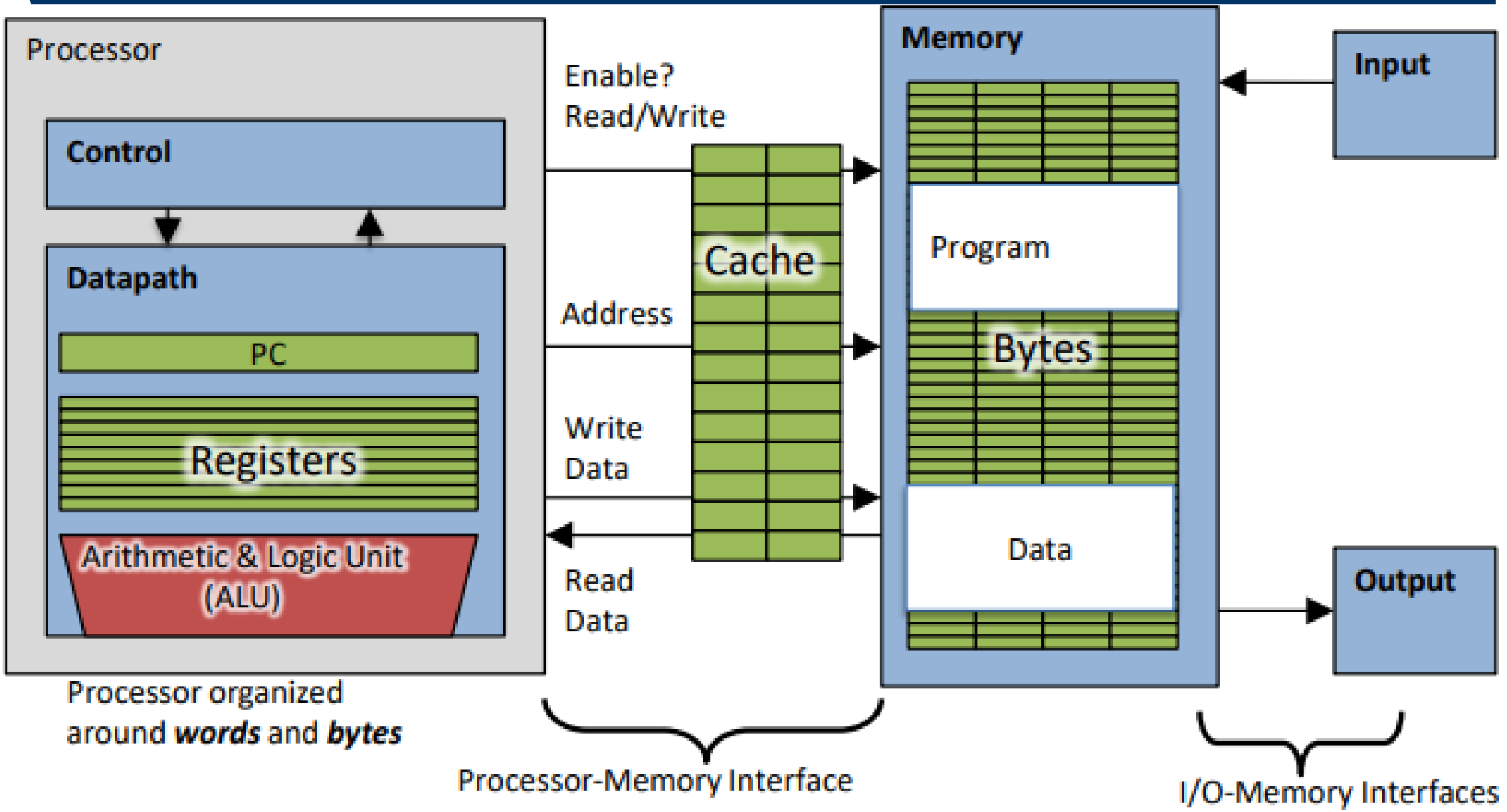
第六章 处理器设计

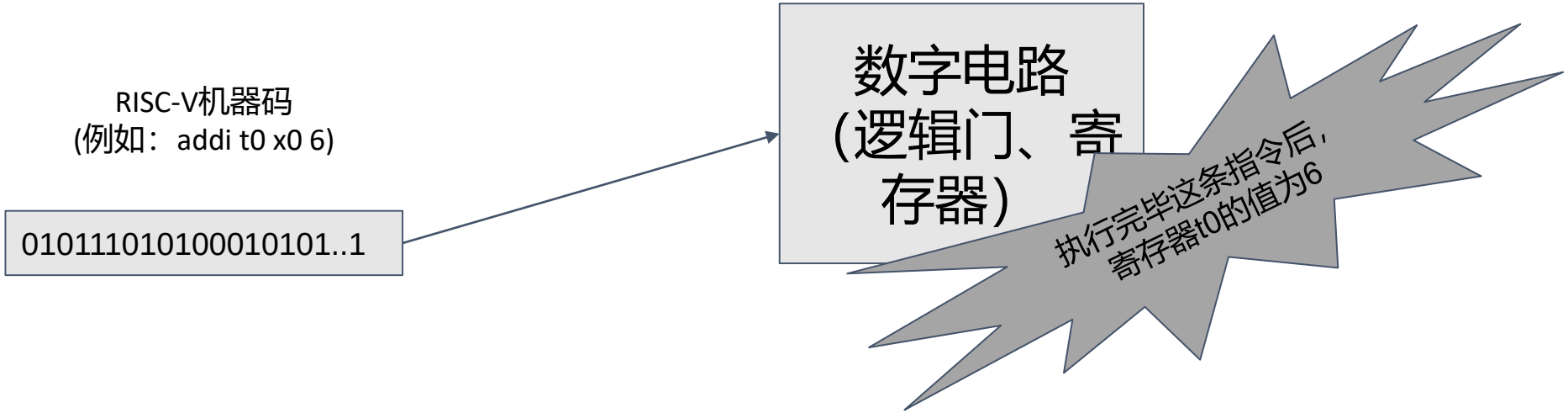
- RISC-V数据通路
  - 数据通路概念
  - RISC-V部分指令的数据通路
- RISC-V控制器

单核计算机系统


Processor
Control
Datapath
PC
Registers
Arithmetic & Logic Unit (ALU)
Enable?
Read/Write
Address
Write Data
Read Data
Cache
Memory
Program
Bytes
Data
Input
Output
Processor organized around words and bytes
Processor-Memory Interface
I/O-Memory Interfaces

CPU

- 功能层面的定义：
  - 构建一个能够通过输入的机器码，执行相应操作、并保持相应状态的数字电路



CPU的组成部分

- **数据通路**是处理器中执行所需操作的硬件
  - 执行控制器的操作（例如，控制器告诉数据通路，执行add指令，则数据通路就会将操作数喂给加法器）

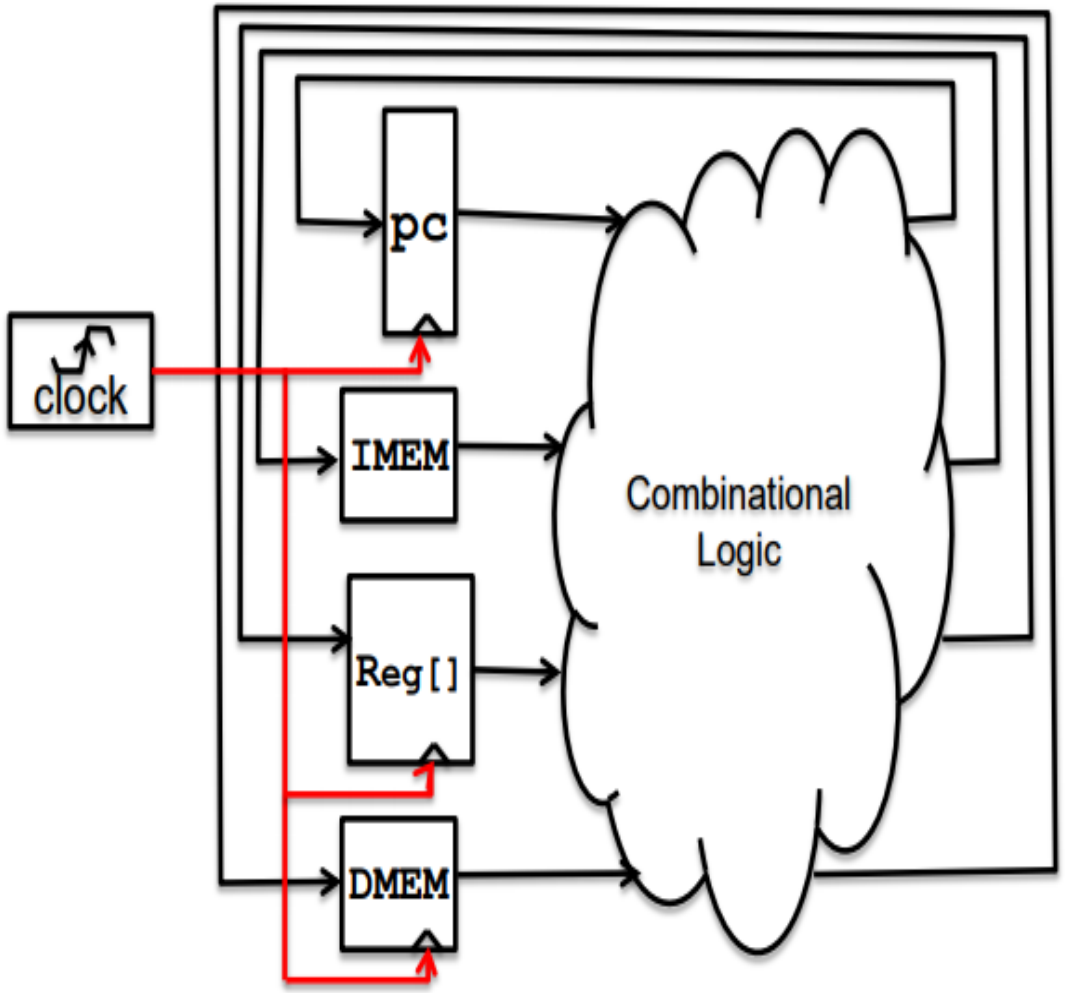
≈≈处理器的**四肢**

- **控制器**是对数据通路要做什么操作进行调度的硬件结构
  - 告诉数据通路：需要执行什么操作？需要读内存吗？需要写寄存器吗？写哪个寄存器？读哪个寄存器？

≈≈处理器的**大脑**

单周期CPU设计模型

- 每个时钟节拍执行一条指令
- 当前状态值通过一系列组合逻辑得到当前指令机器码，以及对应的操作码类型，操作数，结果。
- 在下一时钟上升沿到来时，所有组合逻辑单元输出的状态值保持或者更新对应寄存器，同时开始下一个时钟周期的执行操作。



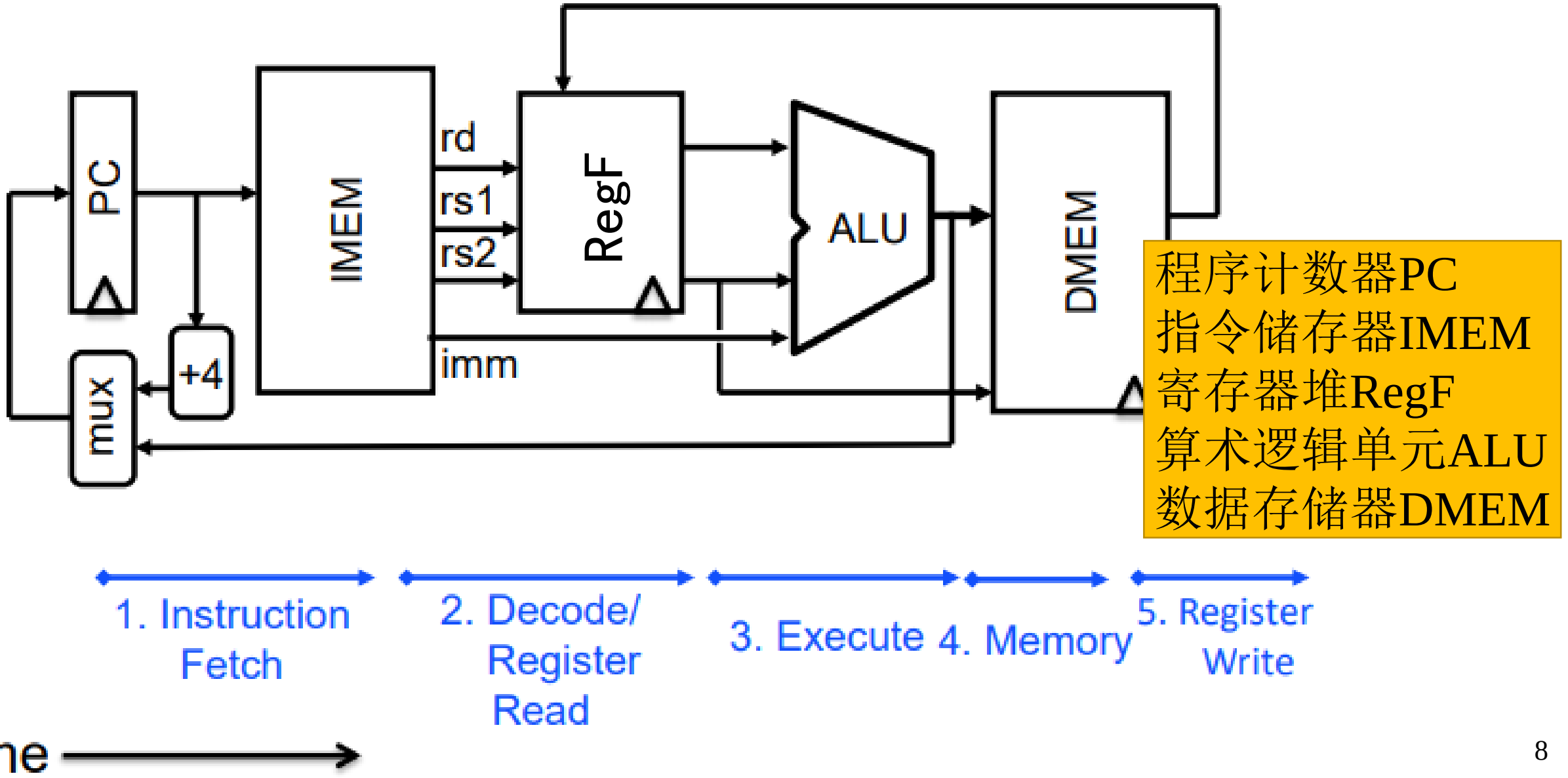
分阶段的数据通路——概述

- 问题：单块的逻辑结构完成对所有指令从取指到执行的所有操作，这种设计**笨重，效率低下**
- 分阶段的数据通路设计：将原来执行一条指令的过程，拆分为几个阶段，然后将这几个阶段对应的电路结构前后串联起来构成完整的数据通路。
  - 电路规模更小，更便于设计实现
  - 便于修改优化一个阶段而不影响其他阶段

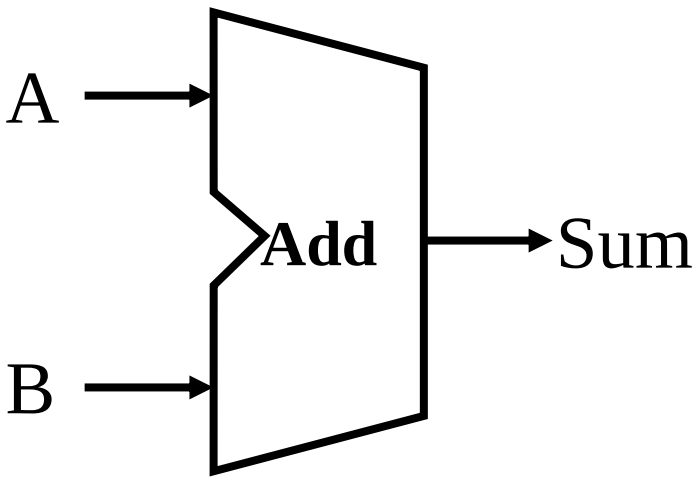
数据通路的五个阶段

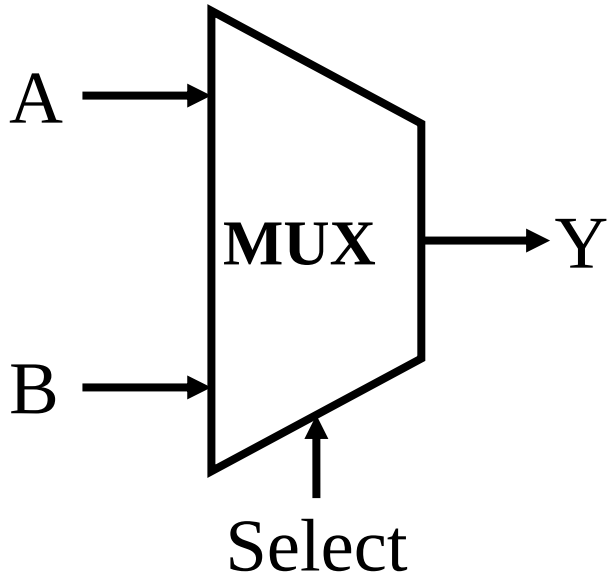
- 取指： Instruction Fetch (IF)
- 译码： Instruction Decode (ID)
- 执行： EXecute (EX) - ALU (Arithmetic-Logic Unit)
- 访存： MEMory access (MEM)
- 写回： Write Back to register (WB)

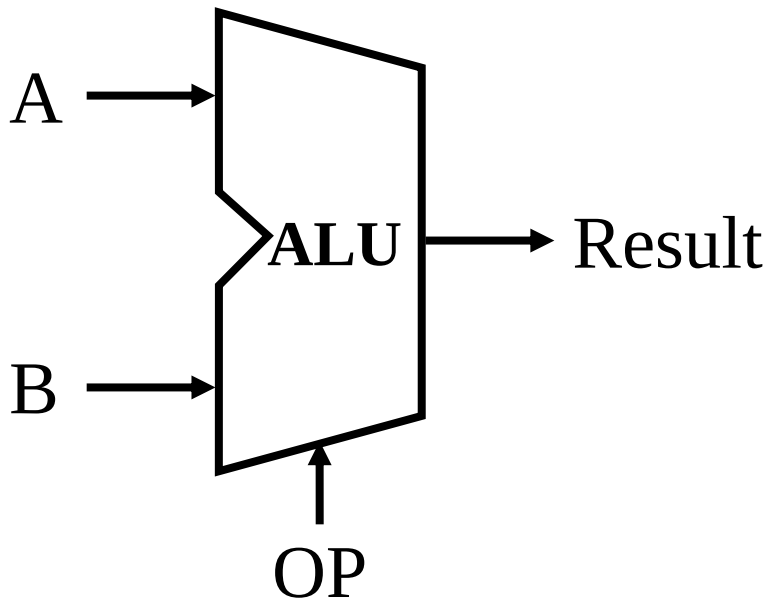
指令执行示意图


PC
IMEM
rd
rs1
rs2
RegF
ALU
DMEM
imm
+4
mux
程序计数器PC
指令储存器IMEM
寄存器堆RegF
算术逻辑单元ALU
数据存储器DMEM
1. Instruction Fetch
2. Decode/ Register Read
3. Execute
4. Memory
5. Register Write
time

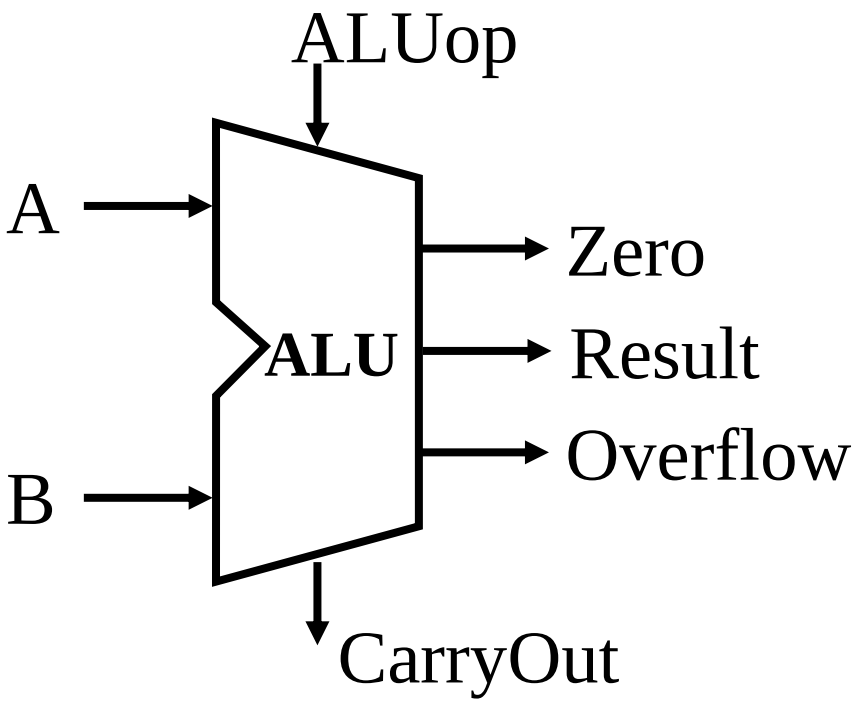
数据通路模块——组合逻辑单元







ALU功能需求

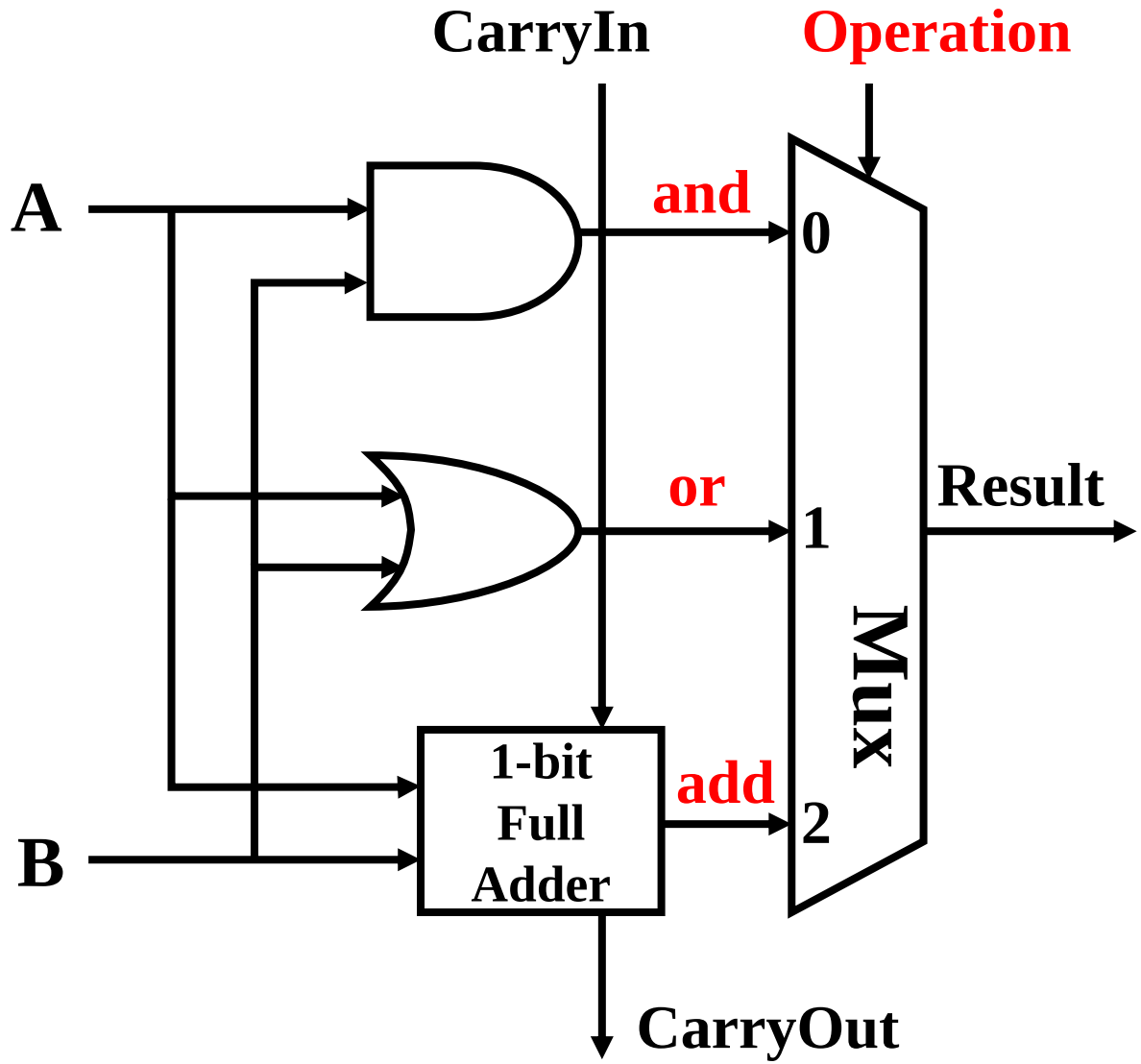


ALU Control(ALUop）	操作
0000	and
0001	or
0010	add
0110	subtract
0111	set on less than
1100	nor

