同济大学计算机系

计算机组成原理实验报告

学　　号 1953484

姓　　名 孙卓阳

专　　业 信息安全

授课老师 郝永涛

一、实验内容

31 条指令单周期 cpu：

1. 本实验应用 Verilog 语言实现 31 条 MIPS 指令 cpu 设计

2.实验目的：

深入理解 CPU 的原理

在理解的基础上画出 31 条指令 CPU 的通路图

学习使用 Verilog 语言设计实现 31 条指令的 CPU

二、实验原理

（1）实现 MIPS 的 31 条指令

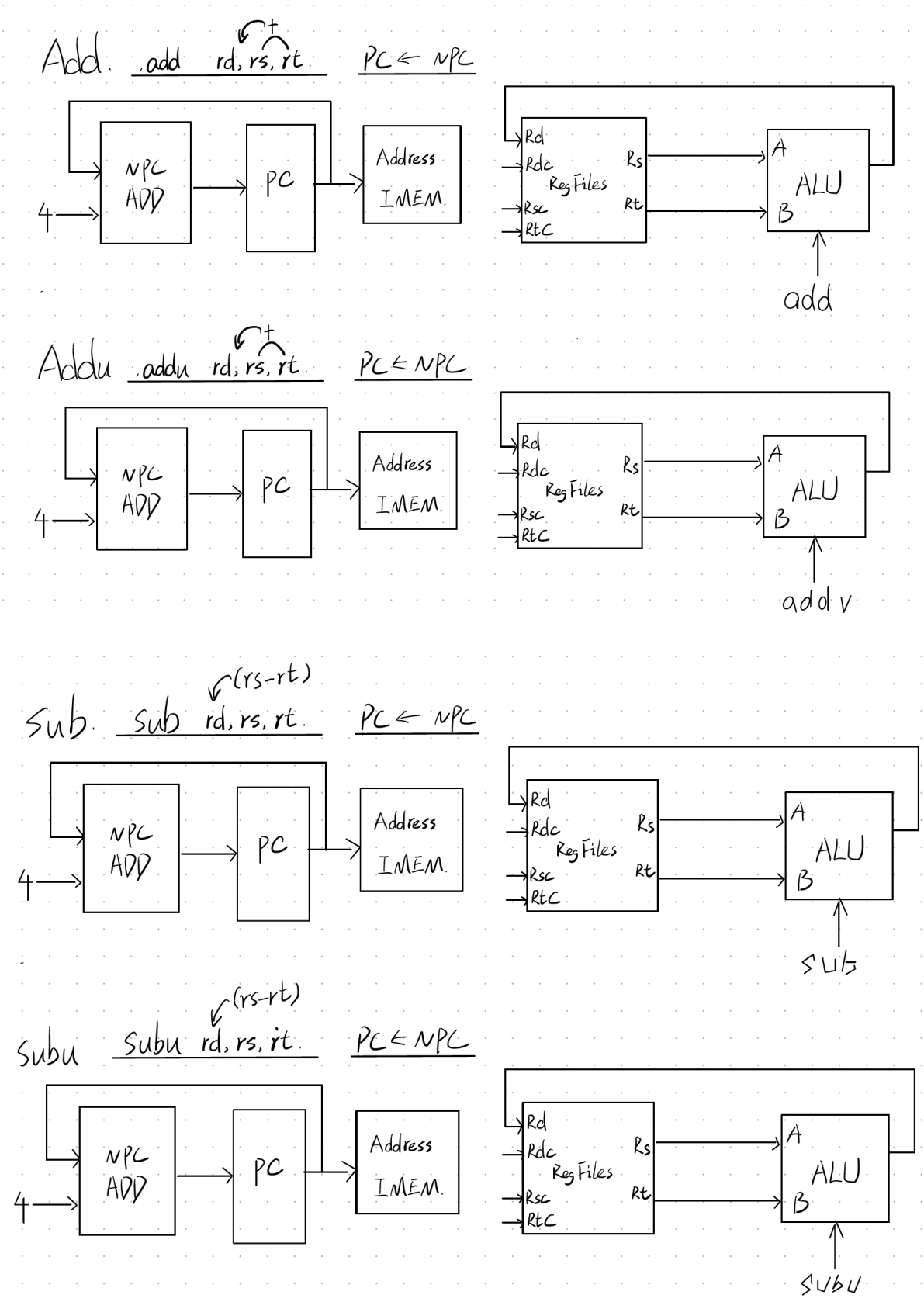
Mnemonic Symbol	Format						Sample
Bit #	31..26	25..21	20..16	15..11	10..6	5..0	
R-type	op	rs	rt	rd	shamt	func	
add	000000	rs	rt	rd	0	100000	add $1,$2,$3
addu	000000	rs	rt	rd	0	100001	addu $1,$2,$3
sub	000000	rs	rt	rd	0	100010	sub $1,$2,$3
subu	000000	rs	rt	rd	0	100011	subu $1,$2,$3
and	000000	rs	rt	rd	0	100100	and $1,$2,$3
or	000000	rs	rt	rd	0	100101	or $1,$2,$3
xor	000000	rs	rt	rd	0	100110	xor $1,$2,$3
nor	000000	rs	rt	rd	0	100111	nor $1,$2,$3
slt	000000	rs	rt	rd	0	101010	slt $1,$2,$3
sltu	000000	rs	rt	rd	0	101011	sltu $1,$2,$3
sll	000000	0	rt	rd	shamt	000000	sll $1,$2,10
srl	000000	0	rt	rd	shamt	000010	srl $1,$2,10
sra	000000	0	rt	rd	shamt	000011	sra $1,$2,10
sllv	000000	rs	rt	rd	0	000100	sllv $1,$2,$3
srlv	000000	rs	rt	rd	0	000110	srlv $1,$2,$3
srav	000000	rs	rt	rd	0	000111	srav $1,$2,$3
jr	000000	rs	0	0	0	001000	jr $31

Bit #	31..26	25..21	20..16	15..0	
I-type	op	rs	rt	immediate	
addi	001000	rs	rt	Immediate(- ~ +)	addi $1,$2,100
addiu	001001	rs	rt	Immediate(- ~ +)	addiu $1,$2,100
andi	001100	rs	rt	Immediate(0 ~ +)	andi $1,$2,10
ori	001101	rs	rt	Immediate(0 ~ +)	andi $1,$2,10
xori	001110	rs	rt	Immediate(0 ~ +)	andi $1,$2,10
lw	100011	rs	rt	Immediate(- ~ +)	lw $1,10($2)
sw	101011	rs	rt	Immediate(- ~ +)	sw $1,10($2)
beq	000100	rs	rt	Immediate(- ~ +)	beq $1,$2,10
bne	000101	rs	rt	Immediate(- ~ +)	bne $1,$2,10
slti	001010	rs	rt	Immediate(- ~ +)	slti $1,$2,10
sltiu	001011	rs	rt	Immediate(- ~ +)	sltiu $1,$2,10
lui	001111	00000	rt	Immediate(- ~ +)	Lui $1, 10
Bit #	31..26	25..0			
J-type	op	Index			
j	000010	address			j 10000
jal	000011	address			jal 10000

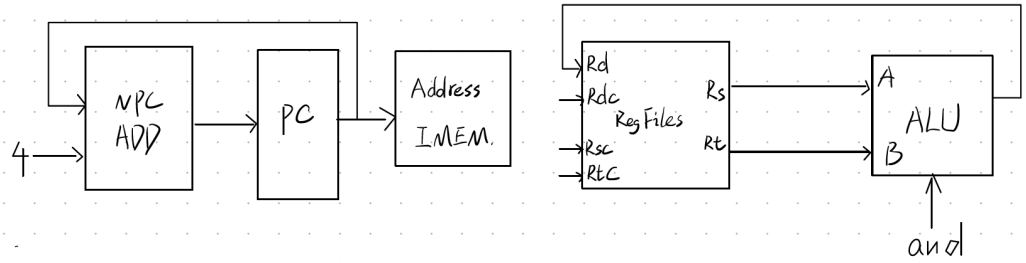
（2）设计思路

- 阅读 31 条指令的详细描述，对每条指令的功能与执行过程有详细的认知
- 确定每条指令执行过程中需要的部件
- 统计总共需要的部件，记录进表格，并标明每个部件的数据来源
- 根据各个部件之间的输入输出关系表格，绘制出整体的数据通路
- 根据每一条指令的数据通路图确定控制信号的值

