Computer Architecture & Mobile Processor

Project #3

-MIPS Pipeline

Nadia Calista Candra The

Dankook University

2021 Spring

Freedays left: 3

1. Project Introduction

Building an enhanced MIPS emulator with 5 pipeline stages by concurrently executing multiple instructions at the same time. In a five stage pipeline, instructions are processed in five distinct stages (or pipeline stages) those are IF (Instruction Fetch), ID (Instruction Decode), EX (Execute Operation), MEM (Access Memory Operand) and WB (Write Back). In a pipeline processor, an instruction is processed with multiple cpu clock cycles. In addition to multi cycle execution, in a pipeline processor, the execution of different instructions are overlapped and five different instructions can be executed at the same time.

2. Motivation

The MIPS emulator with 5 pipeline stages mimics the CPU, so it is good to understand the operational behaviour inside the CPU and to understand how on each stage of the pipeline can connect all of the instructions. Other than that, getting used to the Linux environment which is more manual and learning Linux by operating and making the project through assam server.

3. Concepts Used

File operations: Files are opened using fopen and it is closed using fclose. Contents are read using the fread function.

Struct statement: To define data types with more than one member and using structure tag to define the variable. Using the member access operator (.) to access the member of the struct. Using array of struct to define structure variables that have different information

Define directive: Using define to define macros within the source code. It will allow the constant value to be declared throughout the code.

Function declaration: Group of statements that perform a task and can define additional functions. Dividing functions that can perform a specific task.

Bzero function: It will erase the data in the memory starting from the location that is pointed by writing zero on it.

4. Program Structure

| flushlatchIfIdEx()

| flushlatchIdExMem()

| flushlatchExMemWb()

| initialize()

Runs in a while loop in the main function

+----------

| fetch()

| writeback()

| decode()

| execute()

| memoryaccess()

+----------

flushlatchIfIdEx() to flush or update the information from Fetch and Decode latch to the Decode and Execute latch.

flushlatchIdExMem() to flush or to update the information from Decode and Execute latch to the Execute and Memory latch.

flushlatchExMemWb() to flush or to update the information from Execute and Memory latch to the Memory and Write back latch.

initialize() to initialize the program and sets the value of all the register files into 0 except for ra into 0xFFFFFFFF

fetch() to fetch instructions from the file that is read and will increment or update the PC.

writeback() to update the register file with the result of an operation or load and to see if it has a writeback or not.

decode() to decode opcode, r type instructions, i type instructions, j type instructions, and control signals such as regdst, branch, memread, memtoreg, memwrite and regwrite. It will also calculate the branch target to see if it will be taken or not.

execute() to execute the program and it will go inside an if else that will differentiate either it has opcode 0x0 or other. If it has an opcode of 0x0 then it will go inside another if else to see its funct. Then it will print what the opcode is and will do the instruction accordingly. It will perform arithmetic or logical operations on either two

register values or a register and an immediate and will also call the fulshlatchIdExMem() function.

memoryacces() to memread and to memwrite if it passes the requirements. It will also load or store the information or value inside the memory or data cache and pass it to the memory result. It will also call the flushlatchExMemWb() function.

In the main function it will load the program using file operations such as fread, fopen and fclose. Functions that are written above will be put into a loop and every cycle that it goes will be added by 1 and will be printed out. Not only that, it will also print out the statistics of execution, memory operation instructions, register operation instructions, branch taken, branch not taken, and jump.

```
void flushlatchExMemWb(){
        memWbLatch[0].alu = exMemLatch[1].alu;
        memWbLatch[0].inst = exMemLatch[1].inst;
        memWbLatch[0].vRs = exMemLatch[1].vRs;
        memWbLatch[0].vRt = exMemLatch[1].vRt;
        memWbLatch[0].reg = exMemLatch[1].reg;
        memWbLatch[0].opcode = exMemLatch[1].opcode;
        memWbLatch[0].funct = exMemLatch[1].funct;
        memWbLatch[0].MemtoReg = exMemLatch[1].MemtoReg;
        memWbLatch[0].RegWrite = exMemLatch[1].RegWrite;
}
```

