浙江大学

计算机组成实验报告

实验名称：	单周期CPU设计（含指令扩展）
姓　　名：	蒋奕
学　　号：	3210103803
专　　业：	计算机科学与技术
课程名称：	计算机组成
实验地点：	东4-509
指导教师：	赵莎

2023 年 5 月 5 日

单周期CPU设计（含指令扩展）

1 实验目的和要求

- 复习寄存器传输控制技术，运用寄存器传输控制技术
- 掌握 CPU 的核心组成：数据通路与控制器，指令执行过程与控制流关系
- 设计数据通路的功能部件和控制器
- 进一步了解计算机系统的基本结构
- 熟练掌握 IP 核的使用方法
- 掌握 CPU 的核心：数据通路组成与原理,设计数据通路,指令执行过程与控制流关系
- 学习测试方案的设计,学习测试程序的设计，设计测试程序

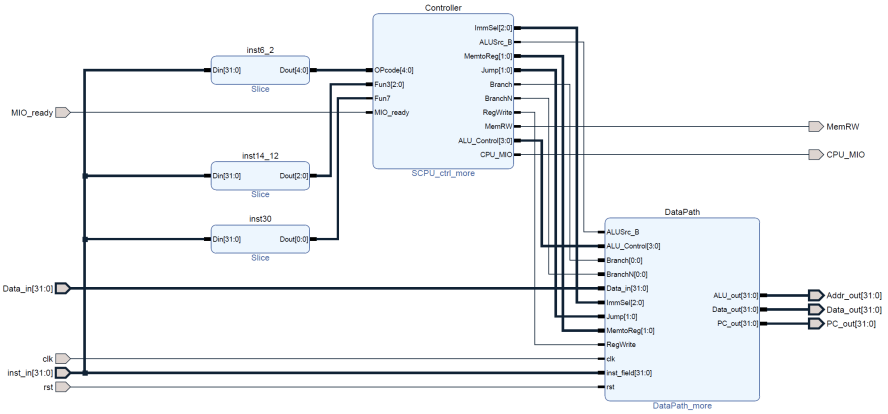
2 实验内容和原理

2.1 实验任务

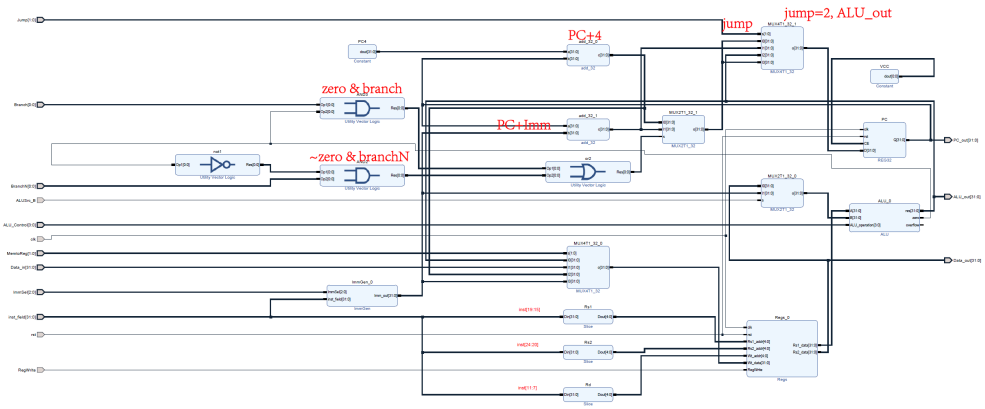
- RTL 代码描述 CPU 设计 SCPU.v
- 设计实现数据通路采用 RTL 实现
- ImmGen立即数生成模块设计采用 RTL 实现
- PC寄存器设计及 PC 通路建立
- 熟练掌握 IP 核的使用方法
- 用硬件描述语言设计实现控制器,设计指令集测试方案并完成测试。设计控制器测试方案并完成测试(译码测试： R格式、访存指令、分支指令，转移指令)

2.2 实验原理

SCPU 原理图如下所示



datapath 原理图如下所示



控制信号设计如下图所示

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	赋值3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	4	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	选择imm
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）	-	-
BranchN	2	Bne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=0）	-	-
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）	选择跳转目标地址ALU输出（JALR:rs1+imm）	-
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	0000 - 1111	4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			

控制信号真值表如下图所示

Inst[31:0]	Banch	Branch N	Jump	ImmSel	ALUSrc_B	ALU_Control	MemRW	RegWrite	MemtoReg
add	0	0	0	*	Reg (0)	Add	Read	1	ALU(0)
sub	0	0	0	*	Reg (0)	Sub	Read	1	ALU(0)
(R-R Op)	0	0	0	*	Reg (0)	*(Op)*	Read	1	ALU(0)
addi	0	0	0	I	Imm (1)	Add	Read	1	ALU(0)
lw	0	0	0	I	Imm (1)	Add	Read	1	Mem(1)
sw	0	0	0	S	Imm (1)	Add	Write	0	*
beq	0	0	0	B	Reg (0)	sub	*	0	*
beq	1	*	0	B	Reg (0)	sub	*	0	*
bne	0	0	0	B	Reg (0)	sub	*	0	*
bne	*	1	0	B	Reg (0)	sub	*	0	*
jalr	*	*	2	I	Imm (1)	Add	*	1	PC+4(2)
jal	*	*	1	J	Imm (1)	*	*	1	PC+4(2)
lui	0	0	0	U	*	*	*	1	Imm(3)

