Lab04实验报告

FPGA基础实验：简单的类MIPS单周期处理器实现–寄存器、存储器与有符号扩展

518021910489 陈沛宇

一、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。

二、实现步骤

1.寄存器

模块描述

寄存器是指令操作的主要对象，32 位的 MIPS 中共有 32 个 32 位的寄存器。

实现方法

当**readReg**信号被置1时即可进行读，但是由于**WriteReg,WriteData,RegWrite** 信号的先后次序不确定，会有信号同步问题。按照实验指导书上提供的方法，将时钟下降沿作为写操作的同步信号即可避免错误。

Register.v

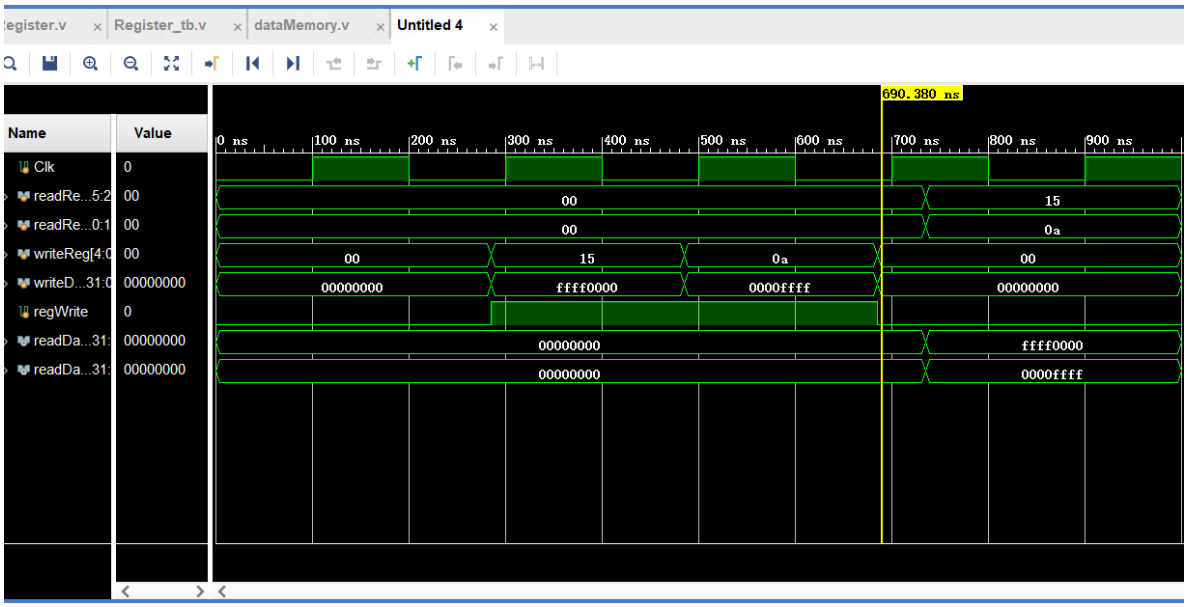
```
always @(readReg1 or readReg2 or writeReg)
        begin
            if (readReg1)
                readData1 = regFile[readReg1];
            else
                readData1 = 0;
            if (readReg2)
                readData2 = regFile[readReg2];
            else
                readData2 = 0;
        end
always @(negedge Clk)
        begin
            if (regWrite)
                regFile[writeReg] = writeData;
        end
```

采用实验指导书给出的激励文件如下。注意时钟信号是如何进行反转的。

Register_tb.v

```
always #100 Clk = !Clk;
    initial begin
        Clk = 0;
        readReg1 = 0;
        readReg2 = 0;
        regWrite = 0;
        writeReg = 0;
        writeData = 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
```

