上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

计算机系统结构实验报告lab4

姓　　名：	张卫鸣
学　　号：	520021911141
专　　业：	计算机科学与技术

2022 年 3 月 22 日

计算机系统结构实验lab4

目录

一、实验概述

（一）实验名称

简单的类MIPS单周期处理器功能部件的设计与实现（二）

（二）实验目的

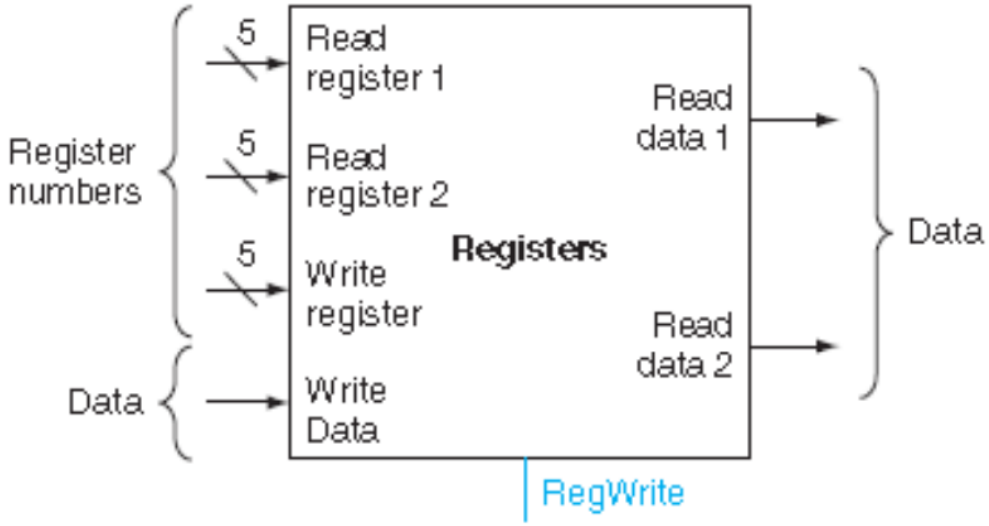
1. 理解寄存器、数据存储器、有符号扩展单元的I/O定义
2. Registers的设计实现
3. Data Memory的设计实现
4. 有符号扩展部件的实现
5. 对功能模块进行仿真

二、实验描述

（一）寄存器模块Registers

1. 寄存器模块描述

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，用作数据的缓存。寄存器的模块定义如下图1所示：



在本实验中，需要使用5位选择信号对寄存器进行寻址。每个寄存器的长度为32位，对应处理器的一个字长，因此数据的输入和输出均为32位。将寄存器模块的输入输出信号进行整理，如下表1所示：

信号	输入/输出	长度	说明
readReg1	input	5	读取寄存器1的编号
readReg2	input	5	读取寄存器2的编号
writeReg	input	5	写入寄存器编号
writeData	input	32	写入数据
regWrite	input	1	写信号
clk	input	1	时钟信号
readData1	output	32	读取寄存器1的结果
readData2	output	32	读取寄存器2的结果

表1

寄存器读取数据的操作一直进行，写数据的操作在时钟下降沿进行。如此设计可以防止在一个周期开始时读取错误的数据导致执行结果错误。

2. 寄存器模块代码

实现寄存器模块的代码如下：

```
module Registers(
    input [25:21] readReg1,
    input [20:16] readReg2,
    input [4:0] writeReg,
    input [31:0] writeData,
    input regWrite,
    input clk,
    output reg [31:0] readData1,
    output reg [31:0] readData2
    );
    reg [31:0] regFile [31:0];
    always @ (readReg1 or readReg2)
    begin
        readData1=regFile[readReg1];
        readData2=regFile[readReg2];
    end
    always @ (negedge clk)
    begin
        if(regWrite)
            regFile[writeReg]=writeData;
        end
endmodule
```