5. Problems and Solutions

I continue to do this project by using project 2's code and try to make the pipeline by using the single cycle code. At first I had a lot of trouble on how to make the latch that is used to save instructions that are waiting to be used on the next stage. However, while attending the lecture one student had the same problem as I did and Professor Yoo taught the whole class on how to make the latch by using structs and array of struct to define which one is going in and which one is going out. I also have a problem calling the loadprog function which I used on the last project, it will show a segmentation fault if I call the loadprog function on the main function. Because I can't

find how to correct the problem, so as a solution I just put the instructions on the function straight to the main function and as soon as I did that, it didn't show any segmentation fault. I also had problems on how to implement the jump instruction. At first I put it on the execute function and didn't know which array to use on the decode and execute latch for making the jump instruction work. However after much more research and studying I found out that it is supposed to be put in the decode function and uses the same instruction as the other instructions to differentiate whether it is jump instruction or not. And because I didn't put the jump instruction at the correct place, it resulted in my jump instruction not working and when I ran the code, it didn't work properly making it won't stop when I put it on the while loop. I also had problems working on how to make the program stop by it self when pc is at FFFFFFFF but after sorting out the lw, sw and main I managed to make it stop when it reached FFFFFFFF. At first the code is not this simple but because it is hard to read I tried to make it as simple as I can and make it work the same way.

