计算机结构实验 lab04

张露伊

2022 年 3 月 26 日

摘要

本实验实现了简单的类 MIPS 处理器的几个部件：寄存器（registers）、数据存储器（data memory）和带符号拓展模块（sign extend）。并分别进行了仿真测试。它们为之后的简单类 MIPS 处理器的实现打下了基础。

目录

1 实验介绍

1.1 实验名称

简单的类 MIPS 单周期处理器存储部件的设计与实现（二）

1.2 实验目的

1. 理解寄存器、数据存储器、有符号拓展单元的 IO 定义
2. Registers 的设计实现
3. Data Memory 的设计实现
4. 有符号扩展部件的实现
5. 对功能模块进行仿真

2 原理分析

2.1 寄存器组模块的分析

寄存器（Register）是指令操作的主要对象，MIPS 中一共有 32 个 32 位的寄存器，用作数据缓存。寄存器读写数据的速度很快，以此来加速计算机对于数据的访问。

本实验中的寄存器模块包括 32 个寄存器，因此需要一个 5 位的选择信号来对寄存器进行寻址。由于寄存器的位数为 32 位，故而寄存器的输入输出应该均为 32 位。

寄存器模块的输入与输出关系如表 1、表 2 所示。

表 1: 寄存器模块输入信号

输入信号	含义
readReg1[24:21]	读取寄存器 1 的编号
readReg2[20:16]	读取寄存器 2 的编号
writeReg[4:0]	写入寄存器的编号
regWrite	写使能信号，高电平有效
writeData[31:0]	写入的数据
clk	时钟信号

表 2: 寄存器模块输出信号

输出信号	含义
readData1[31:0]	读取寄存器 1 的内容
readReg2[31:0]	读取寄存器 2 的内容

其中由于不确定 writeReg、writeReg、regWrite 信号的先后次序，采用时钟的下降沿作为写操作的同步信号，防止发生错误。对于读取操作，寄存器模块会根据寄存器选择信号 readReg1、readReg2 和寄

存器内容立即响应并输出结果。对于写入操作，寄存器模块会在时钟下降沿且 regWrite 信号为高电平时，将 writeData 值写入 writeReg 指定的寄存器。

2.2 数据存储器模块的分析

数据存储器模块（Data memory）是用来存储运行完成的数据，或者初始化的数据。内存模块的编写与 registers 相似，由于写数据也要考虑信号同步，因此也需要时钟。数据存储模块通过 MemWrite 和 MemRead 来判断是读出数据还是写入数据，他们都是高电平有效，MemWrite 只在时钟下沿且高电平时才会有效。

该模块的输入输出信号如下：

表 3: DataMemory 输入输出信号

输入信号	含义
address[31:0]	读/写的内存地址
writeData[31:0]	写入的数据
memWrite	写使能信号
memRead	读使能信号
Clk	时钟信号
输出信号	含义
readData[31:0]	内存读取结果

2.3 带符号扩展模块的分析

符号扩展模块 (Sign extend) 可以根据主控制器单元模块 (Ctr) 的信号，以带符号扩展或无符号扩展对来自指令的 16 位数进行扩展，扩展结果为一个 32 位的数。

该模块的输入输出信号如下：

表 4: Sign extend 输入输出信号

输入信号	含义
inst[15:0]	指令中的立即数
输出信号	含义
data[31:0]	扩展后的数

3 功能实现

3.1 寄存器组模块的实现

寄存器会持续进行读操作，而由 regWrite 控制写操作。为实现信号同步，保证信号的完整性，写操作仅在时钟下降沿进行。核心代码如下：

```
1 reg [31:0] RegFile[31:0];
```

```
2
3  assign readData1 = RegFile[readReg1];
4  assign readData2 = RegFile[readReg2];
5
6  always @ (negedge clk)
7      begin
8          if(regWrite)
9              RegFile[writeReg] = writeData;
10     end
11 initial begin
12     RegFile[0] = 0;
13 end
```

3.2 数据存储器模块的实现

当 memRead 或 address 信号发生变化或时钟处于下降沿时，内存模块会根据 address 指定的内存地址内容更新数据读取数据，并将其作为结果输出。写操作由 memWrite 信号控制。与寄存器模块相似，为实现信号同步，保证信号的完整性，写操作仅在时钟下降沿进行。核心代码如下：

```
1  reg [31:0] memFile [0:63];
2  reg [31:0] ReadData;
3  always @(memRead or address)
4      begin
5          if(memRead)
6              begin
7                  if(address<63)
8                      ReadData = memFile[address];
9                  else
10                     ReadData = 0;
11             end
12     end
13
14 always @(negedge clk)
15     begin
16         if(memWrite)
17             if(address<63)
18                 memFile[address] = writeData;
19         if(memRead)
20             if(address<63)
21                 ReadData =  writeData;
22     end
```

3.3 带符号扩展模块的实现

带符号扩展模块（sign extend）可以在高 16 位填入立即数第 15 位实现，实现代码如下：

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

endmodule
```

4 结果验证

4.1 寄存器组模块的验证

编写激励文件进行仿真，完整激励文件见 Registers_tb.v。仿真过程中，输入的测试信号如下：

```
Clk = 0;
ReadReg1 = 0;
ReadReg2 = 0;
WriteReg = 0;
WriteData = 0;
RegWrite = 0;

#285;    //285
RegWrite = 1;
WriteReg = 5'b10101;
WriteData = 32'b11111111111111110000000000000000;

#200;    //485
WriteReg = 5'b01010;
WriteData = 32'b00000000000000001111111111111111;

#200;    //685
RegWrite = 0;
WriteReg = 5'b00000;
WriteData = 32'b00000000000000000000000000000000;

