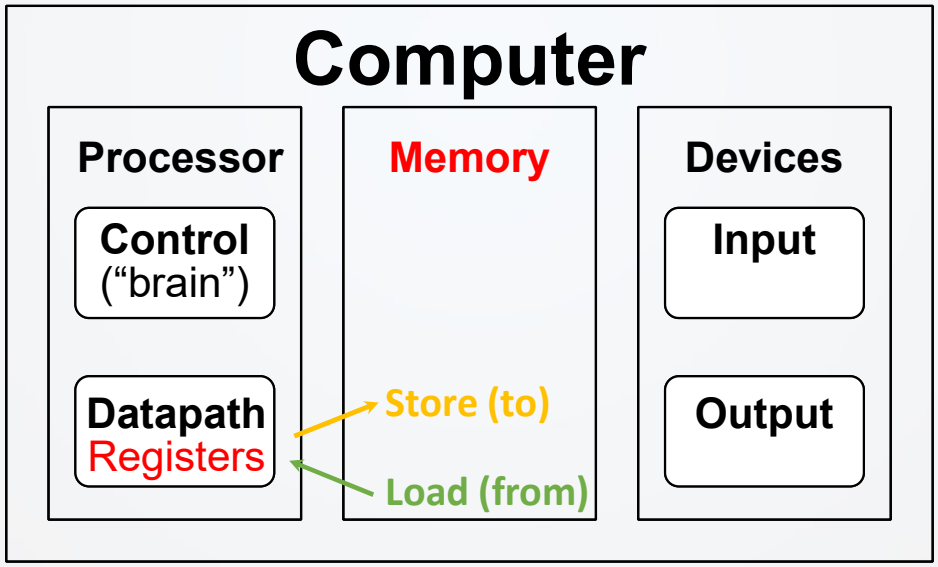
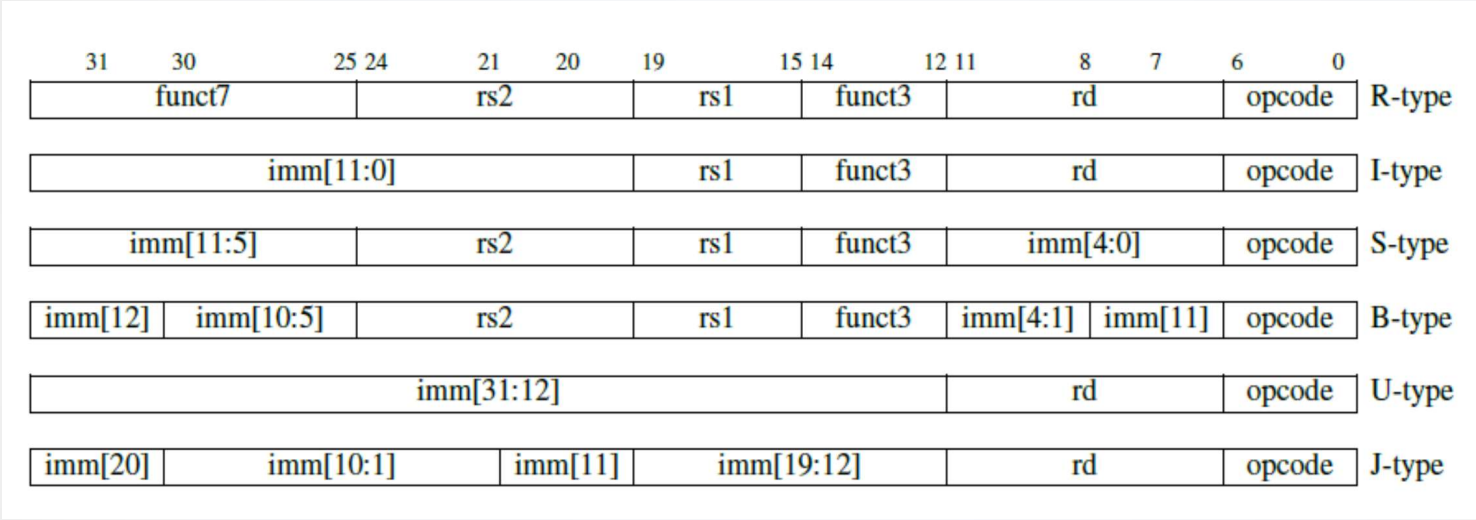
计算机体系结构实验

Five Components of a Computer

- Data Transfer instructions are between registers (Datapath) and Memory
  - Allow us to fetch and store operands in memory



RISC-V指令



Register

Immediate

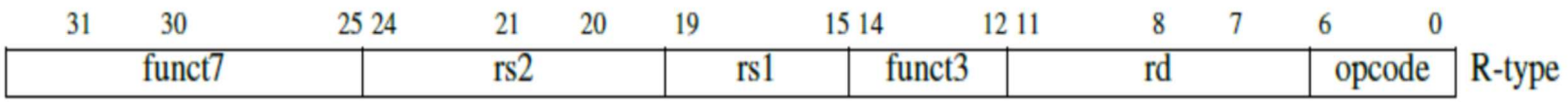
Store

Branch

Upper Immediate

Jump

R-type

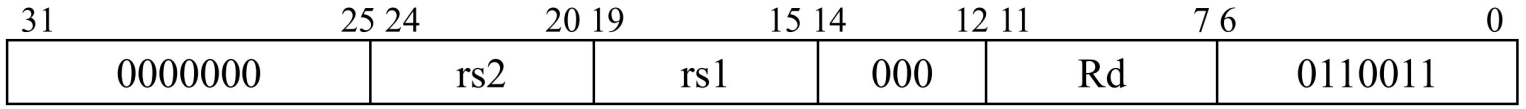


add rd, rs1, rs2 x[rd] = x[rs1] + x[rs2]

加 *(Add)*. R-type, RV32I and RV64I.

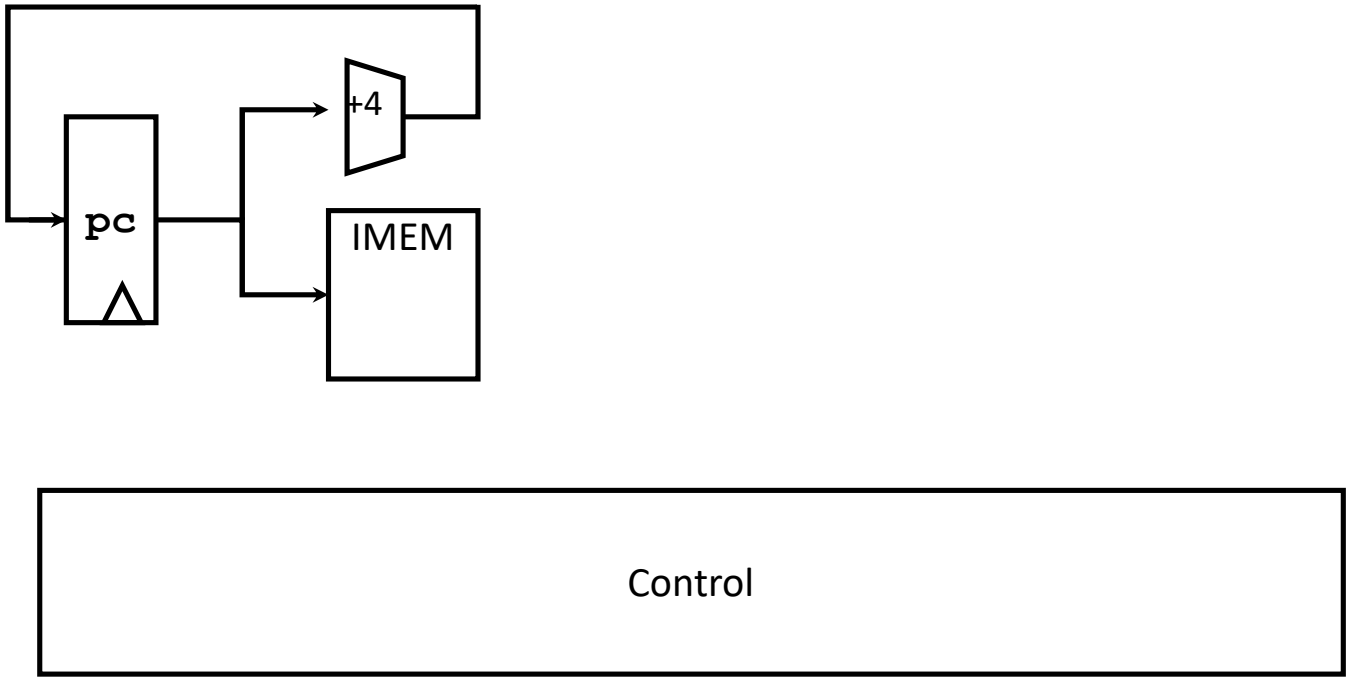
把寄存器 x[*rs2*]加到寄存器 x[*rs1*]上，结果写入 x[*rd*]。忽略算术溢出。

压缩形式：**c.add** rd, rs2; **c.mv** rd, rs2

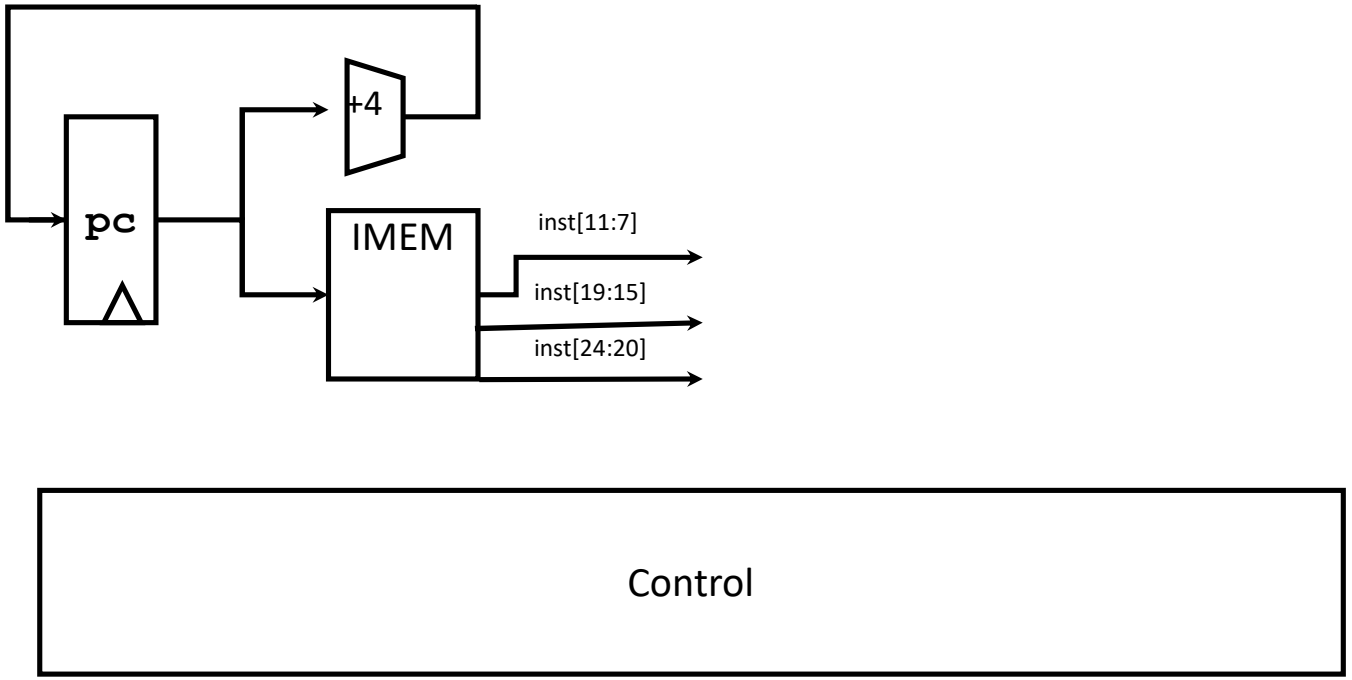


ASel = 0; BSel = 0; ALUSel = 0; WBSel = 1; RegWEn = 1;

