计算机体系结构实验

实验4报告书

作者：李珉超

学号：515030910361

2017.4.2

1 实验介绍

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

1.ISE 13.4 的使用

2.register的实现

3.Data Memory的实现

4.有符号扩展的实现

2 实验过程

2.1 Register

2.1.1 实验原理

Register 是CPU中读取速度最快的存储模块。

由于一共有32个可访问的32位存储单元，所以使用5带宽的数据作为32位寄存器的地址。

Register有两个readReg和一个writeReg，因此对应于两个readData和一个writeData，它们的带宽都是32位的。

此外，Register有一个regWrite信号，高位有效，表明此时writeData上的数据可以在时钟下降沿的时候写到writeReg的寄存器里。 par 而从寄存器里写数据，是随时都可以进行的。

此外，需要clock信号，这是设备的时钟信号；以及reset信号，以将寄存器的值全部清零。

2.1.2 实验模块

```
module register(reset, clock, readReg1, readReg2,
    writeReg, writeData, regWrite,
    readData1, readData2
    );

    input reset;
    input clock;
    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;
    reg [31:0] readData2;
```

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

    always @ (negedge reset)
    begin
        regFile[0] = 0;regFile[1] = 0;regFile[2] = 0;regFile[3] = 0;
        regFile[4] = 0;regFile[5] = 0;regFile[6] = 0;regFile[7] = 0;
        regFile[8] = 0;regFile[9] = 0;regFile[10] = 0;regFile[11] = 0;
        regFile[12] = 0;regFile[13] = 0;regFile[14] = 0;regFile[15] = 0;
        regFile[16] = 0;regFile[17] = 0;regFile[18] = 0;regFile[19] = 0;
        regFile[20] = 0;regFile[21] = 0;regFile[22] = 0;regFile[23] = 0;
        regFile[24] = 0;regFile[25] = 0;regFile[26] = 0;regFile[27] = 0;
        regFile[28] = 0;regFile[29] = 0;regFile[30] = 0;regFile[31] = 0;
    end

    always @ (regWrite or writeReg or negedge clock)
    begin
    if(regWrite == 1)
    begin
        if(writeReg != 0)
        begin
            regFile[writeReg] = writeData;
        end
    end
    end
endmodule
```

2.1.3 仿真

初始化clock，readReg1，readReg2，writeReg，writeData，regWrite为0

然后将regWrite设为1，并做两次读入。

再将regWrite设为0，再做两次读取。

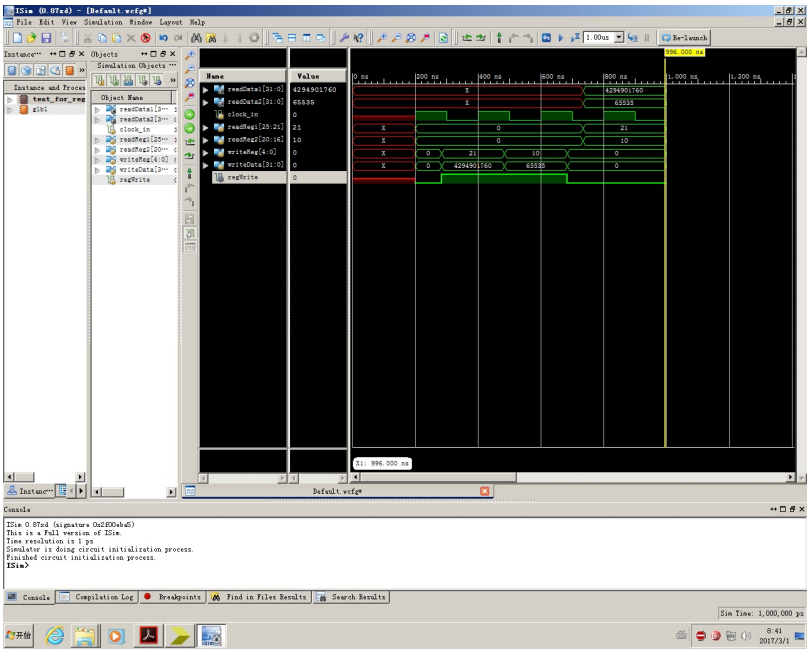
并对实现了的5种opCode进行100ns的测试，观察全部的输出信号是否正确。

