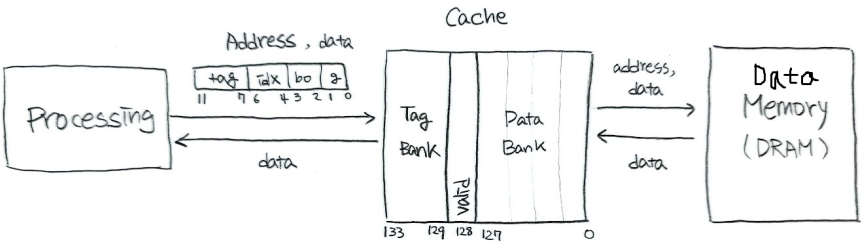
Report: Lab6

20190533 Hyemin Lee
20190235 Gangtae Park

1. Introduction

Lab6은 cache lab으로, lab5에서 구현했던 pipelined CPU에 cache를 추가하는 랩이다. Cache는 CPU가 연산을 위해서 메모리에 있는 값을 필요로 하거나 메모리에 값을 저장할 때 시간이 많이 걸리는 것을 보완하기 위해서 개발된 중간자 역할의 저장소이다. 항상 메모리에 직접적으로 읽고 쓰는 것보다 사용했던 값은 cache에 따로 저장해둠으로써 memory stage에서 소요되는 cycle 수를 감소시키는 것이 목적이다.

2. Design



우리는 direct-mapped cache를 구현하여 processing part와 data memory를 연결하였다. (instruction memory를 위한 cache는 사용하지 않았다.) 메뉴얼에 따라 cache-line 당 4 words를 저장하고 data bank의 용량을 128B로 맞추었다. 이에 따라 cache의 data bank에 총 8줄의 cache-line이 배정되었다. 그래서 12bit의 address 중에서 0-1bit은 offset이, 2-3bit은 block이, 4-6bit은 index가 되었고, 남는 7-11bit은 tag가 되었다. 본 cache는 write hit이 발생하였을 때는 **write-through** policy를 이용해서 cache와 memory값을 업데이트하고, write miss가 발생했을 때는 **write-allocate** policy를 이용해서 cache에 값을 채운다. 그리고 blocking cache여서 memory를 접근하고 있는 동안 다음 instruction이 진행되지 않고 기다리게 된다.

3. Implementation

(431~end line: Main implementation) 어떤 instruction이 mem stage에 도착하면 해당 instruction이 LW 또는 SW인지 확인한다. LW 또는 SW이면 접근하려는 주소가 cache에 있는지 hit 여부를 확인한다. Hit은 접근하려는 주소의 tag와 이에 대응하는 cache-line의 tag값이 같으며 valid bit이 1일 때 성립한다. LW이냐 SW이냐, hit이냐 miss이냐에 따라 어떻게 작동하는지는 아래에 자세히 설명하였다. **(150 line: cache_flag**를 이용한 모든 **stage** 멈춤**)** Blocking cache를 구현하기 위해서 cache_flag를 사용하였다. cache_flag가 올라가면 cache와 관련된 작업을 제외한 모든 작업이 멈추고, cache_flag가 내려가야지만 재개한다.

Read Hit (1cycle)	cache access(1cycle)만 하여 cache에서 바로 값을 읽어서 반환한다. 따라서 cache_flag는 이용하지 않는다.
Read Miss (1+8+1 cycle)	cache access(1cycle)에서 miss임을 판단하면 cache_flag를 올리고 mem_reading 변수에 9를 넣어 memory에서 값을 읽어올 준비를 한다. **(299~309 line)** Clock이 posedge가 될 때마다 mem_reading은 1씩 줄어들며,8,6,4,2가 될 때마다 각각 block 값을 3,2,1,0으로 바꾼 메모리 주소의 데이터를 읽어온다. 읽어온 값은 순서대로 임시 버퍼에 저장되며, mem_reading이 0이 되면 cache

