컴퓨터구조실험

과제 이름: Project #4

담당교수: 이성원

학 과: 컴퓨터정보공학부

학 번: 2019202021

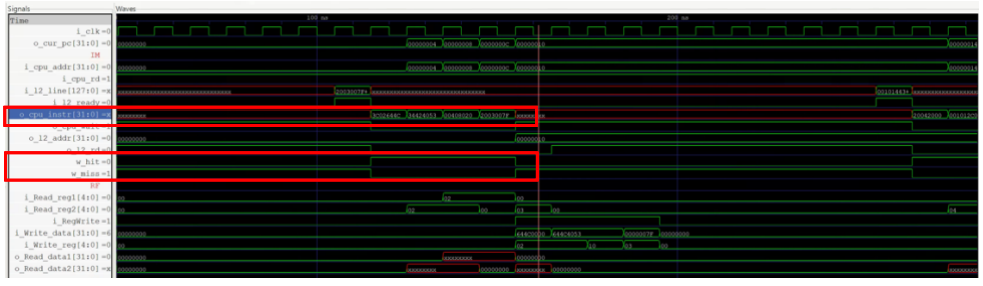
이 름: 정 성 엽

제출일: 2023/6/15

1. Introduction
   A. 과제 소개
      - 정렬 프로그램과 Benchmarks 프로그램에 각각 적합한 cache를 찾고, 각각의 프로그램의 적합한 cache가 왜 다른 건지 프로그램을 분석하여 비교해야 한다. 이때 AMAT을 이용하여 적합한 Cache configuration을 분석한다.
2. 실험 내용
   A. 정렬 프로그램
      - 먼저 Insertion Sort와 Random Access 프로그램의 Instruction cache와 Data cache의 타이밍에 대해 논한다. 이 때 Microprocessor 는 MIPS 기반의 5-stage pipelined 구조 machine이다. Instruction cache는 direct-mapped로 cache는 32 blocks이다. Data cache는 2-ways set associative cache로 32block이다. Replacement policy는 LRU이고 각 block은 4 32bit words이다.
      - 명령어 순서

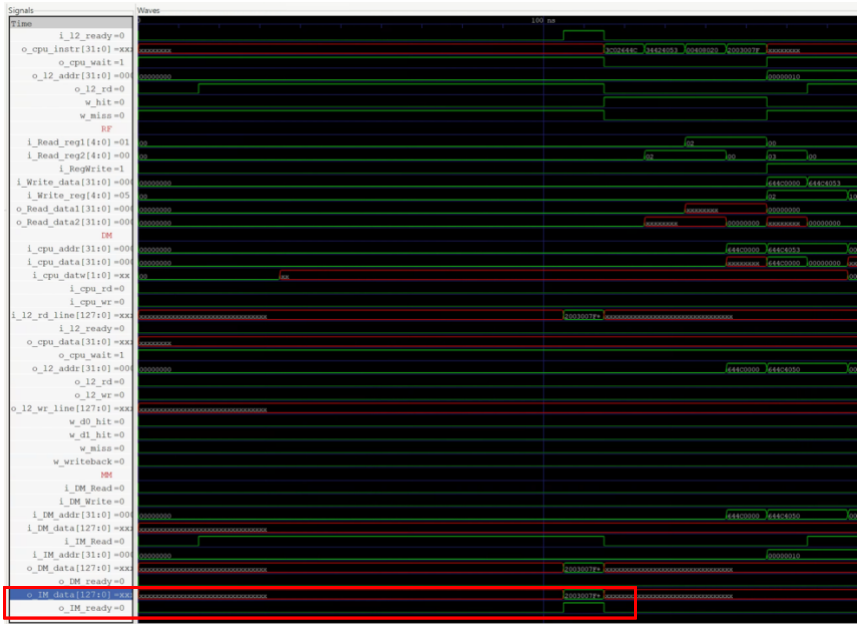
```
00111100000000100110010001001100  // 0x00        lui     $2, 0x644c
00110100010000100100000001010011  // 0x04        ori     $2, $2, 0x4053
00000000010000001000000000100000  // 0x08        add     $16, $2, $0
00100000000000110000000001111111  // 0x0C        addi    $3, $0, 0x1ff
00100000000001000010000000000000  // 0x10        addi    $4, $0, 0x2000
00000000000100000000100101100000  // 0x14  L1:   sll     $2, $16, 11
00000010000000101000000000100110  // 0x18        xor     $16, $16, $2
00000000000010000000101000100011  // 0x1C        sra     $2, $16, 17
00000010000000101000000000100110  // 0x20        xor     $16, $16, $2
00000000000010000000100001000000  // 0x24        sll     $2, $16, 2
00000010000000101000000000100110  // 0x28        xor     $16, $16, $2
00110010000001010000001111111100  // 0x2C        andi    $5, $16, 0x03fc
00000000101001000010100000100000  // 0x30        add     $5, $5, $4
00000010000000000000100000101010  // 0x34        slt     $1, $16, $0
00010100001000000000000000000001  // 0x38        bne     $1, $0, L2
10101100101100000000000000000000  // 0x3C        sw      $16, 0($5)
10001100101001100000000000000000  // 0x40  L2:   lw      $6,  0($5)
00100000110001111111111111111111  // 0x44        addi    $3, $3, -0x1
00010100011000001111111111110010  // 0x48        bne     $3, $0, L1
00000000000000000000000000001101  // 0x4C        break
00000100000000011111111111111111  // 0x50  L3:   b       L3
```

      - Random access
         1. 어떻게 Instruction or Data가 hit operates 또는 miss operates 되었는가
            A. Instruction cache는 direct-mapped로 cache hit 여부는 하나의 w_hit요소로 확인할 수 있다.

