Computer Organization Report

Lab07 cache

2019-12172 이재원

1 Introduction

Understand the structure of cache and measure the performance improvement of cache.

2 Design

2.1 Baseline – without cache

2.1.1 NOP

In this lab, adding ports like ready signals to memory is not allowed. Therefore, without using ready signals in memory modules there are several ways to design cpu model.

First, adding a counter to the processor can be the solution to this problem. However, this way assumes that the processor knows the delay of the memory. Also, if the delay of memory changes, the processor wouldn't work correctly.

Because of these problems, in this lab, the memory module is designed to print NOP when delayed. This can be interpreted in real world that if there is no signal from the memory when read, nothing happens (NOP).

2.1.2 Stall

When there is no signal (NOP) from the memory when read, the processor needs to stop i.e. stall. If delay occurs at instruction memory, pc to IF should be stalled and for data memory access, stalls should be done at MEM/WB register.

2.1.3 Latency model

Therefore, there is counter at memory modules to design delay. However, for write signal there is a counter in the processor at hazard control units.

2.2 Cache

2.2.1 Separated cache, write through / no allocate

2.2.2 Latency model

2.2.2.1 Read hit

Cache (1cycle)

2.2.2.2 Read miss

Cache (1cycle) + memory_read(4cycle) + cache(1cycle) = 6cycles

2.2.2.3 Write hit

Cache (1cycle) + memory_write(4cycle) + cache(1cycle) = 6cycles

Need to access to memory because of write through.

2.2.2.4 Write miss

Cache(1cycle) + memory_read(4cycle) + cache(1cycle) + memory_write(4cycle) + cache(1cylcle) = 11 cycles

Need to read memory because memory is accessed based on lines. If you want to write one word, you need to read one line (4 words) and write the word, and write the changed line to memory.

2.2.3 Replacement policy & conflict

In this lab, it is replaced by the index of input address. Therefore, there is no case of conflict because if the index is same it is replaced.

2.2.4 Processor

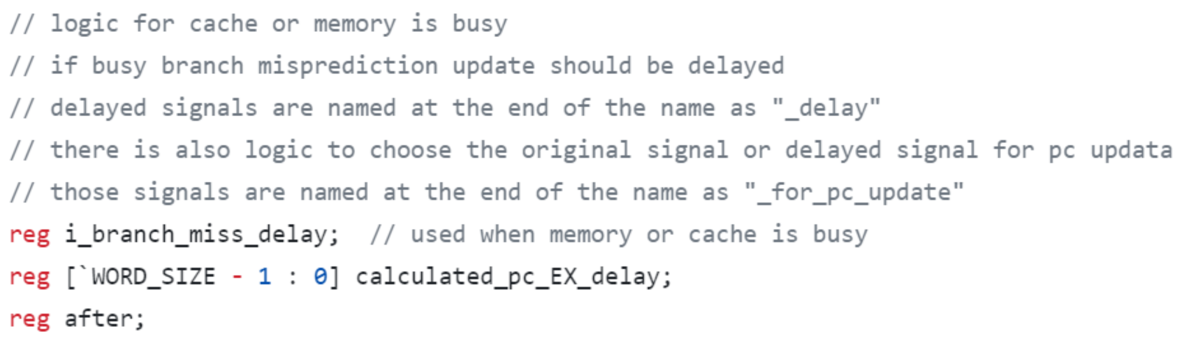
As baseline (without cache) assumes that the processor doesn't know the latency of memory or cache there is no change to cpu unit. However, instead of counters used for write in the datapath, writeDone signal is printed by cache.

3 Implementation

3.1 Processor

3.1.1 Branch misprediction update

In datapath, you should not update pc as soon as branch misprediction is detected because memory or cache can be busy. Therefore, logic is added for updating pc when not busy both to baseline and cache.



```
// logic for cache or memory is busy
// if busy branch misprediction update should be delayed
// delayed signals are named at the end of the name as "_delay"
// there is also logic to choose the original signal or delayed signal for pc updata
// those signals are named at the end of the name as "_for_pc_update"
reg i_branch_miss_delay;  // used when memory or cache is busy
reg [`WORD_SIZE - 1 : 0] calculated_pc_EX_delay;
reg after;
```

Figure 1. Code for branch misprediction in *datapath.v*

3.2 Hazard

3.2.1 Instruction stall

Pc is not updated when delayed.