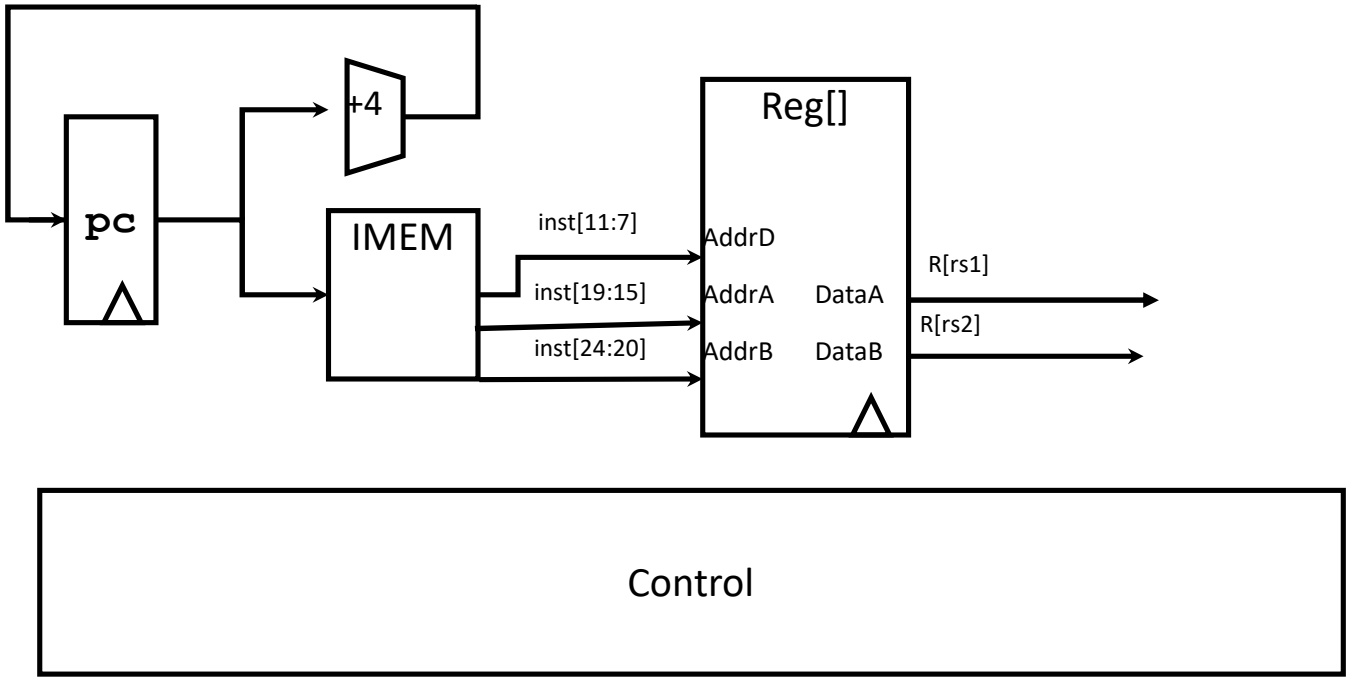
Implementing R-Types



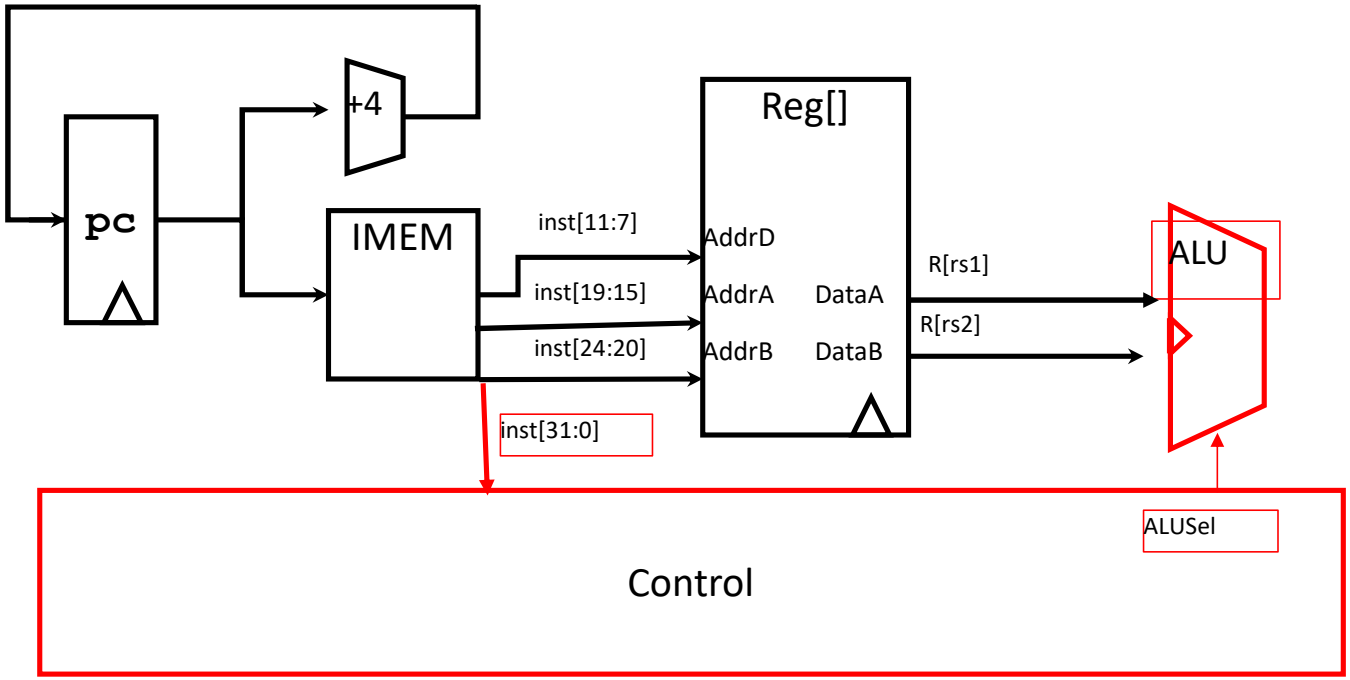
Implementing R-Types



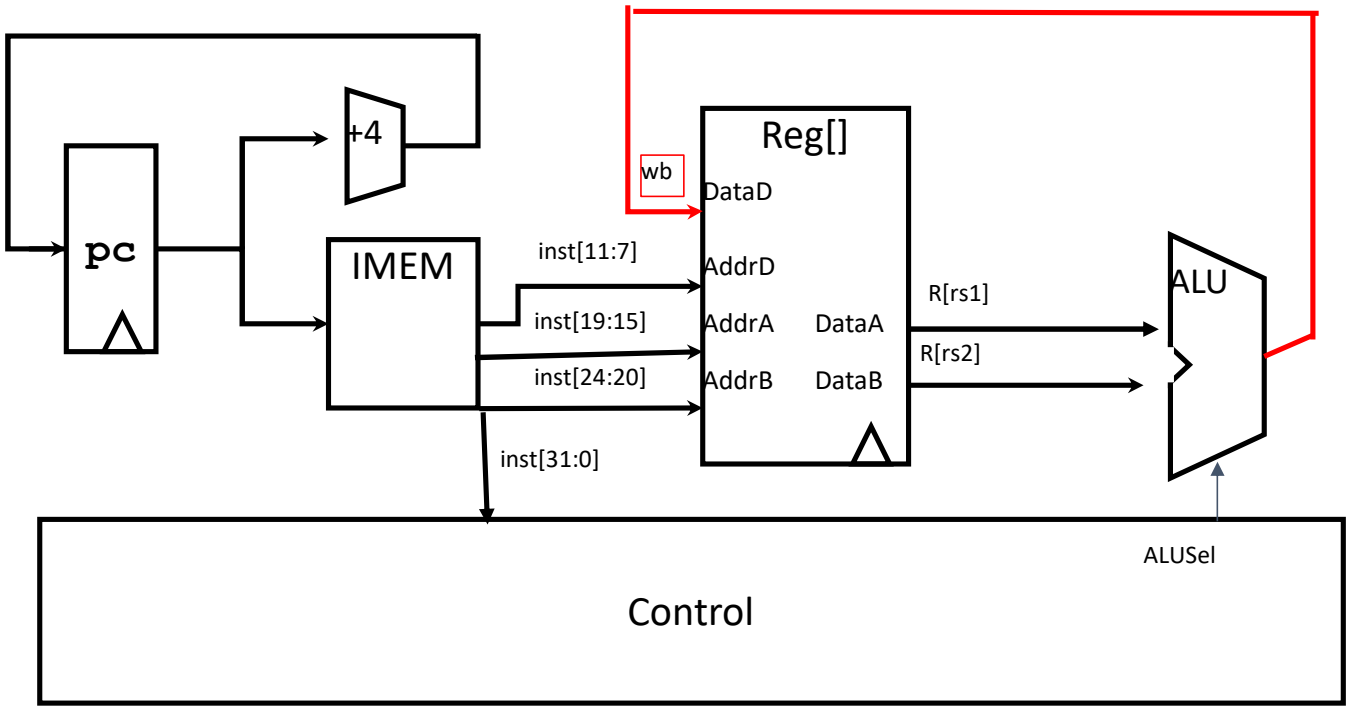
Implementing R-Types



Implementing R-Types



Implementing R-Types



I-type

imm[11:0]	rs1	funct3	rd	opcode	I-type

addi rd, rs1, immediate　　x[rd] = x[rs1] + sext(immediate)

加立即数*(Add Immediate)*. I-type, RV32I and RV64I.

