实验报告

学 号 21311274 姓 名 林宇浩

【实验题目】单周期 CPU 设计(3)

【实验目的】用 FPGA 芯片实现单周期的 MIPS CPU。

【实验工具】

BASYS3 实验板，Vivado 软件

【注意事项】

* 在忽略溢出的前提下，addi 与 addiu 等价， add 与 addu 等价。addiu 也会进行符号扩展。
* slt 和 slti 都是有符号比较。可采用减法判定有符号数大小。
* 数据存储器需要在时钟上升沿写入数据。
* 可采用 for 语句对寄存器组进行初始化。

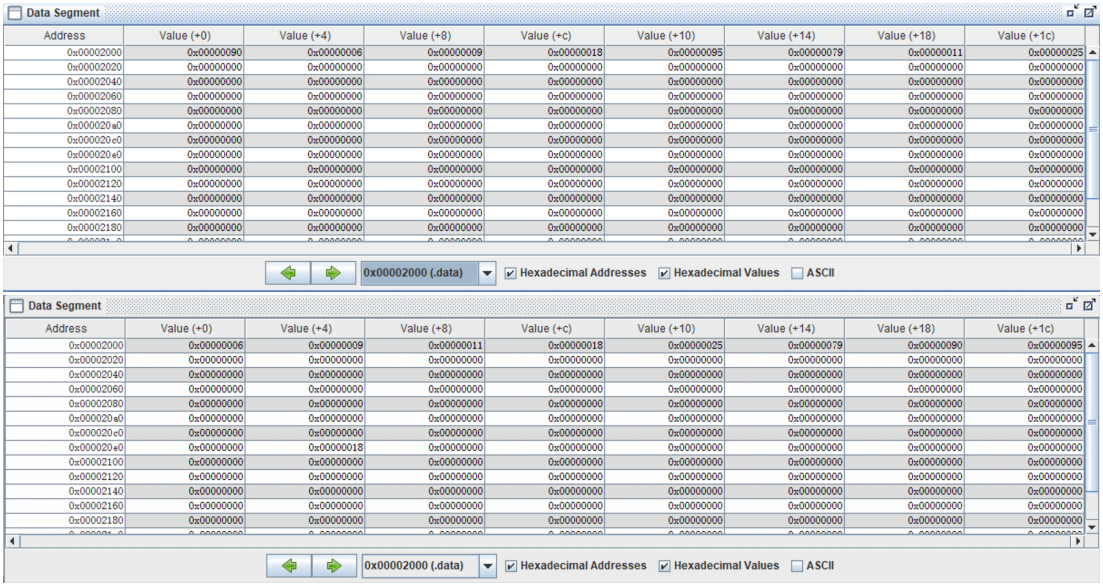
【实验要求】

设计单周期 CPU，可以正确运行课本中的冒泡排序程序。

【实验要求】

1、(sort.asm)用 Mars 运行课本中的冒泡排序程序(sort8.asm)，要求排序以下 8 个 32 位数：

给出排序前和排序后的数据段（Data Segment）截屏：



源程序：

```
# bubble sort for eight numbers
.data
nums: .word 0x90,6,9,0x18,0x95,0x79,0x11,0x25
.space 200
sptop:

.text
.globl main

main:
  la $sp,200
  #la $sp,sptop                # 仅在 Mars 运行时启用，dump 时注销
  la $a0,0
```

```
  #la $a0,nums                # 仅在 Mars 运行时启用，dump 时注销
  li $a1,8
  jal sort
  la $s0,0
  #la $s0,nums                # 仅在 Mars 运行时启用，dump 时注销
  lw $s1,0($s0)
  lw $s1,4($s0)
  lw $s1,8($s0)
  lw $s1,12($s0)
  lw $s1,16($s0)
  lw $s1,20($s0)
  lw $s1,24($s0)
  lw $s1,28($s0)
loop:
   j loop

#v(数组) in $a0, k(个数) in $a1, i s0, j s1
sort:
   addi $sp,$sp,-20      # make room on stack for 5 registers

   sw $ra, 16($sp)
   sw $s3,12($sp)
   sw $s2, 8($sp)
   sw $s1, 4($sp)
   sw $s0, 0($sp)

   move $s2, $a0
   move $s3, $a1

   move $s0, $zero           # i = 0
for1tst:
   slt    $t0, $s0, $s3          # $t0 = 0 if $s0 ≥ $s3 (i ≥ n)
   beq  $t0, $zero, exit1    # go to exit1 if $s0 ≥ $s3 (i ≥ n)
   addi $s1, $s0,-1             # j = i  -  1
for2tst:
   slti    $t0, $s1, 0             # $t0 = 1 if $s1 < 0 (j < 0)
   bne  $t0, $zero, exit2    # go to exit2 if $s1 < 0 (j < 0)
   sll  $t1, $s1, 2               # $t1 = j * 4
   add  $t2, $s2, $t1          # $t2 = v + (j * 4)
   lw   $t3, 0($t2)               # $t3 = v[j]
   lw   $t4, 4($t2)               # $t4 = v[j + 1]