3 实验设备和环境

计算机（Intel Core i5以上，4GB内存以上）系统,Sword2.0/Sword4.0开发板,Xilinx VIVADO 2017.4及以上开发工具

4 实验实现方法、步骤与调试

本实验主要采用Verilog语言描述并且实现原理图中的各个模块。然后替代lab4-2中的各个同名模块，并且自己设计测试程序进行调试直到功能正确。

SCPU 实现代码如下：

```
`timescale 1ns / 1ps
module SCPU(
    input wire clk,
    input wire rst,
    input wire MIO_ready,
    input wire [31:0] inst_in,
    input wire [31:0] Data_in,
    output wire MemRW,
    output wire CPU_MIO,
    output wire [31:0] Addr_out,
    output wire [31:0] Data_out,
    output wire [31:0] PC_out
    );
    wire [2:0] ImmSel;//changed from [1:0]
    wire [1:0] MemtoReg;
    wire [3:0] ALU_Control;//changed from [3:0]
    wire [3:0] ALU_operation;
    assign ALU_operation = ALU_Control;
    wire [1:0] Jump;
    wire Branch,BranchN,RegWrite,ALUSrc_B;
    SCPU_ctrl mySCPU_ctrl(
        .OPcode(inst_in[6:2]),
        .Fun3(inst_in[14:12]),
        .Fun7(inst_in[30]),
        .MIO_ready(MIO_ready),
        .ImmSel(ImmSel),
        .ALUSrc_B(ALUSrc_B),
        .MemtoReg(MemtoReg),
        .Jump(Jump),
        .Branch(Branch),
        .BranchN(BranchN),
        .RegWrite(RegWrite),
        .MemRW(MemRW),
        .ALU_Control(ALU_Control),
        .CPU_MIO(CPU_MIO)
    );
    DataPath myDataPath(
        .ALUSrc_B(ALUSrc_B),
        .ALU_operation(ALU_operation),//ALU_Control ALU_Control
        .Branch(Branch),
        .BranchN(BranchN),
        .Data_in(Data_in),
        .ImmSel(ImmSel),
        .Jump(Jump),
```

```
        .MemtoReg(MemtoReg),
        .RegWrite(RegWrite),
        .clk(clk),
        .rst(rst),
        .inst_field(inst_in),
        //output
        .ALU_out(Addr_out),
        .Data_out(Data_out),
        .PC_out(PC_out)
    );
endmodule
```

1: SCPU.v

datapath 实现代码如下：

```
`timescale 1ns / 1ps
module DataPath(
    input wire clk,
    input wire rst,
    input wire [1:0] Jump,
    input wire Branch,
    input wire BranchN,
    input wire [1:0] MemtoReg,
    input wire ALUSrc_B,
    input wire RegWrite,
    input wire [2:0] ImmSel,
    input wire [3:0] ALU_operation,
    input wire [31:0] Data_in,
    input wire [31:0] inst_field,
    output reg [31:0] PC_out,
    output reg [31:0] Data_out,
    output reg [31:0] ALU_out
    );
    wire zero;
    //generate imm
    wire [31:0] Imm_out;
    ImmGen myImmGen(
        .ImmSel(ImmSel),
        .inst_field(inst_field),
        .Imm_out(Imm_out)
    );
    //pc
    wire rs_rt_equal;
    wire [31:0] PC_next;
    wire [31:0] PC_jump;
    assign PC_next = PC_out + 4;
    assign PC_jump = PC_out + Imm_out;
    always @(posedge clk or posedge rst) begin
        if(rst)//reset pc -> 0
        begin
            PC_out <= 0;
        end
        else //if(Jump)
        begin
            case(Jump)
                2'b00://Branch related
                    if(Branch == 0 && BranchN == 0)
                        begin
                            PC_out <= PC_next;
```

```
45                      end
                    else if(Branch == 1 && zero == 1)//beq
47                      begin
                            PC_out <= PC_jump;
49                      end
                    else if(BranchN == 1 && zero == 0)//bne
51                      begin
                            PC_out <= PC_jump;
53                      end
                    else begin
55                      PC_out <= PC_next;
                    end
57              2'b01://jal
                    PC_out <= PC_jump;
59              2'b10://jalr: rs1data+imm
                    PC_out <= ALU_out;
61              default: //jump==2'b11
                    PC_out <= PC_next;
63          endcase
        end
65  end
    //reg ok
67  wire [4:0] Rs1_addr;
    wire [4:0] Rs2_addr;
69  wire [4:0] Wt_addr;
    assign Rs1_addr = inst_field[19:15];
71  assign Rs2_addr = inst_field[24:20];
    assign Wt_addr = inst_field[11:7];
73  reg [31:0] Wt_data;
    wire [31:0] Rs1_data;
75  wire [31:0] Rs2_data;
    //wt_data
77  always @(*) begin
        case(MemtoReg)
79          2'b00: Wt_data = ALU_out;
            2'b01: Wt_data = Data_in;
81          2'b10: Wt_data = PC_next;
            2'b11: Wt_data = Imm_out;//PC_next;
83      endcase
    end
85  regs myRegs(
        .clk(clk),
87      .rst(rst),
        .Rs1_addr(Rs1_addr),
89      .Rs2_addr(Rs2_addr),
        .Wt_addr(Wt_addr),
91      .Rs1_data(Rs1_data),
        .Rs2_data(Rs2_data),
93      .Wt_data(Wt_data),
        .RegWrite(RegWrite)
95  );
    assign rs_rt_equal = (Rs1_data == Rs2_data) ? 1 : 0;
97  //alu ok
    reg [31:0] alu_a;
99  reg [31:0] alu_b;
    wire [31:0] alu_out;
101 always @(*) begin
        alu_a = Rs1_data;
103     case(ALUSrc_B)
```

```
            1'b0: alu_b = Rs2_data;//data_out
            1'b1: alu_b = Imm_out;
        endcase
    end
    ALU myALU(
        .A(alu_a),
        .B(alu_b),
        .ALU_operation(ALU_operation),
        .res(alu_out),
        .zero(zero)
    );
    //data out and slu out
    always @(*) begin
        Data_out = Rs2_data;
        ALU_out = alu_out;
    end
endmodule
```

2: datapath.v

ImmGen 实现代码如下：

```
`timescale 1ns / 1ps
module ImmGen(
    input wire [2:0] ImmSel,
    input wire [31:0] inst_field,
    output reg [31:0] Imm_out
    );
    always @(*) begin
        case (ImmSel)
            3'b000://u-type
                Imm_out = {inst_field[31:12],12'h000};
            3'b001://i
                Imm_out = { {20{inst_field[31]}}, inst_field[31:20]};
            3'b010://s
                Imm_out = { {20{inst_field[31]}}, inst_field[31:25], inst_field[11:7]};
            3'b011://b
                Imm_out = { {20{inst_field[31]}}, inst_field[7], inst_field[30:25],
                    inst_field[11:8], 1'b0};
                //20 + 1 + 6 + 4 + 1 == 32
            3'b100://j
                Imm_out = { {12{inst_field[31]}}, inst_field[19:12], inst_field[20],
                    inst_field[30:21], 1'b0};
            default:
                Imm_out = 32'hxxxxxxxx;
        endcase
    end
endmodule
```