	update(1cycle)를 진행하고 요청한 값을 반환하면서 cache_flag를 0으로 돌린다.
Write Hit (1+8 cycle)	cache access(1cycle)에서 hit임을 판단하면 cache_flag를 올리고 mem_writing에 9를 넣어 memory와 cache에 값을 적을 준비를 한다. mem_writing도 clk이 posedge가 될 때마다 1씩 줄어들며, mem_writing이 8,6,4,2가 될 때마다 각각 block 값을 3,2,1,0으로 바꾼 메모리 주소에 데이터를 적는다. 이때, mem_writing이 8일 때는 cache에도 동시에 적는다. mem_writing이 1이 되면 cache_flag를 내려서 다음 cycle부터 다른 instruction들도 실행할 수 있게 한다.
Write Miss (1+8+1+8 cycle)	cache access(1cycle)에서 miss임을 판단하면 cache_flag를 올리고 mem_reading에 9를 넣어 read miss에서와 동일하게 memory에서 값을 읽어서 비어있는 cache를 채운다. 그리고 cache update를 할 때 mem_writing에 9를 넣어서 write hit일 때와 동일하게 cache와 memory에 원하는 값을 넣고 cache_flag를 내린다.

4. Evaluation

ModelSim을 이용해 TB_RISCV_inst, TB_RISCV_forloop, 그리고 TB_RISCV_sort testbench들을 시뮬레이션하였다. Cache가 포함된 CPU의 경우, READ_TOT, READ_MISS, WRITE_HIT, WRITE_MISS의 개수를 함께 출력하였다. Hit ratio는 HIT/(HIT+MISS)로 READ와 WRITE의 경우를 나누어서 계산하였다. Cache가 포함되지 않은 경우는 모든 READ와 WRITE가 miss가 된 것으로 예상하면 되기 때문에, READ HIT을 READ MISS로, WRITE HIT을 WRITE MISS로 간주했을 때 최종 cycle이 어떻게 되는지 계산하였다.

TB_RSICV_inst의 경우 모든 testcase(26개)를 통과하였고 총 **50 cycle**의 결과를 얻었다. READ_HIT과 WRITE_MISS는 1번씩 발생했고 READ_MISS와 WRITE_HIT은 발생하지 않았음을 알 수 있다. **Cache**가 없을 때의 경우는 **59 cycle**일 것이다. Hit ratio는 다음과 같다.

	READ	WRITE
HIT RATIO	HIT/(HIT+MISS) = 1/(1+0) = **1**	HIT/(HIT+MISS) = 0/(0+1) = **0**

```
# Test #   25 has been passed
# Test #   26 has been passed
# READ TOT :           1
# READ MISS :          0
# WRITE HIT :          0
# WRITE MISS :         1
# Finish:         50 cycle
# Success.
# ** Note: $finish    : C:/Users/rkdxo/Desktop/KAIST/3-2/EE312/lab6/te
stbench/TB_RISCV_inst.v(179)
#    Time: 615 ns  Iteration: 1  Instance: /TB_RISCV_inst
```

▲ TB_RISCV_inst with cache

TB_RISCV_forloop의 경우 모든 testcase(17개)를 통과하였고 총 **226 cycle**의 결과를 얻었다. READ_HIT은 22번, READ_MISS는 발생하지 않았으며 WRITE_HIT과 WRITE_MISS는 각각 9번, 3번 발생했음을 알 수 있다. **Cache**가 없을 경우는 **577 cycle**일 것이다. Hit ratio는 다음과 같다.

	READ	WRITE
HIT RATIO	HIT/(HIT+MISS) = 22/(22+0) = **1**	HIT/(HIT+MISS) = 9/(9+3) = **0.75**

```
# Test #   16 has been passed
# Test #   17 has been passed
# READ TOT :          22
# READ MISS :          0
# WRITE HIT :          9
# WRITE MISS :         3
# Finish:        226 cycle
# Success.
# ** Note: $finish    : C:/Users/rkdxo/Desktop/KAIST/3-2/EE312/lab6/te
stbench/TB_RISCV_forloop.v(167)
#    Time: 2375 ns  Iteration: 1  Instance: /TB_RISCV_forloop
```

▲ TB_RISCV_forloop with cache

TB_RISCV_sort의 경우 또한 마찬가지로 모든 testcase(40개)를 통과하였고 총 **28293 cycle**의 결과를 얻었다. READ_MISS, WRITE_HIT, WRITE_MISS는 각각 735번, 875번, 32번 발생했다. READ_HIT은 READ_TOT에서 READ_MISS를 뺀 값인 4071번 발생했음을 알 수 있다. **Cache**가 없을 경우는 **79807 cycle**일 것이다. Hit ratio는 다음과 같다.

