计算机组成原理

期末大作业报告

学　　号_____20074221_____

姓　　名_____游佳慧_____

指导教师_____魏坚华_____

提交日期_____2022.6.10_____

成绩评价表

报告内容	报告结构	报告最终成绩
□丰富正确 □基本正确 □有一些问题 □问题很大	□完全符合要求 □基本符合要求 □有比较多的缺陷 □完全不符合要求	
报告与 Project 功能一致性	报告图表	总体评价
□完全一致 □基本一致 □基本不一致	□符合规范 □基本符合规范 □有一些错误 □完全不正确	

教师签字:_____

目录

1 总体数据通路结构设计

1.1 总体数据通路结构图

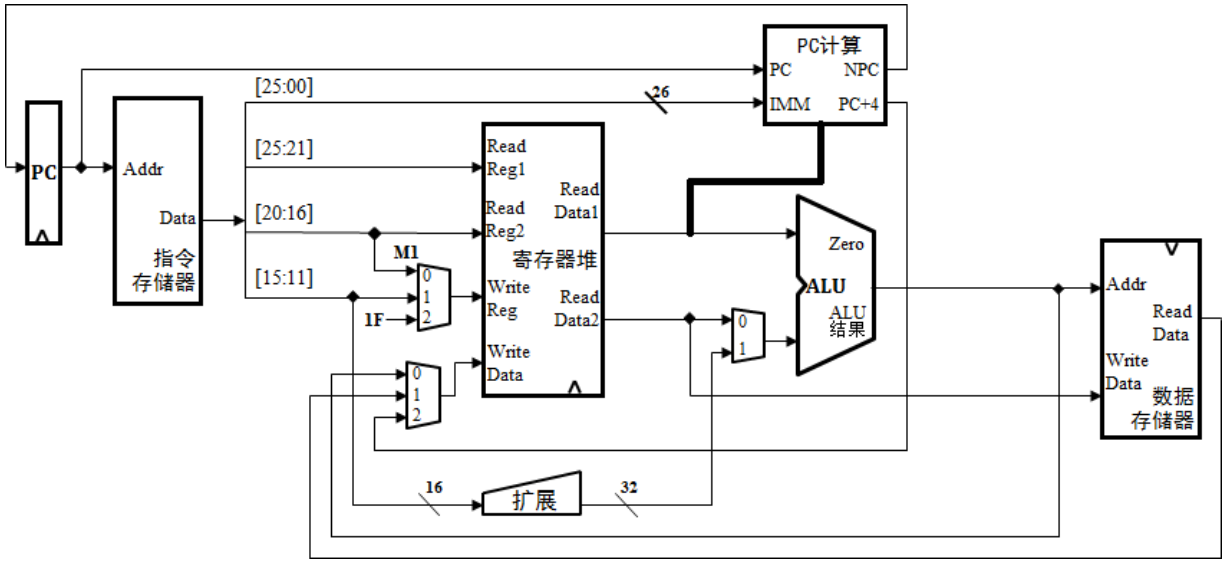

图 1：总体数据通路结构图

```
module mips(clk,rst);
      input clk,rst;
      wire [31:0] ins;
      wire [31:0] npc;
      wire [31:0] cpc;
      wire [31:0] memout;
      wire [31:0] write_data;
      wire [31:0] bushA,bushB;
      wire [31:0] extout;
      wire [31:0] alu_out;
      wire [31:0] sltout;
      wire [31:0] jalPC;
      wire [31:0] b;
      wire [1:0] MemtoReg;     //写入寄存器数据 00:ALU_out 01:DM 10:JALpc
11:SLTout
      wire [1:0] regdst;       //写寄存器选择 00:rt 01:rd 10:$31
      wire [1:0] extop;        //扩展方法 00:zero 01:sign 10:lui
```

```
    wire [1:0] aluctr;      //ALU 计算方法
    wire alusrc;            //B 端输入数据 0: busB 1: imm16
    wire MemWrite;          //DM 写使能
    wire RegWrite;          //GPR 写使能

    wire if_jr;
    wire if_beq;
    wire if_j;
    wire overflow;
    wire zero;

    wire [4:0] m1out;      //write reg
    wire [9:0] im_addr;
    wire [9:0] dm_addr;
    integer i;

    controller my_controller(ins,if_jr,if_beq,if_j,MemWrite,
                             MemtoReg,RegWrite,regdst,alusrc,aluctr,extop);
    pc my_pc(clk,rst,npc,cpc,im_addr);
    im_1k my_im_1k(im_addr,ins);
    calculate_pc my_calculate_pc(cpc,ins,if_beq,zero,
                                 if_j,npc,if_jr,jalPC,bushA);
    gpr my_gpr(clk,rst,RegWrite,overflow,ins,m1out,write_data,bushA,bushB);
    ext my_ext(extop,ins,extout);
    dm_1k my_dm_1k(dm_addr,bushB,MemWrite,clk,memout);
    ALU my_ALU(bushA,b,aluctr,alu_out,zero,overflow,sltout,dm_addr);
    mux1 my_mux1(regdst,ins,m1out);
    mux2 my_mux2(MemtoReg,write_data,alu_out,memout,jalPC,sltout);
    mux3 my_mux3(alusrc,bushB,extout,b);
endmodule
```

