计算机系统结构实验报告 实验四

惠宇龙 518030910059

1 实验概述

1.1 实验描述

简单的类 MIPS 单周期处理器部件实现——寄存器与存储器与有符号扩展。在本实验中先后实现了寄存器、存储器与有符号扩展三个基本的部件，也为后面深入设计复杂CPU的实验做好了铺垫。

1.2 实验目的

- 理解寄存器与存储器、有符号扩展
- 实现寄存器、数据存储器和符号扩展
- 使用行为仿真

2 寄存器模块

2.1 寄存器模块的设计原理

寄存器是有限存贮容量的高速存贮部件，它们可用来暂存指令、数据和地址，寄存器是指令操作的主要对象。MIPS 处理器中一共有 32 个 32 位通用寄存器。如下图所示，MIPS寄存器需要6个输入信号和2个读取数据的输出信号。

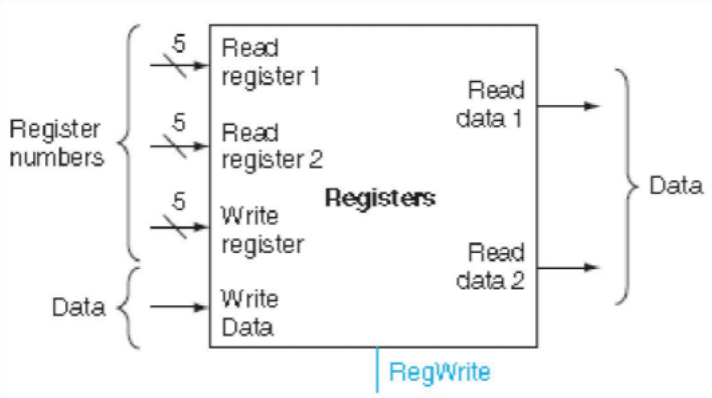


图1.寄存器模块示意图

2.2 寄存器模块的实现

按照指导书所给的模板，将具体的写入数据过程和读取数据的过程进行填充。在always中使用了无阻塞的赋值语句，以避免出现时序异常等错误。按照指导书的要求，在时钟下降沿向寄存器内写入新数据。

```
module Reg(
 input Clk,
 input [25:21] readReg1,
 input [20:16] readReg2,
 input [4:0] writeReg,
 input [31:0] writeData,
 input regwrite,
 output reg [31:0] readData1,
 output reg [31:0] readData2
    );

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

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

 always @ (negedge Clk)
 begin
 if(regwrite)
     regFile[writeReg]<= writeData;
 end

endmodule
```