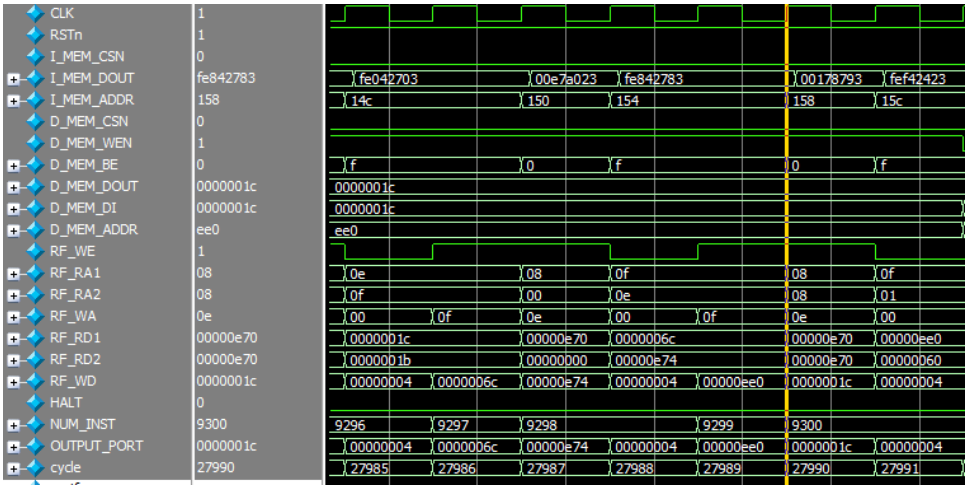
	READ	WRITE
HIT RATIO	HIT/(HIT+MISS) = 4071/(4071+735) = **0.847**	HIT/(HIT+MISS) = 875/(875+32) = **0.965**

```
# Test #   39 has been passed
# Test #   40 has been passed
# READ TOT :        4806
# READ MISS :        735
# WRITE HIT :        875
# WRITE MISS :        32
# Finish:      28293 cycle
# Success.
# ** Note: $finish    : C:/Users/rkdxo/Desktop/KAIST/3-2/EE312/lab6/te
stbench/TB_RISCV_sort.v(193)
#    Time: 283045 ns  Iteration: 1  Instance: /TB_RISCV_sort
```

▲ TB_RISCV_sort with cache

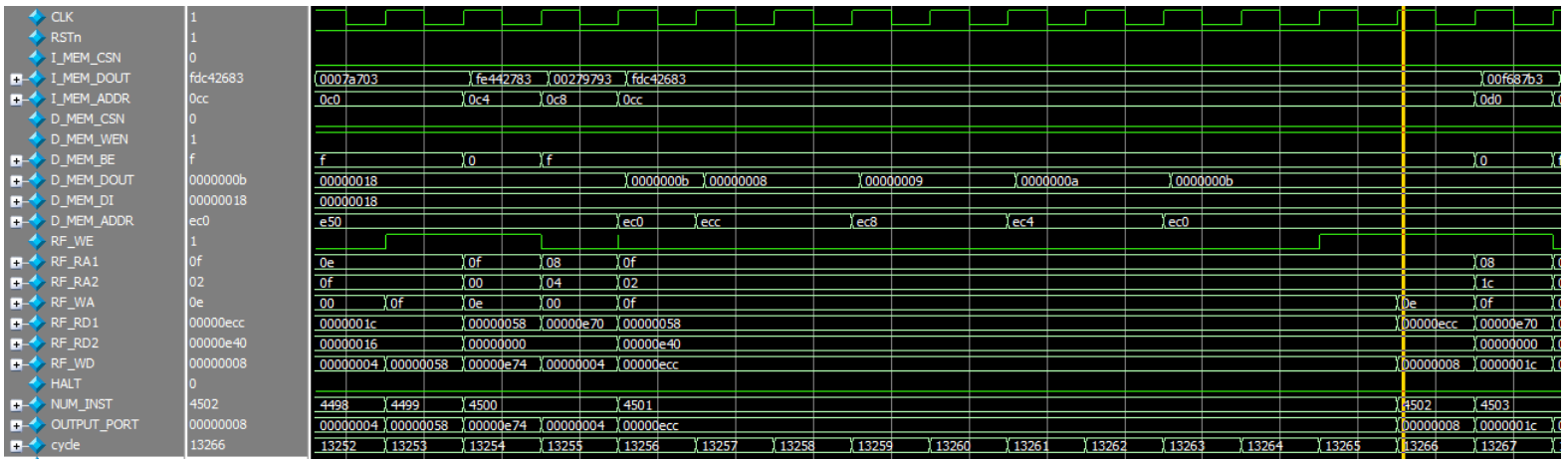
따라서 **cache**가 존재함으로써 **TB_RISCV_inst, TB_RISCV_forloop, TB_RISCV_sort** 각각의 **testbench**에서 **9, 315, 51514 cycle** 씩 **cycle**이 줄어든 것을 확인할 수 있었다**.** 또한, READ HIT, READ MISS, WRITE HIT, WRITE MISS 네 경우 각각에 대해서 ModelSim에서 관찰한 waveform은 다음과 같다. 다음 waveform들은 모두 TB_RISCV_sort를 시뮬레이션했을 때의 형태들이다.