```
module test_for_register;
    // Inputs
    reg clock;
    reg [25:21] readReg1;
    reg [20:16] readReg2;
    reg [4:0] writeReg;
    reg [31:0] writeData;
    reg regWrite;

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

    // Instantiate the Unit Under Test (UUT)
    register uut (
    .clock(clock),
    .readReg1(readReg1),
    .readReg2(readReg2),
    .writeReg(writeReg),
```

```
.writeData(writeData),
.regWrite(regWrite),
.readData1(readData1),
.readData2(readData2),
);

always #(100) clock = ~clock; initial begin
    // Initialize Inputs
    #200;
    clock = 0;
    readReg1 = 0;
    readReg2 = 0;
    writeReg = 0;
    writeData = 0;
    regWrite = 0;

    // Add stimulus here
    #85;
    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
```

2.1.4 仿真图



仿真显示，将数据输入到第10和第21寄存器，然后从中读取到了正确的数据。

2.2 DataMemory

2.2.1 实验原理

Data memory 是计算机结构中最主要的外存设备。

在本实验中，data memory的address端口只有一个，是一个32位带宽的数据。

Data memory有一个读数据端口readData和一个写数据端口writeData，它们都是32带宽的。

另外还有两个信号memRead和writeRead，说明是否需要读或者写数据。

当然，还需要clock信号，以实现时钟下降沿读入。

2.2.2 实验模块

```
module data_memory(clock, address,
    writeData, memWrite, memRead, readData
    );

    input clock;
    input [31:0] address;
    input [31:0] writeData;
    input memWrite;
    input memRead;
    output [31:0] readData;

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

    always @ (negedge clock)
    begin
        if(memWrite == 1)
        begin
        memFile[address] = writeData;
```

```
        end
    end

    always @ (address)
    begin
        readData = memFile[address];
    end
endmodule
```

2.2.3 仿真

初始化所有输入信号为0。

先设置memWrite为1，并将数据写入mem[15]

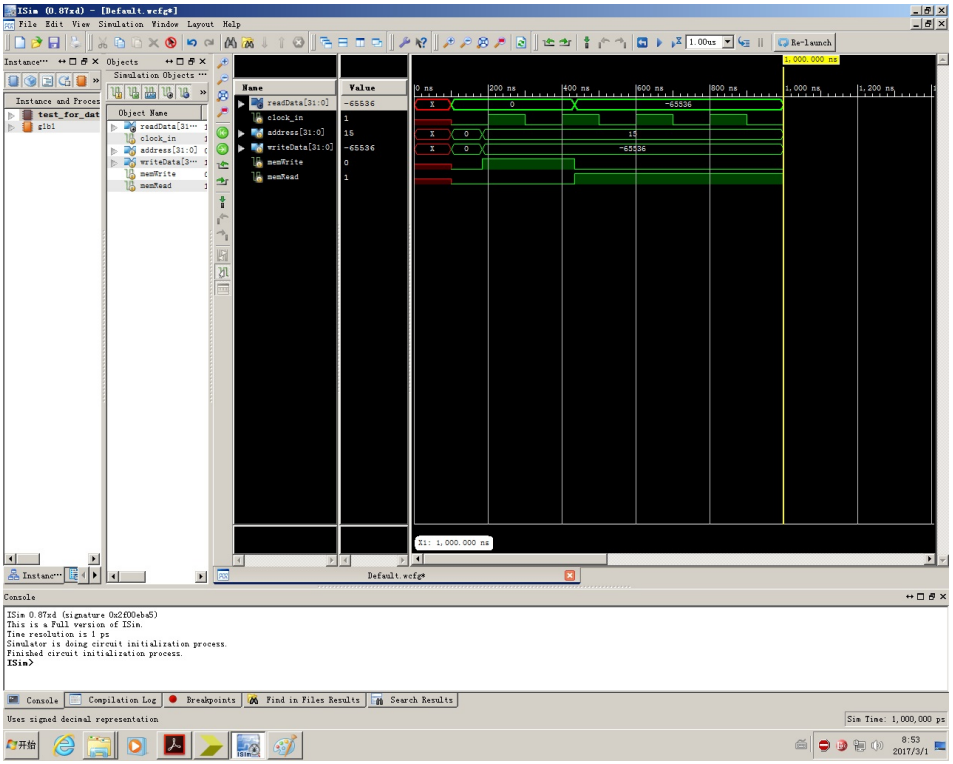
再将memWrite设置为0，并将memRead设置为1

由于address的值不变，故从刚刚读入的地址处读取其中的数据。