3.2.2 MEM / WB stall

For memory read operations it is stalled when delay.

As counter is in datapath of baseline (without cache) write operations stall is determined by counter.

In cache there is writeDone signal that tells the processor that after writeDone no need to stall.

3.3 Cache

3.3.1 Structure

4 line, 4 word wide cache. As line number is 4, index is log(4) = 2 bits. As block size is 4, block offset is log(4) = 2bits. Therefore, tag is 16 - 2 - 2 = 12 bits.

From this the capacity of data bank and tag bank is determined. In the code, data bank and tag bank are declared as register and is updated at posedge clk. The update logic is write through, no allocate.

The output ports are wires so it is in always (*) block.

3.4 Waveform

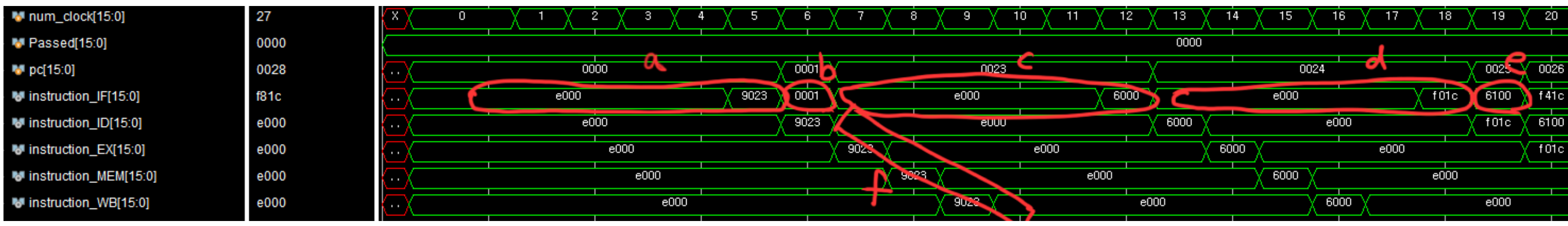


Figure 2. Instruction cache read hit (b, e) & miss (a, c, d). f is the part when it is flushed in the pipeline registers due to flush at branch misprediction.

Instruction cache can be found at IF stage. e000 in the waveform means NOP. Read miss is Cache (1cycle) + memory_read(4cycle) + cache(1cycle) = 6 cycles long. Read hit takes 1 cycle. As there is no case of instruction write, there is no waveform of instruction cache hit / miss.

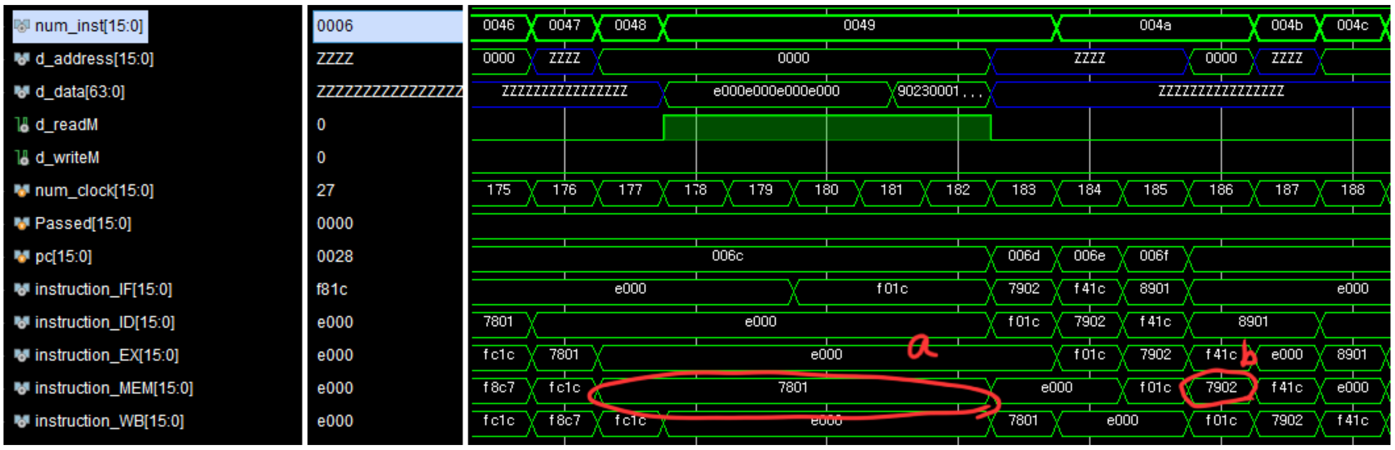


Figure 3. Data cache read miss (a), read hit(b).

