**同济大学计算机系**

**计算机组成原理实验报告**

****

**学 号 1952650**

**姓 名 陈子翔**

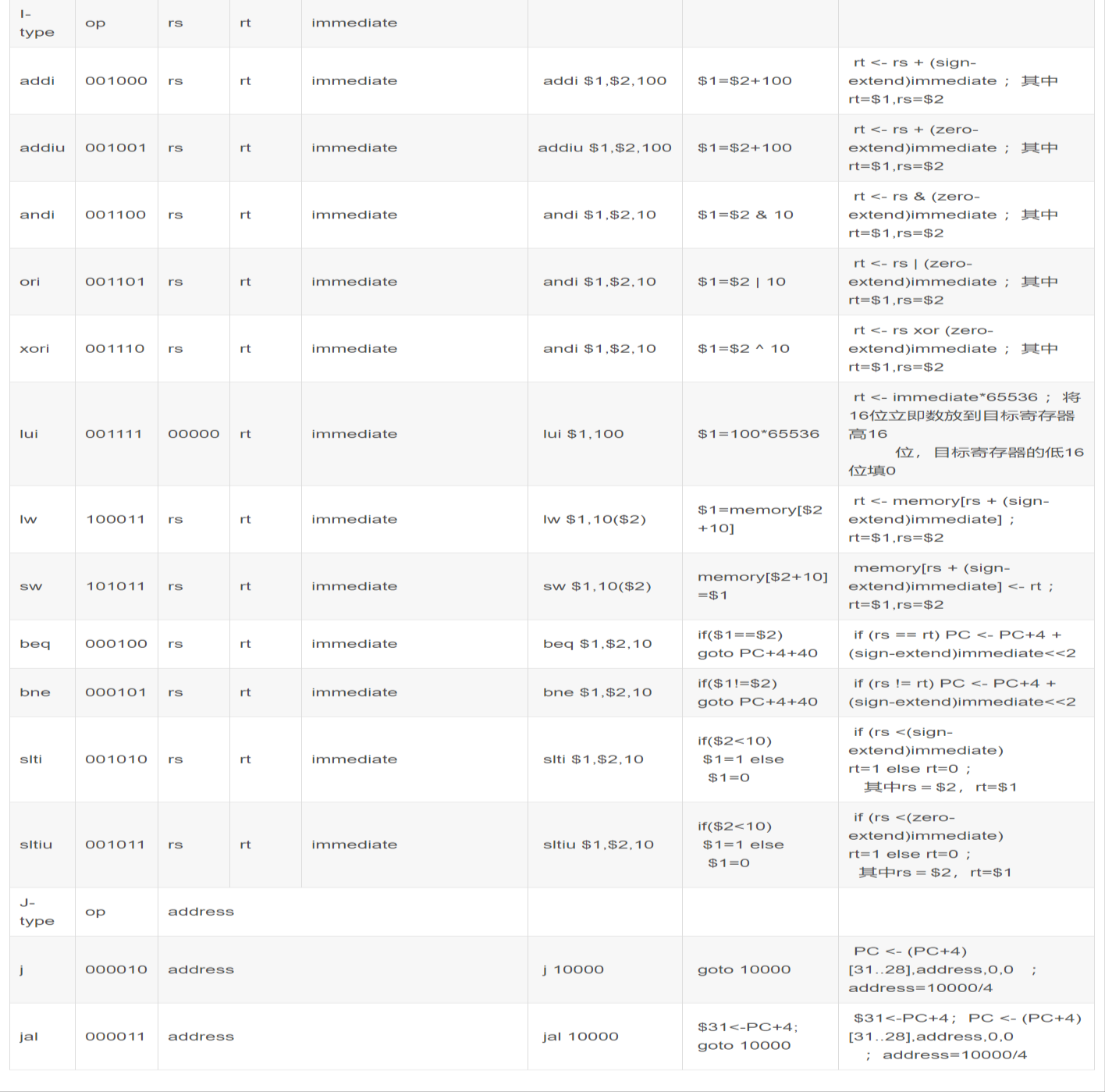
**专 业 信息安全**

**授课老师 郝老师**

1. 实验内容

使用Verilog HDL语言实现31条MIPS指令的CPU的设计和仿真，使其实现以下指令。





1. 实验目的
2. 深入掌握CPU的构成及工作原理。
3. 设计31条指令的CPU的数据通路及控制器。
4. 使用Verilog HDL设计实现31条指令的CPU下板运行。
5. 实验原理

## 3.1实验原理

中央处理器（CPU）由运算器和控制器组成。其中，控制器的功能是负责协调并控制计算机各部件执行程序的指令序列，包括取指令、分析指令和执行指令；运算器的功能是对数据进行加工。CPU的具体功能包括：

1. 指令控制。完成取指令、分析指令和执行指令的操作，即程序的顺序控制。
2. 操作控制。一条指令的功能往往由若干操作信号的组合来实现。CPU管理并产生由内存取出的每条指令的操作信号，把各种操作信号送往相应的部件，从而控制这些部件按指令的要求进行动作。
3. 时间控制。对各种操作加以时间上的控制。时间控制要为每条指令按时间顺序提供应有的控制信号。
4. 数据加工。对数据进行算术和逻辑运算。
5. 中断处理。对计算机运行过程中出现的异常情况和特殊请求进行处理。

单周期CPU是对所有指令都选用相同的执行时间来完成，称为单指令周期方案。此时每条指令都在固定的时钟周期内完成，指令之间串行执行，即下一条指令只能在前一条指令执行结束后才能启动。在单周期处理器中，一条指令执行过程中数据通路的任何资源都不能被重复使用，因此，任何需要被多次使用的资源都需要设置多个。

CPU在执行指令的不同阶段所涉及的数据流有所不同，可分为以下几个步骤：

1. 取指周期：根据程序计数器PC中的内容从主存中取出一条指令并存放在指令寄存器IR中。PC存放的是指令的地址，根据此地址从内存单元中取出的是指令，取指令的同时，PC通过自增到存储下一条指令的地址。
2. 指令译码：对在取指周期中得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。
3. 间址周期：取操作数有效地址，将指令中的地址码送到地址寄存器MAR并送至地址总线，此后控制单元CU向存储器发读命令，以获取有效地址并存至数据寄存器MDR。
4. 执行周期：根据指令寄存器IR中的指令字的操作码核操作数通过算数逻辑单元ALU操作产生执行结果。最后根据指令要求将执行结果或访问存储器得到的数据写回相应的寄存器。

## 3.2指令格式

一条指令就是机器语言的一个语句，它是一组有意义的二进制代码。一条指令通常包括操作码字段和地址码字段两部分：

|  |  |
| --- | --- |
| 操作码字段 | 地址码字段 |

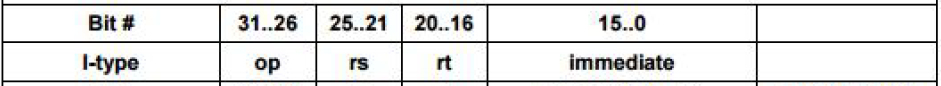
其中，操作码字段指出指令中该指令应该执行什么性质的操作和具有何种功能。操作码是识别指令、了解指令功能及区分操作数地址内容的组成和使用方法等的关键信息。

MIPS框架下的CPU具有以下三种指令类型：

1．R-type



2．I-type



3．J-type



其中：

op：31位~26位，6位操作码，决定执行何种指令，R型指令的操作码均为000000，其余指令操作码各不相同。

rs：5位地址码，代表第一个源操作数地址，只读。

rt：5位地址码，代表第二个源操作数地址，可读可写。

rd：5位地址码，代表目的操作数地址，只写。

shamt：5位数据，专用于移位指令中代表位移量。

func：6位功能码，与操作码配合使用，在R类指令中，操作码均为0，用不同的功能码区分执行何种指令。

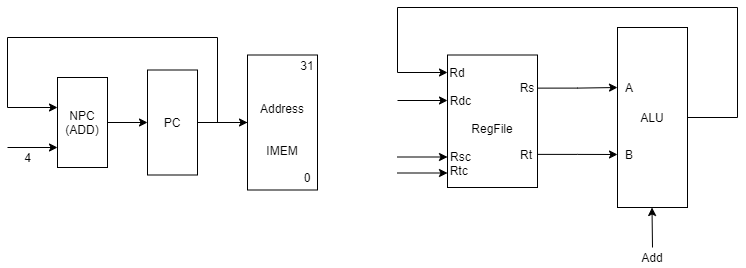
immediate：16位立即数，用作无符号的逻辑操作数、有符号的算数操作数、数据加载/数据保存指令的数据地址字节偏移量和分支指令中相对程序计数器的有符号偏移量。