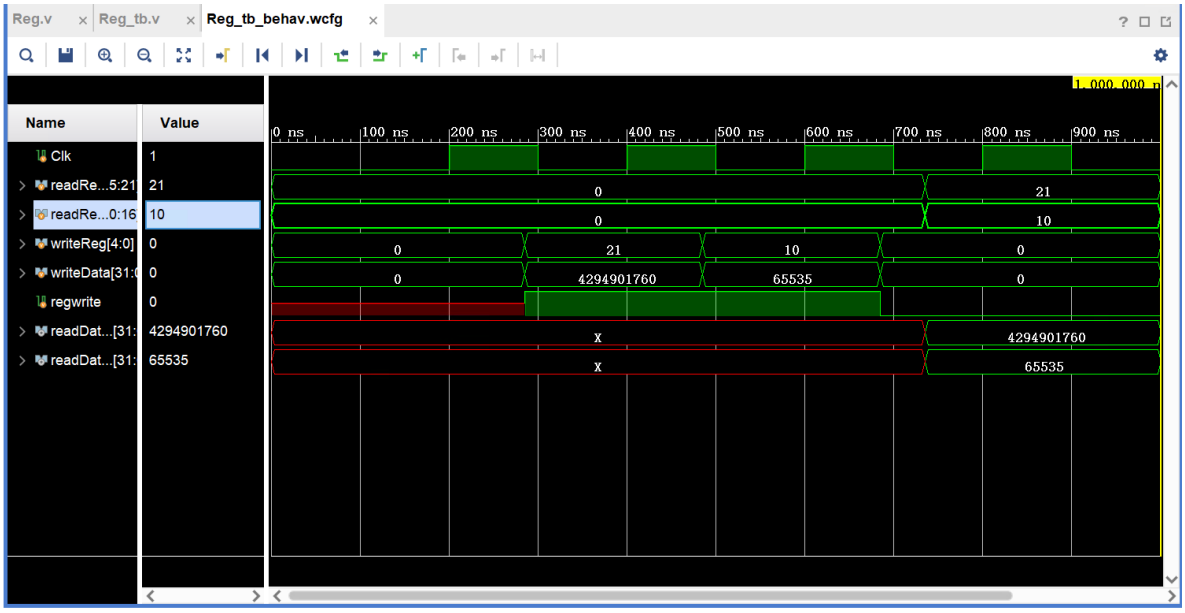
2.3 寄存器模块的验证

按照指导书的信息，补充激励文件的代码。（报告中略去了模块实例化的部分）

```
always
    #100 Clk=!Clk;
    initial begin
    writeData=0;
    writeReg=0;
    readReg1=0;
```

```
    readReg2=0;
    Clk=1'b0;
    #100 Clk=1'b0;
    #185;
    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
```

运行测试，可以得到下列波形



观察到该波形与参考书所给的波形一致，该模块实现成功。

3 存储器模块

3.1 存储器模块的设计原理

存储器本模块与 register 类似，在Verilog中，可以用reg进行表示。由于写数据也要考虑信号同步，因此也需要时钟信息，详细的端口情况如下图：

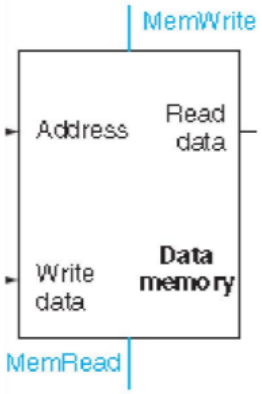


图2.存储器模块示意图

3.2 存储器模块的实现

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 DM(
    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 @ (address or memread or memwrite)
begin
    if(memread) readData<=memFile[address];
end

always @ (negedge Clk)
 begin
  if (memwrite)
      memFile[address]<=writeData;
 end

endmodule
```

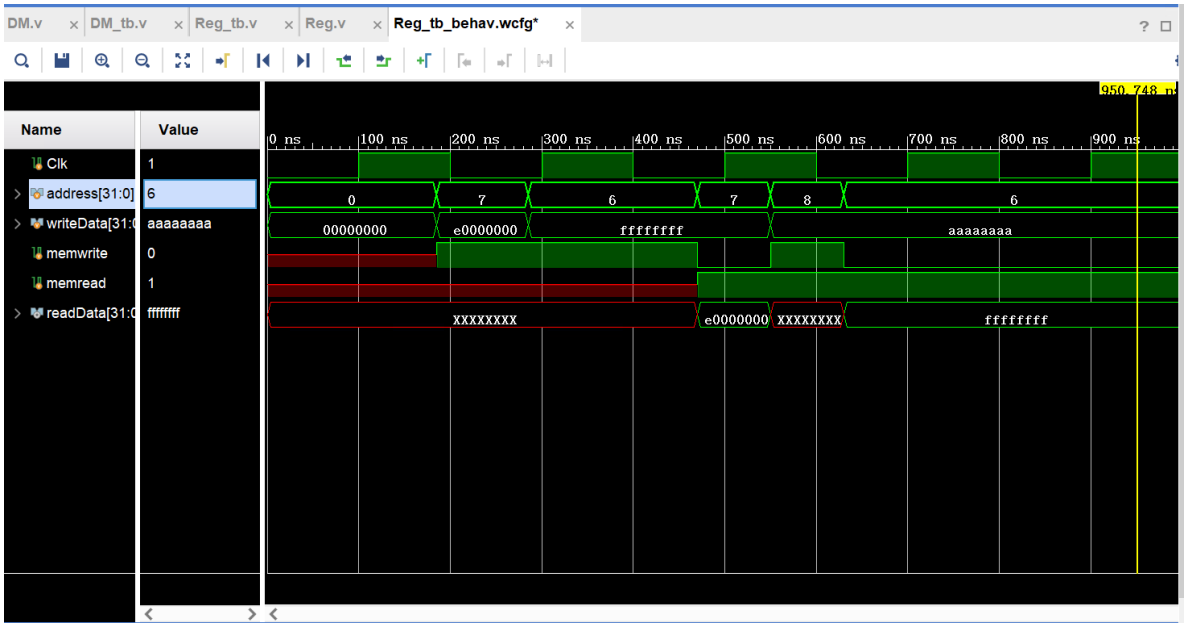
3.3 存储器模块的验证

按照参考书所给出的激励文件，测试存储器的读写情况。

