Computer Organization & Design实验与课程设计

Lab4-1
CPU设计—数据通路

Ma De（马德）

made@zju.edu.cn

2022

College of Computer Science, Zhejiang University

Course Outline

- 一、实验目的
- 二、实验环境
- 三、实验目标及任务

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. Sword 2.0/Sword4.0开发板

3. Xilinx VIVADO2017.4及以上开发工具

材料

无

实验目标及任务

- 目标：熟悉RISC-V RV32I的指令特点，了解数据通路的原理，设计并测试数据通路

- 任务一：设计实现数据通路（采用RTL实现）
  - ALU和Regs调用Exp01设计的模块（可直接加RTL）
  - PC寄存器设计及PC通路建立
  - ImmGen立即数生成模块设计
  - 此实验在Exp4-0的基础上完成，替换Exp4-0的数据通路核
- 任务二：设计数据通路测试方案并完成测试
  - 通路测试：I-格式通路、R-格式通路
  - 部件测试：ALU、Register Files

RISC-V RV32I数据通路的原理介绍

- 实现不少于下列指令

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

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

S-Type： sw；

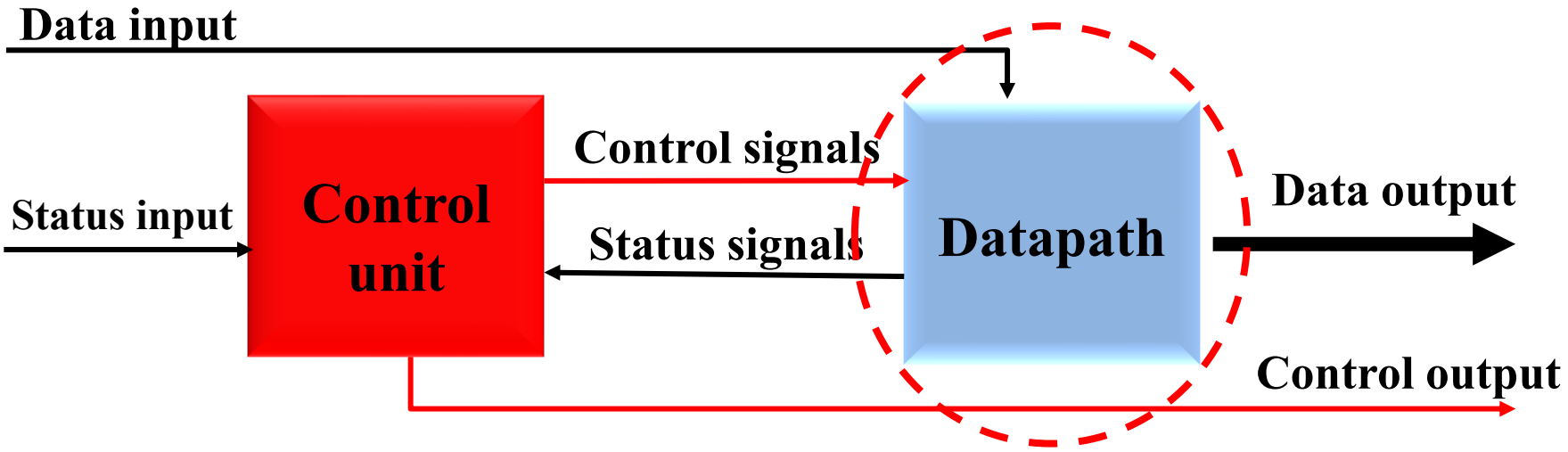
B-Type： beq；

J-Type： Jal；

CPU organization

Digital circuit

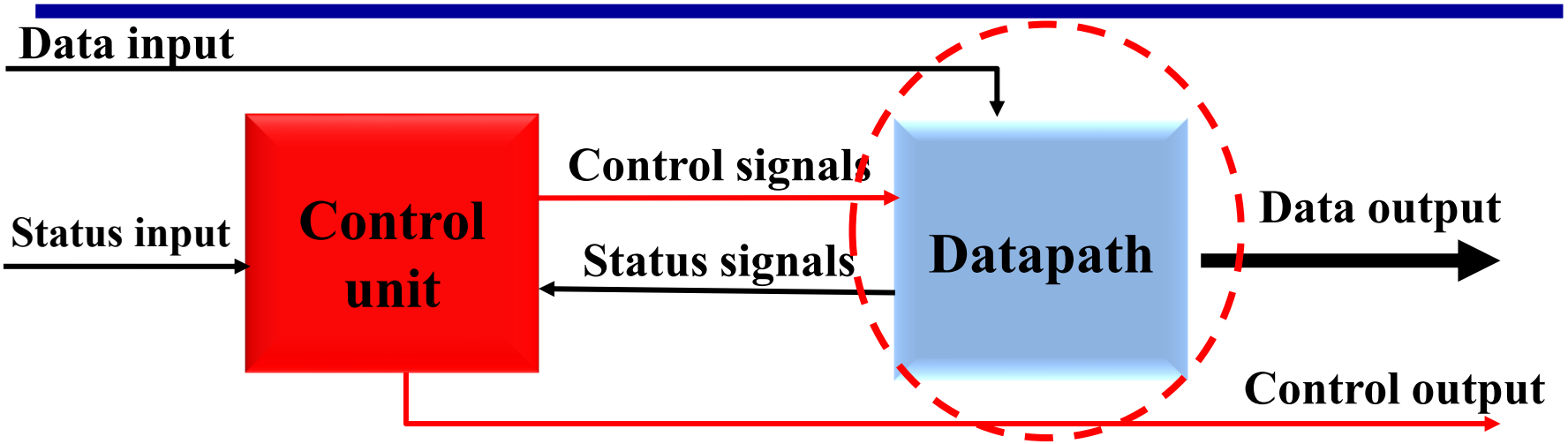
- General circuits that controls logical event with logical gates --Hardware



Computer organization

- Special circuits that processes logical action with instructions -Software

Datapath

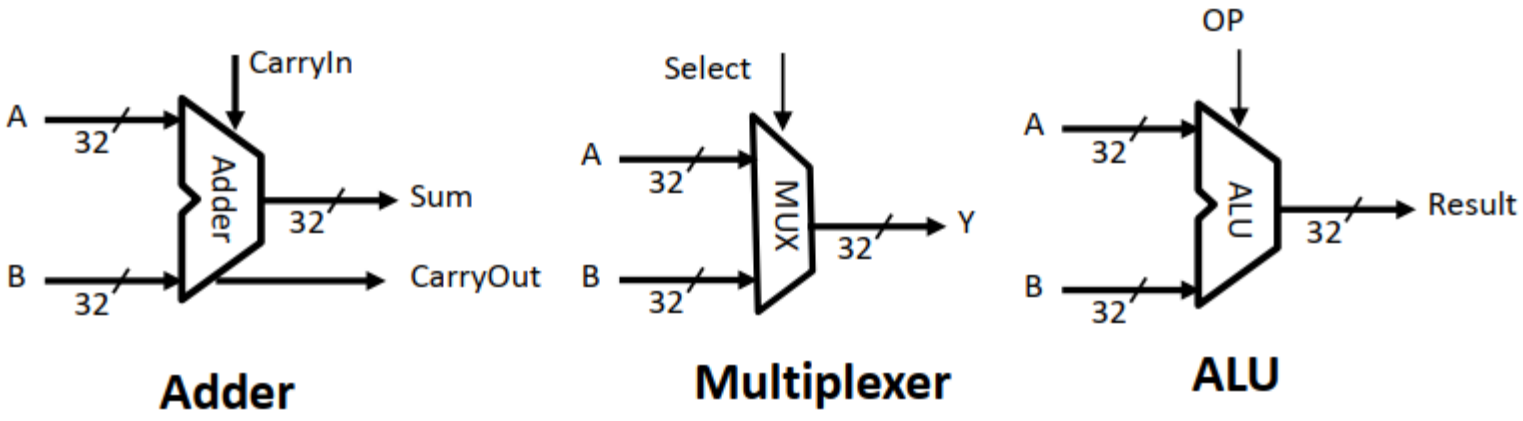


- 数据通路
  - 数据通路作为处理器的一部分，包含了完成处理器所要求的的操作所必须的硬件，包括运算单元、寄存器组、状态寄存器等

Datapath

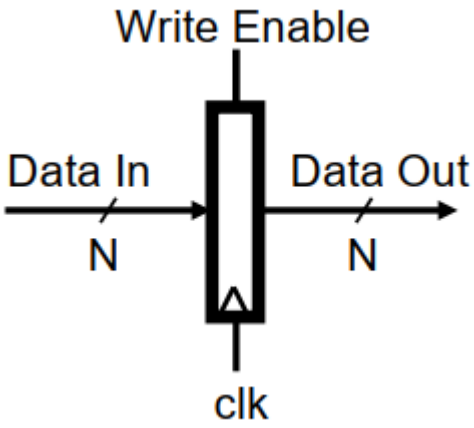
数据通路部件：

- 组合逻辑单元： 加法器、多路选择器、**ALU**算数运算单元



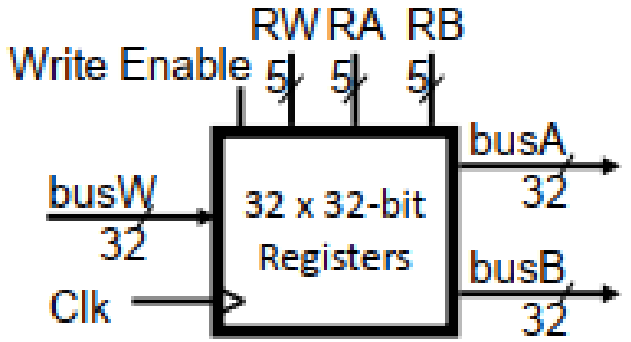
Datapath

- 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

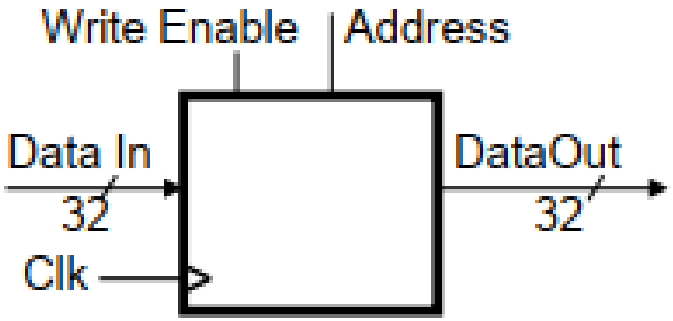


  - **寄存器读写操作:**
    - RA (number) 作为地址选择register存储的数据传输到输出端口busA（读）
    - RB (number) 作为地址选择register存储的数据传输到输出端口busB（读）
    - RW (number) 作为地址选择register接收输入端口busW的数据，当Write Enable=1且时钟边沿有效时（写）

  - **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）

Datapath

State Required by RV32I ISA

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

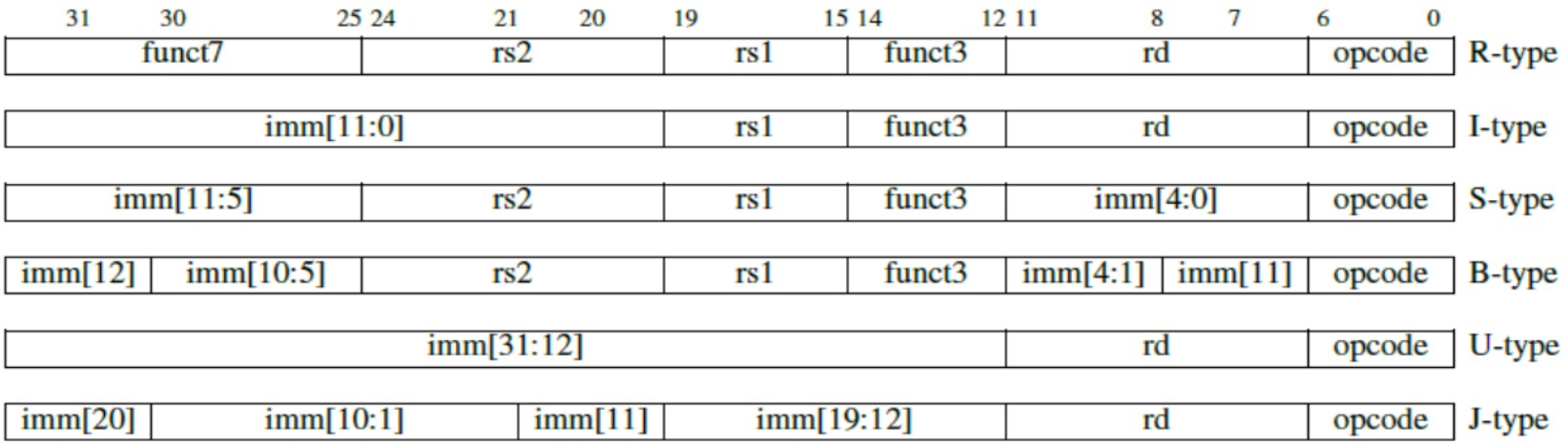
RV Instructions

- 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。

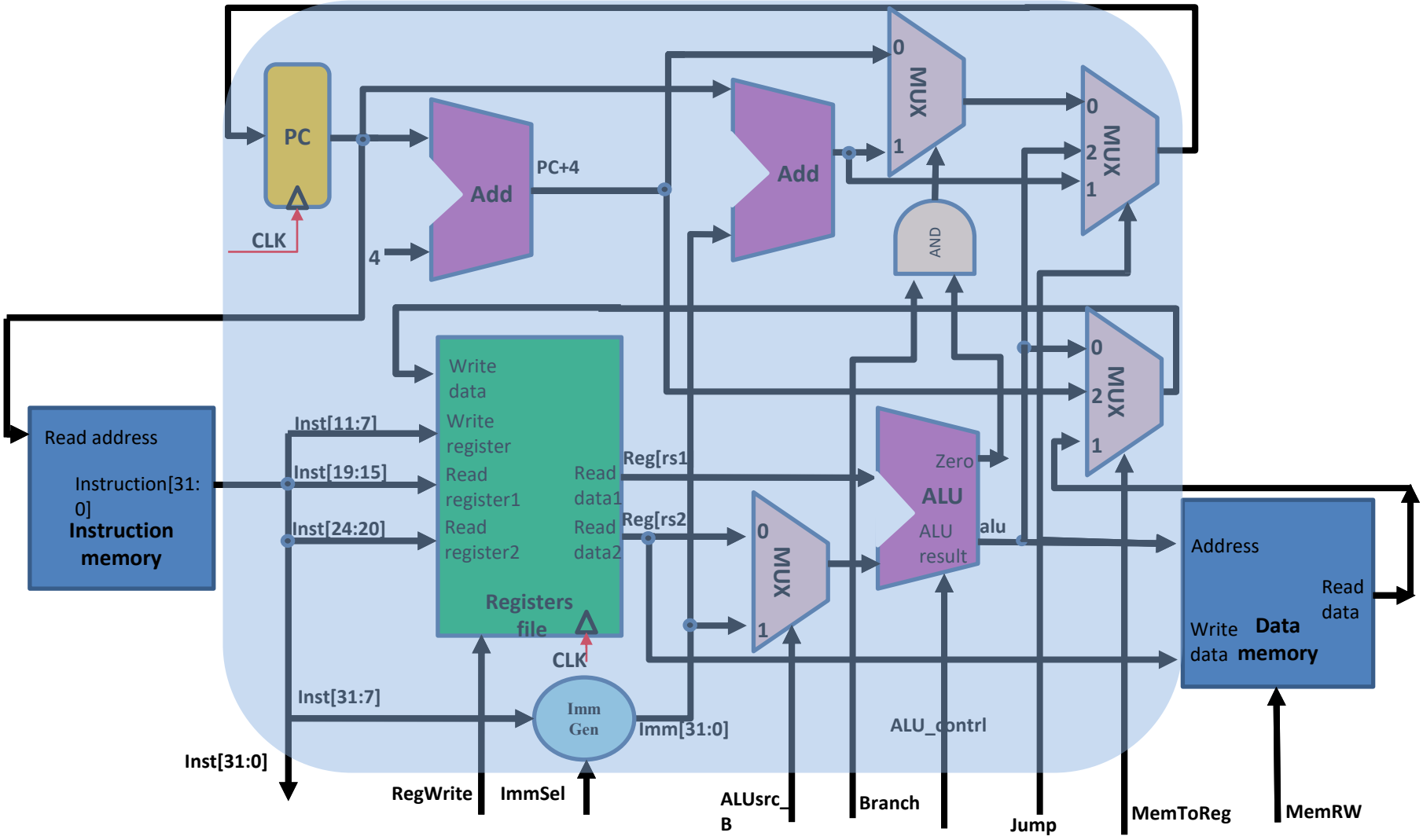
(2) 指令译码(ID)：对取指令操作中得到的指令进行分析并译码，确定这条指令需要完成的操作，从而产生相应的操作控制信号，用于驱动执行状态中的各种操作。

