Chapter 4

处理器——指令集的实现

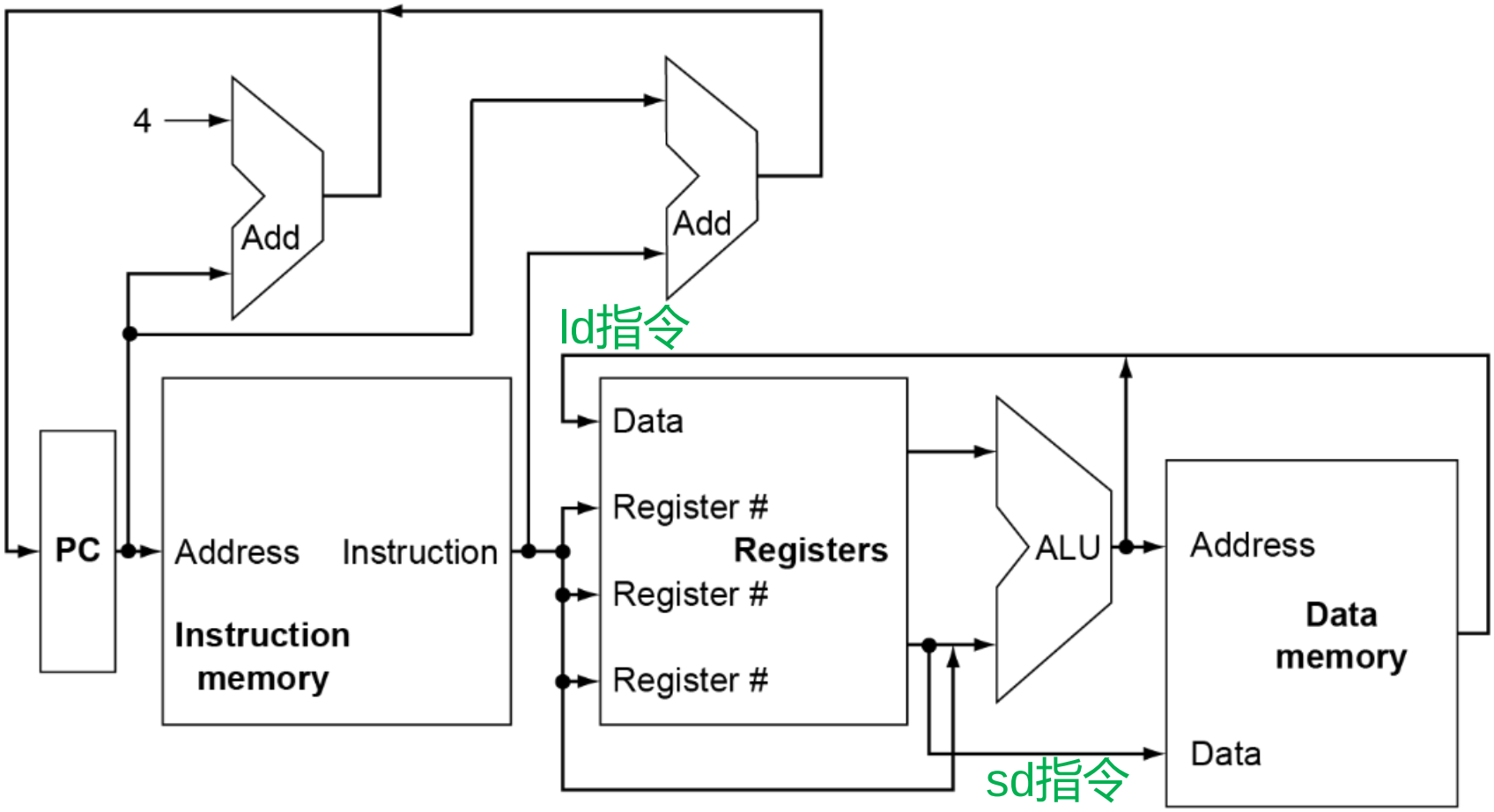
简介

- 影响CPU性能的因素
  - Time = # of Instruction * CPI * 时钟周期
  - # of Instruction
    - 由指令集架构（ISA）和编译器决定
  - CPI and 时钟周期
    - 由CPU硬件决定
- 将讨论两种 RISC-V 的实现
  - 一种简化版本（单周期CPU)
  - 一种更真实的版本（流水线CPU）
- 简化版本可以学习CPU的绝大部分概念
  - 内存访问指令: `ld`, `sd`
  - 算术逻辑指令: `add`, `sub`, `and`, `or`
  - 控制转移指令: `beq`

