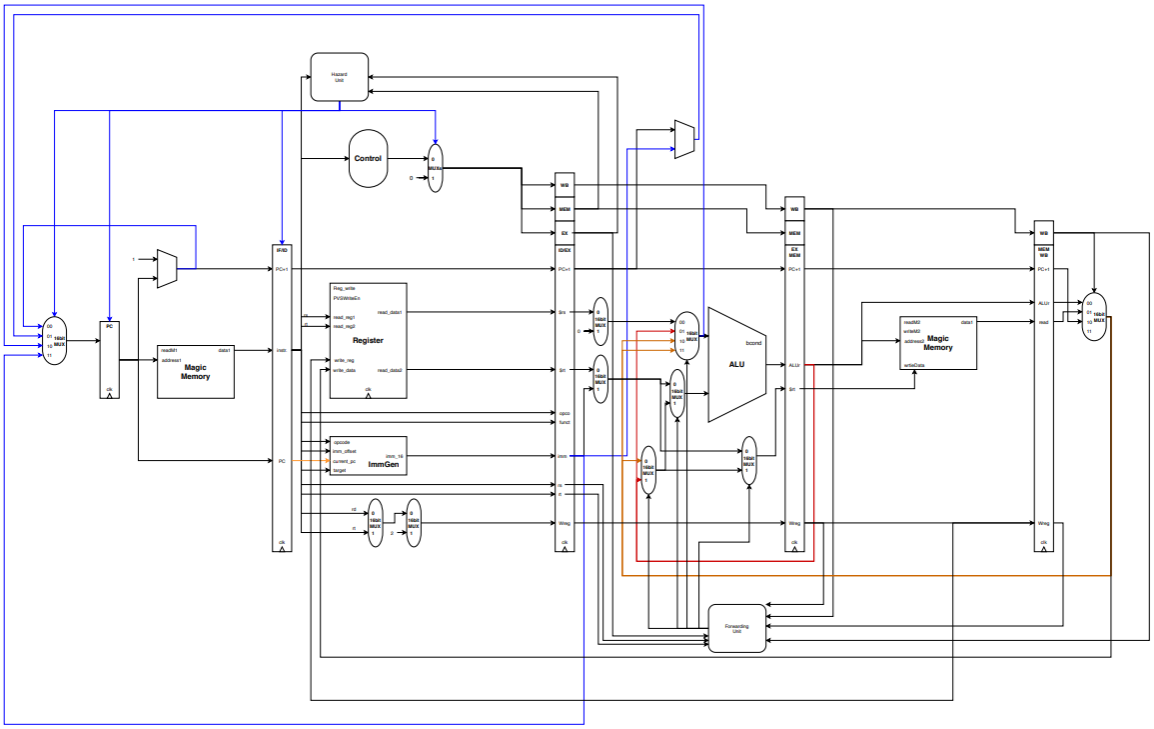
LAB 6. Cache

201800038 박형규, 20180480 성창환

1. Introduction

이번 Lab에서는 이전 Lab에서 구현한 TSC Pipelined CPU에 2-way set associative, single level cache를 추가시켜 Instruction Memory와 Data Memory에 접근하기 전에 Cache를 확인하도록 하는 CPU와 Baselined CPU로 이전 Lab과는 다르게 모든 Memory에 접근하는데 2cycle이 걸리도록 하는 CPU를 설계하도록 할 것이다.

2. Design



기본적인 CPU의 Design은 이전 Lab에서 구현했던 Pipelined CPU와 동일하다. 여기서 총 Capacity가 32words인 2-way set associative를 추가하여 Cached CPU를 구현하였다. Cache는 Instruction Cache와 Data Cache를 나누어 구현하였으며 이 두 Cache를 통합하여 관리하는 모듈은 cache.v의 cache 모듈이다.

3. Implementation

3.1 no cached CPU

No Cached CPU에서는 이전 Lab에서 구현한 Pipelined CPU에서 memory에 접근하는 latency를 2cycle로 늘려주었다. 이를 구현하기 위하여 먼저 MUX16_pc라는 모듈을 새롭게 만들었다. 이는 IFstate를 받아와 IFstate가 1일때는 일반 MUX와 동일하게 작동하고, 0일때는 새로운 pc값을 받아오는 것이 아니라 현재 pc값을 다시 한번 출력해주는 형태이다. 이는 IF 에서 Instruction Memory에 접근하는 것이 2Cycle 걸리는 것을 구현하기 위하여 IFstate이라는 것을 사용한데 있다. 그 뒤에 IFandID module에서는 위의 MUX16_pc에서 사용한 IFstate signal을 만들어준다. IFstate는 0이나 1의 값을 갖게 되는데, 처음에는 0의 값을 가지다가 1cycle이 지난 뒤에 1로 바뀌어 1일 때 정상적인 IFandID module의 기능을 수행하게 된다. 이와 비슷하게 MEMandWB module에서는 MEMstate signal이 존재하는데, MEMstate signal이 0의 값을 가질 경우 한 cycle동안 멈춰있게 되고, 1인 경우 정상적인 MEMandWB module의 기능을 수행하게 된다. 또한 forwarding을 구현함에 있어서 EX 단계에서 MEM 단계의 data를 받아와 ALU 연산을 진행하는 경우에 IF에서 2cycle을 소모하게 되기 때문에 ALU 연산이 중복되어 일어나는 것을 방지하기 위하여 EXandMEM module 에서 EXisValidInstruction일 경우에만 MEMALUresult 를 업데이트 해주었다.

