# 计算机系统结构实验Lab04: 单周期CPU部件设计（二）

常烁晨 521021910369

*2023.4.7*

## 摘要

lab04 的实验目标是完成第二部分类MIPS单周期处理器功能部件的设计。在本实验中需要设计的三个功能部件分别是寄存器（Registers）、数据存储器（dataMemory）、有符号扩展部件（signext）。

lab04 所设计的部件与前面 lab03 设计的三个功能部件组合起来，就组成了最基础的单周期 CPU 的主要部分。与前面一样，实验报告主要展示源代码、激励文件和仿真结果。

# 目录

# 1、实验目的

（1）首先理解单周期CPU的寄存器（Registers）、数据存储（dataMemory）、有符号扩展部件（signext）的原理，了解MIPS指令的编码格式。

（2）分别完成寄存器（Registers）、数据存储器（dataMemory）、有符号扩展部件（signext）模块的编写。

（3）使用 Vivado ，分别进行三个功能部件的仿真，确保仿真结果与实验报告要求的一致。

# 2、源代码实现

## 2.1 Registers

MIPS架构的CPU中含有 32 个 32 位寄存器，Registers 模块需要编写这样的寄存器文件，用于在CPU中保存 32 个 32 位二进制数。寄存器文件有五个输入信号和两个输出信号，分别是两个读寄存器的编号、一个写寄存器的编号、一个读写控制信号、一个写入数据，还有两个读出数据。

这里选择时钟的下降沿作为写操作的同步信号。源代码如下。

```
`timescale 1ns / 1ps
module Registers(
    input [25:21] readReg1, input [20:16] readReg2, input [4:0] writeReg,
    input [31:0] writeData, input regWrite, input clk,
    output [31:0] readData1, output [31:0] readData2 );
    reg [31:0] RegFile[31:0]; reg [31:0] ReadData1; reg [31:0] ReadData2;
    assign readData1 = ReadData1; assign readData2 = ReadData2;

    initial begin       RegFile[0] = 0;      end
    always @ (readReg1 or readReg2)
        begin
            ReadData1 = RegFile[readReg1]; ReadData2 = RegFile[readReg2];
        end
    always @ (negedge clk)
        begin
            if(regWrite)
                RegFile[writeReg] = writeData;
        end
endmodule
```

寄存器文件的功能是当时钟下降沿时判断是否有写信号。若有，则把对应的写寄存器编号对应的寄存器写入给定数据。当出现读信号时，从寄存器文件对应寄存器的值输出。

## 2.2 dataMemory

数据内存模块用来模拟计算机中保存数据的内存。根据哈佛结构，指令内存和数据内存分离，此模块完成的是数据内存的编写。与寄存器文件相似，写数据需要考虑同步问题，因此在时钟下降沿完成同步操作。

下面是数据内存源代码，由 64 个 32 位存储单元组成。

```
`timescale 1ns / 1ps
module dataMemory(
    input clk, input [31 : 0] address, input [31 : 0] writeData,
    input memWrite, input memRead, output [31 : 0] readData );
    reg [31:0] ReadData; reg [31:0] MemFile[63:0];
    assign readData=ReadData;
    integer i;
    initial begin
    for(i=0;i<64;i=i+1)
        MemFile[i]=0;
    ReadData=0;
        end
    always @ (memRead or address)
    begin
        if (memRead)
        begin
            if(address <= 63)
                ReadData = MemFile[address];
            else
                ReadData = 0;
        end
    end

    always @ (negedge clk)
    begin

        if(memWrite)
            if(address<=63)
                MemFile[address] = writeData;
    end
```

数据内存有四个输入信号和一个输出信号，分别是输入地址、写入数据、读控制信号、写控制信号，读出数据。首先内存初始化，之后在时钟下降沿检查是否有写控制信号，若有，则将对应地址的内存单元更新为输入数据。当有读控制信号时，将对应地址的内存单元数据输出。

## 2.3 signext

带符号扩展单元的功能是把 16 位有符号数扩展为 32 位有符号数。带符号数的扩展方式十分简单，只需在高 16 位补上原本 16 位数的最高位即可。以下展示源代码。

```
`timescale 1ns / 1ps
module signext(
    input [15:0] inst,
    output [31:0] data
    );
    assign data = { {16 {inst[15]}}, inst[15 : 0] };
endmodule
```