把符号位扩展的立即数加到寄存器 x[*rs1*]上，结果写入 x[*rd*]。忽略算术溢出。

压缩形式：**c.li** rd, imm; **c.addi** rd, imm; **c.addi16sp** imm; **c.addi4spn** rd, imm

31 … 20	19 … 15	14 … 12	11 … 7	6 … 0
immediate[11:0]	rs1	000	rd	0010011

lw rd, offset(rs1)　　x[rd] = sext(M[x[rs1] + sext(offset)][31:0])

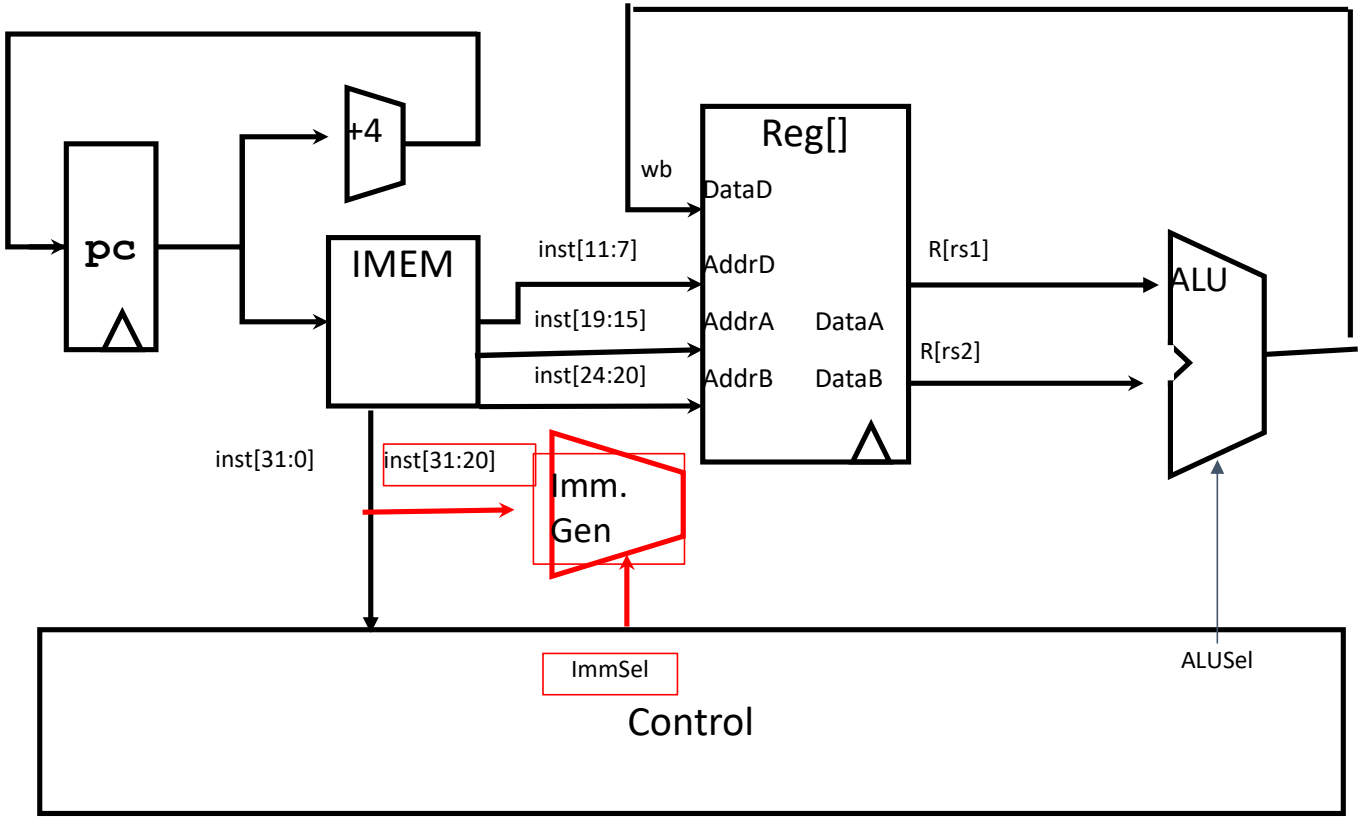
字加载 *(Load Word)*. I-type, RV32I and RV64I.

从地址 x[*rs1*] + *sign-extend*(*offset*)读取四个字节，写入 x[*rd*]。对于 RV64I，结果要进行符号位扩展。

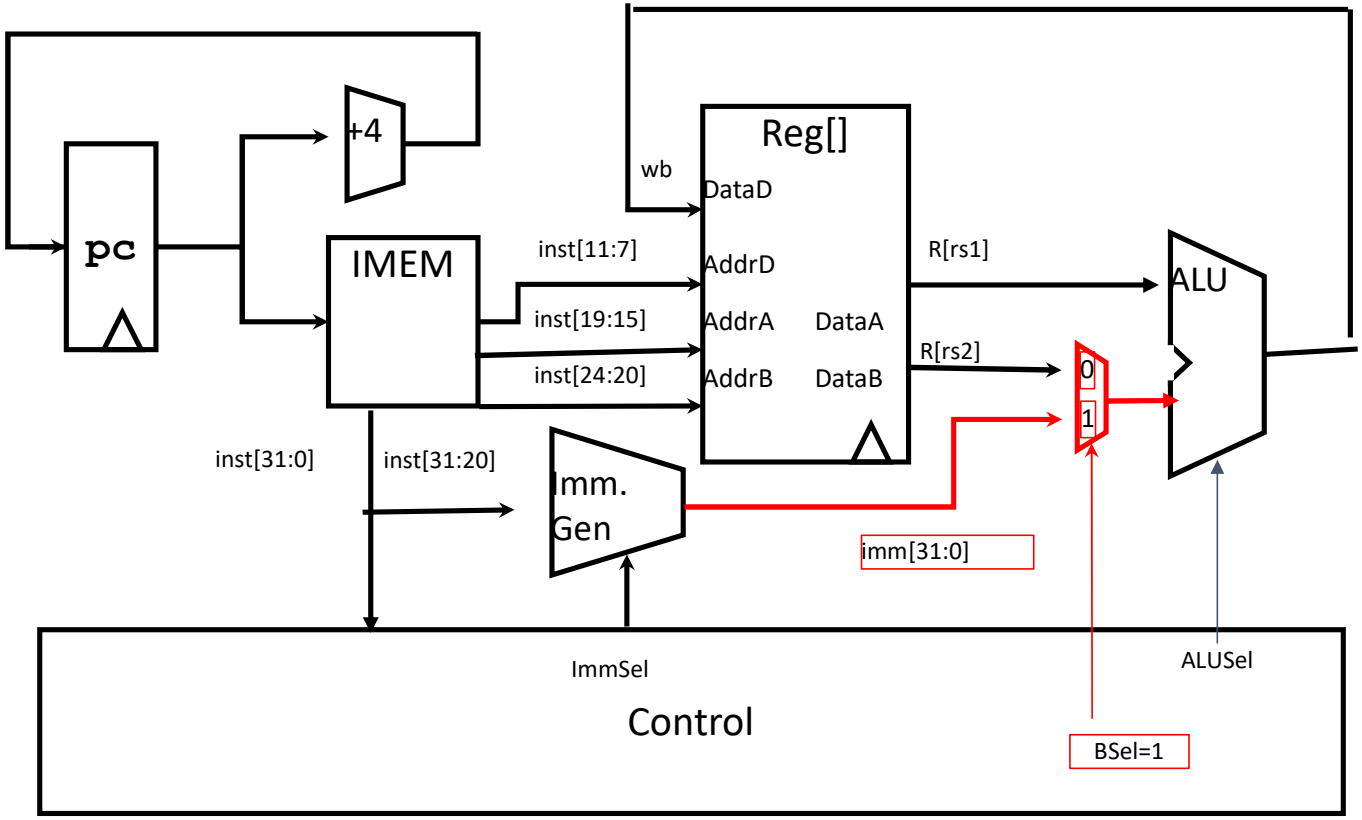
压缩形式：**c.lwsp** rd, offset; **c.lw** rd, offset(rs1)

