浙江大学

计算机组成实验报告

实验名称：	单周期CPU设计（不含指令扩展）
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 4 月 29 日

单周期CPU设计（不含指令扩展）

1 实验目的和要求

- 复习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
- 设计数据通路的功能部件
- 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

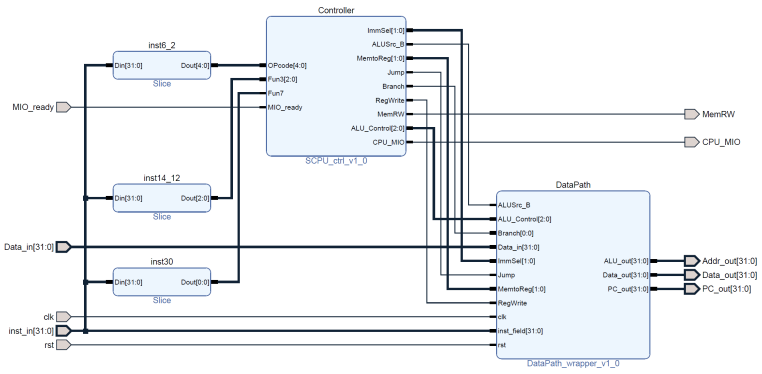
2 实验内容和原理

2.1 实验任务

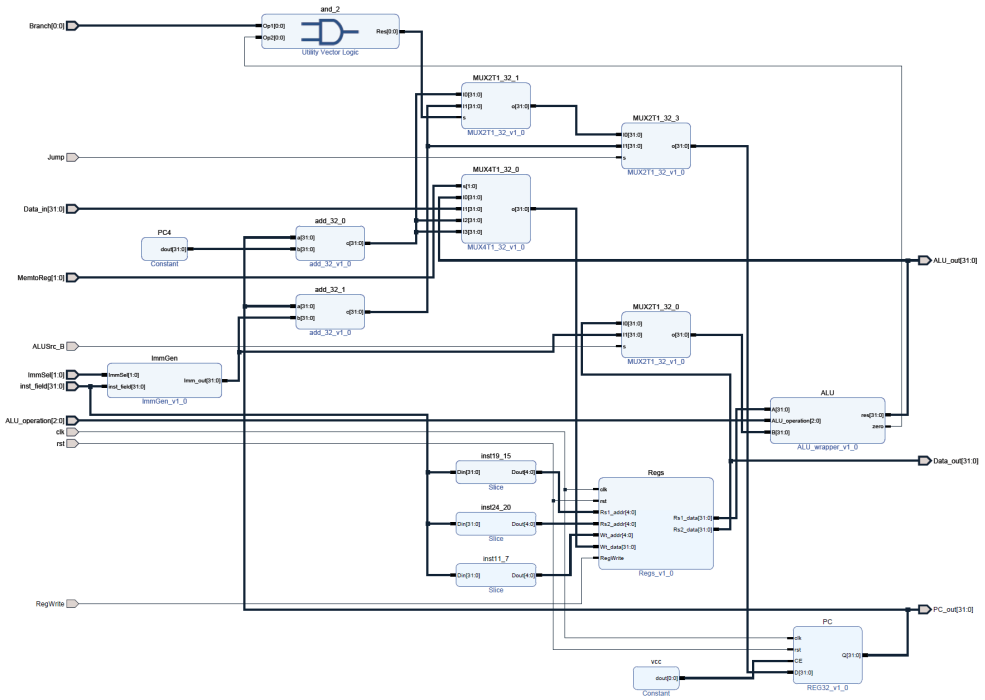
- RTL 代码描述 CPU 设计 SCPU.v
- 设计实现数据通路采用 RTL 实现
- ImmGen立即数生成模块设计采用 RTL 实现
- PC寄存器设计及 PC 通路建立
- 熟练掌握 IP 核的使用方法
- 用硬件描述语言设计实现控制器,设计控制器测试方案并完成测试(译码测试： R格式、访存指令、分支指令，转移指令)