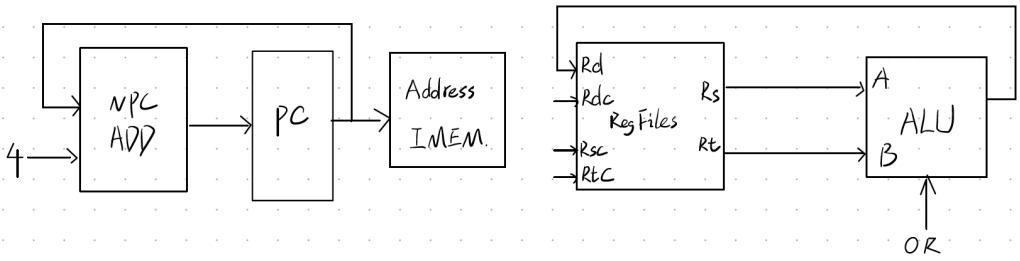
（3）每条指令的数据通路图



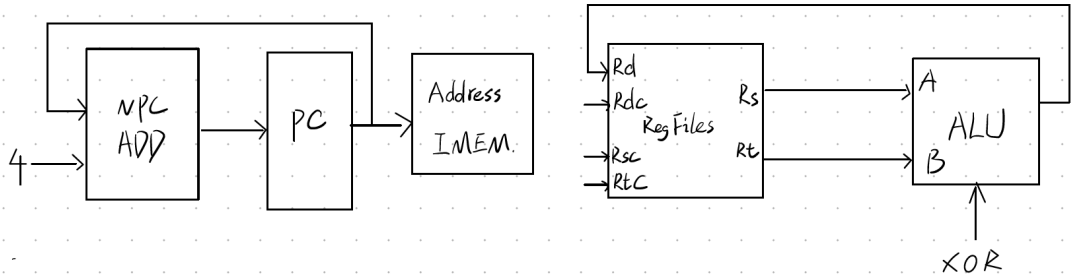
AND. and rd, rs, rt. PC ← NPC



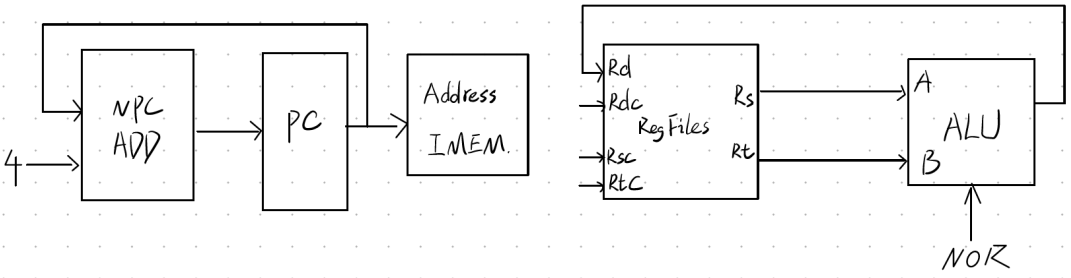
OR. or rd, rs, rt. PC ← NPC



異或
XOR. xor rd, rs, rt. PC ← NPC

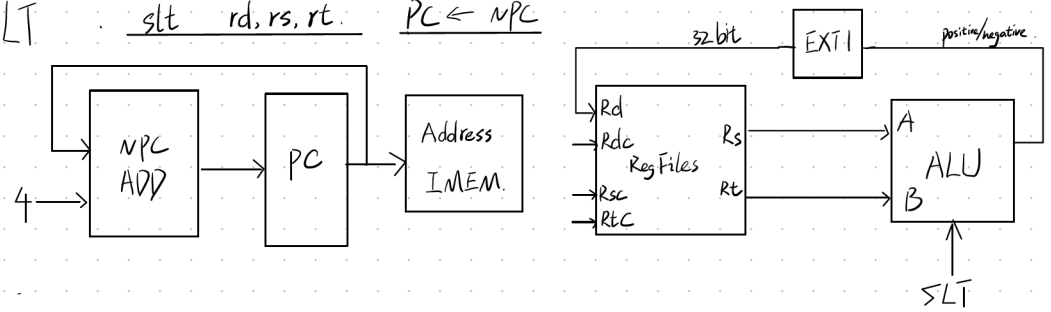


(Not or)
NOR. nor rd, rs, rt. PC ← NPC

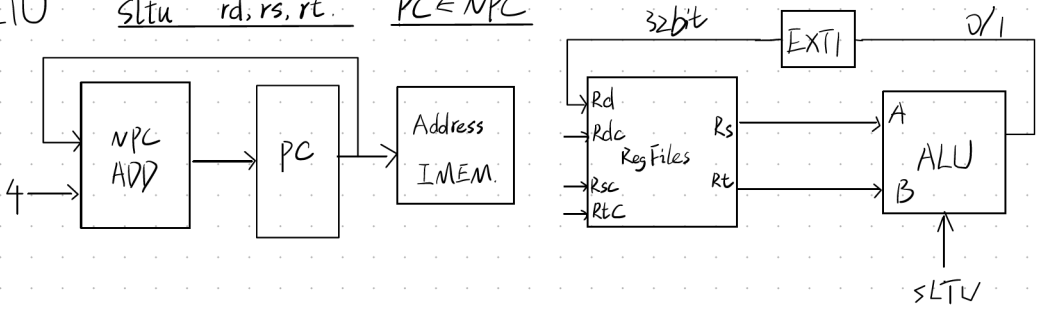


Set on Less Than. 看作有符号数比较 (rs < rt) 结果为 boolean. 须扩展至32位存入 rd.

SLT slt rd, rs, rt. PC ← NPC

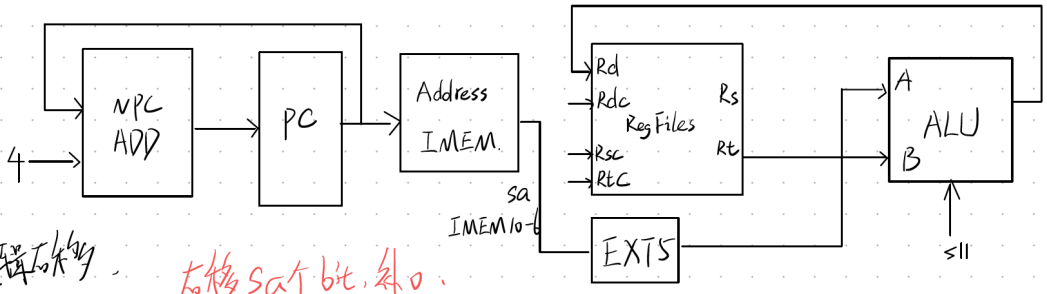


SLTU 无符号数比较 sltu rd, rs, rt. PC ← NPC



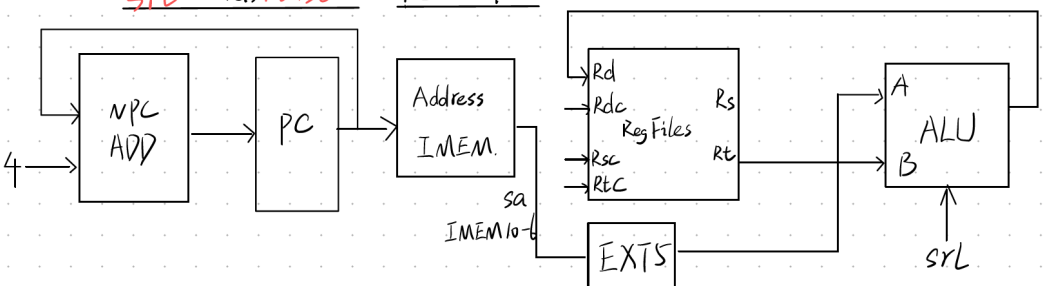
逻辑左移 左移 sa 个 bit, 补 0.

SLL sll rd, rt, sa. PC ← NPC



逻辑右移 右移 sa 个 bit, 补 0.

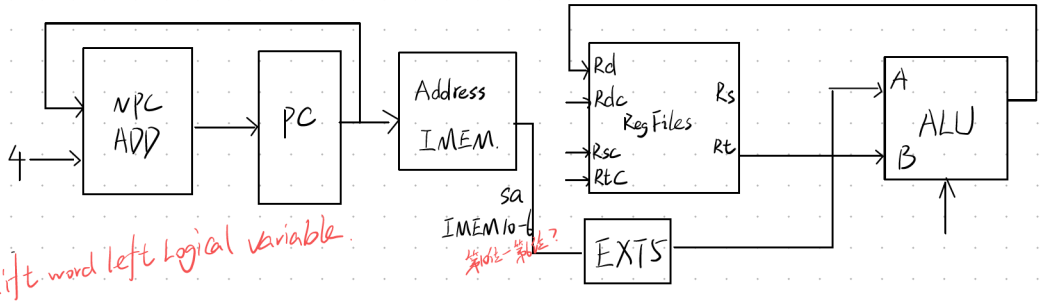
SRL srl rd, rt, sa PC ← NPC



算术右移

SRA 右移 sa 个 bit，补符号位。

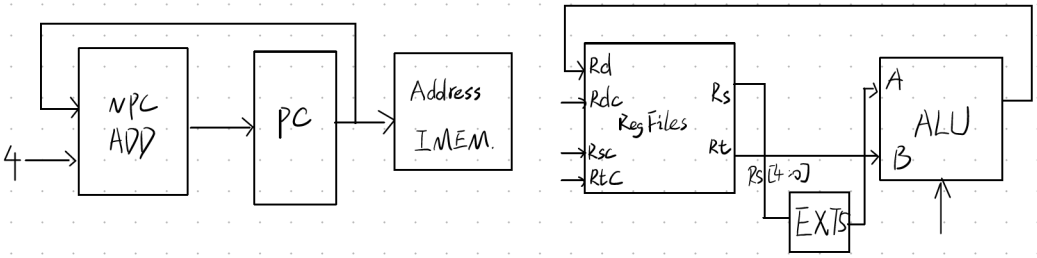
sra rd, rt, sa. PC ← NPC



shift word left Logical variable.

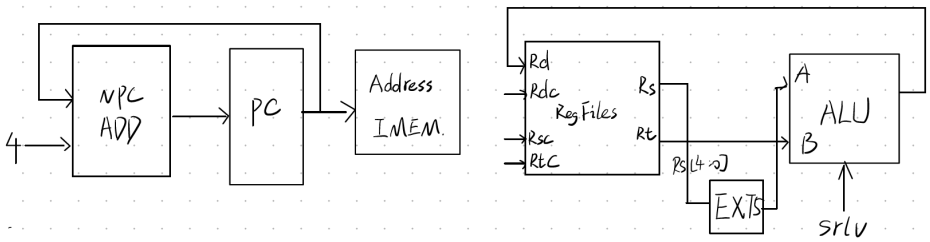
SLLV

sllv rd, rt, rs. PC ← NPC



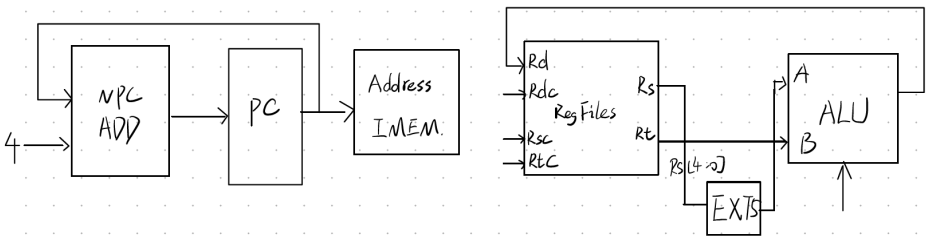
SRLV 同上一条。

srlv rd, rt, rs. PC ← NPC

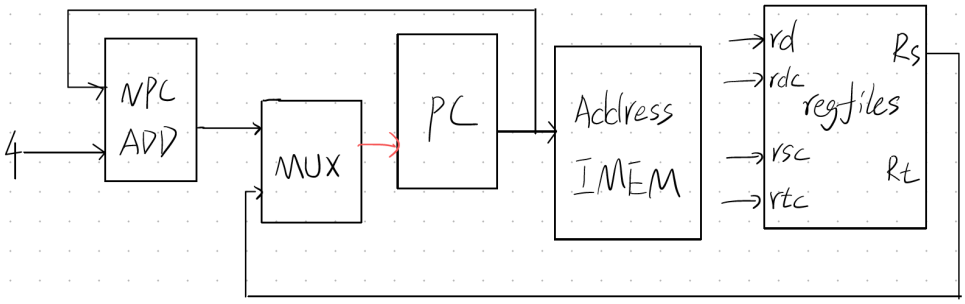


SRAV

srav rd, rt, rs PC ← NPC

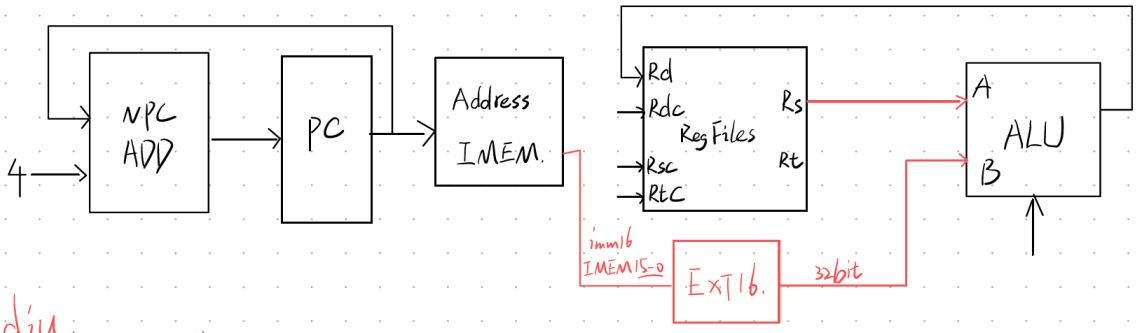


jr jump register.