(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

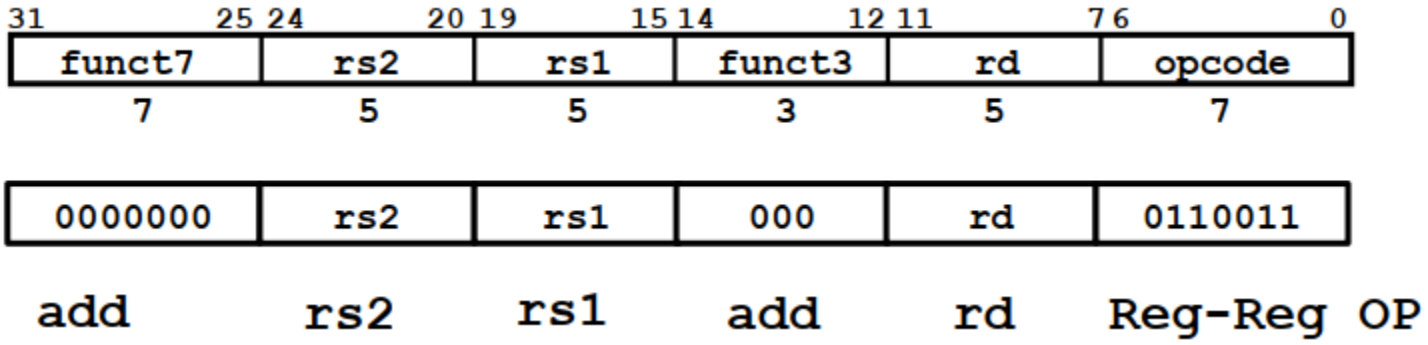
PC：程序计数器，存放指令地址

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



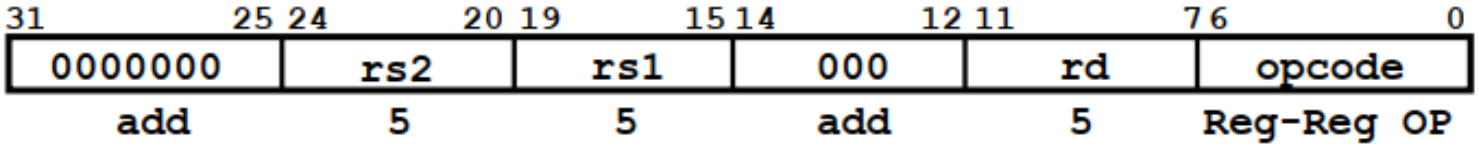
add rd , rs1, rs2

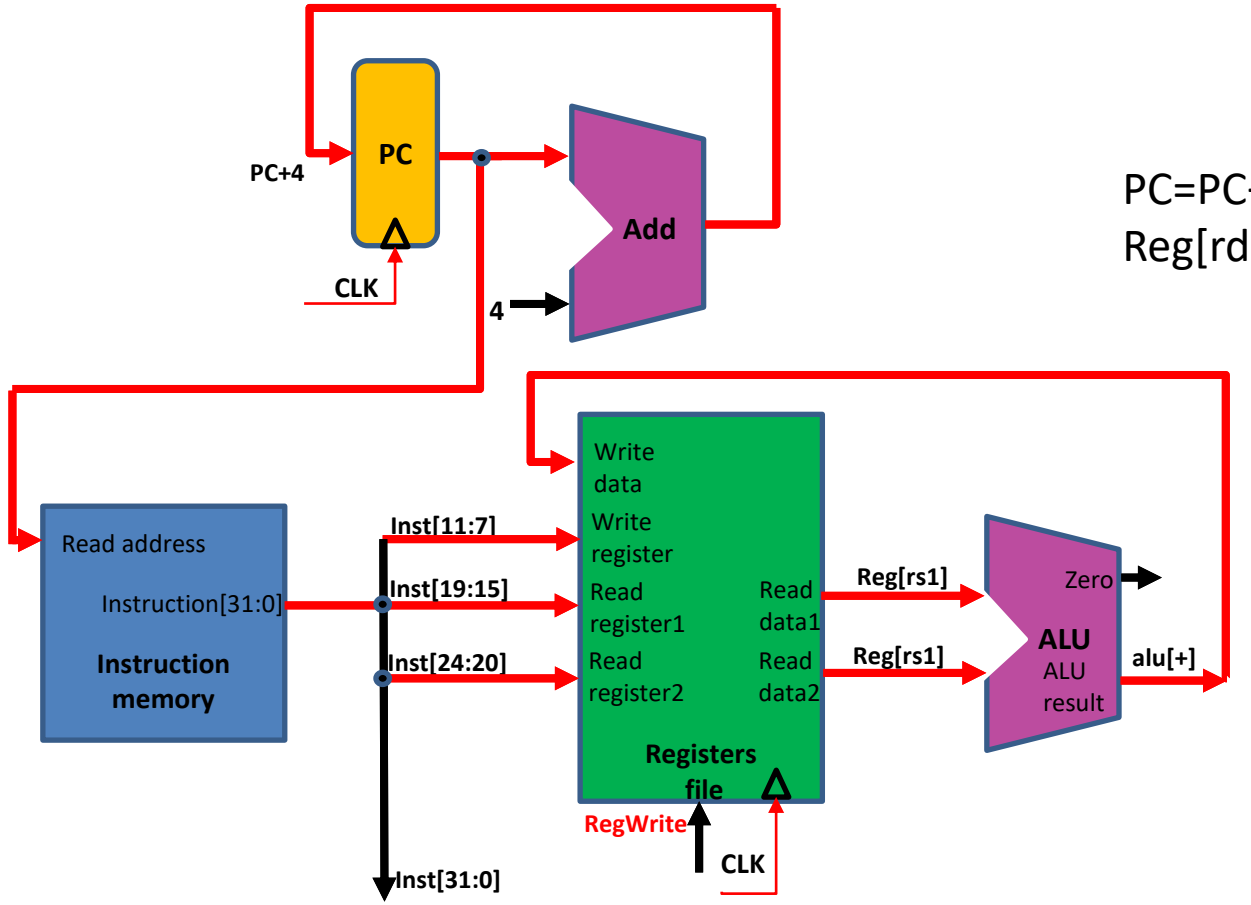
功能：rd←rs1 + rs2。

Eg：add x9,x21,x9

0000000 _01001_10101_000_01001_0110011

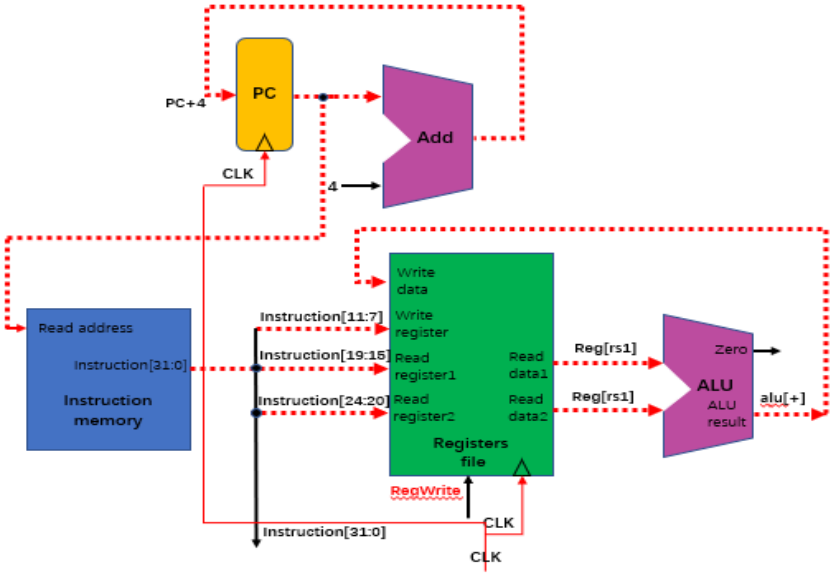
Datapath for add

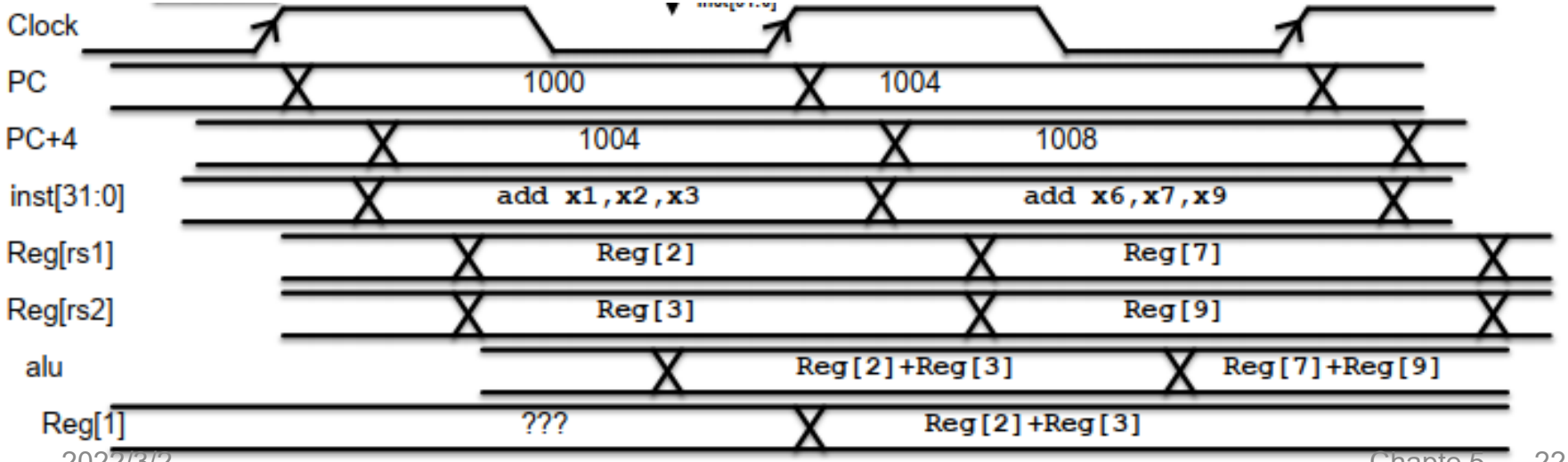




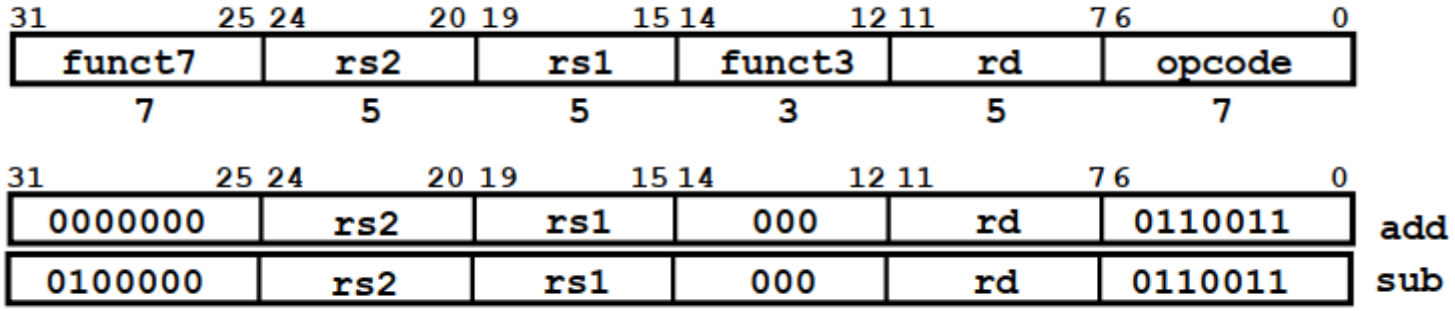
PC=PC+4
Reg[rd]=Reg[rs1]+Reg[rs2]