31 … 20	19 … 15	14 … 12	11 … 7	6 … 0
offset[11:0]	rs1	010	rd	0000011

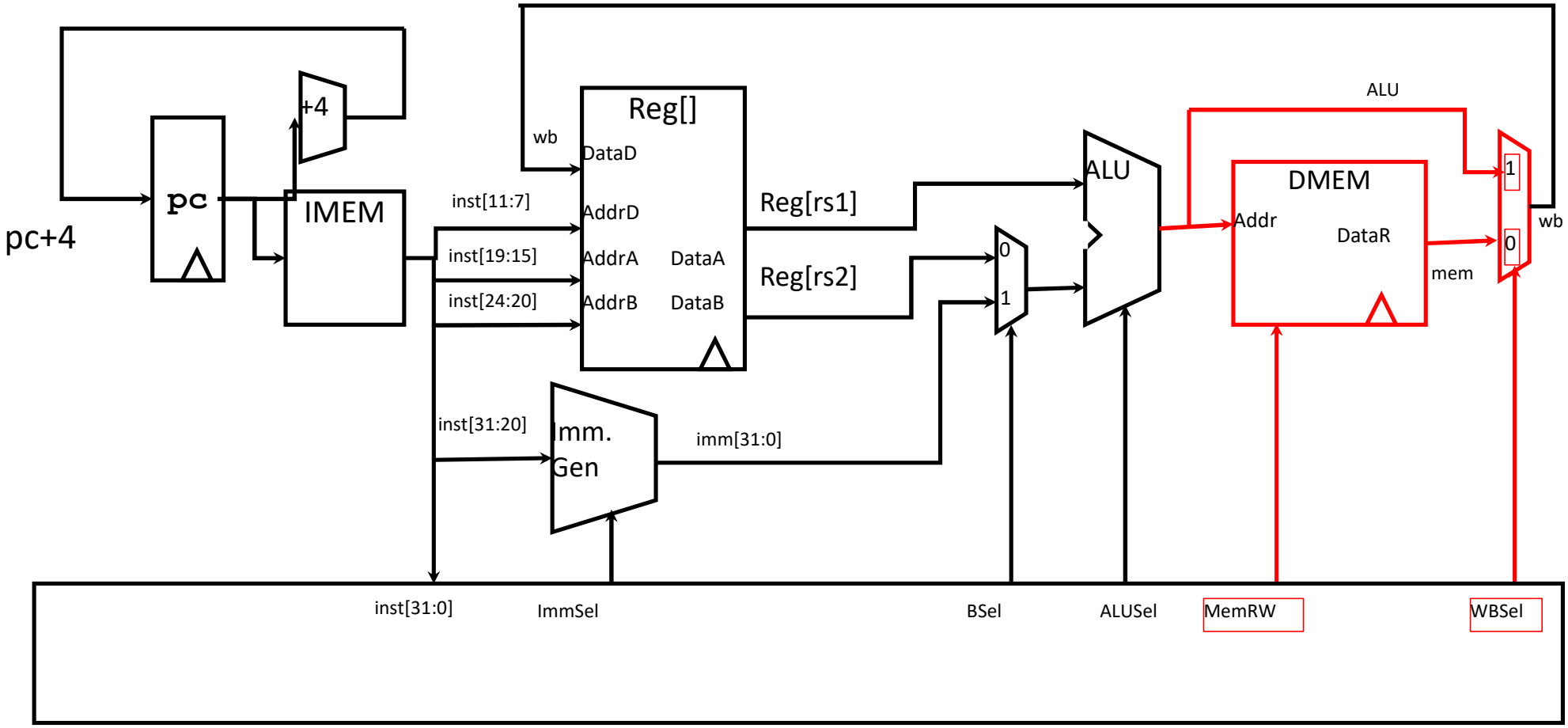
Implementing arithmetic I-Types



Implementing arithmetic I-Types



Adding `lw` to datapath



S-type

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type

SW rs2, offset(rs1) $M[x[rs1] + sext(offset) = x[rs2][31:0]$

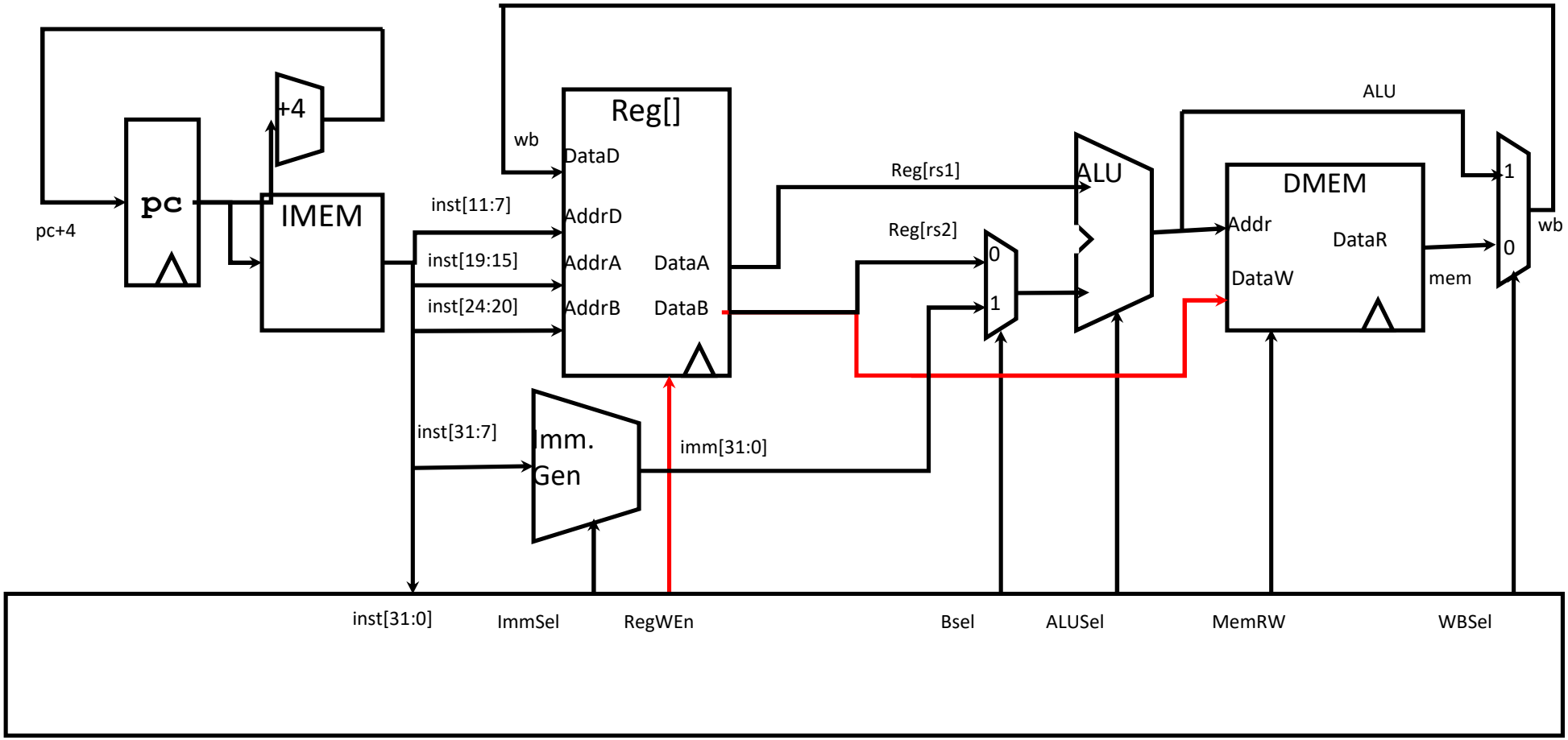
存字(Store Word). S-type, RV32I and RV64I.

将 x[*rs2*]的低位 4 个字节存入内存地址 x[*rs1*]+*sign-extend(offset)*。

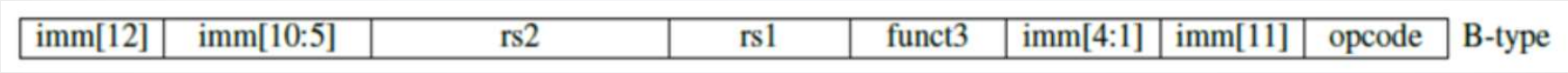
压缩形式： **c.swsp** rs2, offset; **c.sw** rs2, offset(rs1)

31 25	24 20	19 15	14 12	11 7	6 0
offset[11:5]	rs2	rs1	010	offset[4:0]	0100011

Adding `sw` to datapath



B-type

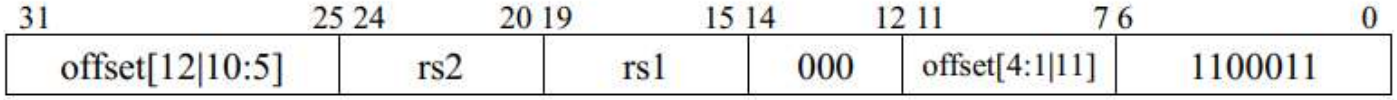


beq rs1, rs2, offset　　if (rs1 == rs2) pc += sext(offset)

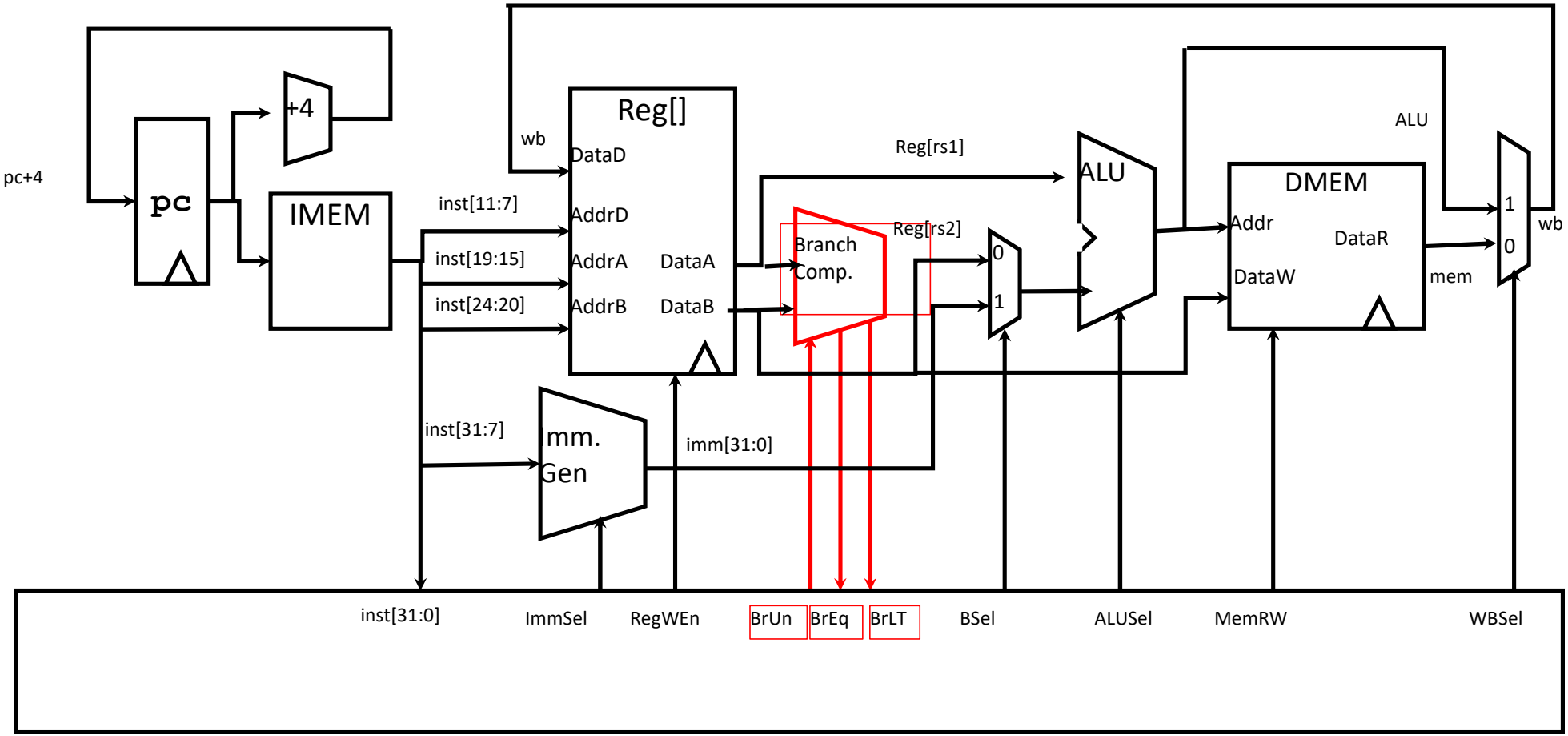
相等时分支 *(Branch if Equal)*. B-type, RV32I and RV64I.

若寄存器 x[*rs1*]和寄存器 x[*rs2*]的值相等，把 *pc* 的值设为当前值加上符号位扩展的偏移 *offset*。

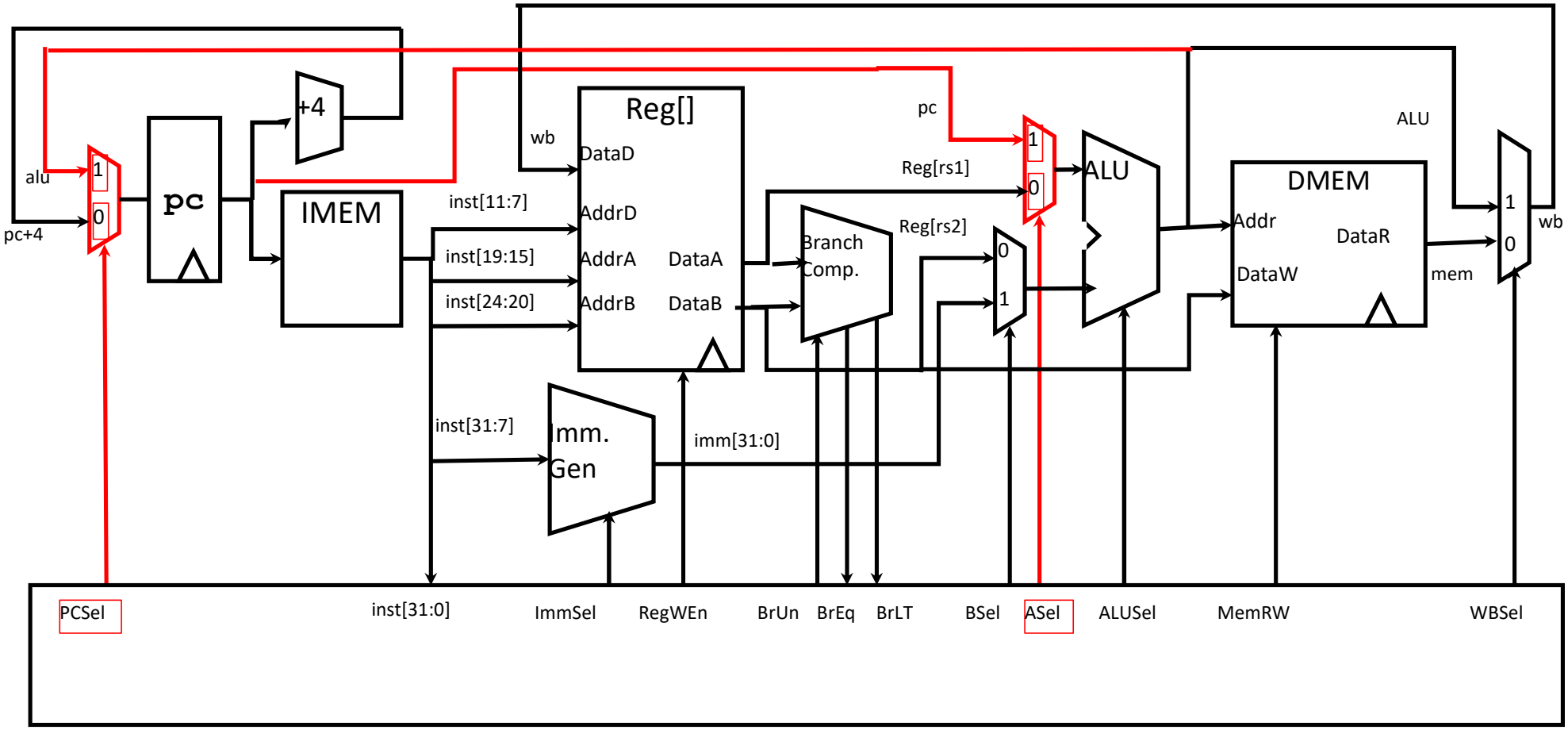
压缩形式：**c.beqz** rs1, offset



Adding branches to datapath



Adding branches to datapath



J-type

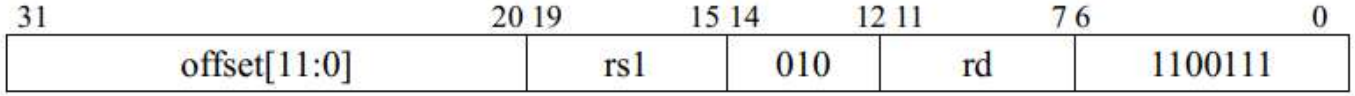
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode	J-type