3: ImmGen.v

SCPU Ctrl 实现代码如下：

```
`timescale 1ns / 1ps
module SCPU_ctrl(
    input [4:0] OPcode,
    input [2:0] Fun3,
    input Fun7,
    input MIO_ready,
```

```
    output reg [2:0] ImmSel,
    output reg ALUSrc_B,
    output reg [1:0] MemtoReg,
    output reg [1:0]Jump,
    output reg Branch,
    output reg BranchN,
    output reg RegWrite,
    output reg MemRW,
    output reg [3:0] ALU_Control,
    output reg CPU_MIO
);
    reg [15:0] CPU_ctrl_signals;
    wire [3:0] Fun;
    `define CPU_ctrl_signals {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,BranchN,Jump,
        ALU_Control,ImmSel}
    assign Fun = {Fun3,Fun7};
    always @(*) begin
        case(OPcode)
        5'b01100: //R-type add;;;
        begin
            `CPU_ctrl_signals = {1'b0,2'b00,1'b1,1'b0,1'b0,1'b0,2'b00,4'bxxxx,3'b001};
            begin
                case(Fun)
                    4'b0000:ALU_Control = 4'b0010 ;//add
                    4'b0001:ALU_Control = 4'b0110;//sub
                    4'b0010:ALU_Control = 4'b1110 ;//sll
                    4'b0100:ALU_Control = 4'b0111;//slt
                    4'b0110: ALU_Control = 4'b1001;//sltu
                    4'b1000:ALU_Control = 4'b1100;//xor
                    4'b1010: ALU_Control = 4'b1101;//srl
                    4'b1011: ALU_Control = 4'b1111;//sra
                    4'b1100:ALU_Control = 4'b0001 ;//or
                    4'b1110:ALU_Control = 4'b0000;//and
                    default: ALU_Control = 4'b0010;
                endcase
            end
        end
        5'b00000: //I-type Lw
        begin
            `CPU_ctrl_signals = {1'b1,2'b01,1'b1,1'b0,1'b0,1'b0,2'b00,4'b0010,3'b001};
        end
        5'b00100://I-type ALU(addi;;;)
        begin
            `CPU_ctrl_signals = {1'b1,2'b00,1'b1,1'b0,1'b0,1'b0,2'b00,4'bxxxx,3'b001};
            begin
                case(Fun3)
                3'b000: ALU_Control = 4'b0010 ;//add
                3'b010: ALU_Control = 4'b0111;//slt
                3'b011: ALU_Control = 4'b1001;//sltu
                3'b100: ALU_Control = 4'b1100;//xor
                3'b110: ALU_Control = 4'b0001;//or
                3'b111: ALU_Control = 4'b0000 ;//and
                3'b001: ALU_Control = 4'b1110 ;//sll
                3'b101:
                    begin
                        case(Fun7)
                            1'b0: ALU_Control = 4'b1101;//srl
                            1'b1: ALU_Control = 4'b1111 ; //sra
                        endcase
```

```
                    end
                default: ALU_Control = 4'bxxxx;
                endcase
            end
        end
        5'b11001: //I-type jalr
        begin
            `CPU_ctrl_signals = {1'b1,2'b10,1'b1,1'b0,1'b0,1'b0,2'b10,4'b0010,3'b001};
        end
        5'b01000: //S-type Sw
        begin
            `CPU_ctrl_signals = {1'b1,2'b00,1'b0,1'b1,1'b0,1'b0,2'b00,4'b0010,3'b010};
        end
        5'b11000://B-type beq,bne branch   buyiyyang
        begin
            `CPU_ctrl_signals = {1'b0,2'b00,1'b0,1'b0,1'b0,1'b0,2'b00,4'b0110,3'b011};
            begin
                case(Fun3)
                    3'b000:begin Branch = 1; BranchN = 0; end
                    3'b001:begin BranchN = 1; Branch = 0;end
                    default: begin Branch = 0;BranchN = 0; end
                endcase
            end
        end
        5'b11011: //J-type jal
        begin
            `CPU_ctrl_signals = {1'b1,2'b10,1'b1,1'b0,1'b0,1'b0,2'b01,4'b0010,3'b100};
        end
        5'b00101: //u-type
        begin
            `CPU_ctrl_signals = {1'bx,2'b11,1'b1,1'b0,1'b0,1'b0,2'b00,4'bxxxx,3'b000};
        end
        5'b01101: //U-type lui
        begin
            `CPU_ctrl_signals = {1'b0,2'b11,1'b1,1'b0,1'b0,1'b0,2'b00,4'b0010,3'b000};
        end
        default:
        begin
            `CPU_ctrl_signals = {1'b0,2'b00,1'b0,1'b0,1'b0,1'b0,2'b00,4'bxxxx,3'b000};
        end
        endcase
    end
endmodule
```

4: SCPUCtrl.v

ALU 实现代码如下：

```
`timescale 1ns / 1ps
module ALU(
    input [31:0] A,
    input [31:0] B,
    input[3:0] ALU_operation,
    output reg[31:0] res,
    output zero,
    output overflow
);
    parameter one = 32'h00000001, zero_0 = 32'h00000000;
    always @(*)
        case (ALU_operation)
```

```
        4'b0000: res=A&B;
        4'b0001: res=A|B;
        4'b0010: res=A+B;
        4'b1100: res=A^B;
        4'b0110: res=A-B;
        4'b1101: res=A>>B[4:0]; // srl
        4'b1111: res=$signed(A)>>>$signed(B[4:0]); //sra
        4'b1001: res=(A < B) ? one : zero_0; //sltu
        4'b0111: res=($signed(A) < $signed(B)) ? one : zero_0; //slt
        4'b1110: res=A<<B[4:0]; //sll
        default: res=32'h00000000;
    endcase
    assign zero = (res==0)? 1: 0;
endmodule
```

5: ALU.v

regs 实现代码如下：

```
`timescale 1ns / 1ps
module regs(
    input clk,
    input rst,
    input RegWrite,
    input [4:0] Rs1_addr, Rs2_addr,Wt_addr ,
    input [31:0] Wt_data,
    output [31:0] Rs1_data, Rs2_data
);
    reg [31:0] register [1:31]; // r1 r31
    integer i;
    assign  Rs1_data = (Rs1_addr== 0) ? 0 : register[Rs1_addr];// read rdata_A
    assign  Rs2_data = (Rs2_addr== 0) ? 0 : register[Rs2_addr]; // read rdata_B
    always @(posedge clk or posedge rst )
    begin if (rst ==1)
                for (i=1; i<32; i=i+1) register[i] <= 0; // reset
            else if ( ( Wt_addr != 0) && (RegWrite == 1 ) )
                register[Wt_addr] <= Wt_data ; // write
    end
endmodule
```