3.2 cached CPU

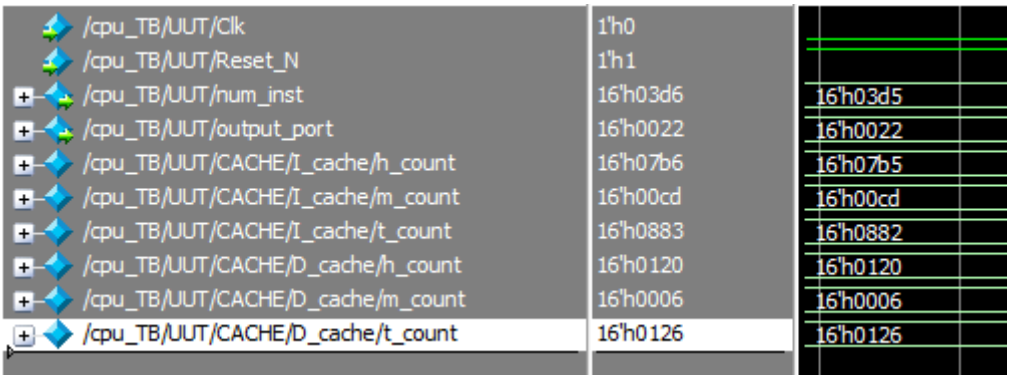
Cached CPU에서는 32words capacity인 2-way set associative cache를 구현하기 위하여 I-cache와 D-cache를 나누어구현하였으며, 2-way set associative 이므로 각각의 Cache가 2개의 set을 가지며 각 set은 8 words capacity, 4 words line을 갖도록 구현하였다.

먼저 I-cache에서는 instruction_address를 받아와서 tag, idx, offset으로 파싱한다. 그리고 tag와 I_cache[idx]의 tag 부분이 일치하고, I_cache[idx]부분의 valid bit가 1이라면 hit으로 나타내었고, 이를 각 set에 대하여 hit1과 hit2로 나타내었다. 만약 hit이라면 1cycle만에 I_cached[idx]에 있는 data를 cached_data로 옮기고 LRU bit를 update해준다. 그 뒤에 h_count도 1 올려준다. 만약 miss라면 6_cycle을 이용하여 memory의 정보를 cache에 올리고 m_count를 update해준다.

다음으로 D-cache에서는 I-cache와 유사하지만 MEM 단계에서는 write가 일어날 수 있으므로 write policy를 규정해야하는데, 이번 랩에서는 write no allocate방식을 사용해서 쓰기명령일 경우 무조건 메모리에 쓰고, miss이면 불러오지않고 hit일때만 캐시에 값을 업데이트하도록 구현했다.

4. Discussion

Cache를 이용하여 구현한 CPU와 memory 접근에 2 cycle이 걸리는 no cached cpu의 성능에 대하여 분석해보도록 하겠다.



총 2472번의 메모리 access가 있었고, 2261번의 hit가 일어나서 0.91의 hit rate를 보여준다.

총 사이클은 3200사이클이 걸렸다.

```
# Clock # 2413
# The testbench is finished. Summarizing...
# All Pass!
# ** Note: $finish    : C:/Users/hyeongkyu/Desktop/POSTECH/3ÇÐ³â/ÄÄÇ»ÅÍ±¸Á¶/Lab6/Lab6_4_20180038_20180480/nocachecpu/cpu_TB.v(153)
#    Time: 241450 ns  Iteration: 2  Instance: /cpu_TB
```

nocache의 경우 총 2413사이클이 걸렸다.

생각보다 꽤 차이가 많이 났는데 이 이유는 cache의경우 ex stage에서 올바른 pc값이 나오고, 또 wrtie의 경우 무조건 6사이클동안 대기하도록 구현했기 때문에 cache의 이점을 잘 못 살린듯 하다.

5. Conclusion

이번 랩을 통해서 no cache pipelined cpu와 cached pipelined cpu를 성공적으로 구현했다. Cache를 사용하는 이유는 memory에 직접 접근하는 것보다 낮은 cost로 data에 접근할 수 있게 하기 위해서이다. 하지만 cache는 miss가 생기는 경우 cache에 한 번 접근한 뒤에 memory에서 데이터를 가져와야 하기 때문에 cache가 없을 때보다 오히려 더 많은 cycle을 소모하게 된다. 하지만 hit일 경우에는 cost를 줄일 수 있고, locality에 따라 근처 address에 접근하거나, 한 번 접근한 address에 접근할 확률이 높기 때문에 hit이 많이 생길 수 있어 cost를 줄일 수 있기 때문에 사용한다. 이번 랩에서는 이러한 cache를 직접 구현하여 보면서 Pipelined CPU에 Cache를 추가했을 때의 장점과 단점에 대하여 알 수 있었다.