jalr rd, offset(rs1)　　t =pc+4; pc=(x[rs1]+sext(offset))&~1; x[rd]=t

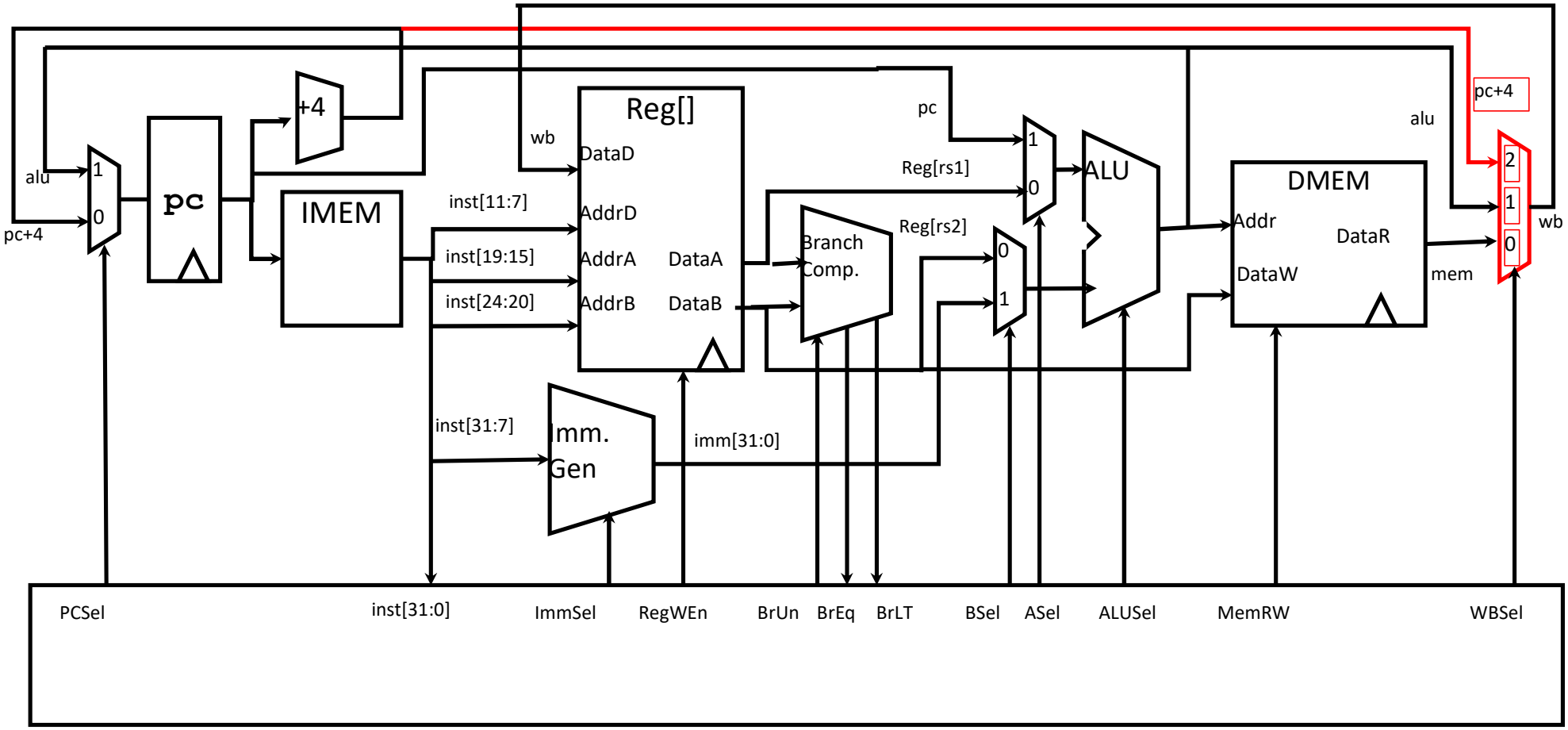
跳转并寄存器链接 *(Jump and Link Register)*. I-type, RV32I and RV64I.

把 *pc* 设置为 x[*rs1*]+*sign-extend*(*offset*)，把计算出的地址的最低有效位设为 0，并将原 *pc+4* 的值写入 f[*rd*]。*rd* 默认为 x1。

