**同济大学计算机系**

**计算机组成原理实验报告**

****

**学 号 1952650**

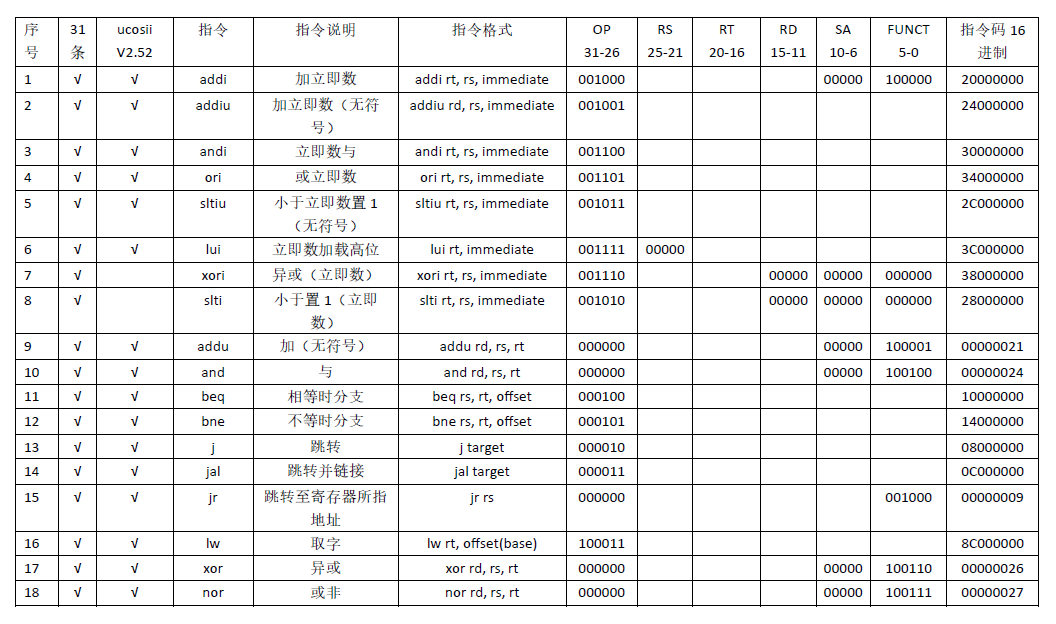
**姓 名 陈子翔**

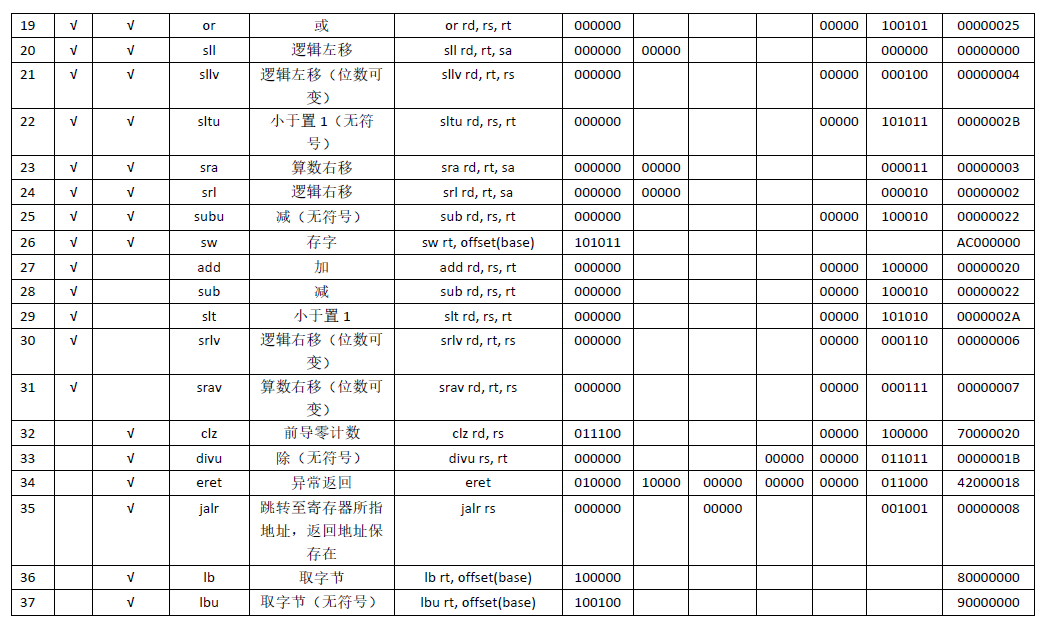
**专 业 信息安全**

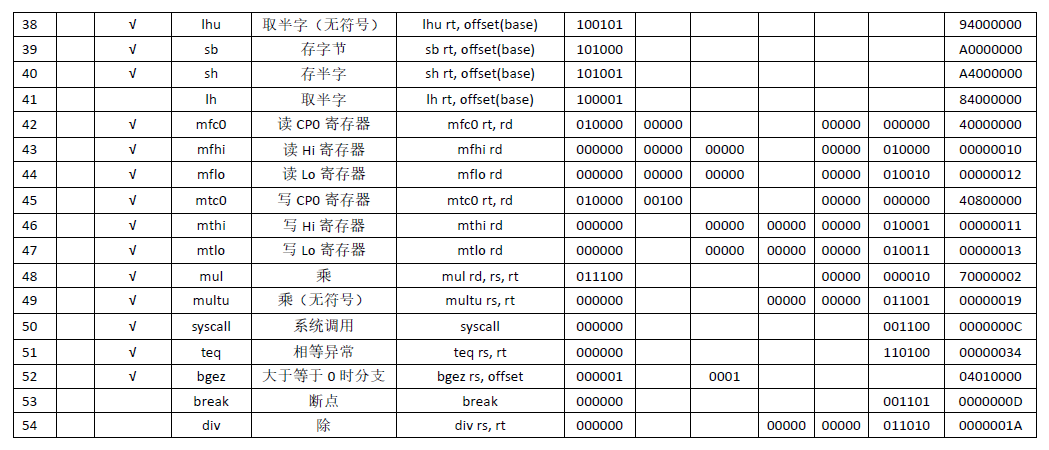
**授课老师 郝老师**

1. 实验内容

使用Verilog HDL语言实现54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54条指令的CPU的数据通路及控制器。
4. 使用Verilog HDL设计实现54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 |

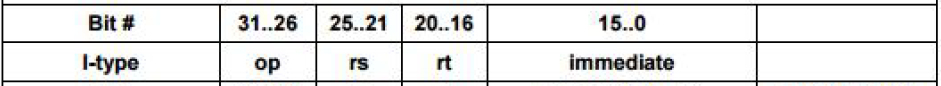
其中，操作码字段指出指令中该指令应该执行什么性质的操作和具有何种功能。操作码是识别指令、了解指令功能及区分操作数地址内容的组成和使用方法等的关键信息。

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。

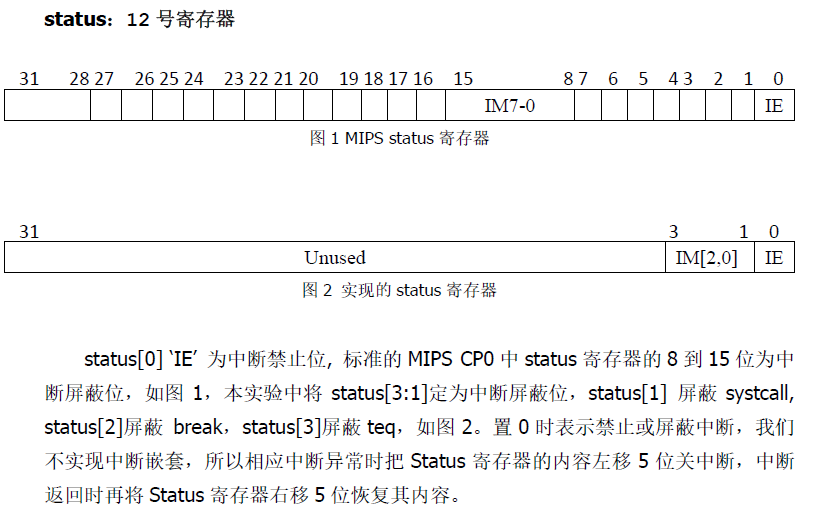
func：6位功能码，与操作码配合使用，在R类指令中，操作码均为0，用不同的功能码区分执行何种指令。

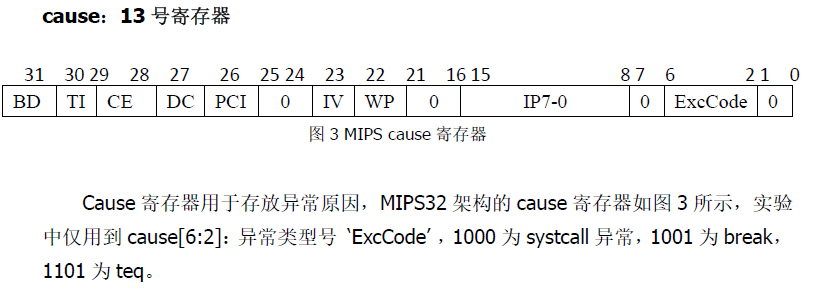
immediate：16位立即数，用作无符号的逻辑操作数、有符号的算数操作数、数据加载/数据保存指令的数据地址字节偏移量和分支指令中相对程序计数器的有符号偏移量。

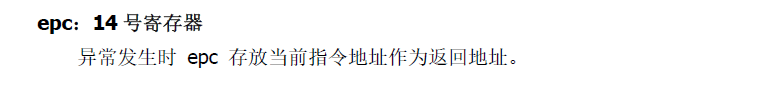
index：26位地址码，用于J类指令中作pc跳转的地址。

## 3.3 CP0中断

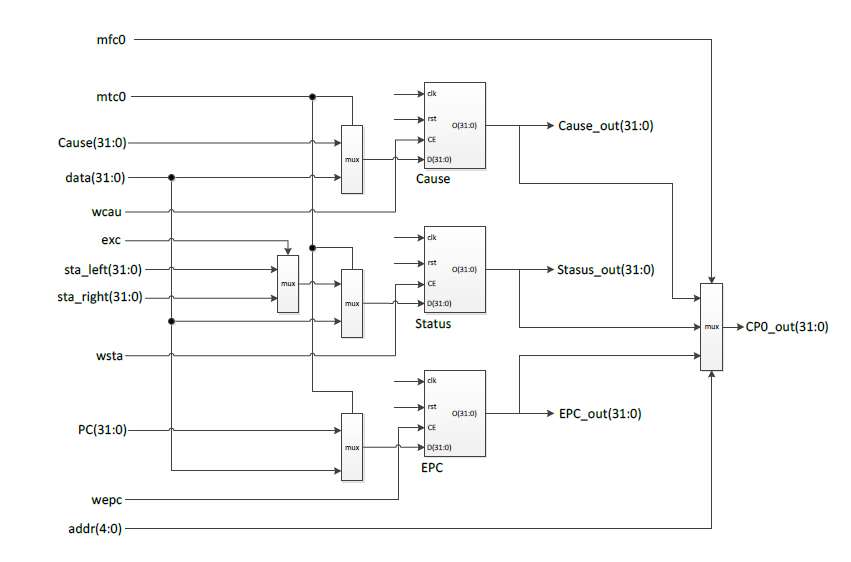
除了基本的运算功能，任何处理器均需要一些部件处理中断、配置选项以及需要某种机制来监控诸如高速缓存和定时器等功能。在MIPS CPU中，异常或中断发生的行为以及怎样处理都是由CP0控制寄存器和几条特殊指令来定义和控制的。CP0中有一系列寄存器来完成异常控制、缓存控制等功能。本CPU实验需用到部分寄存器：







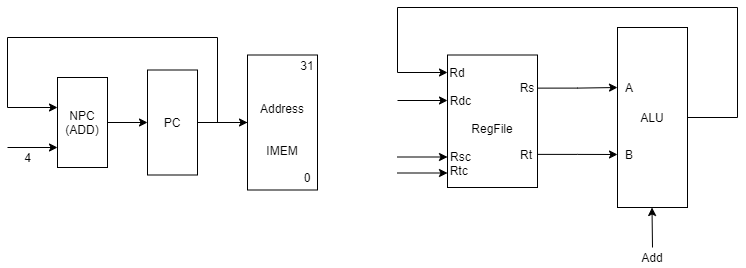
CP0数据通路图如下：



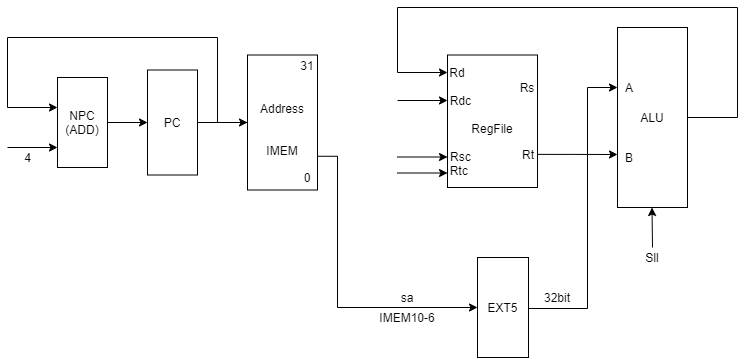
## 3.4数据通路设计

对于相似的指令，相同的数据通路图不再重复画出。

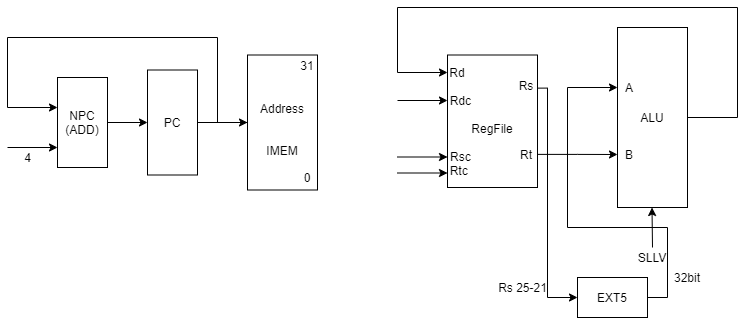
1. ADD/ADDU/SUB/SUBU/AND/OR/XOR/NOR/SLT/SLTU