# 3、仿真实现

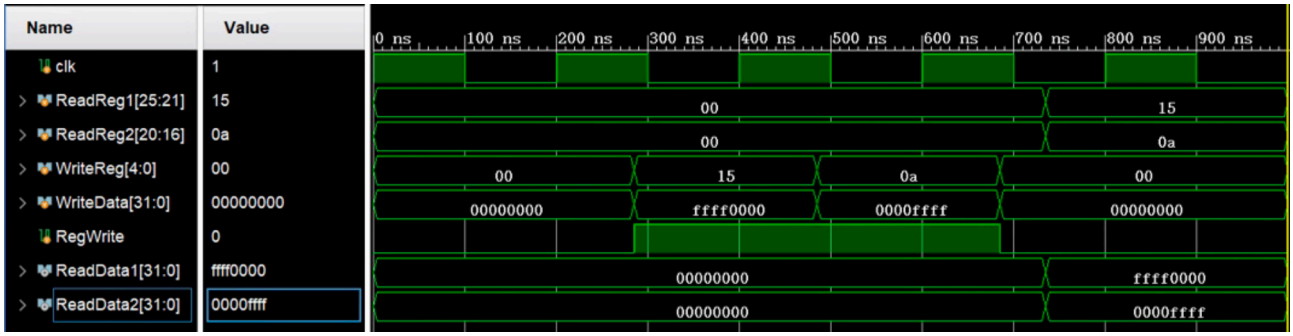
## 3.1 Registers_tb

此处实验指导书提供了仿真结果。依据仿真结果编写对应激励文件如下。

```
`timescale 1ns / 1ps
module Registers_tb();
    reg Clk; 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;
Registers u0(.readReg1(ReadReg1), .readReg2(ReadReg2), .writeReg(WriteReg),
             .writeData(WriteData), .regWrite(RegWrite), .clk(Clk),
             .readData1(ReadData1), .readData2(ReadData2));
     initial begin
        Clk = 0; ReadReg1 = 0; ReadReg2 = 0; WriteReg = 0;
        WriteData = 0; RegWrite = 0;

        #200;   Clk = 1;
        #85;    RegWrite = 1; WriteReg = 5'b10101;
        WriteData = 32'b11111111111111110000000000000000;
        #15;Clk = 0;          #100; Clk = 1;
       #85;     WriteReg = 5'b01010;
        WriteData = 32'b00000000000000001111111111111111;
        #15;Clk = 0;          #100;Clk = 1;
        #85;    RegWrite = 0; WriteReg = 5'b00000;
        WriteData = 32'b00000000000000000000000000000000;
        #15;Clk = 0;          #35; ReadReg1 = 5'b10101;ReadReg2 = 5'b01010;
        #65;Clk = 1;          #100; Clk = 0;
        #100;   end
endmodule
```

仿真结果如下，与实验指导书一致，证明寄存器文件正确。

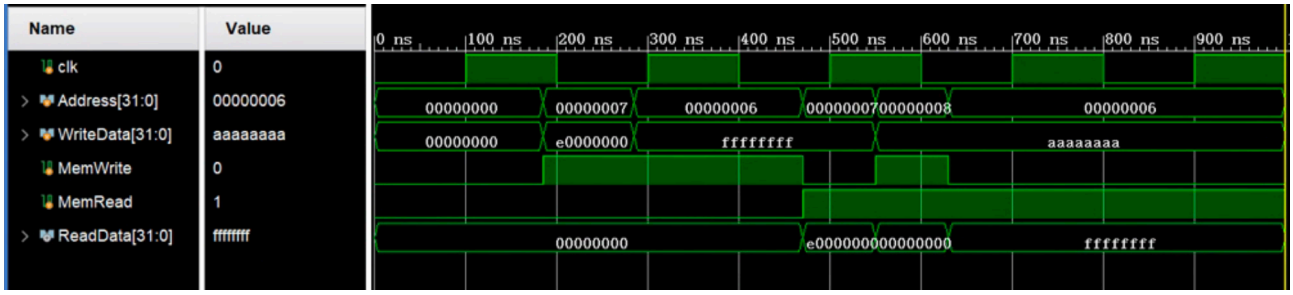



## 3.2 dataMemory_tb

依据仿真样例给出激励文件如下。（补充实验指导书的注释即可）

仿真结果如下，与实验指导书一致，证明寄存器文件正确。

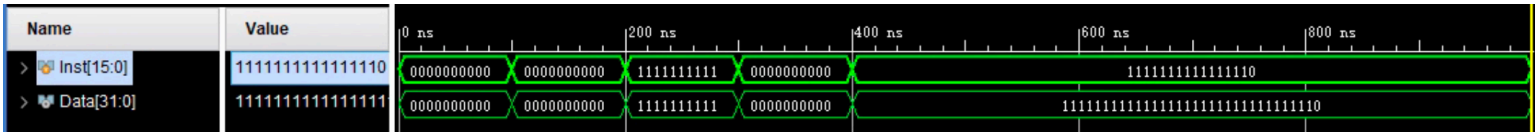
```
    ……
always #100 Clk = ~Clk;
initial begin
    Clk = 0; Address = 0;
    WriteData = 0; MemWrite = 0; MemRead = 0;
    ……
    #185;
    MemRead = 1'b1;
    MemWrite = 0'b0;
    Address = 32'b00000000000000000000000000000111;
    ……
    #80;
    MemWrite = 0;
    MemRead = 1;
    Address = 32’b00000000000000000000000000000110;
    ……
```




## 3.3 signext_tb

原理十分简单，比照指导书上的仿真图例编写激励文件即可。

```
`timescale 1ns / 1ps
module signext_tb();
    reg [15 : 0] Inst; wire [31 : 0] Data;
    signext u0(.inst(Inst), .data(Data));
    initial begin       Inst = 0;
        #100;           Inst = 1;
        #100;           Inst = 16'b1111111111111111;
        #100;           Inst = 2;
        #100;           Inst = 16'b1111111111111110;
    end
endmodule
```



# 4、致谢

感谢刘老师以及三位实验助教学长学姐的悉心指导和帮助。

本次实验我完成的比较顺利，由于本实验结构与 lab3 比较相似，因此完成 lab3 后可以比较得心应手完成 lab4 。尽管如此，还是感谢助教学长学姐指导我关于 signext 的原理。

感谢上海交通大学的计算机系统结构实验室，为我们准备了相关的实验环境以及 FPGA 开发板，学校和学院提供的优秀设备让我们受益匪浅。