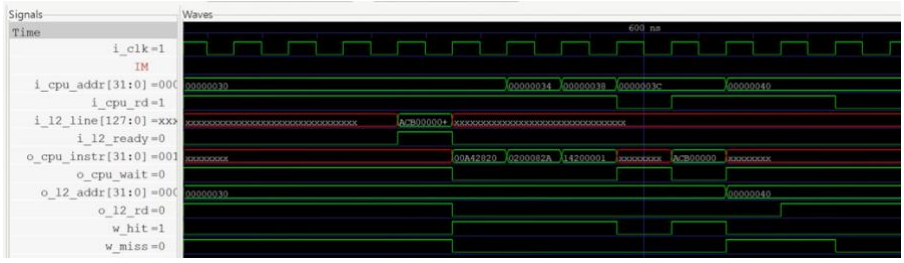


먼저 o_cpu_instr의 0x3X02644C는 lui 명령어로 instruction cache

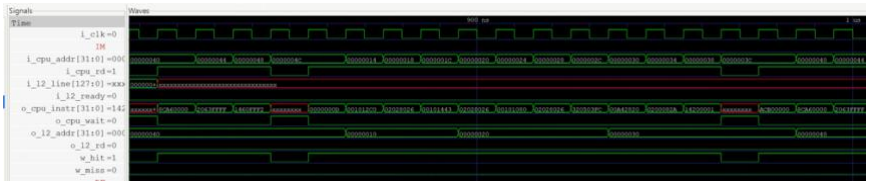
에 hit 하였음을 확인 할 수 있다. 하지만 이전에는 miss가 되고 있기 때문에 Instruction cache memory는 main memory에 접근하여 data를 받아옴을 확인할 수 있다. Main memory의 파형을 확인하면 다음과 같다.



o_IM_ready가 1로 set 되어 128bit data가 instruction cache로 전달되었다. 이때 명령어를 4개씩 가져옴을 확인할 수 있는데 이는 4 32bit word 단위로 block을 구성하기 때문이다. 이후 특이점이 없다가 bne 명령어 이후 특이한 점이 생기는데 다음과 같다

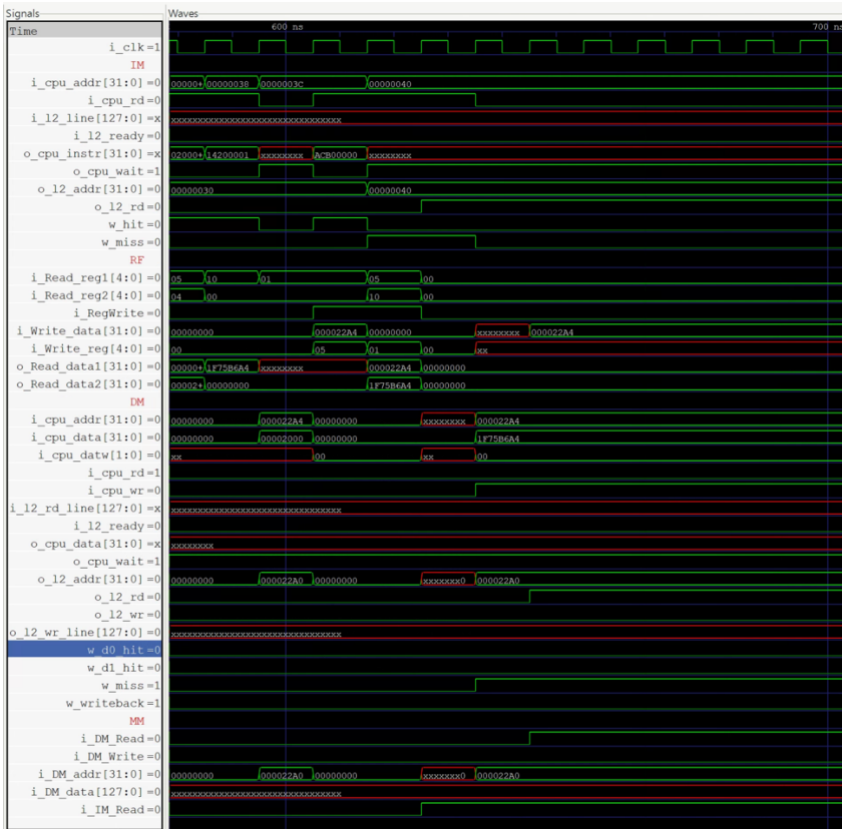


branch 명령어이기 때문에 cpu가 wait하여 cpu_instr이 존재하지 않기 때문에 hit를 하지 못했다. 하지만 not equal 하지 않아서 sw 명령어가 그대로 진행되고 있다. 이후 0x48번의 bne가 작동하여 0x14로 이동하면 이미 instruction을 저장하여 cache에 있는 것을 사용하기 때문에 계속 hit가 나타나는 것을 확인할 수 있다.

