Lab05 实验报告

王正 518021910079

一、 实验名称

简单的类 MIPS 单周期处理器的实现

二、 实验目的

完成单周期的类 MIPS 处理器

三、 功能实现

在顶层模块内，将之前两个实验所实现的是所有逻辑模块实例化，在模块之间用信号线连接，使所有模块成为一个整体，并能正确执行指令。

在连接各个模块之前，还需要额外实现模块，使得处理器功能完善。

指令存储器:

由于数据和指令分开存储，因此还需要一个指令存储器。指令存储器按字节寻址的，一个指令占据四个地址的存储空间。这样是为了和后续 PC + 4、立即数左移两位作为跳转地址等运算相协调。

```
module inst_memory(
    input reset,
    input [31:0] readAddress,
    output reg [31:0] inst
    );
    reg [7:0]instMemFile[0:63];
    initial
    begin
        $readmemb("C:/Archlabs/lab05/inst.txt",instMemFile);
     end
    always@(readAddress or reset)
    begin
        if(reset)
            inst=0;
        else
            inst ={ instMemFile[readAddress+3],instMemFile[readAddress+2],
            instMemFile[readAddress+1], instMemFile[readAddress] };
    end
endmodule
```

数据选择器:

将两路数据和选择信号作为数据，输出一路选择的数据。这样实现顶层代码时，

代码可读性会相对好一些。在顶层模块中，既有对 32 位的数据选择器，也有 5 位的数据选择器

```
module mux32(
    input sel,
    input [31:0] input1,
    input [31:0] input2,
    output [31:0] out
    );

    assign out = sel?input1:input2;
endmodule
```

```
module mux5(
    input sel,
    input [4:0] input1,
    input [4:0] input2,
    output [4:0] out
    );

    assign out = sel?input1:input2;
endmodule
```

寄存器模块的修改:

添加 reset 信号，当 reset 为时，将寄存器清零。

```
always @(negedge Clk)
begin
    if(reset)
     begin
     for(i=0;i<32;i=i+1)
         begin
             regFile[i]=0;
         end
     end
```

存储器 dataMemory 的修改:

改为按字节寻址，写入时将一个字的数据写入四个连续地址中:

```
always@ (memRead, address,reset)
begin
    if (reset)
        readData=0;
    else if(memRead)
        readData={memFile[address+3], memFile[address+2], memFile[address+1], memFile[address]};
        else
            readData=0;
end

 always @(negedge Clk)
 begin
   if(memWrite==1'b1)
    begin
        memFile[address]=writeData[7:0];
        memFile[address+1]=writeData[15:8];
        memFile[address+2]=writeData[23:16];
        memFile[address+3]=writeData[31:24];
    end
 end
endmodule
```

顶层模块 Top:

通过若干连接线将之前实现的模块连接起来。

```
module Top(
    input Clk,
    input reset
    //output reg [31:0] PC
    );
    wire [31:0] INST;
    reg [31:0] PC;
    wire [4:0] WRITE_REG;
    wire [31:0] WRITE_DATA;

    wire [3:0] ALU_CTR;
    wire ZERO;
    wire [31:0] ALU_RES;
    wire [1:0] ALU_OP;
    wire REG_DST;
    wire JUMP;
    wire BRANCH;
    wire MEM_READ;
    wire MEM_WRITE;
    wire MEM_TO_REG;
    wire ALU_SRC;
    wire REG_WRITE;
    wire [31:0] READ_DATA;
    wire [31:0] READ_DATA1;
    wire [31:0] READ_DATA2;
    wire [31:0] DATA;
    wire [31:0]IMM_SEXT;
    wire [31:0]ALU_SRC_B;
```

```
    initial begin
        PC=0;
    end

    Ctr mainCtr(
        .OpCode(INST[31:26]),
        .regDst(REG_DST),
        .aluSrc(ALU_SRC),
        .memToReg(MEM_TO_REG),
        .regWrite(REG_WRITE),
        .memRead(MEM_READ),
        .memWrite(MEM_WRITE),
        .branch(BRANCH),
        .aluOp(ALU_OP),
        .jump(JUMP)
    );//

    mux5 WriteRegMux(
        .sel(REG_DST),
        .input1(INST[15:11]),
        .input2(INST[20:16]),
        .out(WRITE_REG)
    );//
```

```
ALU alu(
    .input1(READ_DATA1),
    .input2(ALU_SRC_B),
    .aluCtr(ALU_CTR),
    .zero(ZERO),
    .aluRes(ALU_RES),
    .reset(reset)
);//

mux32 AluSrcMux(
    .sel(ALU_SRC),
    .input1(IMM_SEXT),
    .input2(READ_DATA2),
    .out(ALU_SRC_B)
);//

ALUCtr mainALUCtr(
    .ALUOp(ALU_OP),
    .Funct(INST[5:0]),
    .reset(reset),
    .ALUCtrOut(ALU_CTR)
);//

dataMemory maindataMemory(
    .Clk(Clk),
    .reset(reset),
    .address(ALU_RES),
    .writeData(READ_DATA2),
    .memWrite(MEM_WRITE),
    .memRead(MEM_READ),
    .readData(READ_DATA)
);//

mux32 RegWriteMux(
    .sel(MEM_TO_REG),
    .input1(READ_DATA),
    .input2(ALU_RES),
    .out(WRITE_DATA)
);//

Registers Registers(
    .reset(reset),
    .readReg1(INST[25:21]),
    .readReg2(INST[20:16]),
    .writeReg(WRITE_REG),
    .writeData(WRITE_DATA),
    .regWrite(REG_WRITE),
    .Clk(Clk),
    .readData1(READ_DATA1),
    .readData2(READ_DATA2)
);//
```

```
    signext mainsignext(
        .inst(INST[15:0]),
        .data(IMM_SEXT),
        .reset(reset)//
    );

    inst_memory inst_memory(
        .reset(reset),
        .readAddress(PC),
        .inst(INST)
    );//

   wire [31:0] PC_4;
   wire [31:0] Branch_addr;
   wire [31:0] sel_Branch_addr;
   wire [31:0] Jump_addr;
   wire [31:0] next;
   wire [31:0] Sext_shift;

  assign PC_4=PC+4;
  assign Jump_addr={PC_4[31:28],INST[25:0]<<2};
  assign Sext_shift=IMM_SEXT<<2;
  assign Branch_addr= PC_4+Sext_shift;

  mux32 BranchMux(
    .sel(BRANCH & ZERO),
    .input1(Branch_addr),
    .input2(PC_4),
    .out(sel_Branch_addr)
  );

    mux32 JumpMux(
    .sel(JUMP),
    .input1(Jump_addr),
    .input2(sel_Branch_addr),
    .out(next)
    );

    always@(negedge Clk)
    begin
       if(reset)
         PC=0;
       else
        PC=next;
    end


endmodule
```

四、 仿真测试

- 测试程序

```
START:
lw $1, 0($0)
lw $2, 4($0)
lw $3, 8($0)
OP:
add $1, $1, $2
or $4, $1, $2
slt $5, $3, $4
sw $4, 12($0)
beq $2, $3, OP
j START
```

- 测试数据:

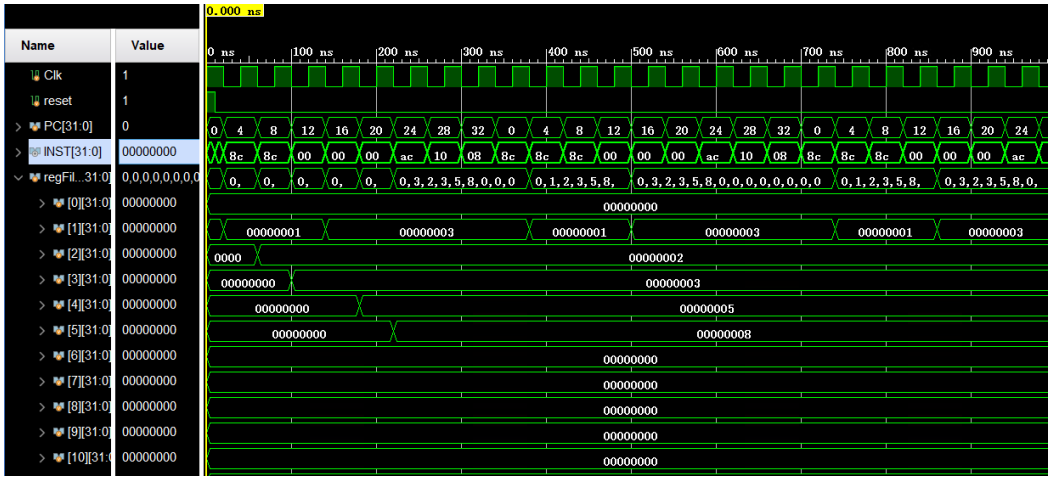
```
0x00-0x03 1
0x04-0x07 2
0x08-0x0B 3
0x0C-0x0F 4
```