ALU设计技巧

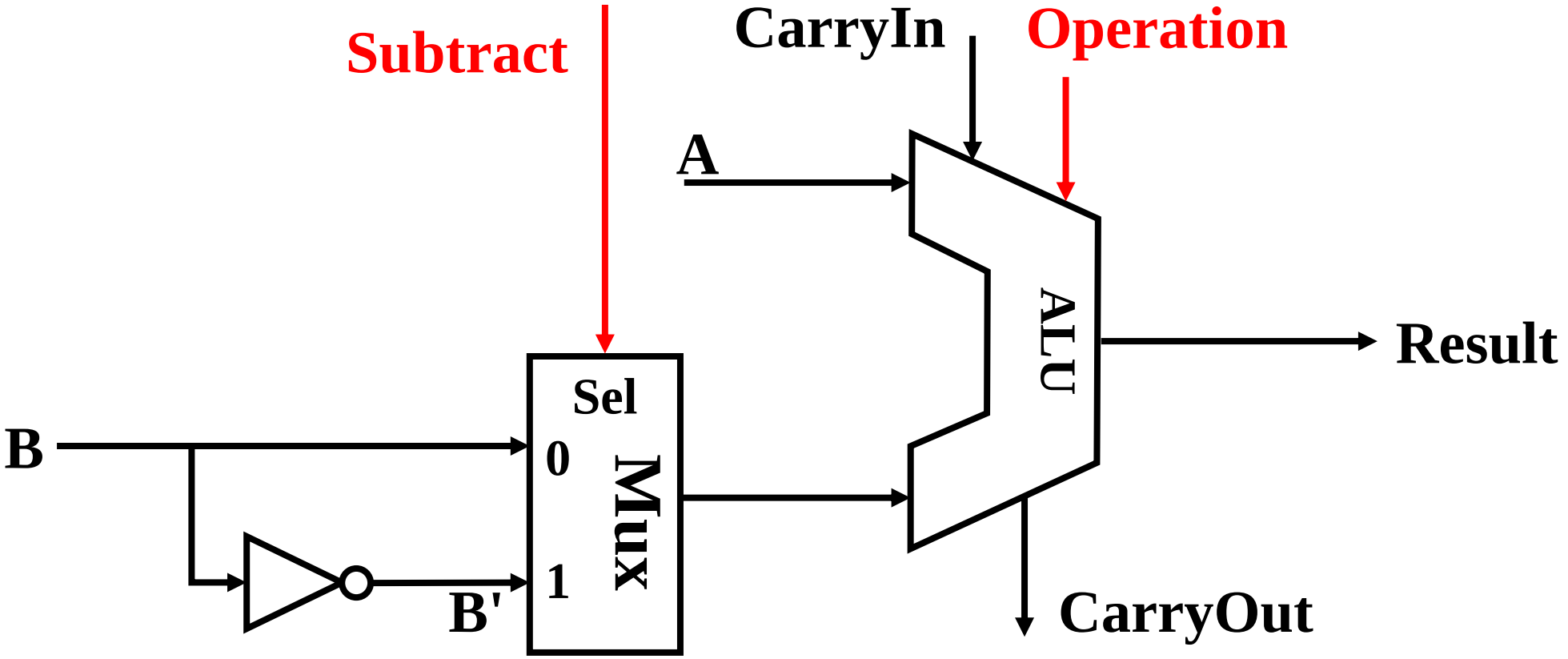
- 从简单开始：然后再进行扩展
- 分而治之：从1位ALU设计开始
- 利用已知的元件或者组件，组合起来实现复杂功能

ALU——简单运算

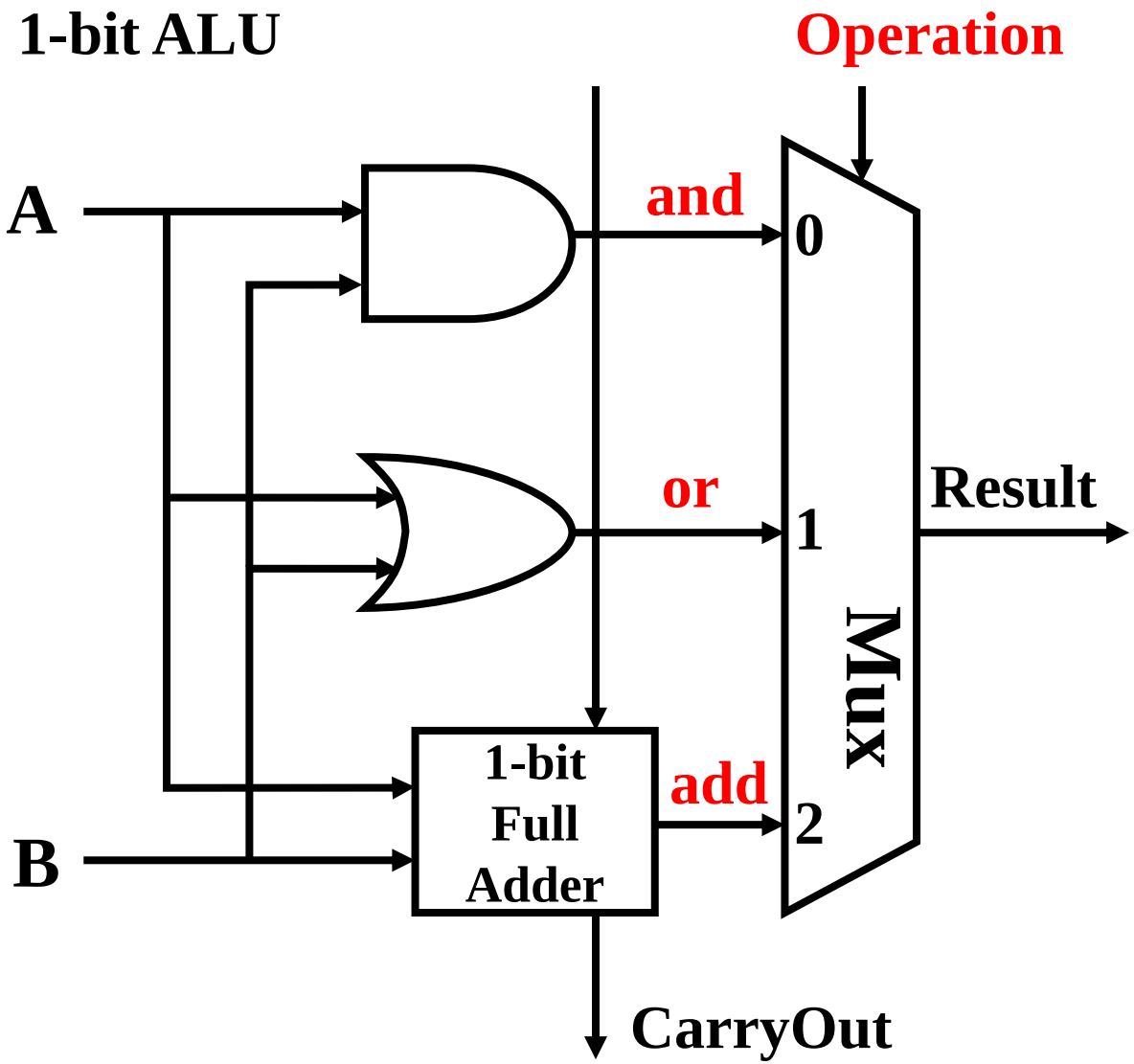

CarryIn
Operation
A
and
0
or
1
Result
Mux
1-bit
Full
Adder
add
2
B
CarryOut

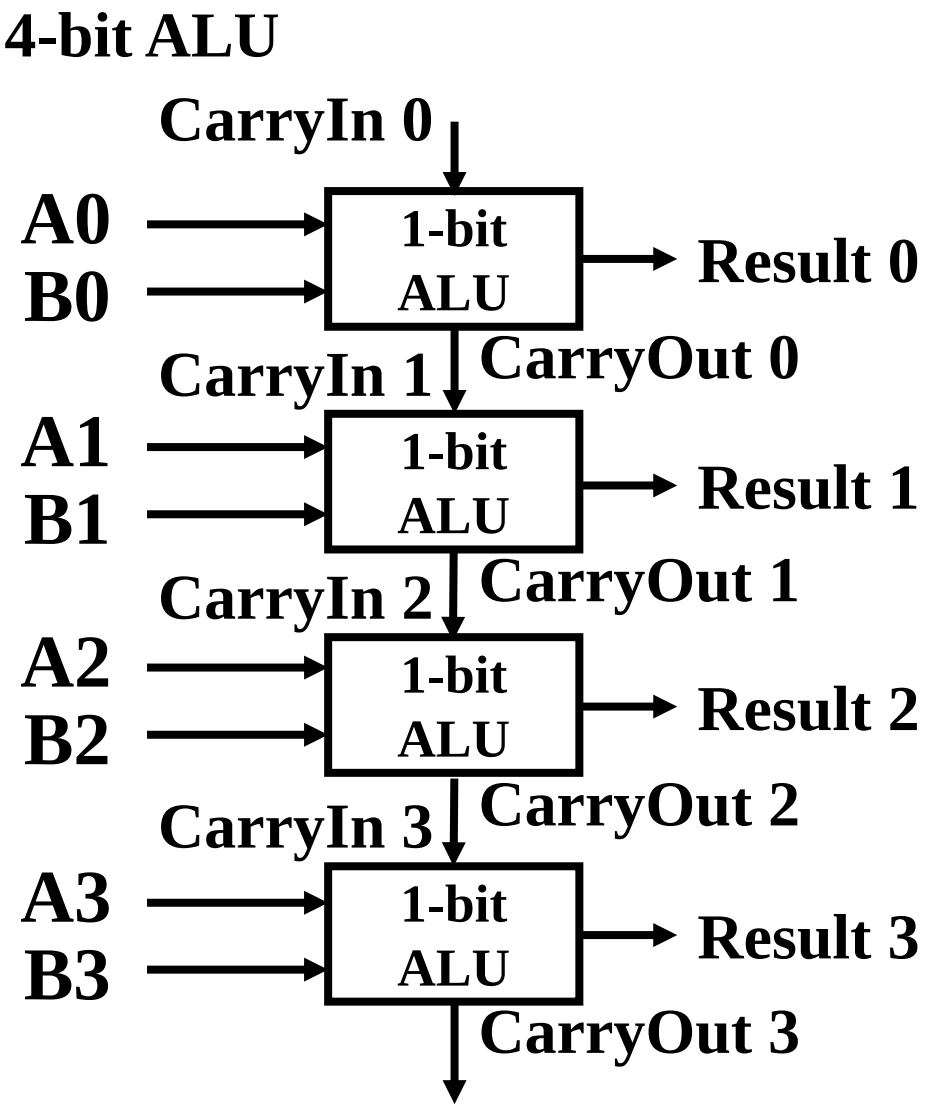
ALU——如何进行减法运算

- A-B=A+B'+1



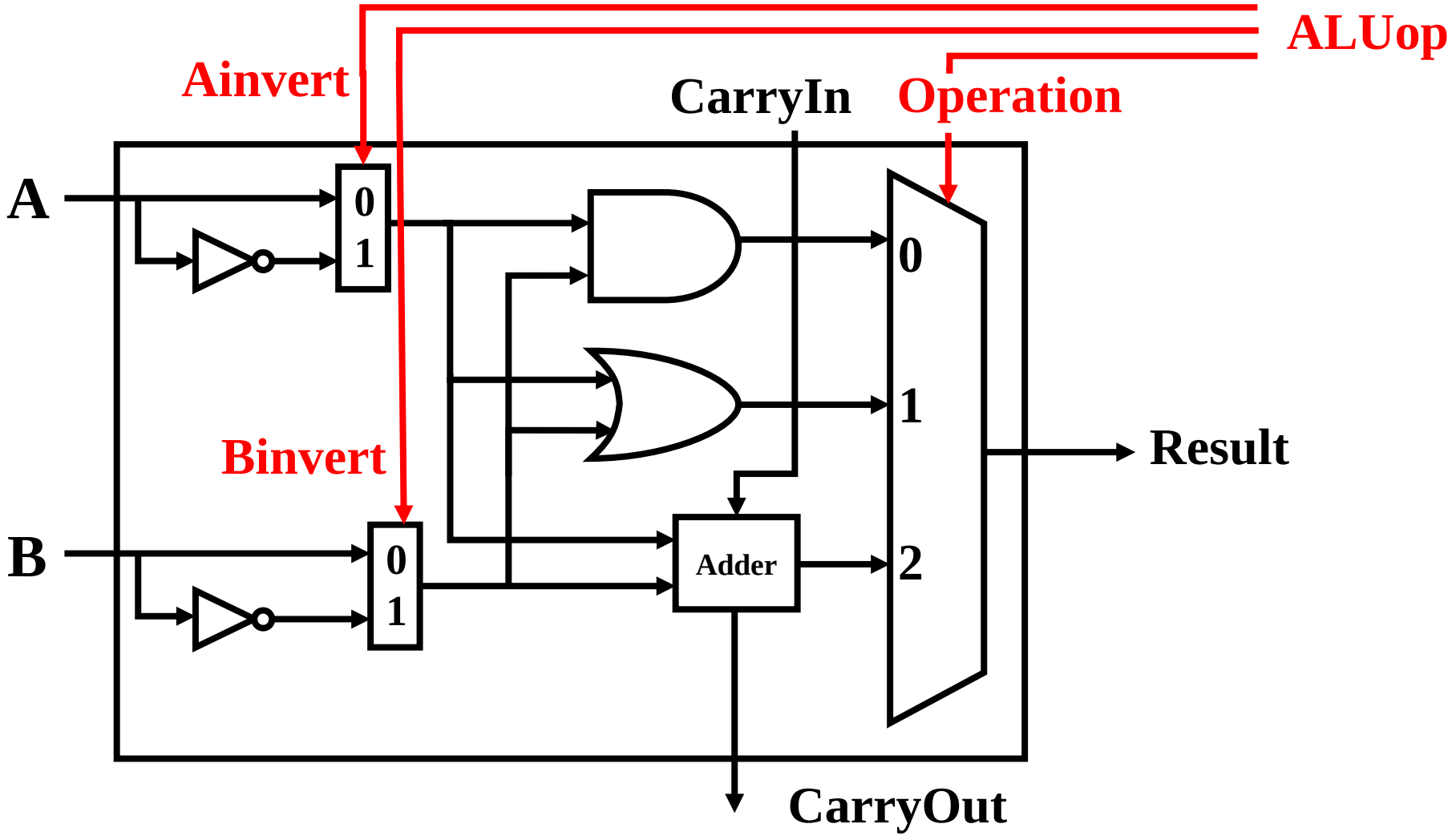
1-bit到多位ALU


1-bit ALU
Operation
A
B
and
or
add
0
1
2
Mux
Result
1-bit
Full
Adder
CarryOut


4-bit ALU
CarryIn 0
A0
B0
1-bit
ALU
Result 0
CarryOut 0
CarryIn 1
A1
B1
1-bit
ALU
Result 1
CarryOut 1
CarryIn 2
A2
B2
1-bit
ALU
Result 2
CarryOut 2
CarryIn 3
A3
B3
1-bit
ALU
Result 3
CarryOut 3

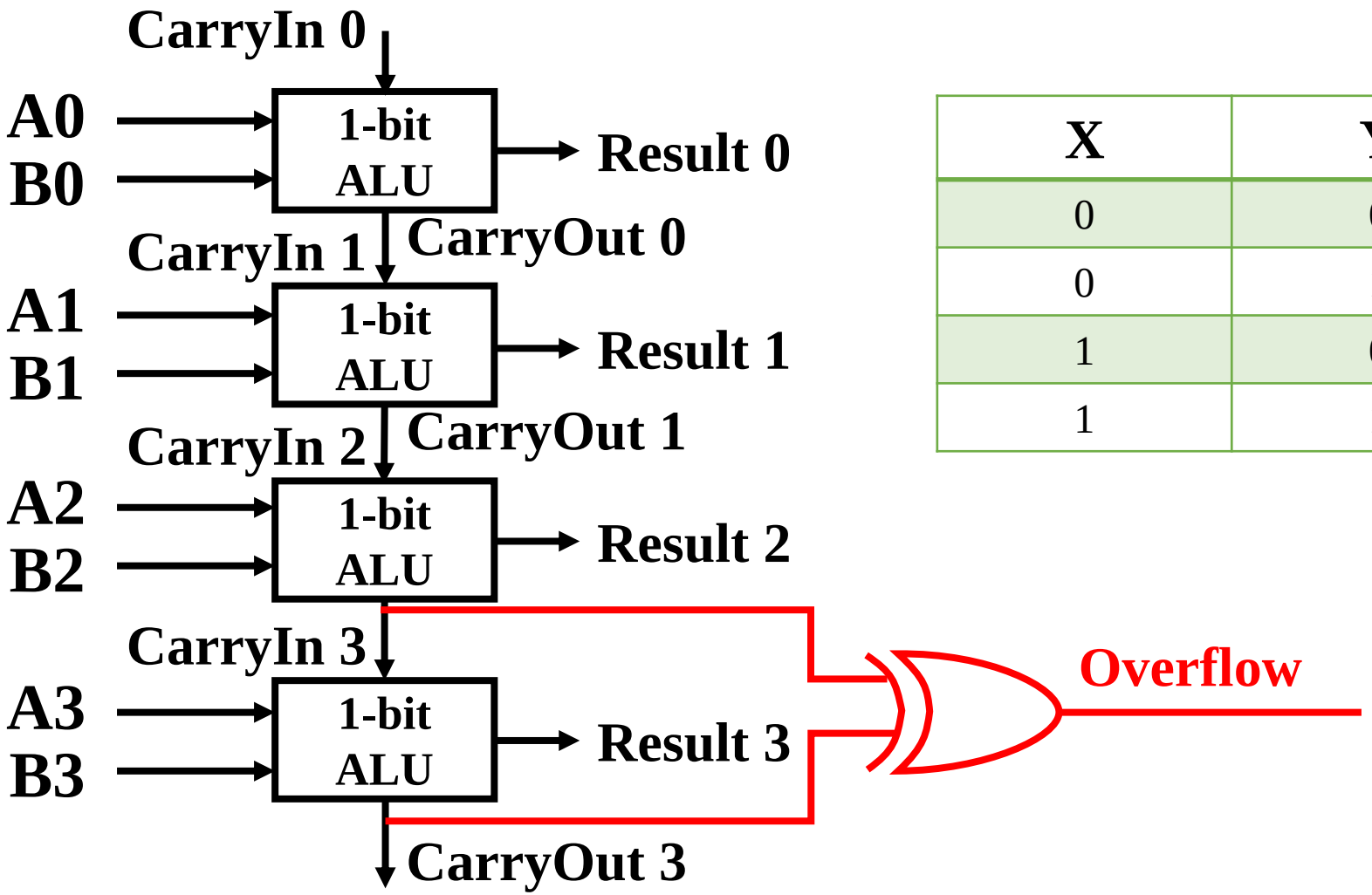
ALU——nor运算

A nor B= （not A） and (not B)



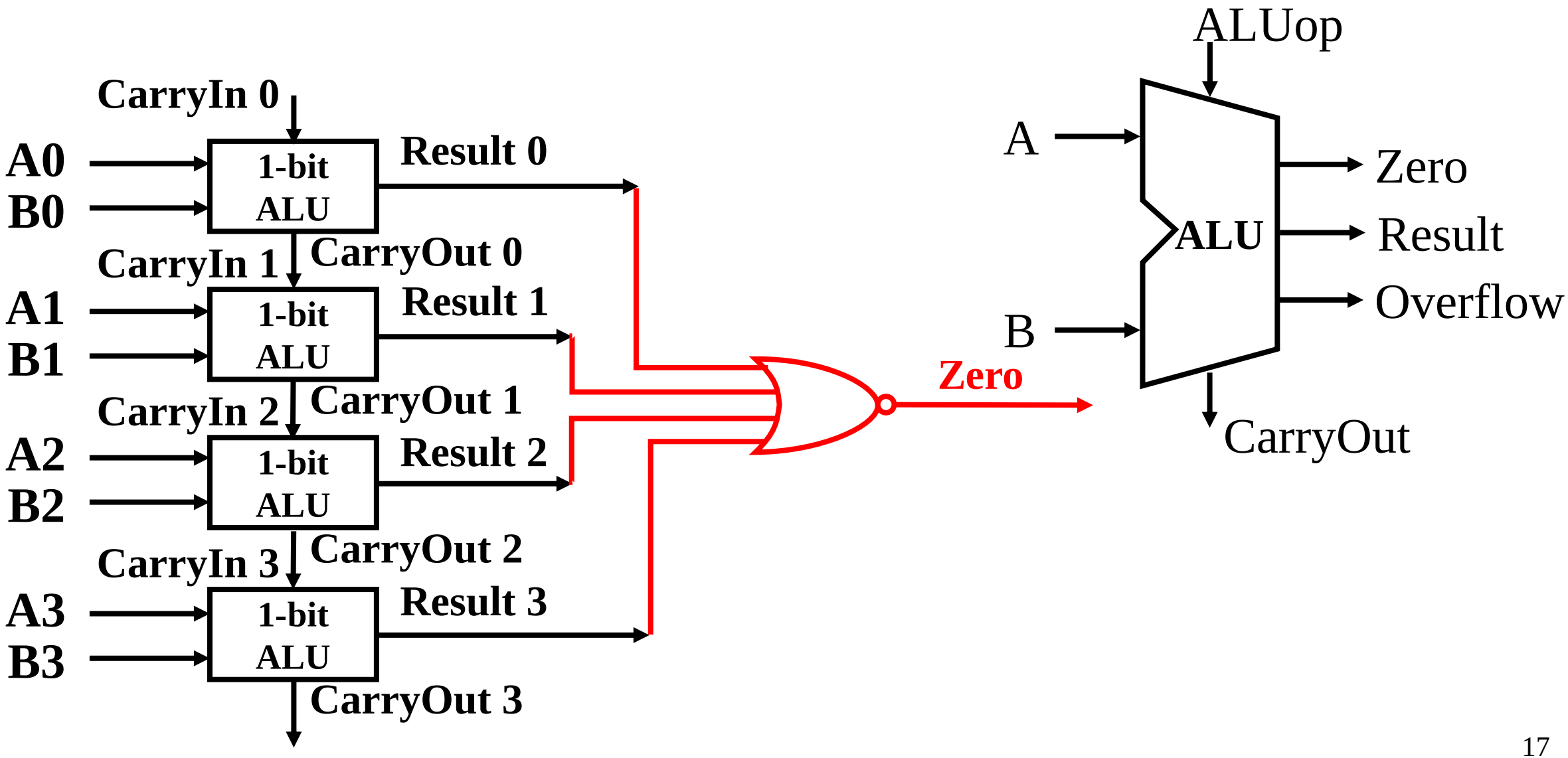
ALU——溢出检测逻辑

- Overflow=CarryIn[N-1] **XOR** CarryOut[N-1]

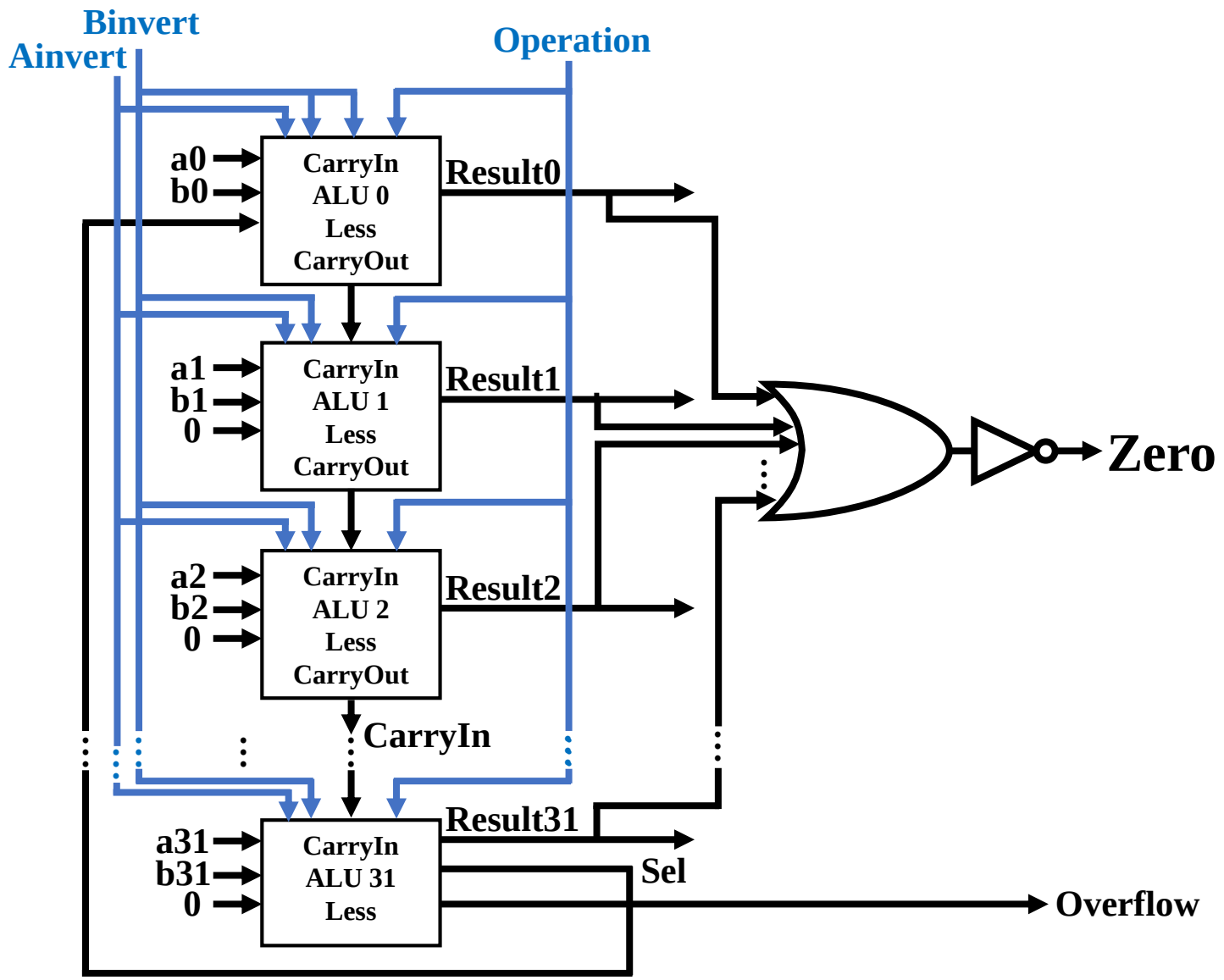


X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0

ALU——判零逻辑


CarryIn 0
A0
B0
1-bit ALU
Result 0
CarryOut 0
CarryIn 1
A1
B1
1-bit ALU
Result 1
CarryOut 1
CarryIn 2
A2
B2
1-bit ALU
Result 2
CarryOut 2
CarryIn 3
A3
B3
1-bit ALU
Result 3
CarryOut 3
Zero
ALUop
A
B
ALU
Zero
Result
Overflow
CarryOut

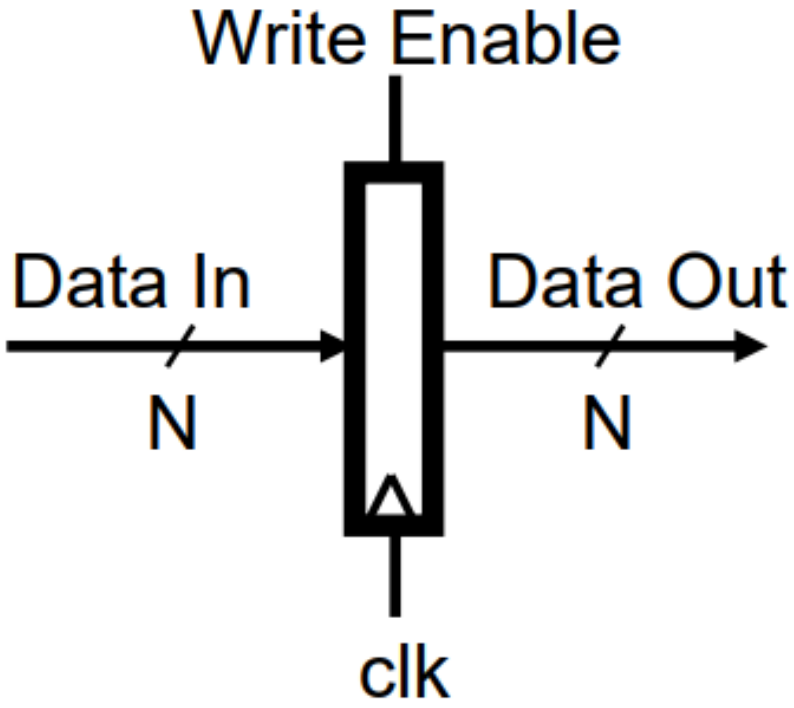
32bit ALU



ALUop	Funciton
0000	and
0001	or
0010	add
0110	subtract
0111	set-on-less-than
1100	nor

