ACM CLASS, SHANGHAI JIAO TONG UNIVERSITY

Five-Stage MIPS Pipeline in Verilog

Nayun Xu (ID: 515030910635)
ACM Class

June 25, 2017

CONTENTS

1 Introduction

This report is a summary on my coursework five-stage MIPS pipe line project. I mainly explain the general structure and my implementaion with Verilog. My project contains a small part of basic MIPS instructions, so it is much simpler than a complete MIPS pipeline. I will explain how I implement forwading, harzard control in this project.
My code and test are available on `https://github.com/Nerer/MIPS-CPU`
MIPS instruction standard in this project is referred to `http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html`
The instructions implemented in this project are :add, addi, sub, and, andi, or, ori, xor, xori, slt, slti, beq, bne, j, jr lb, lui, lw, sb, sw.

2 General Architecture

2.1 Five-Stage Pipeline CPU

The architecture of this project is similar to figure 2.1 which is taught in Computer System class.

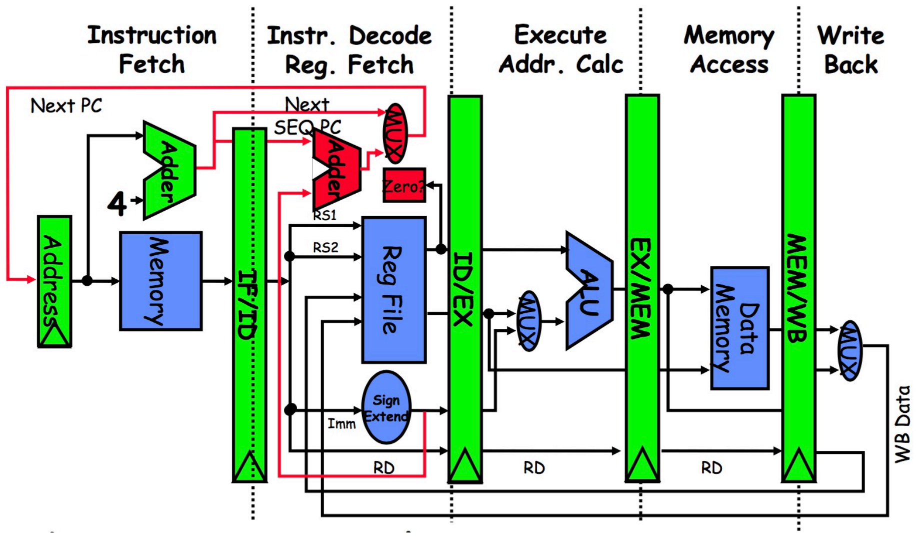


Figure 2.1: Five-Stage MIPS Pipeline

So my project contains five stage modules: `stage_if.v`,`stage_id.v`,`stage_ex.v`,`stage_mem.v` and `stage_wb.v`, four transition buffers:`trans_if_id.v`,`trans_id_ex.v`,`trans_ex_mem.v` and `trans_mem_wb.v`, one regitser file: `register.v`
And there is a `cpu.v` to build connections between thse modules.

2.2 Memory

In this project, Harvard Architecture is used. So I have `rom.v` for instructions and `ram.v` for other data.

2.3 SOPC

In this project, I use `sopc.v` to build connections between memory and cpu(the pipeline).

3 Hazard and Solution

3.1 RAW Hazard

In this project, only RAW(Read After Write) is possible to happen. It means that some instruction being decoding wants to read data from a register that is destination of an ealier instruction when the result hasn't been written into the register.

3.2 Solution for RAW Hazard

3.2.1 Forwarding

If after excution stage, the result to be written into the destination register has been got, send this result to instruction decoding stage. And instruction decoding stage can estimate whether it can use this result. The major structure is shown in 3.1

