CS520 Computer Architecture

Project 3 – Spring 2022

Due date: 5/11/2022

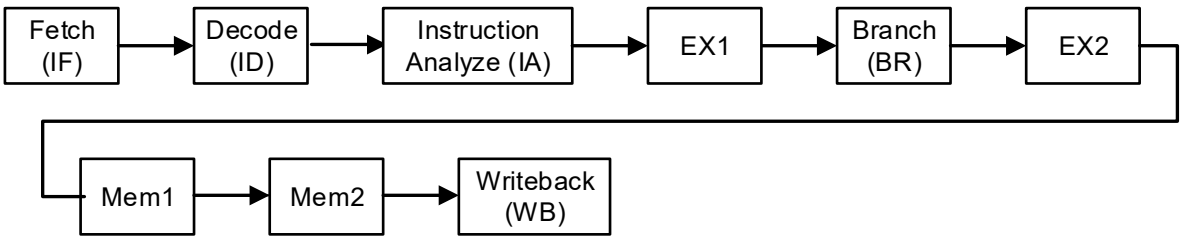
1. RULES

(1) All students must work alone. Cooperation is not allowed.
(2) Sharing of code between students 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.
(3) You must do all your work in the C/C++.
(4) 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)
- 1 stage for decode (ID)
- 1 stage for instruction and dependency analyze (IA)
- 1 stage for branch (BR)
- 1 stages for addition and subtraction (Execution unit 1: EX1)
- 1 stages for multiplication and division (Execution unit 2: EX2)
- 2 stages for memory operation (Memory unit: Mem1 and Mem2)
- 1 stage for write back (WB)
- 16 x 4B registers
- 64KB memory (code for 0-999, data for 1000 – 65535)
  memory.map is the memory map file for this project.
- Instruction formats

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)

The supported instructions have 19 different types as follows. The pipeline only supports integer arithmetic operations with 16 integer registers (R0 - R15), where each has 4B. All numbers between 0 and 1 will be discarded (floor).

Mnemonic	Opcode	Description		
		Destination	Left Operand	Right Operand
set	0x00	set Rx, #Imm (Set an immediate value to register Rx)		
		Register Rx		Immediate value
add	0x10	add Rx, Ry, Rz (Compute Rx = Ry + Rz)		
		Register Rx	Register Ry	Register Rz
add	0x11	add Rx, Ry, #Imm (Compute Rx = Ry + an immediate valve)		
		Register Rx	Register Ry	Immediate value
sub	0x20	sub Rx, Ry, Rz (Compute Rx = Ry – Rz)		
		Register Rx	Register Ry	Register Rz
sub	0x21	sub Rx, Ry, #Imm (Compute Rx = Ry - an immediate valve)		
		Register Rx	Register Ry	Immediate value
mul	0x30	mul Rx, Ry, Rz (Compute Rx = Ry * Rz)		
		Register Rx	Register Ry	Register Rz
mul	0x31	mul Rx, Ry, #Imm (Compute Rx = Ry * an immediate valve)		
		Register Rx	Register Ry	Immediate value
div	0x40	div Rx, Ry, Rz (Compute Rx = Ry / Rz)		
		Register Rx	Register Ry	Register Rz
div	0x41	div Rx, Ry, #Imm (Compute Rx = Ry / an immediate valve)		
		Register Rx	Register Ry	Immediate value
ld	0x50	ld Rx, #Addr (load the data stored in #Addr into register Rx)		
		Register Rx		Immediate value
ld	0x51	ld Rx, Rz (load into register Rx the data stored in the address at Ry)		
		Register Rx		Register Rz
st	0x60	st Ry, #Addr (store the content of register Rx into the address #Addr. E.g.)		
			Register Ry	Immediate value
st	0x61	st Ry, Rz (store the content of register Rx into the address at Ry)		
			Register Ry	Register Rz
bez	0x70	bez Ry, #Imm (branch to #Imm if Rx==0)		
			Register Ry	Immediate value

bgez	0x71	bgez Ry, #Imm (branch to #imm if Rx >= 0)		
			Register Ry	Immediate value
blez	0x72	blez Ry, #Imm (branch to #imm if Rx <= 0)		
			Register Ry	Immediate value
bgtz	0x73	bgtz Ry, #Imm (branch to #imm if Rx > 0)		
			Register Ry	Immediate value
bltz	0x74	bltz Ry, #Imm (branch to #imm if Rx < 0)		
			Register Ry	Immediate value
ret	0xFF	ret (exit the current program)		

All pipeline stages require 1 cycle to complete. A data dependency check and register reads are done at the IA stage. If there is dependency, the instruction must wait at the IA stage until the result becomes available. Once the register is ready, the instruction can read and proceed to the next stage.

There are four different execution stages, Ex1, Ex2, Mem1, and Mem2. The Ex1 is for addition and subtraction instructions, and the Ex2 stage is for multiplication and division. A memory instruction's effective address is prepared at the Ex1 stage, and the memory instruction is completed at the end of the Mem2 stage.

The branch target address is computed at the EX1 stage. The branch condition is checked at the Br stage, and the PC is updated at the same cycle. After the PC update, a new instruction is fetched at the next cycle (i.e., cycle 1: calculate target address, cycle 2: condition check + pc update, cycle 3: fetch a new instruction). To handle branch instruction easily, you can assume that you detect the branch instruction at the fetch stage and stall the instruction fetch.

The memory map file contains the snapshot of the system's main memory. The file position 0 to 65535 is mapped to the main memory address 0 to 65535. The data at the file position presents the data in the corresponding location of the main memory. 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.

Your register file has two read ports and one write port, supporting up to two register reads and one register write per cycle. The initial memory values are set in the given memory map file (memory.map).

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 as follows (the format is already coded).

```
--------------------------------
REG[ 0]  |  Value=0
--------------------------------
REG[ 1]  |  Value=0
--------------------------------
REG[ 2]  |  Value=0
--------------------------------
REG[ 3]  |  Value=0
--------------------------------
REG[ 4]  |  Value=0
--------------------------------
REG[ 5]  |  Value=0
--------------------------------
REG[ 6]  |  Value=0
--------------------------------
REG[ 7]  |  Value=0
--------------------------------
REG[ 8]  |  Value=0
--------------------------------
REG[ 9]  |  Value=0
--------------------------------
REG[10]  |  Value=0
--------------------------------
REG[11]  |  Value=0
--------------------------------
REG[12]  |  Value=0
--------------------------------
REG[13]  |  Value=0
--------------------------------
REG[14]  |  Value=0
--------------------------------
REG[15]  |  Value=0
--------------------------------
================================

Stalled cycles due to data hazard: 0
Stalled cycles due to control hazard: 0

Total stalls: 0
Total execution cycles: 0
Total instruction simulated: 0
IPC:
```

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.

The pipeline receives a program name.

e.g. sim test_01.asm

5. What to submit

You must hand in project3.c. Also, you must submit a cover page with the project title, the Honor Pledge, and your full name as electronic signature of the Honor Pledge. A cover page is posted on the project website.

6. Penalties

Various deductions (out of 100 points):
-8 points for each date late during the first 5 days.
Since it is the end of the semester, we cannot grade your submission after 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.