i) READ HIT



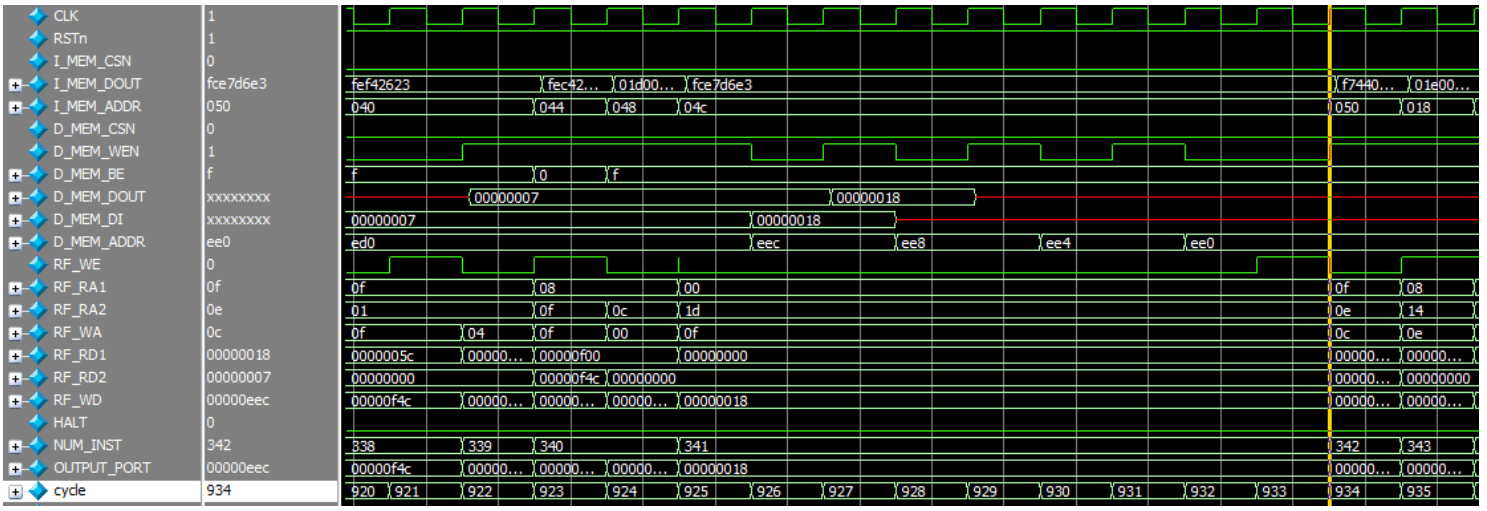
27989 cycle이 READ HIT에 해당하는 1개의 cycle이다. 다음 cycle에서 output에 불러온 값인 1c를 출력한 것을 확인할 수 있다.