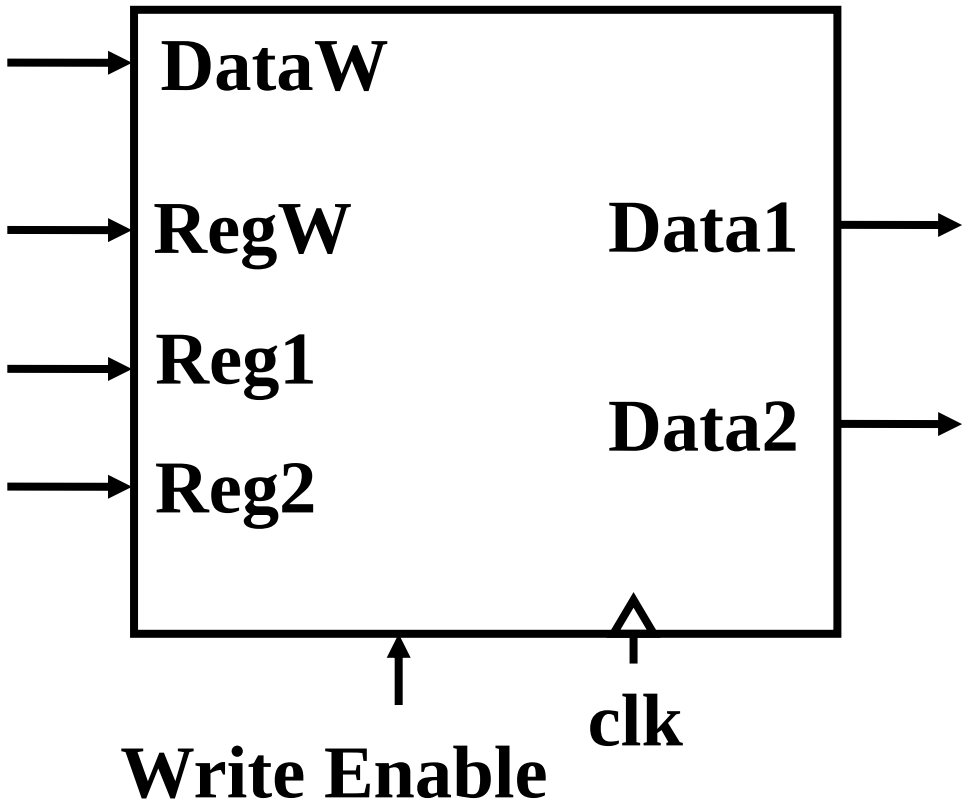
数据通路模块——状态与时序单元

- 寄存器堆和存储器



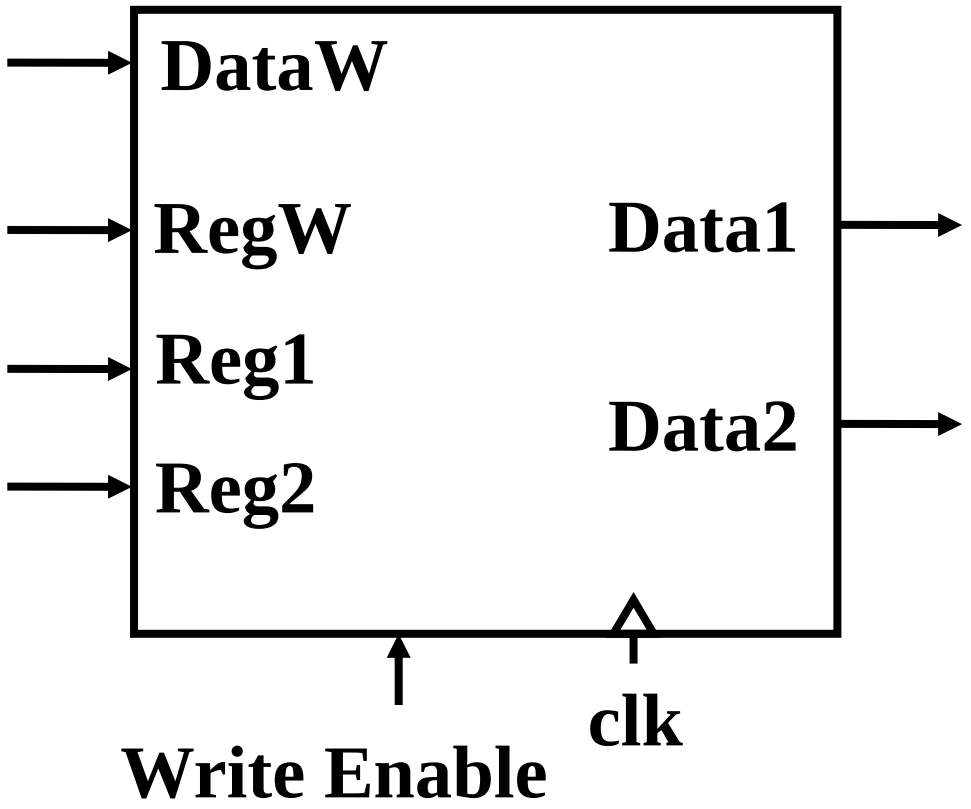
数据通路模块——寄存器堆(reg file)

- 寄存器文件由32个寄存器组成：
  - 两个输出总线：Data1和Data2
  - 一个输入总线：DataW
  - Reg1选择寄存器输出到Data1上
  - Reg2选择寄存器输出到Data2上
  - RegW选择寄存器，当写使能为1时，将DataW上的数据写入



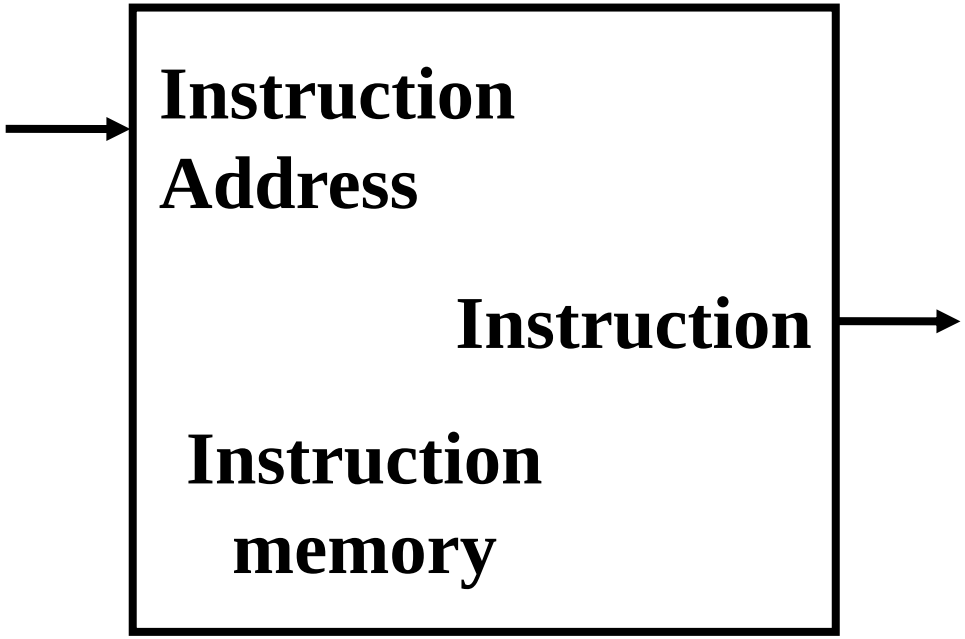
数据通路模块——寄存器堆(reg file)

- Clk输入仅在写寄存器时起作用
- 在读操作时，寄存器堆的行为类似于普通的组合逻辑功能
  - 当Reg1或Reg2有效时，经过一定访问响应时间，得到有效的Data1或Data2输出的数据



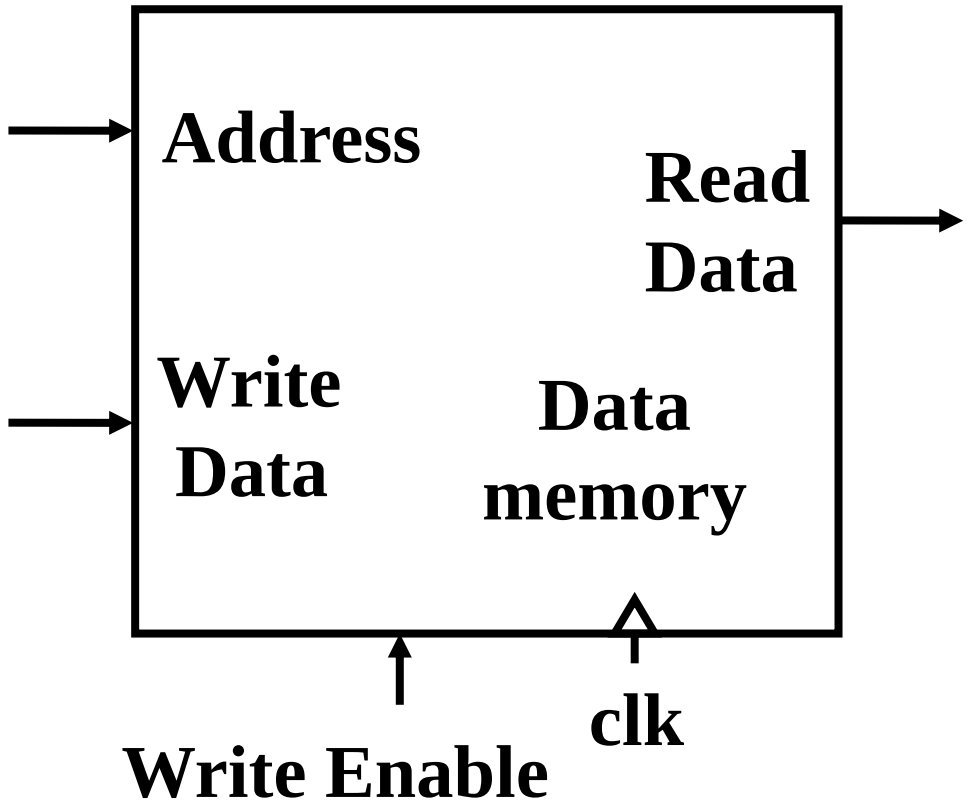
数据通路模块——存储器

- 指令存储器
  - 输入总线： Instruction Address
  - 输出总线： Instruction



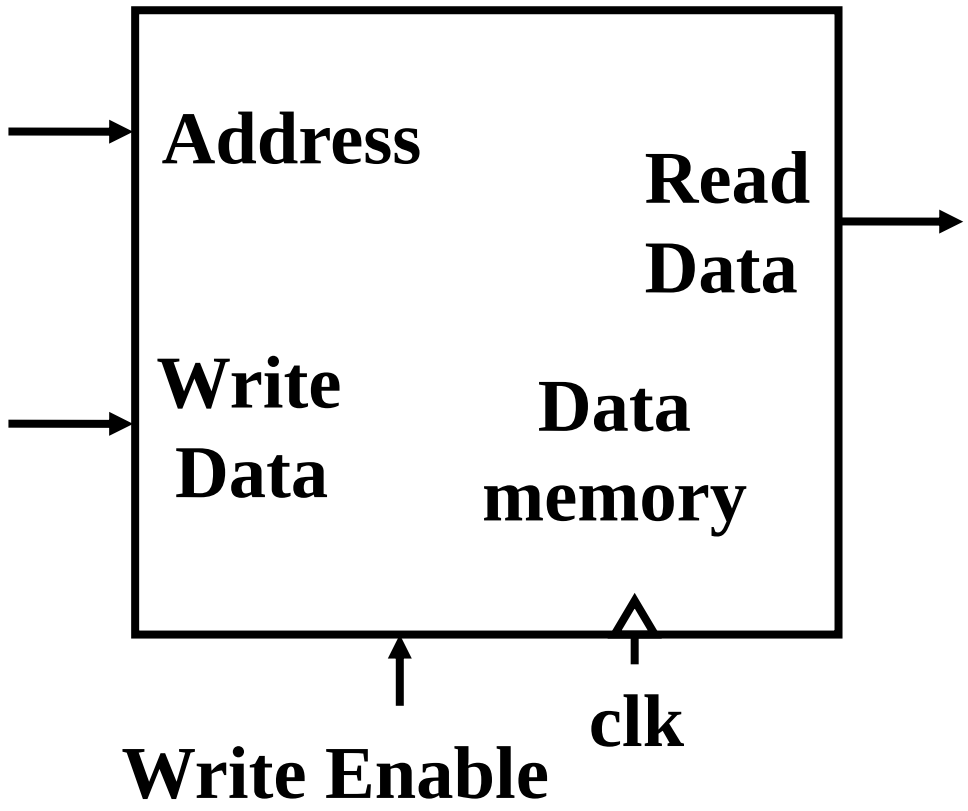
数据通路模块——存储器

- 数据存储器
  - 输入总线：Write Date
  - 输出总线：Read Data
  - 写使能端
  - 地址端
  - 时钟信号端



数据通路模块——存储器之读写

- 读存储器时，由地址端输入的地址，选择存储器中对应地址上的数据，输出到Read Data
- 写存储器时，将写使能置1，表示有效，将Write Data输入的数据写到选择的地址位置上



RISC-V 主要状态单元——寄存器

- 由32个寄存器(0-31)构成的寄存器文件
- 由指令中的rs1字段指定读取的源寄存器1，由rs2字段指定读取的源寄存器2，由rd字段指定写入的目的寄存器
- 另外x0寄存器中恒为0值，对它的写入操作可以忽略
- 程序计数器（PC寄存器），保存着当前执行指令的地址

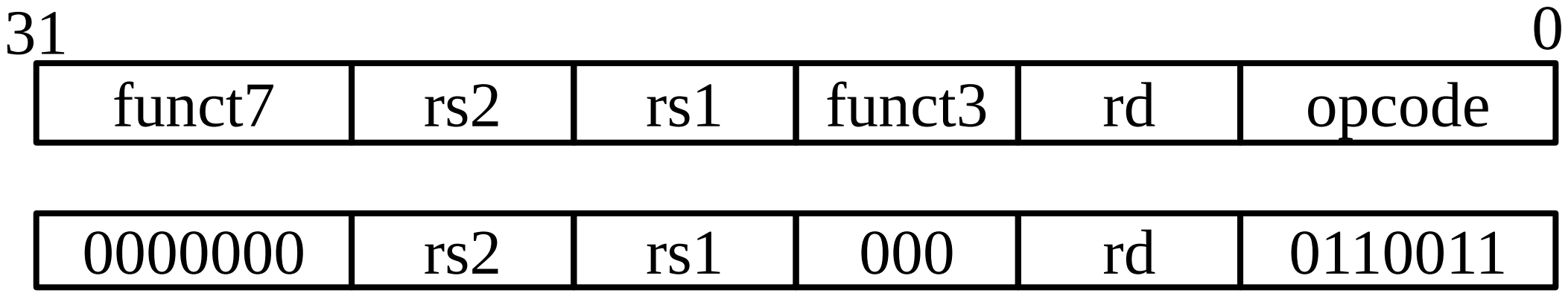
RISC-V 主要状态单元——存储器

- 将指令和数据保存在一个64位的**字节寻址**的存储空间中
- 使用**分立**的两块存储器分别作为指令存储器Imem和数据存储器Dmem
- 从指令存储器**读取**指令，在数据存储器中**读写**数据

第六章 处理器设计

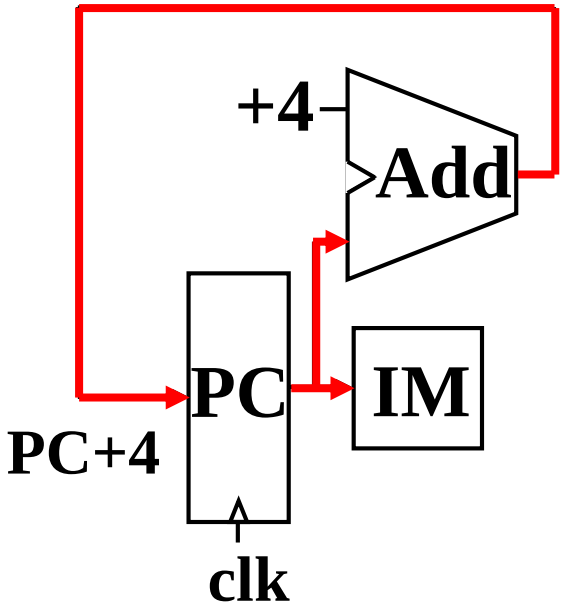
- RISC-V数据通路
  - 数据通路概念
  - RISC-V部分指令的数据通路
- RISC-V控制器

实现add指令



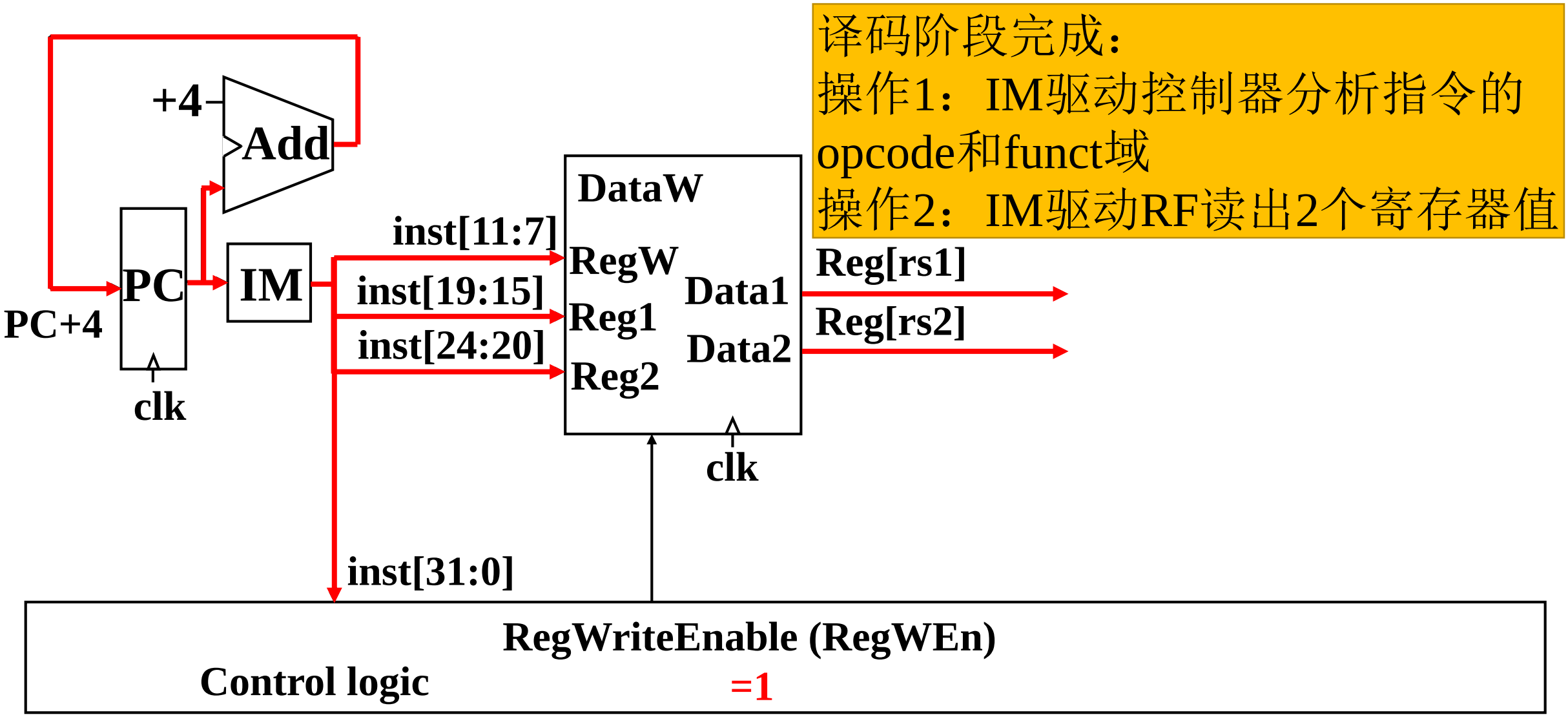
- add rd, rs1, rs2
- 指令对机器状态进行两次更改：
  - Reg[rd] = Reg[rs1] + Reg[rs2]
  - PC = PC +4

add指令的数据通路

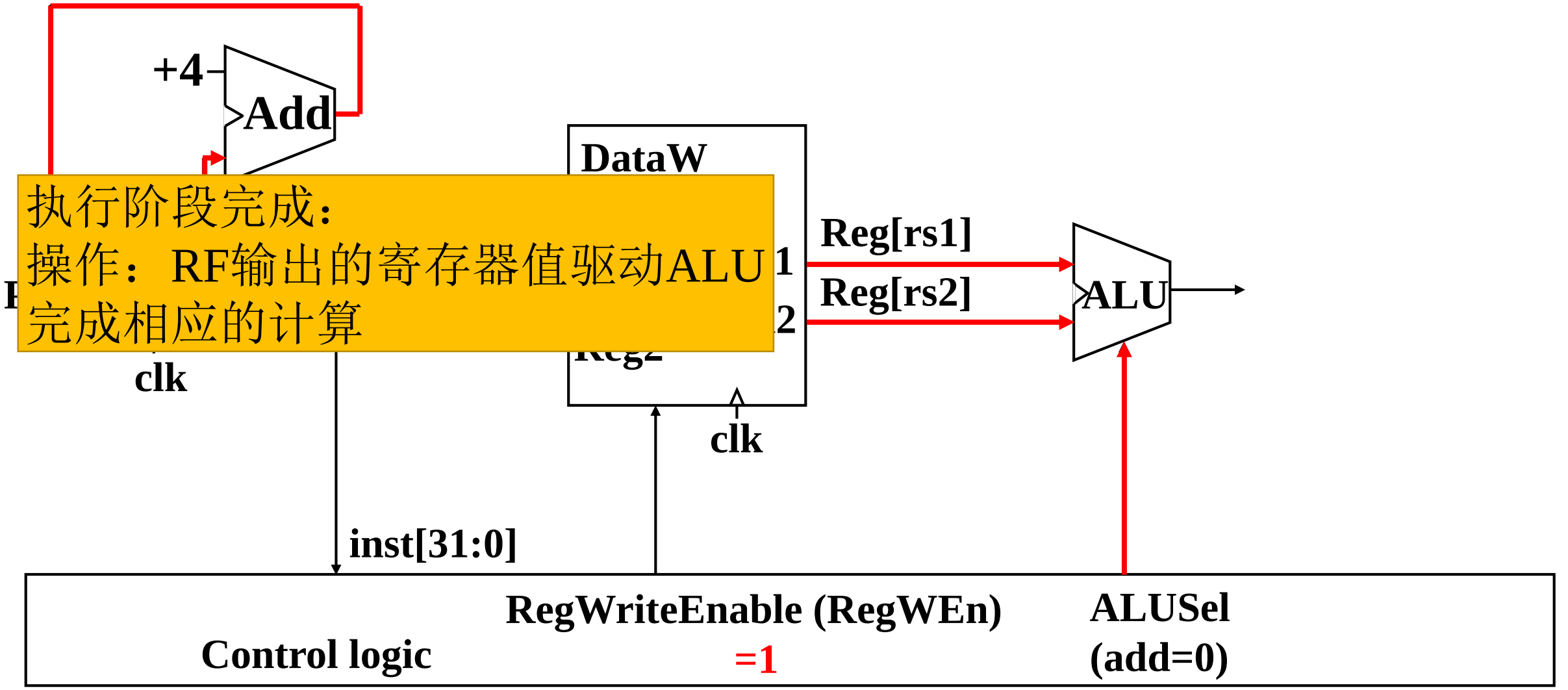


取值阶段完成：
操作1：PC驱动IM输出指令
操作2：PC驱动计算下一个PC值

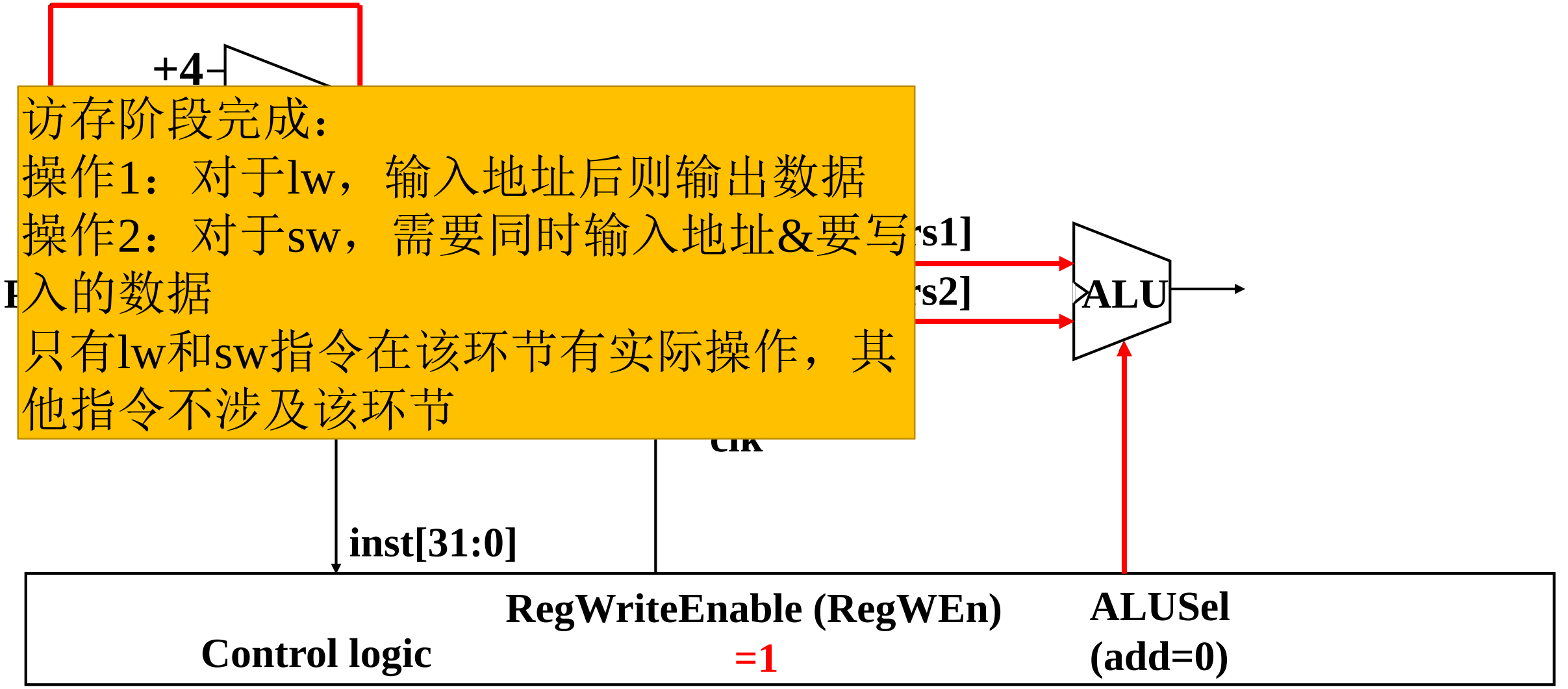
add指令的数据通路


译码阶段完成：
操作1：IM驱动控制器分析指令的opcode和funct域
操作2：IM驱动RF读出2个寄存器值
+4
Add
PC
IM
PC+4
clk
inst[11:7]
inst[19:15]
inst[24:20]
inst[31:0]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
Reg[rs1]
Reg[rs2]
RegWriteEnable (RegWEn)
Control logic
=1