Timing Diagram for add





Implementing the sub Instruction



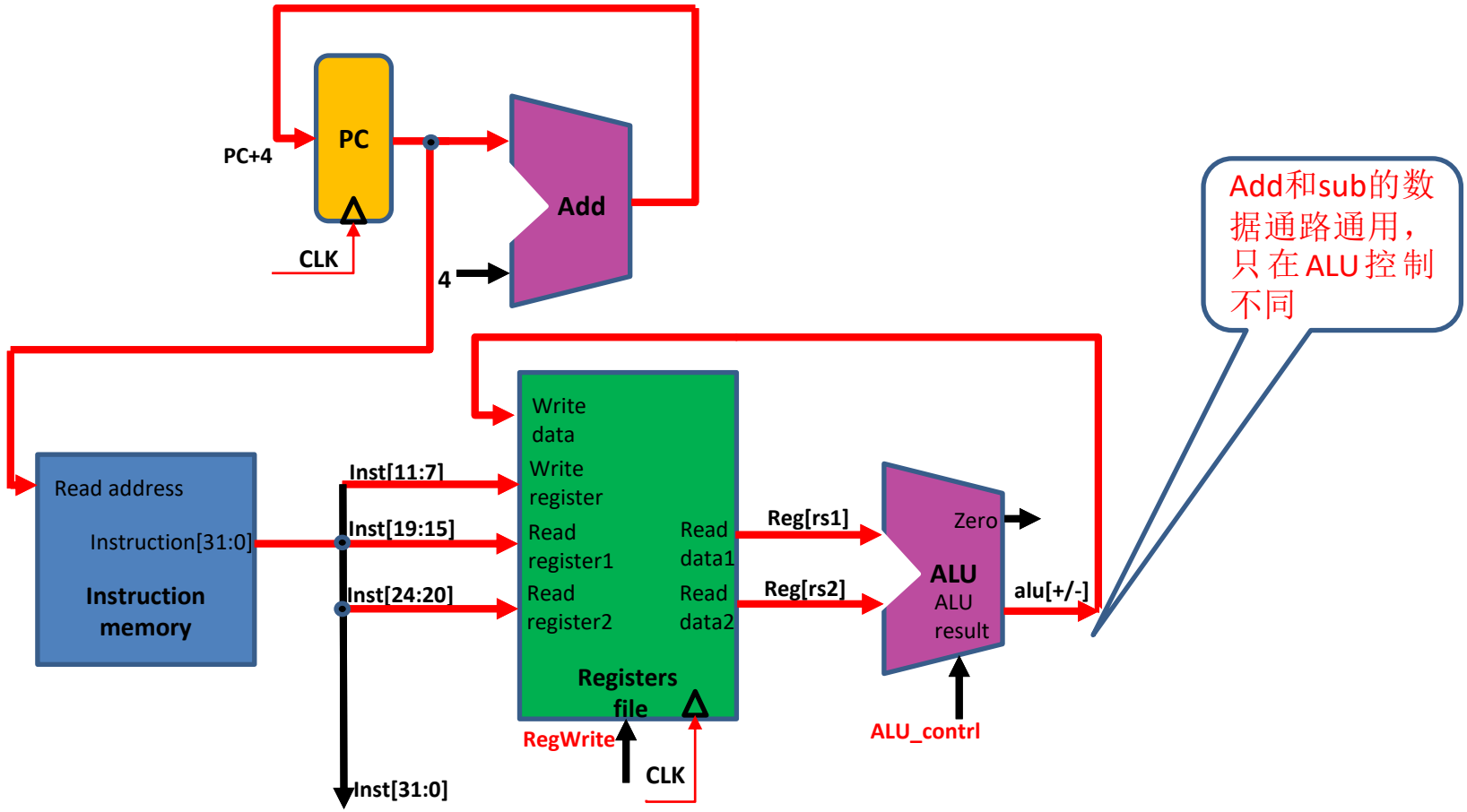
sub rd , rs1 , rs2

功能：rd←rs1 – rs2

Eg：sub x9,x21,x9

0100000 _01001_10101_000_01001_0110011

Datapath for sub/add



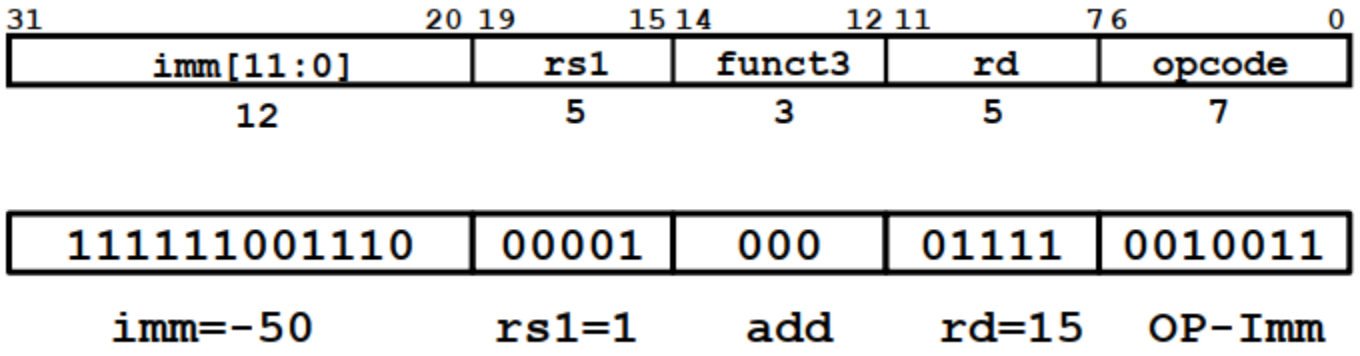
Implementing other R-Format instructions

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

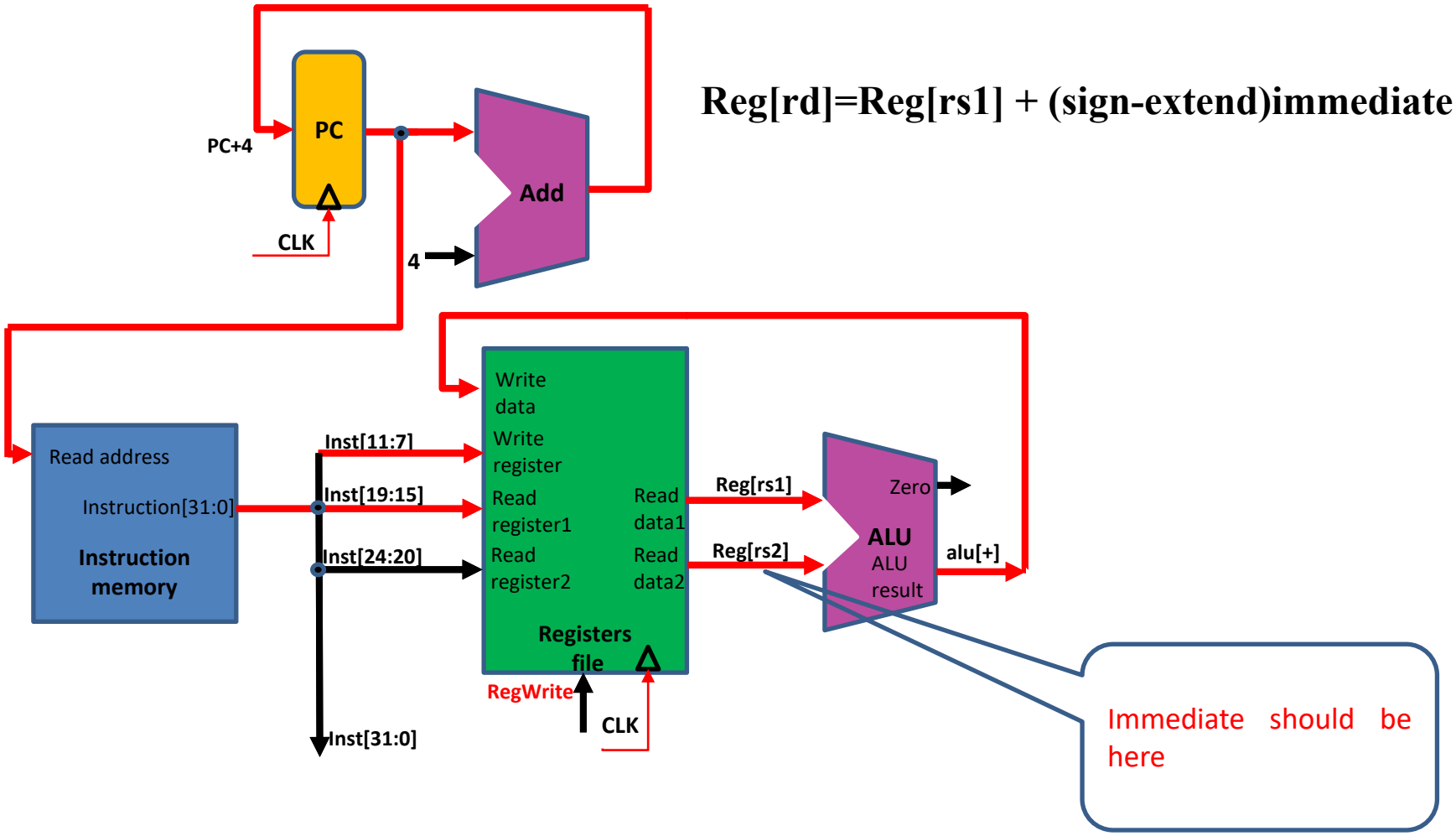


addi rd , rs1 ,immediate

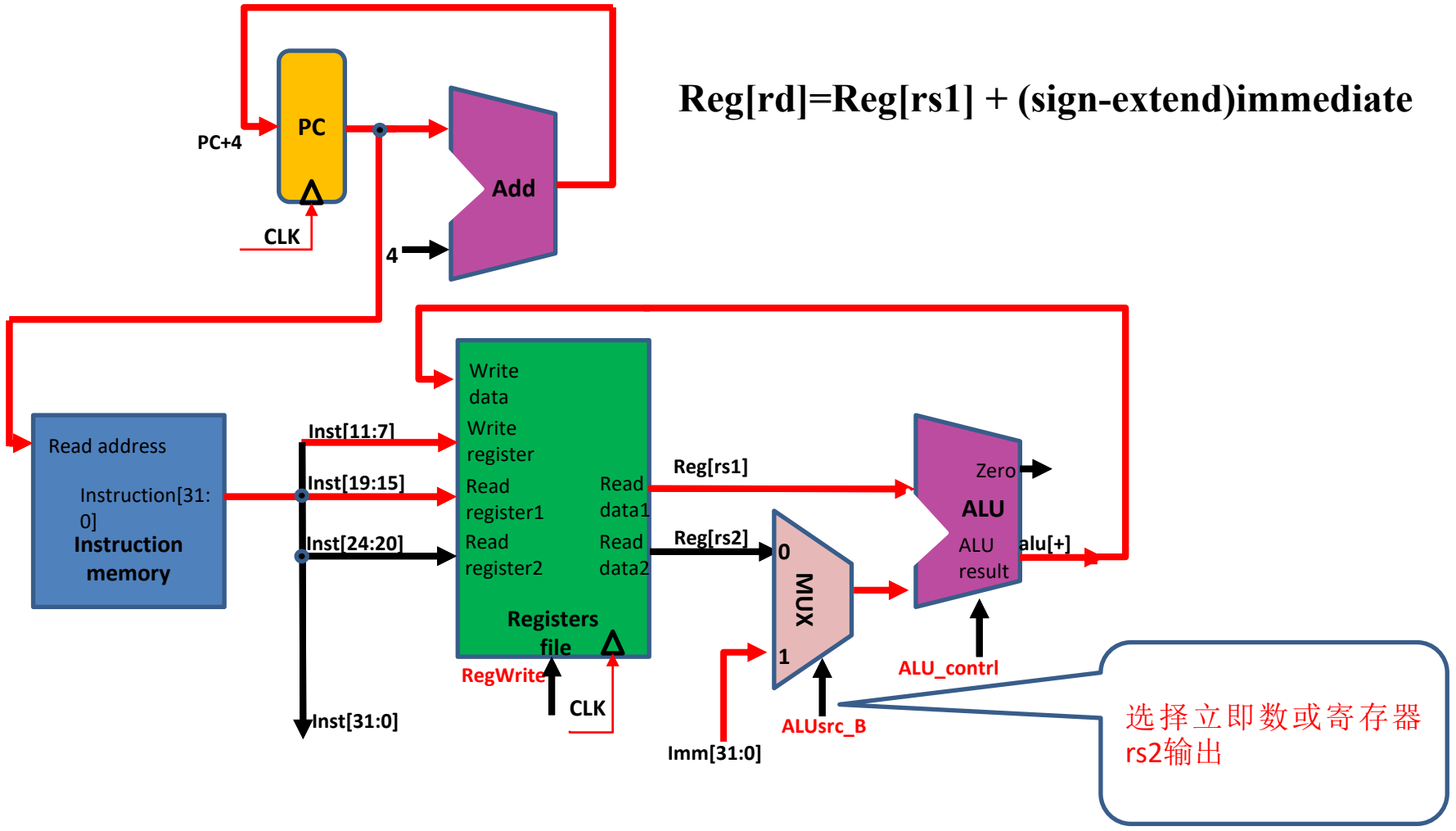
功能：rd←rs1 + (sign-extend)immediate；12位immediate符号扩展再参与“加”运算。

