Lab 8: Pipelined Implementation of RISC-V with Hazards Handling

Objectives

The objective of this lab is to complete the implementation of the RISC-V pipelined datapath including data and control hazards handling mechanisms and to test it on the Nexys A7 board.

This lab is organized as follows:

1. An introduction
2. Experiments to be conducted
3. Lab Report Requirements

Introduction

In the previous labs, you should have completed the Verilog modeling and simulation of the pipelined RISC-V datapath according to Figure 1 assuming that we only need to support the following RV32I ISA subset:

- Memory reference: LW (I-Format), SW (S-Format)
- Arithmetic/logical: ADD, SUB, AND, OR (R-Format)
- Control transfer: BEQ (SB-Format)

The goal of this lab is to extend it to handle data and control hazards correctly.

In this lab we are going to conduct the following experiments:

1. Implement, integrate, and test the forwarding unit
2. Implement, integrate, and test the hazard detection unit
3. Implement, integrate, and test instructions flushing

Note: Students should be working in pairs to implement the following experiments

Three important notes:

1. Not all signals monitored come from the same stage which means that during a program's execution, each group of signals will be associated with a different instruction in the pipeline.
2. The pipeline description includes 2 extra fields in the ID/EX pipeline register which are Rs1 and Rs2. These are needed by the forwarding unit which will be implemented in experiment 1.
3. As studied, we will assume that the write back stage is complete in the first half of the clock cycle, which means that writing in the register file should be activated by the falling edge of the RISC-V clock instead of its rising edge.

