Programming Assignment #3: 64-Bit RISC-V Pipeline

Prof. Jae W. Lee (jaewlee@snu.ac.kr)
Department of Computer Science and Engineering
Seoul National University

TA (snu-arc-uarch-ta@googlegroups.com)

Contents

Goal of this project

- **You should implement three features in a RISC-V pipeline.**
  - The base pipeline structure is given. Three tasks:

    #1 Implement a BEQ instruction and resolve control hazard

    #2 Implement a forwarding unit and resolve data hazard

    #3 Moving the branch decision from MEM stage to the ID Stage

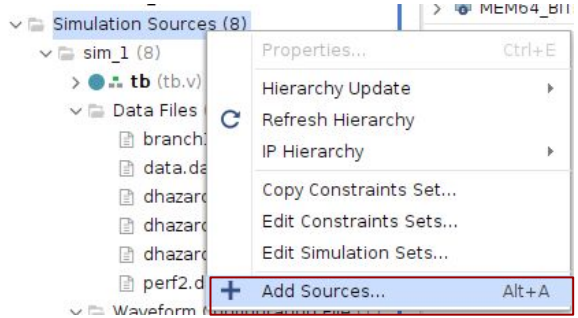
Environment setup

- You should use Xilinx ISE or Vivado to solve PA3

- You can get skeleton code and test bench from git repo
  - `git clone https://github.com/SNU-ARC/2023_fall_comarch_PA3`

Environment setup

- **Before change test bench, you should add test bench to project**



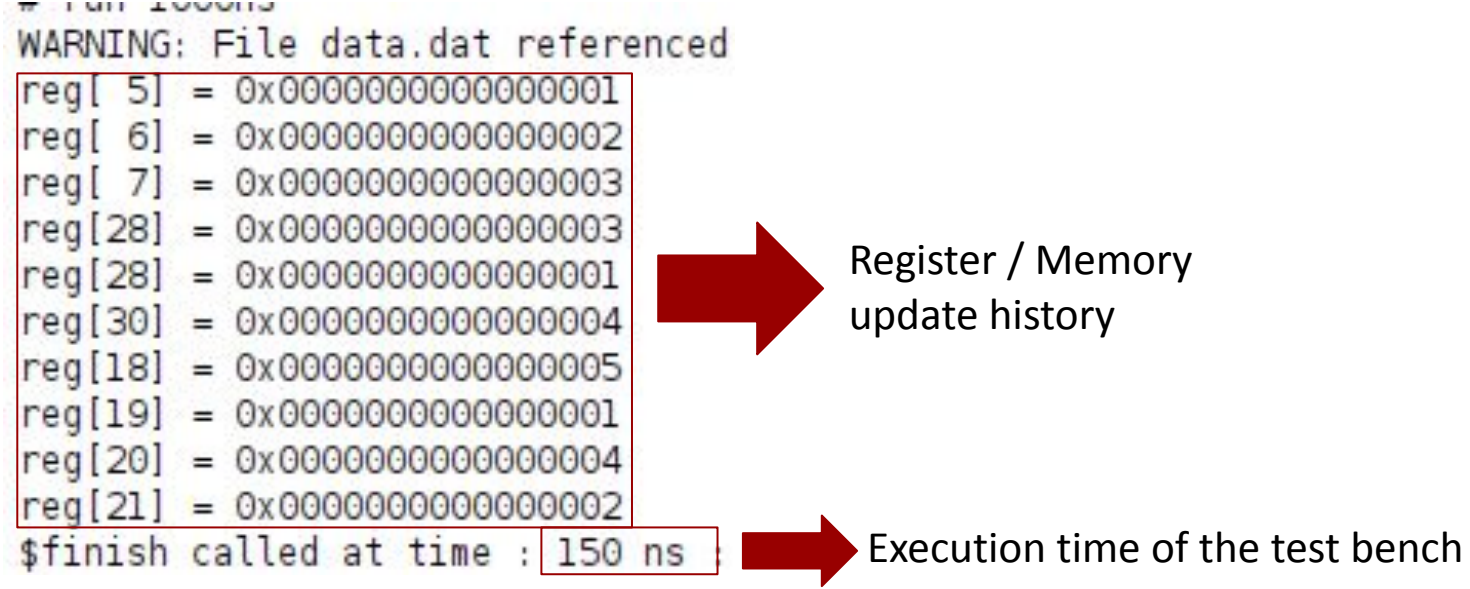
- **Then, you can change test bench by editing the memory.v file**

```
module rom
(
    input   [31:0] address,
    output  [31:0] data_out
);
    parameter FILE = "dhazard3.dat";
    parameter ROM32_BITMASK = 32'h1fc;

    reg [31:0] data[127:0]; //address 8:2 is offset

    initial
    begin
        //Read inst memory file.
        $readmemh(rom.FILE, data);
    end

    //64'h1fc: Bitmask for 8:2
    assign data_out = (address & (~(ROM32_BITMASK))) ? 32'h0 : data[address[8:2]];
```

Please replace **file name (*.dat)** at here!!!