   slt  $t0, $t4, $t3             # $t0 = 0 if $t4 ≥ $t3
   beq  $t0, $zero, exit2     # go to exit2 if $t4 ≥ $t3
   move $a0, $s2               # 1st param of swap is v (old $a0)
   move $a1, $s1               # 2nd param of swap is j
   jal  swap                      # call swap procedure
```

```
   addi $s1, $s1,-1             # j -= 1
    j    for2tst                     # jump to test of inner loop
exit2:
    addi $s0, $s0, 1             # i += 1
     j    for1tst                     # jump to test of outer loop
exit1:
    lw $ra,16($sp)
    lw $s0, 0($sp)
    lw $s1, 4($sp)
    lw $s2, 8($sp)
    lw $s3,12($sp)
    addi $sp,$sp, 20
    jr $ra


swap:
    sll $t1, $a1, 2   # $t1 = k * 4
    add $t1, $a0, $t1 # $t1 = v+(k*4)
                              #   (address of v[k])
    lw $t0, 0($t1)    # $t0 (temp) = v[k]
    lw $t2, 4($t1)    # $t2 = v[k+1]
    sw $t2, 0($t1)    # v[k] = $t2 (v[k+1])
    sw $t0, 4($t1)    # v[k+1] = $t0 (temp)
    jr $ra            # return to calling routine
```

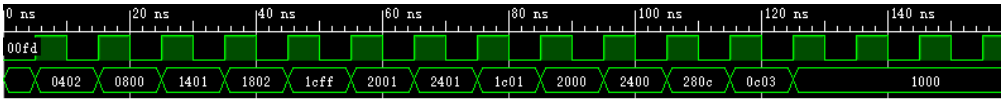
用“单周期 CPU 设计(a).pdf”的方法 dump 机器代码（注意注释掉 Mars 使用的相关语句）：

```
241d00c8
24040000
24050008
0c00000e
24100000
8e110000
8e110004
8e110008
8e11000c
8e110010
8e110014
8e110018
8e11001c
0800000d
23bdffec
afbf0010
afb3000c
afb20008
afb10004
afb00000
00049021
```

00059821
00008021
0213402a
11000010
2211ffff
2a280000
1500000b
00114880
02495020
8d4b0000
8d4c0004
018b402a
11000005
00122021
00112821
0c000030
2231ffff
0800001a
22100001
08000017
8fbf0010
8fb00000
8fb10004
8fb20008
8fb3000c
23bd0014
03e00008
00054880
00894820
8d280000
8d2a0004
ad2a0000
ad280004
03e00008

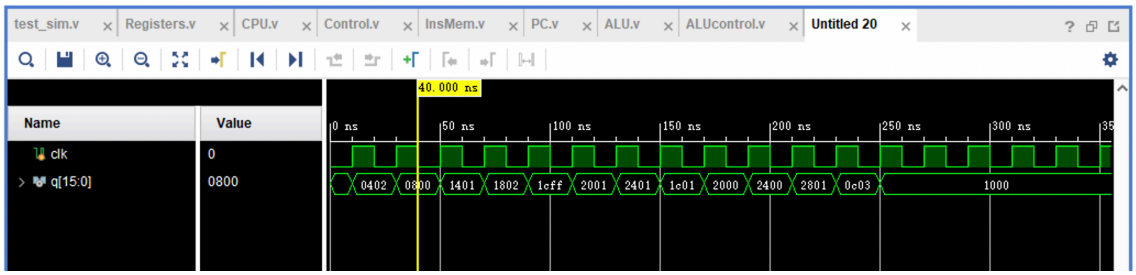
2、（仿真）实现执行 file3.asm 并仿真，仿真执行包含最后读取数据区数据的 8 条语句的 PC 的低 8 位和 WriteData 的低 8 位。