index：26位地址码，用于J类指令中作pc跳转的地址。

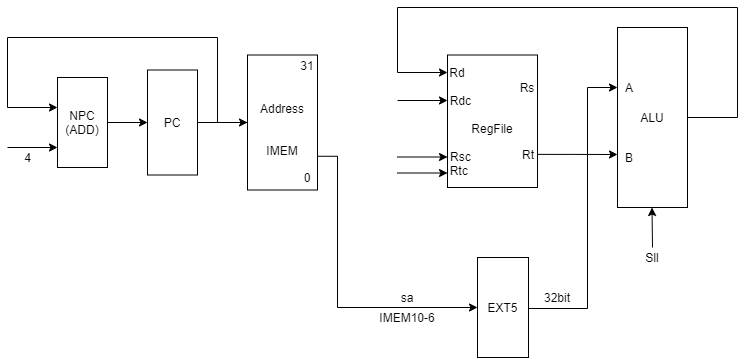
## 3.3数据通路设计

对于相似的指令，相同的数据通路图不再重复画出。

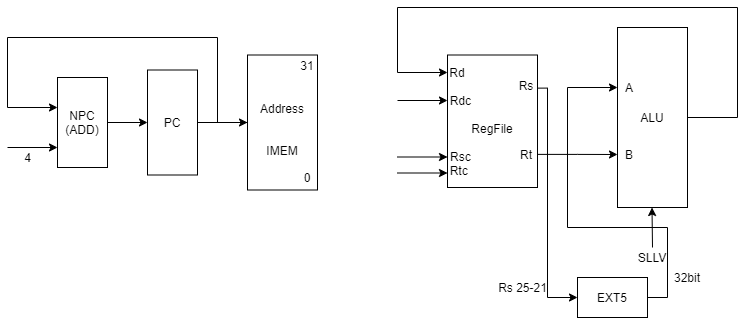
1. ADD/ADDU/SUB/SUBU/AND/OR/XOR/NOR/SLT/SLTU