Environment setup

- **When a register write or memory write occurs, you can verify the location and value though the console windows in vivado program**



Explanation

- **There are 7 skeleton code files (.v), 4 sample test bench, 5 test bench (.s, .dat) for evaluation (+ 1 hidden test bench)**

- **You can only modify the following three files, and you must submit them**
  - riscv-pipeline.v
  - control.v
  - inst_decoder.v

Explanation of skeleton code

- **alu.v**
  - Xilinx design file for alu
  - AND, OR, XOR, ADD, SUB Operations are supported
- **control.v (submit needed)**
  - Xilinx design file for control signal
  - AND(i) / OR(i) / XOR(i) / ADD(i) / SUB / Load / Store Instructions are supported
- **inst_decoder.v (submit needed)**
  - Xilinx design file for instruction decoder
  - AND(i) / OR(i) / XOR(i) / ADD(i) / SUB / Load / Store Instructions are supported
- **memory.v**
  - Xilinx design file for instruction / data memory
  - Instruction Load / Data Load / Data Store function are supported

Explanation of skeleton code

- **reg.v**
  - Xilinx design file for registers
  - We assume that **write is in the first half of the clock cycle** and the **read is in the second half**, so the read delivers what is written
- **riscv_pipeline.v (submit needed)**
  - xilinx design file for riscv pipeline
  - Implementation of base pipeline
- **tb.v**
  - Xilinx design for clock / signal generator
  - Implementation of reference clock and reset signal

Explanation of test bench

- **sample1.dat**
  - Containing only single branch instruction
- **sample2.dat**
  - Containing multiple branch instruction
- **sample3.dat**
  - Containing EX/MEM Hazard
- **sample4.dat**
  - Containing EX/MEM Hazard, Double data hazard

Explanation of test bench

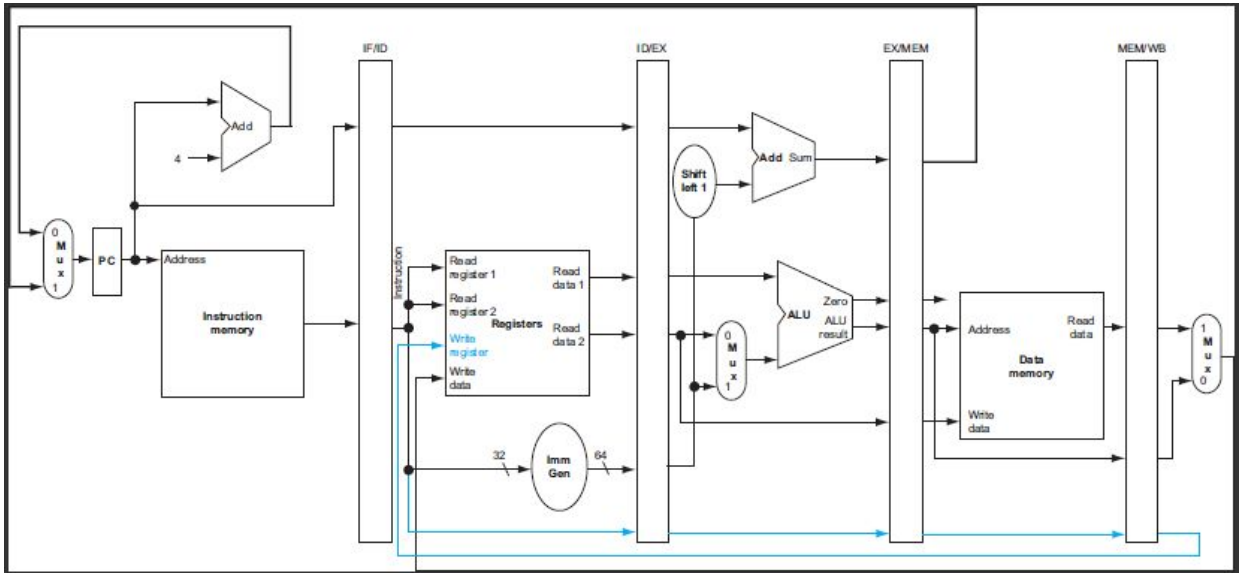
- **branch.dat**
  - Check correctness
  - Containing nested loop using branch instructions
- **dhazard.dat**
  - Check correctness
  - Containing data hazard and control hazard