6: regs.v

5 实验结果与分析

SCPU CTRL 仿真激励代码如下：

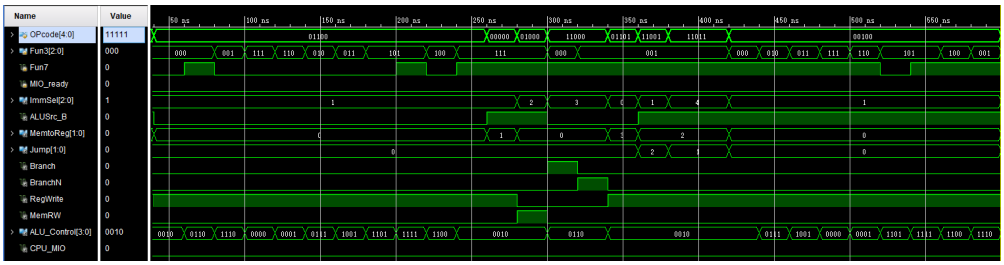
```
`timescale 1ns / 1ps
module SCPU_ctrl_tb();
    reg[4:0] OPcode; //OPcode
    reg[2:0] Fun3; //Function
    reg Fun7; //Function
    reg MIO_ready; //CPU Wait
    wire [2:0]ImmSel;
    wire ALUSrc_B;
    wire [1:0] MemtoReg;
    wire [1:0] Jump;
    wire Branch;
```

```
    wire BranchN;
    wire RegWrite;
    wire MemRW;
    wire [3:0]ALU_Control;
    wire CPU_MIO;
    SCPU_ctrl SCPU_ctrl_U(
        .OPcode (OPcode ),
        .Fun3 (Fun3),
        .Fun7 (Fun7),
        .MIO_ready (MIO_ready),
        .ImmSel (ImmSel),
        .ALUSrc_B (ALUSrc_B),
        .MemtoReg (MemtoReg),
        .Jump (Jump ),
        .Branch (Branch),
        .BranchN(BranchN),
        .RegWrite (RegWrite ),
        .MemRW (MemRW ),
        .ALU_Control (ALU_Control),
        .CPU_MIO (CPU_MIO )
    );
    initial begin
        OPcode = 0;
        Fun3 = 0;
        Fun7 = 0;
        MIO_ready = 0;
        #40;
        OPcode = 5'b01100; //R
        Fun3 = 3'b000;Fun7 = 1'b0;
        #20;
        Fun3 = 3'b000;Fun7 = 1'b1;
        #20;
        Fun3 = 3'b001;Fun7 = 1'b0;
        #20;
        Fun3 = 3'b111;Fun7 = 1'b0;
        #20;
        Fun3 = 3'b110;Fun7 = 1'b0;
        #20;
        Fun3 = 3'b010;Fun7 = 1'b0;
        #20;
        Fun3 = 3'b011;Fun7 = 1'b0;
        #20;
        Fun3 = 3'b101;Fun7 = 1'b0;
        #20;
        Fun3 = 3'b101;Fun7 = 1'b1;
        #20;
        Fun3 = 3'b100;Fun7 = 1'b0;
        #20;
        Fun3 = 3'b111;Fun7 = 1'b1;
        #20;
        OPcode = 5'b00000;Fun3 = 3'b111;Fun7 = 1'b1;
        #20;
        OPcode = 5'b01000;Fun3 = 3'b111;Fun7 = 1'b1;
        #20;
        OPcode = 5'b11000;Fun3 = 3'b000;Fun7 = 1'b1;
        #20;
        OPcode = 5'b11000;Fun3 = 3'b001;Fun7 = 1'b1;
        #20;
        OPcode = 5'b01101;Fun3 = 3'b001;Fun7 = 1'b1;
```

```
        #20;
        OPcode = 5'b11001;Fun3 = 3'b001;Fun7 = 1'b1;
        #20;
        OPcode = 5'b11011;Fun3 = 3'b001;Fun7 = 1'b1;
        #20;
        OPcode = 5'b00100;Fun3 = 3'b000;Fun7 = 1'b1;
        #20;
        OPcode = 5'b00100;Fun3 = 3'b010;Fun7 = 1'b1;
        #20;
        OPcode = 5'b00100;Fun3 = 3'b011;Fun7 = 1'b1;
        #20;
        OPcode = 5'b00100;Fun3 = 3'b111;Fun7 = 1'b1;
        #20;
        OPcode = 5'b00100;Fun3 = 3'b110;Fun7 = 1'b1;
        #20;
        OPcode = 5'b00100;Fun3 = 3'b101;Fun7 = 1'b0;
        #20;
        OPcode = 5'b00100;Fun3 = 3'b101;Fun7 = 1'b1;
        #20;
        OPcode = 5'b00100;Fun3 = 3'b100;Fun7 = 1'b1;
        #20;
        OPcode = 5'b00100;Fun3 = 3'b001;Fun7 = 1'b1;
        #20;
    end
endmodule
```

7: SCPU Ctrl tb.v

仿真波形图如下：



实验用到的RAM如下所示：

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;

自己设计指令集以及第一轮LOOP循环的实验结果如下表格所示(PC从第一条指令开始时候的初始值为0，在此之后每条指令的位置PC加4)