Eg：addi x15,x1,-50
111111001110_00001_ 000 _01111 _0010011

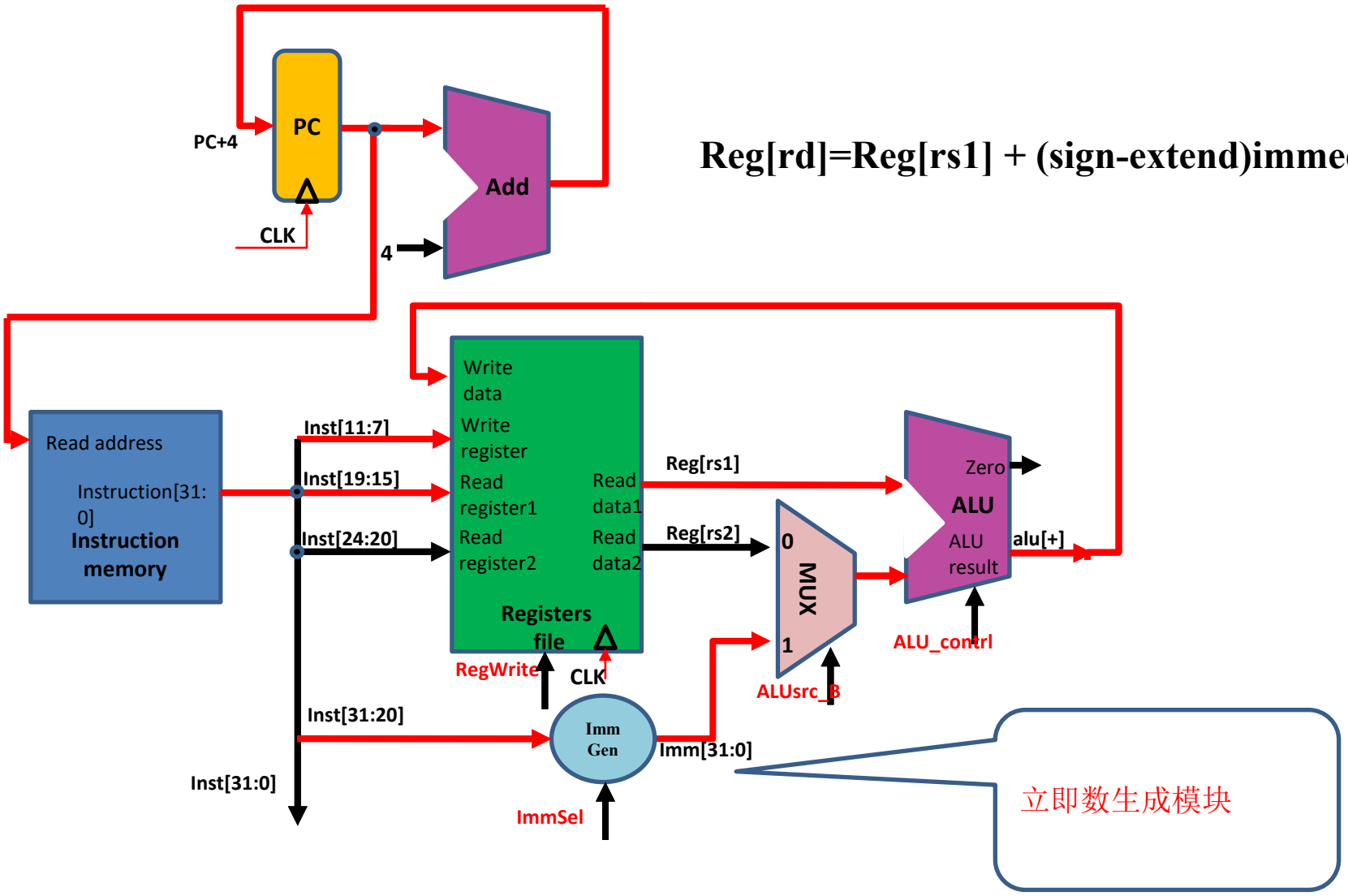
Adding addi to Datapath



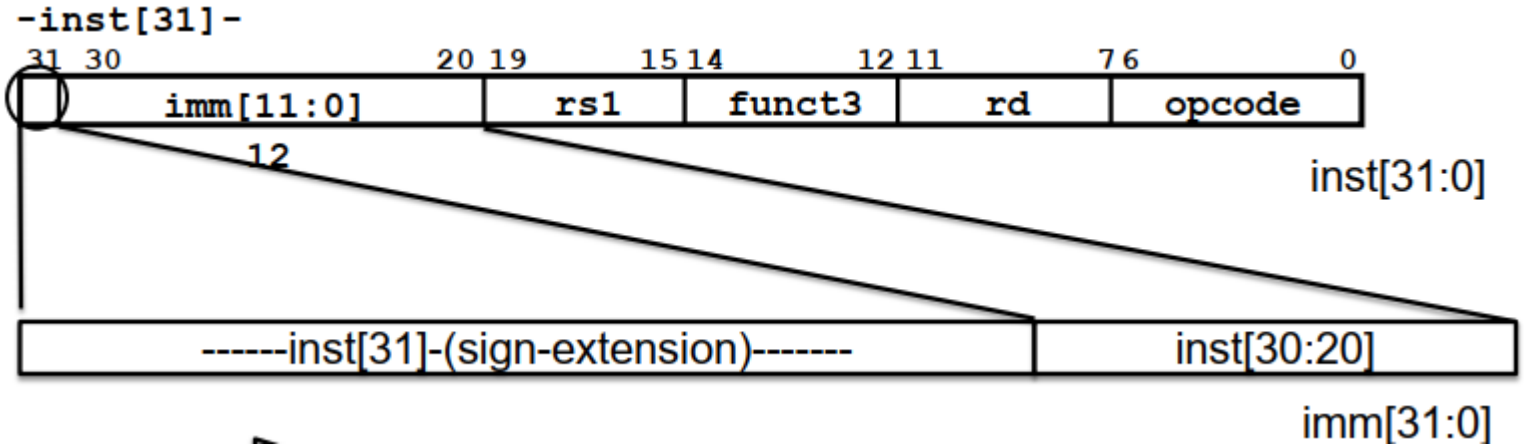
Adding addi to Datapath

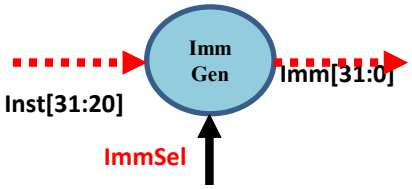


Adding addi to Datapath



I-Format immediates

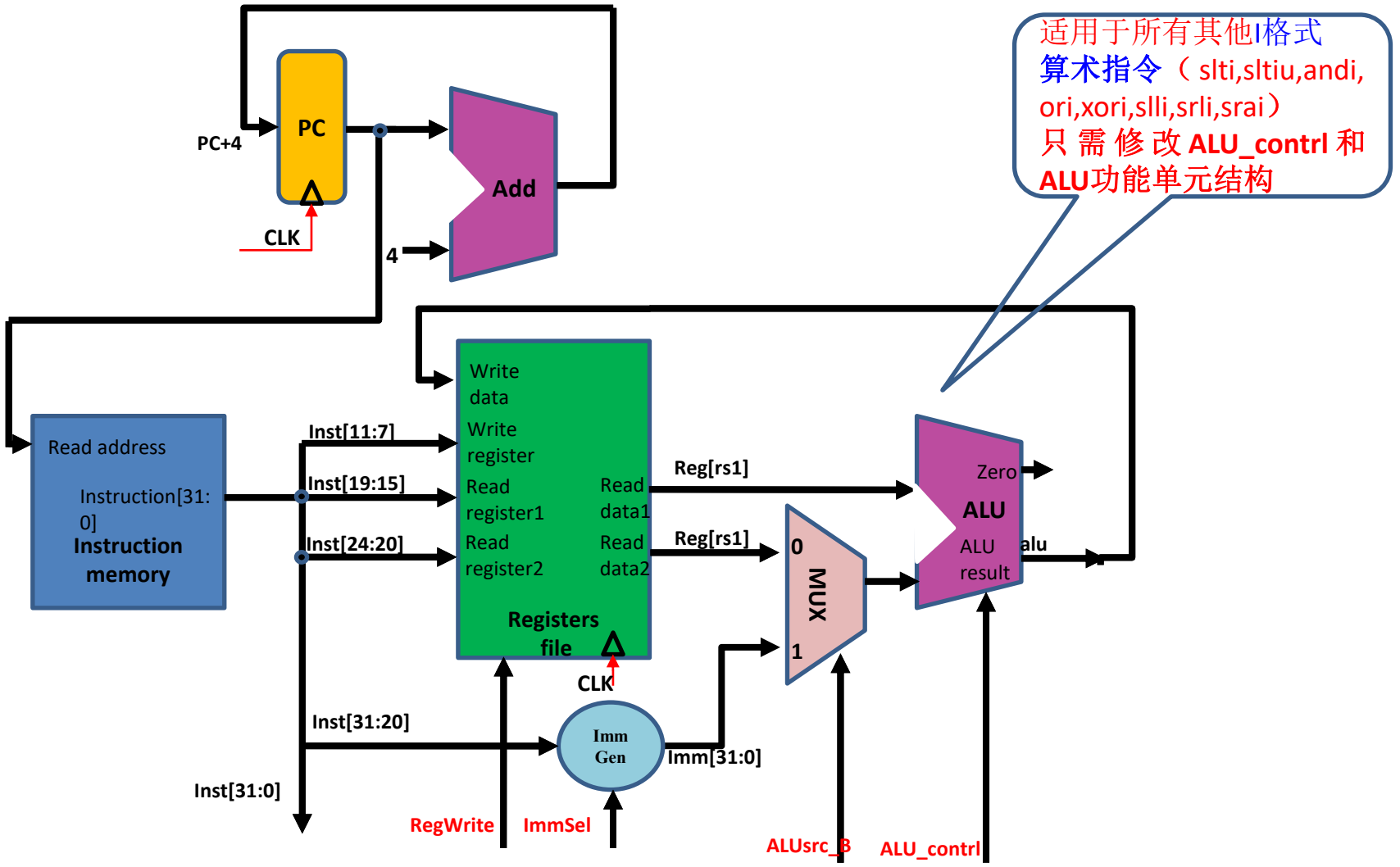




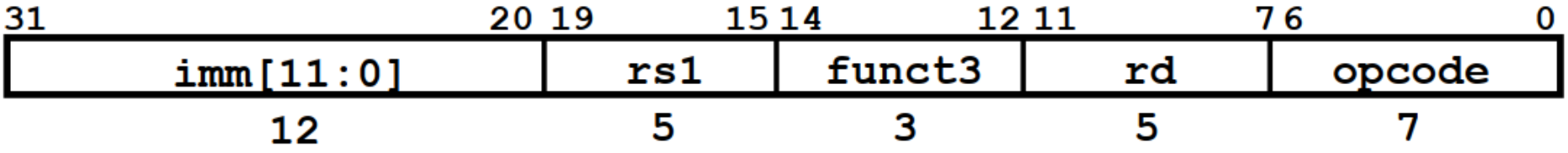
1. **High 12 bits of instruction (inst[31:20]) copied to low 12 bits of immediate (imm[11:0])**

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
PC+4
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)

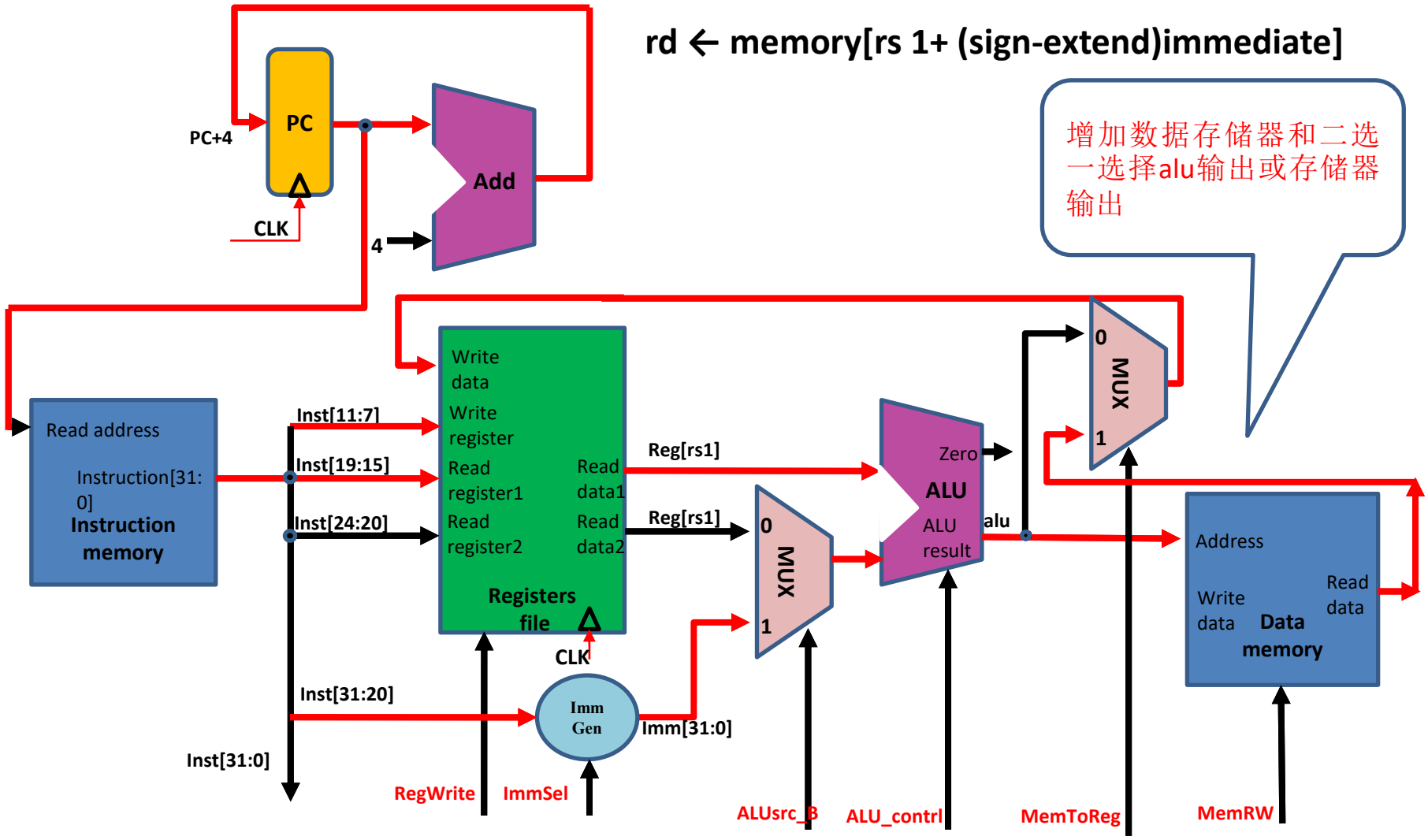
lw rd , immediate(rs1) **读存储器**

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

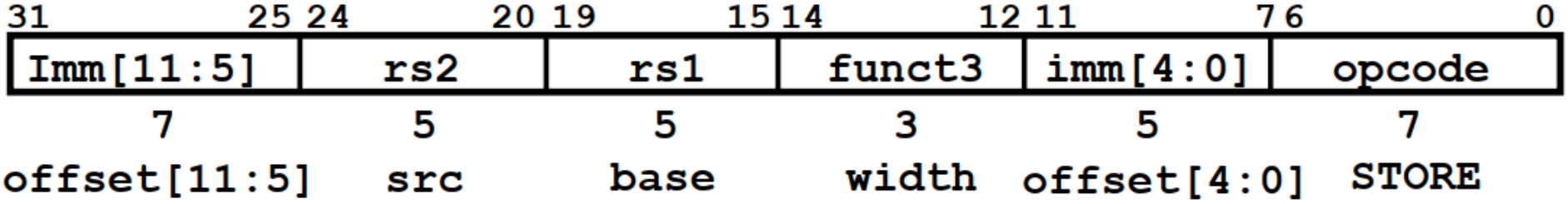


All RV32 Load Instructions

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

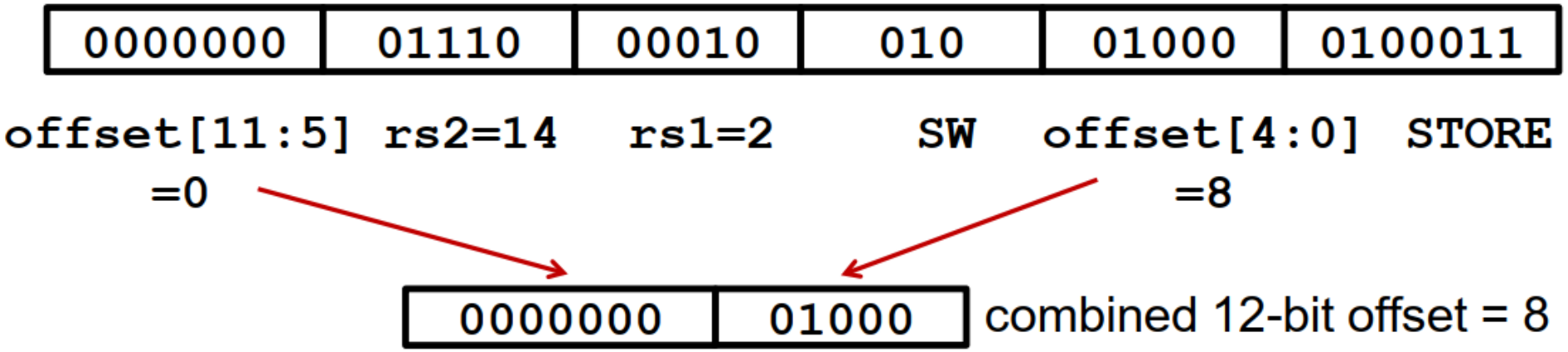


sw rs2 ,immediate(rs1) 写存储器

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)