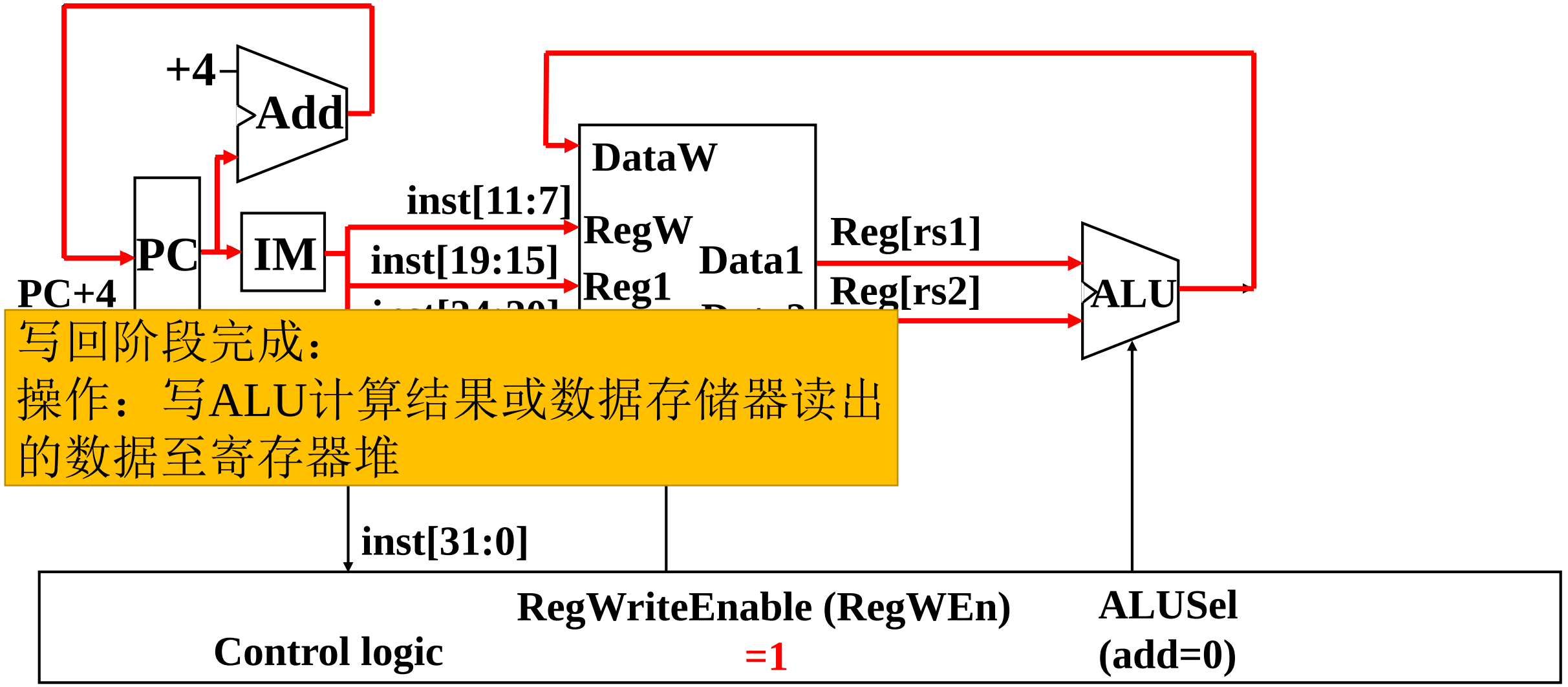
add指令的数据通路


+4
Add
DataW
执行阶段完成：
操作：RF输出的寄存器值驱动ALU
完成相应的计算
Reg[rs1]
Reg[rs2]
1
2
ALU
clk
clk
inst[31:0]
RegWriteEnable (RegWEn)
=1
ALUSel
(add=0)
Control logic

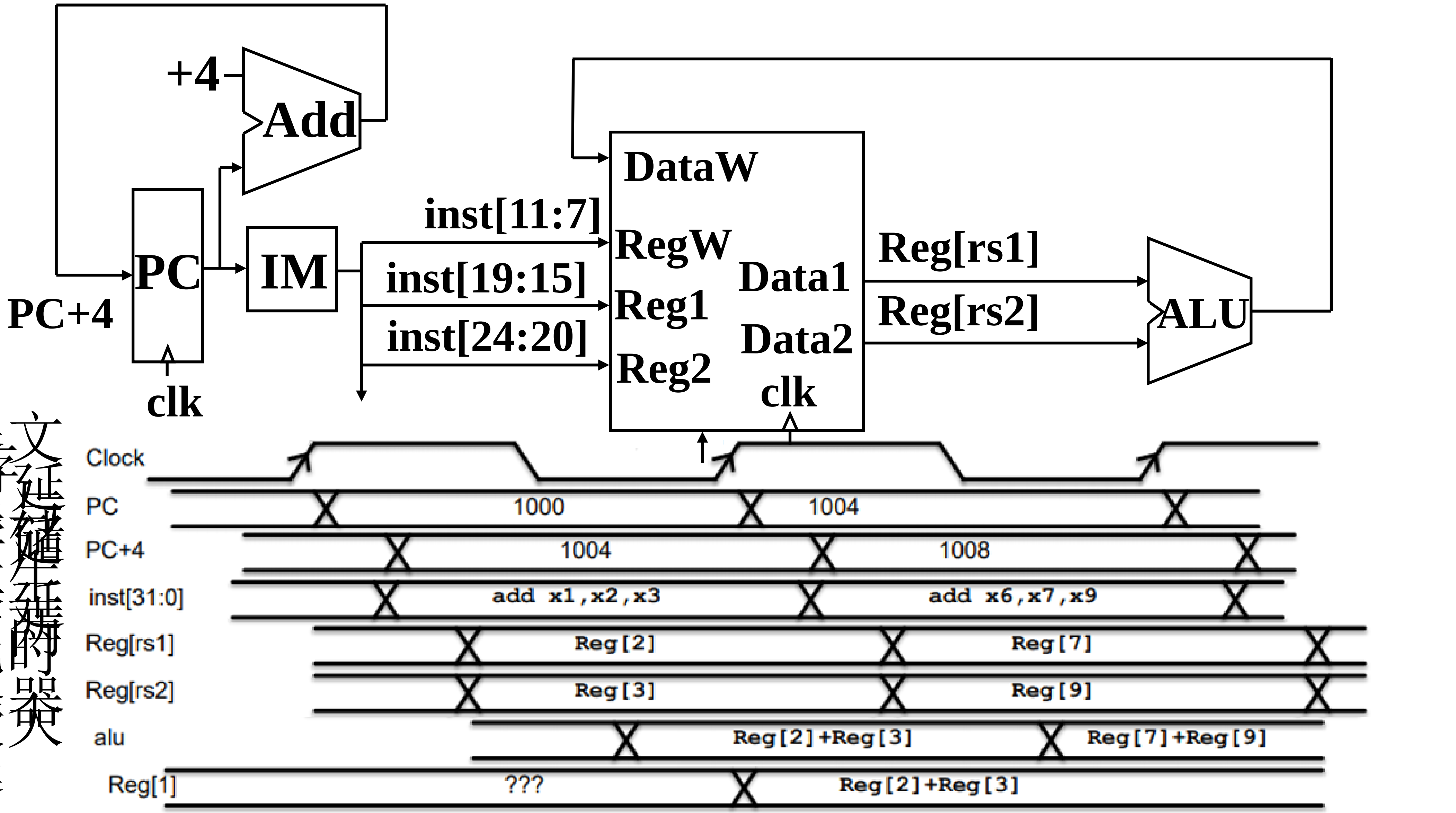
add指令的数据通路


+4
访存阶段完成：
操作1：对于lw，输入地址后则输出数据
操作2：对于sw，需要同时输入地址&要写入的数据
只有lw和sw指令在该环节有实际操作，其他指令不涉及该环节
rs1]
rs2]
ALU
clk
inst[31:0]
RegWriteEnable (RegWEn)
=1
ALUSel
(add=0)
Control logic

add指令的数据通路

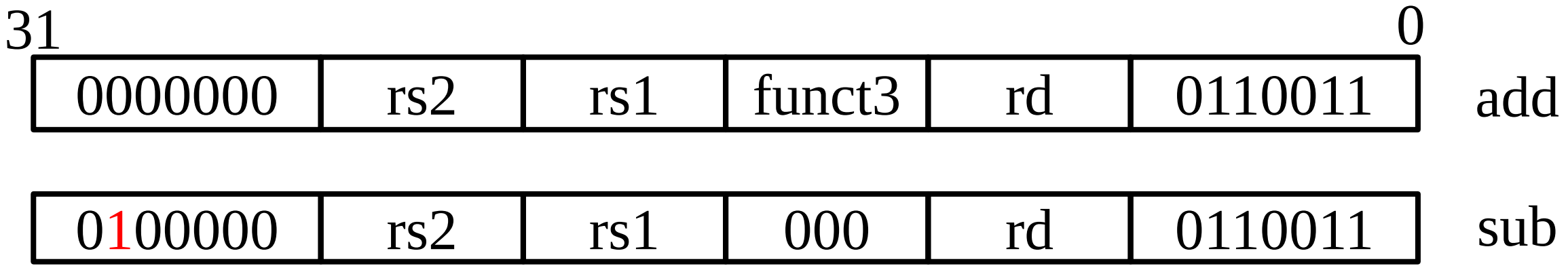

+4
Add
PC
IM
PC+4
inst[11:7]
inst[19:15]
DataW
RegW
Reg1
Data1
Reg[rs1]
Reg[rs2]
ALU
写回阶段完成：
操作：写ALU计算结果或数据存储器读出的数据至寄存器堆
inst[31:0]
Control logic
RegWriteEnable (RegWEn)
=1
ALUSel
(add=0)

add指令的时序图



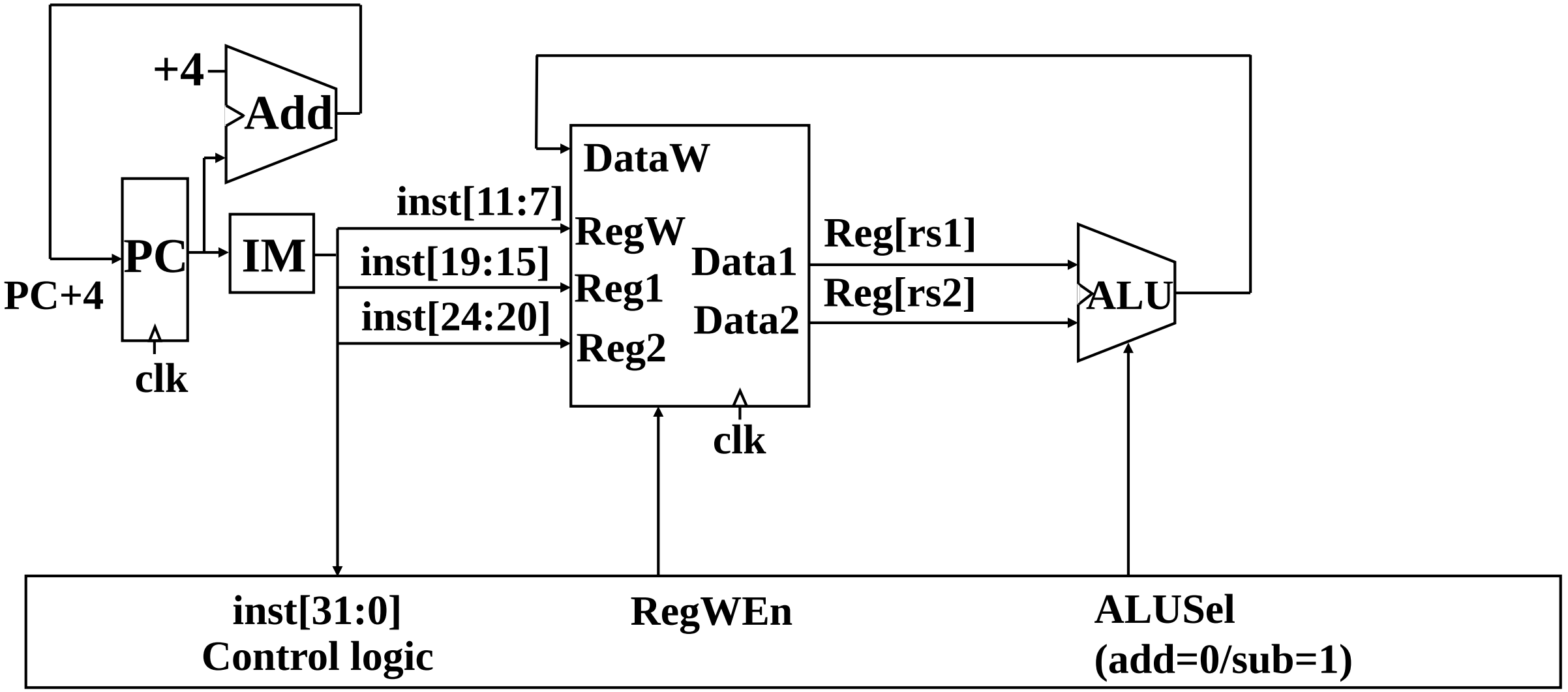
+4
Add
PC
IM
PC+4
clk
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
Reg[rs1]
Reg[rs2]
ALU
Clock
PC
1000
1004
PC+4
1004
1008
inst[31:0]
add x1,x2,x3
add x6,x7,x9
Reg[rs1]
Reg[2]
Reg[7]
Reg[rs2]
Reg[3]
Reg[9]
alu
Reg[2]+Reg[3]
Reg[7]+Reg[9]
Reg[1]
???
Reg[2]+Reg[3]

实现sub指令



- sub rd, rs1, rs2
- 几乎与加法相同
- inst[30]在加法和减法之间进行选择

sub指令的数据通路


+4
Add
PC
IM
PC+4
clk
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
Reg[rs1]
Reg[rs2]
ALU
inst[31:0]
Control logic
RegWEn
ALUSel
(add=0/sub=1)

实现R型指令

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

funct3和funct7字段不同，选择ALU的不同功能

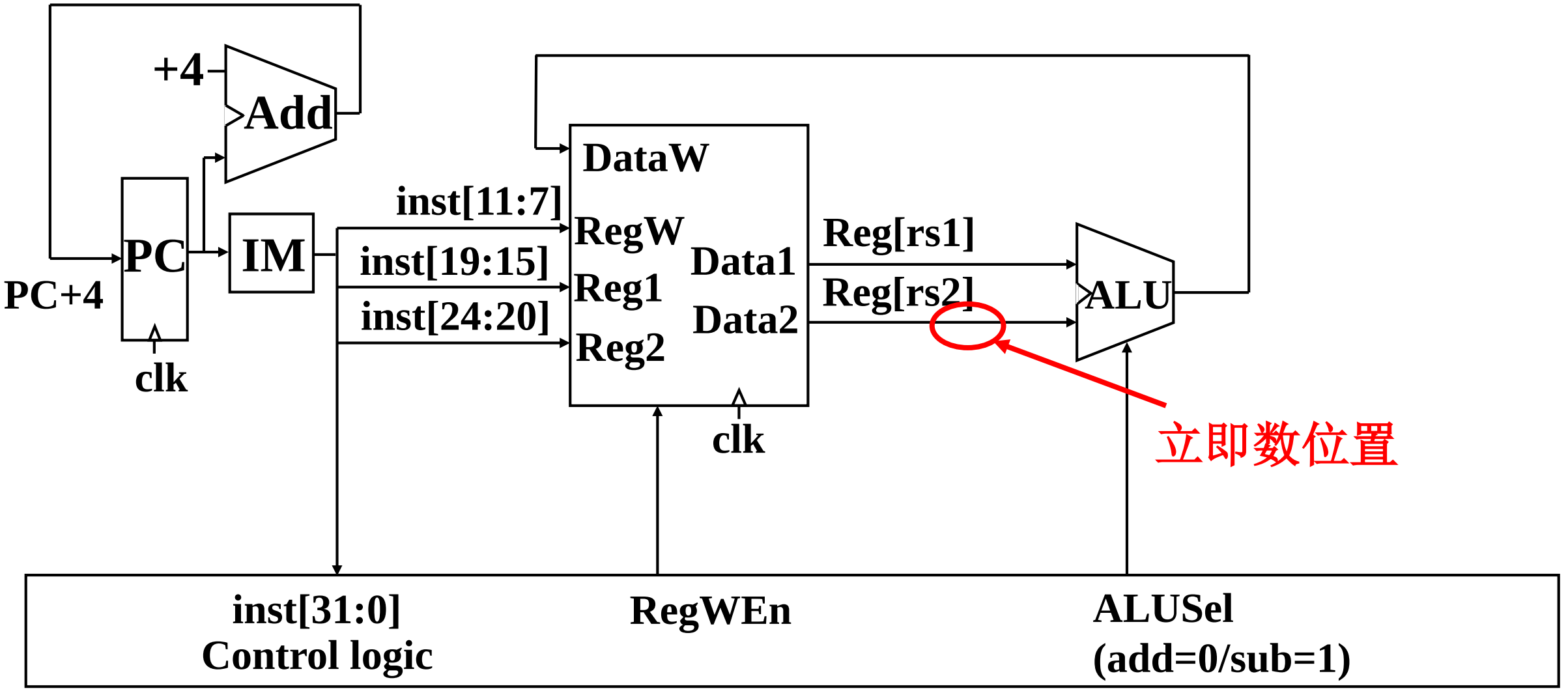
实现RISC-V addi指令

- addi x15, x1, -50

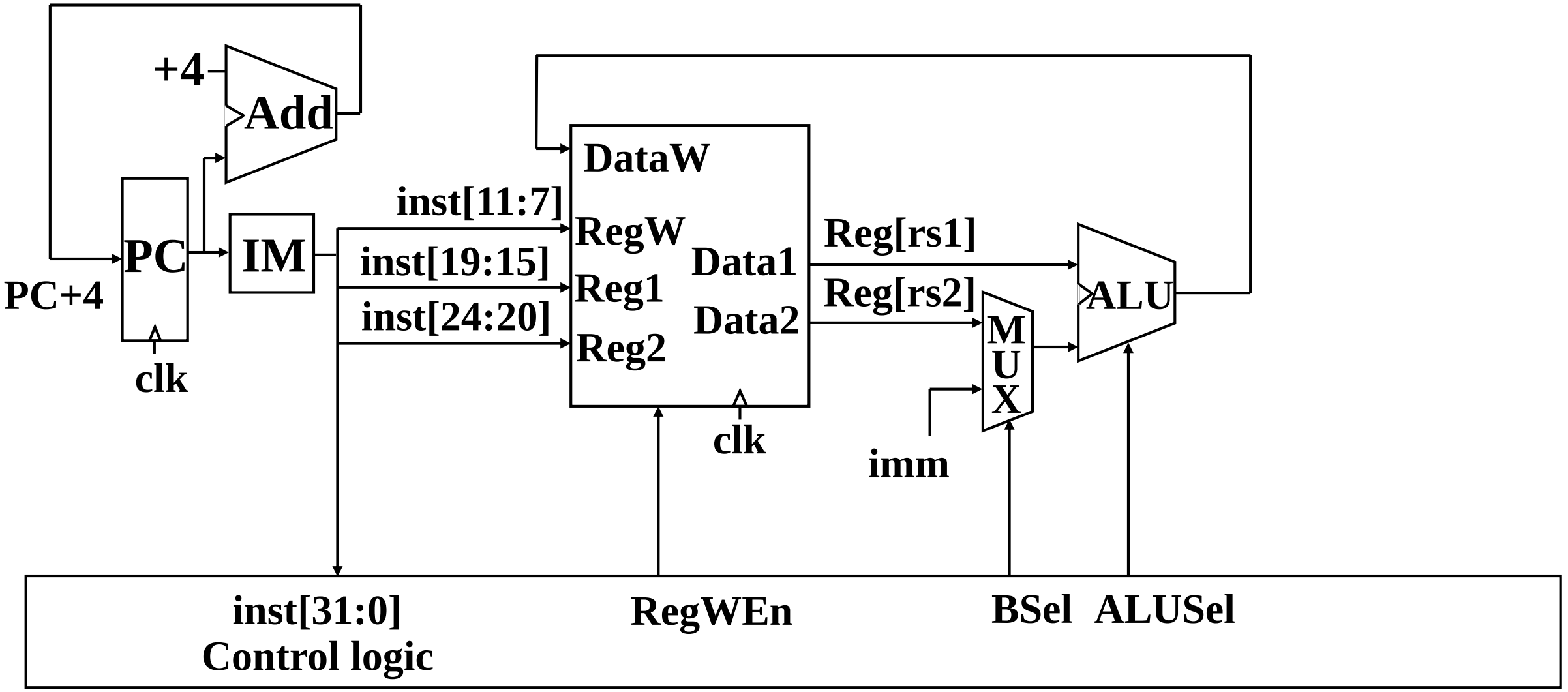
31 0

imm[11:0]	rs1	funct3	rd	opcode
111111001110	00001	000	01111	0010011
imm= -50	rs1=1	add	rd=15	OP-Imm

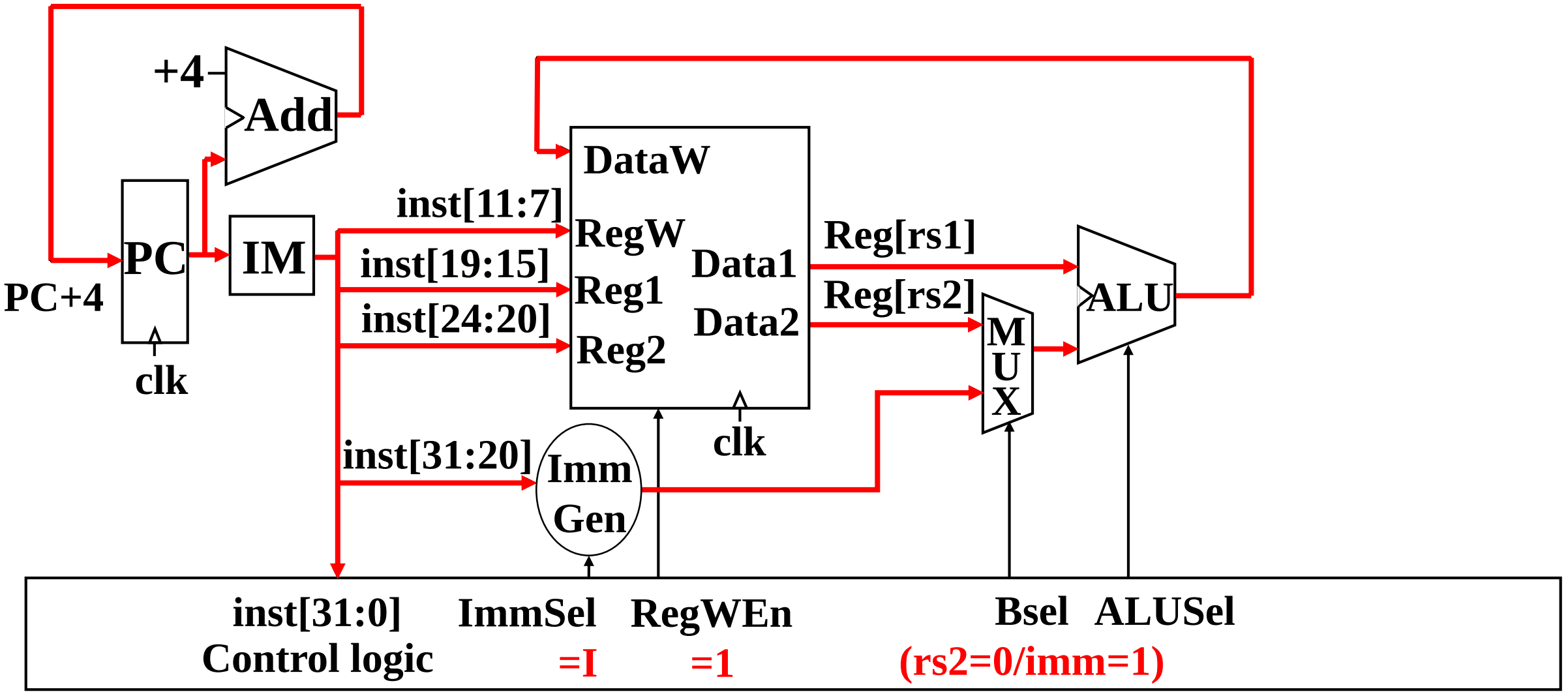
add指令的数据通路


+4
Add
PC
IM
PC+4
clk
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
Reg[rs1]
Reg[rs2]
ALU
立即数位置
inst[31:0]
Control logic
RegWEn
ALUSel
(add=0/sub=1)

