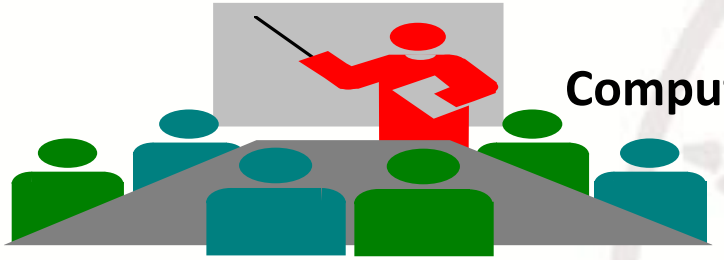


Computer Organization & Design
实验与课程设计

实验六

CPU设计-控制器

施青松
Asso. Prof. Shi Qingsong
College of Computer Science and Technology, Zhejiang University
zjsqs@zju.edu.cn

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

实验目的

1. 运用寄存器传输控制技术
2. 掌握CPU的核心：指令执行过程与控制流关系
3. 设计控制器
4. 学习测试方案的设计
5. 学习测试程序的设计

实验环境

实验设备

1. 计算机（Intel Core i5以上，4GB内存以上）系统

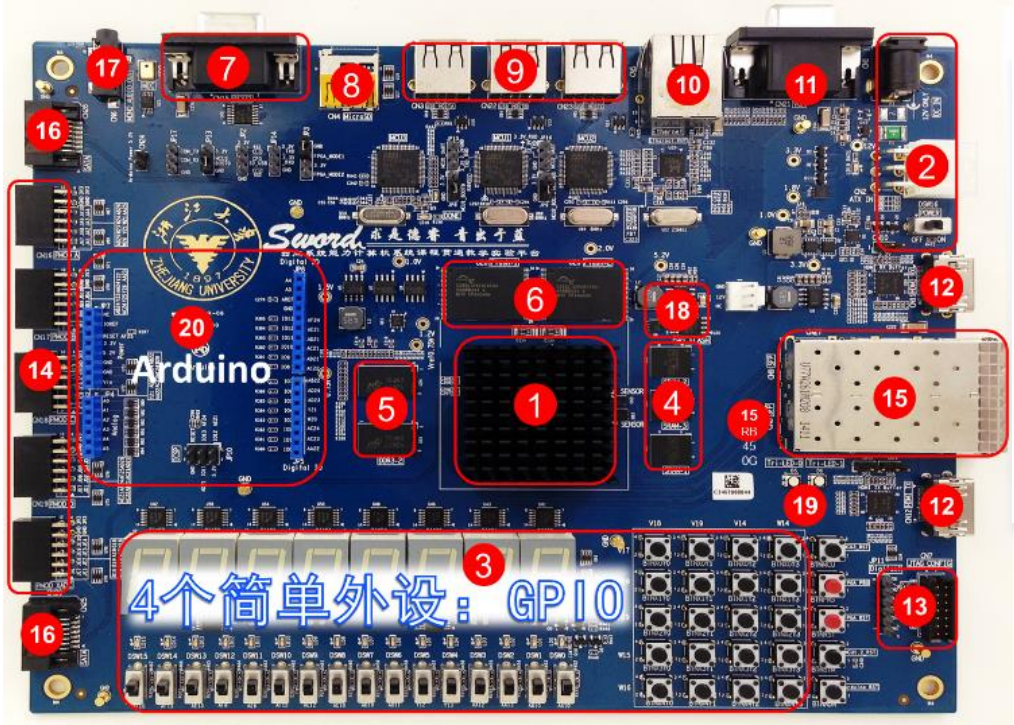
2.计算机软硬件课程贯通教学实验系统(Sword)

3. Xilinx ISE14.4及以上开发工具

材料

无

计算机软硬件课程贯通教学实验系统



贯通教学实验平台主要参数

▼ **核心芯片**

Xilinx Kintex™-7系列的XC7K160/325资源：

162,240个，Slice：25350，片内存储：11.7Mb

▼ **存储体系** 支持32位存储层次体系结构

6MB SRAM静态存储器：支持32Data，16位TAG

512M BDDR3动态存储：支持32Data

32MB NOR Flash存储 ：支持32位Data

▼ **基本接口** 支持微机原理、SOC或微处理器简单应用

4×5+1矩阵按键、16位滑动开关、16位LED、8位七段数码管

▼ **标准接口** 支持基本计算机系统实现

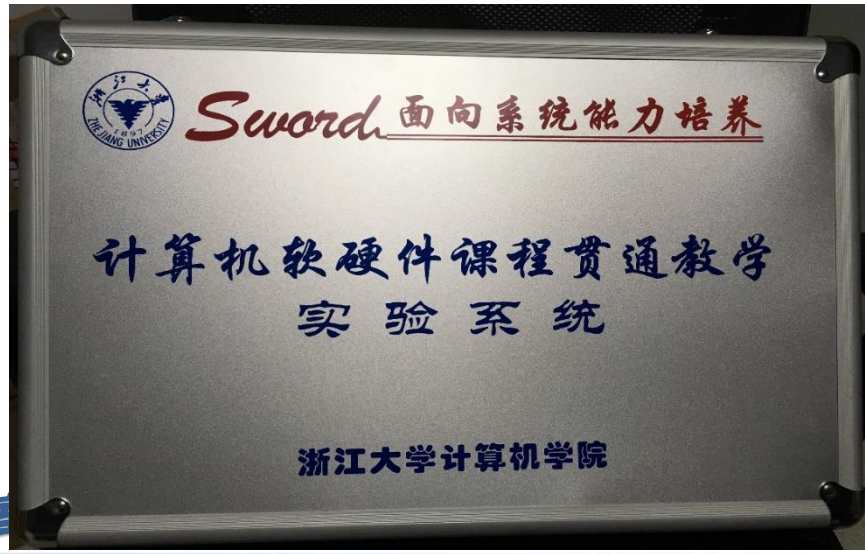
12位VGA接口（RGB656）、USB-HID（键盘）

▼ **通讯接口** 支持数据传输、调试和网络

UART接口、 10M/100M/1000M以太网、 SFP光纤接口

▼ **扩展接口** 支持外存、多媒体和个性化设备

MicroSD(TF）、PMOD、 HDMI、Arduino



Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

实验任务

1. 设计9+条指令的控制器

- 用硬件描述语言设计实现控制器
  - 根据Exp05数据通路及指令编码完成控制信号真值表
- 替换Exp05的控制器核
- 此实验在Exp05的基础上完成

2. 设计控制器测试方案：

- OP译码测试：R-格式、访存指令、分支指令，转移指令
- 运算控制测试：Function译码测试

3. 设计控制器测试程序

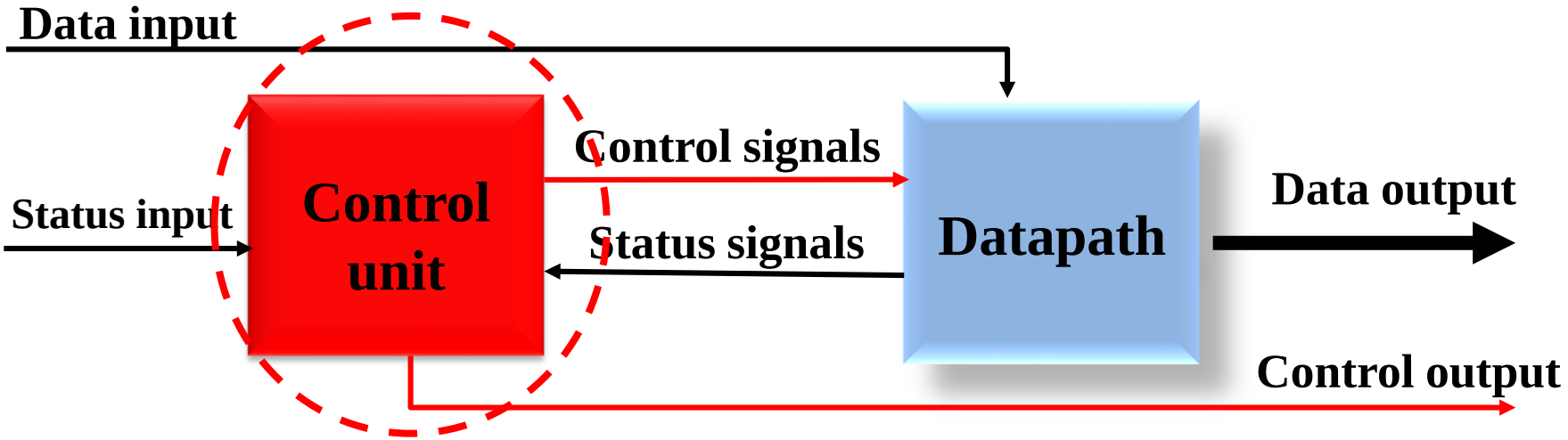
Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

CPU organization

Digital circuit

- General circuits that controls logical event with logical gates --Hardware

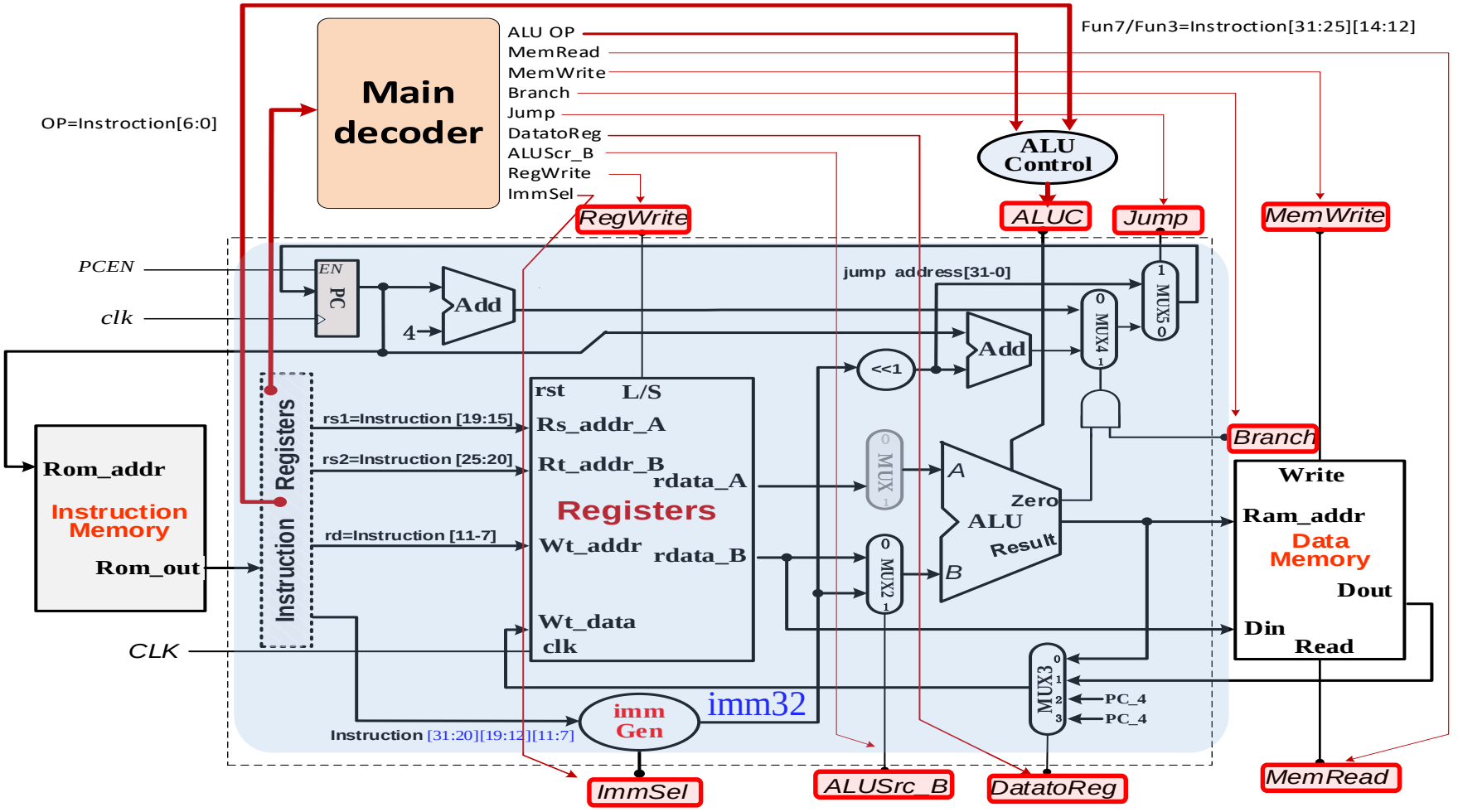


Computer organization

- Special circuits that processes logical action with instructions -Software

控制对象：数据通路结构

- 写出九条指令控制信号真值表



控制信号定义

- 通路与操作控制

信号	源数目	功能定义	赋值0时动作	赋值1时动作
ImmSel	4	立即数选择	=00：imm32 =01：imm32=ims32	=10：imm32=imsb32 =11：imm32=imuj32
ALUSrc_B	2	ALU端口B输入选择	选择寄存器B数据	选择32位立即数（符号扩展后）
DatatoReg	3	寄存器写入数据选择	选择存储器、ALU或PC+4数据	
Branch	2	Beq指令目标地址选择	选择PC+4目标地址	选择转移地址（Zero=1）
Jump	2	J指令目标地址选择	选择Jal目标地址	由Branch决定输出
RegWrite	-	寄存器写控制	禁止寄存器写	使能寄存器写
MemWrite	-	存储器写控制	禁止存储器写	使能存储器写
MemRead	-	存储器读控制	禁止存储器读	使能存储器读
ALUC	000-111	3位ALU操作控制	参考表Exp04	Exp04