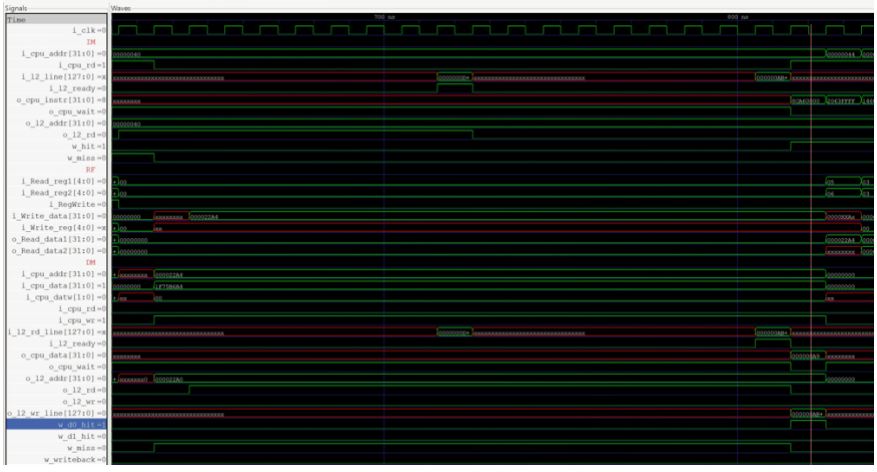


B. Data cache는 2-ways set associate cache로 cache hit 여부는

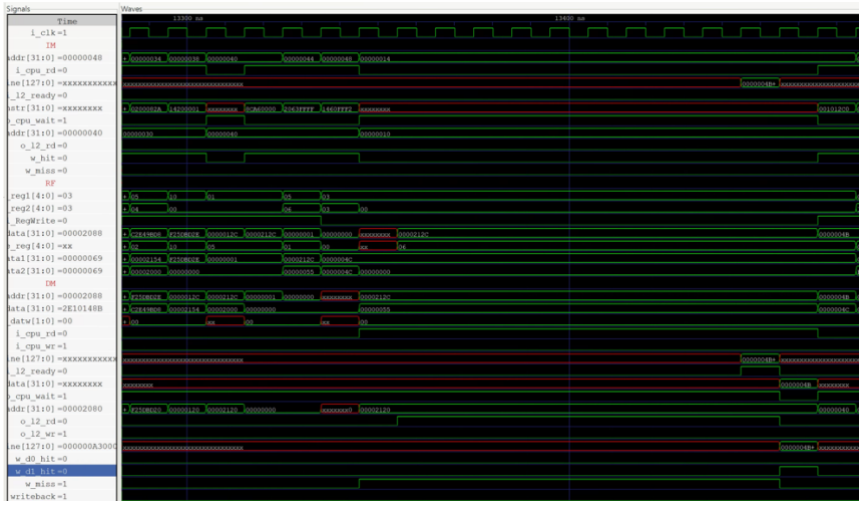
2개의 w_d0_hit, w_d1_hit로 확인할 수 있다. 메모리를 사용하지 않는 명령어에서는 Data cache의 hit과 miss의 변화가 없었지만 sw 명령어를 처음 사용할 때 miss가 발생하였다.



sw 명령어가 실행되고 해당 명령어의 mem 단계에서 w_miss가 발생했다. 이는 처음 해당 주소의 메모리를 접근했기 때문이다.



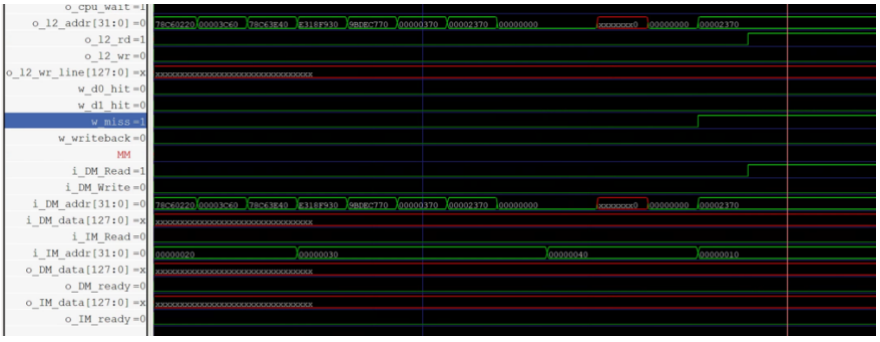
처음으로 data memory cache에 대해 hit가 발생하는 경우이다. 이 때, 중복되는 지금까지 miss된 적이 1번이므로 hit되었다면 w_d0가 hit된다.



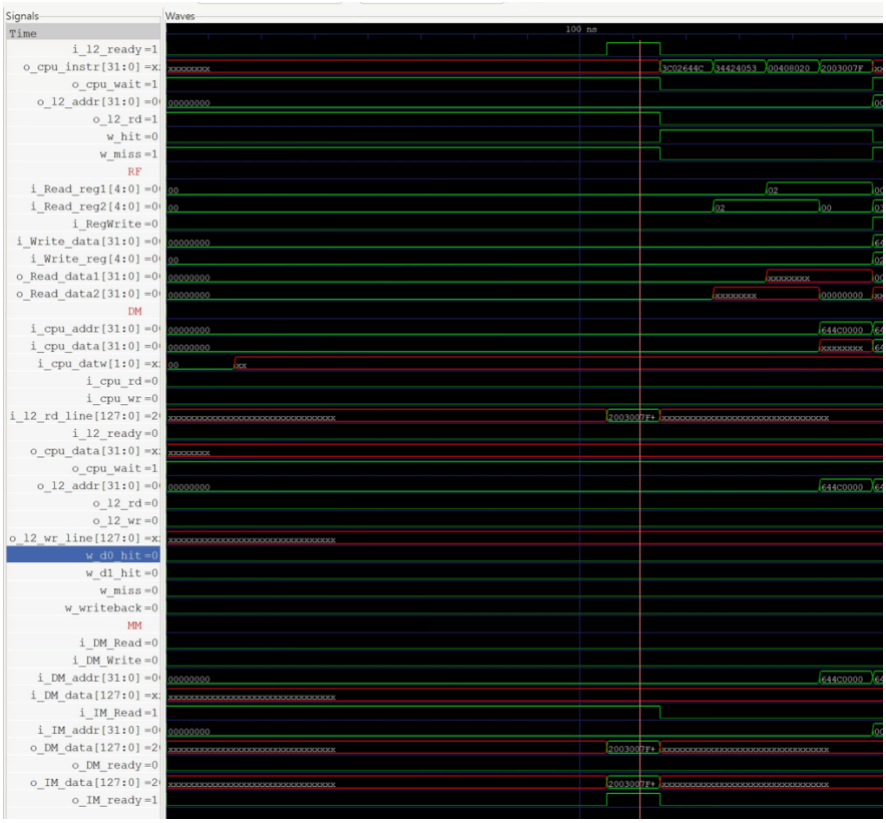
w_d1이 처음 hit 되는 경우는 비교적 매우 뒷 쪽에서 찾을 수 있었다. 이는 이전 miss를 통해 저장된 경우는 물론 해당 tag와 Index로 똑같이 들어가야 저장되기 때문에 시간이 꽤 소요되었다.

