

计算机系统结构实验

## 实验 4 报告

简单的类 MIPS 单周期处理器  
部件实现-寄存器、存储器与有符号扩展

李子龙

518070910095

2021 年 5 月 21 日

### 目录

<b>1</b>	<b>实验目的</b>	<b>2</b>
<b>2</b>	<b>原理分析</b>	<b>2</b>
2.1	寄存器堆 . . . . .	2
2.2	内存单元模块 . . . . .	2
2.3	带符号扩展 . . . . .	3
<b>3</b>	<b>代码实现</b>	<b>3</b>
3.1	寄存器堆 . . . . .	3
3.2	内存单元模块 . . . . .	4
3.3	带符号扩展 . . . . .	5
<b>4</b>	<b>仿真结果</b>	<b>5</b>
4.1	寄存器堆 . . . . .	5
4.2	内存单元模块 . . . . .	5
4.3	带符号扩展 . . . . .	7
<b>5</b>	<b>实验心得</b>	<b>8</b>

## 1 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. DataMemory的实现
4. 有符号扩展的实现
5. 使用行为仿真

## 2 原理分析

### 2.1 寄存器堆

寄存器堆是指令操作的主要对象。5 位的 `readReg1`, `readReg2`, `writeReg` 将用于指定需要的寄存器地址，即访问  $32 (= 2^5)$  个寄存器的其中一个。在时钟下沿，并且 `regWrite` 变为可用时，才会写入数据。（在这里规定均在时钟下沿写入数据，这样才不会读取错误的数  
据）读取得到的数据将会通过 `readData1` 和 `readData2` 输出。

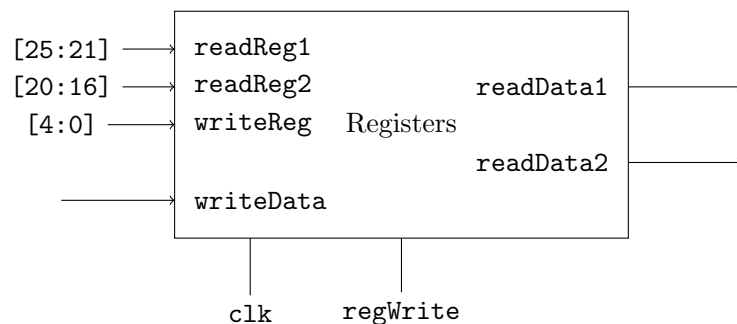


图 1: 寄存器堆模块

### 2.2 内存单元模块

内存模块获取 32 位地址 `address` 的输入。读写控制信号是独立的，一个时钟周期最多激活存储器读 `memRead` 和存储器写 `memWrite` 中的其中一个。需要写入的数据通过 `writeData` 获取，输出的数据从 `readData` 输出。

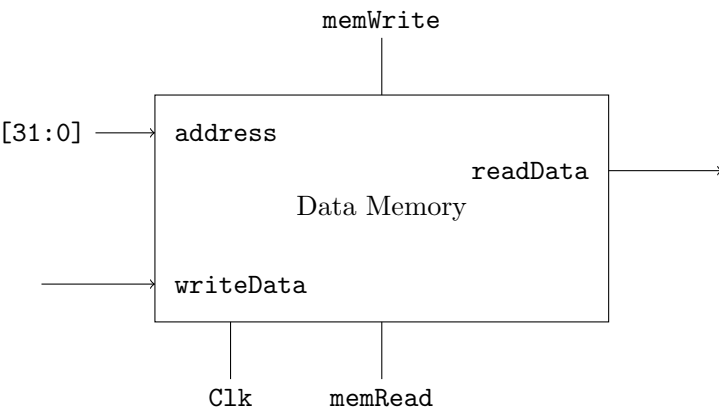


图 2: 内存模块

2.3 带符号扩展

将 16 位数符号扩展为 32 位的符号数。

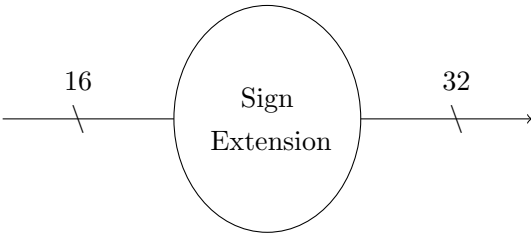


图 3: 带符号扩展

3 代码实现

3.1 寄存器堆

32 位的 MIPS 中共有 32 个 32 位的寄存器，在寄存器内部用下面的语句表示。

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

按照原理，安排读取语句和写入语句。

Listing 1: Registers.v

```
1 reg [31:0] RegFile [31:0];
2 reg [31:0] ReadData1;
3 reg [31:0] ReadData2;
4
```

```
5     always @(readReg1 or readReg2) begin
6         ReadData1 = RegFile[readReg1];
7         ReadData2 = RegFile[readReg2];
8     end
9
10    always @(negedge clk) begin
11        if(regWrite)
12            RegFile[writeReg] = writeData;
13    end
14
15    assign readData1 = ReadData1;
16    assign readData2 = ReadData2;
```