主控制器信号真值表

- 分析填写控制器输出信号真值表

Instruction	format	Imm Sel	ALU Src_B	Data toReg	Reg Write	Mem Read	Mem Write	Branch	Jump	ALU op1	ALU op0
ALU	**R**										
lw	**I**										
sw	**S**										
beq	**SB**										
jal	**UJ**										

ALU操作译码

Second level

- 参考实验四　　Func7= Inst[31:25]　Funct3=Inst[14:12]

opcode	ALUop		Funct7			Funct3			ALUC 210
	ALU_{op1}	ALU_{op0}	$F7_6$	$F7_5$	$F7_{4-0}$	$F3_2$	$F3_1$	$F3_0$	
L/S	0	0	x	x	xxxxx	x	x	x	010
beq	x	1	x	x	xxxxx	x	x	x	110
R-type	1	x	x	0	xxxxx	0	0	0	010
			x	1	xxxxx	0	0	0	110
			x	0	xxxxx	1	1	1	000
			x	0	xxxxx	1	1	0	001
			x	0	xxxxx	0	1	0	111

x：don't care

$$ALUC_2 = ALUop_0 + ALU_{op1}(F7_5\,\overline{F3_2}\,\overline{F3_1}\,\overline{F3_0} + \overline{F7_5}\,\overline{F3_2}F3_1\overline{F3_0})$$

$$ALUC_1 = \overline{ALU_{op1}(\overline{F7_5}\,F32F3_1F3_0 + \overline{F7_5}\,F32F3_1\overline{F3_0})}$$

$$ALUC_0 = ALU_{op1}(\overline{F7_5}\,F32F3_1\overline{F3_0} + \overline{F7_5}\,\overline{F3_2}F3_1\overline{F3_0})$$

ALU操作译码化简

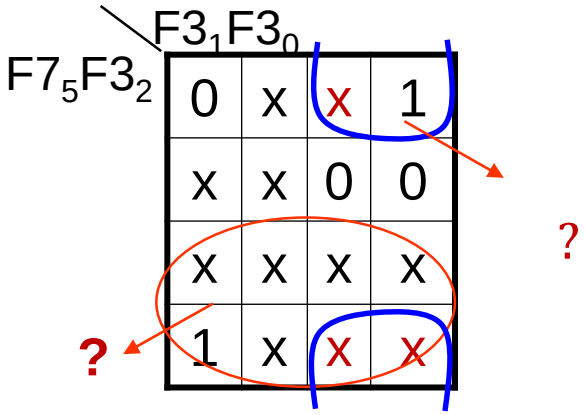
- 对Funct变量化简
  - ALUop 单独考虑
    - 化简后合并

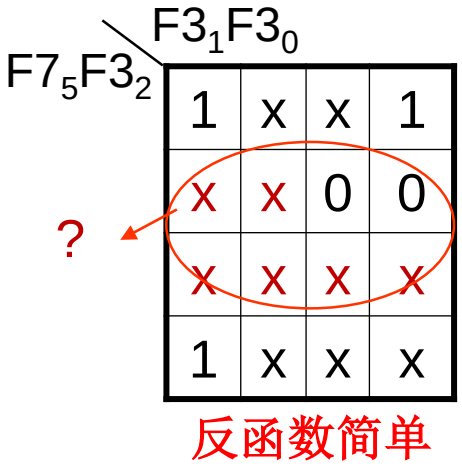
$ALUC_0 = ALU_{op1}F3_1\overline{F3_0}$

$ALUC_1 = \overline{ALU_{op1}F32} = \overline{ALU_{op1}} + \overline{F32}$

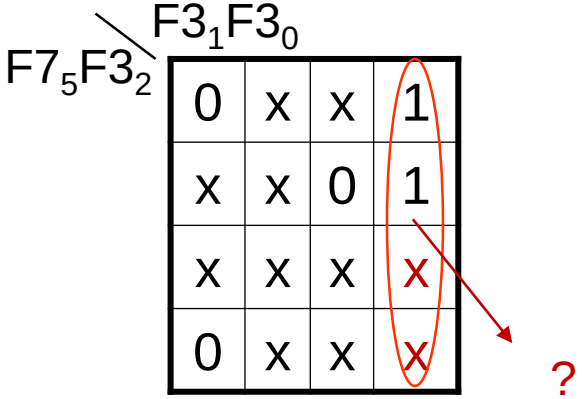
$ALUC_2 = ALUop_0 + ALU_{op1}F7_5 + ALU_{op1}\overline{F3_2}F3_1\overline{F3_0}$

???



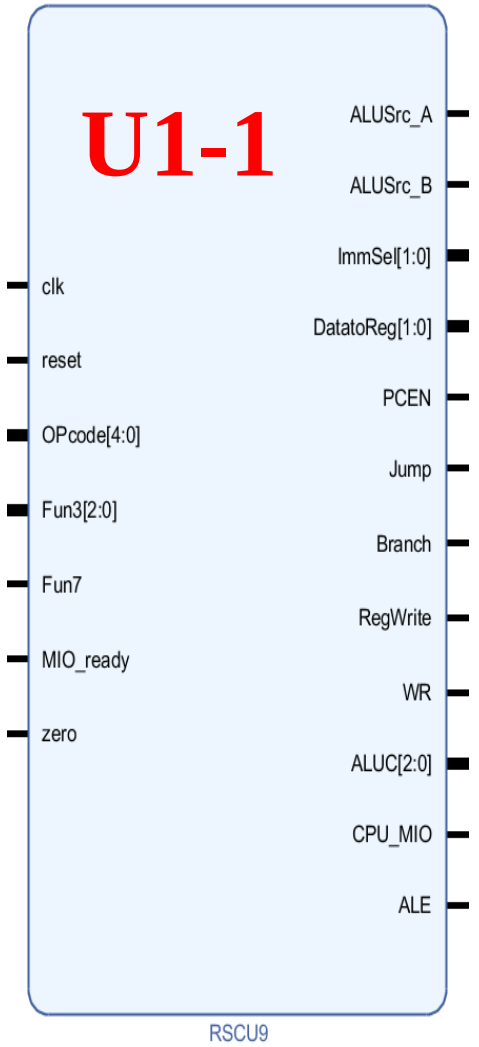


反函数简单



CPU部件之控制器接口： RSCU9

- **Controller**
  - CPU主要部件之一：编码转换成命令
  - 寄存器传输控制单元：
    控制运算和通路控制器
- **基本功能**
  - 产生微操作控制信号：ALU运算控制
  - 数据传输通道控制：选择所需的路径
  - 时序控制：单周期时序在那里？
- **本实验设计RSCU9**
  - 采用HDL结构化描述设计
  - 替换实验四RSCU9.edf
  - 实现最简自主CPU



控制器端口信号标准：RSCU9.v

```
module    RSCU9(input clk,
                input reset,
                input[4:0]OPcode,                    //OPcode: inst[6:2]
                input[2:0]Fun3,                      //Function: inst[14:12]
                input     Fun7,                      //Function: inst[30]
                input MIO_ready,                     //CPU Wait, 复杂时序交互(保留)
                input zero,                          //ALU result = 0

                output reg ALUSrc_A,                 //ALU源操作数1选择
                output reg ALUSrc_B,                 //ALU源操作数2选择
                output reg [1:0]ImmSel,              //立即数选择控制
                output reg [1:0]DatatoReg,           //写回控制选择
                output PCEN,
                output reg Jump,                     //UJ跳转控制
                output reg Branch,                   //SB分支跳转控制
                output reg RegWrite,                 //寄存器堆写使能
                output reg WR,                       //存储器读写使能
                output reg [2:0]ALUC,                //ALU控制
                output reg CPU_MIO,                  //存储器操作信号
                output ALE                           //存储器访问有效
               );

endmodule
```

Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理
- 实验操作与实现

CPU之控制器设计

–控制实验五设计的数据通路

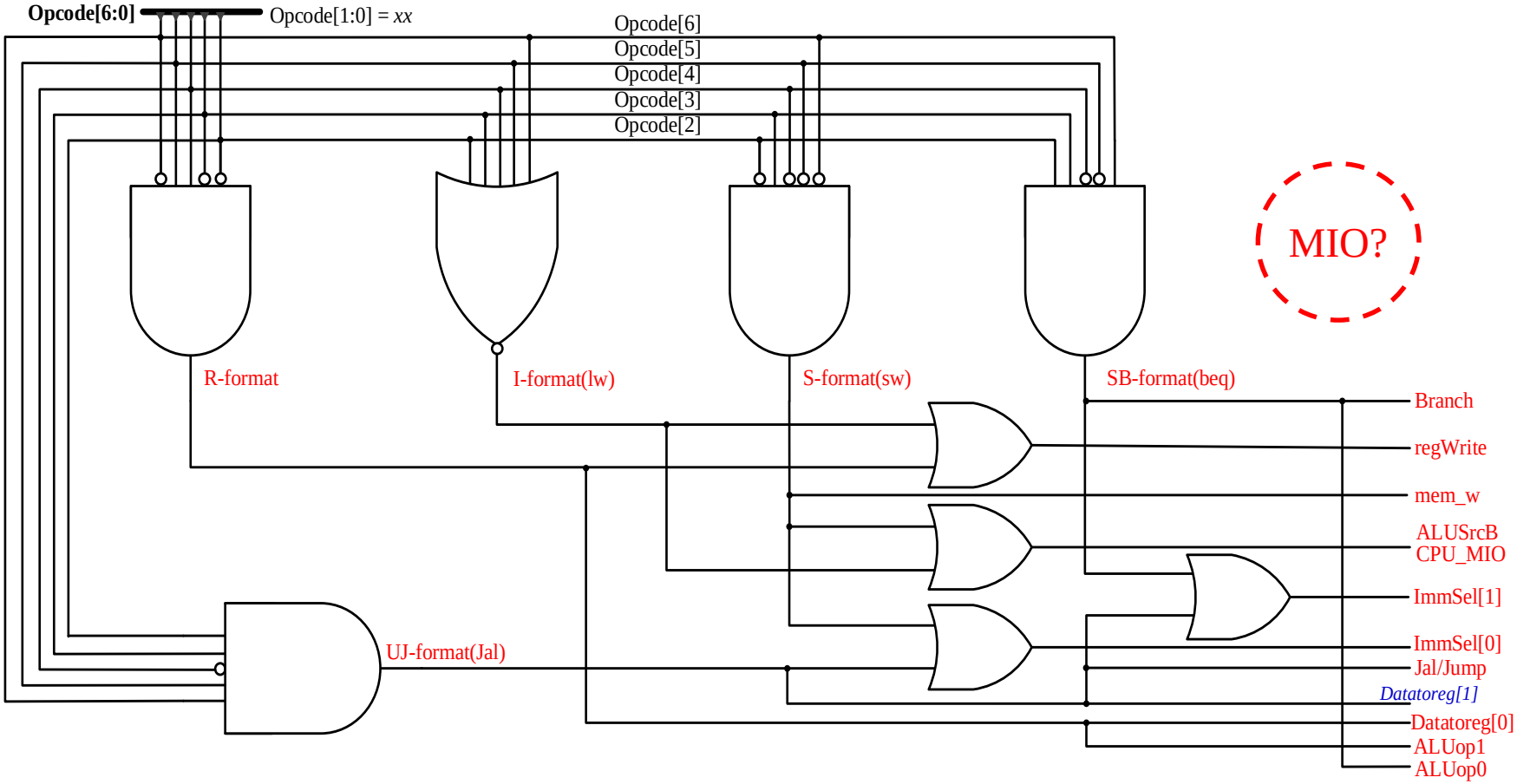
设计工程：OExp06-CSTEH-RSCPU9

- **设计CPU之控制器**
  - 根据理论课分析讨论设计实验五数据通路的的控制器
  - HDL或原理图描述均可
    - 但必须用函数表达式结构描述
  - 仿真测试控制器模块
- **集成替换验证通过的数据通路模块**
  - 替换实验五(Exp05)中的RSCU9.edf核
  - 顶层模块延用OExp05
    - 模块名： OwnRSCPU
- **测试控制器模块**
  - 设计测试程序(RISC-V汇编)测试：
  - OP译码测试：R-格式、访存指令、分支指令，转移指令
  - 运算控制测试：Function译码测试