#50;     //735
ReadReg1 = 5'b10101;
```

```
24 ReadReg2 = 5'b01010;
```

仿真结果截图如下：

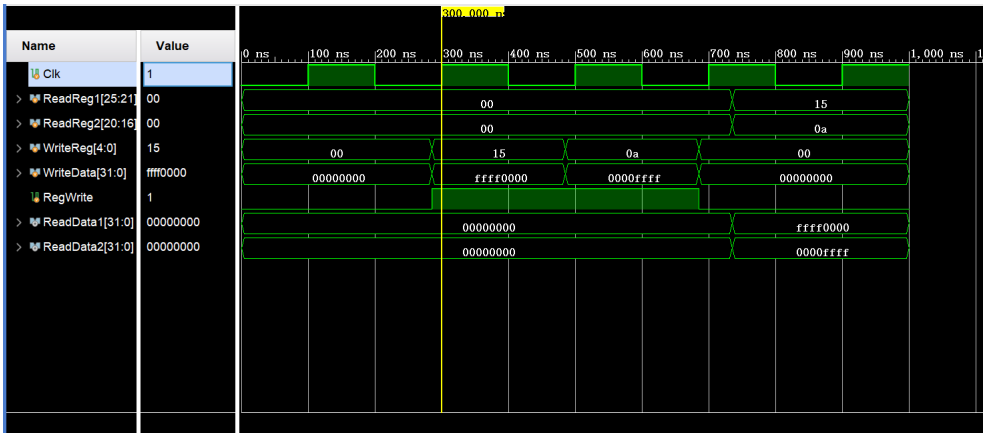

图 1: Registers 仿真结果

从仿真结果看出，寄存器功能模块可以正确地保存以及读取数据。功能实现正确。

4.2 数据存储器模块的验证

编写激励文件进行仿真，完整激励文件见 dataMemory_tb.v。仿真过程中，输入的测试信号如下：

```
1  initial begin
2      Clk = 0;
3      Address = 0;
4      WriteData = 0;
5      MemWrite = 0;
6      MemRead = 0;
7
8      #185;
9      //185
10     MemWrite = 1;
11     Address = 32'b00000000000000000000000000000111;
12     WriteData = 32'b11100000000000000000000000000000;
13     #100;
14
15     //285
16     MemWrite = 1;
17     WriteData = 32'hffffffff;
18     Address = 32'b00000000000000000000000000000110;
19     #185;
20
```

```
    //470
    MemRead = 1;
    MemWrite = 0;
    #80;

    //550
    MemWrite = 1;
    Address = 8;
    WriteData = 32'haaaaaaaa;
    #80;

    //630
    MemWrite = 0;
    MemRead = 1;
end
```

仿真结果截图如下：

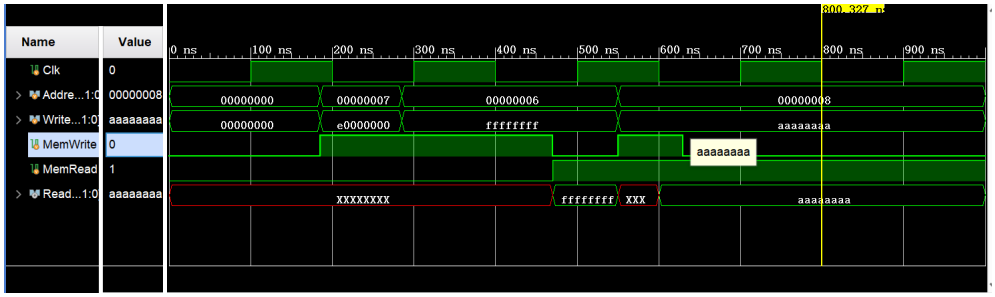

图 2: dataMemory 仿真结果

当地址为初始化时，读取的数据即为 x，符合实验预期。说明模块的实现是正确的。

4.3 带符号扩展模块的验证

编写激励文件进行仿真，完整激励文件见 signext_tb.v。仿真过程中，输入的测试信号如下：

```
initial begin
    inst = 16'h0000;
    #200;
    inst = 16'h0001;
    #200
    inst = 16'hffff;
    #200
    inst = 16'h0002;
    #200;
    inst = 16'hfffe;
```

```
11 end
```

仿真结果截图如下：

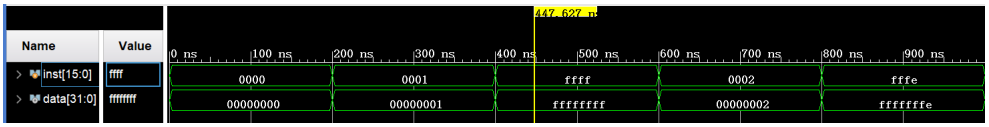

图 3: signext 仿真结果

仿真结果符合实验预期，说明模块的实现是正确的。

5 心得体会

本次实验的三个模块的实现都比较简单，只要充分理解了输入输出信号以及信号的含义，就可以根据模块的工作原理来对模块进行实现。

在实现寄存器时，发现自己不明白 RegFile 的意思，但是通过实验指导书给出的代码部分样例可以推测出 RegFile 表示的是一个寄存器类型的变量，前面的 [31:0] 表示寄存器的位数为 32 位，而后面的 [31:0] 表示的是这样的寄存器有 32 个。而这也符合寄存器组模块的组成。之后通过查阅相关资料发现的确如此。因此在遇到不会的语法或者是变量时，可以通过对它的使用的位置及作用进行合理的推测。而后面的 memFile 同样如此。利用寄存器来模拟内存，省略了逻辑地址到物理的地址的转换过程，address 直接表示了物理地址。