EE312 <Lab 6: Cache Lab>

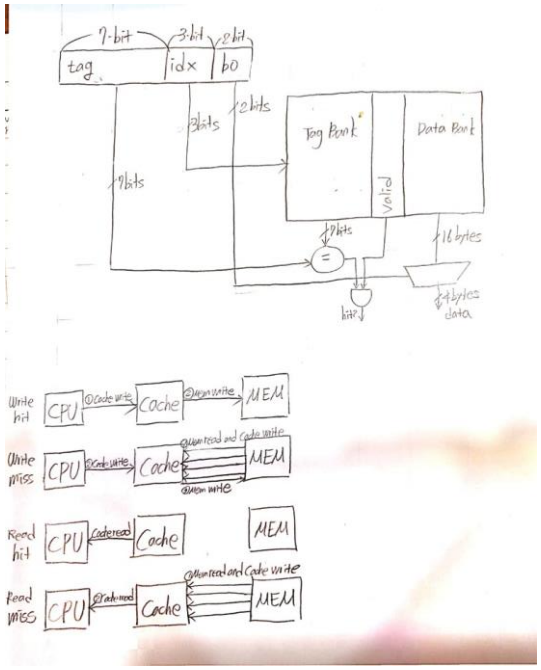
20170616 정희진

1. Instruction

Lab6은 cache를 직접 디자인 해보는 랩이다. cache의 structure, replacement, 그리고 write policy는 목적에 따라 다를 수 있다. 제출된 Cache.v 파일은 direct-mapped cache이고, 때문에 특정한 replacement policy가 적용되지 않았다. write policy는 write-through와 write allocate 방식을 이용했다. 또한 blocking cache방법을 이용하여, cache miss가 일어날때마다 pipelined CPU가 stall하도록 작성하였다. RISCV_TOP.v 파일은 Lab5에서 제출한 파일에서 조금만 바꿔 제출한 것이다.

2. Design

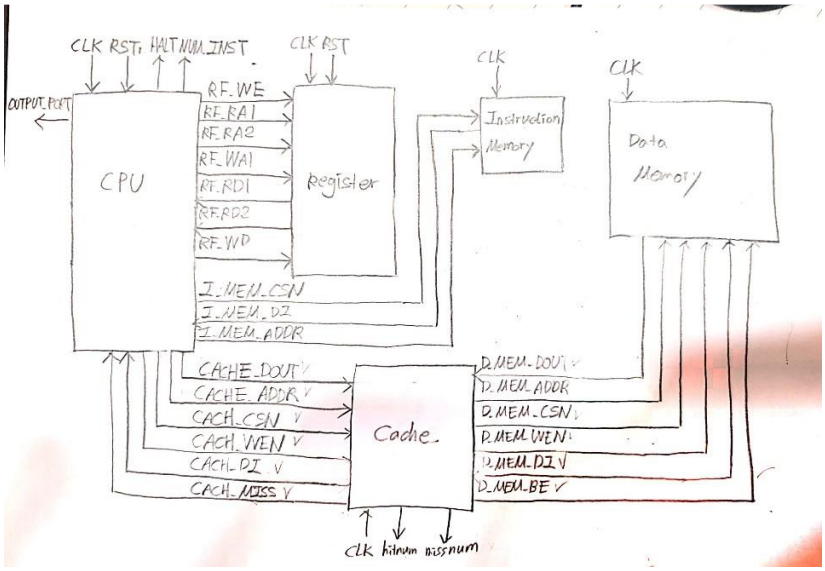
cache design은 아래의 그림과 같다.



3. Implementation

(1) TB_RISCV_***.v

Cache.v 파일이 생겨나면서 TB파일 또한 조금씩 바꿔줘야 되었다. 먼저 RISCV_TOP.v -> Cache.v -> Mem_Model.v 순으로 data가 이동할 수 있도록 TB에서 바꿔주었다. 또한 hit rate를 알 수 있도록 hit 개수와 miss 개수를 display할 수 있도록 바꿔주었다



(2) RISCV_TOP.v

Lab5 때 제출한 pipelined CPU 파일에서 cache miss가 날 때마다 stall을 해주도록 바꿔주었다. 이는 CACHE_MISS 변수를 통해 조절할 수 있다.