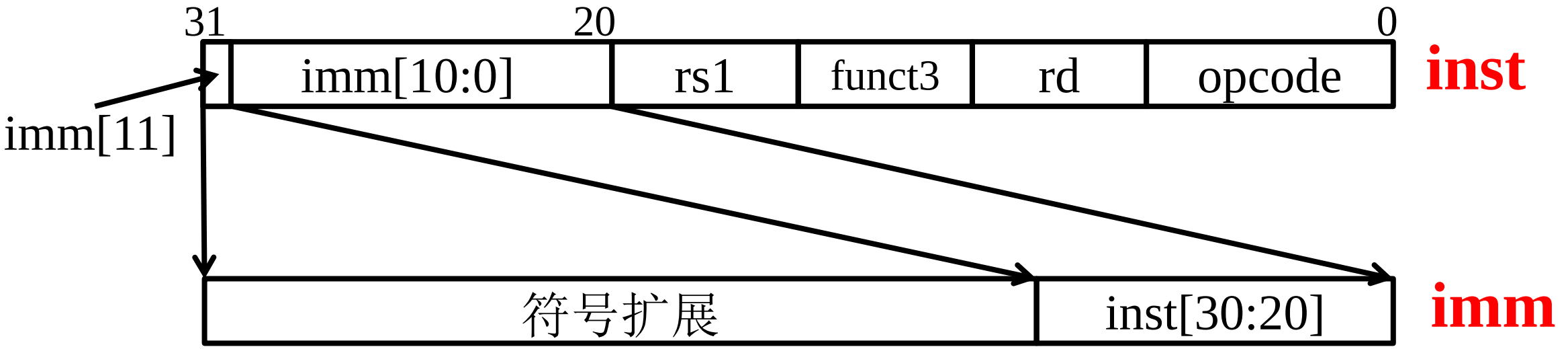
addi指令的数据通路



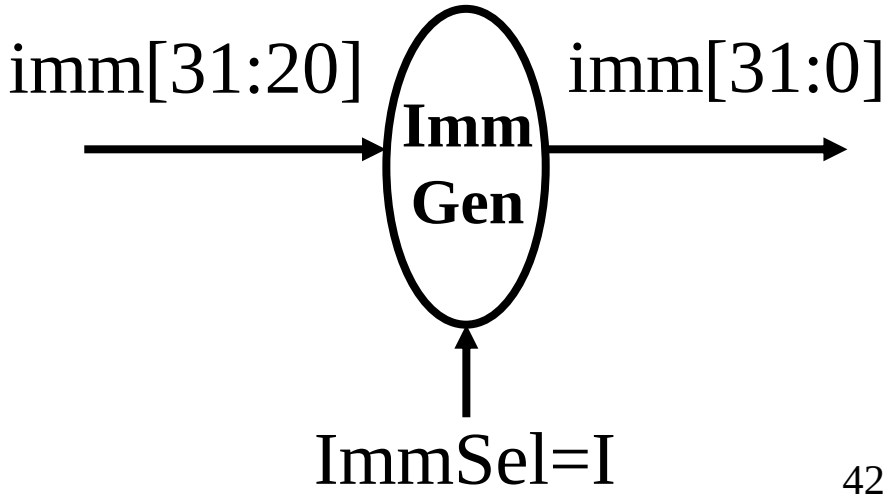
addi指令的数据通路



I型指令的立即数生成

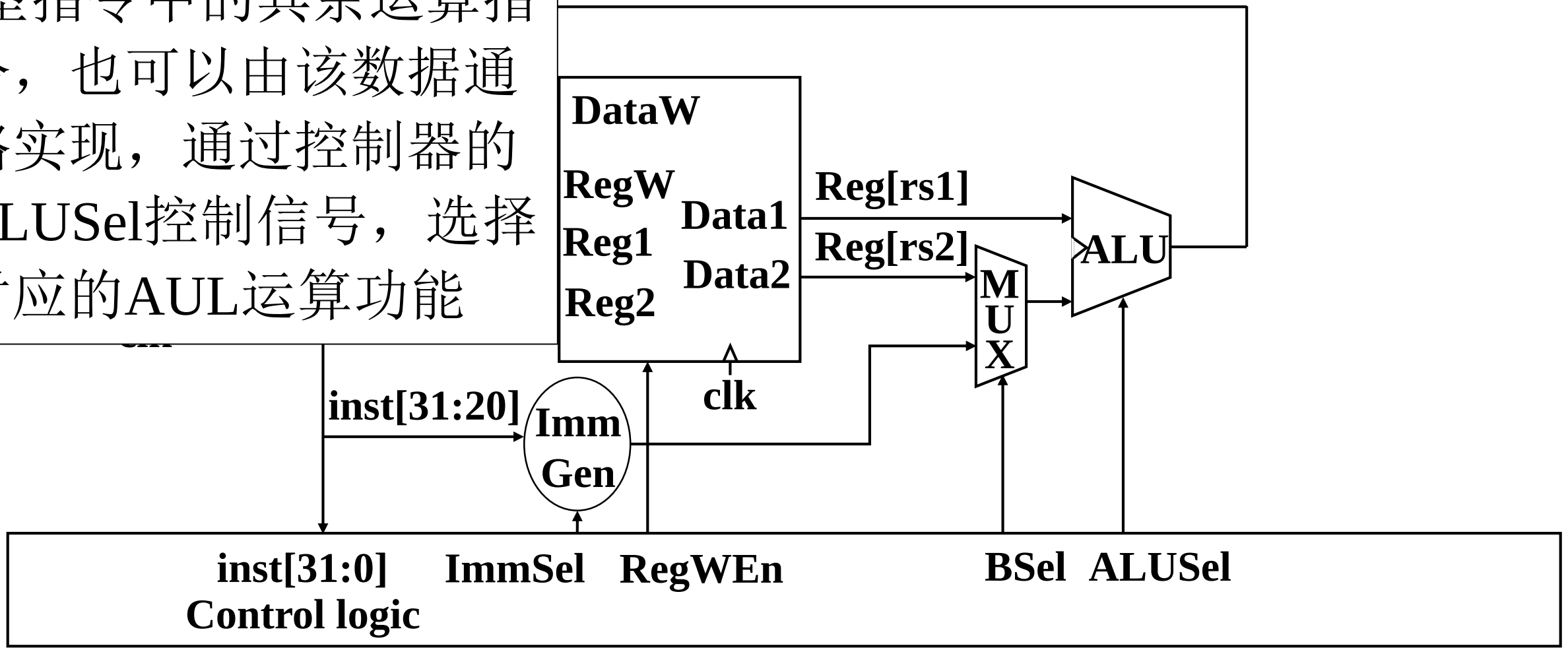


- 高12位复制到立即数的低12位
- 通过将指令的最高比特位复制填充到立即数的高20位完成符号扩展

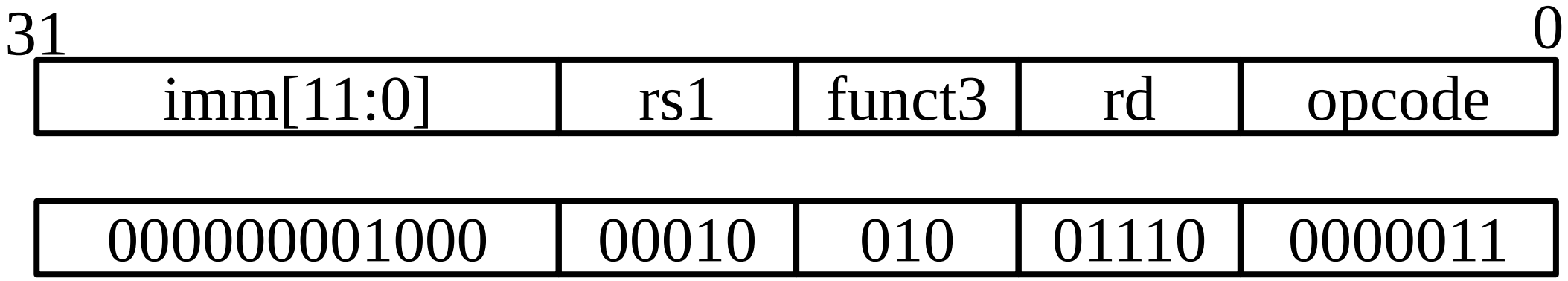


R+I 数据通路

I型指令中的其余运算指令，也可以由该数据通路实现，通过控制器的ALUSel控制信号，选择对应的AUL运算功能

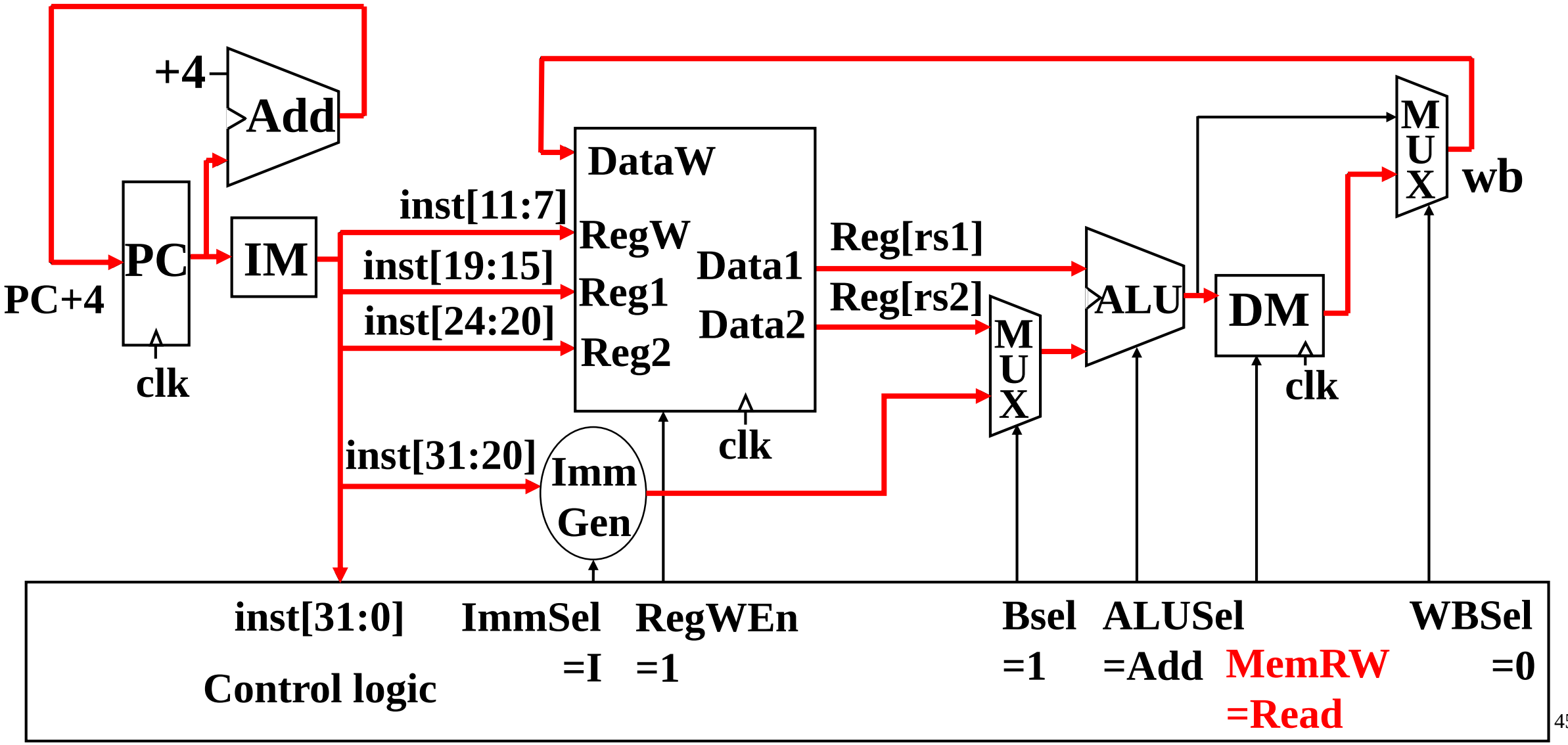


实现lw指令（load word）



- lw x14, 8(x2)
- 寄存器rs1中的值作为基地址，加上立即数值，得到目标访问地址
  - 与addi操作十分相似，但用于计算地址，而不是获得最终结果
- 从存储器读出的数据装入寄存器rd中

lw指令的数据通路


+4
Add
PC
IM
PC+4
clk
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
Reg[rs1]
Reg[rs2]
inst[31:20]
Imm
Gen
M
U
X
ALU
DM
clk
M
U
X
wb
inst[31:0]
Control logic
ImmSel
=I
RegWEn
=1
Bsel
=1
ALUSel
=Add
MemRW
=Read
WBSel
=0

RISC-V 访存装载指令

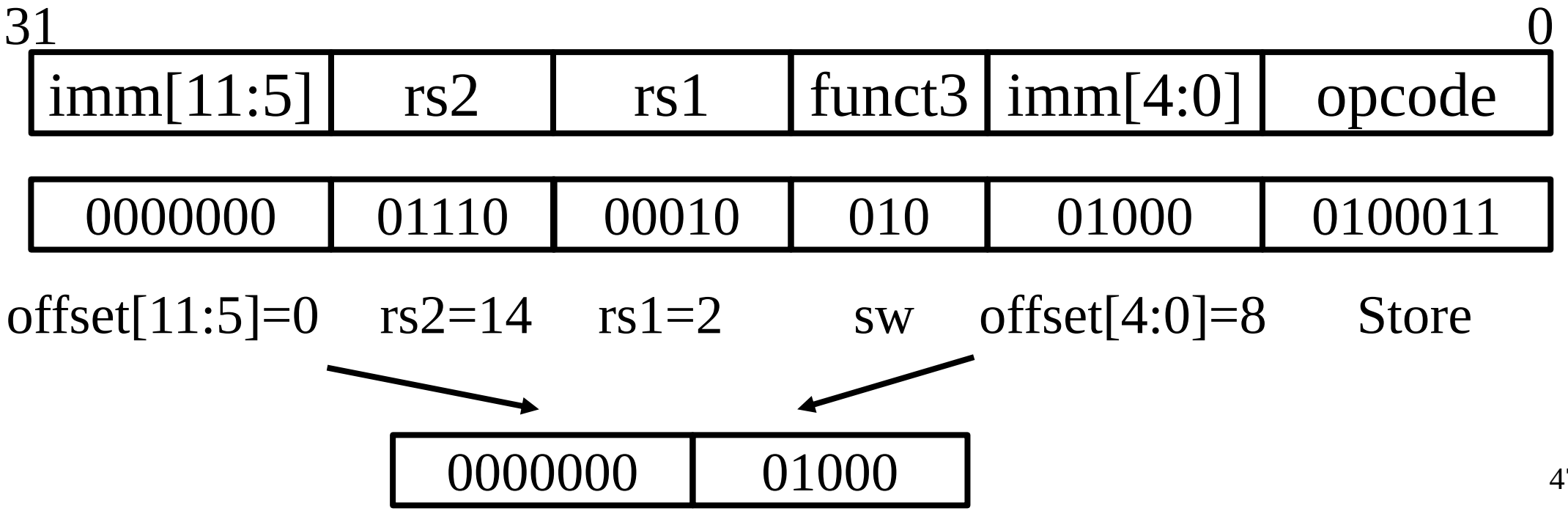
imm[11:0]	rs1	000	rd	0000011	lb
imm[11:0]	rs1	010	rd	0000011	lh
imm[11:0]	rs1	011	rd	0000011	lw
imm[11:0]	rs1	100	rd	0000011	lbu
imm[11:0]	rs1	110	rd	0000011	lhu

- 为了支持8位和16位的访存装载指令
  - 增加额外的逻辑电路，用于从存储器取出的字数据中提取出半字或者字节数据
  - 写回到寄存器前，进行符号扩展或零扩展

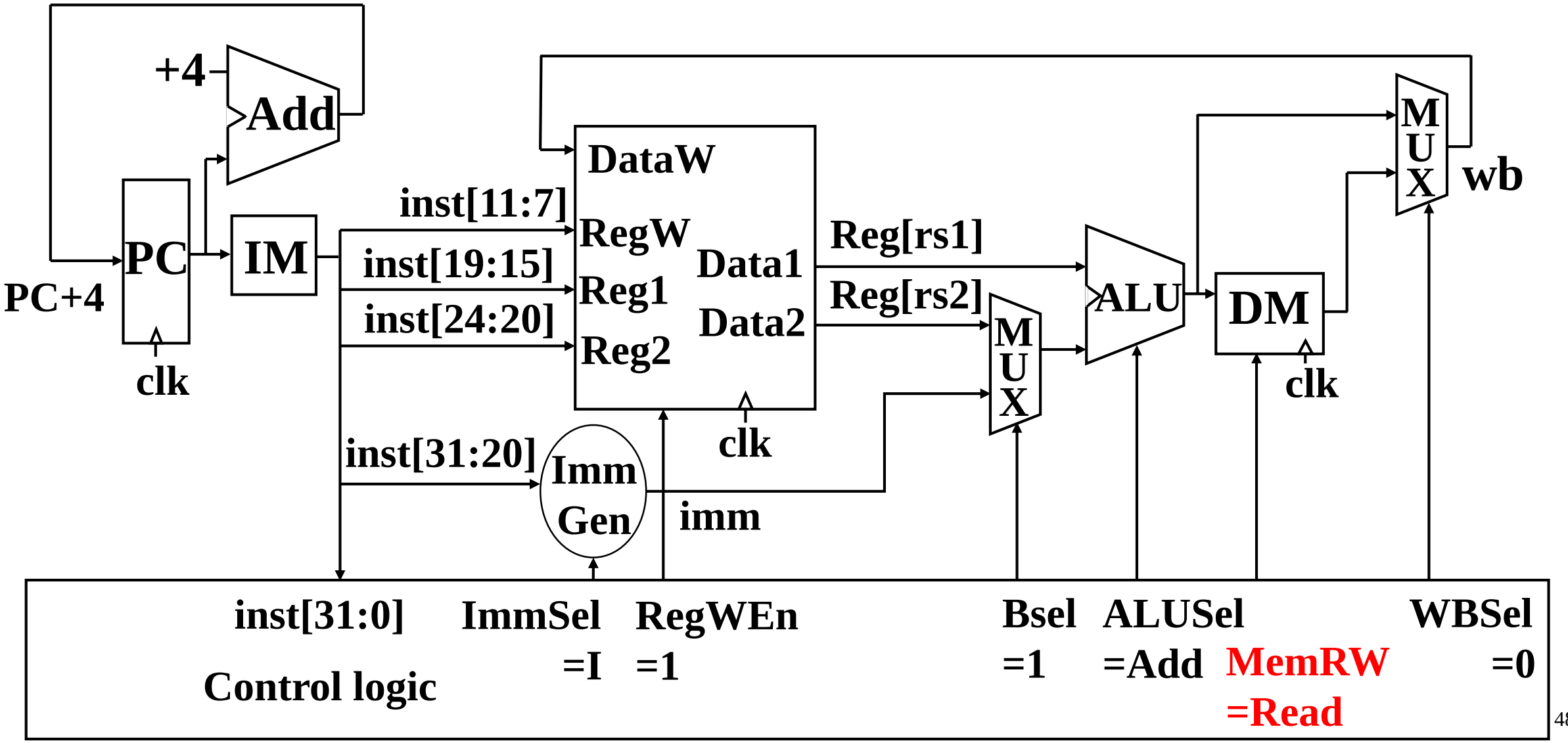
实现sw指令

- 读取两个寄存器，rs1作为提供基地址的源寄存器，rs2作为提供带保存数据的源寄存器，以及立即数偏移量

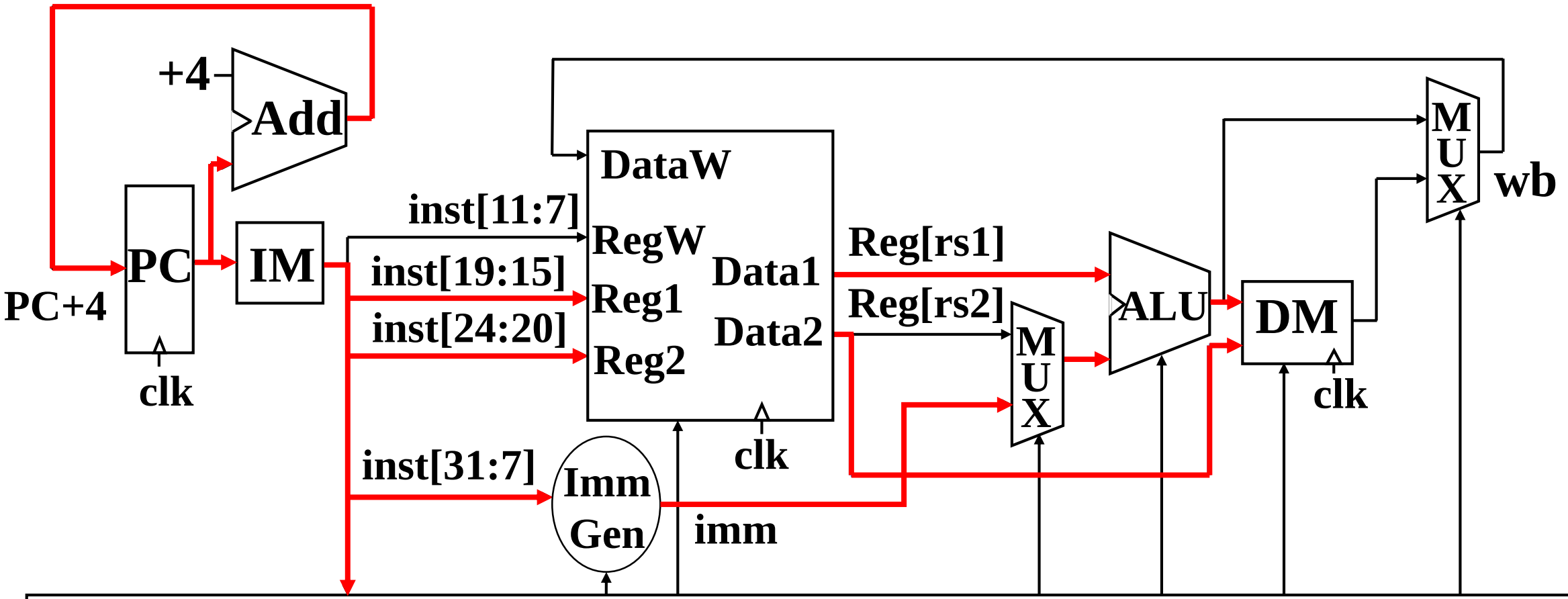
sw x14, 8(x2)



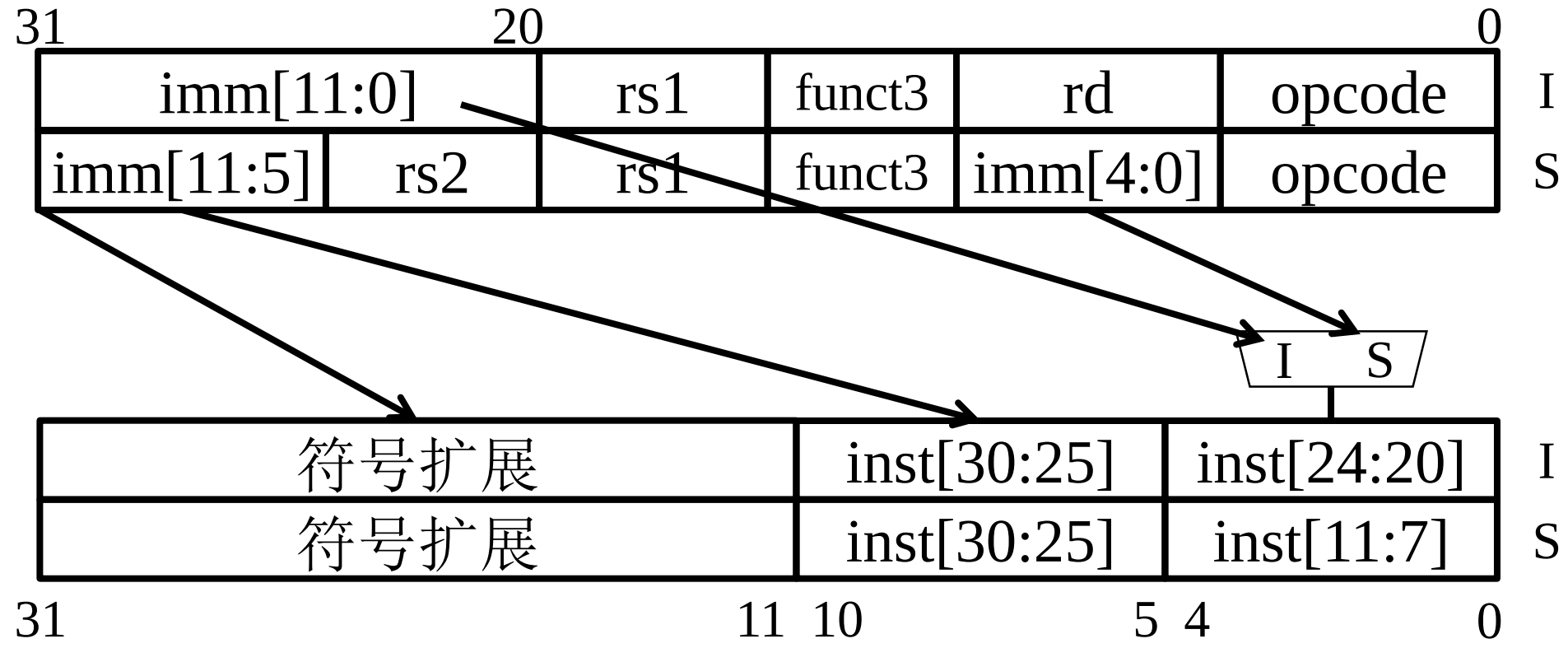
lw指令的数据通路


+4
Add
PC
IM
PC+4
clk
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
Reg[rs1]
Reg[rs2]
M
U
X
ALU
DM
clk
M
U
X
wb
inst[31:20]
Imm
Gen
imm
inst[31:0]
Control logic
ImmSel
=I
RegWEn
=1
Bsel
=1
ALUSel
=Add
MemRW
=Read
WBSel
=0

sw指令的数据通路

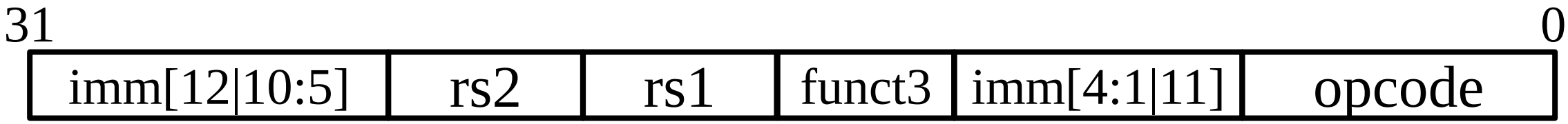

+4
Add
PC
IM
PC+4
clk
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
Reg[rs1]
Reg[rs2]
M
U
X
ALU
DM
clk
M
U
X
wb
inst[31:7]
Imm
Gen
imm
inst[31:0]
Control logic
ImmSel
=S
RegWEn
=0
Bsel
=1
ALUSel
=Add
MemRW
=Write
WBSel
=*

