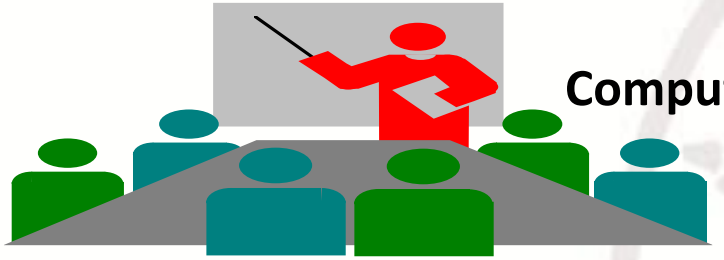


Computer Organization & Design
实验与课程设计

实验五

CPU设计-数据通路

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：数据通路组成与原理

3. 设计数据通路

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

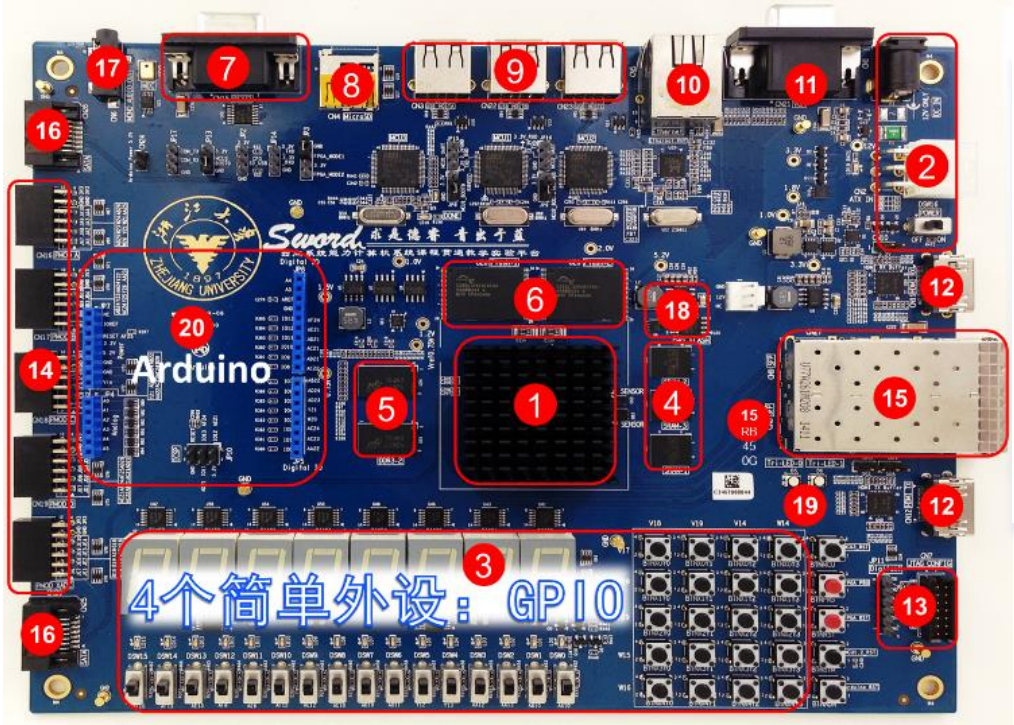
2.计算机软硬件课程贯通教学实验系统(Sword)

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

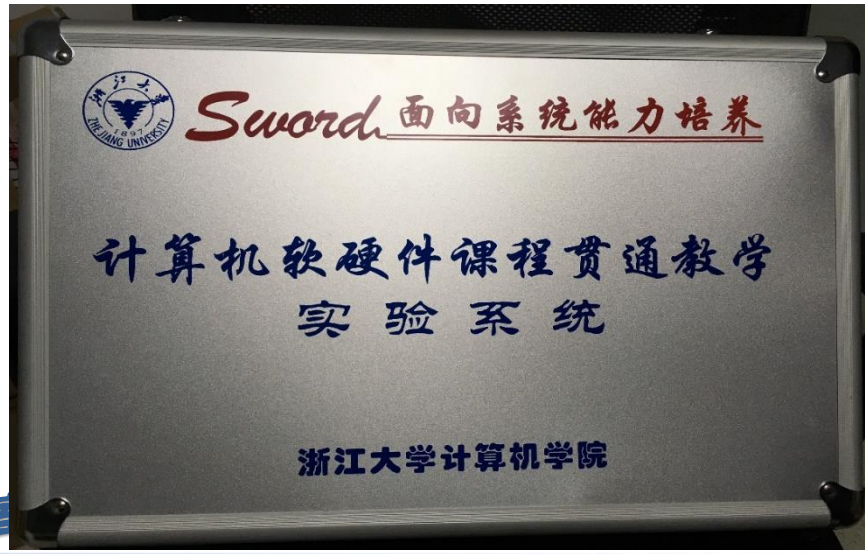
12位VGA接口（RGB656）、USB-HID（键盘）

▼ **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+条指令的数据通路

- 用逻辑原理图设计实现数据通路
  - ALU和Regs调用Exp04设计的模块
- 替换Exp04的数据通路核
- 此实验在Exp04的基础上完成

2. 设计数据通路测试方案：

- 部件测试： ALU、Register Files
- 通路测试：
  - R-格式通路、I-格式通路(LW)、S-格式通路
  - SB-格式通路、UJ-格式通路

3. 设计数据通路测试程序

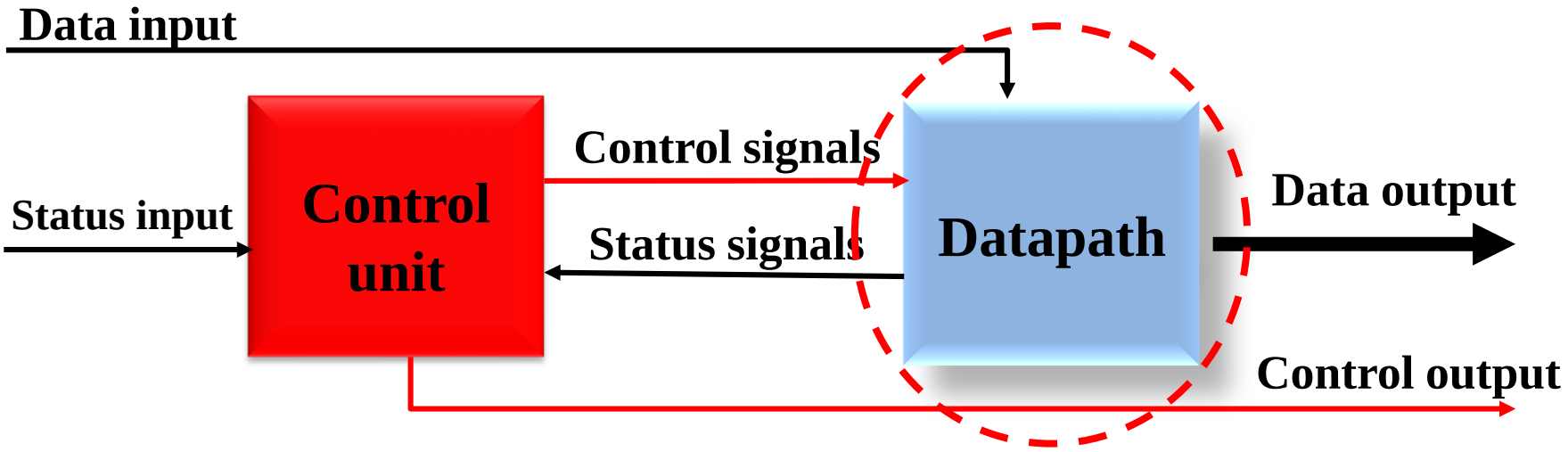
Course Outline

- 实验目的与实验环境
- 实验任务
- 实验原理(详细参见后面附录)
- 实验操作与实现

CPU organization

Digital circuit

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

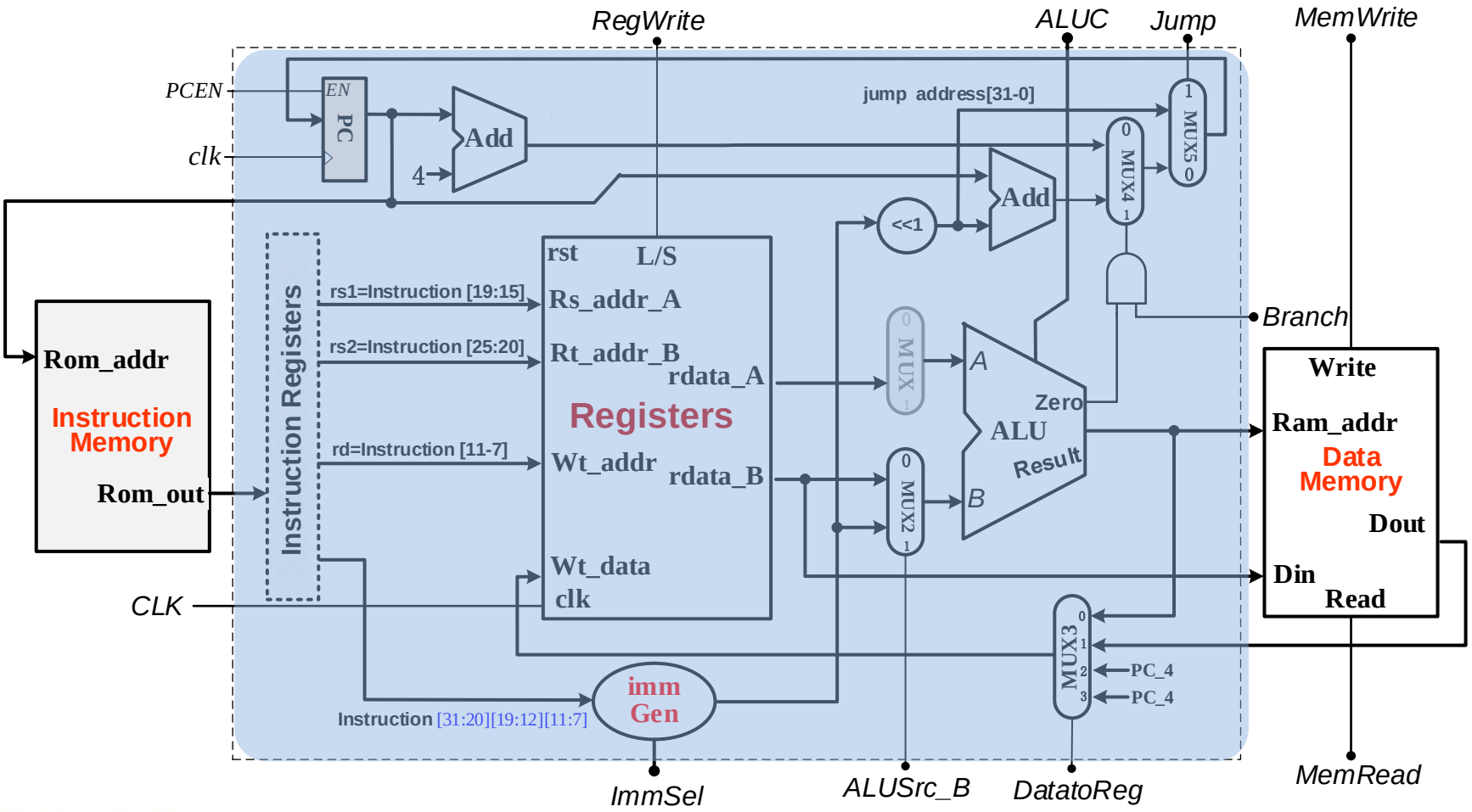


Computer organization

- Special circuits that processes logical action with instructions -Software

单周期数据通路结构

- 找出九条指令的通路：5个MUX



控制信号定义

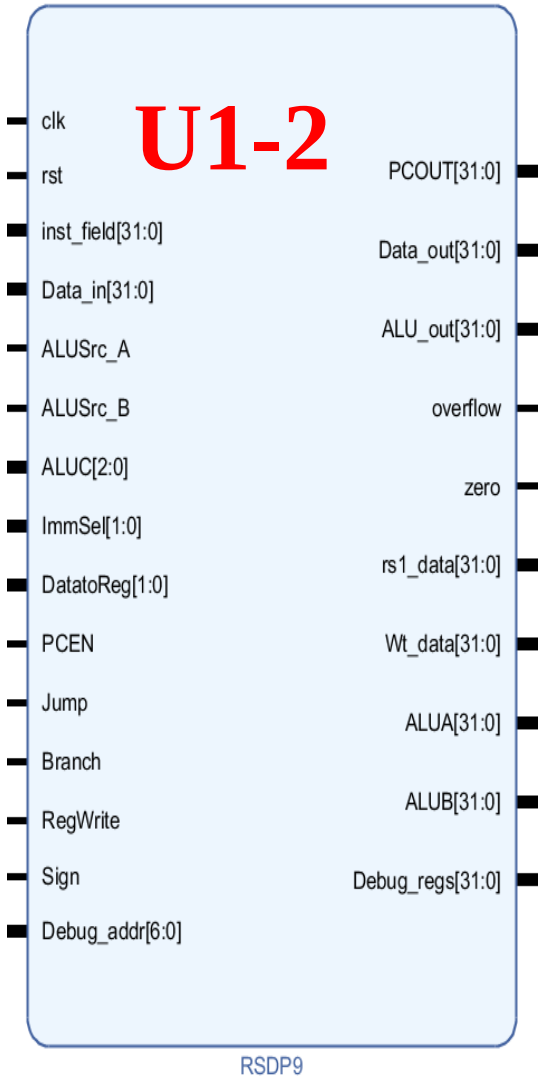
- 通路与操作控制

信号	源数目	功能定义	赋值0时动作	赋值1时动作
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）
ImmSel	4	立即数选择	选择指令立即数：= 00/01/10/11	
DatatoReg	3	寄存器写入数据选择	选择：存储器/ALU/PC+4	
Branch	2	Beq指令目标地址选择	选择PC+4地址	选择转移地址（Zero=1）
Jump	2	J指令目标地址选择	由Bran	地址
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
PCEN	-	控制PC指针修改	禁止PC寄存器写	使能PC寄存器写
ALU Control	000-111	三位ALU操作控制	参考表 Lab3	Lab3

请填写对应操作

CPU部件之数据通路接口：RSDP9

- **Datapath**
  - CPU主要部件之一
  - 寄存器传输控制对象：通用数据通路
- **基本功能**
  - 具有通用计算功能的算术逻辑部件
  - 具有通用目的寄存器
  - 具有通用计数所需的尽可能的路径
- **本实验设计RSDP9**
  - 采用HDL结构化描述设计
    - 调用实验1~4部件
  - 替换实验四RSDP9.edf



数据通路接口信号标准：RSDP9

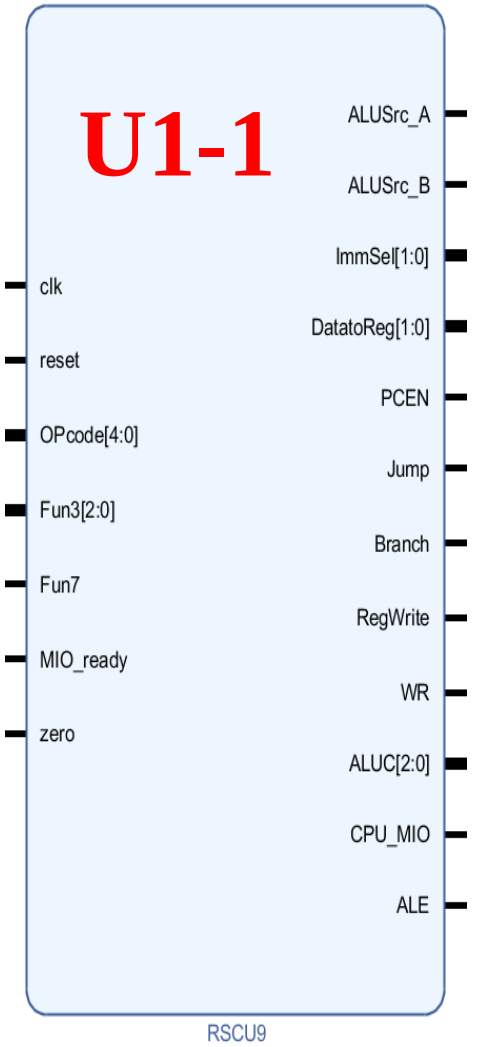
```
module     RSDP9(input wire clk,                          //寄存器时钟
                 input wire rst,                          //寄存器复位
                 input wire[31:0] inst_field,             //指令数据域
                 input wire[31:0] Data_in,                //CPU数据输入
                 input wire       ALUSrc_A,               //寄存器A通道控制
                 input wire       ALUSrc_B,               //寄存器B通道控制
                 input wire[2:0]  ALUC,                   //ALU操作功能控制
                 input wire[1:0]  ImmSel,                 //RISV-V多立即数选择
                 input wire[1:0]  DatatoReg,              //REG写数据通道选择
                 input PCEN,                              //PC使能信号
                 input wire       Jump,                   //UJ跳转控制
                 input wire       Branch,                 //SB跳转控制
                 input wire       RegWrite,               //寄存器写信号
                 input Sign,                              //符号标志（保留）

                 output wire[31:0] PCOUT,                 //PC地址输出
                 output wire[31:0] Data_out,              //CPU数据输出
                 output wire[31:0] ALU_out,               //ALU运算结果输出
                 output overflow,                         //溢出(保留)
                 output zero,                             //ALU操作为"0"
                                                          //Debug signals
                 output rs1_data,                         //rs1寄存器输出
                 output Wt_data,                          //寄存器写数据
                 output ALUA,                             //ALU A通道输入
                 output ALUB,                             //ALU B通道输入
                 input [6:0]Debug_addr,                   //测试定位
                 output [31:0]Debug_regs                  //测试、调试信号
                 );

endmodule
```

CPU部件之2-控制器： RSCU9

- **Controller**
  - CPU主要部件之一
  - 寄存器传输控制单元：
    控制运算和通路控制器
- **基本功能**
  - 指令译码
  - 产生操作控制信号：ALU运算控制
  - 产生指令所需的路径选择
- **本实验用IP 软核RSCU9配合**
  - 核调用模块RSCU9.edf
  - 核端口信号定义模块(空文档)：
    - RSCU9.v



