Computer Organization Assignment #3

Caches

Due : 23:59, Friday, Dec. 7, 2012

I. Introduction

Cache와 Main memory로 구성 된 Memory Hierarchy 구조를 구현하고 성능을 비교한다. 이번 프로젝트에서는 single-cycle MIPS에 cache를 포함한 memory hierarchy 구조를 구현한다. 시뮬레이션만으로 실제 cache와 main memory와 유사한 환경으로 실험할 수 있도록 모델링 된 memory 구조를 사용한다.

II. Theory

실제 SOC(System On Chip) 등 모든 하드웨어 아키텍쳐 설계 시 가장 문제가 되는 것이 바로 DRAM이다. 성능을 높이기 위해서는 첫 째 DRAM에는 되도록 접근을 하지 않도록 하는 것이 중요하고, 꼭 해야 한다면 적은 비용으로 최대한 속도를 덜 느리게 하는 방법을 사용한다. 그런 의미에서 비싸지만 속도가 빠른 cache를 locality를 고려하여 사용하는 것이다. 이번 프로젝트에서는 실제 cache나 main memory와는 다를 수 밖에 없지만 되도록 유사한 딜레이를 갖도록 모델링 된 디자인을 다루게 된다. 예를 들어 read에서 cache hit이면 2cycle 걸리고, cache miss 이면 5cycle이 걸리는 등 임의로 모델링 한 것이다. 그렇다면 우리는 메모리 시스템이 어떻게 모델링 되었는지 cache protocol에 대해 알아야 설계를 할 수 있다.

아래의 1~2를 모델링하고자 하는 캐시의 스펙이라고 가정하고, 그 동작과 동일한 동작을 시뮬레이션하는 캐시 시뮬레이터를 설계한다. assignment#2에 자신이 설계한 디자인에 아래의 캐시 스펙과 동일한 캐시와 컨트롤러를 모델링하여 성능을 비교한다. 스펙에 사용 된 시그날을 그대로 사용 할 필요는 없으며, 같은 동작을 하도록 모델링 하는 것만 목표로 한다. C로 모델링하는 만큼 구현하는 방법은 다양하다.

1. cache write signal protocol

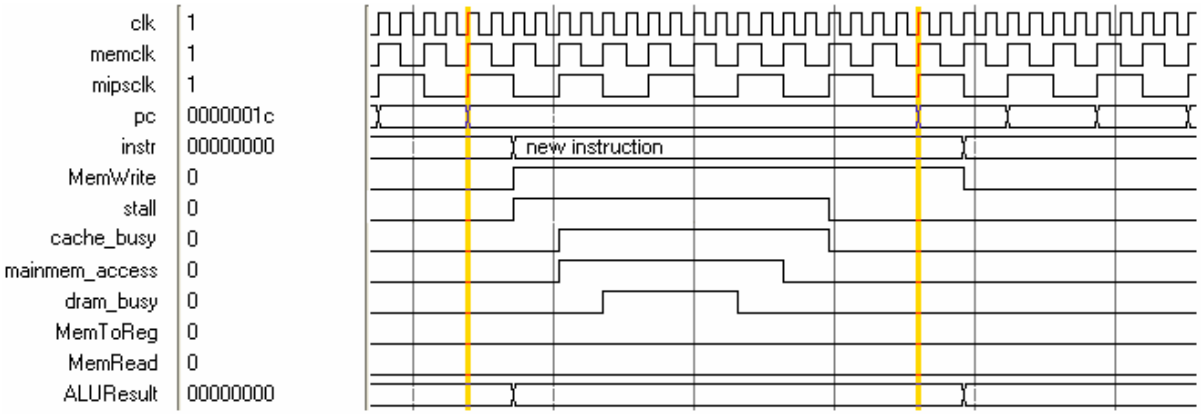

Figure 1cache write

본 cache는 write-through cache로 cache에 입력 시 항상 main memory에도 동시에 write를 해 줘야 한다.(Write-back cache가 훨씬 널리 사용 되지만 Write back을 사용하면 그에 상응하는 또 다른 cache protocol을 알아야 하기 때문에 훨씬 복잡해 진다)