I+S 立即数生成



- 立即数低5位由I型或S型对应的控制信号选择
- 立即数中的其他位连接到指令中的固定位置

实现B型指令

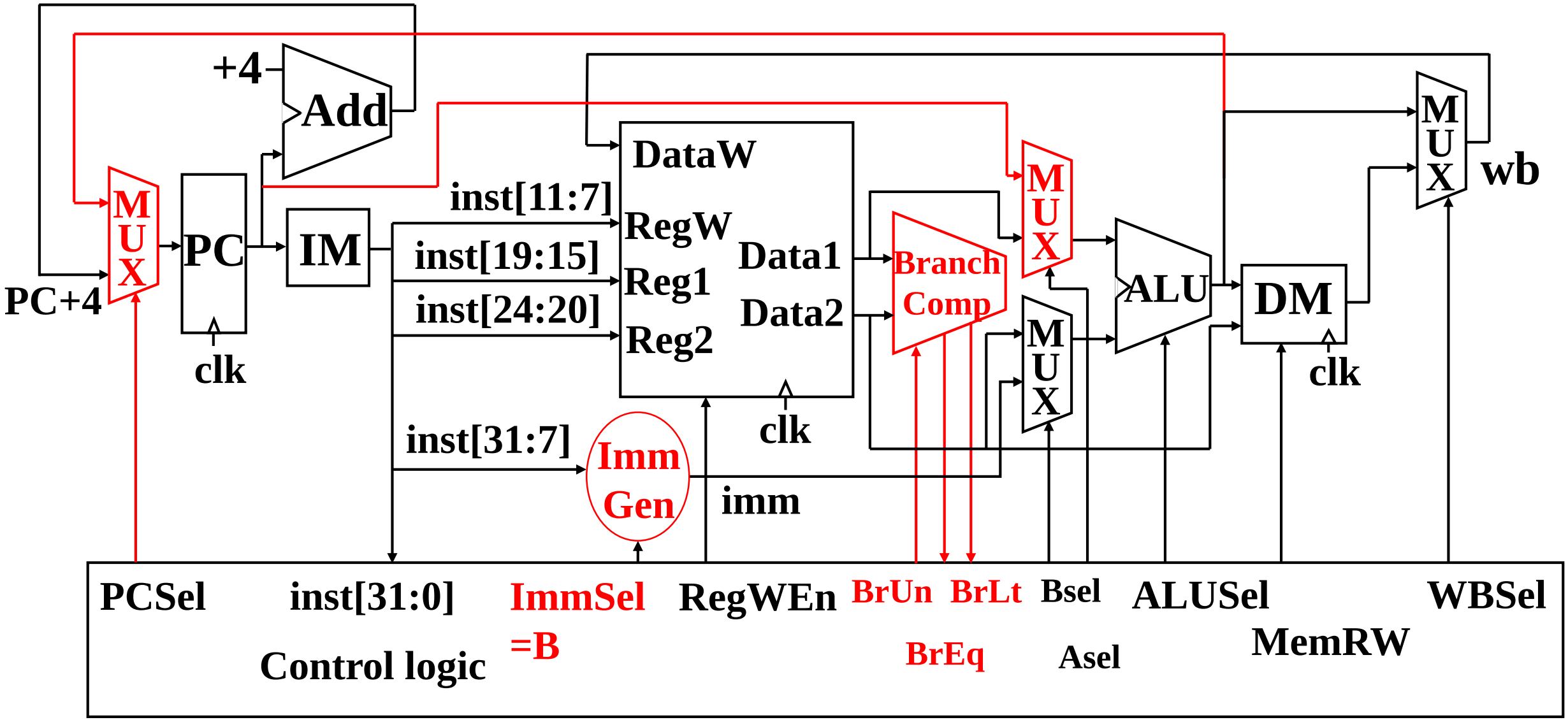


- B型指令格式与S型指令格式基本相同
- 但立即数字段以2字节为增量表示-4096到+4094的偏移量
- 12位立即数字段表示13位有符号字节地址的偏移量
  - 偏移量的最低位始终为零，因此无需存储

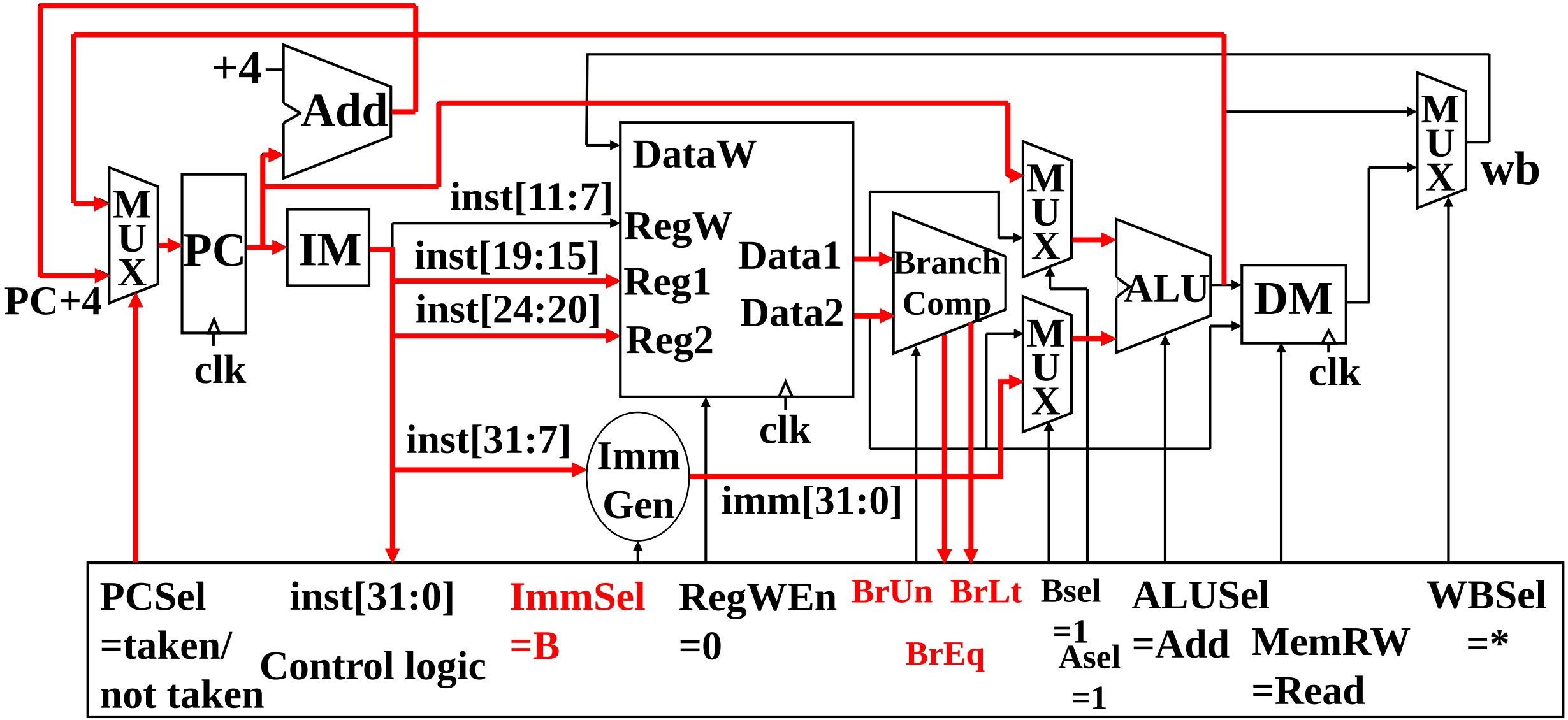
B型指令的数据通路

- 六个指令：beq、bne、blt、bge、bltu、bgeu
- PC不同的状态变化：
  - PC+ 4　不发生分支转移
  - PC + immediate　　发生分支转移
- 需要比较rsl和rs2的数值关系，并计算PC+立即数的结果
- 只有一个ALU
- 需要更多硬件

B型指令的数据通路


+4
Add
MUX
PC+4
PC
clk
IM
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
inst[31:7]
Imm Gen
imm
Branch Comp
MUX
MUX
MUX
ALU
DM
clk
wb
PCSel
inst[31:0]
ImmSel
=B
RegWEn
BrUn
BrLt
BrEq
Bsel
Asel
ALUSel
MemRW
WBSel
Control logic

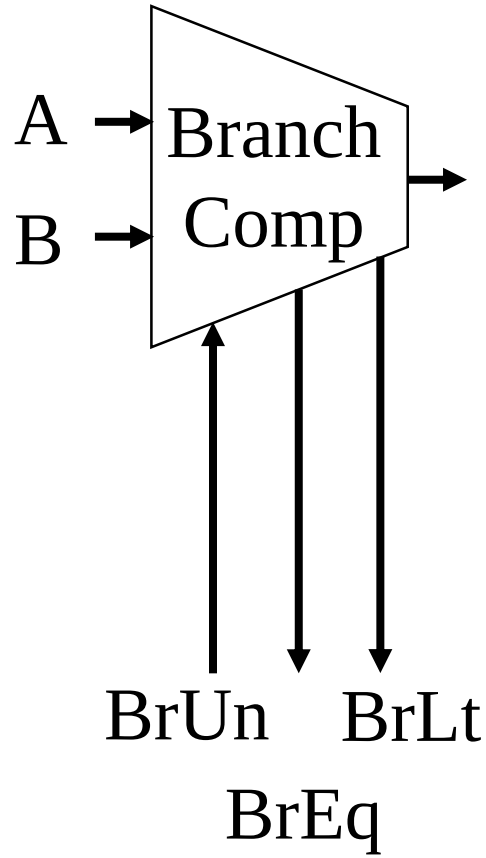
B型指令的数据通路


+4
Add
M
U
X
PC+4
PC
clk
IM
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
inst[31:7]
Imm
Gen
imm[31:0]
Branch
Comp
ALU
DM
clk
wb
PCSel
=taken/
not taken
inst[31:0]
Control logic
ImmSel
=B
RegWEn
=0
BrUn
BrLt
BrEq
Bsel
=1
Asel
=1
ALUSel
=Add
MemRW
=Read
WBSel
=*

分支跳转比较器

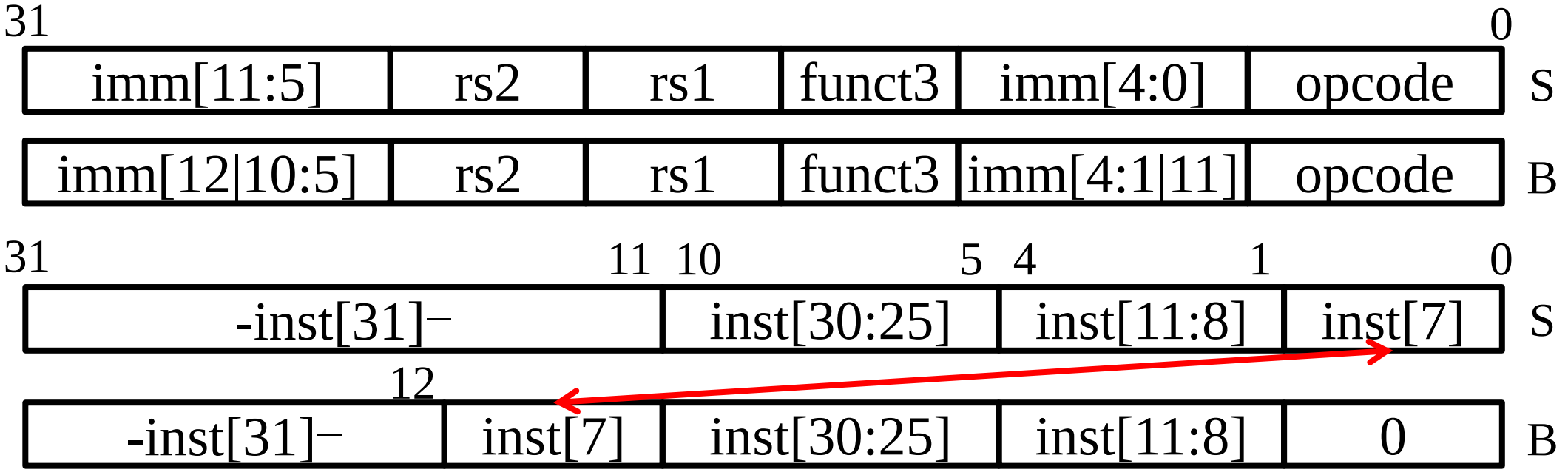
- 当A = B时，输出 BrEq = 1，否则为0
- 当A < B时，输出 BrLt = 1，否则为0
- 输入 BrUn = 1时，选择无符号比较结果

  输入 BrUn = 0时，选择有符号比较结果
- 对于bge，可以根据BrLt信号取反判断A ≥ B

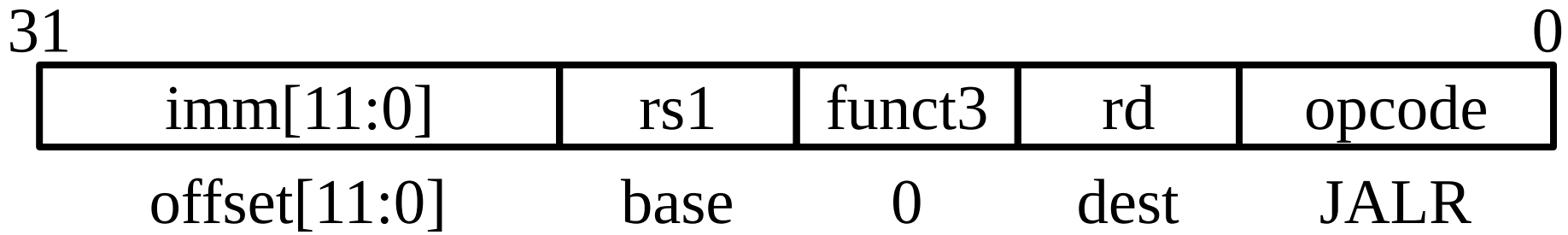


B型指令立即数生成

- 与S型指令立即数编码方式十分相似，除了S型立即数最低比特位在B型立即数中变为了第12比特位
- 只有一位数据在编码位置上有差异
  - 只需要两个单比特两路选择器

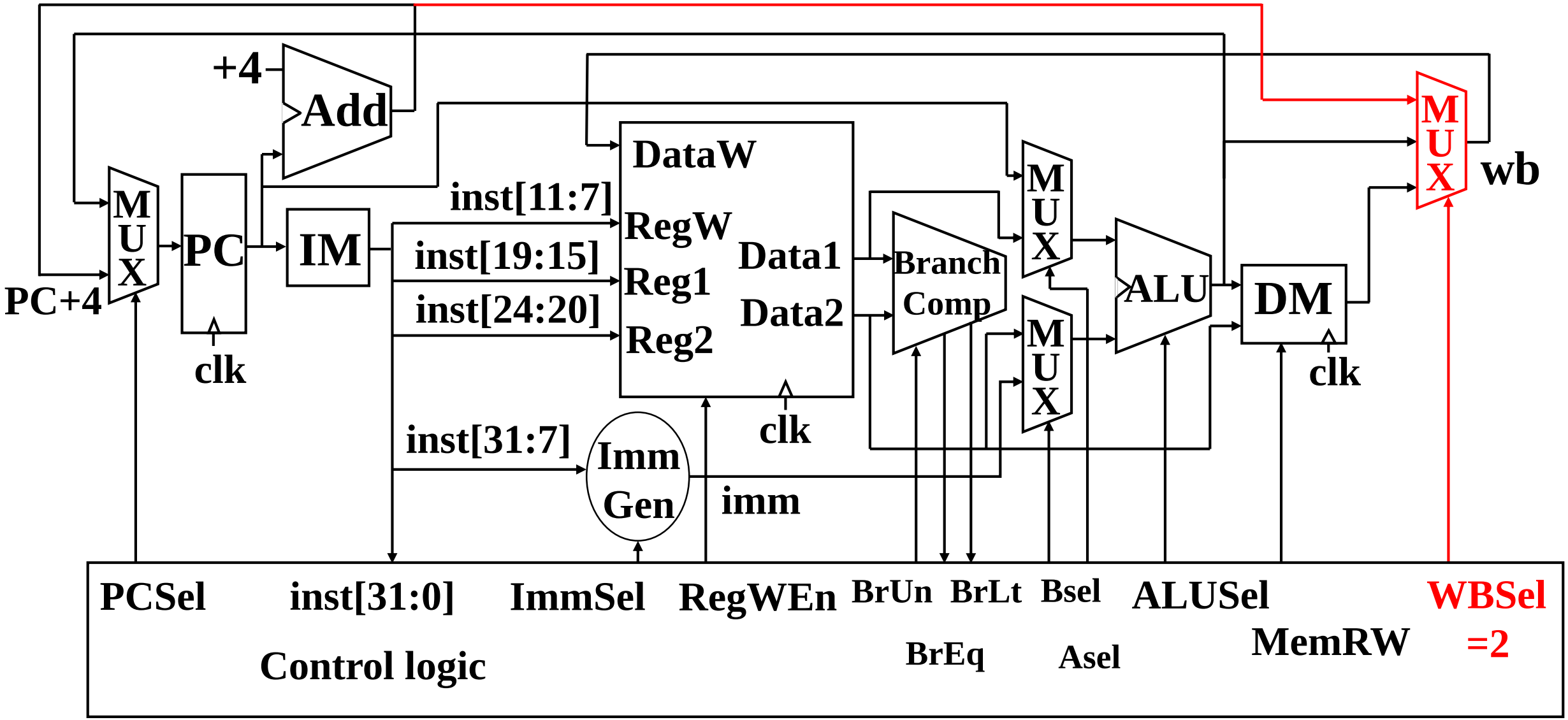


实现jalr指令

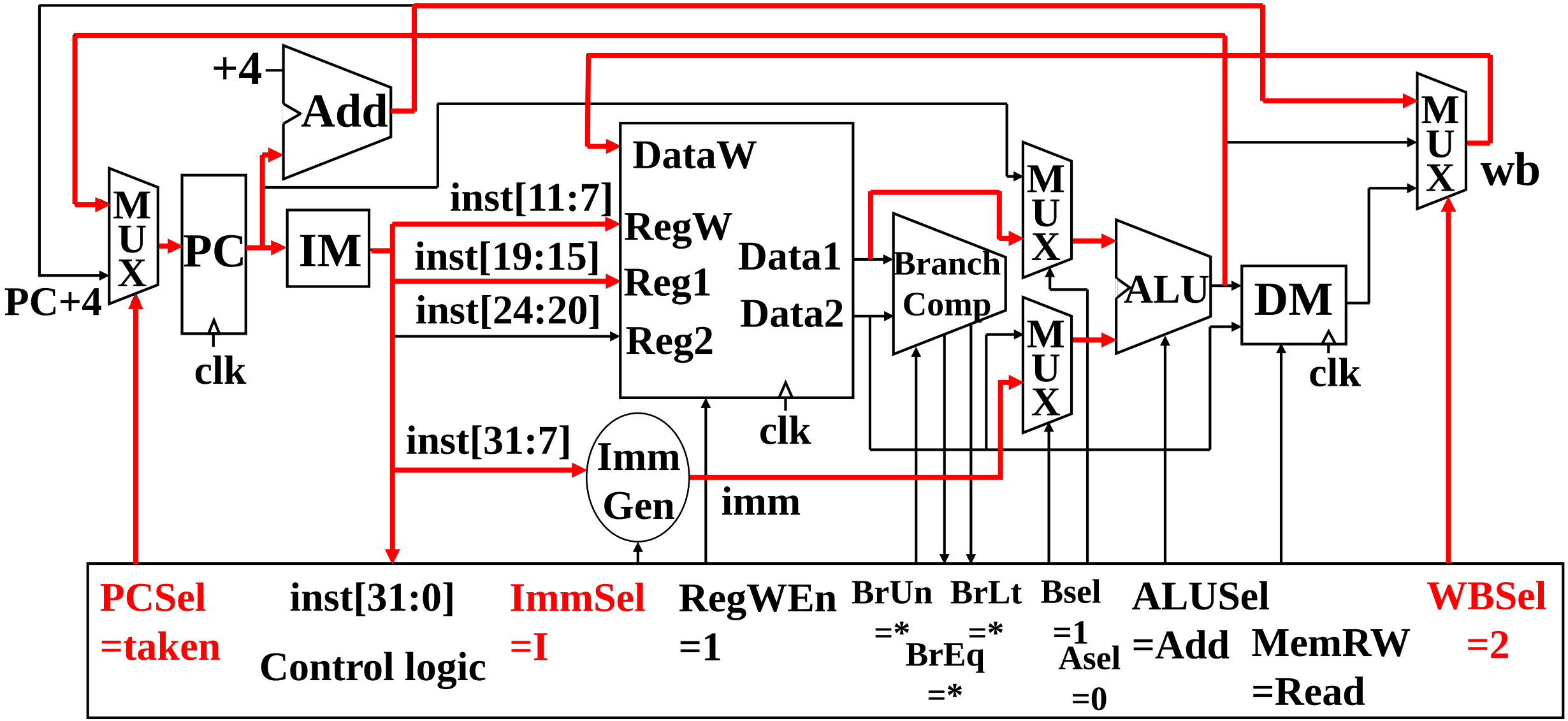


- jalr rd, rs, immediate
- 两个状态变化
  - 将 PC + 4 写入rd（返回地址）
  - 设置 PC = rs + immediate
  - 立即数用法与I型算术和装载指令一样

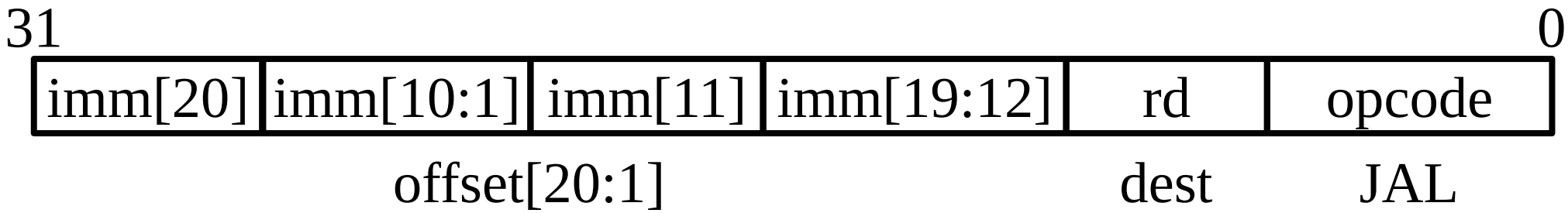
jalr指令的数据通路



jalr指令的数据通路

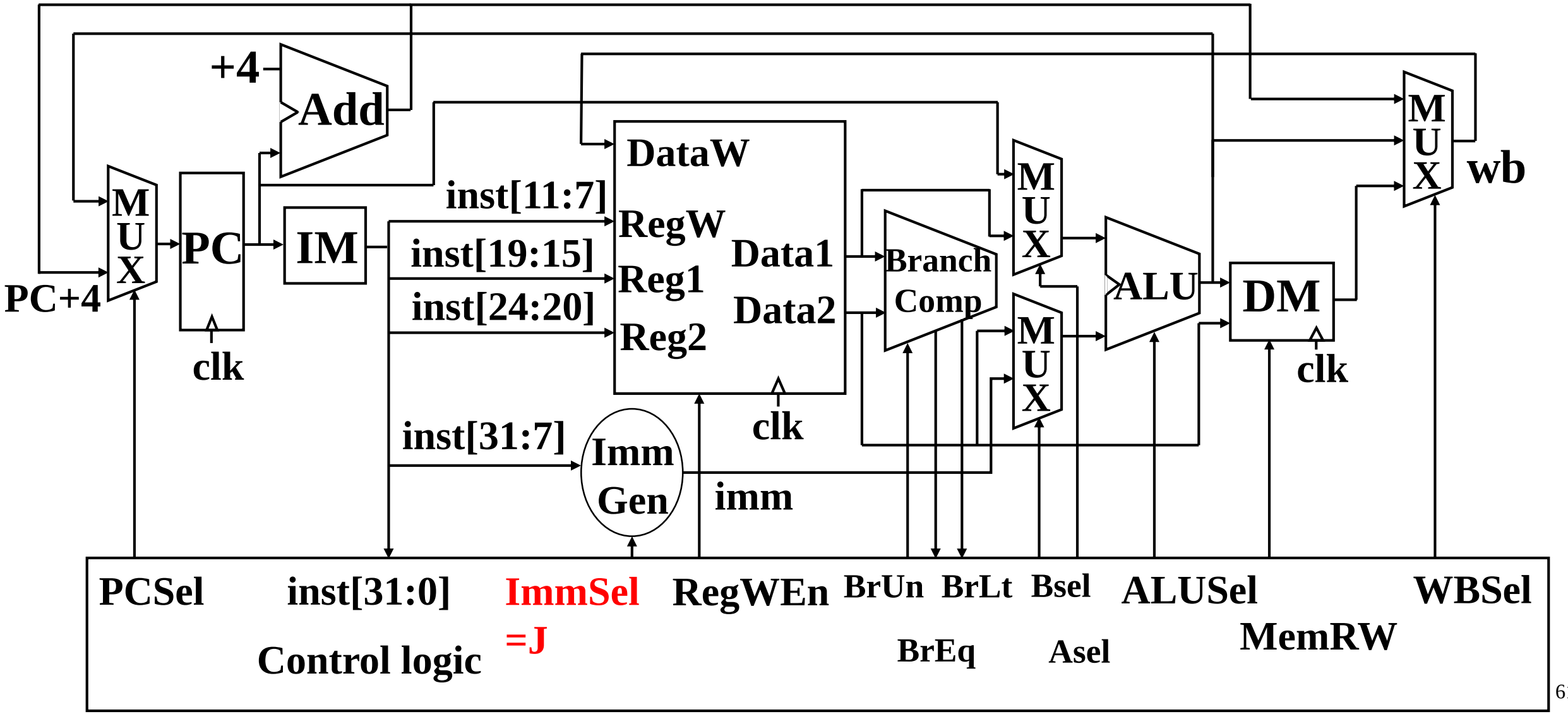


实现jal指令

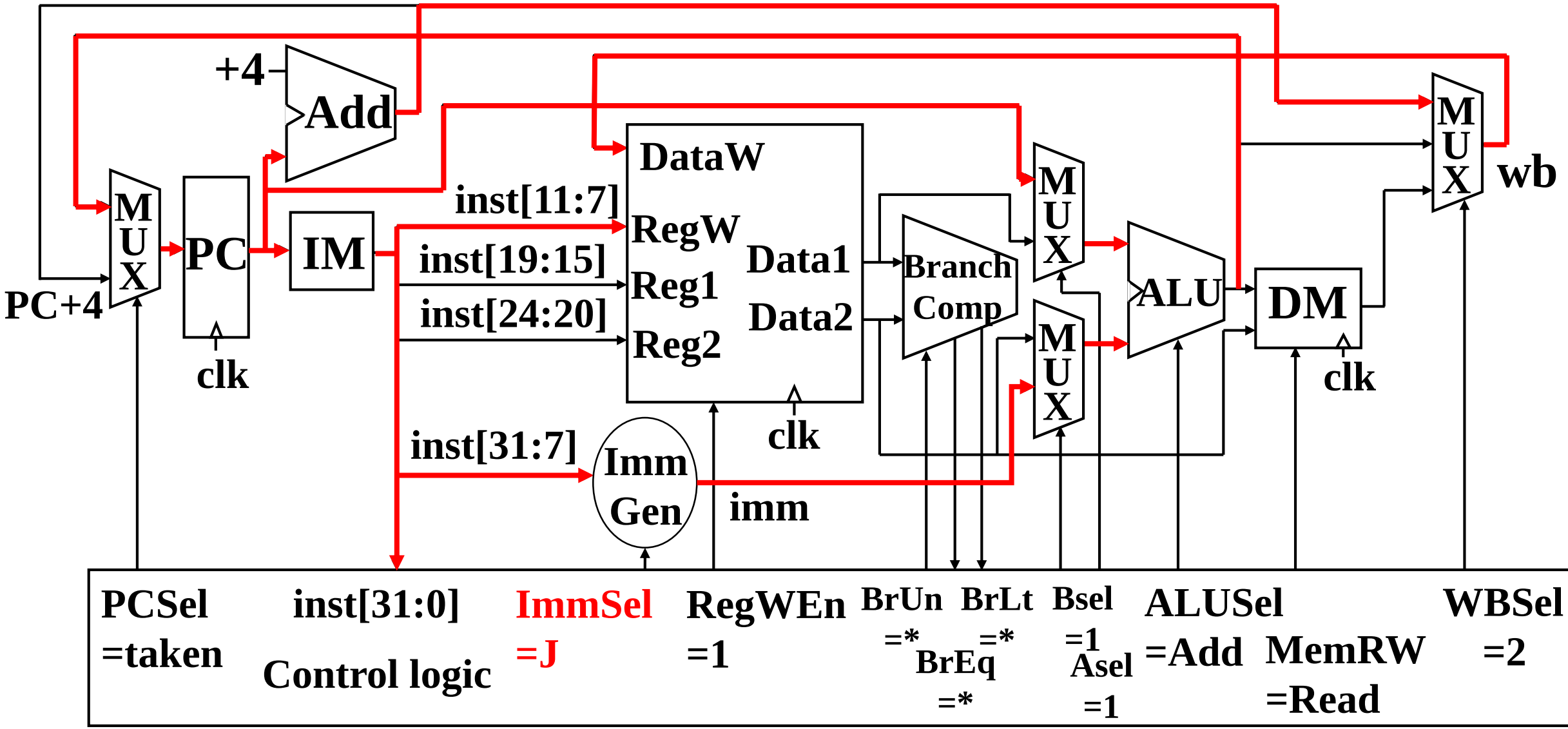


- jal将 PC + 4 写入目的寄存器rd中
- 设置 PC = PC + offset
- 立即数字段共有20位，对应一个$\pm 2^{19}$的以2字节为单位的偏移量