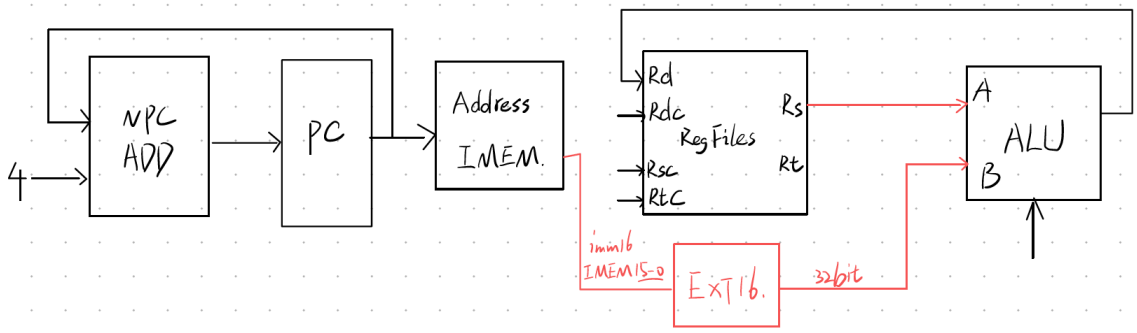
addi

addi rt,rs,,immediate PC ← NPC



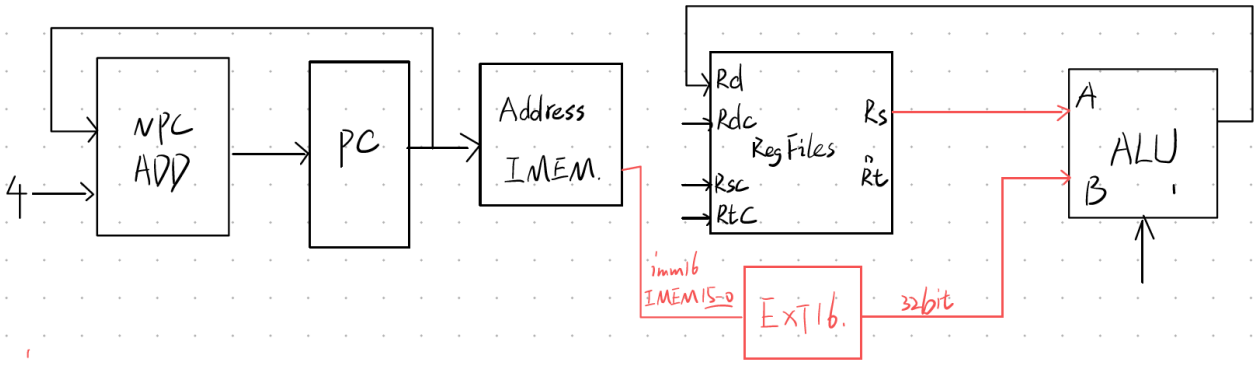
addiu.

addiu rt,rs,,immediate PC ← NPC



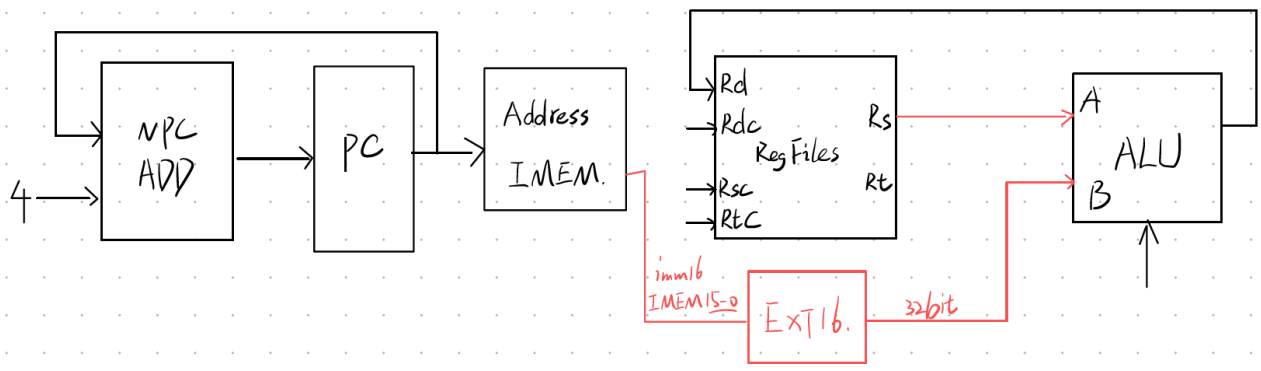
andi

andi rt,rs,,immediate PC ← NPC

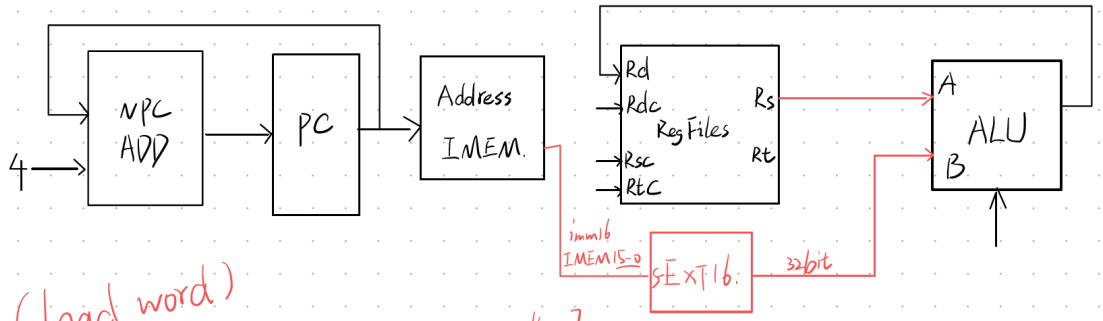


ori

ori rt,rs,,immediate PC ← NPC

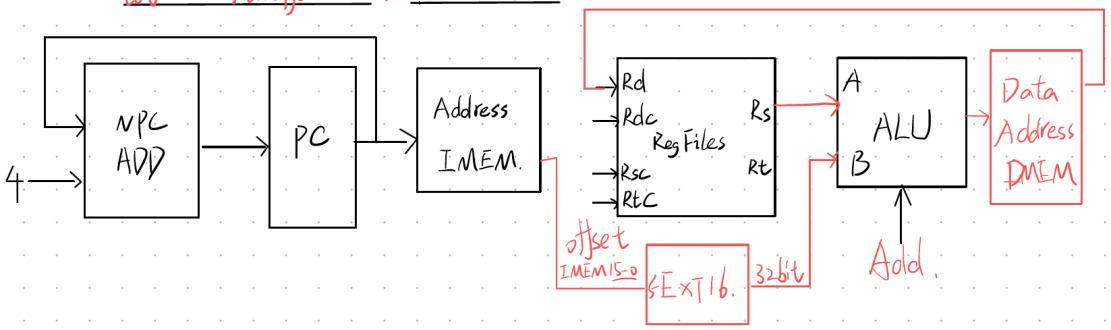


Xor'i xor'i rt,rs,,immediate PC ← NPC



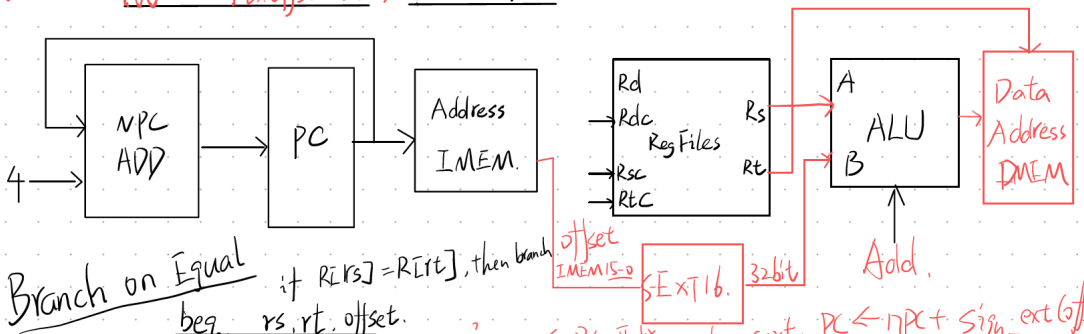
Lw (load word) rt ← memory[rs + sign_ext_offset]

lw rt, offset(base) PC ← NPC



Sw (Store word) memory[rs + sign_ex_offset] ← rt.

