计算机系统结构实验 Lab4

David Wang

2020 年 5 月

1 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
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

2 寄存器的实现

2.1 模块描述

寄存器是指令操作的主要对象，32 位 MIPS 处理器共有 32 个 32 位寄存器。

2.2 模块实现

寄存器的读取操作是组合逻辑，只要遇到 readReg1,readReg2 中的一个，即可读取。但是对于写寄存器来说，如果在 writeReg 信号达到高电平之前 writeReg 没有选中正确的寄存器，或者 writeData 不是正确的数据，则会发生错误。所以写操作应该采用时序逻辑，可以约定采用时钟下降沿作为写操作的同步信号。

```
1    module Registers(
2    input clock,
3    input reset,
4    input [25:21] readReg1,
5    input [20:16] readReg2,
6    input [4:0] writeReg,
7    input [31:0] writeData,
8    input regWrite,
9    output reg [31:0] readData1,
10   output reg [31:0] readData2
11   );
12
13   reg [31:0] regFile[31:0];
14   integer i;
15   always @ (readReg1 or readReg2 )
16   begin
17       regFile[0]=0;
18       readData1=regFile[readReg1];
19       readData2=regFile[readReg2];
```

```
    end
    always @ (negedge clock)
    begin
        if(reset)
        begin
            for(i=0;i<32;i=i+1) regFile[i]=0;
        end
        else
        begin
        if(regWrite==1 && writeReg) regFile[writeReg]=writeData;
        if(regWrite && readReg1==writeReg) readData1=writeData;
        if(regWrite && readReg2==writeReg) readData2=writeData;
        end
    end
    always @ (reset)
endmodule
```

2.3 仿真程序

```
    module Registers_tb();
    reg clock;
    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;
    reg reset;
    Registers re(
        .clock(clock),
        .readReg1(readReg1),
        .readReg2(readReg2),
        .writeReg(writeReg),
        .writeData(writeData),
        .regWrite(regWrite),
        .readData1(readData1),
        .readData2(readData2),
        .reset(reset)
    );

    always #100 clock=~clock;
    initial begin
        clock=0;
        readReg1=0;
        readReg2=0;
        writeReg=0;
        writeData=0;
        regWrite=0;
        reset=1;
        #100;
        reset=0;
        #300;
        regWrite=1'b1;
        writeReg=5'b10101;
        writeData=32'hffff0000;
```

```
38          #200;
39          writeReg=5'b01010;
40          writeData=32'h0000ffff;
41
42          #200;
43          regWrite=1'b0;
44          writeReg=5'b00000;
45          writeData=32'h00000004;
46
47          #200;
48          readReg1=5'b10101;
49          readReg2=5'b01010;
50      end
51  endmodule
```

2.4 仿真波形

从下面的图中可以看出，在给出 readReg1 和 readReg2 时，寄存器可以给出正确的寄存器值，在给出 writeReg 和 regWrite 信号时，寄存器可以被正确写入。如果写入的目标寄存器是 $0，则寄存器无法被写入。

