CS520 Computer Architecture

Project 1 – Fall 2021

Due data: 10/22/2021

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 and optimizations.

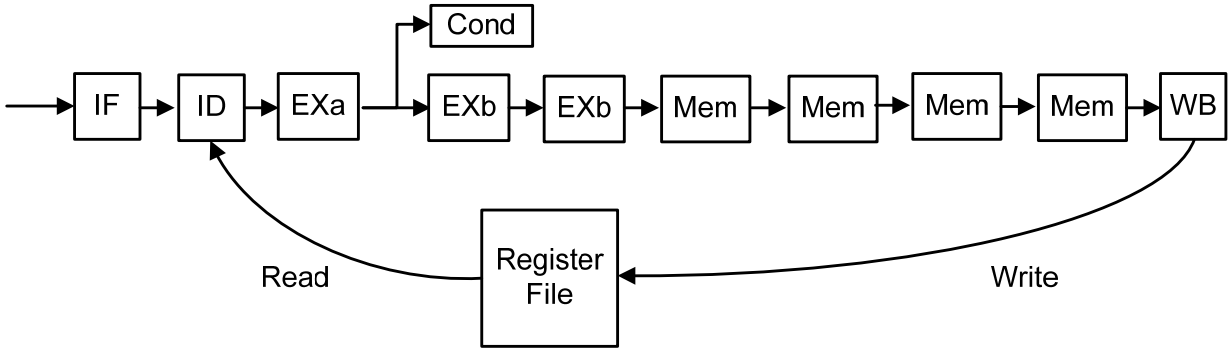
3. Simple Pipeline

Model simple pipeline with the following stages.

- 1 stage for fetch (IF)
- 1 stage for decode (ID)
- 1 stages for addition and subtraction (Execution unit A: EXa)
- 2 stages for multiplication and division (Execution unit B: EXb)
- 4 stages for memory operation (Memory unit: MEM)
- 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

```
set Rx, #Imm   // set value to register Rx
ld  Rx, #Addr      // load into register Rx the data stored in the address #Addr. E.g.) ld R1, 0x0010
ld  Rx, Ry         // load into register Rx the data stored in the address at Ry
st  Ry, #Addr      // store the content of register Rx into the address #Addr. E.g.) st R1, 0x0010
st  RX, Ry         // store the content of register Rx into the address at Ry
add  Rx, Ry, Rz    // Compute Rx = Ry + Rz
add  Rx, Ry, #Imm   // Compute Rx = Ry + #Imm (an immediate valve)
sub  Rx, Ry, Rz       // Compute Rx = Ry – Rz
sub  Rx, Ry, #Imm   // Compute Rx = Ry - #Imm (an immediate valve)
mul  Rx, Ry, Rz       // Compute Rx = Ry * Rz
mul  Rx, Ry, #Imm  // Compute Rx = Ry * #Imm (an immediate valve)
```

```
div  Rx, Ry, Rz        // Compute Rx = Ry / Rz
div  Rx, Ry, #Imm   // Compute Rx = Ry / #Imm (an immediate valve)
bez Rx, #Imm     // branch to #Imm if Rx==0
bgez Rx, #Imm   // branch to #imm if Rx >= 0
blez Rx, #Imm   // branch to #imm if Rx <= 0
bgtz Rx, #Imm   // branch to #imm if Rx > 0
bltz Rx, #Imm   // branch to #imm if Rx < 0
ret                  // exit the current program
```



The dependency check (RAW and WAW hazard) and register read is done at the ID stage. If there is dependency, the instruction must wait at the ID stage until the result becomes available. In case the data is ready, the stage reads the data and does not need to read it again. For example, a dependent instruction can move to the next stage without a register read at the next cycle after the dependency is resolved if it has already read the registers during the stalls. Otherwise, it needs extra cycle to read the register.

The branch target address is computed at the EXa stage. The branch condition is checked at the next cycle (EXb: cond) and 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).

Your register file only has one read port and one write port. Each register only allows one operation at a cycle. If one register receives both read and write operations at the same cycle, write operation is always performed first. If your instruction has two register operands, the instruction needs two cycles to read both operands (structural hazard). The register is always updated at the WB stage.

In the given project c file (project1.c), you have to complete Fetch (fetch()) and Decode (decode()) stages in order to make your simulator print correct output. The initial memory values are set in the given memory map file (memory_map.txt).

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:
REG 1:
...
REG 15:

(A) Stalled cycles due to data hazard:
(B) Stalled cycles due to control hazard:
(C) Stalled cycles due to structural hazard:

(A+B+C) Total stalls:
Total execution cycles:
IPC:

Also, the content in the memory map file (memory_map.txt) must be correct too. 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 and final memory_map.txt 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

We will provide you the simulator codes as a library. But you have to complete the fetch and the decode stages (functions) in the given project1.c file.

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 project1.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):
-5 points for each date late during the first 6 days.
-10 points for each date late after the first 6 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.