Outline

- **CPU的组件**
- 数据通路
- 控制器

CPU 预览


4
Add
Add
ld指令
PC
Address
Instruction
Instruction memory
Data
Register #
Registers
Register #
Register #
ALU
Address
Data memory
Data
sd指令

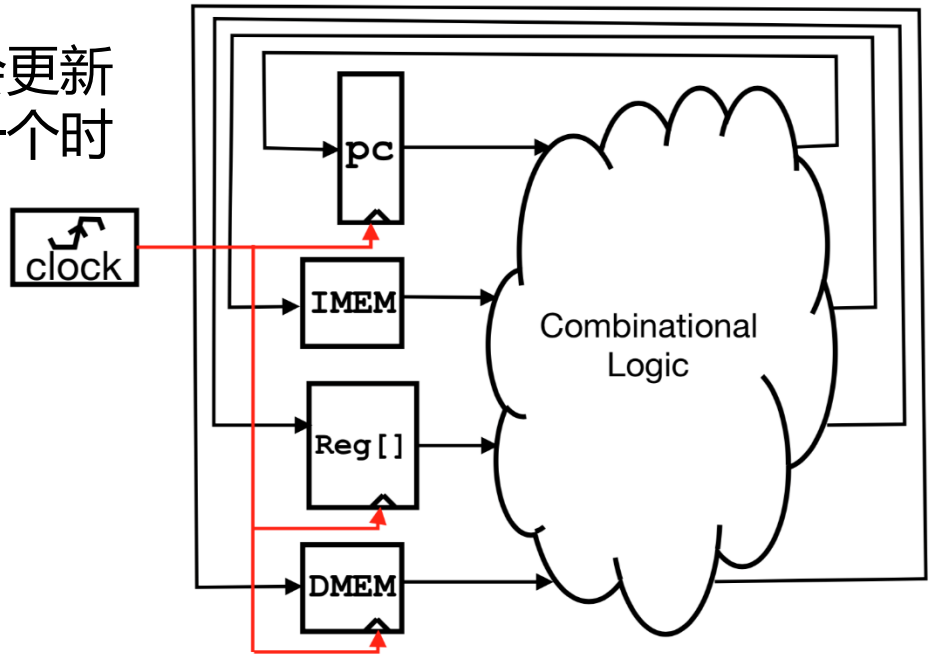
需要“状态”

每条指令的执行，都会读取和更新“状态”:

- **寄存器 (x0..x31)**
  - 寄存器文件 **Reg** 包括32 个 64 位register: Reg[0].. Reg[31]
    - 指令中的 *rs1*、*rs2* 字段，指定第一、二个读寄存器（来源）
    - 指令中的 *rd* 字段，指定写寄存器（目的）
    - x0永远是0（写入**Reg[0] 会**忽略写操作**)**
- **程序计数器 (PC)**
  - 记录和更新当前指令的地址
- **内存 (MEM)**
  - 可以存储指令和数据，字节寻址
  - 分开使用指令内存 (**IMEM**) 和数据内存(**DMEM**)
    - 后面会将IMEM/DMEM换成指令缓存和数据缓存
  - 从指令内存读取指令（取指）——可假设IMEM是只读的
  - load指令/store指令才会访问数据内存