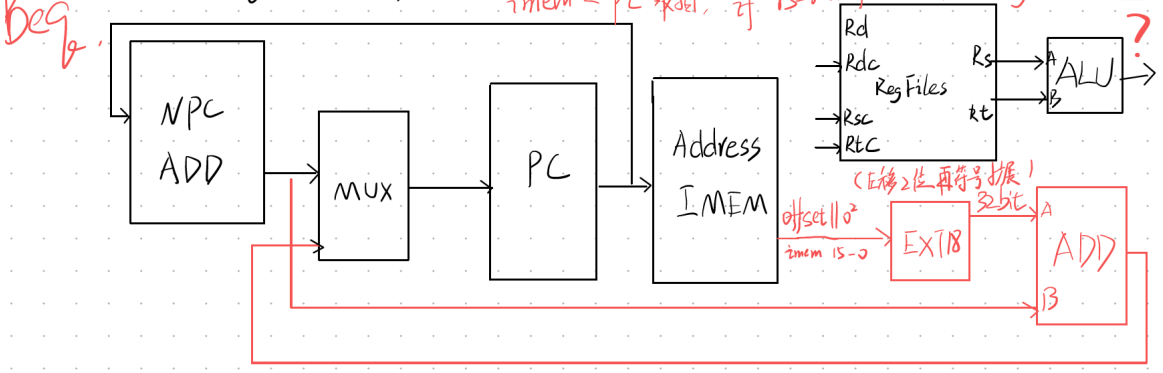
sw rt, offset(base) PC ← NPC



Beq. Branch on Equal if R[rs] = R[rt], then branch

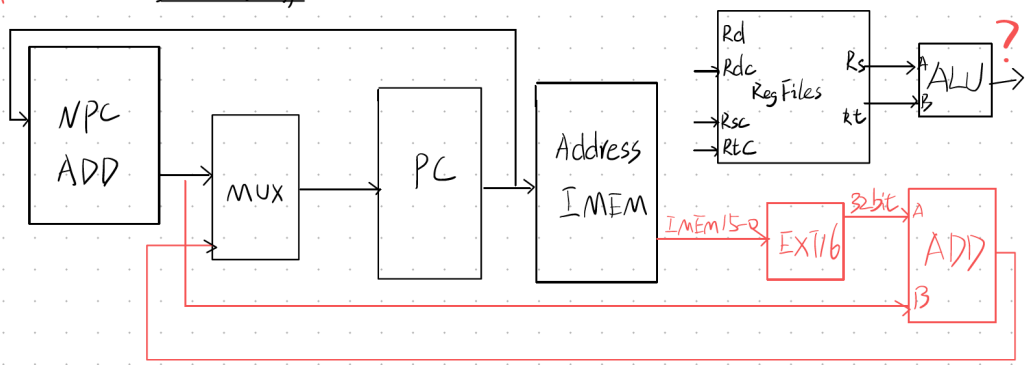
beq rs, rt, offset.

imem ← PL 取指, if rs=rt, PC ← nPC + sign_ext(offset||0)



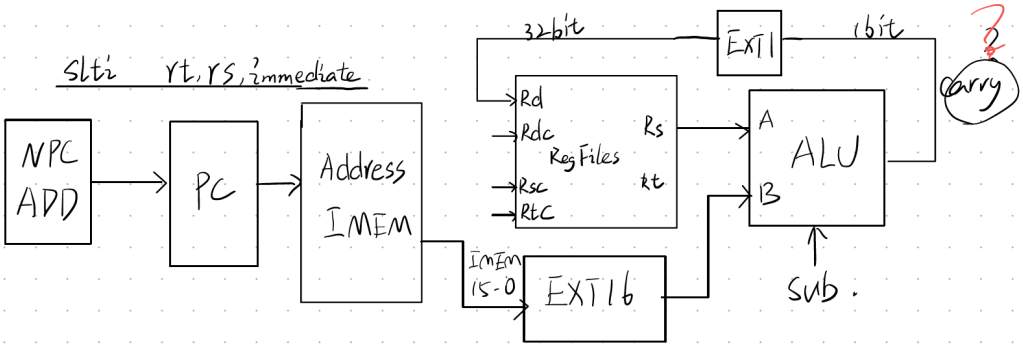
bne branch on not equal

bne rs, rt, offset.



