南通大学信息科学技术学院

《计算机组成原理》 课程设计

报告册

组 长 计214-2130110420-朱恒宇

成 员 计214-2130110418-张允轩

成 员 计214-2130110419-赵 延

成 员 计214-2130110422-朱续杰

指导教师 顾晖

设计时间 2023.6.12~2023.6.16

2022 —— 2023 学年第 2 学期

一、设计目的

1．巩固《计算机组成原理》这门课程的知识，通过知识的综合运用，使学生加深对计算机系统各模块的工作原理及相互联系的认识，加深计算机工作中“时间-空间”概念的理解，从而清晰地建立计算机的整机概念。

2．学习设计和调试计算机的基本步骤和方法，培养学生独立开展科学研究的工作能力，使学生取得解决计算机领域工程复杂问题时必须的工程实践经验和能力，理解团队工作的责任。培养学生担任团队成员或负责人的角色的能力，培养学生独立完成团队分配的工作的能力。

二、设计小组情况

本课程设计小组由朱恒宇，张允轩，赵延，朱续杰人组成，小组成员及分工情况，见下表。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 班级 | 学号 | 姓名 | 角色 | 分工 |
| 1 | 计214 | 2130110420 | 朱恒宇 | 组长 | (2) IRI20型指令数据通路设计及指令执行阶段控制信号的取值分析  (3) 3R型指令数据通路设计及指令执行阶段控制信号的取值分析  (4) 2RI12型指令数据通路设计及指令执行阶段控制信号的取值分析  (5) I26型指令数据通路设计及指令执行阶段控制信号的取值分析  (6) 2RI16型指令数据通路设计及指令执行阶段控制信号的取值分析  (9) 测试程序设计与编写  (10) 模型机软硬件仿真联调  (11) 课程设计报告的整理与撰写 |
| 2 | 计214 | 2130110418 | 张允轩 | 组员 | (7) 控制器设计 |
| 3 | 计214 | 2130110419 | 赵延 | 组员 | (8) 模型机整机互连 |
| 4 | 计214 | 2130110422 | 朱续杰 | 组员 | (1) 取指部件设计 |

二、课程设计题目

1.课程设计任务与要求

本次课程设计的任务是：设计并实现RV32I 架构的模型机硬件系统。该模型机的指令系统包括以下14条指令：设计了几条指令，下表中就列出几条指令。要求在EDA工具（Vivado）仿真环境下完成电路的建模设计、仿真调试与运行。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 类型 | 指令 | 功能 | 说明 |
| 1 | 1RI20 | LU12I.W rd,si20 | GR[rd] ⟵si20 || 12’b0 | ①GR[rd]的高20位为si20，低12位为0  ②符号||表示拼接 |
| 2 | 3R | ADD.W rd, rj, rk | GR[rd]⟵GR[rj]+GR[rk] | 加法 |
| 3 | 3R | SUB.W rd, rj, rk | GR[rd]⟵GR[rj]-GR[rk] | 减法 |
| 4 | 3R | SLT rd, rj, rk | if (GR[rj]<GR[rk]) GR[rd]⟵1 else GR[rd]⟵0 | 带符号数的大小比较 |
| 5 | 3R | SLTU rd, rj, rk | if (GR[rj]<GR[rk]) GR[rd]⟵1 else GR[rd]⟵0 | 无符号数的大小比较 |
| 6 | 3R | NOR rd, rj, rk | GR[rd]⟵ | 或非 |
| 7 | 3R | AND rd, rj, rk | GR[rd]⟵GR[rj] ∧ GR[rk] | 与 |
| 8 | 3R | OR rd, rj, rk | GR[rd]⟵GR[rj] ∨ GR[rk] | 或 |
| 9 | 2RI12 | ADDI.W rd, rj, si12 | GR[rd]⟵GR[rj]+ SignExtend(si12) | ①立即数加法  ②si12是12位立即数，进行符号位扩展后与GR[rj]相加 |
| 10 | 2RI12 | LD.W rd ,rj,si12 | Addr⟵GR[rj] + SignExtend (si12) ,GR[rd] ⟵M[Addr] | ①将内存Addr单元的值取出后存入R[rd]  ②si12是12位立即数，进行符号位扩展后与GR[rj]相加后得到内存单元的地址Addr |
| 11 | 2RI12 | ST.W rd, rj, si12 | Addr⟵GR[rj] + SignExtend (si12) , M[Addr]⟵GR[rd] | ①把GR[rd]的值存入内存Addr单元  ②内存单元的地址Addr的计算方法与LD.W指令相同 |
| 12 | I26 | B offs26 | PC⟵PC+ SignExtend (offs26||2’b0) | 无条件跳转到目标地址 |
| 13 | 2RI16 | BEQ rj, rd, offs16 | if (GR[rj]=GR[rd])  PC⟵PC+ SignExtend (offs16||2’b0) | ①符号||表示拼接  ②GR[rj]=GR[rd]时，跳转到目标地址 |
| 14 | 2RI16 | BLT rj, rd, offs16 | if (signed(GR[rj])<signed(GR[rd]))  PC⟵PC+ SignExtend (offs16||2’b0) | ①符号||表示拼接  ②GR[rj]与GR[rd]视作带符号数  ③GR[rj]<GR[rd]时，跳转到目标地址 |

2. 课程设计完成条件

一台安装了EDA工具软件（Vivado）的计算机。

3.设计思路与方案

1. 设计思路
   1. 逐条分析要实现的LA32R指令的格式与功能。
   2. 设计取指令、执行指令所需的执行部件，并逐一进行仿真测试，完成功能验证。
   3. 按指令功能，设计各指令的数据通路，将执行部件进行互连，并进行仿真测试，完成功能验证。
   4. 设计控制部件，并进行仿真测试，实现指令译码功能验证。
   5. 将执行部件与控制部件进行互连，完成模型机的设计，并进行仿真测试，通过分析测试程序的运行结果，完成模型机功能的验证。
2. 设计方案

本次设计的模型机电路如图1所示。该模型机是单总线结构总线结构，数据总线32位，地址总线32位。该电路包括运算器、存储器、控制器和通用寄存器等模块。其中，运算器可以完成加、减、或、与、或非、直通、有符号数比较以及无符号数比较功能。存储器可以完成存数取数功能，容量是2的10次方。控制器是采用时序逻辑实现的硬布线控制器。时序产生器可以完成产生时钟信号功能,主频是10MHZ。有32个通用寄存器，可以存储32位数据。

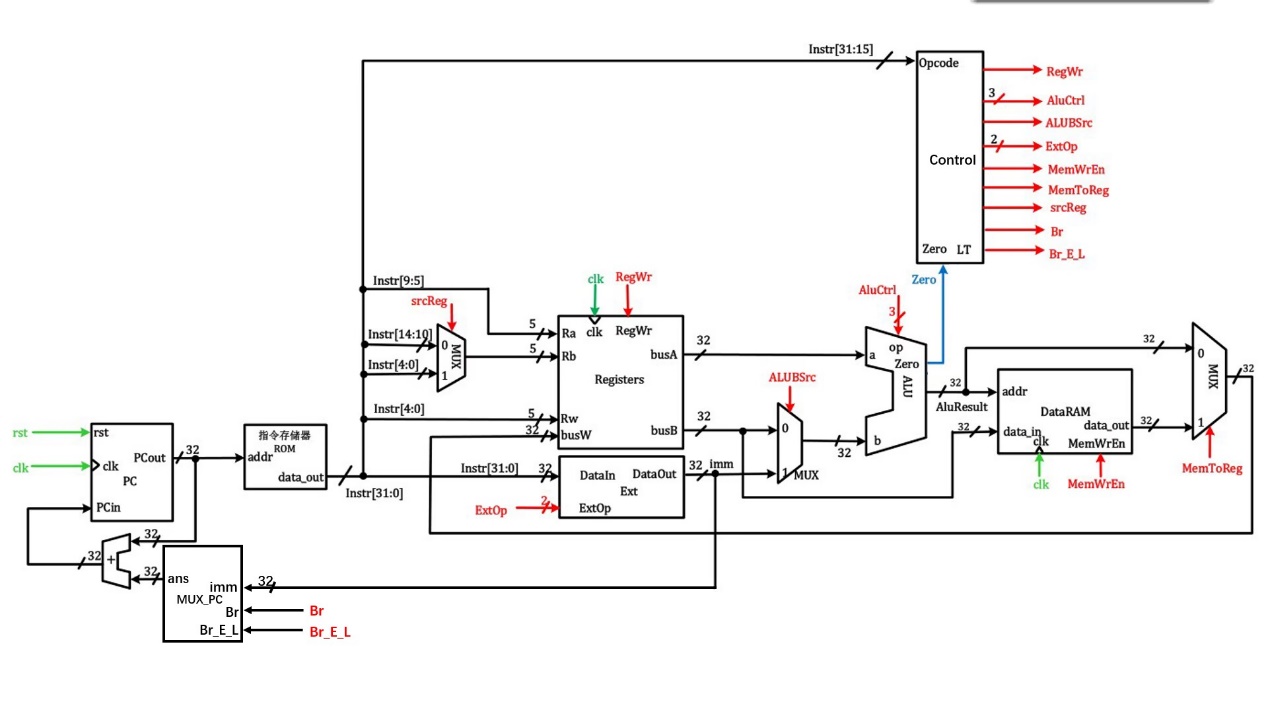


图1模型机电路框图

图1中各功能器件上标注的控制信号的功能说明见下表。

表1微操作信号说明

|  |  |
| --- | --- |
| 信号名称 | 信号功能说明 |
| clk | 脉冲信号，上升沿时刻，配合其他信号从而使部件工作。 |
| rst | 异步复位信号，上升沿时刻，是程序计数器复位，从而实现开机操作。 |
| RegWr | 写使能控制信号，clk上升沿时刻，若RegWr为1，则busW上的数据被存入Rw指定的寄存器中 |
| ALUSrc | ALUBSrc =1，将Ext模块扩展后的值作为ALU模块b的输入。ALUBSrc =0时，将Registers模块busb的值作为ALU模块b的输入； |
| AluCtrl | 运算类型选择输入 |
| ExtOp | 扩展方式选择信号 |
| MemWrEn | 写使能信号。当MemWrEn为0时，数据从addr地址端口指定的内存单元读出。  当MemWrEn为1时，配合clk时钟信号工作，在clk上升沿，数据存入由addr地址端口指定的内存单元。 |
| MemToReg | MUX数据输入选择信号。根据不同指令，选择将ALU的运算结果或是DataRAM取出的值作为要存入寄存器堆的值。 |
| srcReg | MUX数据输入选择信号。根据不同指令，选择指令提供的rk或rd值作为寄存器堆Rb的输入。 |
| Br | 无条件跳转控制信号 |
| Br\_E\_L | 有条件跳转控制信号 |

3.设计原理与内容

1. 通用寄存器堆

通用寄存器堆包含32个寄存器，记为：r0~r31。每个寄存器的位宽都是：32位。其中，r0的值恒为0。其电路原理图如下：



该模块的功能及引脚信号说明如下：

|  |  |
| --- | --- |
| 信号名称 | 功能说明 |
| busA和busB | 两路32位数据输出信号 |
| Ra(5位) | 读寄存器编号输入信号，该编号指定的寄存器的值经过“取数延迟”后，输出到busA |
| Rb(5位) | 读寄存器编号输入信号，该编号指定的寄存器的值经过“取数延迟”后，输出到busB |
| Rw(5位) | 写寄存器编号输入信号，该编号指定的数据要写入哪个寄存器 |
| busW | 32位数据输入信号 |
| clk | 写操作时钟控制信号，上升沿有效 |
| RegWr | 写使能控制信号，clk上升沿时刻，若RegWr为1，则busW上的数据被存入Rw指定的寄存器中 |