图 2：顶层设计

2 模块定义

2.1 GPR 模块定义

2.1.1 模块设计

```
module gpr(clk,reset,RegWrite,overflow,ins,write_reg,write_data,bushA,bushB);
    input clk;
    input reset;
    input RegWrite;
    input overflow;
    input[4:0] write_reg;       //write address
    input [31:0] write_data;
    input [31:0] ins;          //32-bit instruct
    output [31:0] bushA,bushB;  //read data
    reg [31:0] register[31:0];  //32 32-bit register
    integer i;
    assign bushA = register[ins[25:21]];
    assign bushB = register[ins[20:16]];
    always@(posedge clk or posedge reset or overflow)
    begin
      if(reset)  begin
            for(i=0;i<32;i=i+1)
              register[i]<=32'd0; end
      else begin
          if(register[30]) register[30] <= 0;
          if(RegWrite)
            begin
                if(overflow)
           //if overflow, the destination register is not modified
                    register[30]<=(register[30]|32'h0000_0001);
                else if(write_reg!=5'd0)
             //if $0, the destination register is not modified
                    register[write_reg]<=write_data;
            end
        end
    end
endmodule
```

2.1.2 基本描述

GPR 是由 32 个 32 位寄存器组成的寄存器组模块，包含修改寄存器内容、读取寄存器内容功能，寄存器的读输出总是对应于读寄存器号，不需要其他控制信号；写寄存器必须明确写使能控制信号。

2.1.3 模块接口

信号名	方向	描述
RegWrite	I	寄存器写使能。 1：当前指令写入寄存器 0：当前指令不写入寄存器
reset	I	复位信号。 1：复位 0：无效
clk	I	时钟信号。
ins[31:0]	I	传入的 32 位指令。
overflow	I	溢出标志。 1：溢出 0：未溢出
write_reg[4:0]	I	当前指令需要写入的寄存器地址。
write_data[31:0]	I	当前指令需要写入的数据。
bushA[31:0]	O	当前指令读出的数据 1。
bushB[31:0]	O	当前指令读出的数据 2。

2.1.4 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，所有寄存器数据清零。
2	取数据	根据 rs 和 rt 的地址从寄存器中取出数据。
3	写数据	若 30 号寄存器为 1，则置为 0； RegWrite 有效时，若 overflow 为 1，则将 1 写入$30， 否则根据 write_reg 的地址将数据写入该地址所对应寄存器中。

2.1.5 功能说明

最开始设计 GPR 模块时未考虑到存入溢出位后将 30 号寄存器重新清零的问题，经改进后该部分代码如下：

```
begin
  if(register[30]) register[30] <= 0;
  if(RegWrite)
    begin
      if(overflow)
        register[30]<=(register[30]|32'h0000_0001);
      else if(write_reg!=5'd0)
        register[write_reg]<=write_data;
    end
end
```

每次判断 30 号寄存器是否不为 0，若是则将其清零。而后当寄存器写使能有效时，判断溢出标志是否有效，若是则将溢出位存入 30 号寄存器，若未发生溢出且所存寄存器不是 0 号寄存器时，将数据写入对应寄存器。

从而实现每次溢出时能实时将溢出位存入 30 号寄存器，且下一条指令未溢出就将 30 号寄存器清零。

2.2 ALU 模块定义

2.2.1 模块设计

```
module ALU(a,b,alu_ctr,alu_out,zero,overflow,sltout,dm_addr);
   input [31:0] a,b;        //32-bit input data
   input [1:0] alu_ctr;     //00 add; 01 sub; 10 or; 11 addi
   output reg zero;
   output reg overflow;
   output reg [31:0] alu_out;
   output reg [31:0] sltout;
   output [9:0] dm_addr;
   assign dm_addr=alu_out[9:0];
   reg signed [31:0] signed_a,signed_b;

   always@(a or b or alu_ctr or alu_out)
   begin
      case(alu_ctr)
         2'b00:
            begin
               alu_out=a+b;
               zero=0;
               overflow=0;
               sltout=32'd0;
            end
         2'b01:
            begin
               alu_out=a-b;
               if(alu_out==0) zero=1;
               overflow=0;
               signed_a=a;
               signed_b=b;
               if(signed_a<signed_b) sltout=32'd1;
               else sltout=32'd0;
            end
         2'b10:
            begin
               alu_out=a|b;
               zero=0;
```

```
                overflow=0;
                sltout=32'd0;
            end
         2'b11:
            begin
                alu_out=a+b;
                zero=0;
                overflow=((alu_out[31] && (!a[31]) &&
(!b[31]))||((~alu_out[31]) && a[31] && b[31])) ? 1 : 0;
                sltout=32'd0;
            end
      endcase
      end
endmodule
```

2.2.2 基本描述

ALU 的主要功能是完成算术运算和逻辑运算，根据 ALU 控制信号判断 ALU 应进行的运算，产生运算结果并生成零标志信号 zero 和溢出信号 overflow。