参考：



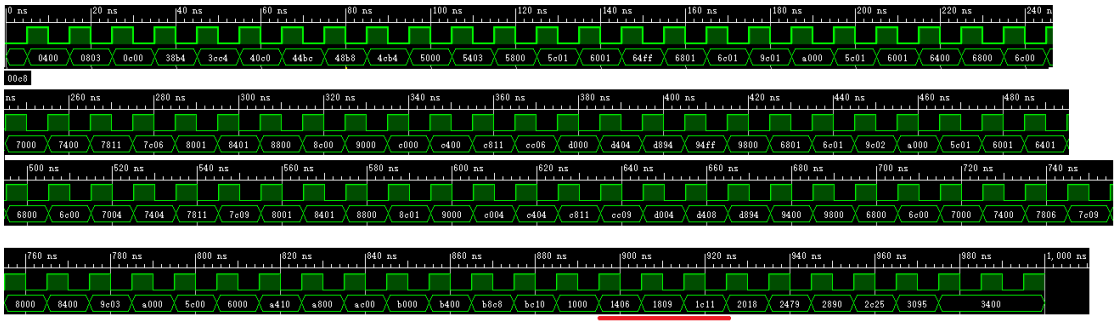
你的仿真图：

实验报告

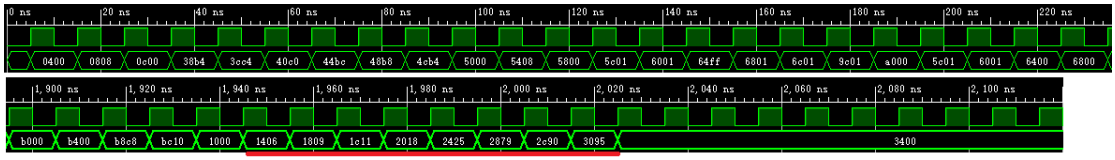


3、（**仿真**）实现执行冒泡排序并仿真，仿真执行包含最后读取数据区数据的 8 条语句的 PC 的低 8 位和 WriteData 的低 8 位。

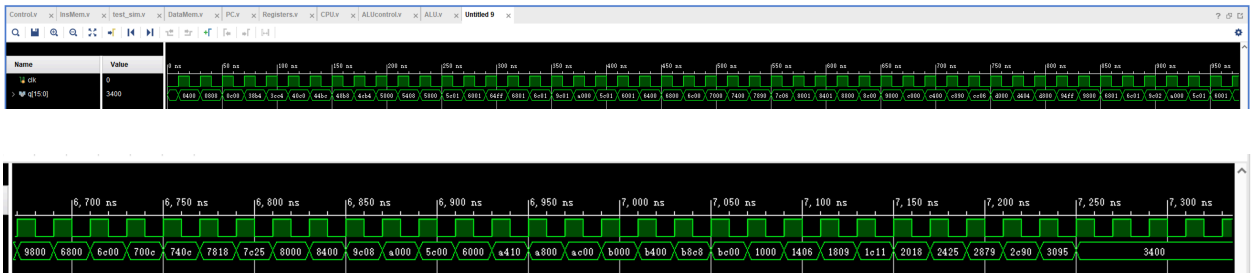
参考：排序 3 个数



参考：排序 8 个数



仿真图：



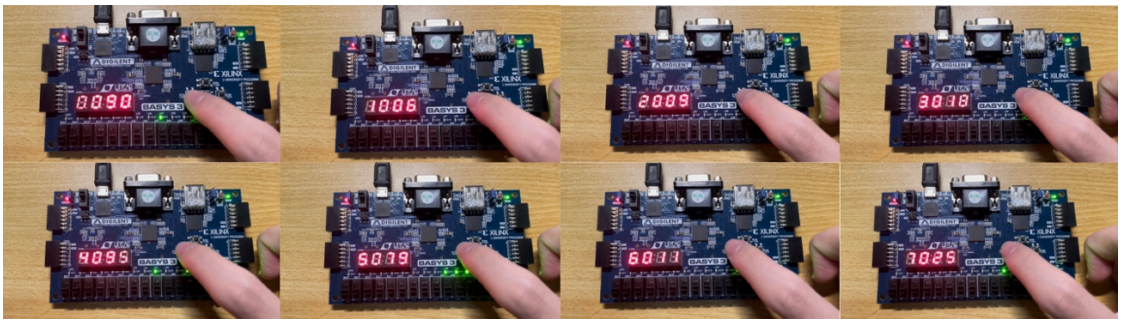
4、（**写板**）实现写板，每次按 BtnL 键在数码管上输出数据存储器中的内容，输出的第一个数字是序号，后三个数字是内容，循环输出。一开始没有排序，在按下按 BtnC 键后，才开始排序，然后按 BtnL 键可以看到排序结果。