注：

1. 寄存器堆的读操作不受clk控制；
2. 0号寄存器的值恒为0，不受写操作的影响。

该模块的Verilog HDL代码设计如下：

module Registers(

    input[31:0] busW,//初始为0,接受mux3\_ans

    input clk,

    input RegWr,

    input[4:0] Ra,//是Instr[9:5]

    input[4:0] Rb,//是MUX选择后输入进来的

    input[4:0] Rw,//是Instr[4:0]

    output[31:0] busA,//32位数据输出信号

    output[31:0] busB//32位数据输出信号,要输出给下一个MUX

    );

    reg[31:0] registers[31:0];

    integer i;

    initial begin

        for(i = 0;i < 32;i = i + 1)

            registers[i] <= 0;

    end

    assign busA = (Ra == 0) ? 32'h0 : registers[Ra];

    assign busB = (Rb == 0) ? 32'h0 : registers[Rb];

    always @(posedge clk) begin

        if (RegWr && (Rw != 0))

            registers[Rw] <= busW;

    end

endmodule

1. 立即数扩展模块

其电路原理图如下：



该模块的功能及引脚信号说明如下：

|  |  |  |
| --- | --- | --- |
| Extop | 功能 | 说明 |
| 0 | Dataout←{{20{DataIn[21]}},DataIn[21:10]} | 对ADDI.W, LD.W, ST.W指令中的12位立即数进行符号位扩展 |
| 1 | Dataout←{{16{DataIn[25]}},{DataIn[25:10]}} | 对BEQ,BLT指令中的立即数进行符号位扩展及其他部分的拼接 |
| 2 | Dataout←{{12{DataIn[24]}},DataIn[24:5]} | 对LUI12I.W指令中的20位立即数高12位补0 |
| 3 | Dataout←{{6{DataIn[25]}},10'b0,DataIn[25:10]} | 对B指令中的立即数进行符号位扩展及其他部分的拼接 |

|  |  |
| --- | --- |
| 信号名称 | 功能说明 |
| Datain | 32位的数据输入信号 |
| Extop | 扩展方式选择信号 |
| Dataout | 32位的数据输出信号 |

该模块的Verilog HDL代码设计如下：

module Ext(

    input [31:0] DataIn,

    input [1:0]ExtOp,

    output reg[31:0] DataOut

    );

    always@(\*)

    begin

        case(ExtOp)

            2'b00:

                DataOut = {{20{DataIn[21]}},DataIn[21:10]};

            2'b01:

                DataOut = {{16{DataIn[25]}},{DataIn[25:10]}};

            2'b10:

                DataOut = {{12{DataIn[24]}},DataIn[24:5]};

            2'b11:

                DataOut = {{6{DataIn[25]}},10'b0,DataIn[25:10]};

            default: DataOut = 32'b0;

        endcase

    end

endmodule

1. 程序计数器PC

其电路原理图如下：



该模块的功能及引脚信号说明如下：

|  |  |
| --- | --- |
| 信号名称 | 功能说明 |
| rst | PC异步清零信号，高电平有效，即：rst为1时，PCdata =0 |
| clk | 时钟信号 |
| offset | 32位偏移量 |
| pc\_inc | 自增控制信号，与clk上升沿配合工作。  在clk上升沿时刻，当pc\_inc=1时，PCdata =原PCdata +1；当pc\_inc=0时，PCdata =原PCdata +offset。 |
| PCdata | 32位数据输出信号 |

该模块的Verilog HDL代码设计如下：

module PC(

    input wire clk,

    input wire rst,

    input wire [31:0] PCin,

    output reg [31:0] PCout

);

    always@(posedge clk)

    begin

        if(rst==0)

        begin

            PCout <= 32'b0;

        end

        else

        begin

            PCout <= PCin;

        end

    end

endmodule

1. 算术逻辑运算单元ALU

其电路原理图如下：



该模块的功能及引脚信号说明如下：

|  |  |  |
| --- | --- | --- |
| **功能编号** | **功能** | **说明** |
| 0 | AddResult =a+b | 加法 |
| 1 | AddResult =a-b | 减法 |
| 2 | AddResult =a⋀b | 与 |
| 3 | AddResult =a⋁b | 或 |
| 4 | AddResult = | 或非 |
| 5 | a<b时AddResult =1 | A和B是带符号数，进行带符号数比较 |
| 6 | a<b时AddResult =1 | A和B是无符号数，进行无符号数比较 |
| 7 | AddResult =b | 直通 |

|  |  |
| --- | --- |
| 信号名称 | 功能说明 |
| a, b | 两个32位的数据输入 |
| op | 运算类型选择输入 |
| AddResult | 运算结果输出 |
| Zero | 0标志位输出。运算结果AddResult为0时，Zero=1；否则Zero=0 |

该模块的Verilog HDL代码设计如下：

module ALU(

    input[31:0] a,

    input[31:0] b,

    input[2:0]AluCtrl,//一共有8个运算方法，需要3位来控制

    output wire Zero,//结果是否为0？是为1，否为0

    output reg[31:0] AluResult

);

    always@(\*)

    begin

        case(AluCtrl)

            3'b000: AluResult = a + b;

            3'b001: AluResult = a - b;

            3'b010: AluResult = a & b;

            3'b011: AluResult = a | b;

            3'b100: AluResult = ~( a | b );

            3'b101: AluResult = ($signed(a) < $signed(b)) ? 32'h1 : 32'h0; // 带符号数比较

            3'b110: AluResult = (a < b) ? 32'h1 : 32'h0; // 无符号数比较

            3'b111:

            begin

                AluResult = b;

            end

            default:

            begin

                AluResult = 32'h00000000;

            end

        endcase

    end

    assign Zero = (AluResult == 0)?1:0;

endmodule

1. 数据存储器

其电路原理图如下：



图中信号说明如下：

|  |  |
| --- | --- |
| 信号名称 | 功能说明 |
| addr | 32位地址总线，用于传送地址，以便按地址访问存储单元。 |
| data\_in | 32位数据输入总线 |
| data\_out | 32位数据输出总线 |
| clk | 时钟信号，上升沿有效 |
| MemWrEn | 写使能信号。  当MemWrEn为0时，数据从addr地址端口指定的内存单元读出。  当MemWrEn为1时，配合clk时钟信号工作，在clk上升沿，数据存入由addr地址端口指定的内存单元。 |

该模块的Verilog HDL代码设计如下：

module DataRAM(

    input[31:0] addr,

    input[31:0] data\_in,

    input clk,

    input MemWrEn,

    output reg[31:0] data\_out

    );

    reg[31:0] mem[0:1023];

    always@(posedge clk)

    begin

        if(MemWrEn)

            mem[addr] <= data\_in;

    end

    always@(\*)

    begin

        if(!MemWrEn)

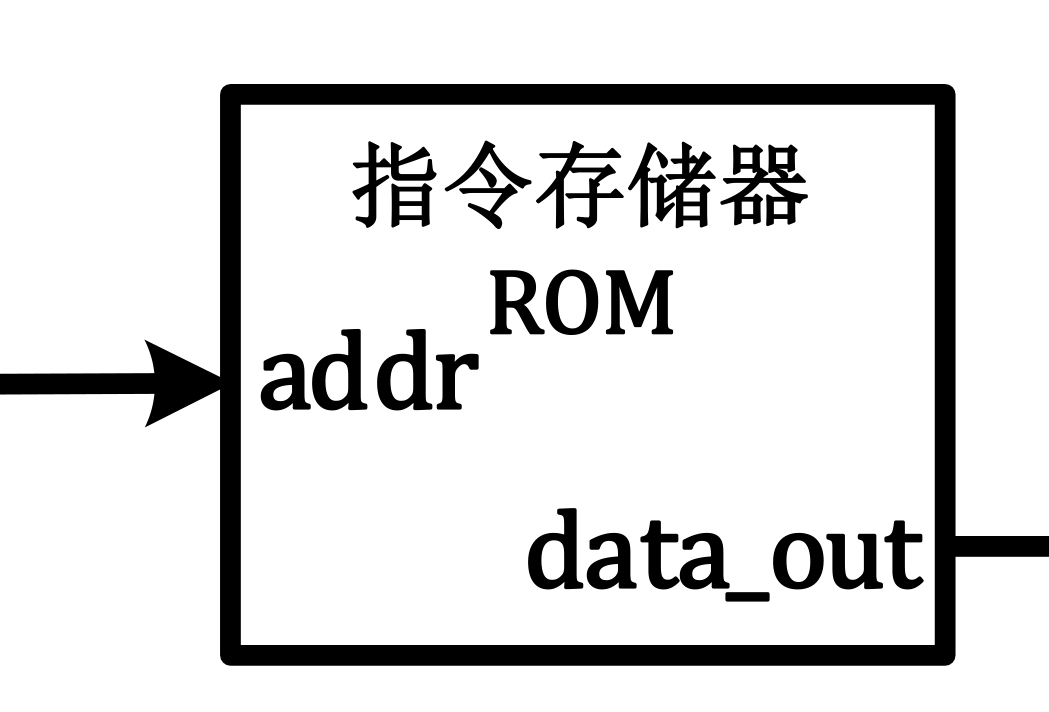
            data\_out <= mem[addr];

    end

endmodule

1. 指令存储器

其电路原理图如下：



该模块的Verilog HDL代码设计如下：

module ROM(

    input wire[31:0] addr,

    output reg[31:0] data\_out

    );

    reg [31:0] rom[127:0];

    initial

        begin

            $readmemb("C:/Users/Judy/Desktop/CPU/design/program.txt",rom);

            data\_out=0;

        end

    always@(\*)

    begin

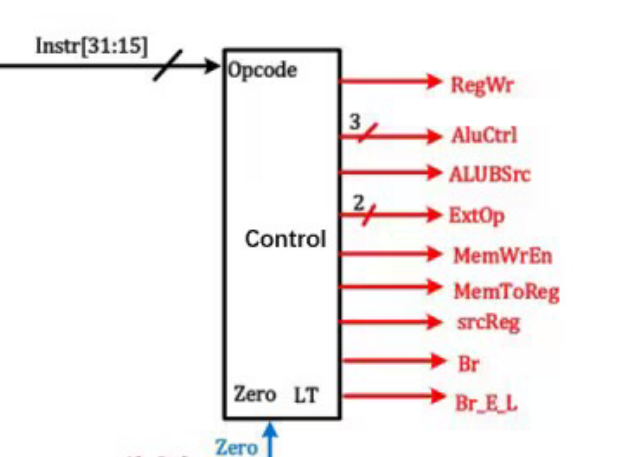
        data\_out <= rom[addr];

    end

endmodule

1. 控制器

其电路原理图如下：



图中输入信号Instr[31:15]为要执行的指令字的部分字段，输出信号RegWr, AluCtrl, ALUBSrc, ExtOp, MemWrEn,MemToReg和srcReg为控制器根据不同指令所产生的控制信号。

该模块的Verilog HDL代码设计如下：

module Control(

    input wire [5:0] op,

    input wire [3:0] fun1,

    input wire [6:0] fun2,

    input wire Zero,Sign,

    output wire srcReg,

    output wire RegWr,

    output wire [1:0] ExtOp,

    output wire ALUBSrc,

    output wire [2:0] ALUCtrl,

    output wire MemWrEn,

    output wire MemToReg,

    output wire Br,

    output wire Br\_E\_L

    );

    reg [11:0] controls;

    assign {srcReg,RegWr,ExtOp,ALUBSrc,ALUCtrl,MemWrEn,MemToReg,Br,Br\_E\_L} = controls;

    always@(\*)

    begin

        case(op)

            6'b000000:begin

                case(fun1)

                    4'b0000:begin

                        case(fun2)

                            7'b0100000: controls <= 12'b010000000000;//add.w 加法

                            7'b0100010: controls <= 12'b010000010000;//sub.w 减法

                            7'b0101001: controls <= 12'b010000100000;//and   与

                            7'b0101010: controls <= 12'b010000110000;//or    或

                            7'b0101000: controls <= 12'b010001000000;//nor   或非

                            7'b0100100: controls <= 12'b010001010000;//sltu 无符号数比较

                            7'b0100101: controls <= 12'b010001100000;//slt  有符号数比较

                        endcase

                    end

                    4'b1010: controls <= 12'b110010000000;//addi.w 立即数加法

                endcase

            end

            6'b001010:begin

                case(fun1)

                    4'b0010: controls <= 12'b110010000100;//ld.w

                    4'b0110: controls <= 12'b100010001000;//st.w

                endcase

            end

            6'b000101:begin

                case(fun1[3])

                    1'b0: controls <= 12'b111011110000;//lu.w 置数

                endcase

            end

            6'b010100: controls <= 12'b00\_11\_0\_000\_0010;//b 无条件跳转指令

            6'b010110:begin

                case(Zero)//判断Zero如果相等 Zero为1

                    1'b1: controls <= 12'b101100010001;//beq 判断相等条件跳转指令

                    1'b0: controls <= 12'b101100010000;

                endcase

            end

            6'b011000: begin

                case(Zero)//判断是否小于 如果小于Zero为0

                    1'b0: controls <= 12'b101101010001;//blt 判断小于条件跳转指令

                    1'b1: controls <= 12'b101101010000;

                endcase

            end

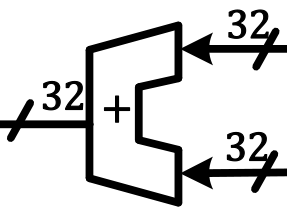
        endcase

    end

endmodule

1. 加法器

其电路原理图如下：



它有两个32位的输入端口a和b，以及一个32位的输出端口ans。在模块中，将输入端口a和b相加，并将结果赋值给输出端口ans。换句话说，这个模块将执行两个32位整数的加法操作，并将结果输出。