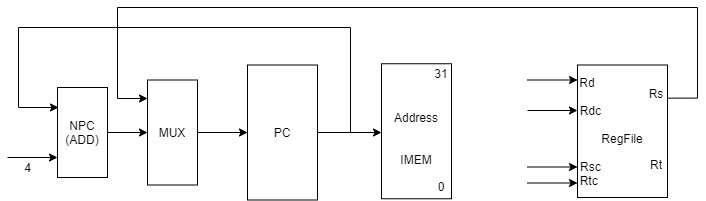
1. SLL/SRL/SRA



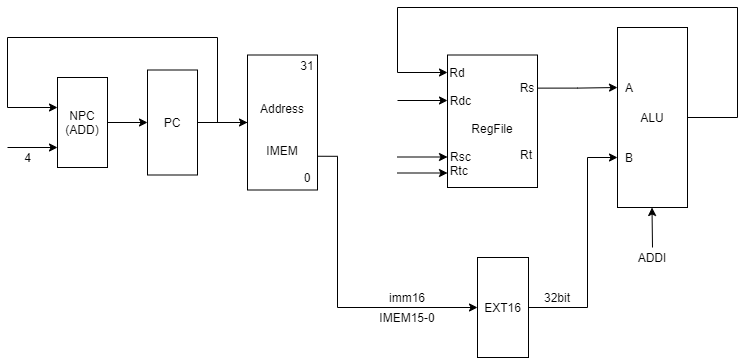
1. SLLV/SRLV/SRAV



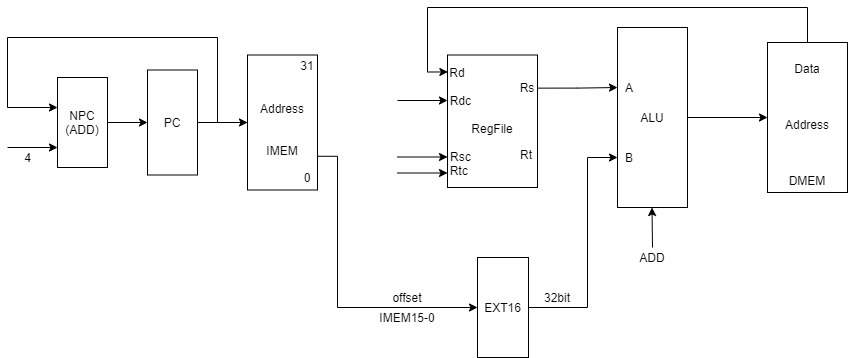
1. JR



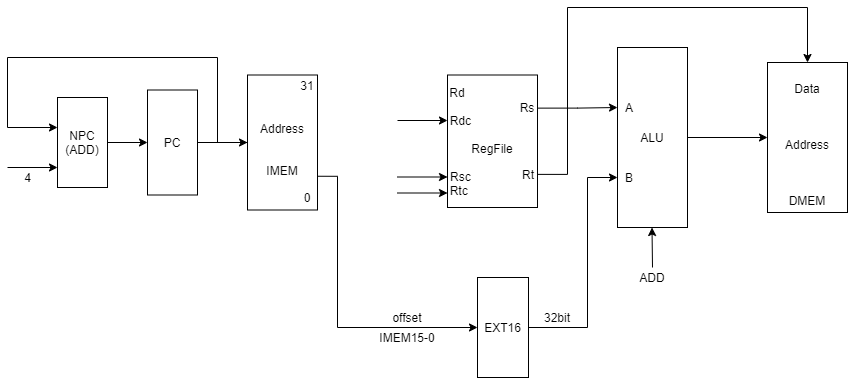
1. ADDI/ADDIU/ANDI/ORI/XORI



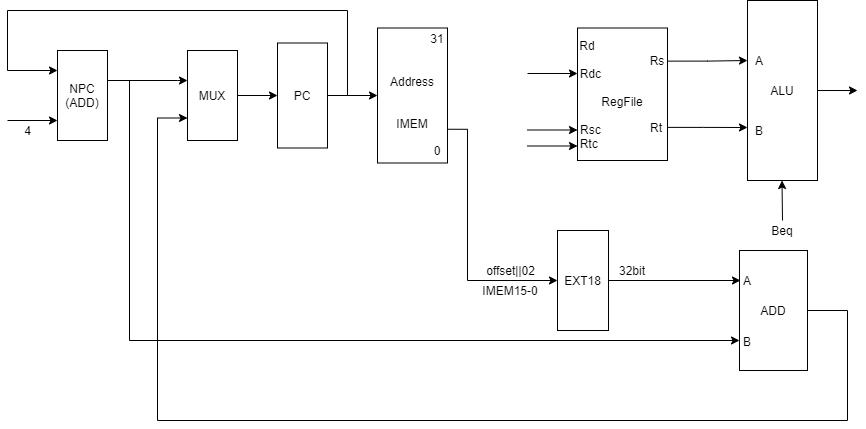
1. LW



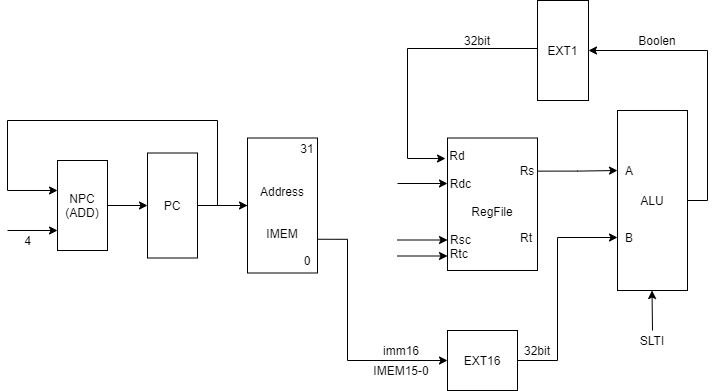
1. SW



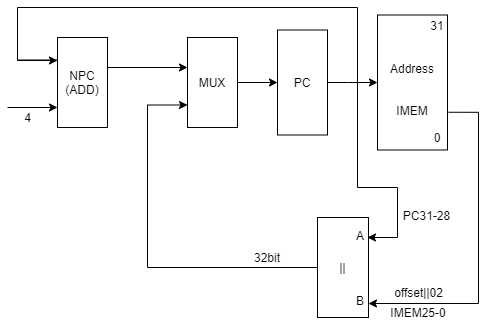
1. BEQ/BNE



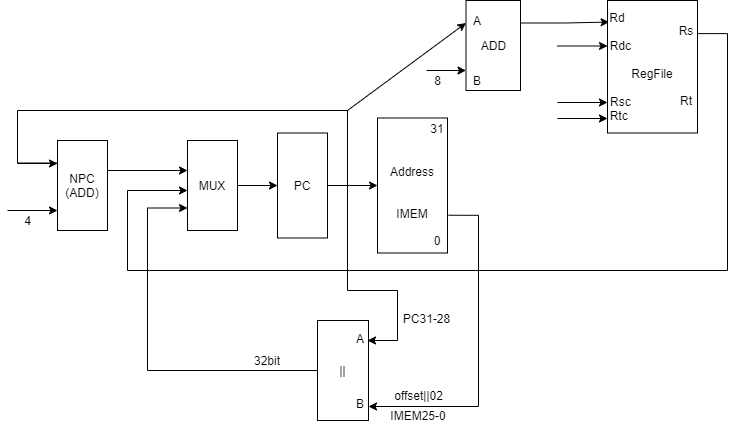
1. SLTI/SLTIU/LUI



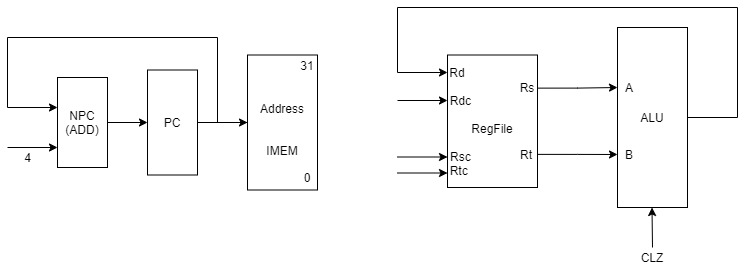
1. J



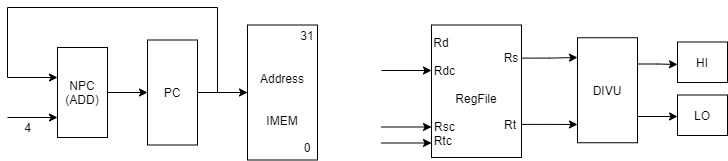
1. JAL



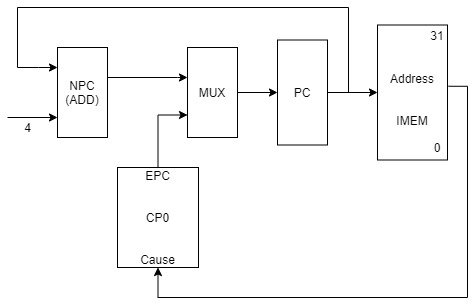
1. CLZ



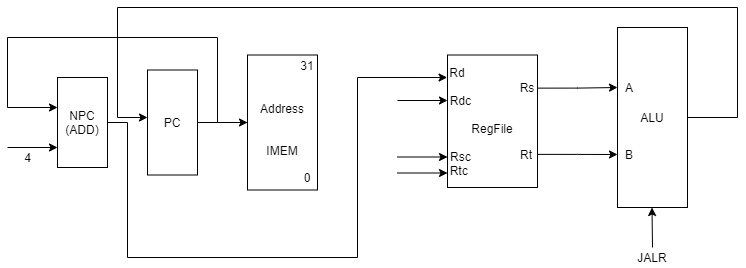
1. DIVU



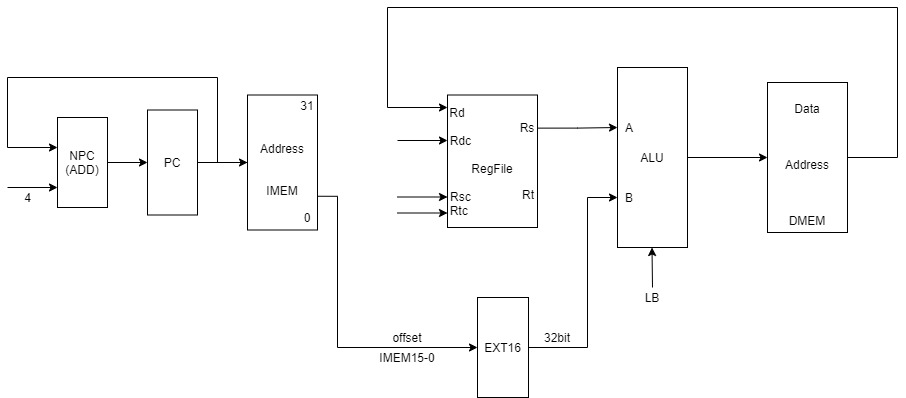
1. ERET



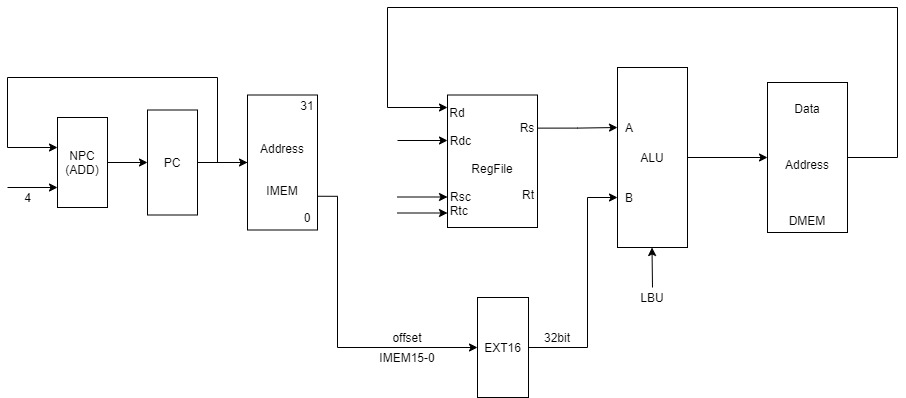
1. JALR



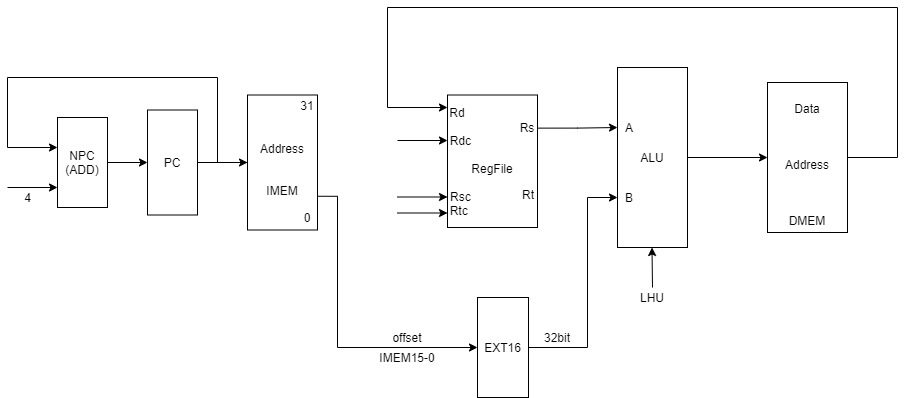
1. LB



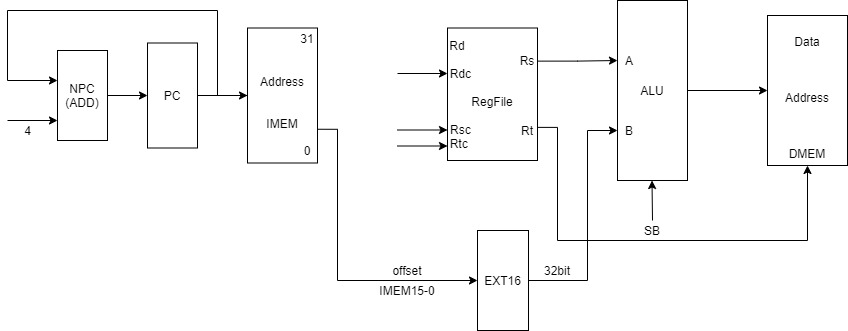
1. LBU



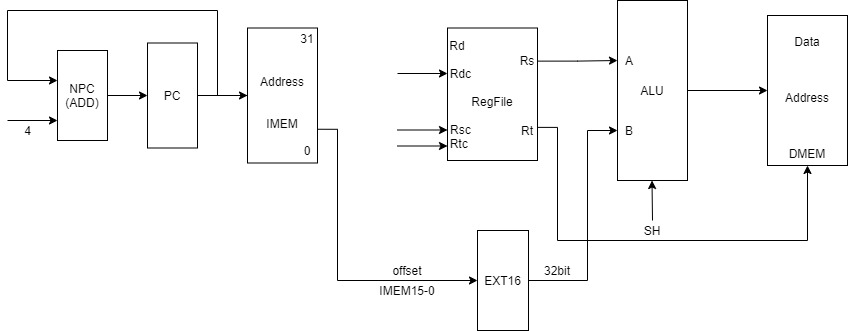
1. LHU



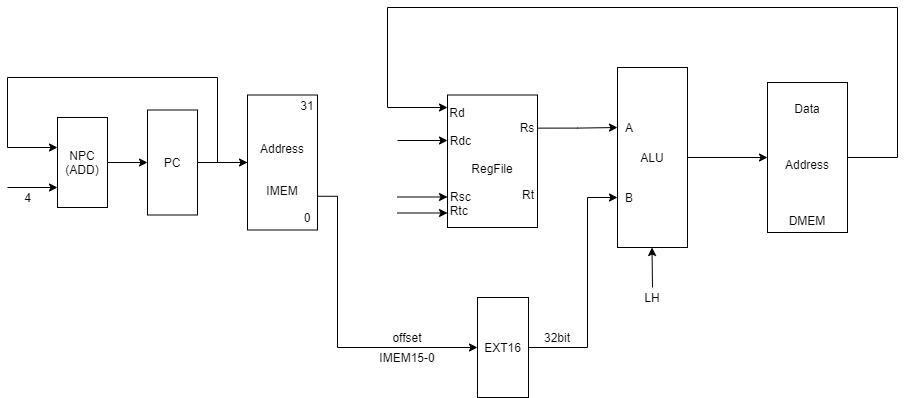