PC	Machine Code	Basic Code	Original Code	Result
0x0	0x00007293	andi x5 x0 0	main:andi x5, x0, 0x0	# x5 = 0
0x4	0x00007313	andi x6 x0 0	andi x6, x0, 0x0	# x6 = 0
0x8	0x88888137	lui x2 559240	lui x2, 0x88888	# x2 = 0x88888000
0xc	0x00832183	lw x3 8(x6)	lw x3, 0x8(x6)	# x3 = 0x80000000
0x10	0x0032A223	sw x3 4(x5)	sw x3, 0x4(x5)	# mem (1) = 0x80000000
0x14	0x00402083	lw x1 4(x0)	lw x1, 0x4(x0)	# x1 = 0x80000000
0x18	0x01C02383	lw x7 28(x0)	nochange:lw x7, 0x1c(x0)	#x7 = 0x80000000
0x1c	0x00338863	beq x7 x3 16	beq x7, x3, cmd_add	
0x20	0x555550B7	lui x1 349525	lui x1,0x55555	# x1 = 0x55555000
0x24	0x0070A0B3	slt x1 x1 x7	slt x1,x1,x7	# x1 = 0x1;
0x28	0xFE0098E3	bne x1 x0 -16	bne x1,x0,nochange	
0x2c	0x007282B3	add x5 x5 x7	cmd_add:add x5, x5, x7	# x5 = 0x80000000
0x30	0x00230333	add x6 x6 x2	add x6, x6, x2	# x6 = 0x88888000
0x34	0x00531463	bne x6 x5 8	bne x6, x5, cmd_sub	
0x38	0x40000033	sub x0 x0 x0	sub x0,x0,x0	
0x3c	0x40530433	sub x8 x6 x5	cmd_sub: sub x8,x6,x5	# x8 = 0x08888000
0x40	0x405304B3	sub x9 x6 x5	sub x9,x6,x5	#x9 = 0x08888000
0x44	0x0080006F	jal x0 8	jal x0, cmd_and	
0x48	0x00007033	and x0 x0 x0	and x0,x0,x0	
0x4c	0x0072F533	and x10 x5 x7	cmd_and: and x10,x5,x7	#x10 = 0x80000000
0x50	0x00157593	andi x11 x10 1	andi x11,x10,0x1	# x11 = 0x0
0x54	0x00B51463	bne x10 x11 8	bne x10,x11 cmd_or	
0x58	0x00006033	or x0 x0 x0	or x0,x0,x0	
0x5c	0x00A5E5B3	or x11 x11 x10	cmd_or:or x11,x11,x10	# x11 = 0x80000000
0x60	0x0015E513	ori x10 x11 1	ori x10,x11,0x1	# x10 = 0x80000001
0x64	0x00558463	beq x11 x5 8	beq x11,x5,cmd_xor	
0x68	0x00004033	xor x0 x0 x0	xor x0,x0,x0	
0x6c	0x00A5C633	xor x12 x11 x10	cmd_xor: xor x12,x11,x10	# x12 = 0x00000001
0x70	0x00164613	xori x12 x12 1	xori x12,x12,0x1	# x12 = 0x00000000
0x74	0x00B61463	bne x12 x11 8	bne x12,x11,cmd_srl	
0x78	0x00000013	addi x0 x0 0	addi x0,x0,0x0	
0x7c	0x0012D293	srli x5 x5 1	cmd_srl: srli x5,x5,0x1	# x5 = 0x40000000
0x80	0x00060463	beq x12 x0 8	beq x12,x0,cmd_sll	
0x84	0x40000033	sub x0 x0 x0	sub x0,x0,x0	
0x88	0x00129293	slli x5 x5 1	cmd_sll: slli x5,x5,0x1	# x5 = 0x80000000
0x8c	0x00B28463	beq x5 x11 8	beq x5,x11,cmd_slt	
0x90	0x00000013	addi x0 x0 0	addi x0,x0,0x0	
0x94	0x001026B3	slt x13 x0 x1	cmd_slt: slt x13,x0,x1	# x13 = 0x0
0x98	0x00503733	sltu x14 x0 x5	sltu x14,x0,x5	# x14 = 0x1
0x9c	0xF65FF06F	jal x0 -156	jal x0,main	

指令集以及输出信号的实验结果如下表格所示

pc	alu_res	mem_wen	dmem_o_data	dmem_i_data	dmem_addr	
0	00000000	0	f0000000	00000000	00000000	
4	00000000	0	f0000000	00000000	00000000	
8	08888000	0	f0000000	08888000	08888000	
c	00000008	0	80000000	08888000	00000008	
10	00000004	1	80000000	80000000	00000004	
14	00000004	0	80000000	00000000	00000004	
18	0000001c	0	80000000	00000000	0000001c	
1c	00000000	0	f0000000	80000000	00000000	goto 2c
20						
24						
28						
2c	80000000	0	f0000000	80000000	80000000	
30	88888000	0	f0000000	88888000	88888000	
34	08888000	0	f0000000	80000000	08888000	goto 3c
38						
3c	08888000	0	f0000000	80000000	08888000	
40	08888000	0	f0000000	80000000	08888000	
44	00000008	0	80000000	08888000	00000008	goto 4c
48						
4c	80000000	0	f0000000	80000000	80000000	
50	00000000	0	f0000000	80000000	00000000	
54	80000000	0	f0000000	00000000	80000000	goto 5c
58						
5c	80000000	0	f0000000	80000000	80000000	
60	80000001	0	f0000000	80000000	80000001	
64	00000000	0	f0000000	80000000	00000000	goto 6c
68						
6c	00000001	0	f0000000	80000001	00000001	
70	00000000	0	f0000000	80000000	00000000	
74	80000000	0	f0000000	80000000	80000000	goto 7c
78						
7c	40000000	0	f0000000	80000000	40000000	
80	00000000	0	f0000000	00000000	00000000	goto 88
84						
88	80000000	0	f0000000	80000000	80000000	
8c	00000000	0	f0000000	80000000	00000000	goto 94
90						
94	00000000	0	f0000000	80000000	00000000	
98	00000001	0	f0000000	80000000	00000001	
9c	ffffff64	0	f0000000	80000000	ffffff64	goto 0

测试程序以及实验结果表明，本实验成功验证了如下指令的正确性：

- R-Type add, sub, and, or, xor, slt sltu ,srl, sra, sll
- I-Type addi, andi, ori, xori, slti, sltiu, srai, slli, jalr
- S-Type sw
- B-Type beq, bne
- J-Type Jal
- U-Type lui

6 实验讨论、心得

部分思考题回答如下：

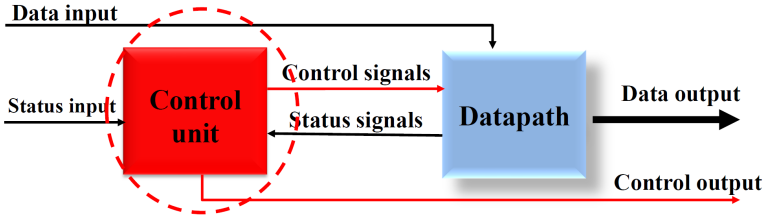
- 设计 bne 指令需要增加控制信号吗: 需要增加branchN信号，为0时候选择PC+4 地址，为1时候选择转移目的地址PC+imm
- 设计 srai 时需要增加新的数据通道吗：不需要
- 指令扩展时控制器用二级译码设计存在什么问题:每一条指令 decode 时间不一样长

7 附录

一些正文里面放不下的原理图：

CPU organization

- **Digital circuit**
  - General circuits that controls logical event with logical gates --Hardware