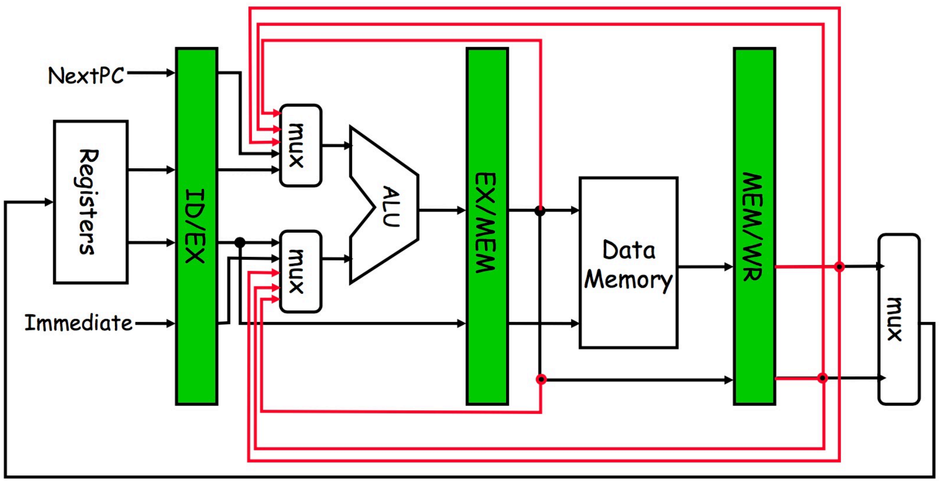


Figure 3.1: Forwarding

3.2.2 Stall

There can be some situations that RAW happens even with forwardig. An example is shown in 3.2

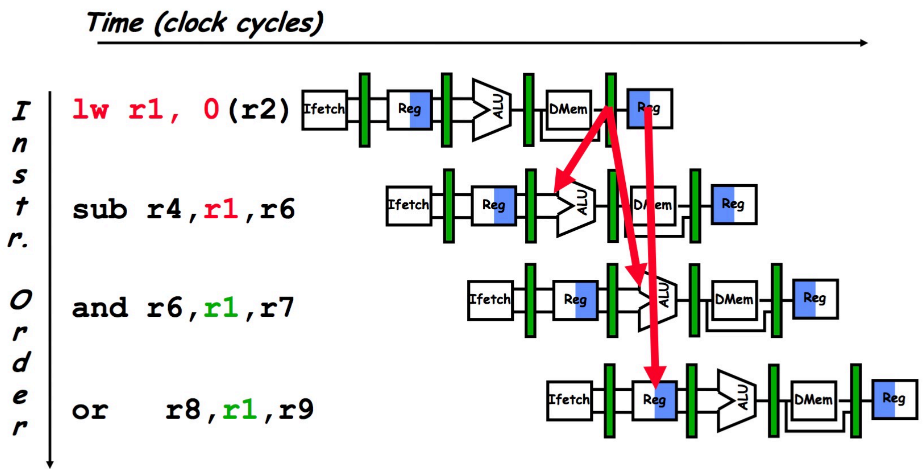


Figure 3.2: RAW Happens Even With Forwarding

To deal with such situation, we just stall all the following instructions for one cycle.

4 Main Modules Elaboration

4.1 Pipeline Modules

There are five stages in this five-stage MIPS pipeline.

Please refer to the code for detailed input and output of each module. I wrote some comments in the code.

4.1.1 Instruction Fetch

Implemented in `stage_if.v`. The main function of this module is to generate `PC`, the address of current instruction, which will be sent to instruction memory (`rom.v`) in `sopc.v` to access instruction. And it also receive control signals to change `PC`.

4.1.2 IF/ID

Implemented in `trans_if_id.v`. The main function of this module is to deal with stall from stage IF and transfer data from stage IF to stage ID. However, in this project, stall in this module will never happen.

4.1.3 Instruction Decode

Implemented in `stage_id.v`. This module is mainly a decoder, which takes instruction from IF module as the input and outputs several control signals. Those signals are

Signal	Meaning
`reg_write_enable`	whether or not is to write register
`reg_write_address`	which register is to be write
`pc_write_enable`	whether or not is to modify `PC`
`pc_write_data`	what to write into `PC`
`mem_write_enable`	whether or not is to write memory
category	the type of computation to perform in ALU in execution stage
operator	the subtype of computation to perform in ALU in execution stage
`operand_a`	the first source of ALU
`operand_b`	the second source of ALU
`stall_request`	whether or not this instruction should be stalled

Table 4.1: A Summary on the decoder's output signals

This module also output some other signals such as the original instruction and some signals to interact with register file.

4.1.4 ID/EX

Implemented in `trans_id_ex.v`. The main function of this module is to deal with stall in stage ID, and transfer data from ID to EX.

4.1.5 Execution

Implemented in `stage_ex.v`. This module performs the computation except memory operation and control operation and implement forwarding.

4.1.6 EX/MEM

Implemented in `trans_ex_mem.v`. The main function of this module is to deal witth stall in stage EX, and transfer data from stage EX to stage MEM. However, in this project, stall in this module will never happen.

