2018300003058计算机系统基础2实验3

--MIPS单周期CPU的设计 辛嘉宇2018300003058

目录：1、简要介绍该目标cpu

2、具体细节、模块的设计

3、测试程序的运行结果分析

4、不足的地方以及遇到的问题

**一、简介**

cpu需要实现以下共42条指令：

• add/sub/and/or/slt/sltu/addu/subu

• addi/ori/lw/sw/beq

• j/jal

• xor/xori/addiu/andi/nor

• sll/sra/srav/lui/slti/sltiu/srl/sllv/srlv/jr/jalr

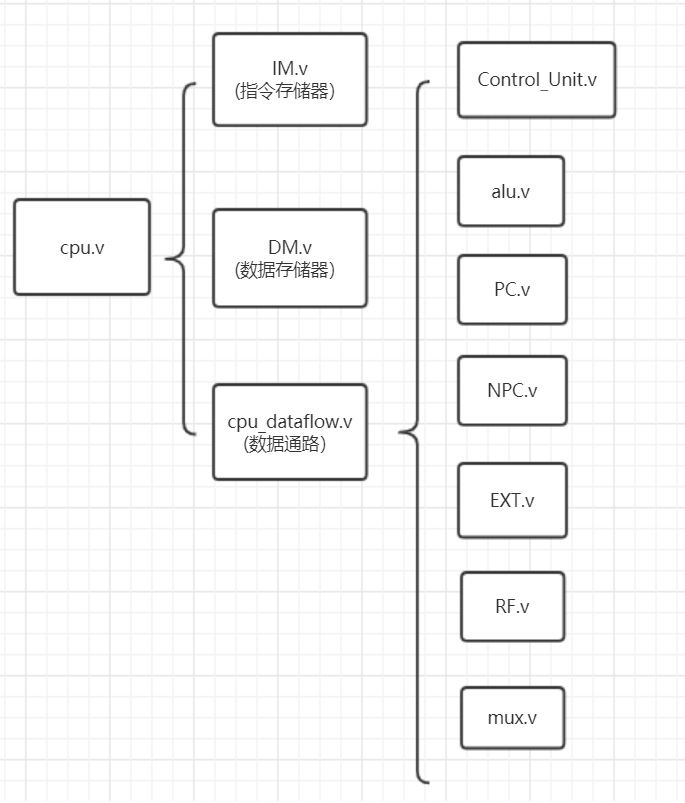
• lb/lh/lbu/lhu/sb/sh (数据在内存中以小端形式存储little endian)

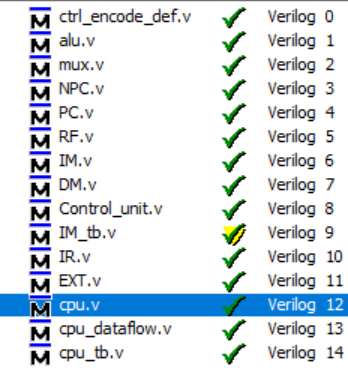
• bne/blez/bltz/bgtz/bgez

参照手册上的cpu设计思路，我将cpu分为以下模块

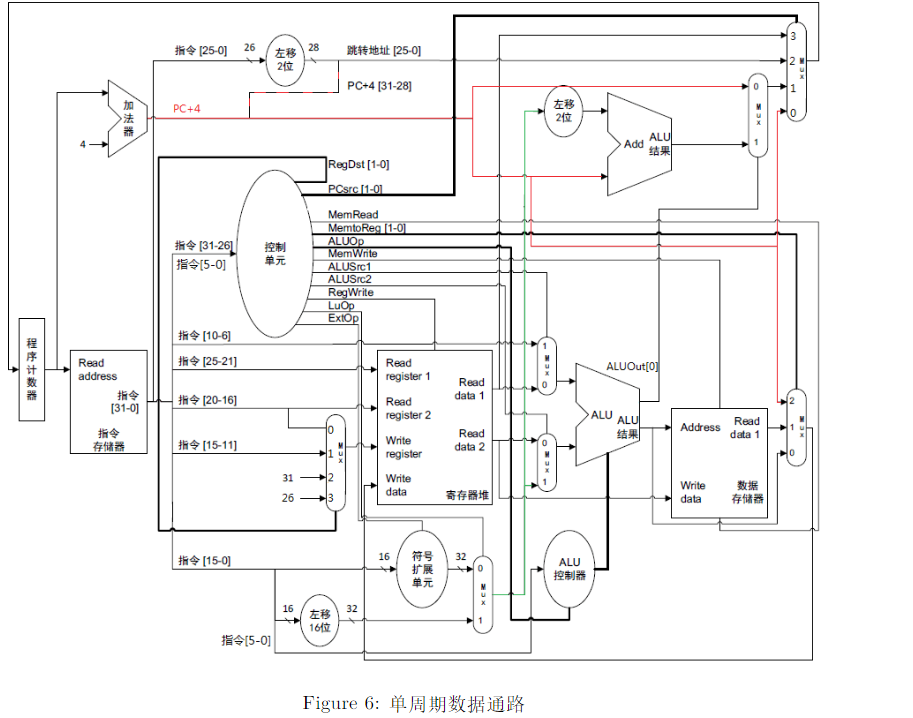
模块结构如下：

分为一个顶层模块(cpu)，三个大模块(cpu\_dataflow, IM, DM),七个基础模块(Control\_Unit, alu, PC, NPC, EXT, RF, mux)





总的数据通路图近似如下，某些地方有一些改动，但大致相同：



**二、具体模块分析**

控制信号--指令真值表(控制单元的基础)

具体表格见附件中(表格太大了放不下)

信号定义：

RegDst: 00--写寄存器为rt

/01--写寄存器为rd

/10--写寄存器为$ra($31)

RegWrite :000--不执行写操作

/001--执行写32位操作(lw\R\_type)

/010--执行写16位操作(lh)

/011-执行写8位操作(lb)

/100-lhu

/101-lbu

ALUsrc1: 0--ALU第一个输入来自寄存器

/1--第一个输入来自shamt(Inst[10:6])

ALUsrc2: 0--ALU第二个输入来自寄存器

/1--第二个输入来自指令低16位(Inst[15:0])

NPCOp: 00--PC+4

/01--分支

/10--立即数跳转

/11--寄存器跳转

MemWrite: 00--不执行写

/01--执行写32位操作(sw)

/10--执行写16位操作(sh)

/11--执行写8位操作(sb)

MemtoReg: 00--寄存器写数据来自ALU输出

/01--来自DM

/10--来自PC+4

BranchZ: 0--指令类型不为4种与0比较的指令

/1--指令类型为4种与0比较的指令之一

EXTOp: 0--0扩展（16->32位）

/1--符号扩展

模块代码如下：

顶层模块代码cpu.v:

module cpu(

input clk, rst,

output [31:0]Inst, PC, ALUOut, MemOut

);

wire [31:0]Read2;

wire [1:0]MemWrite;

cpu\_dataflow df( clk, rst, Inst, MemOut, PC, Read2, ALUOut, MemWrite);

IM instmem( PC, Inst);

DM datamem( clk, rst, Read2, ALUOut, MemWrite, MemOut);

endmodule

大模块之一--指令存储器IM.v:

module IM(

input [31:0] PC,

output reg [31:0] instruction

);

reg [31:0] mem [0:255];

initial begin

$readmemh("mipstest\_branch.txt", mem);

end

always@(PC) begin

assign instruction = mem[PC >> 2];

$display("the instruction now is %h", instruction);

end

endmodule

大模块之二--数据存储器DM.v:

module DM(

input clk,rst,

input [31:0]MemIn,

input [31:0]addr,

input [1:0]MemWrite,

output [31:0]MemOut

);

reg [0:7]DataMem[0:1023];

integer i;

initial

for(i=0;i<1024;i=i+1) DataMem[i]=0;

always@(posedge clk,posedge rst)

begin

if(rst)

for(i=0;i<1024;i=i+1) DataMem[i]<=0;

else

begin

case(MemWrite)

2'b01:

begin

DataMem[addr]=MemIn[7:0];//sw

DataMem[addr+1]=MemIn[15:8];

DataMem[addr+2]=MemIn[23:16];

DataMem[addr+3]=MemIn[31:24];

$display("DataMem[%2d]=%2h",addr,MemIn[7:0]);

$display("DataMem[%2d]=%2h",addr+1,MemIn[15:8]);

$display("DataMem[%2d]=%2h",addr+2,MemIn[23:16]);

$display("DataMem[%2d]=%2h",addr+3,MemIn[31:24]);

end

2'b10:

begin

DataMem[addr]=MemIn[7:0];//sh

DataMem[addr+1]=MemIn[15:8];

$display("DataMem[%2d]=%2h",addr,MemIn[7:0]);

$display("DataMem[%2d]=%2h",addr+1,MemIn[15:8]);

end

2'b11:

begin

DataMem[addr]=MemIn[7:0];//sb

$display("DataMem[%2d]=%2h",addr,MemIn[7:0]);

end

default:DataMem[addr]<=DataMem[addr];

endcase

end

end

assign MemOut={DataMem[addr+3],DataMem[addr+2],DataMem[addr+1],DataMem[addr]};

endmodule

大模块之三--数据通路cpu\_dataflow.v:

module cpu\_dataflow(

input clk, rst,

input [31:0]Inst, MemOut, RegWriteData,

output [31:0]pc, Read2, ALUOut,

output [1:0]MemWrite

);

wire [1:0]RegDst, NPCOp, MemtoReg;

wire [2:0]RegWrite;

wire ALUsrc1, ALUsrc2, EXTOp;

wire [3:0]ALUOp;

wire Zero,Sign,BranchZ;

wire [4:0]WriteReg;

wire [31:0]ExtImm;

wire [31:0]Read1, Jraddr;

wire shamt;

wire [31:0]ALUa;

wire [31:0]nextpc;

wire [31:0]ALUbb;

wire [31:0]ALUb;

control\_unit cu( Inst[31:26], Inst[5:0], Inst[20:16], Zero, Sign,

RegDst, RegWrite, NPCOp, MemWrite, MemtoReg, ALUsrc1, ALUsrc2, BranchZ, EXTOp, ALUOp);

EXT extension( Inst[15:0], EXTOp, ExtImm);