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之数据通路设计

–调用实验0设计的多路器

–调用实验0的基本运算模块

–调用实验1/4设计的ALU和Regs

设计工程： OExp05-CSTEH-CPUSDP

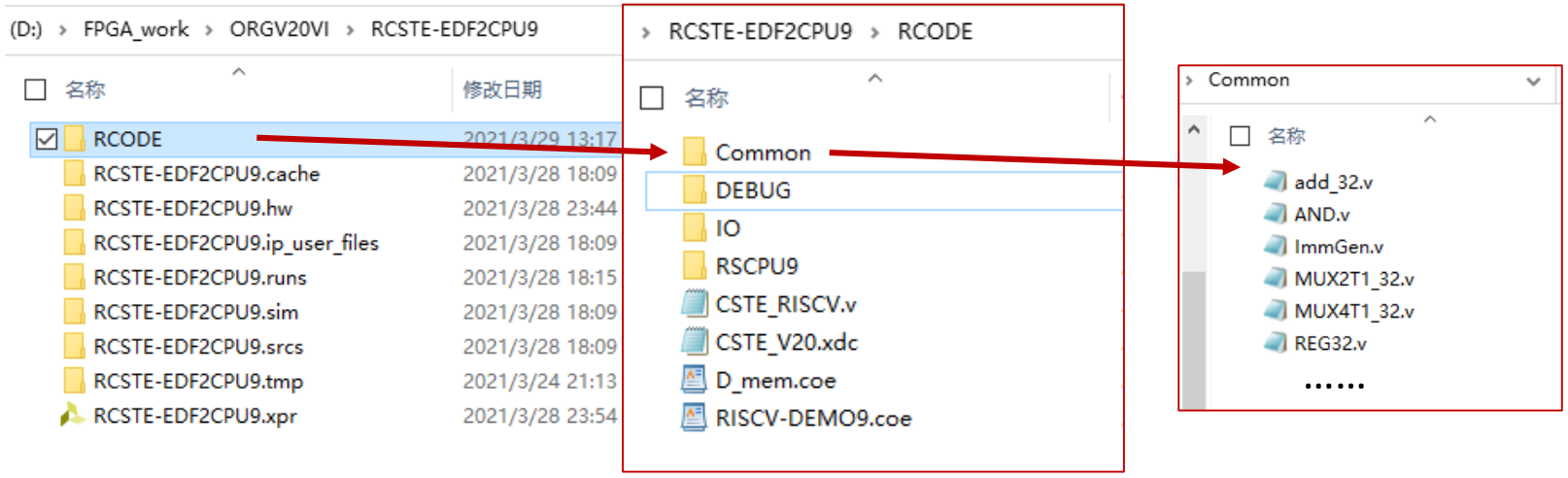
- **设计CPU之数据通路**
  - 根据理论课分析讨论设计9+条指令的数据通路
  - 原理图或HDL描述均可
  - 仿真测试DataPath模块
- **集成替换验证通过的数据通路模块**
  - 替换实验四(Exp04)中的RSDP9.edf核
  - 顶层模块延用Exp04
    - 模块名： OExp05-CSTEH-CPUSDP
- **测试数据通路模块**
  - 设计测试程序(RISCV汇编)测试：
  - ALU功能
  - R-指令、S、B、J指令通路

设计要点

复制实验0~4设计的基本元件至当前工程核目录：

add_32、and32、or32、ADC32、xor32、nor32、srl32、mux2to1_32 、 mux2to1_5、mux8to1_32、or_bit_32 ALU、Regs、REG32

参考目录结构：

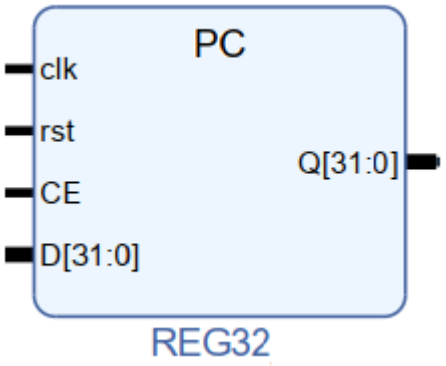


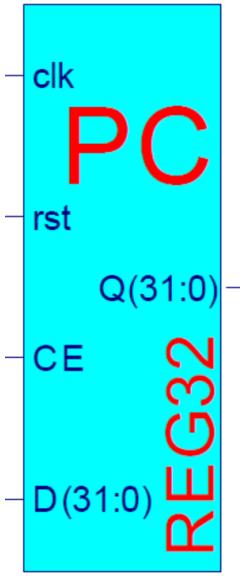
若所有模块都为自己设计的verilog文件，则可直接添加.v文件

设计要点

- 设计32位寄存器
  - 用途：PC指针、数据、地址或指令锁存
  - 参考逻辑实验Exp10，用行为描述实验
  - 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
```

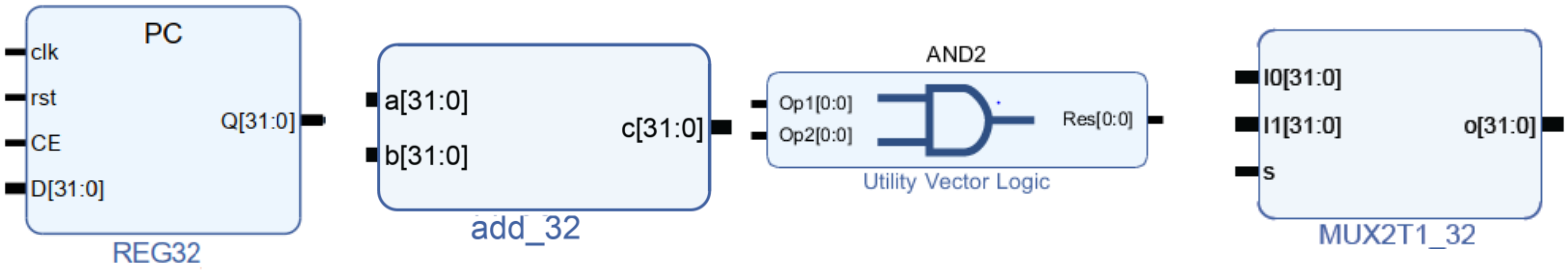




REG32接口

设计要点

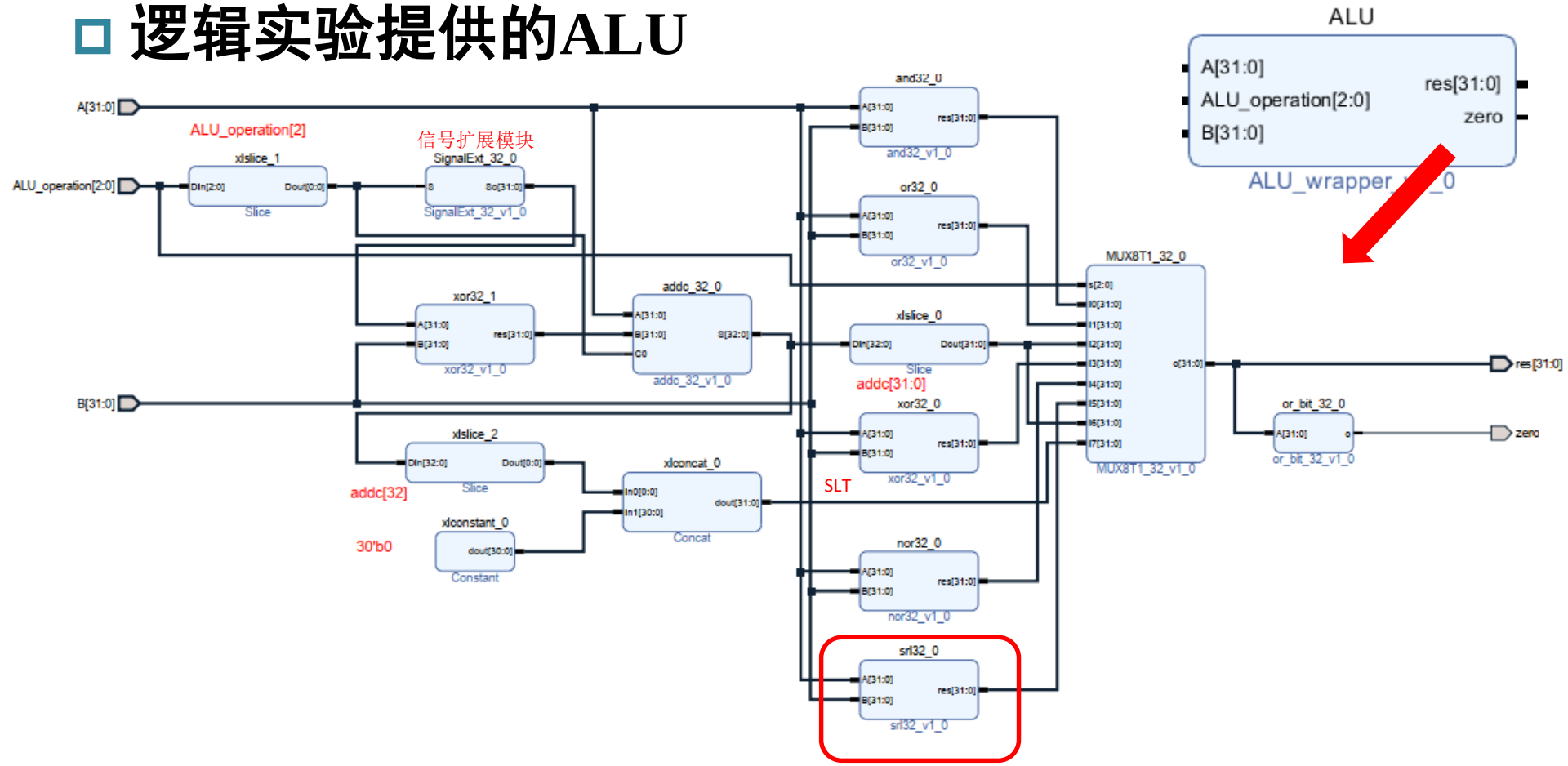
- 建立DataPath描述输入模板
- 设计PC通路
  - 调用REG32、add_32、AND2(库)和MUX2T1_32模块
    - HDL或BD



  - 设计：
    - 顺序执行(PC+4)、Jump和Beq时的PC值
    - 计算和通路
  - 注意常数的电路实现：
    - “4” = ? ? ? ?
    - Branch_offset = ? ? ? ?
    - Jump_addr = ? ? ? ?

设计要点：Exp04的ALU模块

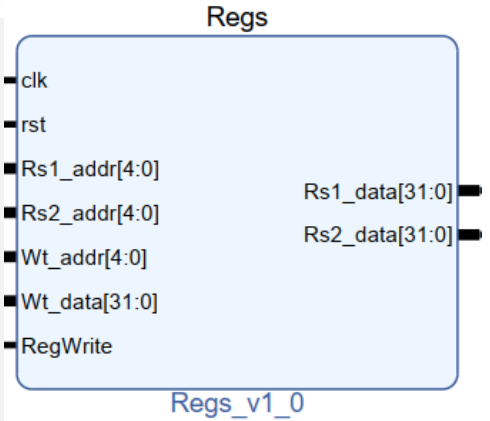
- 逻辑实验提供的ALU



Lab01的srl32模块，因沿用老版SP开发板，只能B右移一位，请自行修改或重新采用HDL描述实现

调用Exp04的Regs模块

此代码留有BUG，请同学自行编写




```
Module  regs( input           clk,  rst,  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 rdata_A = (Rs1_addr== 0) ? 0 : register[Rs1_addr];                  // read
    assign rdata_B = (Rs2_addr== 0) ? 0 : register[Rs2_addr];                  // read