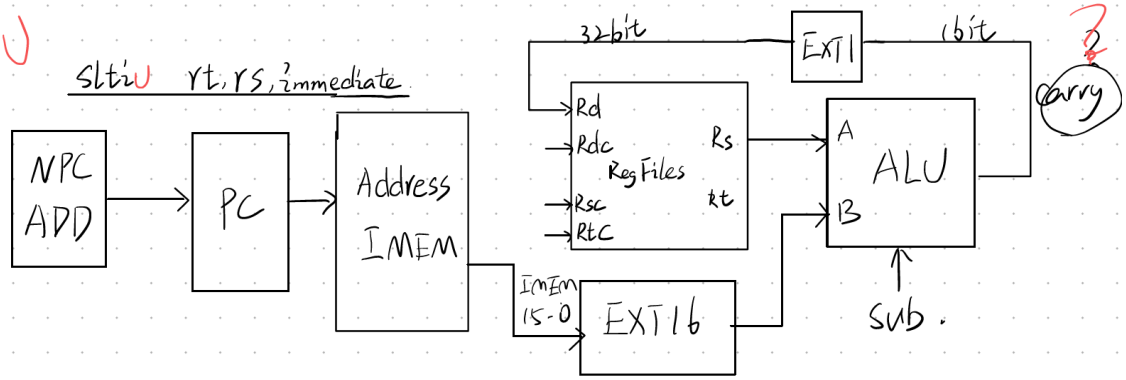
SLTI

slti rt, rs, immediate



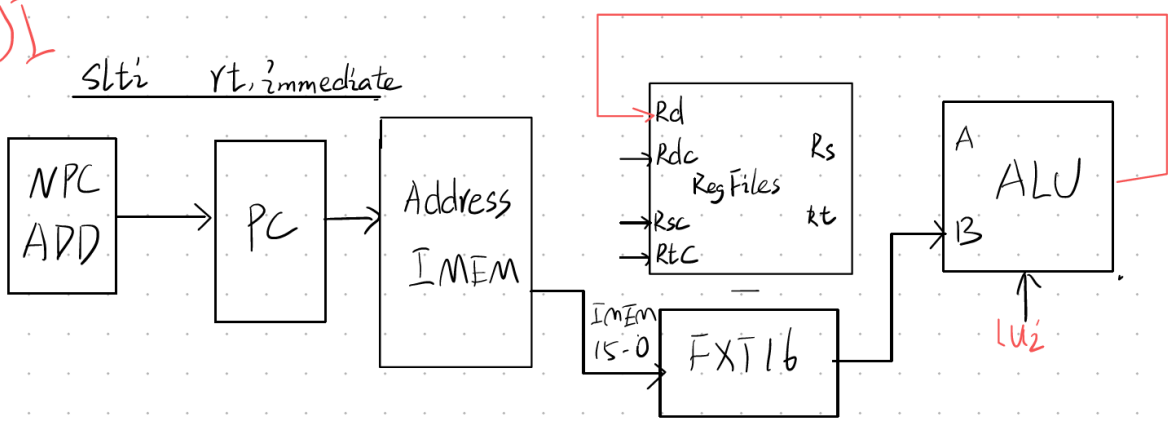
SLTIU

sltiu rt, rs, immediate

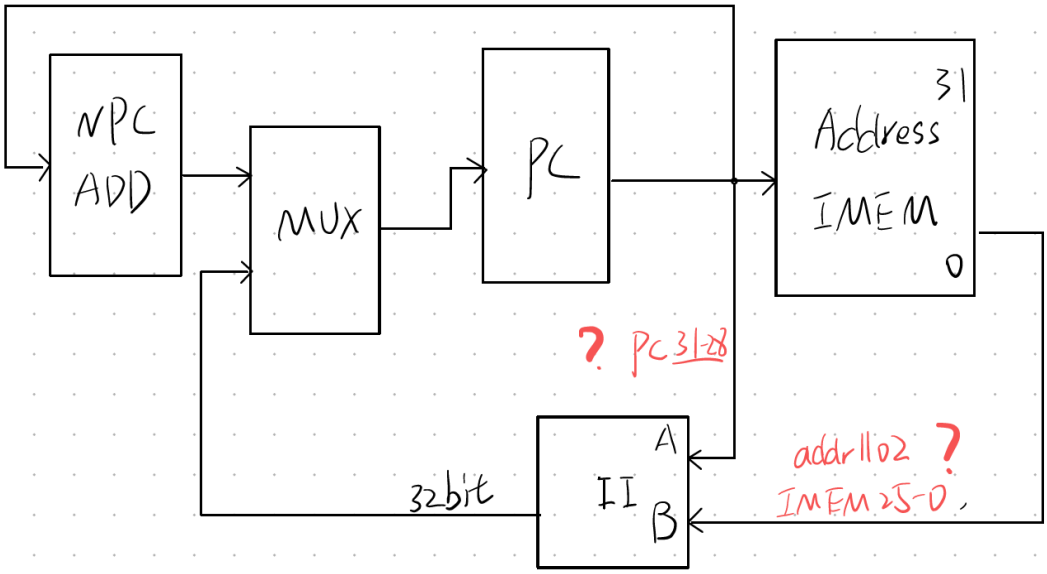


LUI

slti rt, immediate

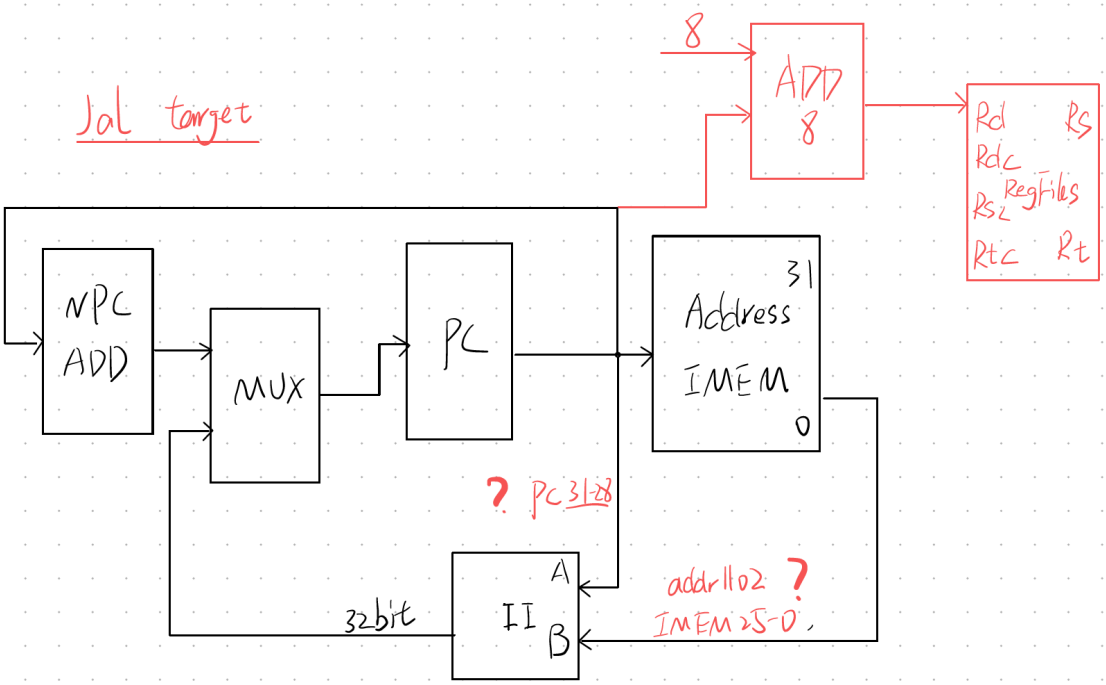


J (Jump) J target



Jal

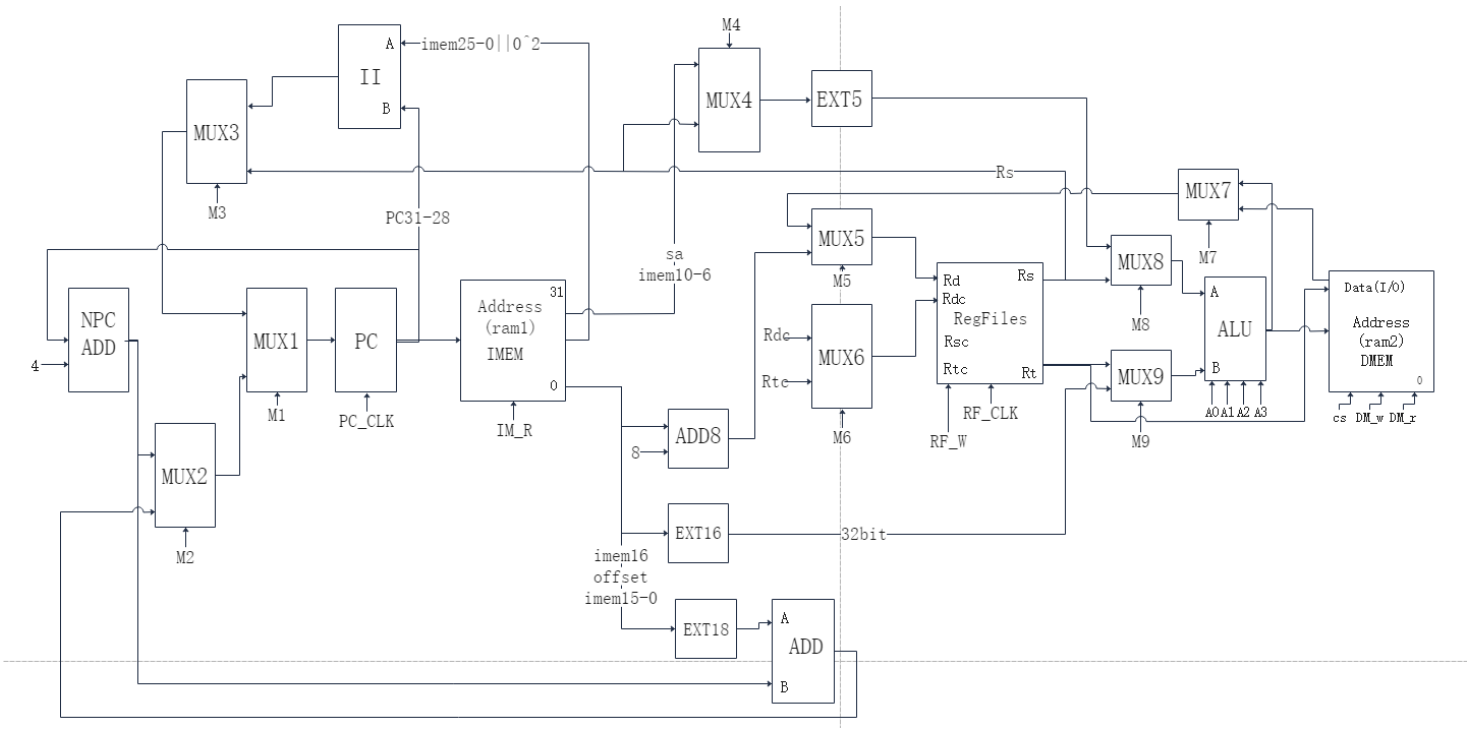
Jal target



（4）输入输出关系表

指令	PC	NPC	IM	RF	ALU		S-EXT16	EXT18	EXT-16	EXT5	EXT1	ADD		||		DMEM		ADD8
				Wdata	A	B						A	B	A	B	Addr	Data	
add	NPC	PC	PC	ALU	Rs	Rt												
addu	NPC	PC	PC	ALU	Rs	Rt												
sub	NPC	PC	PC	ALU	Rs	Rt												
subu	NPC	PC	PC	ALU	Rs	Rt												
and	NPC	PC	PC	ALU	Rs	Rt												
or	NPC	PC	PC	ALU	Rs	Rt												
xor	NPC	PC	PC	ALU	Rs	Rt												
nor	NPC	PC	PC	ALU	Rs	Rt												
slt	NPC	PC	PC	EXT1	Rs	Rt					ALU							
sltu	NPC	PC	PC	EXT1	Rs	Rt					ALU							
sll	NPC	PC	PC	ALU	EXT5	Rt				sa								
srl	NPC	PC	PC	ALU	EXT5	Rt				sa								
sra	NPC	PC	PC	ALU	EXT5	Rt				sa								
sllv	NPC	PC	PC	ALU	EXT5	Rt				Rs								
srlv	NPC	PC	PC	ALU	EXT5	Rt				Rs								
srav	NPC	PC	PC	ALU	EXT5	Rt				Rs								
jr	Rs	PC	PC															
addi	NPC	PC	PC	ALU	Rs	EXT16			Imm16									
addiu	NPC	PC	PC	ALU	Rs	EXT16			Imm16									
andi	NPC	PC	PC	ALU	Rs	EXT16			Imm16									
ori	NPC	PC	PC	ALU	Rs	EXT16			Imm16									
xori	NPC	PC	PC	ALU	Rs	EXT16			Imm16									
lw	NPC	PC	PC	DM	Rs	S-EXT16	Offset									ALU		
sw	NPC	PC	PC		Rs	S-EXT16	Offset									ALU	Rt	
beq	ADD	PC	PC		Rs	Rt		Offset				EXT18	NPC					
bne	ADD	PC	PC		Rs	Rt		Offset				EXT18	NPC					
slti	NPC	PC	PC	EXT1	Rs	EXT16			Imm16		ALU							
sltiu	NPC	PC	PC	EXT1	Rs	EXT16			Imm16		ALU							
lui	NPC	PC	PC	ALU		EXT16												
j	||	PC	PC											PC31-28	IMEM25-0			
jal	||	PC	PC	ADD8										PC31-28	IMEM25-0			PC

（5）整体的数据通路



（6）控制信号设计

	add	addu	sub	subu	and	or	xor	nor	slt	sltu	sll	srl	sra	sllv	srlv	srav	jr	addi	addiu	andi	ori	xori	lw	sw	beq	bne	slti	sltiu	lui	j	jal
PC_CLK	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
IM_R	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Rsc4-0	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21				IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21	IM25-21
Rtc4-0	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16								IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16	IM20-16
M1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0
M2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
M3																	1													0	0
M4											0	0	0	1	1	1															
M5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	0				0	0	0		1
M6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		1	1	1	1	1	1	1			1	1	1		
M7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		0	0	0	0	0	1				0	0	0		
M8	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0		1	1	1	1	1	1	1	1	1	1	1			
M9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		1	1	1	1	1	1	1	0	0	1	1	1		
A0	0	0	1	1	0	1	0	1	1	0	X	1	0	X	1	0		0	0	0	1	0	0	0	1	1	1	0	x		
A1	1	0	1	0	0	0	1	1	1	1	1	0	0	1	0	0		1	0	0	0	1	1	1	1	1	1	1	0		
A2	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	1		0	0	1	1	1	0	0	0	0	0	0	0		
A3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1		0	0	0	0	0	0	0	0	0	1	1	1		
Rdc4-0	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11	IM15-11		20-16	20-16	20-16	20-16	20-16	20-16	20-16			20-16	20-16	20-16		
RF_W	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	0	1	1	1	0	1
RF_CLK	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	0	1	1	1	0	1
CS	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
DM_w	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
DM_r	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0

三、测试过程及结果

前仿真：

部分指令，自己的 cpu 输出结果与 mars 的结果对比

```
D:\>fc mips_31_mars_simulate.txt result.txt
正在比较文件 mips_31_mars_simulate.txt 和 RESULT.TXT
FC: 找不到差异
```

```
D:\>fc  _2_lwsw.txt result.txt
正在比较文件  _2_lwsw.txt 和 RESULT.TXT
FC: 找不到差异


D:\>fc  _2_lwsw2.txt result.txt
正在比较文件  _2_lwsw2.txt 和 RESULT.TXT
FC: 找不到差异
```

四、各模块实现代码

（1）sccomp_dataflow ，本实验的顶层模块，用于连接 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 [10:0] data_addr;
    wire [31:0] inst_addr;
    wire IM_R, DM_cs, DM_w, DM_r;
    wire [31:0] wdata, rdata, r;

    assign inst_addr = pc - 32'h0040_0000;
    assign data_addr = (r - 32'h10010000) /4;

    mips_31_mars_simulate imem (inst_addr[12:2], inst);
    Dram dmem (
        clk_in,
        DM_cs, DM_w, DM_r,
        data_addr, wdata, rdata
        );
    cpu sccpu(
        clk_in, reset, inst, rdata,
        IM_R,
        DM_cs, DM_r, DM_w,
        pc, r, wdata
        );
endmodule
```

（2）Iram，ip 核实现，用于指令的读取

```
// (c) Copyright 1995-2021 Xilinx, Inc. All rights reserved.
//
// This file contains confidential and proprietary information
// of Xilinx, Inc. and is protected under U.S. and
```

```
5. // international copyright and other intellectual property
6. // laws.
7. //
8. // DISCLAIMER
9. // This disclaimer is not a license and does not grant any
10.// rights to the materials distributed herewith. Except as
11.// otherwise provided in a valid license issued to you by
12.// Xilinx, and to the maximum extent permitted by applicable
13.// law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND
14.// WITH ALL FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTIES
15.// AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING
16.// BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-
17.// INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and
18.// (2) Xilinx shall not be liable (whether in contract or tort,
19.// including negligence, or under any other theory of
20.// liability) for any loss or damage of any kind or nature
21.// related to, arising under or in connection with these
22.// materials, including for any direct, or any indirect,
23.// special, incidental, or consequential loss or damage
24.// (including loss of data, profits, goodwill, or any type of
25.// loss or damage suffered as a result of any action brought
26.// by a third party) even if such damage or loss was
27.// reasonably foreseeable or Xilinx had been advised of the
28.// possibility of the same.
29.//
30.// CRITICAL APPLICATIONS
31.// Xilinx products are not designed or intended to be fail-
32.// safe, or for use in any application requiring fail-safe
33.// performance, such as life-support or safety devices or
34.// systems, Class III medical devices, nuclear facilities,
35.// applications related to the deployment of airbags, or any
36.// other applications that could lead to death, personal
37.// injury, or severe property or environmental damage
38.// (individually and collectively, "Critical
39.// Applications"). Customer assumes the sole risk and
40.// liability of any use of Xilinx products in Critical
41.// Applications, subject only to applicable laws and
42.// regulations governing limitations on product liability.
43.//
44.// THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS
45.// PART OF THIS FILE AT ALL TIMES.
46.//
47.// DO NOT MODIFY THIS FILE.
48.
```