该模块的Verilog HDL代码设计如下：

module ADD(

    input wire[31:0] a,

    input wire[31:0] b,

    output wire[31:0] ans

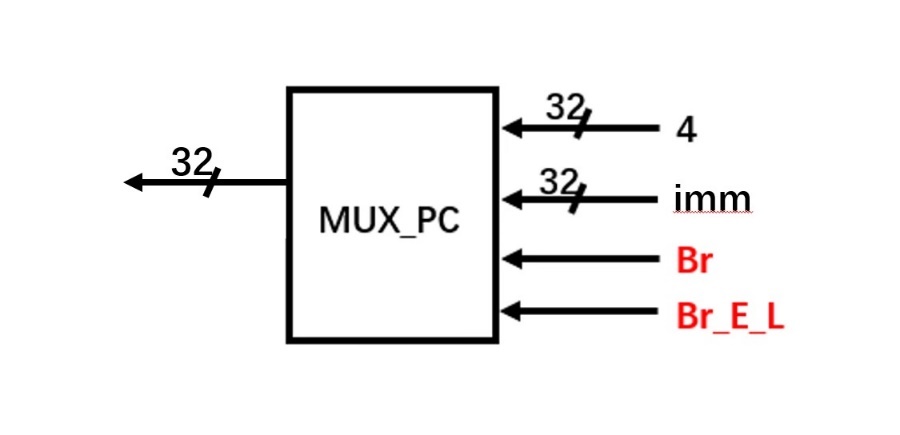
);

    assign ans = a+b;

endmodule

1. 分支选择器

其电路原理图如下：



这段代码实现了一个名为 `Mux\_PC` 的模块，其功能是根据输入的条件选择性地将 `data\_in` 的值赋给 `ans`。

该模块具有以下输入端口：Br：单比特输入线，表示一个条件。Br\_E\_L：单比特输入线，表示另一个条件。Zero：单比特输入线，表示第三个条件。data\_in：32 位输入线，传递数据值。还有一个输出端口：ans：32 位寄存器输出线，用于存储结果值。

在 `always` 块中，使用条件语句对 `ans` 进行赋值。根据条件，如果 `Br` 或者 `Br\_E\_L` 为真，则将 `data\_in` 的值赋给 `ans`。否则，将 `ans` 设置为二进制值 `32'b00000000000000000000000000000001`。

该模块的功能是根据条件选择性地将输入数据传递给输出寄存器。

该模块的Verilog HDL代码设计如下：

module Mux\_PC(

    input wire Br,

    input wire Br\_E\_L,

    input wire Zero,

    input wire[31:0] data\_in,

    output reg [31:0] ans

);

    always@(\*)

    begin

        if(Br || Br\_E\_L)

            ans <= data\_in;

        else

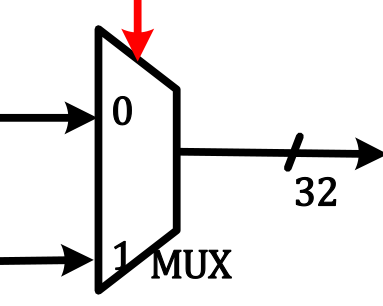
            ans=32'b00000000000000000000000000000001;

    end

endmodule

1. 32位多路复用器

其电路原理图如下：



它有两个32位的输入信号a和b，一个控制信号ALUBSrc，以及一个32位的输出信号result。根据代码逻辑，当ALUBSrc的值为0时，多路复用器的输出信号result等于输入信号a；否则，输出信号result等于输入信号b。这意味着根据控制信号ALUBSrc的值，多路复用器会将其中一个输入信号传递到输出信号上，实现信号的选择和切换功能。

该模块的Verilog HDL代码设计如下：

module Mux\_32bit(

    input wire [31:0] a,

    input wire [31:0] b,

    input wire ALUBSrc,

    output wire[31:0] result

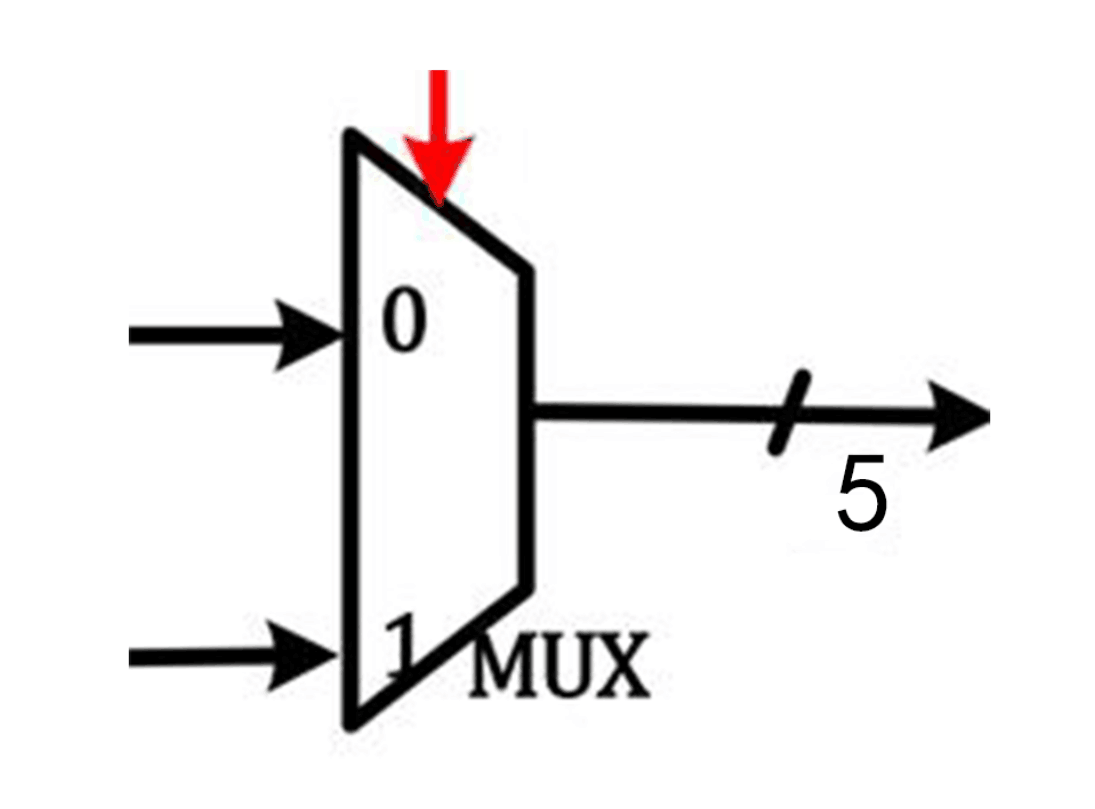
    );

    assign result=(ALUBSrc==0)?a:b;

endmodule

1. 5位多路复用器

其电路原理图如下：



该代码实现了一个5位多路复用器（MUX），根据输入的选择信号`srcReg`来选择并输出不同的数据。

模块`Mux\_5bit`有以下端口：

输入端口：a：5位宽的输入信号a。b：5位宽的输入信号b。srcReg：选择信号，用于确定输出的是信号a还是信号b。

输出端口：result：5位宽的输出信号，根据选择信号选择对应的输入信号输出。

代码中的`assign`语句使用了条件运算符（三目运算符）：`(srcReg==0)?a:b`。当`srcReg`等于0时，输出信号`result`等于输入信号`a`，否则输出信号`result`等于输入信号`b`。这意味着如果选择信号`srcReg`为0，则输出选择信号`a`；否则，输出选择信号`b`。

该模块的Verilog HDL代码设计如下：

module Mux\_5bit(

    input wire [4:0] a,

    input wire [4:0] b,

    input wire srcReg,

    output wire[4:0] result

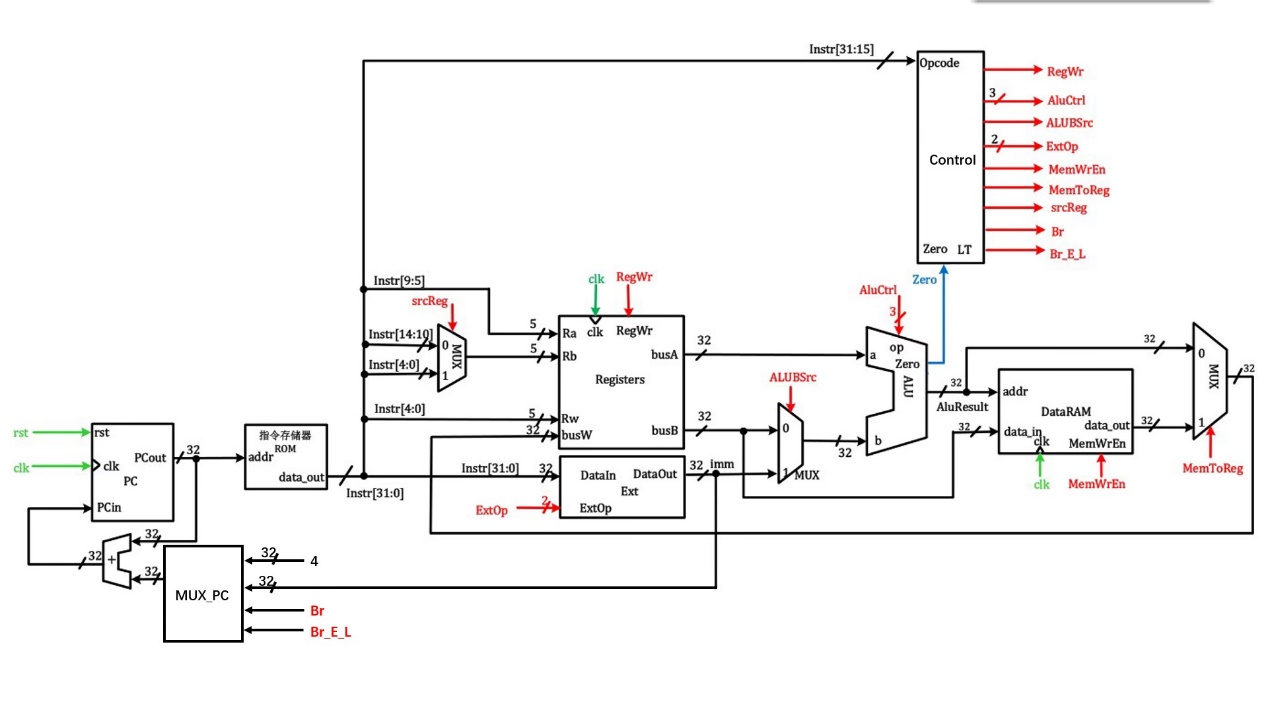
    );

    assign result=(srcReg==0)?a:b;

endmodule

1. 单周期LA32R CPU电路

其电路原理图如下：



该代码实现了一个 CPU\_top 模块，用于构建一个简单的 CPU。以下是各个模块的功能描述：

Control 模块：根据指令（Instr）的不同字段解析，生成控制信号，包括源寄存器选择（srcReg）、寄存器写使能（RegWr）、立即数操作（ExtOp）、ALU 第二个操作数来源选择（ALUBSrc）、ALU 控制信号（ALUCtrl）、内存写使能（MemWrEn）、写回寄存器选择（MemToReg）、无条件跳转信号（Br）、条件跳转信号（Br\_E\_L）以及零标志位（Zero）。

PC 模块：根据时钟信号（clk）和复位信号（rst）控制 PC 的工作，将输入信号（adder\_ans）作为 PC 的输入，将 PC 的输出（PCout）提供给其他模块。ROM 模块：指令存储器，根据给定地址（PCout）读取对应的指令（Instr）。

ADD 模块：加法器，将输入信号（mux\_pc\_ans）和 PC 输出信号（PCout）相加，并将结果（adder\_ans）提供给其他模块。

Mux\_PC 模块：信号选择器，根据条件跳转信号（Br、Br\_E\_L）和零标志位（Zero）选择信号，将立即数（imm）或者 PC 输出信号（PCout）作为输出（mux\_pc\_ans）。