    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
    assign Debug_regs = (Debug_addr == 0) ? 0 : register[Debug_addr];          //TEST
endmodule
```

代码来自李亚民教授

设计要点：CPU之数据通路建立

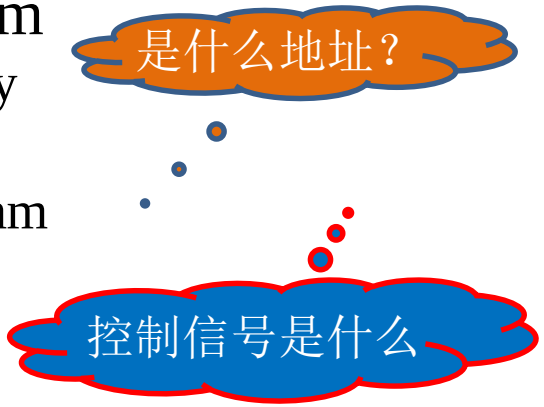
- 设计R-格式和I-格式数据通路
  - 根据理论课分析讨论的数据通路选择MUX
  - Regs数据通道设计
    - R-格式源地址通道选择：rs1、rs2
    - R-格式目的地址通道选择：rd
    - R-格式目的数据通道选择：form ALU
    - I-格式（立即数操作）源地址通道选择：rs1、Imm
    - I-格式（立即数操作）目的地址通道选择：rd
    - I-格式（立即数操作）目的数据通道选择：from ALU
    - I-格式（lw操作）源地址通道选择：rs1、Imm
    - I-格式（lw操作）目的地址通道选择：rd
    - I-格式（lw操作）目的数据通道选择：from Memory
  - ALU数据通路设计
    - ALU输入端口A有通道选择吗？
    - ALU输入端口B通道选择
      - 调用ImmGen模块
      - R-格式：from Regs 2 port
      - I-格式：Where from

是什么地址？

控制信号是什么

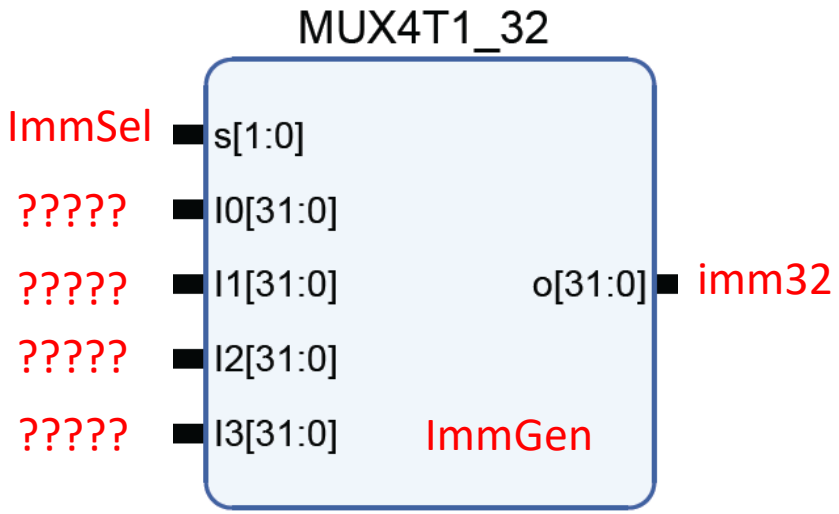
设计要点：通路遍历

- 设计S-格式和B-格式、J-格式数据通路
  - 根据理论课分析讨论的数据通路选择MUX
  - Regs数据通道设计
    - SW-格式源地址通道选择：rs1、imm
    - SW-格式目的地址通道选择：from memory
    - SW-格式目的数据通道选择：rs2
    - Beq-格式源地址通道选择：PC+4、PC+imm
    - Beq-格式目的地址通道选择：PC
    - J格式的源地址通道是什么？
  - ALU数据通路设计
    - ALU输入端口A有通道选择吗？
    - ALU输入端口B通道选择
      - 调用ImmGen模块(MUX0：32位四选1)
      - S-格式：from Imm
      - SB-格式：Where from
      - UJ-格式：Where from



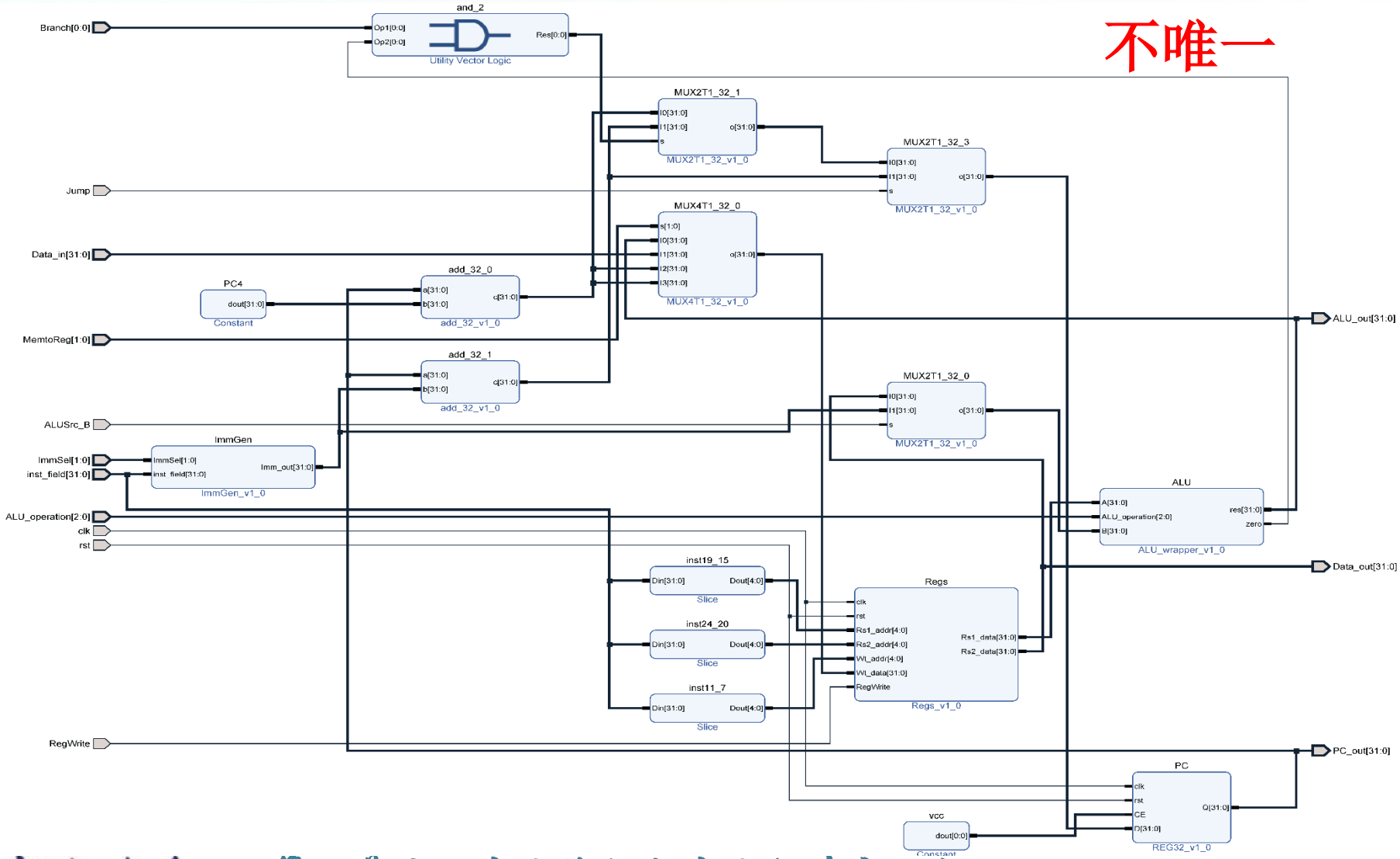
设计要点：ImmGen立即数生成设计

- **ImmGen**
  - 数据通路主要部件之一
  - 立即数生成单元
- **基本功能**
  - 具有输入指令产生立即数的逻辑功能
  - 具有转移指令偏移量左移位的功能
- **接口要求- ImmGen**
  - 数据通路接口如右图
    - 就是一个四选一
    - 选择信号：ImmSel
  - SB、UJ目标地址
    - 乘2(…, 1”b0)
      - 可以合并
      - 也可以外部独立

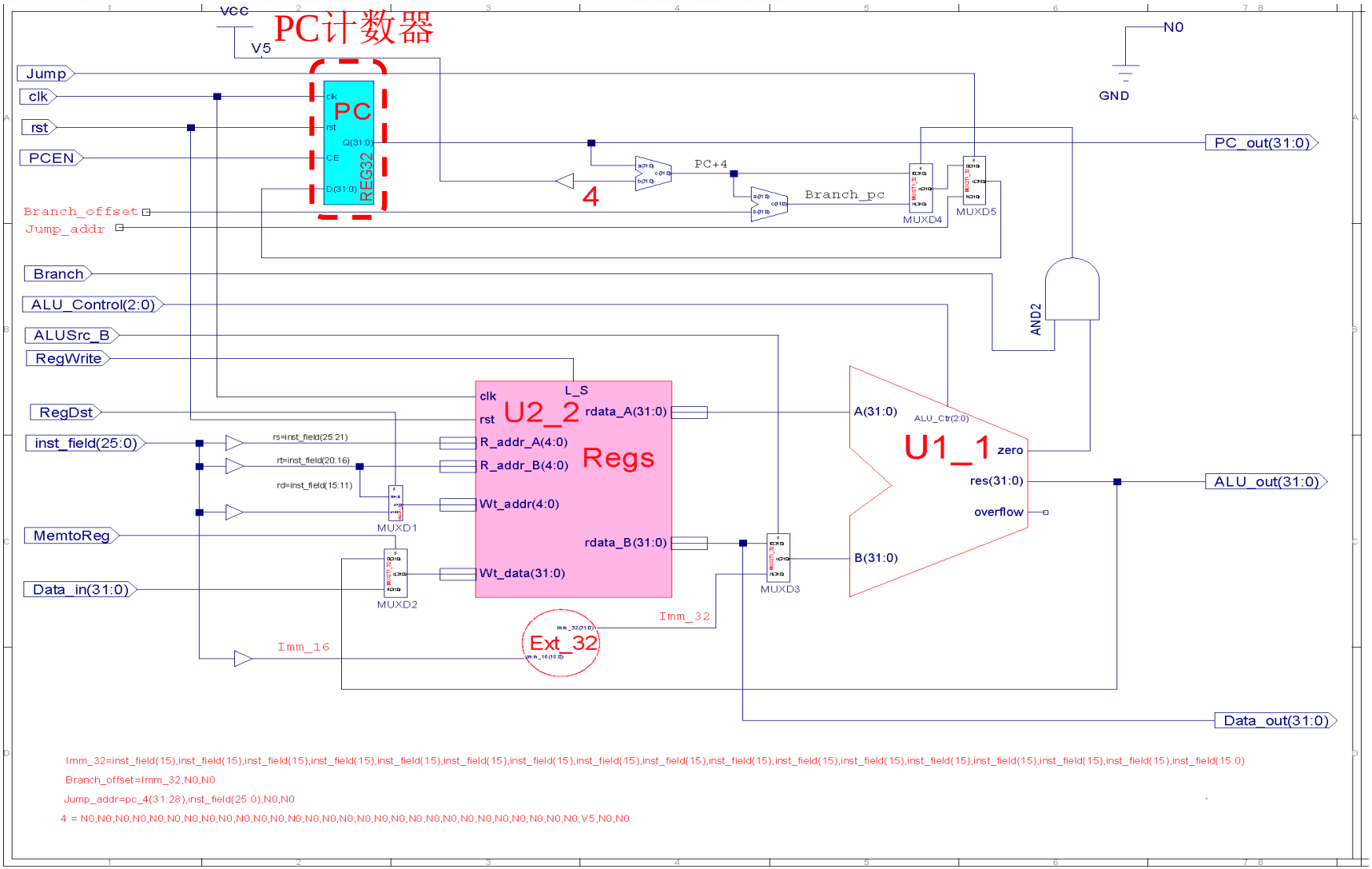


设计要点：数据通路BD参考结构图

不唯一



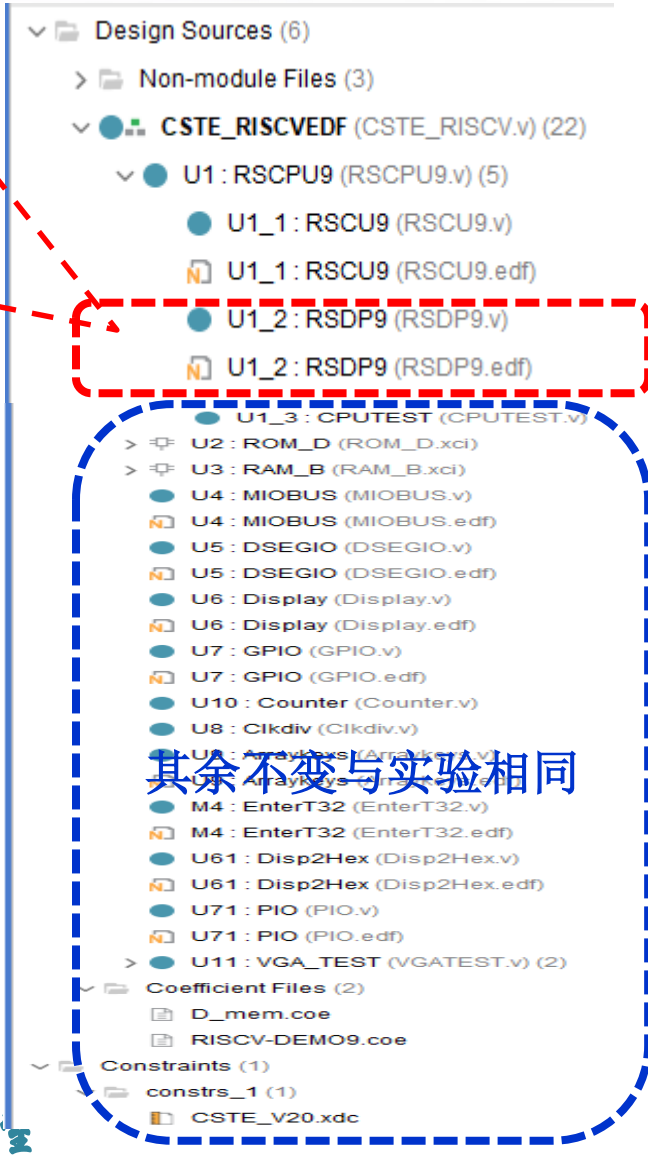
数据通路逻辑结构图(ISE-较清晰)



设计要点：DataPath替换集成前

- 集成替换
  - 仿真正确后替换Exp04的数据通路IP核
- **清理OExp04工程**
  - 移除工程中的数据通路核
    - Exp04工程中移除数据通路核关联
  - 删除工程中RSDP核文件
    - RSDP9.edf和RSDP9.v 文件
    - 在Project菜单中运行：
      Cleanup Project Files …
  - 建议用OExp04资源重建工程
    - 除RSDP9.edf核
  - 添加DataPath核后，参照OExp04完成CPU核集成

OExp04需要清理的核



设计要点：DataPath替换集成后

- 集成替换DapaPath核后的模块层次结构

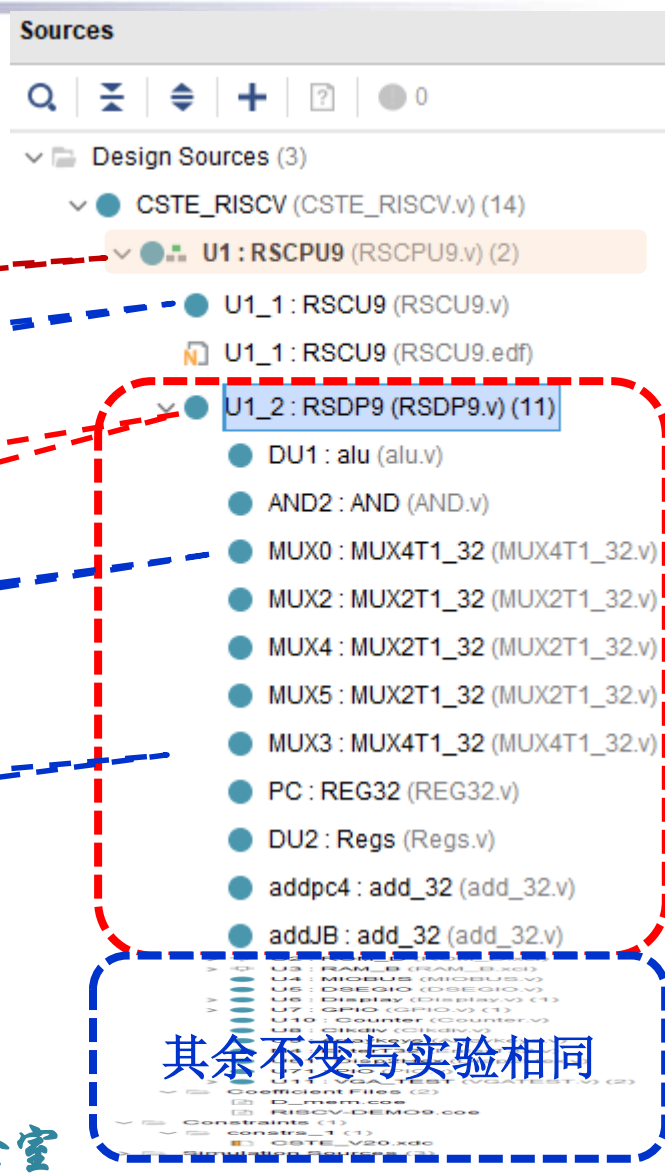
CPU顶层描述，与实验四相同

控制器不变仍然使用edf网表IP核

OExp05完成数据通路替换后的模块调用关系

MUX0：ImmGen

其余实验0~4设计模块，结构化调用描述



设计要点：数据通路仿真测试

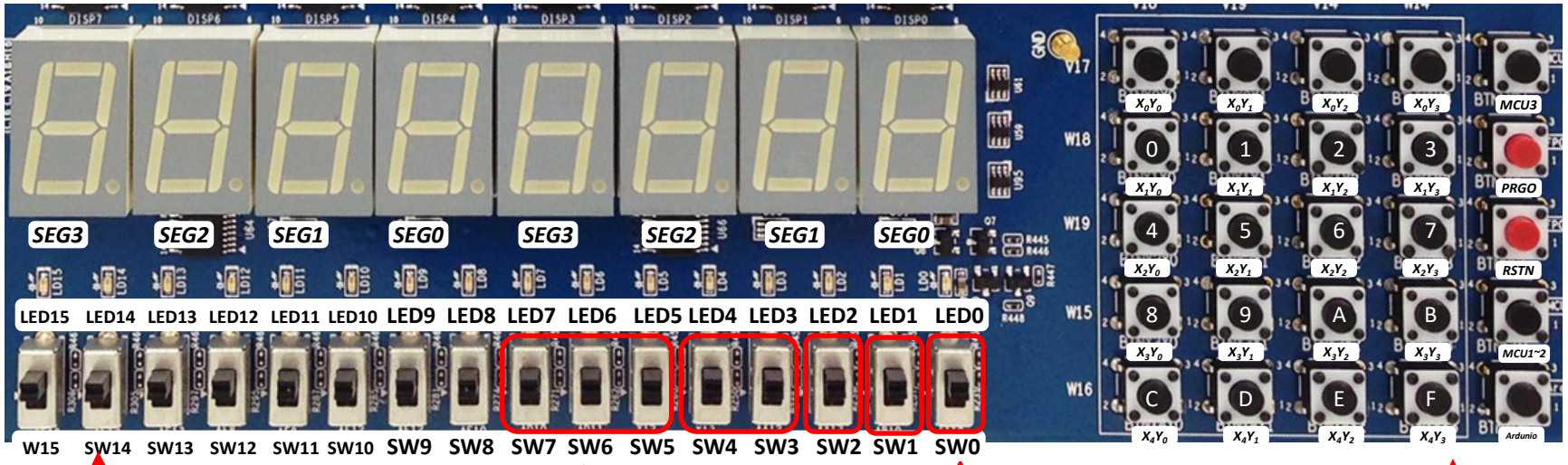
- 数据通路仿真调试
  - DataPath模块仿真
    - HDL或原理图描述检查没有Errors后仿真测试
    - 仿真激励代码设计要点
      - 只做功能性测试，不做性能和完备性测试
      - **通路功能测试**
        - » 选择9条指令所有可能通路的代表指令
        - » 激励输入：
          - 计算出对应指令的输入控制信号和代表数据
          - clk、rst
      - **ALU功能测试**
        - » 选择add、and、sub、or、slt指令
          - 计算出对应指令的输入控制信号和代表数据
        - » 选择Beq、Load和Stroe测试地址计算
      - **Regs功能测试**
        - » add指令代表作寄存器遍历测试

若包含提供的edf格式IP，则无法进行行为仿真

物理验证

- 使用DEMO程序目测数据通路功能
  - 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s5(x21)+1显示
- 用汇编语言设计测试程序
  - 测试ALU功能
  - 测试Regs访问
  - 测试R-格式指令通路
  - 测试I-格式指令(lw)通路
  - 测试B-格式指令通路

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)

测试程序参考：ALU和Regs

- 设计ALU和Regs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

```
#baseAddr 0000
loop:   nor r1,r0,r0;       //r1=FFFFFFFF
        slt r2,r0,r1;       //r2=00000001
        add r3,r2,r2;       //r3=00000002
        add r4,r3,r2;       //r4=00000003
        add r5,r4,r3;       //r5=00000005
        add r6,r5,r4;       //r6=00000008
        add r7,r6,r5;       //r7=0000000d
        add r8,r7,r6;       //r8=00000015
        add r9,r8,r7;       //r9=00000022
        add r10,r9,r8;      //r10=00000037
        add r11,r10,r9;     //r11=00000059
        add r12,r11,r10;    //r12=00000090
        add r13,r12,r11;    //r13=000000E9
        add r14,r13,r12;    //r14=00000179
        add r15,r14,r13;    //r15=00000262
        add r16,r15,r14;    //r16=000003DB
        add r17,r16,r15;    //r17=000006D3
        add r18,r17,r16;    //r18=00000A18
        add r19,r18,r17;    //r19=000010EB
        add r20,r19,r18;    //r20=00001B03
        add r21,r20,r19;    //r21=00003bEE
        add r22,r21,r20;    //r22=000046F1
        add r23,r22,r21;    //r23=000080DF
        add r24,r23,r22;    //r24=0000C9D0
        add r25,r24,r23;    //r25=00014AAF
        add r26,r25,r24;    //r26=0001947F
        add r27,r26,r25;    //r27=0012DF2E
        add r28,r27,r26;    //r28=001473AD
        add r29,r28,r27;    //r29=002752DB
        add r30,r29,r28;    //r30=003BC688
        add r31,r30,r29;    //r31=00621963
j loop;
```

测试程序参考

- 设计通道测试程序替换DEMO程序
  - 通道测试参考设计。测试结果通过CPU输出信号单步观察
  - 通道功能由传输数据结果来指示，如立即数通道观察：14+$zero

```
#baseAddr 0000
start:                                  //通道结果由后一条指令读操作数观察
        lw   r5, 14($zero);             //取测试常数55555555。存储器读通道
start_A:
        add r1, r5, $zero;              //r1: 寄存器写通道。R5:寄存器读通道A输出
        nor r2, $zero, r1;              //r1: 寄存器读通道B输出。R2:ALU输出通道
        lw   r5, 48($zero);  //取测试常数AAAAAAAA。立即数通道:00000048
        beq r2, r5 start_A;             //循环测试
        j     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通道切换功能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

设计要点：ROM功能调试代码

- **设计32位指令存储器：**
  - SWORD实验平台 ROM用Distributed Memory
- **ROM初始化文件(RISCV-DEMO9.coe)**
  - 这是一段功能测试程序：CPU仿真另行设计

注意：这是RISC-V代码与OExp02不同

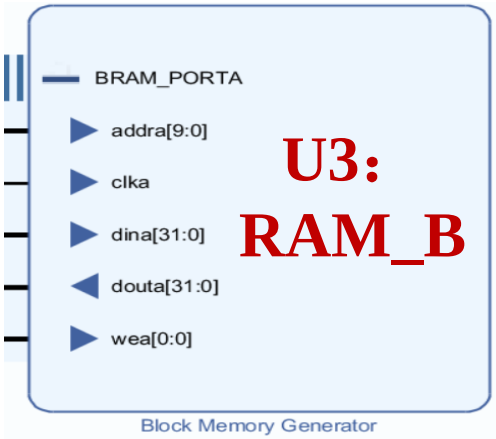
```
memory_initialization_radix=16;
memory_initialization_vector=
0200006F,00000033,00000033,00000033,00000033,00000033,00000033,00000033,00C02283,00502333,
006303B3,00638E33,00738733,01CE02B3,005282B3,01C28EB3,01DE8F33,01EF0F33,01CF0433,01EF0F33,
01EF0F33,01DF0FB3,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,
01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,
01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF0F33,01EF04B3,01E4E633,00948933,012902B3,005282B3,
406006B3,00C4A223,0004A583,00B585B3,00B585B3,00B4A023,006A8AB3,01592023,01402B03,0004A583,
00B585B3,00B585B3,00B4A023,0004A583,0055FC33,006B0B33,040B0E63,0004A583,00E70BB3,017B8CB3,
019B8BB3,0175FC33,000C0C63,037C0463,00E70BB3,037C0663,01592023,FB9FF06F,00D78463,0080006F,
00D687B3,00F92023,FA5FF06F,0609AA83,01592023,F99FF06F,0209AA83,01592023,F8DFF06F,01402B03,
00F787B3,0067E7B3,00E989B3,0089F9B3,006A8AB3,00DA8463,00C0006F,00E00AB3,006A8AB3,0004A583,
00B58C33,018C0C33,0184A023,00C4A223,F6DFF06F;
```

设计要点：数据存储器模块测试

- 设计存储器模块测试程序
  - 7段码显示器的地址是E0000000/FFFFFFE0
  - LED显示地址是F0000000/FFFFFF00
  - 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OExp03/04**

```
memory_initialization_radix=16;
memory_initialization_vector=
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;
```

RAM初始化数据。红色数据为七段LED图形



设计测试记录表格

- 学会实验数据的统计
  - 参考大学物理实验
    - 本实验没有有效数精确计算，但有大量数据表格
- ALU和Regs测试结果记录
  - 自行设计记录表格
- 通道测试结果记录
  - 自行设计记录表格
- 数据存储模块测试记录
  - 自行设计记录表格

思考题

- 扩展下列指令，数据通路将作如何修改：

  R-Type：　sra, sll, sltu；

  I-Type：　addi, andi, ori, xori, lui, slti , srai, slli, sltiu

  B-Type：　bne, blt；

  UJ-Type：　Jal；

  U-Type：　lui；

- 增加I-Type算术运算指令是否需要修改本章设计的数据通路？


END

Appendix(供参考，接口信号以教学班约定)

RISC-V RV32I数据通路的原理介绍

- 最终实现不少于下列指令(实验七完成)

  R-Type： add, sub, and, or, xor, slt,srl；

  I-Type： addi, andi, ori, xori, slti, srli,lw;