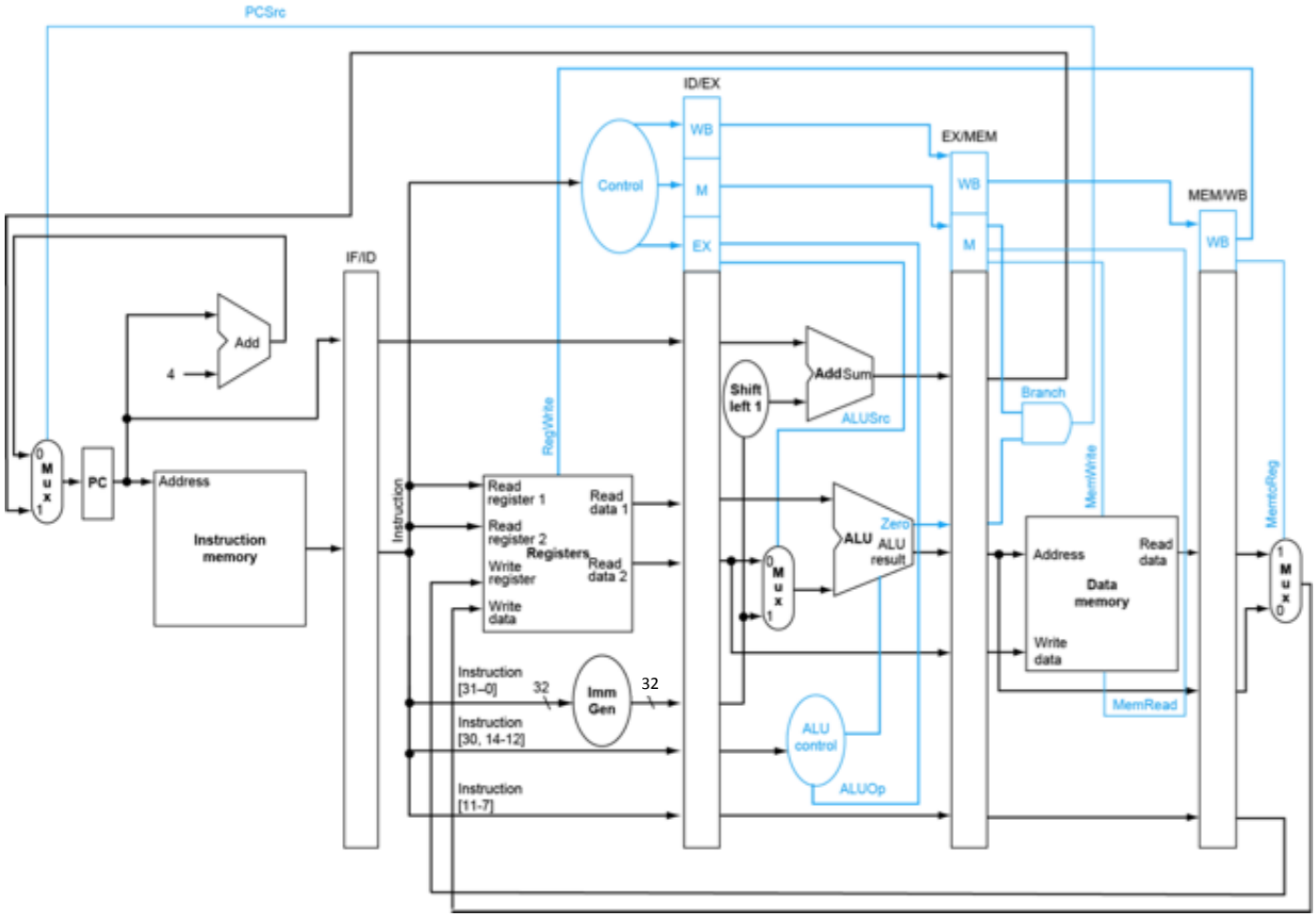

Figure 1. RISC-V Pipelined Datapath (ignoring data and control hazards)

Experiments

Experiment 1: Implement, integrate, and test the forwarding unit

The forwarding unit is responsible for forwarding operands to the ALU inputs if these operands are available in the EX/MEM register or at the output of the MUX after the MEM/WB register at the beginning of the EX stage. The forwarding unit should produce two outputs forwardA and forwardB according to the following conditions:

- **EX hazard (Fwd from EX/MEM pipeline reg):**

```
if ( EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
      and (EX/MEM.RegisterRd == ID/EX.RegisterRs1) )

      forwardA = 10


if ( EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
      and (EX/MEM.RegisterRd == ID/EX.RegisterRs2) )

      forwardB = 10
```

- **MEM hazard (Fwd from MEM/WB pipeline reg):**

```
if ( MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
      and (MEM/WB.RegisterRd == ID/EX.RegisterRs1) )
      and not ( EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
                  and (EX/MEM.RegisterRd == ID/EX.RegisterRs1) )

      forwardA = 01


if ( MEM/WB.RegWrite and (MEM/WB.RegisterRd ≠ 0)
      and (MEM/WB.RegisterRd == ID/EX.RegisterRs2) )
      and not ( EX/MEM.RegWrite and (EX/MEM.RegisterRd ≠ 0)
            and (EX/MEM.RegisterRd == ID/EX.RegisterRs2) )

      forwardB = 01
```

Accordingly the forwarding unit has two outputs: `forwardA` and `forwardB`, and 6 inputs: `ID/EX.RegisterRs1`, `ID/EX.RegisterRs2`, `EX/MEM.RegisterRd`, `MEM/WB.RegisterRd`, `EX/MEM.RegWrite`, and `MEM/WB.RegWrite`.

The `forwardA` and the `forwardB` outputs should be used to control the selection lines of two newly added 4x1 multiplexers providing inputs to the EX stage instead of the ID/EX register.

Experiment steps:

a) Implement the forwarding unit module and modify the top-level RISC-V module to instantiate it in the right place along with both multiplexers.
b) Test the resulting circuit by Vivado simulation and on the Nexys A7 board afterwards on the test program of the previous lab.
c) Test the resulting circuit after removing the unneeded NOP instructions from the test program.
d) Finally, identify the results that will be incorrect due to remaining unhandled hazards if all NOPs are removed.

Experiment 2: Implement, integrate, and test the hazard detection unit

The hazard detection unit is responsible for stalling the datapath for one clock cycle in case of a load-use hazard. This detection is possible during the decoding of the using instruction according to the following condition:

```
If ( (IF/ID.RegisterRs1==ID/EX.RegisterRd) OR (IF/ID.RegisterRs2==ID/EX.RegisterRd) )
      and ID/EX.MemRead and ID/EX.RegisterRd ≠ 0

      stall = 1
```

Accordingly the hazard detection has one output: `stall` and 4 inputs: `IF/ID.RegisterRs1`, `IF/ID.RegisterRs2`, `ID/EX.RegisterRd`, and `ID/EX.MemRead`.