1. SB



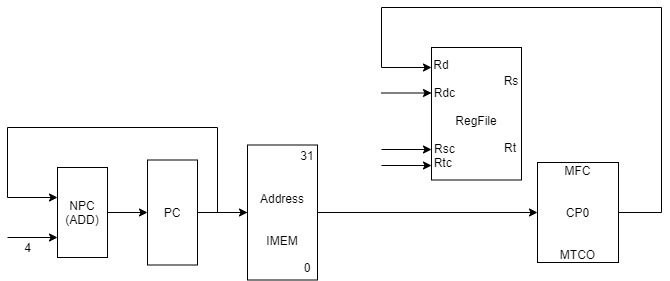
1. SH



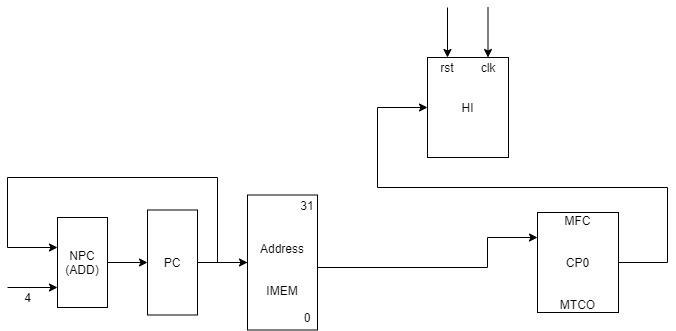
1. LH



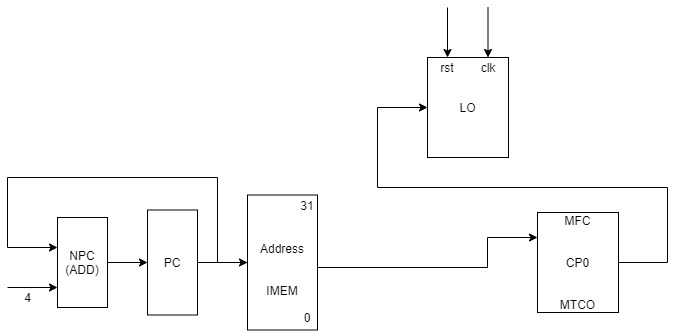
1. MFC0



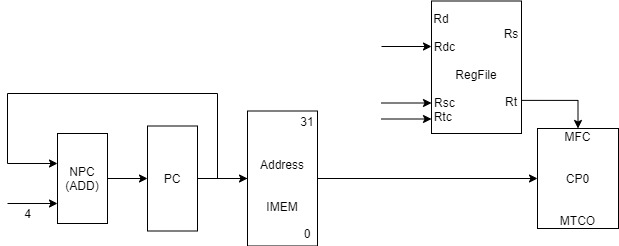
1. MFHI



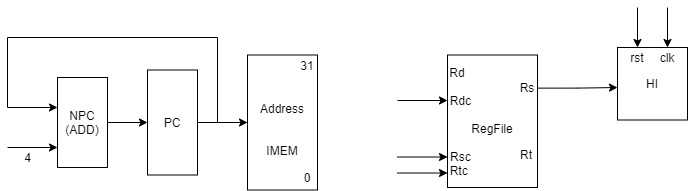
1. MFLO



1. MTC0



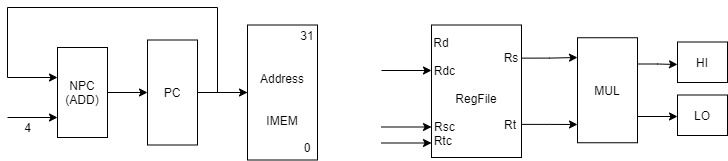
1. MTHI



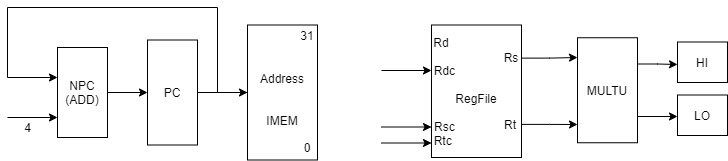
1. MTLO



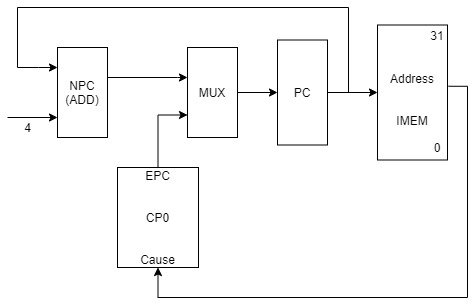
1. MUL



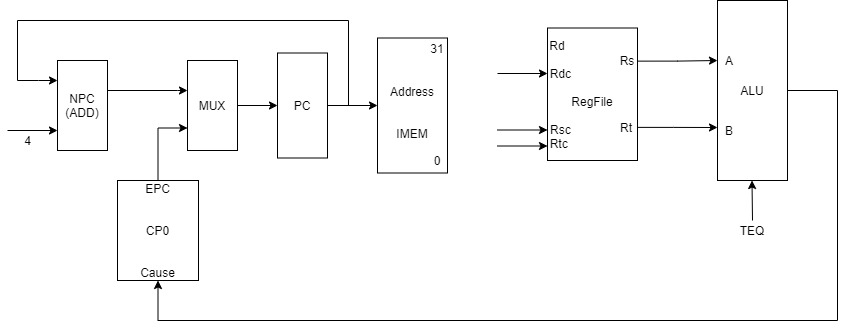
1. MULTU



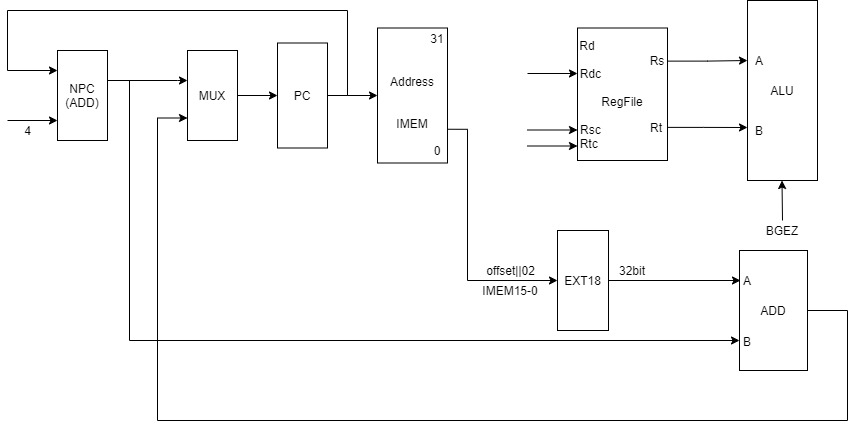
1. SYSCALL



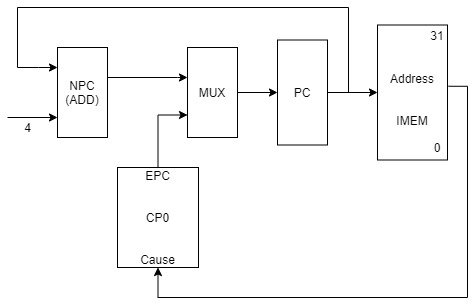
1. TEQ



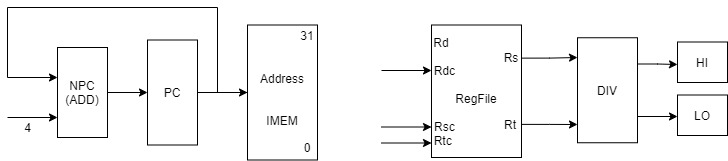
1. BGEZ



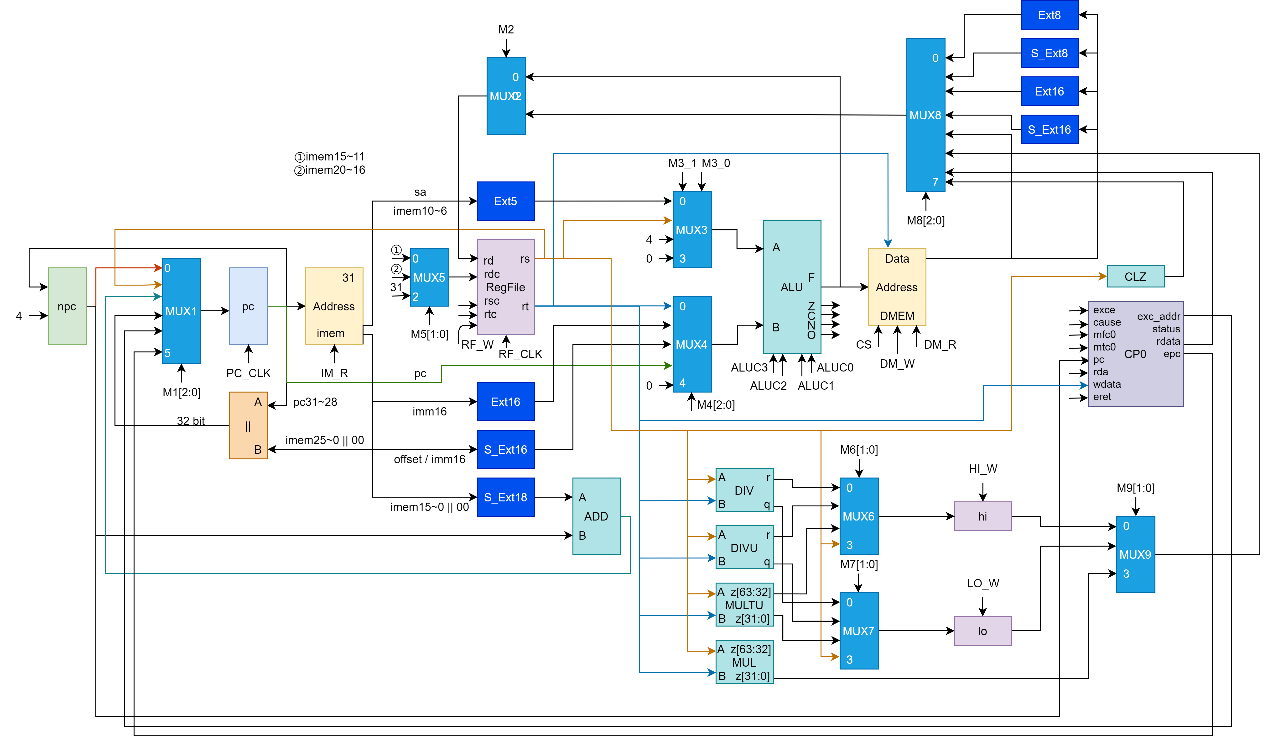
1. BREAK



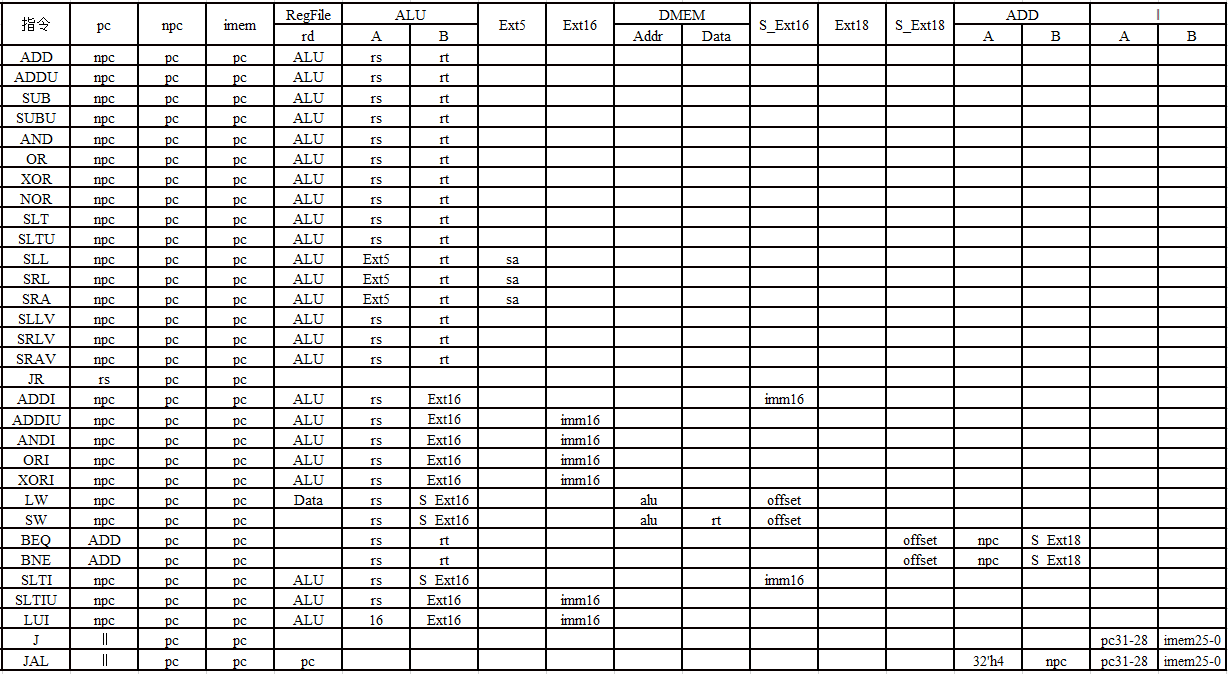
1. DIV



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

CLZ：前置0个数计数器

HI：HI寄存器

LO：LO寄存器

MUL：有符号乘法器

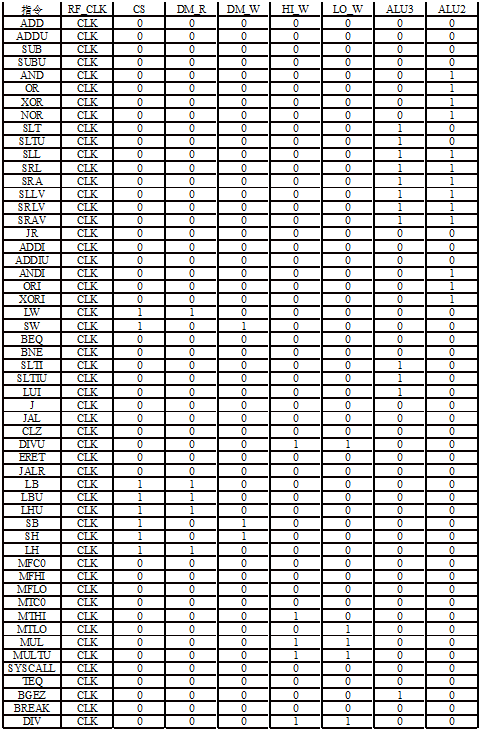
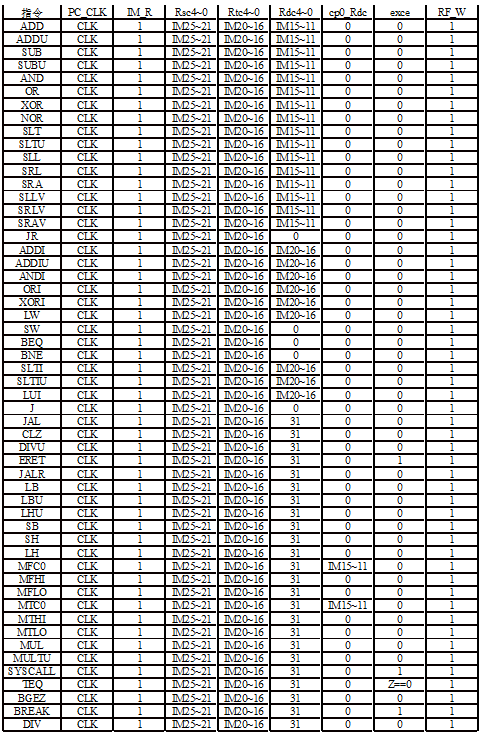
MULTU：无符号乘法器