1) positive mipsclk에 pc값이 바뀐다.
2) 1 memclk 후에 positive memclk에 의해 IF stage에서 new instruction이 fetch 된다. 이 때 들어오는 new instruction은 sw와 같은 memory write 동작을 수행하는 instruction이라고 가정한다.
3) 동시에 MemWrite가 1로 바뀐다.
4) MemWrite가 1로 바뀌면서 stall 또한 1로 바뀐다. 이는 stall인 동안 새로운 instruction이 수행되지 못하도록 한다.
5) MemWrite는 cache module의 input이고, 이 MemWrite이 1이 되는 순간부터 1 memclk 후에 cache_busy와 mainmem_access가 1로 assert 된다.
6) mainmem_access signal은 main memory에 access 하겠다는 신호이다(read, write 모두에 해당)
7) main memory는 mainmem_access 신호를 감지하여 이 신호가 1이 되었을 때로부터 1memclk 후에 dram_busy를 1로 바꾼다.
8) dram_busy signal은 3 memclk으로 모델링 되어있다. 메모리에 데이터를 쓰는 데 3 memclk으로 잡았다고 보면 된다.
9) dram_busy가 3cycle 유지되고 데이터를 다 쓰고 나면 0으로 떨어지는데, 이 때로부터 1 memclk 이후에 mainmem_access가 0으로 바뀐다. cache_busy는 필요에 따라 좀 더 1을 유지하다 0으로 바뀌면 된다.
10) 이후에는 메모리 쓰기가 끝났으니 pc값이 바뀌면서 다음 instruction이 fetch되고 정상적으로 동작을 수행한다.