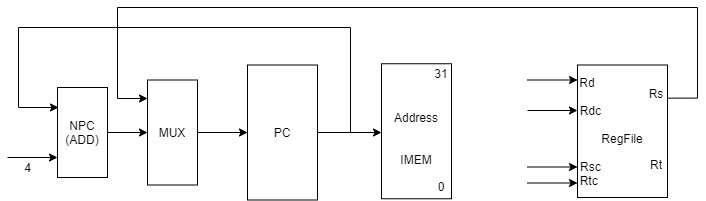
1. SLL/SRL/SRA



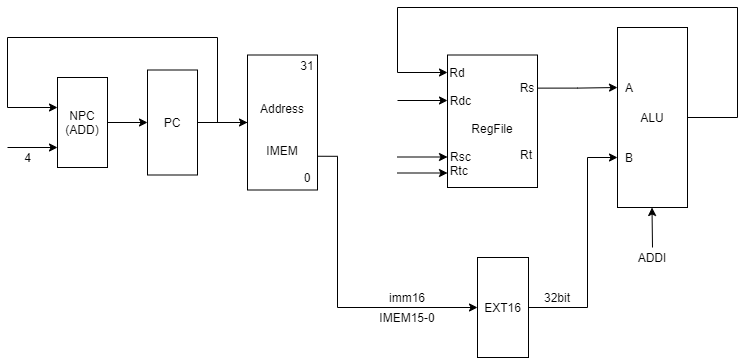
1. SLLV/SRLV/SRAV



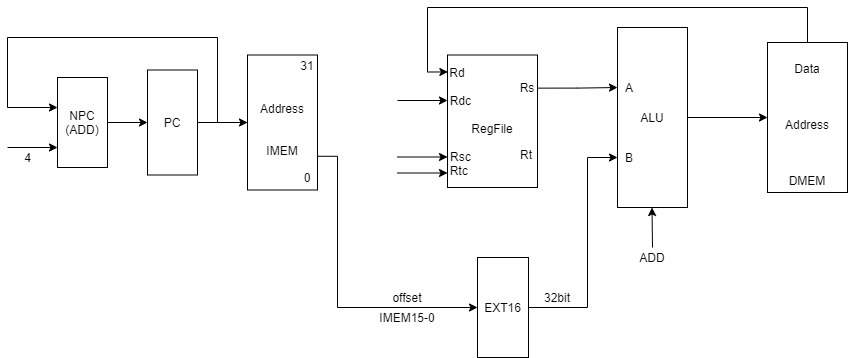
1. JR



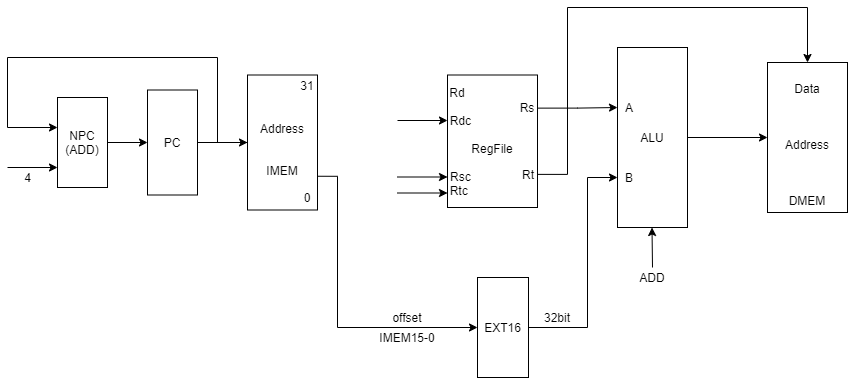
1. ADDI/ADDIU/ANDI/ORI/XORI



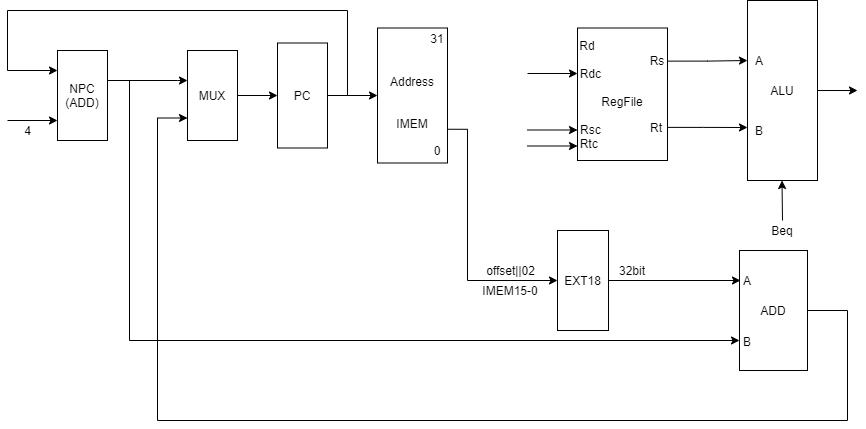
1. LW



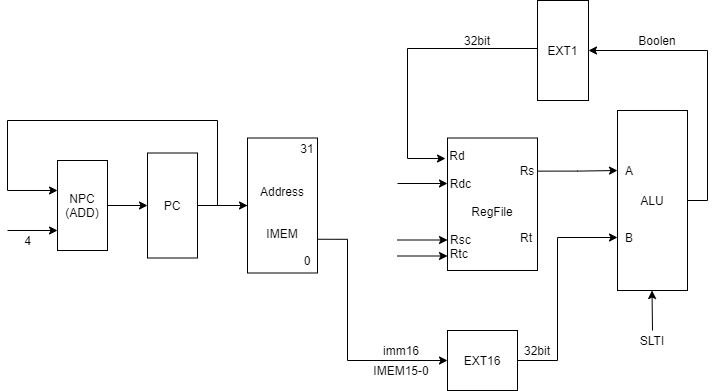
1. SW



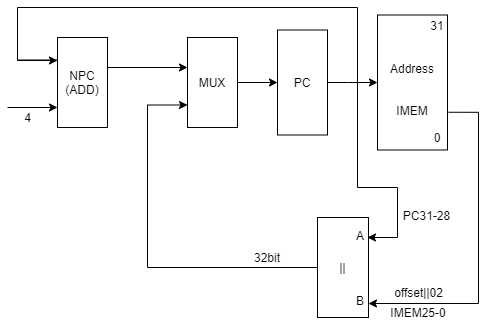
1. BEQ/BNE



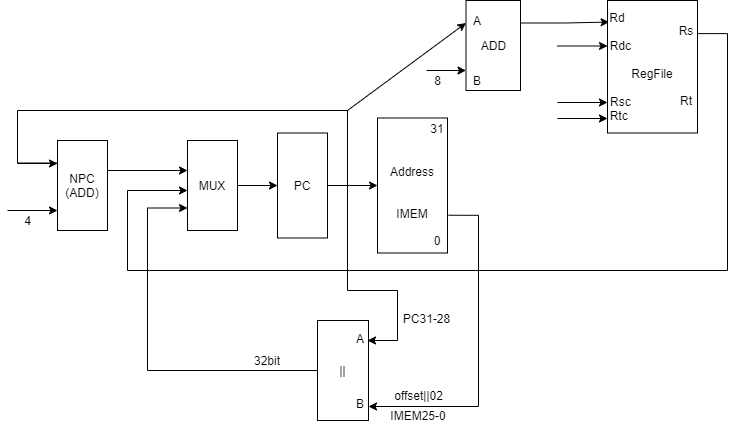
1. SLTI/SLTIU/LUI



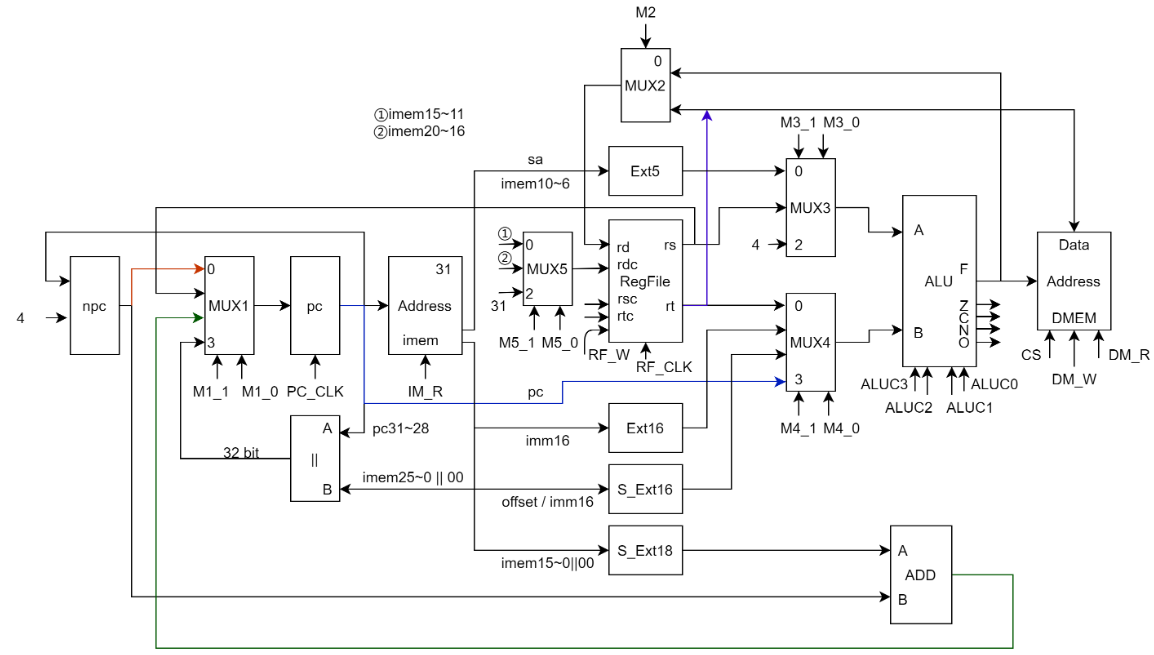
1. J



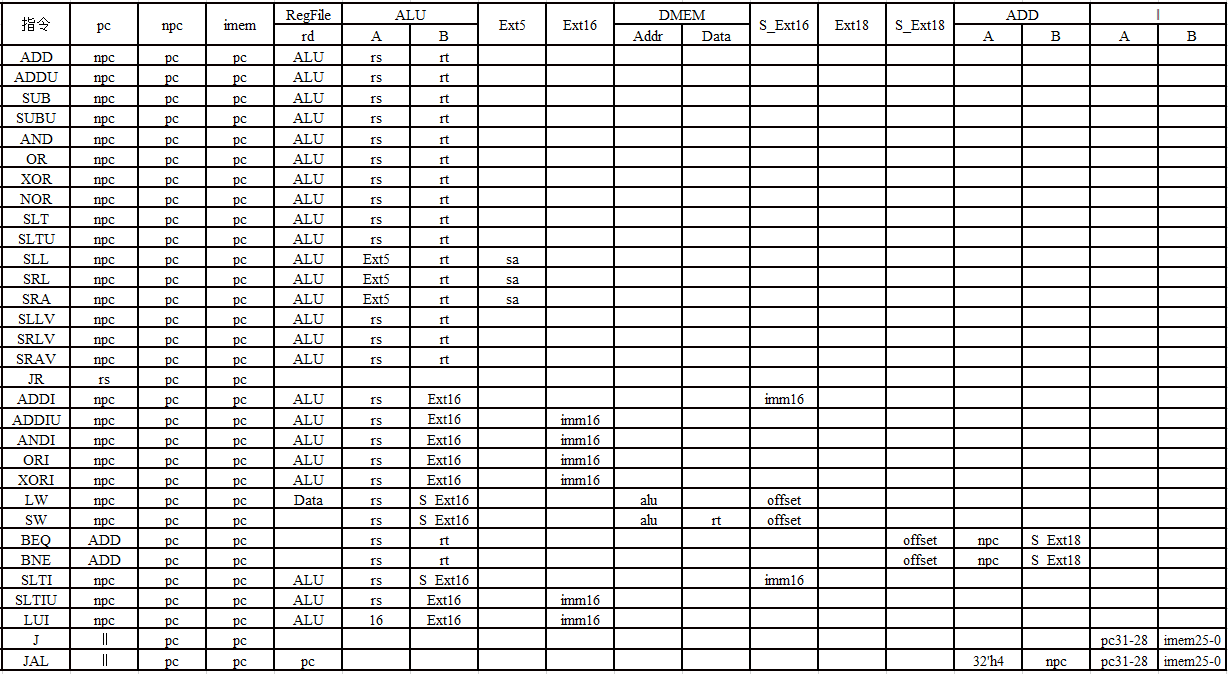
1. JAL



总数据通路：



各部件输入输出关系如下图：

部件功能说明：

NPC：即PC+4，可用简单加法实现

PC：指令计数器

IMEM：指令存储器

DMEM：数据存储器

RegFile：寄存器堆

ALU：算术逻辑单元，实现算数运算与逻辑运算

Ext5：将5位数据无符号扩展为32位

Ext16：将16位数据无符号扩展为32位

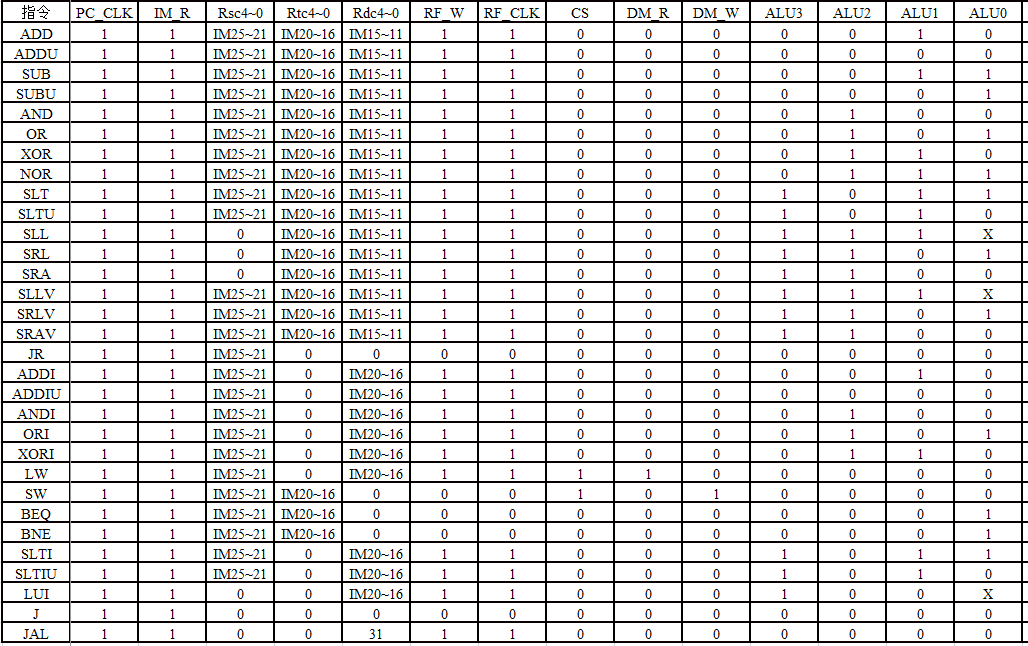
S\_Ext16：将16位数据有符号扩展为32位

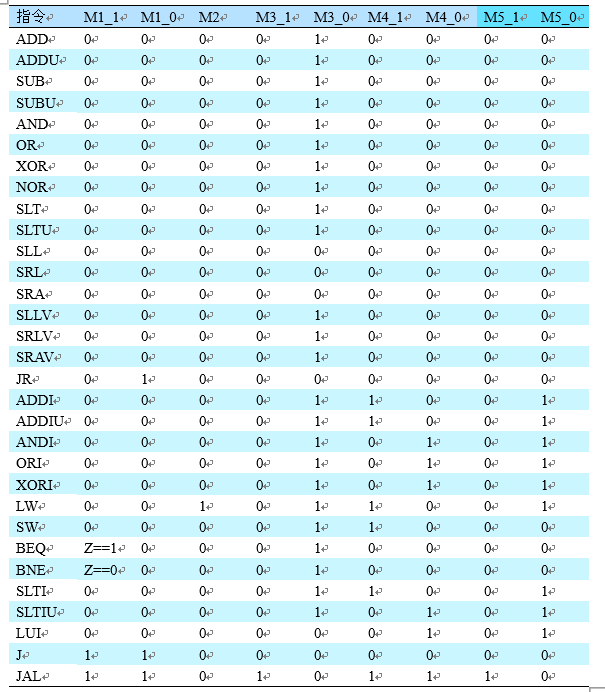
ADD：加法器

||：数据拼接

## CPU控制器设计

各指令控制信号表：





控制信号说明：

PC\_CLK：pc寄存器时钟

IM\_R：指令寄存器IMEM读有效信号

Rsc4~0：rs寄存器选择输入控制端

Rtc4~0：rt寄存器选择输入控制端

Rdc4~0：rd寄存器选择输入控制端

RF\_W：RegFile写信号

RF\_CLK：RegFile时钟

CS：数据存储器片选信号

DM\_R：数据存储器读信号

DM\_W：数据存储器写信号

ALU3：ALU控制端3

ALU2：ALU控制端2

ALU1：ALU控制端1

ALU0：ALU控制端0

M1\_1：MUX1选择器控制端1

M1\_0：MUX1选择器控制端0

M2：MUX2选择器控制端

M3\_1：MUX3选择器控制端1

M3\_0：MUX3选择器控制端0

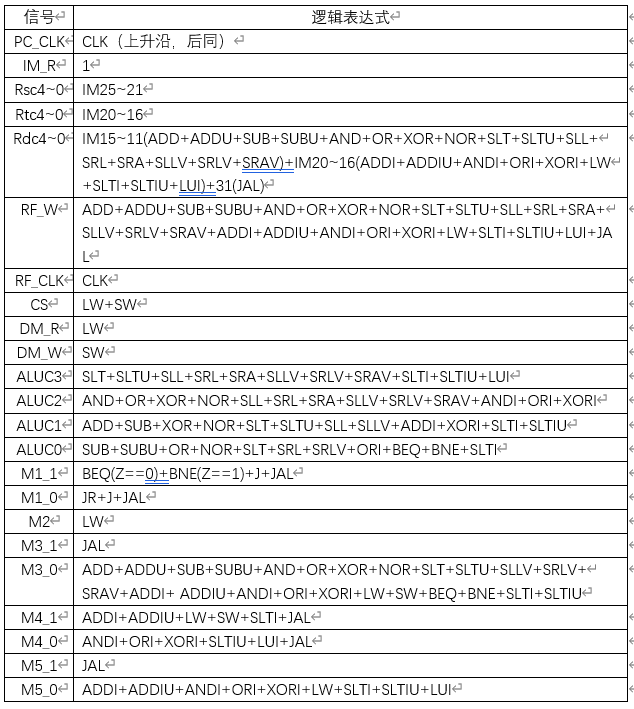
M4\_1：MUX4选择器控制端1

M4\_0：MUX4选择器控制端0

M5\_1：MUX5选择器控制端1

M5\_0：MUX5选择器控制端0

各微指令逻辑表达式如下图：



1. 模块建模

## 4.1 sccomp\_dataflow //顶层模块，调用cpu、imem、dram三个模块

module sccomp\_dataflow(

    input clk\_in,

    input reset,

    output [31:0] inst,

    output [31:0] pc

    );

    //DM信号

    wire CS,DM\_W,DM\_R;

    wire [31:0] DM\_addr;

    wire [31:0] DM\_WData,DM\_RData;

    wire [31:0] instr\_addr;

    wire [31:0] dm\_addr;

    assign instr\_addr = pc - 32'h0040\_0000;

    assign dm\_addr = (DM\_addr - 32'h1001\_0000) / 4;

    //调用IP核

    imem IM(instr\_addr[12:2],inst);

    Dram Dram(clk\_in,CS,DM\_W,DM\_R,dm\_addr,DM\_WData,DM\_RData);

    cpu sccpu(clk\_in,reset,inst,DM\_addr,DM\_RData,DM\_WData,CS,DM\_W,DM\_R,pc);

endmodule

## 4.2 cpu //CPU模块，实现指令译码、数据通路设计、控制器设计等核心功能

module cpu(

    input clk,

    input rst,

    //data(instruction) from iram

    input [31:0] IM,

    //data exchanged with dram

    output [31:0] DM\_addr,

    input [31:0] DM\_RData,

    output [31:0] DM\_WData,

    //control of dram

    output CS,

    output DM\_W,

    output DM\_R,

    //output of pcre

    output [31:0] PC\_out

    );

    //pcreg

    wire PC\_CLK;

    wire ena;

    wire [31:0] PC\_in;

    //wire [31:0] PC\_out;

    //NPC

    wire [31:0] NPC;