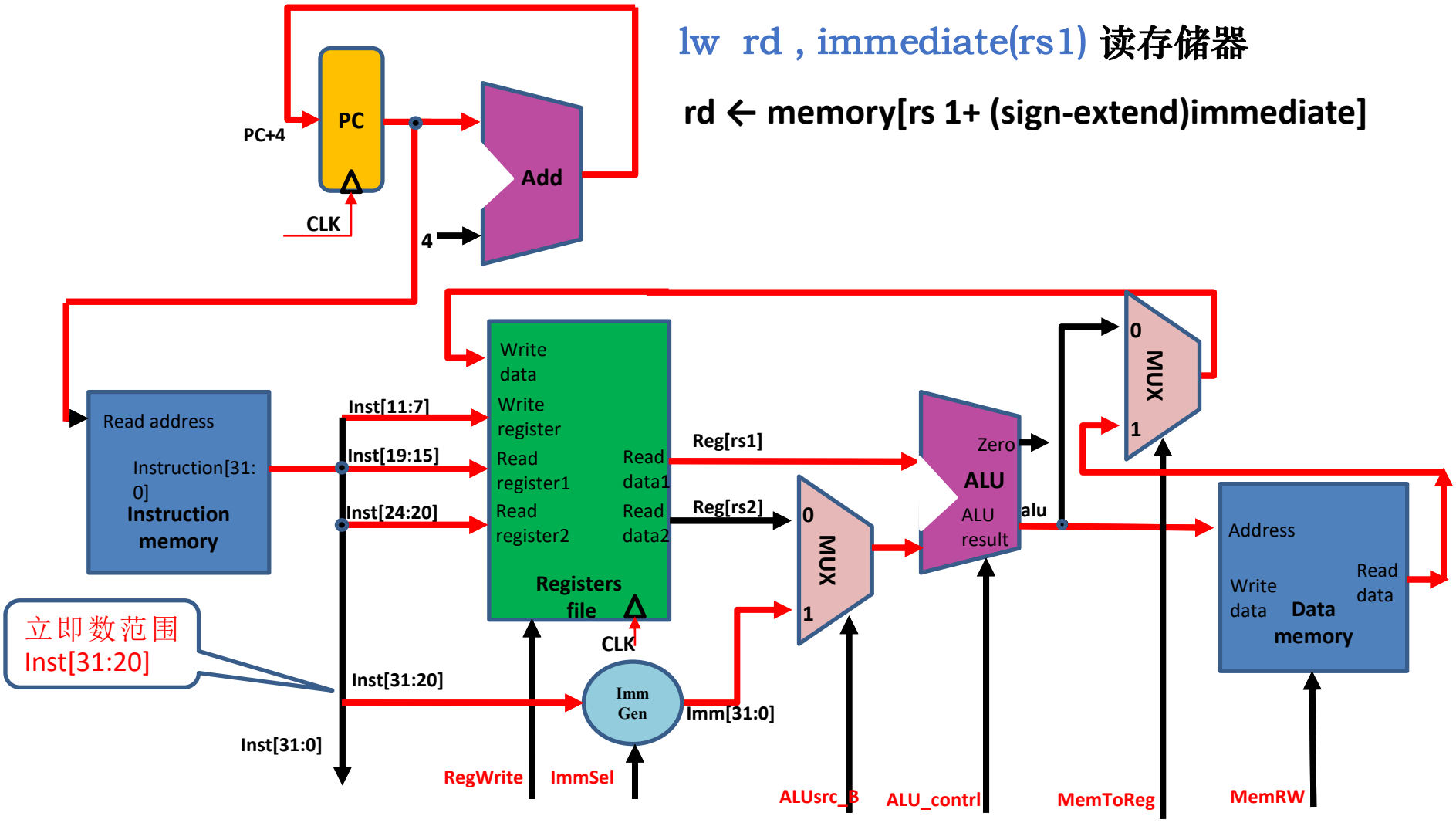
0000000 _01110_00010_010_01000_0100011



Datapath with lw

lw rd , immediate(rs1) 读存储器

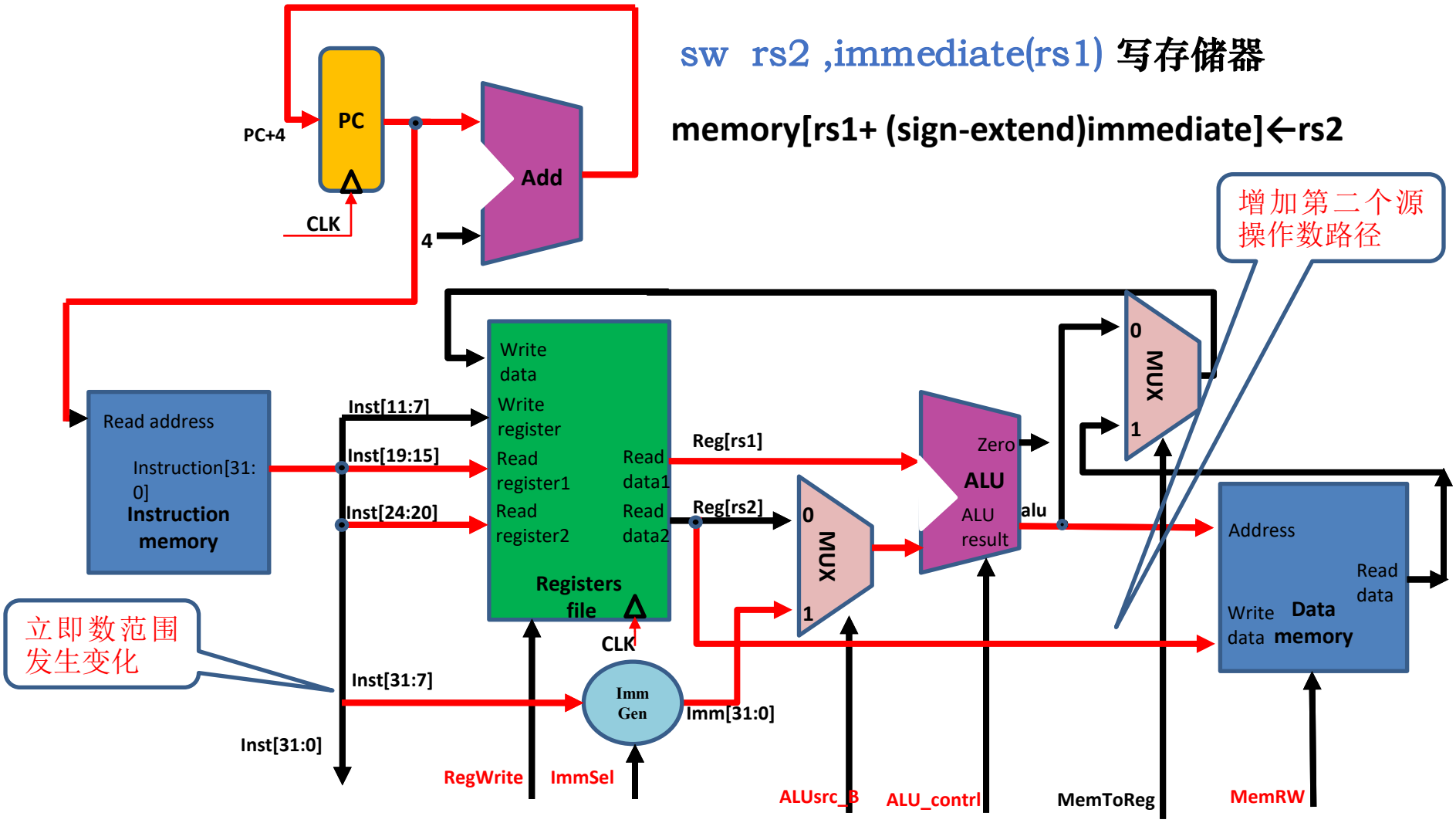
rd ← memory[rs 1+ (sign-extend)immediate]



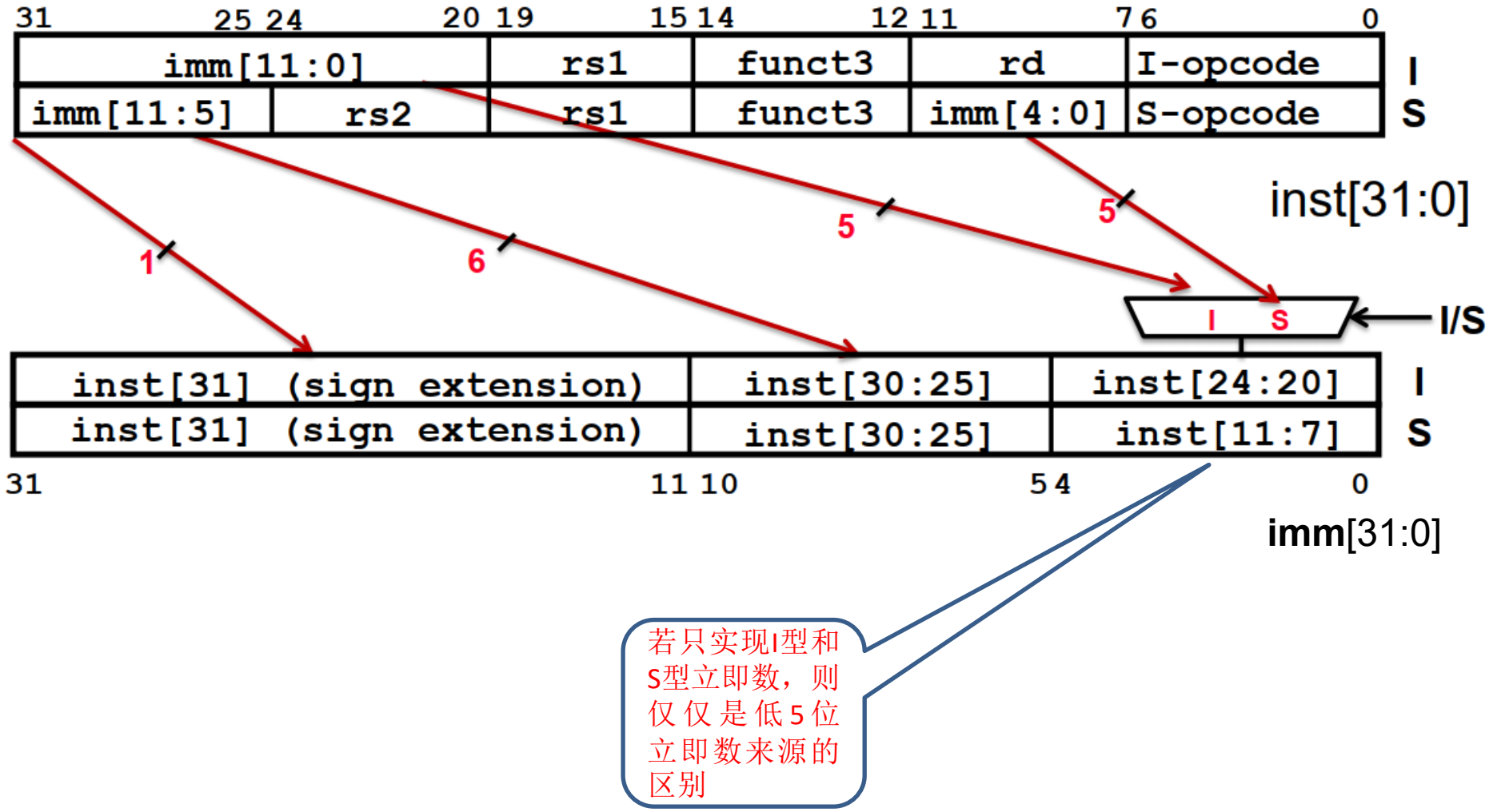
Adding sw to Datapath

sw rs2 ,immediate(rs1) 写存储器

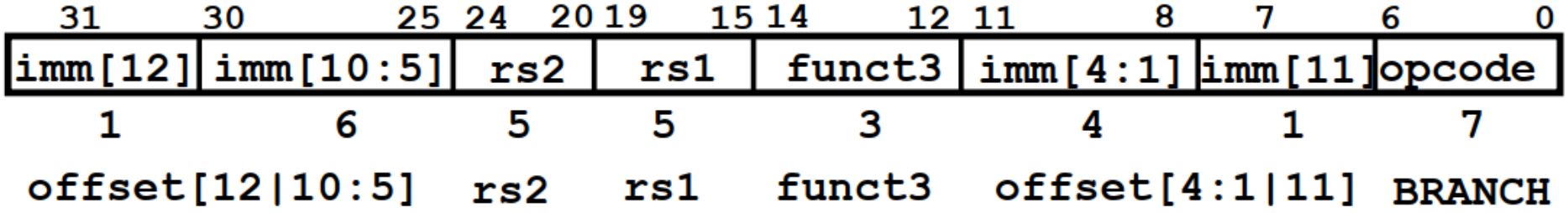
memory[rs1+ (sign-extend)immediate]←rs2



I+S Immediate Generation



Implementing Branches(B-Format)



