Lab04　简单的类 MIPS 单周期处理器功能部件的设计与实现（二）

w1049

目录

1　实验目的

1. 理解寄存器、数据存储器、有符号扩展单元的 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

寄存器组由 32 个寄存器组成，每个寄存器的位宽为 32 位。寄存器组有两个读端口，可以异步读出两个寄存器的数据；一个写端口，在写使能信号为 1 的时钟下降沿将数据写入寄存器。选择在下降沿写入可以为后续解决数据冒险提供方便。

2.2　数据存储器

数据储存器即内存，位宽也是 32 位。数据存储器的读和写都需要使能信号，读仍然是异步的，写在时钟下降沿进行。这里的写入在上升沿或是下降沿都可以，因为一方面，真正的内存可能有自己的时钟，另一方面，真正的内存速度比较慢，不可能在一个时钟周期内完

成写入，处理器直接接触的是缓存。实际上，这里的数据存储器与寄存器组基本无异，区别仅在于读取也需要使能信号，以防访问非法地址；而寄存器组的地址只有 5 位，不会出现非法地址。

2.3 有符号扩展单元

有符号扩展单元接受一个 16 位立即数，将其符号位扩展为 32 位，即将高 16 位全部填充为符号位。后续可能需要无符号扩展，可以给有符号扩展单元增加一个控制信号。

3 功能实现

3.1 寄存器组

由于我使用异步读，所以不论地址是否有效，都会读出相应的数据；至于数据是否要使用，应该由后续使用这个数据的模块判断，这样可以简化实现。写入只会在下降沿进行，当时钟到达下一个上升沿时，数据已经稳定了，也就是后半个周期可以读出正确数据。

读取是组合逻辑，只需要简单的 `assign`，而写入是时序逻辑，还需要判断写使能信号。读写的代码如下：

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

数据的初始化可以暂时写在 `initial` 块中，后续可以考虑使用 `reset` 来控制:

```
integer i;
initial begin
    for (i = 0; i < 32; i = i + 1)
        regFile[i] <= 0;
end
```

3.2 数据存储器

数据存储器与寄存器组几乎一致，只是需要考虑读使能：

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

3.3 有符号扩展单元

实现符号扩展时，可以了解 Verilog 的语法：

```
assign data = {{16{inst[15]}}, inst};
```

4 仿真结果

三个仿真波形图见图 1、图 2、图 3。

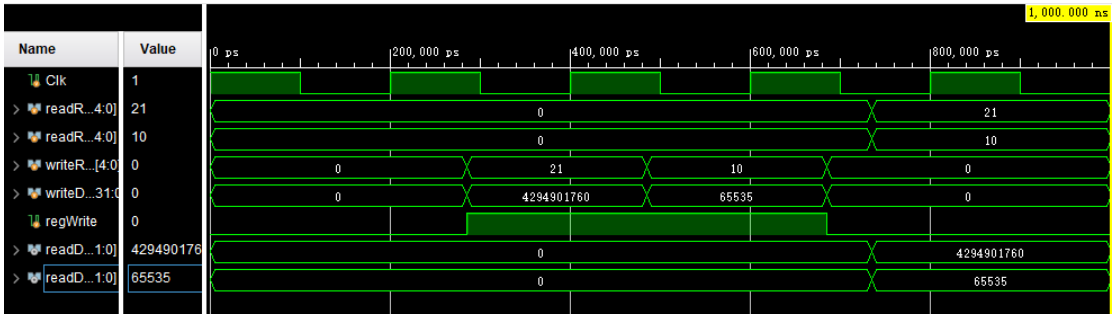

图 1: 寄存器组仿真波形

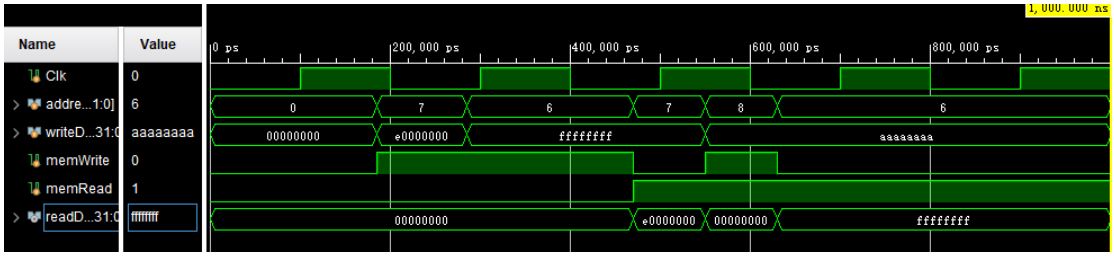

图 2: 数据存储器仿真波形

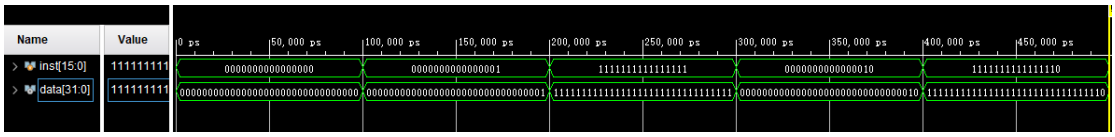

图 3: 符号扩展仿真波形

5 总结与反思

本次实验已经将处理器的大部分组成模块实现了，后续只要拼接起来就可以实现一个简单的单周期处理器了。实验中有几个值得注意的地方：

- 读取和写入在什么时候进行；
- 如何初始化；
- 怎么拼接数字。

在实验报告中和代码中，我也对这些问题进行了说明。