2.2 实验原理

SCPU 原理图如下所示



datapath 原理图如下所示

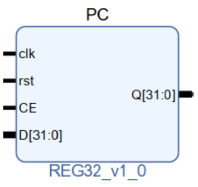


本实验里面，为了实现功能与调试bug时候方便，没有采用单独的把ALU模块从wrapper后调用到本实验，而是直接用Verilog代码的形式自己重新实现了一份，具体见后面的代码。

pc 设计如下图所示

PC寄存器设计及PC通路建立

- **设计32位寄存器**
  - 用途：PC指针、数据、地址或指令锁存
  - 采用行为描述实线
  - 模块名：REG32
    - 上升沿触发：clk
    - 使能信号：CE
    - 同步复位：rst=1
    - 数据输入：D(31:0)
    - 数据输出：Q(31:0)
  - 参考描述结构




```
module  REG32(input clk,
                ……
      always @(posedge clk or posedge rst)
                if (rst==？ ) Q <= ？ ;
                else if (？ )  Q <= ？ ;
endmodule
```

本实验里面，为了实现功能方便，没有采用单独的PC模块，而是将其融合到了SCPUCTRL模块里面，具体见后面的代码。

控制信号设计如下图所示

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（详见实验4-2）		
ImmSel	00-11	2位立即数组合控制	参考表ImmSel（详见实验4-2）		

控制信号真值表如下图所示

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

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lab2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 [1:0] ImmSel;
wire [1:0] MemtoReg;
wire [2:0] ALU_Control;
wire Jump,Branch,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),
.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),
.Data_in(Data_in),
.ImmSel(ImmSel),
.Jump(Jump),
.MemtoReg(MemtoReg),
.RegWrite(RegWrite),
.clk(clk),
.rst(rst),
.inst_field(inst_in),
.ALU_out(Addr_out),
.Data_out(Data_out),
.PC_out(PC_out)
);
endmodule
```

datapath 实现代码如下：

```
`timescale 1ns / 1ps
module DataPath(
input wire clk,
input wire rst,
input wire Jump,
```

```
    input wire Branch,
    input wire [1:0] MemtoReg,
    input wire ALUSrc_B,
    input wire RegWrite,
    input wire [1:0] ImmSel,
    input wire [2:0] 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 [31:0] Imm_out;
    ImmGen myImmGen(
    .ImmSel(ImmSel),
    .inst_field(inst_field),
    .Imm_out(Imm_out)
    );
    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)//jal
    begin
    PC_out <= PC_jump;
    end
    else if(Branch)//neq
    begin
    PC_out <= (rs_rt_equal) ? PC_jump : PC_next;
    end
    else
    begin
    PC_out <= PC_next;
    end
    end
    wire [4:0] Rs1_addr;
    wire [4:0] Rs2_addr;
    wire [4:0] Wt_addr;
    assign Rs1_addr = inst_field[19:15];
    assign Rs2_addr = inst_field[24:20];
    assign Wt_addr = inst_field[11:7];
    reg [31:0] Wt_data;
    wire [31:0] Rs1_data;
    wire [31:0] Rs2_data;
    always @(*) begin
    case(MemtoReg)
    2'b00: Wt_data = ALU_out;
    2'b01: Wt_data = Data_in;
    2'b10: Wt_data = PC_next;
    2'b11: Wt_data = PC_next;
    endcase
    end
    regs myRegs(
```

```
    .clk(clk),
    .rst(rst),
    .Rs1_addr(Rs1_addr),
    .Rs2_addr(Rs2_addr),
    .Wt_addr(Wt_addr),
    .Rs1_data(Rs1_data),
    .Rs2_data(Rs2_data),
    .Wt_data(Wt_data),
    .RegWrite(RegWrite)
    );
    assign rs_rt_equal = (Rs1_data == Rs2_data) ? 1 : 0;
    reg [31:0] alu_a;
    reg [31:0] alu_b;
    wire zero;
    wire [31:0] alu_out;
    always @(*) begin
    alu_a = Rs1_data;
    case(ALUSrc_B)
    1'b0: alu_b = Rs2_data;
    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)
    );
    always @(*) begin
    Data_out = Rs2_addr;
    ALU_out = alu_out;
    end
    endmodule
```

ImmGen 实现代码如下：