(3) Cache.v

cache의 cycle에 따른 implementation은 다음 그림과 같다.

	cycle	0	1	2	3	4
write miss	CACHE_CSN	0	0	0	0	0
	CACHE_WEN	0	0	0	0	0
	CACHE_MISS	1	1	1	1	0
	MEMREAD	1	1	1	1	0
	MEMWRITE	0	0	0	0	1
	MEMADDR	word1	word2	word3	word3	data
write hit	CACHE_CSN	0				
	CACHE_WEN	0				
	CACHE_MISS	0				
	MEMREAD	0				
	MEMWRITE	1				
	MEMADDR	data				
read miss	CACHE_CSN	0	0	0	0	0
	CACHE_WEN	1	1	1	1	1

	CACHE_MISS	1	1	1	1	0
	MEMREAD	1	1	1	1	1
	MEMWRITE	0	0	0	0	0
	MEMADDR	word1	word2	word3	word4	data
read hit	CACHE_CSN	0				
	CACHE_WEN	1				
	CACHE_MISS	0				
	MEMREAD	1				
	MEMWRITE	0				
	MEMADDR	data				

CACHE_CSN와 CACHE_WEN는 Mem_Model.v에 있는 CSN, WEN와 같은 역할을 한다. CACHE_MISS는 cache miss를 알려주며, 이 변수가 1이 될 때 CPU가 stall된다. MEMREAD와 MEMWRITE는 MEM에서 데이터를 읽을 것인지 쓸 것인지 정해주고, MEMADDR은 각 clock마다 MEM에 input되는 address를 뜻한다(또는 cache에 data가 쓰일 자리를 알려준다).

4. Evaluation

모든 testcase를 통과하였으며 속도도 빨랐다.

1) TB_RISCV_inst.v

```
# Test #    1 has been passed
# Test #    2 has been passed
# Test #    3 has been passed
# Test #    4 has been passed
# Test #    5 has been passed
# Test #    6 has been passed
# Test #    7 has been passed
# Test #    8 has been passed
# Test #    9 has been passed
# Test #   10 has been passed
# Test #   11 has been passed
# Test #   12 has been passed
# Test #   13 has been passed
# Test #   14 has been passed
# Test #   15 has been passed
# Test #   16 has been passed
# Test #   17 has been passed
# Test #   18 has been passed
# Test #   19 has been passed
# Test #   20 has been passed
# Finish:         23 cycle,          0 hit          0 miss
# Success.
# ** Note: $finish    : C:/Users/jhj/Desktop/Lab6_Á¤ÈñÁø_20170616/TB_RISCV_inst.v(201)
#    Time: 345 ns  Iteration: 1  Instance: /TB_RISCV
```

data memory에 접근하지 않았다.

2) TB_RISCV_forloop.v,

```
# Finish:        133 cycle         26 hit          8 miss
# Success.
# ** Note: $finish    : C:/Users/jhj/Desktop/Lab6_Á¤ÈñÁø_20170616/TB_RISCV_forloop.v(193)
#    Time: 1445 ns  Iteration: 1  Instance: /TB_RISCV
```

hit rate = 76.5%

3) TB_RISCV_sort.v

```
# Finish:      19800 cycle       4445 hit       1268 miss
# Success.
# ** Note: $finish    : C:/Users/jhj/Desktop/Lab6_Á¤ÈñÁø_20170616/TB_RISCV_sort.v(219)
#    Time: 198115 ns  Iteration: 1  Instance: /TB_RISCV
```

hit rate = 77.8%

5. Discussion

이번 랩에서 만든 cache는 miss가 날 때는 5 clock, hit일 때는 1 clock이 걸리도록 설계하였다. 만약 CPU와 memory간의 거리를 생각하면서 clock을 더 늘일 수도 있을 것이다. 그렇다면 결과에 보여지는 cycle의 크기는 더욱 커질 것이다. 이번 랩은 저번 랩보다 쉽게 작성할 수 있었다. 다만 저번 랩에서 작성한 파일을 이용했는데, pipeline register의 개수가 많아 모두 고려해야 된 점이 어려웠다. 그 외로는 개념적인 부분에서의 어려움이 있었지만 개념을 이해한 후에 코드로 작성하는 것은 어렵지 않았다.

6. Conclusion

모든 testcase를 잘 통과한 것으로 보아 의도한 대로 잘 설계된 것으로 보인다. 랩 코드를 작성하니 수업시간 때 배웠던 내용들을 좀더 잘 이해할 수 있게 되었다.