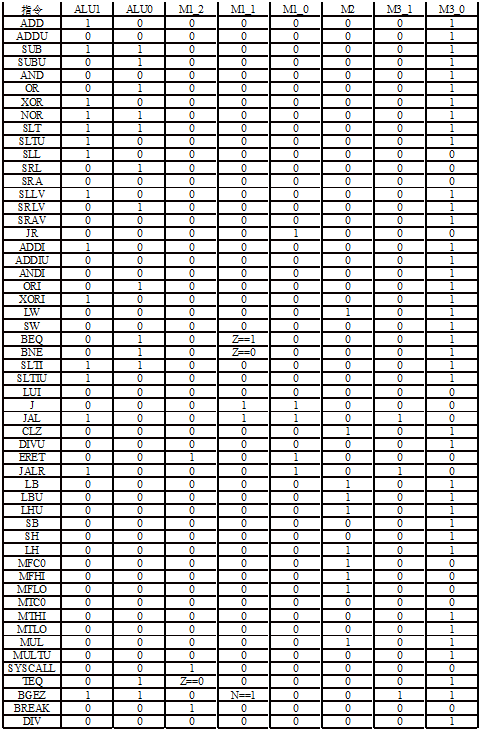
DIV：有符号除法器

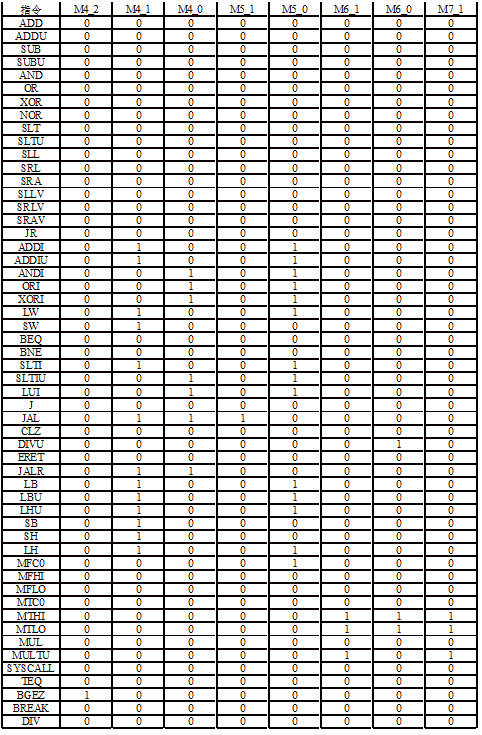
DIVU：无符号除法器

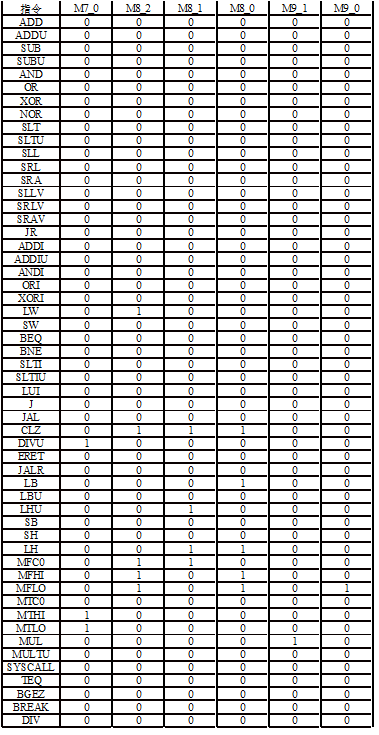
## CPU控制器设计

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

CP0\_Rdc：CP0中rd寄存器选择输入控制端

EXCE：异常处理标志

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

HI\_W：HI寄存器写信号

LO\_W：LO寄存器写信号

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\_2：MUX1选择器控制端2

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\_2：MUX4选择器控制端2

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

M6\_1：MUX6选择器控制端1

M6\_0：MUX6选择器控制端0

M7\_1：MUX7选择器控制端1

M7\_0：MUX7选择器控制端0

M8\_2：MUX8选择器控制端2

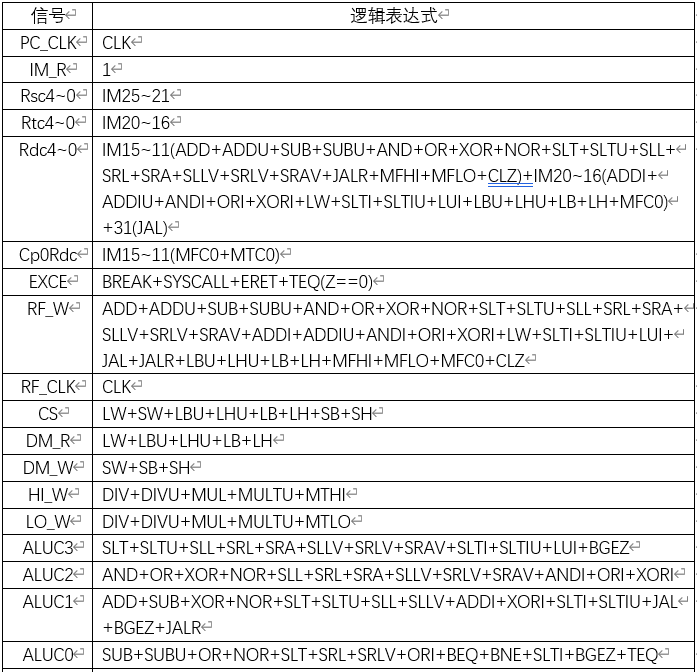
M8\_1：MUX8选择器控制端1

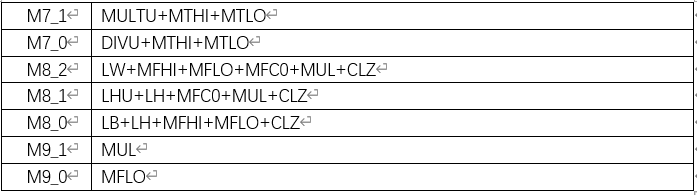
M8\_0：MUX8选择器控制端0

M9\_1：MUX9选择器控制端1

M9\_0：MUX9选择器控制端0

各微指令逻辑表达式如下图：



1. 后仿真与下板

## 4.1 后仿真功能说明

run behavioral simulation-----行为级仿真，行为级别的仿真通常也说功能仿真。

post-synthesis function simulation-----综合后的功能仿真。

post-synthesis timing simulation-----综合后带时序信息的仿真，综合后带时序信息的仿真比较接近于真实的时序。

post-implementation function simulation-----布线后的功能仿真。

post-implementation timing simulation-----（布局布线后的仿真） 执行后的时序仿真，该仿真时最接近真实的时序波形

前仿真即为行为级仿真，不涉及时序信息。后仿真即为综合后、布线后的带有时序信息的仿真。

## 4.2 后仿真代码修改

### 4.2.1 顶层模块接口更改

module sccomp\_dataflow(

    input clk\_in,

    input reset,

    output [7:0] o\_seg,

    output [7:0] o\_sel

    //output [31:0] inst,

    //output [31:0] pc

    );

由于后仿真需要进行管脚设置，需输出的值为七段数码管相关变量，故添加七段数码管段选与位选信号。

### 4.2.2 对系统时钟进行分频操作

reg [21:0] cnt;

    always @ (posedge clk\_in, posedge reset)

      if (reset)

        cnt <= 0;

      else

        cnt <= cnt + 1'b1;

wire new\_clk=cnt[21];

降低CPU主频控制pc值在七段数码管上的显示速度。

### 4.2.3 七段数码管模块

module seg7x16(

    input clk,

     input reset,

     input cs,

     input [31:0] i\_data,

     output [7:0] o\_seg,

     output [7:0] o\_sel

    );

    reg [14:0] cnt;

     always @ (posedge clk, posedge reset)

      if (reset)

        cnt <= 0;

      else

        cnt <= cnt + 1'b1;

    wire seg7\_clk = cnt[14];

     reg [2:0] seg7\_addr;

     always @ (posedge seg7\_clk, posedge reset)

       if(reset)

          seg7\_addr <= 0;

        else

          seg7\_addr <= seg7\_addr + 1'b1;

     reg [7:0] o\_sel\_r;

     always @ (\*)

       case(seg7\_addr)

          7 : o\_sel\_r = 8'b01111111;

          6 : o\_sel\_r = 8'b10111111;

          5 : o\_sel\_r = 8'b11011111;

          4 : o\_sel\_r = 8'b11101111;

          3 : o\_sel\_r = 8'b11110111;

          2 : o\_sel\_r = 8'b11111011;

          1 : o\_sel\_r = 8'b11111101;

          0 : o\_sel\_r = 8'b11111110;

        endcase

     reg [31:0] i\_data\_store;

     always @ (posedge clk, posedge reset)

       if(reset)

          i\_data\_store <= 0;

        else if(cs)

          i\_data\_store <= i\_data;

     reg [7:0] seg\_data\_r;

     always @ (\*)

       case(seg7\_addr)

          0 : seg\_data\_r = i\_data\_store[3:0];

          1 : seg\_data\_r = i\_data\_store[7:4];

          2 : seg\_data\_r = i\_data\_store[11:8];

          3 : seg\_data\_r = i\_data\_store[15:12];

          4 : seg\_data\_r = i\_data\_store[19:16];

          5 : seg\_data\_r = i\_data\_store[23:20];

          6 : seg\_data\_r = i\_data\_store[27:24];

          7 : seg\_data\_r = i\_data\_store[31:28];

        endcase

     reg [7:0] o\_seg\_r;

     always @ (posedge clk, posedge reset)

       if(reset)

          o\_seg\_r <= 8'hff;

        else

          case(seg\_data\_r)

            4'h0 : o\_seg\_r <= 8'hC0;

          4'h1 : o\_seg\_r <= 8'hF9;

          4'h2 : o\_seg\_r <= 8'hA4;

          4'h3 : o\_seg\_r <= 8'hB0;

          4'h4 : o\_seg\_r <= 8'h99;

          4'h5 : o\_seg\_r <= 8'h92;

          4'h6 : o\_seg\_r <= 8'h82;

          4'h7 : o\_seg\_r <= 8'hF8;

          4'h8 : o\_seg\_r <= 8'h80;

          4'h9 : o\_seg\_r <= 8'h90;

          4'hA : o\_seg\_r <= 8'h88;

          4'hB : o\_seg\_r <= 8'h83;

          4'hC : o\_seg\_r <= 8'hC6;

          4'hD : o\_seg\_r <= 8'hA1;

          4'hE : o\_seg\_r <= 8'h86;

          4'hF : o\_seg\_r <= 8'h8E;

          endcase

     assign o\_sel = o\_sel\_r;

     assign o\_seg = o\_seg\_r;

endmodule

## 4.3 下板操作

完成后仿真后便可进行下板验证，由于七段数码管选择输出程序计数器pc的值，故开发板七段数码管处可稳定显示计数器pc的值。可以通过对时钟分频控制pc在数码管中的显示速度。

1. 模块建模

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

`timescale 1ns / 1ps

module sccomp\_dataflow(

    input clk\_in,

    input reset,

    output [31:0] inst,

    output [31:0] pc

    );

    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 instr\_addr = pc;

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

    imen 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

## 5.2 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;

    reg [31:0] temp;

    always @(posedge clk)

    begin

        temp <= rs;

    end

    //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;

    //DIV

    wire DIV\_start;

    wire [31:0] DIV\_dividend;

    wire [31:0] DIV\_divisor;

    wire [31:0] DIV\_q;

    wire [31:0] DIV\_r;

    wire DIV\_busy;

    //DIVU

    wire DIVU\_start;

    wire [31:0] DIVU\_dividend;

    wire [31:0] DIVU\_divisor;

    wire [31:0] DIVU\_q;

    wire [31:0] DIVU\_r;

    wire DIVU\_busy;

    //MULT

    wire [31:0] MULT\_a;

    wire [31:0] MULT\_b;

    wire [63:0] MULT\_z;

    //multu

    wire [31:0] multu\_a;

    wire [31:0] multu\_b;

    wire [63:0] multu\_z;

    //HI

    wire [31:0] HI\_data\_in;

    wire [31:0] HI\_data\_out;

    wire HI\_W;

    //LO

    wire [31:0] LO\_data\_in;

    wire [31:0] LO\_data\_out;

    wire LO\_W;

    //CP0

    wire [31:0] CP0\_pc;

    wire [4:0] CP0\_Rd;

    wire [31:0] CP0\_wdata;

    wire CP0\_exception;

    wire [4:0] CP0\_cause;

    wire [31:0] CP0\_rdata;

    wire [31:0] CP0\_status;

    wire [31:0] CP0\_epc;

    wire [31:0] CP0\_exc\_addr;

    //Iram

    //wire [31:0] IM;

    //EXT

    wire [31:0] Ext5;

    wire [31:0] Ext16;

    wire [31:0] S\_Ext16;

    wire [31:0] S\_Ext18;

    wire [7:0] DExt8in;

    wire [7:0] DS\_Ext8in;

    wire [15:0] DExt16in;

    wire [15:0] DS\_Ext16in;

    wire [31:0] DExt8;

    wire [31:0] DS\_Ext8;

    wire [31:0] DExt16;

    wire [31:0] DS\_Ext16;

    //MUX

    wire M1\_2;

    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\_2;

    wire M4\_1;

    wire M4\_0;

    wire [31:0] M4\_out;

    wire M5\_1;

    wire M5\_0;

    wire [4:0] M5\_out;

    wire M6\_1;

    wire M6\_0;

    wire [31:0] M6\_out;

    wire M7\_1;

    wire M7\_0;

    wire [31:0] M7\_out;

    wire M8\_2;

    wire M8\_1;

    wire M8\_0;

    wire [31:0] M8\_out;

    wire M9\_1;

    wire M9\_0;

    wire [31:0] M9\_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;

    //CLZ

    wire [31:0] CLZ\_in;

    wire [31:0] CLZ\_out;

    //31条指令

    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);

    //cpu54剩余指令

    wire CLZ = (IM[31:26]==6'b011100) && (IM[5:0]==6'b100000);

    wire DIVU = (IM[31:26]==6'b0) && (IM[5:0]==6'b011011);

    wire ERET = (IM[31:26]==6'b010000) && (IM[25:21]==6'b10000) && (IM[5:0]==6'b011000);

    wire JALR = (IM[31:26]==6'b0) && (IM[20:16]==5'b0) && (IM[5:0]==6'b001001);

    wire LB = (IM[31:26]==6'b100000);

    wire LBU = (IM[31:26]==6'b100100);

    wire LHU = (IM[31:26]==6'b100101);

    wire SB = (IM[31:26]==6'b101000);

    wire SH = (IM[31:26]==6'b101001);

    wire LH = (IM[31:26]==6'b100001);

    wire MFC0 = (IM[31:26]==6'b010000) && (IM[25:21]==5'b0) && (IM[5:0]==6'b0);

    wire MFHI = (IM[31:26]==6'b0) && (IM[5:0]==6'b010000);

    wire MFLO = (IM[31:26]==6'b0) && (IM[5:0]==6'b010010);

    wire MTC0 = (IM[31:26]==6'b010000) && (IM[25:21]==5'b00100) && (IM[5:0]==6'b0);

    wire MTHI = (IM[31:26]==6'b0) && (IM[5:0]==6'b010001);

    wire MTLO = (IM[31:26]==6'b0) && (IM[5:0]==6'b010011);

    wire MUL = (IM[31:26]==6'b011100) && (IM[5:0]==6'b000010);

    wire MULTU = (IM[31:26]==6'b0) && (IM[5:0]==6'b011001);

    wire SYSCALL = (IM[31:26]==6'b0) && (IM[5:0]==6'b001100);

    wire TEQ = (IM[31:26]==6'b0) && (IM[5:0]==6'b110100);

    wire BGEZ = (IM[31:26]==6'b000001) && (IM[20:16]==5'b00001);

    wire BREAK = (IM[31:26]==6'b0) && (IM[5:0]==6'b001101);

    wire DIV = (IM[31:26]==6'b0) && (IM[5:0]==6'b011010);

    //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||JALR||LBU||LHU||

                  LB||LH||MFHI||MFLO||MFC0||CLZ||MUL;

    assign RF\_CLK = clk;

    //alu

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

    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||BGEZ||JALR;

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

    assign alu\_a = M3\_out;

    assign alu\_b = M4\_out;

    //MUX

    assign M1\_2 = BREAK||SYSCALL||ERET||(TEQ && (zero == 0));

    assign M1\_1 = (BEQ && (zero == 1))||(BNE && (zero == 0))||J||JAL||(BGEZ && (negative == 1'b0));

    assign M1\_0 = JR||J||JAL||JALR||ERET;

    assign M2 = LW||LBU||LHU||LB||LH||MFHI||MFLO||MFC0||CLZ||MUL;

    assign M3\_1 = JAL||JALR;