设计要点

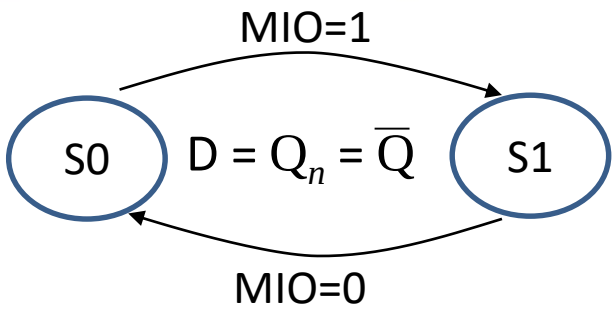
- **设计主控制器模块**
  - 写出控制器的函数表达式
  - 结构描述实现电路
- **设计ALU操作译码**
  - 写出ALU操作译码函数表达式
  - 结构描述实现电路
  - 使用DEMO作功能初步调试
- **仿真二个控制器电路模块**
  - 可以单独或合并仿真，但最后要合并为一个控制模块

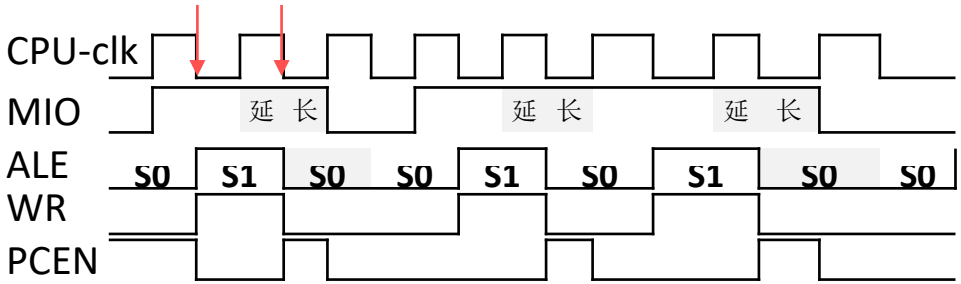
指令译码-主控制器逻辑电路



指令译码-地址锁存信号

现态	输入	次态	输出		
Q	MIO	Qn	ALE	WR	PCEN
0	0	1	0	0	1
1	1	0	1	mem_w	0

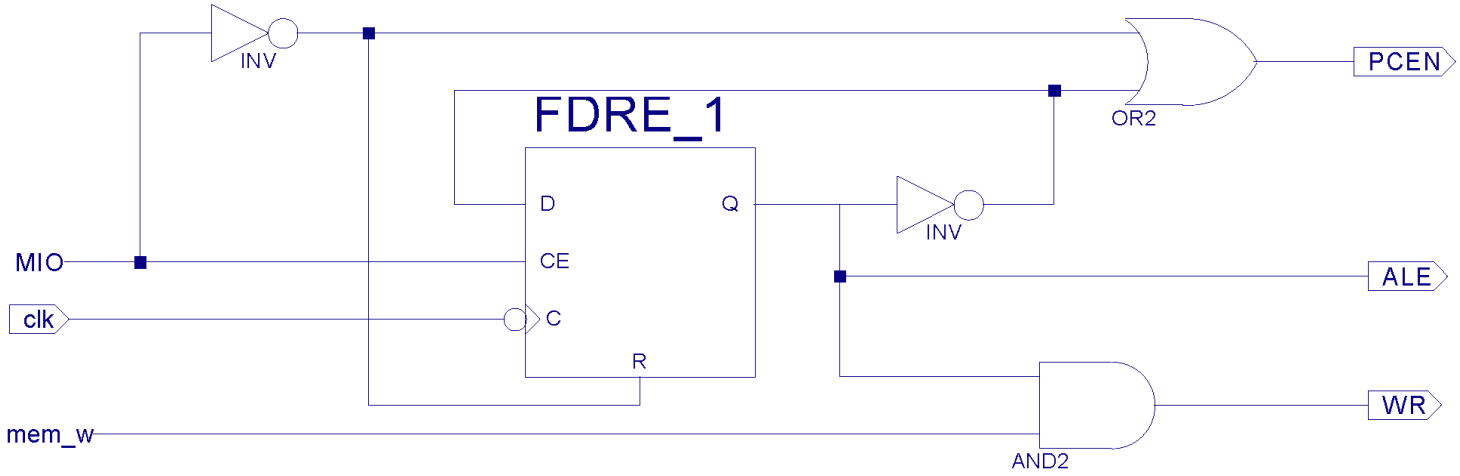




ALE = Q

WR = Q mem_w

PCEN= $\overline{Q}$ + $\overline{MIO}$



主控制器HDL描述结构

指令译码器参考描述

```
`define CPU_ctrl_signals
        {ImmSel, ALUSrc_B, DatatoReg, RegWrite, MemRead, MemWrite, Branch, Jump, ALUop}
        assign WR = MemWrite&&(~MemRead);
        always @* begin
                case(OPcode)
                 5‘b01100: begin CPU_ctrl_signals = ? ;  end     //ALU
                 5’b00000: begin CPU_ctrl_signals = ? ;  end     //load
                 5’b01000: begin CPU_ctrl_signals = ? ;  end     //store
                 5’b11000: begin CPU_ctrl_signals = ? ;  end     //beq
                 5’b11011: begin CPU_ctrl_signals = ? ;  end     //jump
                 5’b00100: begin CPU_ctrl_signals = ? ;  end     //addi，增加ALUop编码
                                        ……
                default:     begin  CPU_ctrl_signals = ? ; end
                endcase
end
```

某些指令仅需部分控制信号，无关信号可赋定值，避免组合逻辑赋值不完整而生成latch

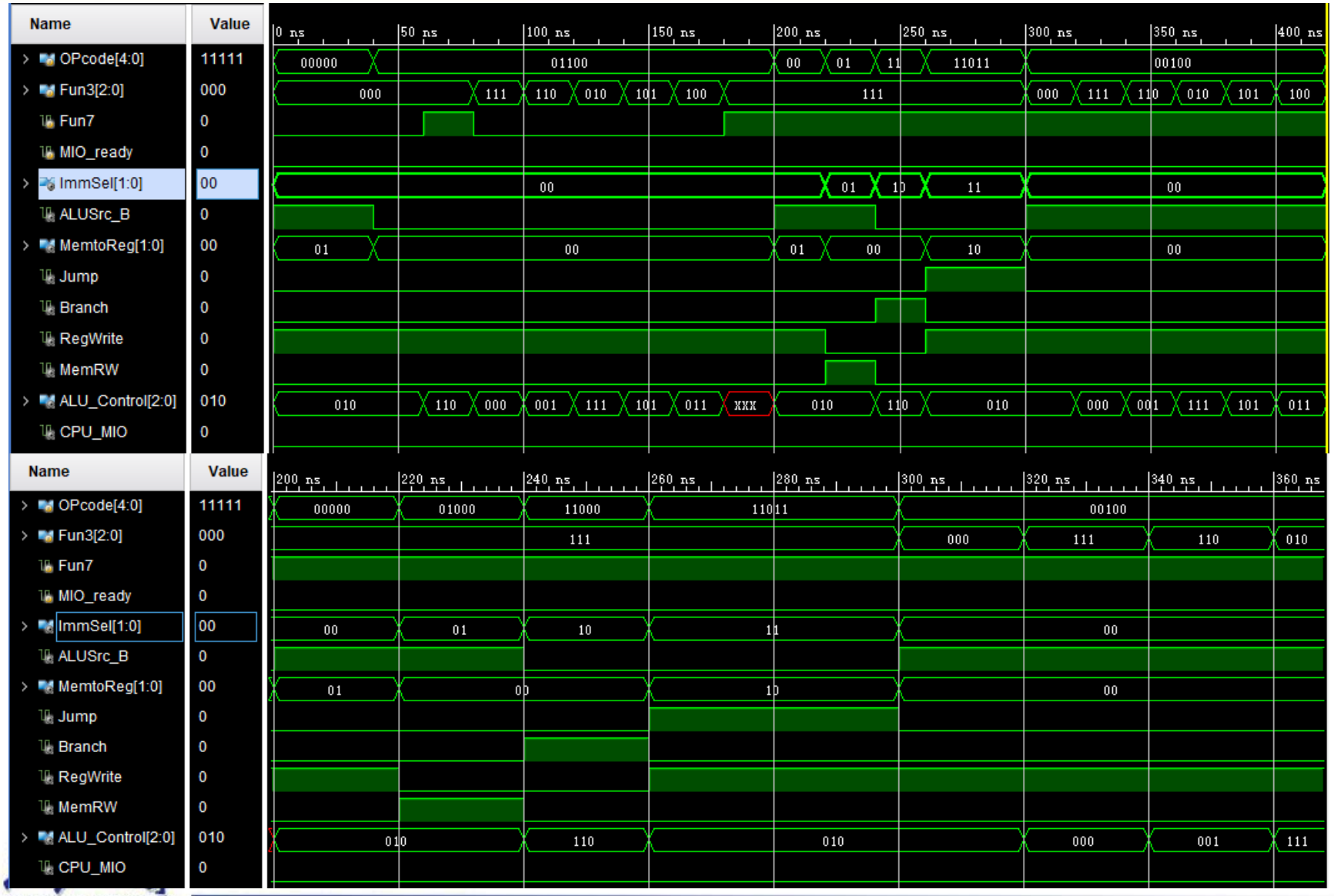
控制器仿真激励代码参考

```
initial begin
        // Initialize Inputs
        OPcode = 0;
        Fun3 = 0; Fun7 = 0;
        MIO_ready = 0;
        #40;
        // Wait 40 ns for global reset to finish。以上是测试模板代码。
        // Add stimulus here
        //检查输出信号和关键信号输出是否满足真值表
        OPcode = 5'b01100; //ALU指令，检查ALUop=2'b10; RegDst=1; RegWrite=1
                Fun3 = 3'b000; Fun7 = 1'b0;//add,检查ALU_Control=3'b010
                #20;
                Fun3 = 3'b000; Fun7 = 1'b1;//sub,检查ALU_Control=3'b110
                #20;
                Fun3 = 3'b111; Fun7 = 1'b0;//and,检查ALU_Control=3'b000
                #20;
                Fun3 = 3'b110; Fun7 = 1'b0;//or,检查ALU_Control=3'b001
                #20;
                Fun3 = 3'b010; Fun7 = 1'b0 ;//slt,检查ALU_Control=3'b111
                #20;
```

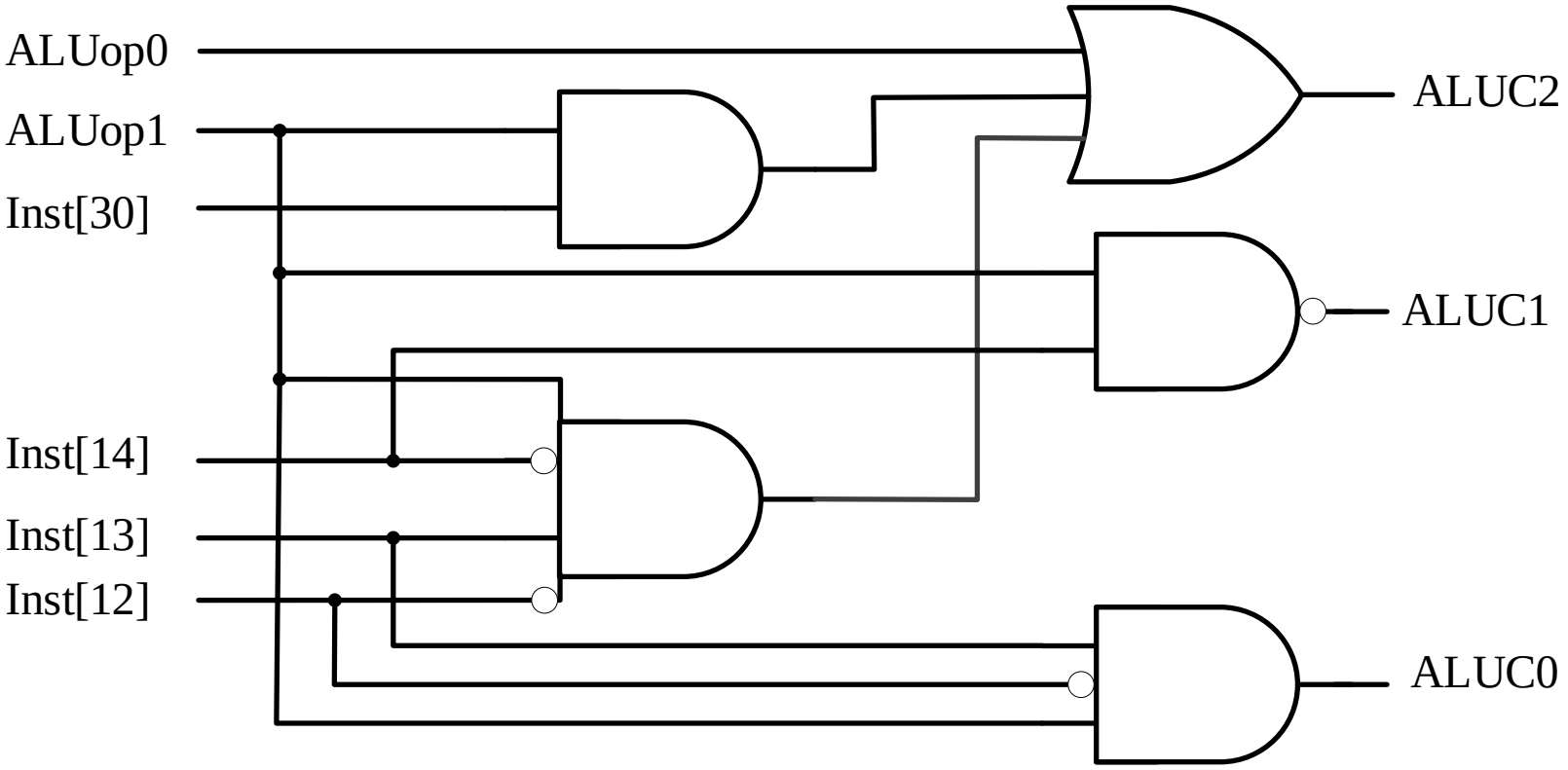
控制器仿真激励代码参考

```
#20;
Fun3 = 3'b101; Fun7 = 1'b0//srl,检查ALU_Control=3'b101
#20;
Fun3 = 3'b100; Fun7 = 1'b0//xor,检查ALU_Control=3'b011
#20;
Fun3 = 3'b111; Fun7 = 1'b1;                    //间隔
#1;
OPcode = 5'b00000;//load指令，检查ALUop=2'b01,
#20;                    // ALUSrc_B=1, MemtoReg=1, RegWrite=1
OPcode = 5'b01000;
#20; //store指令，检查ALUop=2'b10, MemRW=1, ALUSrc_B=1
OPcode = 5'b11000;//beq指令，检查ALUop=2'b11, Branch=1
#20;
OPcode = 5'b11011;//jump指令，检查Jump=1
OPcode = 5'b00100; //I指令，检查ALUop=2'b01; RegWrite=1
#20;
Fun3 = 3'b000;       //addi,检查ALU_Control=3'b010
.....
#20;
OPcode = 5'h1f;                                 //间隔
 Fun3 = 3'b000; Fun7 = 1'b0;                    //间隔
end
```

