CS520 Computer Architecture

Project 1 – Spring 2023

Due date: 3/6/2023

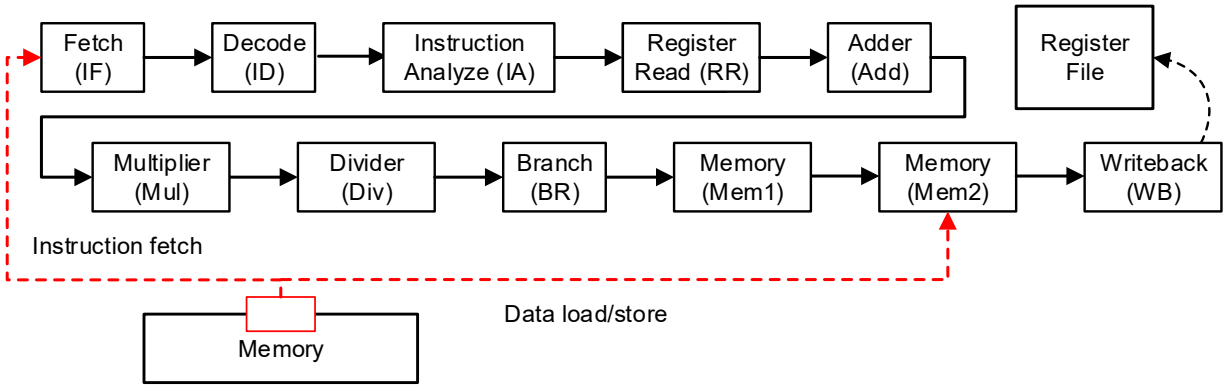
1. RULES

(1) You are allowed to work in a group of not more than two students per group, where both members must have an important role in making sure all members are working together.
(2) All groups must work separately. Cooperation between groups is not allowed.
(3) Sharing of code between groups is considered cheating and will receive appropriate action in accordance with University policy. The TAs will scan source code through various tools available to us for detecting cheating. Source code that is flagged by these tools will be dealt with severely.
(4) You must do all your work in C/C++.
(5) Your code must be compiled on remote.cs.binghamton.edu or the machines in the EB-G7 and EB-Q22. This is the platform where the TAs will compile and test your simulator. They all have the same software environment.

2. Project Description

In this project, you will construct a simple pipeline with an instruction decoder.

3. Simple Pipeline



Model simple pipeline with the following three stages.

- 1 stage for fetch (**IF**): fetch an instruction from memory
- 1 stage for decode (**ID**): decode an instruction
- 1 stage for instruction analyze (**IA**): analyze an instruction's dependency
- 1 stage for register read (**RR**): access the register file to read registers
- 1 stage for adder (**Add**): Adder operates on the operands if needed
- 1 stage for multiplier (**Mul**): Multiplier operates on the operands if needed
- 1 stage for divider (**Div**): Divider operates on the operands if needed
- 1 stage for branch (**BR**): PC is replaced with the branch destination address if needed
- 2 stages for memory (**Mem1, Mem2**): Access memory if needed
- 1 stage for register writeback (**WB**): Write the result into the register file

The pipeline supports 4B fixed-length instructions, which have 1B for opcode, 1B for destination, and 2B for two operands. The destination and the left operand are always registers. The right operand can be either register or an immediate value.

Opcode (1B)	Destination (1B)	Left Operand (1B)	Right Operand (1B)
Opcode (1B)	Destination (1B)	Operand (2B)	
Opcode (1B)	None (3B)		

Instruction: The supported instructions have 11 different types, as listed in the following table. The pipeline only supports integer arithmetic operations with 128 integer registers (R0 – R127), each having 4B. All numbers between 0 and 1 are discarded (floor). Instructions' operands are either registers or immediate values, and the destinations are registers. These instructions do not have data dependencies in the pipeline. The load instruction reads 4B from the specified address in the memory map file.