2. Cache read signal protocol

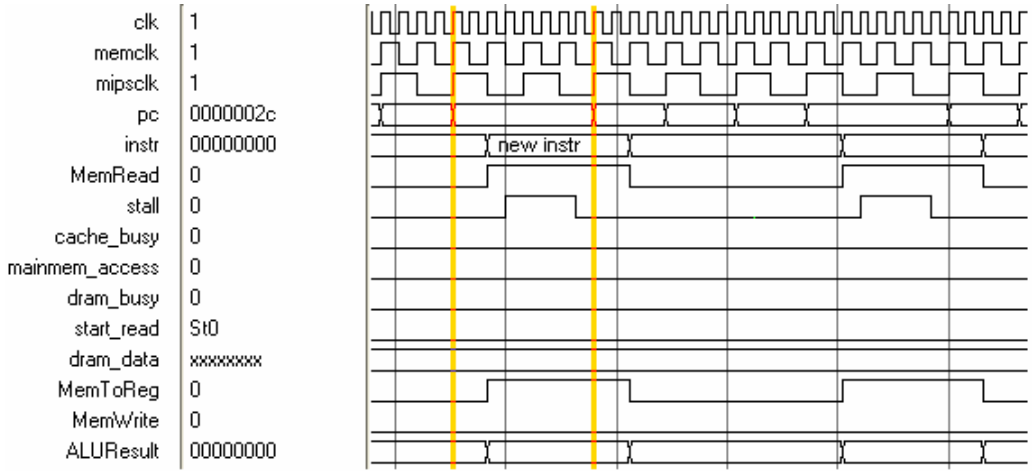

Figure 2Cache read hit

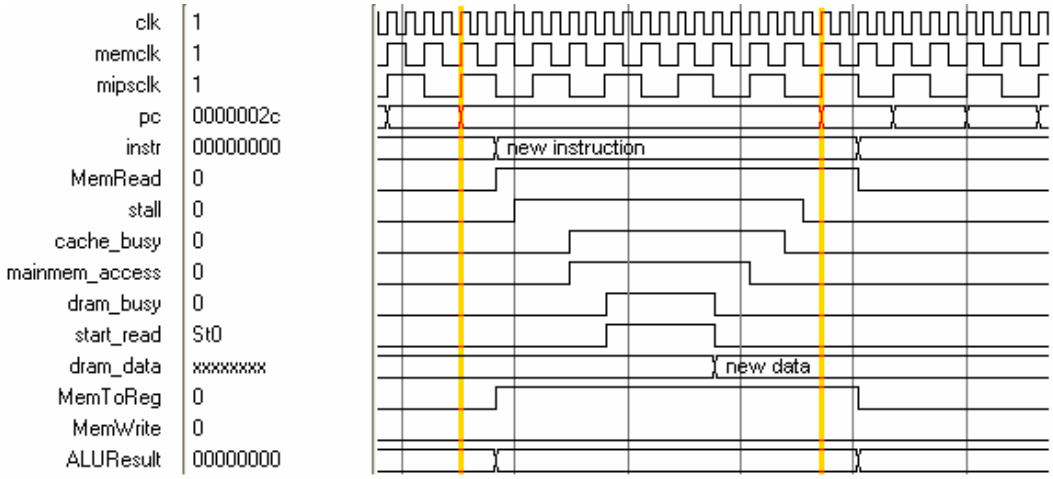

Figure 3Cache read miss with 1-word block

1) positive mipsclk에 pc값이 바뀐다. 1 memclk 후에 positive memclk에 의해 IF stage에서 new instruction이 fetch 된다. 이 때 들어오는 new instruction은 lw와 같은 memory read 동작을 수행하는 instruction이라고 가정한다.
2) instruction이 fetch 되고 1memclk 후에 MemRead signal이 1로 바뀐다.
3) 그리고 1memclk 후의 positive edge에서 stall이 1인 상태로 있게 되고, cache read miss냐 cache read hit이냐에 따라 cache_busy를 0으로 유지할 것인지 1로 바꿀 것인지에 대해 결정해야 한다.
4) **[cache hit인 경우 : Figure 2]** 설계한 cache의 cache_busy signal이 low(0)를 유지해야 한다. MemStage module에서는 이를 감지하여 stall을 0으로 바꾼다. 즉, read operation에서 이 stall이 적어도 한 번의 mipsclk을 지연시키므로 read hit이라 해도 2 mipsclk이 소모된다.
5) **[cache miss인 경우 : Figure 3]** cache miss인 경우에는 data를 main memory에 fetch 해야 한다. 우선 cache hit과 다르게 MemRead가 1이 되는 시점에서 2memclk 이후에 cache_busy signal을

high로 바꾸고,mainmem_access signal 또한 1로 바꾼다. stall은 cache_busy가 low가 되고 1memclk 지날 때 까지 high를 유지하다 low로 바뀐다.

6) 만약 main memory가 access 1번에 1개의 word만을 얻도록 설계하고자 한다면 cache module의 **read_block** signal을 0으로 만들어야 한다. 첫 번째 word가 start_read signal의 negative edge 이후로 1memclk 다음에 read 된다. 이는 dram_busy가 high 가 되고 나서 3 memclk 이후의 일이다. 단순히 1 word만 읽으면 되므로 dram_busy signal은 start read와 동시에 low가 된다. mainmem_access는 dram_busy가 low가 되고 나서 1 memclk 후에 0이 된다. cache_busy signal은 필요에 따라 dram_busy가 low가 된 후에 적절한 시간을 high인 상태로 유지하면 된다.

7) **[figure4 read miss with 4-word block]** 만약에 main memory가 access 1번에 4개의 word를 얻도록 설계하고자 한다면 read_block_signal을 1로 바꿔야 한다. 첫 번째 word가 start_read signal의 negative edge인 때 ~~로부터 1 memclk 이후에~~ 읽어진다. 이는 dram_busy가 high가 되고 난 후로 3memclk 이후의 일이다. 남은 3개의 word는 memclk의 positive edge 마다 읽어진다. 특히 마지막 word는 dram_busy가 low가 되면서 동시에 읽어진다. mainmem_access signal은 dram_busy가 low가 되고 난 후로 1memclk 이후에 low가 된다. cache_busy는 필요에 따라 적당히 high를 유지하도록 한다.

8) cache_busy가 low가 되면 MemStage module에서 감지되어 stall signal 또한 동시에 low가 된다.

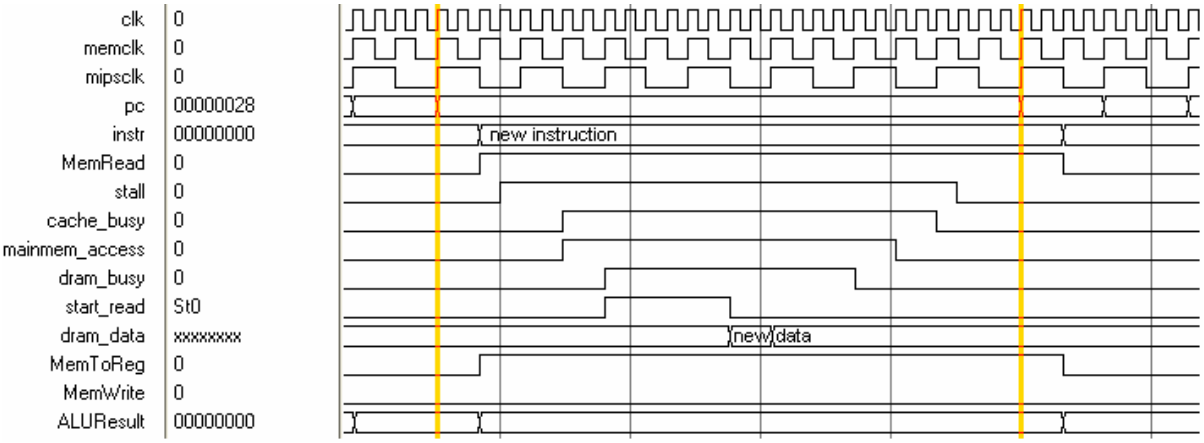

Figure 4Cache read miss with 4-word block

III. Requirements

1. cache protocol 파악하기
2. cache 설계 (set associativity, replacement policy 결정)
3. 성능 측정 및 분석

IV. Implementation Details

1. cache 설계 (set associativity, replacement policy 결정)

설계하는 cache는 다음과 같은 동작이 가능해야 한다.