```
    `timescale 1ns / 1ps
    module ImmGen(
    input wire [1:0] ImmSel,
    input wire [31:0] inst_field,
    output reg [31:0] Imm_out
    );
    always @(*) begin
    case (ImmSel)
    2'b00 : Imm_out = { {20 {inst_field[31]} }, inst_field[31:20]};//addi/lw(I)
    2'b01 : Imm_out = { {20 {inst_field[31]} }, inst_field[31:25], inst_field[11:7]};//sw(s)
    2'b10 : Imm_out = { {20 {inst_field[31]} }, inst_field[7], inst_field[30:25], inst_field
        [11:8], 1'b0};//beq(sb)
    2'b11 : Imm_out = { {12 {inst_field[31]} }, inst_field[19:12], inst_field[20],
        inst_field[30:21], 1'b0};//jal(uj)
    endcase
    end
    endmodule
```

SCPU Ctrl 实现代码如下：

```
module SCPU_ctrl(
input [4:0] OPcode,
input [2:0] Fun3,
input Fun7,
input MIO_ready,
output reg [1:0] ImmSel,
output reg ALUSrc_B,
output reg [1:0] MemtoReg,
output reg Jump,
output reg Branch,
output reg RegWrite,
output reg MemRW,
output reg [2:0] ALU_Control,
output reg CPU_MIO
);
reg [1:0] ALUop;
`define CPU_ctrl_signals {ALUSrc_B,MemtoReg,RegWrite,MemRW,Branch,Jump,ALUop,ImmSel}
always @*
begin
case(OPcode)//ALUop modified to ppt
5'b01100: begin `CPU_ctrl_signals = {1'b0,2'b00,1'b1,1'b0,1'b0,1'b0,2'b10,2'b00}; end //ALU(R)
5'b00000: begin `CPU_ctrl_signals = {1'b1,2'b01,1'b1,1'b0,1'b0,1'b0,2'b00,2'b00}; end //load(I)
5'b01000: begin `CPU_ctrl_signals = {1'b1,2'b00,1'b0,1'b1,1'b0,1'b0,2'b00,2'b01}; end //store(S)
5'b11000: begin `CPU_ctrl_signals = {1'b0,2'b00,1'b0,1'b0,1'b1,1'b0,2'b01,2'b10}; end //beq(B)
5'b11011: begin `CPU_ctrl_signals = {1'b0,2'b10,1'b1,1'b0,1'b0,1'b1,2'b00,2'b11}; end //jump, ALUop = xx
5'b00100: begin `CPU_ctrl_signals = {1'b1,2'b00,1'b1,1'b0,1'b0,1'b0,2'b11,2'b00}; end //imm(I)
endcase
end
wire [3:0] Fun;
assign Fun = {Fun3,Fun7};
always @(*)
begin//bug
case(ALUop)
2'b00: ALU_Control = 3'b010 ;//add address for load and store and jump
2'b01: ALU_Control = 3'b110 ;//sub for beq
2'b10://ALU
case(Fun)
4'b0000: ALU_Control = 3'b010 ;//add
4'b0001: ALU_Control = 3'b110 ;//sub
4'b1110: ALU_Control = 3'b000 ;//and
4'b1100: ALU_Control = 3'b001 ;//or
4'b0100: ALU_Control = 3'b111 ;//slt
4'b1010: ALU_Control = 3'b101 ;//srl
4'b1000: ALU_Control = 3'b011 ;//xor
default: ALU_Control = 3'bx;
endcase
2'b11://imm
case(OPcode)
5'b00000: ALU_Control = 3'b010; //load
5'b00100://imm
case(Fun3)
3'b010: ALU_Control=3'b111;//slt
```

```
    3'b000: ALU_Control=3'b010;//add
    3'b100: ALU_Control=3'b011;//xor
    3'b110: ALU_Control=3'b001;//or
    3'b111: ALU_Control=3'b000;//and
    3'b101: ALU_Control=3'b101;//srl
    default: ALU_Control=3'bx;
    endcase
    default: ALU_Control=3'bx;
    endcase
    endcase
    end
    endmodule
```

ALU 实现代码如下：