控制器模块时序仿真结果



ALU操作译码器逻辑电路



ALU操作译码器HDL描述结构

ALU控制器参考描述

```
assign Fun = {Fun3,Fun7};
always @* begin
    case(ALUop)
    2'b00: ALU_Control = ? ;                          //add计算地址
    2'b01: ALU_Control = ? ;                          //sub比较条件
    2'b10: case(Fun)
              4'b0000:  ALUC = 3'b010 ;               //add
              4’b0001: ALUC =   ?   ;                 //sub
              4’b1110: ALUC =   ?   ;                 //and
              4’b1100: ALUC =   ?   ;                 //or
              4’b0100: ALUC =   ?   ;                 //slt
              4’b1010: ALUC =   ?   ;                 //srl
              4’b1000: ALUC =   ?  ;                  //xor
                    ……
              default:   ALU_Control=3'bx;
           endcase
    2'b11: case(Fun3)
              3'b000: ALUC =3'b010;                   //addi
                    ……
    endcase
```

设计要点：控制器集成替换前

- 集成替换
  - 仿真正确后替换Exp05的控制器IP核
- **移除工程中的控制器核**
  - Exp05工程中移控制器核关联
  - 删除工程中控制器核文件
    - RSCU9.edf和RSCU9.v 文件
    - 重新添加关联自己设计的RSDP9 .v
  - 建议用OExp04资源重建工程
    - 除RSDP9.edf核
  - 添加Controller核后，参照OExp04完成CPU核集成

OExp05需要清理的核

其余不变与实验五相同

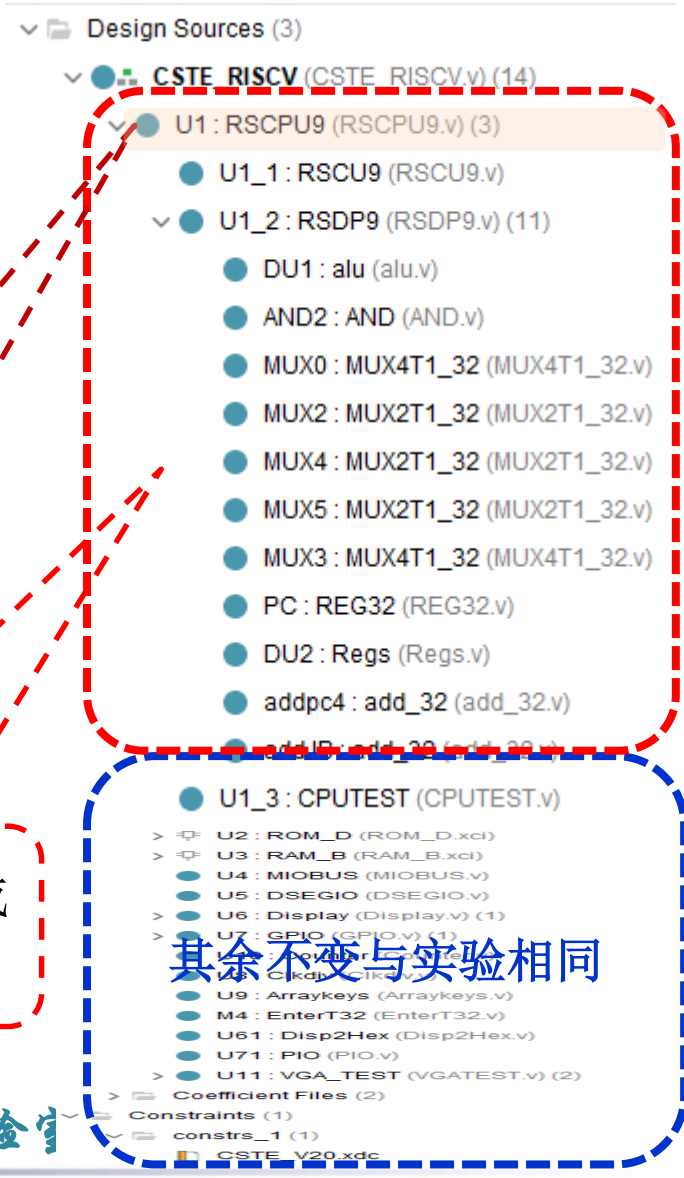
设计要点：CPU集成替换后

- 集成替换RSCU9核后CPU模块层次结构
  - CPU部分已经完全是自己的代码
  - 实验七扩展设计CPU指令功能
  - 实验八扩展设计CPU中断功能

设计一个简单CPU也不难吧！
加个中断
小场景应用也行！

CPU顶层描述，与实验四/五 相同

OExp06控制器替换完成后的CPU模块层次关系

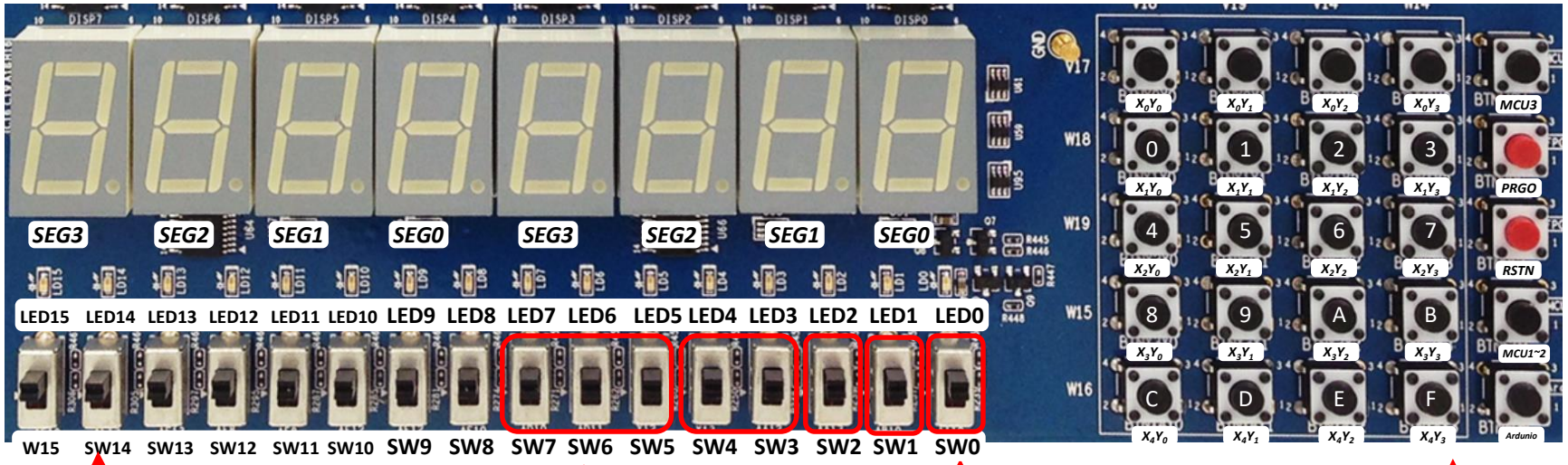


其余不变与实验相同

物理验证

- 使用DEMO程序目测控制器功能(同实验五)
  - DEMO接口功能
    - SW[7:5]=000， SW[2]=0(全速运行)
      - SW[4:3]=00， SW[0]=0，点阵显示程序：跑马灯
      - SW[4:3]=00， SW[0]=0，点阵显示程序：矩形变幻
      - SW[4:3]=01， SW[0]=1，内存数据显示程序：0～F
      - SW[4:3]=10， SW[0]=1，当前寄存器R9+1显示
- 用汇编语言设计测试程序
  - 测试ALU指令(R-格式译码、 Function译码)
  - 测试LW指令(I-格式译码)
  - 测试SW指令(S-格式译码)
  - 测试分支指令(B-格式译码)
  - 测试转移指令(J-格式译码)

设计要点：物理验证接口（详细参见实验二）



SW[0]=文本图形选择

SW[1]=高低16位选择

没有使用

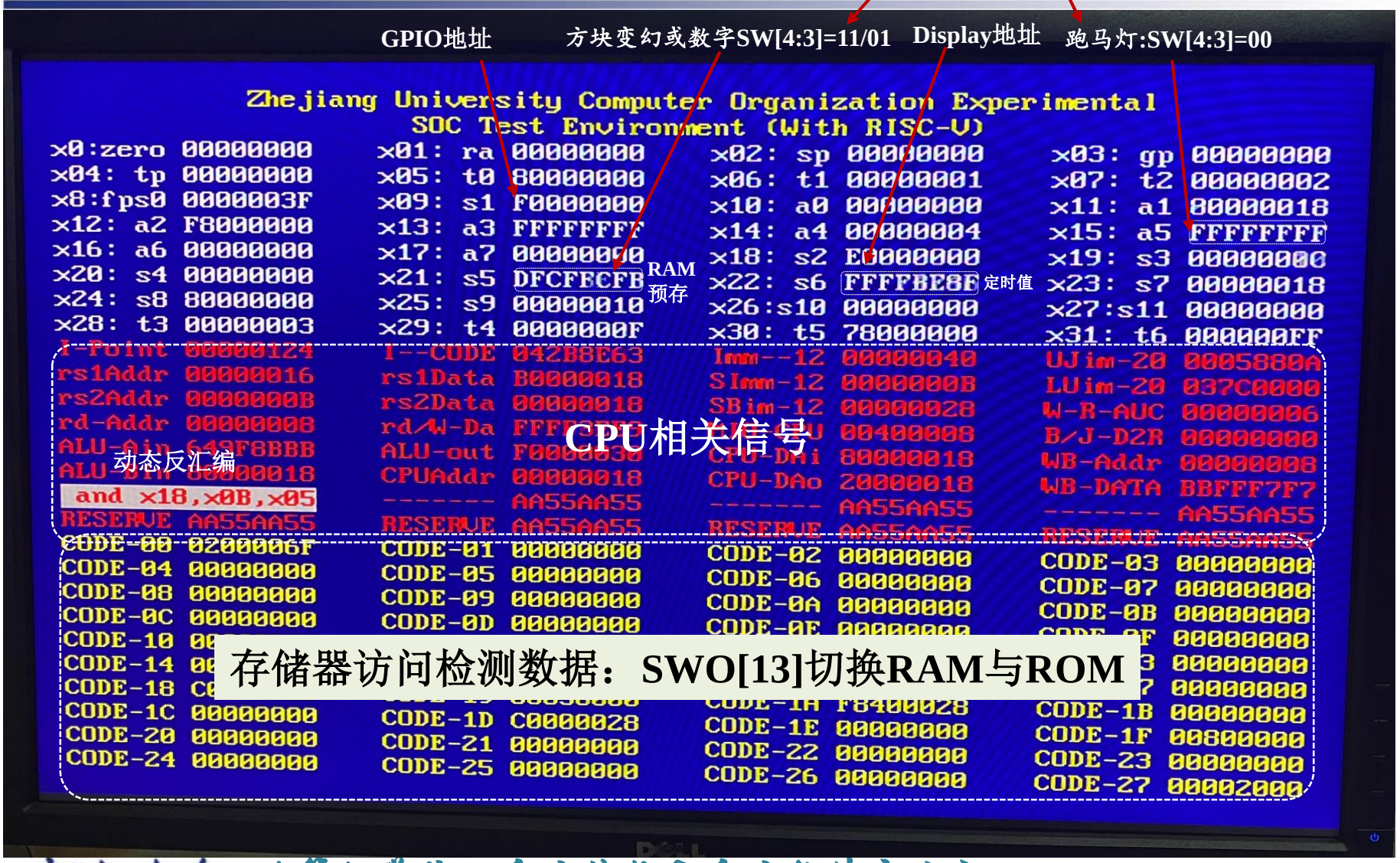
SW[2]=CPU单步时钟选择

SW[13]=0选择测试ROM
SW[]13=1选择测试RAM
SW[14]=0/1测试数据翻页

SW[7:5]=显示通道选择
SW[7:5]=000：CPU程序运行输出
SW[7:5]=001：测试PC字地址
SW[7:5]=010：测试指令字
SW[7:5]=011：测试计数器
SW[7:5]=100：测试RAM地址
SW[7:5]=101：测试CPU数据输出
SW[7:5]=110：测试CPU数据输入