4.1.7 Memory Access

Implemented in `stage_mem.v`. Memory access. Receive memory instruction and output signals to interact with `ram.v`. This module also implements forwarding.

4.1.8 MEM/WB

Implemented in `trans_mem_wb.v`. The main function of this module is to deal with stall in stage MEM, and transfer data from stage MEM to stage WB.

4.1.9 Write Back

Not implemented as a single module, since after MEM/WB, we have got all the information `register.v` needs, register file can finish the write back stage.

4.1.10 Register File

Implemented in `register.v`. The main function of this module is to simulate register storage and deal with write and read operations.

4.1.11 CPU

implemented in `cpu.v`. This module build connections between pipeline modules above, form data path and control path.

4.2 Memory Modules

4.2.1 RAM

Implemented in `ram.v`. This module simulates memory storage and deal with memory access.

4.2.2 ROM

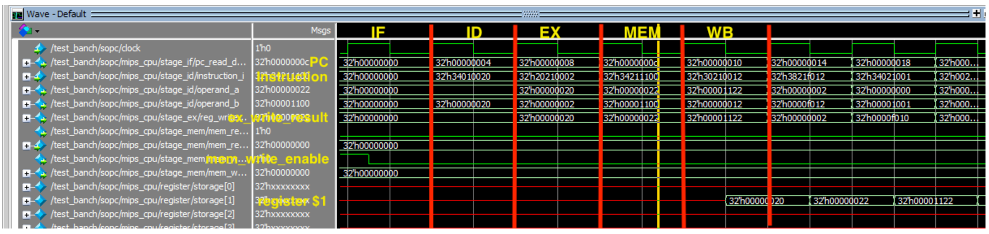
Implemented in `rom.v`. This module simulates instruction storage and deal with instruction access.

4.3 SOPC Module

Implemented in `sopc.v`.This module build connections between Pipeline modules and Memory modules.

5 Test Summary

All test code is available on https://github.com/Nerer/MIPS-CPU/tree/master/test. They are from Zhiyong Fang and Lianmin Zheng. The picture of test result is too large to display in this report. I have simulate the complex test data in presentation. So here I just show a simple example 5.1. We can see clearly five stages in it.



```
/test_banch/sopc/clock                              1'h0
/test_banch/sopc/mips_cpu/stage_if/pc_read_d...     32'h0000000c   32'h00000000  32'h00000004  32'h00000008  32'h0000000c  32'h00000010  32'h00000014  32'h00000018  32'h000...
/test_banch/sopc/mips_cpu/stage_id/instruction_i                   32'h00000000  32'h34010020  32'h20210002  32'h34211100  32'h30210012  32'h3821f012  32'h34021001  32'h002...
/test_banch/sopc/mips_cpu/stage_id/operand_a        32'h00000022   32'h00000000                32'h00000020  32'h00000022  32'h00001122  32'h00000002  32'h00000000  32'h000...
/test_banch/sopc/mips_cpu/stage_id/operand_b        32'h00001100   32'h00000000  32'h00000020  32'h00000002  32'h00001100  32'h00000012  32'h0000f012  32'h00001001  32'h000...
/test_banch/sopc/mips_cpu/stage_ex/reg_wr...                       32'h00000000                32'h00000020  32'h00000022  32'h00001122  32'h00000002  32'h0000f010  32'h000...
/test_banch/sopc/mips_cpu/stage_mem/mem_re...       1'h0
/test_banch/sopc/mips_cpu/stage_mem/mem_re...       32'h00000000   32'h00000000
/test_banch/sopc/mips_cpu/stage_mem...
/test_banch/sopc/mips_cpu/stage_mem/mem_w...        32'h00000000   32'h00000000
/test_banch/sopc/mips_cpu/register/storage[0]       32'hxxxxxxxx
/test_banch/sopc/mips_cpu/register/storage[1]                                                                           32'h00000020  32'h00000022  32'h00001122
/test_banch/sopc/mips_cpu/register/storage[2]       32'hxxxxxxxx
```

Figure 5.1: Simulation

6 Further Enhancement

Out of order execution. Traditional 5-stage MIPS pipeline is in-order execution. I think it is difficult to do improvement. Maybe we can use out of order execution such as Toma-sulo to improve the performance. But I didn't have enough time and ability to do this because I am really not familiar with Verilog.