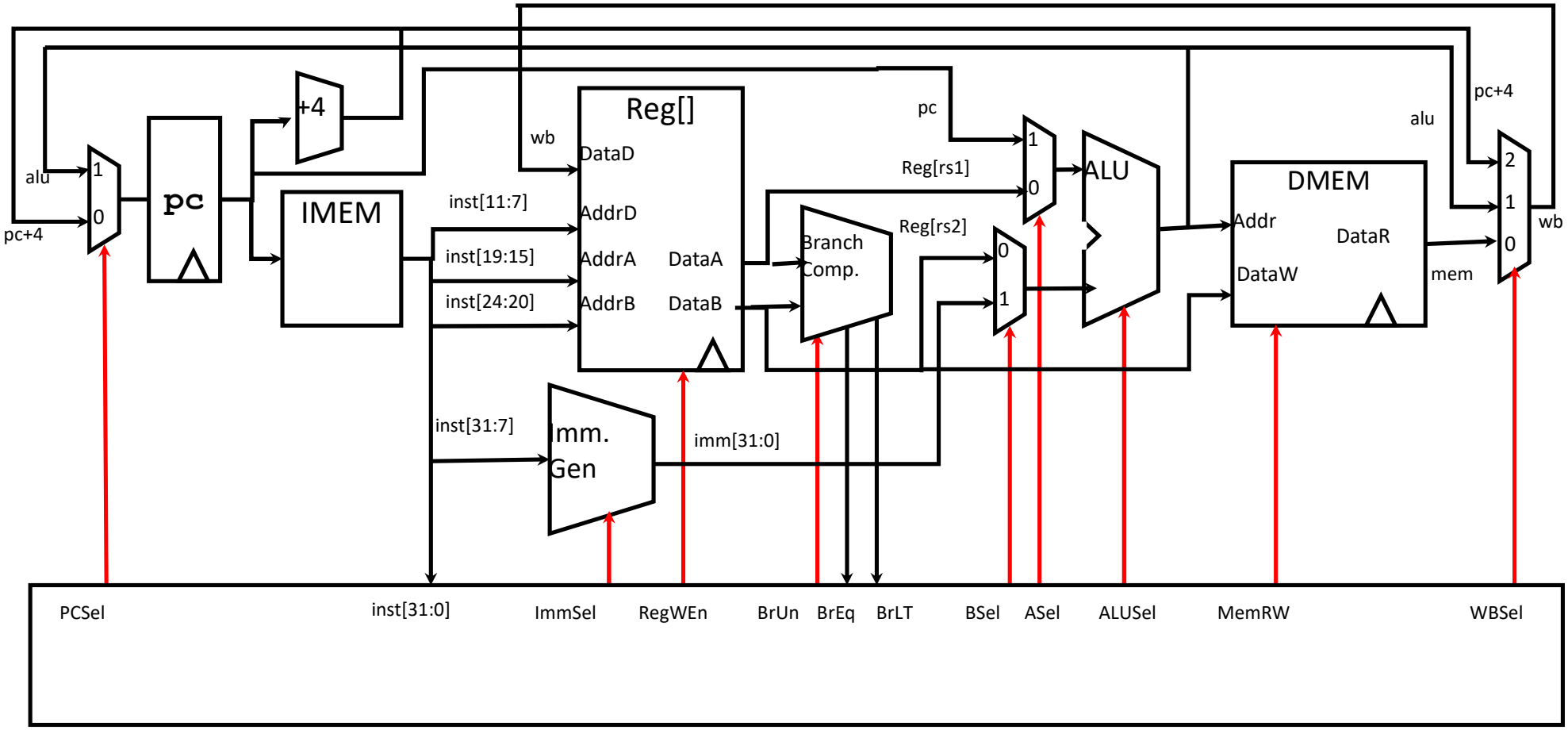
压缩形式：**c.jr** rs1；**c.jalr** rs1



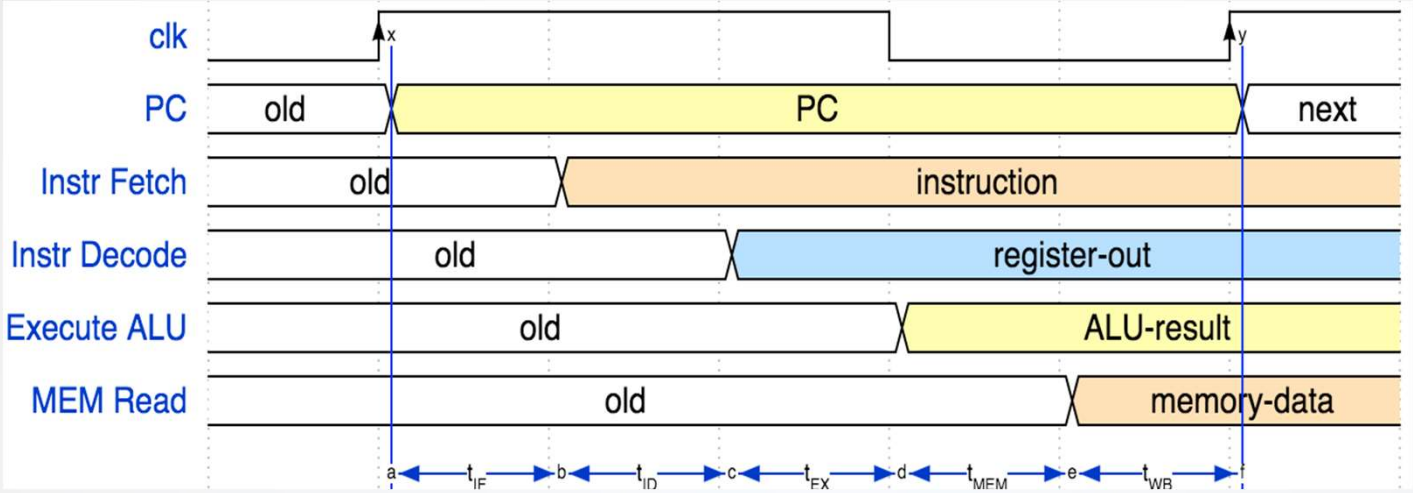
Adding `jalr` to datapath



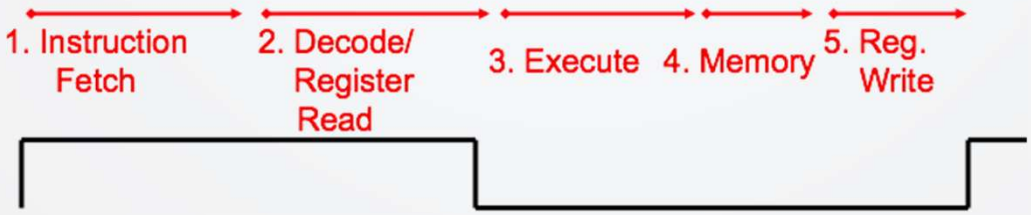
Single-Cycle RISC-V RV32I Datapath



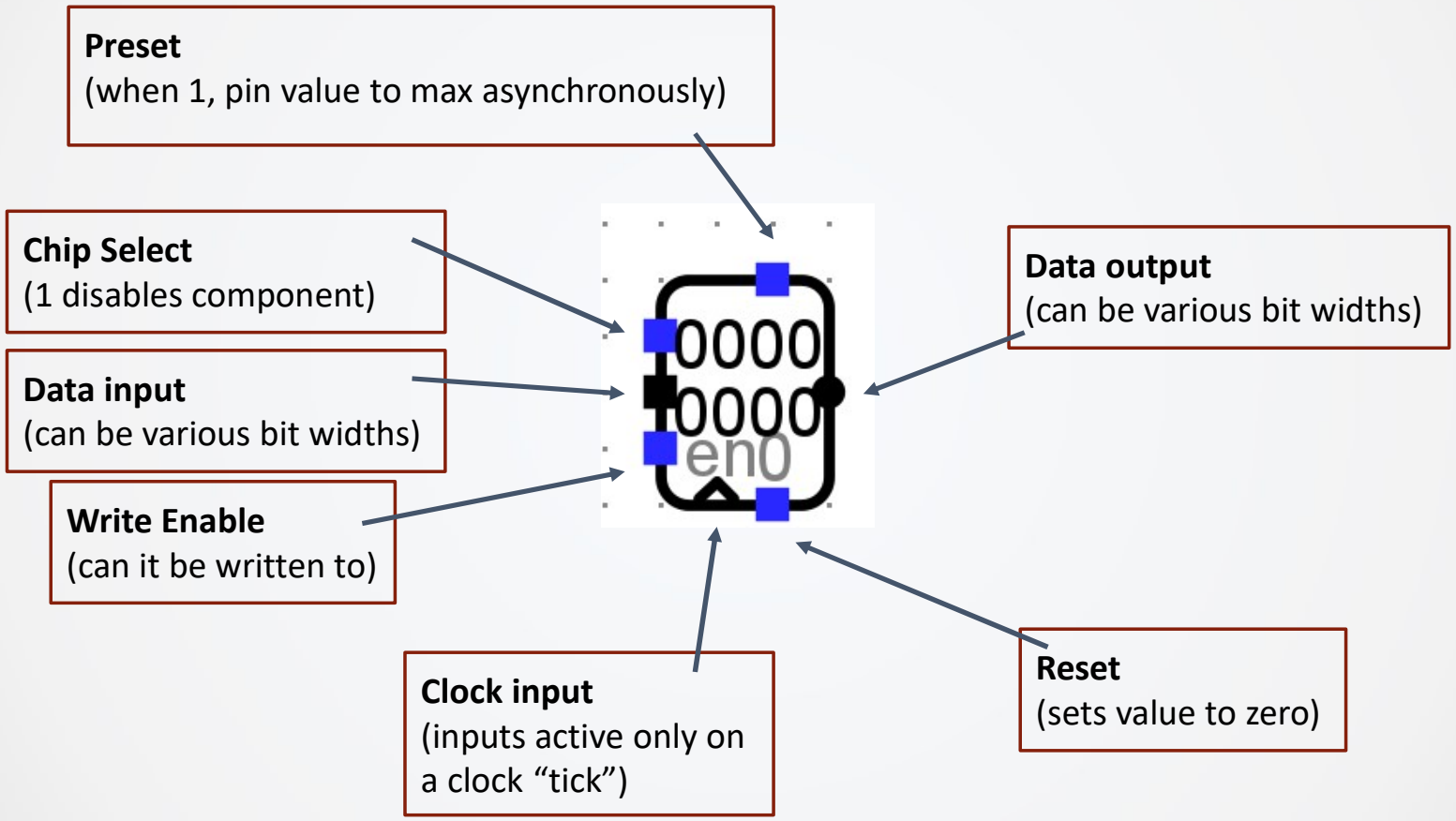
单周期CPU时序



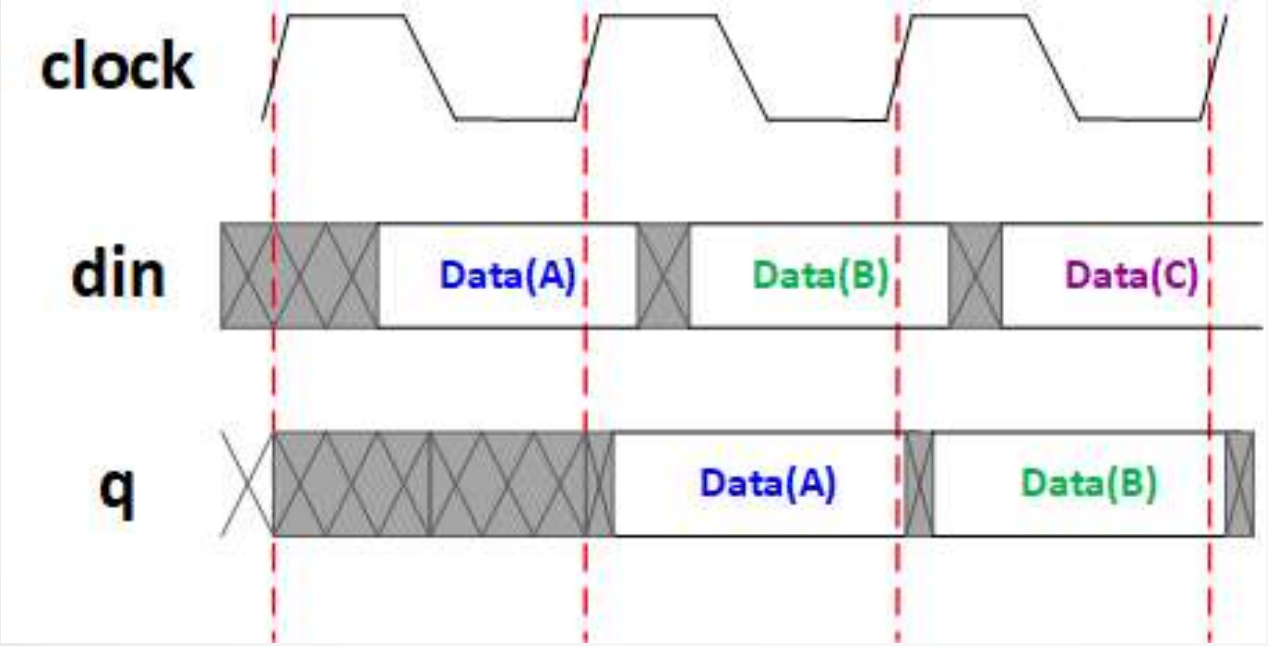
IF	ID	EX	MEM	WB	Total
IMEM	Reg Read	ALU	DMEM	Reg W	
200 ps	100 ps	200 ps	200 ps	100 ps	800 ps



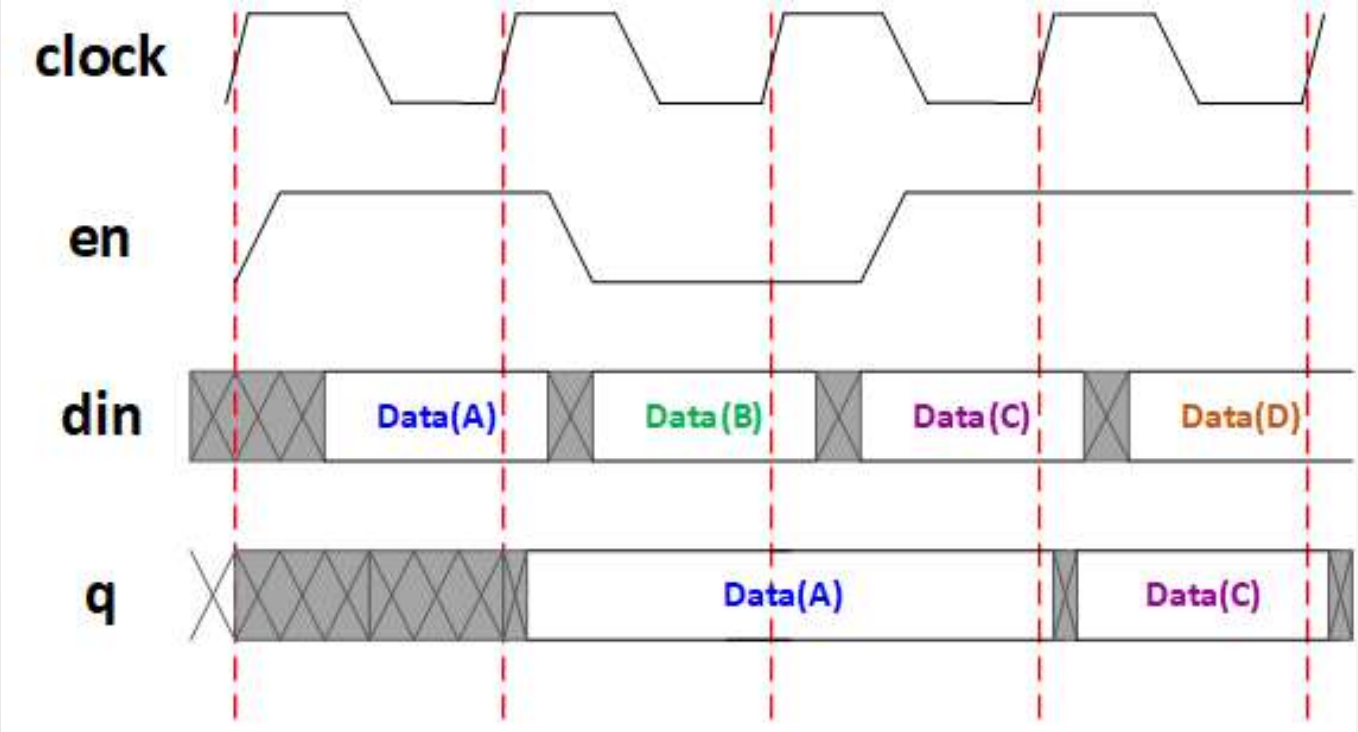
寄存器


Preset
(when 1, pin value to max asynchronously)
Chip Select
(1 disables component)
Data input
(can be various bit widths)
Write Enable
(can it be written to)
Clock input
(inputs active only on a clock “tick”)
Reset
(sets value to zero)
Data output
(can be various bit widths)
0000
0000
en0

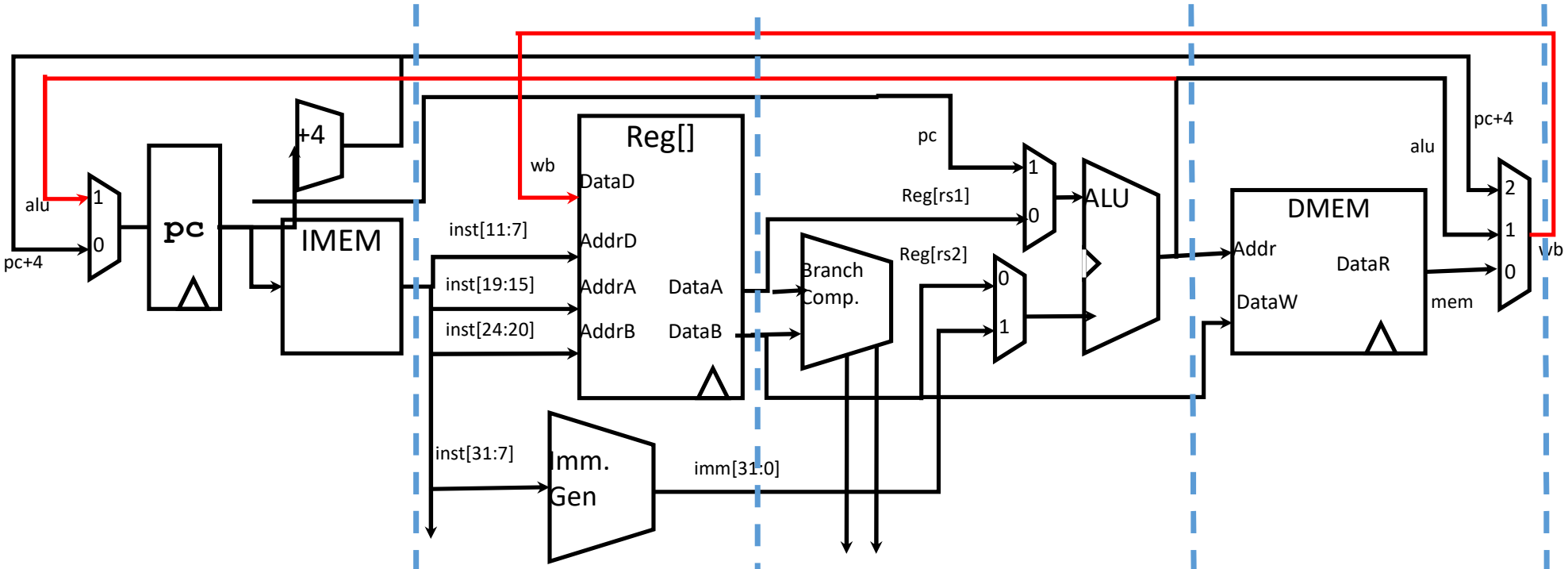
D触发器时序



带使能端D触发器时序



Pipelining RISC-V RV32I Datapath



Instruction Fetch (F)