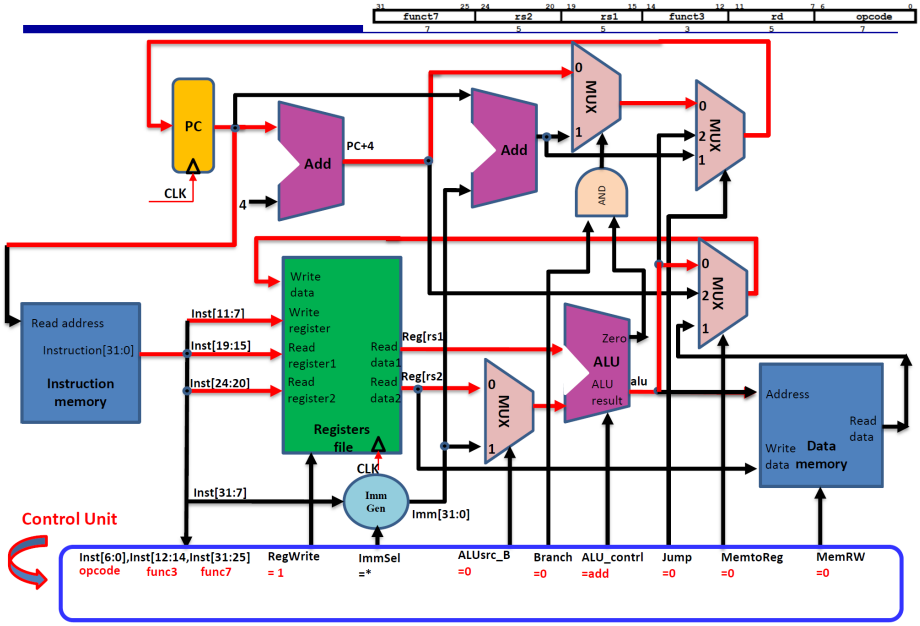


- **Computer organization**
  - Special circuits that processes logical action with instructions --Software

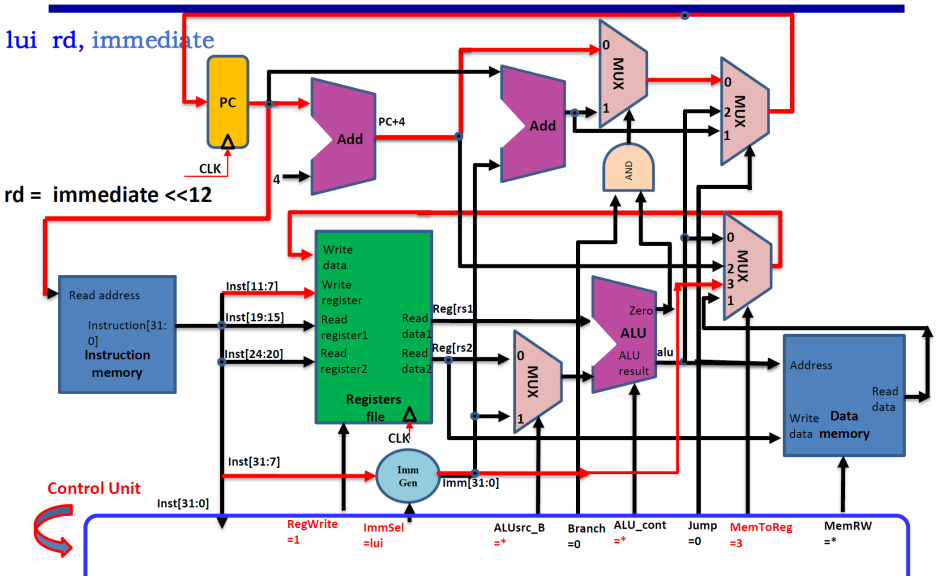
ImmSel

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]	001
	0010011	Addi;slti;sltiu;xori;ori;andi;slli;srai		
	1100111	jalr		
S-type	0100011	Sw;sb;sh	(sign-extend) instr[31:25],[11:7]	010
B-type	1100011	Beq;bne;blt;bge;bltu;bgeu	(sign-extend) instr[31],[7],[30:25],[11:8], 1'b0	011
J-type	1101111	jal	(sign-extend) instr[31],[19:12],[20],[30:21],1'b0	100
U-type	0010111	auipc	instr[31:12],12'h000	000
	0110111	lui		

