

SHANGHAI JIAOTONG UNIVERSITY



计算机系统结构实验报告 – Lab4

姓名：刘欣鹏

学号：516030910259

完成时间：2017/3/21

目录

[1. 概述 3](#_Toc479014376)

[1.1 实验名称 3](#_Toc479014377)

[1.2 实验目的 3](#_Toc479014378)

[1.3实验范围 3](#_Toc479014379)

[2. 实验描述 3](#_Toc479014380)

[2.1 寄存器单元模块REGISTER 3](#_Toc479014381)

[2.1.1模块描述 3](#_Toc479014382)

[2.1.2 register模块代码 4](#_Toc479014383)

[2.1.3 仿真测试代码 5](#_Toc479014384)

[2.1.4 仿真波形 6](#_Toc479014385)

[2.1.5 实验结论 6](#_Toc479014386)

[2.2 内存单元模块MEMORY 6](#_Toc479014387)

[2.2.1 模块描述 6](#_Toc479014388)

[2.2.2 Memory单元模块代码 6](#_Toc479014389)

[2.2.3 仿真测试代码 6](#_Toc479014390)

[2.2.4 仿真波形 7](#_Toc479014391)

[2.2.5 实验结论 7](#_Toc479014392)

[2.3 带符号扩展 7](#_Toc479014393)

[2.3.1 模块描述 8](#_Toc479014394)

[2.3.2带符号扩展模块代码 8](#_Toc479014395)

[2.3.3 仿真测试代码 8](#_Toc479014396)

[2.3.4 仿真波形 8](#_Toc479014397)

[2.3.5 实验结论 8](#_Toc479014398)

[3. 实验心得 8](#_Toc479014400)

# 1. 概述

## 1.1 实验名称

简单的类MIPS单周期处理器实现——寄存器与内存

## 1.2 实验目的

1、理解CPU的寄存器与内存

2、使用Verilog语言设计存储器件

3、使用ISim进行行为仿真

## 1.3实验范围

本次实验将覆盖以下范围:

(1) ISE的使用

(2) Register的实现

(3) Data Memory的实现

(4)有符号拓展的实现

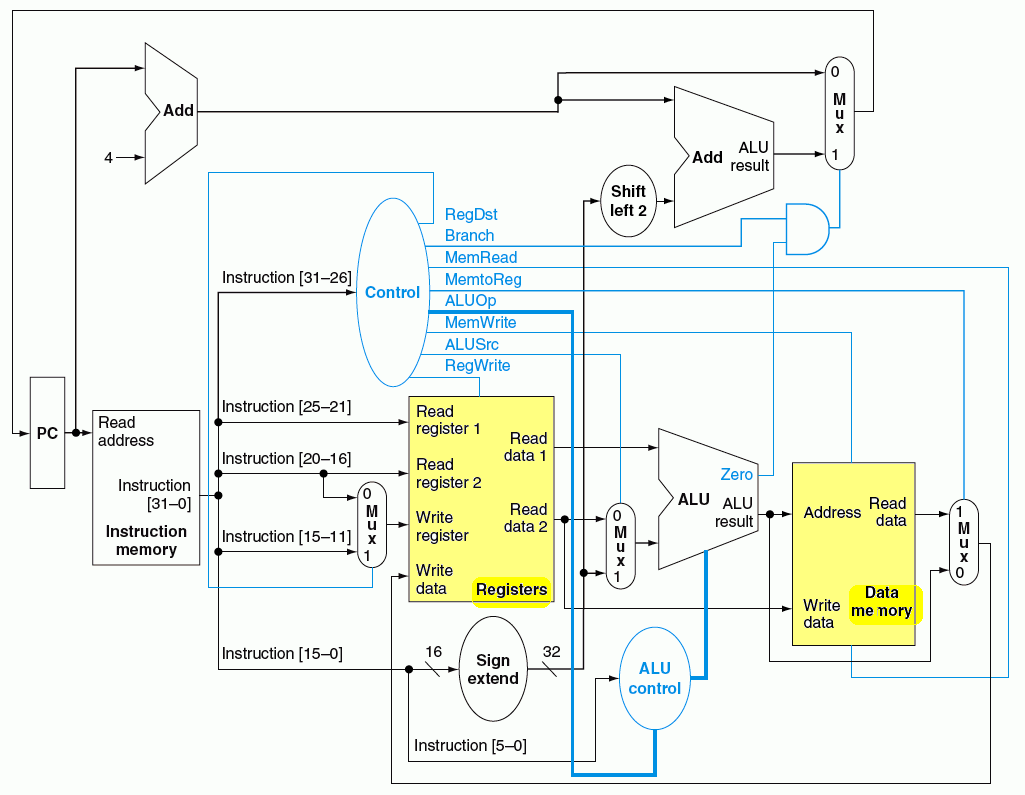
# 2. 实验描述

## 2.1 寄存器单元模块REGISTER

### 2.1.1模块描述

寄存器是指令操作的主要对象，MIPS中一共有32个32位的寄存器。

由于不确定writeReg，writeData和regWrite的信号顺序，统一采用时钟信号的下降沿作为写操作的同步信号，防止发生错误。



Mips处理器基本架构图——寄存器和存储器单元

### 2.1.2 register模块代码

**`timescale 1ns / 1ps**

**module register(**

**input clock\_in,**

**input [25:21] readReg1,**

**input [20:16] readReg2,**

**input [4:0] writeReg,**

**input [31:0] writeData,**

**input regWrite,**

**output [31:0] readData1,**

**output [31:0] readData2**

**);**

**reg [31:0] regFile[31:0];**

**reg [31:0] readData1,readData2;**

**initial begin**

**regFile[0]=0;**

**end**

**always @ (readReg1 or readReg2)**

**begin**

**readData1=regFile[readReg1];**

**readData2=regFile[readReg2];**

**end**

**always @ (negedge clock\_in)**

**begin**

**if (writeReg!=0 && regWrite==1)**

**regFile[writeReg]=writeData;**

**end**

**endmodule**

### 2.1.3 仿真测试代码

仿真代码主要部分如下，完整代码见test\_for\_register.v。

**always #100 clock\_in=~clock\_in;**

**initial begin**

**// Initialize Inputs**

**clock\_in = 0;**

**readReg1 = 0;**

**readReg2 = 0;**

**writeReg = 0;**

**writeData = 0;**

**regWrite = 0;**

**#285;**

**regWrite=1;writeReg=5'b10101;**

**writeData=32'b11111111111111110000000000000000;**

**#200;**

**writeReg=5'b01010;**

**writeData=32'b00000000000000001111111111111111;**

**#200;**

**regWrite=0;writeReg=5'b00000;**

**writeData=32'b00000000000000000000000000000000;**

**#500;**

**readReg1=5'b10101;**

**readReg2=5'b01010;**

**#285;**

**regWrite=1;writeReg=5'b10101;**

**writeData=32'b11111111111111110000000000000000;**

**#200;**

**writeReg=5'b11010;**

**writeData=32'b10000000000000001111111111111111;**

**#200;**

**regWrite=0;writeReg=5'b01000;**

**writeData=32'b00000000001000000000000000000000;**

**#50;**

**readReg1=5'b11010;**

**readReg2=5'b01010;**

**#285;**

**regWrite=1;writeReg=5'b11101;**

**writeData=32'b11111111010111110000000000000000;**

**#200;**

**writeReg=5'b01010;**

**writeData=32'b00000000000000001111111111111111;**

**#200;**

**regWrite=0;writeReg=5'b00000;**

**writeData=32'b00000000000000000000000000000000;**

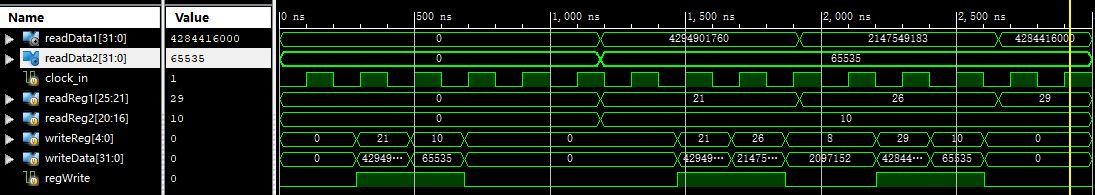
**#50;**

**readReg1=5'b11101;**

**readReg2=5'b01010;**

**end**

### 2.1.4 仿真波形



Register模块仿真波形

### 2.1.5 实验结论

仿真波形说明Register模块能够正确地根据readReg信号读取寄存器，并在时钟的下降沿根据regWrite，writeReg将writeData写入寄存器。Register单元模块实验成功。