  S-Type： sw；

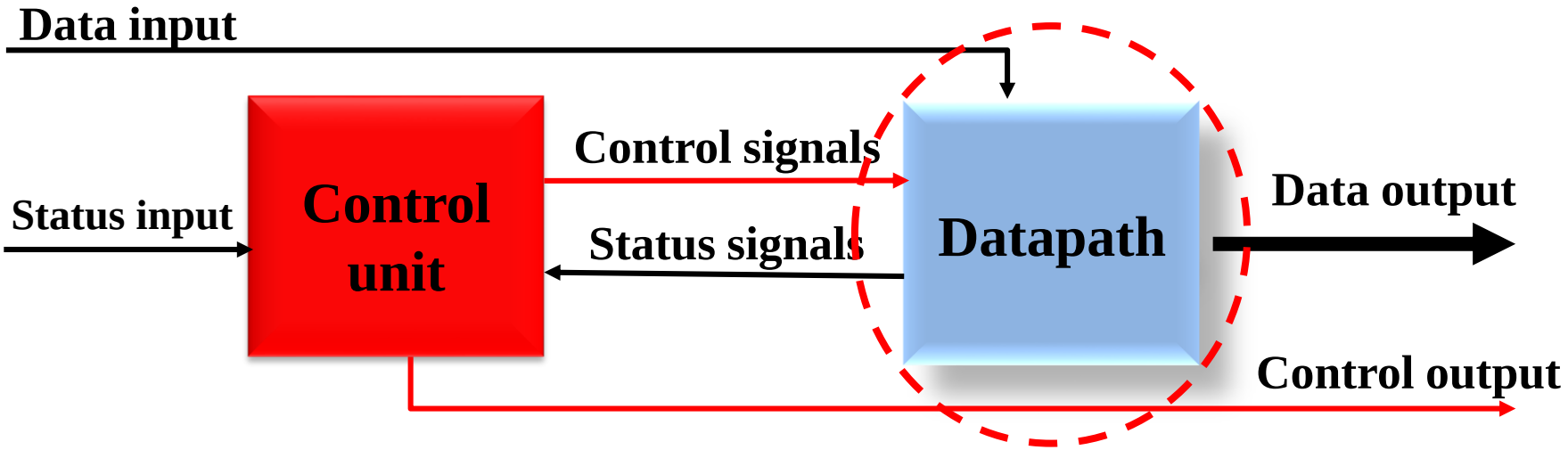
  B-Type： beq, bne, blt；

  J-Type： Jal, Jalr；

CPU organization

Digital circuit

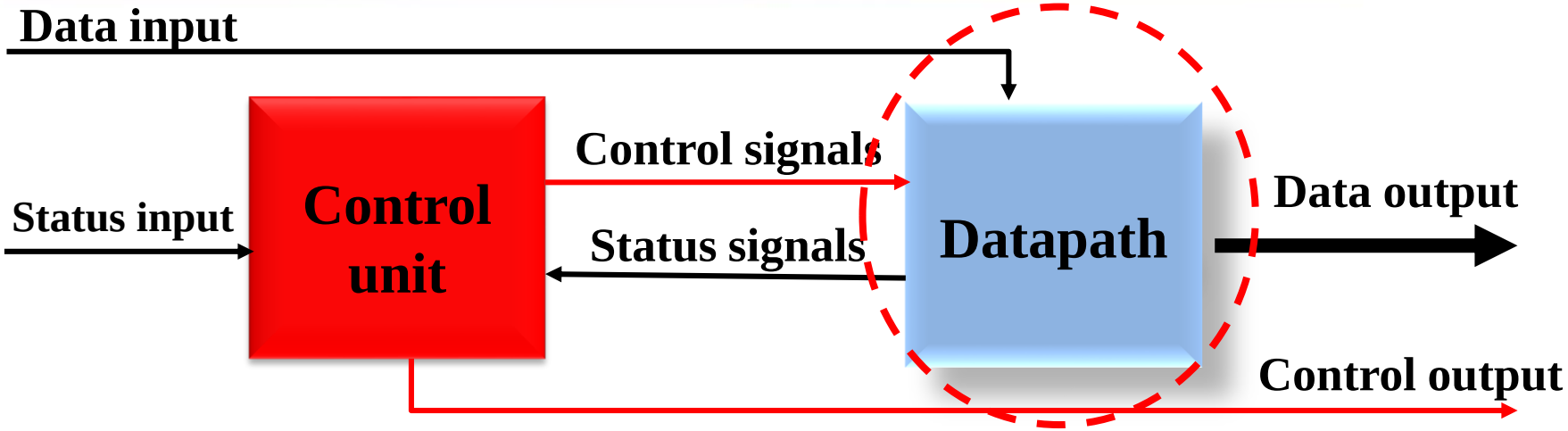
- General circuits that controls logical event with logical gates --Hardware



Computer organization

- Special circuits that processes logical action with instructions -Software

Datapath

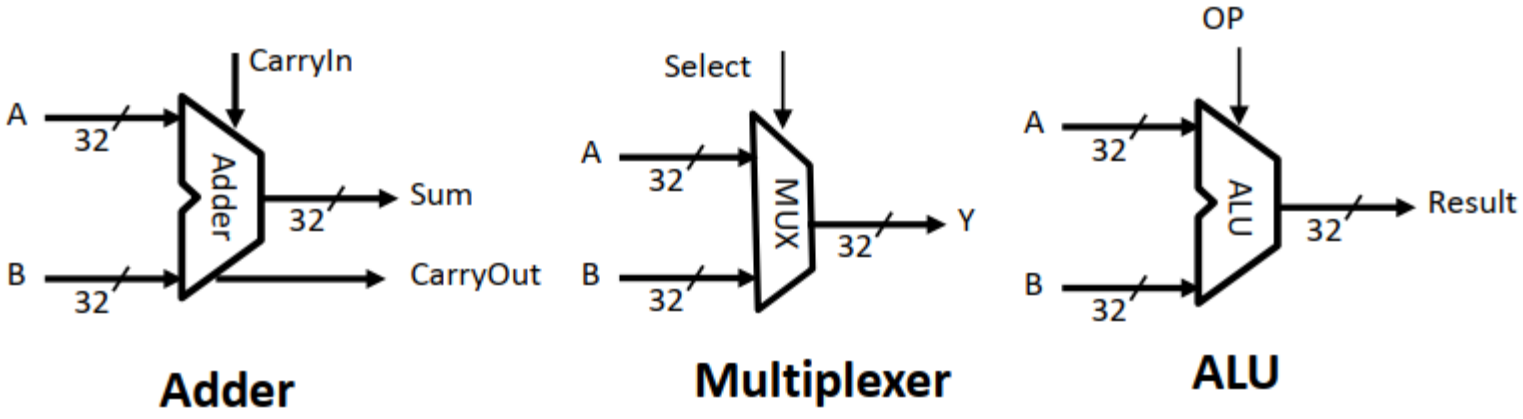


- 数据通路
  - 数据通路作为处理器的一部分，包含了完成处理器所要求的的操作所必须的硬件，包括运算单元、寄存器组、状态寄存器等

Datapath

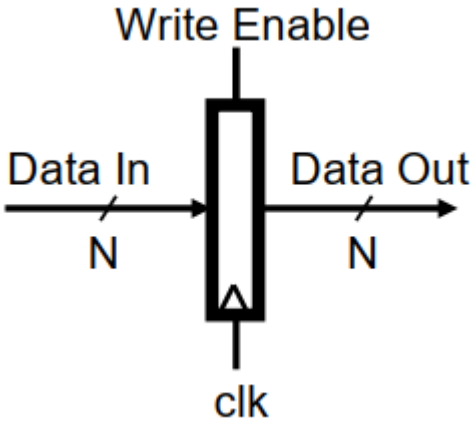
数据通路部件：

- 组合逻辑单元：　加法器、多路选择器、ALU算数运算单元



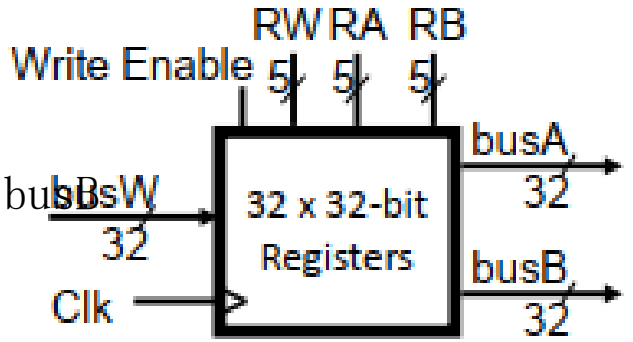
Datapath

- 数据通路部件：
- 时序逻辑单元（状态元件）：　　Register
  - Write Enable:
    - 置0，数据输出保持原状态不变
    - 置1，在有效时钟边沿到来，数据输出为数据输入值



Datapath

- 数据通路部件：
- 时序逻辑单元：　　寄存器堆
  - **Register files：　由32个register构成**
    - 两个32位的数据输出端口：busA、busB
    - 一个32位的数据输入端口：busW

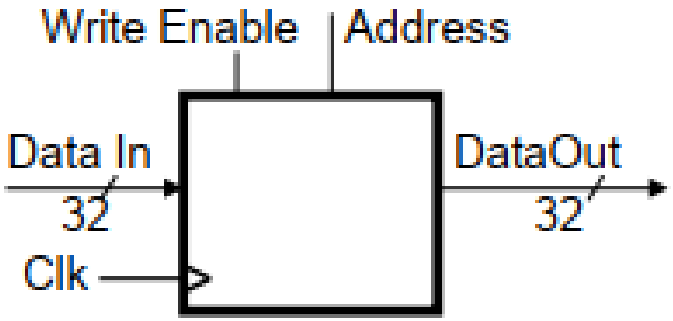


- **寄存器读写操作：**
  - RA（number）作为地址选择register存储的数据传输到输出端口busA（读）
  - RB（number）作为地址选择register存储的数据传输到输出端口busB（读）
  - RW（number）作为地址选择register接收输入端口busW的数据，当Write Enable=1且时钟边沿有效时（写）

- **时钟（clk）：**
  - 写操作时，clk是影响因子，在有效的时钟边沿到来，数据被写入
  - 读操作时，clk非影响因子，只要RA、RB有效，经过短暂的器件延迟，数据即从busA、busB输出（此时属于组合逻辑操作）

Datapath

- 时序逻辑单元：
  - 存储器
    - 一个数据输入端口：DataIn
    - 一个数据输出端口：DataOut



  - 读写操作：
    - 读：Address 作为地址选择存储的数据输出到端口DataOut
    - 写：Address 作为地址接收端口DataIn输入的数据，当Write Enable = 1且时钟边沿有效
  - 时钟(CLK)
    - 写操作时，clk是影响因子，在有效的时钟边沿到来，数据被写入
    - 读操作时，clk非影响因子，只要Address有效，经过短暂的器件延迟，数据即从DataOut输出（此时属于组合逻辑操作）

每条指令执行时在取指之后会更新以下状态元件:

- 通用寄存器Registers (x0..x31)
  - 寄存器堆为32个32位的寄存器:: Reg[0]..Reg[31]
  - 指令的rs1字段指定了第一个源操作寄存器的读地址
  - 指令的rs2字段指定了第二个源操作寄存器的读地址
  - 指令的rd字段指定了目标操作寄存器的写地址
  - 寄存器 x0值永远为0; 写操作无效
- Program Counter (PC)
  - 保存当前指令的地址
- Memory (MEM)
  - 在32位宽的存储空间内保存指令或数据
  - 本实验会采用分开的存储用于存储指令 (IMEM) 和数据(DMEM)
  - 本实验采用的指令存储器只支持只读模式
  - 只有Load/store 操作才会访问数据存储器

Table of RV Registers

Register(s)	Alt.	Description
x0	zero	The zero register, always zero
x1	ra	The return address register, stores where functions should return
x2	sp	The stack pointer, where the stack ends
x5-x7, x28-x31	t0-t6	The temporary registers
x8-x9, x18-x27	s0-s11	The saved registers
x10-x17	a0-a7	The argument registers, a0-a1 are also return value

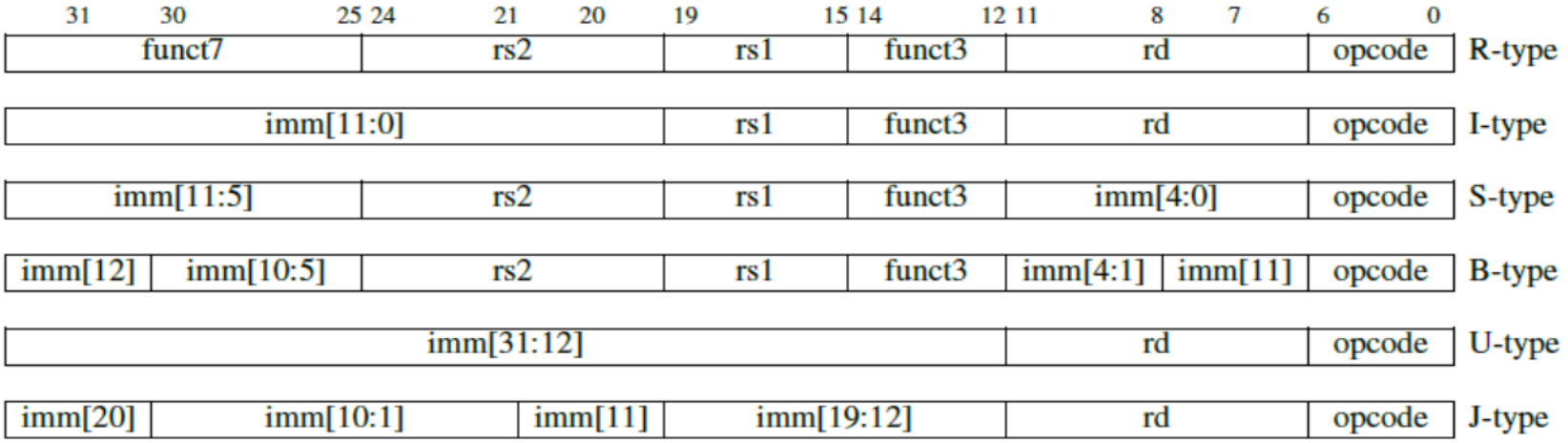
RV Instructions

- RISC-V指令分类：
  - 1. RV32I，它是 RISC-V 固定不变的基础整数指令集
  - 2. RV32M，乘法和除法
  - 3. RV32F 和 RV32D，浮点操作
  - 4. RV32A，原子操作

- RISC-V　RV32I 六种基本指令格式：用于寄存器-寄存器操作的 R 类型指令，用于短立即数和访存 load 操作的 I 型指令，用于访存 store 操作的 S 型指令，用于条件跳转操作的 B 类型指令，用于长立即数的 U 型指令和用于无条件跳转的 J 型指令。

- 本实验主要实现RV32I的常见指令

RV Instructions



opcode：为操作码；用于表示指令格式和指令操作的字段

rs1：只读。为第1个源操作数寄存器，寄存器地址（编号）是00000~11111，00~1F；

rs2：只读。为第2个源操作数寄存器，寄存器地址（编号）是00000~11111，00~1F；

rd：只写。为目的操作数寄存器，寄存器地址（同上）；

funct3/7：为功能码，在指令中用来指定指令的功能与操作码配合使用；

immediate：为立即数，用作无符号的逻辑操作数、有符号的算术操作数、数据加载（Load）/数据保存（Store）指令的数据地址字节偏移量和分支指令中相对程序计数器（PC）的有符号偏移量；

单周期CPU

单周期CPU指的是一条指令的执行在一个时钟周期内完成，然后开始下一条指令的执行，即一条指令用一个时钟周期完成。单周期CPU，是在一个时钟周期内完成这五个阶段的处理。

(1) 取指令(IF)：根据程序计数器PC中的指令地址，从存储器中取出一条指令，同时，PC根据指令字长度自动递增产生下一条指令所需要的指令地址，但遇到“地址转移”指令时，则控制器把“转移地址”送入PC。

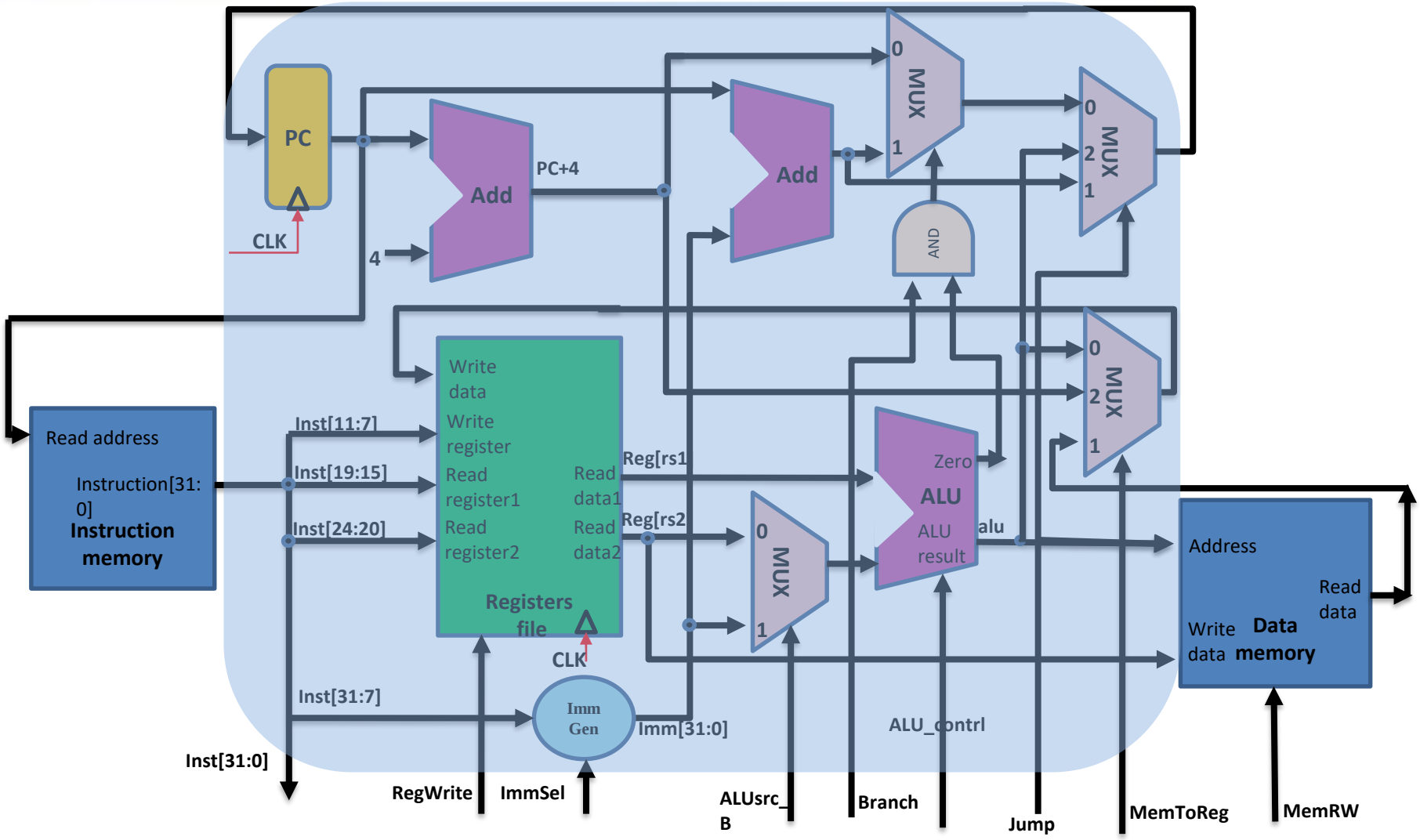
(2) 指令译码(ID)：对取指令操作中得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