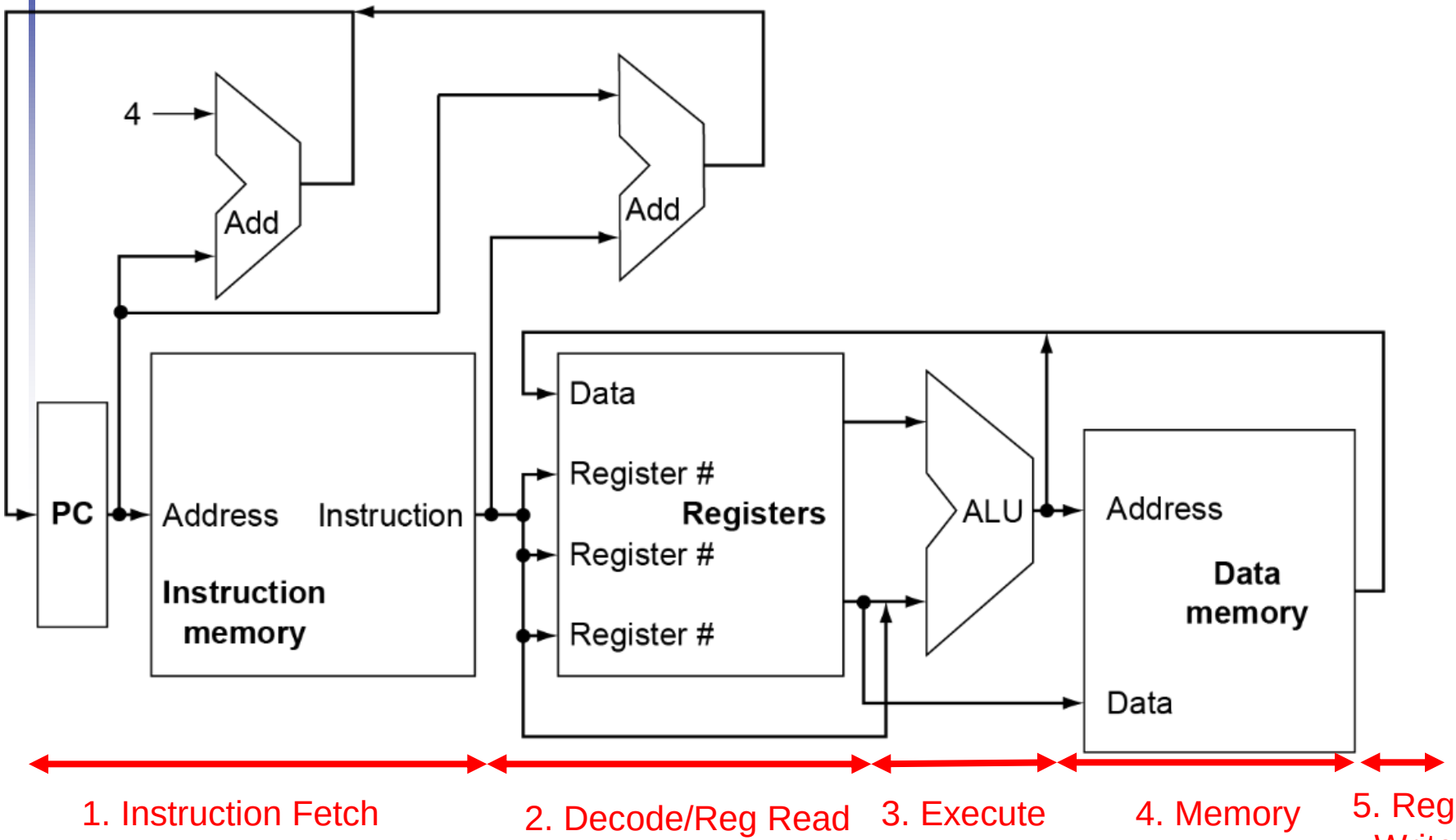
单周期 RISC-V 机器

- 当前状态，作为组合逻辑的输入；组合逻辑进行运算，其输出会更新状态（在下一个时钟上升时）
- 在时钟上升沿，组合逻辑的输出会更新所有的状态逻辑；随之，开始下一个时钟的读取—计算—更新操作
  - 读写PC：每条指令
  - 读/写DMEM：load/store指令
  - 读/写reg：每条指令/R型指令
  - 读IMEM：每条指令



- 分开的IMEM和DMEM:
  - 简化版本中, 内存是异步读 (not clocked), 同步写 (is clocked)

CPU 预览(数据通路)