```
    `timescale 1ns / 1ps
    module ALU(
    input [31:0] A,
    input [31:0] B,
    input [2:0] ALU_operation,
    output reg[31:0] res,
    output zero
    );
    parameter one = 32'h00000001;
    parameter zero_0 = 32'h00000000;
    always @(*) begin
    case (ALU_operation)
    3'b000: res=A&B;//and
    3'b001: res=A|B;//or
    3'b010: res=A+B;//add
    3'b011: res=A^B;//xor
    3'b110: res=A-B;//sub
    3'b101: res=B>>1; // srl
    3'b111: res=($signed(A) < $signed(B)) ? one : zero_0;//slt
    default: res=32'h00000000;
    endcase
    end
    assign zero = (res==0)? 1: 0;
    endmodule
```

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
```

```
14      always @(posedge clk or posedge rst )
        begin if (rst ==1)
16      for (i=1; i<32; i=i+1) register[i] <= 0; // reset
        else if ( ( Wt_addr != 0) && (RegWrite == 1 ) )
18      register[Wt_addr] <= Wt_data ; // write
        end
20      endmodule
```

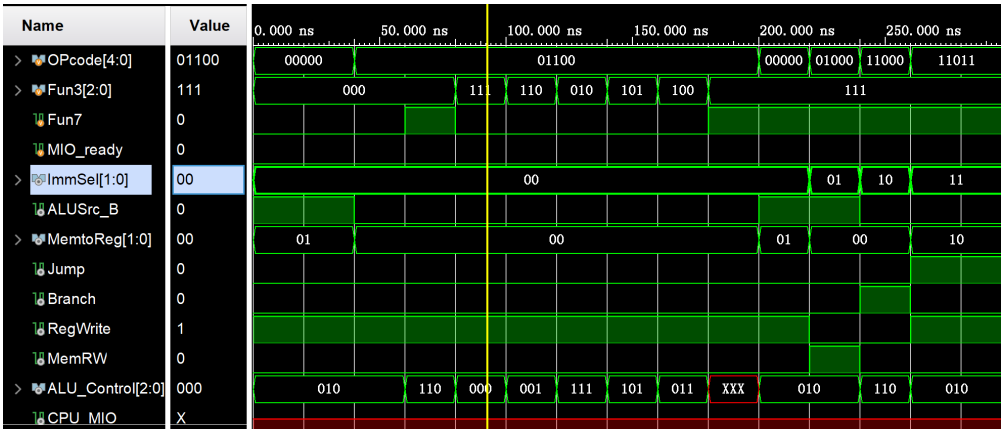
5 实验结果与分析

SCPU CTRL 仿真激励代码如下：

```
1       module SCPU_ctrl_tb();
        reg[4:0]OPcode; //OPcode
3       reg[2:0]Fun3; //Function
        reg Fun7; //Function
5       reg MIO_ready; //CPU Wait
        wire [1:0]ImmSel;
7       wire ALUSrc_B;
        wire [1:0]MemtoReg;
9       wire Jump;
        wire Branch;
11      wire RegWrite;
        wire MemRW;
13      wire [2:0]ALU_Control;
        wire CPU_MIO;
15      SCPU_ctrl SCPU_ctrl_U(
        .OPcode (OPcode ),
17      .Fun3 (Fun3 ),
        .Fun7 (Fun7 ),
19      .MIO_ready (MIO_ready ),
        .ImmSel (ImmSel ),
21      .ALUSrc_B (ALUSrc_B ),
        .MemtoReg (MemtoReg ),
23      .Jump (Jump ),
        .Branch (Branch ),
25      .RegWrite (RegWrite ),
        .MemRW (MemRW ),
27      .ALU_Control (ALU_Control),
        .CPU_MIO (CPU_MIO )
29      );
        initial begin
31      OPcode = 0;
        Fun3 = 0;
33      Fun7 = 0;
        MIO_ready = 0;
35      #40;
        OPcode = 5'b01100; //R
37      Fun3 = 3'b000;Fun7 = 1'b0; //add
        #20;
39      Fun3 = 3'b000;Fun7 = 1'b1; //sub
        #20;
41      Fun3 = 3'b111;Fun7 = 1'b0; //and
        #20;
43      Fun3 = 3'b110;Fun7 = 1'b0; //or
```

```
    #20;
    Fun3 = 3'b010;Fun7 = 1'b0; //slt
    #20;
    Fun3 = 3'b101;Fun7 = 1'b0; //srl
    #20;
    Fun3 = 3'b100;Fun7 = 1'b0; //xor
    #20;
    Fun3 = 3'b111;Fun7 = 1'b1;
    #20;
    OPcode = 5'b00000; //load
    #20;
    OPcode = 5'b01000; //store
    #20;
    OPcode = 5'b11000; //beq
    #20;
    OPcode = 5'b11011; //jal
    #40;
    OPcode = 5'b00100; //I
    Fun3 = 3'b000; //addi
    #20;
    Fun3 = 3'b111; //andi
    #20;
    Fun3 = 3'b110; //ori
    #20;
    Fun3 = 3'b010; //slti
    #20;
    Fun3 = 3'b101; //srli
    #20;
    Fun3 = 3'b100; //xori
    #20
    OPcode = 5'h1f;
    Fun3 = 3'b000;
    Fun7 = 1'b0;
    end
    endmodule
```