(3) 指令执行(EXE)：根据指令译码得到的操作控制信号，具体地执行指令动作，然后转移到结果写回状态。

(4) 存储器访问(MEM)：所有需要访问存储器的操作都将在这个步骤中执行，该步骤给出存储器的数据地址，把数据写入到存储器中数据地址所指定的存储单元或者从存储器中得到数据地址单元中的数据。

(5) 结果写回(WB)：指令执行的结果或者访问存储器中得到的数据写回相应的目的寄存器中。

单周期数据通路结构


PC
CLK
Add
PC+4
4
Add
MUX
0
1
MUX
0
2
1
AND
MUX
0
2
1
Read address
Instruction[31:0]
Instruction memory
Inst[11:7]
Inst[19:15]
Inst[24:20]
Write data
Write register
Read register1
Read register2
Registers file
Read data1
Read data2
CLK
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]
Inst[31:0]
ALU_contrl
RegWrite
ImmSel
ALUsrc_B
Branch
Jump
MemToReg
MemRW

单周期数据通路结构

Instruction Memory：指令存储器，
  Readaddress，指令存储器地址输入端口
  Instruction，指令存储器数据输出端口（指令代码输出端口）

Data Memory：数据存储器，
  Address，数据存储器地址输入端口
  Writedate，数据存储器数据输入端口
  Readdata，数据存储器数据输出端口

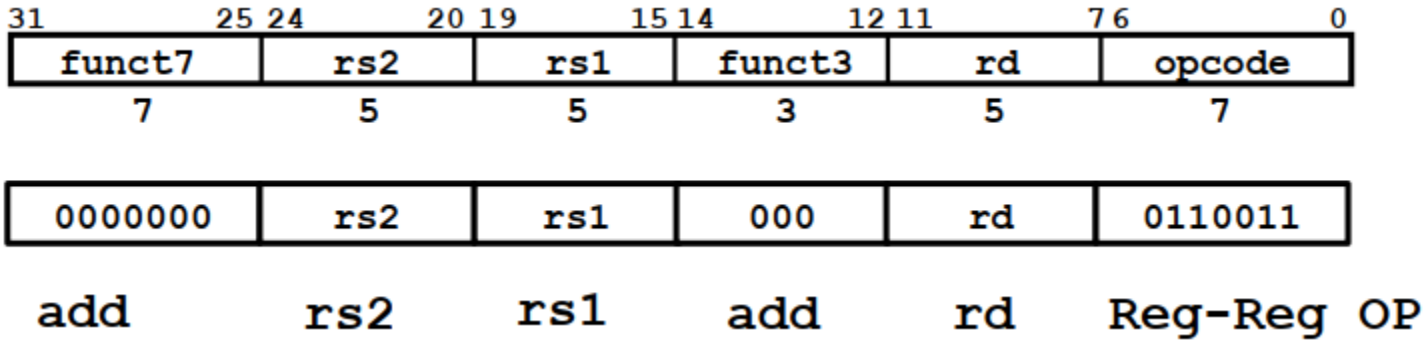
PC：程序计数器，存放指令地址

Registers：寄存器组
  Read Reg1，rs1寄存器地址输入端口
  Read Reg2，rs2寄存器地址输入端口
  Write Reg，将数据写入的寄存器端口，其地址rd字段
  Write Data，写入寄存器的数据输入端口
  Read Data1，rs1寄存器数据输出端口
  Read Data2，rs2寄存器数据输出端口

ImmGen：立即数生成单元

ALU：　算术逻辑单元
  result，ALU运算结果
  zero，运算结果标志，结果为0，则zero=1；否则zero=0

Implementing the add Instruction



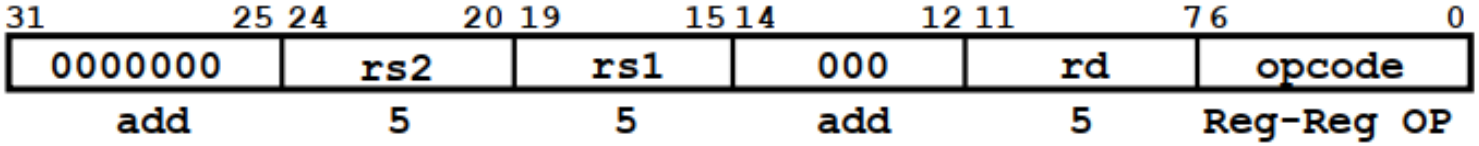
add rd , rs1, rs2

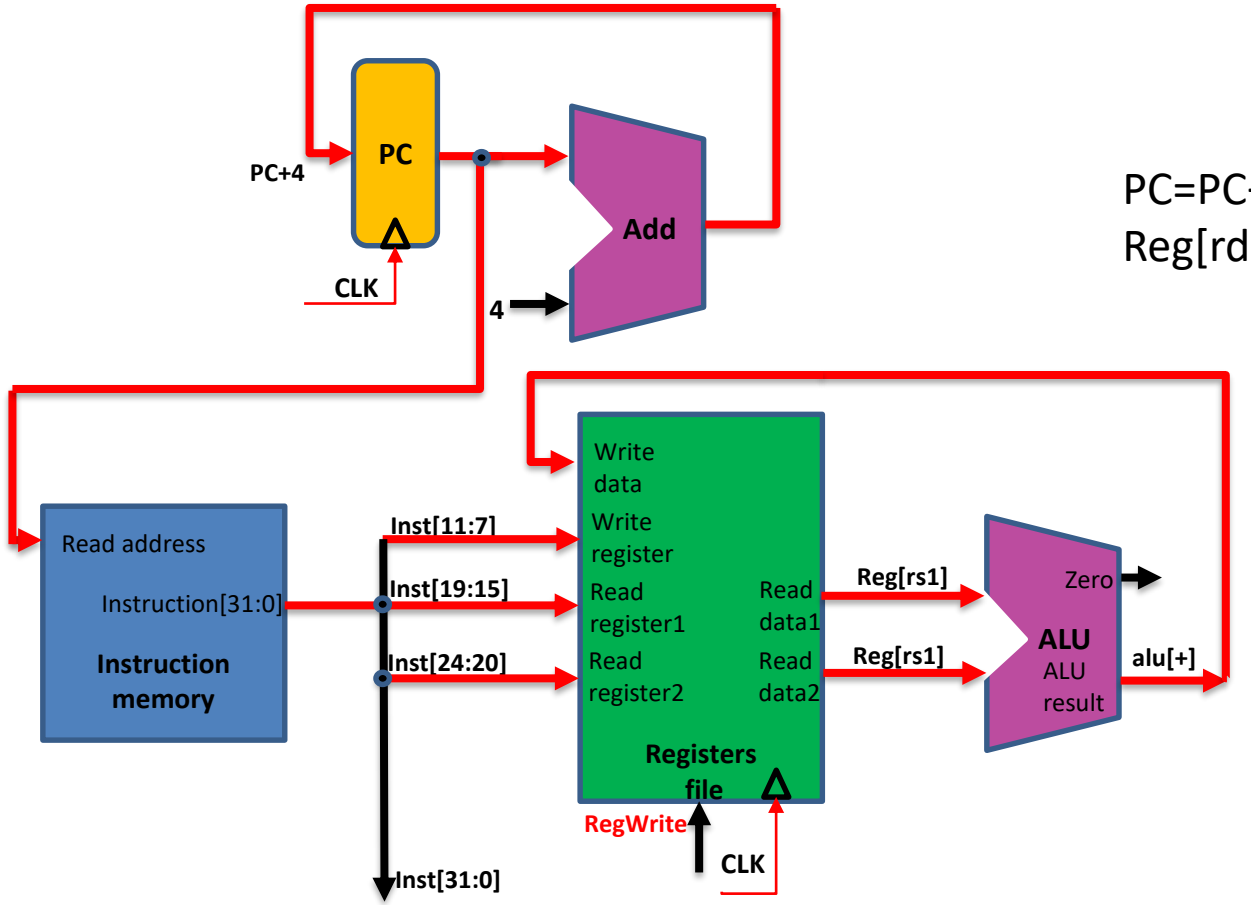
功能：rd←rs1 + rs2。

Eg：add x9,x21,x9

0000000 _01001_10101_000_01001_0110011

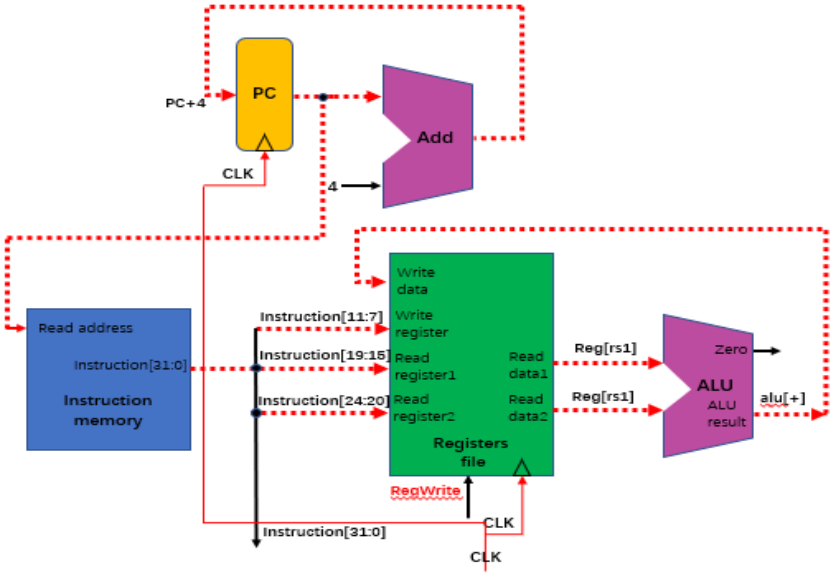
Datapath for add

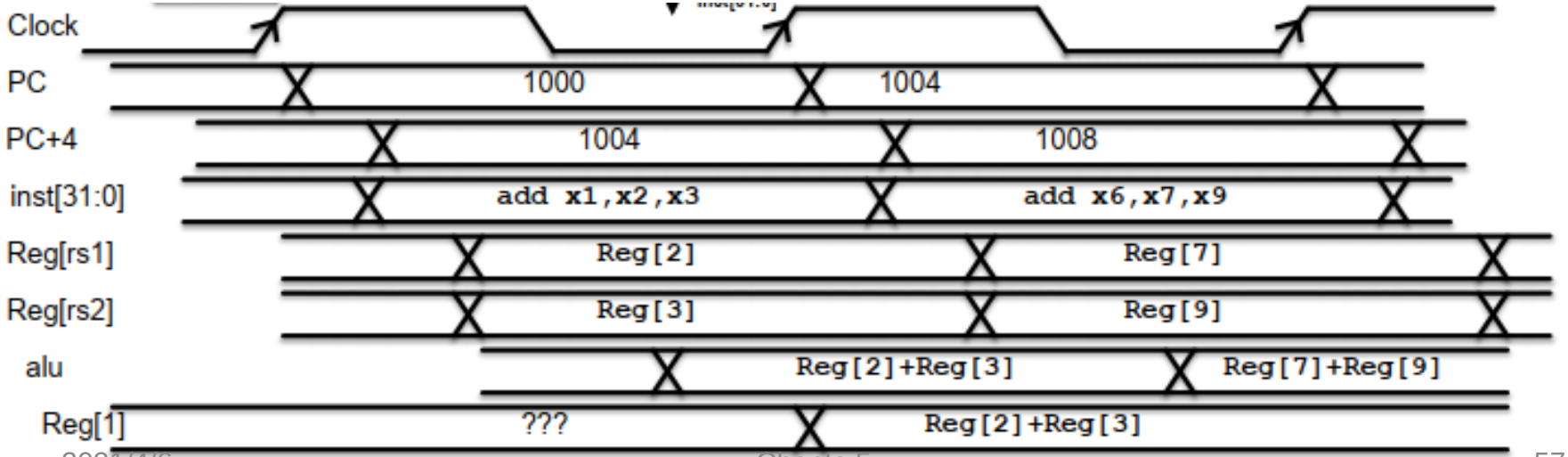




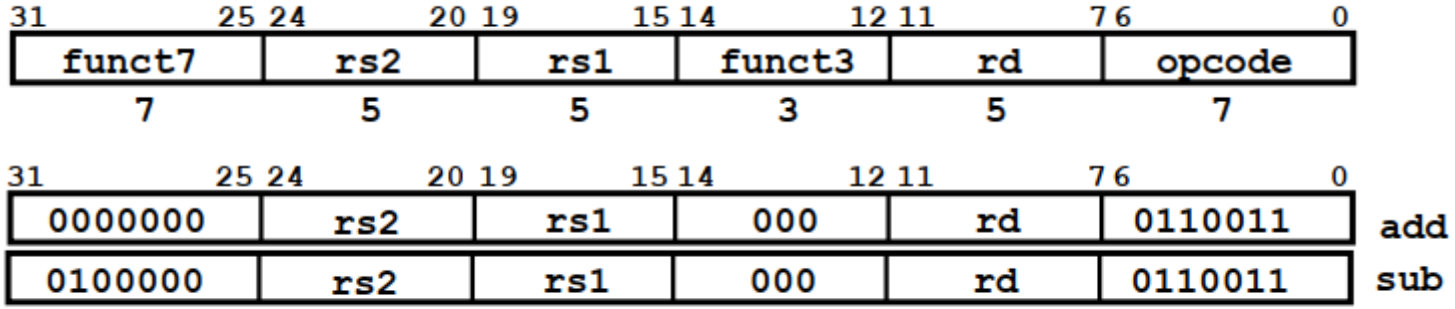
PC=PC+4
Reg[rd]=Reg[rs1]+Reg[rs2]

Timing Diagram for add





Implementing the sub Instruction



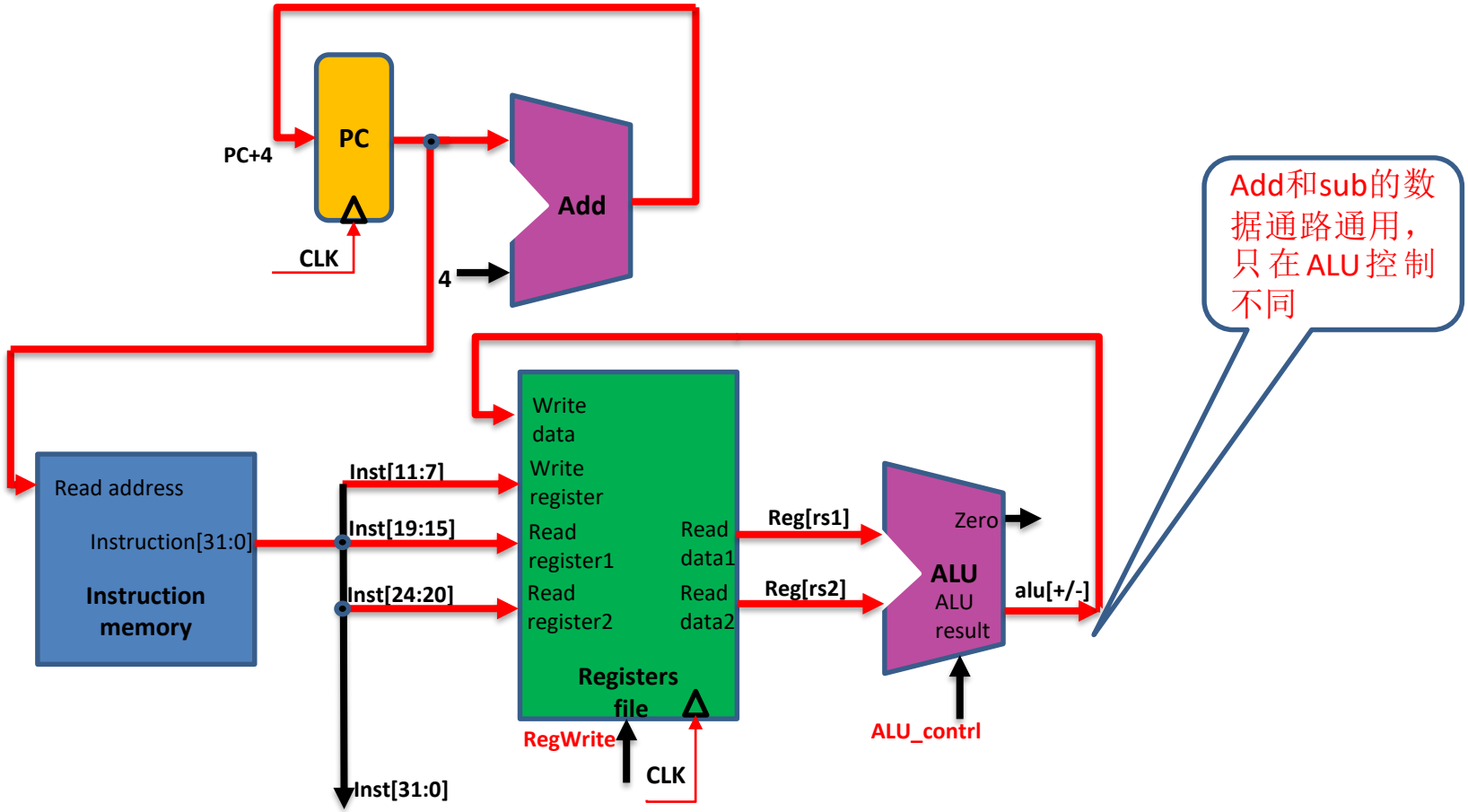
sub rd , rs1 , rs2

功能：rd←rs1 – rs2

Eg：sub x9,x21,x9

0100000 _01001_10101_000_01001_0110011

Datapath for sub/add



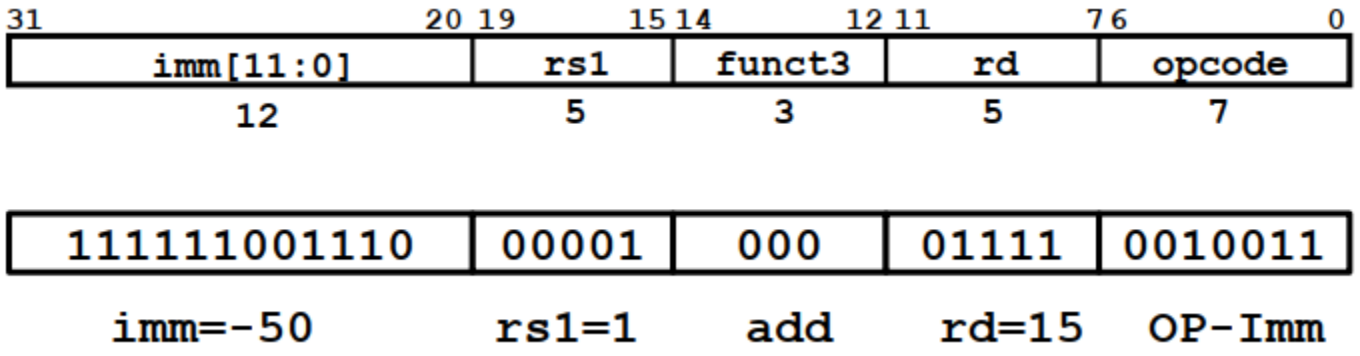
Implementing other R-Format instructions

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

All implemented by decoding funct3 and funct7 fields and
selecting appropriate ALU function