SW[4:3]=00，点阵显示程序：跑马灯
SW[4:3]=00，点阵显示程序：矩形变幻
SW[4:3]=01，内存数据显示程序：0～F
SW[4:3]=10，当前寄存器+1显示(用户扩展保留)

VGA DEBUG 显示


七段Display上显示内容
GPIO地址
方块变幻或数字SW[4:3]=11/01
Display地址
跑马灯:SW[4:3]=00
Zhejiang University Computer Organization Experimental
SOC Test Environment (With RISC-V)
x0:zero 00000000
x01: ra 00000000
x02: sp 00000000
x03: gp 00000000
x04: tp 00000000
x05: t0 80000000
x06: t1 00000001
x07: t2 00000002
x8:fps0 0000003F
x09: s1 F0000000
x10: a0 00000000
x11: a1 00000018
x12: a2 F8000000
x13: a3 FFFFFFFF
x14: a4 00000004
x15: a5 FFFFFFFF
x16: a6 00000000
x17: a7 00000000
x18: s2 E0000000
x19: s3 00000000
x20: s4 00000000
x21: s5 DFCFECFB
x22: s6 FFFFBE8F
x23: s7 00000018
RAM预存
定时值
x24: s8 80000000
x25: s9 00000010
x26:s10 00000000
x27:s11 00000000
x28: t3 00000003
x29: t4 0000000F
x30: t5 78000000
x31: t6 000000FF
CPU相关信号
动态反汇编
and x18,x0B,x05
存储器访问检测数据：SWO[13]切换RAM与ROM

测试程序参考：ALU指令

- 设计ALU指令测试程序替换DEMO程序
  - ALU、Regs测试参考设计，测试结果通过CPU输出信号单步观察
  - SW[7:5]=100, Addr_out = ALU输出
  - SW[7:5]=101，Data_out= 寄存器B输出

```
loop:   addi x1,x0,1;          //x1=00000001
        slt  x2,x0,x1;         //x2=00000001
        add x3,x2,x2;          //x3=00000002
        add x4,x3,x3;          //x4=00000004
        add x5,x4,x2;          //x5=00000005
        add x6,x5,x5;          //x6=0000000A
        sub x7,x6,x5;          //x7=00000005
        and x8,x7,x5;          //x9=0000000A
        sub x10,x8,x6;         //x10=00000000
        or   x11,x5,x6;        //x11=0000000F
        or   x12,x11,x7;       //x12=0000000A
        slt   x13,x7,x7;        //x13=00000000
        ……
         beq x0,x0, loop;
```

测试程序参考： LW/SW

- 设计LW/SW程序替换DEMO程序
  - 参考OExp05通道测试设计。测试结果通过CPU输出信号单步观察
  - 存储器地址通过Addr_out通道4观察： 13+x0

```
#baseAddr 0000
start:                                  //通道结果由后一条指令读操作数观察
        lw   x5, 0x34(x0);              //取测试常数55555555。存储器读通道
start_A:
        add x1, x5, x0;                 //r1: 寄存器写通道。R5:寄存器读通道A输出
        xor x2, x0, x1;                 //r1: 寄存器读通道B输出。R2:ALU输出通道
        lw   x5, 0x48(x0);  //取测试常数AAAAAAAA。立即数通道:00000048
        beq x2, x5 test_sw;             //循环测试
        jal   x0,  start;               //循环测试。
test_sw: ……                            //增加写SW测试，如34和48单元交换
        beq x0,x0,start;                //循环测试。立即数通道：00000014
```

- 测试的完备性
  - 上述测试正确仅表明地址计算、存储单元和总线传输部分正确
  - 要测试其完全正确，必须遍历所有可能的情况

动态LW/SW测试

- 利用七段显示设备可以设计动态测试程序
  - 7段码显示器的地址是E0000000/FFFFFFE0
  - LED显示地址是F0000000/FFFFFF00
  - SW指令输出测试结果：sw
  - 请设计存储器模块测试程序
    - 测试结果在7段显示器上指示
- **RAM初始化数据同OExp05**

F0000000, 000002AB, 80000000, 0000003F, 00000001, FFF70000,
0000FFFF, 80000000, 00000000, 11111111, 22222222, 33333333,
44444444, 55555555, 66666666, 77777777, 88888888, 99999999,
aaaaaaaa, bbbbbbbb, cccccccc, dddddddd, eeeeeeee, FFFFFFFF,
557EF7E0, D7BDFBD9, D7DBFDB9, DFCFFCFB, DFCFBFFF, F7F3DFFF,
FFFFDF3D, FFFF9DB9, FFFFBCFB, DFCFFCFB, DFCFBFFF, D7DB9FFF,
D7DBFDB9, D7BDFBD9, FFFF07E0, 007E0FFF, 03bdf020, 03def820,
08002300;

设计测试记录表格

- **ALU指令**测试结果记录
  - 自行设计记录表格
- **LW/SW指令测试结果记录**
  - 自行设计记录表格
- **动态存储模块测试记录**
  - 自行设计记录表格

思考题

- 单周期控制器时序体现在那里？
- 设计bne指令需要增加控制信号吗？
- 扩展下列指令，控制器将作如何修改：

  R-Type：　　sra, sll, sltu；

  I-Type：　　srai, slli, sltiu, jalr;

  B-Type：　　bne；

  U-Type：　　lui；

  - 此时用二级译码有优势吗？
- 动态存储模块测试七段显示会出现什么问题？


END

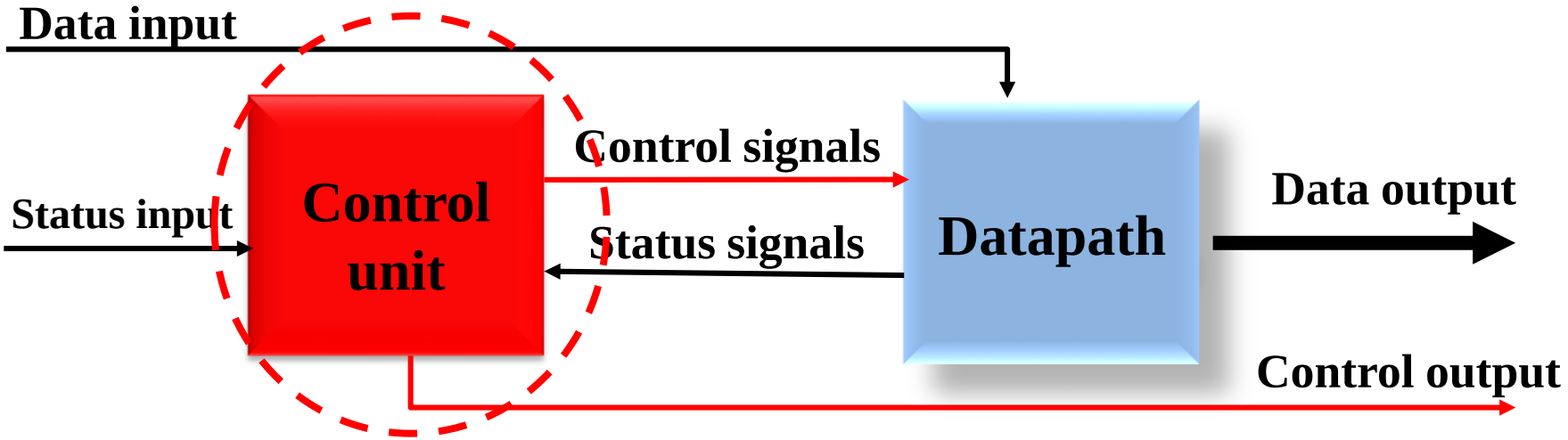
Appendix(供参考，接口信号以教学班约定)

RISC-V RV32I控制器的原理介绍

CPU organization

Digital circuit

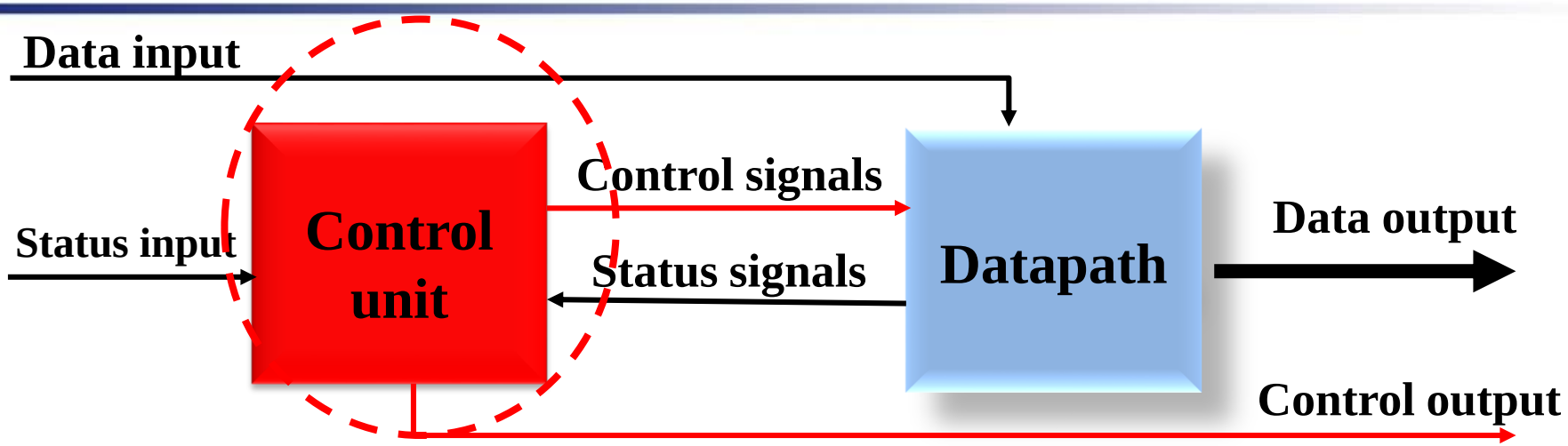
- General circuits that controls logical event with logical gates --Hardware



Computer organization

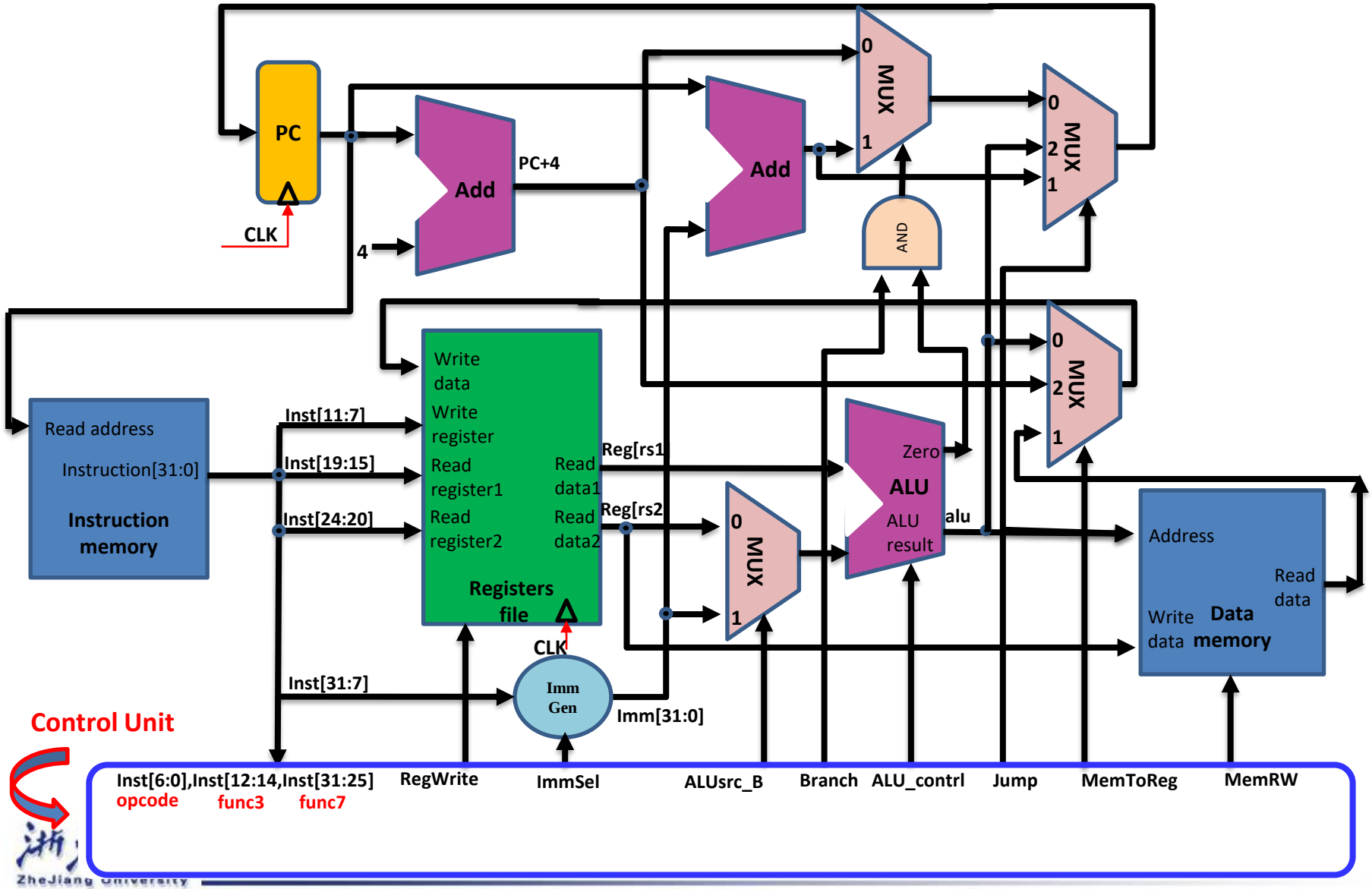
- Special circuits that processes logical action with instructions -Software