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

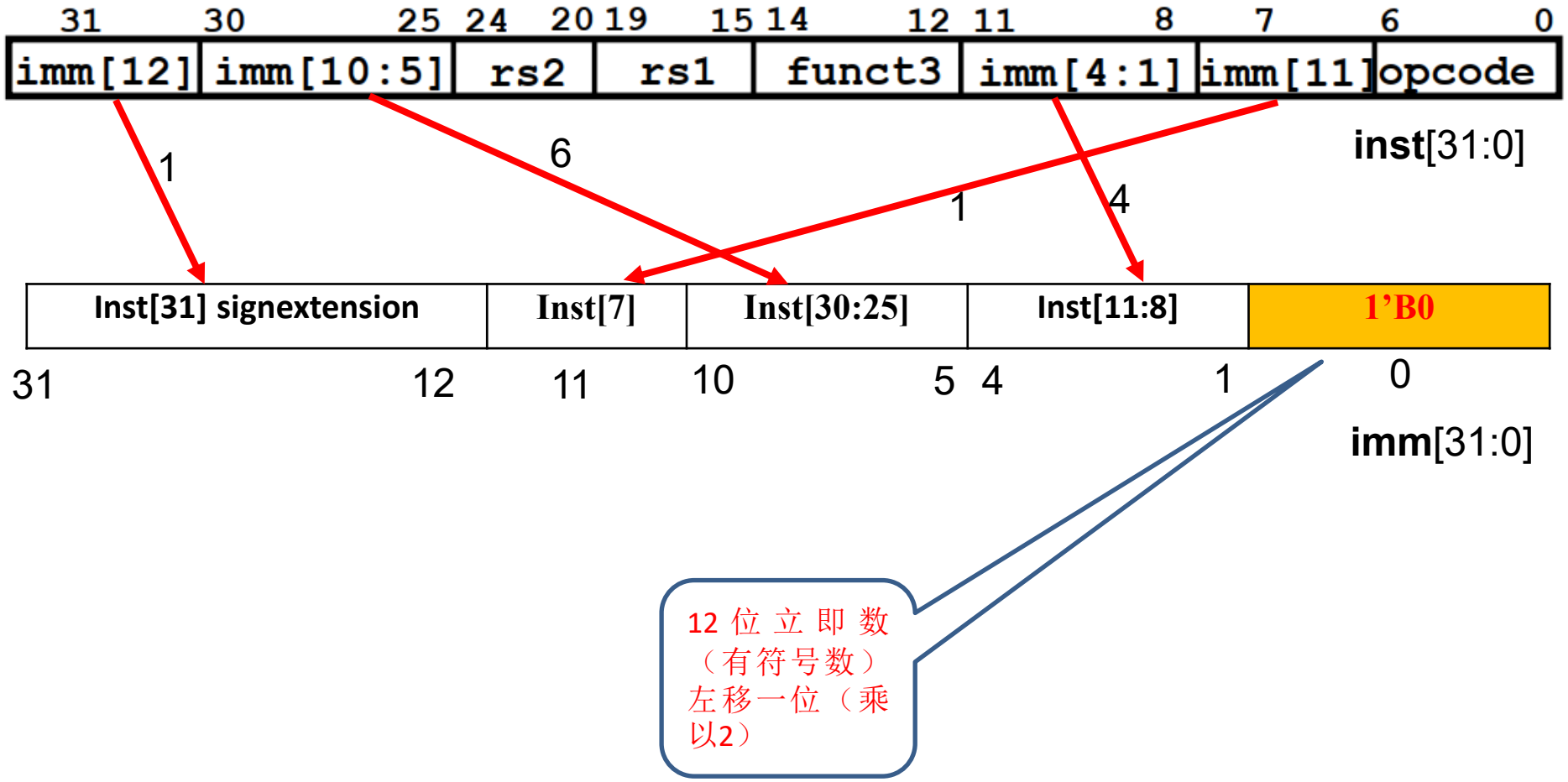
beq rs1,rs2,immediate　　相等则跳转

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

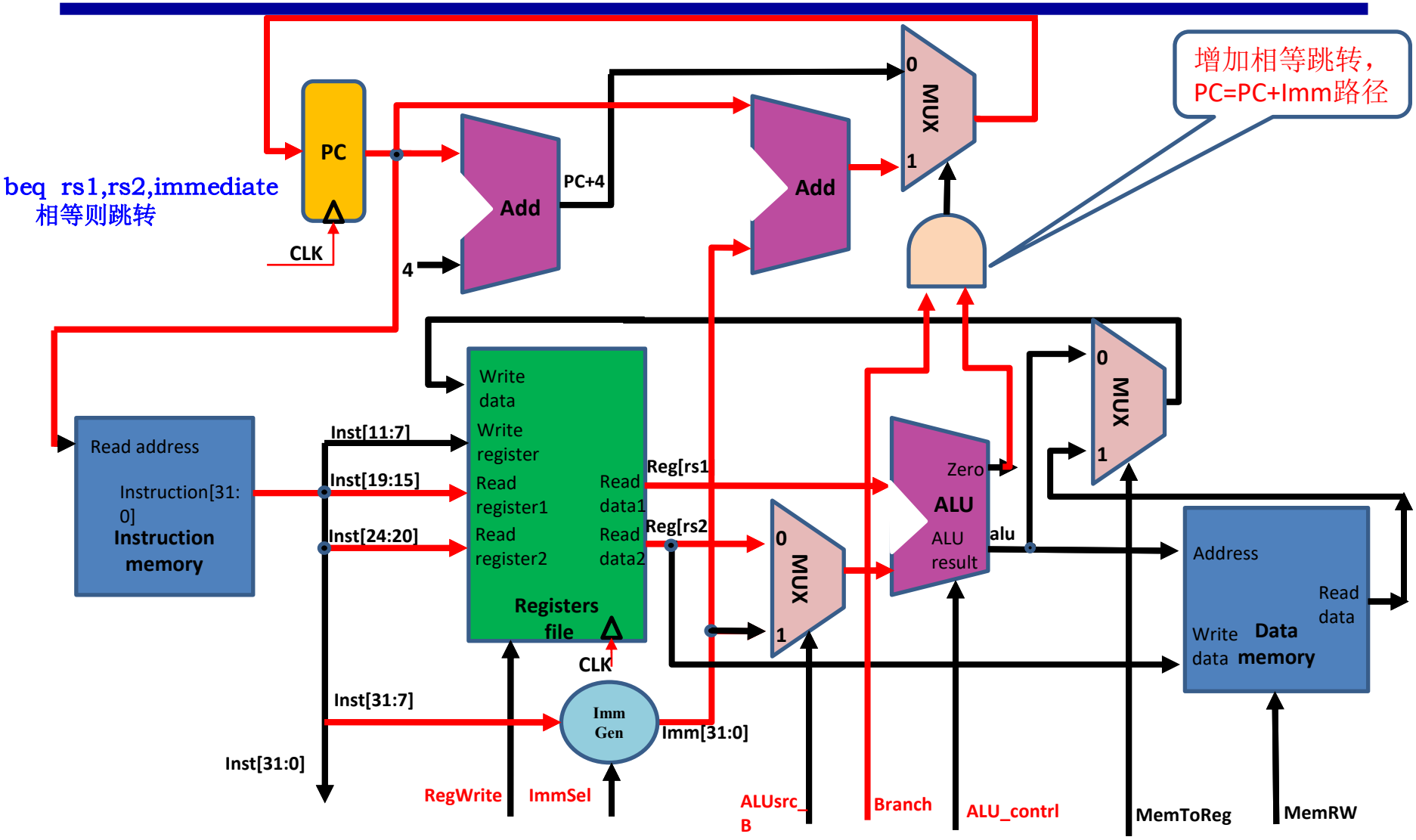
Eg　beq x5,x6,100

0000 0110 0110 0010 1000 0010 0110 0011

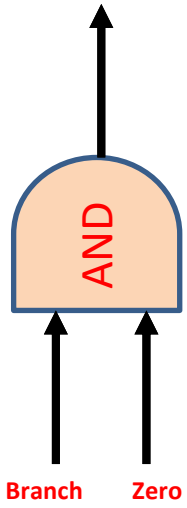
Branches Immediate Generation



Adding branches to Datapath

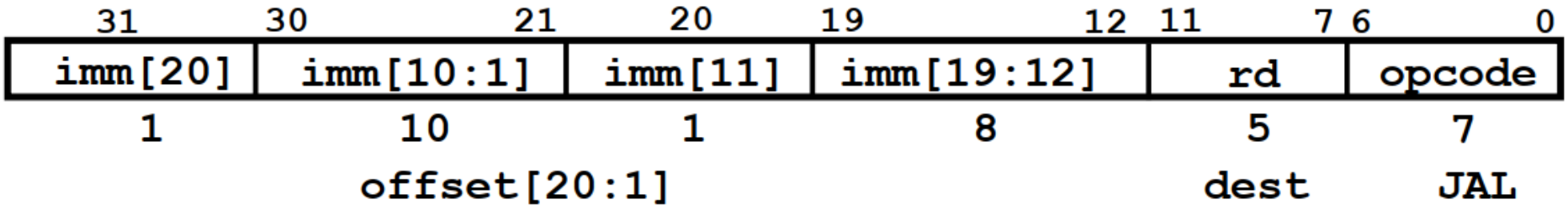


Branch Comparator



- 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）

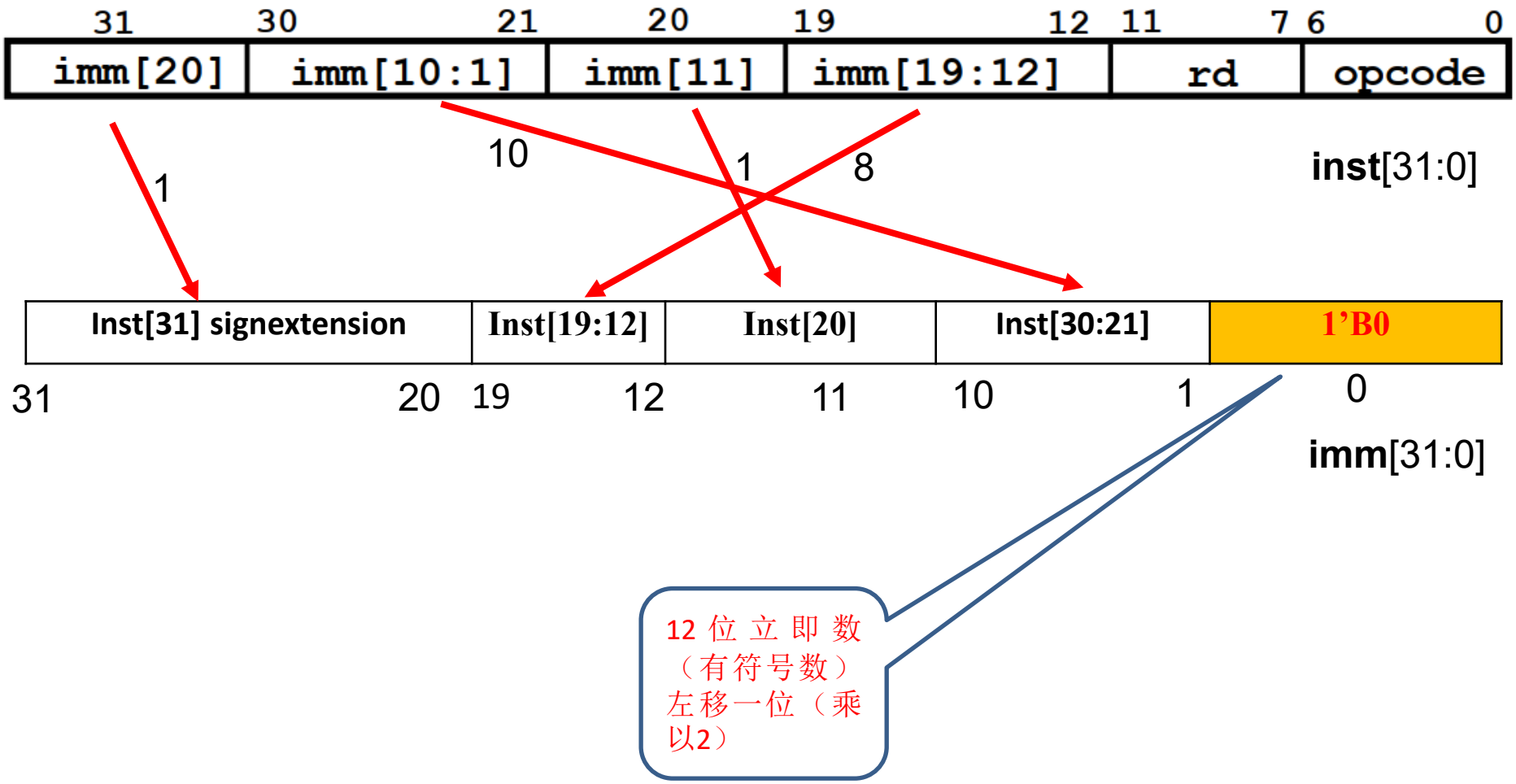
jal rd, immediate 跳转链接

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

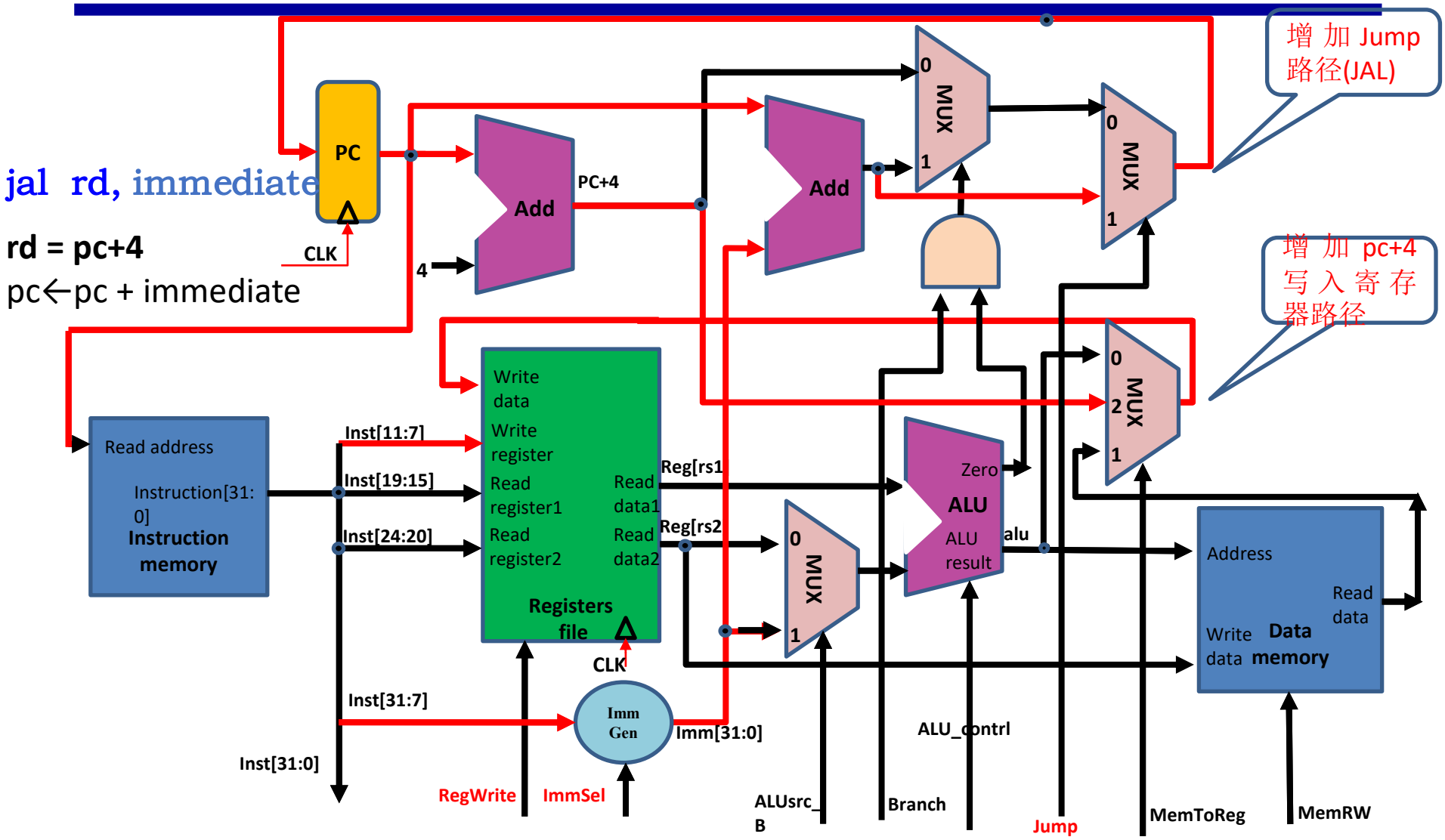
Eg jal x1,100

0000 0110 0100 0000 0000 0000 1110 1111

JAL Immediate Generation



Adding JAL to Datapath

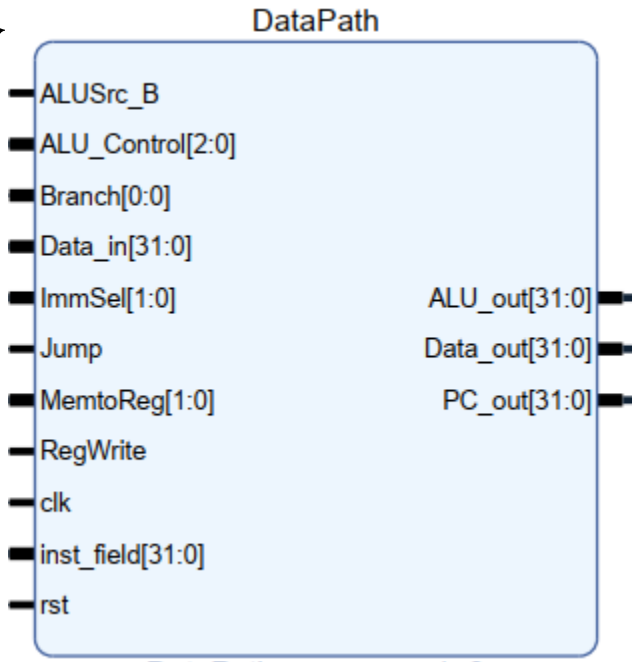


控制信号定义通路与操作控制

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）		

CPU部件之数据通路接口：Data_path

- **Data_path**
  - 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
- **接口要求- Data_path**
  - 数据通路接口信号如右图



数据通路接口信号标准- Data_path.v

```
module      Data_path( input clk,                  //寄存器时钟
                       input rst,                  //寄存器复位
                       input[31:0]inst_field,      //指令数据域[31:7]
                       input ALUSrc_B,             //ALU端口B输入选择
                       input [1:0]MemtoReg,        //Regs写入数据源控制
                       input Jump,                 //J指令
                       input Branch,               //Beq指令
                       input RegWrite,             //寄存器写信号
                       input[31:0]Data_in,         //存储器输入
                       input[2:0]ALU_Control,      //ALU操作控制
                       input[1:0]ImmSel,           //ImmGen操作控制

                       output[31:0]ALU_out,        //ALU运算输出
                       output[31:0]Data_out,       //CPU数据输出
                       output[31:0]PC_out          //PC指针输出
                       );

endmodule
```

- 任务一：设计实现数据通路（根据原理图写RTL代码）
  - ALU和Regs调用Exp01设计的模块
  - PC寄存器设计及PC通路建立
  - ImmGen立即数生成模块设计
  - 此实验在Exp4-0的基础上完成，替换Exp4-0的数据通路核

