Computer Architecture and Mobile Processor

Project 4 – Simple Pipelined MIPS With Cache



Student Name : Jason Melvin Tedjokusumo

Student ID : 32205061

Major : SW 융합모바일시스템공학과

Professor : 유시환교수님

Submission Date : June 20, 2022

Free Days Left : 0 days

* Introduction

This project is the continuation of the 3rd project, to make a simple Pipelined MIPS emulator with the cache. The fact that all instructions and data are kept in memory makes Von Neumann computers exceedingly memory-intensive, and memory access latency is often a hundred times slower than register access latency. This project will add cache functionality to the pipeline project to increase the access time rate to memory and significantly boost the program execution time.

* Important Concept

Memory access for a CPU with a cache is checked first with the cache before the memory is accessed. Data is accessed directly from the cache rather than memory if it is in the cache (a cache hit). The CPU retrieves the data from memory if it is not in the cache (cache miss). Keep in mind that cache is located within the CPU so that the pipeline may access it rapidly.

We must establish a number of regulations for cache storage in order to make the most of it. Cache line, set associativity, and the total number of entries in the cache must all be defined first. Cache block (or cache line) comprises some neighboring bytes from the requested location to maximize spatial locality. At the same time, all of the data in a cache line is read from memory and written back to it. There are numerous cache lines that make up the total cache.

The tag is a an additional information that remembers the memory address, which is represented in some bits. There are various methods for mapping memory regions onto cache lines, including set-associative mapping, fully-associative mapping, and direct mapping. The cache line index may also be taken into account to minimize the size of tag bits. Several cache lines exist in a cache, and some address bits can be used to identify the cache line's index. For direct mapping, the cache line entry is based on the address, and the cache line index is fixed. The cache line index has no bearing on the address for fully associative mapping. In other words, it doesn't matter where the memory is located in the cache line. A hybrid cache-memory mapping called set-associative mapping groups certain cache lines into sets and computes the set index similarly to how direct mapping does.

Cache conflict miss happens when the cache is entirely utilized and another cache miss occurs. There is no more space for data storage because every cache line entry has already been used. In this instance, one of the entries (a cache line) must be removed from the cache in order to make room for fetching the desired memory region into a cache line. The cache must therefore decide which cache line needs to be replaced. Maximizing temporal locality is a sound justification for replacement entry. The least recently accessed entry in the cache is chosen using the LRU (Least Recently Used) method. In other words, LRU aims to keep in the cache the entries that are most likely to be accessed again.

The write policy must be decided upon last. The write operation can be used either on the cache line (write-back) or through the memory when a cache line is modified (Write-through). Data consistency between the memory and the cache is hampered when we write data just to the cache line. (For a single processor system, that is acceptable) Memory access speed is reflected in pipeline execution when data is written to memory via the cache.

* Unique Considerations for Implementation

Some adjustments are made using the code from the prior project. Two structs are added to the "header.h" file, and some variables are added to the already-existing struct. Two new functions that are used to implement the cache are added to the main file. Additionally, a few global variables for the cache are defined right away. The new struct types have a defined range when they are declared.

The main function begins at line 40. The first step is to declare local variables. The instruction memory stores the instructions from the binary input file, which is opened sequentially. This step is followed by closing the binary input file. Line 70 creates a while loop to iterate over each instruction. The code will be deciphered after each instruction has been read from memory. Following the execution of the instruction, the control logic is followed to access the memory. The output of the instructions is then written using the writeback function. The writeback function is called after the IF function in this code since it is related to the pipeline concept. After that, certain variables are assigned to the latch instructions. Next, the pc value is checked; if it is 0xfffffff, the system will terminate. Cycle and instruction count counters are continuously increased. The modified micro-architectural states are then printed. After the while loop is closed, the final output of the executed instructions is printed together with any additional information. Additional information includes the total amount of cycles, memory access count, register operation count, branch instruction count, and cache hit/miss percentage.

Under the main function, the IF function is established to get an instruction from the cache and set the pc value. The control signal's CS function is then established. Its goal is to set the value of the control signal in accordance with the opcode value using an if-else if statement. The decode stage's pc and instruction value of the latch are then updated using the IDLatchUpdate function.