Explanation of test bench

- **perf1.dat**
  - Check correctness and **performance (≤ 970 ns)**
  - Test bench for evaluating the performance of static branch predictor
- **perf2.dat**
  - Check correctness and **performance (200 ns)**
  - Test bench for evaluating the performance of forwarding
- **perf3.dat**
  - Check correctness and **performance (980 ns)**
  - Test bench for evaluating the performance of moving branch decision

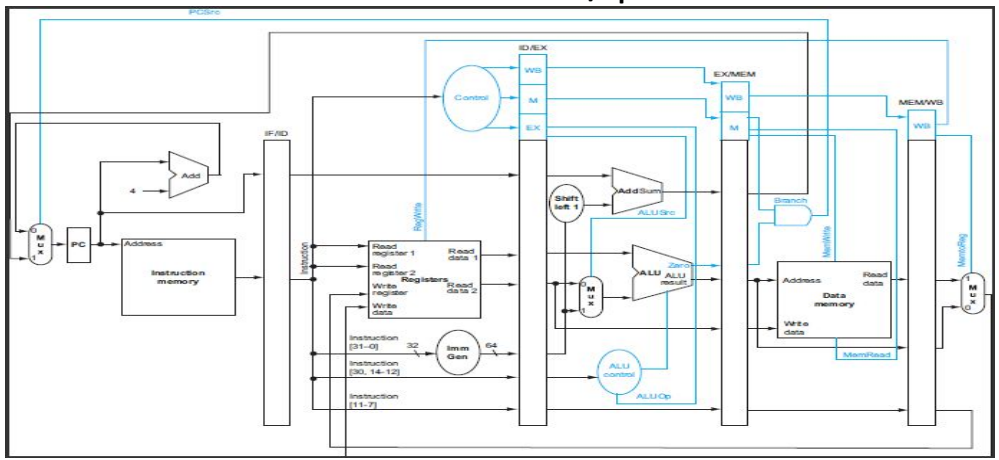
Base pipeline overview

- **Supported instructions**
  - Arithmetic instructions: ADD(i), SUB, AND(i), OR(i), XOR(i)
  - Memory instructions: LD, SD
  - Other instruction: HALT (stop simulation)



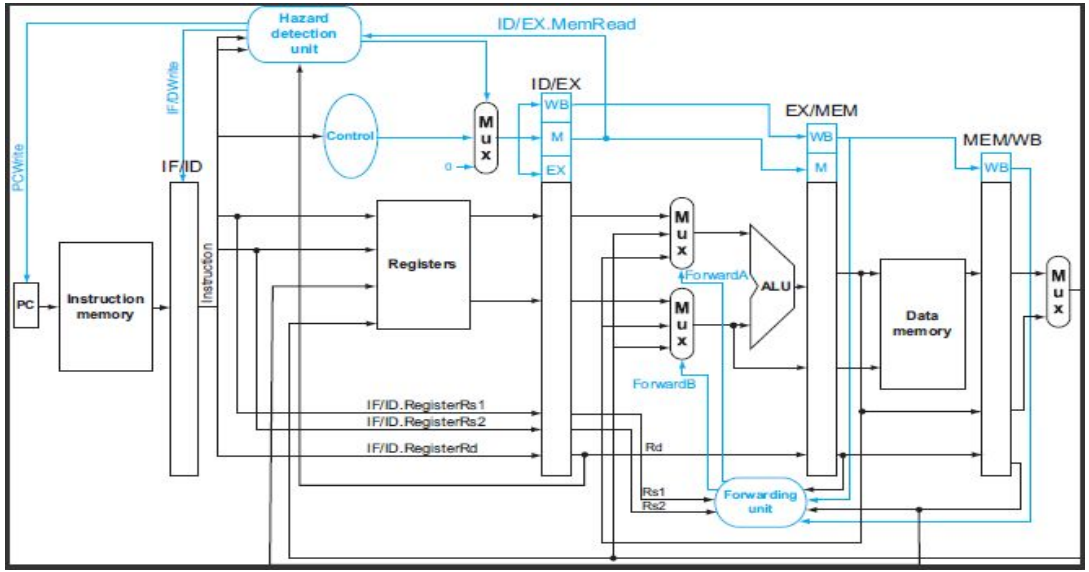
Problem 1. Branch instruction (BEQ)