ii) READ MISS



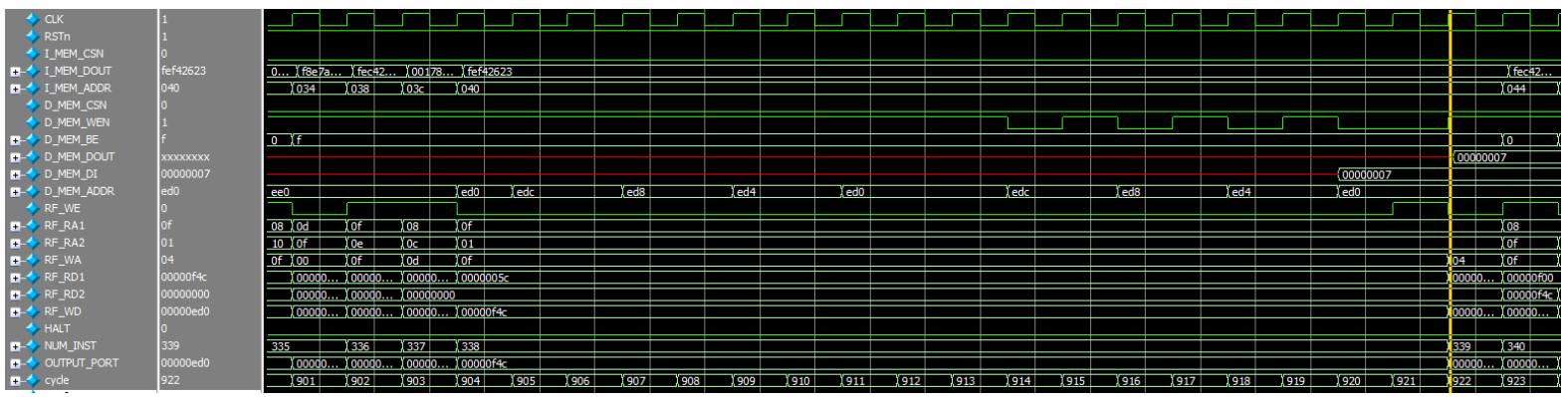
13256 cycle부터 13265 cycle까지 10 cycle이 READ MISS에 해당한다. 2~9 cycle 동안 D_MEM_ADDR이 4번 바뀌는 것을 볼 수 있고, output에 불러온 값(불러온 값은 8이고 메모리의 ecc 주소에서 참조했다.)을 출력한다.

iii) WRITE HIT



925 cycle 부터 933 cycle까지의 9 cycle이 WRITE HIT에 해당한다. 2~9 cycle 동안 D_MEM_ADDR이 4번 바뀌는 것을 볼 수 있고, store을 한 메모리 주소(eec)가 다음 cycle에서 output으로 출력된다.

iv) WRITE MISS



904 cycle 부터 921 cycle까지의 18 cycle이 WRITE MISS에 해당한다. 2~9번째 cycle은 cache에 해당하는 word를 저장하고, 11~18번째 cycle 동안 메모리에 값을 저장하는 것을 볼 수 있다. 예를 들어, D_MEM_ADDR이

ed0인 메모리에는 처음엔 값이 들어있지 않지만 17번째 cycle에서 7이 저장된다. Store을 한 메모리 주소(ed0)가 다음 cycle에서 output으로 출력된다.

5. Discussion

WRITE cache 부분에서 D_MEM_ADDR의 메모리 주소에 D_MEM_DOUT에 있는 값을 저장하는 것을 결정해주는 시그널인 D_MEM_WEN을 조절하는 문제를 발견하는데 어려움이 있었다. WRITE MISS의 경우 cache에 word를 저장할 때는 D_MEM_WEN이 1이고, 메모리에 값들을 저장할때 0으로 설정해주어야 하는데 18 cycle 동안 D_MEM_WEN을 0으로 해두었더니 원하지 않는 순간에 메모리 업데이트가 이루어져 쓰레기값이 저장되는 상황이 발생했었다.

6. Conclusion

Lab6는 앞서 구현했던 pipeline CPU에서 cache를 구현하는 랩이었다. Cache를 직접 구현해보니 메모리와 CPU간의 관계에 대해서 더욱 잘 이해할 수 있었다. 또한, 문제점을 찾기 위해서 처음부터 구현했던 single cycle CPU, multi cycle CPU, 그리고 pipeline CPU를 다시 한번 검토해보면서 전체적인 흐름을 깨달을 수 있었다.