The ID function is then built to decode each instruction according to its opcode value. The instruction is decoded to reveal the opcode and operands. An if statement is then created to extract the command's sign extended instantaneous value. The destination of the jump address is also determined in the following line of code. By using the IDLatchUpdate method, the decode latch is updated sequentially. The target of the j and jal instructions is initialized after this function using the if-else if statement, which executes in accordance with the instruction's opcode.

After the ID function is formed, the EXE function follows. First, variables v0 and v1 store the values of the v0 and v1 latches. The if-else if statement is then used to address the data dependency by forwarding. The index latch value is used to initialize the v0 and v1 variables after handling the data dependent. The instruction is then carried out in accordance with its control signal. During the execution process, the counter for a particular variable is increased. Latch variables are continually initialized one by one before moving on to the following step.

The MEM function is produced for memory access after the EX function. For load and store word instructions that call for access to the memory register, use this function. Based on the control signal, specific code is run, and the latch variables are initialized to move on to the next instruction (memread and memwrite).

Using an if-else statement based on the memtoreg or writereg control signal, the WB function is used to write the result after the MEM function. If the registry operation is successful, the regops counter variable will likewise be increased. The functions to read and write memory are constructed individually in the next line.

Cache is implemented in this function. Variables for the cache are defined and initialized with a specific value at the start of this function. Depending on whether the cache index is valid or not, an if-else sentence is formed after this variable. The memory address is saved in the cache storage in this function, and depending on its function, it is processed individually.

* Build Configuration / Environment

I am using C programming language in VScode. First of all, you need to install Vscode and mingw. Mingw is a C/C++ toolset, because Vscode doesn’t have a C compiler. After installing VScode, you have to download several extensions, which is C/C++ provided by Microsoft and its extensions and Code runner to run codes.

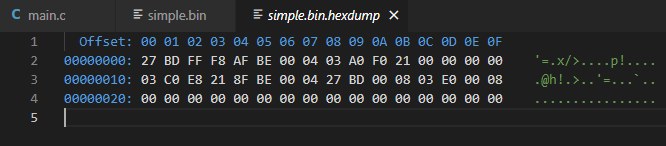
Sometimes when you want to use scanf() function, you need to insert something to the terminal, and we can use run in terminal to give the output in the terminal. Click on File > Preference > Settings > Extensions > Run Code Configuration > scroll down to Run in Terminal and check it.

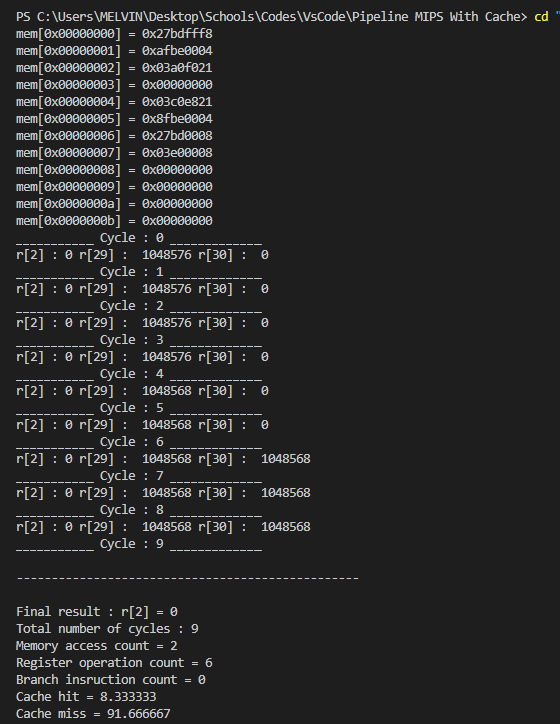
Because we will handle binary input file, so we have to install hexdump for Vscode in the extensions. This extension could show the binary input (which is impossible for human to read) in a unique hexadecimal format.

* Working Proofs

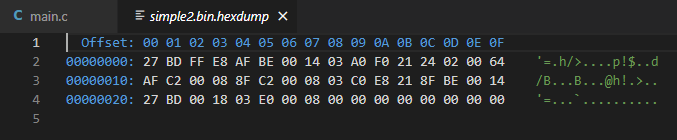
Below are the binary input files showed in hexadecimal dump file and the results of each binary input file. Because the results and some input binary files are too long, I will crop some of it and present the final results and print outputs.