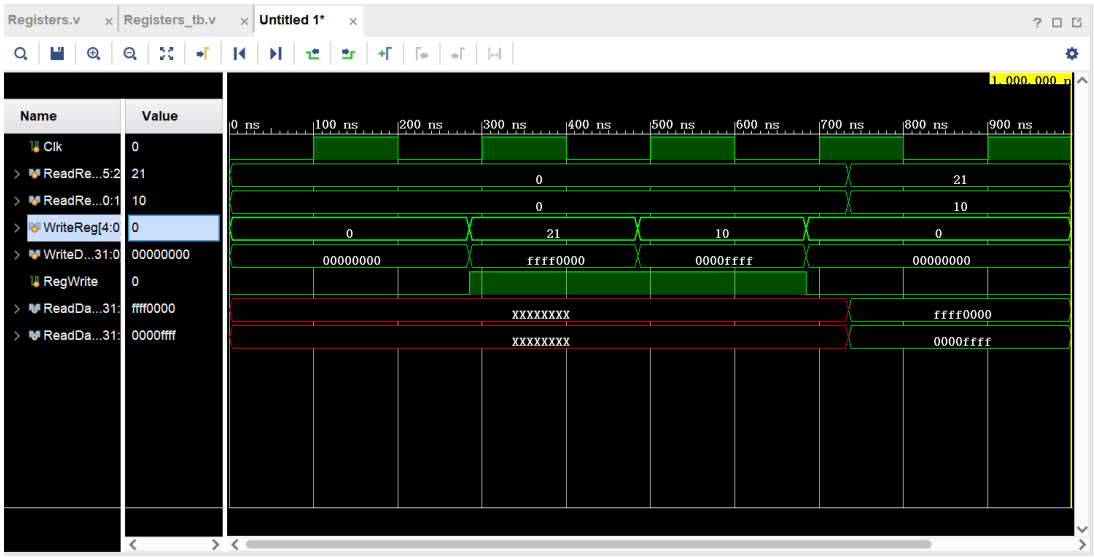
3. 仿真测试代码

仿真代码如下：

```
module Registers_tb(

    );
    reg [25:21] ReadReg1;
    reg [20:16] ReadReg2;
    reg [4:0] WriteReg;
    reg [31:0] WriteData;
    reg RegWrite;
    reg Clk;
    wire [31:0] ReadData1;
    wire [31:0] ReadData2;
    Registers u0(.readReg1(ReadReg1),
                 .readReg2(ReadReg2),
                 .writeReg(WriteReg),
                 .writeData(WriteData),
                 .regWrite(RegWrite),
                 .clk(Clk),
                 .readData1(ReadData1),
                 .readData2(ReadData2)
    );
    always #100 Clk=~Clk;
    initial begin
        Clk=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;

        #50;
        ReadReg1=5'b10101;
```

```
45          ReadReg2=5'b01010;
46      end
47  endmodule
```

4. 仿真波形

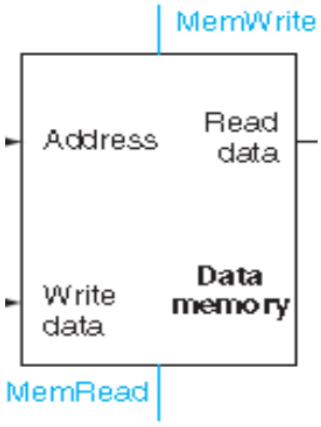


我们测试的波形结果里有着红色的x，这是因为寄存器数据没有初始化，按照实验指导书上的步骤没有这样的要求，所以我并没有对其进行初始化。其他的部分执行结果都是正确的，说明寄存器的实现是正确的。

（二）数据存储器模块dataMemory

1. 数据存储器模块描述

Data memory需要支持数据读取、数据写入的功能，主要接口如图3所示。内存单元模块同样按字进行寻址，不考虑页表、虚拟地址与物理地址的转化，只考虑直接操作物理地址的情况。我们设定有效内存大小的上限为64个字。数据存储器的输入输出信号如表2所示。



信号	输入/输出	长度	说明
clk	input	1	时钟信号
address	input	32	内存地址
writeData	input	32	写入的数据
memWrite	input	1	写信号
memRead	input	1	读信号
readData	output	32	内存读取数据

