计算机系统结构实验报告　实验 2

简单的类 MIPS 单周期处理器实现：寄存器、存储器与带符号扩展

黄骁 520021910388

2022 年 3 月 22 日

摘要

本实验实现了简单的类 MIPS 处理器的几个重要部件：寄存器 (Register)、存储器 (Data Memory) 以及带符号扩展单元 (Sign Extension)，其作用分别是暂时存放一些数据与运算结果、用于较大量存储数据以及对立即数进行有符号拓展。对于存储部件（如寄存器、存储器等），其需要实现读取数据、存放数据和写入数据的功能。本实验通过软件仿真的形式进行实验结果的验证。

目录

1 实验目的

本次实验有如下几个实验目的：

1. 理解寄存器、数据存储器、带符号扩展单元的 IO 定义
2. Registers 的设计实现
3. Data Memory 的设计实现
4. 带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 (Register) 原理分析

寄存器（Register）是中央处理器内用来暂存指令、数据和地址的电脑存储器，主要接口如图 1 所示。寄存器是指令操作的主要对象，MIPS 中一共有 32 个 32 位的寄存器，用作数据的缓存。

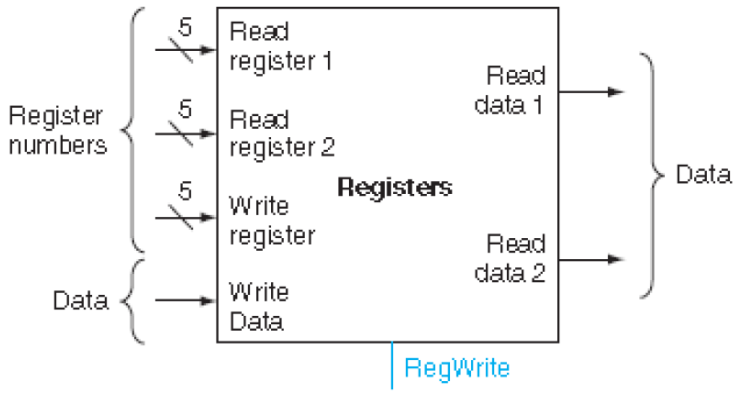

图 1: 寄存器模块的 IO 定义

寄存器的读取数据的操是组合逻辑，只要遇到 ReadReg1，readReg2 中的一个就可以读取。而写操作应当采用时序逻辑，因为如果在 writeReg 信号达到高电平之前 writeReg 没有选中正确的寄存器，或者 writeData 不是正确的数据就会发生错误。可以约定采用时钟下降沿作为写操作的同步信号。

2.2 存储器 (Data Memory) 原理分析

存储器 (Data Memory) 的功能是存储大量的数据，其主要包括只读存储器 (ROM) 以及随机访问存储器 (RAM)。我们设计实现的是后者，其需要支持数据读取、数据写入的功能，主要接口如图 2 所示。内存模块的编写与 register 类似，由于写数据也要考虑信号同步，因此也需要时钟。与寄存器类似地，我们让存储器的写数据的操作仅在时钟下降沿进行。

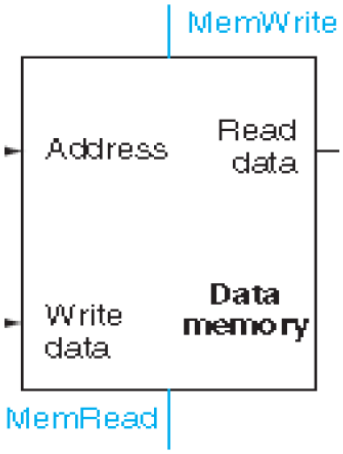

图 2: 内存模块的 IO 定义

2.3 带符号扩展单元 (Sign Extension) 原理分析

带符号扩展单元 (Sign Extension) 的主要功能是将指令中的 16 位有符号立即数拓展为 32 位有符号立即数，因此需要将这个 16 位有符号数的符号位填充在 32 位有符号立即数的高 16 位，再将低 16 位复制到 32 位有符号立即数的低 16 位即可。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。

3 功能实现

3.1 寄存器 (Register) 功能实现

寄存器 (Register) 一直进行读取操作，而由 RegWrite 信号控制是否进行写入操作。在第 2.1 节中我们提到，对于写入操作寄存器将延迟到时钟下降沿时进行写入，这样是为了实现信号同步，同时防止读取到错误数据。而寄存器的读取操作可以一直进行，因为我们会用控制信号来判断读取数据是否为我们需要的数据，而不需要寄存器进行读取数据的检查。

