Homework 5 Report

李京叡, 0616329

***Abstract*—This electronic document is the homework 5 report for the course—Microprocessor Systems: Principles and Implementation at NCTU in 2020 fall.**

I. Introduction

A cache plays an important role in modern processors because it can affect the processor's performance significantly. Since reading data from memory takes much more clock cycles, it is better to create a mechanism to mediate between memory and processor and this is the purpose of inventing a cache. The idea of cache memory is that some active portion of a low-speed memory is stored in duplicate in a higher-speed cache memory. When a memory request is generated, the request is first presented to the cache memory, and if the cache cannot meet the request, the request is the presented to main memory. Since cache memory is much faster than main memory (I have done some experiments in the following to evaluate the number of clock cycles spent by each of them), if there are more hits and lesser misses in cache, the performance of the processor should be improved. In this assignment, I focus on analyzing and improving the performance of I-cache.

II. Method to Decreasing Miss Rate

There are many ways to improving the cache memory, including both hardware and software approaches. For hardware approaches, the first method is to enlarge the block size. This approach makes a block contains more surrounding data, more hits per block for code with high locality. However, too large block size reduces the number of blocks in a given cache size, which may increase misses. The second method is to make a higher associativity, which helps reduce cache entries being discarded by conflict. However, this method may increase the time to check whether the request is hit or miss. Another approach is hardware prefetch. The cache fetches extra blocks on a miss. This idea is under the assumption that subsequent blocks will be referenced soon. Another two ways are in terms of software. The first one is to prefetch by software. This is done by compiler instructing the processor to preload data into a register or load into cache. The other method to improve miss rate is by compiler optimization. With optimization, compiler generates less code, which means fewer instruction misses and better data memory organization can also contribute to fewer data misses.

III. Analysis of I-Cache

To analyze I-cache, I insert counters to count the number of hit instructions and miss instructions in order to evaluate the miss rate. The counter starts at the point the input address is different from the previous one and stops at the time p_ready_o is set to high. Originally, I set the starting point at the point while p_strobe_i is rising, but it is not practical, since p_strobe_i could be at the high level for many consecutive clock cycles if the instructions hit for a few instructions in a row. It's hard to determine which point is the starting point of an instruction request, so I change the counter's starting point at the time the input instruction address is changed. Since the resource on the FPGA board is limited, I simply decrease the D-cache to 2 KB in order to put the counters and ILA debug core into the board and have the space to change the size of I-cache. I also change the degree of associative of the cache to compare the performance of each scheme. The following table shows the miss instruction count under different conditions. I found that the miss rate is too small (about 10^{-4} %), so I just show the miss instruction count in the table. The number of miss instructions is counted until the program counter arrives the instruction "ret" in main function.

i-cache size	miss	associative	Dhrystone
4KB	1117	4 way	0.51
2KB	1784	4 way	0.51
4KB	1106	8 way	0.51
2KB	1806	8 way	0.51
4KB	1353	1 way	0.51
2KB	6002273	1 way	0.44

From my test, for 4KB-size cache, the number of misses for 8 way and 4 way associative cache is almost same. 8 way associative cache is slightly better than 4 way associative cache. Direct-mapped cache has the worst performance, however, the difference between it and other two scheme is not very big. In fact, the difference between them is under second decimal place. For 2KB-size cache, it is apparent that the number of misses is much more than 4 KB-size cache. Especially for direct-mapped cache, it misses 6002273 instructions. One thing worth noticing is that 8 ways associative cache doesn't perform better than 4 way associative cache. Since the size of the cache is fixed, more ways causes less lines. Though more ways contribute to less conflicts, it also decreases the number of lines in the cache. For Dhrystone, the result is pretty much same as misses. 2KB-size direct-mapped cache performs the worst in Dhrystone, it is only 0.44 DMIPS/Mhz. Other schemes perform about 0.51 DMIPS/MHz in Dhrystone program.

I also create another two counters to count the clock cycles an instruction request needs to compare the condition of miss and hit respectively. From my observation, the processor spend within 3 clock cycles if the instruction hits, however, if it fetches the instruction from the memory when instruction is miss, the counter can count up to 27 clock cycles. It's a large gap between them, and this is why decreasing miss rate helps improving a processor's performance a lot.

