**同济大学**

**计算机科学与技术系**

**计算机组成原理实验报告**

****

**学 号 1551445**

**姓 名 童佳燕**

**专 业 计算机科学与技术**

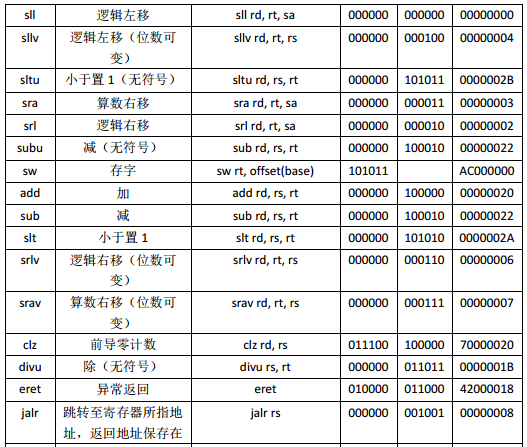
**授课老师 郝泳涛**

**日 期 2018年6月4日**

1. 实验内容：54条指令CPU设计

在本次实验中，我们要求要求画出54条指令的CPU的通路图，并使用Verilog HDL 语言实现54条MIPS指令的CPU的设计和仿真。

需要实现的54条 MIPS 指令

1. **实验步骤**

本实验完成的是54条MIPS指令单周期CPU，实验步骤如下：

1. 阅读每条指令，对每条指令所需执行的功能与过程都有充分的了解
2. 确定每条指令在执行过程中所用到的部件以及数据的流向
3. 使用表格列出指令所用部件，在表格中填入每个部件的数据输入来源

表格如下图

表3.1.指令ADD-JR相关部件数据来源表

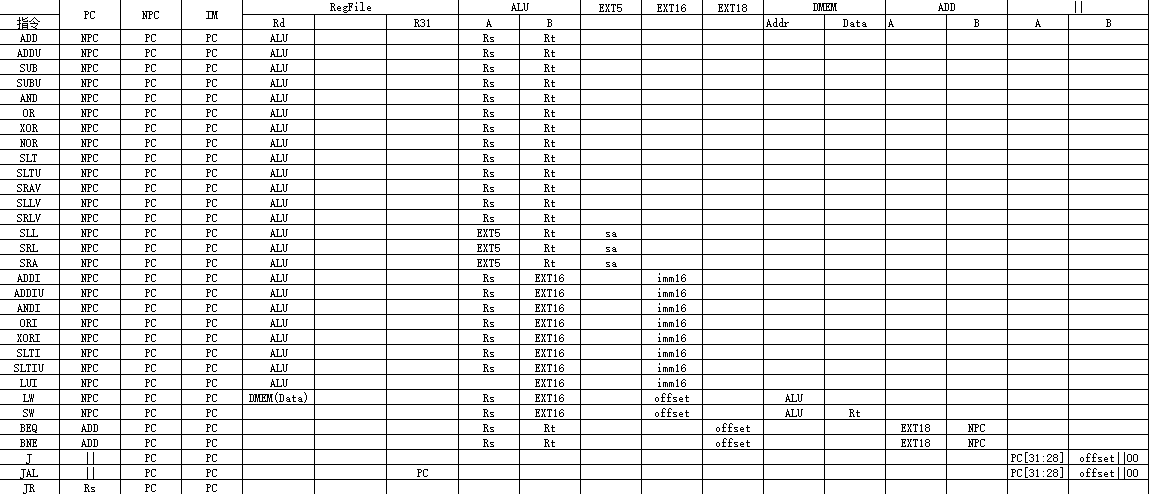
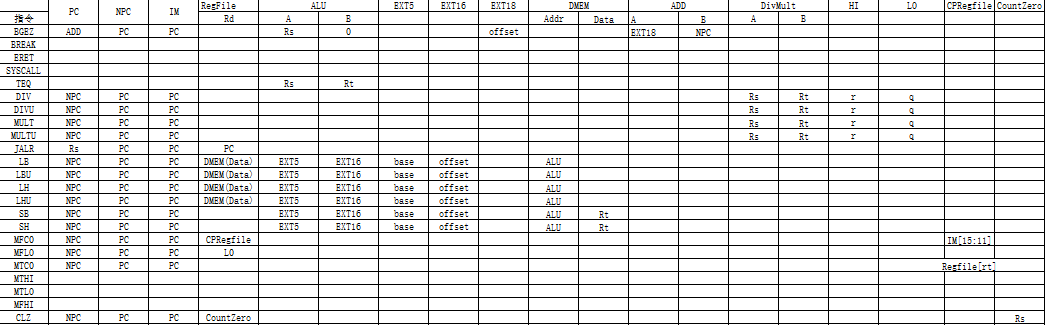
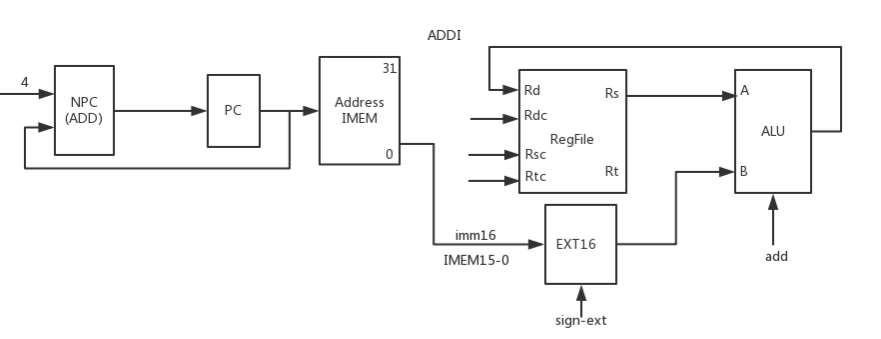