所有的R型指令，数据通路通用，只是ALU控制操作不同；而opcode相同，通过func3和func7共同决定

Implementing I-Format - addi instruction



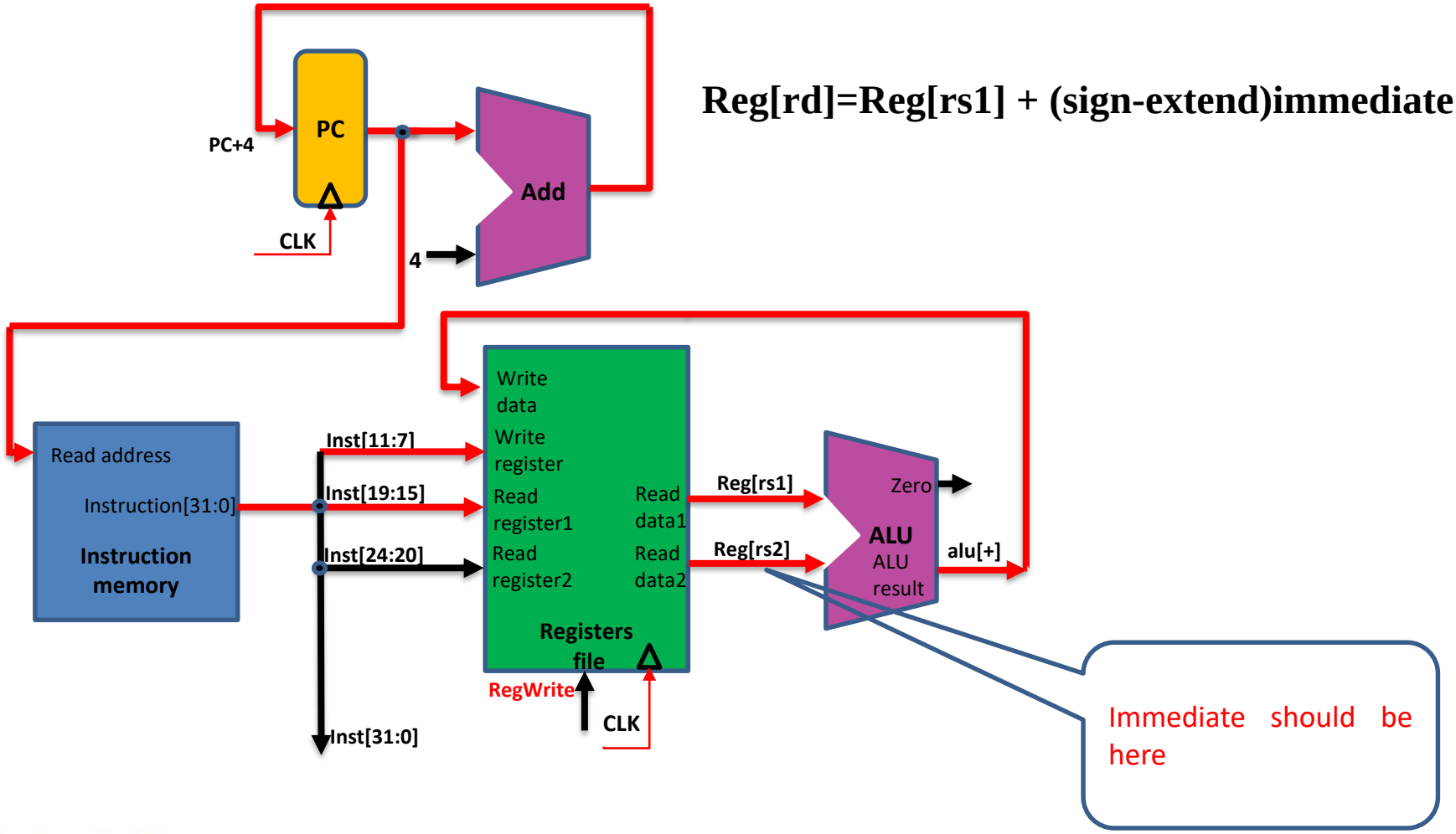
addi rd , rs1 ,immediate

功能：rd←rs1 + (sign-extend)immediate；12位immediate符号扩展再参与“加”运算。

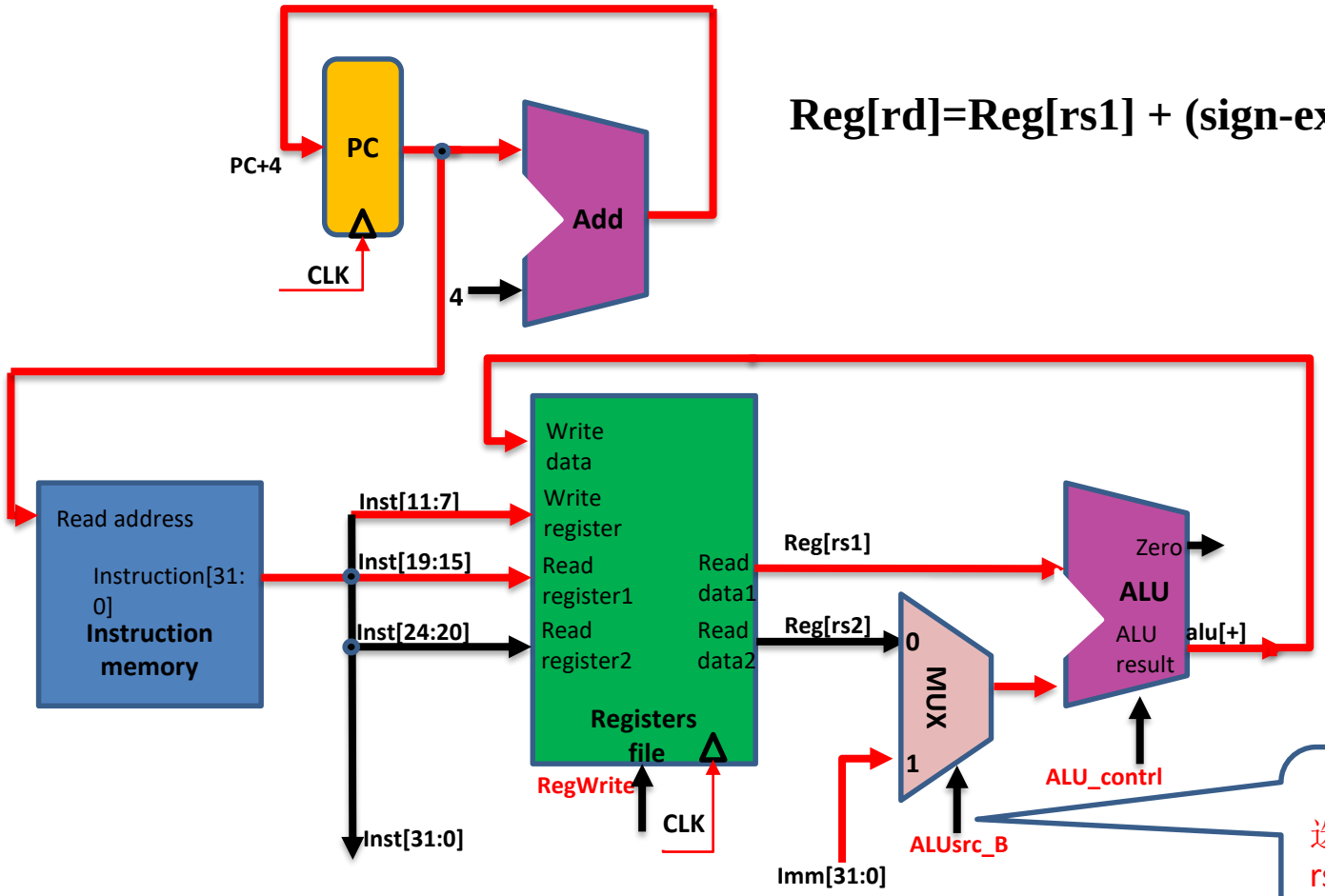
Eg：addi x15,x1,-50

111111001110_00001_ 000 _01111 _0010011

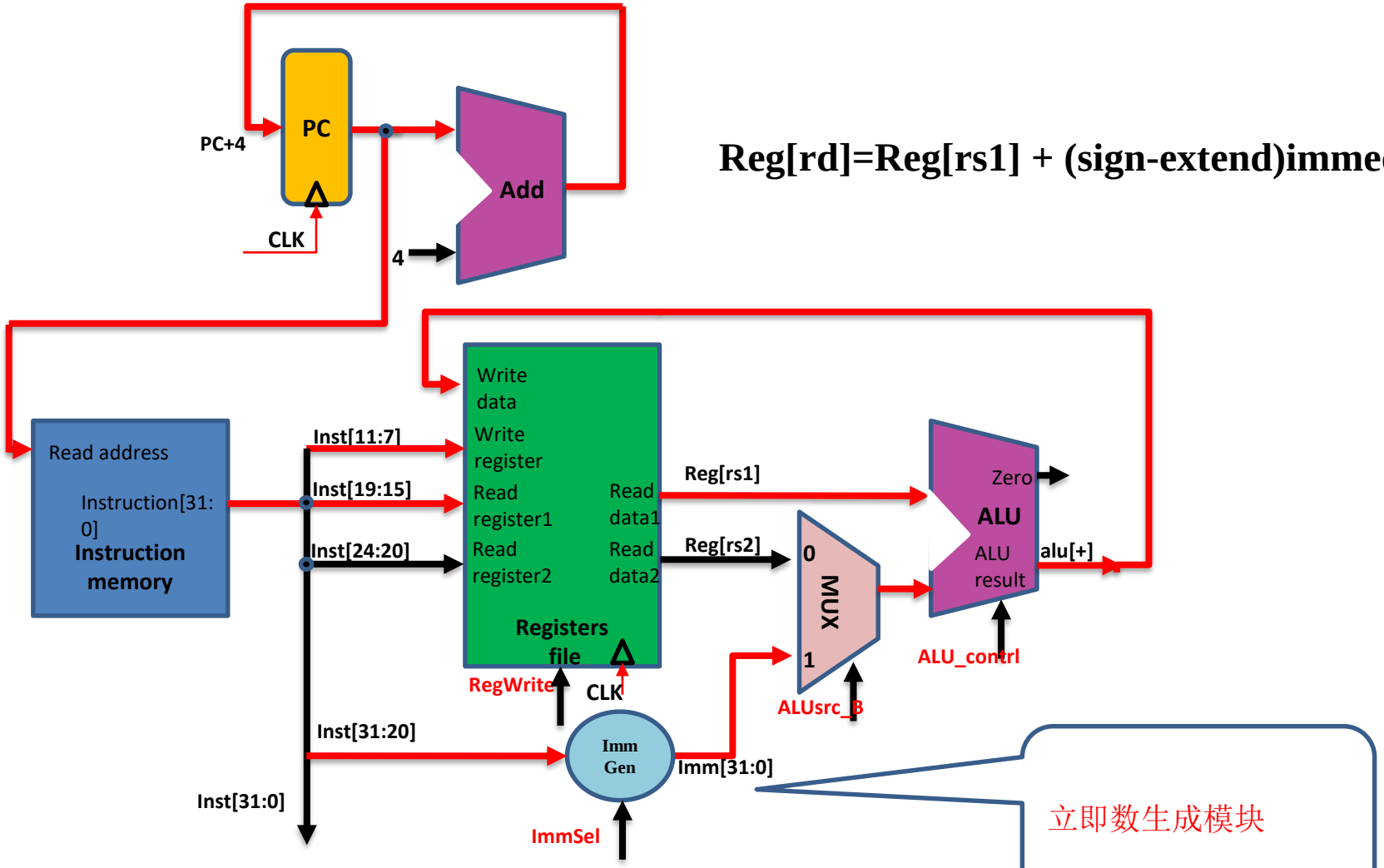
Adding addi to Datapath



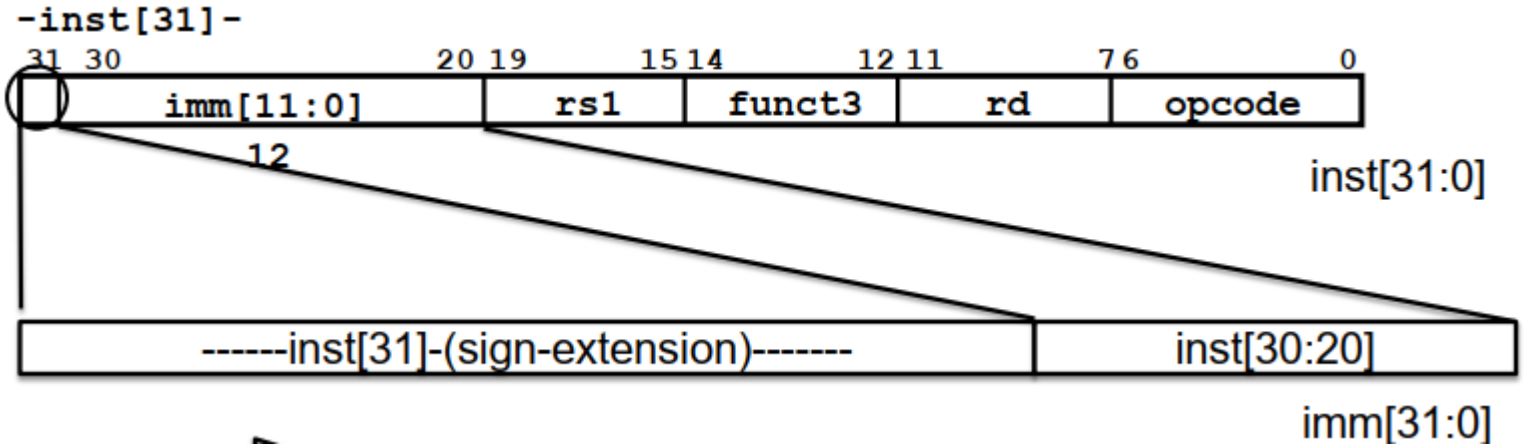
Adding addi to Datapath

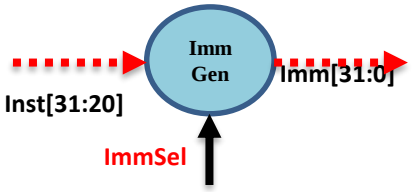


Adding addi to Datapath



I-Format immediates

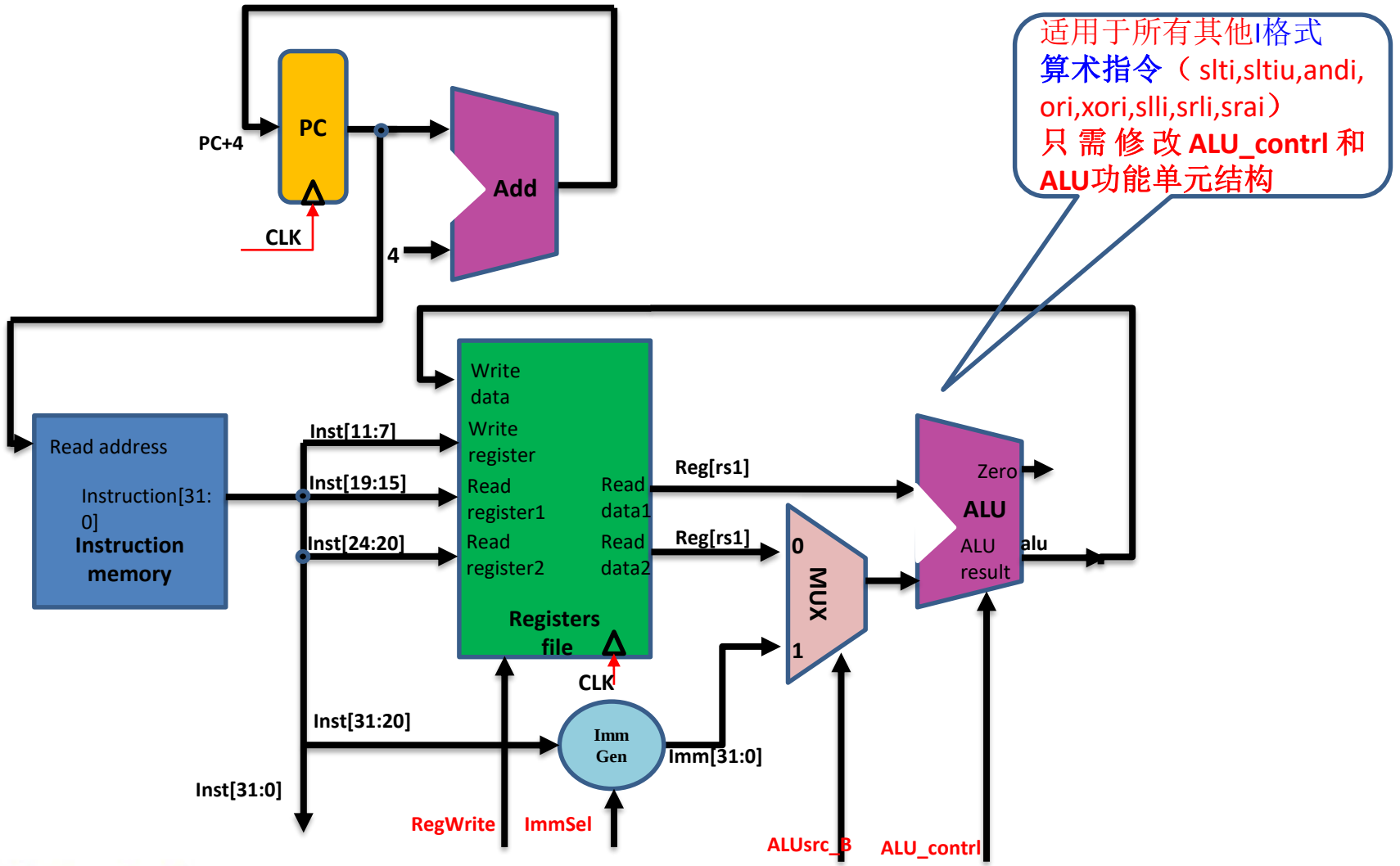




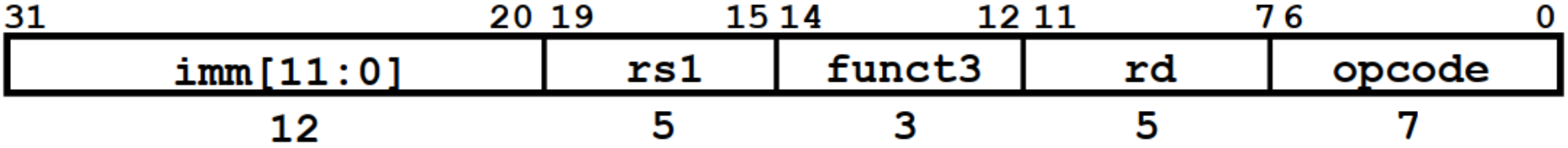
1. High 12 bits of instruction (inst[31:20]) copied
   to low 12 bits of immediate (imm[11:0])