    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||DIV||DIVU||MUL||MULTU||

                BGEZ||LBU||LHU||LB||LH||SB||SH||MTHI||MTLO||CLZ||TEQ;

    assign M4\_2 = BGEZ;

    assign M4\_1 = ADDI||ADDIU||LW||SW||SLTI||JAL||JALR||LBU||LHU||LB||LH||SB||SH;

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

    assign M5\_1 = JAL;

    assign M5\_0 = ADDI||ADDIU||ANDI||ORI||XORI||LW||SLTI||SLTIU||LUI||LBU||LHU||LB||LH||MFC0;

    assign M6\_1 = MTHI||MTLO||MULTU;

    assign M6\_0 = DIVU||MTHI||MTLO;

    assign M7\_1 = MTHI||MTLO||MULTU;

    assign M7\_0 = DIVU||MTHI||MTLO;

    assign M8\_2 = LW||MFHI||MFLO||MFC0||CLZ||MUL;

    assign M8\_1 = LHU||LH||MFC0||CLZ;

    assign M8\_0 = LB||LH||MFHI||MFLO||CLZ||MUL;

    assign M9\_1 = MUL;

    assign M9\_0 = MFLO;

    //Dram

    assign CS = LW||SW||LBU||LHU||LB||LH||SB||SH;

    assign DM\_R = LW||LBU||LHU||LB||LH;

    assign DM\_W = SW||SB||SH;

    assign DM\_addr = alu\_c;

    assign DM\_WData = (SH && DM\_addr[1]==1'b0) ? {DM\_RData[31:16], rt[15:0]} : (SH && DM\_addr[1]==1'b1) ? {rt[15:0], DM\_RData[15:0]}:(SB && DM\_addr[1:0]==2'b00) ? {DM\_RData[31:8], rt[7:0]} : (SB && DM\_addr[1:0]==2'b01) ? {DM\_RData[31:15], rt[7:0], DM\_RData[7:0]}:(SB && DM\_addr[1:0]==2'b10) ?{DM\_RData[31:24], rt[7:0], DM\_RData[15:0]}:(SB && DM\_addr[1:0]==2'b11) ?{rt[7:0], DM\_RData[23:0]}: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;

    //CLZ

    assign CLZ\_in = rs;

    //DIV

    assign DIV\_start = DIV&&~DIV\_busy;

    assign DIV\_dividend = rs;

    assign DIV\_divisor = rt;

    //DIVU

    assign DIVU\_start = DIVU&&~DIVU\_busy;

    assign DIVU\_dividend = rs;

    assign DIVU\_divisor = rt;

    //MULT

    assign MULT\_a = rs;

    assign MULT\_b = rt;

    //multu

    assign multu\_a = rs;

    assign multu\_b = rt;

    //HI

    assign HI\_data\_in = M6\_out;

    assign HI\_W = DIV||DIVU||MULTU||MTHI;

    //LO

    assign LO\_data\_in = M7\_out;

    assign LO\_W = DIV||DIVU||MULTU||MTLO;

    //CP0

    assign CP0\_pc = NPC;

    assign CP0\_Rd = IM[15:11];

    assign CP0\_wdata = rt;

    assign CP0\_exception = BREAK||SYSCALL||(TEQ && (zero == 0));

    assign CP0\_cause = BREAK ? 5'b01001:SYSCALL ? 5'b01000:TEQ ? 5'b01101:0;

    //assign NPC = PC\_out + 4;

    assign NPC = (DIV\_busy||DIVU\_busy) ? PC\_out : (PC\_out + 4);

    //EXT

    assign DS\_Ext8in = (LB && DM\_addr[1:0]==2'b00) ? DM\_RData[7:0]:(LB && DM\_addr[1:0]==2'b01) ? DM\_RData[15:8]:(LB && DM\_addr[1:0]==2'b10) ? DM\_RData[23:16]:(LB && DM\_addr[1:0]==2'b11) ? DM\_RData[31:24]:8'b0;

    assign DExt8in = (LBU && DM\_addr[1:0]==2'b00) ? DM\_RData[7:0]:(LBU && DM\_addr[1:0]==2'b01) ? DM\_RData[15:8]:(LBU && DM\_addr[1:0]==2'b10) ? DM\_RData[23:16]:(LBU && DM\_addr[1:0]==2'b11) ? DM\_RData[31:24]:8'b0;

    assign DS\_Ext16in = (LH && DM\_addr[1]==1'b0) ? DM\_RData[15:0]:(LH && DM\_addr[1]==1'b1) ? DM\_RData[31:16]:16'b0;

    assign DExt16in = (LHU && DM\_addr[1]==1'b0) ? DM\_RData[15:0]:(LHU && DM\_addr[1]==1'b1) ? DM\_RData[31:16]:16'b0;

    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);

    Ext8 DE8(DExt8in,DExt8);

    S\_Ext8 S\_DE8(DS\_Ext8in,DS\_Ext8);

    Ext16 DE16(DExt16in,DExt16);

    S\_Ext16 S\_DE16(DS\_Ext16in,DS\_Ext16);

    CLZ clz(CLZ\_in,CLZ\_out);

    MUX8 MUX1(NPC,temp,ADD\_C,Connect\_C,CP0\_exc\_addr,CP0\_epc,CP0\_rdata,0,M1\_2,M1\_1,M1\_0,M1\_out);

    assign M2\_out = M2==0 ? alu\_c:M8\_out;

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

    MUX8 MUX4(rt,Ext16,S\_Ext16,PC\_out,0,0,0,0,M4\_2,M4\_1,M4\_0,M4\_out);

    MUX\_5bit MUX5(IM[15:11],IM[20:16],5'b11111,0,M5\_1,M5\_0,M5\_out);

    MUX MUX6(DIV\_r,DIVU\_r,multu\_z[63:32],rs,M6\_1,M6\_0,M6\_out);

    MUX MUX7(DIV\_q,DIVU\_q,multu\_z[31:0],rs,M7\_1,M7\_0,M7\_out);

    MUX MUX9(HI\_data\_out,LO\_data\_out,MULT\_z[31:0],0,M9\_1,M9\_0,M9\_out);

    MUX8 MUX8(DExt8,DS\_Ext8,DExt16,DS\_Ext16,DM\_RData,M9\_out,CP0\_rdata,CLZ\_out,M8\_2,M8\_1,M8\_0,M8\_out);

    MULT MULT(clk,rst,MULT\_a,MULT\_b,MULT\_z);

    multu multu(clk,rst,multu\_a,multu\_b,multu\_z);

    DIV div(DIV\_dividend,DIV\_divisor,DIV\_start,clk,rst,DIV\_q,DIV\_r,DIV\_busy);

    DIVU divu(DIVU\_dividend,DIVU\_divisor,DIVU\_start,clk,rst,DIVU\_q,DIVU\_r,DIVU\_busy);

    pcreg pcreg\_HI(clk,rst,HI\_W,HI\_data\_in,HI\_data\_out);

    pcreg pcreg\_LO(clk,rst,LO\_W,LO\_data\_in,LO\_data\_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'b1,RF\_W,rsc,rtc,rdc,rd,rs,rt);

    CP0 CP0(clk,rst,MFC0,MTC0,CP0\_pc,CP0\_Rd,CP0\_wdata,CP0\_exception,ERET,CP0\_cause,CP0\_rdata,CP0\_status,CP0\_epc,CP0\_exc\_addr);

endmodule

## 5.3 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

## 5.4 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 [1023:0];

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

    always @ (posedge clk)

    begin

        if(ena && (DM\_W))

            RAM[addr] <= data\_in;

    end

endmodule

## 5.5 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

## 5.6 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

## 5.7 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

## 5.8 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

## 5.9 MUX8 //八选一数据选择器

module MUX8(

    input [31:0] iC0,

    input [31:0] iC1,

    input [31:0] iC2,

    input [31:0] iC3,

    input [31:0] iC4,

    input [31:0] iC5,

    input [31:0] iC6,

    input [31:0] iC7,

    input iS2,

    input iS1,

    input iS0,

    output [31:0] oZ

    );

reg[31:0] tmpt;

always@(\*)

begin

    if({iS2,iS1,iS0} == 3'b000)

        tmpt = iC0;

    else if({iS2,iS1,iS0} == 3'b001)

        tmpt = iC1;

    else if({iS2,iS1,iS0} == 3'b010)

        tmpt = iC2;

    else if({iS2,iS1,iS0} == 3'b011)

        tmpt = iC3;

    else if({iS2,iS1,iS0} == 3'b100)

        tmpt = iC4;

    else if({iS2,iS1,iS0} == 3'b101)

        tmpt = iC5;

    else if({iS2,iS1,iS0} == 3'b110)

        tmpt = iC6;

    else

        tmpt = iC7;

end

assign oZ = tmpt;

endmodule

## 5.10 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

## 5.11 Ext8 //将8位数据无符号扩展为32位

module Ext8(

    input [7:0] data\_in,

    output [31:0] data\_out

    );

    assign data\_out = {24'b0,data\_in};

endmodule

## 5.12 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

## 5.13 S\_Ext8 //将8位数据有符号扩展为32位

module S\_Ext8(

    input [7:0] data\_in,

    output [31:0] data\_out

    );

    assign data\_out = {{24{data\_in[7]}},data\_in};

endmodule

## 5.14 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

## 5.15 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

## 5.16 MULT//有符号乘法器，将两数乘积扩展为有符号64位输出

module MULT(

input clk, //乘法器时钟信号

input reset, //复位信号，低电平有效

input [31:0] a, //输入数 a(被乘数)

input [31:0] b, //输入数 b（乘数）

output [63:0] z //乘积输出 z

);

  reg [63:0] temp;

  wire [31:0] a0,b0;//a，b绝对值

  reg [31:0] a1=0,b1=0;//时钟时刻a，b

  reg as=0,bs=0,zs=0;

  reg [63:0] stored00=0; reg [63:0] stored01=0; reg [63:0] stored02=0; reg [63:0] stored03=0; reg [63:0] stored04=0; reg [63:0] stored05=0; reg [63:0] stored06=0; reg [63:0] stored07=0;

  reg [63:0] stored08=0; reg [63:0] stored09=0; reg [63:0] stored10=0; reg [63:0] stored11=0; reg [63:0] stored12=0; reg [63:0] stored13=0; reg [63:0] stored14=0; reg [63:0] stored15=0;

  reg [63:0] stored16=0; reg [63:0] stored17=0; reg [63:0] stored18=0; reg [63:0] stored19=0; reg [63:0] stored20=0; reg [63:0] stored21=0; reg [63:0] stored22=0; reg [63:0] stored23=0;

  reg [63:0] stored24=0; reg [63:0] stored25=0; reg [63:0] stored26=0; reg [63:0] stored27=0; reg [63:0] stored28=0; reg [63:0] stored29=0; reg [63:0] stored30=0; reg [63:0] stored31=0;

 reg [63:0] add00\_01, add02\_03, add04\_05, add06\_07, add08\_09, add10\_11, add12\_13, add14\_15,add16\_17, add18\_19, add20\_21, add22\_23, add24\_25, add26\_27, add28\_29, add30\_31;

 reg [63:0] add00\_03, add04\_07, add08\_11, add12\_15, add16\_19, add20\_23, add24\_27, add28\_31;

 reg [63:0] add00\_07, add08\_15, add16\_23, add24\_31;

 reg [63:0] add00\_15, add16\_31;

  assign a0 = as ? (~a1+1) : a1;

  assign b0 = bs ? (~b1+1) : b1;

  assign z = zs ? (~temp+1):temp;

  always @(\*)

  begin

   if(reset)

   begin

     temp<=0;  as=0;bs=0;zs=0; a1=0;b1=0;

     stored00<=0; stored01<=0; stored02<=0; stored03<=0; stored04<=0; stored05<=0; stored06<=0; stored07<=0;

     stored08<=0; stored09<=0; stored10<=0; stored11<=0; stored12<=0; stored13<=0; stored14<=0; stored15<=0;

     stored16<=0; stored17<=0; stored18<=0; stored19<=0; stored20<=0; stored21<=0; stored22<=0; stored23<=0;

     stored24<=0; stored25<=0; stored26<=0; stored27<=0; stored28<=0; stored29<=0; stored30<=0; stored31<=0;

     add00\_01 <= 0; add02\_03 <= 0; add04\_05 <= 0; add06\_07 <= 0; add08\_09 <= 0; add10\_11 <= 0; add12\_13 <= 0; add14\_15 <= 0;

     add16\_17 <= 0; add18\_19 <= 0; add20\_21 <= 0; add22\_23 <= 0; add24\_25 <= 0; add26\_27 <= 0; add28\_29 <= 0; add30\_31 <= 0;

     add00\_03 <= 0; add04\_07 <= 0; add08\_11 <= 0; add12\_15 <= 0; add16\_19 <= 0; add20\_23 <= 0; add24\_27 <= 0; add28\_31 <= 0;

     add00\_07 <= 0; add08\_15 <= 0; add16\_23 <= 0; add24\_31 <= 0;

     add00\_15 <= 0; add16\_31 <= 0;

   end

   else

   begin

    as <= a[31];    bs <= b[31];

    a1 <= a;        b1 <= b;

    stored00 <= b0[0]? {32'b0, a0} :64'b0;         stored01 <= b0[1]? {31'b0, a0, 1'b0} :64'b0;

    stored02 <= b0[2]? {30'b0, a0, 2'b0} :64'b0;   stored03 <= b0[3]? {29'b0, a0, 3'b0} :64'b0;

    stored04 <= b0[4]? {28'b0, a0, 4'b0} :64'b0;   stored05 <= b0[5]? {27'b0, a0, 5'b0} :64'b0;

    stored06 <= b0[6]? {26'b0, a0, 6'b0} :64'b0;   stored07 <= b0[7]? {25'b0, a0, 7'b0} :64'b0;

    stored08 <= b0[8]? {24'b0, a0, 8'b0} :64'b0;   stored09 <= b0[9]? {23'b0, a0, 9'b0} :64'b0;

    stored10 <= b0[10]? {22'b0, a0, 10'b0} :64'b0; stored11 <= b0[11]? {21'b0, a0, 11'b0} :64'b0;

    stored12 <= b0[12]? {20'b0, a0, 12'b0} :64'b0; stored13 <= b0[13]? {19'b0, a0, 13'b0} :64'b0;

    stored14 <= b0[14]? {18'b0, a0, 14'b0} :64'b0; stored15 <= b0[15]? {17'b0, a0, 15'b0} :64'b0;

    stored16 <= b0[16]? {16'b0, a0, 16'b0} :64'b0; stored17 <= b0[17]? {15'b0, a0, 17'b0} :64'b0;

    stored18 <= b0[18]? {14'b0, a0, 18'b0} :64'b0; stored19 <= b0[19]? {13'b0, a0, 19'b0} :64'b0;

    stored20 <= b0[20]? {12'b0, a0, 20'b0} :64'b0; stored21 <= b0[21]? {11'b0, a0, 21'b0} :64'b0;

    stored22 <= b0[22]? {10'b0, a0, 22'b0} :64'b0; stored23 <= b0[23]? {9'b0, a0, 23'b0} :64'b0;

    stored24 <= b0[24]? {8'b0, a0, 24'b0} :64'b0;  stored25 <= b0[25]? {7'b0, a0, 25'b0} :64'b0;

    stored26 <= b0[26]? {6'b0, a0, 26'b0} :64'b0;  stored27 <= b0[27]? {5'b0, a0, 27'b0} :64'b0;

    stored28 <= b0[28]? {4'b0, a0, 28'b0} :64'b0;  stored29 <= b0[29]? {3'b0, a0, 29'b0} :64'b0;