表3.2.指令ADD-JR相关部件数据来源表

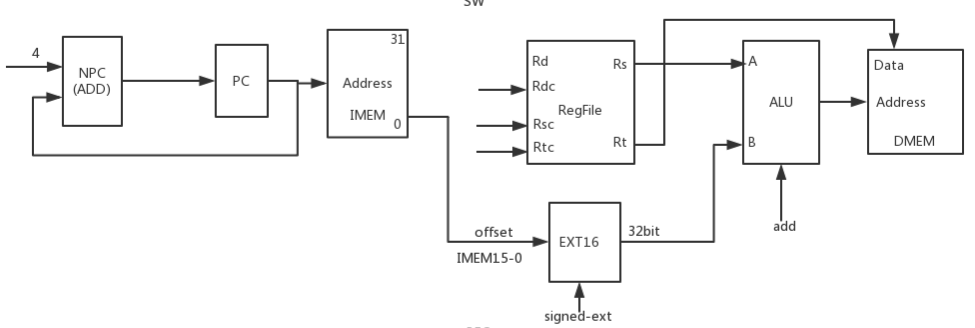


1. 根据表格所涉及部件和部件的数据输入来源，绘制每个指令的数据流图，并在绘制图的过程中记录所需控制信号。由于54条指令设计大量指令数据流图，报告中仅呈现部分代表性指令，详细指令数据流图见附件1.1,1.2,1.3.

图4.1 ADDI指令数据流图

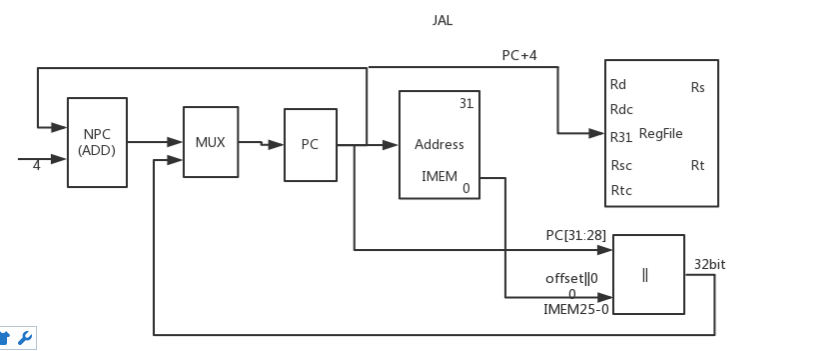
说明：NPC部件执行PC+4操作，准备下一条指令地址；PC寄存器在时钟上升沿到来时作为addr读取IMEM中的32位指令；根据指令中对应位的数据，在时钟下降沿到来时取出Rs，与16位imm扩展后的数据一起送入ALU部件执行加法；最后将结果送到Regfile的rd中。

图4.2 SW指令数据流图



说明：NPC部件执行PC+4操作，准备下一条指令地址；PC寄存器在时钟上升沿到来时作为addr读取IMEM中的32位指令；根据指令中对应位的数据，在时钟下降沿到来时取出Rs，与16位imm扩展后的数据一起送入ALU部件执行加法，作为数据存储器写入数据的地址；DMEM部件根据ALU的结果是写入数据，完成SW的全部操作。

图4.3 JAL指令数据流图



说明：JAL指令在J的基础上将返回地址保存在Regfile的31号寄存器中。下一条指令，在之前的指令的基础上，会有两个数据来源：PC+4会让PC[31:28]||offset||00。NPC部件执行PC+4操作，准备PC+4地址，||部件执行数据连接操作，准备PC[31:28]||offset||00地址。同时PC+4作为返回地址送到Regfile的31号寄存器进行存储