Mux\_5bit 模块：2选1选择器1，根据指令的字段（Instr[14:10]和Instr[4:0]）以及源寄存器选择信号（srcReg）选择一个5位的结果（mux1\_ans）。

Mux\_32bit 模块：2选1选择器2，根据 ALU 第二个操作数来源选择信号（ALUBSrc）选择一个32位的结果（mux2\_ans），作为 ALU 的第二个操作数。

Mux\_32bit 模块：2选1选择器3，根据内存写使能信号（MemToReg）选择 ALU 结果（AluResult）或者数据存储器输出（data\_out）作为输出（mux3\_ans）。

Registers 模块：寄存器，根据控制信号进行读写操作，包括写数据（mux3\_ans）、时钟信号（clk）、寄存器写使能信号（RegWr）、寄存器 A 和 B 的选择信号（Instr[9:5]和mux1\_ans）、写入数据总线 A（busA）和总线 B（busB）。

Ext 模块：立即数拓展，根据指令（Instr）和立即数操作信号（ExtOp）将指令中的数据拓展为32位的立即数（imm）。

ALU 模块：运算器，根据输入操作数（busA、mux2\_ans）和 ALU 控制信号（AluCtrl）进行运算，同时输出运算结果（AluResult）和零标志位（Zero）。

DataRAM 模块：RAM 存储器，根据地址（AluResult）读写数据，时钟信号（clk）控制操作，读取的数据提供给数据输出（data\_out），写入的数据来自总线 B（busB）。

通过以上各个模块的协同工作，CPU\_top 模块实现了一个基本的 CPU 功能，包括指令的存取、指令解析、寄存器读写、运算、内存读写等。具体的功能细节需要根据各个模块的实现来确定。

该模块的Verilog HDL代码设计如下：

module CPU\_top(

    input clk,input rst,

    output wire [31:0] Instr,                       //ROM的输出Instr

    output wire [31:0] mux\_pc\_ans,                  //Mux\_PC的输出muxPcAns

    output wire [31:0]PCout,                        //PC的输出PCout

    output wire [31:0]adder\_ans,                    //ADD的输出adder\_ans

    output wire srcReg,                             //Control的mux1控制信号srcReg

    output wire RegWr,                              //Control的muxRegisters控制信号RegWr

    output wire [1:0] ExtOp,                        //Control的mux1控制信号ExtOp

    output wire ALUBSrc,                            //Control的mux1控制信号ALUBSrc

    output wire [2:0] ALUCtrl,                      //Control的mux1控制信号ALUCtrl

    output wire MemWrEn,                            //Control的mux1控制信号MemWrEn

    output wire MemToReg,                           //Control的mux1控制信号MemToReg

    output wire Br,                                 //Control的无条件跳转信号Br

    output wire Br\_E\_L,                             //Control的条件跳转信号Br\_E\_L

    output wire [4:0]mux1\_ans,                      //mux1的输出(5位)

    output wire [31:0] busA,output wire [31:0] busB,//Registers的输出

    output wire [31:0] imm,                         //Ext的输出

    output wire [31:0] mux2\_ans,                    //mux2的输出(32位)

    output wire Zero,output wire [31:0] AluResult, //ALU的输出Zero,Sign和AluResult

    output wire [31:0] data\_out,                    //DataRAM的输出data\_out

    output wire [31:0] mux3\_ans                     //mux3的输出(32位)

);

    //控制器

    Control control(.op(Instr[31:26]),.fun1(Instr[25:22]),.fun2(Instr[21:15]),.Zero(Zero),.srcReg(srcReg),.RegWr(RegWr),.ExtOp(ExtOp),.ALUBSrc(ALUBSrc),.ALUCtrl(ALUCtrl),.MemWrEn(MemWrEn),.MemToReg(MemToReg),.Br(Br),.Br\_E\_L(Br\_E\_L));

    PC pc(.clk(clk),.rst(rst),.PCin(adder\_ans),.PCout(PCout));//PC

    ROM rom(.addr(PCout),.data\_out(Instr));//指令存储器

    ADD add(.a(mux\_pc\_ans),.b(PCout),.ans(adder\_ans));//加法器

    Mux\_PC mux\_pc(.Br(Br),.Br\_E\_L(Br\_E\_L),.Zero(Zero),.data\_in(imm),.ans(mux\_pc\_ans));//信号选择器

    Mux\_5bit mux1(.a(Instr[14:10]),.b(Instr[4:0]),.srcReg(srcReg),.result(mux1\_ans));//2选1选择器1

    Mux\_32bit mux2(.a(busB),.b(imm),.ALUBSrc(ALUBSrc),.result(mux2\_ans));//2选1选择器2

    Mux\_32bit mux3(.a(AluResult),.b(data\_out),.ALUBSrc(MemToReg),.result(mux3\_ans));//2选1选择器3

    Registers registers(.busW(mux3\_ans),.clk(clk),.RegWr(RegWr),.Ra(Instr[9:5]),.Rb(mux1\_ans),.Rw(Instr[4:0]),.busA(busA),.busB(busB));//寄存器

    Ext ext(.DataIn(Instr),.ExtOp(ExtOp),.DataOut(imm));//立即数拓展

    ALU alu(.a(busA),.b(mux2\_ans),.AluCtrl(ALUCtrl),.Zero(Zero),.AluResult(AluResult));//运算器

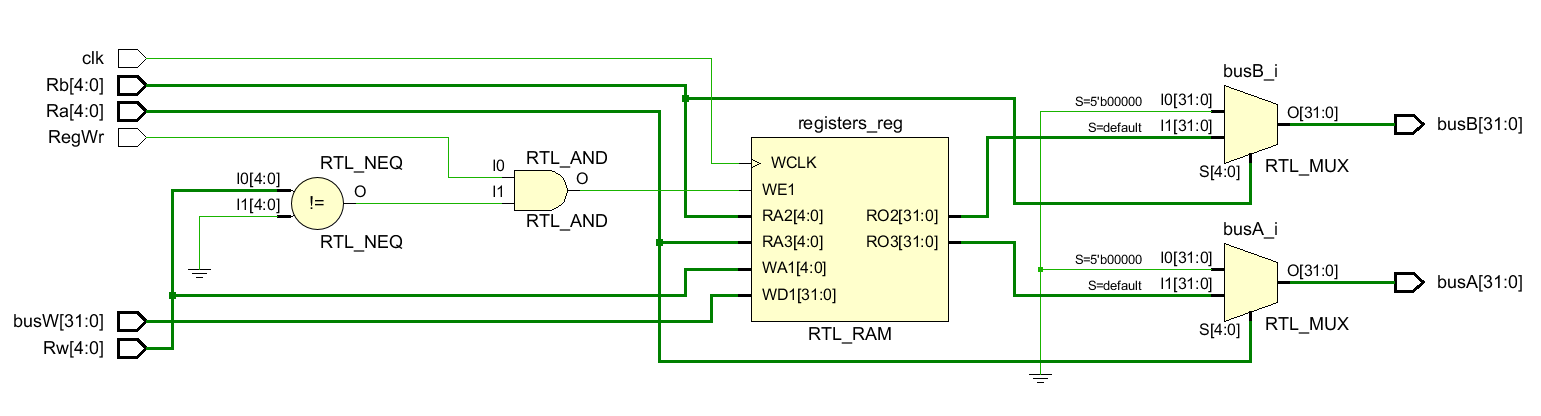
    DataRAM dataRAM(.addr(AluResult),.data\_in(busB),.clk(clk),.MemWrEn(MemWrEn),.data\_out(data\_out));//RAM存储器

endmodule

4.设计调试与结果

1. 通用寄存器堆

在Vivado环境中，输入Verilog HDL设计代码，保存为.v文件。检查无误后，系统生成的RTL原理图如下：



根据该模块的功能，设计如下仿真激励文件，进行功能仿真测试。

module sim\_Registers;

reg [4:0] Ra, Rb, Rw;

reg [31:0] busW;

wire [31:0] busA, busB;

reg clk, RegWr;

Registers dut (.Ra(Ra),.Rb(Rb),.Rw(Rw),.busW(busW),.busA(busA),.busB(busB),.clk(clk),.RegWr(RegWr));

initial begin

    clk = 0;

    RegWr = 0;

    Ra = 0;

    Rb = 0;

    Rw = 0;

    busW = 32'h0;

    RegWr = 1;Rw = 1;busW = 32'h123;#10;

    RegWr = 0;Ra = 1;#10;

    Rb = 1;#10;

    RegWr = 1;Rw = 2;busW = 32'hABC;#10;

    RegWr = 0;Ra = 2;#10;

    Rb = 2;#10;

    Ra = 0;#10;

    RegWr = 1;Rw = 0;busW = 32'hFFF;#10;

    RegWr = 0;Ra = 0;#10;

    Rb = 0;#10;

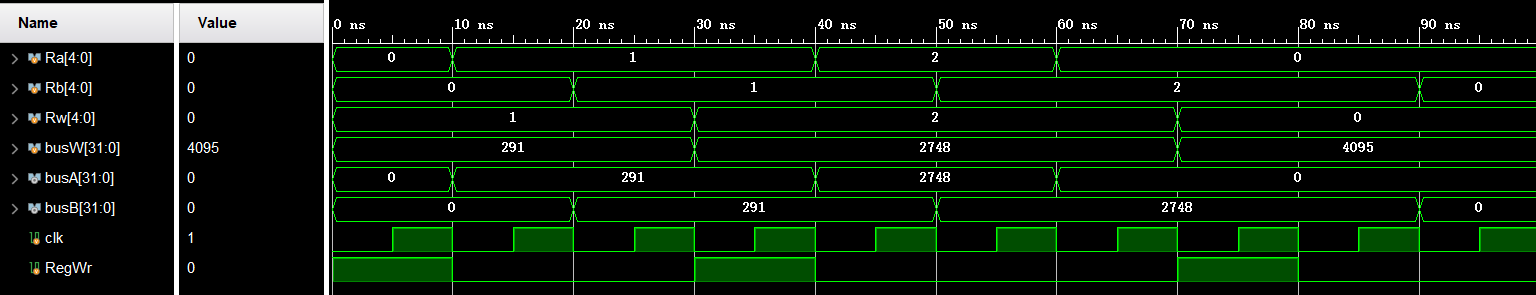
    $finish;

end

always #5 clk = ~clk;

endmodule

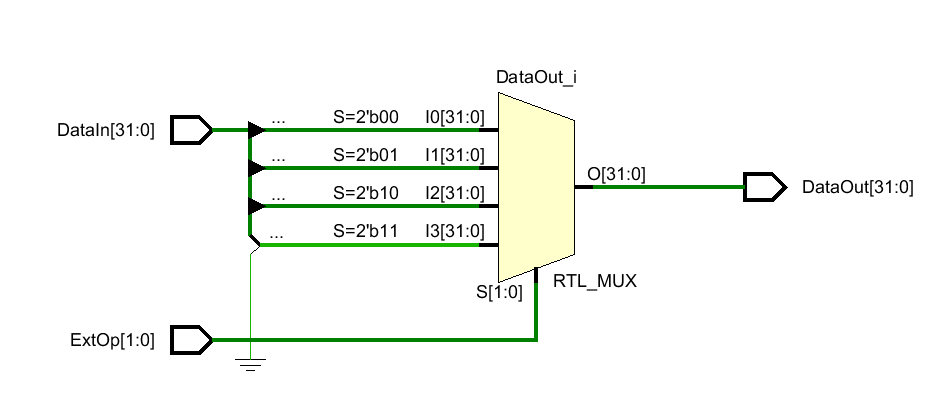
仿真结果如下：



分析仿真波形，可以发现实现了模块具备的功能。

1. 立即数扩展模块

在Vivado环境中，输入Verilog HDL设计代码，保存为.v文件。检查无误后，系统生成的RTL原理图如下：



根据该模块的功能，设计如下仿真激励文件，进行功能仿真测试。

module sim\_Ext;

    reg [31:0] DataIn;

    reg [1:0] ExtOp;

    wire [31:0] DataOut;

    Ext dut (.DataIn(DataIn),.ExtOp(ExtOp),.DataOut(DataOut));

    initial begin

        DataIn = 32'hABCDEF01;

        ExtOp = 2'b00;

        #10;

        DataIn = 32'hABCDEF01;

        ExtOp = 2'b01;

        #10;

        DataIn = 32'hABCDEF01;

        ExtOp = 2'b10;

        #10;

        DataIn = 32'hABCDEF01;

        ExtOp = 2'b11;