IV. STREAM BUFFER

The way I try to implement to improve the cache performance is by adding an extra stream buffer. This is the method I found on the internet. The idea is proposed by Norman P. Jouppi.[1]. In this design, the processor maintains a buffer to do the prefetch. When a miss occurs, the stream buffer prefetches a few successive lines starting from the miss target. Every access to the cache would compare its address against to the entries in the caches and the first entry of the buffer as well. The buffer uses FIFO scheme, so the cache can access only the first entry of the buffer. If the access misses in the cache but hits in the buffer, the cache can be reloaded in a single cycle from the stream buffer. This is much faster than reloading the cache from the main memory since stream buffer is on-chip device. It can respond the request in a single cycle. I try following the logic and implement the design on the I-cache. Figure 1 shows schematic diagram of my design.

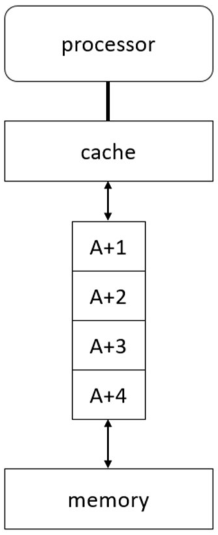

Figure 1

I create a buffer with a capacity of 4 lines and add another three states, which are "BufferCheck", "BufferUpdate", and "BufferRotate" to control the workflow. In the "Next" state, if the requested instruction is miss, it will not go to "RdFromMem" state directly. Instead, it goes to "BufferCheck" state first to check if the request hits in the buffer. If not, it goes to "RdFromMem" state to read instruction from the main memory. After "RdFromMem" state, if goes to "BufferUpdate" state to read 4 extra successive lines beginning from miss target into buffer. After these procedure has been done, it goes to "RdFromMemFinish" state and this is the end of one instruction request misses both in the cache and the buffer. On the other hand, if the request hits in the buffer, which means the first entry of the buffer contains the requested instruction, it goes to "BufferRotate" state. In "BufferRotate" state, all the entries in the buffer move forward for one entry and the buffer will read one following line from the memory. Then, it goes back to the "Next" state to deal with the next instruction request. I follow the logic of this design and implement this design and run Dhrystone program. The performance doesn't improve, even drops a lot. Dhrystone performance drops to 0.29 DMIPS/MHz, but I think it is reasonable. Since my design is serial, with stream buffer mechanism, there are more memory accesses than the original design. For example, in "BufferUpdate" state, we need to reload all content in the buffer and it needs to read data from main memory for 4 times. I use ILA to observe the waveform when it is running, and find that my design takes about 4 times more clock cycles when the instruction misses both in the cache and the buffer. Also, even if the instruction hits in the buffer, it takes time to load another one lines from main memory to fill the buffer. I think a better implementation is to revise the memory and broaden the data bandwidth of the memory making it able to output 4 lines in a time or design the procedures into pipeline.

V. COURSE REVIEW

After taking this course, it seems to me that this course is an advanced course of Computer Organization. There are many domain knowledge required in designing a processor. When I do the homework, sometimes, I need to review the content of Computer Organization, but I think this is a good chance to review the knowledge I have learned before. Besides, in Computer Organization, I write Verilog to simulate the behavior of a processor. In this course, I implement the design onto FPGA board. In this course, I also learn many techniques to analyze a circuit. In Digital Circuit Lab, I just dedicate to write the Verilog code that can run normally and meet the homework requirements. I never had a deeper look to how my code work on FPGAs. I think ILA is a handy tool. I only could write a testbench and run simulation before. ILA provides another way to debug my design. In this course, I also practice English writing a lot. At the first glance of report template provided by teacher, I thought it was just like writing a thesis. I haven't read many theses, not to mention writing a thesis. It seemed to be a difficult task to me. I just try to deliver my thoughts and works I have done in English as clearly as possible in every report. Maybe my writing is still not formal, but I think at least during this semester, my English writing has improved a lot.

VI. REFERENCE

[1] N. P. Jouppi, "Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers," [1990] Proceedings. The 17th Annual International Symposium on Computer Architecture, Seattle, WA, USA, 1990,pp.364-373, doi: 10.1109/ISCA.1990.134547.