Lab04 实验报告

王正 518021910079

一、 实验名称

简单的类 MIPS 单周期处理器部件实现——寄存器与存储器

二、 实验目的

1. 理解CPU 的寄存器、存储器、有符号扩展
2. Register 的实现
3. Data Memory 的实现
4. 有符号扩展的实现
5. 使用行为仿真

三、 功能实现

1. 寄存器 Registers

寄存器的读取是组合逻辑，写入是时序逻辑，所以只要遇到 readReg 信号就可以读取，在下降沿进行写入操作。

```
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];

    integer i=0;
    initial begin
     for(i=0;i<32;i=i+1)
            begin
                regFile[i]=0;
            end
    end
```

```
    always @(readReg1 or readReg2 or writeReg)
    begin
        readData1=regFile[readReg1];
        readData2=regFile[readReg2];
    end

    always @(negedge Clk)
    begin


        if(regWrite)
            begin
                regFile[writeReg]=writeData;
            end
    end
endmodule
```

2. 数据存储器模块 dataMemory

数据存储器模块和寄存器模块类似，写数据也要考虑信号同步，实现代码和寄存器基本相同。

```
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[63:0];

    integer i=0;

    initial begin
        for(i=0;i<32;i=i+1)
        begin
            memFile[i]=0;
        end
    end


    always@ (memRead, address)
    begin
        readData=memFile[address];
    end

    always @(negedge Clk)
    begin
        if(memWrite==1'b1)
        begin
            memFile[address]=writeData;
        end
    end
endmodule
```

3. 符号扩展模块

判断一个 16 位带符号数是否表示的是负数，只要看其最高位是否为 1。如果为 1，那么扩展后的高 16 位全为 1，否则全为 0。按照此逻辑编写模块即可。

```
module signext(
    input [15:0] inst,
    output reg [31:0] data
    );
    always@(inst)
    begin
        if(inst[15] == 0)
        assign data = {16'b0000000000000000,inst};

        if(inst[15] ==1)
        assign data = {16'b1111111111111111,inst};
    end
endmodule
```

四、 激励文件

1. Registers_tb:

```
module Registers_tb(

    );
    reg Clk;
    reg [25:21] readReg1;
    reg [20:16] readReg2;
    reg [4:0] writeReg;
    reg [31:0] writeData;
    reg regWrite;

    wire [31:0] readData1;
    wire [31:0] readData2;

    Registers u0(
        .Clk(Clk),
        .readReg1(readReg1),
        .readReg2(readReg2),
        .writeReg(writeReg),
        .writeData(writeData),
        .readData1(readData1),
        .readData2(readData2),

        .regWrite(regWrite)
     );
     parameter DELAY=100;
     always #(DELAY) Clk=!Clk;


     initial begin
        Clk=0;
        readReg1=0;
        readReg2=0;
        writeReg=0;
        writeData=0;
        regWrite=0;

        #285;
        regWrite=1'b1;
        writeReg=5'b10101;
        writeData=32'b11111111111111110000000000000000;

        #200;
        writeReg=5'b01010;
        writeData=32'b00000000000000001111111111111111;

        #200;
        regWrite = 1'b0;
        writeReg = 5'b00000;
        writeData = 32'b00000000000000000000000000000000;

        #50;
        readReg1=5'b10101;
        readReg2=5'b01010;
    end
endmodule
```

2. dataMemory_tb

```
module dataMemory_tb(

    );
    reg Clk;
    reg [31:0] address;
    reg [31:0] writeData;
    reg  memWrite;
    reg memRead;

    wire [31:0] readData;

    dataMemory u0(
        .Clk(Clk),
        .address(address),
        .writeData(writeData),
        .memWrite(memWrite),
        .memRead(memRead),
        .readData(readData)
        );

       parameter DELAY=100;
     always #(DELAY) Clk=!Clk;


    initial begin
       Clk=0;
       address=0;
       writeData=0;
       memWrite=0;
       memRead=0;

       #185;
       memWrite =1'b1;
       address = 32'b00000000000000000000000000000111;
       writeData = 32'b11100000000000000000000000000000;

       #100;
       memWrite = 1'b1;
       writeData = 32'hffffffff;
       address = 32'b00000000000000000000000000000110;

       #185;
       memRead=1'b1;
       memWrite= 1'b0;
       address=7;
```

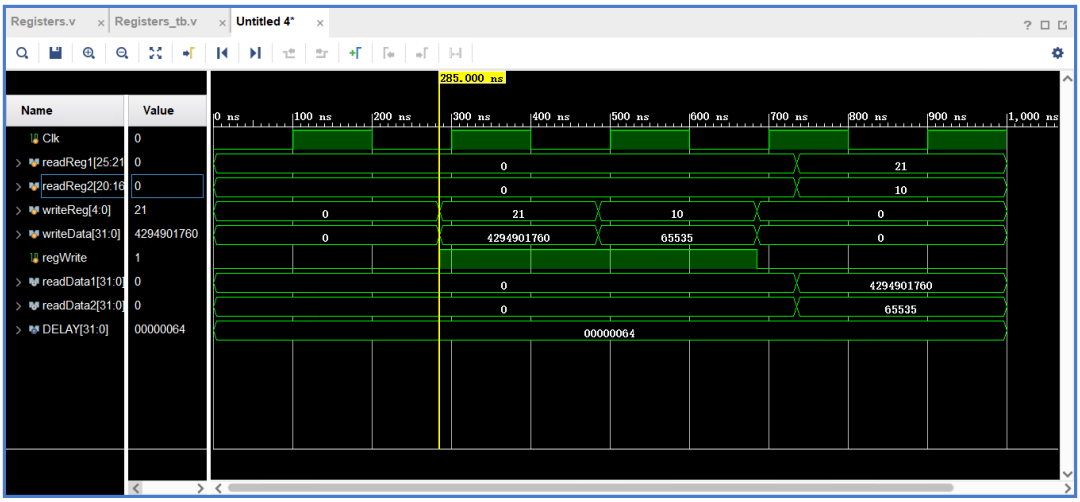
```
        #80;
        memWrite=1;
        address= 8;
        writeData=32'haaaaaaaa;

        #80;
        memWrite = 0;
        memRead = 1;
        address=6;

    end

endmodule
```