2.2.3 模块接口

信号名	方向	描述
a[31:0]	I	参与运算的第一个输入数据。
b[31:0]	I	参与运算的第二个输入数据
alu_ctr[1:0]	I	ALU 控制信号。 00：无溢出加 01：减法运算 10：或运算 11：带溢出加
alu_out [31:0]	O	ALU 运算结果。
zero	O	运算结果是否为零的标志位。 1：运算结果为 0 0：运算结果非 0

overflow	O	溢出标志位。 1：溢出 0：未溢出
sltout[31:0]	O	slt 指令结果输出
dm_addr[9:0]	O	输出 dm 地址

2.2.4 功能定义

序号	功能名称	功能描述
1	算术运算	两数无溢出加、减、或、带溢出加
2	dm 地址	输出对应 dm 地址

2.3 EXT 模块定义

2.3.1 模块设计

```
module ext(extop,ins,extout);
    input [1:0] extop;
    input [31:0] ins;
    output reg [31:0] extout;

    always@(ins or extop)
    begin
      case(extop)
          2'b00: extout = {16'h0000,ins[15:0]};       //zero extend
          2'b01: extout = {{16{ins[15]}},ins[15:0]}; //sign extend
          2'b10: extout = {ins[15:0],16'h0000};       //lui
          default: extout=0;
      endcase
    end
endmodule
```

2.3.2 基本描述

EXT 的主要功能是完成 16 位立即数扩展，根据 EXTop 信号的不同值分别进行 0 扩展、符号位扩展或 lui 指令高位复制扩展，扩展为 32 位立即数输出。

2.3.3 模块接口

信号名	方向	描述
ins[31:0]	I	传入的 32 位指令。
extop[1:0]	I	符号扩展控制信号。 00：高位 0 扩展 01：符号位扩展 10：低 16 位补零扩展
extout [31:0]	O	完成扩展的 32 位立即数。

2.3.4 功能定义

序号	功能名称	功能描述
1	0 扩展	16 位->32 位，高 16 位补零
2	符号位扩展	16 位->32 位，最高有效位（符号位）复制填满高 16 位
3	低 16 位补零扩展	16 位->32 位，低 16 位补零

2.4 DM 模块定义

2.4.1 模块设计

```
module dm_1k(addr, din, we, clk, dout);
    input [9:0] addr ;
    input [31:0] din ;   // 32-bit input data
    input we ;          // dm write enable
    input clk ;         // clock
    output [31:0] dout ;  // 32-bit dm output
    reg[7:0] dm[1023:0] ;

    always@(posedge clk)
    begin
        if(we) {dm[addr+3],dm[addr+2],dm[addr+1],dm[addr]}<=din;
             //将数据写入对应地址
    end
    assign dout={dm[addr+3],dm[addr+2],dm[addr+1],dm[addr]};
             //从对应地址读出的数据
      //小端序 高字节数据保存在高地址存储单元
endmodule
```

2.4.2 基本描述

DM 是数据存储器，主要功能是完成存储器读写。当写入使能有效时，根据输入的地址将输入的数据写入存储器的相应位置，或输出从该地址读取的数据。

2.4.3 模块接口

信号名	方向	描述
addr[9:0]	I	需要读或写的存储器地址。
din[31:0]	I	需要写入的数据。
clk	I	时钟信号。
we	I	写入使能信号。 1：允许写入

		0：不允许写入
dout[31:0]	O	从输入地址读出的数据。

2.4.5 功能定义

序号	功能名称	功能描述
1	读数据	根据输入的寄存器地址读出数据。
2	写数据	根据输入的地址将输入数据写入存储器的相应位置。

2.5 Controller 模块定义

2.5.1 模块设计

```
module controller(ins,if_jr,if_beq,if_j,MemWrite,MemtoReg,
RegWrite,regdst,alusrc,aluctr,extop);

    input  [31:0] ins;      //32-bit instruct
    output [1:0] MemtoReg;  //写入寄存器数据 00:ALU_out 01:DM 10:JALpc
11:SLTout
    output [1:0] regdst;    //写寄存器选择 00:rt 01:rd 10:$31
    output [1:0] extop;     //扩展方法 00:zero 01:sign 10:lui
    output [1:0] aluctr;   //ALU 计算方法
    output alusrc;          //B 端输入数据 0: busB 1: imm16
    output MemWrite;        //DM 写使能
    output RegWrite;        //GPR 写使能
    output if_jr,if_beq,if_j;
    wire addu,subu,ori,lw,sw,beq,lui,j,addiu,addi,slt,jal,jr;

    //根据 opcode 和 funct 字段确定指令类型
    assign addu = (ins[31:26]==6'd0 && ins[5:0]==6'b100001)?1:0;
    assign subu = (ins[31:26]==6'd0 && ins[5:0]==6'b100011)?1:0;
    assign slt  = (ins[31:26]==6'd0 && ins[5:0]==6'b101010)?1:0;
    assign jr   = (ins[31:26]==6'd0 && ins[5:0]==6'b001000)?1:0;
    assign j    = (ins[31:26]==6'b000010)?1:0;
    assign jal  = (ins[31:26]==6'b000011)?1:0;
    assign beq  = (ins[31:26]==6'b000100)?1:0;
    assign addi = (ins[31:26]==6'b001000)?1:0;
    assign addiu= (ins[31:26]==6'b001001)?1:0;
    assign ori  = (ins[31:26]==6'b001101)?1:0;
    assign lw   = (ins[31:26]==6'b100011)?1:0;
    assign sw   = (ins[31:26]==6'b101011)?1:0;
    assign lui  = (ins[31:26]==6'b001111)?1:0;

  //设置控制信号
    assign if_jr   = jr;                                    //1 为 jr 指令
    assign if_beq  = beq;                                   //1 为 beq 指令
    assign if_j    = j||jal;                               //q 为 j 指令
    assign MemWrite= sw;                                   //1 数据存储器写使能有效
```

```
    assign MemtoReg= {(slt||jal),(lw||slt)};              //选择传入寄存器数据
    assign RegWrite= addu||subu||ori||lw||lui||addiu||addi||slt||jal; //1 寄存
器堆存储器写使能有效
    assign regdst  = {jal,(addu||subu||slt)};              //选择写入的寄存器
    assign alusrc  = ori||lw||sw||lui||addiu||addi;      //选择 ALU 第二个操作数
    assign aluctr  = {(ori||lui||addi),(subu||beq||addi||slt)};       //选择
ALU 计算类型 00 add; 01 sub; 10 or; 11 addi
    assign extop   = {lui,(lw||sw||addiu||addi)};                  //选择扩展方法
00 0 扩展；01 符号扩展；10 lui 扩展
    endmodule
```