    //regfile

    wire [31:0] rs;

    wire [31:0] rt;

    wire [31:0] rd;

    wire [4:0] rdc;

    wire [4:0] rsc;

    wire [4:0] rtc;

    wire RF\_CLK;

    wire RF\_W;

    //alu

    wire [3:0] aluc;

    wire [31:0] alu\_a;

    wire [31:0] alu\_b;

    wire [31:0] alu\_c;

    wire zero;

    wire carry;

    wire negative;

    wire overflow;

    //Iram

    //wire [31:0] IM;

    //EXT

    wire [31:0] Ext5;

    wire [31:0] Ext16;

    wire [31:0] S\_Ext16;

    wire [31:0] S\_Ext18;

    //MUX

    wire M1\_1;

    wire M1\_0;

    wire [31:0] M1\_out;

    wire M2;

    wire [31:0] M2\_out;

    wire M3\_1;

    wire M3\_0;

    wire [31:0] M3\_out;

    wire M4\_1;

    wire M4\_0;

    wire [31:0] M4\_out;

    wire M5\_1;

    wire M5\_0;

    wire [4:0] M5\_out;

    //ADD

    wire [31:0] ADD\_A;

    wire [31:0] ADD\_B;

    wire [31:0] ADD\_C;

    //Connect||

    wire [3:0] Connect\_A;

    wire [27:0] Connect\_B;

    wire [31:0] Connect\_C;

    //指令

    wire ADD = (IM[31:26]==6'b0) && (IM[5:0]==6'b100000);

    wire ADDU = (IM[31:26]==6'b0) && (IM[5:0]==6'b100001);

    wire SUB = (IM[31:26]==6'b0) && (IM[5:0]==6'b100010);

    wire SUBU = (IM[31:26]==6'b0) && (IM[5:0]==6'b100011);

    wire AND = (IM[31:26]==6'b0) && (IM[5:0]==6'b100100);

    wire OR = (IM[31:26]==6'b0) && (IM[5:0]==6'b100101);

    wire XOR = (IM[31:26]==6'b0) && (IM[5:0]==6'b100110);

    wire NOR = (IM[31:26]==6'b0) && (IM[5:0]==6'b100111);

    wire SLT = (IM[31:26]==6'b0) && (IM[5:0]==6'b101010);

    wire SLTU = (IM[31:26]==6'b0) && (IM[5:0]==6'b101011);

    wire SLL = (IM[31:26]==6'b0) && (IM[5:0]==6'b000000);

    wire SRL = (IM[31:26]==6'b0) && (IM[5:0]==6'b000010);

    wire SRA = (IM[31:26]==6'b0) && (IM[5:0]==6'b000011);

    wire SLLV = (IM[31:26]==6'b0) && (IM[5:0]==6'b000100);

    wire SRLV = (IM[31:26]==6'b0) && (IM[5:0]==6'b000110);

    wire SRAV = (IM[31:26]==6'b0) && (IM[5:0]==6'b000111);

    wire JR = (IM[31:26]==6'b0) && (IM[5:0]==6'b001000);

    wire ADDI = (IM[31:26]==6'b001000);

    wire ADDIU = (IM[31:26]==6'b001001);

    wire ANDI = (IM[31:26]==6'b001100);

    wire ORI = (IM[31:26]==6'b001101);

    wire XORI = (IM[31:26]==6'b001110);

    wire LW = (IM[31:26]==6'b100011);

    wire SW = (IM[31:26]==6'b101011);

    wire BEQ = (IM[31:26]==6'b000100);

    wire BNE = (IM[31:26]==6'b000101);

    wire SLTI = (IM[31:26]==6'b001010);

    wire SLTIU = (IM[31:26]==6'b001011);

    wire LUI = (IM[31:26]==6'b001111);

    wire J = (IM[31:26]==6'b000010);

    wire JAL = (IM[31:26]==6'b000011);