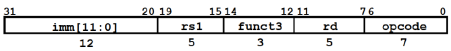
Control unit---add

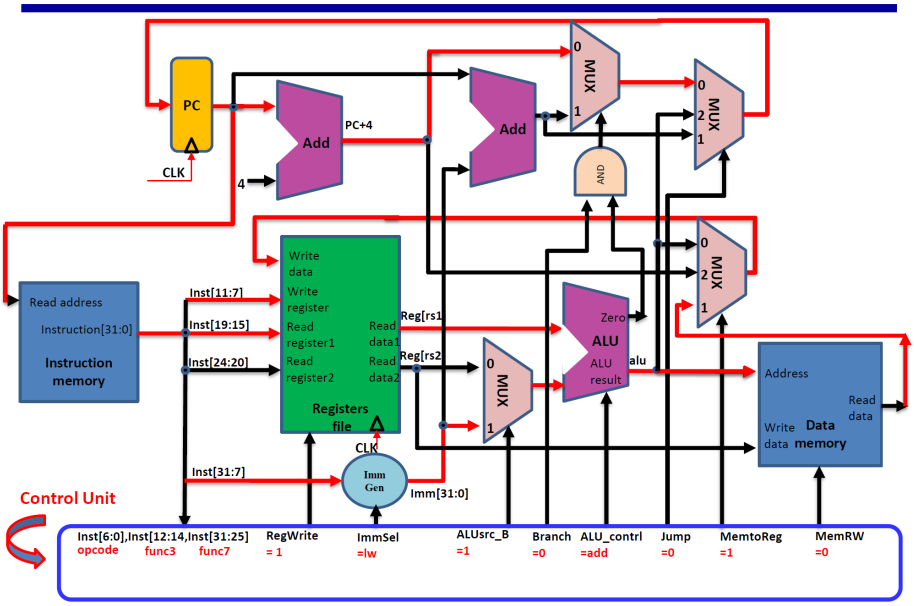


Control unit-- LUI

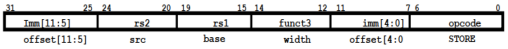


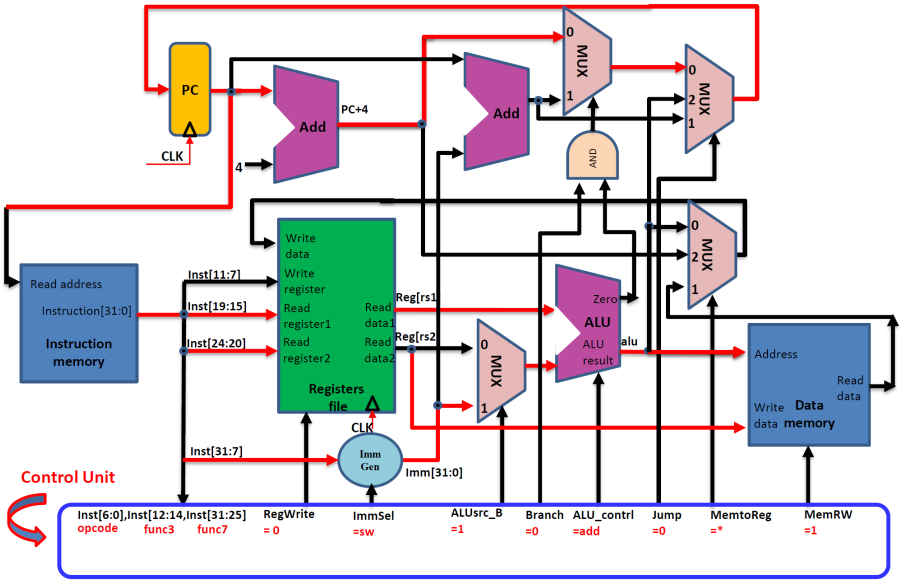
Control unit---lw



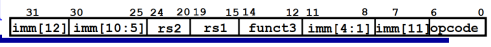


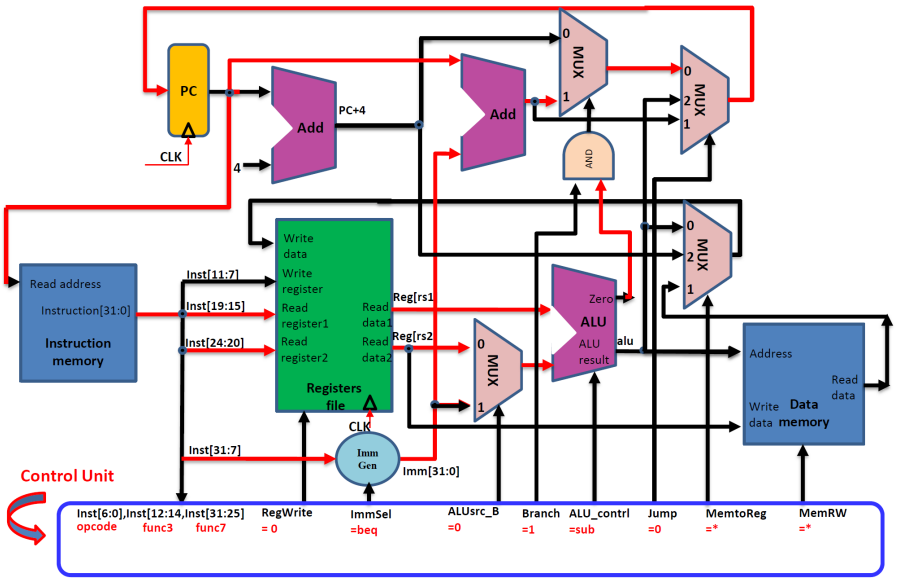
Control unit---sw



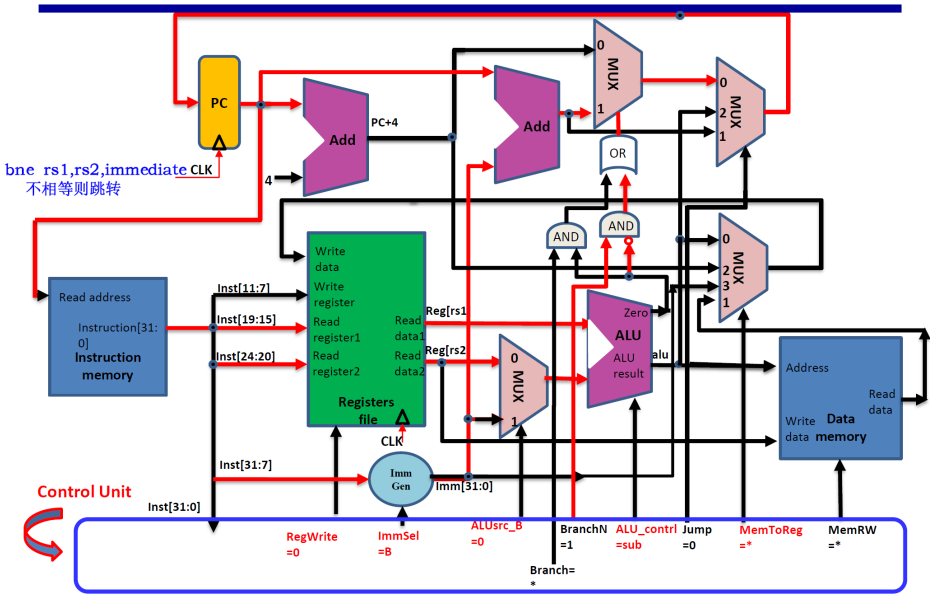


Control unit---beq

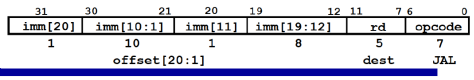




Control unit-- bne



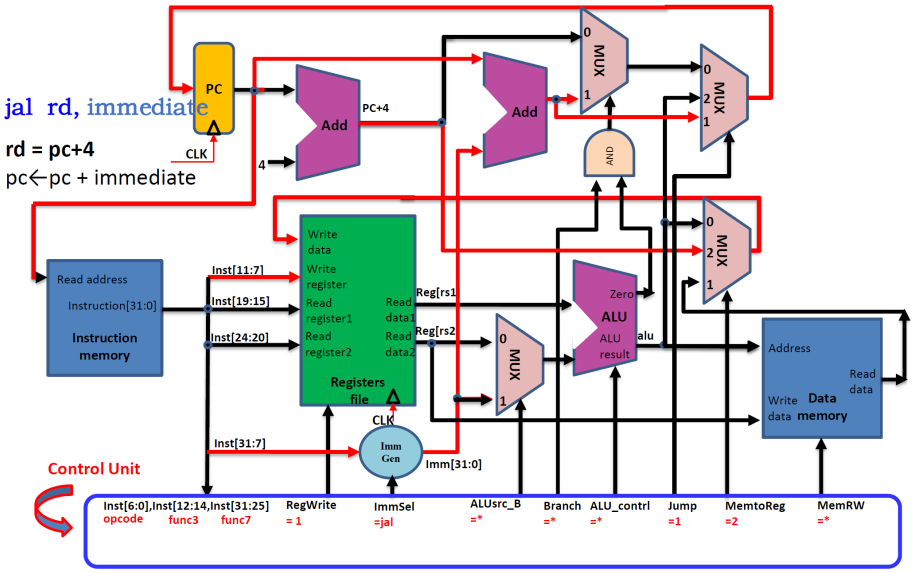