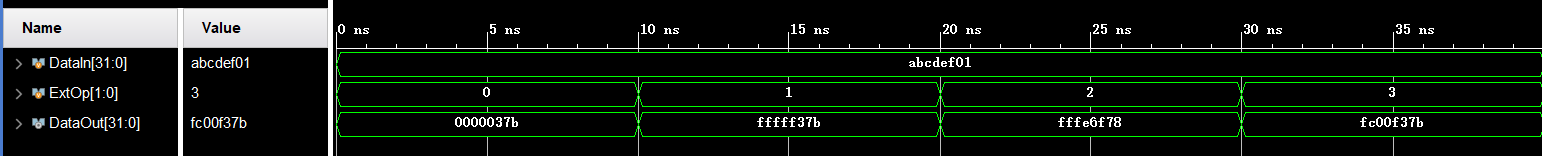
        #10;

        $finish;

    end

endmodule

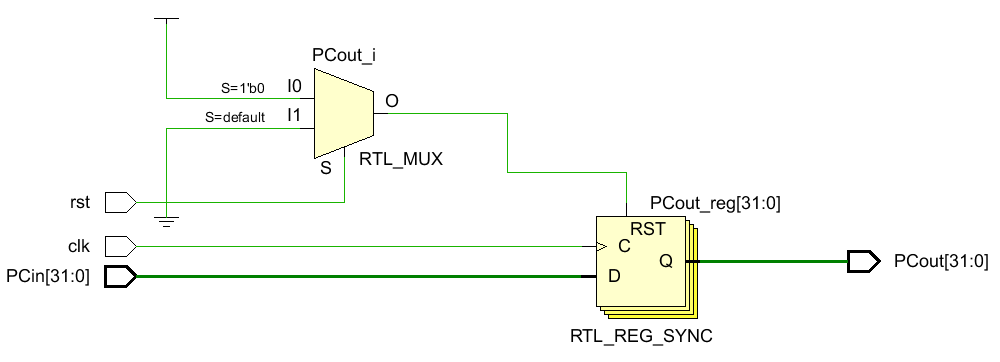
仿真结果如下：



分析仿真波形，可以发现实现了模块具备的功能。

1. 程序计数器PC

在Vivado环境中，输入Verilog HDL设计代码，保存为.v文件。检查无误后，系统生成的RTL原理图如下：



根据该模块的功能，设计如下仿真激励文件，进行功能仿真测试。

module sim\_PC;

    reg rst, clk, pc\_inc;

    reg [31:0] offset;

    wire [31:0] PCdata;

PC dut (.rst(rst),.clk(clk),.offset(offset),.pc\_inc(pc\_inc),.PCdata(PCdata));

initial begin

    clk = 0;

    rst = 1;

    pc\_inc = 0;

    offset = 0;

    #10 rst = 0;

    #10 pc\_inc = 1;

    #10 clk = 1;

    #10 clk = 0;

    #20 pc\_inc = 0;

    #10 offset = 5;

    #10 clk = 1;

    #10 clk = 0;

    #10 clk = 1;

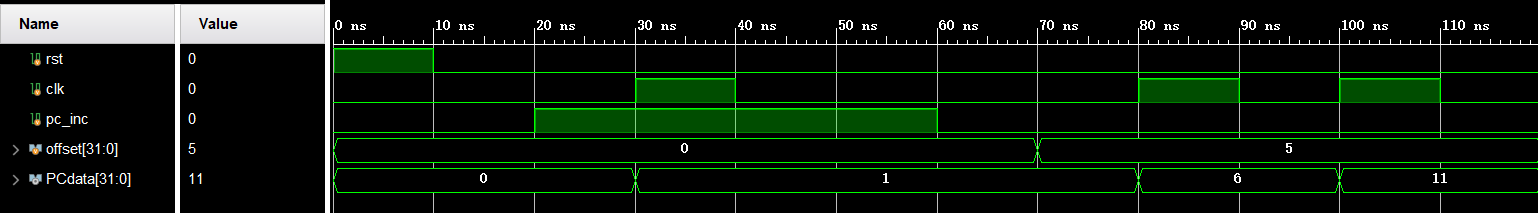
    #10 clk = 0;

    #10 $finish;

end

endmodule

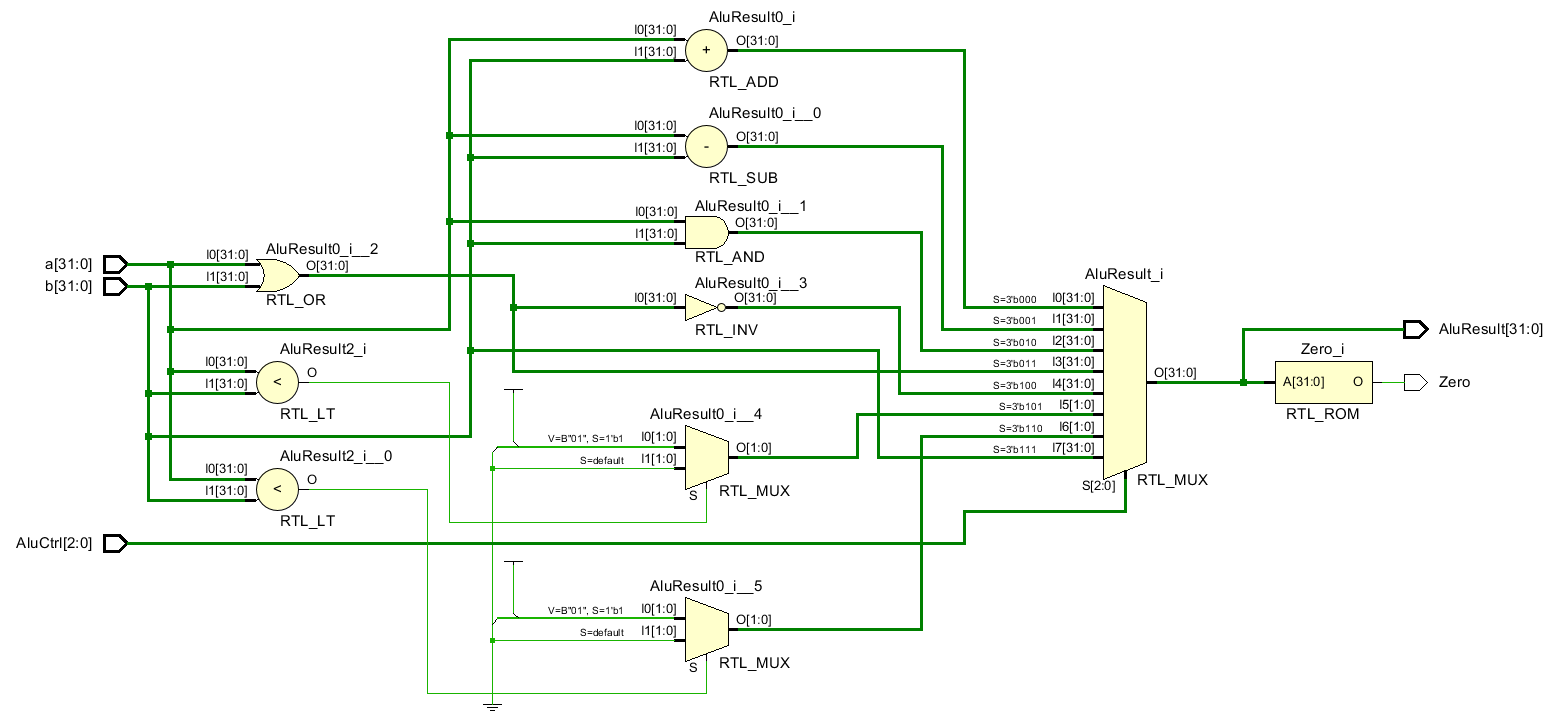
仿真结果如下：



分析仿真波形，可以发现实现了模块具备的功能。

1. 算术逻辑运算单元ALU

在Vivado环境中，输入Verilog HDL设计代码，保存为.v文件。检查无误后，系统生成的RTL原理图如下：



根据该模块的功能，设计如下仿真激励文件，进行功能仿真测试。

module sim\_ALU;

    reg [31:0] a, b;

    reg [2:0] op;

    wire [31:0] AddResult;

    wire Zero;

ALU dut(.a(a), .b(b), .op(op), .AddResult(AddResult), .Zero(Zero));

initial begin

    op = 3'b000; a = 32'h00000001; b = 32'h00000001;

    #100 $display("a+b = %d, Zero = %d", AddResult, Zero);

    op = 3'b001; a = 32'h00000001; b = 32'h00000001;

    #100 $display("a-b = %d, Zero = %d", AddResult, Zero);

    op = 3'b010; a = 32'h0000000F; b = 32'h00000005;

    #100 $display("a&b = %d, Zero = %d", AddResult, Zero);

    op = 3'b011; a = 32'h0000000F; b = 32'h00000005;

    #100 $display("a|b = %d, Zero = %d", AddResult, Zero);

    op = 3'b100; a = 32'h0000000F; b = 32'h00000005;

    #100 $display("~(a|b) = %d, Zero = %d", AddResult, Zero);

    op = 3'b101; a = 32'hFFFFFFF0; b = 32'h0000000F; // -16 < 15

    #100 $display("a<b (signed) = %d, Zero = %d", AddResult, Zero);

    op = 3'b110; a = 32'hFFFFFFF0; b = 32'h0000000F; // 4294967280 > 15

    #100 $display("a<b (unsigned) = %d, Zero = %d", AddResult, Zero);

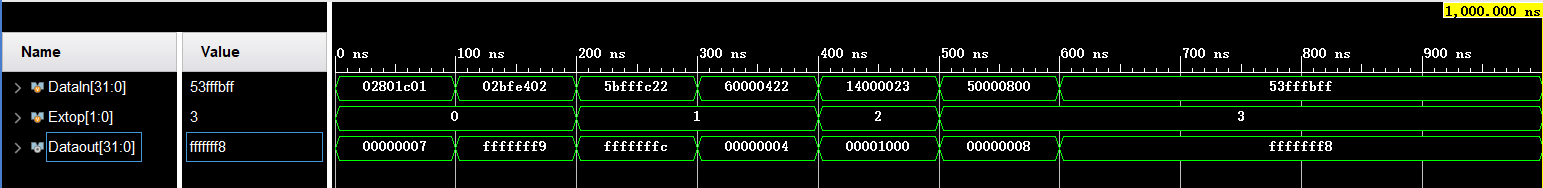
    op = 3'b111; a = 32'h00000000; b = 32'hFFFFFFFF;

    #100 $display("b = %d, Zero = %d", AddResult, Zero);

end

endmodule

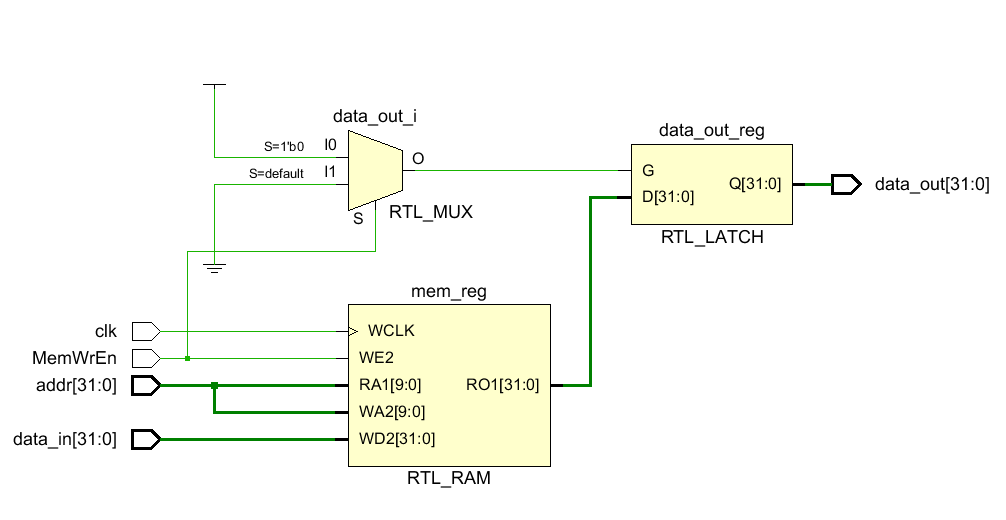
仿真结果如下：



分析仿真波形，可以发现实现了模块具备的功能。

1. 数据存储器

在Vivado环境中，输入Verilog HDL设计代码，保存为.v文件。检查无误后，系统生成的RTL原理图如下：



根据该模块的功能，设计如下仿真激励文件，进行功能仿真测试。

module sim\_RAM;

    reg [31:0] addr;

    reg [31:0] data\_in;

    reg clk;

    reg MemWrEn;

    wire [31:0] data\_out;