```
49.
50.// IP VLNV: xilinx.com:ip:dist_mem_gen:8.0
51.// IP Revision: 10
52.
53.`timescale 1ns/1ps
54.
55.(* DowngradeIPIdentifiedWarnings = "yes" *)
56.module mips_31_mars_simulate (
57.  a,
58.  spo
59.);
60.
61.input wire [10 : 0] a;
62.output wire [31 : 0] spo;
63.
64.  dist_mem_gen_v8_0_10 #(
65.    .C_FAMILY("artix7"),
66.    .C_ADDR_WIDTH(11),
67.    .C_DEFAULT_DATA("0"),
68.    .C_DEPTH(2048),
69.    .C_HAS_CLK(0),
70.    .C_HAS_D(0),
71.    .C_HAS_DPO(0),
72.    .C_HAS_DPRA(0),
73.    .C_HAS_I_CE(0),
74.    .C_HAS_QDPO(0),
75.    .C_HAS_QDPO_CE(0),
76.    .C_HAS_QDPO_CLK(0),
77.    .C_HAS_QDPO_RST(0),
78.    .C_HAS_QDPO_SRST(0),
79.    .C_HAS_QSPO(0),
80.    .C_HAS_QSPO_CE(0),
81.    .C_HAS_QSPO_RST(0),
82.    .C_HAS_QSPO_SRST(0),
83.    .C_HAS_SPO(1),
84.    .C_HAS_WE(0),
85.    .C_MEM_INIT_FILE("mips_31_mars_simulate.mif"),
86.    .C_ELABORATION_DIR("./"),
87.    .C_MEM_TYPE(0),
88.    .C_PIPELINE_STAGES(0),
89.    .C_QCE_JOINED(0),
90.    .C_QUALIFY_WE(0),
91.    .C_READ_MIF(1),
92.    .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
```

（3）Dram，数据存储器，用于向内存中读写数据

```
`timescale 1ns / 1ps

module Dram(
    input clk,
    input DM_cs,
    input DM_w,
    input DM_r,
    input [10:0] addr,
    input [31:0] data_in,
    output [31:0] data_out
    );

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

    assign data_out = (DM_r && DM_cs) ? mem[addr] : 32'hz;

    always@(posedge clk) begin
```

```
        if(DM_w && DM_cs) begin
            mem[addr] <= data_in;
        end
    end
endmodule

```

（4）CPU，链接所有指令执行需要的实例化部件

```
`timescale 1ns / 1ps
module cpu(
    input         clk,
    input         reset,
    input  [31:0] inst,
    input  [31:0] rdata,
    output        IM_r,
    output        DM_cs,
    output        DM_r,
    output        DM_w,
    output [31:0] pc,
    output [31:0] addr,
    output [31:0] wdata
    );
    //npc
    wire [31:0] npc;
    //pc
    wire [31:0] pc_data;
    wire pc_clk;
    //regfiles
    wire [31:0] rd;
    wire [4:0] rdc;
    wire [4:0] rtc, rsc;
    wire RF_w;
    wire RF_clk;
    wire [31:0] rs, rt;
    //alu
    wire [31:0] op1, op2;
    wire [3:0] aluc;
    wire [31:0] r;
    //mux
    wire m1, m2, m3, m4, m5, m6, m7, m8, m9;
    wire [31:0] data1, data2, data3, data5, data7, data8, data9;
    wire [4:0] data4, data6;
    //sign
```

```
36.     wire zf, of, cf, nf;
37.     //states
38.     wire [31:0] choose;
39.     wire [31:0] ext5_data;
40.     wire ext16_sign;
41.     wire [31:0] ext16_data;
42.     wire [31:0] ext18_data;
43.     wire [31:0] add_data;
44.     wire [31:0] add8_data;
45.     wire [31:0] ii_data;
46.
47.     assign npc = pc + 4;
48.     assign pc = pc_data;
49.     pcreg cpu_pc (
50.         pc_clk,
51.         reset,
52.         data1,
53.         pc_data
54.         );
55.     //regfile
56.     assign rd = data5, rdc = data6;
57.     assign rsc = inst[25:21], rtc = inst[20:16];
58.     regfile cpu_ref (
59.         RF_clk, reset, RF_w,
60.         rsc, rtc, rdc, rd,
61.         rs, rt
62.         );
63.     //alu
64.     assign addr = r;
65.     assign wdata = rt;
66.     assign op1 = data8, op2 = data9;
67.     alu cpu_alu (
68.         op1, op2, aluc,
69.         r,
70.         zf, cf, nf, of
71.         );
72.     //mux
73.     mux mux1(data3, data2, m1, data1);
74.     mux mux2(npc, add_data, m2, data2);
75.     mux mux3(ii_data, rs, m3, data3);
76.     mux5 mux4 (inst[10:6], rs[4:0], {choose[30], m4}, data4);
77.     mux mux5(data7, add8_data, m5, data5);////////////
78.     mux5 mux6 (inst[15:11], inst[20:16], {choose[30], m6}, data6)
    ;
```

```
    mux mux7(r, rdata, m7, data7);
    mux mux8(ext5_data, rs, m8, data8);
    mux mux9(rt, ext16_data, m9, data9);
    //states
    instr_decoder choice (inst, choose);
    operation cpu_states (
        clk, zf, choose, pc_clk, IM_r,
        m1, m2, m3, m4, m5, m6, m7, m8, m9,
        aluc,
        RF_w, RF_clk,
        DM_cs, DM_w, DM_r,
        ext16_sign
        );
    //ext5
    extend5 cpu_ext5_4 (data4, ext5_data);
    //ext16
    extend16 cpu_ext16 (inst[15:0], ext16_sign, ext16_data);
    //ext18
    extend18 cpu_ext18 (inst[15:0], ext18_data);
    //add
    add cpu_add (ext18_data, npc, add_data);
     //add8
     add8 cpu_add8 (pc_data, add8_data);
     //ii
     II cpu_ii (pc_data[31:28], inst[25:0], ii_data);

endmodule
```

（5）instr_decoder，指令译码器，将不同指令译码

```
`timescale 1ns / 1ps

module instr_decoder(
    input [31:0] inst,
    output reg [31:0] choose
    );
    always @(*)
    begin
    if(inst[31:26] == 6'b0)begin
        if(inst[5:0] == 6'b100000)     choose = 32'b0000_0000_0000_0000_0000_0000_0000_0001;//add
        else if(inst[5:0] == 6'b100001)choose = 32'b0000_0000_0000_0000_0000_0000_0000_0010;//addu
```

```
12.         else if(inst[5:0] == 6'b100010)choose = 32'b0000_0000_000
    0_0000_0000_0000_0000_0100;//sub
13.         else if(inst[5:0] == 6'b100011)choose = 32'b0000_0000_000
    0_0000_0000_0000_0000_1000;//subu
14.         else if(inst[5:0] == 6'b100100)choose = 32'b0000_0000_000
    0_0000_0000_0000_0001_0000;//and
15.         else if(inst[5:0] == 6'b100101)choose = 32'b0000_0000_000
    0_0000_0000_0000_0010_0000;//or
16.         else if(inst[5:0] == 6'b100110)choose = 32'b0000_0000_000
    0_0000_0000_0000_0100_0000;//xor
17.         else if(inst[5:0] == 6'b100111)choose = 32'b0000_0000_000
    0_0000_0000_0000_1000_0000;//nor
18.         else if(inst[5:0] == 6'b101010)choose = 32'b0000_0000_000
    0_0000_0000_0001_0000_0000;//slt
19.         else if(inst[5:0] == 6'b101011)choose = 32'b0000_0000_000
    0_0000_0000_0010_0000_0000;//sltu
20.         else if(inst[5:0] == 6'b000000)choose = 32'b0000_0000_000
    0_0000_0000_0100_0000_0000;//sll
21.         else if(inst[5:0] == 6'b000010)choose = 32'b0000_0000_000
    0_0000_0000_1000_0000_0000;//srl
22.         else if(inst[5:0] == 6'b000011)choose = 32'b0000_0000_000
    0_0000_0001_0000_0000_0000;//sra
23.         else if(inst[5:0] == 6'b000100)choose = 32'b0000_0000_000
    0_0000_0010_0000_0000_0000;//sllv
24.         else if(inst[5:0] == 6'b000110)choose = 32'b0000_0000_000
    0_0000_0100_0000_0000_0000;//srlv
25.         else if(inst[5:0] == 6'b000111)choose = 32'b0000_0000_000
    0_0000_1000_0000_0000_0000;//srav
26.         else if(inst[5:0] == 6'b001000)choose = 32'b0000_0000_000
    0_0001_0000_0000_0000_0000;//jr
27.     end
28.     else if(inst[31:26] == 6'b001000)choose = 32'b0000_0000_0000_
    0010_0000_0000_0000_0000;//addi
29.     else if(inst[31:26] == 6'b001001)choose = 32'b0000_0000_0000_
    0100_0000_0000_0000_0000;//addiu
30.     else if(inst[31:26] == 6'b001100)choose = 32'b0000_0000_0000_
    1000_0000_0000_0000_0000;//andi
31.     else if(inst[31:26] == 6'b001101)choose = 32'b0000_0000_0001_
    0000_0000_0000_0000_0000;//ori
32.     else if(inst[31:26] == 6'b001110)choose = 32'b0000_0000_0010_
    0000_0000_0000_0000_0000;//xori
33.     else if(inst[31:26] == 6'b100011)choose = 32'b0000_0000_0100_
    0000_0000_0000_0000_0000;//lw
```

```
    else if(inst[31:26] == 6'b101011)choose = 32'b0000_0000_1000_0000_0000_0000_0000_0000;//sw
    else if(inst[31:26] == 6'b000100)choose = 32'b0000_0001_0000_0000_0000_0000_0000_0000;//beq
    else if(inst[31:26] == 6'b000101)choose = 32'b0000_0010_0000_0000_0000_0000_0000_0000;//bne
    else if(inst[31:26] == 6'b001010)choose = 32'b0000_0100_0000_0000_0000_0000_0000_0000;//slti
    else if(inst[31:26] == 6'b001011)choose = 32'b0000_1000_0000_0000_0000_0000_0000_0000;//sltiu
    else if(inst[31:26] == 6'b001111)choose = 32'b0001_0000_0000_0000_0000_0000_0000_0000;//lui
    else if(inst[31:26] == 6'b000010)choose = 32'b0010_0000_0000_0000_0000_0000_0000_0000;//j
    else if(inst[31:26] == 6'b000011)choose = 32'b0100_0000_0000_0000_0000_0000_0000_0000;//jal
    else choose = 32'bz;
    end

endmodule
```

（7）operation，负责控制信号的生成

按控制信号设计表进行逻辑运算，得到当前的控制信号

```
`timescale 1ns / 1ns
module operation(
    input clk,
    input zf,
    input [31:0] i,
    output PC_CLK,
    output IM_R,
    output M1,
    output M2,
    output M3,
    output M4,
    output M5,
    output M6,
    output M7,
    output M8,
    output M9,
    output [3:0] aluc,
    output RF_W,
    output RF_CLK,
```