Control unit---jal



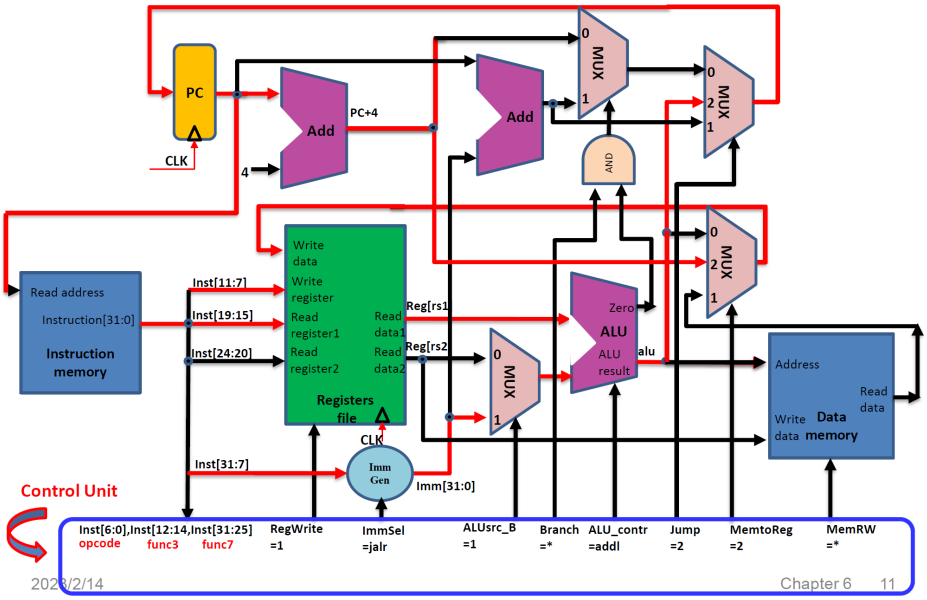
jal rd, immediate

rd = pc+4

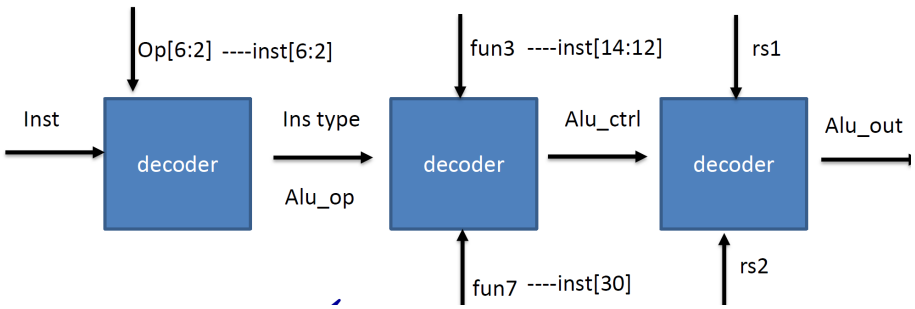
pc←pc + immediate



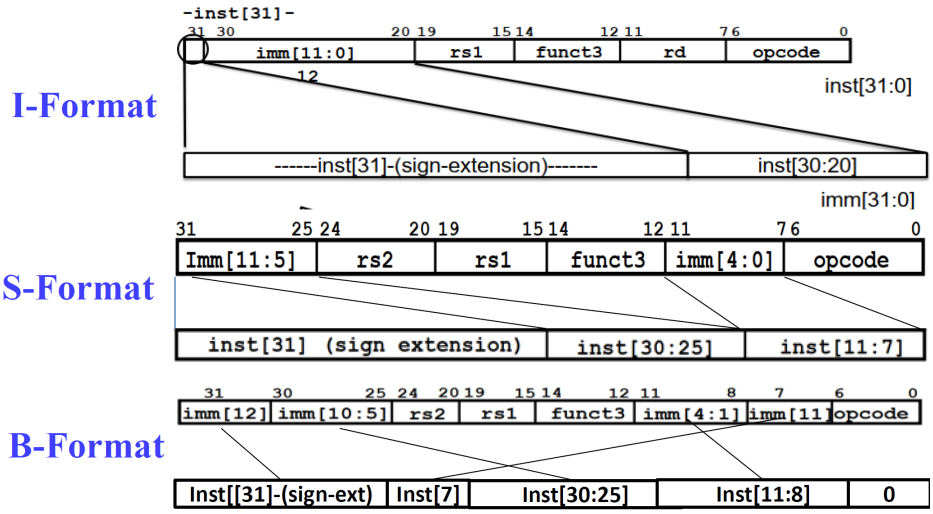
Control unit---jalr



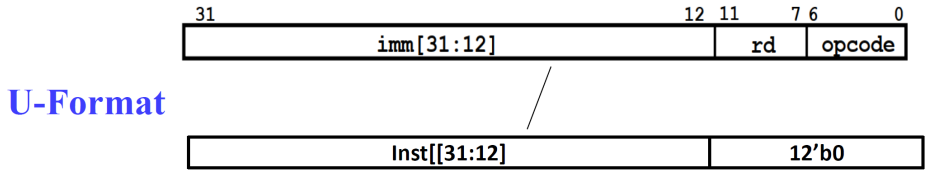
ALU操作译码----多级译码

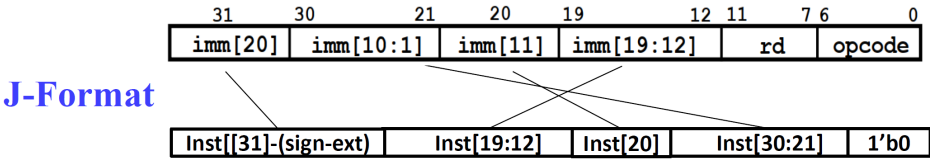


ImmSel

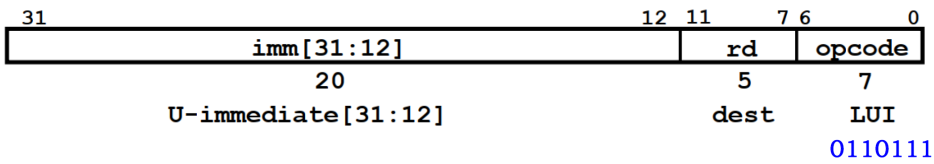


ImmSel





Adding U-Format



lui rd, immediate

功能：rd = immediate <<12

Eg: lui x5,0x12345

X5 = 0x12345000　　取左移12位的20位立即数