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设计的ALU和Regs

RTL设计实现数据通路

设计工程：OExp04-Datapath

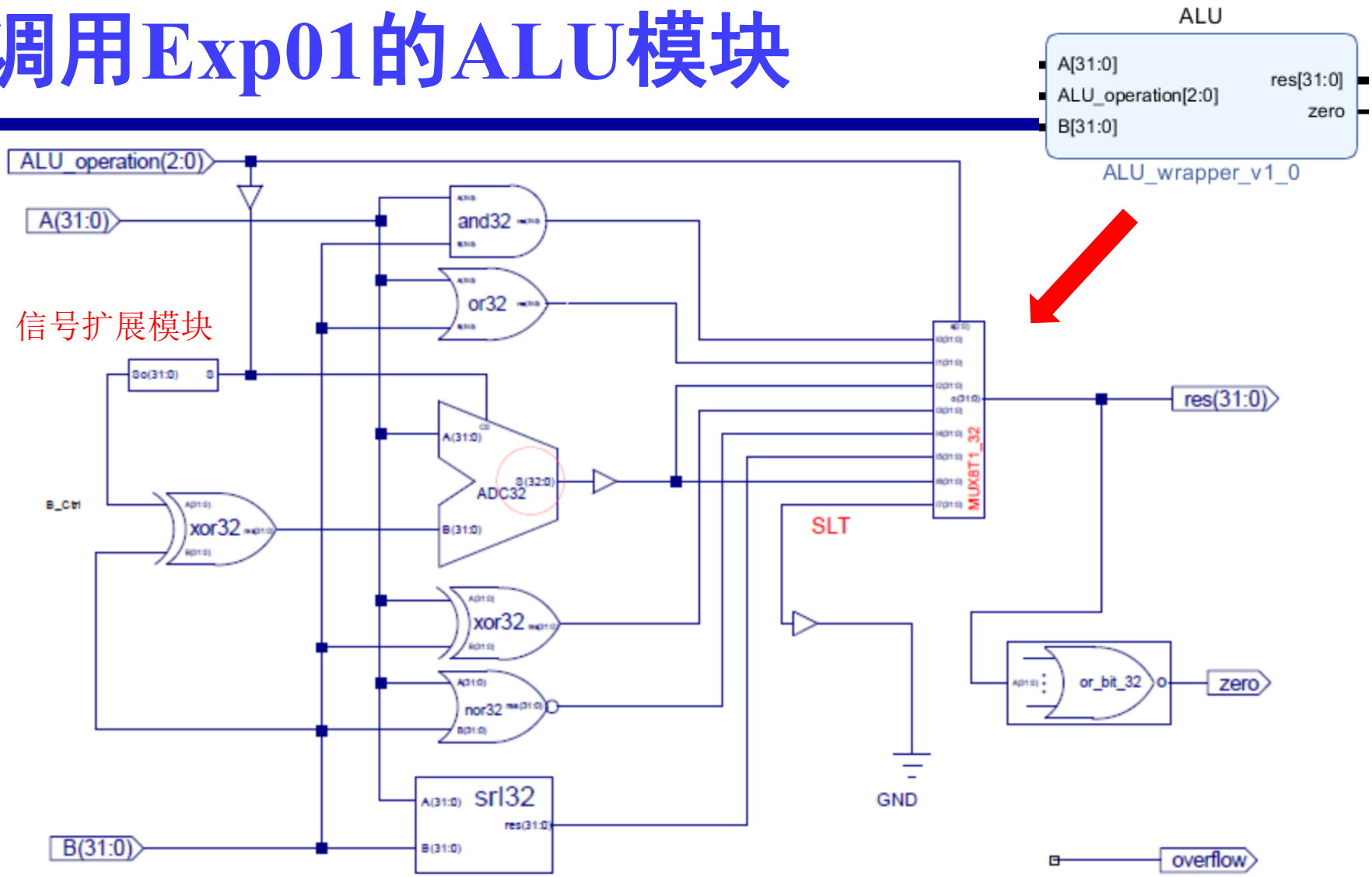
- **设计CPU之数据通路**
  - 根据原理介绍分析讨论设计13+条指令的数据通路
  - Add,sub,and,or.xor,srl,slt;addi,andi,ori,xori,srli,slti;lw,sw,
  - beq,jal
  - 仿真测试DataPath模块
- **集成替换验证通过的数据通路模块**
  - 替换实验四(Exp4-0)中的Data_Path核
  - 顶层模块延用Exp04
    - 模块名：DataPath.v
- **测试数据通路模块**
  - 设计测试程序(RISCV汇编)测试：
  - ALU功能
  - I-指令、R-指令、S、B、J指令通路

设计要点

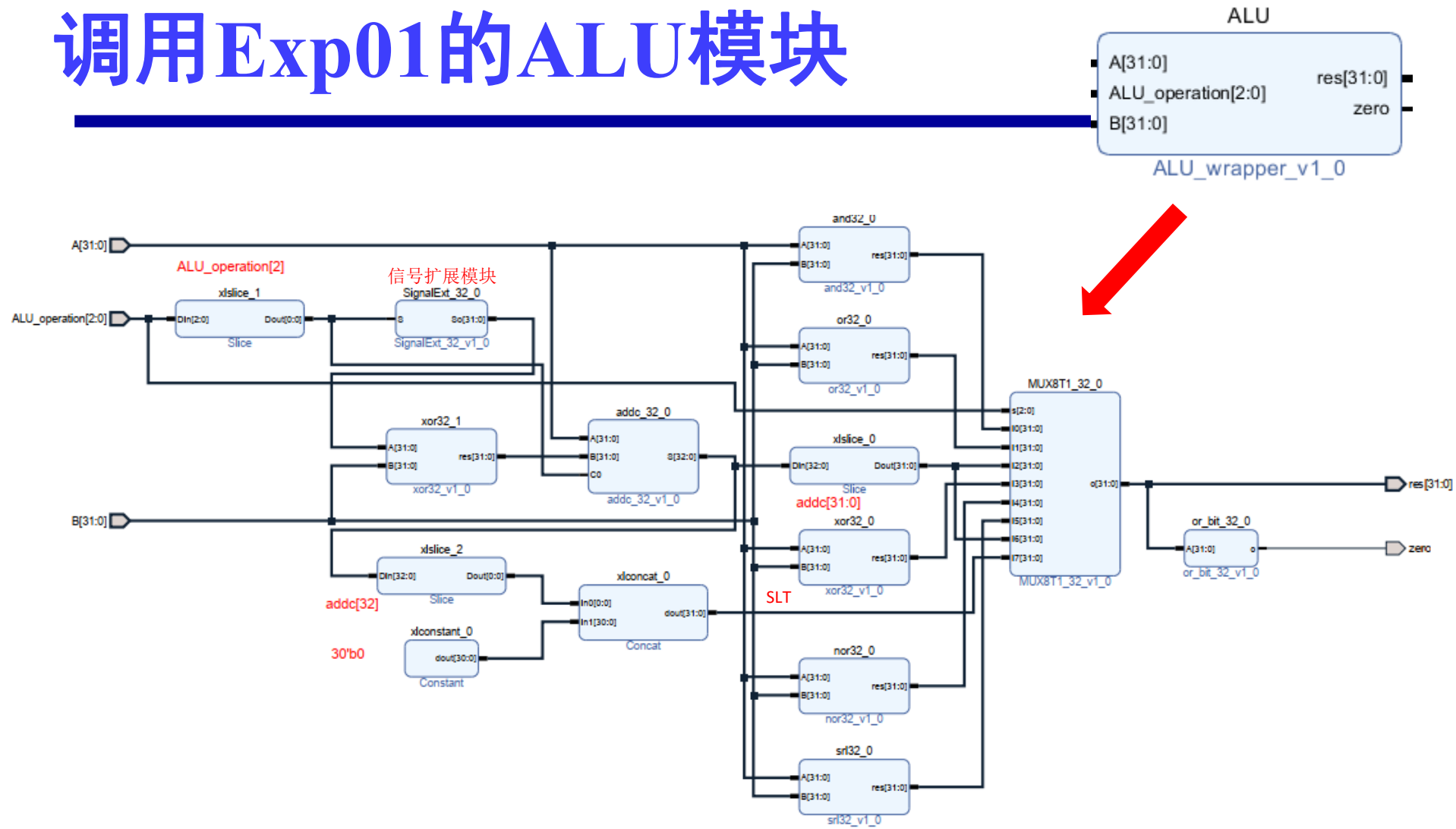
添加下列模块到当前工程IP目录：

and32、or32、ADC32、xor32、nor32、srl32、SignalExt_32、mux8to1_32、or_bit_32 add_32、 mux2to1_32 、 mux2to1_5、ALU、Regs、Ext_32、REG32、ImmGen

调用Exp01的ALU模块

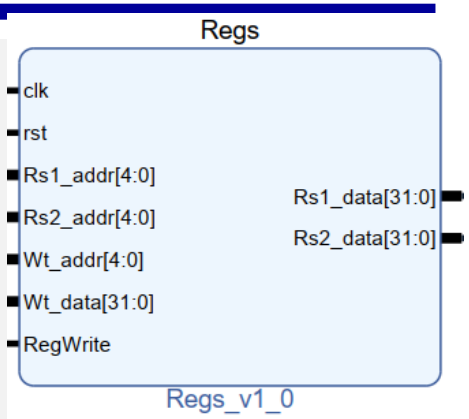


调用Exp01的ALU模块



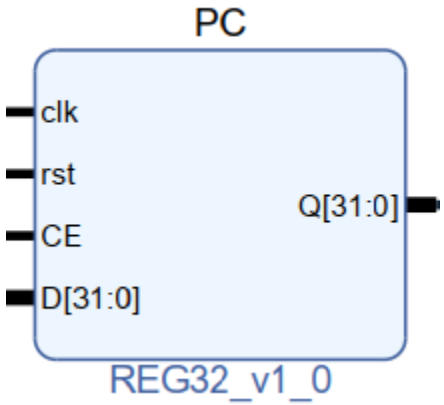
调用Exp01的Regs模块

```
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

endmodule
```



PC寄存器设计及PC通路建立

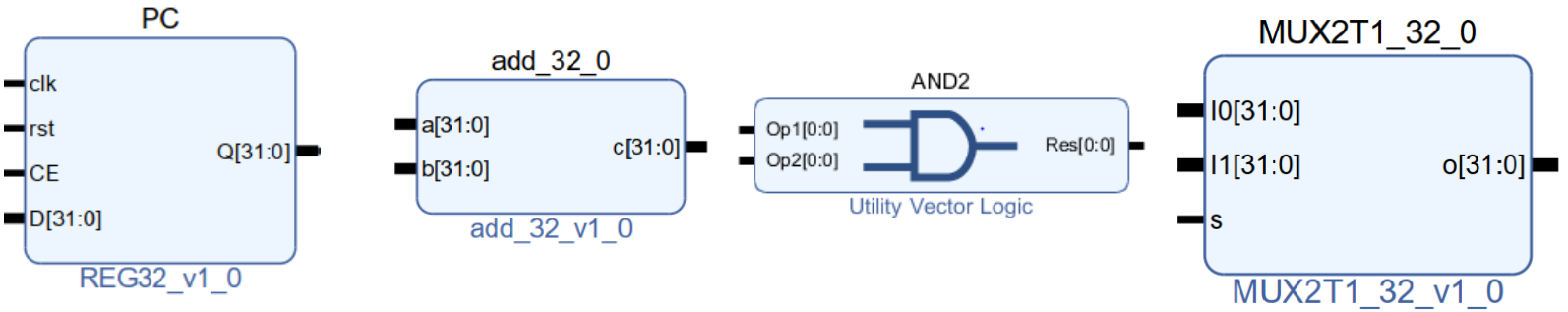
- 设计32位寄存器
  - 用途：PC指针、数据、地址或指令锁存
  - 采用行为描述实线
  - 模块名：REG32
    - 上升沿触发：clk
    - 使能信号：CE
    - 同步复位：rst=1
    - 数据输入：D(31:0)
    - 数据输出：Q(31:0)
  - 参考描述结构




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

PC寄存器设计及PC通路建立

- 建立DataPath顶层连接
- 设计PC通路
  - 调用REG32、add_32、AND2(库)和MUX2T1_32模块



  - 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”　　＝？？？？

ImmGen立即数生成模块设计

- **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**
  - 数据通路接口信号如右图

ImmGen接口信号标准—ImmGen.v

```
module      ImmGen ( input wire [1:0] ImmSel,        //立即数操作控制
                     input wire [31:0] inst_field,   //指令数据域[31:7]
                     outputreg  [31:0] Imm_out       //立即数输出
                       );
always@*begin
    case(ImmSel)
       2'b00:Imm_out = {{20{inst_field[31]}},inst_field[31:20]};  //addi\lw(I)
       2'b01:Imm_out =  ……;                                      //sw(s)
       2’b10:Imm_out =  ……;                                      //beq(b)
       2’b11:Imm_out =  ……;                                      //jal(j)
   endcase
 end
endmodule
```

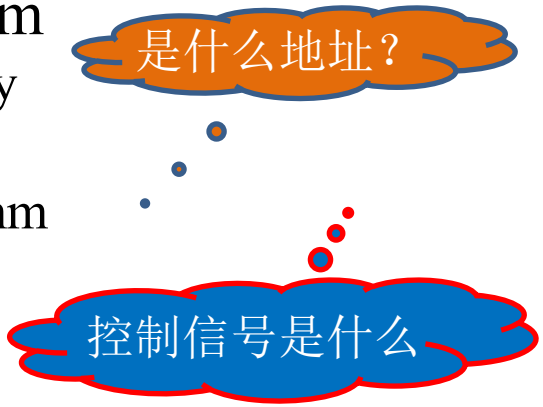
根据指令产生
对应立即数

CPU之数据通路建立

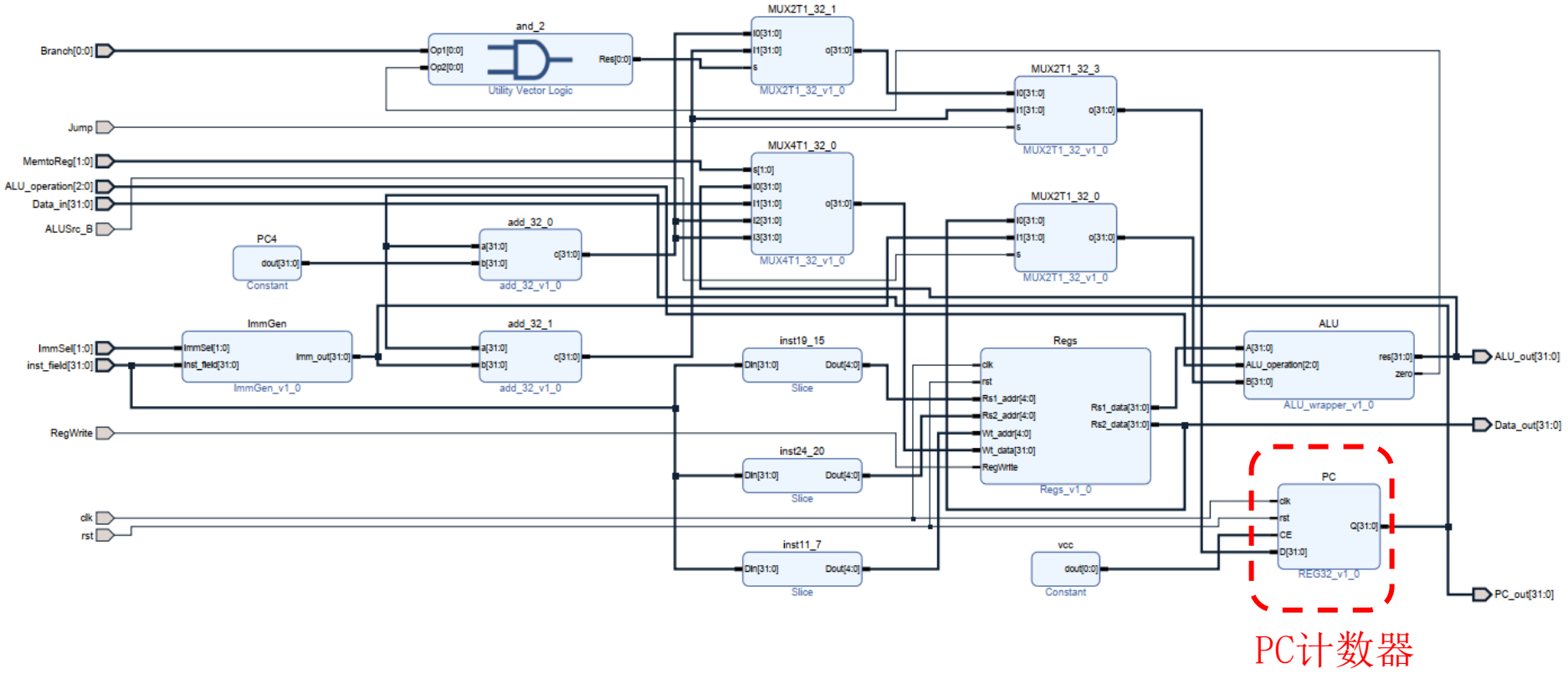
- 设计R-格式和I-格式数据通路
  - 根据理论课分析讨论的数据通路选择MUX
  - Regs数据通道设计
    - R-格式源地址通道选择：rs1、rs2
    - R-格式目的地址通道选择：rd
    - R-格式目的数据通道选择：form ALU
    - I-格式（立即数操作）源地址通道选择：rs1、Imm
    - I-格式（立即数操作）目的地址通道选择：rd
    - I-格式（立即数操作）目的数据通道选择：from ALU
    - I-格式（lw操作）源地址通道选择：rs1、Imm
    - I-格式（lw操作）目的地址通道选择：rd
    - I-格式（lw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