- **Read hit** - cache에서 읽어온 데이터를 CPU에 전달함
- **Read miss** – main memory에서 block에 해당하는 데이터를 읽어와 cache에 fetch하고 CPU에 전달함
- **Write hit** – cache block에 데이터를 쓰고 동시에 main memory에도 쓴다(Write through)
- **Write miss** – 만약 설계하고자 하는 cache의 block size가 1 word 라면 cache와 main memory에 동시에 write 해야 한다. 만약 cache block size가 1 word보다 크다면 main memory에만 데이터를 업데이트 한다.

만약 cache block이 1 word 보다 크다면 읽어올 때 연속적인 4개의 주소에서 값을 읽어 block에 저장하는 식이다.

※ **제한사항**

- Cache의 총 size는 **1kB**를 넘어서는 안 된다. [valid bits, tags, data 모두 합쳐]
- main memory에 read나 write할 때는 당연히 word align되어야 한다.
  [몇 개의 word를 동시에 읽고 쓰더라도 이는 word 단위로 다루어야 한다.]
- Main memory의 size는 256kB가 최대라고 가정 한다. 즉, 16bit 만으로 main memory의 모든 entry를 addressing 할 수 있다.(256kB = 2^18B) 이러한 사실을 이용하면 tag의 길이를 줄일 수 있다.

Requirement #1

1. 본인이 설계 할 cache의 구체적인 명세를 정한다. 즉, direct, set associativity 중 어떤 것을 사용할 것이며 set의 크기는 어떻게 정할 것인가? 그리고 set associativity를 사용한다면 replacement policy는 어떻게 정할 것인가? 에 대한 내용을 정한다.
   [여러 개를 따로 구현하면 더 좋다. 성능 비교를 위해]
2. Cache를 설계하여 메모리 접근 instruction으로 구성 된 imem_data.txt가 제대로 동작하는지 simulation을 통해 검증한다.

※ 1 word block에 direct-mapped cache가 가장 구현하기 쉽지만 이것만 구현하면 좋은 평가를 받을 수 없다. 하지만 set associativity를 사용하더라도 올바로 동작하지 않으면 더 나쁜 평가를 받게 됨.

2. 성능 측정 및 분석

Assignment#2에서 사용했던 예제인 bubble sort 예제를 똑같이 사용 한다. total_cycles, total_instr, total_writes, total_reads 등 성능 측정을 위한 데이터를 얻을 수 있는 변수들을 추가한다. 이 4가지 값 외에도 코드를 응용하여 필요하다고 생각되는 데이터를 만들어서 자신이 설계한 cache의 성능이 어떻다 라는 것을 증명하여 나름의 결론을 내린다.

HINT : hit/miss를 따로 count하여 miss rate, miss penalty 등의 개념을 도입해야 할 것이다. 정석적인 방법은 교과서 5장을 참조한다.

Requirement #2

bubble sort 프로그램을 사용하여 본인이 설계 한 cache의 성능을 측정한다. 성능을 측정하는 방법에는 제한이 없으며 최대한 교과서에서 배운 방법을 사용하는 것이 좋다. 만약 2가지 이상의 cache를 따로 설계하여 성능을 비교한 데이터를 제시하면 높은 점수를 받는다. 기본적으로 cache가 동작하지 않을 때와 cache를 사용했을 때의 성능 비교는 반드시 들어가야 한다.

V. Submission

제출과제
1. 보고서(이메일과 hardcopy 모두 제출)
① Requirement#1 ② Requirement#2
2. 본인이 설계한 ISS 프로젝트와 출력 파일을 압축하여 이메일 제출

제출방법
1. 조교 이메일 modesty@sdgroup.snu.ac.kr 로 보고서, 코드파일을 압축하여 제출. 2. 보고서 Hardcopy는 12월 8일(토) 기말고사 전에 제출
※ **12/7일 금요일 저녁 11시 59분 까지 제출분만 인정(메일 도착 시간 기준)** **(지각 제출은 받지 않겠습니다)**

※ 한 학기 동안 수고 많으셨습니다.

조교 김지천 드림.