寄存器的部分实现代码如下，完整的代码实现可以参见附录 A.1。

```
reg [31 : 0] regFile [31 : 0];

    always @ (readReg1 or readReg2)
    begin
        if(readReg1)
            readData1 = regFile[readReg1];
        else
            readData1 = 0;
        if(readReg2)
            readData2 = regFile[readReg2];
        else
            readData2 = 0;
    end
```

```
14
15      always @ (negedge clk)
16      begin
17          if (regWrite)
18              regFile[writeReg] = writeData;
19      end
```

3.2 存储器 (Data Memory) 功能实现

存储器 (Data Memory) 由 MemRead 信号控制是否进行读取操作，而由 MemWrite 信号控制是否进行写入操作。在第 2.2 节中我们提到，对于写入操作存储器将延迟到时钟下降沿时进行写入，这样是为了实现信号同步，同时防止读取到错误数据。同时在第 2.2 节中我们提到，需要加入对存储器的地址进行检查的机制，以防止写入不存在的存储单元。

存储器的部分实现代码如下，完整的代码实现可以参见附录 A.2。

```
1       reg [31 : 0] memFile [0 : 1023];
2
3       always @ (address)
4       begin
5           if (memRead && !memWrite)
6               readData = memFile[address];
7           else
8               readData = 0;
9       end
10
11      always @ (negedge clk)
12      begin
13          if (memWrite)
14              memFile[address] = writeData;
15      end
```

3.3 带符号扩展单元 (Sign Extension) 功能实现

带符号扩展单元 (Sign Extension) 将 16 位有符号立即数扩展为 32 位有符号立即数。在第 2.3 节中我们提到，将指令中的 16 位有符号立即数拓展为 32 位有符号立即数，只需要将这个 16 位有符号数的符号位填充在 32 位有符号立即数的高 16 位，再将低 16 位复制到 32 位有符号立即数的低 16 位即可。我们这里使用 Verilog 提供的拼接操作，将 16 为有符号数的符号位复制 16 份，与原数拼接后得到符号扩展后的结果。

带符号扩展单元的部分实现代码如下，完整的实现代码可以参见附录 A.3。

```
1   assign data = { {16 {inst[15]}}, inst[15 : 0] };
```

4 结果验证

4.1 寄存器 (Register) 结果验证

我们使用 Verilog 编写激励文件，采用软件仿真的形式对寄存器 (Register) 模块进行测试（代码实现参见附录 A.1）。我们在激励文件中对寄存器读写的不同情况进行了测试，测试结果如图 3 所示。

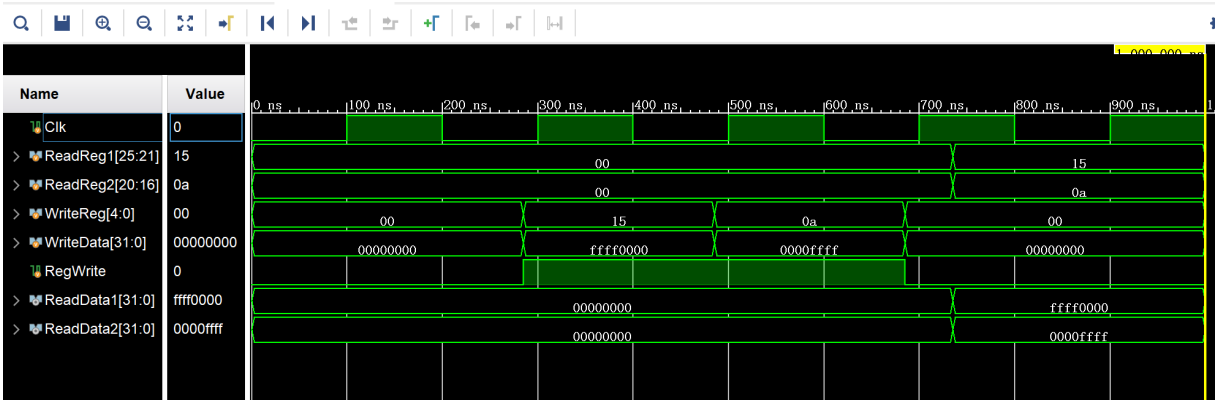

图 3: 对于寄存器 (Register) 的测试结果

我们的测试结果和实验指导书上的测试结果基本一致，因此寄存器的仿真成功。

4.2 存储器 (Data Memory) 结果验证

我们使用 Verilog 编写激励文件，采用软件仿真的形式对存储器 (Data Memory) 模块进行测试（代码实现参见附录 A.2）。我们在激励文件中对存储器读写的不同情况（甚至包含了同时读写的特殊情况）进行了测试，测试结果如图 4 所示。