2. Immediate is sign-extended by copying value
of inst[31] to fill the upper 20 bits of the immediate value (imm[31:12])

R+I Datapath


适用于所有其他I格式算术指令（slti,sltiu,andi,ori,xori,slli,srli,srai）只需修改ALU_contrl和ALU功能单元结构
PC+4
PC
CLK
Add
4
Read address
Instruction[31:0]
Instruction memory
Inst[11:7]
Inst[19:15]
Inst[24:20]
Write data
Write register
Read register1
Read register2
Registers file
Read data1
Read data2
CLK
Reg[rs1]
Reg[rs1]
0
MUX
1
Zero
ALU
ALU result
alu
Inst[31:20]
Imm Gen
Imm[31:0]
Inst[31:0]
RegWrite
ImmSel
ALUsrc_B
ALU_contrl

Implementing I-Format - lw instruction



Imm[11:0]	Rs1(5位)	Func3(010)	rd(5位)	0000011 (op)

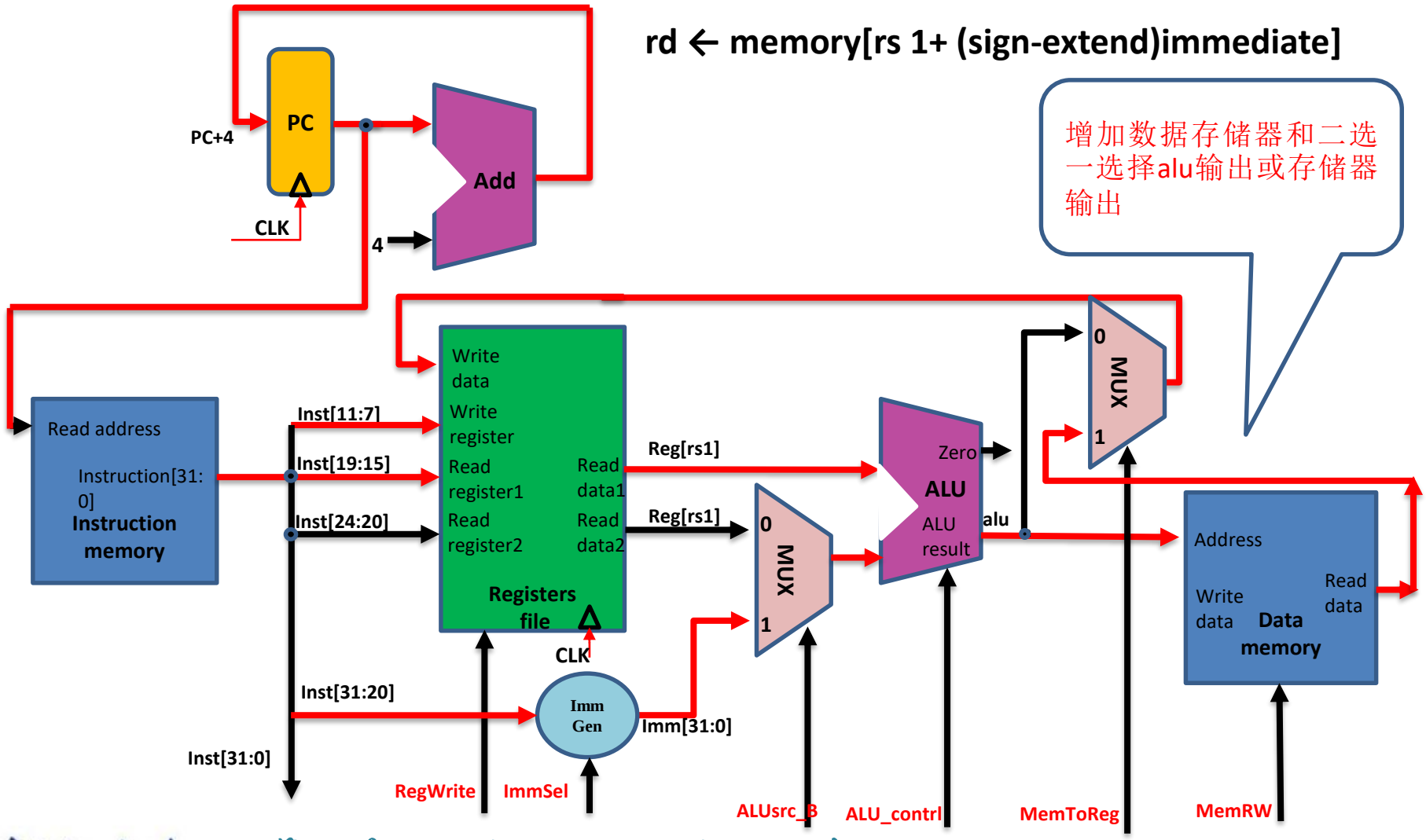
lw rd , immediate(rs1) **读存储器**

功能：rd ← memory[rs 1+ (sign-extend)immediate]；immediate符号扩展再相加。即读取rs1寄存器内容和立即数符号扩展后的数相加作为地址的内存单元中的数，然后保存到rd寄存器中。

Eg lw x9,240(x10) x9 = memory[x10+240]

0000111 10000_01010_010_01001_0000011

Adding lw to Datapath

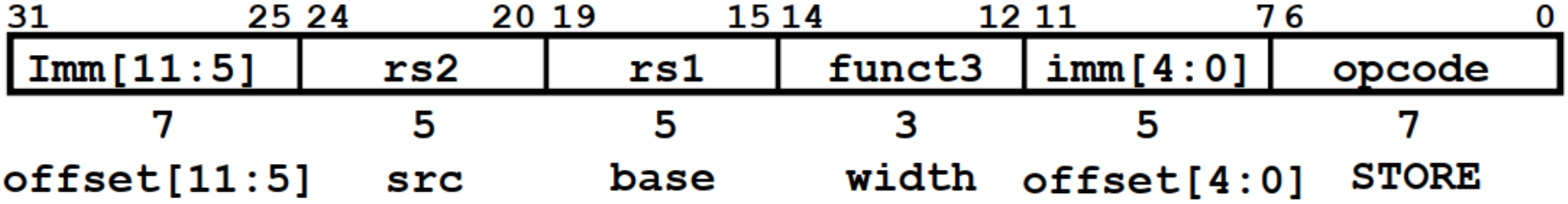


All RV32 Load Instructions

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

funct3 field

Implementing S-Format - sw instruction

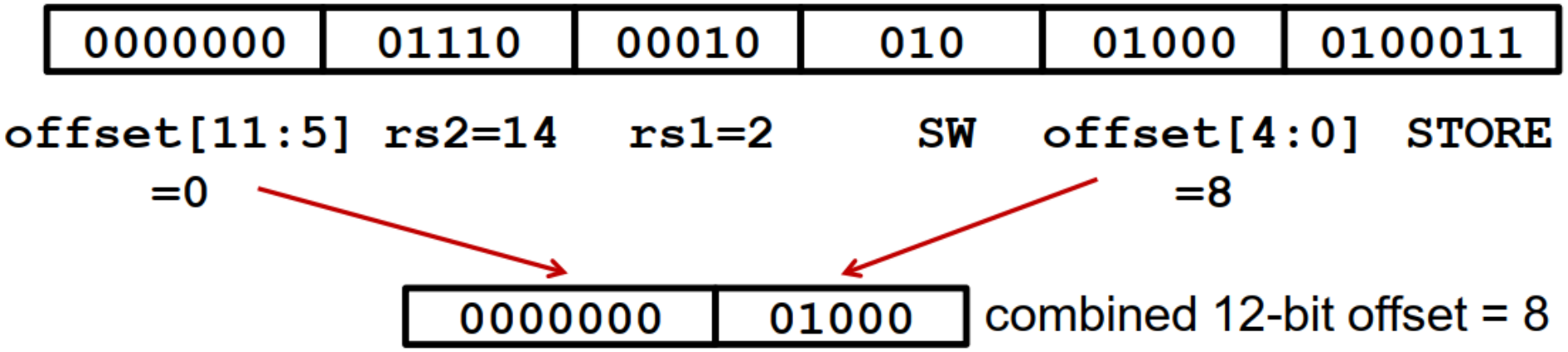


sw rs2 ,immediate(rs1) **写存储器**

功能：memory[rs1+ (sign-extend)immediate]←rs2；immediate符号扩展再相加。即将rs2寄存器的内容保存到rs1寄存器内容和立即数符号扩展后的数相加作为地址的内存单元中。

Eg: sw x14, 8(x2)

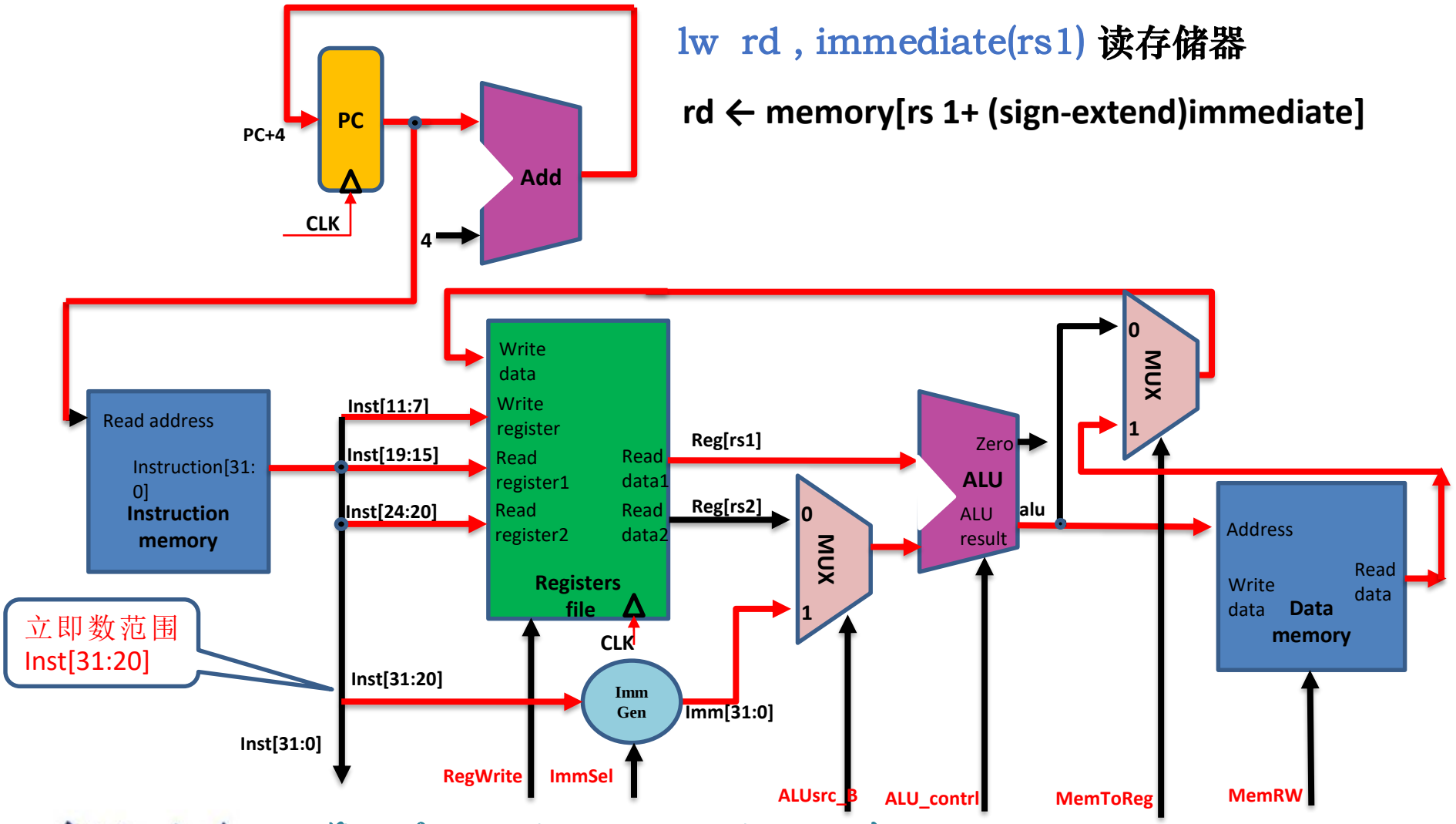
0000000 _01110_00010_010_01000_0100011



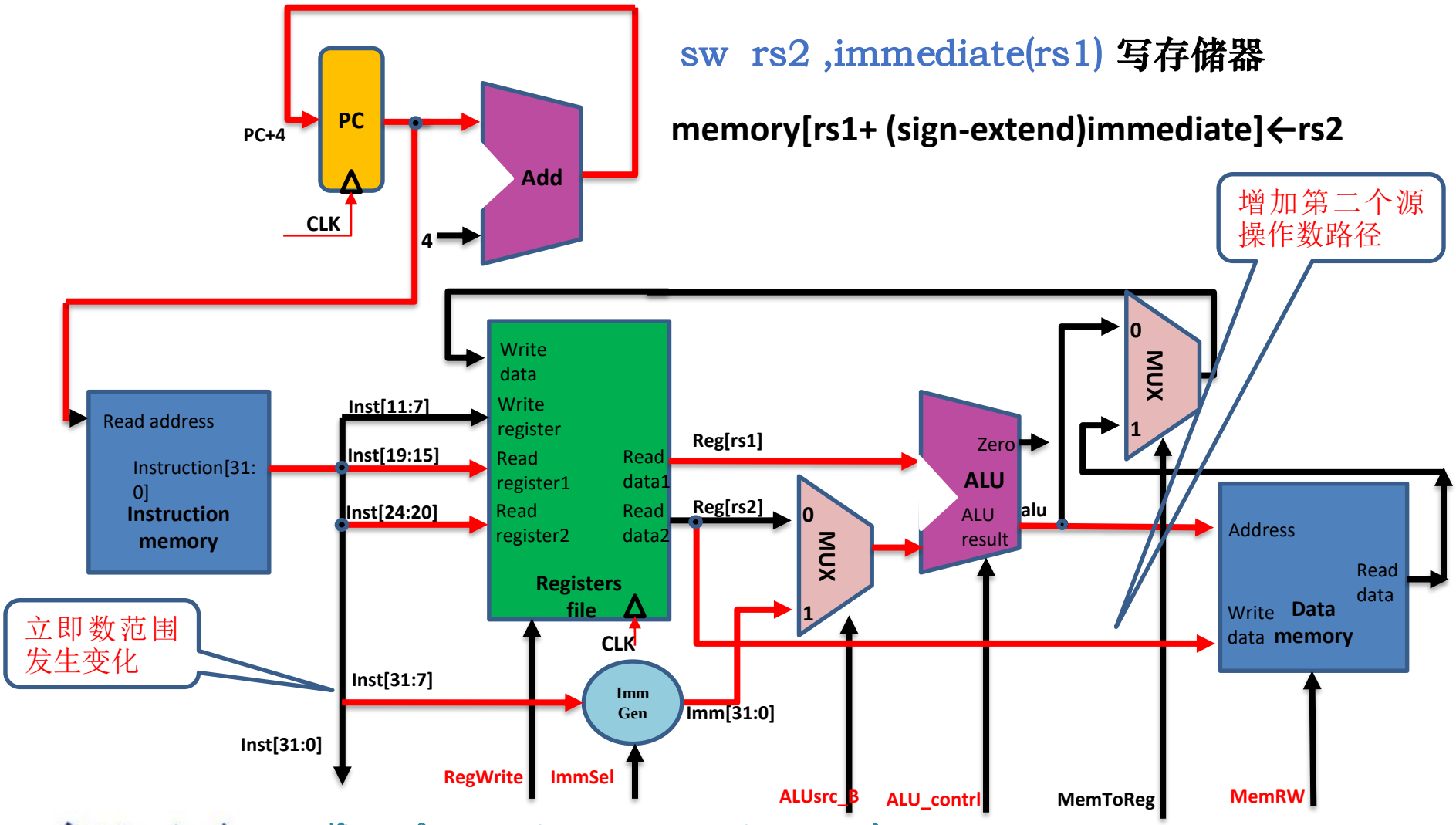
Datapath with lw

lw rd , immediate(rs1) 读存储器

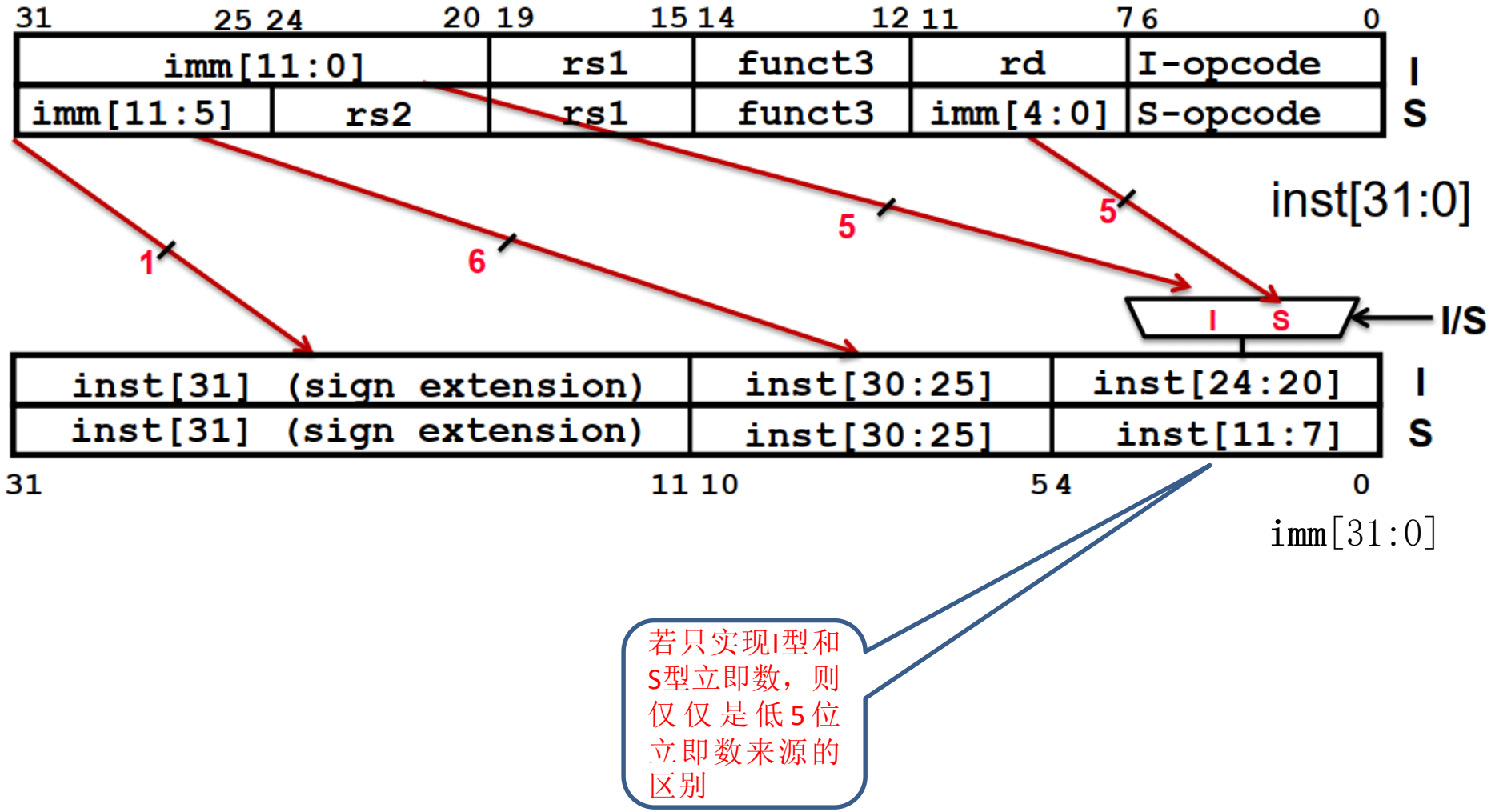
rd ← memory[rs 1+ (sign-extend)immediate]