Control unit

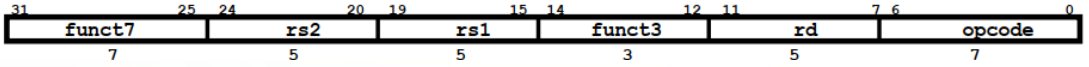


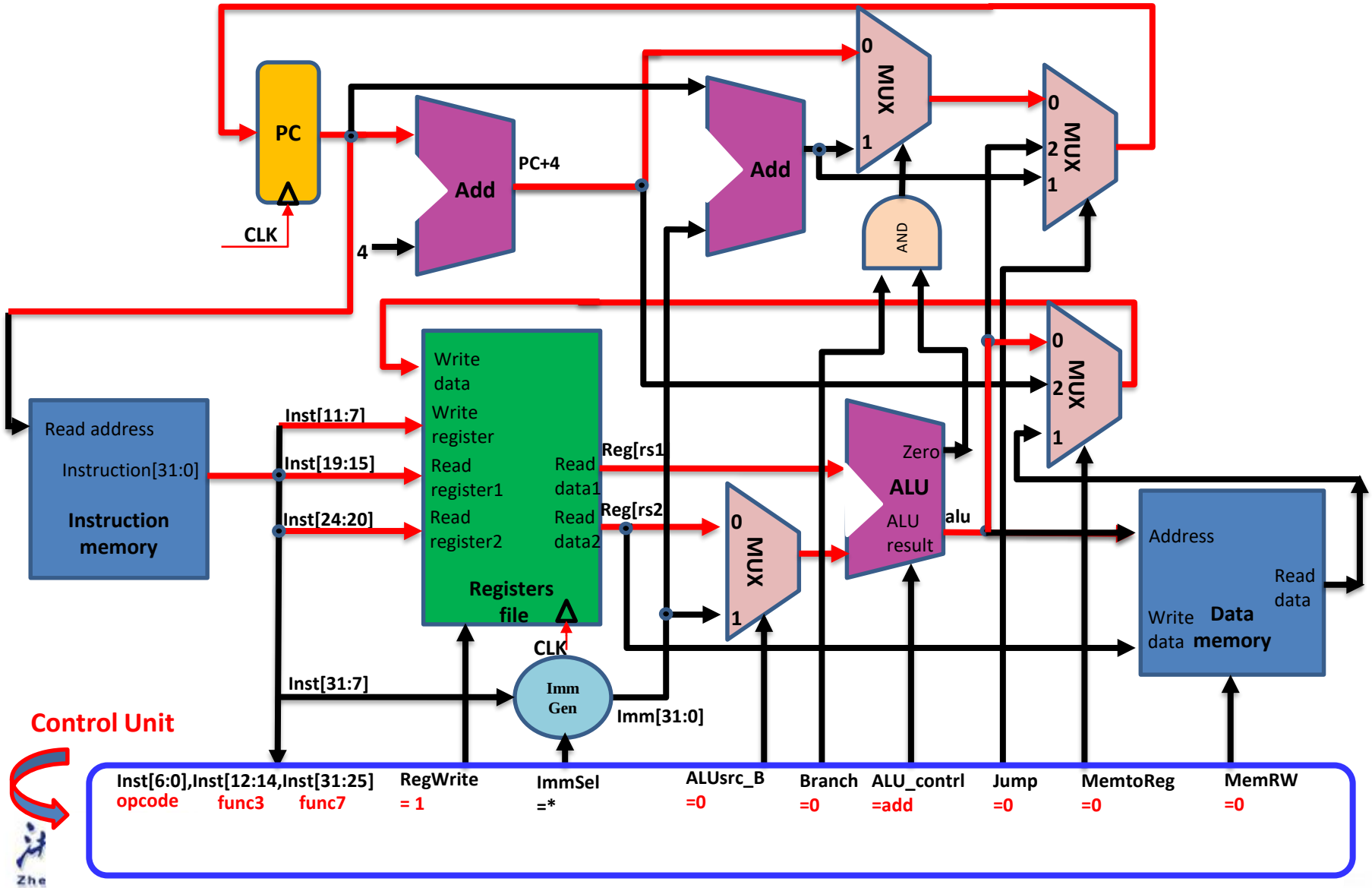
- 控制单元
- 控制单元作为处理器的一部分，用以告诉数据通路需要做什么。包含了取指单元（PC及地址计算单元）、译码单元、控制单元（时序信号形成及微操作控制信号形成电路）、中断等

控制对象：数据通路结构

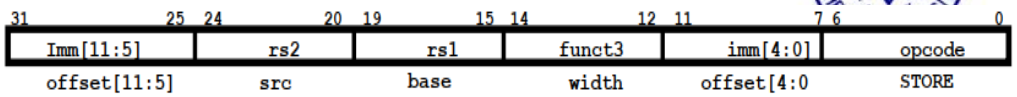

PC
CLK
Add
4
PC+4
Add
MUX
0
1
MUX
0
2
1
AND
MUX
0
2
1
Write
data
Write
register
Read
register1
Read
register2
Registers
file
Read
data1
Read
data2
CLK
Read address
Instruction[31:0]
Instruction
memory
Inst[11:7]
Inst[19:15]
Inst[24:20]
Reg[rs1
Reg[rs2
MUX
0
1
Zero
ALU
ALU
result
alu
Address
Read
data
Write
data
Data
memory
Inst[31:7]
Imm
Gen
Imm[31:0]
Control Unit
Inst[6:0],Inst[12:14,Inst[31:25]
opcode
func3
func7
RegWrite
ImmSel
ALUsrc_B
Branch
ALU_contrl
Jump
MemToReg
MemRW