表4.1 54条单周期CPU控制信号表

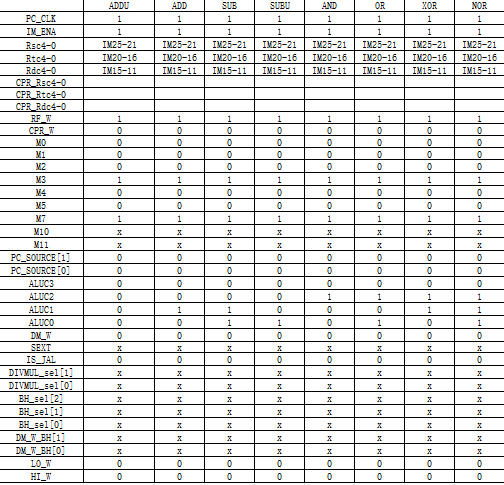


表4.2 54条单周期CPU控制信号表

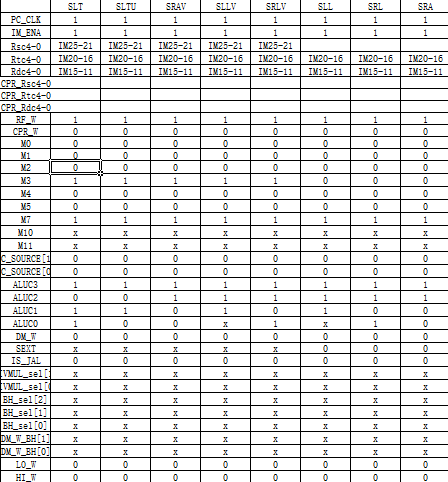


表4.3 54条单周期CPU控制信号表

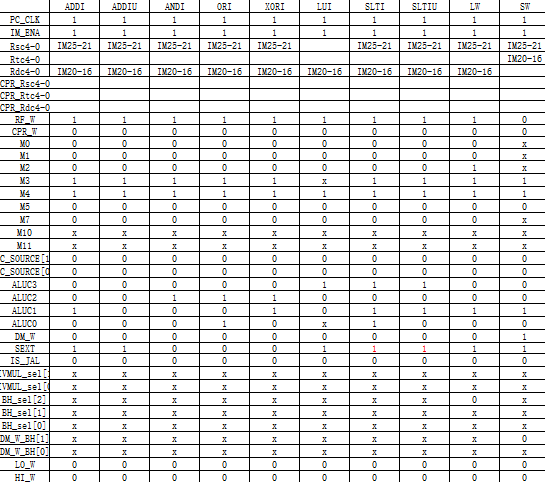


表4.4 54条单周期CPU控制信号表

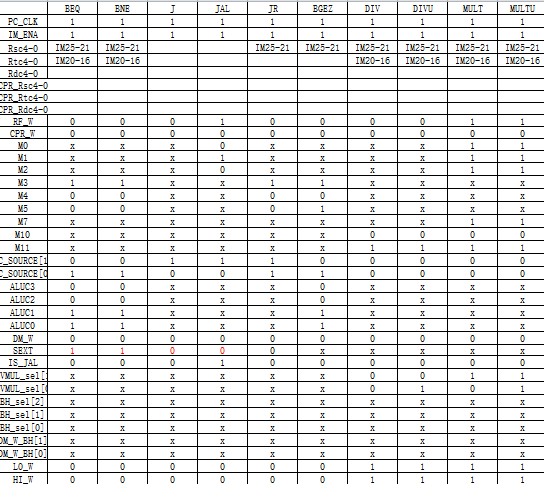


表4.5 54条单周期CPU控制信号表

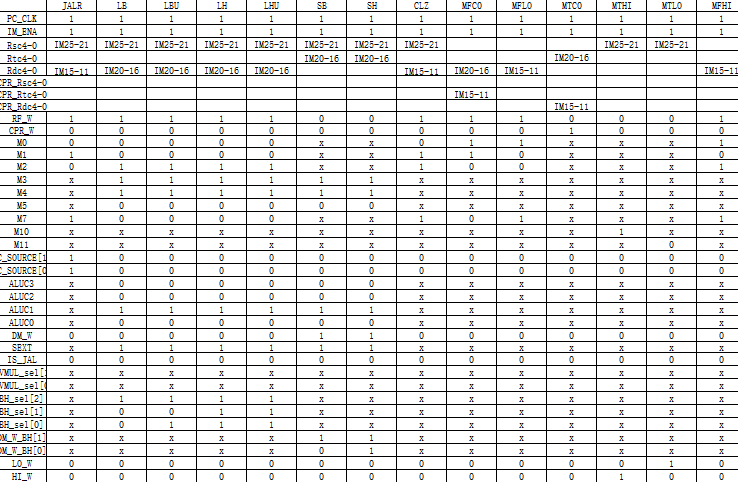


表4.6 ALU部件选择信号说明

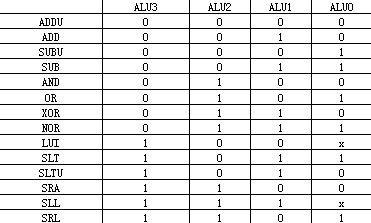


表4.7 rf\_wdata数据来源选择信号说明

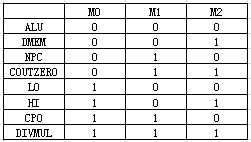


表4.8 乘除法器功能选择信号说明

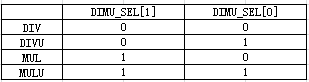
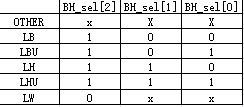