```
20.     output DM_cs,
21.     output DM_w,
22.     output DM_r,
23.     output ext16_sign
24.     );
25.
26.     assign PC_CLK = clk;
27.     assign IM_R = 1;
28.     //j || jr || jal
29.     assign M1 = ~(i[16] || i[29] || i[30]);
30.     //beq || bne
31.     assign M2 = (i[24] & zf) || (i[25] & ~zf);
32.     //jr
33.     assign M3 = i[16];
34.     //sllv || srlv || srav
35.     assign M4 = i[13] || i[14] || i[15];
36.     //jal
37.     assign M5 = i[30];
38.     //addi || addiu || andi || ori || xori || lw || sw || slti ||
    sliu || lui
39.     assign M6 = i[17] || i[18] || i[19] || i[20] || i[21] || i[22
   ] || i[23] || i[26] || i[27] || i[28];
40.     //lw
41.     assign M7 = i[22];
42.     // !(sll || srl || sra || sllv || srav)
43.     assign M8 = ~(i[10] || i[11] || i[12] || i[13] || i[14] || i[
   15]);
44.     //
45.     assign M9 = i[17] || i[18] || i[19] || i[20] || i[21] || i[22
   ] || i[23] || i[26] || i[27] || i[28];
46.
47.     //sub || subu || or || nor || slt || srl || srlv || ori || be
   q || bne || slti
48.     assign aluc[0] = i[2] || i[3] || i[5] || i[7] | i[8] || i[11]
    || i[14] || i[20] || i[24] || i[25] || i[26];
49.     //add || sub || xor || nor || slt || sltu || sll || sllv || a
   ddi || xori || lw || sw || beq || bne || slti || sltiu
50.     assign aluc[1] = i[0] || i[2] || i[6] || i[7] || i[8] || i[9]
    || i[10] || i[13] || i[17] || i[21] || i[22] || i[23] || i[24] |
   | i[25] || i[26] || i[27];
51.     //and || or || xor || nor || sll || srl || sra || sllv || srl
   v || srav || andi || ori || xori
```

```
    assign aluc[2] = i[4] || i[5] || i[6] || i[7] || i[10] || i[11] || i[12] || i[13] || i[14] || i[15] || i[19] || i[20] || i[21];
    //slt || sltu || sll || srl || sra || sllv || srav || slti || sltiu || lui
    assign aluc[3] = i[8] || i[9] || i[10] || i[11] || i[12] || i[13] || i[14] || i[15] || i[26] || i[27] || i[28];
    //~(jr || sw || beq || bne || j
    assign RF_W = ~(i[16] || i[23] || i[24] || i[25] || i[29]);
    assign RF_CLK = clk;
    //sw
    assign DM_cs = i[22] || i[23];
    //lw
    assign DM_w = i[23];
    //sw
    assign DM_r = i[22];

    assign ext16_sign = (i[17] || i[18] || i[22] || i[23] || i[26]);
endmodule
```

（8）pcreg，pc 寄存器

```
module pcreg(
    input clk,
    input rst,
    input [31:0] data_in,
    output reg [31:0] data_out
    );
    parameter pc_base = 32'h00400000;
    always @(negedge clk or posedge rst) begin
        if (rst)
            data_out <= pc_base;
        else
            data_out <= data_in;
    end
endmodule
```

（9）regfile，寄存器堆，包含 32 个寄存器，初值为 0

```
module regfile(
    input clk,
    input rst,
```

```
    input RF_w,
    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 = array_reg[rsc];
    assign rt = array_reg[rtc];

    always @(negedge clk or posedge rst) begin
        if (rst)
            for(i=0; i<32; i = i + 1)
                array_reg[i] <= 0;
        else if (RF_w && rdc != 0)
            array_reg[rdc] <= rd;
    end
endmodule
```

（10）alu，逻辑运算单元

```
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 [32:0] r_temp;
    reg [31:0] b0;
    always @(*) begin
    case(aluc)
        4'b0000: begin
            r_temp = a + b;
            r = r_temp[31:0];
            carry = r_temp[32];
            if(r == 0)
```

```
                zero = 1;
            else zero = 0;
            //negative
            if(r[31] == 1)
                negative = 1;
            else
                negative = 0;
        end

    4'b0010: begin
        r=a+b;
        if(~a[31] & ~b[31] & r[31] || a[31] & b[31] & ~r[31])
            overflow = 1;
        else
            overflow = 0;
        if(r == 0)
            zero = 1;
        else
            zero = 0;
        if(r[31] == 1)
            negative = 1;
        else
            negative = 0;
    end

    4'b0001: begin
        r = a - b;
        if(a >= b)
            carry = 0;
        else
            carry = 1;
        if(r == 0)
            zero = 1;
        else
            zero = 0;
        if(r[31] == 1)
            negative = 1;
        else
            negative = 0;
    end

    4'b0011: begin
        b0 = ~b + 1;
        r = a + b0;
```

```
64.     if(b[31] != 1'b1)
65.         if(~a[31] & ~b0[31] & r[31] || a[31] & b0[31] & ~r[31])
66.             overflow = 1;
67.         else
68.             overflow = 0;
69.     else
70.         if(a[31] == 0)
71.             overflow = 1;
72.         else
73.             overflow = 0;
74.         if(r == 0)
75.             zero = 1;
76.         else
77.             zero = 0;
78.         if(r[31] == 1)
79.             negative = 1;
80.         else
81.             negative = 0;
82.     end
83.
84.     4'b0100: begin
85.         r = a & b;
86.         if(r == 0)
87.             zero = 1;
88.         else
89.             zero = 0;
90.         if(r[31] == 1)
91.             negative = 1;
92.         else
93.             negative = 0;
94.     end
95.
96.     4'b0101: begin
97.         r = a | b;
98.         if(r == 0)
99.             zero = 1;
100.         else
101.             zero = 0;
102.         if(r[31] == 1)
103.             negative = 1;
104.         else
105.             negative = 0;
106.     end
107.
```

```
    4'b0110:  begin
        r = a ^ b;
        if(r == 0)
            zero = 1;
        else
            zero = 0;
        if(r[31] == 1)
            negative = 1;
        else
            negative = 0;
    end

    4'b0111:  begin
        r = ~(a | b);
        if(r == 0)
            zero = 1;
        else
            zero = 0;
        if(r[31] == 1)
            negative = 1;
        else
            negative = 0;
    end

    4'b1000: begin
        r = {b[15:0], 16'b0};
        if(r == 0)
            zero = 1;
        else
            zero = 0 ;
        if(r[31] == 1)
            negative = 1;
        else
            negative = 0;
    end

    4'b1001: begin
        r = {b[15:0], 16'b0};
        if(r == 0)
            zero = 1;
        else
            zero = 0;
        if(r[31] == 1)
            negative = 1;
```

```
152.         else
153.             negative = 0;
154.     end
155.
156.     4'b1011: begin
157.         if(a[31] == 0 && b[31] == 1)
158.             r = 0;
159.         else if(a[31] == 1 && b[31] == 0)
160.             r = 1;
161.         else if(a[31] == 1 && b[31] == 1)
162.             r = (a < b) ? 1 : 0;
163.         else
164.             r=(a < b) ? 1 : 0;
165.             if(a == b)
166.                 zero = 1;
167.             else
168.                 zero = 0;
169.             if(r == 1)
170.                 negative=1;
171.             else
172.                 negative=0;
173.     end
174.
175.     4'b1010: begin
176.         if(a < b) begin
177.             r = 1;
178.             carry = 1;
179.         end
180.         else begin
181.             r = 0;
182.             carry = 0;
183.         end
184.         if(a == b)begin
185.             r = 0;
186.             zero = 1;
187.         end
188.         else
189.             zero = 0;
190.         if(r[31] == 1)
191.             negative = 1;
192.         else
193.             negative = 0;
194.     end
195.
```

```
196.     4'b1100: begin
197.         r = ($signed(b)) >>> a;
198.         if(a != 0)
199.             carry = b[a-1];
200.         else
201.             carry = 0;
202.         if(r == 0)
203.             zero = 1;
204.         else
205.             zero = 0;
206.         if(r[31] == 1)
207.             negative = 1;
208.         else
209.             negative = 0;
210.     end
211.
212.     4'b1111: begin
213.         r = b << a;
214.         if(a != 0)
215.             carry = b[32-a];
216.         else
217.             carry = 0;
218.         if(r == 0)
219.             zero = 1;
220.         else
221.             zero = 0;
222.         if(r[31] == 1)
223.             negative = 1;
224.         else
225.             negative = 0;
226.     end
227.
228.     4'b1110: begin
229.         r = b << a;
230.         if(a != 0)
231.             carry = b[32-a];
232.         else
233.             carry = 0;
234.         if(r == 0)
235.             zero = 1;
236.         else
237.             zero = 0;
238.         if(r[31] == 1)
239.             negative = 1;
```

```
        else
            negative = 0;
    end

    4'b1101: begin
        r = b >> a;
        if(a != 0)
            carry = b[a-1];
        else
            carry = 0;
        if(r == 0)
            zero = 1;
        else
            zero = 0;
        if(r[31] == 1)
            negative = 1;
        else
            negative = 0;
    end
    endcase
end
endmodule
```

（11）mux，输入数据为 32 位的双路选择器，0 选 a，1 选 b

```
`timescale 1ns / 1ns
module mux(
    input [31:0] a,
    input [31:0] b,
    input choose,
    output reg [31:0] z
    );
    always @(*)
    begin
    case(choose)
        1'b1:z <= b;
        1'b0:z <= a;
   endcase
   end
endmodule
```

（12）mux5，输入数据为 5 位的双路选择器

```
`timescale 1ns / 1ns
module mux5(
    input [4:0] a,
    input [4:0] b,
    input [1:0] choose,
    output [4:0] z
    );
    reg [4:0] temp_z;
    always @(*)
    begin
    case(choose)
        2'b00:temp_z <= a;
        2'b01:temp_z <= b;
        2'b10:temp_z <= 5'b11111;
        2'b11:temp_z <= 5'b11111;
        default:temp_z <= 5'bz;
    endcase
    end
    assign z = temp_z;
endmodule
```

（13）extend5，将 5 位的数据扩展为 32 位