```
module test_for_data_memory;
    // Inputs
    reg clock;
    reg [31:0] address;
    reg [31:0] writeData;
    reg memWrite;
    reg memRead;

    // Outputs
    wire [31:0] readData;

    // Instantiate the Unit Under Test (UUT)
    data_memory uut (
    .clock(clock),
    .address(address),
    .writeData(writeData),
    .memWrite(memWrite),
```

```
        .memRead(memRead),
        .readData(readData)
        );

        always #100 clock = ~clock;
        initial begin
            // Initialize Inputs
            #100;
            clock = 0;
            address = 0;
            writeData = 0;
            memWrite = 0;
            memRead = 0;

            // Add stimulus here
            #85;
            memWrite = 1'b1;
            address = 32'b00000000000000000000000000001111;
            writeData = 32'b11111111111111110000000000000000;

            #250;
            memRead = 1'b1;
            memWrite = 1'b0;

        end
endmodule
```

2.2.4 仿真图



仿真表明，从正确的地址读取了正确的数据。

2.3 Signext

2.3.1 实验原理

Sign extend是CPU中专门为I-type服务的硬件模块。

对于I-type的MIPS指令，低16位是immediate number，作为ALU的操作数之一，必须将16位数扩展到32位。

注意，16位数是有符号数，扩展也必须是有符号的扩展。

我的实现方式是，看16位的最高位。如果是0，那么直接输出16位到32位；否则先将16位的数取正，然后再在32位内取负。

2.3.2 实验模块

```
module signext(inst, data);
    input [15:0] inst;
    output [31:0] data;
    assign data = inst[15] ? (32'b11111111111111111111111111111111 - (16'b1111111111111111
- inst + 1) + 1) : inst;
endmodule
```

2.3.3 仿真

先初始化inst为0

然后测试inst为16'b0000000011111111和16'b1000000011111111的情况。

```
module test_for_singext;
    // Inputs
    reg [15:0] inst;

    // Outputs
    wire [31:0] data;

    // Instantiate the Unit Under Test (UUT)
    signext uut (
    .inst(inst),
    .data(data)
    );

    initial begin
        // Initialize Inputs
        inst = 0;

        // Wait 100 ns for global reset to finish
        #100;
```

```
        // Add stimulus here
        #200 inst = 16'b0000000011111111;
        #200 inst = 16'b1000000011111111;
    end
endmodule
```

2.3.4 仿真图

注意到，仿真结果表明，对于正数，负数的扩展均正确。

3 实验心得

了解了寄存器和内存的针脚设计的基本结构。了解了CPU中数据读写和信号之间的关系和具体操作。

由于每一个指令都需要读取寄存器，所以register不需要一个信号来指明register被读取。而写入数据到寄存器中则不是每个指令都需要的，所以需要一个regWrite的信号，否则每个周期都会刷新register的数据。

同理，因为不是每个周期都需要读或写内存的数据，所以读和写都需要信号。

由于不确定WriteReg，WriteData，RegWrite的顺序，我们采用一个事件作为写入的触发事件，这里以时钟下降沿为事件。而读取操作，因为数据总是已经在file中准备好了，所以不需要触发事件。