测试结果



测试结果与实验指导书一致，逻辑正确。

2.内存单元模块

模块描述

存储器本模块与**register**类似，由于写数据也要考虑信号同步，因此也需要时钟。内存单元的实现，也可用系统**Block Memory**来生成。

实现方法

主要考虑的问题仍然是信号先后次序不确定导致的信号同步的问题，可以按照寄存器实现的逻辑来实现内存单元模块。

dataMemory.v

```
always @(address)
        begin
            if(memRead && !memWrite)
                readData=memFile[address];
            else
                readData=0;
        end
    always @(negedge Clk)
        begin
            if(memWrite)
            memFile[address]=writeData;
        end
```

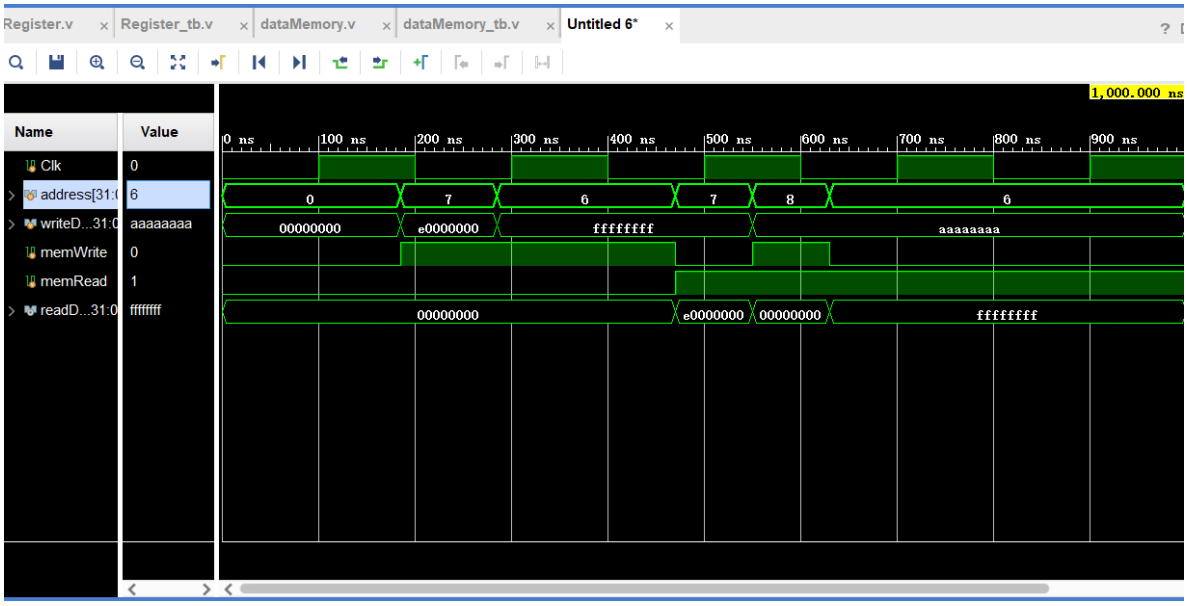
激励文件按照实验指导书给出的数据编写即可。留意到激励文件中在修改**memWrite**为1后并未修改**memRead**为0，考虑到代码稳定性反过去修改了**dataMemory**的第一个条件分支，将判断条件扩充为了两个。同样需要注意如何进行时钟信号反转。

dataMemory_tb.v

```
always #100 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
            address =6 ;
            memWrite = 0;
            memRead = 1;
        end
```

测试结果



测试结果与实验指导书的结果一样，逻辑正确。

3.带符号拓展

模块描述

带符号拓展模块实现将16位有符号数扩展为32位有符号数。在二进制数据表示中采用补码。正数的补码与原码相同，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。