表2

同样，存储器的写数据操作仅在时钟下降沿进行。

2. 数据存储器模块代码

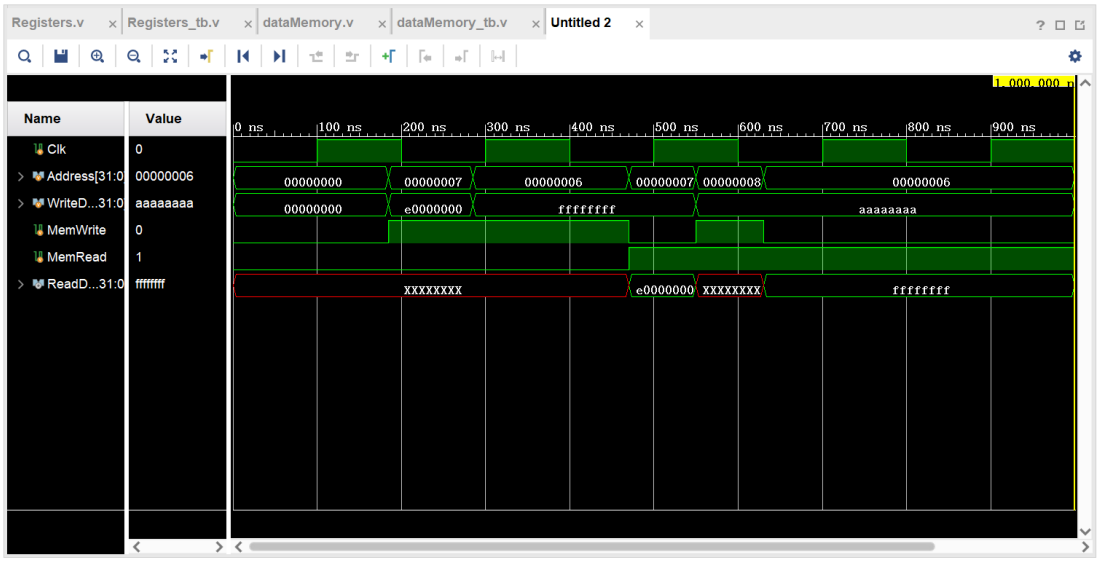
```
module dataMemory(
    input clk,
    input [31:0] address,
    input [31:0] writeData,
    input memWrite,
    input memRead,
    output reg [31:0] readData
    );
    reg [31:0] memFile [0:63];
    always @ (memRead or address)
    begin
        if(memRead)
        begin
            if(address<=63)
                readData=memFile[address];
            else
                readData=0;
        end
    end
    always @ (negedge clk)
    begin
        if(memWrite&&address<=63)
            memFile[address]=writeData;
    end
endmodule
```

3. 仿真测试代码

```
module dataMemory_tb(

    );
    reg Clk;
```

```
    reg [31:0] Address;
    reg [31:0] WriteData;
    reg MemWrite;
    reg MemRead;
    wire [31:0] ReadData;
    dataMemory u(.clk(Clk),
                 .address(Address),
                 .writeData(WriteData),
                 .memWrite(MemWrite),
                 .memRead(MemRead),
                 .readData(ReadData)
    );
    always #100 Clk=~Clk;
    initial begin
        Clk=0;
        Address=0;
        WriteData=0;
        MemWrite=0;
        MemRead=0;

        #185;
        MemWrite=1;
        Address=32'b00000000000000000000000000000111;
        WriteData=32'b11100000000000000000000000000000;

        #100;
        MemWrite=1;
        WriteData=32'hffffffff;
        Address=32'b00000000000000000000000000000110;

        #185;
        MemRead=1;
        MemWrite=0;
        Address=32'b00000000000000000000000000000111;

        #80;
        MemWrite=1;
        Address=8;
        WriteData=32'haaaaaaaa;

        #80;
        MemWrite=0;
        MemRead=1;
        Address=32'b00000000000000000000000000000110;
    end
endmodule
```

4. 仿真波形

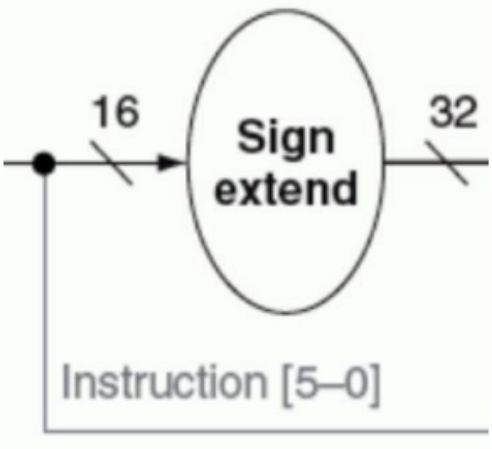


同样，我们的测试结果里出现了红色的x，也是由于存储器数据没有进行初始化；后面的x是由于我们将写操作延迟在时钟下降沿，读取的数据还没有初始化。这是合理的，且其他执行结果都正确。

