ercesiMIPS Lab Report

Lab1 Single Cyclic CPU with 7-9(11) MIPS Instructions

Name: Firstname Lastename
StudentID: xxxxxx
Class#: xxxxx

CS 11007 Computer Organization and Architecture
(Spring, 2017)

Northwestern Polytechnical University, China
Faculty of Computer Science
ERCESI

27 April 2017

Abstract

Please enter the abstract here, just summary your lab works.

Contents

1 Overview

The structure of Single cyclic MIPS CPU has been introduced in CS 11007 class lecture.

- **Supported Instructions** includes *sub, add, or, ori, lw, sw, lst, beq, j*, or adding *addi, and, andi* for better programming experience in assembly. All these instructions can be supported without exception detecting (overflow detecting)
- **All instructions work in one cycle.** For the very beginning stage, Single Cyclic CPU model is a great example to explain how CPU works.
- **Consisted of Data Path, Control Unit and Memory Unit.** To illustrate the typical systematic idea of computer, we recommend you design your first CPU with two separated modules, CPath and DPath, in such coding style, both blocks can also be easily verified separately. Additionally, if more complement MIPS ISA is chosen, this structure will be high efficient to be extended.
- **Chisel3 is also recommended.** Chisel is a powerful structural hardware description language, with more efficient expression for block, operation, and IO bundles compared with Verilog. However, the most significant feature of Chisel is that it can express the structure of system without detailed circuits coding. Further more, we prefer Chisel3 instead of Chisel2, which relies on verilator for verilog simulations instead of Synopsys vcs. The difference between these tow versions can be referenced here: `https://github.com/ucb-bar/chisel3/wiki/Chisel3-vs-Chisel2`.

2 System Design

2.1 System Overview

Please describe your CPU system design here, Figures are recommended for detailed illustration and add the figure using latex could be reference the insertion of Fig. 1. If you need cite a reference like this [1], and a sample bibliography

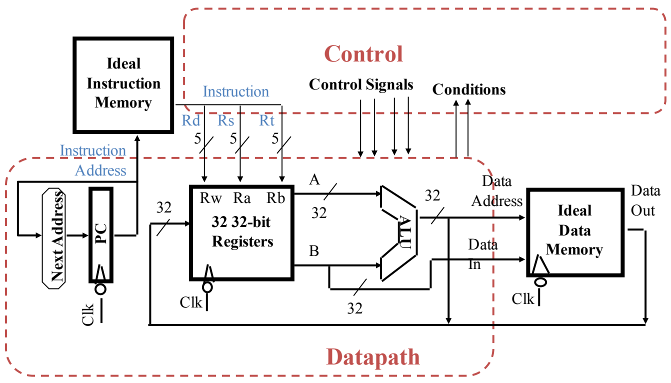

Figure 1: Single Cyclic CPU Block Diagram

item is attached at the end.

2.2 Interface Definition

In this section, the interfaces among the blocks of top level should be described in details. Tables can be used in latex as this. Table 1

2.3 (Sequential) Logic of Interface

Please define the logic of interfaces of your top level. Figures and tables also will be helpful for expressing ideas in academic style.

3 Blocks Design

This section is for detailed introduction of block function, interface definition, logic implementation (FSM design), etc. for every single block.

Table 1: Signals Definition for Test Mode

Signal Name	**Direction**	**Width**	**Function**
boot	Input	1-bit	Trigger the boot test mode, set to 0 in CPU regular process mode
test_im_wr	Input	1-bit	Instruction memory write enable in test mode,set to 0 in CPU regular process mode. In test mode, it will be set to 1 when if writing instructions to imem, otherwise it is set to 0.
test_im_re	Input	1-bit	Instruction memory read enable in test mode,set to 0 in CPU regular process mode. In test mode, it will be set to 1 when if reading instructions out, otherwise it is set to 0.
test_im_addr	Input	32-bit	Instruction memory address
test_im_in	Input	32-bit	Instruction memory data input for test mode.
test_im_out	Output	32-bit	Instruction memory data output for test mode.
test_dm_wr	Input	1-bit	Data memory write enable in test mode,set to 0 in CPU regular process mode. In test mode, it will be set to 1 when if writing data to dmem, otherwise it is set to 0.
test_dm_re	Input	1-bit	Data memory read enable in test mode,set to 0 in CPU regular process mode. In test mode, it will be set to 1 when if reading data out, otherwise it is set to 0.
test_dm_addr	Input	32-bit	Data memory address
test_dm_in	Input	32-bit	Data memory input for test mode.
test_dm_out	Output	32-bit	Data memory output for test mode.
valid	Output	1-bit	If CPU stopped or any exception happens, valid signal is set to 0.

3.1 ALU

3.1.1 Function

3.1.2 Interface Definition

3.1.3 Logic Design

3.2 Control Unit

3.2.1 Function

3.2.2 Interface Definition

3.2.3 Logic Design

3.3 Data Path

3.3.1 Function

3.3.2 Interface Definition

3.3.3 Logic Design

4 Lab Records

The whole lab process, all design events, problems and relevant solutions are described here. Name your Subsections according to demand.

Appendix A Code

Please add your code with Scala syntax highlight support like below.This is app A

```scala
class TopIO extends Bundle() {
  val boot = Input(Bool())
// imem and dmem interface for Tests
  val test_im_wr    = Input(Bool())
  val test_im_rd    = Input(Bool())
  val test_im_addr  = Input(UInt(32.W))
  val test_im_in    = Input(UInt(32.W))
  val test_im_out   = Output(UInt(32.W))

  val test_dm_wr    = Input(Bool())
  val test_dm_rd    = Input(Bool())
  val test_dm_addr  = Input(UInt(32.W))
  val test_dm_in    = Input(UInt(32.W))
  val test_dm_out   = Output(UInt(32.W))

  val valid     = Output(Bool())
}
class Top extends Module() {
  val io    = IO(new TopIO())//in chisel3, io must be wrapped in IO(...)
  //...
  when (io.boot & io.test_im_wr){
    imm(test_im_addr) := test_im_in
    } .elsewhen (io.boot & io.test_dm_wr){
    // please finish it
    } //...
}
```

References

[1] P. Erdős, *A selection of problems and results in combinatorics*, Recent trends in combinatorics (Matrahaza, 1995), Cambridge Univ. Press, Cambridge, 2001, pp. 1–6.