RAM dut (.addr(addr),.data\_in(data\_in),.data\_out(data\_out),.clk(clk),.MemWrEn(MemWrEn));

always begin

    #50 clk = ~clk;

end

initial begin

    clk = 0;

    addr = 0;

    data\_in = 0;

    MemWrEn = 0;

    // 写入数据

    addr = 32'h4;

    data\_in = 32'h123;

    MemWrEn = 1;

    #75;

    // 读出数据

    addr = 32'h4;

    MemWrEn = 0;

    #125;

    // 写入数据

    addr = 32'h8;

    data\_in = 32'h456;

    MemWrEn = 1;

    #75;

    // 读出数据

    addr = 32'h8;

    MemWrEn = 0;

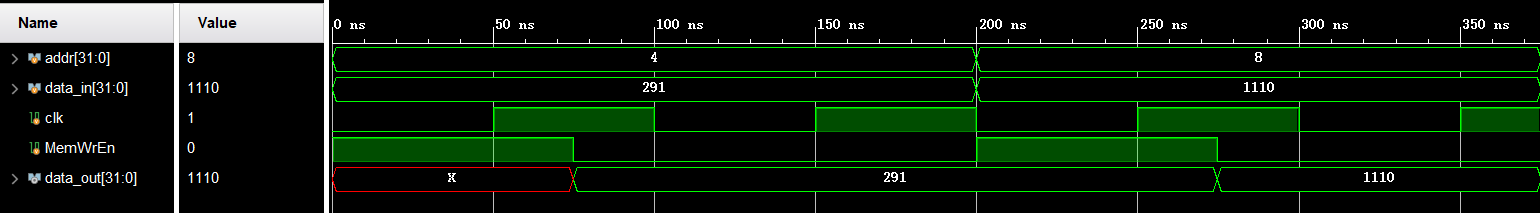
    #100;

    $finish;

end

endmodule

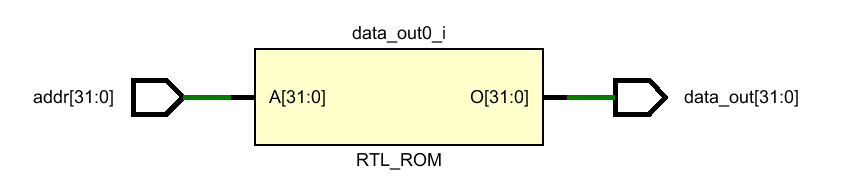
仿真结果如下：



分析仿真波形，可以发现实现了模块具备的功能。

1. 指令存储器

在Vivado环境中，输入Verilog HDL设计代码，保存为.v文件。检查无误后，系统生成的RTL原理图如下：



根据该模块的功能，设计如下仿真激励文件，进行功能仿真测试。

module ROM\_tb;

  reg [31:0] addr;

  wire [31:0] data\_out;

  ROM dut(.addr(addr),.data\_out(data\_out));

  initial begin

    $readmemb("C:/Users/Judy/Desktop/CPU/design/program.txt", dut.rom);

    #10 addr = 0;

    #10 addr = 1;

    #10 addr = 2;

    #10 addr = 2;

    #10 addr = 3;

    #10 addr = 4;

    #10 addr = 5;

    #10 addr = 6;

    #10 addr = 7;

    #10 addr = 8;

    #10 addr = 9;

    #10 addr = 10;

    #10 addr = 11;

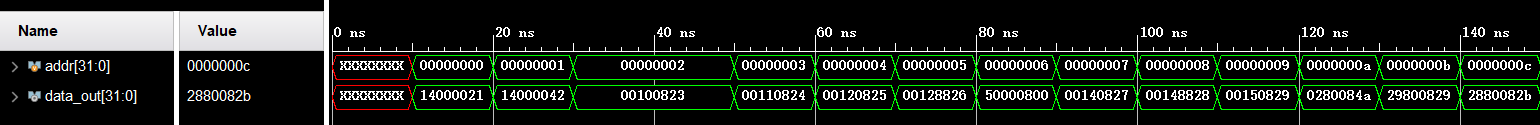
    #10 addr = 12;

    #10 $finish;

  end

endmodule

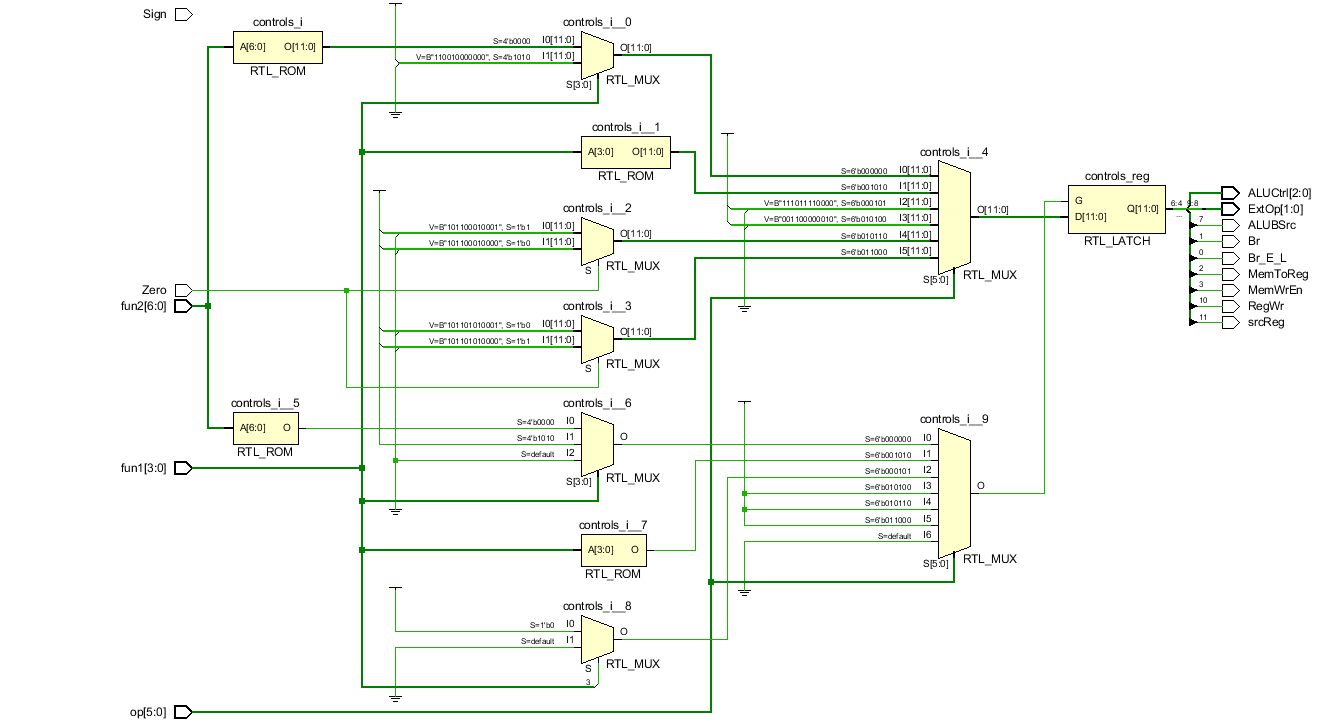
仿真结果如下：



分析仿真波形，可以发现实现了模块具备的功能。

1. 控制器

在Vivado环境中，输入Verilog HDL设计代码，保存为.v文件。检查无误后，系统生成的RTL原理图如下：



根据该模块的功能，设计如下仿真激励文件，进行功能仿真测试。

module sim\_control;

  reg [16:0] Opcode;

  wire RegWr, ALUBSrc, MemWrEn, MemToReg, srcReg;

  wire [2:0] AluCtrl;

  wire [1:0] ExtOp;

  Control control(.Opcode(Opcode),.RegWr(RegWr),.AluCtrl(AluCtrl),.ALUBSrc(ALUBSrc),.ExtOp(ExtOp),.MemWrEn(MemWrEn),.MemToReg(MemToReg),.srcReg(srcReg));

  initial begin

    Opcode = 17'b00010100000000000;

    #10;

    Opcode = 17'b00010101111000000;

    #10;

    Opcode = 17'b000000\_0000\_01\_00000;

    #10;

    Opcode = 17'b000000\_0000\_01\_00101;

    #10;

    Opcode = 17'b000000\_0000\_01\_00100;

    #10;

    Opcode = 17'b001010\_0110\_0000000;

    #10;

    Opcode = 17'b001010\_0010\_0000000;

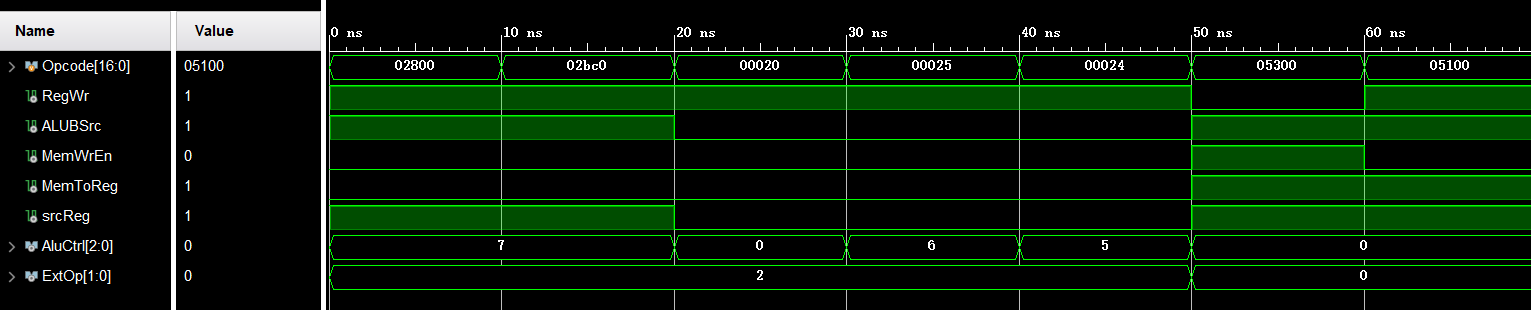
    #10;

    $finish;

  end

endmodule

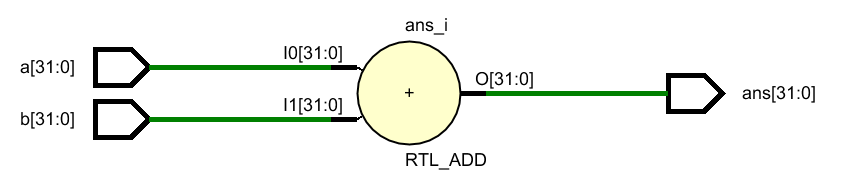
仿真结果如下：



分析仿真波形，可以发现实现了模块具备的功能。

1. 加法器

在Vivado环境中，输入Verilog HDL设计代码，保存为.v文件。检查无误后，系统生成的RTL原理图如下：



根据该模块的功能，设计如下仿真激励文件，进行功能仿真测试。

module sim\_ADD;

  reg [31:0] a;

  reg [31:0] b;

  wire [31:0] ans;

  ADD dut (.a(a),.b(b),.ans(ans));

  initial begin

    a = 10;

    b = 20;