RF RegFile( clk, rst, RegWrite, Inst[25:21], Inst[20:16], WriteReg, RegWriteData, Read1, Read2, Inst[25:21], Jraddr);

mux4\_5 RegD( Inst[20:16], Inst[15:11], 5'd31, 5'd31, RegDst, WriteReg);

mux2\_32 ALU\_a( Read1, {27'b0,Inst[10:6]}, ALUsrc1, ALUa);

mux2\_32 ALU\_bb( Read2, ExtImm, ALUsrc2, ALUbb);

mux2\_32 ALU\_b( ALUbb, 32'b0, BranchZ, ALUb);

mux4\_32 Mem2Reg( ALUOut, MemOut, pc+4, 32'b0, MemtoReg, RegWriteData);

alu ALU( ALUa, ALUb, ALUOp, ALUOut, Zero, Sign);

NPC NextPC( pc, NPCOp, Inst[25:0], nextpc, Jraddr);

PC PC\_unit( clk, rst, nextpc, pc);

endmodule

基础模块之一--控制单元Control\_Unit.v:

`timescale 1ns/1ps

module control\_unit(

input [5:0]op,funct,

input [4:0]rt,

input Zero,Sign,

output [1:0]RegDst,

output [2:0]RegWrite,

output [1:0]NPCOp, MemWrite, MemtoReg,

output ALUsrc1, ALUsrc2, BranchZ, EXTOp,

output [3:0]ALUOp

);

wire rtype=~|op;

wire i\_add=rtype&funct[5]&~funct[4]&~funct[3]&~funct[2]&~funct[1]&~funct[0];//100000

wire i\_addu=rtype&funct[5]&~funct[4]&~funct[3]&~funct[2]&~funct[1]&funct[0];//100001

wire i\_sub=rtype&funct[5]&~funct[4]&~funct[3]&~funct[2]&funct[1]&~funct[0];//100010

wire i\_subu=rtype&funct[5]&~funct[4]&~funct[3]&~funct[2]&funct[1]&funct[0];//100011

wire i\_and=rtype&funct[5]&~funct[4]&~funct[3]&funct[2]&~funct[1]&~funct[0];//100100

wire i\_or=rtype&funct[5]&~funct[4]&~funct[3]&funct[2]&~funct[1]&funct[0];//100101

wire i\_nor=rtype&funct[5]&~funct[4]&~funct[3]&funct[2]&funct[1]&funct[0];//100111

wire i\_xor=rtype&funct[5]&~funct[4]&~funct[3]&funct[2]&funct[1]&~funct[0];//100110

wire i\_sll=rtype&~funct[5]&~funct[4]&~funct[3]&~funct[2]&~funct[1]&~funct[0];//000000

wire i\_sllv=rtype&~funct[5]&~funct[4]&~funct[3]&funct[2]&~funct[1]&~funct[0];//000100

wire i\_srl=rtype&~funct[5]&~funct[4]&~funct[3]&~funct[2]&funct[1]&~funct[0];//000010

wire i\_srlv=rtype&~funct[5]&~funct[4]&~funct[3]&funct[2]&funct[1]&~funct[0];//000110

wire i\_sra=rtype&~funct[5]&~funct[4]&~funct[3]&~funct[2]&funct[1]&funct[0];//000011

wire i\_srav=rtype&~funct[5]&~funct[4]&~funct[3]&funct[2]&funct[1]&funct[0];//000111

wire i\_jr=rtype&~funct[5]&~funct[4]&funct[3]&~funct[2]&~funct[1]&~funct[0];//001000

wire i\_jalr=rtype&~funct[5]&~funct[4]&funct[3]&~funct[2]&~funct[1]&funct[0];//001001

wire i\_slt=rtype&funct[5]&~funct[4]&funct[3]&~funct[2]&funct[1]&~funct[0];//101010

wire i\_sltu=rtype&funct[5]&~funct[4]&funct[3]&~funct[2]&funct[1]&funct[0];//101011

//18

wire i\_addi=~op[5]&~op[4]&op[3]&~op[2]&~op[1]&~op[0];//001000

wire i\_addiu=~op[5]&~op[4]&op[3]&~op[2]&~op[1]&op[0];//001001

wire i\_andi=~op[5]&~op[4]&op[3]&op[2]&~op[1]&~op[0];//001100

wire i\_xori=~op[5]&~op[4]&op[3]&op[2]&op[1]&~op[0];//001110

wire i\_beq=~op[5]&~op[4]&~op[3]&op[2]&~op[1]&~op[0];//000100

wire i\_bne=~op[5]&~op[4]&~op[3]&op[2]&~op[1]&op[0];//000101

wire i\_blez=~op[5]&~op[4]&~op[3]&op[2]&op[1]&~op[0];//000110

wire i\_bgtz=~op[5]&~op[4]&~op[3]&op[2]&op[1]&op[0];//000111

wire i\_bltz=~op[5]&~op[4]&~op[3]&~op[2]&~op[1]&op[0]&~rt[0];//000001 00000

wire i\_bgez=~op[5]&~op[4]&~op[3]&~op[2]&~op[1]&op[0]&rt[0];//000001 00001

wire i\_j=~op[5]&~op[4]&~op[3]&~op[2]&op[1]&~op[0];//000010

wire i\_jal=~op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0];//000011

wire i\_slti=~op[5]&~op[4]&op[3]&~op[2]&op[1]&~op[0];//001010

wire i\_sltiu=~op[5]&~op[4]&op[3]&~op[2]&op[1]&op[0];//001011

wire i\_ori=~op[5]&~op[4]&op[3]&op[2]&~op[1]&op[0];//001101

wire i\_lui=~op[5]&~op[4]&op[3]&op[2]&op[1]&op[0];//001111

wire i\_lb=op[5]&~op[4]&~op[3]&~op[2]&~op[1]&~op[0];//100000

wire i\_lh=op[5]&~op[4]&~op[3]&~op[2]&~op[1]&op[0];//100001

wire i\_lw=op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0];//100011

wire i\_lbu=op[5]&~op[4]&~op[3]&op[2]&~op[1]&~op[0];//100100

wire i\_lhu=op[5]&~op[4]&~op[3]&op[2]&~op[1]&op[0];//100101

wire i\_sb=op[5]&~op[4]&op[3]&~op[2]&~op[1]&~op[0];//101000

wire i\_sh=op[5]&~op[4]&op[3]&~op[2]&~op[1]&op[0];//101001

wire i\_sw=op[5]&~op[4]&op[3]&~op[2]&op[1]&op[0];//101011

//24

assign RegDst[1]=i\_jalr|i\_jal;

assign RegDst[0]=i\_add|i\_addu|i\_sub|i\_subu|i\_and|i\_or

|i\_xor|i\_nor|i\_slt|i\_sltu|i\_sll|i\_sllv

|i\_srl|i\_srlv|i\_sra|i\_srav;

assign RegWrite[2]=i\_lhu|i\_lbu;

assign RegWrite[1]=i\_lb|i\_lh;

assign RegWrite[0]=i\_add|i\_addu|i\_sub|i\_subu|i\_and|i\_or

|i\_xor|i\_nor|i\_slt|i\_sltu|i\_sll|i\_sllv

|i\_srl|i\_srlv|i\_sra|i\_srav|i\_jalr

|i\_addi|i\_addiu|i\_slti|i\_sltiu|i\_andi

|i\_ori|i\_xori|i\_lui|i\_jal|i\_lb|i\_lw|i\_lbu;

assign ALUsrc1=i\_sll|i\_srl|i\_sra;

assign ALUsrc2=i\_addi|i\_addiu|i\_slti|i\_sltiu|i\_andi|i\_ori|i\_xori

|i\_lui|i\_lb|i\_lh|i\_lw|i\_lbu|i\_lhu|i\_sb|i\_sh|i\_sw;

assign ALUOp[3]=i\_nor|i\_sll|i\_sllv|i\_srl|i\_srlv|i\_sra|i\_srav|i\_lui;

assign ALUOp[2]=i\_or|i\_xor|i\_slt|i\_sltu|i\_slti|i\_sltiu|i\_ori

|i\_xori|i\_lui;

assign ALUOp[1]=i\_sub|i\_subu|i\_and|i\_xor|i\_sltu|i\_sltiu|i\_srl|i\_srlv

|i\_sra|i\_srav|i\_andi|i\_xori|i\_beq|i\_bne

|i\_blez|i\_bgtz|i\_bltz|i\_bgez;

assign ALUOp[0]=i\_add|i\_addu|i\_and|i\_xor|i\_slt|i\_slti|i\_sll|i\_sllv

|i\_sra|i\_srav|i\_addi|i\_addiu|i\_andi|i\_xori

|i\_lb|i\_lh|i\_lw|i\_lbu|i\_lhu|i\_sb|i\_sh|i\_sw;

assign NPCOp[1]=i\_jr|i\_jalr|i\_j|i\_jal;

assign NPCOp[0]=i\_jr|i\_jalr|(i\_beq&Zero)|(i\_bne&(~Zero))|(i\_blez&(Sign|Zero))|(i\_bgtz&(~Sign&~Zero))|(i\_bltz&Sign)|(i\_bgez&(~Sign));

assign MemWrite[1]=i\_sb|i\_sh;

assign MemWrite[0]=i\_sw|i\_sb;

assign MemtoReg[1]=i\_jalr|i\_jal;

assign MemtoReg[0]=i\_lb|i\_lh|i\_lw|i\_lbu|i\_lhu;

assign BranchZ=i\_blez|i\_bgtz|i\_bltz|i\_bgez;

assign EXTOp=i\_addi|i\_slti|i\_lb|i\_lh|i\_lw|i\_lbu|i\_lhu|i\_sb|i\_sh|i\_sw;

endmodule

(具体信号含义请参照开头的定义,Zero/Sign为ALU输出至控制单元的信号值，分别代表了ALU运算结果是否为0/ALU运算结果的符号位)

基础模块之二--算术逻辑单元alu.v:

`include "ctrl\_encode\_def.v"

module alu(A, B, ALUOp, C, Zero, Sign);

input signed [31:0] A, B;

input [3:0] ALUOp;

output signed [31:0] C;

output Zero,Sign;

reg [31:0] C;

integer i;

always @( \* ) begin

case ( ALUOp )

`ALU\_NOP: C = A; // NOP

`ALU\_ADD: C = A + B; // ADD/ADDI/ADDU

`ALU\_SUB: C = A - B; // SUB/SUBU

`ALU\_AND: C = A & B; // AND/ANDI

`ALU\_OR: C = A | B; // OR/ORI

`ALU\_SLT: C = (A < B); // SLT/SLTI

`ALU\_SLTU: C = ({1'b0, A} < {1'b0, B});