    //pcreg

    assign PC\_CLK = clk;

    assign ena = 1;

    assign PC\_in = M1\_out;

    //Regfiles

    assign rd = M2\_out;

    assign rdc = M5\_out;

    assign rsc = IM[25:21];

    assign rtc = IM[20:16];

    assign RF\_W = ADD||ADDU||SUB||SUBU||AND||OR||XOR||NOR||SLT||SLTU||SLL||SRL||SRA||SLLV||SRLV||

                  SRAV||ADDI||ADDIU||ANDI||ORI||XORI||LW||SLTI||SLTIU||LUI||JAL;

    assign RF\_CLK = clk;

    //alu

    assign aluc[3] = SLT||SLTU||SLL||SRL||SRA||SLLV||SRLV||SRAV||SLTI||SLTIU||LUI;

    assign aluc[2] = AND||OR||XOR||NOR||SLL||SRL||SRA||SLLV||SRLV||SRAV||ANDI||ORI||XORI;

    assign aluc[1] = ADD||SUB||XOR||NOR||SLT||SLTU||SLL||SLLV||ADDI||XORI||SLTI||SLTIU||JAL;

    assign aluc[0] = SUB||SUBU||OR||NOR||SLT||SRL||SRLV||ORI||BEQ||BNE||SLTI;

    assign alu\_a = M3\_out;

    assign alu\_b = M4\_out;

    //MUX

    assign M1\_1 = (BEQ && (zero == 1))||(BNE && (zero == 0))||J||JAL;

    assign M1\_0 = JR||J||JAL;

    assign M2 = LW;

    assign M3\_1 = JAL;

    assign M3\_0 = ADD||ADDU||SUB||SUBU||AND||OR||XOR||NOR||SLT||SLTU||SLLV||SRLV||SRAV||ADDI||

                ADDIU||ANDI||ORI||XORI||LW||SW||BEQ||BNE||SLTI||SLTIU;

    assign M4\_1 = ADDI||ADDIU||LW||SW||SLTI||JAL;

    assign M4\_0 = ANDI||ORI||XORI||SLTIU||LUI||JAL;

    assign M5\_1 = JAL;

    assign M5\_0 = ADDI||ADDIU||ANDI||ORI||XORI||LW||SLTI||SLTIU||LUI;

    //Dram

    assign CS = LW||SW;

    assign DM\_R = LW;

    assign DM\_W = SW;

    assign DM\_addr = alu\_c;

    assign DM\_WData = rt;

    //Connect||

    assign Connect\_A = PC\_out[31:28];

    assign Connect\_B = {IM[25:0],2'b00};

    assign Connect\_C = {Connect\_A,Connect\_B};

    //ADD

    assign ADD\_A = S\_Ext18;

    assign ADD\_B = NPC;

    assign ADD\_C = ADD\_A + ADD\_B;

    assign NPC = PC\_out + 4;

    Ext5 E5(IM[10:6],Ext5);

    Ext16 E16(IM[15:0],Ext16);

    S\_Ext16 S\_E16(IM[15:0],S\_Ext16);

    S\_Ext18 S\_E18({IM[15:0],2'b00},S\_Ext18);

    MUX MUX1(NPC,rs,ADD\_C,Connect\_C,M1\_1,M1\_0,M1\_out);

    assign M2\_out = M2==0?alu\_c:DM\_RData;

    MUX MUX3(Ext5,rs,4,0,M3\_1,M3\_0,M3\_out);

    MUX MUX4(rt,Ext16,S\_Ext16,PC\_out,M4\_1,M4\_0,M4\_out);

    MUX\_5bit MUX5(IM[15:11],IM[20:16],31,0,M5\_1,M5\_0,M5\_out);

    pcreg pcreg(PC\_CLK,rst,ena,PC\_in,PC\_out);

    alu alu(alu\_a,alu\_b,aluc,alu\_c,zero,carry,negative,overflow);

    regfile cpu\_ref(RF\_CLK,rst,1,RF\_W,rsc,rtc,rdc,rd,rs,rt);

endmodule

## 4.3 alu //算术逻辑单元模块，实现多种逻辑运算与算术运算

module alu(

    input [31:0] a,

    input [31:0] b,

    input [3:0] aluc,

    output reg [31:0] r,

    output reg zero,

    output reg carry,

    output reg negative,

    output reg overflow

    );

    reg [31:0] oTmpt;

    always@(\*)

        case(aluc)

            4'b0000:r=a+b;

            4'b0010:r=$signed(a)+$signed(b);

            4'b0001:r=a-b;

            4'b0011:r=a-b;

            4'b0100:r=a&b;

            4'b0101:r=a|b;

            4'b0110:r=a^b;

            4'b0111:r=~(a|b);

            4'b1000:r={b[15:0],16'b0};

            4'b1001:r={b[15:0],16'b0};

            4'b1011:r=($signed(a)<$signed(b))?1:0;

            4'b1010:r=(a<b)?1:0;

            4'b1100:r=$signed(b)>>>a;

            4'b1110:r=b<<a;

            4'b1111:r=b<<a;

            4'b1101:r=b>>a;

            default:r=32'h00000000;

        endcase

    //zero

    always@(\*)

        if(aluc==4'b1011||aluc==4'b1010)

            if(a==b)

                zero=1'b1;

            else

                zero=1'b0;

        else if(r==0)

            zero=1'b1;

        else

            zero=1'b0;

    //negative

    always@(\*)

        if(aluc==4'b0010||aluc==4'b0011)

            if($signed(r)<0)

                negative=1'b1;

            else

                negative=1'b0;

        else if(aluc==4'b1011)

            if(r==1'b1)

                negative=1'b1;

            else

                negative=1'b0;

        else

            if(r[31]==1'b1)

                negative=1'b1;

            else

                negative=1'b0;

    //carry

    always@(\*)

        if(aluc==4'b0000)

            if(a[31]==1'b1&&b[31]==1'b1)

                carry=1'b1;

            else if(a[31]==1'b1&&b[31]==1'b0&&r[31]==1'b0)

                carry=1'b1;

            else if(a[31]==1'b0&&b[31]==1'b1&&r[31]==1'b0)

                carry=1'b1;

            else

                carry=1'b0;

        else if(aluc==4'b0001)

            if(a<b)

                carry=1'b1;

            else

                carry=1'b0;

        else if(aluc==4'b1010)

            if(a<b)

                carry=1'b1;

            else

                carry=1'b0;

        else if(aluc==4'b1100)

            begin

            oTmpt=$signed(b)>>>(a-1);

            carry=oTmpt[0];

            end

        else if(aluc==4'b1101)

            begin

            oTmpt=b>>>(a-1);

            carry=oTmpt[0];

            end

        else if(aluc==4'b1110||aluc==4'b1111)

            begin

            oTmpt=b<<(a-1);

            carry=oTmpt[31];

            end

    //overflow

    always@(\*)

        if(aluc==4'b0010)

            if(a[31]==b[31]&&~r[31]==a[31])

                overflow=1'b1;

            else

                overflow=1'b0;

        else if(aluc==4'b0011)

            if(a[31]==0&&b[31]==1&&r[31]==1)

                overflow=1'b1;

            else if(a[31]==1&&b[31]==0&&r[31]==0)

                overflow=1'b1;

            else

                overflow=1'b0;

endmodule

## 4.4 Dram //数据存储器模块，用于向内存中读写数据

module Dram(

    input clk,

    input ena,

    input DM\_W,

    input DM\_R,

    input [31:0] addr,

    input [31:0] data\_in,

    output [31:0] data\_out

    );

    reg [31:0] RAM [31:0];

    assign data\_out = (ena && DM\_R) ? RAM[addr] : 32'hzzzzzzzz;

    always @ (posedge clk)

    begin

        if(ena && DM\_W)