CPU之数据通路建立

- 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
      - S-格式：from Imm



数据通路参考逻辑结构图



PC计数器

本图仅供概览，详细连接图请参见完整版pdf文档

数据通路顶层设计

- 利用verilog描述，在DataPath.v顶层进行模块调用和连接

```
module DataPath(
    input wire clk,
    input wire rst,
    input wire[31:0] inst_field,
    input wire[31:0] Data_in,
    input wire[2:0]  ALU_Control,
    input wire[1:0]  ImmSel,
    input wire[1:0]  MemtoReg,
    input wire       ALUSrc_B,
    input wire       Jump,
    input wire       Branch,
    input wire       RegWrite,

    output wire[31:0] PC_out,
    output wire[31:0] Data_out,
    output wire[31:0] ALU_out
);
```

```
.....
.....

wire[31:0] Regs_0_Rs1_data;
wire[31:0] MUX2T1_32_0_o;
wire       ALU_0_zero;
ALU_0 ALU_U(
        .A(Regs_0_Rs1_data),
        .ALU_operation(ALU_Control),
        .B(MUX2T1_32_0_o),
        .res(ALU_out),
        .zero(ALU_0_zero)
);

.....
.....
```

DataPath替换集成

- 集成替换
  - 仿真正确后封装为核并替换Exp04-0的数据通路IP核
- 清理Exp04-0工程
  - 移除工程中的数据通路核
    - Exp04-0 工程中移除数据通路核
    - 右键点击，Remove File From Project
  - 删除工程中CPU核文件
    - Data_path.edf和 Data_path.v 文件
  - 建议用Exp04-0资源重建工程
    - 删除Data_path核
  - 添加设计后的DataPath核，参照Exp04-0完成CPU核集成

DataPath替换集成

- 集成替换DapaPath核后的模块层次结构

CPU替换集成

- 集成替换
  - 参照lab0将本次设计的CPU替换Exp02的SOC系统中的SCPU核

- **CPU替换集成SOC的替换方法与DataPath替换集成CPU相同**

- 完成单周期CPU替换集成后，即可设计测试方案进行功能测试

- 任务二：设计数据通路测试方案并完成测试
  - 通路测试：I-格式通路、R-格式通路
  - 部件测试：ALU、Register Files

数据通路仿真调试

- DataPath模块仿真
  - 原理图检查没有Errors和warnings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13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、xor、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比较、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

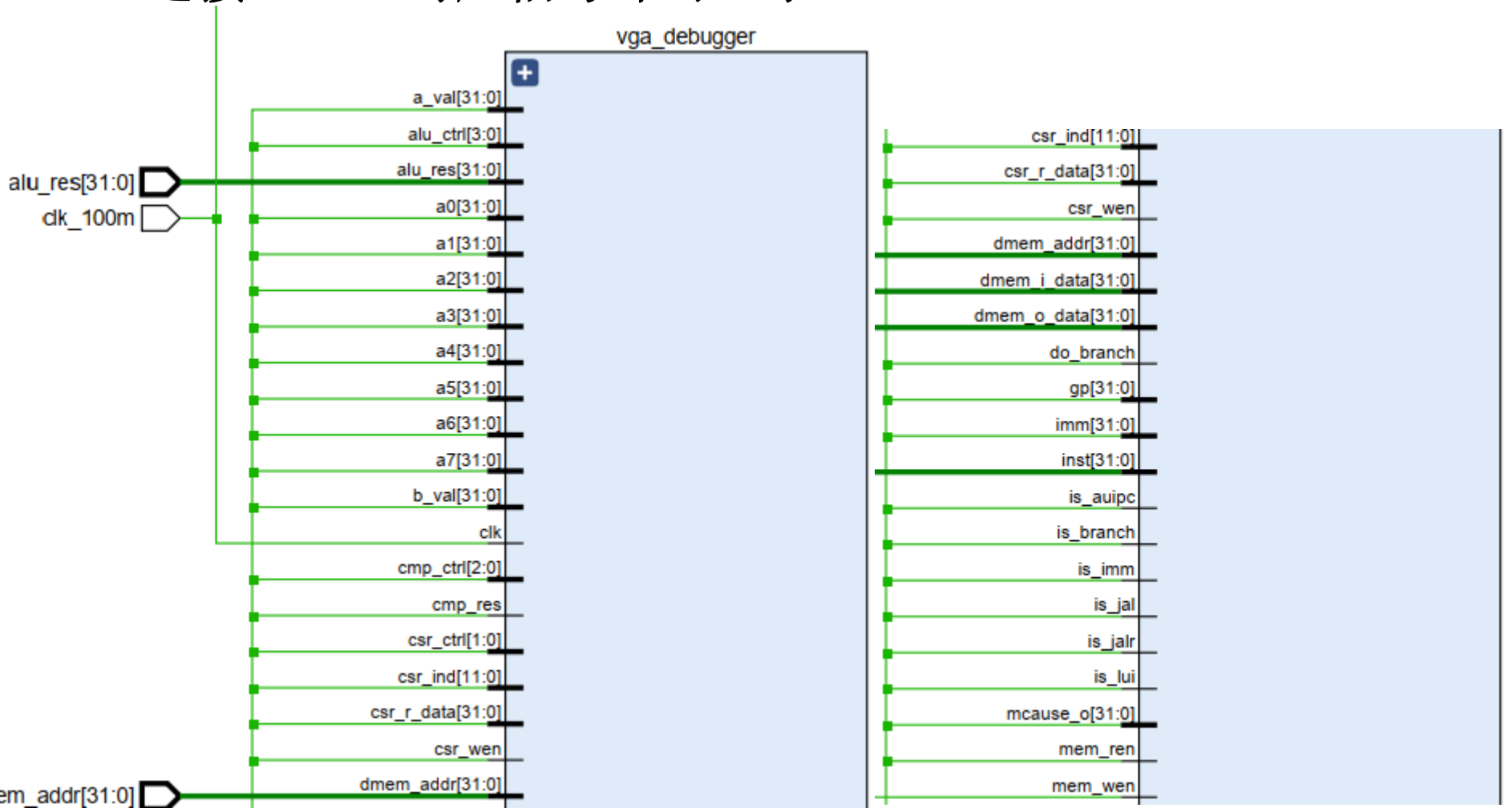
若包含提供的edf格式IP，则无法进行行为仿真

物理验证

- **使用DEMO程序目测数据通路功能**
  - DEMO接口功能
    - SW[8]=0， SW[2]=0(全速运行)
    - SW[8]=0, SW[2]=0(自动单步)
    - SW[8]=1, SW[2]=x(手动单步)
    - SW[10], 从0到1 (手动单步)
- **用汇编语言设计测试程序**
  - 测试ALU功能
  - 测试Regs访问
  - 测试I-格式指令通路
  - 测试R-格式指令通路

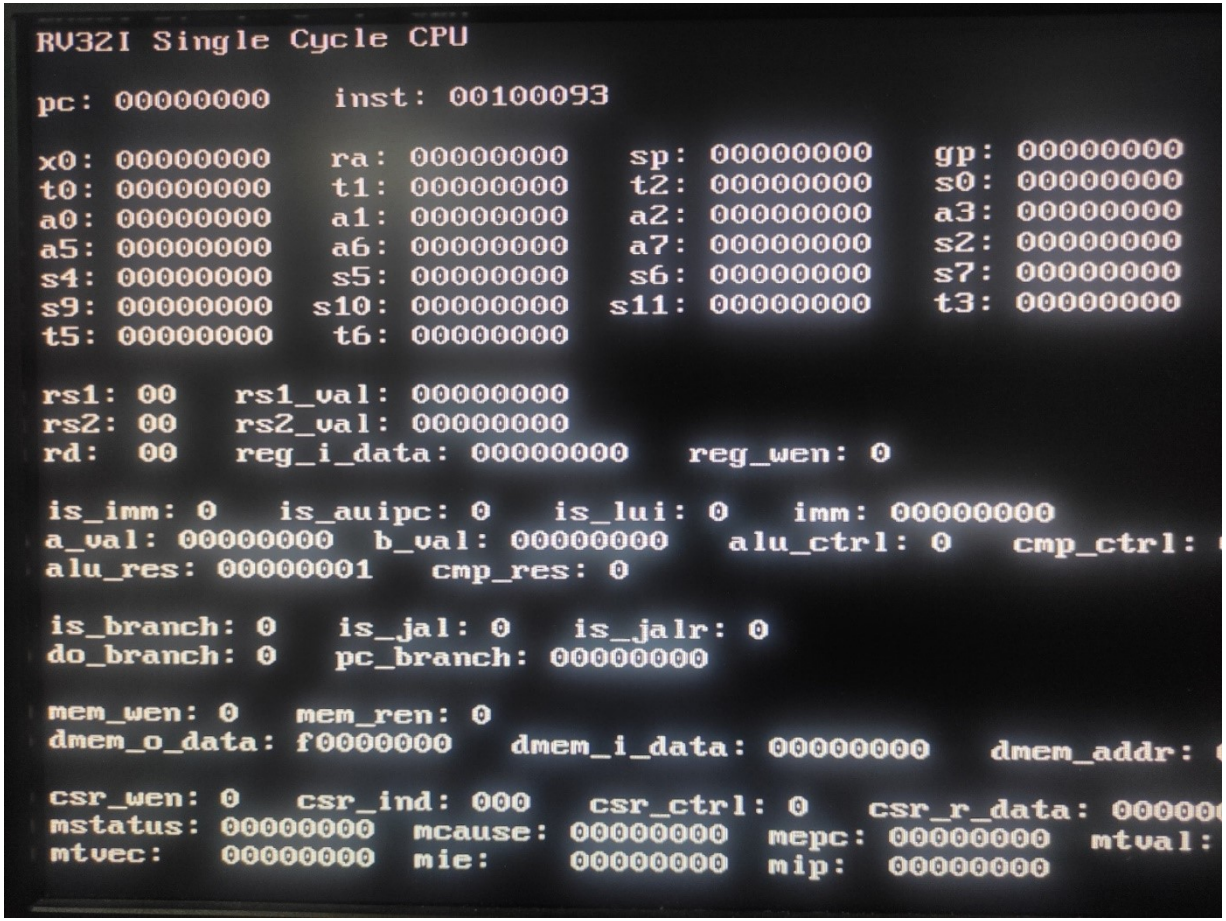
VGA信号调试

- VGA模块预留有很多调试端口，只需将端口重新引出，连接上CPU对应信号即可显示。

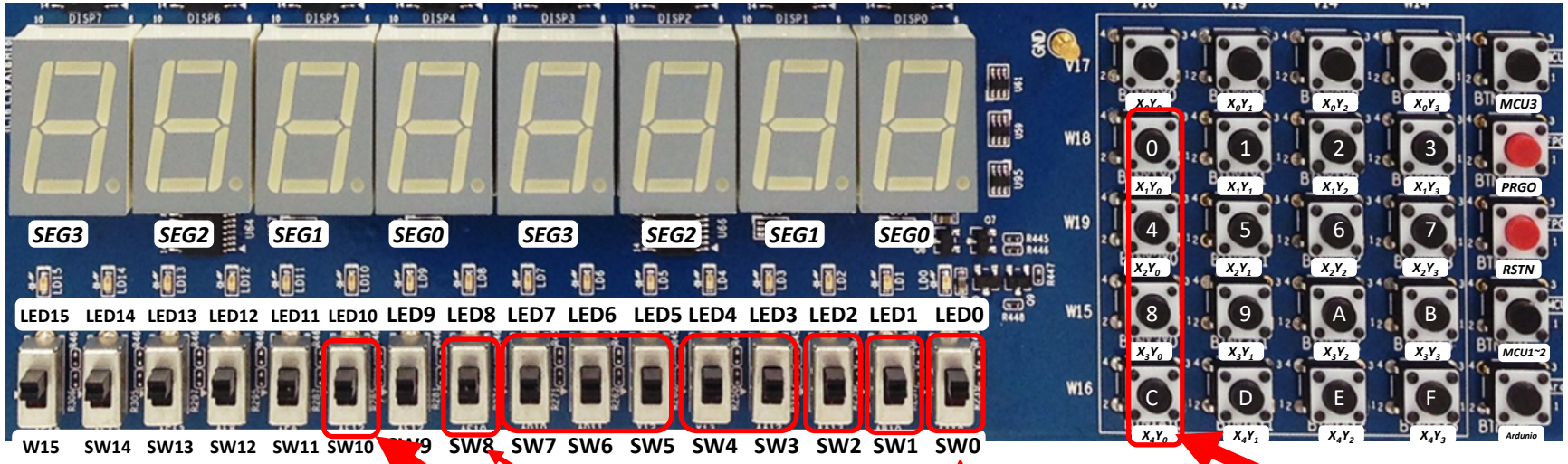


物理验证

- 使用DEMO程序目测数据通路功能
  - DEMO接口功能
    - SW[8]=1(手动单步运行) 液晶屏显示CPU信息



物理验证-DEMO接口功能



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[0]=文本图形选择

SW[1]=高低16位选择

BTN_y[0]做单步STEP输入

SW[8][2]=CPU单步时钟选择

SW[10]做单步STEP输入

SW[10]更稳定

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:
        addi x1,x0,1;       //x1=00000001
        slt x2,x0,x1;       //x2=00000001
        add x3,x2,x2;       //x3=00000002
        add x4,x3,x2;       //x4=00000003
        add x5,x4,x3;       //x5=00000005
        add x6,x5,x4;       //x6=00000008
        add x7,x6,x5;       //x7=0000000d
        add x8,x7,x6;       //x8=00000015
        add x9,x8,x7;       //x9=00000022
        add x10,x9,x8;      //x10=00000037
        add x11,x10,x9;     //x11=00000059
        add x12,x11,x10;    //x12=00000090
        add x13,x12,x11;    //x13=000000E9
        add x14,x13,x12;    //x14=00000179
        add x15,x14,x13;    //x15=00000262
        add x16,x15,x14;    //x16=000003DB
        add x17,x16,x15;    //x17=000006D3
        add x18,x17,x16;    //x18=00000A18
        add x19,x18,x17;    //x19=000010EB
        add x20,x19,x18;    //x20=00001B03
        add x21,x20,x19;    //x21=00003bEE
        add x22,x21,x20;    //x22=000046F1
        add x23,x22,x21;    //x23=000080DF
        add x24,x23,x22;    //x24=0000C9D0
        add x25,x24,x23;    //x25=00014AAF
        add x26,x25,x24;    //x26=0001947F
        add x27,x26,x25;    //x27=0012DF2E
        add x28,x27,x26;    //x28=001473AD
        add x29,x28,x27;    //x29=002752DB
        add x30,x29,x28;    //x30=003BC688
        add x31,x30,x29;    //x31=00621963
beq x0,x0,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：13+x0

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

- 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

数据存储器模块测试(reserve)

- 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Exp0**

```
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

- 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：　　srai,slli,sltiu

  B-Type：　　bne；

  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