`ALU\_XOR: C = A^B;

`ALU\_NOR: C = ~(A|B);

`ALU\_SLL: C = B<<A[5:0];

`ALU\_SRL: C = B>>A[5:0];

`ALU\_SRA: C = $signed(B)>>>A[5:0];

`ALU\_LUI: C = {B[15:0],16'b0};

default: C = A; // Undefined

endcase

end // end always

assign Zero = (C == 32'b0);

assign Sign = C[31];

endmodule

基础模块之三--PC寄存器PC.v:

module PC( clk, rst, NPC, PC );

input clk;

input rst;

input [31:0] NPC;

output reg [31:0] PC;

always @(posedge clk, posedge rst)

if (rst)

PC <= 32'h0000\_0000;

// PC <= 32'h0000\_3000;

else

PC <= NPC;

endmodule

基础模块之四--下一地址计算单元NPC.v:

`include "ctrl\_encode\_def.v"

module NPC(PC, NPCOp, IMM, NPC, Jraddr); // next pc module

input [31:0] PC; // pc

input [1:0] NPCOp; // next pc operation

input [25:0] IMM; // immediate

input [31:0] Jraddr;

output reg [31:0] NPC; // next pc

wire [31:0] PCPLUS4;

assign PCPLUS4 = PC + 4; // pc + 4

always @(\*) begin

case (NPCOp)

`NPC\_PLUS4: NPC = PCPLUS4;

`NPC\_BRANCH: NPC = PCPLUS4 + {{14{IMM[15]}}, IMM[15:0], 2'b00};

`NPC\_JUMP: NPC = {PCPLUS4[31:28], IMM[25:0], 2'b00};

default: NPC = Jraddr;

endcase

end // end always

endmodule

基础模块之五--16-32位符号扩展/0扩展单元EXT.v:

module EXT( Imm16, EXTOp, Imm32 );

input [15:0] Imm16;

input EXTOp;

output [31:0] Imm32;

assign Imm32 = (EXTOp) ? {{16{Imm16[15]}}, Imm16} : {16'd0, Imm16}; // signed-extension or zero extension

endmodule

基础模块之六--寄存器堆RF.v:

module RF( input clk,

input rst,

input [2:0] RFWr,

input [4:0] A1, A2, A3,

input [31:0] WD,

output [31:0] RD1, RD2,

input [4:0] reg\_sel,

output [31:0] reg\_data);

reg [31:0] rf[0:31];

integer i;

initial rf[0]=0;

always @(posedge clk, posedge rst)

if (rst) begin // reset

for (i=1; i<32; i=i+1)

rf[i] <= 0; // i;

end

else

begin

case(RFWr)

3'b001://lw

begin

rf[A3] <= WD;

// $display("r[00-07]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", 0, rf[1], rf[2], rf[3], rf[4], rf[5], rf[6], rf[7]);

// $display("r[08-15]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[8], rf[9], rf[10], rf[11], rf[12], rf[13], rf[14], rf[15]);

// $display("r[16-23]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[16], rf[17], rf[18], rf[19], rf[20], rf[21], rf[22], rf[23]);

// $display("r[24-31]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[24], rf[25], rf[26], rf[27], rf[28], rf[29], rf[30], rf[31]);

$display("r[%2d] = 0x%8X,", A3, WD);

end

3'b010://lh

begin

rf[A3] <= {{16{WD[15]}},WD[15:0]};

// $display("r[00-07]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", 0, rf[1], rf[2], rf[3], rf[4], rf[5], rf[6], rf[7]);

// $display("r[08-15]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[8], rf[9], rf[10], rf[11], rf[12], rf[13], rf[14], rf[15]);

// $display("r[16-23]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[16], rf[17], rf[18], rf[19], rf[20], rf[21], rf[22], rf[23]);

// $display("r[24-31]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[24], rf[25], rf[26], rf[27], rf[28], rf[29], rf[30], rf[31]);

$display("r[%2d] = 0x%8X,", A3, {{16{WD[15]}},WD[15:0]});

end

3'b011://lb

begin

rf[A3] <= {{24{WD[7]}},WD[7:0]};

// $display("r[00-07]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", 0, rf[1], rf[2], rf[3], rf[4], rf[5], rf[6], rf[7]);

// $display("r[08-15]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[8], rf[9], rf[10], rf[11], rf[12], rf[13], rf[14], rf[15]);

// $display("r[16-23]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[16], rf[17], rf[18], rf[19], rf[20], rf[21], rf[22], rf[23]);

// $display("r[24-31]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[24], rf[25], rf[26], rf[27], rf[28], rf[29], rf[30], rf[31]);

$display("r[%2d] = 0x%8X,", A3, {{24{WD[7]}},WD[7:0]});

end

3'b100://lhu

begin

rf[A3] <= {16'b0,WD[15:0]};

// $display("r[00-07]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", 0, rf[1], rf[2], rf[3], rf[4], rf[5], rf[6], rf[7]);

// $display("r[08-15]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[8], rf[9], rf[10], rf[11], rf[12], rf[13], rf[14], rf[15]);

// $display("r[16-23]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[16], rf[17], rf[18], rf[19], rf[20], rf[21], rf[22], rf[23]);

// $display("r[24-31]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[24], rf[25], rf[26], rf[27], rf[28], rf[29], rf[30], rf[31]);

$display("r[%2d] = 0x%8X,", A3, {16'b0,WD[15:0]});

end

3'b101://lbu

begin

rf[A3] <= {24'b0,WD[7:0]};

// $display("r[00-07]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", 0, rf[1], rf[2], rf[3], rf[4], rf[5], rf[6], rf[7]);

// $display("r[08-15]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[8], rf[9], rf[10], rf[11], rf[12], rf[13], rf[14], rf[15]);

// $display("r[16-23]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[16], rf[17], rf[18], rf[19], rf[20], rf[21], rf[22], rf[23]);

// $display("r[24-31]=0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X, 0x%8X", rf[24], rf[25], rf[26], rf[27], rf[28], rf[29], rf[30], rf[31]);

$display("r[%2d] = 0x%8X,", A3, {24'b0,WD[7:0]});

end

default: rf[A3]<=rf[A3];

endcase

end

assign RD1 = (A1 != 0) ? rf[A1] : 0;

assign RD2 = (A2 != 0) ? rf[A2] : 0;

assign reg\_data = (reg\_sel != 0) ? rf[reg\_sel] : 0;

endmodule

基础模块之七--多路选择器(内含32位的2选1、5位的4选1、32位的4选1)mux.v:

// mux2

module mux2\_32 #(parameter WIDTH = 32)

(d0, d1,

s, y);

input [WIDTH-1:0] d0, d1;

input s;

output [WIDTH-1:0] y;

reg [WIDTH-1:0] y\_r;

always@(\*)

begin

y\_r= ( s == 1'b1 ) ? d1:d0;

end

assign y = y\_r;

endmodule

// mux4

module mux4\_5 #(parameter WIDTH = 5)

(d0, d1, d2, d3,

s, y);

input [WIDTH-1:0] d0, d1, d2, d3;

input [1:0] s;

output [WIDTH-1:0] y;

reg [WIDTH-1:0] y\_r;

always @( \* ) begin

case ( s )

2'b00: y\_r = d0;

2'b01: y\_r = d1;

2'b10: y\_r = d2;

2'b11: y\_r = d3;

default: ;

endcase

end // end always

assign y = y\_r;

endmodule

module mux4\_32 #(parameter WIDTH = 32)

(d0, d1, d2, d3,

s, y);

input [WIDTH-1:0] d0, d1, d2, d3;

input [1:0] s;

output [WIDTH-1:0] y;

reg [WIDTH-1:0] y\_r;

always @( \* ) begin

case ( s )

2'b00: y\_r = d0;

2'b01: y\_r = d1;

2'b10: y\_r = d2;

2'b11: y\_r = d3;

default: ;

endcase

end // end always

assign y = y\_r;

endmodule

宏定义文件ctrl\_encode\_def.v:

// NPC control signal

`define NPC\_PLUS4 2'b00

`define NPC\_BRANCH 2'b01

`define NPC\_JUMP 2'b10

// ALU control signal

`define ALU\_NOP 4'b0000

`define ALU\_ADD 4'b0001

`define ALU\_SUB 4'b0010

`define ALU\_AND 4'b0011

`define ALU\_OR 4'b0100

`define ALU\_SLT 4'b0101

`define ALU\_SLTU 4'b0110

`define ALU\_XOR 4'b0111

`define ALU\_NOR 4'b1000

`define ALU\_SLL 4'b1001

`define ALU\_SRL 4'b1010

`define ALU\_SRA 4'b1011

`define ALU\_LUI 4'b1100

顶层模块测试文件cpu\_tb.v:

`timescale 1ns/1ps

module cpu\_tb;

reg clk,rst;

wire [31:0]Inst, PC, ALUOut, MemOut;

initial

begin

clk=0;

rst=1;

#500;

clk=~clk;

rst=0;

#500;

end

always #500

begin

clk=~clk;

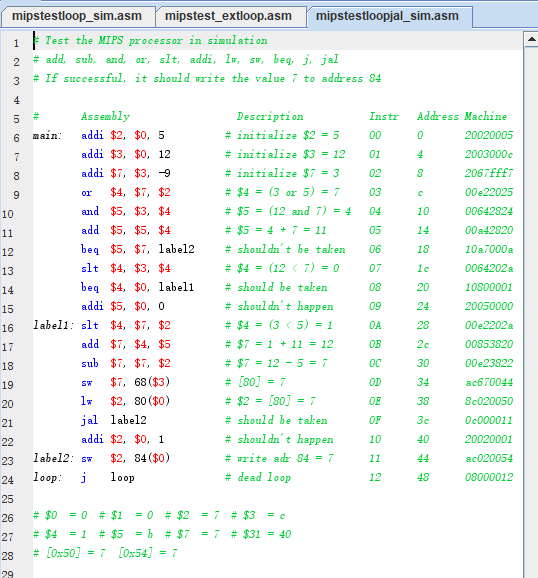
end

cpu cpu\_test( clk, rst, Inst, PC, ALUOut, MemOut);

endmodule

**三、测试程序运行结果分析**

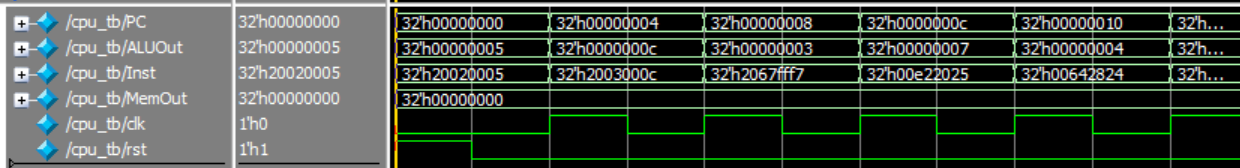
测试程序之一--mipstesloopjal\_sim.asm:

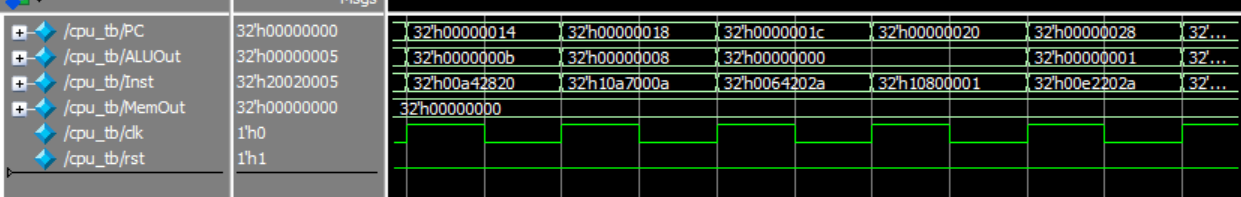


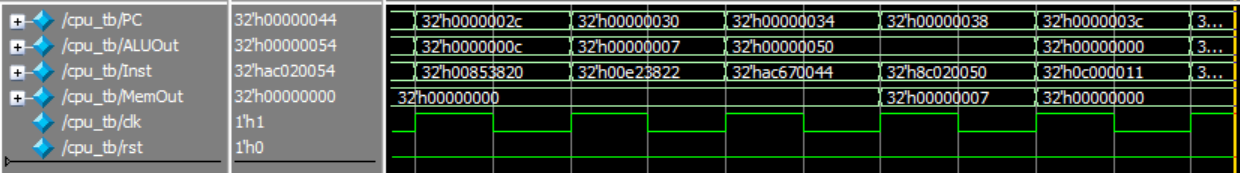
分析得PC跳转情况为：

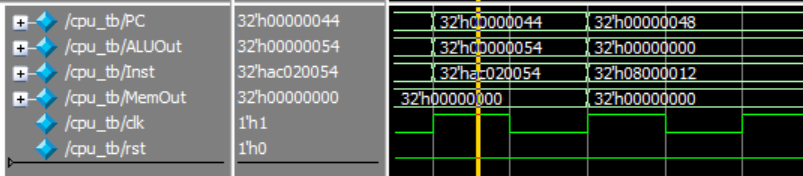
0x00->0x4->...->0x20->0x28->...->0x3c->0x44->0x48

PC波形图如下:



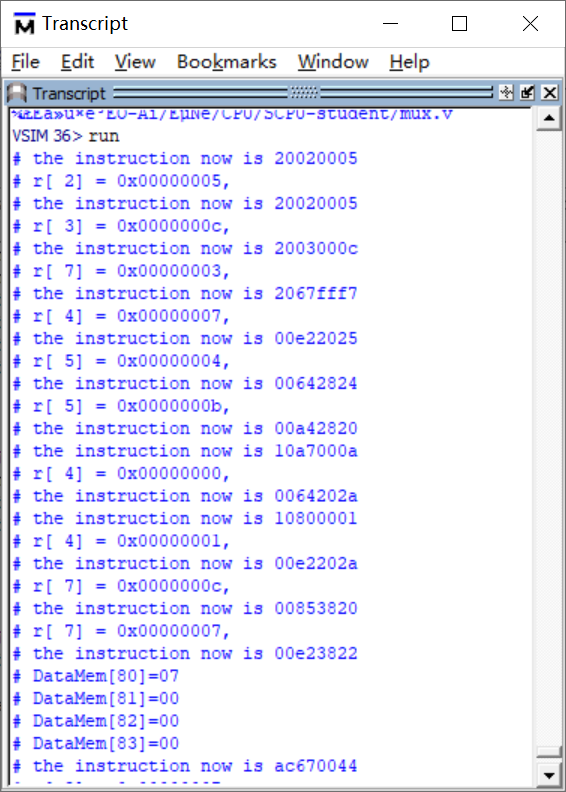


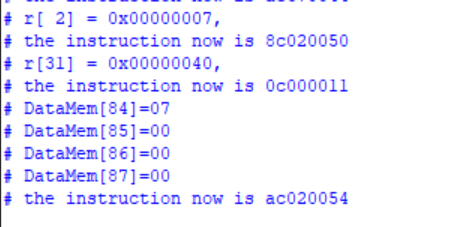




PC跳转情况与预期结果一致

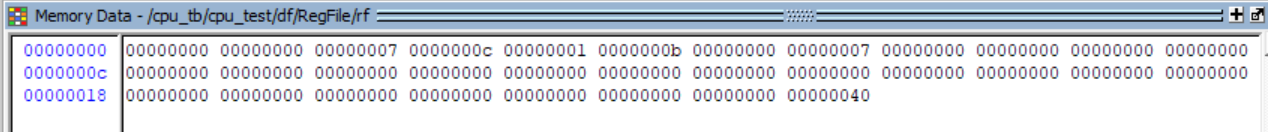
执行过程如下：

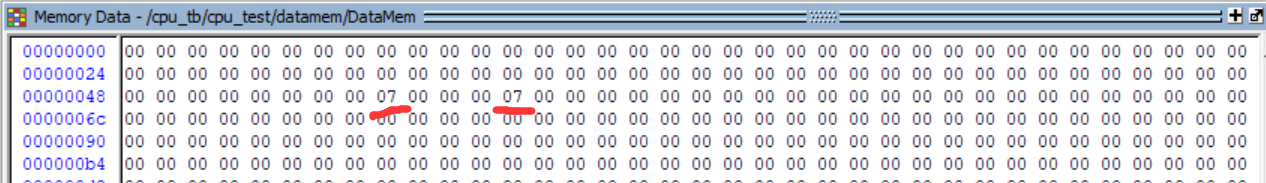




也与预期结果一致

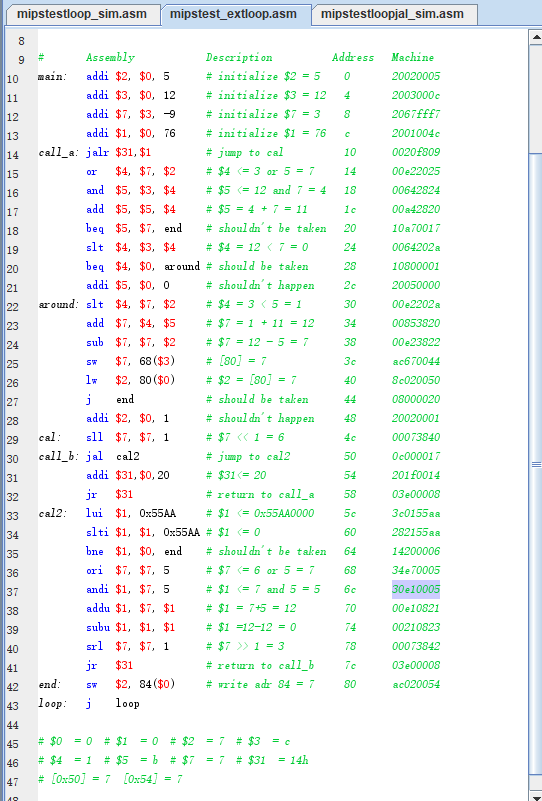
最终的寄存器堆、数据存储器内容如下：





与预期结果一致，说明无误

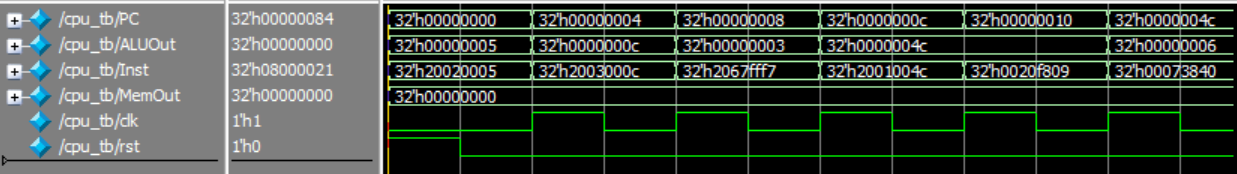
测试程序之二--mipstest\_extloop:



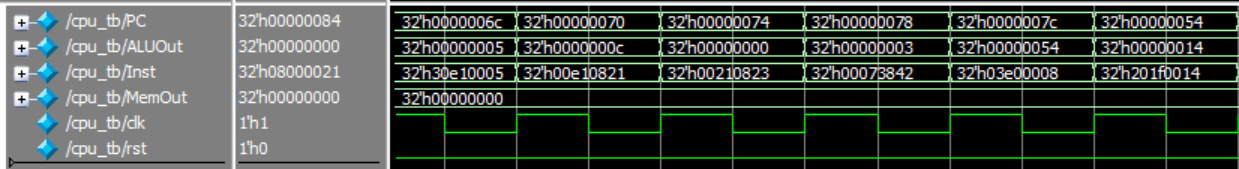
分析得PC跳转情况为：

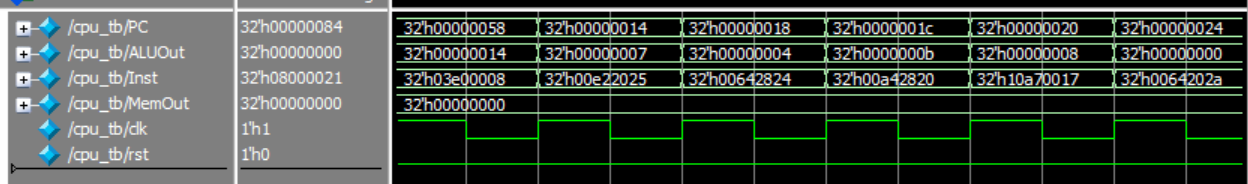
0x00->...->0x10->0x4c->0x50->0x5c->...->0x7c->0x50->0x5c->...->0x7c->0x54->0x58-> 0x14->...->0x28->0x30->...->0x44->0x80->0x84