拍排序视频(sort8.mp4)放在打包文件中,一定要尽量短，上传文件限制 80M。

完成后拍照 8 个数据排序前的结果，拼在一张图上(最好不要超过 300KB)：

[1] [2] [3] [4]

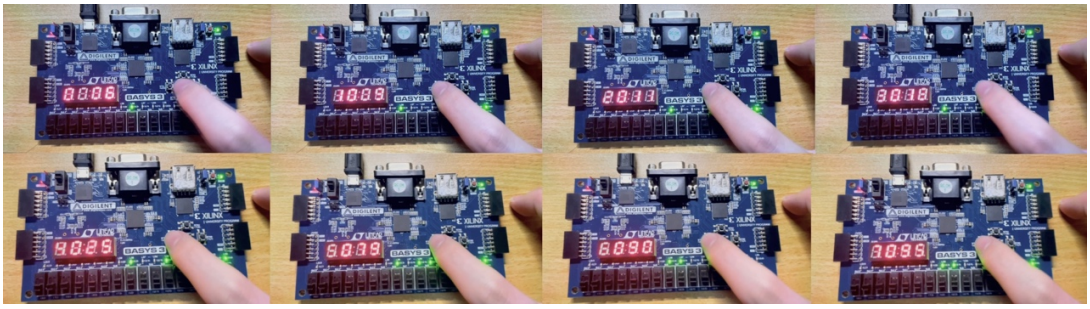
[5] [6] [7] [8]



拍照 8 个数据排序后的结果，拼在一张图上(最好不要超过 300KB)：

[1] [2] [3] [4]

[5] [6] [7] [8]



5、模块源码（名字可变，也可增减）：

[PC]

```
module PC(clk,pc,readAddress,mux5,ins,jal,ra,start);
    input clk,jal,start;
    input [31:0] mux5,ins,ra;
    output reg [31:0] pc;
    output reg [31:0] readAddress;
    reg ok;

    initial begin
        ok = 0;
```

```
        pc = 4;
        readAddress=0;
    end

    always@(posedge start)
        ok = 1;

    always@(posedge clk) begin
        if(ok==0) begin
            pc = 4;
            readAddress=0;
        end
        else if(ins[31:26]==0&&ins[5:0]==6'b001000) begin//jr
            readAddress=ra;
            pc=ra;
            pc=pc+4;
        end
        else begin
            readAddress = mux5;
            pc = mux5;
            pc = pc+4;
        end
    end;
endmodule

[InsMem]
```

```
module InsMem(input wire [31:0] pc, output reg[31:0] ins);

    reg[31:0] mem[255:0];

    initial begin

        mem[0] = 32'h241d00c8;

        mem[1] = 32'h24040000;

        mem[2] = 32'h24050008;//

        mem[3] = 32'h0c00000e;

        mem[4] = 32'h24100000;

        mem[5] = 32'h8e110000;

        mem[6] = 32'h8e110004;

        mem[7] = 32'h8e110008;

        mem[8] = 32'h8e11000c;

        mem[9] = 32'h8e110010;

        mem[10] = 32'h8e110014;

        mem[11] = 32'h8e110018;

        mem[12] = 32'h8e11001c;

        mem[13] = 32'h0800000d;

        mem[14] = 32'h23bdffec;

        mem[15] = 32'hafbf0010;

        mem[16] = 32'hafb3000c;

        mem[17] = 32'hafb20008;

        mem[18] = 32'hafb10004;

        mem[19] = 32'hafb00000;

        mem[20] = 32'h00049021;

        mem[21] = 32'h00059821;

        mem[22] = 32'h20100000;//
```

```
mem[23] = 32'h0213402a;

mem[24] = 32'h11000010;

mem[25] = 32'h2211ffff;

mem[26] = 32'h2a280000;

mem[27] = 32'h1500000b;

mem[28] = 32'h00114880;

mem[29] = 32'h02495020;

mem[30] = 32'h8d4b0000;

mem[31] = 32'h8d4c0004;

mem[32] = 32'h018b402a;

mem[33] = 32'h11000005;

mem[34] = 32'h00122021;

mem[35] = 32'h00112821;

mem[36] = 32'h0c000030;

mem[37] = 32'h2231ffff;

mem[38] = 32'h0800001a;

mem[39] = 32'h22100001;

mem[40] = 32'h08000017;

mem[41] = 32'h8fbf0010;

mem[42] = 32'h8fb00000;

mem[43] = 32'h8fb10004;

mem[44] = 32'h8fb20008;

mem[45] = 32'h8fb3000c;

mem[46] = 32'h23bd0014;

mem[47] = 32'h03e00008;

mem[48] = 32'h00054880;
```

```
        mem[49] = 32'h00894820;
        mem[50] = 32'h8d280000;
        mem[51] = 32'h8d2a0004;
        mem[52] = 32'had2a0000;
        mem[53] = 32'had280004;
        mem[54] = 32'h03e00008;

//        $readmemh("C:\Users\lin\Desktop\filmtext.txt",mem);
    end
    always@(pc)
    ins = mem[pc>>2]; //要用 always
endmodule
```