Instruction Decode/Register Read (D)

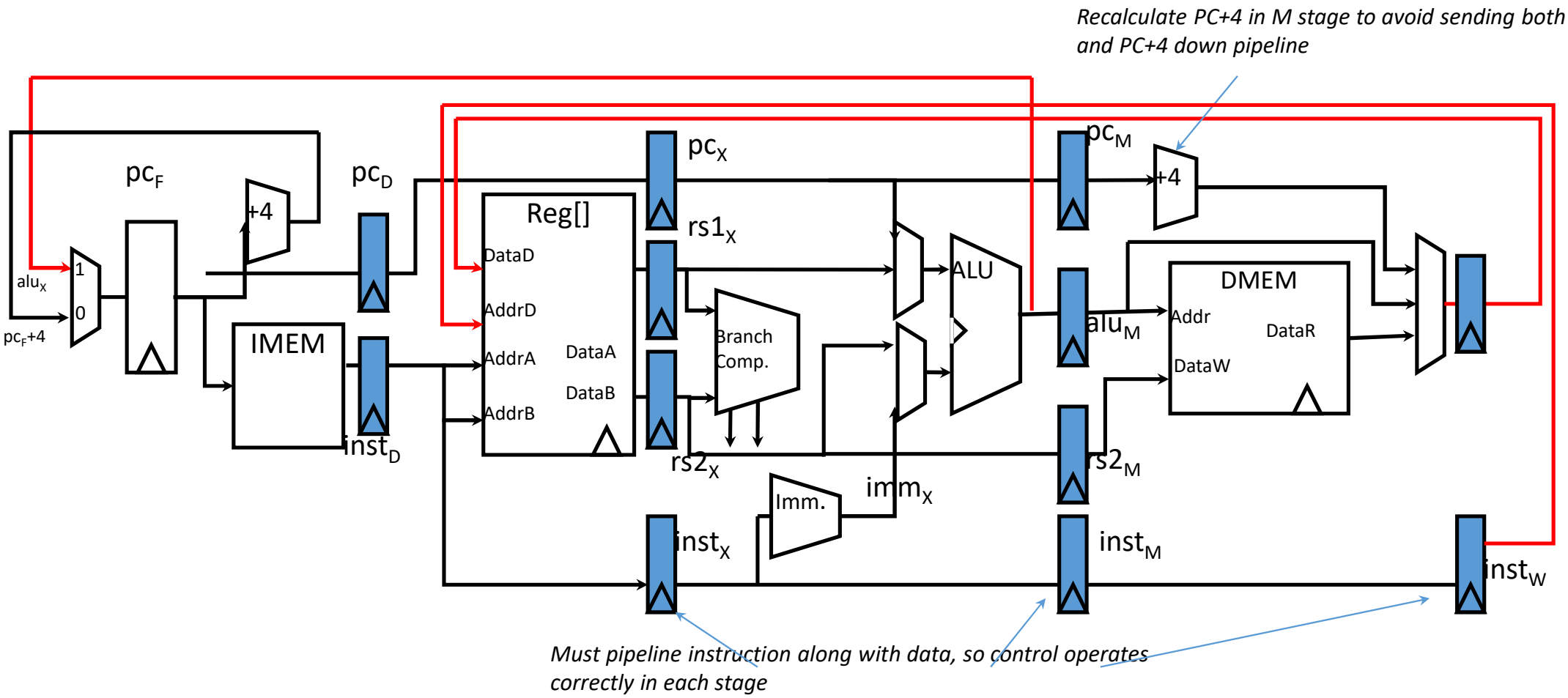
ALU Execute (X)

Memory Access (M)

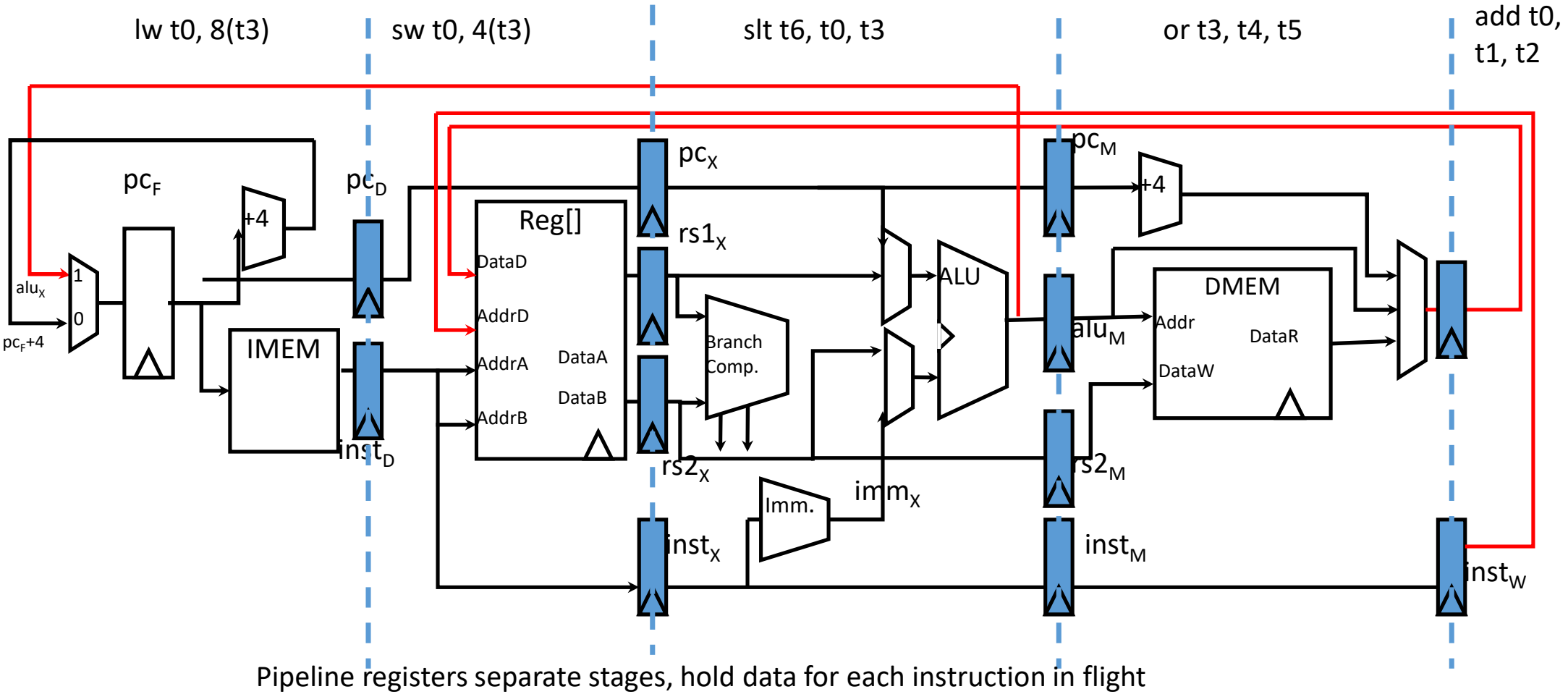
Write Back (W)

NOTE: Control signals are also pipelined!

Pipelined RISC-V RV32I Datapath

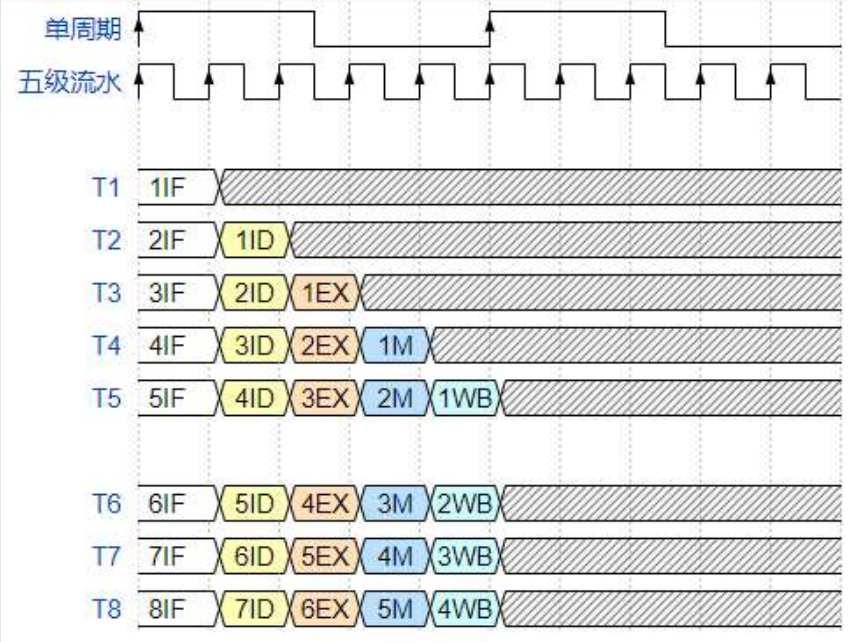


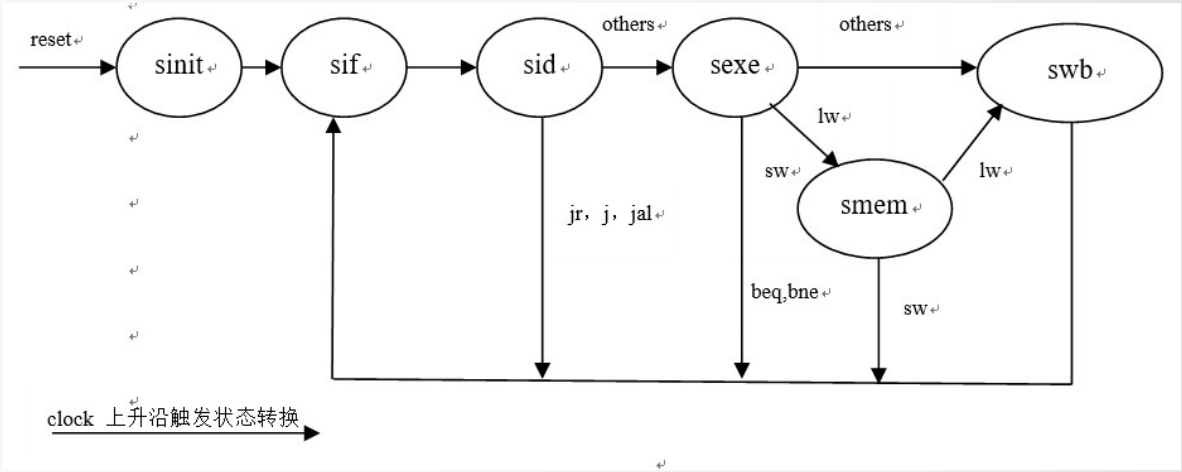
Each stage operates on different instruction