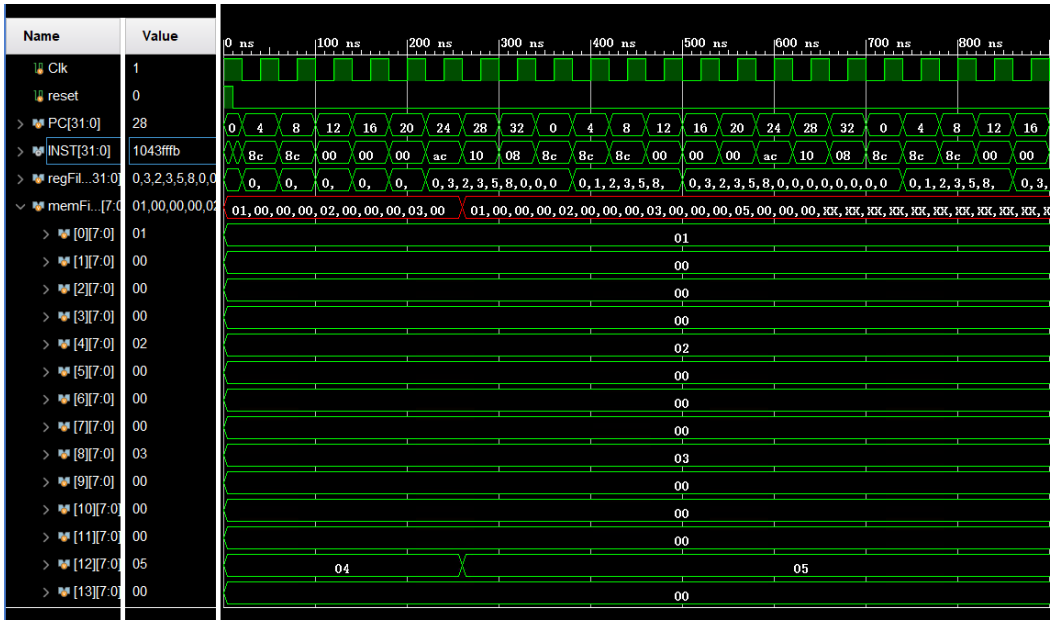
- 仿真激励文件

clk 周期为 40ns，reset 保持高电平 10ns:

```
module Top_tb();
    reg Clk;
    reg reset;
    always #20 Clk=!Clk;
    Top uo(
        .Clk(Clk),
        .reset(reset) );
    initial begin
    Clk=1;
    reset=1;
    #10
    reset=0;
    #2000;
    end
endmodule
```

◆ 仿真波形





通过观察图中 PC、INST、regFile、memFile 的数值，仿真波形符合逻辑预期，单周期处理器仿真测试成功。

五、 心得体会

这次实验是对 Lab03 和 Lab04 的一个综合，要将之前所实现的所有部件组合在一起，并且需要添加一些新的部件。因此，这次实验的难度也是大幅度提升。

而这次实验的关键点在于正确地把合适的模块用正确的连线相连。这项工作耗时耗力，需要有足够的耐心和细心。通过对程序的反复调试，我对单周期处理器的工作原理有了更深入的了解。