2. 언제 어떻게 main memory(L2)가 operate되었는가

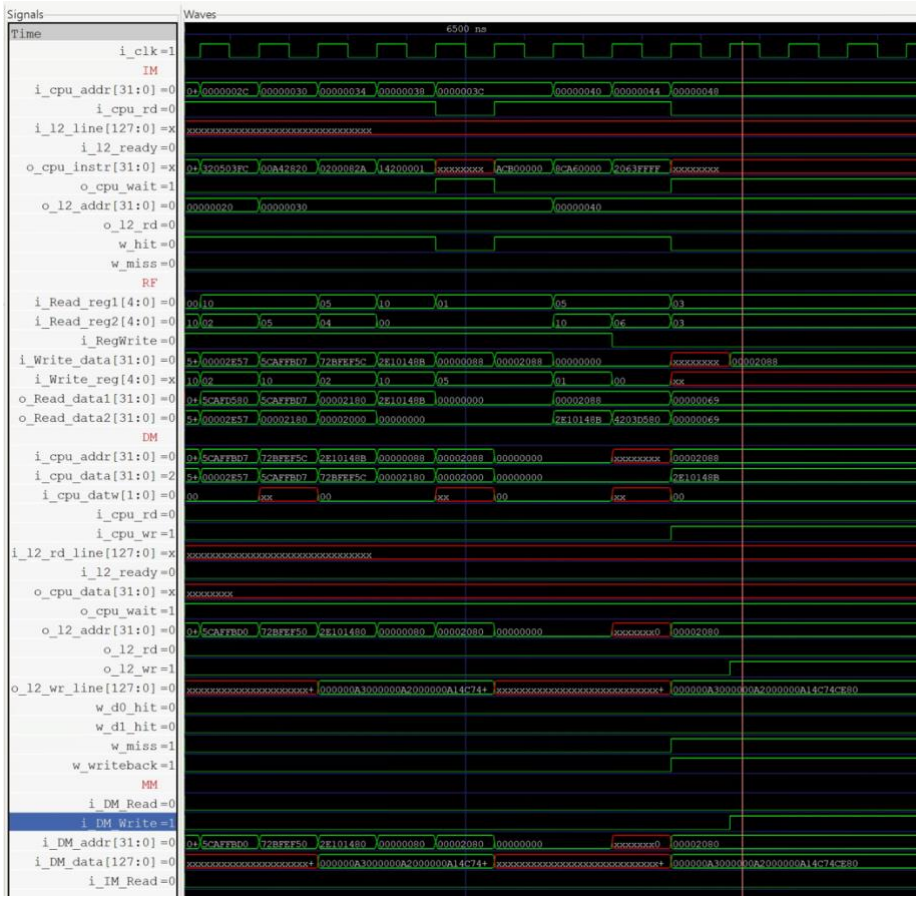
위에서 말한 것과 같이 sw 명령어가 실행되면 mem 단계에서 cache에 miss가 발생하고 이 다음 사이클에 해당 값을 main memory에서 read를 한다. 이는 아래 사진에서 확인할 수 있다.



i_IM_READ 또한 위에서 말한 것과 같이 처음에는 명령어가 Cache에 저장되어 있지 않기 때문에 memory에 접근하여 명령어에 대한 data를 읽는다.

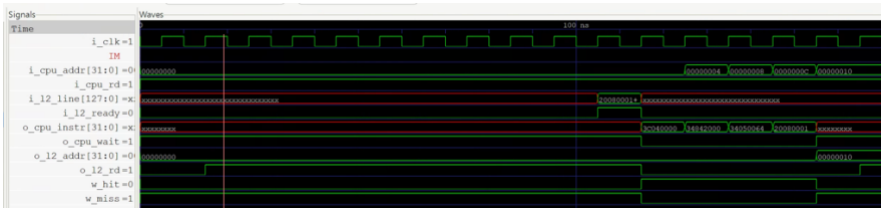


처음 i_DM_WRITE는 sw에서 miss 발생함과 같이 writeback이 set 된다면 set 된다.



- Insertion sort
  1. 어떻게 Instruction or Data가 hit operates 또는 miss operates 되었는가
     A. Instruction hit/miss operates

        대체적으로 Random Access와 Insertion Sort의 hit와 miss의 경우는 크게 다르지 않다. 먼저 명령어를 처음 가져오는 경우 miss이며 가져온 이후에는 hit가 된다.

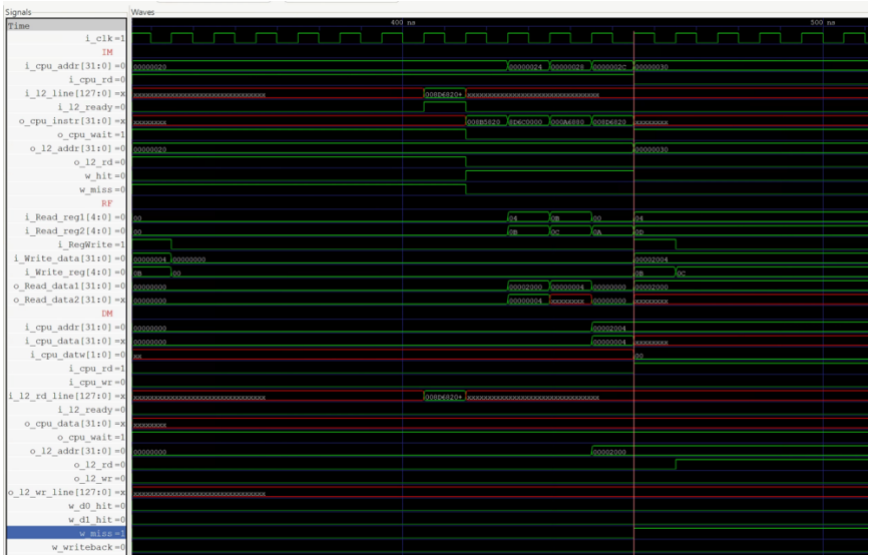
        다시 호출되어도 hit가 된다.