[CPU]

```
module CPU(clk,clk2,sm_wei,sm_duan,start,btnl);
    input clk;//用于 cpu
    input clk2;//smg
    input start;//btnc
    input btnl;//btnl
    output wire [3:0] sm_wei;
    output wire [6:0] sm_duan;

//    wire[15:0] q;
//    assign q[15:8] = readAddress[7:0];
//    assign q[7:0] = mux3[7:0];
```

```
        wire regDst,jump,branch,memRead,memToReg,aluop,memWrite,aluSrc,regWrite,jal;

        wire [31:0] pc,ra;

        wire [31:0] readAddress;

        wire [31:0] ins;

        wire [4:0] mux1;

        wire [31:0] readData1;

        wire [31:0] readData2;

        wire [3:0] aluOp,aluCtr;

        wire [31:0] data,aluResult,readData;

        wire [31:0] mux2,mux3,mux4,mux5;

        wire [27:0] jumpAdr;

        assign jumpAdr=ins[25:0]<<2;

        wire[31:0] shiftLeft;

        assign shiftLeft=data<<2;

        wire[15:0] smg;



        PC uPC(clk,pc,readAddress,mux5,ins,jal,ra,start);

        InsMem uInsMem(readAddress,ins);

        Control
uControl(ins[31:26],regDst,jump,branch,memRead,memToReg,aluOp,memWrite,aluSrc,regWrite,jal,i
ns);

        Mux #(5) Mux1(ins[20:16], ins[15:11], regDst, mux1);

        Registers
uRegisters(clk,regWrite,ins[25:21],ins[20:16],mux1,mux3,readData1,readData2,jal,pc,ra);

        SignExtend uSignExtend(ins[15:0],data);

        Mux #(32) Mux2(readData2, data, aluSrc, mux2);

        ALUcontrol uALUcontrol(aluOp,ins[5:0],aluCtr);
```

```
    ALU uALU(aluCtr,readData1,mux2,aluResult,zero,ins[10:6]);
    DataMem uDataMem(clk,memWrite,memRead,aluResult,readData2,readData,btnl,smg);
    Mux #(32) Mux3(aluResult,readData,memToReg,mux3);
    Mux #(32) Mux4(pc,pc+shiftLeft,branch & zero,mux4);
    Mux #(32) Mux5(mux4,{pc[31:28],jumpAdr},jump,mux5);

    display udisplay(clk2,smg,sm_wei,sm_duan);
endmodule

[CPU_sim]
module CPU_sim();
    reg clk,clk2,start,btnl;
    wire [3:0] sm_wei;
    wire [6:0] sm_duan;

     CPU uCPU(clk,clk2,sm_wei,sm_duan,start,btnl);

     initial begin
        clk = 0;
        clk2 =0;
        start=0;
        btnl=0;
     end

     reg [3:0] count=0;
     always begin
```

```
        #5;
        if(count<8) begin
            count = count + 1;
            btnl=~btnl;
            #5;
            btnl=~btnl;
        end
    end
    always #1500 count = 0;

    always #100 start = 1;

    always #2 clk = ~clk;
    always #1 clk2 = ~clk2;

endmodule
```