2.5.2 基本描述

Controller 主要功能是完成对指令功能的判断和确定每条指令对应的控制信号。根据输入指令的操作码和功能码判断指令，并输出各单元的写使能信号、各多选器的选择信号和 ALU 的控制信号。

2.5.3 模块接口

信号名	方向	描述
ins[31:0]	I	传入的 32 位指令。
aluctr[1:0]	O	ALU 控制信号。 00：无溢出加 01：减法运算 10：或运算 11：带溢出加
alusrc	O	选择第二个 ALU 操作数。 0：操作数为寄存器取出的值 1：操作数为经 EXT 扩展后的 32 位立即数
extop[1:0]	O	控制 EXT 的扩展方式
if_beq	O	beq 指令标志。 1：是 beq 指令 0：非 beq 指令
if_j	O	j 指令标志。 1：是 j 指令 0：非 j 指令

if_jr	O	jr 指令标志。 1：是 jr 指令 0：非 jr 指令
MemWrite	O	DM 写使能信号。
RegWrite	O	GPR 写使能信号。
MemtoReg	O	选择写入寄存器的数据。 0：写入的数据是 ALU 计算输出结果 1：写入的数据是 DM 输出结果 2：写入的数据是 PC+4 3：写入的数据是 sltout
regdst	O	写入寄存器的目标寄存器号来源。

2.5.4 功能定义

序号	功能名称	功能描述
1	译码	将 ins[31:0]转换成对应指令。
2	产生控制信号	对输入指令的所有控制信号赋值。

2.6 PC 模块定义

2.6.1 模块设计

```
module pc(clk,reset,npc,cpc,addr);
    input clk,reset;
    input[31:0] npc;
    output reg [31:0] cpc;
    output [9:0] addr;
    assign addr = cpc[9:0];  //IM address

    always@(posedge clk or posedge reset)
    begin
      if(reset)
          cpc<=32'h0000_3000;
      else
         cpc<=npc;
    end
    endmodule
```

2.6.2 基本描述

PC 主要功能是完成储存指令地址的功能。将要执行的指令编码对应的地址储存起来，然后在下一个 clk 上升沿到来时将储存的指令编码对应的地址送出去，并继续更新指令编码对应的地址。

2.6.3 模块接口

信号名	方向	描述
clk	I	时钟信号
reset	I	复位信号
npc[31:0]	I	下一条指令编码对应的地址
cpc[31:0]	O	输出即将执行的指令编码对应的地址
addr[9:0]	O	IM 地址

2.6.4 功能定义

序号	功能名称	功能描述
1	复位	当复位信号有效时，PC 被设置为 0x0000_3000，即第一条指令编码对应的地址。
2	送出下一条指令地址	将即将执行的指令编码对应的地址传送给 IM 模块。

2.7 calculate_pc 模块定义

2.7.1 模块设计

```
module calculate_pc(cpc,ins,if_beq,zero,if_j,npc,if_jr,jalPC,bushA);
    input [31:0] ins;        //32-bit instruct
    input [31:0] cpc;        //now PC
    input [31:0] bushA;     //target address in GPR rs
    input if_beq,if_j,zero,if_jr;
    output [31:0] jalPC;
    output reg [31:0] npc;
    reg beq_jump;
    reg [2:0] choose;
    assign jalPC = cpc+4;

    always@(choose,ins,if_j,beq_jump,if_beq,zero,if_jr,bushA)
    begin
      beq_jump=if_beq && zero;
      choose={if_j,beq_jump,if_jr};
      case(choose)
          3'b000: npc = cpc+32'h4;                  //pc=pc+4
          3'b010: npc = cpc+32'h4+({{16{ins[15]}},ins[15:0]}<<2);   //beq
          3'b100: npc = {cpc[31:28],ins[25:0],2'b0};    //j jal
          3'b001: npc = bushA;
          default:npc = npc;
      endcase
    end
    endmodule
```

