计算机系统结构实验报告 实验 4

姓名：卞思沅

学号：518021910656

日期：2020/05/27

目录：

1. 实验概述

1.1 实验名称

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

1.2 实验目的

1．理解 CPU 的寄存器、存储器、有符号扩展

1．Register 的实现

2．Data Memory 的实现

3．有符号扩展的实现

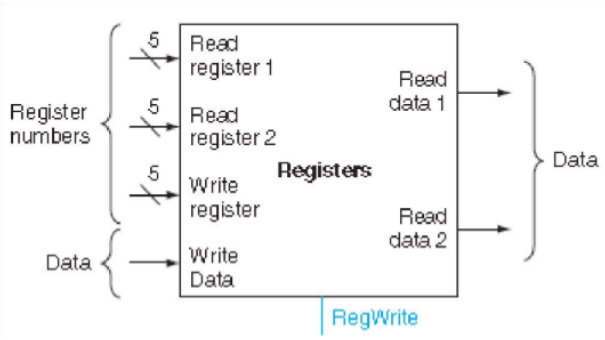
4．使用行为仿真

2. 实验描述

2.1 寄存器

2.1.1 寄存器模块简述

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。



寄存器模块

编写功能：由于不确定 WriteReg,WriteData,RegWrite 信号的先后次序，可采用时钟的下降沿作为写操 作的同步信号，以防止发生错误。

```
31
32      reg [31:0] regFile[31:0];
33
34
35      always @(readReg1 or readReg2 or writeReg)
36          begin
37              // ToDo
38          end
39
40      always @ (negedge Clk)
41          begin
42              // ToDo
43          end
44
```

仿真测试：添加激励信号，进行行为仿真。使用 Clk 作为时钟输入，仿真周期自定，时钟周期 可设为 200ns。

2.1.2 寄存器模块实验代码及结果

Register.v:

```
23  module Registers(
24      input reset,
25      input Clk,
26      input [25:21] readReg1,
27      input [20:16] readReg2,
28      input [4:0] writeReg,
29      input [31:0] writeData,
30      input regWrite,
31      output [31:0] readData1,
32      output [31:0] readData2
33      );
34
35      reg [31:0] regFile[31:0];
36      reg [31:0] ReadData1;
37      reg [31:0] ReadData2;
38      integer i;
39
40      always @(readReg1 or readReg2 or writeReg)
41          begin
42              ReadData1 = regFile[readReg1];
43              ReadData2 = regFile[readReg2];
44          end
45      always @ (negedge Clk)
46          begin
47              if(reset)
48                  begin
49                  for(i=0;i<31;i=i+1)
50                      regFile[i] = 32'h0000_0000;
51                  end
```

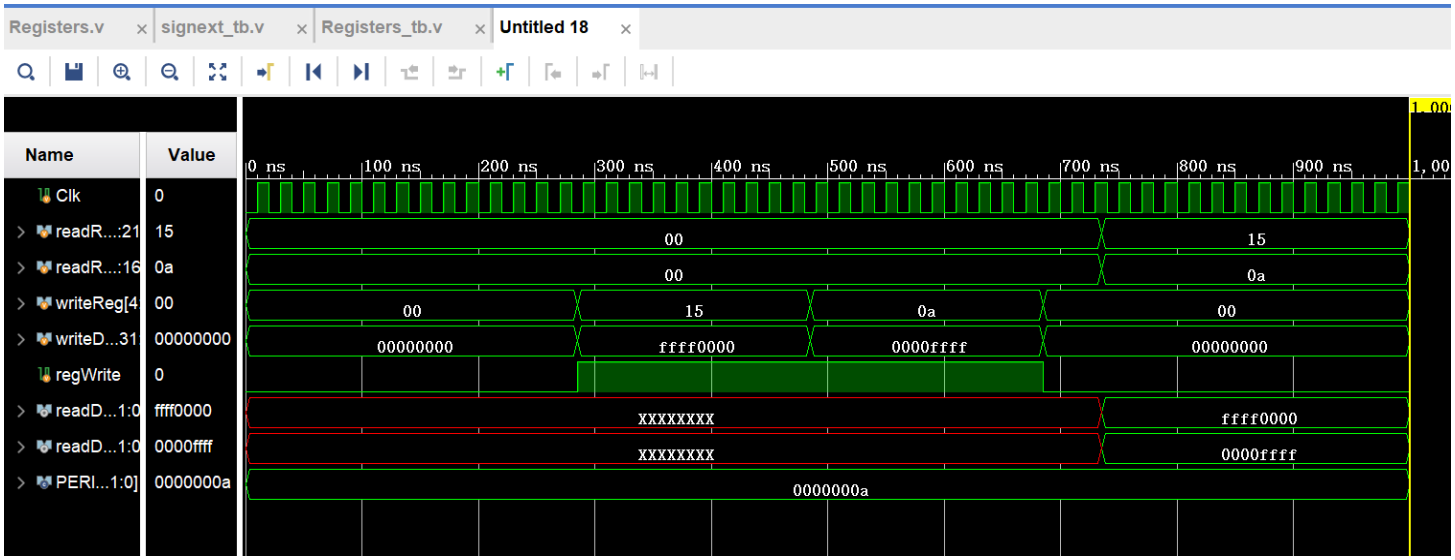
```
52              else if(writeReg)
53                  regFile[writeReg] = writeData;
54          end
55
56      assign readData1 = ReadData1;
57      assign readData2 = ReadData2;
58  endmodule
```