[DataMem]

```
module DataMem(clk,memWrite,memRead,address,writeData,readData,btnl,smg);
    input clk,btnl;
    input memWrite,memRead;
    input [31:0] address;
    input [31:0] writeData;
    output [31:0] readData;
    output [15:0] smg;
    reg [31:0] mem[500:0];
```

```
    initial begin
//        mem[0] = 32'h00000011;
//        mem[1] = 6;
//        mem[2] = 9;
        mem[0] = 32'h00000090;
        mem[1] = 6;
        mem[2] = 9;
        mem[3] = 32'h00000018;
        mem[4] = 32'h00000095;
        mem[5] = 32'h00000079;
        mem[6] = 32'h00000011;
        mem[7] = 32'h00000025;
    end
    always@(posedge clk) begin
        if (memWrite) mem[address>>2]=writeData;
    end
    assign readData=mem[address>>2];

    reg [3:0] count;
    initial begin
        count = 0;
    end
    always@(posedge btnl) begin
        count = count+1;
        if(count>7)
            count = 0;
```

```
    end

    assign smg={count,mem[count][11:0]};


endmodule


[ALU]
module ALU(aluCtr,A,B,aluResult,zero,shamt,ins);
        input [3:0] aluCtr;
        input [31:0] A,ins;
        input [31:0] B;
        input [4:0] shamt;
        output [31:0] aluResult;
        output zero;
        reg [32-1:0] aluResult;
        reg C,zero;
        always@(*)
        begin
            C=0;
            case(aluCtr)
                4'b0000:begin aluResult=A&B; end    //按位与
                4'b0001:begin aluResult=A|B; end    //按位或
                4'b1100:begin aluResult=A^B; end    //按位异或
                4'b0101:begin aluResult=B>>A; end //将 B 右移 A 位
                4'b0010:begin {C,aluResult}=A+B; zero = aluResult==0;end //加法
                4'b0110:begin {C,aluResult}=A-B; zero = aluResult==0;end //减法
                4'b1110:begin {C,aluResult}=A-B; zero = aluResult!=0; aluResult=zero; end
```

```
//bne
                    4'b0111:begin
                        if(A[31]==0&&B[31]==0) aluResult=A<B;
                        if(A[31]==1&&B[31]==0) aluResult=1;
                        if(A[31]==0&&B[31]==1) aluResult=0;
                        if(A[31]==1&&B[31]==1) aluResult=A<B;
                    end//A<B 则 F=1，否则 F=0 slt
                    4'b0011:begin aluResult=B<<shamt; end   //将 B 左移 A 位
                    4'b1000:begin aluResult=0; end   //j
                endcase;
            end
  endmodule


[Control]

module
Control(opCode,regDst,jump,branch,memRead,memToReg,aluop,memWrite,aluSrc,regWrite,jal,ins);
    input [5:0] opCode;
    input [31:0] ins;
    output reg regDst;
    output reg jump;
    output reg branch;
    output reg memRead;
    output reg memToReg;
    output reg[3:0] aluop;
    output reg memWrite;
    output reg aluSrc;
```

```
    output reg regWrite;
    output reg jal;

    always@(opCode) begin
        case(opCode)
            6'b000010: begin //J
            regDst = 0;  aluSrc = 0; memToReg = 0;
            regWrite = 0; memRead = 0; memWrite = 0; branch = 0; aluop = 4'b1000; jump = 1; jal=0;
            end
            6'b000011: begin //jal
            regDst = 0;  aluSrc = 0; memToReg = 0;
            regWrite = 0; memRead = 0; memWrite = 0; branch = 0; aluop = 4'b0000; jump = 1; jal=1;
            end
            6'b000000: begin//R
                if(ins[5:0]==6'b001000) begin//jr
                    regDst = 0;  aluSrc = 0; memToReg = 0;
                    regWrite = 0; memRead = 0; memWrite = 0; branch = 0; aluop = 4'b1000; jump
= 1; jal=0;
                end
                else begin
                    regDst = 1;  aluSrc = 0; memToReg = 0;
                    regWrite = 1; memRead = 0; memWrite = 0; branch = 0; aluop = 4'b1111; jump
= 0; jal=0;
                end
            end