3. signext_tb

```
module signext_tb();
    reg [15:0] inst;
    wire [31:0] data;

    signext u0(
    .inst(inst),
    .data(data) );

    initial begin
        //inst=0;

       // #100;
        inst=16'b0000000000000000;
         #100;
        inst=16'b0000000000000001;
         #100;
        inst=16'b1111111111111111;
         #100;
        inst=16'b0000000000000010;
         #100;
        inst=16'b1111111111111110;
        #100;
    end
endmodule
```

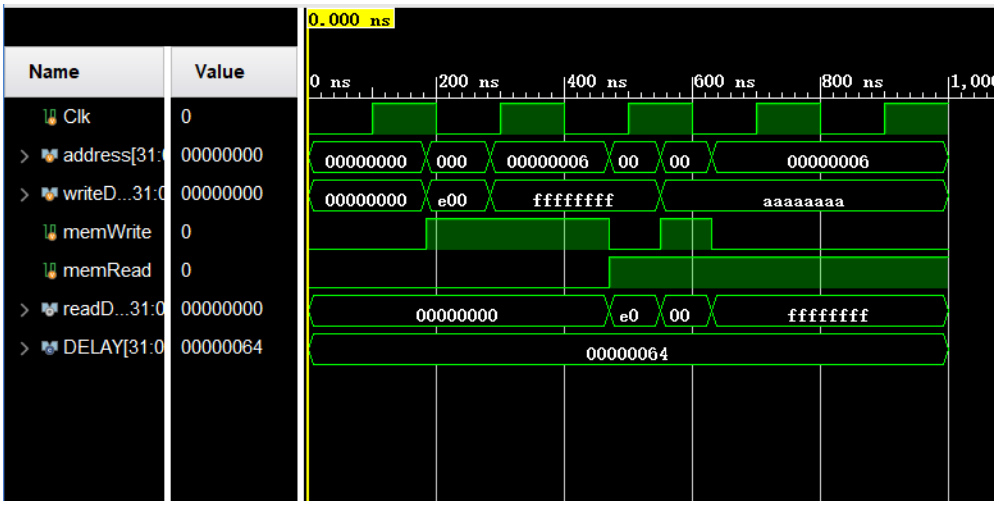
五、 结果展示

1. Registers 仿真波形



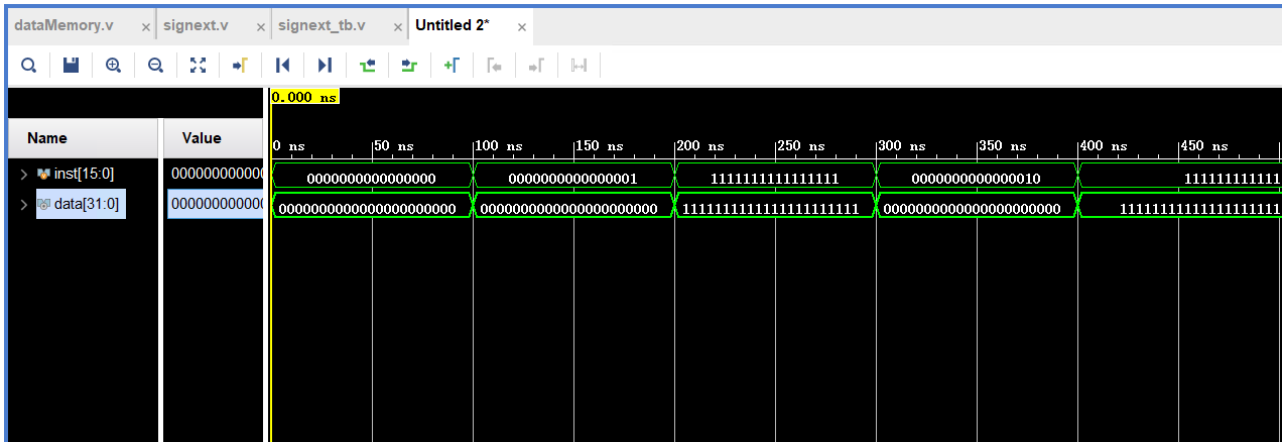
波形符合预期，仿真成功。

2. DataMemory 仿真波形



分别观察不同激励信号下存储器的输出信号的内部状态，符合预期。

3. signext 仿真波形



由波形可知，仿真结果满足预期设计。

六、 心得体会

本次实验实现了简单 MIPS 处理器中的存储器。通过这次实验，我认识到各模块中组合逻辑和时序逻辑的区别，使我对寄存器，数据存储器和符号位扩展的工作原理有了更进一步的认识。