Just like instruction read, data cache read takes same cycle. This can be checked by looking at instruction at MEM stage.

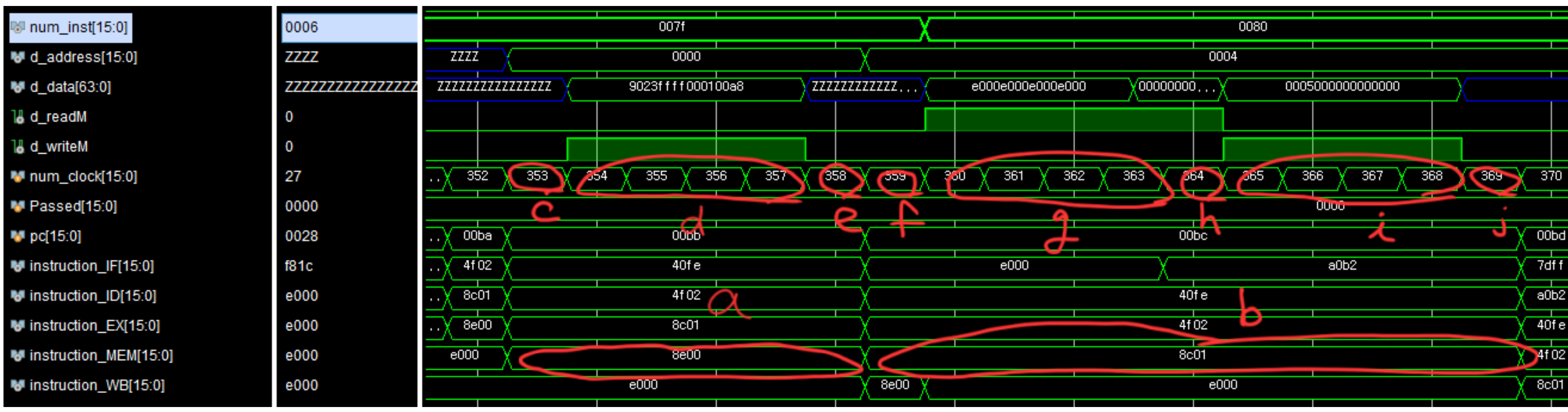


Figure 4. Data cache write hit (a) and write miss (b). c, d, e, f, g, h, i, j, shows cycle of each step.

Cache write hit (a) is cache *1cycle (c) + memory_write 4cycle (d) + cache 1cylce (e)*.

Cache write miss (b) is constituted of cache *1cycle (f) + memory_read 4cycle (g) + cache 1cylce (h) + memory_read 4cycle (i) + cache 1cylce (j)*.

4 Discussion

4.1 Hit /miss

4.1.1 Code

Can be checked by looking at the value of *num_cache_miss, num_cache_hit* in *cache.v* module. It is updated at the first cycle of cache.

4.1.2 Result

	hit	*miss*	*total*	*hit rate*
instruction cache	4113	172	4285	95.99%
data cache	493	9	502	98.21%
total	4606	181	4787	96.22%

Note that sum of number of hits and misses are 4285, where total number of instructions are 3036. This is because there are instructions fetched through cache but not counted at the end of pipeline due to branch misprediction. Total number of instructions are counted at the end of the pipeline, whereas hit or miss is counted at the start of pipeline. One example is Figure 2. (b) , where instruction '0001' is fetched, but flushed due to jump misprediction. For one jump misprediction, 1 or 2 instructions that will be flushed are fetched depending on whether cache is busy or not. Likewise, in case of one I-type branch misprediction, 2 or 3 instructions are fetched that are not counted at number of instructions.

4.2 Performance

	total instructions	*cycles*	*IPC*
baseline	3036	7088	0.42833
cache		5110	0.594129

Cpu with cache is 0.594/0.428 -1 = 38.71% faster than baseline (without cache).

As hit rate of cache is more than 95%, there was a dramatic increase in performance.

5 Conclusion

Implemented separated write through no allocate cache, and calculated hit ratio and performance improvement.