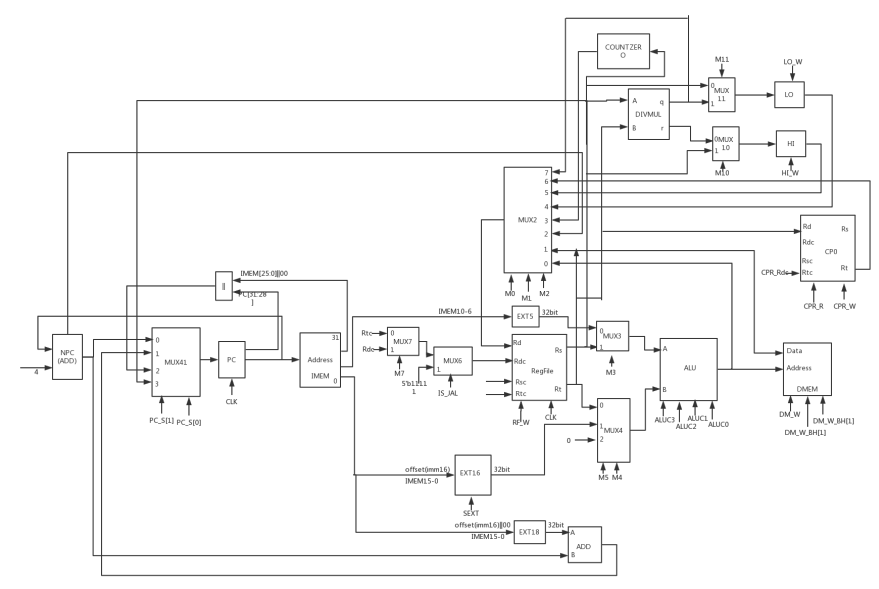
表4.9 LH/LB/LHU/LBU指令选择信号说明



阶段性总结：结合上一次31条MIPS指令单周期CPU的实验过程，这一步骤从整体过程中是至关重要，也是耗时最多的一步。完成这一步，要求我们明确每一条指令的功能以及工作流程。在这个基础上将指令的工作分配给相应的部件，同时设计需要的控制信号来正确的调控各个部件的工作。这一步完成之后，我们可以明确所有部件之间数据流的传输以及所有控制信号的工作内容，以及来源，在此基础上可以完成控制器部件的代码设计。

1. 根据表格所涉及部件和部件的数据输入来源，画出整个数据通路

图5.1数据通路图



Ps:由于在载入图片过程中进行了一定程度的压缩，因此图片清晰度受损，清晰的PDF版本请见附录5.0

三、测试过程及结果

在完成CPU主体部分之后，添加一个tb.v文件，在tb模块中实例化CPU，并使用测试指令初始化指令寄存器，同时对regfile中的数据进行输出，用于查看CPU工作的正确性。

Tb.v代码如下:

|  |
| --- |
| module sccomp\_df\_tb();  reg clk;  reg rst;  always  #1 clk<=!clk;    integer file\_output;  integer counter=0;  reg[31:0]p;    sccomp\_dataflow inst\_scc(clk, rst);  initial begin  //修改路径  file\_output=$fopen("F:\\result\_my.txt");  clk=0;  rst=1;  p=1;  #2  rst=0;  end    always@(posedge clk)begin  p<=inst\_scc.ip\_in;  end  always @(posedge clk)begin    // if(p!=inst\_scc.ip\_in)begin  if(1==1)begin  counter=counter+1;  $fdisplay(file\_output,"pc: %h",inst\_scc.ip\_in);  $fdisplay(file\_output,"instr: %h",inst\_scc.inst);    $fdisplay(file\_output,"regfile0: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[0]);  $fdisplay(file\_output,"regfile1: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[1]);  $fdisplay(file\_output,"regfile2: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[2]);  $fdisplay(file\_output,"regfile3: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[3]);  $fdisplay(file\_output,"regfile4: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[4]);  $fdisplay(file\_output,"regfile5: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[5]);  $fdisplay(file\_output,"regfile6: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[6]);  $fdisplay(file\_output,"regfile7: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[7]);  $fdisplay(file\_output,"regfile8: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[8]);  $fdisplay(file\_output,"regfile9: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[9]);  $fdisplay(file\_output,"regfile10: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[10]);  $fdisplay(file\_output,"regfile11: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[11]);  $fdisplay(file\_output,"regfile12: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[12]);  $fdisplay(file\_output,"regfile13: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[13]);  $fdisplay(file\_output,"regfile14: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[14]);  $fdisplay(file\_output,"regfile15: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[15]);  $fdisplay(file\_output,"regfile16: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[16]);  $fdisplay(file\_output,"regfile17: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[17]);  $fdisplay(file\_output,"regfile18: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[18]);  $fdisplay(file\_output,"regfile19: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[19]);  $fdisplay(file\_output,"regfile20: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[20]);  $fdisplay(file\_output,"regfile21: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[21]);  $fdisplay(file\_output,"regfile22: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[22]);  $fdisplay(file\_output,"regfile23: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[23]);  $fdisplay(file\_output,"regfile24: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[24]);  $fdisplay(file\_output,"regfile25: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[25]);  $fdisplay(file\_output,"regfile26: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[26]);  $fdisplay(file\_output,"regfile27: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[27]);  $fdisplay(file\_output,"regfile28: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[28]);  $fdisplay(file\_output,"regfile29: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[29]);  $fdisplay(file\_output,"regfile30: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[30]);  $fdisplay(file\_output,"regfile31: %h",inst\_scc.sccpu.cpu\_ref.array\_reg[31]);  end  end  endmodule |

经过modelsim仿真之后会生成result.txt文件，使用TkDiff软件将其与测试文件对应的正确结果进行比对，以验证CPU工作的正确性。

测试过程中经验总结最重要的一个方法：**打印相关接口数据，顺藤摸瓜**

【以regfile无法写入数据为例】

解决思路：

* 列出regfile相关的变量：RF\_W,wdata,wdata
* 将这些变量加入cpu的接口列表中，同时注释掉cpu模块代码主体中对这些变量的声明
* 修改对应的tb文件中对cpu的实例化
* simulation，观察modelsim中这些变量的值，是否符合预想

以上这个思路是在debug我的cpu的时候最常用的方法，可以帮助**检查到出错的端口**。在此基础上，**顺藤摸瓜，一步一步排查出错的量，直到找到问题的根源**。