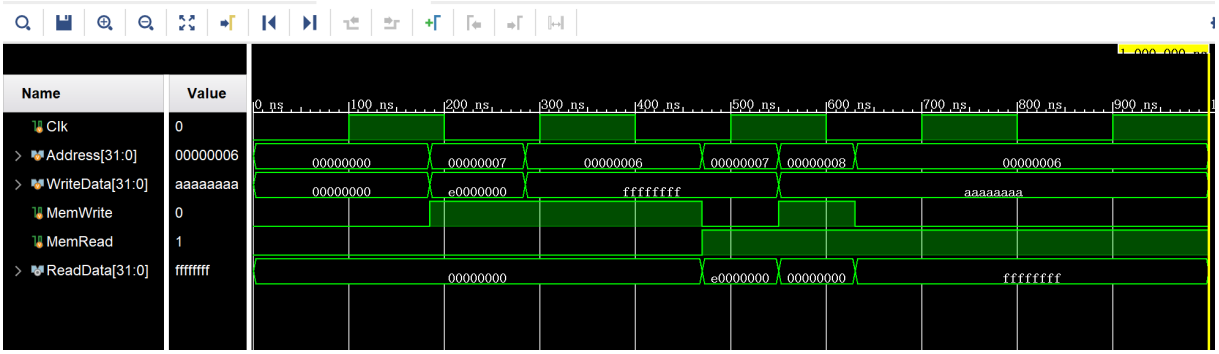

图 4: 对于存储器 (Data Memory) 的测试结果

我们的测试结果和实验指导书上的测试结果基本一致，因此存储器的仿真成功。

4.3 带符号扩展单元 (Sign Extension) 结果验证

我们使用 Verilog 编写激励文件，采用软件仿真的形式对带符号扩展单元 (Sign Extension) 模块进行测试（代码实现参见附录 A.3）。我们在激励文件中对带符号扩展单元的不同输入都进行了测试，测试结果如图 5 所示。

我们的测试结果和实验指导书上的测试结果基本一致，因此带符号扩展单元仿真成功。

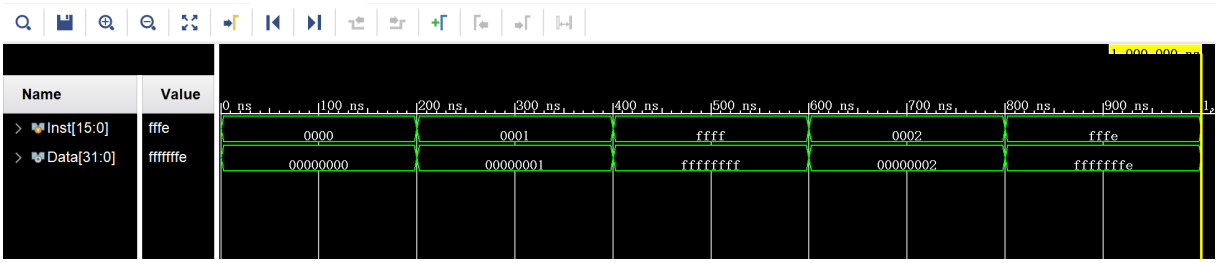

图 5: 对于带符号扩展单元 (Sign Extension) 的测试结果

5 总结与反思

本实验设计并实现了类 MIPS 处理器的三个重要组成部件：寄存器 (Register)、存储器 (Data Memory)、带符号扩展单元 (Sign Extension)，并且通过软件仿真模拟的方法验证了它们的正确性，为后面的单周期类 MIPS 处理器以及流水线处理器的实现奠定基础。控制器和运算器，它们都是组合逻辑单元，而本次实验主要实现的是存储器，存储器的读取功能依然是组合逻辑，而写入功能则是时序逻辑。写入需要考虑时钟同步的问题，要在专门的块中处理。认识到各模块中组合逻辑和时序逻辑的区别，对于后续实现整体的处理器结构是有帮助的。

本次实验的一个重要思想是：延迟存储器、寄存器的写操作到时间下降沿。这样可以同步信号，保证读取数据的正确性，保证程序运行正确。在本次实验中，我对这 MIPS 处理器寄存器、存储器以及带符号扩展单元的有了更加深入的认识，并且有浓厚的兴趣进行最后两次实验。

附录 A　设计文件完整代码实现

A.1　寄存器 (Register) 的代码实现

参见代码文件 `Registers.v` 及 `Registers_tb.v`。

A.2　存储器 (Data Memory) 的代码实现

参见代码文件 `dataMemory.v` 及 `dataMemory_tb.v`。

A.3　带符号扩展单元 (Sign Extension) 的代码实现

参见代码文件 `signext.v` 及 `signext_tb.v`。