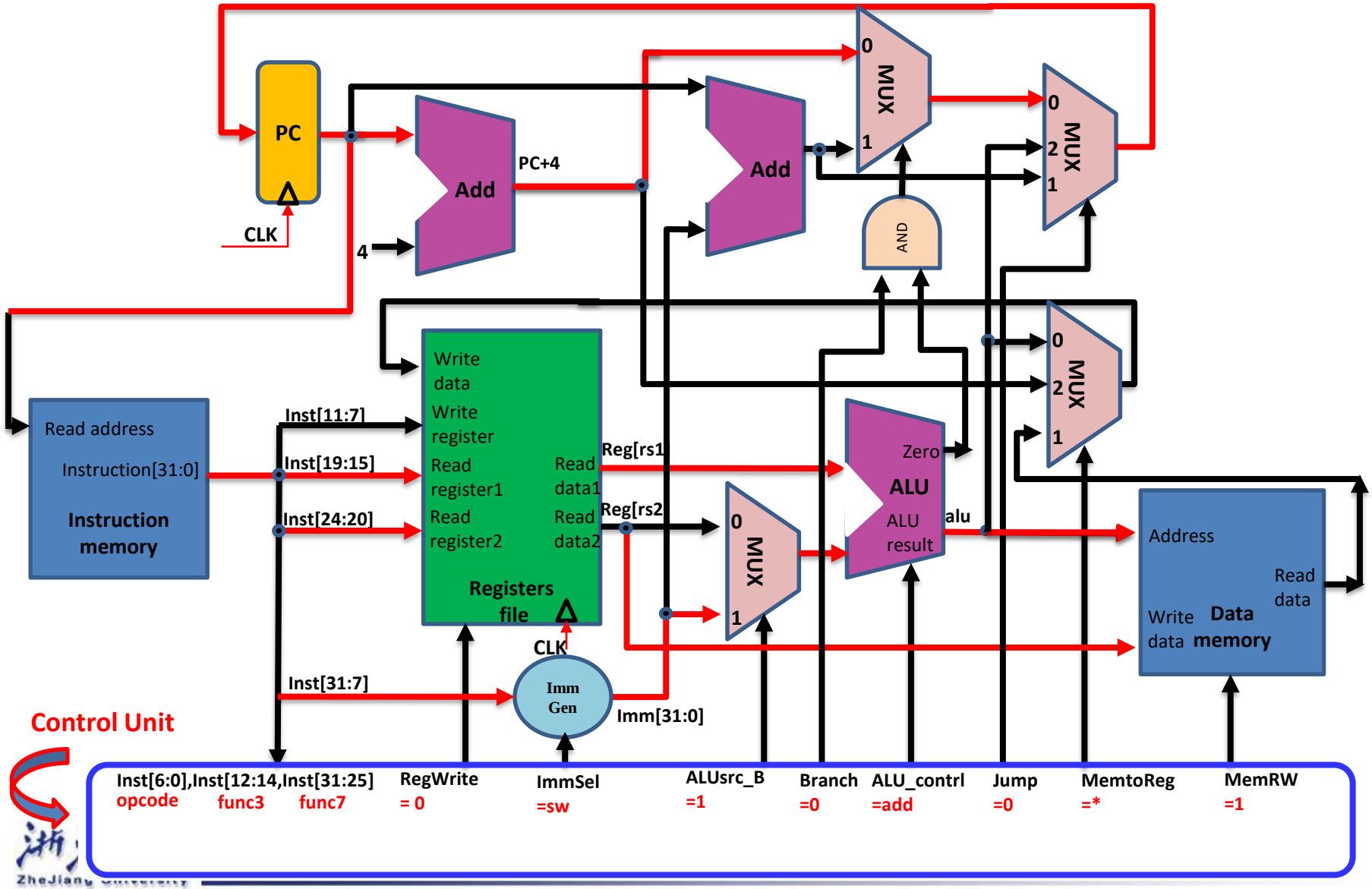
Control unit--add



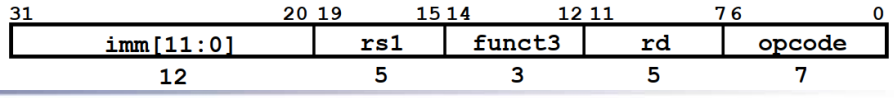


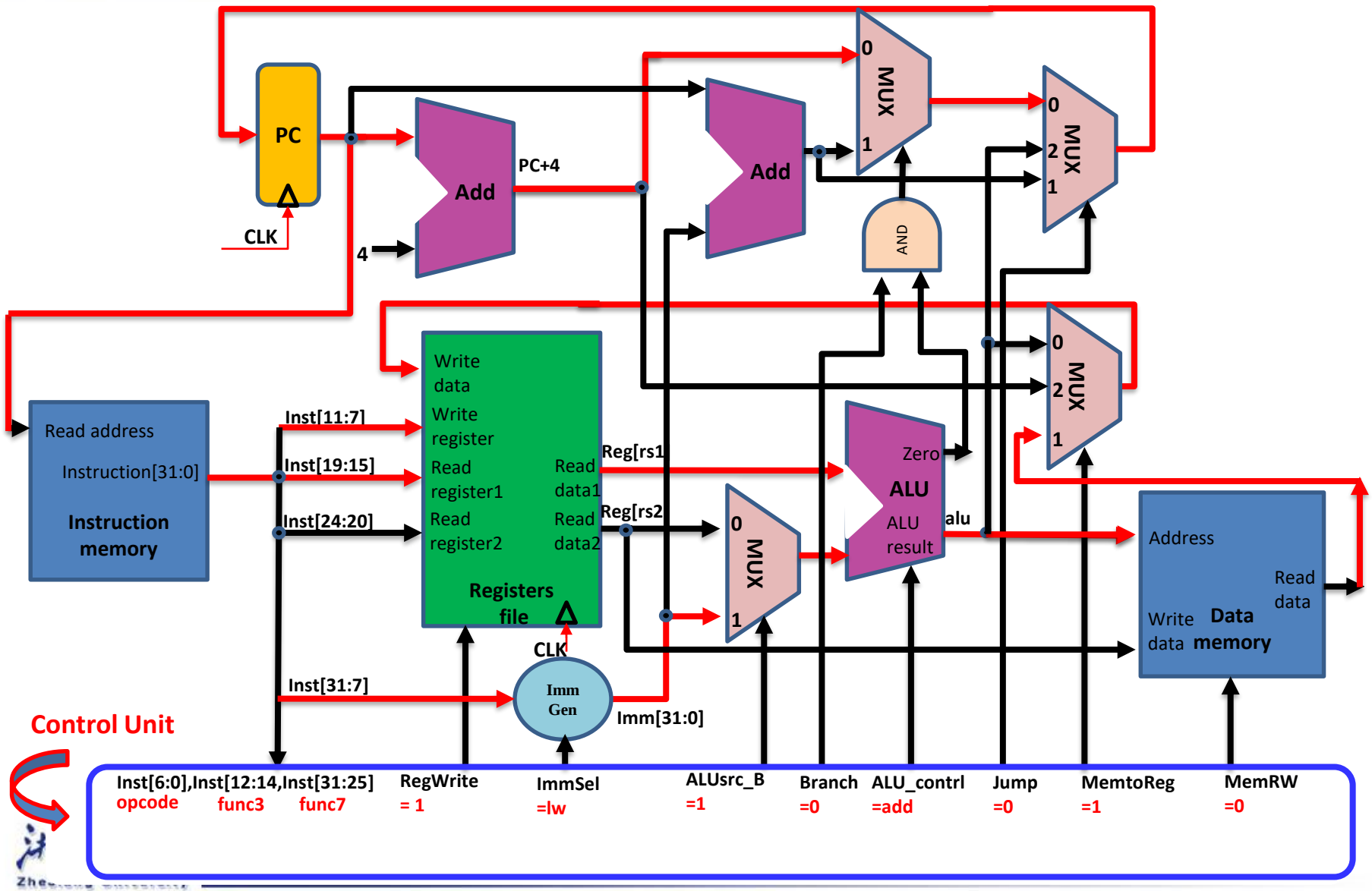
Control unit--sw



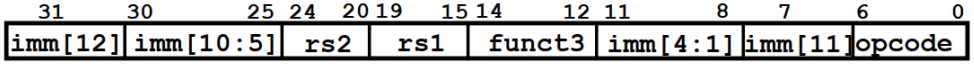


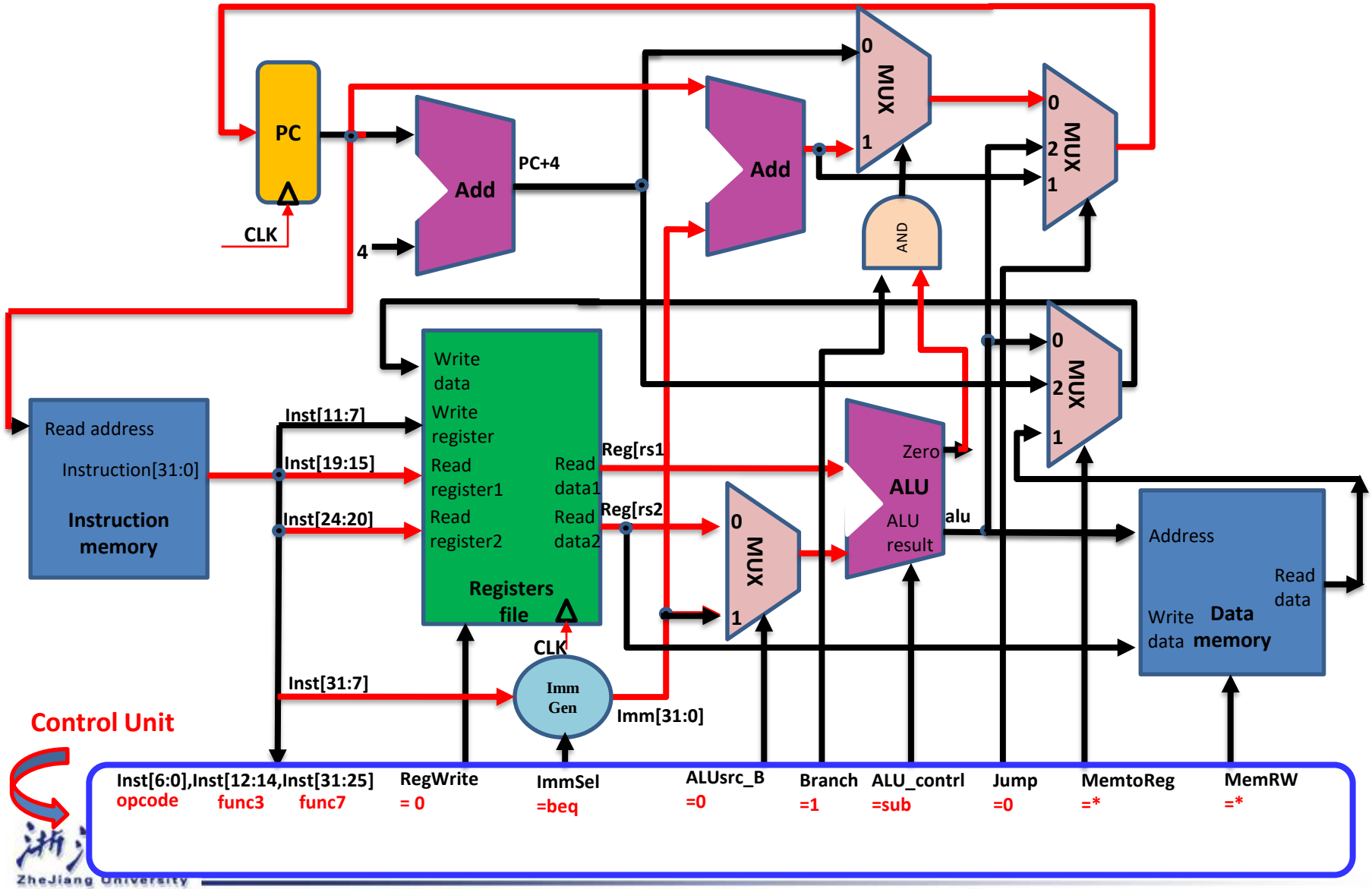
Control unit--lw



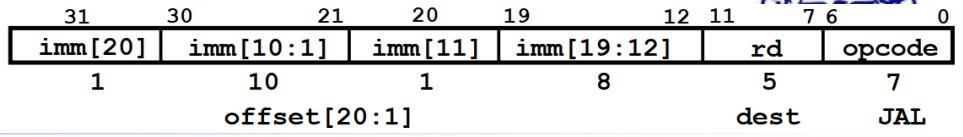


Control unit--beq





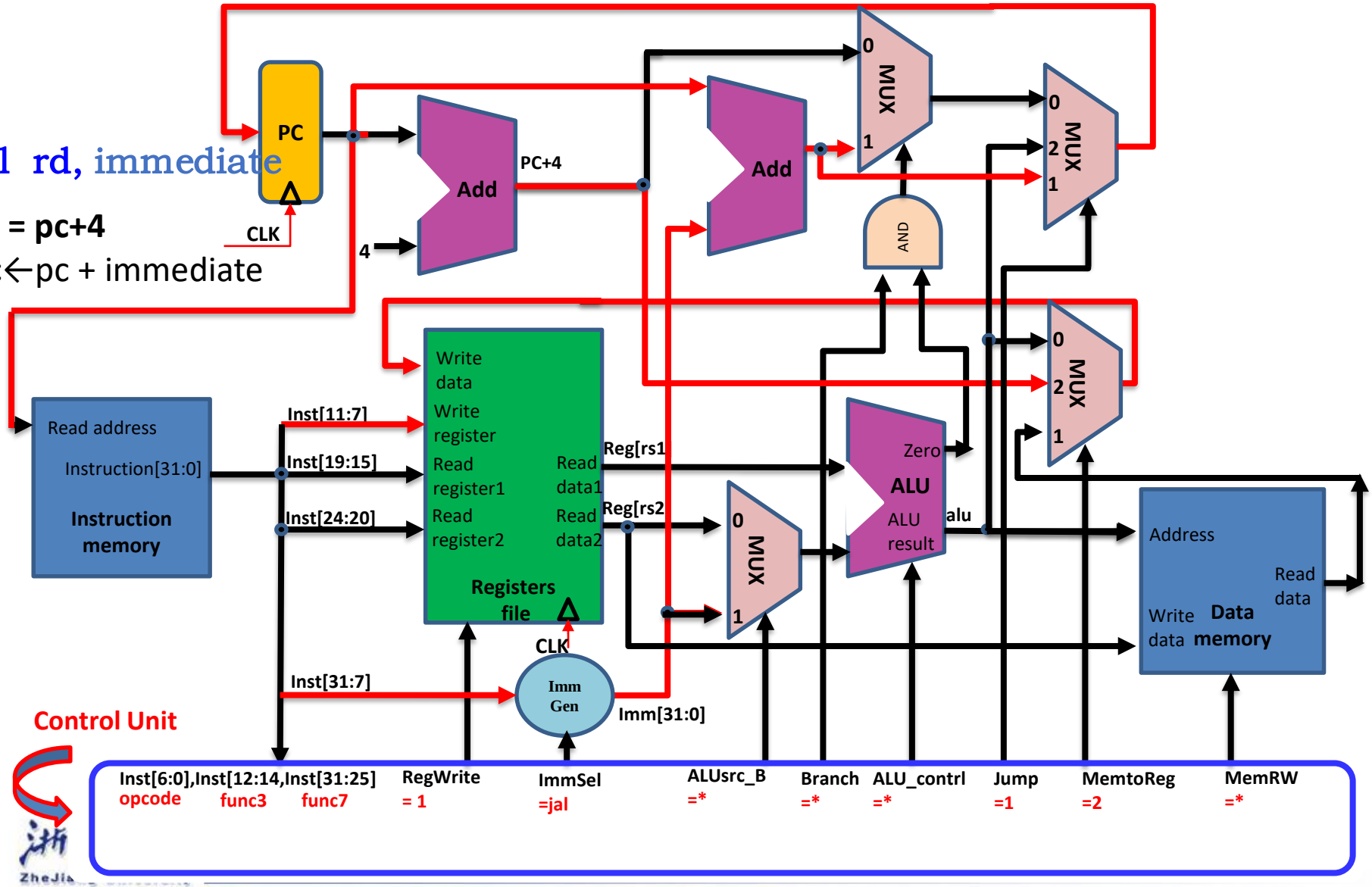
Control unit--jal



jal rd, immediate

rd = pc+4

pc←pc + immediate



控制信号定义

通路与操作控制

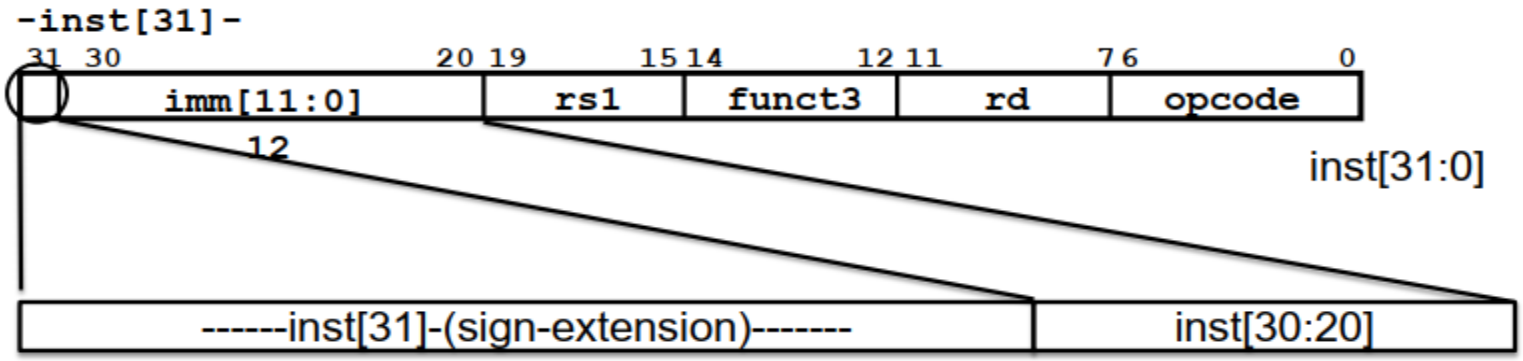
信号	源数目	功能定义	赋值0时动作	赋值1时动作	赋值2时动作
ALUSrc_B	2	ALU端口B输入选择	选择源操作数寄存器2数据	选择32位立即数（符号扩展后）	-
MemToReg	3	寄存器写入数据选择	选择ALU输出	选择存储器数据	选择PC+4
Branch	2	Beq指令目标地址选择	选择PC+4地址	选择转移目的地址 PC+imm（zero=1）	-
Jump	3	J指令目标地址选择	由Branch决定输出	选择跳转目标地址 PC+imm（JAL）	-
RegWrite	-	寄存器写控制	禁止寄存器写	使能寄存器写	-
MemRW	-	存储器读写控制	存储器**读使能**，存储器写禁止	存储器**写使能**，存储器读禁止	-
ALU_Control	000-111	3位ALU操作控制	参考表ALU_Control		
ImmSel	000-111	3位立即数组合控制	参考表ImmSel		