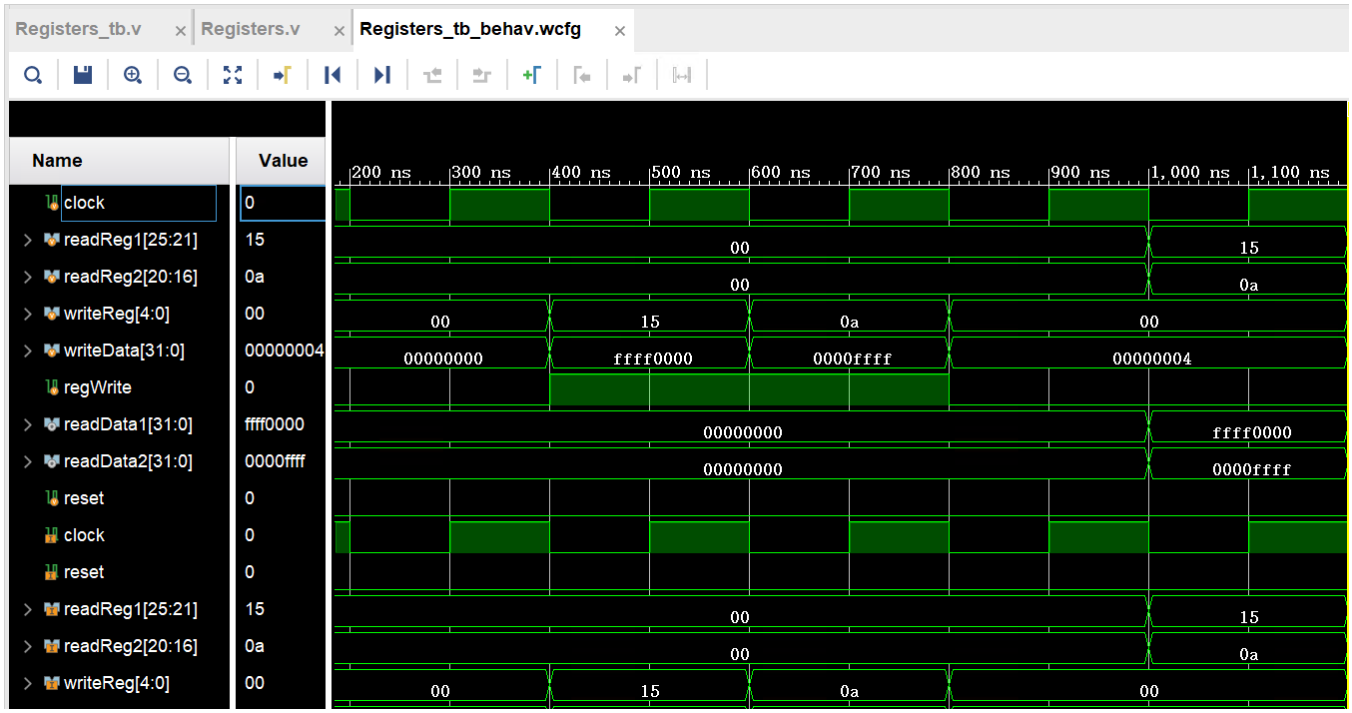

图 1:


Registers_tb.v
Registers.v
Registers_tb_behav.wcfg
Name
Value
readData1[31:0]
ffff0000
readData2[31:0]
0000ffff
reset
0
clock
0
reset
0
readReg1[25:21]
15
readReg2[20:16]
0a
writeReg[4:0]
00
writeData[31:0]
00000004
regWrite
0
readData1[31:0]
ffff0000
readData2[31:0]
0000ffff
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1,000 ns
1,100 ns
00000000
ffff0000
0000ffff
00
15
0a
00000004

图 2:

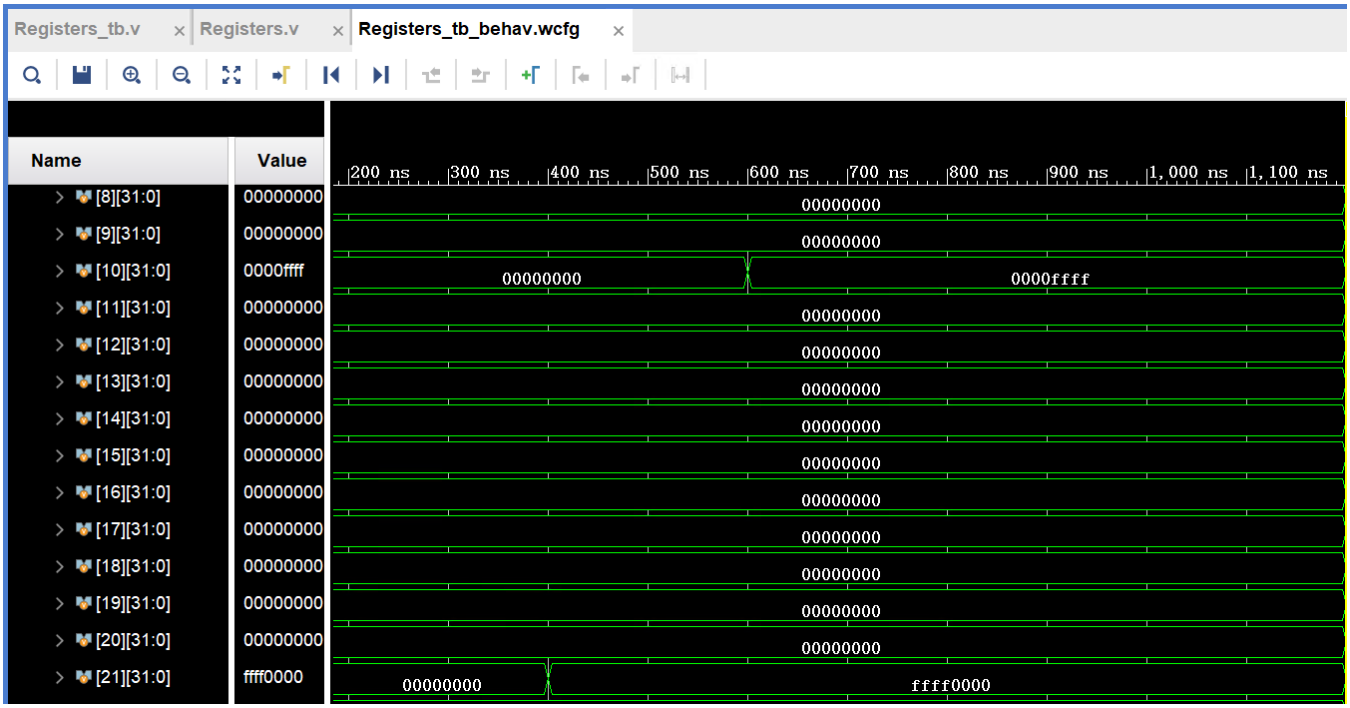
Registers_tb.v
Registers.v
Registers_tb_behav.wcfg
Name
Value
[8][31:0]
[9][31:0]
[10][31:0]
[11][31:0]
[12][31:0]
[13][31:0]
[14][31:0]
[15][31:0]
[16][31:0]
[17][31:0]
[18][31:0]
[19][31:0]
[20][31:0]
[21][31:0]
00000000
0000ffff
ffff0000
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1,000 ns
1,100 ns

图 3:

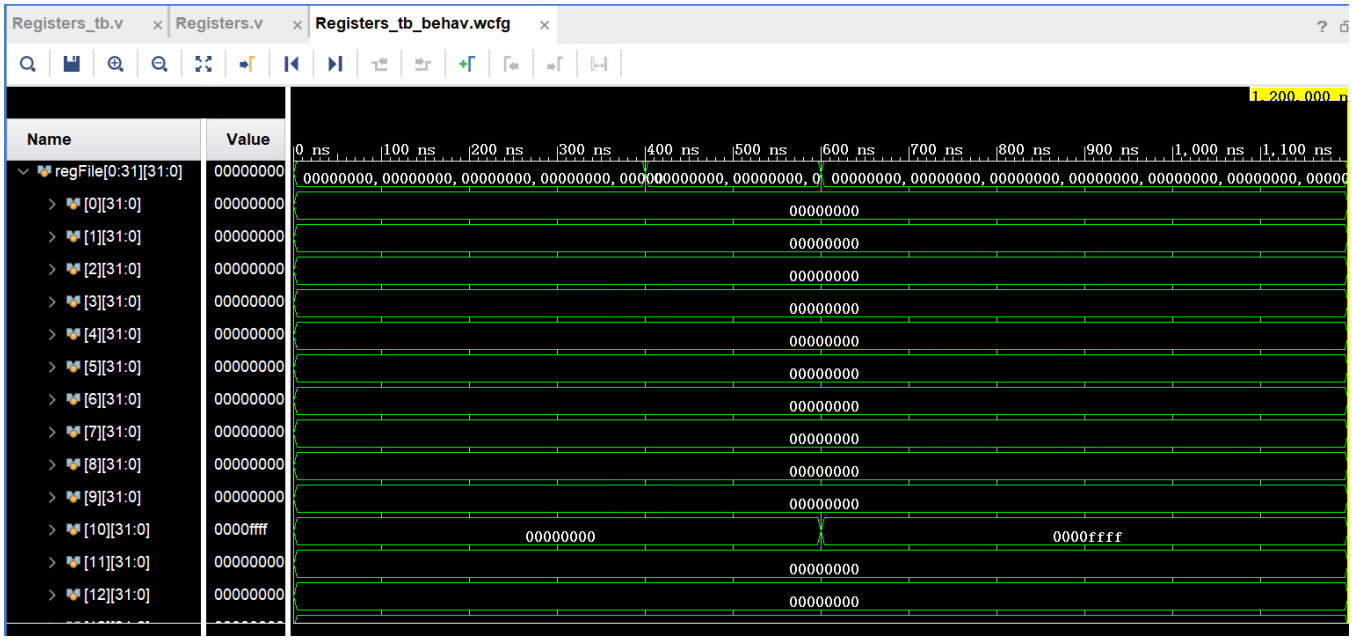

图 4:

3 数据存储器

3.1 模块实现

数据存储器模块与寄存器模块类似，因为写数据，也需要考虑同步，也需要时钟信号。这里依然统一约定在时钟的下降沿写数据。

```
    module dataMemory(
    input clock,
    input [31:0] address,
    input [31:0] writeData,
    input memWrite,
    input memRead,
    output reg [31:0] readData
    );
    reg [31:0] memFile [31:0];
    always @ (memRead or address)
    begin
        readData=memFile[address];
    end
    always @ (negedge clock)
    begin
        if(memWrite==1) memFile[address]=writeData;
    end
endmodule
```

3.2 仿真程序

```
    module dataMemory_tb();
    reg clock;
    reg [31:0] address;
    reg [31:0] writeData;
    reg memWrite;
    reg memRead;
```

```
    wire [31:0] readData;

    dataMemory dm(
        .clock(clock),
        .address(address),
        .writeData(writeData),
        .memWrite(memWrite),
        .memRead(memRead),
        .readData(readData)
    );
    always #100 clock=~clock;
    initial begin
        clock=0;
        address=0;
        writeData=0;
        memWrite=0;
        memRead=0;
        #185;
        memWrite=1'b1;
        address=32'h00000007;
        writeData=32'he0000000;
        #100;
        memWrite=1'b1;
        writeData=32'hffffffff;
        address=32'h00000006;

        #185;
        memRead=1'b1;
        memWrite=1'b0;

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

        #80;
        memWrite=0;
        memRead=1;

    end
endmodule
```

3.3 仿真波形

如下面的两个图所示，波形变化符合预期。

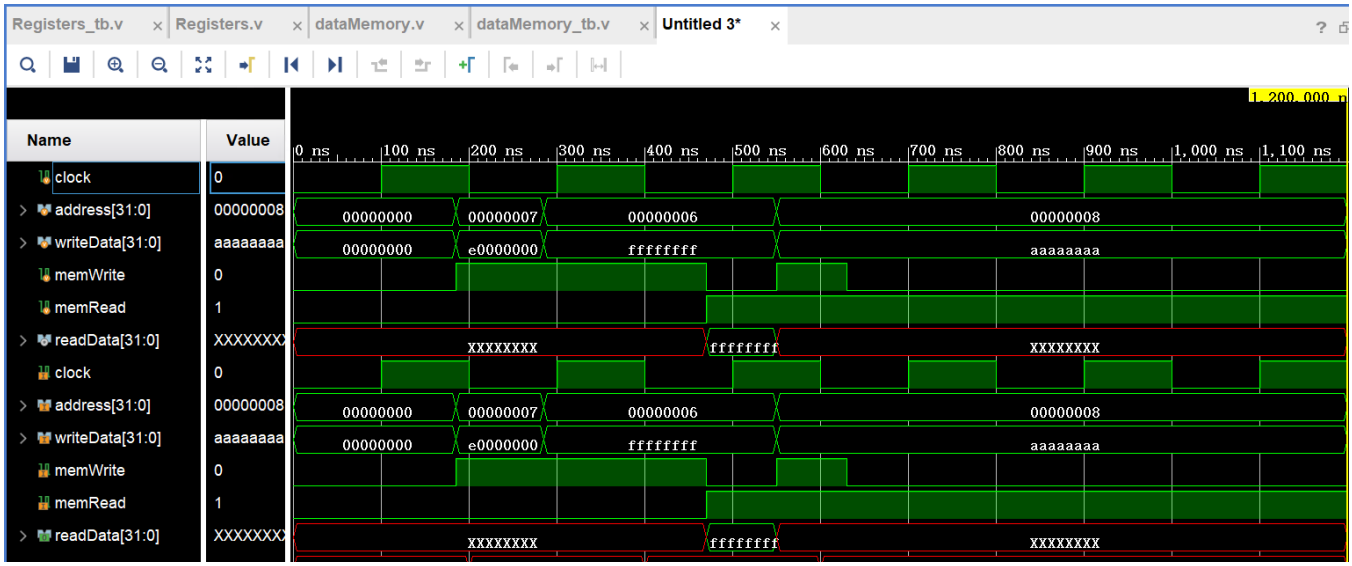
Registers_tb.v
Registers.v
dataMemory.v
dataMemory_tb.v
Untitled 3*
Name
Value
clock
address[31:0]
writeData[31:0]
memWrite
memRead
readData[31:0]
clock
address[31:0]
writeData[31:0]
memWrite
memRead
readData[31:0]
0
00000008
aaaaaaaa
0
1
XXXXXXXX
0
00000008
aaaaaaaa
0
1
XXXXXXXX
1,200,000
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1,000 ns
1,100 ns
00000000
00000007
00000006
00000008
00000000
e0000000
ffffffff
aaaaaaaa
XXXXXXXX
ffffffff
XXXXXXXX
00000000
00000007
00000006
00000008
00000000
e0000000
ffffffff
aaaaaaaa
XXXXXXXX
ffffffff
XXXXXXXX