2.7.2 基本描述

根据输入的控制信号，计算下一条指令地址。

2.7.3 模块接口

信号名	方向	描述
ins[31:0]	I	输出 32 位 MIPS 指令

cpc[31:0]	I	PC 模块传入的地址
if_beq	I	beq 指令标志。 1：是 beq 指令 0：非 beq 指令
if_j	I	j 指令标志。 1：是 j 指令 0：非 j 指令
if_jr	I	jr 指令标志。 1：是 jr 指令 0：非 jr 指令
zero	I	ALU 计算结果为 0 标志。 1：计算结果为 0 0：计算结果非 0
bushA[31:0]	I	jr 指令的跳转地址
jalPC[31:0]	O	输出当前指令地址+4
npc[31:0]	O	输出下一条指令编码对应的地址给 PC

2.7.4 功能定义

序号	功能名称	功能描述
1	计算下一条指令地址	如果当前指令是 beq 指令，并且 zero 为 1，则 PC←PC+4+(sign_ext(ins[15:0])<<2)； 否则如果当前指令是 J 类型指令，则 PC←{cpc[31:28],ins[25:0],2'b0}； 否则如果当前指令是 Jr 指令，则 PC←bushA； 否则 PC←PC+4。

2.8 IM 模块定义

2.8.1 模块设计

```
module im_lk(addr,dout);
    input [9:0] addr;
    output [31:0] dout;
    reg [7:0] im[1023:0];
    assign dout={im[addr],im[addr+1],im[addr+2],im[addr+3]};
endmodule
```

2.8.2 基本描述

IM 的主要功能是作为指令存储器存储指令，根据输入的指令地址取出相应的指令。

2.8.3 模块接口

信号名	方向	描述
addr[9:0]	I	指令选择地址
dout[31:0]	O	输出 32 位指令

2.8.4 功能定义

序号	功能名称	功能描述
1	取指令	根据译码地址 addr 从 IM 中取出指令。

2.9 MUX1 模块定义

2.9.1 模块设计

```
module mux1(regdst,ins,m1out); //choose write register
    input [1:0] regdst;
    input [31:0] ins;
    output reg[4:0] m1out;
    always@(regdst or ins)
    begin
      case(regdst)
          2'b00: m1out = ins[20:16]; //rt
          2'b01: m1out = ins[15:11]; //rd
          2'b10: m1out = 5'd31;     //$31
          default: m1out = 5'd0;
      endcase
    end
endmodule
```

2.9.2 基本描述

多路选择器，为 gpr 选择写入的寄存器。

2.9.3 模块接口

信号名	方向	描述
regdst[1:0]	I	寄存器选择端 0：ins[20:16]；1：ins[15:11] 2：$31
ins[31:0]	I	32 位指令
m1out[40]	O	选择结果输出

2.9.4 功能定义

序号	功能名称	功能描述
1	选择器	选择将数据写入哪个寄存器

2.10 MUX2 模块定义

2.10.1 模块设计

```
module mux2(MemtoReg,write_data,alu_out,dm_out,jalPC,sltout);
//choose write data in register
    input [1:0] MemtoReg;
    input [31:0] alu_out;
    input [31:0] dm_out;
    input [31:0] jalPC;
    input [31:0] sltout;
    output reg[31:0] write_data;

    always@(MemtoReg or alu_out or dm_out or jalPC or sltout)
    begin
      case(MemtoReg)
          2'd0: write_data = alu_out;
          2'd1: write_data = dm_out;
          2'd2: write_data = jalPC;
          2'd3: write_data = sltout;
          default: write_data = 32'd0;
      endcase
    end
endmodule
```

2.10.2 基本描述

多路选择器，选择写入寄存器的数据。

2.10.3 模块接口

信号名	方向	描述
MemtoReg[1:0]	I	选择写入寄存器的数据。 0：写入的数据是 ALU 计算输出结果 1：写入的数据是 DM 输出结果 2：写入的数据是 PC+4 3：写入的数据是 sltout
alu_out [31:0]	I	alu 计算结果

dm_out [4:0]	I	dm 输出内容
jalPC[31:0]	I	PC+4
sltout[31:0]	I	slt 指令结果
write_data[31:0]	O	选择结果输出

2.10.4 功能定义

序号	功能名称	功能描述
1	选择器	选择写入寄存器的数据。

2.11 MUX3 模块定义

2.11.1 模块设计

```
module mux3(alusrc,bushB,extout,b);    //choose the second input to ALU
    input [31:0] bushB;
    input [31:0] extout;
    input alusrc;
    output reg[31:0] b;
    always@(alusrc or bushB or extout)
    begin
      case(alusrc)
          1'd0:b=bushB;
          1'd1:b=extout;
      endcase
    end
    endmodule
```

2.11.2 基本描述

多路选择器，选择 ALU 的第二个输入数据。

2.11.3 模块接口

信号名	方向	描述
alusrc[1:0]	I	寄存器选择端 0：bushB；1：extout
bushB[31:0]	I	rt 寄存器数值
extout[31:0]	I	扩展器输出
b[31:0]	O	输出数据