### 3.2 内存单元模块

如果是读数据，如果 `memRead` 被激活或者地址发生了改变，首先会判断地址是否在物理地址空间内（这里存储器只有 64 个，地址空间却为 32 位），如果是就会从 `readData` 输出数据；否则是无效地址将清空读到的数据；如果是写数据，如果 `memWrite` 被激活，而且在物理地址空间内，则会在时钟下沿从 `writeData` 读取数据，并写入寄存器。

**Listing 2:** dataMemory.v

```
1     reg [31:0] MemFile[0:63];
2
3     always @(memRead or address) begin
4         if(address<31'h00000020)
5             readData = MemFile[address];
6         else
7             readData = 31'h00000000;
8     end
9
10    always @(negedge Clk) begin
11        if(memWrite && address<31'h00000020)
12            MemFile[address] = writeData;
13    end
```

### 3.3 带符号扩展

这里采用了扩展复制第一位至前16位的方法进行符号扩展。

**Listing 3:** signext.v

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

## 4 仿真结果

### 4.1 寄存器堆

寄存器堆的激励文件如下。时钟周期被设定为 100ns。

**Listing 4:** Registers\_tb.v

```
1 initial begin
2     readReg1 = 0;
3     readReg2 = 0;
4     writeReg = 0;
5     writeData = 0;
6     regWrite = 0;
7     clk = 0;
8
9     #285;
10    regWrite = 1'b1;
11    writeReg = 21;
12    writeData = 32'b11111111111111111000000000000000;
13
14    #200;
15    writeReg = 10;
16    writeData = 32'b00000000000000000111111111111111;
17
18    #200;
19    regWrite = 1'b0;
20    writeReg = 0;
21    writeData = 32'b00000000000000000000000000000000;
22
23    #50;
24    readReg1 = 5'b10101;
25    readReg2 = 5'b01010;
26 end
```

图 4 显示了仿真结果。在时钟下沿稳定写入信号，向地址 21 写入 FFFF0000，向地址 10 写入 0000FFFF，然后读取地址 21 和 10 的数据，分别为 FFFF0000 和 0000FFFF 输出到 readData1 和 readData2。

### 4.2 内存单元模块

内存单元模块的激励文件如下。

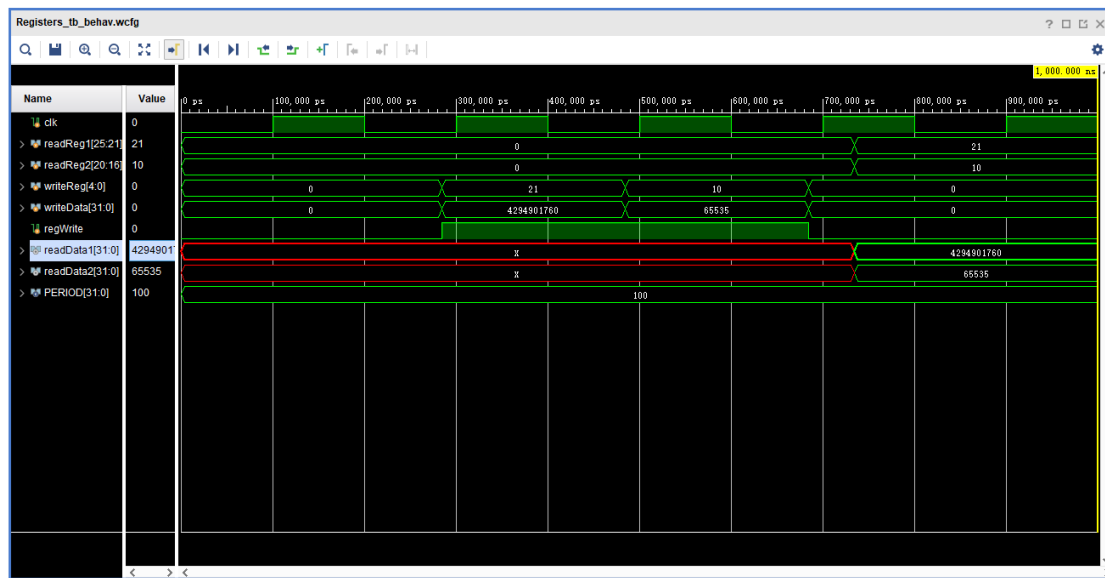


图 4: 寄存器堆的仿真结果

**Listing 5:** dataMemory\_tb.v

```

1      initial begin
2          address = 0;
3          writeData = 0;
4          Clk = 0;
5          memWrite = 0;
6          memRead = 0;
7
8          #185;
9          memWrite = 1'b1;
10         address = 32'b00000000000000000000000000000000111;
11         writeData = 32'b111000000000000000000000000000000;
12
13         #100;
14         memWrite = 1'b1;
15         writeData = 32'hfffffff;
16         address = 32'b00000000000000000000000000000000110;
17
18         #185;
19         memRead = 1'b1;
20         memWrite = 1'b0;
21         address = 32'b00000000000000000000000000000000111;
22
23         #80;
24         memWrite = 1;
25         address = 8;
26         writeData = 32'haaaaaaaa;
27
28         #80;
29         memWrite = 0;
30         memRead = 1;