```
`timescale 1ns / 1ns
module extend5(
    input [4:0] a,
    output [31:0] b
    );
    assign b = {27'b0, a};
endmodule
```

（14）extend16，将 16 位的数据扩展为 32 位

```
`timescale 1ns / 1ns
module extend16(
    input [15:0] a,
    input sign,
    output [31:0] b
    );
    assign b = sign ? {{16{a[15]}}, a} : {16'b0, a};
endmodule
```

（15）extend18，将 18 位的数据扩展为 32 位

```
`timescale 1ns / 1ns
module extend18 (
    input [15:0] a,
    output [31:0] b
    );
    assign b = {{14{a[15]}}, a, 2'b00};
endmodule
```

（16）add，加法器

```
`timescale 1ns / 1ns
module add(
    input [31:0] a,
    input [31:0] b,
    output reg [31:0] r
    );
    always@(*)begin
        r = a + b;
    end
endmodule
```

（17）add8

```
`timescale 1ns / 1ns
module add8(
    input [31:0] a,
    output [31:0] r
    );
    assign r=a+4;
endmodule
```

（18）II，连接一个 4 位和 26 位的数据

```
`timescale 1ns / 1ps

module II(
    input [3:0] a,
    input [25:0] b,
    output [31:0] data_out
```

```
    );
    assign data_out = {a, b, 2'b00};
Endmodule
```

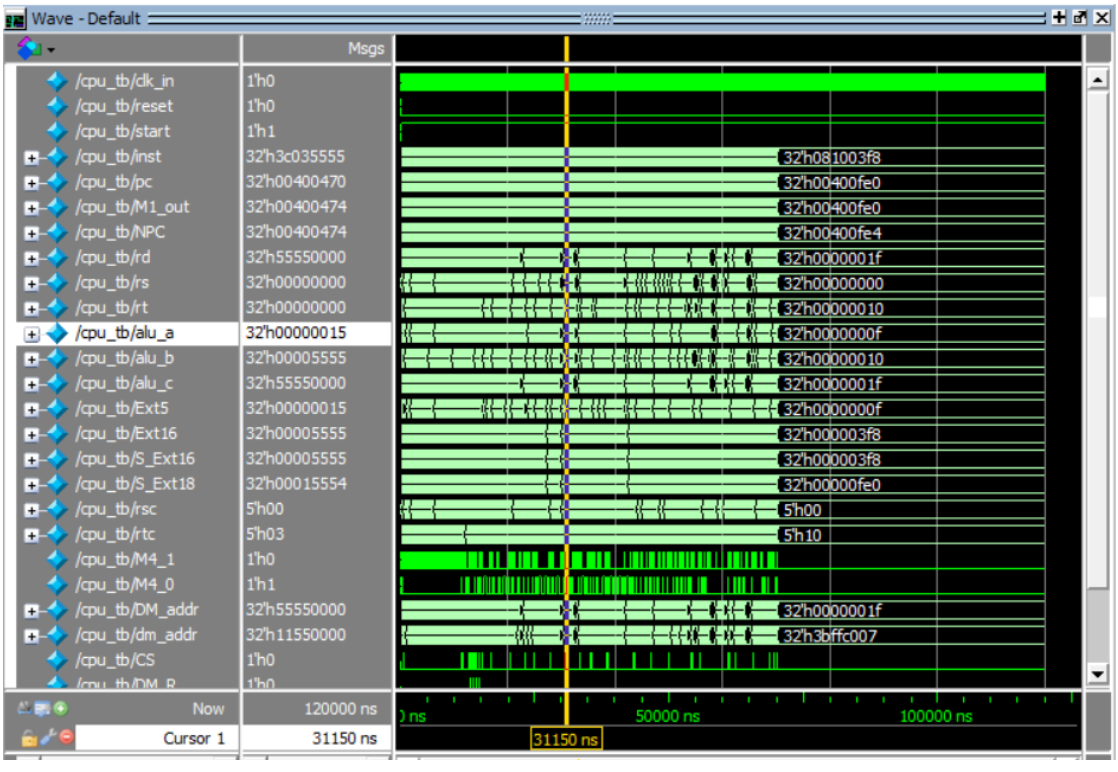
五、 测试模块

```
`timescale 1ps/1ps
module cpu_tb();
    reg clk_in;
    reg reset;
    reg start;
    wire[31:0] inst;
    wire[31:0] pc;

    sccomp_dataflow my_cpu(clk_in, reset, inst, pc);

    integer file_output;
    integer counter = 0;

    initial begin
        file_output = $fopen("E:/Personal/Desktop/result_xlf.txt");
    end

    initial begin
        clk_in = 0;
        start  = 0;
        forever begin
            //#1 clk_in = ~clk_in;
            #50 clk_in = ~clk_in;
        end
    end

    initial begin
        reset     = 1;
        // #6 reset  = 1;
        // #50 reset = 0;
        #225 reset = 0;
        start = 1;
    end

    always @(posedge clk_in) begin
        if (start)begin
            counter                             = counter + 1;
```

```
38.              counter = counter + 1;
39.
40.             $fdisplay(file_output, "regfile0: %h", my_cpu.sccpu.c
   pu_ref.array_reg[0]);
41.             $fdisplay(file_output, "regfile1: %h", my_cpu.sccpu.c
   pu_ref.array_reg[1]);
42.             $fdisplay(file_output, "regfile2: %h", my_cpu.sccpu.c
   pu_ref.array_reg[2]);
43.             $fdisplay(file_output, "regfile3: %h", my_cpu.sccpu.c
   pu_ref.array_reg[3]);
44.             $fdisplay(file_output, "regfile4: %h", my_cpu.sccpu.c
   pu_ref.array_reg[4]);
45.             $fdisplay(file_output, "regfile5: %h", my_cpu.sccpu.c
   pu_ref.array_reg[5]);
46.             $fdisplay(file_output, "regfile6: %h", my_cpu.sccpu.c
   pu_ref.array_reg[6]);
47.             $fdisplay(file_output, "regfile7: %h", my_cpu.sccpu.c
   pu_ref.array_reg[7]);
48.             $fdisplay(file_output, "regfile8: %h", my_cpu.sccpu.c
   pu_ref.array_reg[8]);
49.             $fdisplay(file_output, "regfile9: %h", my_cpu.sccpu.c
   pu_ref.array_reg[9]);
50.             $fdisplay(file_output, "regfile10: %h", my_cpu.sccpu.
   cpu_ref.array_reg[10]);
51.             $fdisplay(file_output, "regfile11: %h", my_cpu.sccpu.
   cpu_ref.array_reg[11]);
52.             $fdisplay(file_output, "regfile12: %h", my_cpu.sccpu.
   cpu_ref.array_reg[12]);
53.             $fdisplay(file_output, "regfile13: %h", my_cpu.sccpu.
   cpu_ref.array_reg[13]);
54.             $fdisplay(file_output, "regfile14: %h", my_cpu.sccpu.
   cpu_ref.array_reg[14]);
55.             $fdisplay(file_output, "regfile15: %h", my_cpu.sccpu.
   cpu_ref.array_reg[15]);
56.             $fdisplay(file_output, "regfile16: %h", my_cpu.sccpu.
   cpu_ref.array_reg[16]);
57.             $fdisplay(file_output, "regfile17: %h", my_cpu.sccpu.
   cpu_ref.array_reg[17]);
58.             $fdisplay(file_output, "regfile18: %h", my_cpu.sccpu.
   cpu_ref.array_reg[18]);
59.             $fdisplay(file_output, "regfile19: %h", my_cpu.sccpu.
   cpu_ref.array_reg[19]);
60.             $fdisplay(file_output, "regfile20: %h", my_cpu.sccpu.
   cpu_ref.array_reg[20]);
```

```
61.             $fdisplay(file_output, "regfile21: %h", my_cpu.sccpu.
    cpu_ref.array_reg[21]);
62.             $fdisplay(file_output, "regfile22: %h", my_cpu.sccpu.
    cpu_ref.array_reg[22]);
63.             $fdisplay(file_output, "regfile23: %h", my_cpu.sccpu.
    cpu_ref.array_reg[23]);
64.             $fdisplay(file_output, "regfile24: %h", my_cpu.sccpu.
    cpu_ref.array_reg[24]);
65.             $fdisplay(file_output, "regfile25: %h", my_cpu.sccpu.
    cpu_ref.array_reg[25]);
66.             $fdisplay(file_output, "regfile26: %h", my_cpu.sccpu.
    cpu_ref.array_reg[26]);
67.             $fdisplay(file_output, "regfile27: %h", my_cpu.sccpu.
    cpu_ref.array_reg[27]);
68.             $fdisplay(file_output, "regfile28: %h", my_cpu.sccpu.
    cpu_ref.array_reg[28]);
69.             $fdisplay(file_output, "regfile29: %h", my_cpu.sccpu.
    cpu_ref.array_reg[29]);
70.             $fdisplay(file_output, "regfile30: %h", my_cpu.sccpu.
    cpu_ref.array_reg[30]);
71.             $fdisplay(file_output, "regfile31: %h", my_cpu.sccpu.
    cpu_ref.array_reg[31]);
72.             $fdisplay(file_output, "pc: %h", pc);
73.             $fdisplay(file_output, "instr: %h", inst);
74.         end
75.     end
76.endmodule
```

六、 测试波形图



七、 总结与体会

刚开始做的时候，还是什么都不懂，通过阅读 31 条指令的详细文档说明，逐渐对 CPU 的运行原理有了一个大体上的感觉。

然后开始花单条指令的数据通路图，在画图的过程中，文档里的文字说明变成了更具象化的数据通路，这更加深了我对每条指令执行步骤与要求的认识。

进入准备阶段最重要的几步，开始根据我对每条指令的理解与总结出的需要的部件，来统计得出两张表格和最终完整的数据通路图。在综合数据的过程中，之前的一些错误的认识互相矛盾，让我发现了对描述要求的理解偏差，及时纠正了过来，比如 beq 指令的具体实现是靠 alu 输出的 zf 标志位来生成控制符，而之前在画数据通路的时候却没能理解到这一层。

之后就是编写代码的过程。由于需要的很多部件都是上学期数字逻辑的平时作业，所以基本上只需要编写控制模块，将这些部件按照输入输出关系连接起来就好了，并不是很费时间。

但是由于编写过程的不细心，很多部件的输入输出关系连错，导致 debug 的时候很痛苦，只能慢慢捋思路，一步一步查。

同时，在一条条测试指令的过程中，一些之前不太确定的地方也得到了修正，如符号位扩展的问题。