    #10;

  end

endmodule

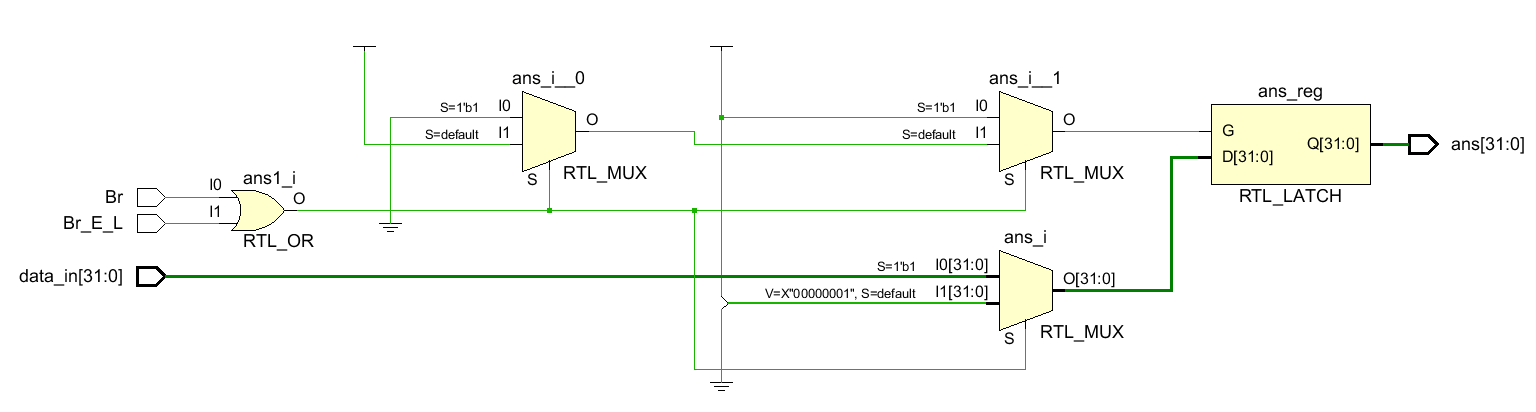
仿真结果如下：



分析仿真波形，可以发现实现了模块具备的功能。

1. 分支选择器

在Vivado环境中，输入Verilog HDL设计代码，保存为.v文件。检查无误后，系统生成的RTL原理图如下：



根据该模块的功能，设计如下仿真激励文件，进行功能仿真测试。

module sim\_Mux\_PC;

    reg Br;

    reg Br\_E\_L;

    reg [31:0] data\_in;

    wire [31:0] ans;

    Mux\_PC dut (.Br(Br),.Br\_E\_L(Br\_E\_L),.data\_in(data\_in),.ans(ans));

    initial begin

        Br = 0;

        Br\_E\_L = 0;

        data\_in = 32'b10101010101010101010101010101010;

        #10;

        Br = 1;

        Br\_E\_L = 0;

        data\_in = 32'b11001100110011001100110011001100;

        #10;

        Br = 0;

        Br\_E\_L = 1;

        data\_in = 32'b11110000111100001111000011110000;

        #10;

        Br = 1;

        Br\_E\_L = 1;

        data\_in = 32'b01010101010101010101010101010101;

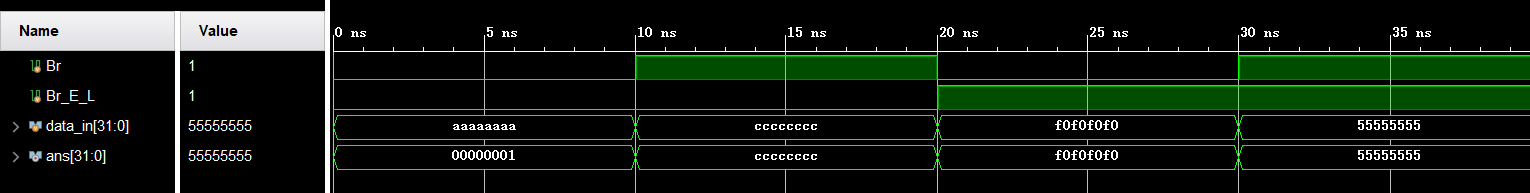
        #10;

        $finish;

    end

endmodule

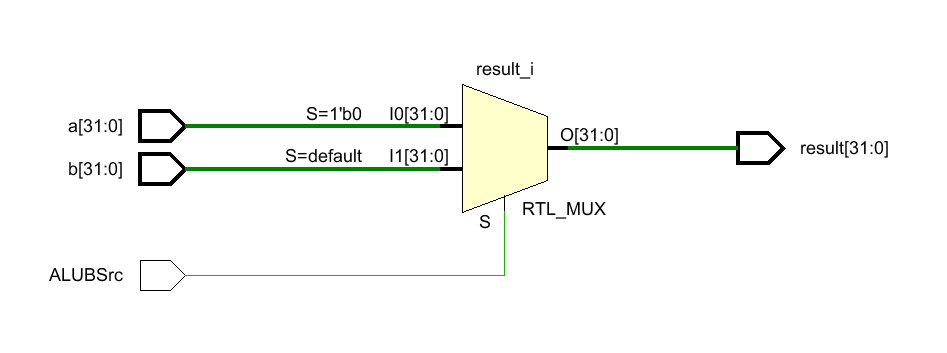
仿真结果如下：



分析仿真波形，可以发现实现了模块具备的功能。

1. 32位多路复用器

在Vivado环境中，输入Verilog HDL设计代码，保存为.v文件。检查无误后，系统生成的RTL原理图如下：



根据该模块的功能，设计如下仿真激励文件，进行功能仿真测试。

module sim\_Mux\_32bit;

reg [31:0] a;

  reg [31:0] b;

  reg ALUBSrc;

  wire [31:0] result;

  Mux\_32bit dut (.a(a),.b(b),.ALUBSrc(ALUBSrc),.result(result));

  initial begin

    a = 32'h01234567;

    b = 32'h89ABCDEF;

    ALUBSrc = 1'b0;

    #10 ALUBSrc = 1'b1;

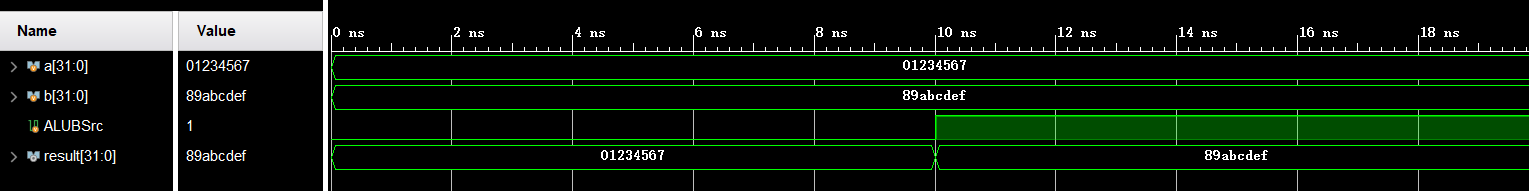
    #10;

    $finish;

  end

endmodule

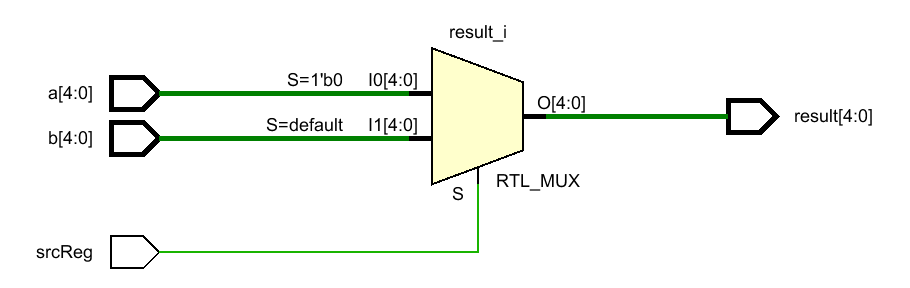
仿真结果如下：



分析仿真波形，可以发现实现了模块具备的功能。

1. 5位多路复用器

在Vivado环境中，输入Verilog HDL设计代码，保存为.v文件。检查无误后，系统生成的RTL原理图如下：



根据该模块的功能，设计如下仿真激励文件，进行功能仿真测试。

module sim\_Mux\_5bit;

    reg [4:0] a;

    reg [4:0] b;

    reg srcReg;

    wire [4:0] result;

    Mux\_5bit uut (.a(a),.b(b),.srcReg(srcReg),.result(result));

    initial begin

        a = 5'b01101;

        b = 5'b11010;

        srcReg = 0;

        #10;

        srcReg = 1;

        #10;

        a = 5'b10101;

        b = 5'b00110;

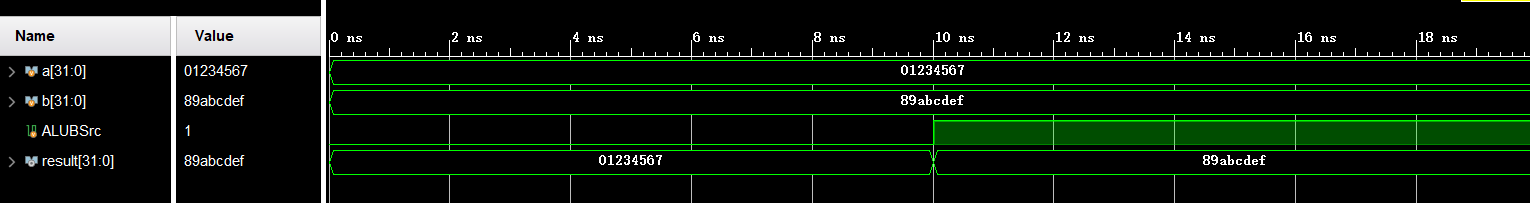
        #10;

        $finish;

    end

endmodule

仿真结果如下：



分析仿真波形，可以发现实现了模块具备的功能。

1. 单周期LA32R CPU电路
   1. 控制信号的取指分析

本次设计的指令系统包括14条指令，控制信号的取值分析如下：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 序号 | 指令 | 6位 | 1位 | 3位 | 7位 | | 5位 | 5位 | 5位 |
| 31 … 26 | 25 | 24… 22 | 21 20 | 19…15 | 14…10 | 9… 5 | 4… 0 |
| 1 | LUI12I.W | 000101 | 0 | si20 | | | | | rd |
| 2 | ADD.W | 000000 | 0000 | | 01 | 00000 | rk | rj | rd |
| 3 | SUB.W | 000000 | 0000 | | 01 | 00010 | rk | rj | rd |
| 4 | SLT | 000000 | 0000 | | 01 | 00100 | rk | rj | rd |
| 5 | SLTU | 000000 | 0000 | | 01 | 00101 | rk | rj | rd |
| 6 | NOR | 000000 | 0000 | | 01 | 01000 | rk | rj | rd |
| 7 | AND | 000000 | 0000 | | 01 | 01001 | rk | rj | rd |
| 8 | OR | 000000 | 0000 | | 01 | 01010 | rk | rj | rd |
| 9 | ADDI.W | 000000 | 1010 | | si12 | | | rj | rd |
| 10 | LD. W | 001010 | 0010 | | si12 | | | rj | rd |
| 11 | ST.W | 001010 | 0110 | | si12 | | | rj | rd |
| 12 | B | 010100 | offs[15:0] | | | | | offs [25:16] | |
| 13 | BEQ | 010110 | offs[15:0] | | | | | rj | rd |
| 14 | BLT | 011000 | offs [15:0] | | | | | rj | rd |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 输出 | | | | | | | | |
| 指令 | AluCtrl[2:0] | RegWr | ALUBSrc | ExtOp | srcReg | MemWrEn | MemToReg | Br | Br\_E\_L |
| LUI12I.W | 111 | 1 | 0 | 10 | 0 | 0 | 0 | 0 | 0 |
| ADD.W | 000 | 1 | 0 | X | 0 | 0 | 0 | 0 | 0 |
| SUB.W | 001 | 1 | 0 | X | 0 | 0 | 0 | 0 | 0 |
| SLT | 101 | 1 | 0 | X | 0 | 0 | 0 | 0 | 0 |
| SLTU | 110 | 1 | 0 | X | 0 | 0 | 0 | 0 | 0 |
| NOR | 100 | 1 | 0 | X | 0 | 0 | 0 | 0 | 0 |
| AND | 010 | 1 | 0 | X | 0 | 0 | 0 | 0 | 0 |
| OR | 011 | 1 | 0 | X | 0 | 0 | 0 | 0 | 0 |
| ADDI.W | 000 | 1 | 1 | 00 | 0 | 0 | 0 | 0 | 0 |
| LD. W | 000 | 1 | 1 | 00 | 0 | 0 | 1 | 0 | 0 |
| ST.W | 000 | 0 | 1 | 00 | 0 | 1 | 0 | 0 | 0 |
| B | X | 0 | X | 11 | X | 0 | X | 1 | 0 |
| BEQ | 001 | 0 | 0 | 01 | 0 | 0 | X | 0 | 1 |
| BLT | 101 | 0 | 0 | 01 | 0 | 0 | X | 0 | 1 |