            RAM[addr] <= data\_in;

    end

endmodule

## 4.5 pcreg //指令寄存器模块，每个时钟下降沿更新一次

module pcreg(

    input clk,

    input rst,

    input ena,

    input [31:0] data\_in,

    output reg [31:0] data\_out

    );

    always@(negedge clk or posedge rst)

        begin

            if(rst==1'b1)

                data\_out=32'h00400000;

            else

                if(ena==1'b1)

                    data\_out=data\_in;

                else if(ena==1'b0)

                    data\_out=data\_out;

        end

endmodule

## 4.6 regfile //寄存器堆模块，包括32个寄存器，根据地址内容进行读写

module regfile(

    input clk,

    input rst,

    input en,

    input rf\_write,

    input [4:0] rsc,

    input [4:0] rtc,

    input [4:0] rdc,

    input [31:0] rd,

    output [31:0] rs,

    output [31:0] rt

    );

    reg [31:0] array\_reg[31:0];

    reg [5:0] i;

    assign rs = en ? array\_reg[rsc] : 32'bz;

    assign rt = en ?array\_reg[rtc] : 32'bz;

    always @(negedge clk or posedge rst) begin

      if (rst) begin

        for(i=0;i<32;i=i+1)

          array\_reg[i] <= 0;

        end

    end

    always@(posedge clk) begin begin

      if (rf\_write && en && (rdc != 0))

        array\_reg[rdc] <= rd;

      end

    end

endmodule

## 4.7 MUX //四选一数据选择器

module MUX(

    input [31:0] iC0,

    input [31:0] iC1,

    input [31:0] iC2,

    input [31:0] iC3,

    input iS1,

    input iS0,

    output [31:0] oZ

    );

reg[31:0] tmpt;

always@(\*)

begin

   if(iS1==0)

        begin

            if(iS0==0)

                tmpt = iC0;

            else

                tmpt = iC1;

        end

   else

        begin

            if(iS0==0)

                tmpt = iC2;

            else

                tmpt = iC3;

        end

end

assign oZ = tmpt;

endmodule

## 4.7 MUX\_5bit //五位四选一数据选择器

module MUX\_5bit(

    input [4:0] iC0,

    input [4:0] iC1,

    input [4:0] iC2,

    input [4:0] iC3,

    input iS1,

    input iS0,

    output [4:0] oZ

    );

reg[31:0] tmpt;

    always@(\*)

    begin

       if(iS1==0)

            begin

                if(iS0==0)

                    tmpt = iC0;

                else

                    tmpt = iC1;

            end

       else

            begin

                if(iS0==0)

                    tmpt = iC2;

                else

                    tmpt = iC3;

            end

    end

    assign oZ = tmpt;

endmodule

## 4.8 Ext5 //将5位数据无符号扩展为32位

module Ext5(

    input [4:0] data\_in,

    output [31:0] data\_out

    );

    assign data\_out = {27'b0,data\_in};

endmodule

## 4.9 Ext16 //将16位数据无符号扩展为32位

module Ext16(

    input [15:0] data\_in,

    output [31:0] data\_out

    );

    assign data\_out = {16'b0,data\_in};

endmodule

## 4.10 S\_Ext16 //将16位数据有符号扩展为32位

module S\_Ext16(

    input [15:0] data\_in,

    output [31:0] data\_out

    );

    assign data\_out = {{16{data\_in[15]}},data\_in};

endmodule

## 4.11 S\_Ext18 //将18位数据有符号扩展为32位

module S\_Ext18(

    input [17:0] data\_in,

    output [31:0] data\_out

    );

    assign data\_out = {{14{data\_in[17]}},data\_in};

endmodule

## 4.12 imem //指令寄存器模块，通过调用ip核实现

`timescale 1ns/1ps

(\* DowngradeIPIdentifiedWarnings = "yes" \*)

module imem (

  a,

  spo

);

input wire [10 : 0] a;

output wire [31 : 0] spo;

  dist\_mem\_gen\_v8\_0\_10 #(

    .C\_FAMILY("artix7"),

    .C\_ADDR\_WIDTH(11),

    .C\_DEFAULT\_DATA("0"),

    .C\_DEPTH(2048),

    .C\_HAS\_CLK(0),

    .C\_HAS\_D(0),

    .C\_HAS\_DPO(0),

    .C\_HAS\_DPRA(0),

    .C\_HAS\_I\_CE(0),

    .C\_HAS\_QDPO(0),

    .C\_HAS\_QDPO\_CE(0),

    .C\_HAS\_QDPO\_CLK(0),

    .C\_HAS\_QDPO\_RST(0),

    .C\_HAS\_QDPO\_SRST(0),

    .C\_HAS\_QSPO(0),

    .C\_HAS\_QSPO\_CE(0),

    .C\_HAS\_QSPO\_RST(0),

    .C\_HAS\_QSPO\_SRST(0),

    .C\_HAS\_SPO(1),

    .C\_HAS\_WE(0),

    .C\_MEM\_INIT\_FILE("imem.mif"),

    .C\_ELABORATION\_DIR("./"),

    .C\_MEM\_TYPE(0),

    .C\_PIPELINE\_STAGES(0),

    .C\_QCE\_JOINED(0),

    .C\_QUALIFY\_WE(0),

    .C\_READ\_MIF(1),

    .C\_REG\_A\_D\_INPUTS(0),

    .C\_REG\_DPRA\_INPUT(0),

    .C\_SYNC\_ENABLE(1),

    .C\_WIDTH(32),

    .C\_PARSER\_TYPE(1)

  ) inst (

    .a(a),

    .d(32'B0),

    .dpra(11'B0),

    .clk(1'D0),

    .we(1'D0),

    .i\_ce(1'D1),

    .qspo\_ce(1'D1),

    .qdpo\_ce(1'D1),

    .qdpo\_clk(1'D0),

    .qspo\_rst(1'D0),

    .qdpo\_rst(1'D0),

    .qspo\_srst(1'D0),

    .qdpo\_srst(1'D0),

    .spo(spo),

    .dpo(),

    .qspo(),

    .qdpo()

  );

endmodule

1. 测试模块建模

（要求列写各建模模块的test bench模块代码）

`timescale 1ns / 1ps

module cpu\_tb(

    );

        reg clk\_in;

        reg reset;

        reg start;

        wire[31:0] inst;

        wire[31:0] pc;

        sccomp\_dataflow sc(clk\_in, reset, inst, pc);

        wire [31:0] M1\_out = sc.sccpu.M1\_out;

        wire [31:0] NPC = sc.sccpu.NPC;

        wire [31:0] rd = sc.sccpu.rd;

        wire [31:0] rs = sc.sccpu.rs;

        wire [31:0] rt = sc.sccpu.rt;

        wire [31:0] alu\_a = sc.sccpu.alu\_a;

        wire [31:0] alu\_b = sc.sccpu.alu\_b;

        wire [31:0] alu\_c = sc.sccpu.alu\_c;

        wire [31:0] Ext5 = sc.sccpu.Ext5;

        wire [31:0] Ext16 = sc.sccpu.Ext16;

        wire [31:0] S\_Ext16 = sc.sccpu.S\_Ext16;

        wire [31:0] S\_Ext18 = sc.sccpu.S\_Ext18;

        wire [4:0] rsc = sc.sccpu.rsc;

        wire [4:0] rtc = sc.sccpu.rtc;

        wire M4\_1 = sc.sccpu.M4\_1;

        wire M4\_0 = sc.sccpu.M4\_0;

        wire [31:0] DM\_addr = sc.DM\_addr;

        wire [31:0] dm\_addr = sc.dm\_addr;

        wire CS = sc.CS;

        wire DM\_R = sc.DM\_R;

        integer file\_output;

        integer counter = 0;

        initial

        begin

            file\_output = $fopen("D:/computer\_composition/MIPS31/result.txt");

        end

        initial

        begin

            clk\_in = 1;

            start  = 0;

            forever begin

                #50 clk\_in = ~clk\_in;

            end

        end

        initial begin

            reset     = 0;

            #6 reset  = 1;

            #50 reset = 0;

            start = 1;

        end

        always @(posedge clk\_in) begin

            if (start)begin

                counter  = counter + 1;

                $fdisplay(file\_output, "regfiles0 = %h", sc.sccpu.cpu\_ref.array\_reg[0]);

                $fdisplay(file\_output, "regfiles1 = %h", sc.sccpu.cpu\_ref.array\_reg[1]);

                $fdisplay(file\_output, "regfiles2 = %h", sc.sccpu.cpu\_ref.array\_reg[2]);

                $fdisplay(file\_output, "regfiles3 = %h", sc.sccpu.cpu\_ref.array\_reg[3]);

                $fdisplay(file\_output, "regfiles4 = %h", sc.sccpu.cpu\_ref.array\_reg[4]);

                $fdisplay(file\_output, "regfiles5 = %h", sc.sccpu.cpu\_ref.array\_reg[5]);

                $fdisplay(file\_output, "regfiles6 = %h", sc.sccpu.cpu\_ref.array\_reg[6]);

                $fdisplay(file\_output, "regfiles7 = %h", sc.sccpu.cpu\_ref.array\_reg[7]);

                $fdisplay(file\_output, "regfiles8 = %h", sc.sccpu.cpu\_ref.array\_reg[8]);

                $fdisplay(file\_output, "regfiles9 = %h", sc.sccpu.cpu\_ref.array\_reg[9]);

                $fdisplay(file\_output, "regfiles10 = %h", sc.sccpu.cpu\_ref.array\_reg[10]);

                $fdisplay(file\_output, "regfiles11 = %h", sc.sccpu.cpu\_ref.array\_reg[11]);

                $fdisplay(file\_output, "regfiles12 = %h", sc.sccpu.cpu\_ref.array\_reg[12]);

                $fdisplay(file\_output, "regfiles13 = %h", sc.sccpu.cpu\_ref.array\_reg[13]);