    stored30 <= b0[30]? {2'b0, a0, 30'b0} :64'b0;  stored31 <= b0[31]? {1'b0, a0, 31'b0} :64'b0;

    add00\_01 <= stored00+stored01; add02\_03 <= stored02+stored03; add04\_05 <= stored04+stored05; add06\_07 <= stored06+stored07;

    add08\_09 <= stored08+stored09; add10\_11 <= stored10+stored11; add12\_13 <= stored12+stored13; add14\_15 <= stored14+stored15;

    add16\_17 <= stored16+stored17; add18\_19 <= stored18+stored19; add20\_21 <= stored20+stored21; add22\_23 <= stored22+stored23;

    add24\_25 <= stored24+stored25; add26\_27 <= stored26+stored27; add28\_29 <= stored28+stored29; add30\_31 <= stored30+stored31;

    add00\_03 <= add00\_01+add02\_03; add04\_07 <= add04\_05+add06\_07; add08\_11 <= add08\_09+add10\_11; add12\_15 <= add12\_13+add14\_15;

    add16\_19 <= add16\_17+add18\_19; add20\_23 <= add20\_21+add22\_23; add24\_27 <= add24\_25+add26\_27; add28\_31 <= add28\_29+add30\_31;

    add00\_07 <= add00\_03+add04\_07; add08\_15 <= add08\_11+add12\_15; add16\_23 <= add16\_19+add20\_23; add24\_31 <= add24\_27+add28\_31;

    add00\_15 <= add00\_07+add08\_15; add16\_31 <= add16\_23+add24\_31;

    temp <= add00\_15+add16\_31;

    zs<=as+bs;

   end

   end

endmodule

## 5.17 MULTU//无符号乘法器，将两数乘积扩展为无符号64位输出

module multu(

input clk, //乘法器时钟信号

input reset, //复位信号，低电平有效

input [31:0] a, //输入数 a(被乘数)

input [31:0] b, //输入数 b（乘数）

output [63:0] z //乘积输出 z

);

  reg [63:0] temp;

  reg [63:0] stored00; reg [63:0] stored01; reg [63:0] stored02; reg [63:0] stored03; reg [63:0] stored04; reg [63:0] stored05; reg [63:0] stored06; reg [63:0] stored07;

  reg [63:0] stored08; reg [63:0] stored09; reg [63:0] stored10; reg [63:0] stored11; reg [63:0] stored12; reg [63:0] stored13; reg [63:0] stored14; reg [63:0] stored15;

  reg [63:0] stored16; reg [63:0] stored17; reg [63:0] stored18; reg [63:0] stored19; reg [63:0] stored20; reg [63:0] stored21; reg [63:0] stored22; reg [63:0] stored23;

  reg [63:0] stored24; reg [63:0] stored25; reg [63:0] stored26; reg [63:0] stored27; reg [63:0] stored28; reg [63:0] stored29; reg [63:0] stored30; reg [63:0] stored31;

 reg [63:0] add00\_01, add02\_03, add04\_05, add06\_07, add08\_09, add10\_11, add12\_13, add14\_15,add16\_17, add18\_19, add20\_21, add22\_23, add24\_25, add26\_27, add28\_29, add30\_31;

 reg [63:0] add00\_03, add04\_07, add08\_11, add12\_15, add16\_19, add20\_23, add24\_27, add28\_31;

 reg [63:0] add00\_07, add08\_15, add16\_23, add24\_31;

 reg [63:0] add00\_15, add16\_31;

  always @(\*)

  begin

   if(reset)

   begin

     temp<=0;

     stored00<=0; stored01<=0; stored02<=0; stored03<=0; stored04<=0; stored05<=0; stored06<=0; stored07<=0;

     stored08<=0; stored09<=0; stored10<=0; stored11<=0; stored12<=0; stored13<=0; stored14<=0; stored15<=0;

     stored16<=0; stored17<=0; stored18<=0; stored19<=0; stored20<=0; stored21<=0; stored22<=0; stored23<=0;

     stored24<=0; stored25<=0; stored26<=0; stored27<=0; stored28<=0; stored29<=0; stored30<=0; stored31<=0;

     add00\_01 <= 0; add02\_03 <= 0; add04\_05 <= 0; add06\_07 <= 0; add08\_09 <= 0; add10\_11 <= 0; add12\_13 <= 0; add14\_15 <= 0;

     add16\_17 <= 0; add18\_19 <= 0; add20\_21 <= 0; add22\_23 <= 0; add24\_25 <= 0; add26\_27 <= 0; add28\_29 <= 0; add30\_31 <= 0;

     add00\_03 <= 0; add04\_07 <= 0; add08\_11 <= 0; add12\_15 <= 0; add16\_19 <= 0; add20\_23 <= 0; add24\_27 <= 0; add28\_31 <= 0;

     add00\_07 <= 0; add08\_15 <= 0; add16\_23 <= 0; add24\_31 <= 0;

     add00\_15 <= 0; add16\_31 <= 0;

   end

   else

   begin

    stored00 <= b[0]? {32'b0, a} :64'b0;         stored01 <= b[1]? {31'b0, a, 1'b0} :64'b0;

    stored02 <= b[2]? {30'b0, a, 2'b0} :64'b0;   stored03 <= b[3]? {29'b0, a, 3'b0} :64'b0;

    stored04 <= b[4]? {28'b0, a, 4'b0} :64'b0;   stored05 <= b[5]? {27'b0, a, 5'b0} :64'b0;

    stored06 <= b[6]? {26'b0, a, 6'b0} :64'b0;   stored07 <= b[7]? {25'b0, a, 7'b0} :64'b0;

    stored08 <= b[8]? {24'b0, a, 8'b0} :64'b0;   stored09 <= b[9]? {23'b0, a, 9'b0} :64'b0;

    stored10 <= b[10]? {22'b0, a, 10'b0} :64'b0; stored11 <= b[11]? {21'b0, a, 11'b0} :64'b0;

    stored12 <= b[12]? {20'b0, a, 12'b0} :64'b0; stored13 <= b[13]? {19'b0, a, 13'b0} :64'b0;

    stored14 <= b[14]? {18'b0, a, 14'b0} :64'b0; stored15 <= b[15]? {17'b0, a, 15'b0} :64'b0;

    stored16 <= b[16]? {16'b0, a, 16'b0} :64'b0; stored17 <= b[17]? {15'b0, a, 17'b0} :64'b0;

    stored18 <= b[18]? {14'b0, a, 18'b0} :64'b0; stored19 <= b[19]? {13'b0, a, 19'b0} :64'b0;

    stored20 <= b[20]? {12'b0, a, 20'b0} :64'b0; stored21 <= b[21]? {11'b0, a, 21'b0} :64'b0;

    stored22 <= b[22]? {10'b0, a, 22'b0} :64'b0; stored23 <= b[23]? {9'b0, a, 23'b0} :64'b0;

    stored24 <= b[24]? {8'b0, a, 24'b0} :64'b0;  stored25 <= b[25]? {7'b0, a, 25'b0} :64'b0;

    stored26 <= b[26]? {6'b0, a, 26'b0} :64'b0;  stored27 <= b[27]? {5'b0, a, 27'b0} :64'b0;

    stored28 <= b[28]? {4'b0, a, 28'b0} :64'b0;  stored29 <= b[29]? {3'b0, a, 29'b0} :64'b0;

    stored30 <= b[30]? {2'b0, a, 30'b0} :64'b0;  stored31 <= b[31]? {1'b0, a, 31'b0} :64'b0;

    add00\_01 <= stored00+stored01; add02\_03 <= stored02+stored03; add04\_05 <= stored04+stored05; add06\_07 <= stored06+stored07;

    add08\_09 <= stored08+stored09; add10\_11 <= stored10+stored11; add12\_13 <= stored12+stored13; add14\_15 <= stored14+stored15;

    add16\_17 <= stored16+stored17; add18\_19 <= stored18+stored19; add20\_21 <= stored20+stored21; add22\_23 <= stored22+stored23;

    add24\_25 <= stored24+stored25; add26\_27 <= stored26+stored27; add28\_29 <= stored28+stored29; add30\_31 <= stored30+stored31;

    add00\_03 <= add00\_01+add02\_03; add04\_07 <= add04\_05+add06\_07; add08\_11 <= add08\_09+add10\_11; add12\_15 <= add12\_13+add14\_15;

    add16\_19 <= add16\_17+add18\_19; add20\_23 <= add20\_21+add22\_23; add24\_27 <= add24\_25+add26\_27; add28\_31 <= add28\_29+add30\_31;

    add00\_07 <= add00\_03+add04\_07; add08\_15 <= add08\_11+add12\_15; add16\_23 <= add16\_19+add20\_23; add24\_31 <= add24\_27+add28\_31;

    add00\_15 <= add00\_07+add08\_15; add16\_31 <= add16\_23+add24\_31;

    temp <= add00\_15+add16\_31;

   end

   end

  assign z=temp;

endmodule

## 5.18 DIV//有符号乘法器，将两有符号数相除输出商和余数

module DIV(

    input [31:0] dividend,      //被除数

    input [31:0] divisor,       //除数

    input start,                //开始运算

    input clock,                //时钟

    input reset,                //复位

    output [31:0] q,            //商

    output [31:0] r,            //余数

    output reg busy             //除法器忙标志位

    );

    reg [5:0] count;            //计数器

    reg [31:0] reg\_q;

    reg [31:0] reg\_r;

    reg [31:0] reg\_b;

    wire [31:0] reg\_r2;

    reg r\_sign;

    wire [32:0] sub\_add = r\_sign? ({reg\_r, reg\_q[31]} + {1'b0, reg\_b}):({reg\_r, reg\_q[31]} - {1'b0, reg\_b});    //加减法器

    assign reg\_r2 = r\_sign ? reg\_r + reg\_b : reg\_r;

    assign r = dividend[31] ? (~reg\_r2 + 1) : reg\_r2;

    assign q = (divisor[31]^dividend[31]) ? (~reg\_q + 1) : reg\_q;

    always @ (posedge clock or posedge reset)

    begin

        if (reset)

        begin

            count <= 0;                         //重置

            busy <= 0;

        end

        else

        begin

            if (start)                          //开始除法运算，初始化

            begin

                reg\_r <= 32'b0;

                r\_sign <= 0;

                if (dividend[31] == 1)

                    reg\_q <= ~dividend+1;

                else

                    reg\_q <= dividend;

                if(divisor[31] == 1)

                    reg\_b <= ~divisor+1;

                else

                    reg\_b <= divisor;

                count <= 0;

                busy <= 1;

            end

            else if (busy)

            begin

                reg\_r <= sub\_add[31:0];         //循环操作

                r\_sign <= sub\_add[32];          //部分余数

                reg\_q <= {reg\_q[30:0],~sub\_add[32]};

                count <= count + 1;             //计数器加一

                if(count == 31)                 //结束除法运算

                    busy <= 0;

            end

        end

    end

endmodule

## 5.19 DIVU//无符号乘法器，将两无符号数相除输出商和余数

module DIVU(

    input [31:0] dividend,          //被除数

    input [31:0] divisor,           //除数

    input start,                    //开始运算

    input clock,                    //时钟

    input reset,                    //复位

    output [31:0] q,                //商

    output [31:0] r,                //余数

    output reg busy                 //除法器忙标志位

    );

    reg [4:0] count;

    reg [31:0] reg\_q;

    reg [31:0] reg\_r;

    reg [31:0] reg\_b;

    reg r\_sign;

    wire [32:0] sub\_add = r\_sign?({reg\_r,q[31]} + {1'b0,reg\_b}):({reg\_r,q[31]} - {1'b0,reg\_b});    //加、减法器

    assign r = r\_sign? reg\_r + reg\_b : reg\_r;

    assign q = reg\_q;

    always @ (posedge clock or posedge reset)

    begin

        if (reset == 1)

        begin                                           //重置

            count <=5'b0;

            busy <= 0;

        end

        else

        begin

            if (start)

            begin                                       //开始除法运算，初始化

                reg\_r <= 32'b0;

                r\_sign <= 0;

                reg\_q <= dividend;

                reg\_b <= divisor;

                count <= 5'b0;

                busy <= 1'b1;

            end

            else if (busy)

            begin                                       //循环操作

                reg\_r <= sub\_add[31:0];                 //部分余数

                r\_sign <= sub\_add[32];                  //如果为负，下次相加

                reg\_q <= {reg\_q[30:0],~sub\_add[32]};

                count <= count +5'b1;                   //计数器加一

                if (count == 31)

                    busy <= 0;                          //结束除法运算

            end

        end

    end

endmodule

## 5.20 CLZ //实现输出数据前导0的个数

/\*module CLZ(

    input [31:0] in,

    output [31:0] out

    );

    assign out = in[31]==1 ? 0:in[30]==1 ? 1:in[29]==1 ? 2:in[28]==1 ? 3:in[27]==1 ? 4:in[26]==1 ? 5:in[25]==1 ? 6:in[24]==1 ? 7:in[23]==1 ? 8:

                in[22]==1 ? 9:in[21]==1 ? 10:in[20]==1 ? 11:in[19]==1 ? 12:in[18]==1 ? 13:in[17]==1 ? 14:in[16]==1 ? 15:in[15]==1 ? 16:in[14]==1 ? 17:

                in[13]==1 ? 18:in[12]==1 ? 19:in[11]==1 ? 20:in[10]==1 ? 21:in[9]==1 ? 22:in[8]==1 ? 23:in[7]==1 ? 24:in[6]==1 ? 25:in[5]==1 ? 26:

                in[4]==1 ? 27:in[3]==1 ? 28:in[2]==1 ? 29:in[1]==1 ? 30:in[0]==1 ? 31:32;

endmodule\*/

module CLZ(

    input [31:0]num,

    output[31:0]cnt

);

assign cnt=num[31]==1?0:num[30]==1?1:num[29]==1?2:num[28]==1?3:num[27]==1?4:num[26]==1?5:num[25]==1?6:num[24]==1?7:num[23]==1?8:num[22]==1?9:num[21]==1?10:num[20]==1?11:num[19]==1?12:num[18]==1?13:num[17]==1?14:num[16]==1?15:num[15]==1?16:num[14]==1?17:num[13]==1?18:num[12]==1?19:num[11]==1?20:num[10]==1?21:num[9]==1?22:num[8]==1?23:num[7]==1?24:num[6]==1?25:num[5]==1?26:num[4]==1?27:num[3]==1?28:num[2]==1?29:num[1]==1?30:num[0]==1?31:32;

endmodule

## 5.21 CP0 //对CPU发生中断或异常时的处理

module CP0(

    input clk,

    input rst,

    input mfc0,

    input mtc0,

    input [31:0]pc,

    input [4:0]addr,

    input [31:0]data,

    input exception,

    input eret,

    input [4:0]cause,

    output reg [31:0]CP0\_out,

    output [31:0]status,

    output [31:0]epc\_out,

    output [31:0]exc\_addr

);

    reg [31:0]memory[31:0];

    integer i;

    initial begin

        for(i = 0; i <= 31; i = i + 1)

        begin

          if(i == 12)

              memory[i] = 32'hf;

          else

                memory[i] = 32'b0;

        end

    end

    reg exception\_valid = 1'b0;

    always @(\*) begin

        if(exception & memory[12][0]) begin

            case(cause[3:0])