（三）带符号扩展模块

1. 带符号扩展模块描述

带符号扩展单元（Sign Extension）将16位有符号立即数扩展为32位有符号立即数（如图5所示）。在I型指令中，我们有时需要将指令中的16位有符号立即数扩展为32位有符号立即数。所采取的方法是对立即数最高位进行判断，如果是1便扩展前16位均为1，如果是0便扩展前16位均为0。



符号扩展模块的输入输出信号如表3所示。

信号	输入/输出	长度	说明
inst	input	16	指令中的立即数
data	output	32	扩展之后的结果

表3

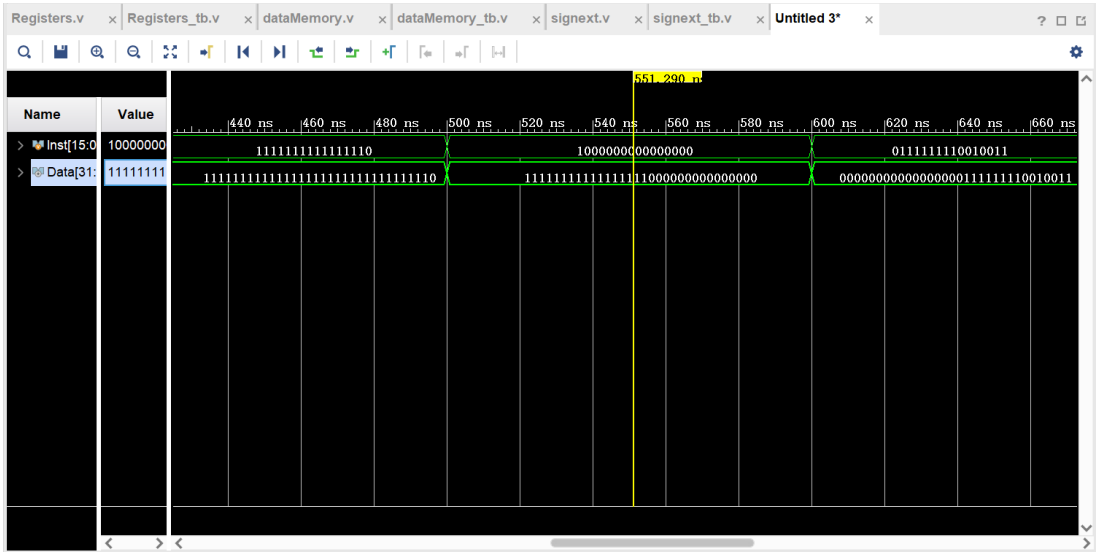
2. 带符号扩展模块代码

```
module signext(
    input [15:0] inst,
    output [31:0] data
    );
    assign data=inst[15]?{16'hffff,inst}:{16'h0000,inst};
endmodule
```

3. 仿真测试代码

```
module signext_tb(

    );
    reg [15:0] Inst;
    wire [31:0] Data;
    signext uu(.inst(Inst),
               .data(Data)
    );
    initial begin
        Inst=0;
        #100;
        Inst=1;
        #100;
        Inst=16'hffff;
        #100;
        Inst=2;
        #100;
        Inst=16'hfffe;
        #100;
        Inst=16'h8000;
        #100;
        Inst=16'h7f93;
    end
endmodule
```

4. 仿真波形



波形图只展示了我们仿真文件的部分结果，但是完整地呈现了符号扩展单元的功能。波形图中每一个16位输入的立即数都得到了正确的扩展，所以实现正确。

三、实验心得

本次实验的整体架构和lab3类似，都是完成MIPS单周期处理器的三个部件，寄存器模块、内存单元模块和符号扩展模块。在完成了lab3之后再做lab4，实验时的方向感就有了，做的总体还是比较顺利的。

有一个问题就是，在设置register和memory的写操作时，一开始并没有意识到是在时钟下降沿，后来才发现应该如此，这样防止读取时出现由于不同步造成的错误结果。

在这两次实验的过程中，我逐渐体会到了是如何一步步构建出一台MIPS处理器的——通过把一个巨大的复杂电路先实现其中的一些子元件的构造，而后再将线与端口相连。这样的思想让我对构建处理器有了更加清晰的认识，从而也为lab5做了一定的准备。

但是这些构建都是基于实验指导书所要求的几条较少的指令，功能还十分有限，在后期扩展指令的时候还需要将这些部件更新。

四、参考资料

2022计算机系统结构实验指导书lab4