                $fdisplay(file\_output, "regfiles14 = %h", sc.sccpu.cpu\_ref.array\_reg[14]);

                $fdisplay(file\_output, "regfiles15 = %h", sc.sccpu.cpu\_ref.array\_reg[15]);

                $fdisplay(file\_output, "regfiles16 = %h", sc.sccpu.cpu\_ref.array\_reg[16]);

                $fdisplay(file\_output, "regfiles17 = %h", sc.sccpu.cpu\_ref.array\_reg[17]);

                $fdisplay(file\_output, "regfiles18 = %h", sc.sccpu.cpu\_ref.array\_reg[18]);

                $fdisplay(file\_output, "regfiles19 = %h", sc.sccpu.cpu\_ref.array\_reg[19]);

                $fdisplay(file\_output, "regfiles20 = %h", sc.sccpu.cpu\_ref.array\_reg[20]);

                $fdisplay(file\_output, "regfiles21 = %h", sc.sccpu.cpu\_ref.array\_reg[21]);

                $fdisplay(file\_output, "regfiles22 = %h", sc.sccpu.cpu\_ref.array\_reg[22]);

                $fdisplay(file\_output, "regfiles23 = %h", sc.sccpu.cpu\_ref.array\_reg[23]);

                $fdisplay(file\_output, "regfiles24 = %h", sc.sccpu.cpu\_ref.array\_reg[24]);

                $fdisplay(file\_output, "regfiles25 = %h", sc.sccpu.cpu\_ref.array\_reg[25]);

                $fdisplay(file\_output, "regfiles26 = %h", sc.sccpu.cpu\_ref.array\_reg[26]);

                $fdisplay(file\_output, "regfiles27 = %h", sc.sccpu.cpu\_ref.array\_reg[27]);

                $fdisplay(file\_output, "regfiles28 = %h", sc.sccpu.cpu\_ref.array\_reg[28]);

                $fdisplay(file\_output, "regfiles29 = %h", sc.sccpu.cpu\_ref.array\_reg[29]);

                $fdisplay(file\_output, "regfiles30 = %h", sc.sccpu.cpu\_ref.array\_reg[30]);

                $fdisplay(file\_output, "regfiles31 = %h", sc.sccpu.cpu\_ref.array\_reg[31]);

                $fdisplay(file\_output, "instr = %h", inst);

                $fdisplay(file\_output, "pc = %h", pc);

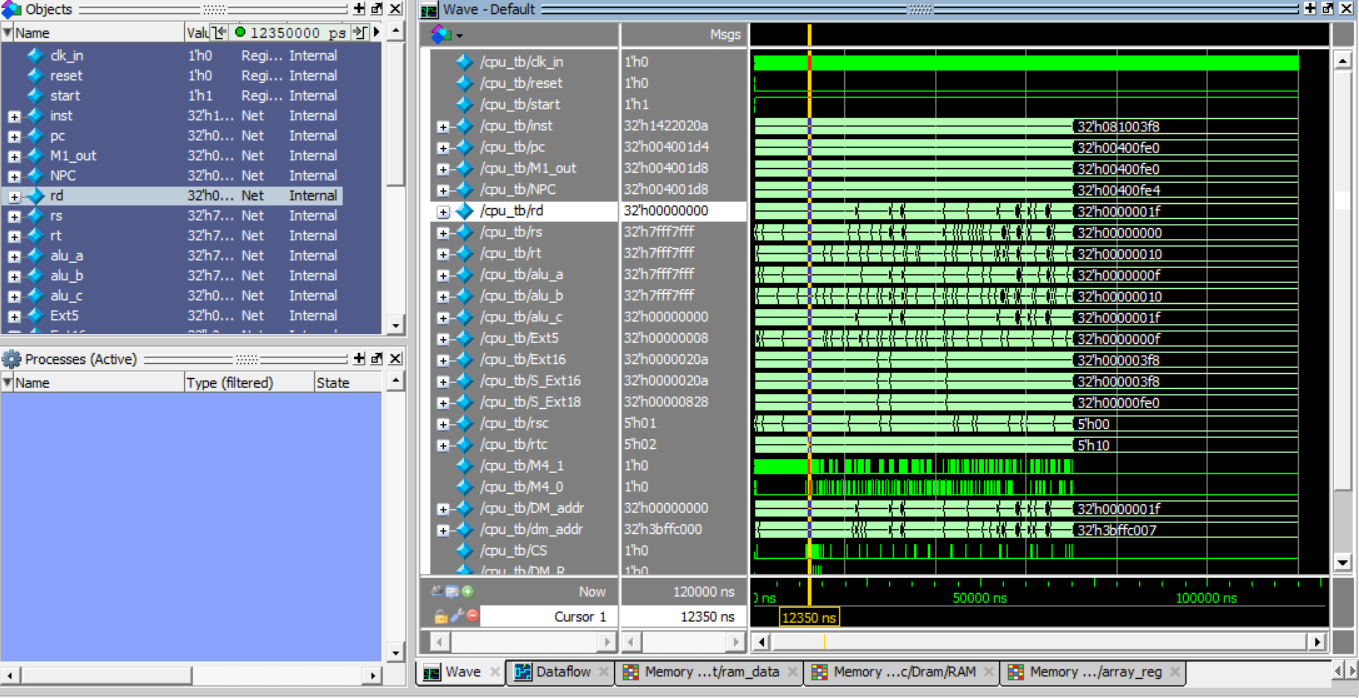
            end

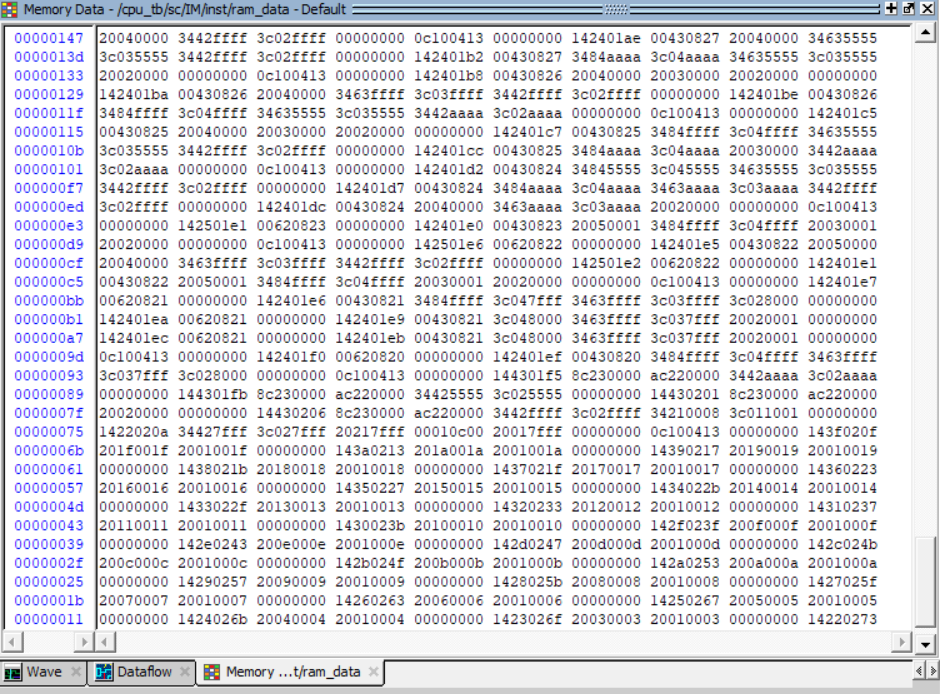
          end

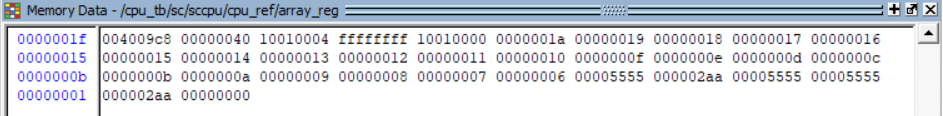
endmodule

1. 实验结果

## 6.1 modelsim仿真波形图



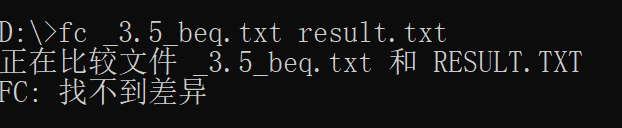


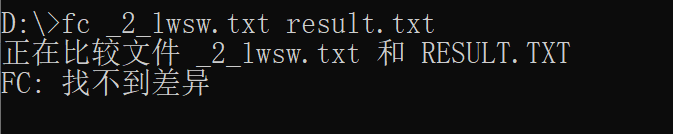


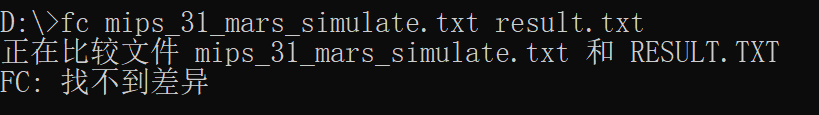
由上图我们可以看到，指令寄存器IM内存已读入相关指令译码后的结果，指令寄存器运行正常。同时寄存器堆regfile中已成功写入数据，寄存器堆也正常工作。