```

```

31     address = 6;
32     end

```

内存模块的仿真结果如图 5 所示。在后面的读取数据时，地址 7 读到了之前写入的 e0000000，地址 6 读到了之前写入的 aaaaaaaa。仿真结果是正确的。

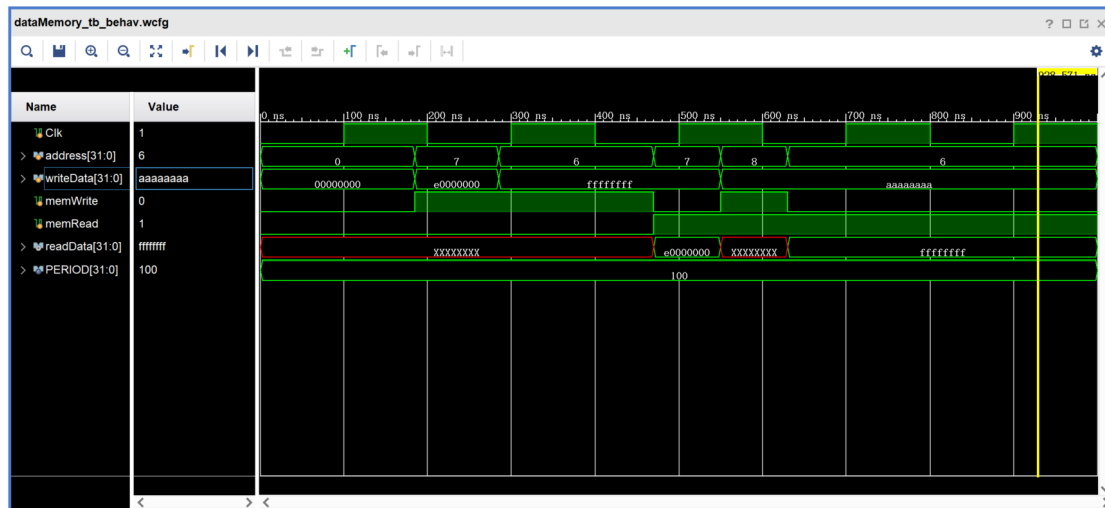


图 5: 存储模块仿真结果

### 4.3 带符号扩展

带符号扩展的激励文件如下。

Listing 6: signext\_tb.v

```

1  initial begin
2      inst = 0;
3
4      #100 inst = 1;
5      #100 inst = -1;
6      #100 inst = 2;
7      #100 inst = -2;
8  end

```

带符号扩展的仿真结果如图 6 所示。对于所有的输入，符号位扩展是正确的。

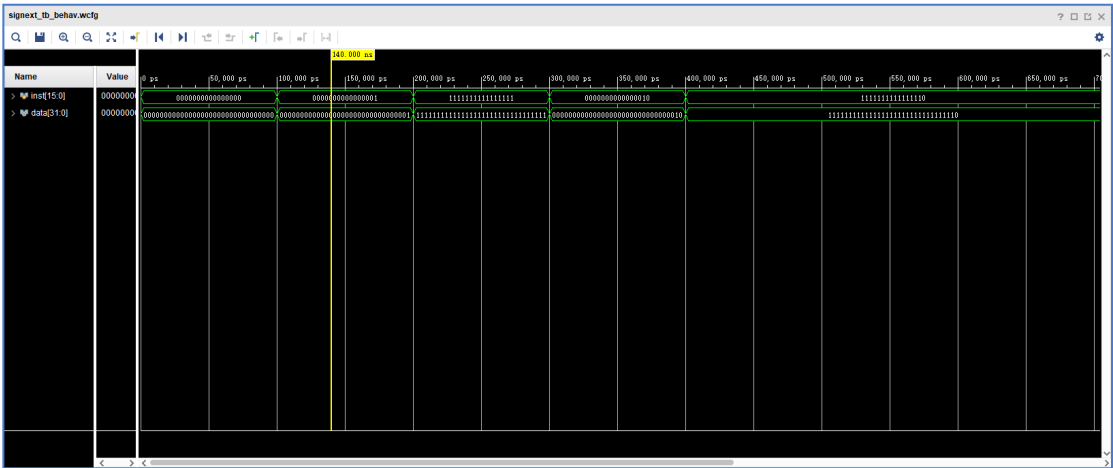


图 6: 带符号扩展仿真结果

## 5 实验心得

通过本次实验，更加熟悉了三个部件的构造方法。特别是时钟下沿的写入，是值得注意的。对于内存地址变化时就应当重新读取这一点也是需要特别注意的，否则会导致地址变化时无响应情形。需要注意是有效地址的内存才可以读取和写入，这一点在之后的设计中也是非常重要的。符号位扩展再次熟悉了拼接操作。