仿真波形图如下：



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)

表 1: 指令集以及实验结果

指令	机器码	ALU输出(16进制)	评注
LOOP: addi x18,x0,1	0x00100913	00000001	
add x19,x18,x18	0x012909B3	00000002	
sub x20,x18,x19	0x41390A33	FFFFFFFF	
sub x20,x20,x19	0x413A0A33	FFFFFFFD	
add x21,x19,x18	0x01298AB3	00000003	
lw x20,0x34(x0)	0x03402A03	00000034	x20=0x55555555
xor x21,x21,x20	0x014ACAB3	55555556	
xori x21,x20,31	0x01FA4A93	5555554A	
or x21,x21,x20	0x014AEAB3	5555555F	
ori x21,x21,124	0x07CAEA93	5555557F	
and x21,x21,x20	0x014AFAB3	55555555	
andi x21,x21,21	0x015AFA93	00000015	
slt x23,x20,x21	0x015A2BB3	00000000	x20>x21
sw x20,0x34(x0)	0x03402A23	00000034	x20=0x55555555
add x23,x23,x0	0x000B8BB3	00000000	
slti x23,x21,5	0x005AAB93	00000000	x21>5
add x23,x23,x0	0x000B8BB3	00000000	
beq x23,x23,SKIP	0x017B8463	00000000	
add x21,x21,x21	0x015A8AB3		skipped
SKIP: add x21,x21,x0	0x000A8AB3	00000015	
jal LOOP	0xFB1FF0EF		回到开始时候的LOOP

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;

以上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, srl
- I-Type addi, andi, ori, xori, slti, srli, lw
- S-Type sw
- B-Type beq
- J-Type Jal