## 6.2 指令执行结果比对

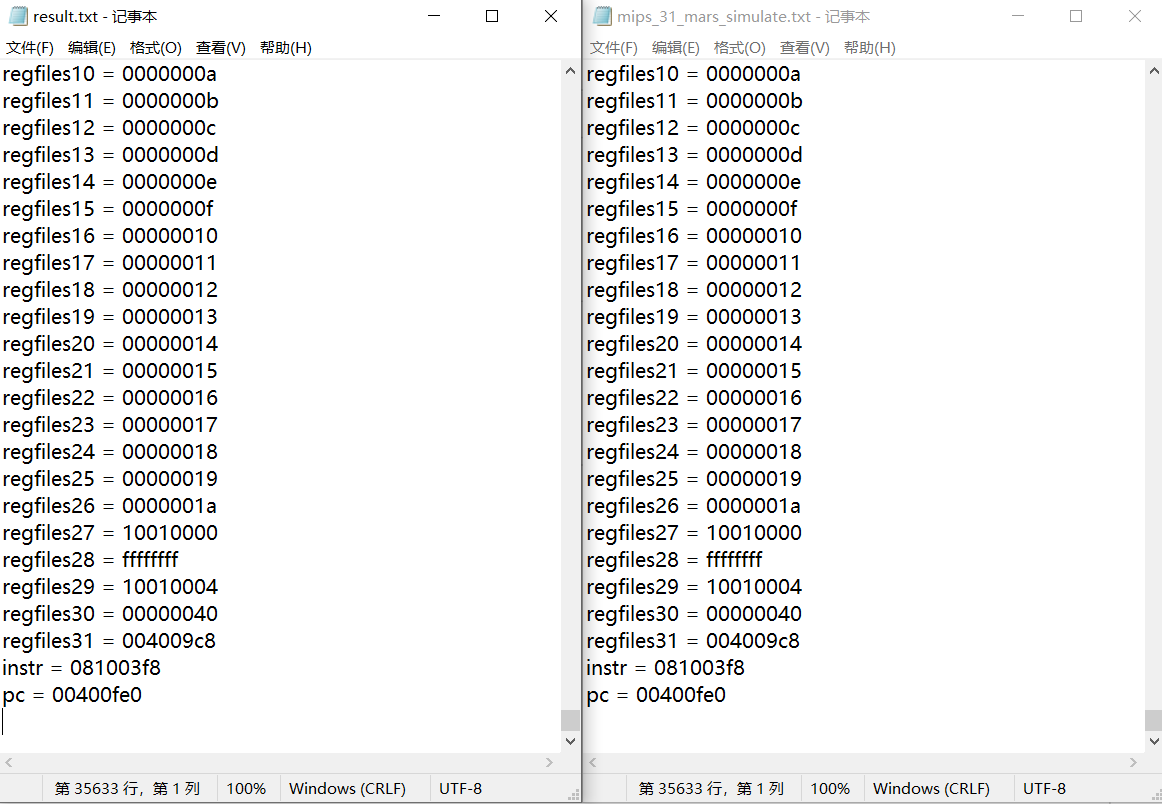
我通过采用cmd控制台的比较文件指令分别对31条指令以及最终版指令运行结果与标准结果进行比对，依次确认比对结果与标准结果并无差异以确认指令运行正确，CPU设计正确。

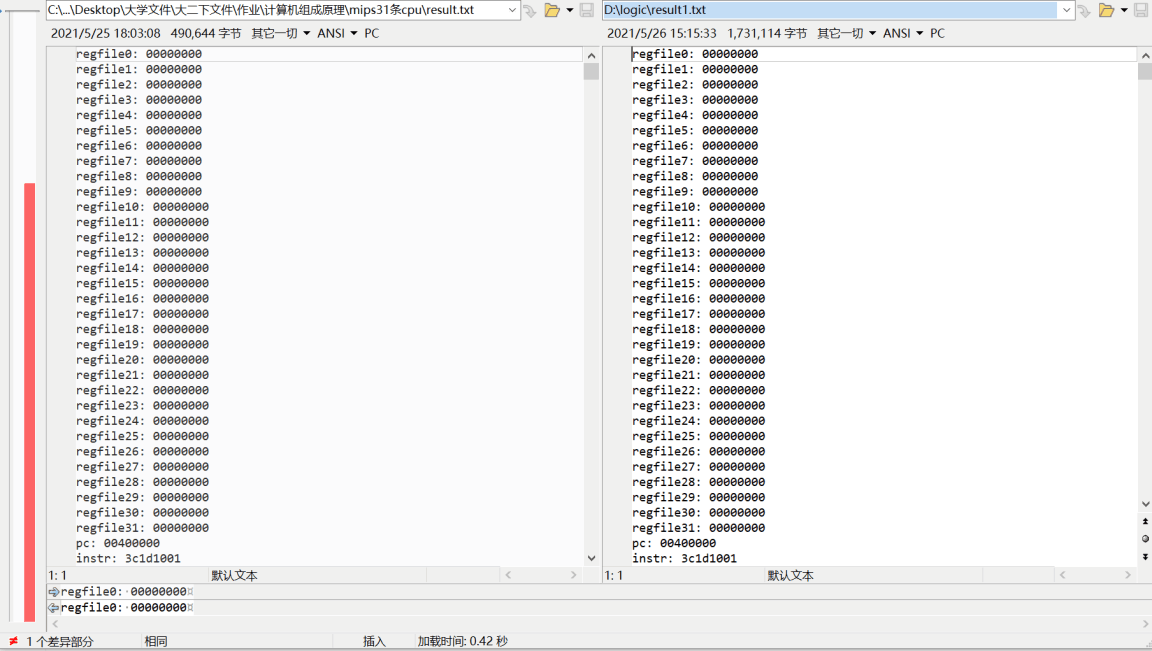






通过文件比对可以看出运行结果与标准并无差异，唯一一处不同在于运行结尾循环终止出，由此证明运行正常。





1. 总结与体会

总体来说，在本次31条指令CPU的设计中，我从最初的对CPU构架处于一知半解的状态，只是在理论层面有所了解而理解不够深刻到通过实际操作完成本次作业实现一个能够运行31条指令的CPU后，对CPU的设计流程、数据通路、控制器等有了更深刻的理解，收获很大。

此次CPU设计过程，我是根据实验指导书上的顺序对不同部件依次进行设计的。首先是CPU数据通路的设计，我根据指令的功能，确定每条指令所用到的部件；其次根据各个指令所用的部件，表格列出，同时在表格中填入每个部件的数据输入来源，再根据数据输入来源画出每条指令的数据通路，最后将所有指令数据通路合并成一个总的数据通路。合并过程涉及数据选择器MUX，对于同一部件有不同输入来源的情况，根据数据选择器输入选择端决定在不同指令执行情况下将什么输入来源送到该部件。

数据通路设计完成后，我接着进行了CPU控制器的设计，具体流程为：首先绘制指令流程图，把每一条机器指令都分解为一系列微操作，编排指令操作时间表，再者进行微操作综合，将相同的微操作综合起来，通过不同指令间的或操作进行赋值得到每个微操作的逻辑表达式，记录各个微操作不同时刻的电平，若为高电平则执行此微操作。最后画出控制器逻辑电路，根据各微操作的逻辑表达式设计CPU控制单元。

数据通路以及控制器均设计完成后，前期准备工作告一段落，就进入了代码编写环节，通过顶层模块调用cpu模块、指令/数据寄存器模块，在CPU模块中调用诸如ALU、regfile、pcreg、Ext等子模块实现CPU中数据通路、控制器的设计。代码完成后，通过IP核读入指令，将寄存器结果写入文件与通过MARS编译后产生的结果进行比对，准确无误后，确实设计正确。

在设计过程中，我遇到了许多由于疏忽而产生的bug，有些错误甚至极难发现修改，debug过程还是比较艰难痛苦的。首先是在将数据通路与控制器通过代码呈现时无法正常读入诸如IP核中的指令数据等，导致modelsim仿真以及寄存器结果文件输出无有效值，均为未知值x或z。我通过排查数据通路是否连接、命名是否正确，最终发现问题出在顶层模块调用子模块时命名出现了错误，instr写成了inst，导致无法正常写入。更多的错误出在依次对单条指令进行执行比对过程中寄存器结果、地址、程序计数器出现不一致的情况。具体的原因包括：控制信号逻辑表达式有误、数据选择器有误，经过仔细排查，重新设计更改指令数据通路，重新写不同微指令的逻辑表达式，最终解决了相关问题。