**四、部分代码**

Dataflow.v

|  |
| --- |
| module sccomp\_dataflow(  input clk\_in,  input reset//高电平有效，复位  //output [31:0] inst,  //output [31:0] pc,  //output [31:0] addr  );  wire [31:0] inst;  wire [31:0] pc;  wire [31:0] addr;  wire [31:0] dram\_data\_out;  wire DM\_W;  wire [31:0] wdata;  wire [31:0] ip\_in;  assign ip\_in=pc-32'h00400000;  wire [1:0] DM\_W\_BH;  wire [2:0] BH\_sel;  cpu sccpu(clk\_in,reset,inst,dram\_data\_out,pc,DM\_W\_BH,BH\_sel,DM\_W,addr,wdata);  iram imem\_inst(ip\_in[12:2],inst);//右移2位，取低11位  dram dm(clk\_in,DM\_W,DM\_W\_BH,BH\_sel,addr-32'h10010000,wdata,dram\_data\_out);  endmodule |

Cpu.v

|  |
| --- |
| module cpu(  input clk\_in,  input reset,//高电平有效  input [31:0] instruction,  input [31:0] dram\_data\_out,  output [31:0] pc,  output [1:0] DM\_W\_BH,  output [2:0] BH\_sel,  output DM\_W,  output [31:0] addr,//数据存储器待存取的地址  output [31:0] wdata  );  wire RF\_W;//寄存器堆的写信号，下降沿+高电平有效  //wire CPR\_R  wire CPR\_W;//特殊寄存器的写信号，下降沿+高电平有效  wire M0;  wire M1;  wire M2;//以上三个主要负责写入寄存器堆的数据的选择  wire M3;  wire M4;  wire M5;//以上三个主要负责传入ALU部件A,B接口的数据选择  wire M7;//选择Rdc  wire M10;//HI寄存器的数据来源选择  wire M11;//LO寄存器的数据来源选择  wire [1:0] PC\_SOURCE;//PC数据来源的选择  wire [3:0] ALUC;  //wire DM\_R,  //wire DM\_W;//数据寄存器的写信号，高电平有效  wire SEXT;//传给扩展部件，指示符号扩展（高电平）还是0扩展（低电平）  wire IS\_JAL;  //wire DIV\_MUL\_RST;//高电平有效。不执行乘除运算时，对DIVMUL部件的寄存器置零；  //wire DIV\_MUL\_START;//START=!RST & ~busy  wire [1:0] DIVMUL\_sel;//乘除法器部件中，有/无符号乘/除法的选择  //wire [2:0] BH\_sel;//LB,LH,LBU,LHU，LW的选择信号  //wire [1:0] DM\_W\_BH;//SW,SH,SB的处理，传入DMEM  wire LO\_W;//写入LO寄存器的写信号，高电平有效，时钟下降沿写入  wire HI\_W;//写入HI寄存器的写信号，高电平有效，时钟下降沿写入  wire MFC0;  wire MTC0;  //接口，信号定义///////////////////////////////////  //拆解指令instruction  wire [5:0] op;  wire [5:0] func;  wire [4:0] rsc;//Rsc  wire [4:0] rtc;//Rtc  wire [4:0] rdc;  wire [4:0] r31=5'b11111;  wire [31:0] rs;  wire [31:0] rt;  wire [4:0] shamt;//5位的offset  wire [15:0] immediate;//16位立即数  wire [17:0]immediate\_tmp;//immediate||00  wire [25:0] j\_addr;//J,Jal指令中的address  //PC四路选择器数据来源  wire [31:0]next\_pc;  wire [31:0]pc4;//PC+4  wire [31:0]pc4offset;//pc+4+offset||00  wire [31:0]pc4offset\_tmp;  wire [31:0]pc4addr00;//(pc+4)[31:28]||address||00  wire [31:0]pc4addr00\_tmp;  wire [31:0]pcrs;  wire [31:0] rf\_wdata;//写入dataram的数据  //ALU数据输入及运算标志  wire [31:0] alu\_a;  wire [31:0] alu\_b;  wire [31:0] alu\_r;  wire zero; //判断运算结果是否为0  wire carry; //判断是否进位  wire negative;//判断结果是否为负  wire overflow;//判断结果是否溢出  wire [31:0] zero\_null;//替死鬼  wire [31:0] zero\_num; //前导0的个数  wire [31:0] LO\_out;//LO寄存器读取的数据  wire [31:0] LO\_in;  wire [31:0] HI\_in;  wire [31:0] HI\_out;//HI寄存器读取的数据  wire [31:0] cpr\_out;//从CP0特殊寄存器堆读取的数据  wire EXCEPTION;  wire ERET;  wire [4:0] CAUSE;  wire [31:0] status;  wire [31:0] exc\_addr;    wire [31:0] q;  wire [31:0] r;    // wire [31:0] dram\_data\_out;  wire [31:0] res; //从乘除法器出来，结果result的低32位写入regfile  //wire [31:0] addr;  ///////////////////////////////////////////////////  //实例化控制器  control\_unit\_54 cu54(op,rsc,rs,rt,func,zero,negative,RF\_W,CPR\_W,M0,M1,M2,M3,M4,M5,M7,M10,M11,PC\_SOURCE,ALUC,DM\_W,SEXT,IS\_JAL,DIVMUL\_sel,BH\_sel,DM\_W\_BH,LO\_W,HI\_W,MFC0,MTC0,EXCEPTION,ERET,CAUSE);  //由于写入地址有rtc和rdc两种，所以要先经过mux7  wire [4:0] rdc\_tmp0;  wire [4:0] rdc\_tmp1;  assign op[5:0] = instruction[31:26];  assign func[5:0] = instruction[5:0];  assign rsc = instruction[25:21];  assign rtc = instruction[20:16];  assign rdc\_tmp1 = instruction[15:11];  assign shamt = instruction[10:6];  assign immediate = instruction[15:0];  assign j\_addr = instruction[25:0];  assign immediate\_tmp ={immediate,{2'b00}} ;//在计算pc+4+offset||00时需要用    //时钟上升沿控制PC的更新，将next\_pc传给PC，其中next\_pc已经经过数据来源选择器  add2 npc(pc,32'h0000\_0004,pc4);  ext18 extoffset(immediate\_tmp,SEXT,pc4offset\_tmp);  add2 pc4offset\_inst(pc4,pc4offset\_tmp,pc4offset);  assign pc4addr00\_tmp = {{pc4[31:28]},j\_addr,{2'b00}};//j jal;  //assign pc4addr00 = pc4addr00\_tmp-32'h0040\_0000;  assign pc4addr00 = pc4addr00\_tmp;  //assign pcrs = rs-32'h0040\_0000;  assign pcrs = rs;  mux32x41 mux41(pc4,pc4offset,pc4addr00,pcrs,PC\_SOURCE,next\_pc);  pcreg pcreg\_inst(clk\_in,reset,next\_pc,pc);  //实例化指令寄存器  //wire im\_ena=1'b1;  //instmem instmen\_inst(im\_ena,pc[9:0],instruction);  //iram iram\_inst(pc,in)  //选择Rdc  mux5x21 mux7(rtc,rdc\_tmp1,M7,rdc\_tmp0);  mux5x21 mux8(rdc\_tmp0,r31,IS\_JAL,rdc);  //选择rf\_wdata    wire [31:0] pc4\_jal;  //assign pc4\_jal = pc4+32'h0040\_0000;  assign pc4\_jal=pc4;  //从这七个数据来源中，根据选择信号M0-M2进行选择，得到rf\_wdata  mux32x81 mux2(alu\_r,dram\_data\_out,pc4\_jal,zero\_num,LO\_out,HI\_out,cpr\_out,res,M0,M1,M2,rf\_wdata);  //实例化regfiles  regfile cpu\_ref(clk\_in,reset,RF\_W,rsc,rtc,rdc,rf\_wdata,rs,rt);  //实例化ALU部件    wire [31:0] tmp\_a;  wire [31:0] tmp\_b;  ext5 extshamt(shamt,SEXT,tmp\_a);  mux32x21 mux3(tmp\_a,rs,M3,alu\_a);  ext16 extimm(immediate,SEXT,tmp\_b);  wire [31:0] zero\_b\_in;  assign zero\_b\_in = 32'h0;  assign zero\_null = 32'h0;  mux32x41 mux4(rt,tmp\_b,zero\_b\_in,zero\_null,{M5,M4},alu\_b);  alu alu\_inst(alu\_a,alu\_b,ALUC,alu\_r,zero,carry,negative,overflow);  //实例化countZero部件  countZero cz(rs,zero\_num);  //实例化乘除法器  divmul divm(rs,rt,DIVMUL\_sel,q,r,res);  //实例化LO部件  mux32x21 mux11(rs,q,M11,LO\_in);  HILOreg LO(LO\_in,LO\_W,clk\_in,LO\_out);  //实例化HI部件  mux32x21 mux10(r,rs,M10,HI\_in);  HILOreg HI(HI\_in,HI\_W,clk\_in,HI\_out);  //实例化CP0，即特殊寄存器堆  CP0 cp0(clk\_in,reset,CPR\_W,MFC0,MTC0,pc,rdc\_tmp1,rt,EXCEPTION,ERET,CAUSE,cpr\_out,status,exc\_addr);    //实例化dmem,之后拿出CPU  //wire [31:0] addr;  //assign addr=alu\_r-32'h10010000;  assign addr=alu\_r;  assign wdata=rt;  //dram dm(clk\_in,reset,DM\_W,DM\_W\_BH,BH\_sel,addr,wdata,dram\_data\_out);  //dram dm(clk\_in,DM\_W,DM\_W\_BH,BH\_sel,alu\_r[9:0],rt,dram\_data\_out);  //dram dm(clk\_in,DM\_W,alu\_r[9:0],rt,dram\_data\_out);  endmodule |