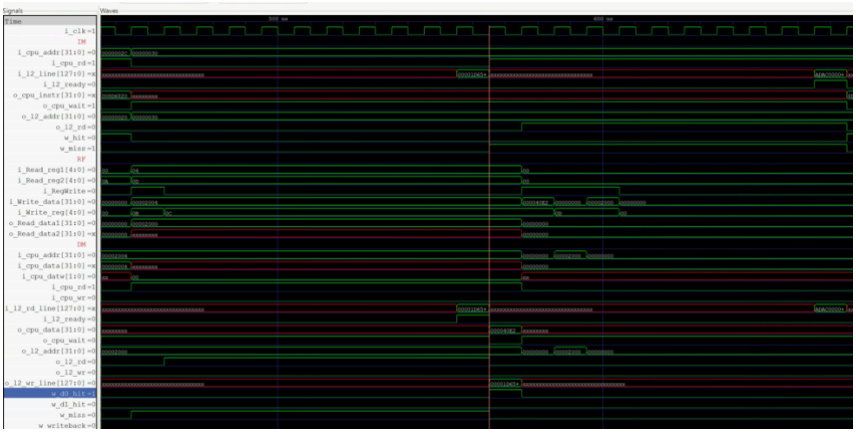
        명령어 호출에서 크게 다른 점이라면 Random Access에 비해 다시 호출된 명령어에 대해 stall되는 경우가 적은 것으로 보이는데 이는 insertion sort로 순차 액세스 패턴을 가져 캐시의 공간 지역성을 활용하여 데이터를 효율적으로 캐시에 로드하였기 때문으로 판단된다.

     B. Data hit/miss operates

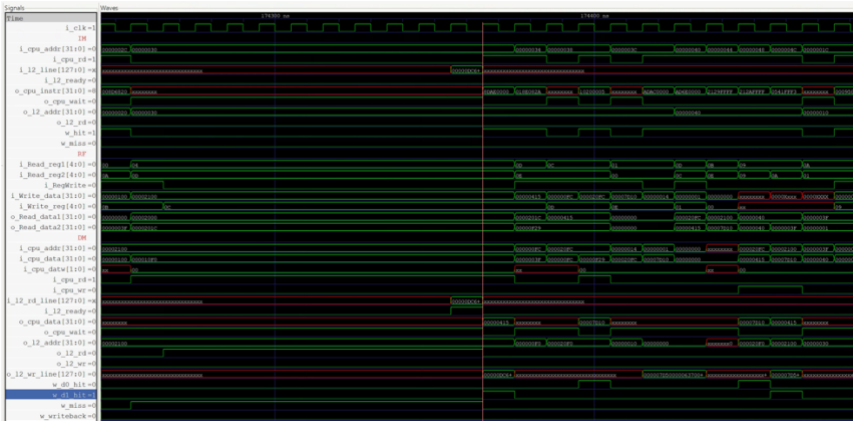
        명령어 순서가 다르기 때문에 lw가 먼저 나와서 random access에 비해 빠르게 data memory cache miss가 발생했다.

        또한 그 이후 과정에서 w_d0_hit가 되었다.

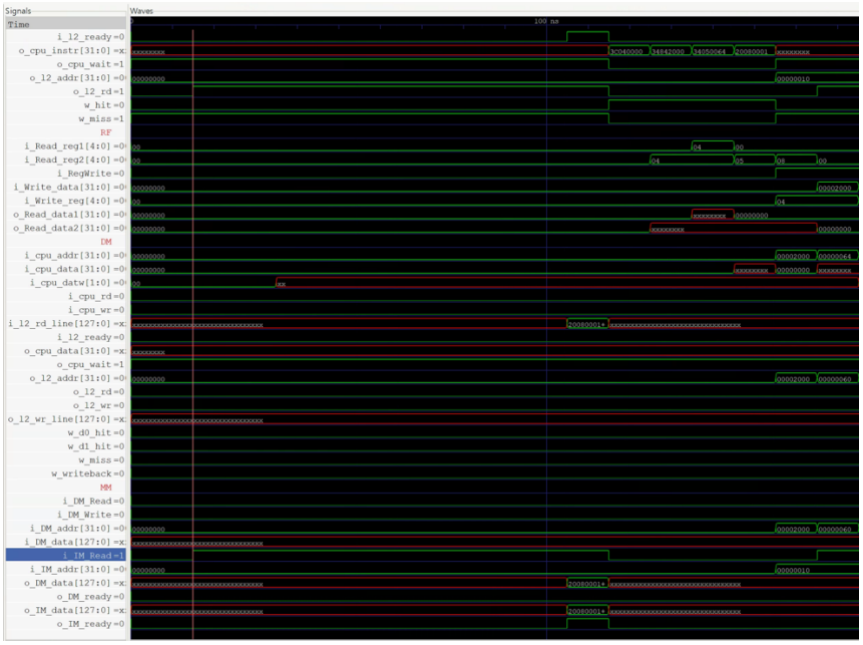


또한 이후 많은 cycle이 지난 후에 w_d1_hit가 발생하였다. 이는 random acccess와 마찬가지로 많은 경우를 지나고 miss가 발생해야 다음 way로 저장하기 때문에 많은 시간이 걸린 것으로 추측할 수 있다.