```
    always #100 Clk=!Clk;

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

    #185;
    memwrite=1'b1;
    address=32'b00000000000000000000000000000111;
    writeData=32'he0000000;
```

```
    #100;
    memwrite=1'b1;
    writeData=32'hffffffff;
    address=32'h00000006;
    #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
```

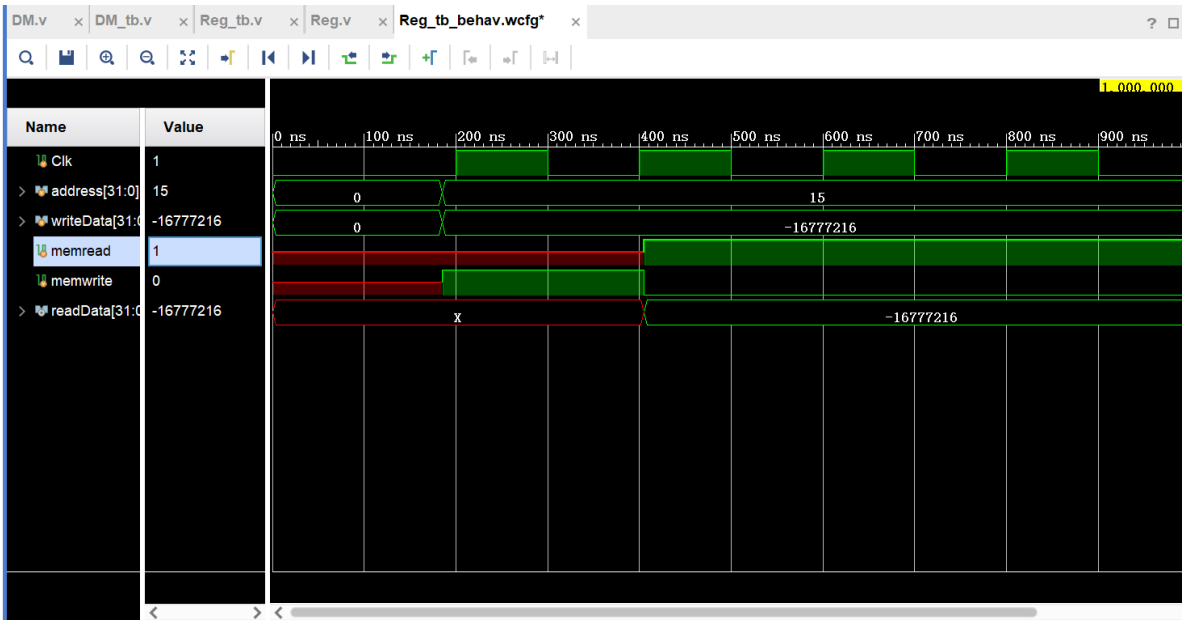
根据此激励文件可以得到如下波形：



另一种激励文件，呈现如下：

```
initial begin
    Clk=0;
    writeData=0;
    address=0;

    #100 Clk=0;
    #85
    memwrite=1;
    address=15;
    writeData=-16777216;

    #220 memread=1;
    memwrite=0;

    end
```

该激励文件可以得到如下的波形：



综上，得到的两种波形均与参考书所给示例一致。本模块实现成功。

4 有符号扩展模块

4.1 有符号扩展模块的设计原理

将16位有符号数扩展为32位有符号数。只需在16位有符号数前面补足符号即可。

4.2 有符号扩展模块的实现

该模块相对简单，需要先判断数据的符号，然后在前面补足该符号。