            6'b100011: begin//lw
            regDst = 0;  aluSrc = 1; memToReg = 1;
```

```
regWrite = 1; memRead = 1; memWrite = 0; branch = 0; aluop = 4'b0000; jump = 0; jal=0;

end

6'b101011: begin//sw

regDst = 0;  aluSrc = 1; memToReg = 0;

regWrite = 0; memRead = 0; memWrite = 1; branch = 0; aluop = 4'b0000; jump = 0; jal=0;

end

6'b000100: begin//beq

regDst = 0;  aluSrc = 0; memToReg = 0;

regWrite = 0; memRead = 0; memWrite = 0; branch = 1; aluop = 4'b0001; jump = 0; jal=0;

end

6'b000101: begin//bne

regDst = 0;  aluSrc = 0; memToReg = 0;

regWrite = 0; memRead = 0; memWrite = 0; branch = 1; aluop = 4'b0110; jump = 0; jal=0;

end

6'b001000: begin//addi

regDst = 0;  aluSrc = 1; memToReg = 0;

regWrite = 1; memRead = 0; memWrite = 0; branch = 0; aluop = 4'b0000; jump = 0;jal=0;

end

6'b001001: begin//addiu(addi)

regDst = 0;  aluSrc = 1; memToReg = 0;

regWrite = 1; memRead = 0; memWrite = 0; branch = 0; aluop = 4'b0000; jump = 0;jal=0;

end

6'b001100: begin//andi

regDst = 0;  aluSrc = 1; memToReg = 0;

regWrite = 1; memRead = 0; memWrite = 0; branch = 0; aluop = 4'b0100; jump = 0; jal=0;

end
```

```
            6'b001101: begin//ori
            regDst = 0;  aluSrc = 1; memToReg = 0;
            regWrite = 1; memRead = 0; memWrite = 0; branch = 0; aluop = 4'b0010; jump = 0; jal=0;
            end
            6'b001110: begin//xori
            regDst = 0;  aluSrc = 1; memToReg = 0;
            regWrite = 1; memRead = 0; memWrite = 0; branch = 0; aluop = 4'b1100; jump = 0; jal=0;
            end
            6'b001010: begin//slti
            regDst = 0;  aluSrc = 1; memToReg = 0;
            regWrite = 1; memRead = 0; memWrite = 0; branch = 0; aluop = 4'b0011; jump = 0; jal=0;
            end
            6'b001111: begin//luui
            regDst = 0;  aluSrc = 1; memToReg = 0;
            regWrite = 1; memRead = 0; memWrite = 0; branch = 0; aluop = 4'b1011; jump = 0; jal=0;
            end
            default: begin
            regDst = 0;  aluSrc = 0; memToReg = 0;
            regWrite = 0; memRead = 0; memWrite = 0; branch = 0; aluop = 4'b0000; jump = 0; jal=0;
            end
        endcase
    end
endmodule

[display]
module display(clk,data,sm_wei,sm_duan);
```

```
input clk;

input [15:0] data;

output [3:0] sm_wei;

output [6:0] sm_duan;


//分频

integer clk_cnt;

reg clk_400Hz;

always @(posedge clk)

    if(clk_cnt==32'd1) begin

        clk_cnt <= 1'b0; clk_400Hz <= ~clk_400Hz;

    end

    else clk_cnt <= clk_cnt + 1'b1;


//位控制

reg [3:0]wei_ctrl=4'b1110;

always @(posedge clk)

    wei_ctrl <= {wei_ctrl[2:0],wei_ctrl[3]}; //位控制


reg [3:0]duan_ctrl;

always @(wei_ctrl or data)

    case(wei_ctrl)

        4'b1110:duan_ctrl=data[3:0];

        4'b1101:duan_ctrl=data[7:4];

        4'b1011:duan_ctrl=data[11:8];

        4'b0111:duan_ctrl=data[15:12];
```

```
        default:duan_ctrl=4'hf;

    endcase


//解码模块
reg [6:0]duan;
always @(duan_ctrl)
    case(duan_ctrl)
        4'h0:duan=7'b100_0000;//0
        4'h1:duan=7'b111_1001;//1
        4'h2:duan=7'b010_0100;//2
        4'h3:duan=7'b011_0000;//3
        4'h4:duan=7'b001_1001;//4
        4'h5:duan=7'b001_0010;//5
        4'h6:duan=7'b000_0010;//6
        4'h7:duan=7'b111_1000;//7
        4'h8:duan=7'b000_0000;//8
        4'h9:duan=7'b001_0000;//9
        4'ha:duan=7'b000_1000;//a
        4'hb:duan=7'b000_0011;//b
        4'hc:duan=7'b100_0110;//c
        4'hd:duan=7'b010_0001;//d
        4'he:duan=7'b000_0111;//e
        4'hf:duan=7'b000_1110;//f
        // 4'hf:duan=7'b111_1111;//不显示
        default : duan = 7'b100_0000;//0
endcase
```

```
    assign sm_wei = wei_ctrl;

    assign sm_duan = duan;

endmodule
```