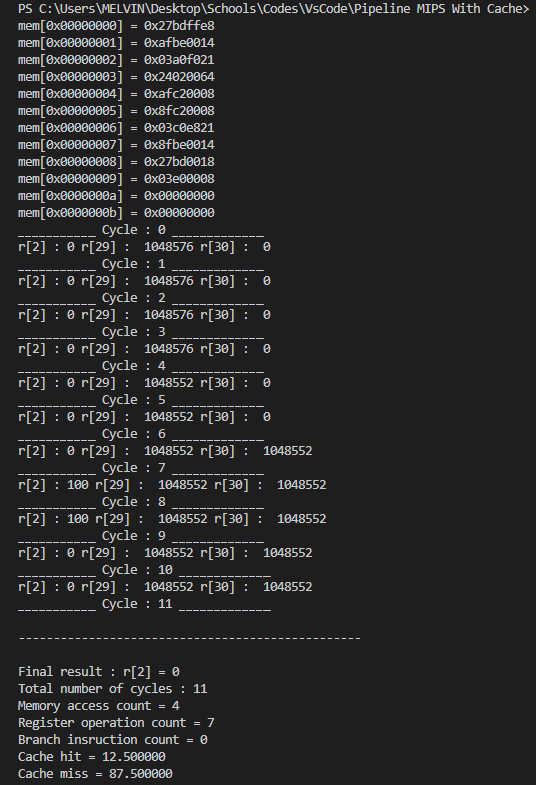
*Simple.bin*

**

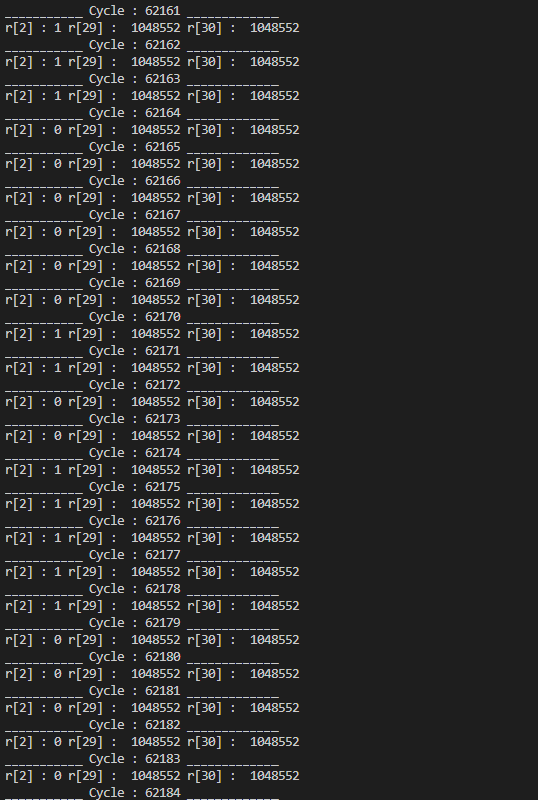


*Simple2.bin*





*Simple3.bin, simple4.bin, fib.bin, gcd.bin, and input4.bin. All resulting in infinite loop (fail attempts)*



* Trial & Errors

There is still a bunch of errors in the program, and I think it’s because the pipeline program itself is not properly function yet. Like there is still some things to be fixed, changing the pipeline structure, or branch and data dependency implementation. And of course, if I still don’t know how to fix the base program, which is to understand the flow of the program, it is gonna be hard to implement the cache because I’m confused with the flow of the program. Fortunately, there is still output for simple.bin and simple2.bin. Perhaps the output is wrong, but glad that somehow its still working.

* Final Thoughts

At first, I thought that this one will be easy, since it’s just adding the cache feature to the program. Unfortunately, the lack of knowledge of my own pipeline became so hard for me to solve and implement the cache. I think I’ll review this project of making cache only for different purpose, not only MIPS pipeline, because its very interesting and probably one of the most useful and utilized feature in the future. I’m just glad to be able to finish this project in time, even if the result is not commendable.