```
module sign(
    input [15:0]inst,
    output reg [31:0] data
    );

    always @ (inst)
    begin
    if (inst[15])
        data[31:16]<=16'b1111111111111111;
    else
        data[31:16]<=16'b0000000000000000;

        data[15:0]<=inst;
    end
endmodule
```

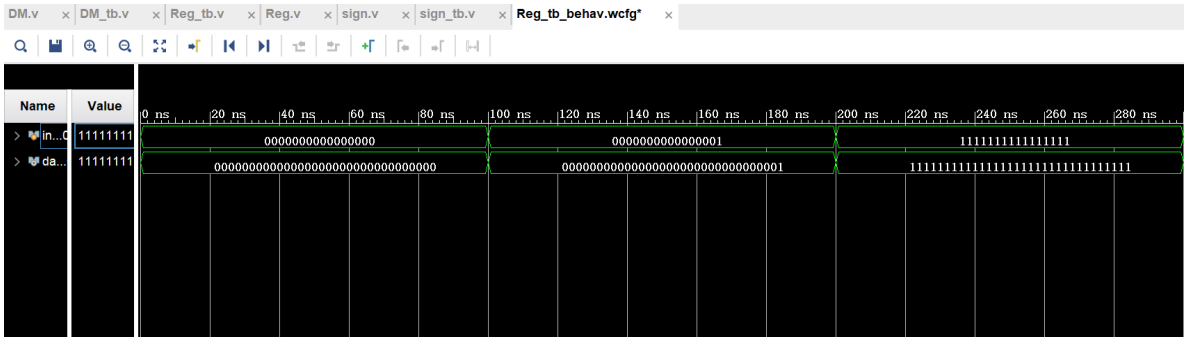
4.3 有符号扩展模块的验证

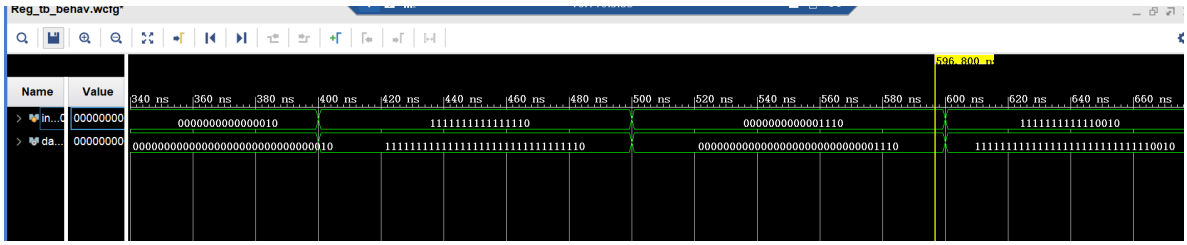
按照参考书所给示例的逻辑，编写激励文件：

```
module sign_tb(

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

```
    sign s0(
    .inst(inst),
    .data(data)
    );

    initial begin
    inst=0;
    #100
    inst=1;
    #100
    inst=-1;
    #100
    inst=2;
    #100
    inst=-2;
    #100
    inst=14;
    #100
    inst=-14;
    end

endmodule
```

该激励文件所产生的波形如下两张图所示：





可以看出，有符号数字已经被正确填充了符号，该模块实现成功。

5 总结反思

本实验实现了几个较为简单的小部件，为后续两个实验实现复杂的CPU做好了铺垫。

有符号扩展模块相对比较简单，需要注意到：我们要给data的高十六位、低十六位两部分端口分别赋值。

两个存储模块的实现是相似的，都是运用了多个reg来储存数据。（实际的存储器大小应当远大于代码所表示的大小，这里只是在原理上进行实现）。存储模块实现的关键点是要准确编写触发条件：当readReg改变或有新数据写入时，需要重新给输出端口赋值，两个触发条件缺一不可。另一方面，按照习惯，我们往往在时钟信号改变时写入数据，在这里统一选择了下降沿。

整体来说，本实验的难点并不多，但是存储单元是CPU中的重要组成，本实验让我对该部件有了更深入的理解。