Control\_unit.v

|  |
| --- |
| module control\_unit\_54(  input [5:0] op,  input [4:0] rsc,  input [31:0] rs,  input [31:0] rt,  input [5:0] func,  //input busy,//由乘除法器传出的信号  input z,//ALU的zero属性  input n,//ALU的negetive属性  //output RF\_R,/  output RF\_W,//寄存器堆的写信号，下降沿+高电平有效  //output CPR\_R  output CPR\_W,//特殊寄存器的写信号，下降沿+高电平有效  output M0,  output M1,  output M2,//以上三个主要负责写入寄存器堆的数据的选择  output M3,  output M4,  output M5,//以上三个主要负责传入ALU部件A,B接口的数据选择  output M7,//选择Rdc  output M10,//HI寄存器的数据来源选择  output M11,//LO寄存器的数据来源选择  output [1:0] PC\_SOURCE,//PC数据来源的选择  output [3:0] ALUC,  //output DM\_R,  output DM\_W,//数据寄存器的写信号，高电平有效  output SEXT,//传给扩展部件，指示符号扩展（高电平）还是0扩展（低电平）  output IS\_JAL,  //output DIV\_MUL\_RST,//高电平有效。不执行乘除运算时，对DIVMUL部件的寄存器置零；  //output DIV\_MUL\_START,//START=!RST & ~busy  output [1:0] DIVMUL\_sel,//乘除法器部件中，有/无符号乘/除法的选择  output [2:0] BH\_sel,//LB,LH,LBU,LHU，LW的选择信号  output [1:0] DM\_W\_BH,//SW,SH,SB的处理，传入DMEM  output LO\_W,//写入LO寄存器的写信号，高电平有效，时钟下降沿写入  output HI\_W, //写入HI寄存器的写信号，高电平有效，时钟下降沿写入  output MFC0,  output MTC0,  output EXCEPTION,  output ERET,  output [4:0] CAUSE  );  wire ADD,ADDU,SUB,SUBU,AND,OR,XOR,NOR,SLT,SLTU;  wire SLL,SRL,SRA,SLLV,SRLV,SRAV,JR,ADDI,ADDIU,ANDI,ORI,XORI;  wire LW,SW,BEQ,BNE,SLTI,SLTIU,LUI,J,JAL,CLZ,DIVU;  wire JALR,LB,LBU,LHU,SB,SH,LH,MFHI,MFLO;  wire MTHI,MTLO,MUL,MULTU,SYSCALL,TEQ,BGEZ,BREAK,DIV;  instruction\_decoder inst\_decode(op,rsc,func,ADD,ADDU,SUB,SUBU,AND, OR,XOR,  NOR,SLT,SLTU,SLL,SRL,SRA,SLLV,SRLV,SRAV,JR,ADDI,ADDIU,ANDI,ORI,XORI,  LW,SW,BEQ,BNE,SLTI,SLTIU,LUI,J,JAL,CLZ,DIVU,ERET,JALR,LB,LBU,LHU,  SB,SH,LH,MFC0,MFHI,MFLO,MTC0,MTHI,MTLO,//字母o  MUL,MULTU,SYSCALL,TEQ, BGEZ,BREAK,DIV);    assign EXCEPTION = BREAK | TEQ &(rs==rt) | SYSCALL;  assign CAUSE[4]=1'b0;  assign CAUSE[3]=BREAK||SYSCALL||(TEQ && rt==rs);  assign CAUSE[2]=TEQ;  assign CAUSE[1]=1'b0;  assign CAUSE[0]=(TEQ && rt==rs)||BREAK;  assign HI\_W = DIV | DIVU | MUL | MULTU | MTHI;  assign LO\_W = DIV | DIVU | MUL | MULTU | MTLO;  assign DM\_W\_BH[0] = SH;  assign DM\_W\_BH[1] = SH | SB;  assign BH\_sel[0] = LBU | LHU;  assign BH\_sel[1] = LH | LHU;  assign BH\_sel[2] = LB | LBU | LH | LHU;  assign DIVMUL\_sel[0] = MULTU | DIVU;  assign DIVMUL\_sel[1] = MULTU | MUL;  //assign DIV\_MUL\_RST = ~MUL & ~MULTU & ~DIV & ~DIVU;  //assign DIV\_MUL\_START = ~DIV\_MUL\_RST & ~busy;  assign IS\_JAL = JAL ;  assign SEXT = ADDI | ADDIU | LUI | SLTI | LW | SW | BEQ | BNE | LB | LBU | LHU | LH | SB | SH;  assign DM\_W = SW | SB | SH;  assign ALUC[0] = SUB | SUBU | OR | NOR | SLT | SRLV | SRL | ORI | SLTI | BEQ | BNE | BGEZ;  assign ALUC[1] = ADD | SUB | XOR | NOR | SLT | SLTU | SLLV | SLL | ADDI | XORI | SLTI | SLTIU | LW | SW | BEQ | BNE | BGEZ | LB | LH | LBU | LHU | SB | SH ;  assign ALUC[2] = AND | OR | XOR | NOR | SRAV | SLLV | SRLV | SLL | SRA | SRL | ANDI | ORI | XORI;  assign ALUC[3] = SRAV | SLLV | SRLV | SLL | SRA | SRL | SLT | SLTU | LUI | SLTI | SLTIU;  assign PC\_SOURCE[0] = BEQ & z | BNE & ~z | JR | BGEZ & ~n | JALR;  assign PC\_SOURCE[1] = J | JAL | JR | JALR;  assign M11 = DIV | DIVU | MUL | MULTU;  assign M10 = MTHI;  assign M7 = ADDU | ADD | SUB | SUBU | AND | OR | XOR | NOR | SLT | SLTU | SRAV | SLLV | SRLV | SLL | SRA | SRL | JALR | CLZ | MFLO | MFHI | MUL | MULTU;  assign M5 = BGEZ;  assign M4 = ADDI | ADDIU | ANDI | ADDI | ADDIU | ORI | XORI | LUI | SLTI | SLTIU | LW | SW | LB | LH | LBU | LHU | SB | SH ;  assign M3 = ADDU | ADD | SUB | SUBU | AND | OR | XOR | NOR | SLT | SLTU | SRAV | SLLV | SRLV | ADDI | ADDIU | ANDI | ORI | XORI | SLTI | SLTIU | SW | LW | BEQ | BNE | JR | BGEZ | LH | LHU | LB | LBU | SB | SH ;  assign M2 = LW | LB | LBU | LH | LHU | CLZ | MFHI | MUL | MULTU;  assign M1 = JAL | JALR | CLZ | MFC0 | MUL | MULTU;  assign M0 = MFC0 | MFLO | MFHI | MUL | MULTU;  assign CPR\_W = MTC0 | EXCEPTION; ////////////////////////////////////  assign RF\_W = ADDU | ADD | SUB | SUBU | AND | OR | XOR | NOR | SLT | SLTU | SRAV | SLLV | SRLV | SLL | SRL | SRA |ADDI | ADDIU | ANDI | ORI | XORI | LUI | SLTI | SLTIU | LW | JAL | JALR | LB | LBU | LH | LHU | CLZ | MFC0 | MFLO | MFHI | MUL | MULTU ;  endmodule |