实现方法

带符号数的符号位位最高位，只需单独对最高位进行判断然后将最高位复制到扩展后的16位即可，可以采用或运算实现。

signext.v

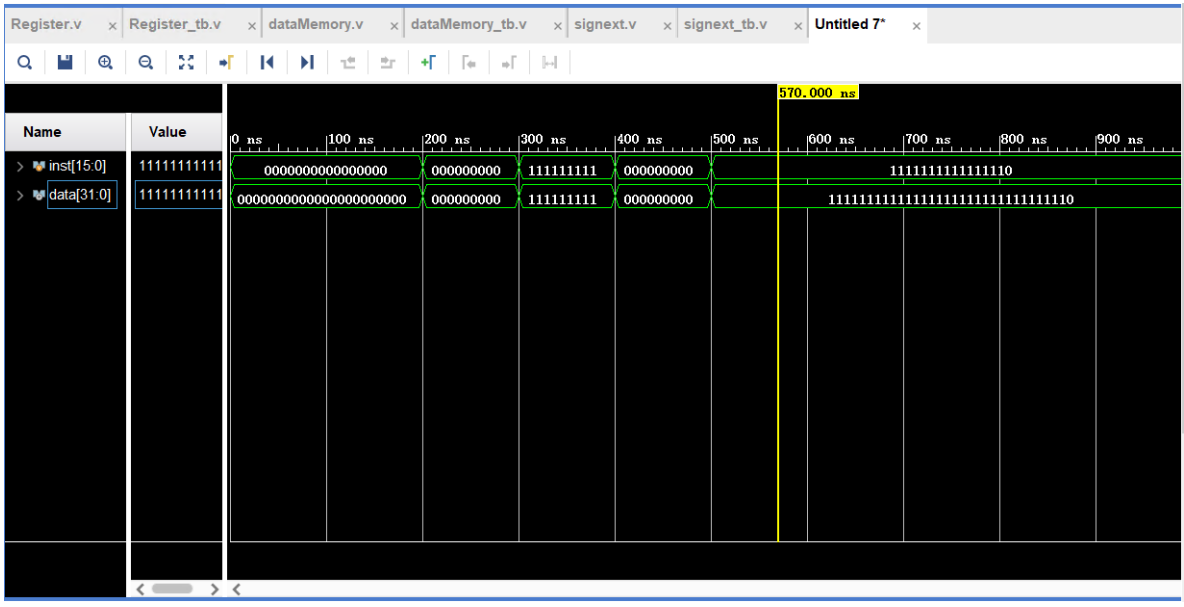
```
assign data=(inst[15])?(inst | 32'hffff0000) : (inst | 32'h00000000);
```

通过分析实验报告书仿真测试结果得出激励文件如下。

signext_tb.v

```
initial begin
        inst=0;
        #100
        inst=16'b0000000000000000;
        #100
        inst=16'b0000000000000001;
        #100
        inst=16'b1111111111111111;
        #100
        inst=16'b0000000000000010;
        #100
        inst=16'b1111111111111110;
    end
```

测试结果



测试结果与实验指导书一致，逻辑正确。

三、总结与心得体会

1.总结

本次遇到了一种新的数据声明，即：

```
reg [n:0] name [n:0];
```

在最开始不太理解这种声明代表的数据类型，在实验进行过程中通过对前半学期实验的同学进行询问了解到这代表这个数据长度n，可以存放n条。理解该数据结构后也帮助我更简单地完成了这几个模块读写部分的设计。

在符号拓展的设计中，一开始没有想到如何用实验指导书上暗示的一行表达式完成拓展的方法，而是使用了if条件分支语句。在查询**Verilog**语法后发现**Verilog**也支持?:表达式，故采用三目运算符可以较简便地实现符号拓展。联想到系统结构课程上学到的条件分支和条件传送两种类型的语句在不同的输入数据情况下会有不同的性能表现，我们以后在程序设计上应该仔细考虑使用什么语句性能最好。

2.心得体会

本次实验相比前三次实验逻辑性更强，需要仔细思考才能得出与实验指导书上一致的结果。同时本次实验也帮助我认识到了组合逻辑和时序逻辑的区别和联系，有利于我在最后两个实验中实现更复杂的处理器模块。