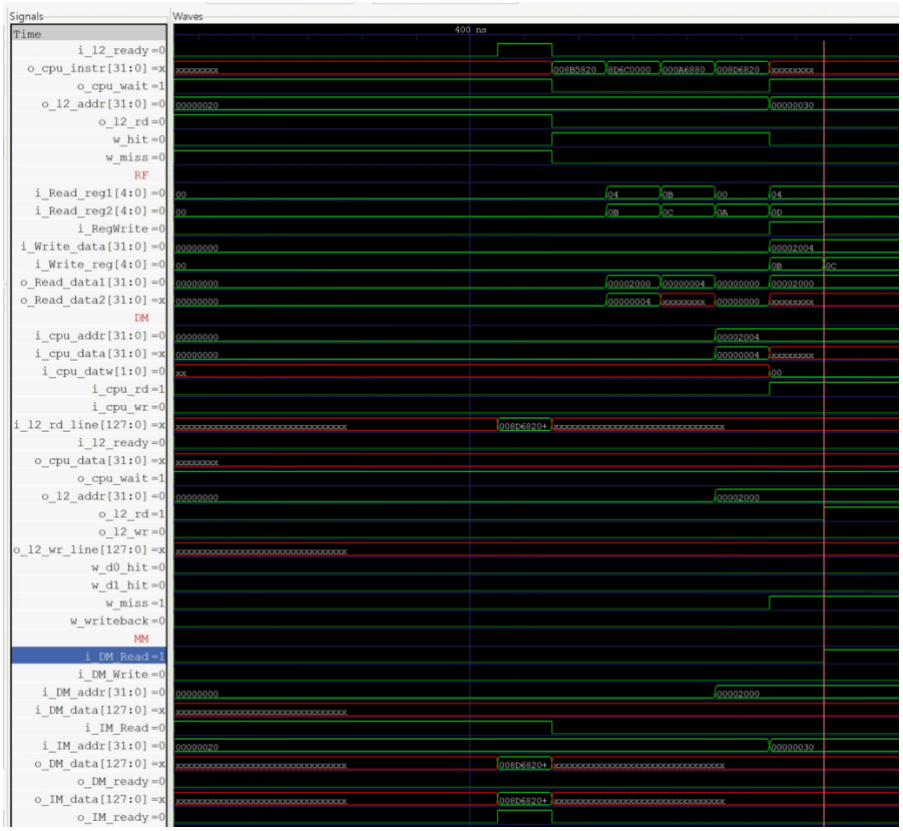


2. 언제 어떻게 main memory(L2)가 operate되었는가

Random Access와 마찬가지로 그리고 위에서 말한 것과 같이 Instruction을 처음 가져올 때 i_IM_Read가 set된다.



그리고 i_DM_Read는 lw 명령어에서 miss가 되었을 때 다음 cycle에서 set되었으며 이에 대한 파형은 아래와 같다.



마지막으로 i_DM_Write는 이상하게도 1로 set 되는 경우를 전혀 찾을 수 없었다.

- 캐시 동작 측면에서 Insertion sort와 Random access의 차이

Insertion Sort는 데이터를 정렬하기 위한 알고리즘으로 데이터를 순차적으로 접근하여, 현재 원소를 정렬된 부분에 삽입하는 방식으로 동작

한다. 캐시 동작 측면에서는 데이터에 연속적인 엑세스 패턴을 가지며, 데이터의 순차적인 접근이 이루어진다. 이러한 순차 엑세스 패턴은 캐시의 공간 지역성을 활용하여 데이터를 효율적으로 캐시에 로드할 수 있다.

Random Access는 데이터에 임의로 접근하는 방식으로 데이터를 인덱스 또는 주소를 통해 직접 엑세스하며, 특정 위치의 데이터를 검색하거나 수정하는데 사용된다. 캐시 동작 측면에서는 데이터에 임의 엑세스 패턴이 발생하며, 캐시에서 해당 데이터를 찾는 과정에서 캐시 미스가 발생할 수 있다. 이 임의 엑세스 패턴은 캐시의 태그 비교 및 데이터 검색 시간에 영향을 미치며, 캐시 효율성을 저하시킬 수 있다.

- 만약 데이터 사이즈가 커진다면

  만약 캐시의 데이터 크기가 커진다면 높은 용량을 활용할 수 있으며, 낮은 캐시 미스 비율을 가진다. 이는 높은 용량과 연계되어 더 많은 데이터를 캐시에 저장할 수 있기 때문에, 캐시에서 원하는 데이터를 찾을 확률이 높아지고, 따라서 캐시 미스가 줄어들게 된다. 그리고 높은 처리량 그리고 높은 캐시 비율로 메모리 엑세스를 줄여 전체 시스템의 성능을 향상 시킬 수 있다.

B. Benchmarks 프로그램

- SimpleScalar라고 불리는 microprocessor로 2 level cache를 설정하고 cc1, ijpeg, perl의 3가지 SPEC CINT95 programs의 성능을 측정한다. SPEC benchmark suite는 컴퓨터 시스템 성능을 측정하도록 설계되었고 이는 많은 OS 기능을 갖춘 고성능 시뮬레이터가 필요하다. 이에 대해 사용하는 것이 SimpleScalar라는 process simulator이다.
- 추가적으로 제안서에서 cache size가 2배되면 cycle time이 4% 그리고 associativity가 2배 되면 2% 증가라고 했지만 명확한 기준을 주어지지 않았기 때문에 cache set가 8일 때를 기준으로 하고 associativity가 1일 때를 기준으로 하여 계산한다.
- cc1 벤치마크 프로그램에 적합한 cache 찾기

  1. Unified cache / Separate cache

# of Sets	Unified cache Miss rate	Unified cache AMAT (cycle)	Split cache		Split cache AMAT (cycle)
			Inst. Miss rate	Data Miss rate	
64	0.3411	69.34	0.3697	0.2248	66.86
128	0.2745	56.06	0.3118	0.1654	55.24
256	0.2203	45.26	0.2558	0.1154	44.42
512	0.1667	34.58	0.2131	0.0793	36.29

2. L1 cache size / L2 cache size

L1I/L1D/L2U	Inst.Miss rate	Data.Miss rate	Unified Cache Miss rate	AMAT
8/8/1024	0.4827	0.3761	0.2450	40.98
16/16/512	0.4274	0.2954	0.3910	48.56
32/32/256	0.3697	0.2248	0.5968	56.08
64/64/128	0.3118	0.1654	0.8267	59.29
128/128/0	x	x	x	x