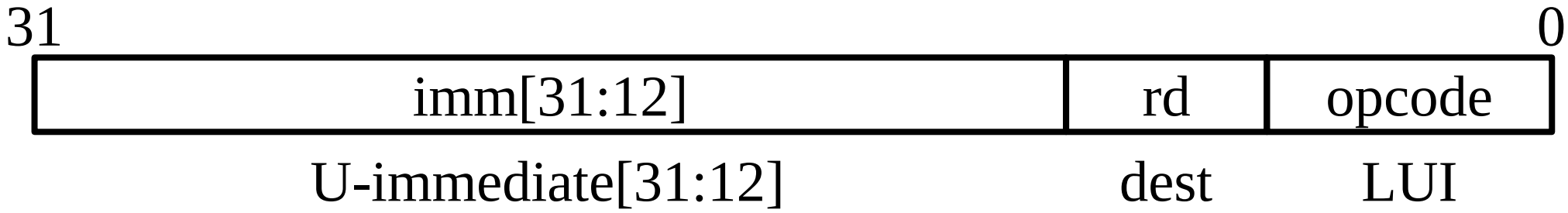
jal指令的数据通路


+4
Add
M
U
X
PC+4
PC
clk
IM
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
inst[31:7]
Imm
Gen
imm
Branch
Comp
M
U
X
M
U
X
M
U
X
ALU
DM
clk
wb
PCSel
inst[31:0]
ImmSel
=J
RegWEn
BrUn
BrLt
Bsel
ALUSel
WBSel
Control logic
BrEq
Asel
MemRW

jal指令的数据通路

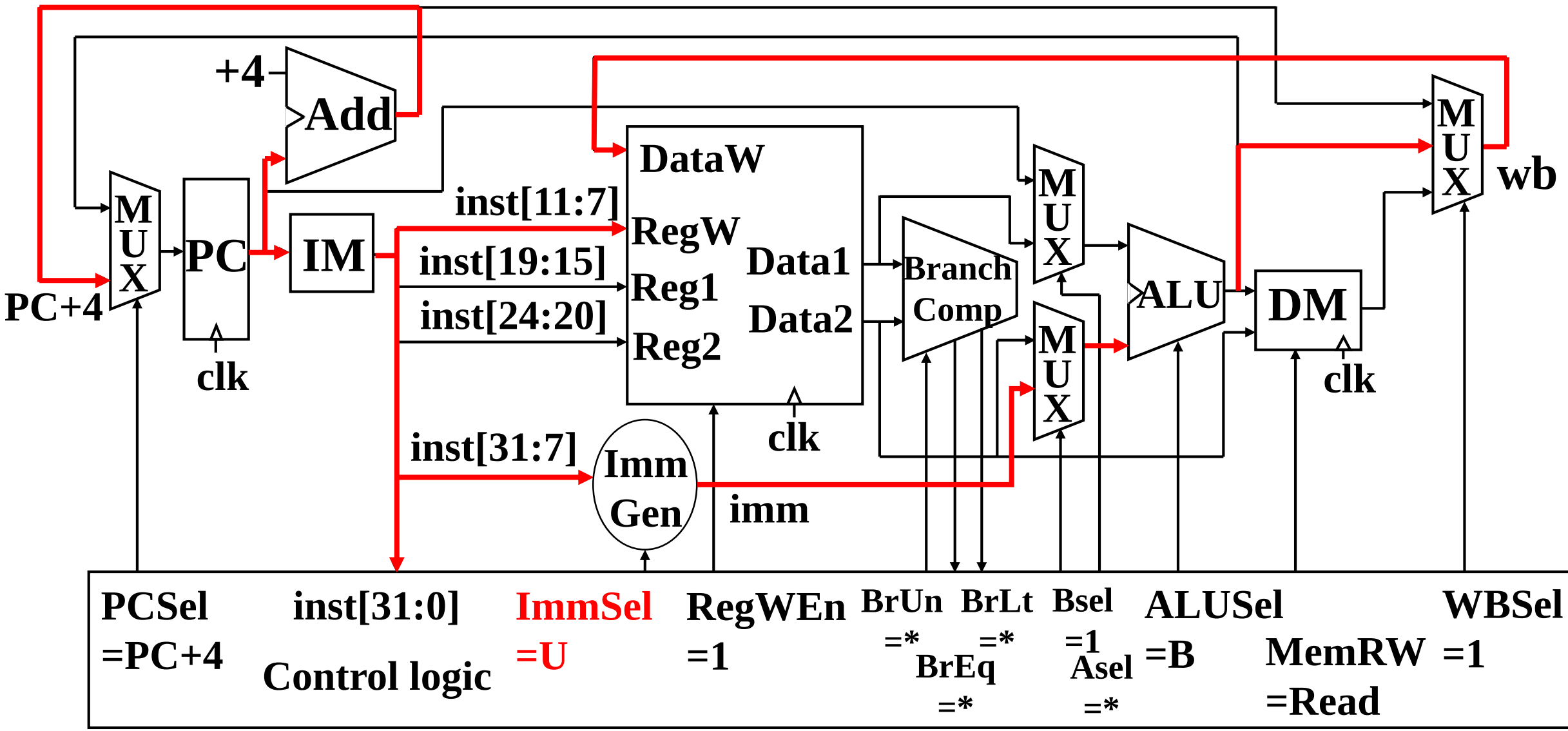

+4
Add
M
U
X
PC+4
PC
clk
IM
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
inst[31:7]
Imm
Gen
imm
Branch
Comp
M
U
X
M
U
X
M
U
X
ALU
DM
clk
wb
PCSel
=taken
inst[31:0]
Control logic
ImmSel
=J
RegWEn
=1
BrUn
=*
BrLt
=*
BrEq
=*
Bsel
=1
Asel
=1
ALUSel
=Add
MemRW
=Read
WBSel
=2

U型指令

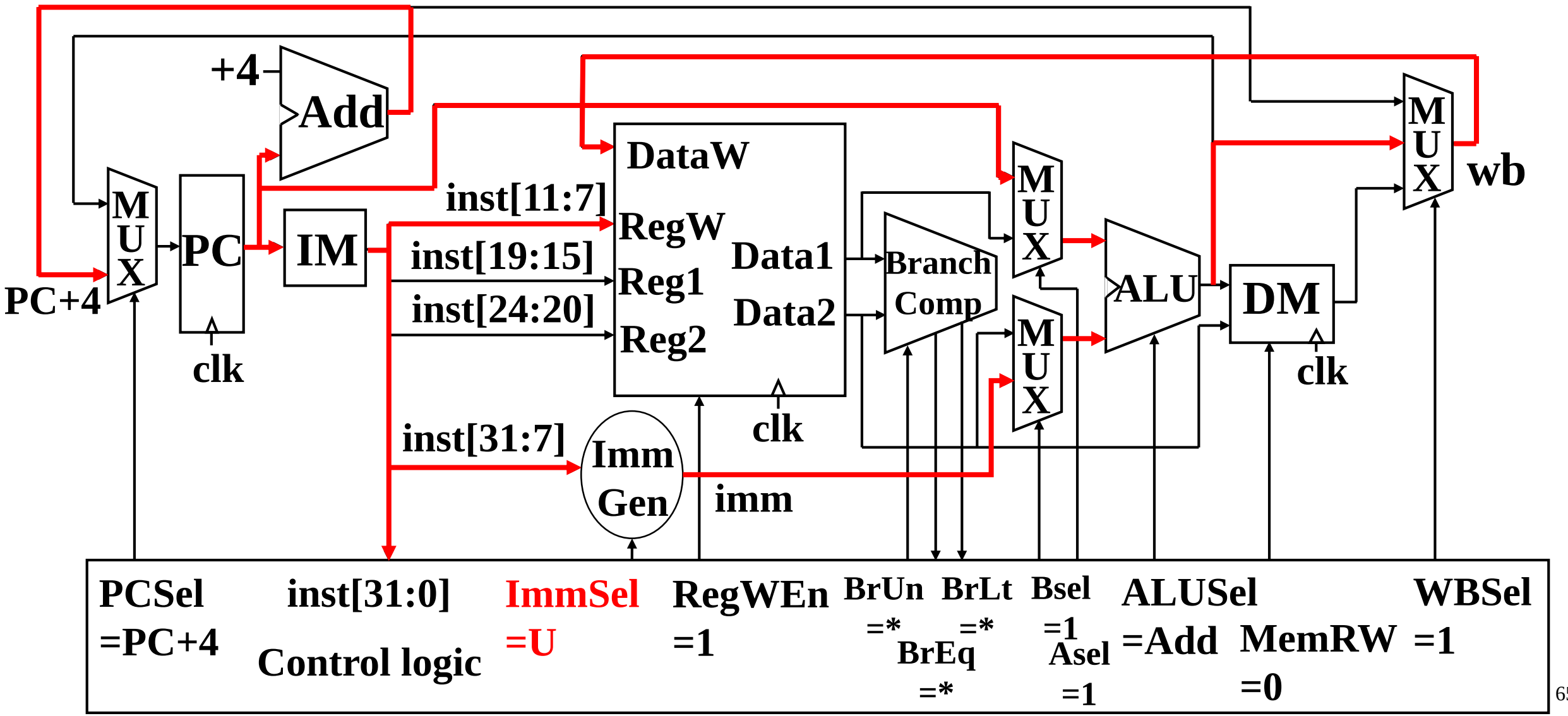


- U型指令进行长立即数操作
  - 高20位为长度为20位的立即数字段
  - 5位的目的地寄存器字段
- 用于两个指令
  - lui——将长立即数写入目的寄存器
  - auipc——将PC与长立即数相加结果写入目的寄存器

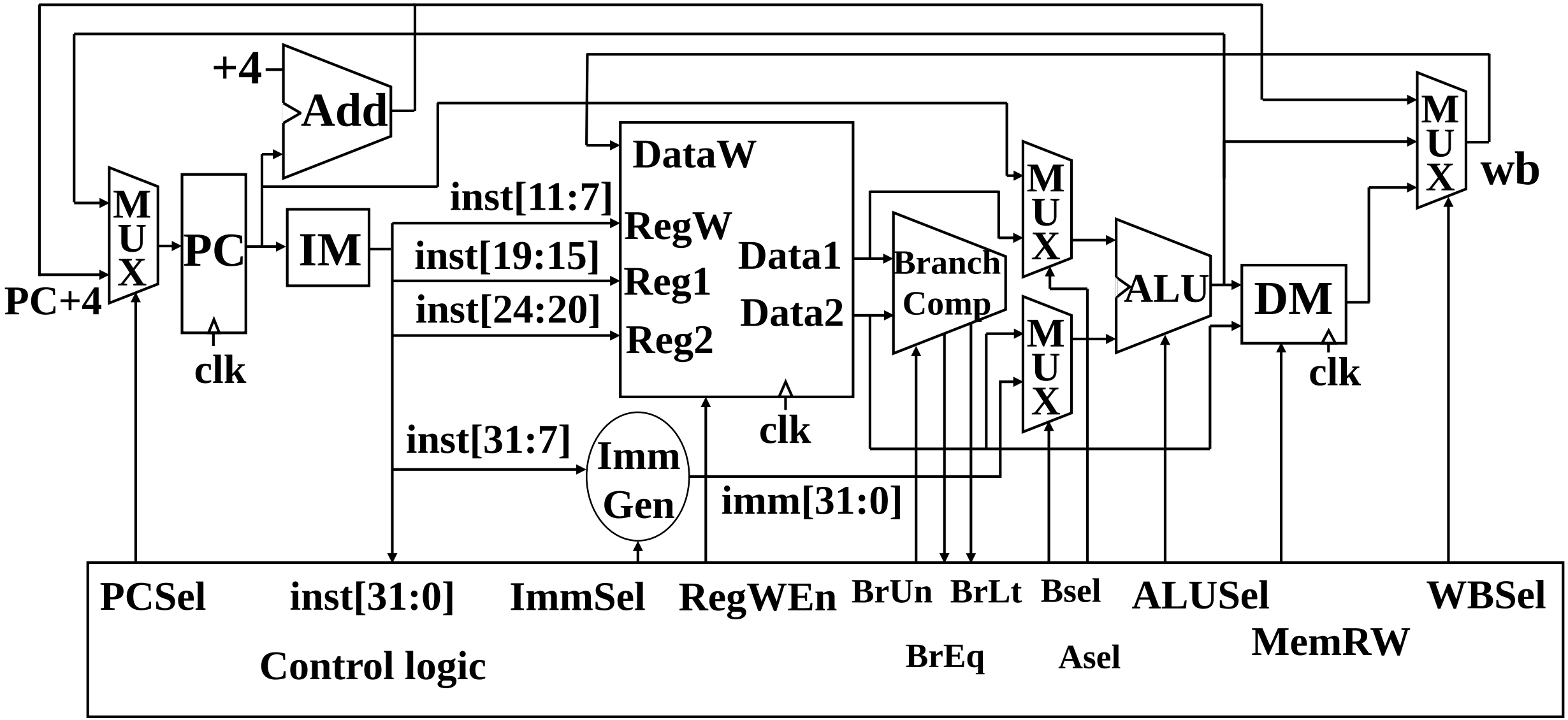
lui指令的数据通路


+4
Add
MUX
PC+4
PC
clk
IM
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
inst[31:7]
Imm Gen
imm
Branch Comp
MUX
MUX
MUX
ALU
DM
clk
MUX
wb
PCSel
=PC+4
inst[31:0]
Control logic
ImmSel
=U
RegWEn
=1
BrUn
=*
BrLt
=*
BrEq
=*
Bsel
=1
Asel
=*
ALUSel
=B
MemRW
=Read
WBSel
=1

auipc指令的数据通路



RISC-V 的数据通路


+4
Add
MUX
PC+4
PC
clk
IM
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
inst[31:7]
Imm
Gen
imm[31:0]
Branch
Comp
MUX
MUX
ALU
DM
clk
MUX
wb
PCSel
inst[31:0]
ImmSel
RegWEn
BrUn
BrLt
Bsel
ALUSel
WBSel
Control logic
BrEq
Asel
MemRW

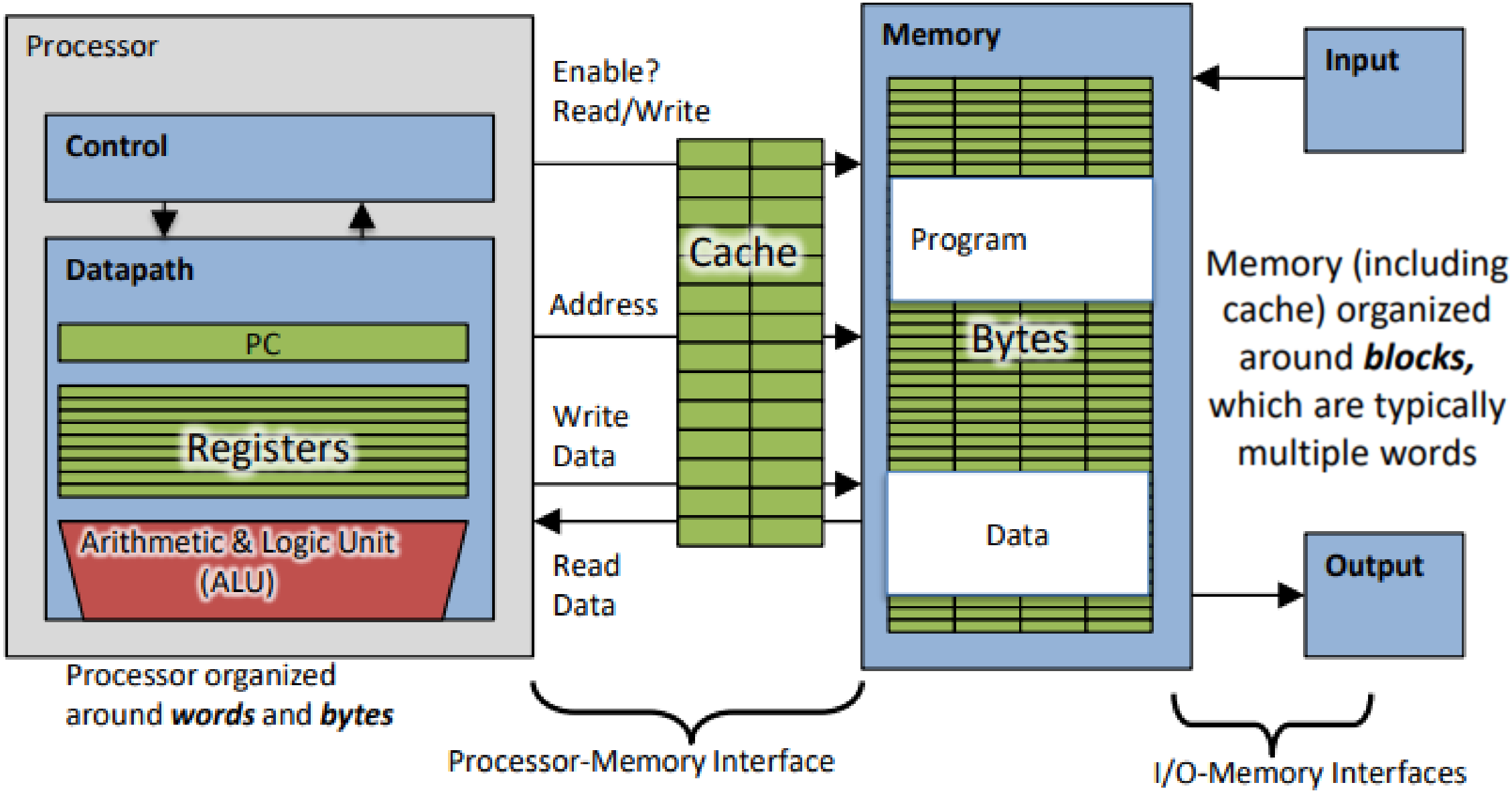
回顾

- 一个完整的数据通路
  - 能够在一个时钟周期内执行所有RISC-V指令
  - 并非所有指令都会用到所有硬件单元
- 5个执行阶段
  - IF、ID、EX、MEM、WB
  - 有的阶段只有部分指令才会用到
- 控制器指定如何执行指令
  - 控制器通过产生控制信号，控制指令如何被执行

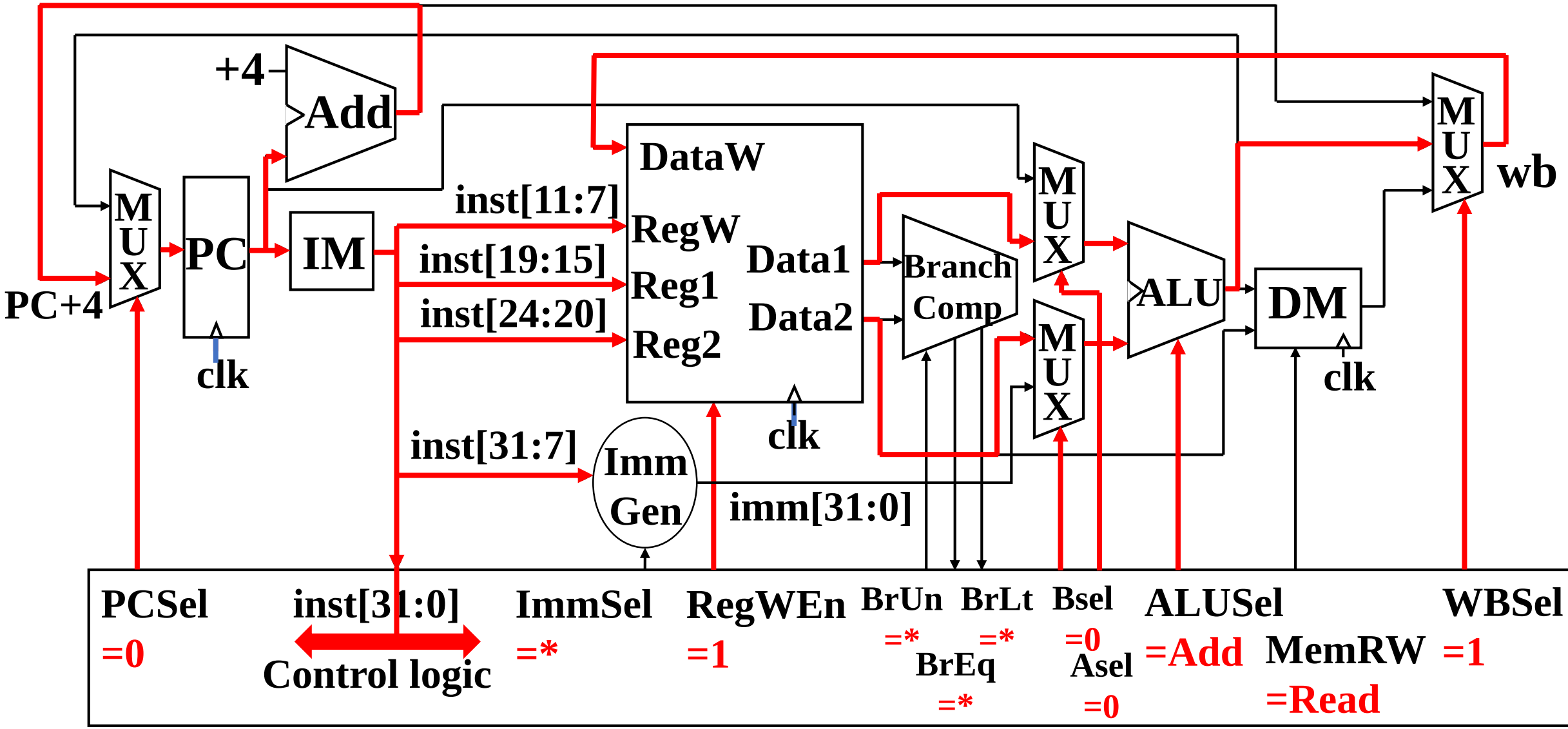
第六章 处理器设计

- RISC-V数据通路
  - 数据通路概念
  - RISC-V部分指令的数据通路
- RISC-V控制器

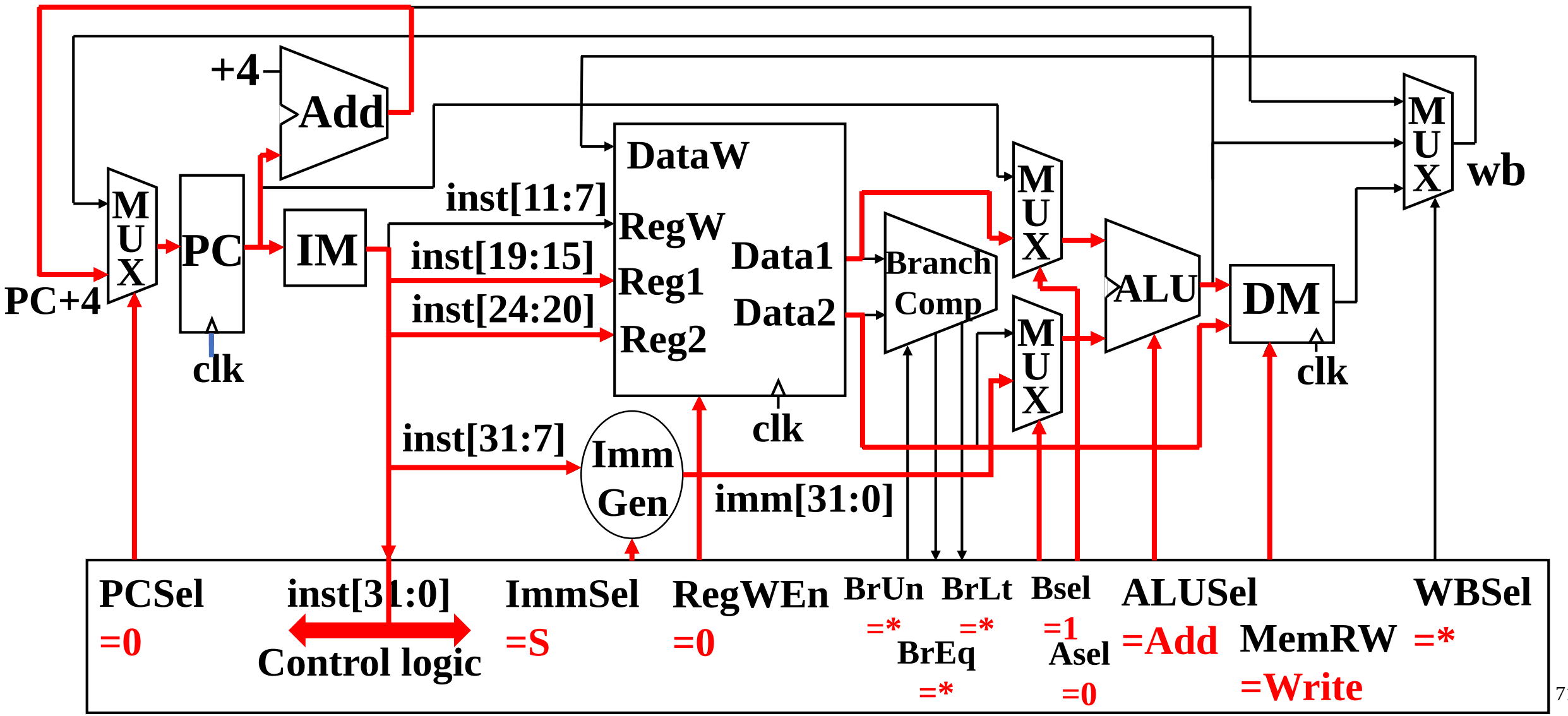
单核计算机系统


Processor
Control
Datapath
PC
Registers
Arithmetic & Logic Unit (ALU)
Enable?
Read/Write
Address
Write Data
Read Data
Cache
Memory
Program
Bytes
Data
Input
Output
Memory (including cache) organized around blocks, which are typically multiple words
Processor organized around words and bytes
Processor-Memory Interface
I/O-Memory Interfaces