2.11.4 功能定义

序号	功能名称	功能描述
1	选择器	选择 ALU 的第二个输入数据。

3 设计的机器指令描述

指令操作码助记符	机器指令代码		指令功能
	opcode	funct	
addu	000000	100001	分别从 rs 和 rt 寄存器中取出两个数做无符号数加法，结果放入 rd 寄存器中。
subu	000000	100011	分别从 rs 和 rt 寄存器中取出两个数做无符号数减法，结果放入 rd 寄存器中。
ori	001101	—	从 rs 寄存器中取出一个数与高位零扩展后的 16 位立即数做或运算，结果放入 rt 寄存器。
lui	110000	—	从 rs 寄存器中取出一个数与低位补零扩展后的 16 位立即数做或运算，结果放入 rt 寄存器。
sw	101011	—	根据基地址+偏移量算出地址，将 rt 寄存器内容写入该地址对应的内存单元中。
lw	100011	—	根据基地址+偏移量算出地址，从该地址对应存储器中取出一个数存入 rt 寄存器中。
beq	000100	—	分别从 s 和 t 寄存器中取出两个数比较是否相等，若相等则进行分支跳转，PC<-PC+4+符号扩展 imm16,若不相等则不跳转，PC<-PC+4。
j	000010	—	无条件跳转，PC<-{PC+4[31:28] imm26}。
addi	001000	—	支持溢出的立即数加法，若溢出，则将$30 第 0 位置 1，否则进行正常加法操作。
addiu	001001	—	不支持溢出的立即数加法。
slt	000000	101010	如果 gpr[rs]<gpr[rt],则 gpr[rd]=1，否则为 0。
jal	000011	—	将 PC+4 存入$31 寄存器并跳转到对应地址。
jr	000000	001000	跳转到 gpr[rs]中对应地址。

4 测试程序

机器码	指令	注释说明
34100001	ori $16, $0, 1	将 16 号寄存器赋值 1
34110003	ori $17, $0, 3	将 17 号寄存器赋值 3
34080001	ori $8, $0, 1	将 8 号寄存器赋值 1
340cabab	ori $12, $0,0xabab	将 12 号寄存器赋值 0xabab
3c0d000a	lui $13, 10	将 13 号寄存器高 16 位赋值 10
00102021	start:addu $4, $0,$16	将 0 号寄存器与 16 号寄存器内容相加放入 4 号寄存器
00082821	addu $5, $0,$8	将 0 号寄存器与 8 号寄存器内容相加放入 5 号寄存器
0c000c32	jal newadd	跳转到 newadd 处，并将下条指令地址存入 31 号寄存器
00028021	addu $16, $0, $2	将 0 号寄存器与 2 号寄存器内容相加放入 16 号寄存器
02288823	subu $17,$17,$8	将 17 号寄存器内容与 8 号寄存器内容相减放入 17 号寄存器
1211fffa	beq $16, $17, start	若 16 号寄存器内容与 17 号寄存器内容相等则跳转到 start，否则向下执行
34080004	ori $8, $0,4	将 8 号寄存器赋值 4
3c017fff	addiu $24,$0,0x7fffffff	将 0 号寄存器内容与 0x7fffffff 相加放入 24 号寄存器
27090003	addiu $9,$24,3	将 24 号寄存器内容+3 放入 9 号寄存器（无溢出加法）
270a0005	addiu $10,$24,5	将 24 号寄存器内容+5 放入 10 号寄存器（无溢出加法）
00000021	addu $0,$0,$0	将 0 号寄存器与 0 号寄存器内容相加放入 0 号寄存器
	#addi $22,$24,6	将 24 号寄存器内容+6，若无溢出，则将结果赋给 22 号寄存器，否则将 30 号寄存器第 0 位置 1
ad090000	start2:sw $9, 0($8)	将 9 号寄存器内容存到 dm[[$8]]处
8d0e0000	lw $14, 0($8)	将 dm[[$8]]处值赋给 14 号寄存器
ad0a0004	sw $10,4($8)	将 10 号寄存器内容存到 dm[[$8]+4]处
8d0f0004	lw $15,4($8)	将 dm[[$8]+4]处值赋给 15 号寄存器