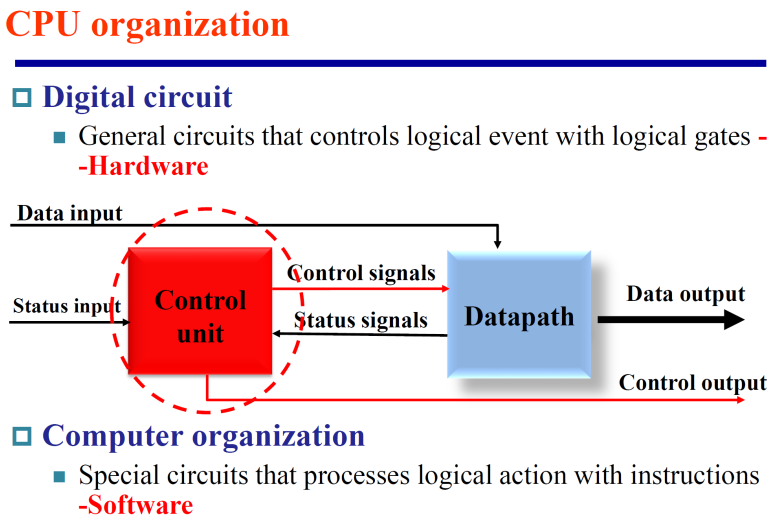
6 实验讨论、心得

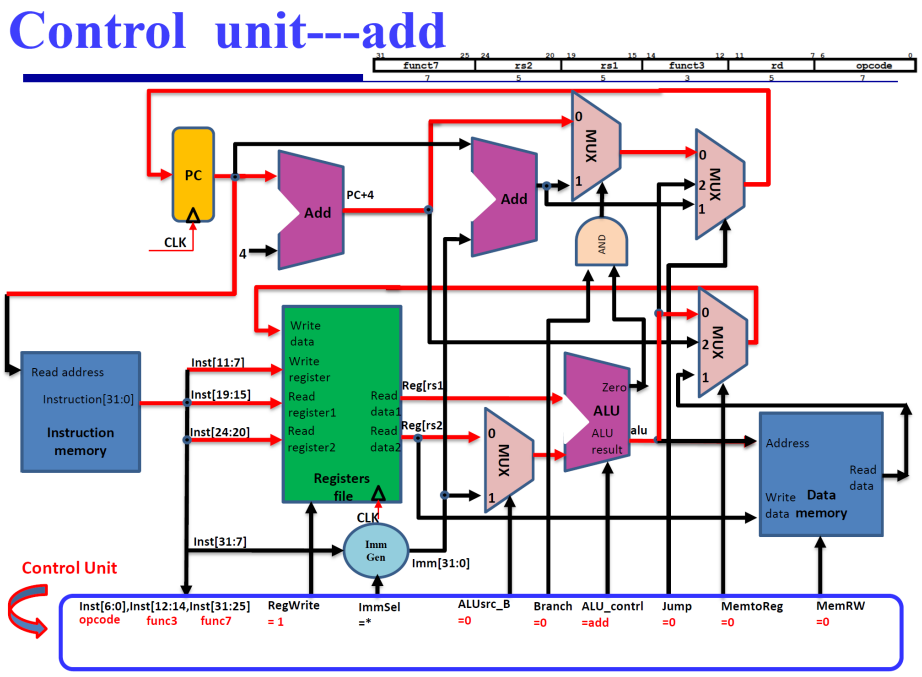
部分思考题回答如下：

- 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
- 单周期控制器时序体现在那里：每次只取一条指令，在执行完一条指令之前不会执行下一条指令的instruction fetch操作