主控制器信号真值表

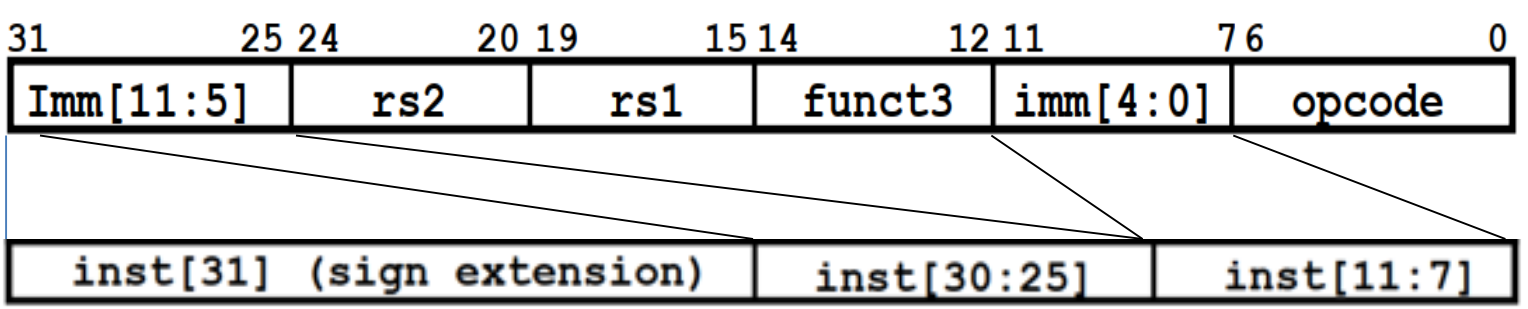
Inst[31:0]	Banch	Jump	ImmSel	ALUSrc_B	ALU_Control	MemRW	RegWrite	MemtoReg
add	0	0	*	Reg	Add	Read	1	ALU
sub	0	0	*	Reg	Sub	Read	1	ALU
(R-R Op)	0	0	*	Reg	*(Op)*	Read	1	ALU
addi	0	0	I	Imm	Add	Read	1	ALU
lw	0	0	I	Imm	Add	Read	1	Mem
sw	0	0	S	Imm	Add	Write	0	*
beq	0	0	B	Reg	Sub	Read	0	*
beq	1	0	B	Reg	sub	Read	0	*
jal	0	1	J	Imm	*	Read	1	PC+4
lui	0	0	U	Imm	Add	Read	1	ALU

ImmSel

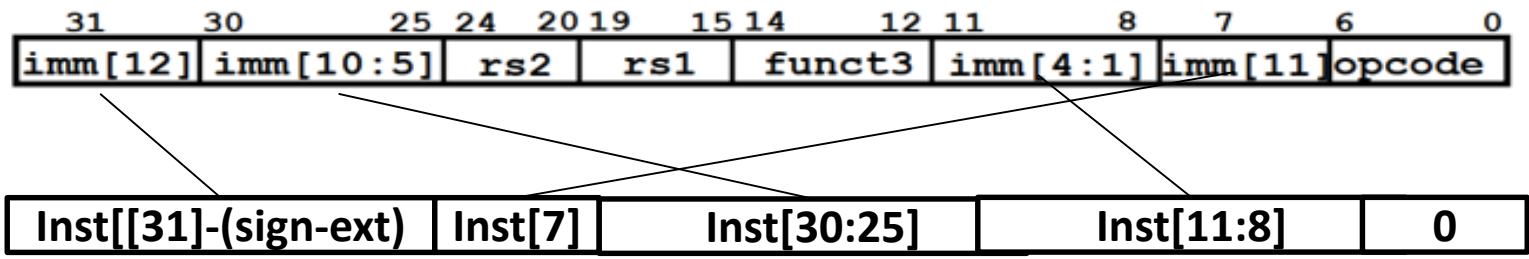
I-Format



S-Format

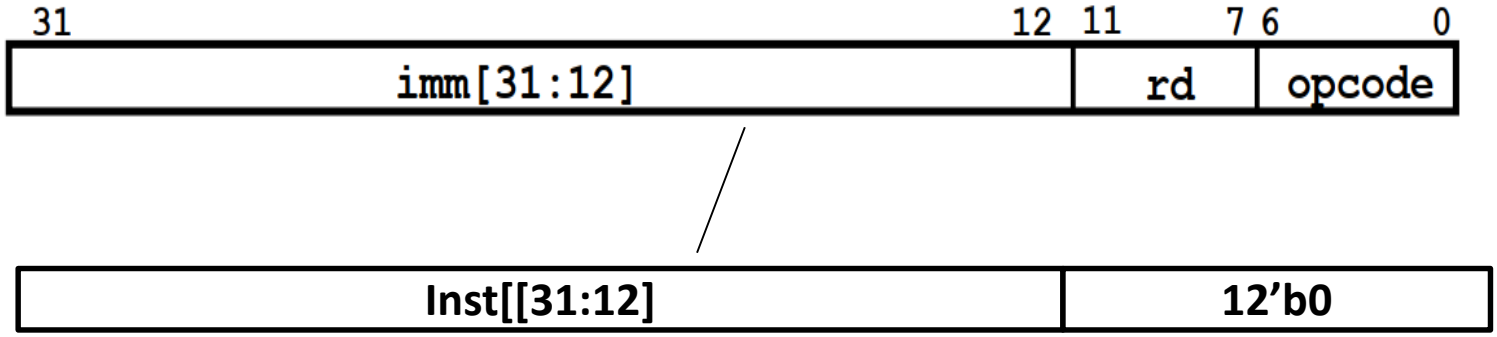


B-Format

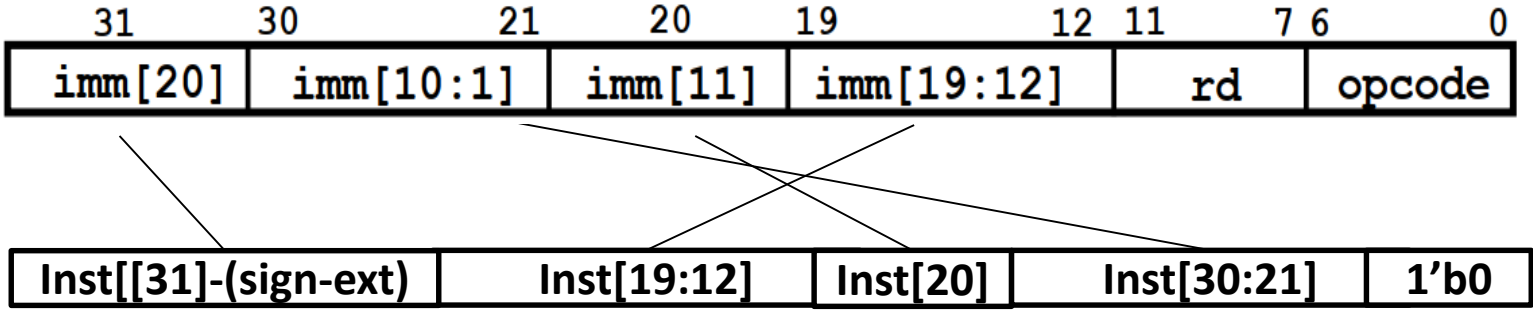


ImmSel

U-Format



J-Format



ImmSel

Instruction type	Instruction opcode[6:0]	Instruction operation	(sign-extend)immediate	Imm Sel
I-type	0000011	Lw;lbu;lh; lb;lhu	**(sign-extend) instr[31:20]**	00
	0010011	Addi;slti;sltiu;xori;ori;andi;		
	1100111	jalr		
S-type	0100011	Sw;sb;sh	**(sign-extend) instr[31:25],[11:7]**	01
B-type	1100011	Beq;bne;blt;bge;bltu;bgeu	**(sign-extend) instr[31],[7],[30:25],[11:8],1'b0**	10
J-type	1101111	jal	**(sign-extend) instr[31],[19:12],[20],[30:21],1'b0**	11

RV32I--encode

imm[31:12]			rd	0110111	LUI
imm[31:12]			rd	0010111	AUIPC
imm[20|10:1|11|19:12]			rd	1101111	JAL
imm[11:0]	rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI

inst[30] inst[14:12] inst[6:2]

0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

0000	pred	succ	00000	000	00000	0001111	I fence
0000	0000	0000	00000	001	00000	0001111	I fence.i
000000000000			00000	000	00000	1110011	I ecall
000000000001			00000	000	00000	1110011	I ebreak
csr			rs1	001	rd	1110011	I csrrw
csr			rs1	010	rd	1110011	I csrrs
csr			rs1	011	rd	1110011	I csrrc
csr			zimm	101	rd	1110011	I csrrwi
csr			zimm	110	rd	1110011	I cssrrsi
csr			zimm	111	rd	1110011	I csrrci

Instruction type encoded using only 9 bits
inst[30], inst[14:12], inst[6:2]

特殊指令本实验暂不做考虑

RV32I--decode

Instruction opcode	op	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
R-type	0110011	add	000	0000000	10	add	010
		sub	000	0100000		sub	110
		sll	001	0000000		sll	-
		slt	**010**	**0000000**		slt	111
		sltu	011	0000000		sltu	-
		xor	**100**	**0000000**		xor	011
		srl	**101**	**0000000**		srl	101
		sra	101	0100000		sra	-
		or	**110**	**0000000**		or	001
		and	**111**	**0000000**		and	000

RV32I--decode

Instruction opcode	op	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
S-Type	0100011	sb	000	-	00	add	010
		sh	001	-		add	010
		sw	010	-		add	010

Instruction opcode	op	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
B-Type	1100011	Beq	000	-	01	sub	110
		Bne	001	-		sub	110

RV32I--decode

Instruction opcode	op	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
U-Type	0110111	lui	-	-	-	--	-
	0010111	auipc	-	-		-	-
Instruction opcode	op	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
J-Type	1101111	jal	-	-	-	-	-

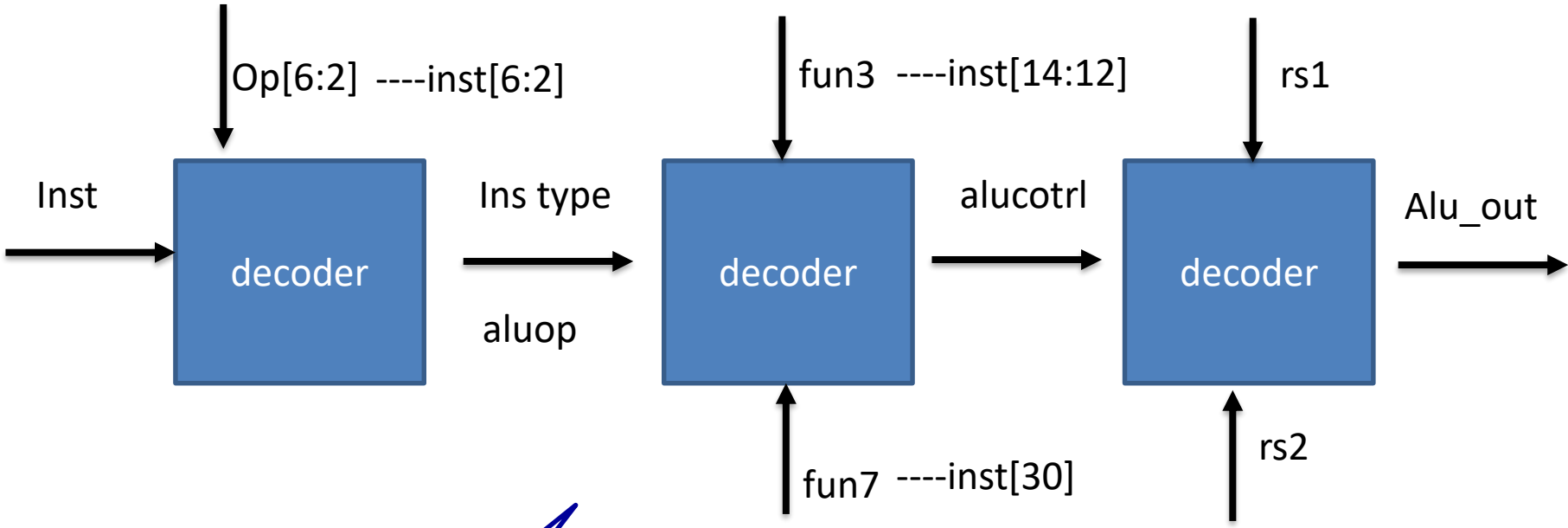
RV32I-decode

Instruction opcode	op	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
I-Type	0000011	Lb	-	-	00	add	010
		Lh	-	-		add	010
		Lw	010	-		add	010
		Lbu	-	-		add	010
		lhu	-	-		add	010

RV32I---decode

Instruction opcode	op	Instruction operation	Funct 3	Funct7	ALUop	Desired ALU action	ALUControl
I-Type	0010011	addi	000	-	11	add	010
		slti	010	-		slt	111
		sltiu	011	-		sltu	-
		xori	100	-		xor	011
		ori	110	-		or	001
		andi	111	-		and	000
		slli	001	0000000		sll	-
		srli	101	0000000		srl	101
		srai	101	0100000		sra	-

ALU操作译码--多级译码



一级译码如何实现，优缺点如何？