Adding sw to Datapath

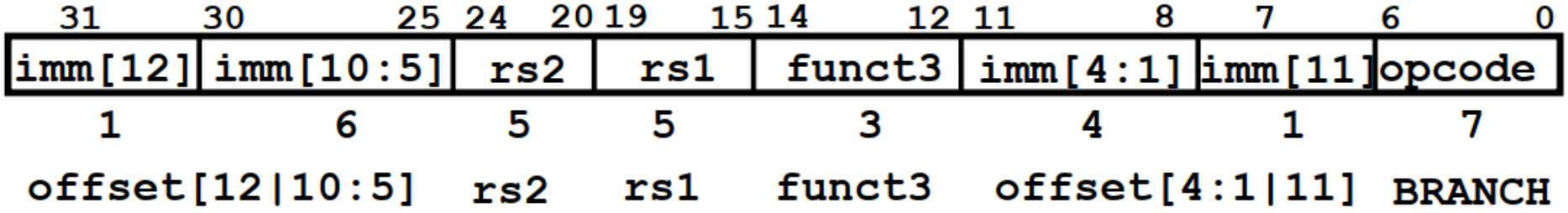


I+S Immediate Generation



若只实现I型和S型立即数，则仅仅是低5位立即数来源的区别

Implementing Branches(B-Format)



Imm[12|10:5]	Rs2(5位)	Rs1(5位)	Func3(000)	Imm[4:1|11]	1100011（op）

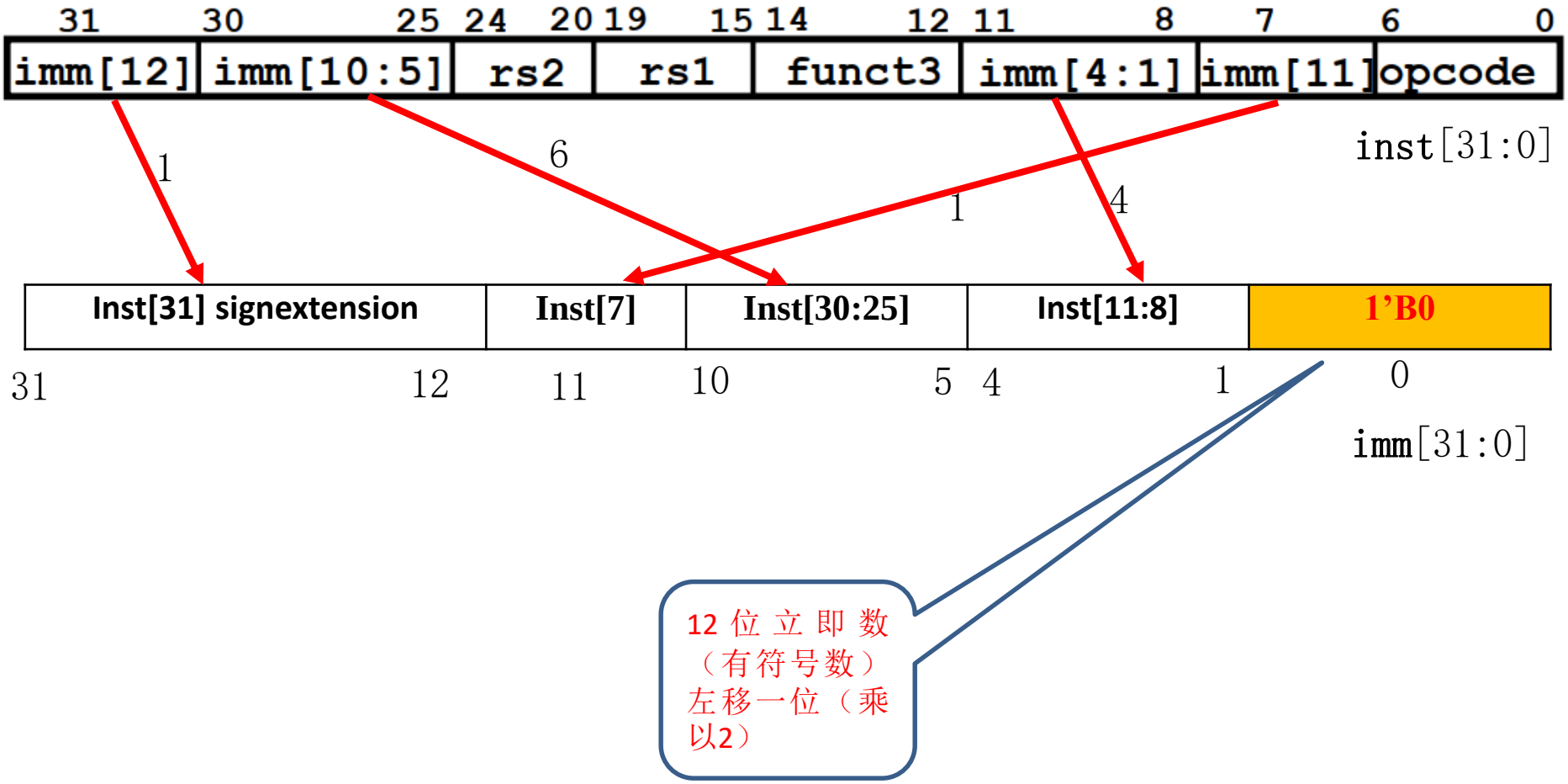
beq rs1,rs2,immediate　　相等则跳转

功能：if(rs1=rs2) pc←pc + (sign-extend)immediate<<1 else pc ←pc + 4

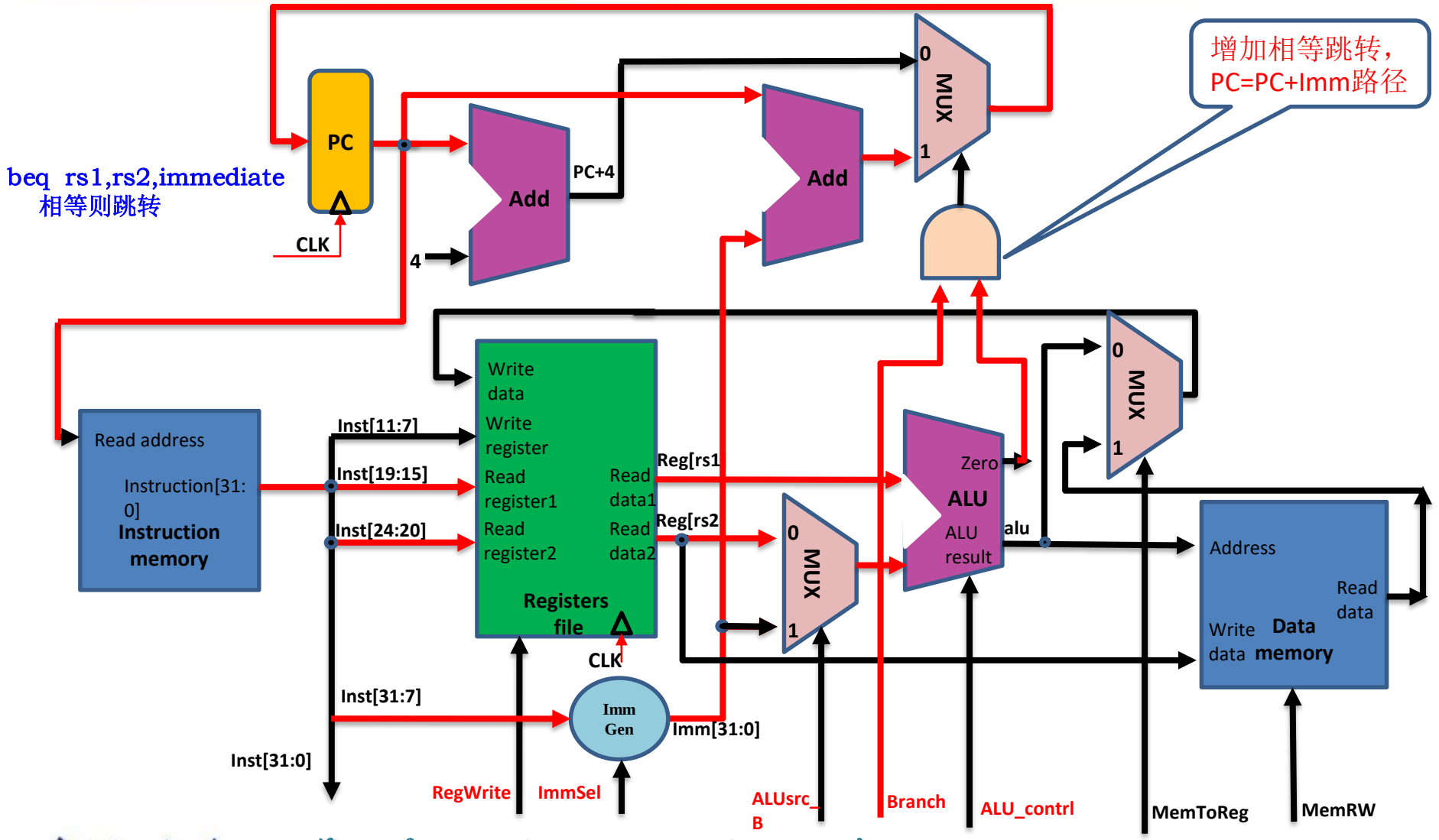
Eg　beq x5,x6,100

0000 0110 0110 0010 1000 0010 0110 0011

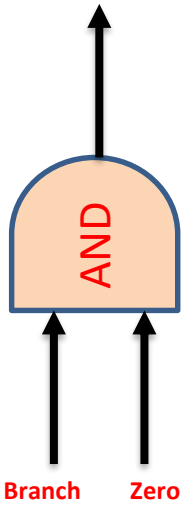
Branches Immediate Generation



Adding branches to Datapath

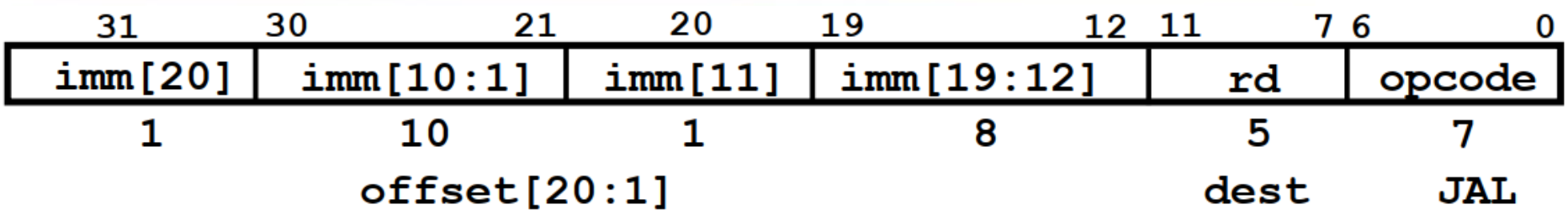


Branch Comparator



- Zero来自ALU的全零判断输出，为1，则源操作数寄存器rs1和rs2的数值相等
- Branch来自控制器的输出，解码指令为分支指令
- 当二者相与操作，输出为1，达到Beq的跳转条件

Adding JAL (J-Format)



imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd(5位)	1101111（op）

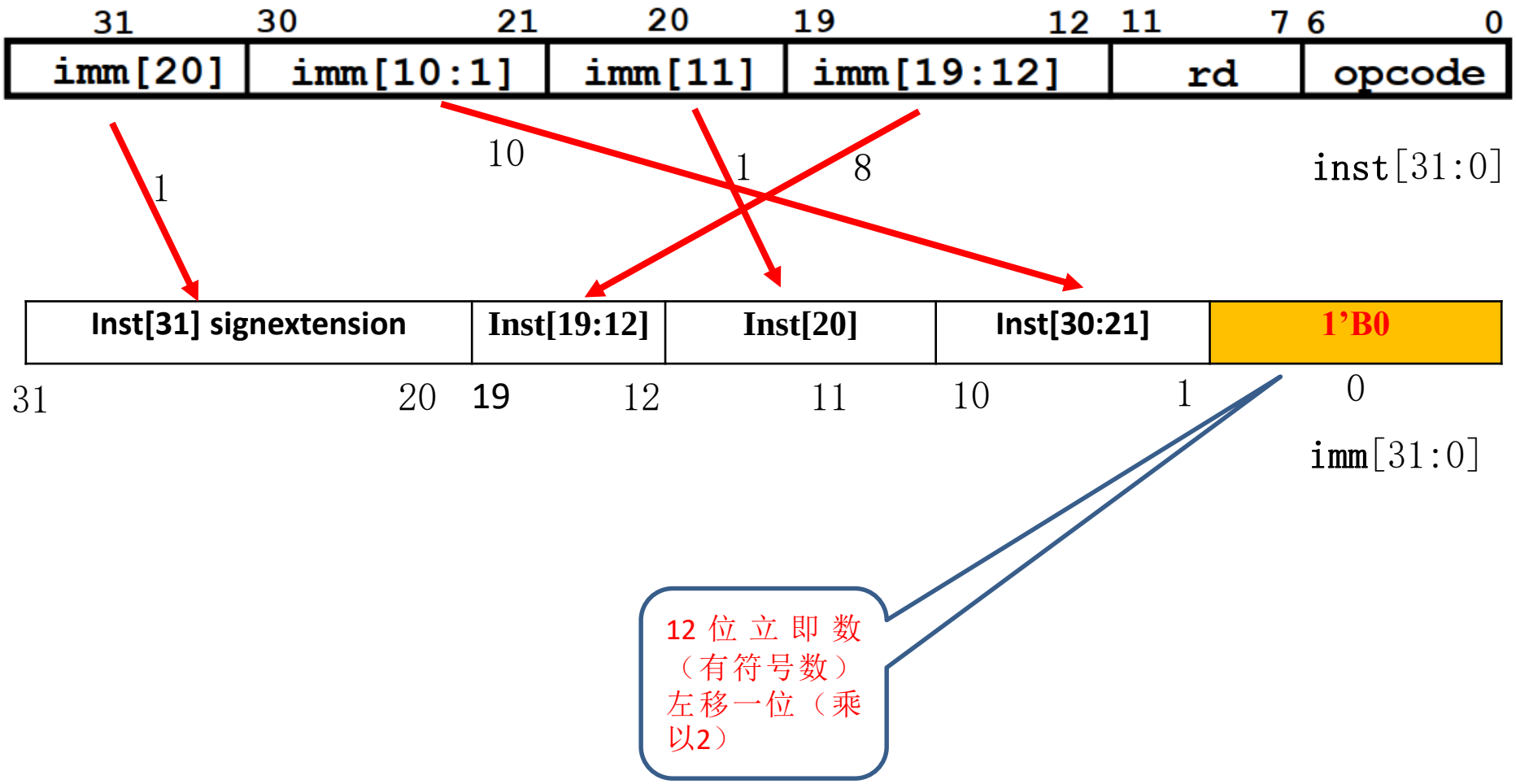
jal rd, immediate　跳转链接

功能：rd = pc+4， pc←pc + (sign-extend)immediate<<1

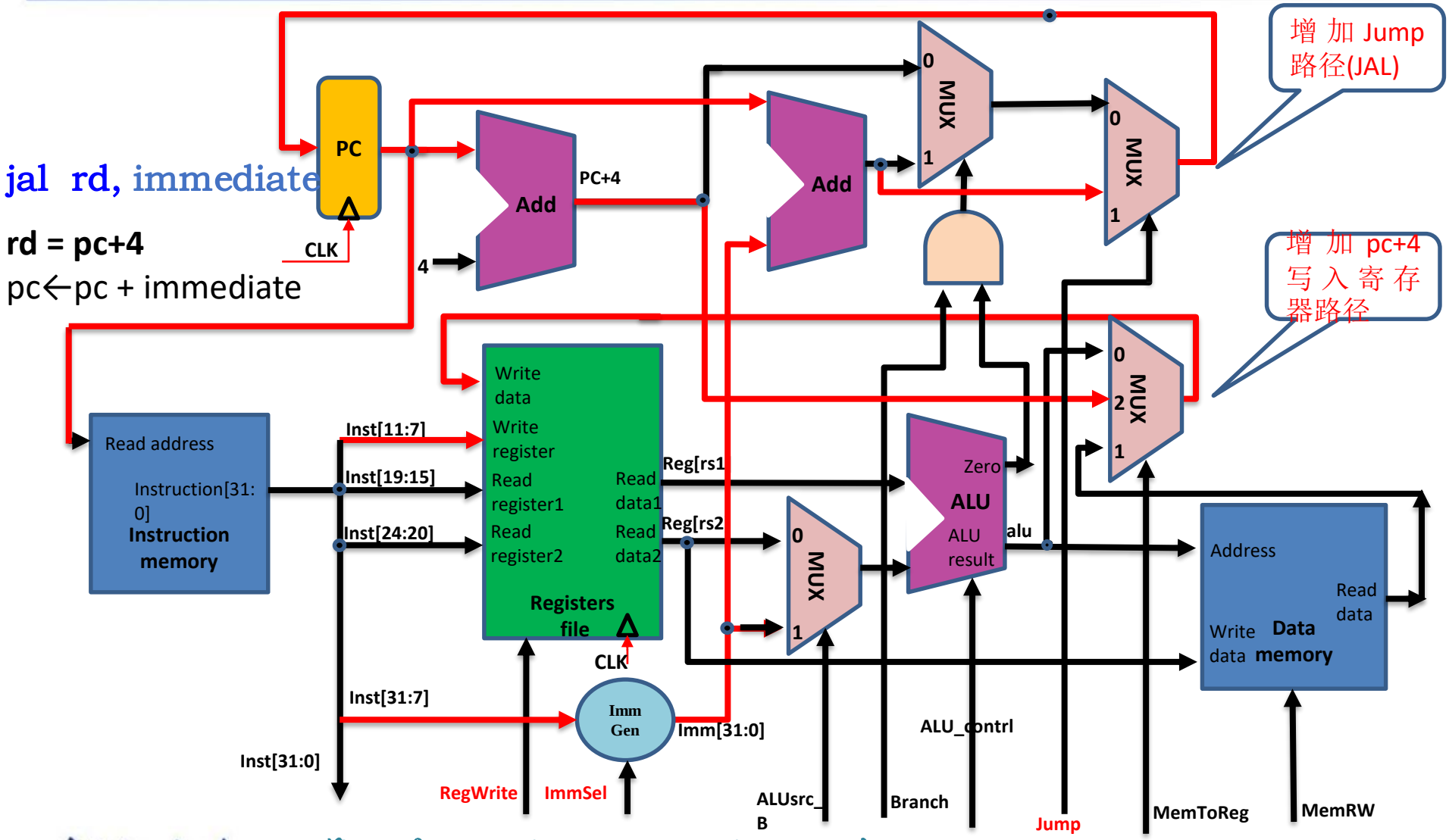
Eg　jal x1,100

0000 0110 0100 0000 0000 0000 1110 1111

JAL Immediate Generation



Adding JAL to Datapath



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）		