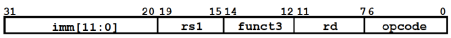
7 附录

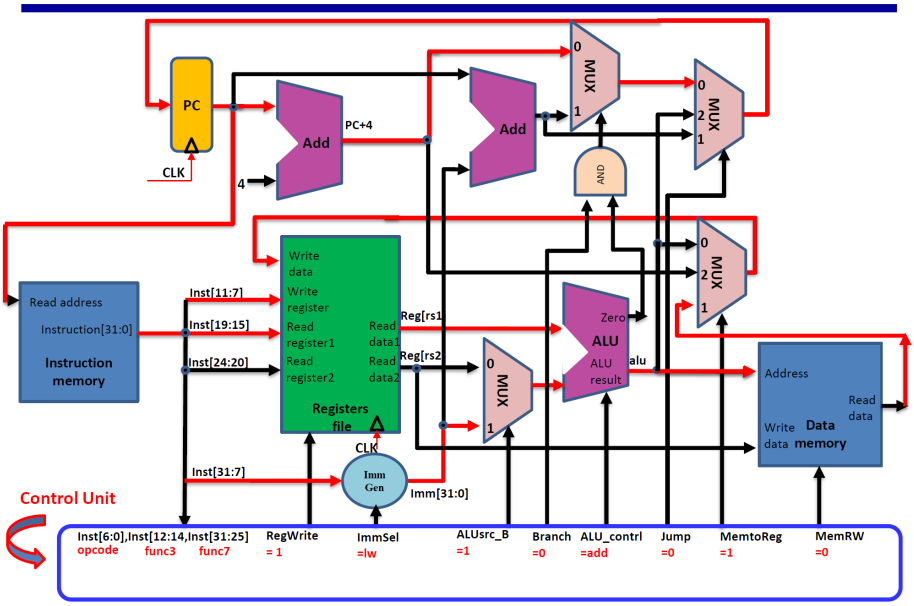
一些正文里面放不下的原理图：



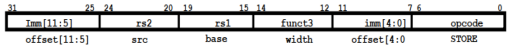


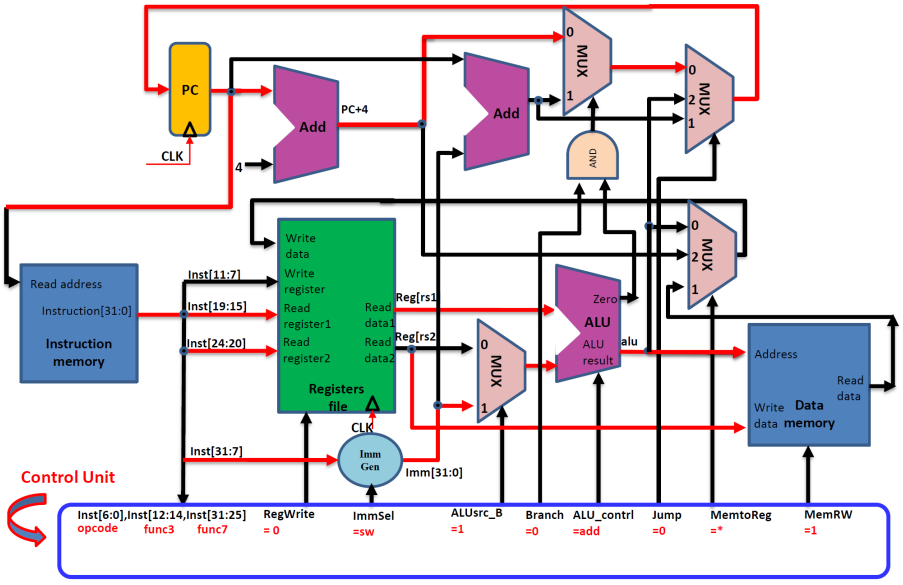
Control unit---lw



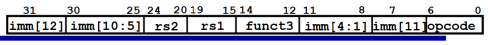


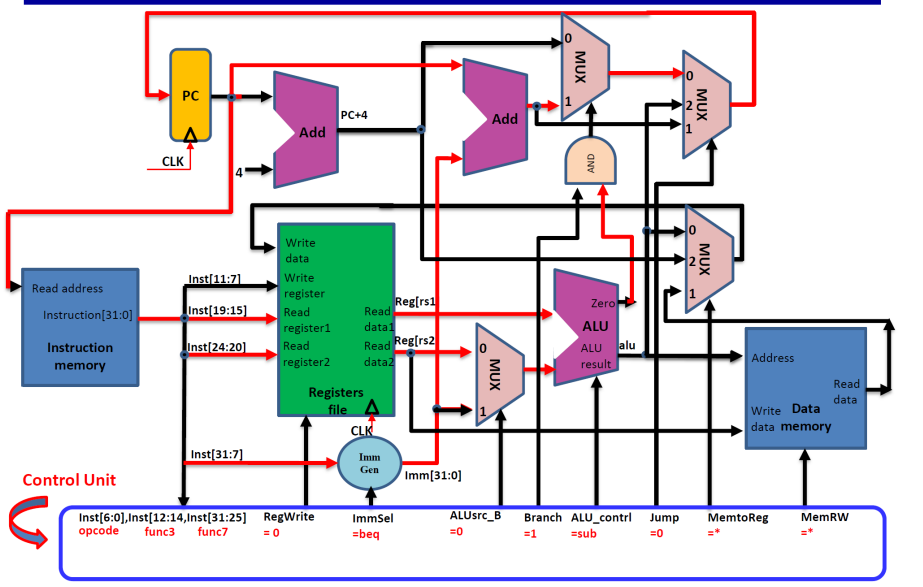