                4'b1000: begin //systcall

                    if(memory[12][1]) begin

                        exception\_valid = 1'b1;

                    end else begin

                        exception\_valid = 1'b0;

                    end

                end

                4'b1001: begin //break

                    if(memory[12][2]) begin

                        exception\_valid = 1'b1;

                    end else begin

                        exception\_valid = 1'b0;

                    end

                end

                4'b1101: begin //teq

                    if(memory[12][3]) begin

                        exception\_valid = 1'b1;

                    end else begin

                        exception\_valid = 1'b0;

                    end

                end

                default: begin

                    exception\_valid = 1'bx;

                end

            endcase

        end else begin

            exception\_valid = 1'b0;

        end

    end

    reg in\_exception = 1'b0;

    always @(negedge clk or posedge rst) begin

        if(rst) begin

            memory[12] <= 32'h0000000f;

            memory[13] <= 32'b0;

            memory[14] <= 32'b0;

            in\_exception <= 1'b0;

        end else begin

            if(mtc0) begin

                memory[addr] <= data;

            end else begin

                if(exception\_valid & (~in\_exception)) begin

                    memory[12] <= {memory[12][26:0], 5'b0};

                    memory[13] <= {26'b0 ,cause, 2'b0};

                    memory[14] <= pc ;//- 32'h4;

                    in\_exception <= 1'b1;

                end else if(eret & in\_exception) begin

                    memory[12] <= {5'b0, memory[12][31:5]};

                    in\_exception <= 1'b0;

                end

            end

        end

    end

    always @(\*) begin

        if(mfc0) begin

            CP0\_out = memory[addr];

        end else begin

            CP0\_out = 32'bx;

        end

    end

    wire [31:0]cause\_ = memory[13];

    assign status = memory[12];//status\_reg;

    assign epc\_out = memory[14];//epc\_reg;

    assign exc\_addr=32'h00400004;

endmodule

## 5.22 imem //指令寄存器模块，通过调用ip核实现

`timescale 1ns/1ps

(\* DowngradeIPIdentifiedWarnings = "yes" \*)

module imen (

  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("imen.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;\*/

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

        //wire [31:0] M8\_out = sc.sccpu.M8\_out;

        wire [31:0] rd = sc.sccpu.rd;

        wire [31:0] rs = sc.sccpu.rs;

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

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

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

        wire [31:0] RF\_W = sc.sccpu.RF\_W;

        wire [31:0] CP0\_rdata = sc.sccpu.CP0\_rdata;

        wire MTHI = sc.sccpu.MTHI;

        wire MFHI = sc.sccpu.MFHI;

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

        //wire M\_LO = sc.sccpu.M\_LO;

        wire LO\_W = sc.sccpu.LO\_W;

        wire [31:0] LO\_data\_in = sc.sccpu.LO\_data\_in;

        wire [63:0] MULT\_z = sc.sccpu.MULT\_z;

        wire [31:0] M9\_out = sc.sccpu.M9\_out;

        wire [31:0] M8\_out = sc.sccpu.M8\_out;

        wire [31:0] M2\_out = sc.sccpu.M2\_out;

        wire [31:0] M7\_out = sc.sccpu.M7\_out;

        wire [31:0] multu\_z = sc.sccpu.multu\_z;

        wire M8\_2 = sc.sccpu.M8\_2;

        wire M8\_1 = sc.sccpu.M8\_1;

        wire M8\_0 = sc.sccpu.M8\_0;

        wire [31:0] DExt8 = sc.sccpu.DExt8;

        wire [31:0] DS\_Ext8 = sc.sccpu.DS\_Ext8;

        wire [31:0] DExt16 = sc.sccpu.DExt16;

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

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

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

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

        wire [31:0] rt = sc.sccpu.rt;

        wire DM\_R = sc.sccpu.DM\_R;

        wire DM\_W = sc.sccpu.DM\_W;

        wire CS = sc.sccpu.CS;

        wire [31:0] DS\_Ext8in = sc.sccpu.DS\_Ext8in;

        wire LB = sc.sccpu.LB;

        wire LBU = sc.sccpu.LBU;

        wire LHU = sc.sccpu.LHU;

        wire SB = sc.sccpu.SB;

        wire SH = sc.sccpu.SH;

        wire LH = sc.sccpu.LH;

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

        wire M1\_2 = sc.sccpu.M1\_2;

        wire M1\_1 = sc.sccpu.M1\_1;

        wire M1\_0 = sc.sccpu.M1\_0;

        wire negative = sc.sccpu.negative;

        wire BGEZ = sc.sccpu.BGEZ;

        wire [31:0] M3\_out = sc.sccpu.M3\_out;

        wire [31:0] M4\_out = sc.sccpu.M4\_out;

        wire [31:0] CP0\_exc\_addr = sc.sccpu.CP0\_exc\_addr;

        integer file\_output;

        integer counter = 0;

        initial

        begin

            file\_output = $fopen("D:/MyResult.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 @(negedge clk\_in) begin

            if (start)begin

                counter  = counter + 1;

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

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

                $fdisplay(file\_output, "regfile0: %h", sc.sccpu.cpu\_ref.array\_reg[0]);

                $fdisplay(file\_output, "regfile1: %h", sc.sccpu.cpu\_ref.array\_reg[1]);

                $fdisplay(file\_output, "regfile2: %h", sc.sccpu.cpu\_ref.array\_reg[2]);

                $fdisplay(file\_output, "regfile3: %h", sc.sccpu.cpu\_ref.array\_reg[3]);

                $fdisplay(file\_output, "regfile4: %h", sc.sccpu.cpu\_ref.array\_reg[4]);

                $fdisplay(file\_output, "regfile5: %h", sc.sccpu.cpu\_ref.array\_reg[5]);

                $fdisplay(file\_output, "regfile6: %h", sc.sccpu.cpu\_ref.array\_reg[6]);

                $fdisplay(file\_output, "regfile7: %h", sc.sccpu.cpu\_ref.array\_reg[7]);

                $fdisplay(file\_output, "regfile8: %h", sc.sccpu.cpu\_ref.array\_reg[8]);

                $fdisplay(file\_output, "regfile9: %h", sc.sccpu.cpu\_ref.array\_reg[9]);

                $fdisplay(file\_output, "regfile10: %h", sc.sccpu.cpu\_ref.array\_reg[10]);

                $fdisplay(file\_output, "regfile11: %h", sc.sccpu.cpu\_ref.array\_reg[11]);

                $fdisplay(file\_output, "regfile12: %h", sc.sccpu.cpu\_ref.array\_reg[12]);

                $fdisplay(file\_output, "regfile13: %h", sc.sccpu.cpu\_ref.array\_reg[13]);

                $fdisplay(file\_output, "regfile14: %h", sc.sccpu.cpu\_ref.array\_reg[14]);

                $fdisplay(file\_output, "regfile15: %h", sc.sccpu.cpu\_ref.array\_reg[15]);

                $fdisplay(file\_output, "regfile16: %h", sc.sccpu.cpu\_ref.array\_reg[16]);

                $fdisplay(file\_output, "regfile17: %h", sc.sccpu.cpu\_ref.array\_reg[17]);

                $fdisplay(file\_output, "regfile18: %h", sc.sccpu.cpu\_ref.array\_reg[18]);

                $fdisplay(file\_output, "regfile19: %h", sc.sccpu.cpu\_ref.array\_reg[19]);

                $fdisplay(file\_output, "regfile20: %h", sc.sccpu.cpu\_ref.array\_reg[20]);

                $fdisplay(file\_output, "regfile21: %h", sc.sccpu.cpu\_ref.array\_reg[21]);

                $fdisplay(file\_output, "regfile22: %h", sc.sccpu.cpu\_ref.array\_reg[22]);

                $fdisplay(file\_output, "regfile23: %h", sc.sccpu.cpu\_ref.array\_reg[23]);

                $fdisplay(file\_output, "regfile24: %h", sc.sccpu.cpu\_ref.array\_reg[24]);

                $fdisplay(file\_output, "regfile25: %h", sc.sccpu.cpu\_ref.array\_reg[25]);

                $fdisplay(file\_output, "regfile26: %h", sc.sccpu.cpu\_ref.array\_reg[26]);

                $fdisplay(file\_output, "regfile27: %h", sc.sccpu.cpu\_ref.array\_reg[27]);

                $fdisplay(file\_output, "regfile28: %h", sc.sccpu.cpu\_ref.array\_reg[28]);

                $fdisplay(file\_output, "regfile29: %h", sc.sccpu.cpu\_ref.array\_reg[29]);

                $fdisplay(file\_output, "regfile30: %h", sc.sccpu.cpu\_ref.array\_reg[30]);

                $fdisplay(file\_output, "regfile31: %h", sc.sccpu.cpu\_ref.array\_reg[31]);