- **Implement branch instruction (BEQ)**
  - Refer to RISC-V reference sheet for instruction encoding
  - Pipeline should be stalled properly to resolve control hazards
  - To minimize stall, you should implement static branch predictor (assume branch is **always not taken**)
  - Related testbench : branch.dat / perf1.dat



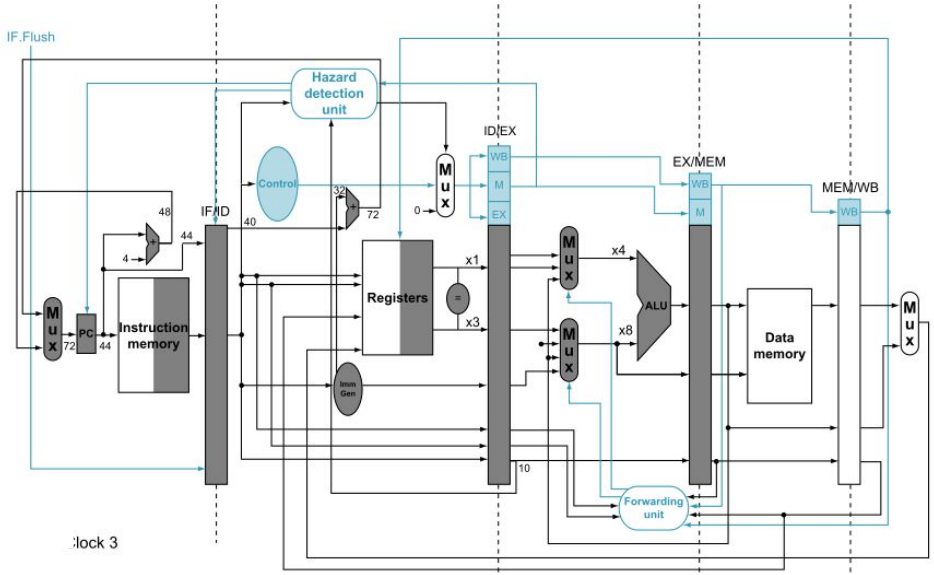
Problem 2. Forwarding unit

- **Implement forwarding unit**
  - Pipeline should be stalled properly to resolve data hazards
  - To minimize stall, you should implement forwarding unit
  - Related testbench : dhazard.dat / perf2.dat



Problem 3. Moving branch decision

- **Moving the branch decision from MEM stage to the ID Stage**
  - Should implement additional forwarding and hazard detection hardware to properly resolve data hazards and minimize stall.
  - Related testbench : perf3.dat



Helper program (Linux)

- **Use provided assembler (G++ is required)**
  - It converts code written in RISC-V assembly into binary format
  - To build program in a **Windows environment**, you will need tools like MinGW
  - List of supported instructions
    - ✓ Arithmetic & Bitwise Operation : ADD(i), SUB, AND(i), OR(i), XOR(i)
    - ✓ Memory operations: LD, SD
    - ✓ Branch operation: BEQ
    - ✓ Others: NOP, HALT
  - **Branch to label is not supported** (Only branch to offset is supported)
- **Usage example**
  - `$> riscv-assembler sample.s >> sample.dat`

```
kjh@kjh-MS-7D91:~/Documents/TA_ComputerArchitecture/bench$ make
g++ -Wall -Werror -I. -c main.cpp -o main.o
g++ -Wall -Werror -I. -c riscv-assembler.cpp -o riscv-assembler.o
g++ main.o riscv-assembler.o   -o riscv-assembler
kjh@kjh-MS-7D91:~/Documents/TA_ComputerArchitecture/bench$ ./riscv-assembler dhazard_sample.s >> dhazard_sample.dat
```

Grading Policy

- **Test bench : 90 %**
  - branch.dat / dhazard.dat : 10% per each
  - perf1.dat / perf2.dat / perf3.dat : 20% per each
  - hidden.dat : 10%
  - If you hard code for a specific situation, there may be disadvantages
- **WriteUp : 10%**
- **For late submission:**
  - A deduction of 10% p per 24 hours
  - After next 120 hours, submission will not be accepted

Submission

- **Write-up**
  - Briefly describe your implementation.
  - Filename: [student_id].txt (example: 2023-12345.txt)
  - Please use ‘UTF-8’ encoding if possible
  - **Please** submit it in **txt** format. Other formats are not accepted.
- **Compress your source code and write-up into a single zip file**
  - Compress **riscv-pipeline.v, control.v, inst_decoder.v** and your report
  - Filename should be [student_id].zip (example: 2023-12345.zip).
  - **Please** submit it in **ZIP** format. Other formats are not accepted.
- **Submission deadline: by 23:59 on November 8, 2023**