Control unit---sw



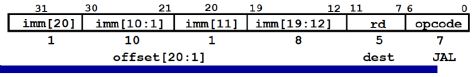


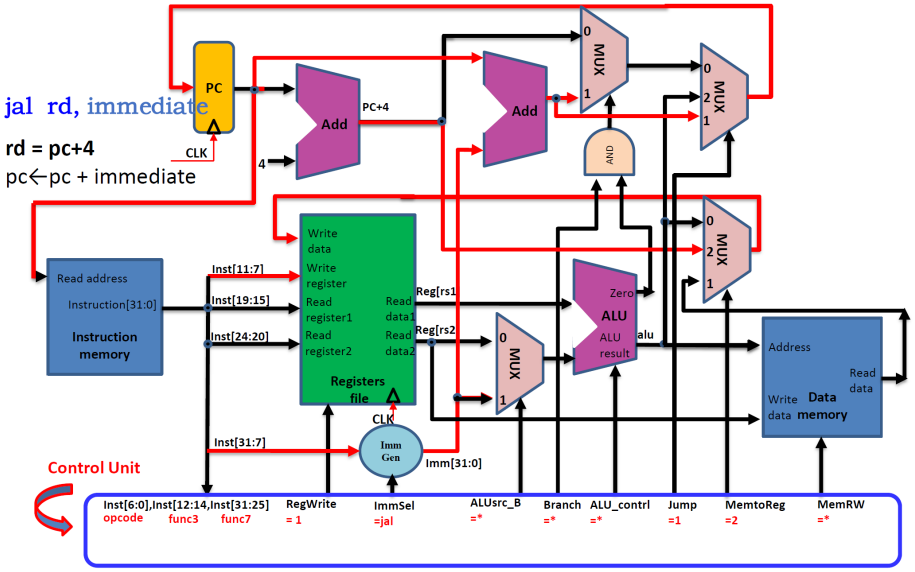
Control unit---beq



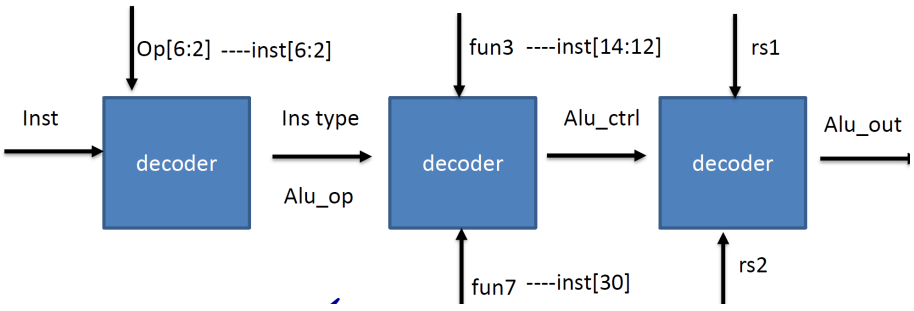


Control unit---jal

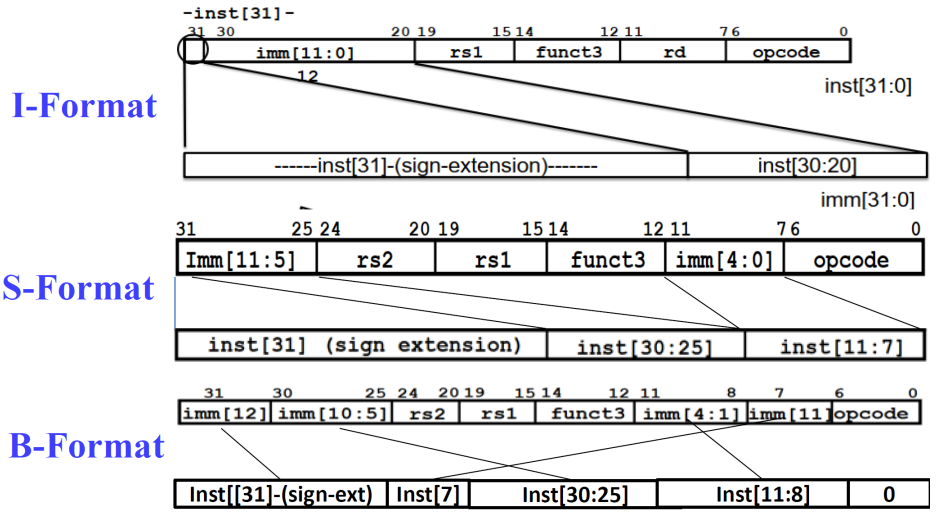




ALU操作译码----多级译码



ImmSel



ImmSel