            end

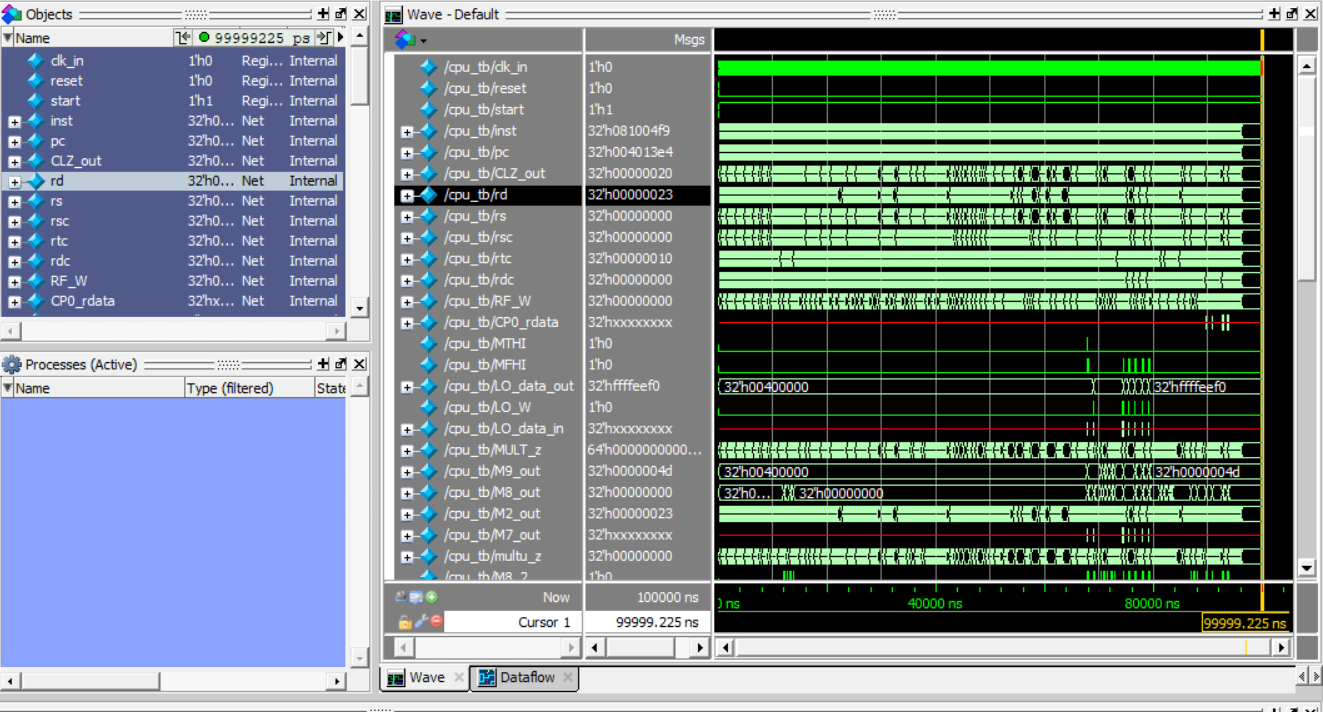
          end

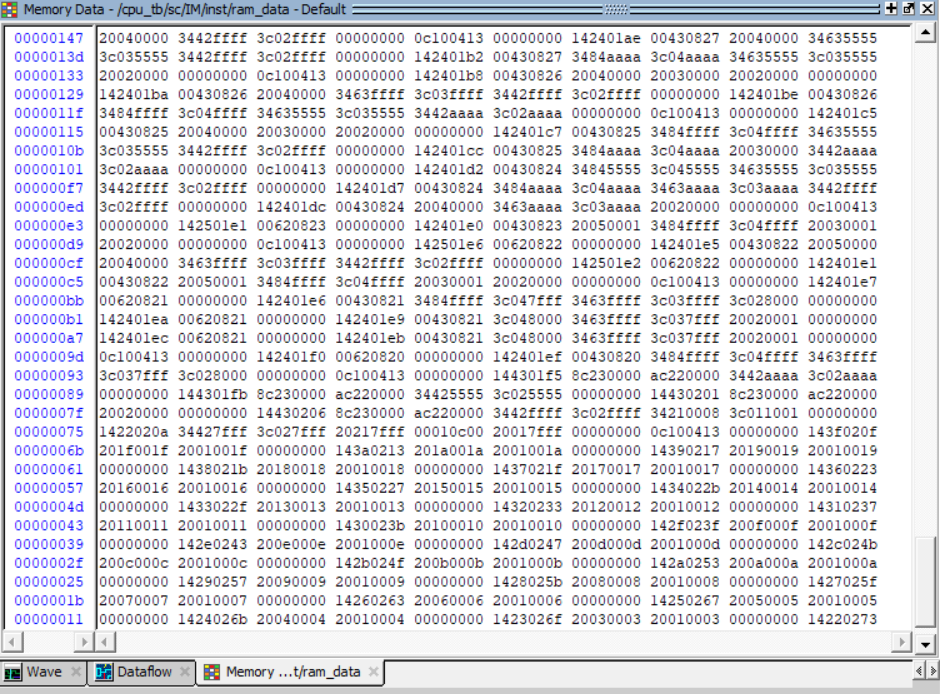
endmodule

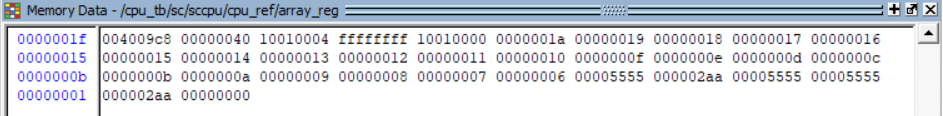
1. 实验结果

## 前仿真

### 6.1.1 modelsim后仿真波形图

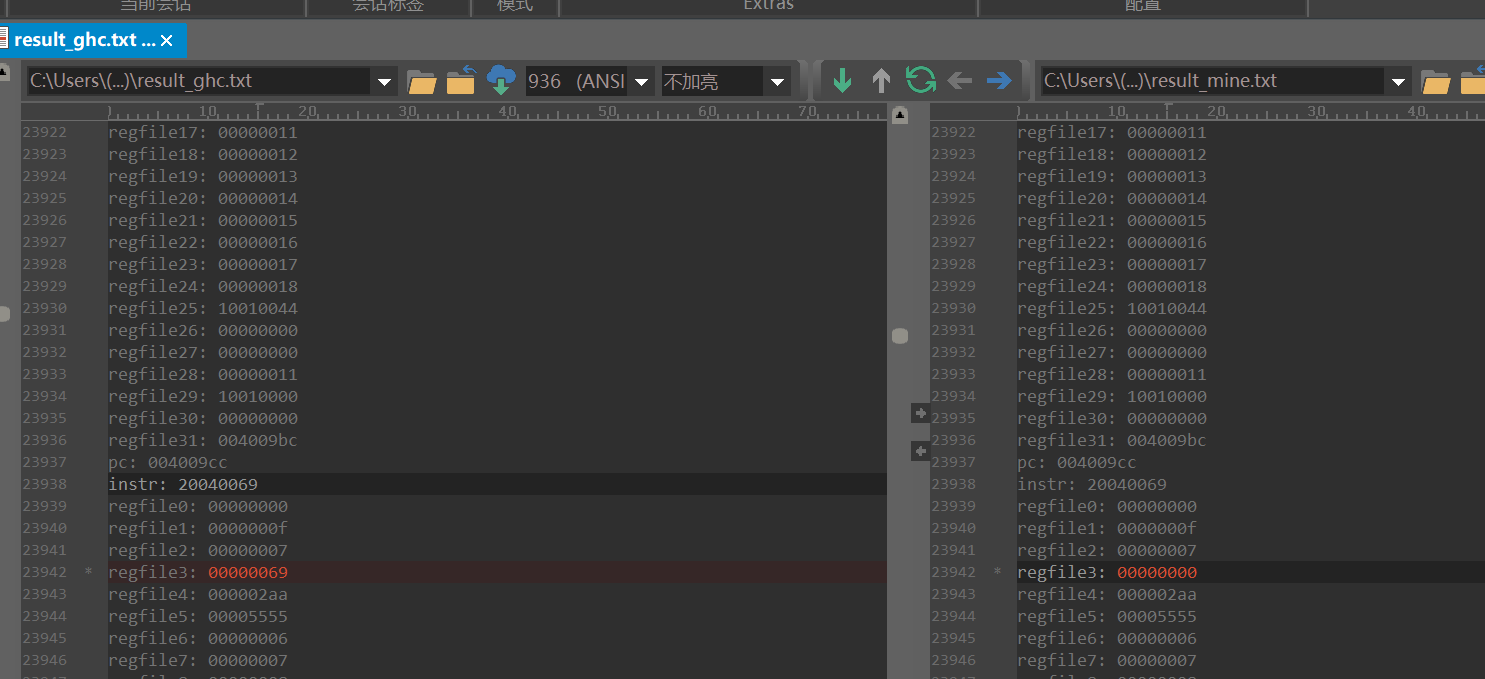




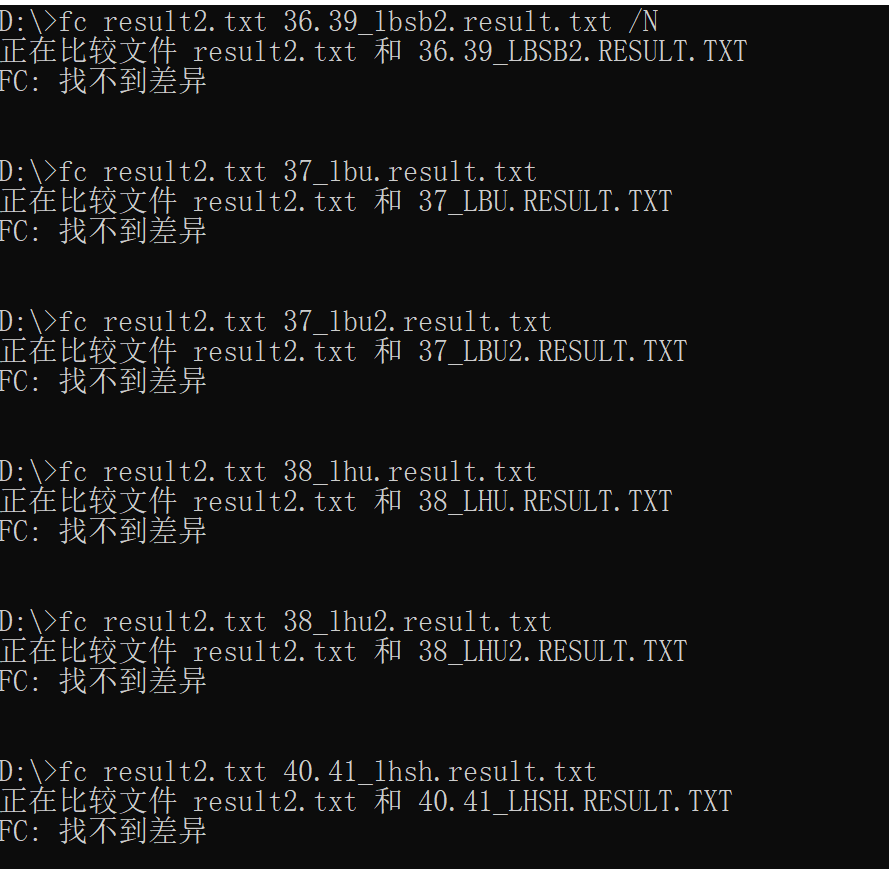
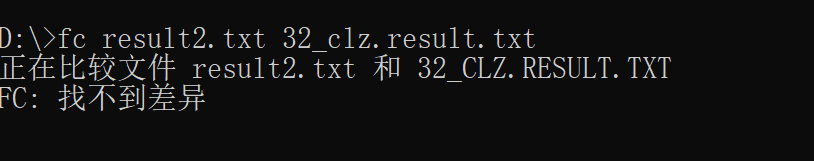


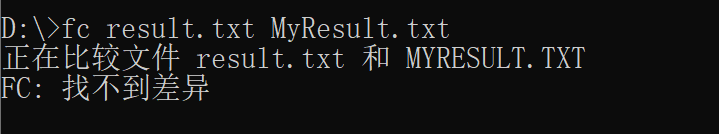
由上图我们可以看到，指令寄存器IM内存已读入相关指令译码后的结果，指令寄存器运行正常。同时寄存器堆regfile中已成功写入数据，寄存器堆也正常工作。

### 6.1.2 指令执行结果比对

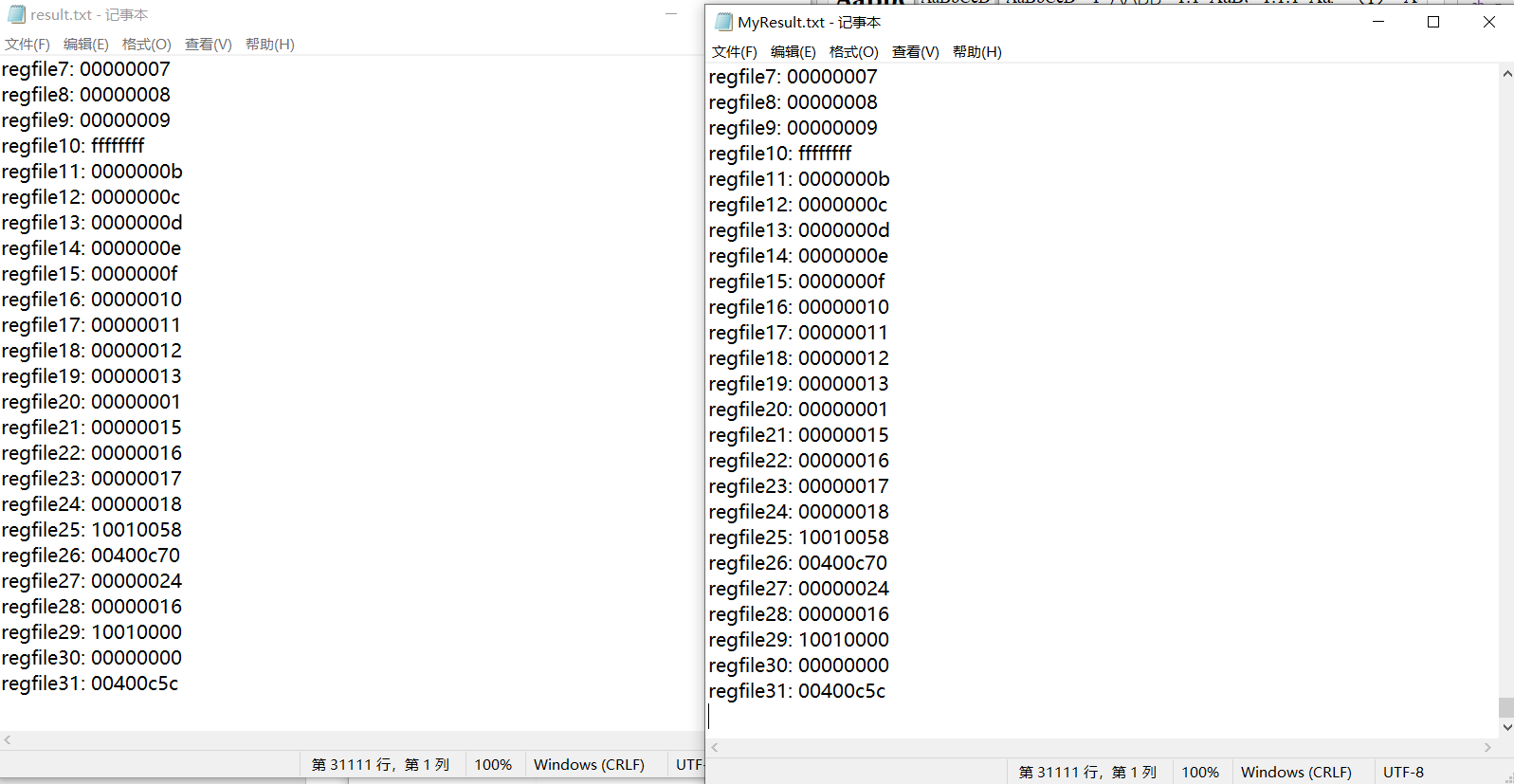


我通过采用cmd控制台的比较文件指令分别对31条指令以及最终版指令运行结果与标准结果进行比对，依次确认比对结果与标准结果并无差异以确认指令运行正确，CPU设计正确。



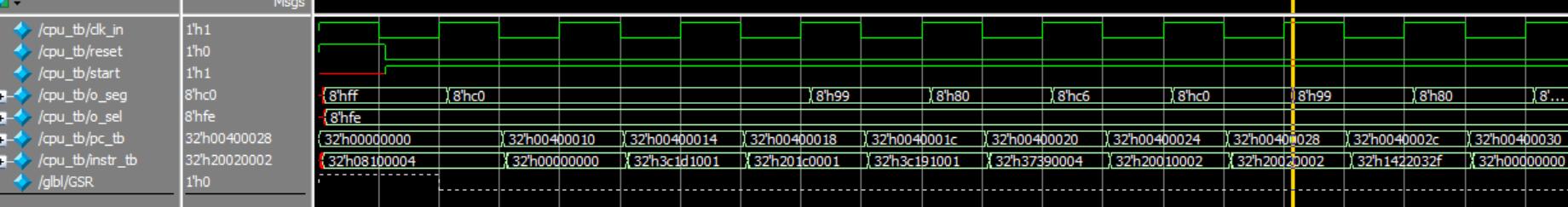


通过文件比对可以看出运行结果与标准并无差异，唯一一处不同在于运行结尾循环终止出，由此证明运行正常。

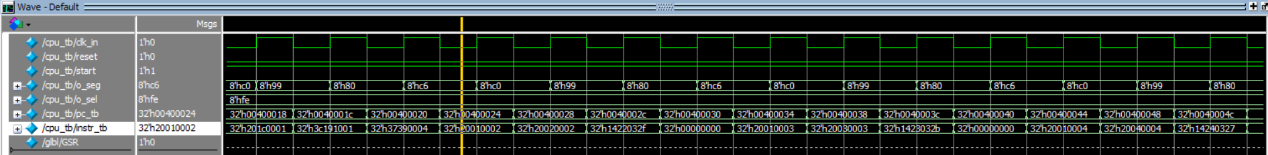


## 后仿真

### 6.2.1 modelsim后仿真波形图

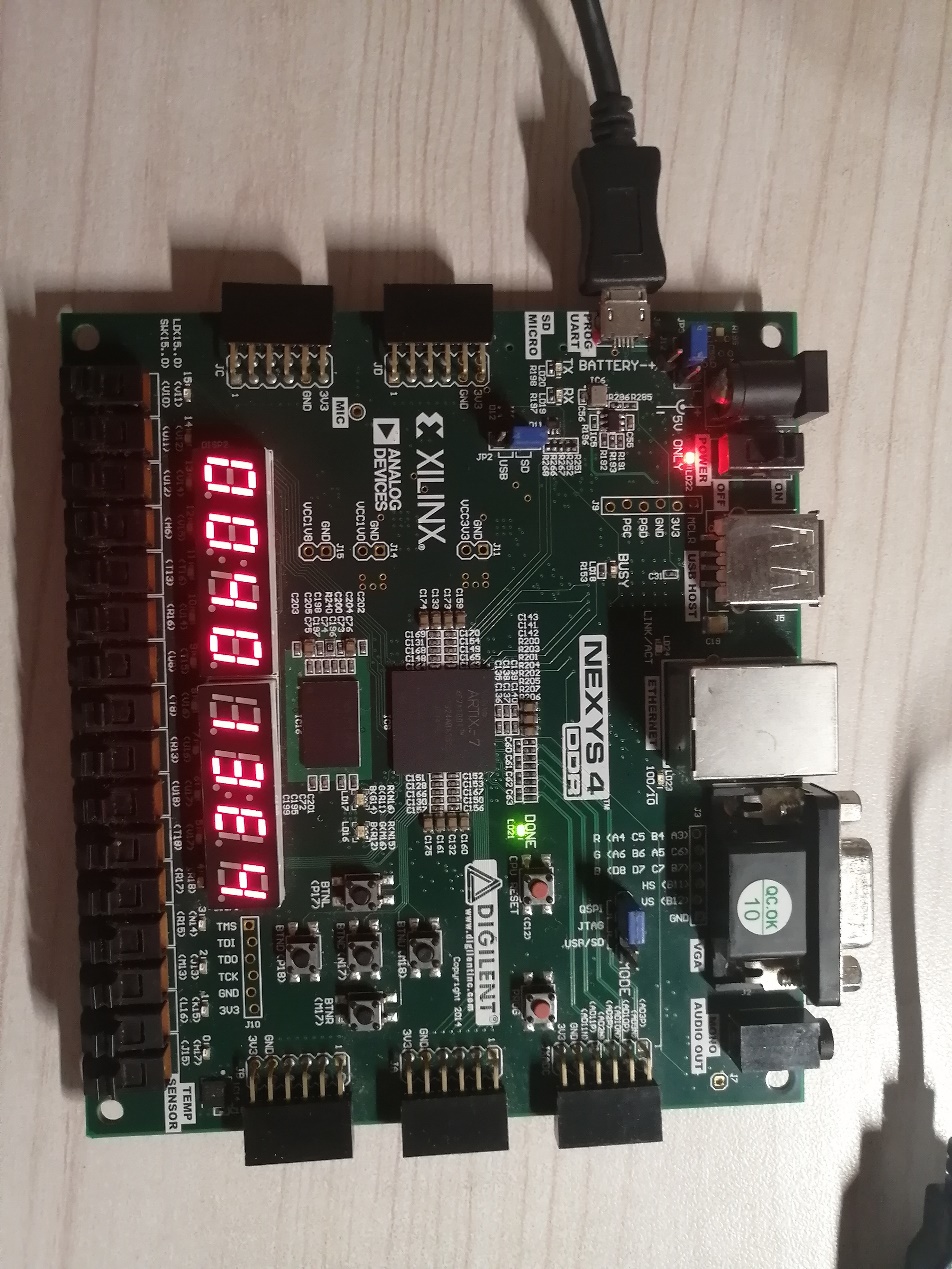


### 6.2.2 相应modelsim前仿真波形图



由modelsim仿真结果我们可以看出前仿真与后仿真波形图基本一致，唯一不同的点在于后仿真波形图pc与instr部分有略微的延迟。

## 下板验证



可以看出下板后七段数码管可以稳定显示程序中的PC值，故下板成功。

1. 总结与体会

总体来说，由于有了31条指令CPU的设计基础以及设计流程的过程熟悉。本次54条指令CPU设计相对来说更有方向，设计初期画图、列部件、写表达式等过程顺畅了许多。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。

由于本次54条指令CPU实验涉及到了 CPU内部对异常以及中断的处理，需要我们首先设计一个中断处理器CP0，我对CPU内部中断处理的机制、寄存器等内部构成有了一定的了解与认识，对CPU的整体框架有了更加深刻的理解。

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。这个过程需要对出错处的指令进行分析，将该指令所涉及到的部件控制信号在tb文件中利用Modelsim打印出来分析出错原因进行倒退回溯，最终寻找出出错原因。

CPU的设计以及后期的debug过程让我深刻感受到了设计一个已有指令的CPU的过程，也通过自己设计一个CPU对CPU的工作机理有了深刻的认识，总之收获颇丰。