3. Large Block size / Small block size

Block Size	Unified cache Miss rate	AMAT
16	0.1667	34.58
64	0.0410	9.44
128	0.0206	5.36
256	0.0118	3.60
512	0.0064	2.52

4. Direct-mapped / Set-Associative

# of Sets	Split Cache Miss rate / AMAT							
	1-way		2-way		4-way		8-way	
64	0.3697, 0.2248	66.86	0.2965, 0.1320	51.12	0.2403, 0.0778	39.99	0.1929, 0.0456	31.37
128	0.3118, 0.1654	55.24	0.2456, 0.0881	41.37	0.1949, 0.0483	31.85	0.1378, 0.0298	22.60
256	0.2558, 0.1154	44.42	0.1986, 0.0564	32.87	0.1408, 0.0313	23.16	0.0858, 0.0192	14.57
512	0.2131, 0.0793	36.29	0.1499, 0.0362	24.78	0.0901, 0.0200	15.27	0.0383, 0.0122	7.39
1024	0.1617, 0.0524	27.43	0.0990, 0.0229	16.76	0.0440, 0.0126	8.27	0.0145, 0.0062	3.67
2048	0.1172, 0.0338	20.02	0.0563, 0.0140	10.16	0.0182, 0.0064	4.26	0.0082, 0.0021	2.58

- ijpeg 벤치마크 프로그램에 적합한 cache 찾기

1. Unified cache / Separate cache

# of Sets	Unified cache Miss rate	Unified cache AMAT (cycle)	Split cache		Split cache AMAT (cycle)
			Inst. Miss rate	Data Miss rate	
64	0.1880	38.72	0.2585	0.2571	52.72
128	0.1006	21.28	0.1089	0.2194	27.72
256	0.0457	10.34	0.0212	0.1484	10.94

512	0.0322	7.68	0.0047	0.0791	5.38

2. L1 cache size / L2 cache size

L1I/L1D/L2U	Inst.Miss rate	Data.Miss rate	Unified Cache Miss rate	AMAT
8/8/1024	0.4912	0.4193	0.0310	16.95
16/16/512	0.3733	0.3511	0.0729	16.84
32/32/256	0.2585	0.2571	0.1261	15.09
64/64/128	0.1089	0.2194	0.4078	16.79
128/128/0	x	x	x	x

3. Large Block size / Small block size

Block Size	Unified cache Miss rate	AMAT
16	0.0322	7.68
64	0.0093	3.10
128	0.0014	1.52
256	0.0008	1.40
512	0.0006	1.36

4. Direct-mapped / Set-Associative

# of Sets	Split Cache Miss rate / AMAT							
	1-way		2-way		4-way		8-way	
64	0.2585, 0.2571	52.72	0.1172, 0.1812	27.31	0.0173, 0.0474	5.85	0.0050, 0.0251	2.96
128	0.1089, 0.2194	27.72	0.0170, 0.1537	10.48	0.0055, 0.0357	3.54	0.0012, 0.0154	1.98
256	0.0212, 0.1484	10.94	0.0054, 0.0711	5.10	0.0014, 0.0160	2.08	0.0004, 0.0113	1.72
512	0.0047, 0.0791	5.38	0.0015, 0.0517	3.69	0.0004, 0.0114	1.76	0.0003, 0.0090	1.64
1024	0.0035, 0.0517	4.04	0.0010, 0.0126	1.95	0.0003, 0.0092	1.69	0.0003, 0.0084	1.65
2048	0.0025, 0.0157	2.36	0.0003, 0.0097	1.75	0.0003, 0.0085	1.70	0.0003, 0.0081	1.68

- perl 벤치마크 프로그램에 적합한 cache 찾기

1. Unified cache / Separate cache

# of Sets	Unified cache Miss rate	Unified cache AMAT	Split cache		Split cache AMAT
			Inst. Miss	Data Miss	

		(cycle)	rate	rate	(cycle)
64	0.3712	75.36	0.4051	0.2391	73.00
128	0.2903	59.22	0.3522	0.1807	62.16
256	0.2303	47.26	0.2793	0.1387	49.32
512	0.1816	37.56	0.2134	0.1081	38.12

2. L1 cache size / L2 cache size

L1I/L1D/L2U	Inst.Miss rate	Data.Miss rate	Unified Cache Miss rate	AMAT
8/8/1024	0.4579	0.3748	0.3072	46.35
16/16/512	0.4342	0.2963	0.4150	51.65
32/32/256	0.4051	0.2391	0.5760	59.39
64/64/128	0.3522	0.1807	0.8108	65.57
128/128/0	x	x	x	x

3. Large Block size / Small block size

Block Size	Unified cache Miss rate	AMAT
16	0.1816	37.56
64	0.0493	11.10
128	0.0254	6.32
256	0.0153	4.30
512	0.0072	2.68

4. Direct-mapped / Set-Associative

# of Sets	Split Cache Miss rate / AMAT							
	1-way		2-way		4-way		8-way	
64	0.4051, 0.2391	73.00	0.3520, 0.1507	60.40	0.2676, 0.1039	45.59	0.1978, 0.0693	33.57
128	0.3522, 0.1807	62.16	0.2807, 0.1108	47.91	0.1999, 0.0728	34.10	0.1251, 0.0460	21.79
256	0.2793, 0.1387	49.32	0.2029, 0.0820	35.09	0.1298, 0.0514	22.80	0.0670, 0.0360	12.85
512	0.2134, 0.1081	38.12	0.1404, 0.0593	24.82	0.0728, 0.0376	13.82	0.0343, 0.0301	7.83
1024	0.1685, 0.0716	29.65	0.0871, 0.0399	16.06	0.0372, 0.0303	8.30	0.0214, 0.0288	5.93
2048	0.1418, 0.0589	25.12	0.0615, 0.0332	12.02	0.0240, 0.0289	6.35	0.0184, 0.0287	5.53