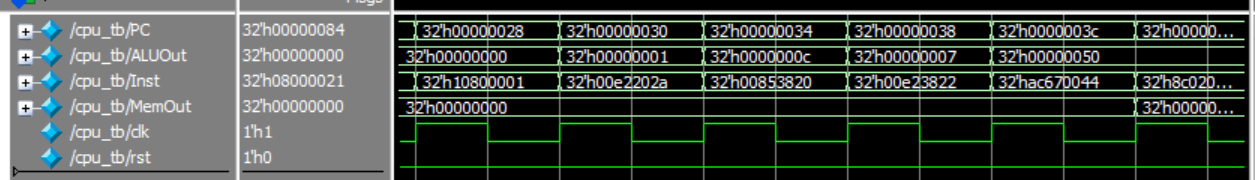
波形图如下：

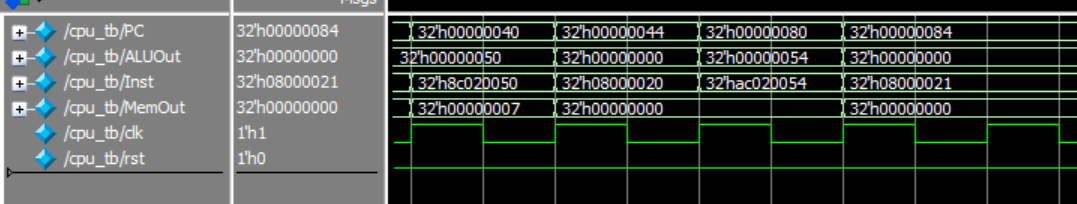






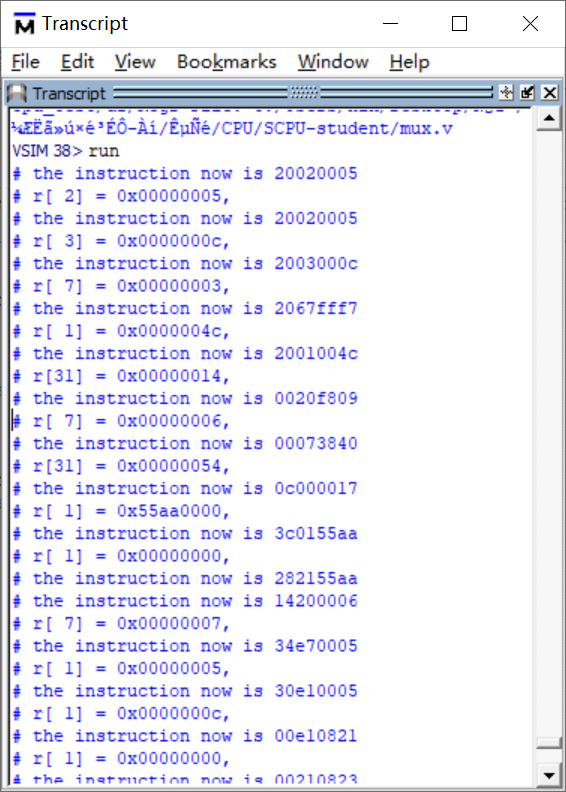


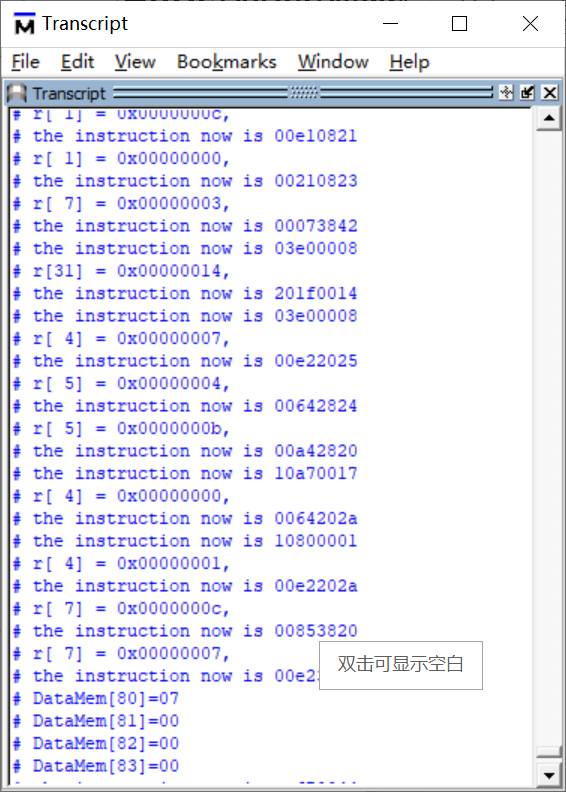


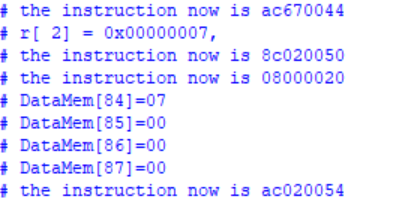


与预期结果一致

执行过程如下：

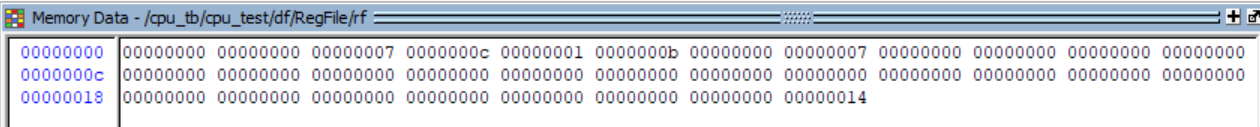


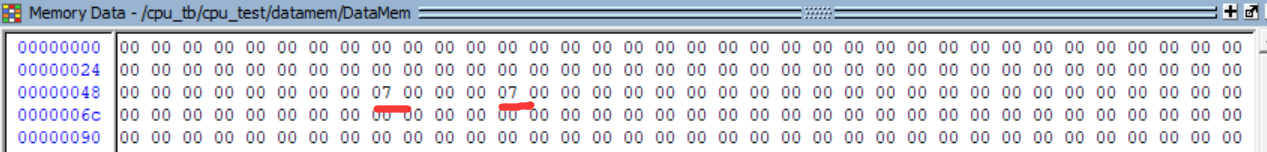




与预期结果相同

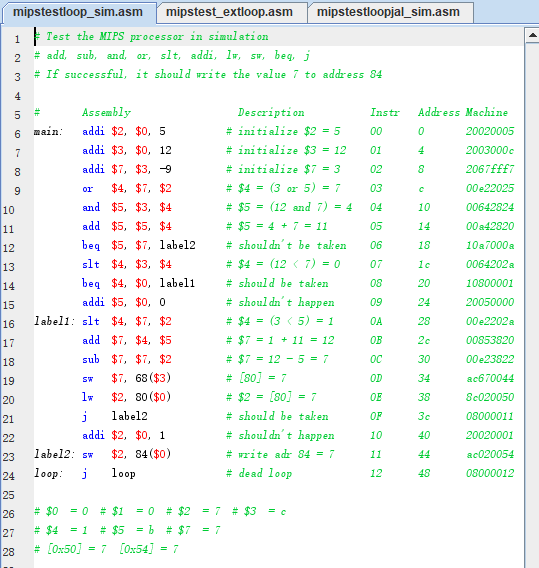
最终寄存器堆、数据存储器内容如下：





与预期结果相同，说明无误

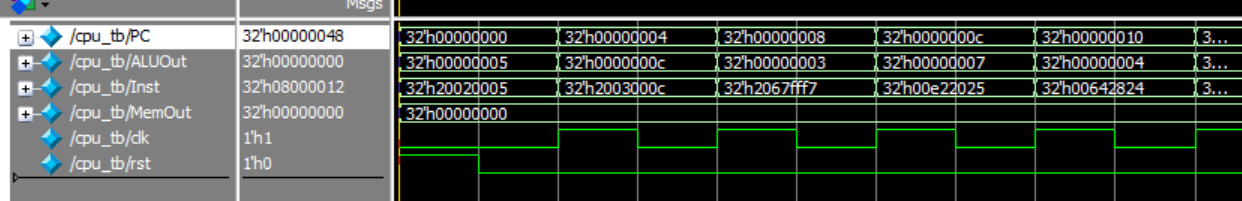
测试程序之三--mipstestloop\_sim:

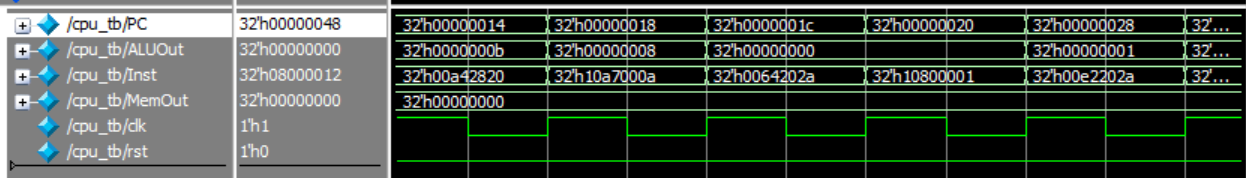


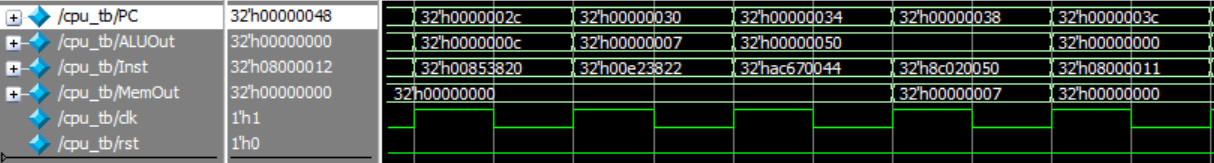
分析得PC跳转结果为：

0x00->...->0x20->0x28->...->0x3c->0x44->0x48

波形图如下：



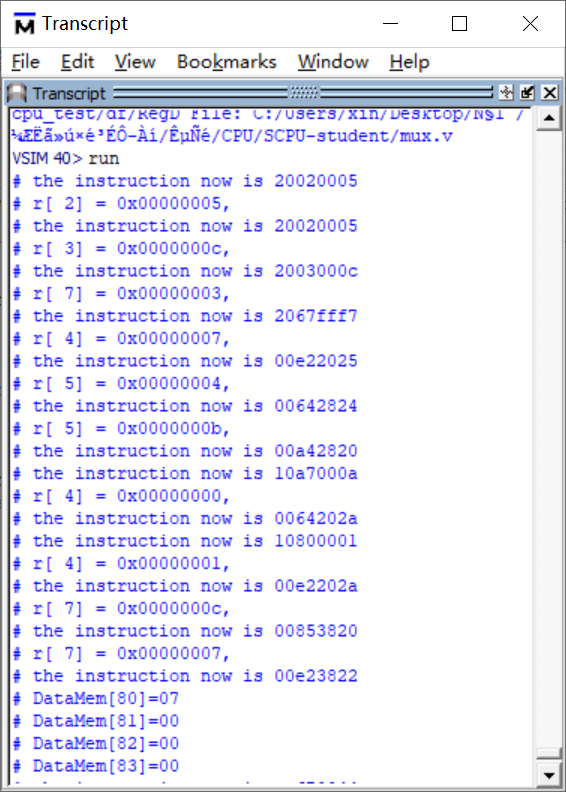


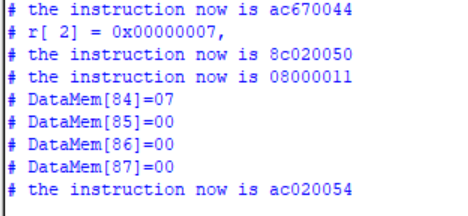




与预期结果一致

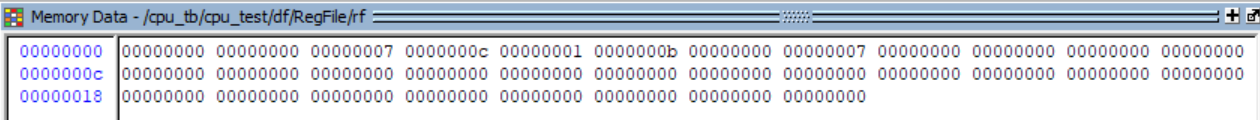
执行过程如下：

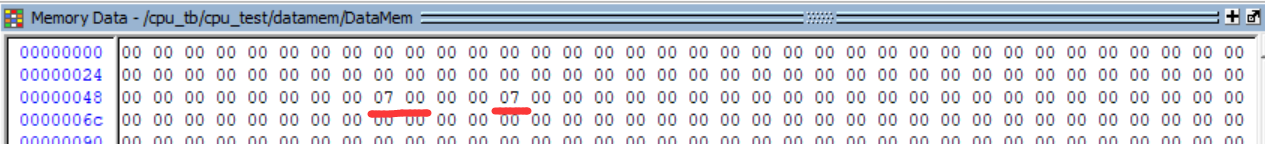




与预期结果一致

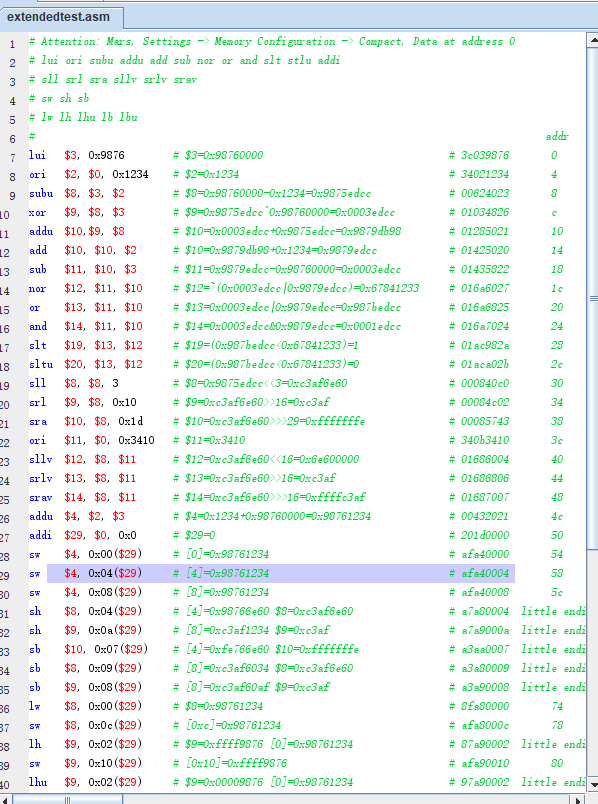
最终寄存器堆、数据存储器内容如下：

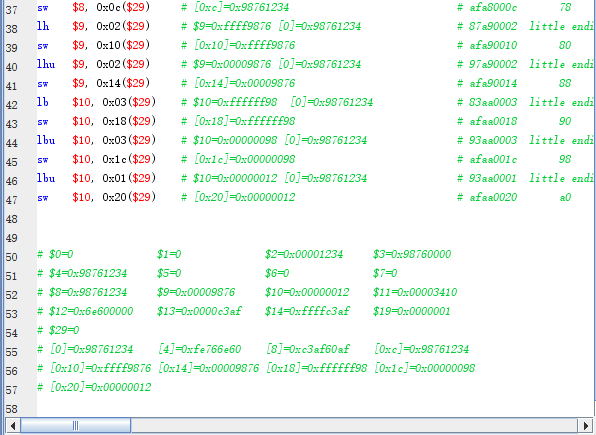




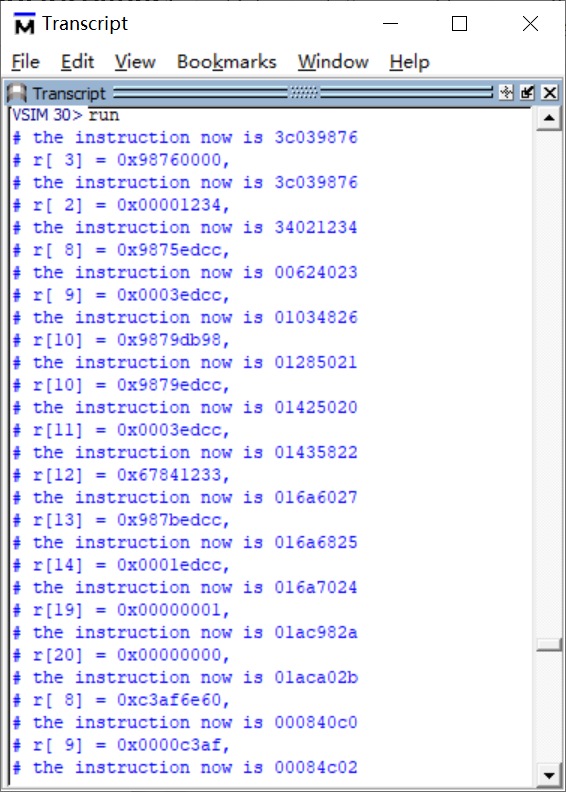
与预期结果一致，说明无误

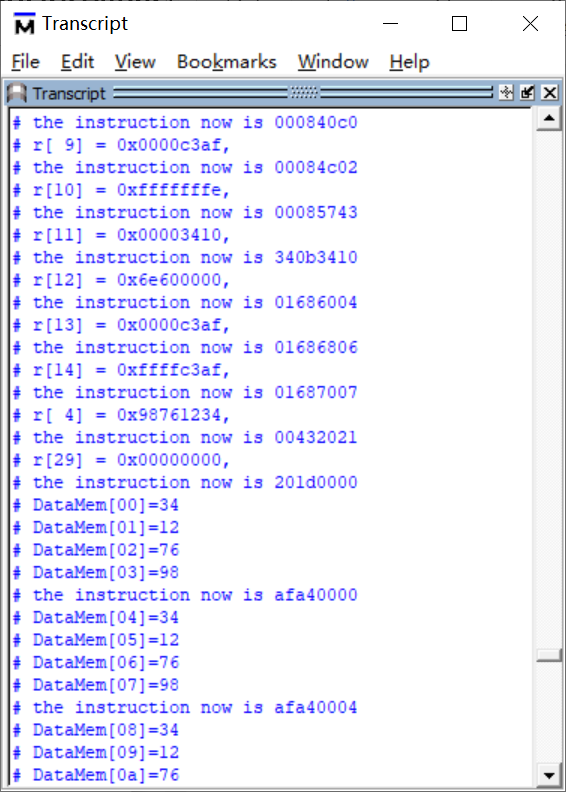
测试程序之四--extendedtest:

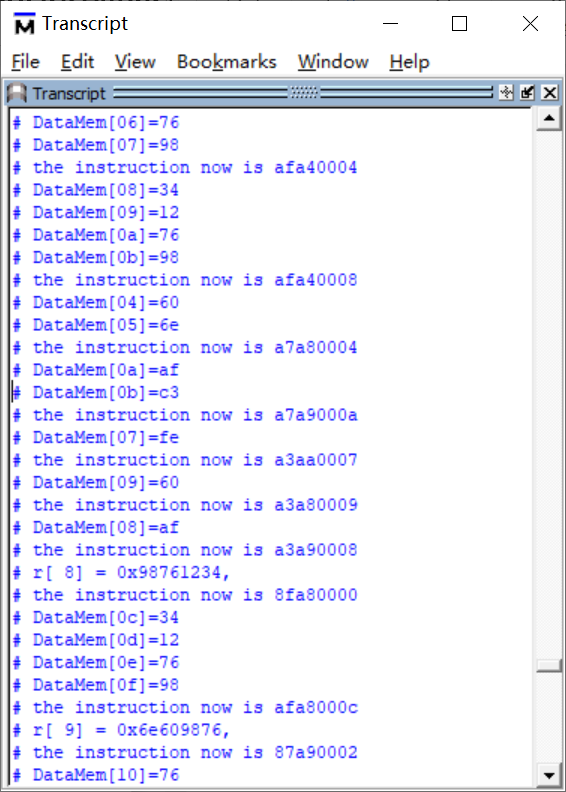


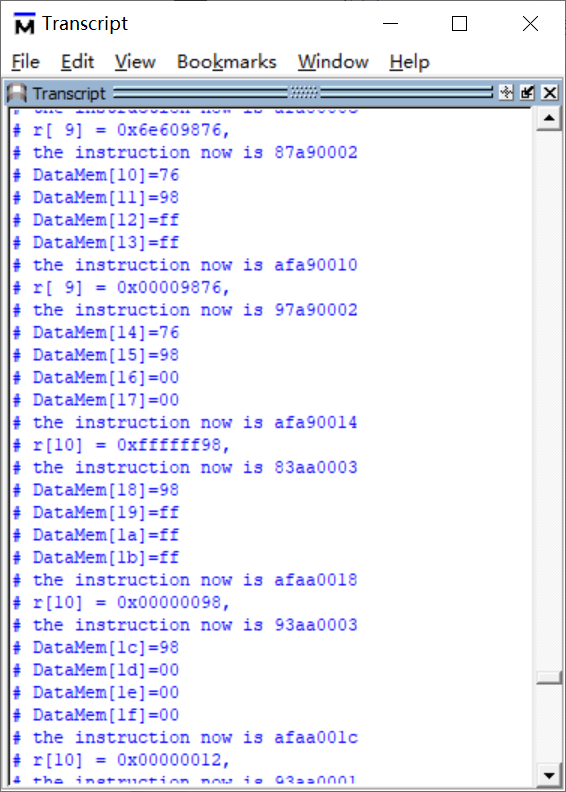


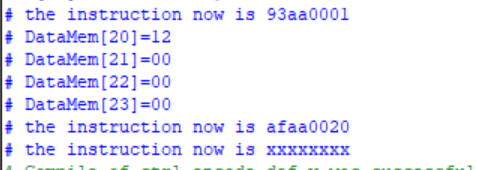
执行过程如下：





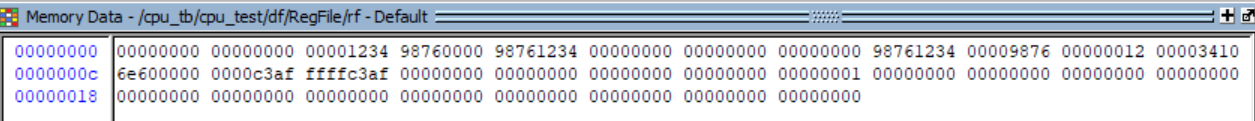


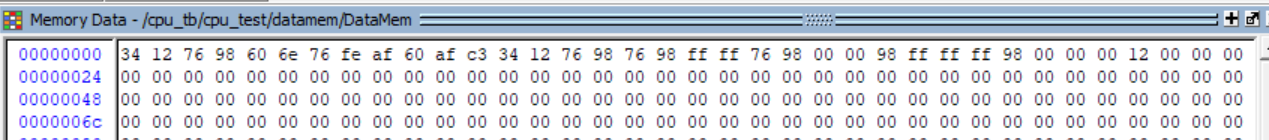




过程与预期相同

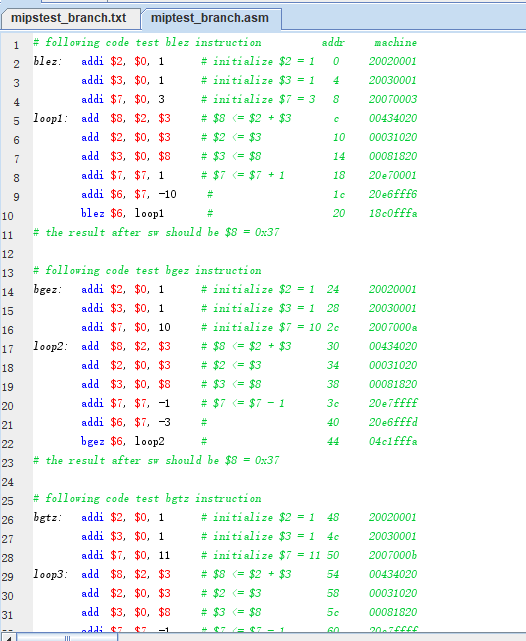
最终寄存器堆、数据存储器内容如下：

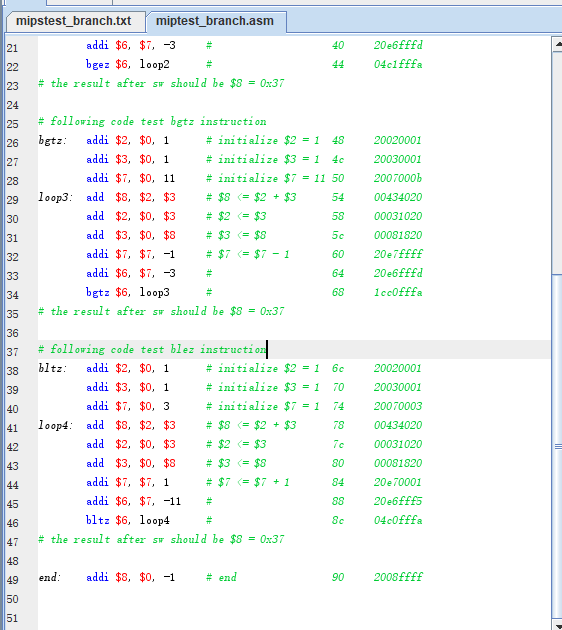




与预期结果相同，说明无误

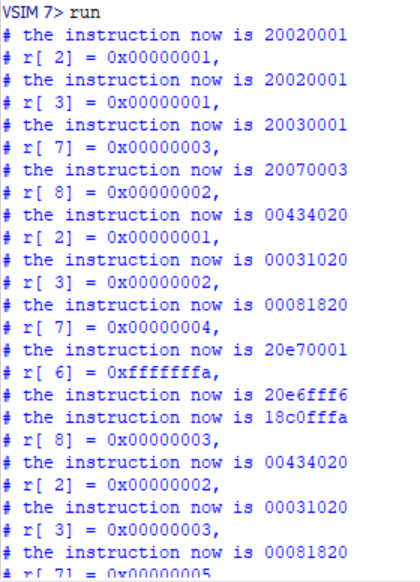
测试程序之五--mipstest\_branch:

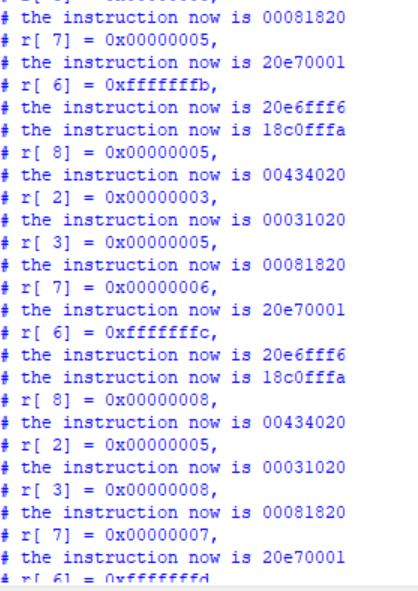


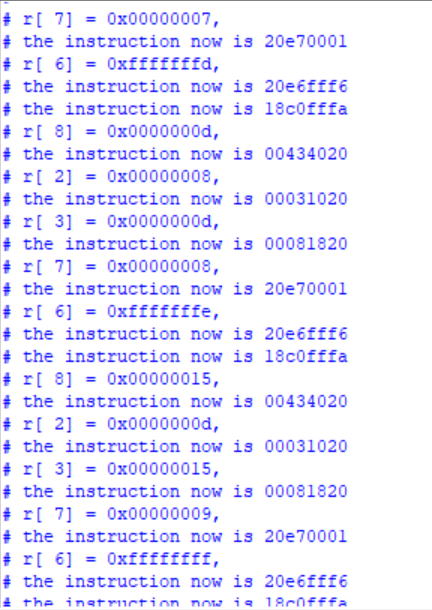


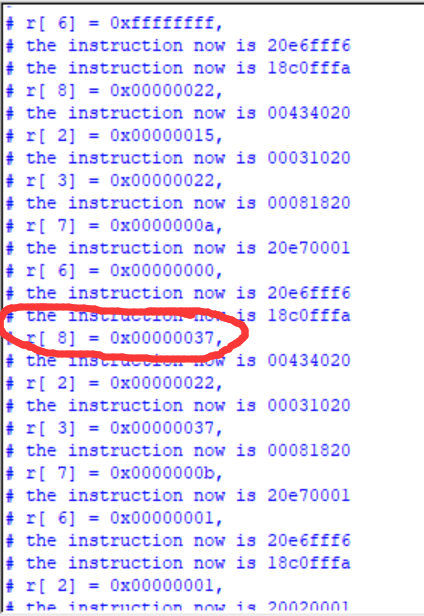
运行过程分析：

**blez:**

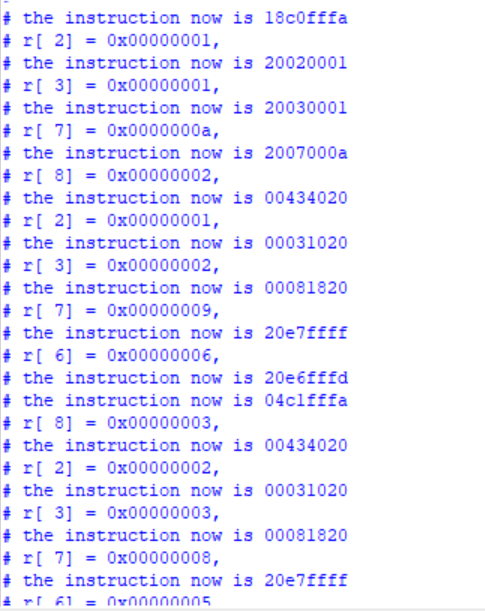


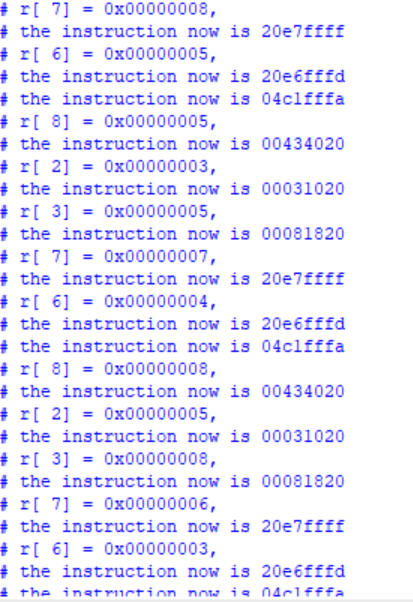


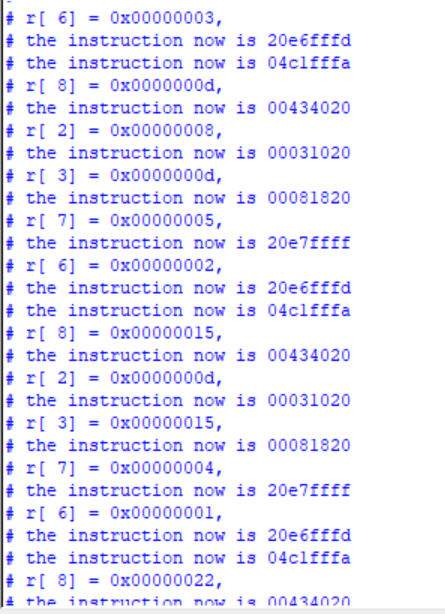


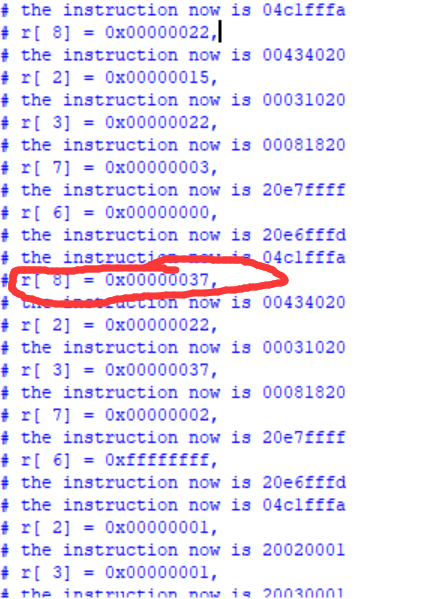


**bgez:**

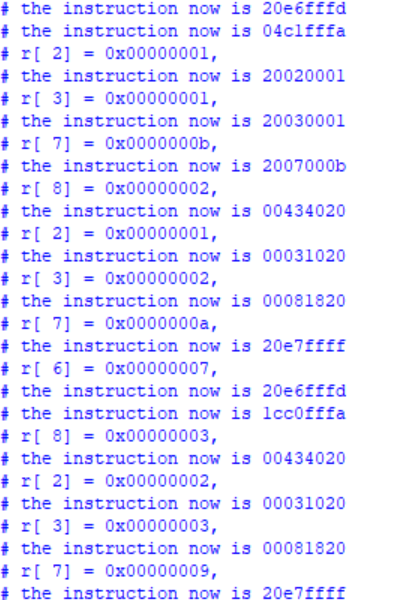


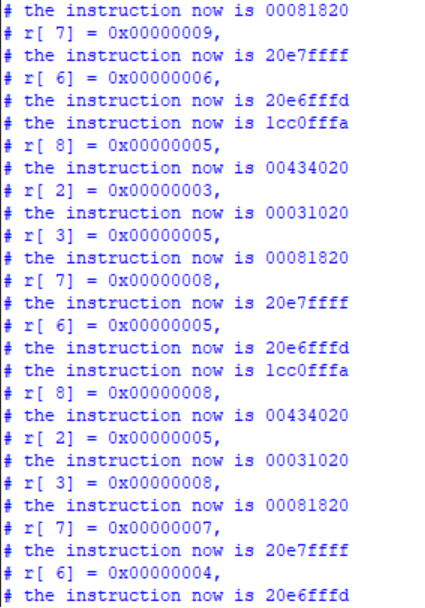


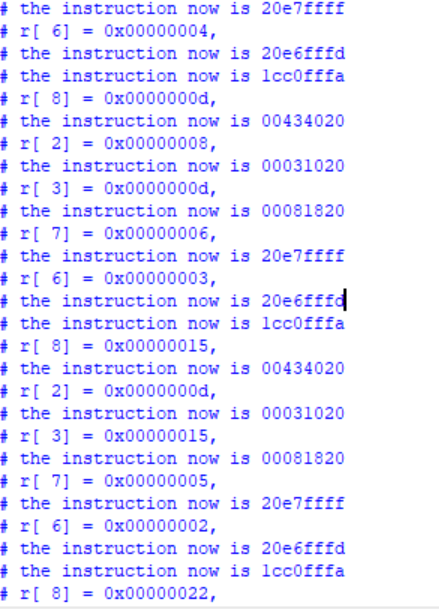


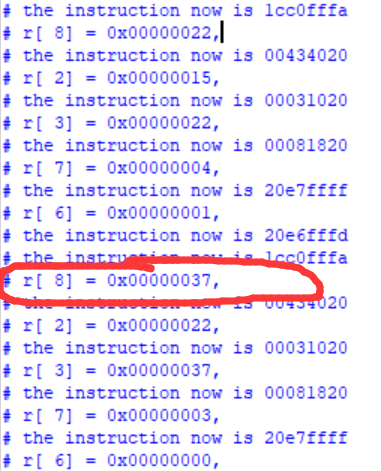


**bgtz:**

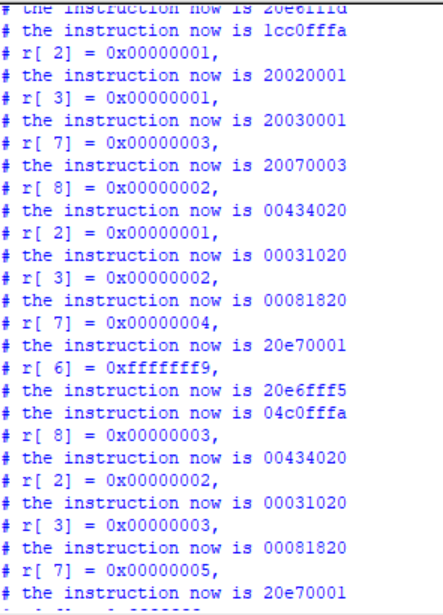


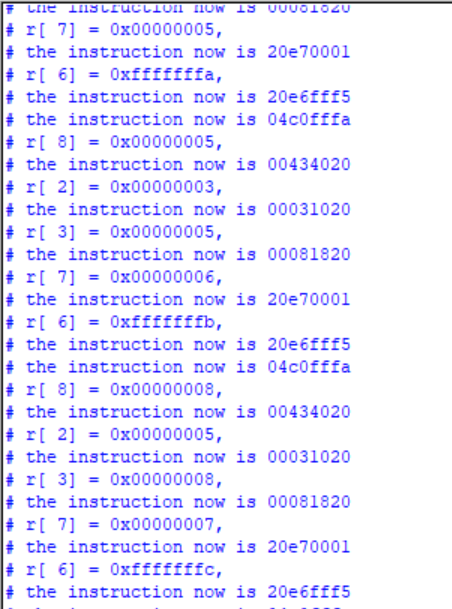


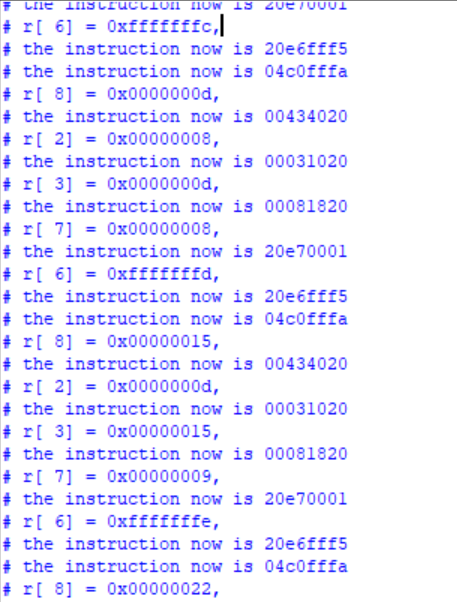


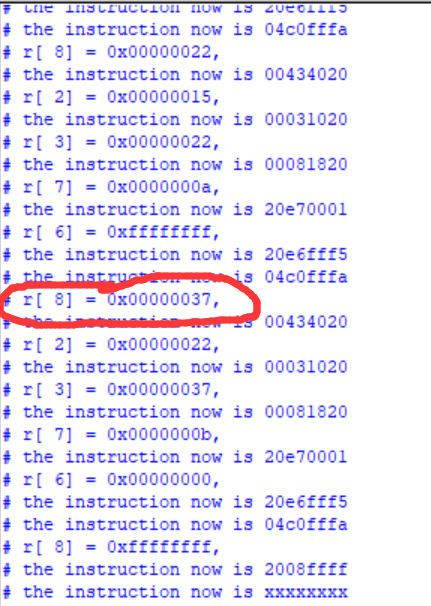


**bltz:**









四种跳转均与预期结果一致，说明无误

**四、反思**

在设计cpu的过程中，遇到的大问题如下：

1、一开始遇到了不知道如何把指令写入的问题，后查资料学习了$readmemh的用法；

设计初期比较头疼的是整体的代码结构究竟该如何规划。我阅读了大量的资料，选择了其中一种思路较为清晰的结构进行重写，使用简洁、易于区分的信号，变量名称，来使自己的代码思路清晰化。

2、设计初期对各条指令不够熟悉，花了大量的时间在分析每一条指令的来龙去脉上。也算是一种准备工作吧。在书上的Branch类指令中，书上的数据通路只给出了beq的思路（Branch&Zero），需要再拓展blez等与0比较的指令时被卡住了。后来与同学、学长交流了一下，提到了Sign符号位，利用Sign和Zero进行组合来对控制信号赋值，使得问题迎刃而解。（但有一点疑惑的是，在mips指令集中，该类指令为何在rt段的值都默认设为1，为何不设为0，这样便可以省下一个多选器）。

3、运行测试代码输出结果时，我一直在寻找一种更好体现过程的方法，后来学习到了$display的用法，可以在transcript框中体现每一条指令的执行结果、赋值过程，一目了然，方便调试。

4、关于lw型指令和sw型指令，我并没有把这两个类型都在DataMem中加以区分。我的方法是：在DataMem中往内存写数据时区分sw/sh/sb,在RegFile中往寄存器写数据时区分lw/lh/lb/lhu/lbu,我感觉这样不会让DataMem显得那么臃肿，也不用额外的读信号，只需要将RegWrite和MemWrite写信号进行稍微拓展就可以了。

5、总体来说，我被困扰最久的是如何规划适合42条指令的数据通路。教程中只支持20多条，如果直接把里面的一部分内容搬上去的话，不仅不理解，扩展指令也会很麻烦。于是我充分理解了每一步的含义，对代码进行重写，找到了一个比较齐全的数据通路图，顺着思路一步一步往下走。在设计控制单元时，信号与指令间的关系很容易出错，我后来调试也修改了其中的错误。还有一点比较不适应的地方在于verilog写电路实际上是一种并行的思想，通俗的来说是在连接一个电路图，而非平时我们写程序的串行执行的思想，后来我才慢慢适应过来。

感谢陈彦文、王嘉梁、生广明、徐泽辉同学，我们在交流中学习了许多东西。攻克这个难关还是挺有成就感的，感谢他们的帮助。