图 5:

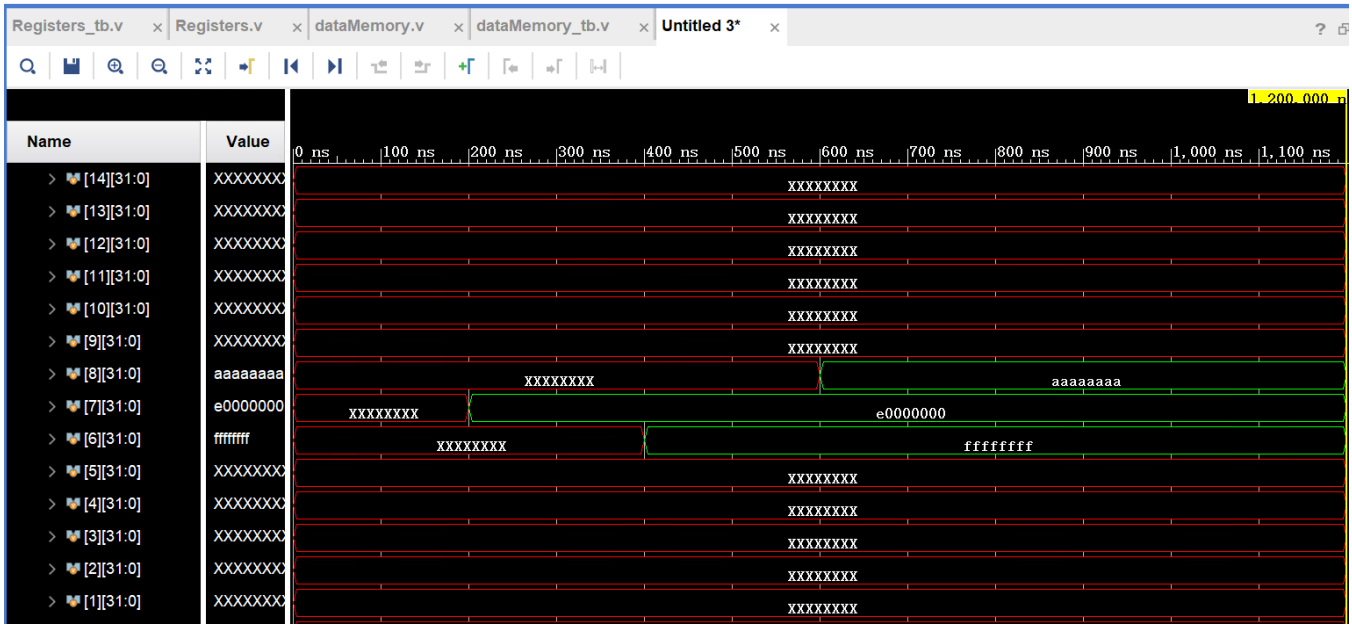
Registers_tb.v
Registers.v
dataMemory.v
dataMemory_tb.v
Untitled 3*
Name
Value
[14][31:0]
[13][31:0]
[12][31:0]
[11][31:0]
[10][31:0]
[9][31:0]
[8][31:0]
[7][31:0]
[6][31:0]
[5][31:0]
[4][31:0]
[3][31:0]
[2][31:0]
[1][31:0]
XXXXXXXX
aaaaaaaa
e0000000
ffffffff
1,200,000
0 ns
100 ns
200 ns
300 ns
400 ns
500 ns
600 ns
700 ns
800 ns
900 ns
1,000 ns
1,100 ns
XXXXXXXX
aaaaaaaa
e0000000
ffffffff