Pipeline registers separate stages, hold data for each instruction in flight

流水与多周期（状态机）





流水：以面积换性能，以空间换时间　VS　**状态机**：以性能换面积，以时间换空间

- 状态机，需要多个时钟周期才能完成一条指令的所有操作，每个时钟周期完成状态机的一个状态（分别为取值、译码、执行、访存和写回）。通过使用状态机，可以省略上述流水中的寄存器开销，还可以复用组合逻辑数据通路，因此面积开销比较小，但是每条指令都需要5个周期才能完成，吞吐量和性能很差。（早期原始的8051单片机）

实验一：RISC-V三段流水仿真

实验目的

Part A 理解流水线基本概念，掌握三段理想流水线RISC-V CPU工作流程

Part B 理解控制冒险的基本原理，掌握控制冲突流水线处理流程

实验原理 — 流水线的相关冲突（hazard）

- 资源相关
  - 取操作数与取指令都需要访问主存
  - 多周期方案中计算PC、分支地址，运算指令 复用ALU
  - 增加部件消除
  - 寄存器组先写后读
- 分支相关
  - 控制IF段进行分支跳转
  - 提前取出指令作废，流水线清空
- 数据相关
  - 指令操作数依赖于前一条指令的执行结果
  - 引起流水线停顿直到数据写回

实验原理— 分支指令

无条件跳转指令

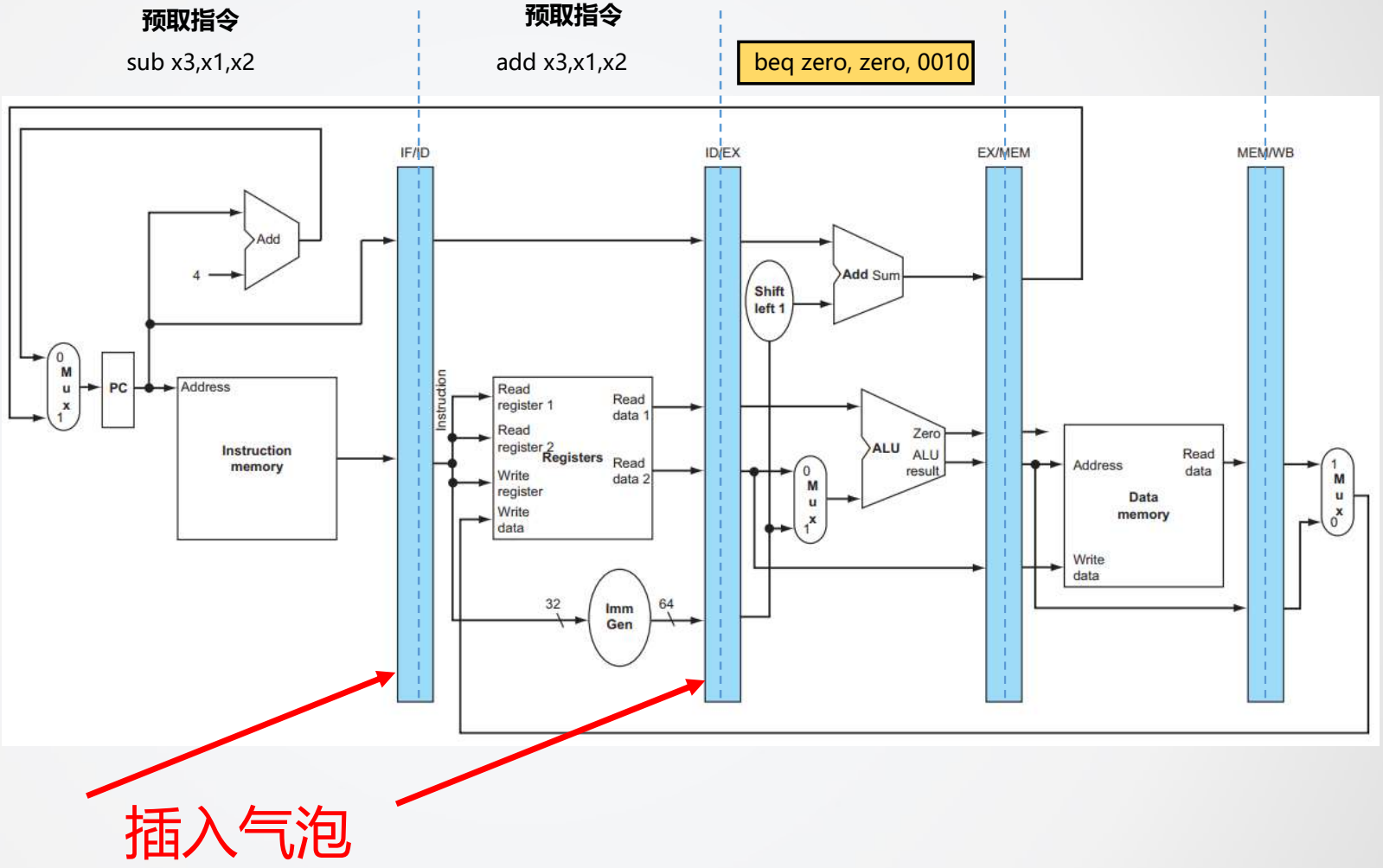
指令汇编格式	功能	操作
jal rd, label	跳转并链接	x[rd] = pc+4; pc += sext(offset)
jalr rd, offset(rs1)	跳转并寄存器链接	t =pc+4; pc=(x[rs1]+sext(offset))&~1; x[rd]=t

有条件跳转指令

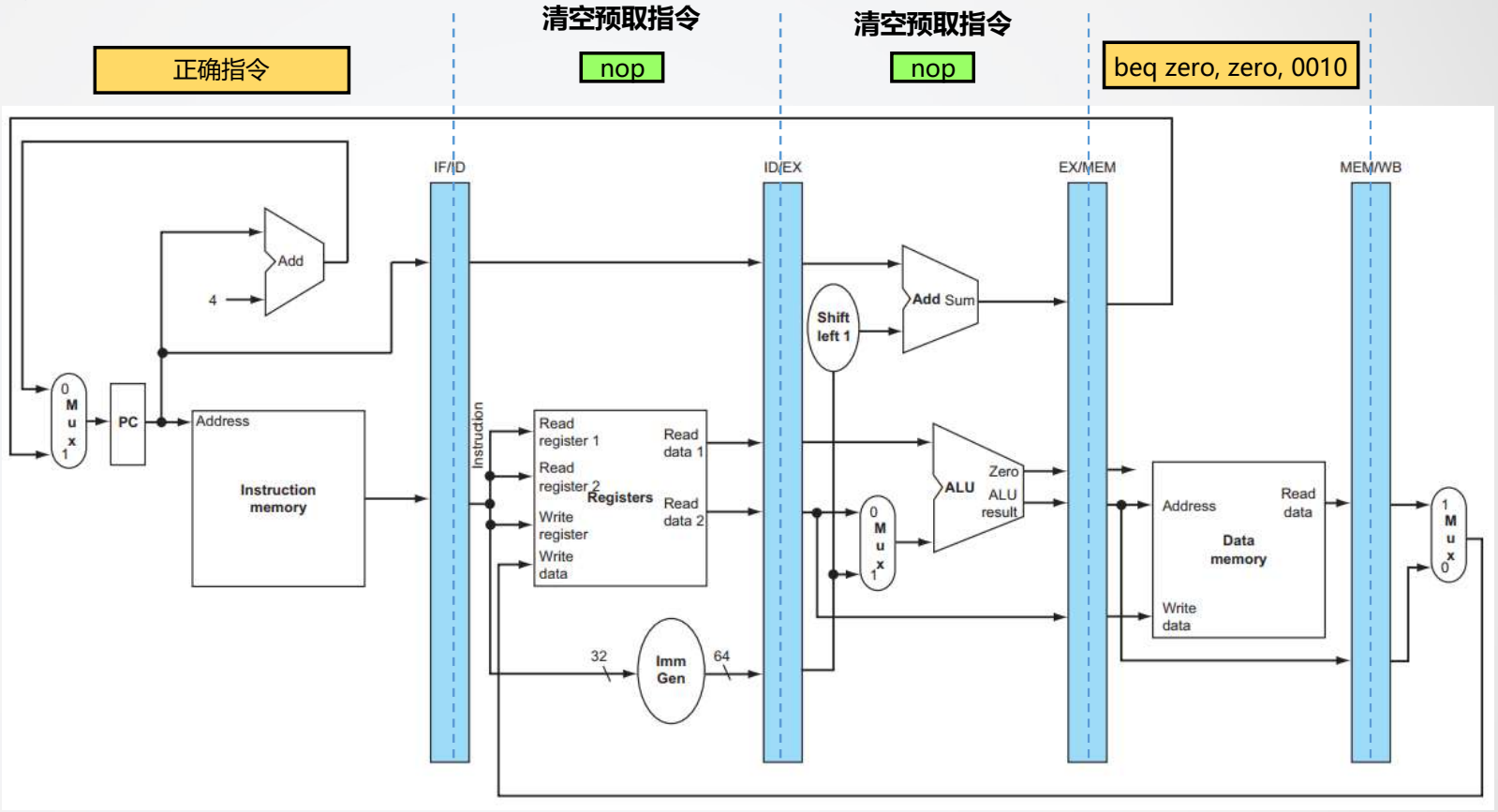
指令汇编格式	功能	操作
beq rs1, rs2, offset	相等时跳转	if (rs1 == rs2) pc += sext(offset)
bne rs1, rs2, offset	不等时跳转	if (rs1 != rs2) pc += sext(offset)
blt rs1, rs2, offset	有符号数比较，小于时跳转	if (rs1< rs2) pc += sext(offset)
bltu rs1, rs2, offset	无符号数比较，小于时跳转	if (rs1< rs2) pc += sext(offset)
BGE rs1, rs2, offset	有符号数比较，大于等于时跳转	if (rs1 ≥ rs2) pc += sext(offset)
BGEU rs1, rs2, offset	无符号数比较，大于等于时跳转	if (rs1 ≥ rs2) pc += sext(offset)

实验原理 — 分支相关

#分支相关测试程序
beq zero, zero, 0010
add x3,x1,x2
sub x3,x1,x2



实验原理 — 分支相关


正确指令
清空预取指令
nop
清空预取指令
nop
beq zero, zero, 0010
IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M u x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 1
Add Sum
ALU
Zero
ALU result
Address
Data memory
Read data
Write data
32
Imm Gen
64

实验原理——分支相关时空图

实验步骤

- 修改流水接口部件，使其能实现同步清零插入气泡的功能。即在流水接口部件中，增加多路选择器，由多路选择器决定寄存器的输入：本段处理完成的数据或清零信号。
- 加载分支相关测试程序进行功能调试，该程序包含有无条件跳转指令和有条件跳转指令，使其能在流水线上正确运行。

关于halt停机判断

- 需要把instruction传递到第三级流水，同时判断第一级流水和第三级流水的instruction是否都0，如果是，则给出halt信号，停止计数。

实验提交

- **时间截点**

  三个星期内，即6月9日之前提交，**否则视为未提交**

- **需提交的内容**
  - 电路文件
  - 运行结果截图
  - 把设计思路写到实验报告中，实验报告格式不限

- **提交邮箱：hitsz_arch2020@163.com**