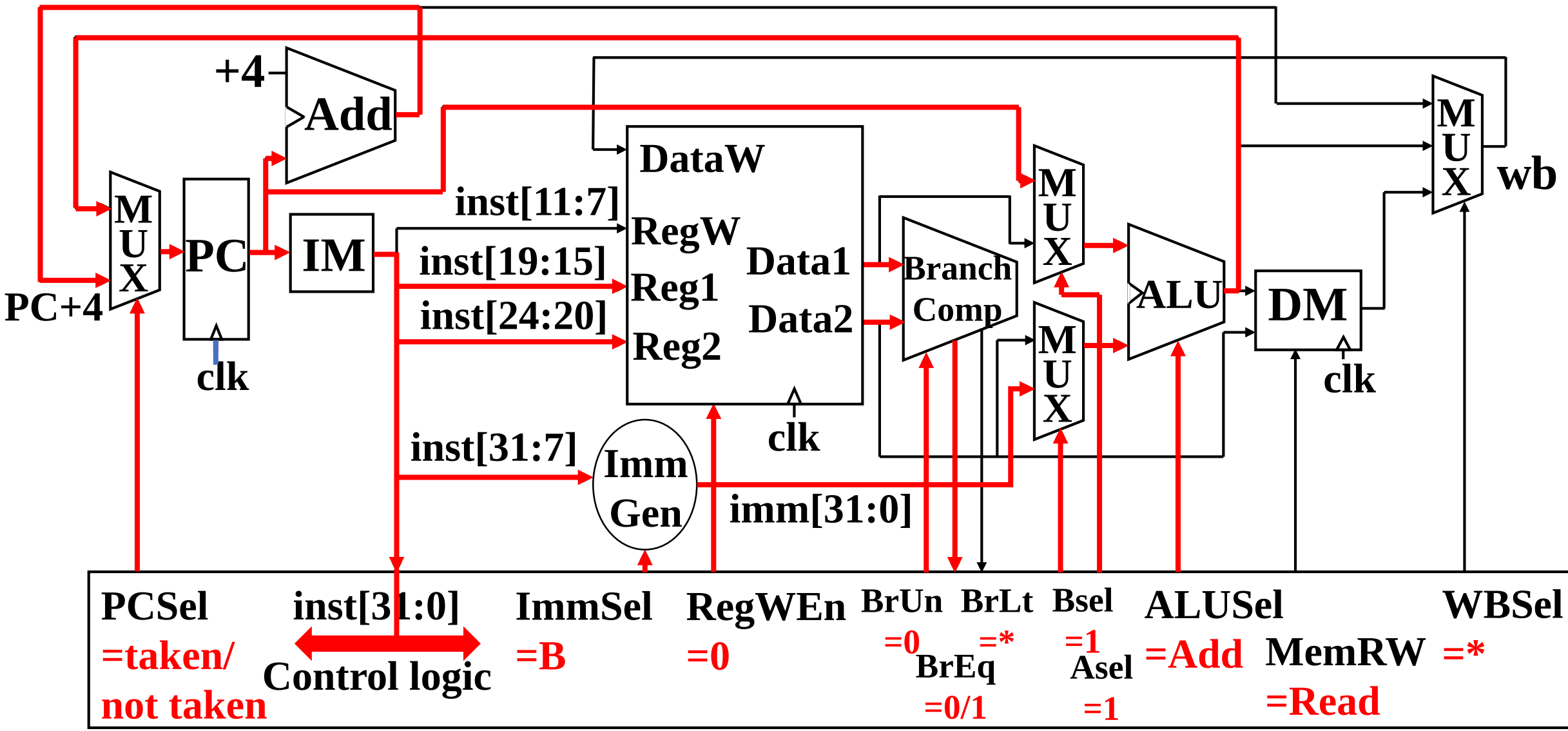
add指令的执行


+4
Add
MUX
PC+4
PC
clk
IM
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
inst[31:7]
Imm Gen
imm[31:0]
Branch Comp
MUX
MUX
MUX
ALU
DM
clk
MUX
wb
PCSel
=0
inst[31:0]
Control logic
ImmSel
=*
RegWEn
=1
BrUn
=*
BrLt
=*
BrEq
=*
Bsel
=0
Asel
=0
ALUSel
=Add
MemRW
=Read
WBSel
=1

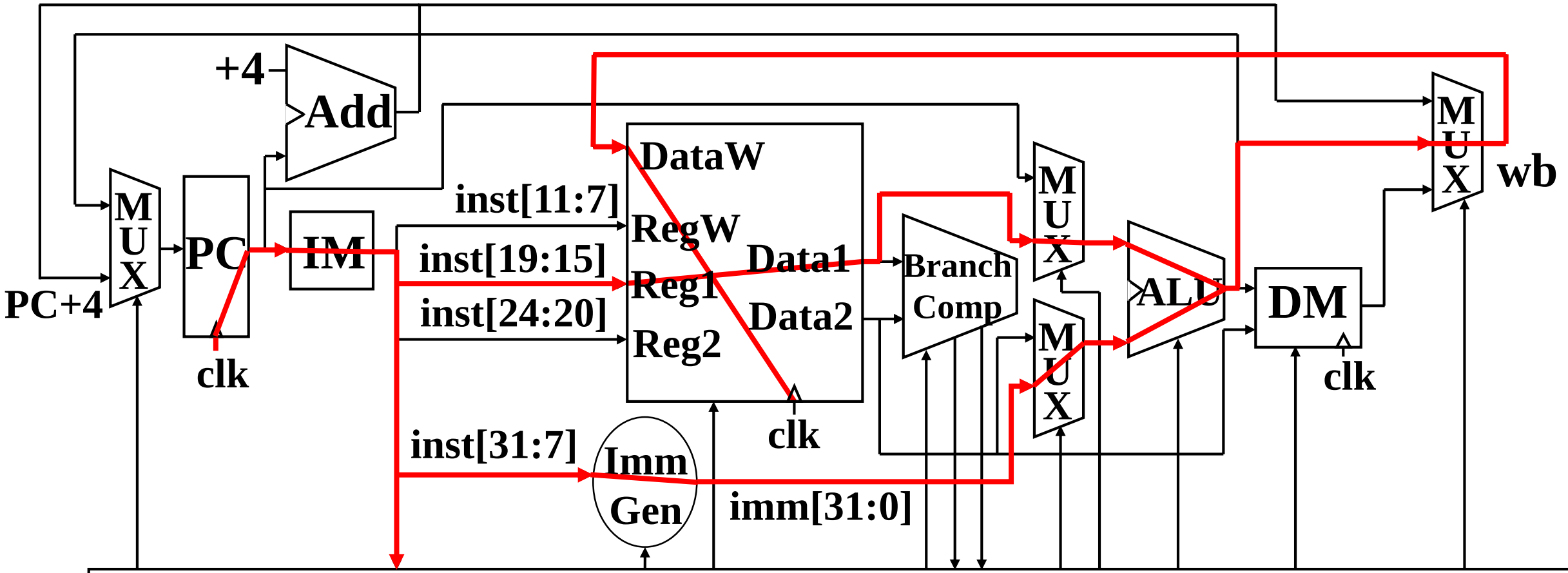
sw指令的执行


+4
Add
M
U
X
PC+4
PC
clk
IM
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
inst[31:7]
Imm
Gen
imm[31:0]
Branch
Comp
ALU
DM
clk
wb
PCSel
=0
inst[31:0]
Control logic
ImmSel
=S
RegWEn
=0
BrUn
=*
BrLt
=*
BrEq
=*
Bsel
=1
Asel
=0
ALUSel
=Add
MemRW
=Write
WBSel
=*

beq指令的执行


+4
Add
M
U
X
PC+4
PC
clk
IM
inst[11:7]
inst[19:15]
inst[24:20]
DataW
RegW
Reg1
Reg2
Data1
Data2
clk
inst[31:7]
Imm
Gen
imm[31:0]
Branch
Comp
ALU
DM
clk
wb
PCSel
=taken/
not taken
inst[31:0]
Control logic
ImmSel
=B
RegWEn
=0
BrUn
=0
BrLt
=*
BrEq
=0/1
Bsel
=1
Asel
=1
ALUSel
=Add
MemRW
=Read
WBSel
=*

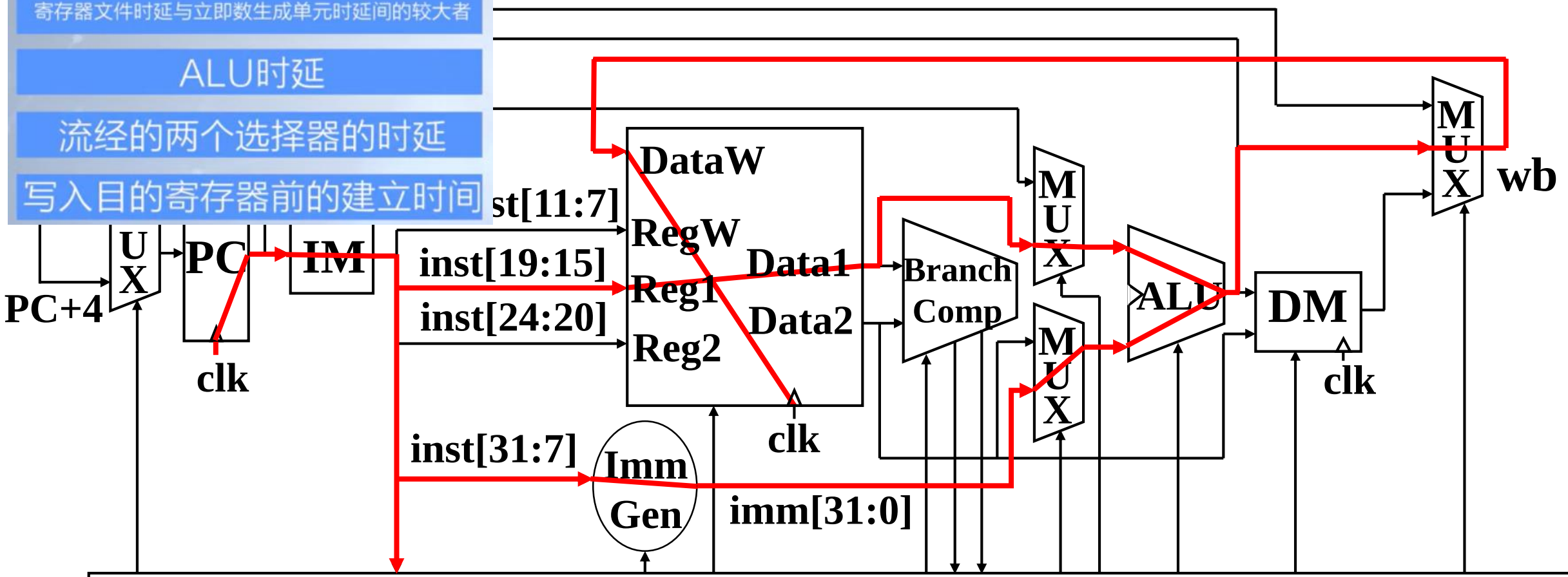
关键路径



$$t_{clk-q} + t_{IMEM} + \max\{t_{Reg} + t_{Imm}\} + t_{ALU} + 2t_{mux} + t_{Setup}$$

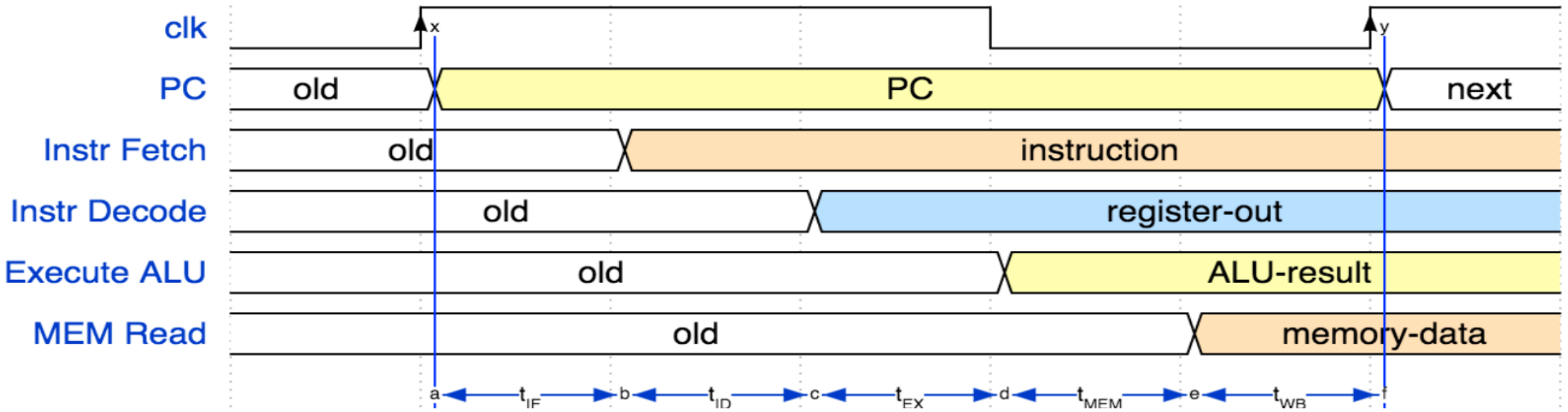
关键路径

- PC寄存器的时延
- 指令存储器的时延
- 寄存器文件时延与立即数生成单元时延间的较大者
- ALU时延
- 流经的两个选择器的时延
- 写入目的寄存器前的建立时间



$$t_{clk-q} + t_{IMEM} + \max\{t_{Reg} + t_{Imm}\} + t_{ALU} + 2t_{mux} + t_{Setup}$$

指令时延



IF	ID	EX	MEM	WB	Total
I-MEM	Reg Read	ALU	D-MEN	Reg W	
200ps	100ps	200ps	200ps	100ps	800ps

指令时延

Instr	IF=200ps	ID=100ps	ALU=200ps	MEM=200ps	WB=100ps	Total
add	X	X	X		X	600ps
beq	X	X	X			500ps
jal	X	X	X			500ps
lw	X	X	X	X	X	800ps
sw	X	X	X	X		700ps

- 最大时钟频率
  - $f_{max} = \frac{1}{800ps} = 1.25GHz$
- 大多数区块大部分时间处于空闲状态
  - $f_{max,ALU} = \frac{1}{200ps} = 5GHz$

控制信号真值表

Inst[31:0]	BrEq	BrLt	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWen	WBSel
add	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
sub	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
(R-R Op)	*	*	+4	*	*	Reg	Reg	(Op)	Read	1	ALU
addi	*	*	+4	I	*	Reg	Imm	Add	Read	1	ALU
lw	*	*	+4	I	*	Reg	Imm	Add	Read	1	Mem
sw	*	*	+4	S	*	Reg	Imm	Add	Write	0	*
beq	0	*	+4	B	*	PC	Imm	Add	Read	0	*
beq	1	*	ALU	B	*	PC	Imm	Add	Read	0	*
bne	0	*	ALU	B	*	PC	Imm	Add	Read	0	*
bne	1	*	+4	B	*	PC	Imm	Add	Read	0	*
blt	*	1	ALU	B	0	PC	Imm	Add	Read	0	*
bltu	*	1	ALU	B	1	PC	Imm	Add	Read	0	*
jalr	*	*	ALU	I	*	Reg	Imm	Add	Read	1	PC+4
jal	*	*	ALU	J	*	PC	Imm	Add	Read	1	PC+4
auipc	*	*	+4	U	*	PC	Imm	Add	Read	1	ALU

RISC V控制器实现

- ROM
  - 便于重新编程
    - 修正错误
    - 添加新的指令
  - 在人工设计控制逻辑时十分常用
- 组合逻辑
  - 现在很多的芯片设计者会使用逻辑综合工具，将控制器真值表实现为门级网表电路

RISC-V 编码

- 从RISC-V 指令集编码可以看出，实际上用来区分某条指令种类的编码只有9位——inst[30]、inst[14:12]、inst[6:2]

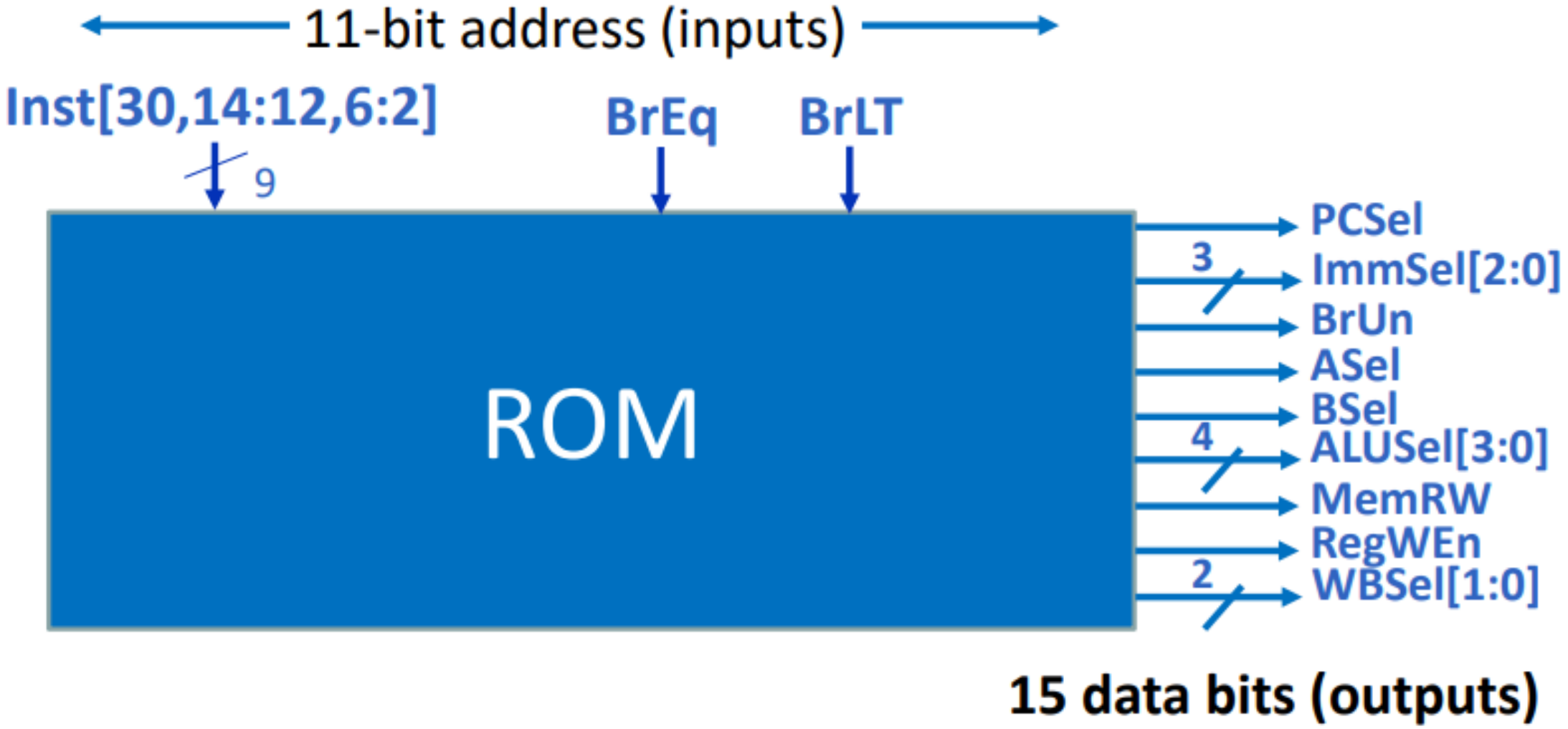
R 型	funct7	rs2	rs1	funct3	rd	opcode
I 型	imm[11:0]		rs1	funct3	rd	opcode
S 型	Imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
SB / B 型	Imm[12,10:5]	rs2	rs1	funct3	imm[4:1,11]	opcode
UJ / J 型	Imm[20,10:1,11,19:12]				rd	opcode
U 型	Imm[31:12]				rd	opcode

组合逻辑控制——例子

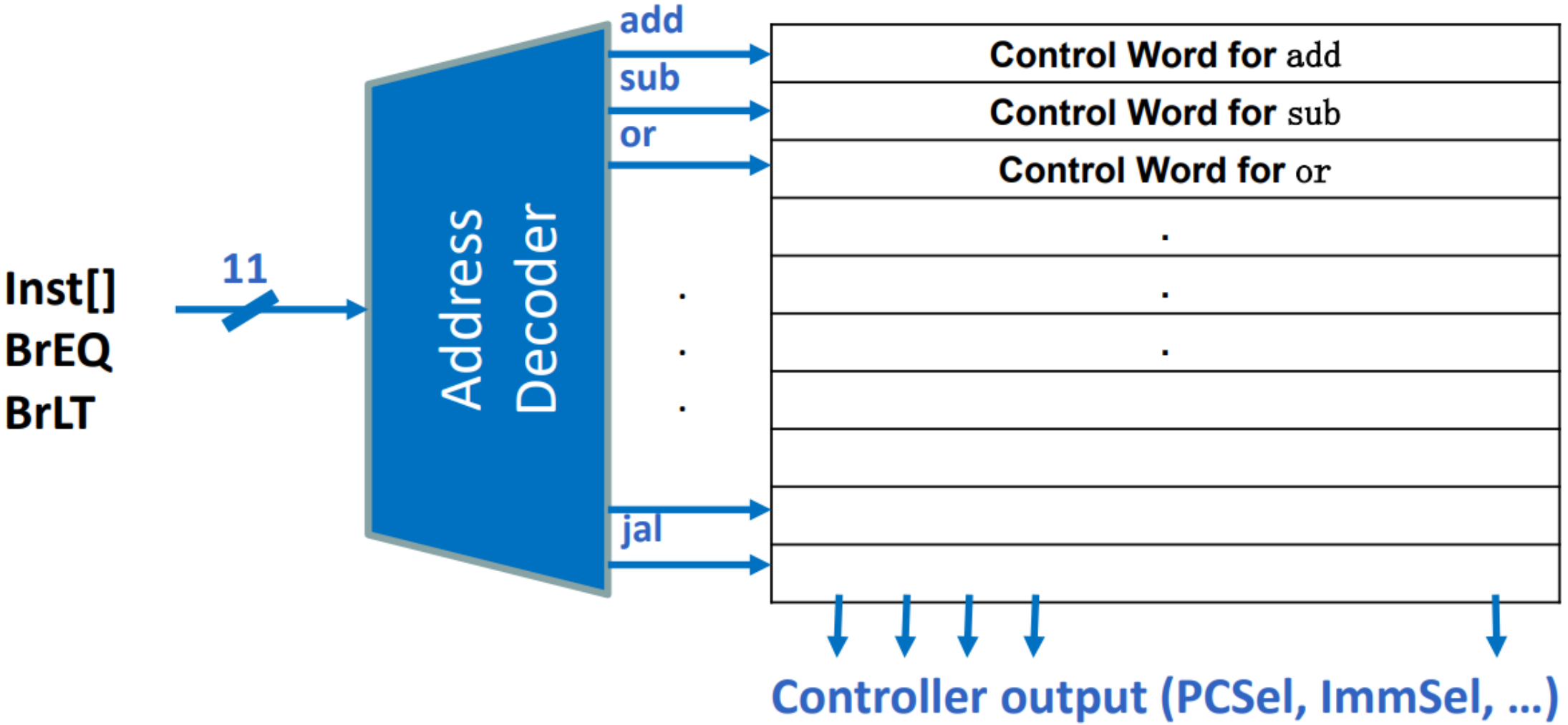
			Inst[14:12] ↓		Inst[6:2] ↓	
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

- BrUn = Inst[14] · Inst[13] · Branch

ROM 控制


11-bit address (inputs)
Inst[30,14:12,6:2]
9
BrEq
BrLT
ROM
PCSel
3
ImmSel[2:0]
BrUn
ASel
BSel
4
ALUSel[3:0]
MemRW
RegWEn
2
WBSel[1:0]
15 data bits (outputs)

ROM 控制器实现


Inst[]
BrEQ
BrLT
11
Address Decoder
add
sub
or
jal
Control Word for add
Control Word for sub
Control Word for or
Controller output (PCSel, ImmSel, ...)

回顾

- 实现了一个处理器
  - 能够在一个时钟周期内执行所有RISC-V指令
  - 并非所有指令都会用到所有硬件单元
  - 关键路径
- 5个执行阶段
  - IF、ID、EX、MEM、WB
  - 有的阶段只有部分指令才会用到
- 控制器指定如何执行指令
  - 基于ROM或组合逻辑实现