图 6:

4 带符号扩展

4.1 模块描述

带符号扩展单元将 16 位有符号数扩展为 32 位有符号数。如果最高位为 0，则结果的高 16 位均为 0。如果最低位为 0，结果的高 16 位均为 1。

4.2 模块实现

```
    module signext(
    input [15:0] inst,
    output  [31:0] data
    );
    assign data= inst[15]?{16'hffff,inst}:{16'h0000,inst};
endmodule
```

4.3 仿真程序

```
    module signext_tb();
    reg [15:0] inst;
    wire [31:0] data;
    signext ext(
        .inst(inst),
        .data(data)
    );
    initial begin
        inst=16'h0000;
        #200
        inst=16'b0000000000000001;
        #200
        inst=16'b1111111111111111;
        #200
        inst=16'b0000000000000010;
        #200
        inst=16'b1111111111111110;
    end
endmodule
```

5 仿真波形

如下图所示，波形符合预期。

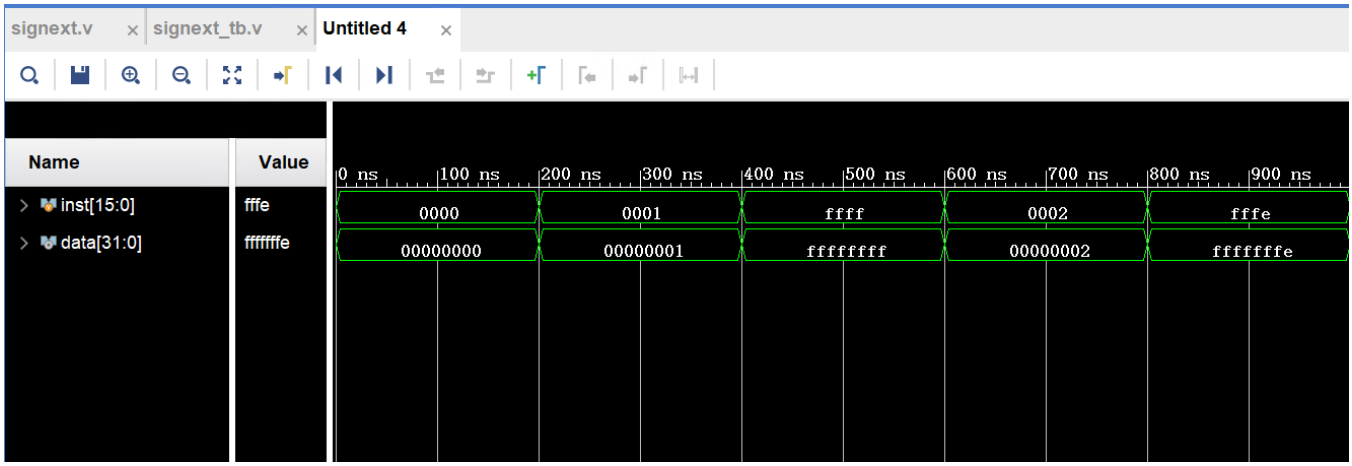

图 7:

6 总结与反思

6.1 实验重点与难点

寄存器和数据存储器的读和写需要不同的判断条件，其中写操作都需要注意同步的问题。

6.2 实验总结与感想

本次实验完成的是比较简单的三个模块。它们是 MIPS 处理器的重要组成部分。它们能否正确运行关系到后续实验的成败。所以完成这次实验必须非常仔细，考虑到的情况必须全面。经过前四个实验，我已经初步掌握了使用 Verilog 进行数字逻辑设计的基本知识。我期待使用这四个实验设计出来的模块组装成一个功能完备的处理器。