3. 검증 전략, 분석 및 결과

A. AMAT를 이용하여 적합한 Cache configuration 분석

- cc1 벤치마크 프로그램
  1. cc1은 C 소스 코드를 어셈블리어로 변환하는 과정을 수행하는 컴파일러의 단계를 벤치마크 하여 일반적으로 소스코드 크기, 복잡성, 제어 흐름, 연산 등 다양한 측면을 포함하여 다양한 시나리오에서 컴파일러 동작을 테스트하고 비교한다.
  2. 시뮬레이션 결과, cc1 벤치마크 프로그램에서 가장 적합한 Cache configuration은 # of set이 512이고 block size가 512인 1-way unified cache이다. 이다. 이 cache는 AMAT이 2.52로 시뮬레이션 한 결과 중 가장 작은 값을 가졌다.
- ijpeg 벤치마크 프로그램
  1. ijpeg는 이미지 압축 알고리즘을 평가하고 비교하기 위한 벤치마크 프로그램으로 이미지를 JPEG 형식으로 압축하는 과정을 수행하는 프로그램이다. ijpeg 벤치마크는 이미지 처리와 압축에 관련된 직업을 수행하며, 압축 알고리즘의 성능, 압축률, 실행 시간 등을 측정하고 비교할 수 있습니다.
  2. 시뮬레이션 결과, ijpeg 벤치마크 프로그램에서 가장 적합한 Cache configuration은 # of set이 512이고 block size가 512인 1-way unified cache이다. 이 cache는 AMAT이 1.36으로 시뮬레이션 한 결과 중 가장 작은 값을 가졌다.
- perl 벤치마크 프로그램
  1. Perl 벤치마크 프로그램은 Perl 언어의 성능을 측정하고 비교하기 위해 사용되는 프로그램이다. Perl 벤치마크는 Perl 인터프리터의 실행 속도, 메모리 사용량, 코드 성능 등을 평가하고 비교하는데 사용된다.
  2. 시뮬레이션 결과, perl 벤치마크 프로그램에서 가장 적합한 Cache configuration은 # of set이 512이고 block size가 512인 1-way unified cache이다. 이 cache는 AMAT이 2.68로 시뮬레이션 한 결과 중 가장 작은 값을 가졌다.

4. 문제점 및 고찰

A. 프로젝트 내용 전체 정리 및 고찰

- 이번 프로젝트는 Insertion Sort와 Random Access하는 Cache의 파형을 보고 비교하며 언제 L1, 또는 L2에 hit or miss 그리고 main memory에 접근하는지 알아보았다. 또한 Linux Ubuntu 환경에서

SimpleScalar 시뮬레이터를 이용하여 3개의 벤치마크 프로그램인 cc1, ijpeg, perl을 각 조건에 맞춰서 최적의 cache는 무엇인지 찾는 것이다. 이 때, simplescalar란 Alpha, PISA, ARM, x86을 가상으로 실행해 볼 수 있는 시뮬레이터이다.

GTK wave의 파형을 확인하는 과정은 생각보다 많은 시간을 소요하였다. GTK wave에서 볼 수 있는 정보가 명령어와 cache의 세부 정보를 확인할 수 있는데 많은 정보들도 있고, RF가 있지만 정확히 어떤 타이밍에 pipeline의 어떤 단계에 진행하고 있는지 다른 파형들을 통해 확인하는 과정은 매우 복잡하고 어려운 점이 많았다.

벤치마크 테스트는 각 프로그램마다 4가지로 나누어 진행하였다. 첫 번째 simulation은 unified인 경우와 split인 경우에 block size와 associativity를 1으로 고정하고 # of set을 바꿔가며 진행하였다. 두 번째 simulation은 block size가 16이고, Associativity가 1로 고정된 상황에서 L1I/L1D/L2U의 사이즈를 바꿔가며 진행하였다. 세번째는 unified cache를 # of sets를 associativity를 1로 고정하고 block size를 바꿔가며 진행하였다. 마지막으로 simulation 4는 block size를 16으로 고정하고 associativity와 # of set을 바꿔가며 실험을 진행하였다.

프로젝트 자체는 어렵지 않았으나 각 simulation을 위해 값을 바꿔가며 하나하나 값을 받고 해당 값을 통해 AMAT을 계산해야 했으며 이 계산하는 과정에서는 cache size가 2배되면 access time 4% 증가 associativity가 2배되면 access time이 2% 증가하는데 이 때 각 요소에 접근하는 시간은 통일되는 것이 아닌 각 cache나 main memory에 따라 달라진다. 또한 split이면 각 비율을 계산해서 적용하며 level이 2개라면 하위 AMAT을 계산하여 상위 AMAT을 계산하는 데 이에 시간이 매우 오래 걸렸다.

마지막으로 L1 cache size각 128/128 그리고 L2 unified cache size가 0인 경우 벤치마크에서 사이즈가 0인 경우를 거부하기 때문에 x로 표현하였다.

B. Benchmarks에 최적화된 cache와 정렬 프로그램에 최적화된 cache 비교 및 분석

- cc1 벤치마크 프로그램에서 가장 적합한 Cache configuration은 # of set이 512이고 block size가 512인 1-way unified cache이다. 이다. 이 cache는 AMAT이 2.52로 시뮬레이션 한 결과 중 가장 작은 값을 가졌다. 시뮬레이션 결과, ijpeg 벤치마크 프로그램에서 가장 적합한 Cache configuration은 # of set이 512이고 block size가 512인 1-way

unified cache이다. 이 cache는 AMAT이 1.36으로 시뮬레이션 한 결과 중 가장 작은 값을 가졌다. 시뮬레이션 결과, perl 벤치마크 프로그램에서 가장 적합한 Cache configuration은 # of set이 512이고 block size가 512인 1-way unified cache이다. 이 cache는 AMAT이 2.68로 시뮬레이션 한 결과 중 가장 작은 값을 가졌다. 신기하게도 모두 같은 경우의 cache가 최적의 경우임을 보이고 있다.