[Mux]

```
module Mux #(parameter width = 8) (input[width-1:0] d0, d1, input s, output [width-1:0] y);

    assign y = s ? d1 : d0;

endmodule
```

[Registers]

```
module
Registers(clk,regWrite,readRegister1,readRegister2,writeRegister,writeData,readData1,readDat
a2,jal,pc,ra);

    input clk,regWrite,jal;

    input [4:0] readRegister1;

    input [4:0] readRegister2;

    input [4:0] writeRegister;

    input [31:0] writeData,pc;

    output [31:0] readData1;

    output [31:0] readData2;

    output [31:0] ra;

    reg [31:0] regFiles[0:31];

    integer i;

    initial begin

        for(i=0;i<32;i=i+1)

            regFiles[i]<=0;
```

```
    end

    assign readData1 = regFiles[readRegister1];

    assign readData2 = regFiles[readRegister2];


    always @(posedge clk) begin

        if(jal==1)

            regFiles[31]=pc;

        else if(regWrite)

        regFiles[writeRegister] = writeData;

    end


    assign ra=regFiles[31];

endmodule


[SignExtend]

module SignExtend(in,data);

    input [15:0] in;

    output reg [31:0] data;

    always@(in) begin

        if(in[15]==0)

            data = {16'h0000,in};

        else

            data = {16'hffff,in};

        end

endmodule
```

[ALUcontrol]

```
module  ALUcontrol(aluOp,funct,aluCtr);
    input [3:0] aluOp;
    input [5:0] funct;
    output reg [3:0] aluCtr;
    always @(aluOp or funct)
        casex({aluOp, funct})
            10'b0000xxxxxx: aluCtr = 4'b0010; // lw, sw, addi, addiu(addi)
            10'b0001xxxxxx: aluCtr = 4'b0110; // beq
            10'b1111100000: aluCtr = 4'b0010; // add
            10'b1111111101: aluCtr = 4'b0010; // addu(add)
            10'b1111100010: aluCtr = 4'b0110; // sub
            10'b1111xx0100: aluCtr = 4'b0000; // and
            10'b1111100101: aluCtr = 4'b0001; // or
            10'b1111000000: aluCtr = 4'b0011; // sll
            10'b1111101010: aluCtr = 4'b0111; // slt
            10'b0010xxxxxx: aluCtr = 4'b0001; // ori
            10'b0011xxxxxx: aluCtr = 4'b0111; // slti
            10'b0110xxxxxx: aluCtr = 4'b1110; // bne
            10'b1000xxxxxx: aluCtr = 4'b1000; // j
            default:aluCtr = 4'b0010;
        endcase
endmodule
```

【完成情况】

是否完成以下步骤？（√完成 ×未完成）

1［√］2[√] 3[√] 4[√] 5[√]

【实验体会】

实验报告

写出实验过程中遇到的问题，解决方法和自己的思考;并简述实验体会（如果有的话）。

本次实验收获良多。在 PC 模块中了解了寄存器变量在最底层模块设置更为方便。在 Control 模块设置中纠正了以为 jr 是 J 型指令的误区。在 Registers 模块中知道了 jal 和 jr 的数据通路设计。在仿真中，学会了使用 add to wave window 功能将模块中的变量添加到仿真图像中。在生成二进制码时遇到了 Multiple Driver Nets 的报错，了解了不能在多个 always 块中对同一个变量进行赋值。

【交实验报告】

每位同学单独完成本实验内容并填写实验报告。

交作业地点：http://172.18.187.251/netdisk/default.aspx?vm=2lorg

FPGA 实验/06、单周期 CPU 设计(3)

截止日期：2022 年 12 月 1 日 23:00（周四）

上传文件：学号_姓名_单周期 CPU 设计 3.doc

学号_姓名_单周期 CPU 设计 3.rar （包含源代码的工程和 MP4 文件）