ad04fffc	sw $4, -4($8)	将 4 号寄存器内容存到 dm[[$8]-4]处
8d12fffc	lw $18, -4($8)	将 dm[[$8]-4]处值赋给 18 号寄存器
00082021	addu $4,$0,$8	将 0 号寄存器与 8 号寄存器内容相加放入 4 号寄存器
00092821	addu $5,$0,$9	将 0 号寄存器与 9 号寄存器内容相加放入 5 号寄存器
0c000c32	jal newadd	跳转到 newadd 处，并将下条指令地址放入 31 号寄存器
0148c82a	slt $25,$10,$8	如果[$10]<[$8],则[$25]=1,否则为 0
13200018	beq $25, $0,end2	若[$25]=0 则跳转到 end2，否则顺序执行
0184a02a	slt $20,$12,$4	如果[$12]<[$4],则[$20]=1,否则为 0
12800001	beq $20, $0, end1	若[$20]=0 则跳转到 end1，否则顺序执行
3c0cffff	lui $12, 65535	给 12 号寄存器高 16 位赋值 65535
34000001	end1:ori $0, $0,1	不执行任何操作
3c13efef	lui $19, 0xefef	给 19 号寄存器高 16 位赋值 0xefef
3c01abab	addiu $3,$0,0xababcdcd	将 0 号寄存器与 0xababcdcd 相加放入 3 号寄存器
24640002	start3:addiu $4, $3, 2	将 3 号寄存器内容+2 赋给 4 号寄存器（无溢出加法）
20770005	addi $23, $3, 5	将 3 号寄存器内容+5，若无溢出，则将结果赋给 23 号寄存器，否则将 30 号寄存器第 0 位置 1
0c000c32	jal newadd	跳转到 newadd 处，并将下条指令地址放入 31 号寄存器
00024021	addu $8, $0, $2	把 0 号寄存器与 2 号寄存器内容相加赋给 8 号寄存器
00082021	addu $4, $0, $8	把 0 号寄存器与 8 号寄存器内容相加赋给 4 号寄存器
00092821	addu $5, $0, $9	把 0 号寄存器与 9 号寄存器内容相加赋给 5 号寄存器
0c000c32	jal newadd	跳转到 newadd 处，并将下条指令地址放入 31 号寄存器
00024821	addu $9, $0, $2	把 0 号寄存器与 2 号寄存器内容相加赋给 9 号寄存器
01004821	addu $9, $8, $0	把 8 号寄存器与 0 号寄存器内容相加赋给 9 号寄存器
3c0a0069	lui $10, 0x69	给 10 号寄存器高 16 位赋值 0x69
11090001	beq $8, $9, start4	若[$8]=[$9]则跳转到 start1，否则顺序执行

1000fff4	beq $0, $0, start3	跳转到 start3
08000c36	start4:j end	跳转到 end
00851021	newadd:addu $2, $4, $5	将 4 号寄存器内容与 5 号寄存器内容相加赋给 2 号寄存器（无溢出加法）
21801234	addi $0,$12,0x1234	将 12 号寄存器内容+0x1234，若溢出，将 30 号寄存器第 0 位置 1
03e00008	jr $31	跳转至 31 号寄存器中存储的地址
201a5678	end2:addi $26,$0,0x5678	给 26 号寄存器赋值 0x5678
	end:	代码结束

5 测试结果

5.1 GPR 运行结果

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0xababcdcd
$v0	2	0xababcdd3
$v1	3	0xababcdcd
$a0	4	0x2babcdd1
$a1	5	0x80000002
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x2babcdd1
$t1	9	0x2babcdd1
$t2	10	0x00690000
$t3	11	0x00000000
$t4	12	0x0000abab
$t5	13	0x000a0000
$t6	14	0x80000002
$t7	15	0x80000004
$s0	16	0x00000003
$s1	17	0x00000001
$s2	18	0x00000002
$s3	19	0xefef0000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0xababcdd2
$t8	24	0x7fffffff
$t9	25	0x00000001
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x00001800
$sp	29	0x00002ffc
$fp	30	0x00000000
$ra	31	0x000030b0
pc		0x000030d8
hi		0x00000000
lo		0x00000000

图 3：Mars 中的 GPR

Memory Data - /test_mips/my

31	000030b0
30	00000000
29	00000000
28	00000000
27	00000000
26	00000000
25	00000001
24	7fffffff
23	ababcdd2
22	00000000
21	00000000
20	00000000
19	efef0000
18	00000002
17	00000001
16	00000003
15	80000004
14	80000002
13	000a0000
12	0000abab
11	00000000
10	00690000
9	2babcdd1
8	2babcdd1
7	00000000
6	00000000
5	80000002
4	2babcdd1
3	ababcdcd
2	ababcdd3
1	ababcdcd
0	00000000

图 4：modelsim 中的 GPR

5.2 DM 运行结果

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)
0x00000000	0x00000002	0x80000002	0x80000004	0x00000000
0x00000020	0x00000000	0x00000000	0x00000000	0x00000000
0x00000040	0x00000000	0x00000000	0x00000000	0x00000000
0x00000060	0x00000000	0x00000000	0x00000000	0x00000000
0x00000080	0x00000000	0x00000000	0x00000000	0x00000000
0x000000a0	0x00000000	0x00000000	0x00000000	0x00000000

图 5：Mars 中的 DM

0000003b	xx xx xx xx
00000037	xx xx xx xx
00000033	xx xx xx xx
0000002f	xx xx xx xx
0000002b	xx xx xx xx
00000027	xx xx xx xx
00000023	xx xx xx xx
0000001f	xx xx xx xx
0000001b	xx xx xx xx
00000017	xx xx xx xx
00000013	xx xx xx xx
0000000f	xx xx xx xx
0000000b	80 00 00 04
00000007	80 00 00 02
00000003	00 00 00 02