Mnemonic	Description		
	Destination	Left Operand	Right Operand
set	set Rx, #Imm (Set an immediate value to register Rx)		
	Register Rx	Immediate value	
add	add Rx, #imm, #imm (Compute Rx = #imm + #imm)		
	Register Rx	Immediate value	Immediate value
add	add Rx, Ry, #Imm (Compute Rx = Ry + #imm)		
	Register Rx	Register Ry	Immediate value
sub	sub Rx, #imm, #imm (Compute Rx = #imm - #imm)		
	Register Rx	Immediate value	Immediate value
sub	sub Rx, Ry, #Imm (Compute Rx = Ry - #imm)		
	Register Rx	Register Ry	Immediate value
mul	mul Rx, #imm, #imm (Compute Rx = #imm × #imm)		
	Register Rx	Immediate value	Immediate value
mul	mul Rx, Ry, #Imm (Compute Rx = Ry × #imm)		
	Register Rx	Register Ry	Immediate value
div	div Rx, #imm, #imm (Compute Rx = #imm / #imm)		
	Register Rx	Immediate value	Immediate value
div	div Rx, Ry, #Imm (Compute Rx = Ry / an immediate valve)		
	Register Rx	Register Ry	Immediate value
ld	ld Rx, #Addr (load the data stored in #Addr into register Rx)		
	Register Rx	Immediate value	
ld	ld Rx, Ry (load into register Rx the data stored in the address at Ry)		
	Register Rx	Register Ry	
ret	ret (exit the current program)		

Pipeline: The pipeline has 11 stages. An instruction is fetched at the IF stage. The instruction is decoded at the ID stage, and its dependency is analyzed at the IA stage. However, **we leave the IA stage empty for simplicity** in this project. Therefore, you do not need to implement the inside of the IA stage. The necessary registers are read at the RR stage before the execution stages. After the RR stage, five execution stages follow for different instruction types. The Add stage executes set, add, and sub instructions. The Mul stage executes mul instructions, and the Div stage executes div instructions. At last, the two Mem stages execute ld instructions. The loaded value is only available at the end of the second Mem stage (Mem2). Finally, destination registers are updated at the WB stage. Each stage takes 1 cycle to complete.

The pipeline has **one limitation**: **the memory unit only has one port** and cannot process two instructions concurrently. Hence, while memory instructions are being executed, the instruction fetch stalls. Also, since there are no branch instructions in this project, you can leave the BR stage empty.

Memory: The memory map file contains the snapshot of the system's main memory. The data in the file presents 4B data in the memory. Therefore, the location of each data shows the corresponding location of the mapped main memory multiplied by four. Also, although the programs are in separate files, they are mapped to the memory address 0 to 999. You do not need to copy the programs to the memory map file.

4. Validation and Other Requirements

4.1. Validation requirements

Sample simulation outputs will be provided on the website. You must run your simulator and debug it until it matches the simulation outputs. Your simulator must print the final contents in the register and performance results correctly (the format is already coded).

Your output must match both numerically and in terms of formatting, because the TAs will “diff” your output with the correct output. You must confirm correctness of your simulator by following these two steps for each program:

1) Redirect the console output of your simulator to a temporary file. This can be achieved by placing “> your_output_file” after the simulator command.

2) Test whether or not your outputs match properly, by running this unix command:
“diff –iw <your_output_file> <posted_output_file>”

The –iw flags tell “diff” to treat upper-case and lower-case as equivalent and to ignore the amount of whitespace between words. Therefore, you do not need to worry about the exact number of spaces or tabs as long as there is some whitespace where the sample outputs have whitespace. Both your outputs must be the same as the solution.

3) Your simulator must run correctly not only with the given programs. Note that TA will validate your simulator with hidden programs.

4.2. Compiling and running simulator

You will hand in source code and the TA will compile and run your simulator. As such, you must be able to compile and run your simulator on machines in EB-G7 and EB-Q22. This is required so that the TAs can compile and run your simulator. You also can access the machine with the same environment remotely at remote.cs.binghamton.edu via SSH. A make file is provided with two commands, make and make clean.

The pipeline simulator receives a program name as follows. The below command must generate your outputs.

e.g., sim program_1.txt

5. What to submit

You must hand in three c files, main.c, cpu.c, and cpu.h. Please follow the following naming rule.

LASTNAME_FIRSTNAME_project1.tar.gz

Also, if you work as a group, you must submit another file, group.txt, explaining each member's role in this project. Finally, you must submit a cover page with the project title, the Honor Pledge, and your full name as an electronic signature of the Honor Pledge. A cover page is posted on the project website.

6. Late submissions/Penalties

This project will be a part of all the following projects. That's why this project's portion in final grading is small. We allocate a long enough time, allowing you to flexibly manage your time to work on this project, although we highly recommend you start this homework as soon as possible.

On the other hand, we, therefore, have limited flexibility in the project's due date because our next project will start immediately after this one. Late submission is only allowed the first five days after the due date, with a penalty. Also, no extension will be allowed.

Various deductions (out of 100 points):
-8 points for each date late during the first 5 days.

Up to -10 points for not complying with specific procedures. Follow all procedures very carefully to avoid penalties.

Cheating: Source code that is flagged by tools available to us will be dealt with according to University Policy. This includes a 0 for the project and other disciplinary actions.