1. 应用程序

1.用所设计的14条指令编写的应用程序如下，该测试程序完成14个全部功能。

|  |  |
| --- | --- |
| 应用程序助记符代码 | 应用程序机器代码 |
| LU12I.W 1,1 | 000101\_000000000000000000001\_00001 |
| LU12I.W 2,2 | 000101\_000000000000000000010\_00010 |
| ADD.W 3,1,2 | 000000\_0000\_0100000\_00010\_00001\_00011 |
| SUB.W 4,1,2 | 000000\_0000\_0100010\_00010\_00001\_00100 |
| SLT 5,1,2 | 000000\_0000\_0100100\_00010\_00001\_00101 |
| SLTU 6,1,2 | 000000\_0000\_0100101\_00010\_00001\_00110 |
|  | 010100\_0000000000000010\_0000000000 |
| NOR 7,1,2 | 000000\_0000\_0101000\_00010\_00001\_00111 |
| AND 8,1,2 | 000000\_0000\_0101001\_00010\_00001\_01000 |
| OR 9,1,2 | 000000\_0000\_0101010\_00010\_00001\_01001 |
| ADDI.W 10,2,2 | 000000\_1010\_000000000010\_00010\_01010 |
| LD.W 9,1,2 | 001010\_0110\_000000000010\_00001\_01001 |
| ST.W 11,1,2 | 001010\_0010\_000000000010\_00001\_01011 |

2.用所设计的2条指令编写的应用程序如下，该测试程序完成字符串连接功能。

|  |  |
| --- | --- |
| 应用程序助记符代码 | 应用程序机器代码 |
| LU12I.W 1,97 | 000101\_000000000000001100001\_00001 |
| ST.W 1,0,0 | 001010\_0110\_000000000000\_00000\_00001 |
| LU12I.W 2,98 | 000101\_000000000000001100010\_00010 |
| ST.W 2,0,1 | 001010\_0110\_000000000001\_00000\_00010 |
| LU12I.W 3,99 | 000101\_000000000000001100011\_00011 |
| ST.W 3,0,2 | 001010\_0110\_000000000010\_00000\_00011 |
| LU12I.W 4,36 | 000101\_000000000000000100100\_00100 |
| ST.W 4,0,3 | 001010\_0110\_000000000011\_00000\_00100 |
| LU12I.W 5,50 | 000101\_000000000000000110010\_00101 |
| ST.W 5,0,8 | 001010\_0110\_000000001000\_00000\_00101 |
| LU12I.W 6,51 | 000101\_000000000000000110011\_00110 |
| ST.W 6,0,9 | 001010\_0110\_000000001001\_00000\_00110 |
| LU12I.W 7,52 | 000101\_000000000000000110100\_00111 |
| ST.W 7,0,10 | 001010\_0110\_000000001010\_00000\_00111 |
| LU12I.W 8,53 | 000101\_000000000000000110101\_01000 |
| ST.W 8,0,11 | 001010\_0110\_000000001011\_00000\_01000 |
| LU12I.W 9,36 | 000101\_000000000000000100100\_01001 |
| ST.W 9,0,12 | 001010\_0110\_000000001100\_00000\_01001 |
| ST.W 1,0,16 | 001010\_0110\_000000010000\_00000\_00001 |
| ST.W 2,0,17 | 001010\_0110\_000000010001\_00000\_00010 |
| ST.W 3,0,18 | 001010\_0110\_000000010010\_00000\_00011 |
| ST.W 5,0,19 | 001010\_0110\_000000010011\_00000\_00101 |
| ST.W 6,0,20 | 001010\_0110\_000000010100\_00000\_00110 |
| ST.W 7,0,21 | 001010\_0110\_000000010101\_00000\_00111 |
| ST.W 8,0,22 | 001010\_0110\_000000010110\_00000\_01000 |

1. 应用程序与模型机联调过程及结果
   * 1. 仿真测试文件内容：

module sim\_CPU\_top;

    reg clk;

    reg rst;

    wire [31:0] Instr;

    wire [1:0] ExtOp;

    wire [2:0] ALUCtrl;

    wire srcReg, RegWr, ALUBSrc, MemWrEn, MemToReg,Br,Br\_E\_L;

    wire [4:0] mux1\_ans;

    wire [31:0] busA;

    wire [31:0] busB;

    wire [31:0] imm;

    wire [31:0] mux2\_ans;

    wire Zero;

    wire [31:0] AluResult;

    wire [31:0] data\_out;

    wire [31:0] mux3\_ans;

    CPU\_top uut(.Instr(Instr), .clk(clk),.rst(rst), .srcReg(srcReg), .RegWr(RegWr),

          .ExtOp(ExtOp), .ALUBSrc(ALUBSrc), .ALUCtrl(ALUCtrl), .MemWrEn(MemWrEn),

          .MemToReg(MemToReg),.Br(Br),.Br\_E\_L(Br\_E\_L),.mux1\_ans(mux1\_ans), .busA(busA), .busB(busB),

          .imm(imm), .mux2\_ans(mux2\_ans),.Zero(Zero),.AluResult(AluResult),

          .data\_out(data\_out), .mux3\_ans(mux3\_ans));

    initial begin

        clk = 0;

        rst = 0;

    end

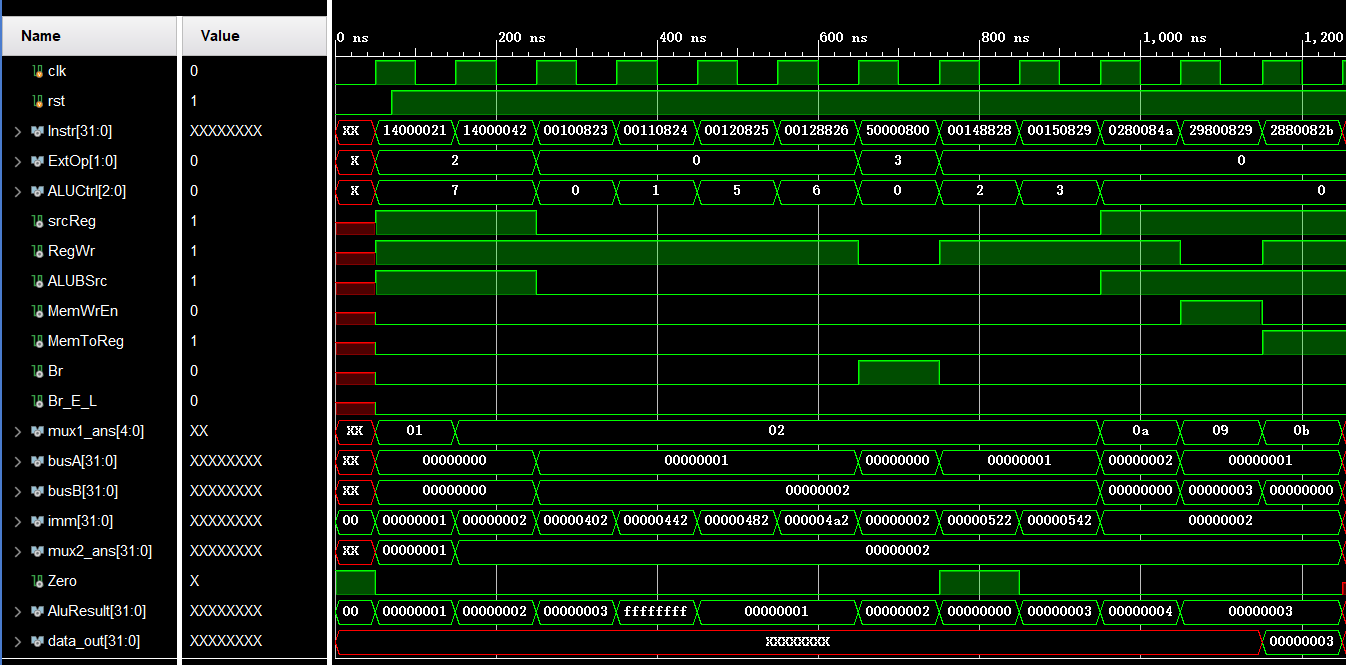
    always #70 rst = 1;

    always #50 clk = ~clk;

endmodule

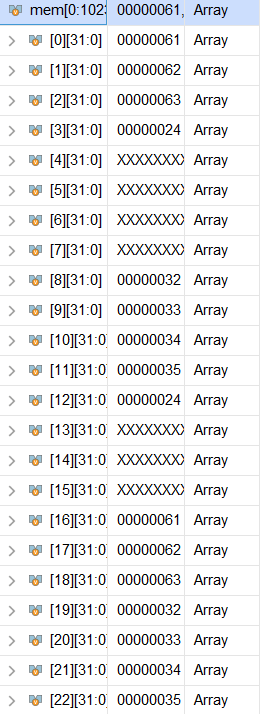
仿真测试文件都一样只有program里面的指令不一样。

* + 1. 1.将一号寄存器置为1；二号寄存器置为2；一号寄存器加二号寄存器为3后赋到三号寄存器；一号寄存器减二号寄存器为负后赋到四号寄存器；一号寄存器与二号寄存器进行有符号位比较后为1赋到五号寄存器；一号寄存器与二号寄存器进行无符号位比较后为1赋到六号寄存器；进行B指令，跳转到and指令；一号寄存器与上二号寄存器后为0赋到七号寄存器；一号寄存器或上二号寄存器后为3赋到八号寄存器；之后进行ld和st指令。由波形图可以看出完成并且正确的执行了上述代码。



2.仿真波形截图其实没什么用，直接看Object.

0号地址到3号地址存进去了”abc$”，8号地址到12号地址存进去了”2345”，16号地址到22号地址存进去了”abc2345”，所以结果正确。



1. 遇到问题与解决方法

* 问题1：ALU可以正确读取指令，但无法输出结果。
* 解决方法：

检查ALU的输入和输出端口：首先，检查ALU的输入和输出端口是否正确连接。确保指令的输入信号正确传递给ALU，而输出信号能够被正确连接到目标寄存器或其他模块。

检查ALU的控制信号：ALU的功能和操作由控制信号决定。确保控制信号正确设置，以使ALU执行所需的操作。例如，检查运算控制信号是否正确设置为加法、减法、逻辑与或逻辑或等。

检查ALU的运算实现：ALU的运算实现需要根据指令的操作码执行正确的操作。检查ALU的运算电路是否正确设计，以确保能够执行所需的算术和逻辑操作。

检查ALU的状态和时序：ALU的状态和时序控制对于正确执行操作非常重要。确保ALU的状态转换和时序控制逻辑正确设计，以确保操作在正确的时间和顺序下进行。

调试和验证：运行仿真或实际的实验，检查ALU的行为和输出结果。通过观察仿真波形或实验结果，可以确定ALU无法输出结果的具体原因。在发现问题后，根据问题的具体情况进行逐步调试和排除故障。

* 问题2：要使0寄存器恒为零，可以考虑以下几种方法
* 解决方法：

初始化：在计算机系统启动或复位时，将0寄存器的初始值设置为零。这可以通过硬件电路或软件初始化程序来实现。

写入控制逻辑：在设计中，可以使用写入控制逻辑来保证0寄存器只能被写入零值。在写入操作时，检测写入数据是否为零，如果不是零，则不执行写入操作。

屏蔽写入信号：通过修改控制逻辑，禁止对0寄存器进行写入操作。这样，即使有写入操作指令，也无法对0寄存器进行写入，从而保持其值为零。

强制清零：在需要清零操作时，可以通过特定的指令或控制信号强制将0寄存器的值设置为零。这样可以确保在需要时随时清零。

三、设计总结体会

* 撰写各类报告与沟通交流：在课程设计过程中，我们学会了如何撰写实验报告、设计报告和总结报告。通过撰写这些报告，我们能够系统地记录实验过程、设计思路、仿真数据和结果，并对所学原理和专业知识进行应用和分析。这帮助我们更好地与同学和老师进行有效的沟通和交流，向他们阐述和解释相关问题，并接受他们的反馈和建议。
* 使用技术和工具进行模拟与分析：为了解决模型机的设计问题，我们学会了选择和合理使用相关技术、软硬件和系统资源，以及现代工程研发工具。这些工具和资源包括仿真软件、开发环境、设计工具和分析工具等。通过灵活运用这些工具，我们能够进行模拟和分析，验证设计的正确性和可行性，发现潜在问题并加以改进。
* 团队合作和协调：在课程设计任务要求中，组织或选择团队并在其中承担相应角色是必要的。我们学会了如何与团队成员合作，根据各自的能力和任务要求分配工作，并协调或指挥团队开展工作。这需要良好的沟通能力、领导能力和团队合作精神，以确保整个团队能够高效协作、充分发挥各自的优势，最终完成课程设计任务。

四、参考文献

[1]龙芯架构32位精简版参考手册:V1.02，龙芯中科技术股份有限公司.

[2]白中英, 戴志涛. 计算机组成原理.第六版立体化教材[M]. 科学出版社, 2019.

[3]计算机组成原理课程设计指导书.LoongArch之LA32R版，南通大学信息科学技术学院计算机组成原理课程组