图 6：modelsim 中的 DM

5.3 波形图

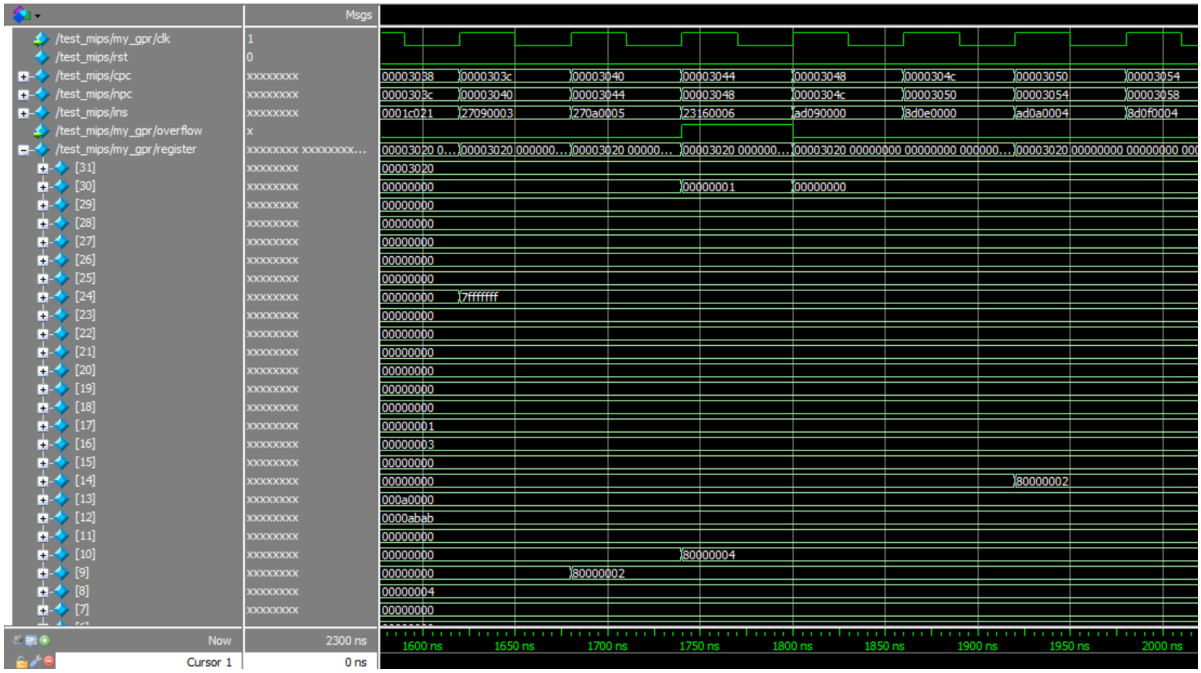

图 7：局部仿真结果截图

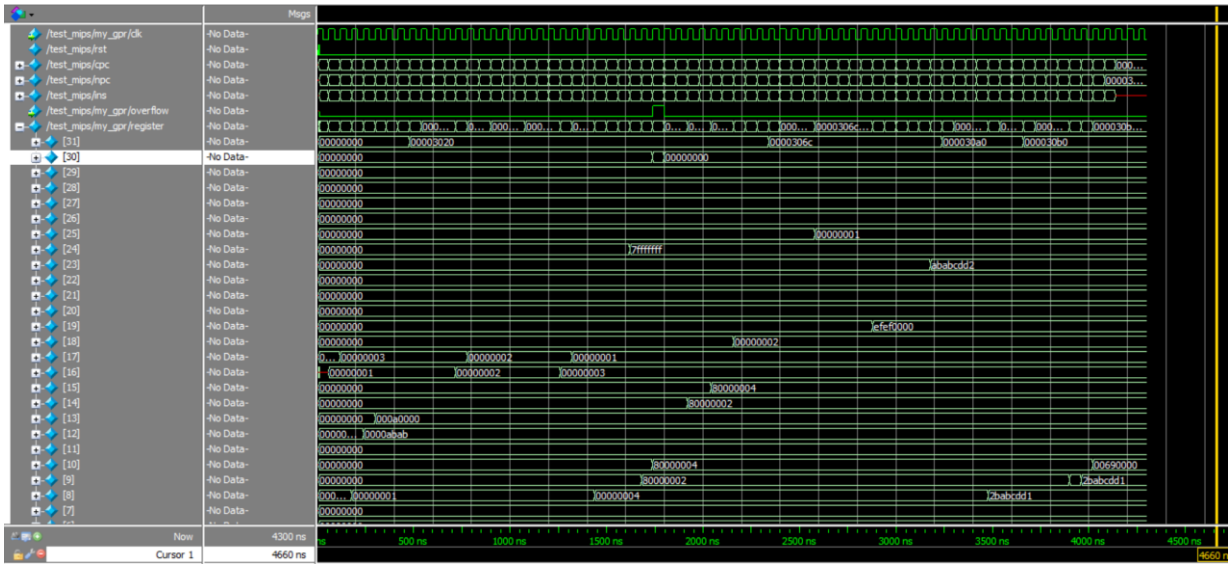

图 8：仿真结果整体截图

6 总结与收获

这次期末大作业主要实现功能即把计算机组成原理大作业中用 logisim 中连线模块转换为 Verilog 代码语言实现，这个过程让我复习了 Verilog 语言的写法，并掌握了之前使用不多的模块间结构化连接方式、testbench 测试方法以及 modelsim 软件。也加强了我对于模块化层次化设计的能力，构造数字系统时采用分层设计和模块设计可以将复杂的问题拆解成若干容易解决的小问题，降低了解决问题的难度，完成这次大作业的经历对于我之后的学习有很大帮助。