The `stall` output should be used to:

i. control the load of the PC register (preventing it from loading when stall = 1)

ii. control the load of the IF/ID register (preventing it from loading when stall = 1)
iii. control the selection line of a newly added 2x1 multiplexer providing the control values to be stored in the ID/EX register in such a way that when stall = 1, zeros are stored instead of all the control values.

Experiment steps:

a) Implement the hazard detection unit module and modify the top-level RISC-V module to instantiate it in the right place along with the mentioned multiplexers.
b) Test the resulting circuit after removing the unneeded NOP instructions, using Vivado simulation and on the Nexys A7 board afterwards.
c) Finally, identify the results that will be incorrect due to remaining unhandled hazards if all NOPs are removed.

Experiment 3: Implement, integrate, and test instructions flushing

When a BEQ instruction is encountered according to the current datapath, nothing happens until it reaches the MEM stage in which the branch outcome is decided. Meanwhile the 3 instructions following the BEQ instruction were introduced in the pipeline.

In case the branch outcome is not to take the branch, nothing needs to be done and execution continues normally; however, if the branch outcome is to take the branch, the `PCSrc` control signal is set to 1 forcing the PC to start fetching instructions from the BEQ target address. This is the correct behavior (assuming an always-not-taken static branch predictor) except that in case the branch is to be taken, the 3 instructions following the BEQ have to be flushed from the pipeline.

No special flushing detection unit needs to be implemented, since the `PCSrc` signal serves as a flag that we need to flush all 3 instructions. So when `PCSrc`=1, we need to flush:

i. The instruction currently being fetched. This is done by converting it to an instruction that does nothing (a NOP instruction) which we can define given the supported instruction set as "add x0, x0, x0". So we use the `PCSrc` signal to control the selection line of a newly added 2x1 multiplexer providing the IF/ID register with the instruction fetched. If `PCSrc` = 1, we select a hardcoded instruction equivalent to NOP instead of the instruction read from the instruction memory

ii. The instruction currently being decoded. This is done by controlling the selection line of the 2x1 multiplexer providing the control values to be stored in the ID/EX register (the same multiplexer when `stall` = 1) in such a way that when `PCSrc` = 1, zeros are stored instead of all the control values. Hint: The selection line of this multiplexer should be the result of ORing `stall` and `PCSrc`.

iii. The instruction currently being executed. This is also done by controlling the selection line of a newly added 2x1 multiplexer providing the control values to be stored in the EX/MEM register in such a way that when `PCSrc` = 1, zeros are stored instead of all the control values.

Experiment steps:

a) Integrate the necessary flushing circuitry in the top-level RISC-V module.
b) Test the resulting circuit after removing all NOP instructions, using Vivado simulation and on the Nexys A7 board afterwards.

Deliverables

General Report Submission Notes:

- Each student is required to submit an individual report.
- The submission should be composed of **two files only**:
  1. YourID_Report.pdf/doc:
     A single pdf/word file having the solution of the lab report questions in correct sequence.
  2. YourID_labwork.zip/rar/…:
     A zipped folder containing your codes for all the lab experiments organized in sequence.
- **Submission deadline**: half an hour before the start of next week's session.
- **Submission method**: On Blackboard

Lab Report [10 pts]

1. [0 pts] Your name and student ID.
2. [2 pts] Verilog code for the 3 experiments.
3. [3 pts] The values obtained after each experiment. Include the most relevant values only for each experiment. These must include all the incorrect values detected due to lack of proper hazards handling. (use screen shots to explain)
4. [3 pts] One photo of the Nexys A7 board for each experiment illustrating an incorrect output must be shown. A clear description of the photo provided including the instructions in each pipeline stage, the value of the selection lines (ledSel and ssdSel) when the photo was taken, and the correct output should be included.
5. [1 pts] If we have a data hazard from memory stage, and data hazard from execution stage, which one will forward the data? Why?
6. [1 pts] Why do we add this condition `(EX/MEM.RegisterRd ≠ 0)` in the forwarding unit?