6. Build Environment

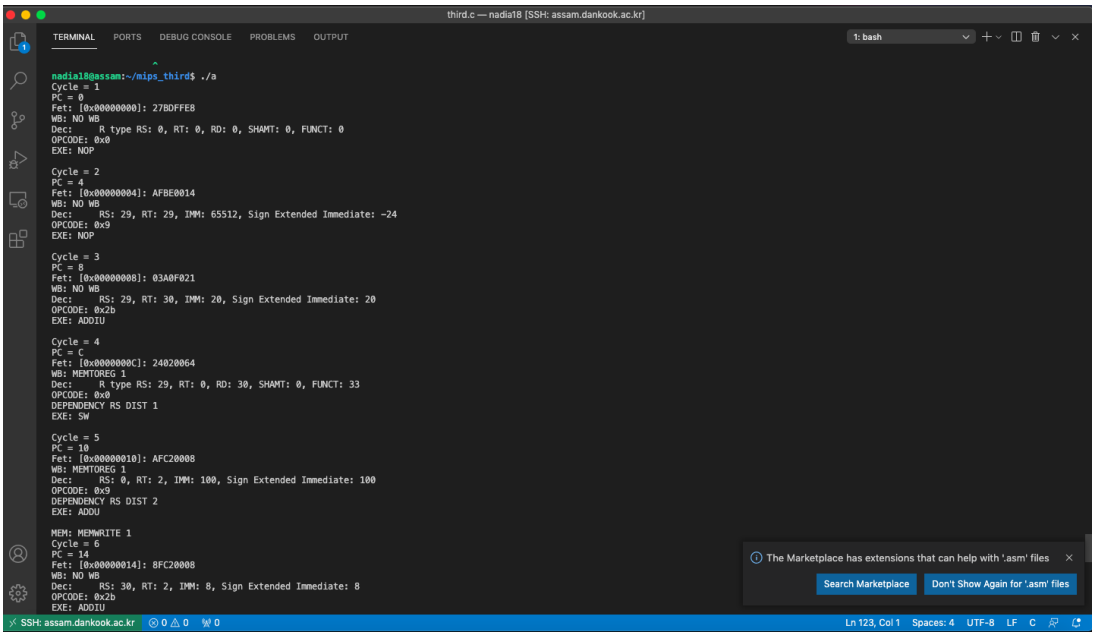
Compilation: Linux Assam, with GCC

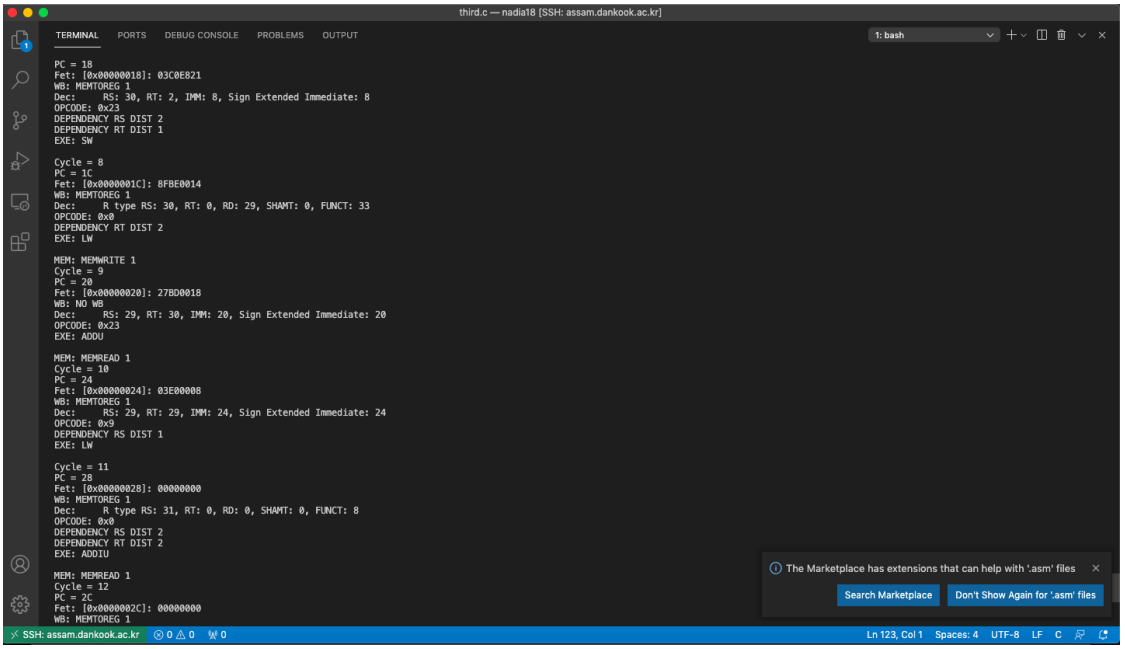
To compile: gcc -o a third.c

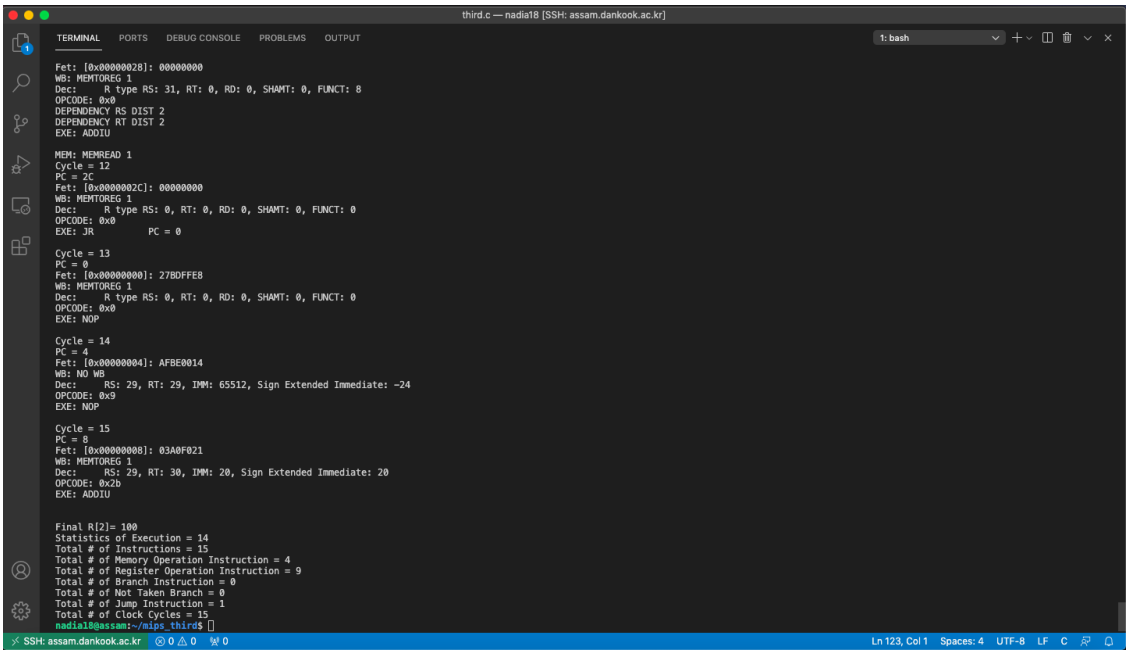
To run: ./a

The binary files are in the same directory (mips_third) as the third.c file

7. Screen Capture







8. Personal Feelings

I felt good finishing this project even though I couldn't submit this project on time. Thankfully there are still 5 free days left for me to use so my deadline is pushed back by around 5 days. However I did feel quite sad because I couldn't submit it on time. By using the project 2 codes and modifying from it is much faster than doing it from the start. As I do this project, I find it really hard to understand how to implement the theories that are learned and write it into the code but I really tried my best to implement it on how I would understand it.