Register_tb.v:

```
23  module Registers_tb(  );
24      reg Clk;
25      reg [25:21] readReg1;
26      reg [20:16] readReg2;
27      reg [4:0] writeReg;
28      reg [31:0] writeData;
29      reg regWrite;
30      wire [31:0] readData1;
31      wire [31:0] readData2;
32
33      Registers u0(
34          .Clk(Clk),
35          .readReg1(readReg1),
36          .readReg2(readReg2),
37          .writeReg(writeReg),
38          .writeData(writeData),
39          .readData1(readData1),
40          .readData2(readData2)
41      );
42
43      parameter PERIOD = 10;
44      always #(PERIOD) Clk = !Clk;
45
46      initial begin
47          //initialize inputs
48          Clk = 0;
49          readReg1 = 5'b00000;
50          readReg2 = 5'b00000;
51          regWrite = 1'b0;
```

```
52          writeReg = 5'b00000;
53          writeData = 32'h0000_0000;
54
55          //current time: 285ns
56          #285;
57          regWrite = 1'b1;
58          writeReg = 5'b10101;
59          writeData = 32'hffff_0000;
60
61          //current time: 485ns
62          #200;
63          writeReg = 5'b01010;
64          writeData = 32'h0000_ffff;
65
66          #200;
67          regWrite = 1'b0;
68          writeReg = 5'b00000;
69          writeData = 32'h0000_0000;
70
71         // Current time:735ns
72         #50;
73         readReg1 = 5'b10101;
74         readReg2 = 5'b01010;
75
76      end
77
78  endmodule
```

仿真结果:

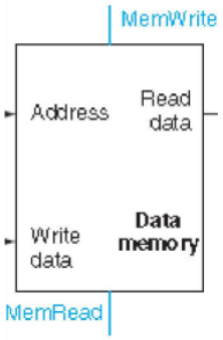


2.2 内存单元模块

2.2.1 内存单元模块简述

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 BlockMemory 来生成。



实验步骤：

1. 新建模块源文件

2. 编写功能，可编写如下格式两个 Verilog 代码的 always 语句块：

```
13
14      reg [31:0] memFile[0:63];
15
16      always@ ( /* conditions */ )
17        begin
18          // ToDo
19        end
20
21      always@ (/* which edg */)
22        begin
23          // ToDo
24        end
```

3. 功能仿真。添加激励信号，设定不同的输入。需要覆盖所有的情况，以保证逻辑的正确。

2.2.2 内存单元模块实验代码及结果

dataMemory.v:

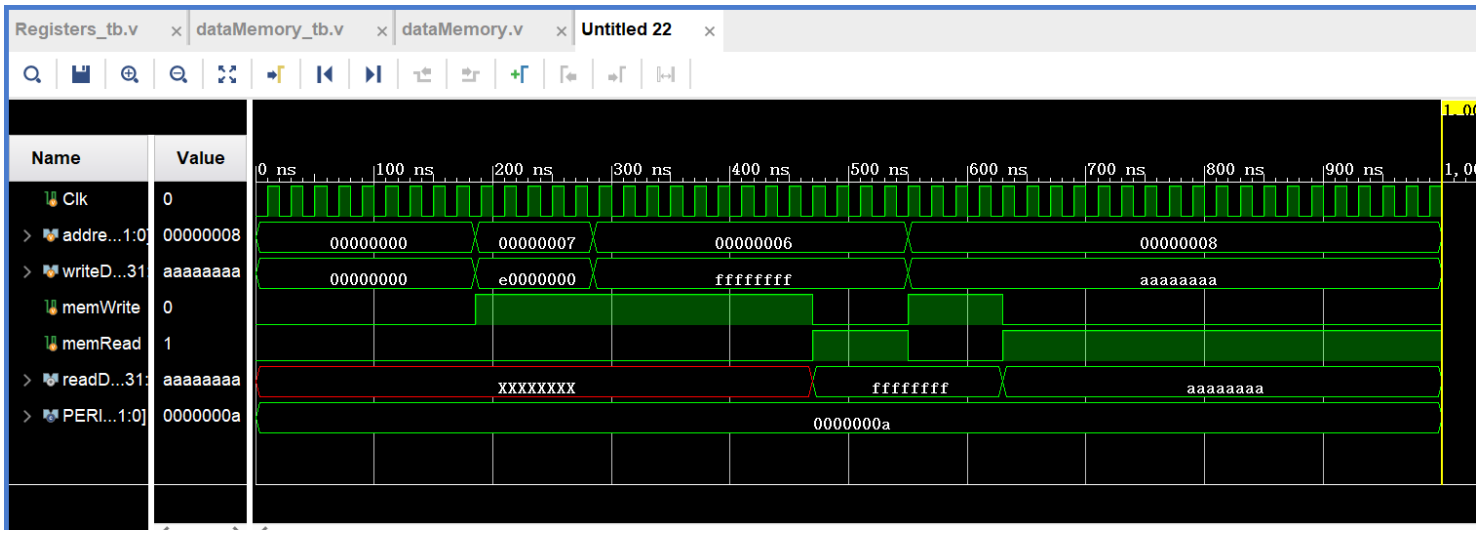
```
23  module dataMemory(
24      input Clk,
25      input [31:0] address,
26      input [31:0] writeData,
27      input memWrite,
28      input memRead,
29      output [31:0] readData
30      );
31
32      reg [31:0] memFile[31:0];
33      reg [31:0] ReadData;
34
35      always @(address or memRead or memWrite)
36          begin
37          if(memRead)
38              ReadData = memFile[address];
39          end
40      always @ (negedge Clk)
41          begin
42              if(memWrite)
43                  memFile[address] = writeData;
44          end
45
46      assign readData = ReadData;
47  endmodule
48
```

dataMemory_tb.v:

```
23  module dataMemory_tb( );
24      reg Clk;
25      reg [31:0] address;
26      reg [31:0] writeData;
27      reg memWrite;
28      reg memRead;
29      wire [31:0] readData;
30
31      dataMemory u0(
32          .Clk(Clk),
33          .address(address),
34          .writeData(writeData),
35          .memWrite(memWrite),
36          .memRead(memRead),
37          .readData(readData)
38      );
39
40      parameter PERIOD = 10;
41      always #(PERIOD) Clk = !Clk;
42
43      initial begin
44          //initialize inputs
45          Clk = 0;
46          address = 0;
47          memWrite = 1'b0;
48          memRead = 1'b0;
49          writeData = 32'h0000_0000;
50          address = 32'h0000_0000;
51
```

```
52          #185;
53          memWrite = 1'b1;
54          address = 32'h0000_0007;
55          writeData = 32'he000_0000;
56
57
58          #100;
59          memWrite = 1'b1;
60          writeData = 32'hffff_ffff;
61          address = 32'h0000_0006;
62
63          #185
64          memRead = 1'b1;
65          memWrite = 1'b0;
66          // ------------
67
68          #80;
69          memRead = 1'b0;
70          memWrite = 1'b1;
71          address = 8;
72          writeData = 32'haaaaaaaa;
73
74          #80;
75          memWrite = 1'b0;
76          memRead = 1'b1;
77          // ----------------
78
79      end
80  endmodule
```

仿真结果:



2.3 带符号扩展模块

2.3.1 带符号扩展模块简述

模块描述：将 16 位有符号数扩展为 32 位有符号数。

实验步骤：

1. 新建模块源文件

2. 实现功能，可能的实现方法之一如下：

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

      assign data=  //How to;

endmodule
```

3. 行为仿真。1. 添加激励信号，进行行为仿真 2. 观察波形是否满足设计逻辑。

2.3.2 带符号扩展模块实验代码及结果

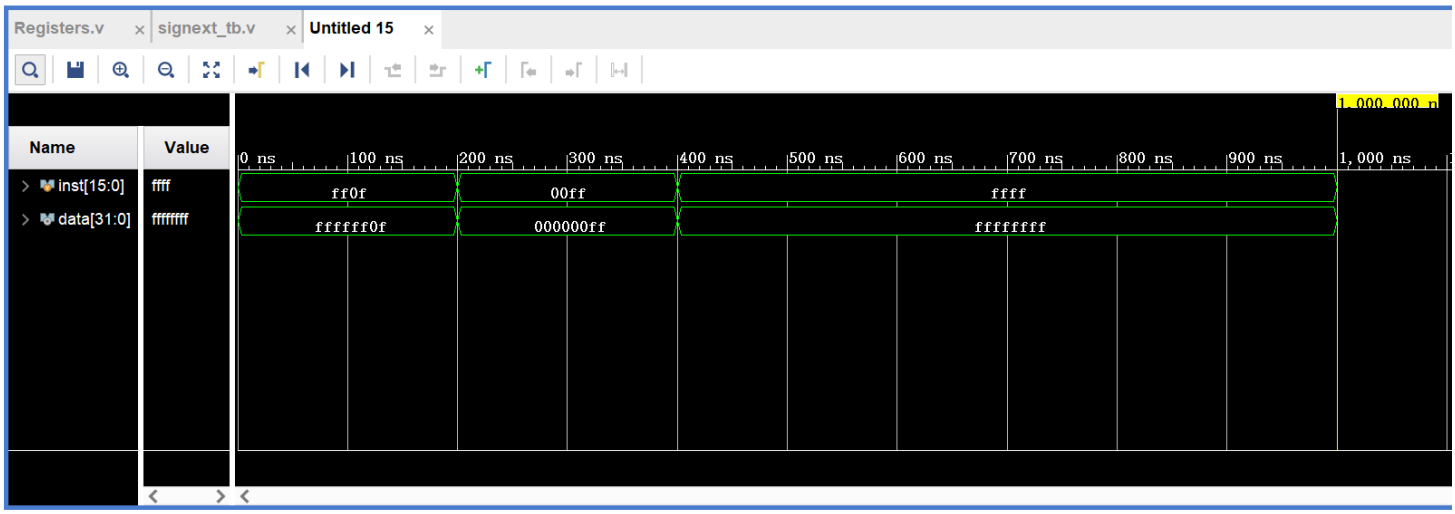
Signext.v:

```
22
23  module signext(
24      input [15:0] inst,
25      output [31:0] data
26      );
27
28      assign data=(inst[15])? (inst|32'hffff0000):(inst|32'h00000000);
29  endmodule
30
```

Signext_tb.v:

```
23  module signext_tb();
24      reg [15:0] inst;
25      wire [31:0] data;
26
27      signext u0(
28          .inst(inst),
29          .data(data)
30      );
31
32      initial begin
33          inst = 16'hff0f;
34
35          #200
36          inst = 16'h00ff;
37
38          #200
39          inst = 16'hffff;
40
41      end
42  endmodule
43
```

仿真结果:



3. 实验心得与总结

本次实验需要自己编写的部分多了很多。就难度而言我认为和上一个模块差不多。在 reg 与 wire 的选择上，我依旧有一些小问题，但是最终都得到了解决。

在代码中，我设置 register 和 memory 的写操作都在时钟下降沿开始。这样便可以同步各操作，以防在读取时出现由于不同步造成的错误结果，或是几个写操作间出现相互冲突。

起初我不太能理解 CLK 的作用。经过查阅资料，理解了我们需要利用时钟的下降沿作为写操作的同步信号，否则在读取时会接收到错误的信号。 此外，在有符号扩展时，我起初并未注意到“有符号”的前提，直接通过“zero padding”，通过补 0 将 16 位整数扩展到了 32 位。在与同学讨论时才注意到这个问题，进行了修改，最后顺利通过仿真测试。

补充：由于我仿真时对波形的观察不仔细，register 和 memory 模块原代码有着部分错误而没有发现（module 中 always@后面接的条件不对，导致有时控制信号变化，module 却无法作出反应）。在 lab5 编写单周期 CPU 代码时才发现问题。这提醒我在行为仿真时需要更加仔细，不能过分相信自己的代码，否则会产生意想不到的问题。发现问题后，我又重新检查了其它模块的仿真代码，并且又发现了少数小问题。

4、参考资料

2020 计算机系统结构实验指导书-LAB04_M