Dmem.v

|  |
| --- |
| module dram(  input clk\_in,//下降沿触发写操作，当wena有效时，写  input wena,//高电平有效  input [1:0] DM\_W\_BH,//写入一个字节，或2个字节的选择信号  input [2:0] BH\_sel,//读取数据时，选择读出B,H，BU,HU或者W  input [31:0] addr,  input [31:0] data\_in,  output [31:0] data\_out  );  reg [31:0]dataram[0:1023];  reg [31:0] data\_out\_tmp;  reg [7:0] tmp\_8;  reg [15:0] tmp\_16;  assign data\_out=data\_out\_tmp;  always @(\*)  begin  if (BH\_sel[2]==0)//LW  begin  data\_out\_tmp=dataram[addr];  end    else if (BH\_sel[1]==0)  begin  tmp\_8=dataram[addr][7:0];  if (BH\_sel[0]==1'b0)//LB  begin  data\_out\_tmp={{24{tmp\_8[7]}},tmp\_8};  end  else //LBU  begin  data\_out\_tmp={24'b0,tmp\_8};  end  end  else  begin  tmp\_16=dataram[addr][15:0];  if(BH\_sel[0]==1'b0)//LH  begin  data\_out\_tmp={{16{tmp\_16[15]}},tmp\_16};  end  else  begin  data\_out\_tmp={16'b0,tmp\_16};  end  end  end  always @(negedge clk\_in)  begin  if (wena==1'b1)  begin  // write  if (DM\_W\_BH[1]==1'b0)  dataram[addr]=data\_in;  else if(DM\_W\_BH[0]==1'b0)//SB,存data\_in的低8位，到dataram[addr]  dataram[addr][7:0]=data\_in[7:0];  else//SH,存data\_in的低16位,到dataram[addr]  dataram[addr][15:0]=data\_in[15:0];  end  end  endmodule |

Regfile.v

|  |
| --- |
| module regfile(  input clk\_in,//下降沿写入数据  input rst,//清零信号，高电平有效，所有寄存器清零，异步  input we,//写信号，高电平有效  input [4:0] raddr1,//读数据的地址1，Rs  input [4:0] raddr2,//读数据的地址2，Rt  input [4:0] waddr,//写数据的地址，Rd  input [31:0] wdata,//写入的数据  output [31:0] rdata1,//读取的数据1  output [31:0] rdata2//读取的数据2  );  reg [31:0] array\_reg [0:32];  assign rdata1 = array\_reg[raddr1];  assign rdata2 = array\_reg[raddr2];  always @(negedge clk\_in or posedge rst)  begin  if (rst==1'b1)  begin  // reset  array\_reg[0]=32'h0;  array\_reg[1]=32'h0;  array\_reg[2]=32'h0;  array\_reg[3]=32'h0;  array\_reg[4]=32'h0;  array\_reg[5]=32'h0;  array\_reg[6]=32'h0;  array\_reg[7]=32'h0;  array\_reg[8]=32'h0;  array\_reg[9]=32'h0;  array\_reg[10]=32'h0;  array\_reg[11]=32'h0;  array\_reg[12]=32'h0;  array\_reg[13]=32'h0;  array\_reg[14]=32'h0;  array\_reg[15]=32'h0;  array\_reg[16]=32'h0;  array\_reg[17]=32'h0;  array\_reg[18]=32'h0;  array\_reg[19]=32'h0;  array\_reg[20]=32'h0;  array\_reg[21]=32'h0;  array\_reg[22]=32'h0;  array\_reg[23]=32'h0;  array\_reg[24]=32'h0;  array\_reg[25]=32'h0;  array\_reg[26]=32'h0;  array\_reg[27]=32'h0;  array\_reg[28]=32'h0;  array\_reg[29]=32'h0;  array\_reg[30]=32'h0;  array\_reg[31]=32'h0;  array\_reg[32]=32'h0;  end  else if(we==1'b1) //写信号有效  begin  if(waddr==5'b00000)  array\_reg[waddr]=32'h0;  else  array\_reg[waddr]=wdata;  end  end  endmodule |

**五、实验总结**

在实验过程中，我对MIPS体系结构下的单周期CPU的工作流程有了深刻的认识，了解CPU内部各个部件之间的关系以及工作时数据的流动。同时在实验过程中大大提高了对verilog语言的掌握程度，对vivado ,modelsim等软件的使用，更加重要的是，在知识架构中填补了非常重要的底层硬件相关的空白，为将来的学习铺设重要的基础。