## 2.2 内存单元模块MEMORY

### 2.2.1 模块描述

内存模块与register模块相似，由于写数据也要考虑信号同步，因此也需要clock\_in。内存单元的实现，也可以用系统Block Memory来生成。

### 2.2.2 Memory单元模块代码

**`timescale 1ns / 1ps**

**module data\_memory(**

**input clock\_in,**

**input [6:0] address,**

**input [31:0] writeData,**

**input memRead,**

**input memWrite,**

**output [31:0] readData**

**);**

**reg [31:0] memFile[0:127];**

**reg [31:0] readData;**

**always @(memRead)**

**begin**

**if (memRead==1) readData=memFile[address];**

**else readData=0;**

**end**

**always @(negedge clock\_in)**

**begin**

**if (memWrite==1) memFile[address]=writeData;**

**end**

**endmodule**

### 2.2.3 仿真测试代码

仿真代码主要部分如下，完整代码见test\_for\_memory.v。

**always #100 clock\_in=~clock\_in;**

**initial begin**

**// Initialize Inputs**

**clock\_in = 0;**

**address = 0;**

**writeData = 0;**

**memRead = 0;**

**memWrite = 0;**

**// Wait 100 ns for global reset to finish**

**#185;**

**memWrite=1;**

**address=7'b0001111;**

**writeData=32'b11111111111111110000000000000000;**

**#250;**

**memRead=1;**

**memWrite=0;**

**#440;**

**memRead=1;**

**address=17;**

**writeData=45536;**

**#150;**

**memWrite=1;**

**#200**

**memWrite=0;**

**memRead=1;**

**#200**

**address=15;**

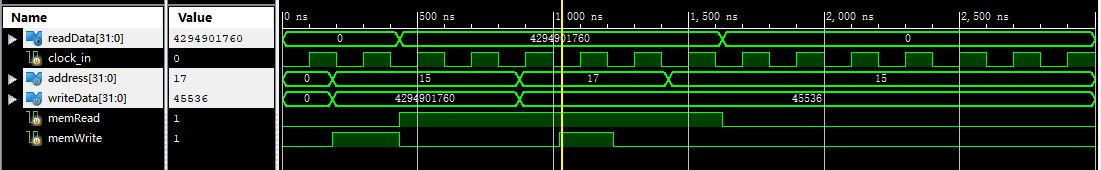
**#200**

**memRead=0;**

**// Add stimulus here**

**end;**

### 2.2.4 仿真波形



Data\_Memory模块仿真波形

### 2.2.5 实验结论

仿真波形说明Data\_Memory模块能够正确地根据memRead，address信号读取内存，并在时钟的下降沿根据memWrite，address信号将writeData写入内存。由于我在内存模块进行了初始化，在开始时刻readData的取值为0。内存单元模块实验成功。

## 2.3 带符号扩展

### 2.3.1 模块描述

将16位有符号数扩展为32位有符号数。

补码：

（1）正数的补码：与原码相同

（2）负数的补码：符号位为1，其余位为该数绝对值的原码按位取反，然后整个数加1。

带符号扩展只需要在前面补足符号即可。

### 2.3.2带符号扩展模块代码

**module signext(**

**input [15:0] inst,**

**output [31:0] data);**

**assign data={{16{inst[15]}},inst[15:0]};**

**endmodule**

### 2.3.3 仿真测试代码

仿真代码主要部分如下，完整代码见test\_for\_signext.v。

**initial begin**

**// Initialize Inputs**

**inst=0;**

**// Wait 100 ns for global reset to finish**

**#100;inst=-50;**

**#100;inst=200;**

**#100;inst=0;**

**#100;inst=7;**

**#100;inst=-7;**

**end**

### 2.3.4 仿真波形



带符号扩展模块仿真波形

### 2.3.5 实验结论

仿真波形说明带符号扩展模块能够成功将16位补码表示拓展为32位补码，带符号扩展模块实验成功。

# 3. 实验心得

本次实验相对来说比较基础，它主要要求我们实现计算机硬件中寄存器、内存这些存储器，以及带符号扩展器的接口。尽管如此，实验中也有许多值得注意的地方，比如寄存器和内存写入需要与时钟信号保持一致，而读取操作则没有相应的要求。同时，这一部分的实现过程也使我接触到了阻塞赋值和非阻塞赋值、组合逻辑与时序逻辑的区分，使我有了更深入地理解。

经过查阅资料，我发现了Verilog语言的拼接语句。这个语句使带符号扩展部分的代码更加精简。这个经历也使我对Verilog语言有了更多的认识。

这次实验的成功除了个人的努力外，也得益于同学之间的讨论、指导老师的讲解和实验指导书的详尽叙述。在此感谢老师和同学们的帮助。