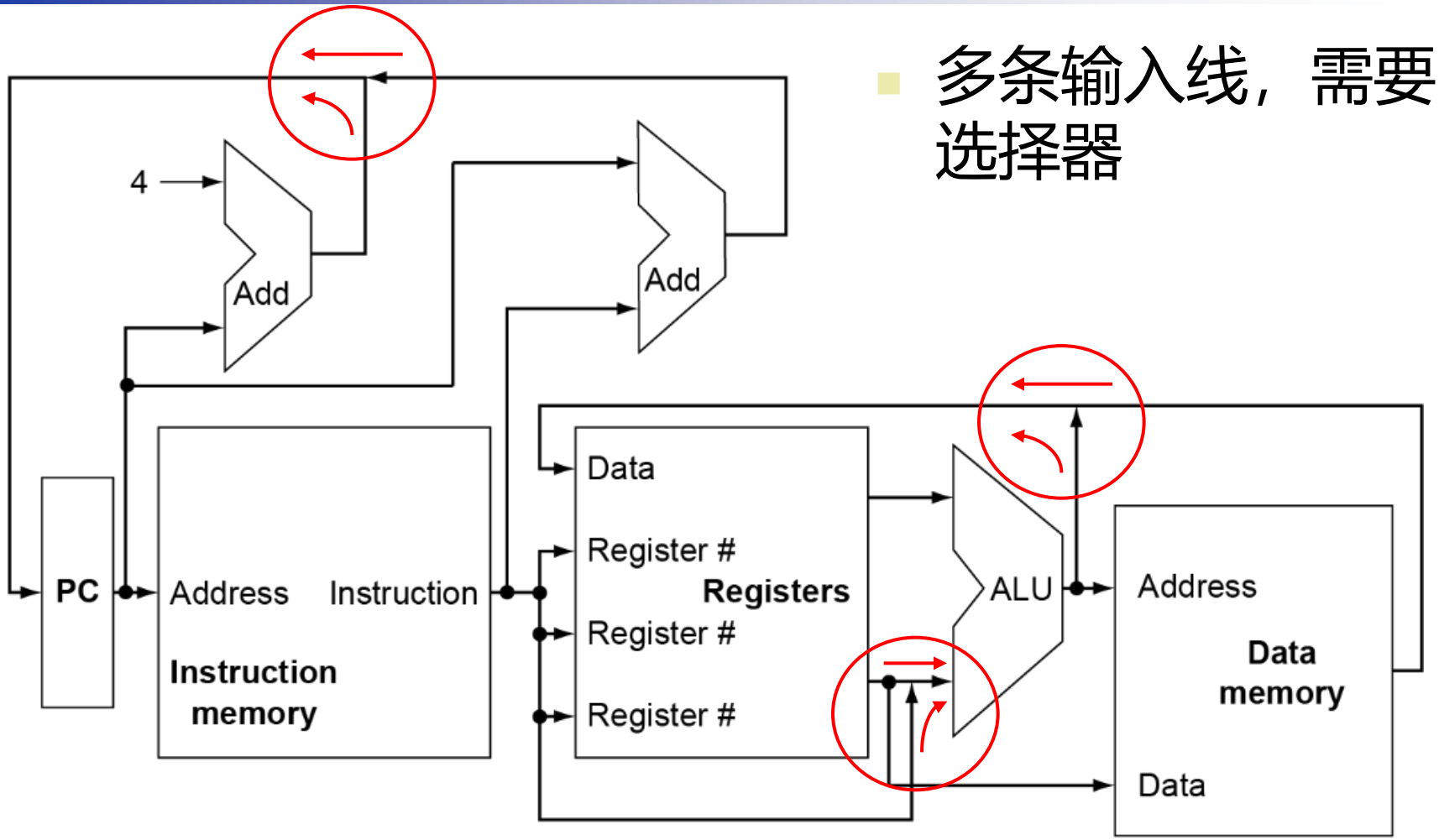

4
Add
Add
PC
Address
Instruction
Instruction memory
Data
Register #
Registers
Register #
Register #
ALU
Address
Data memory
Data
1. Instruction Fetch
2. Decode/Reg Read
3. Execute
4. Memory
5. Reg Write
Clock
时间线

单周期内指令的执行

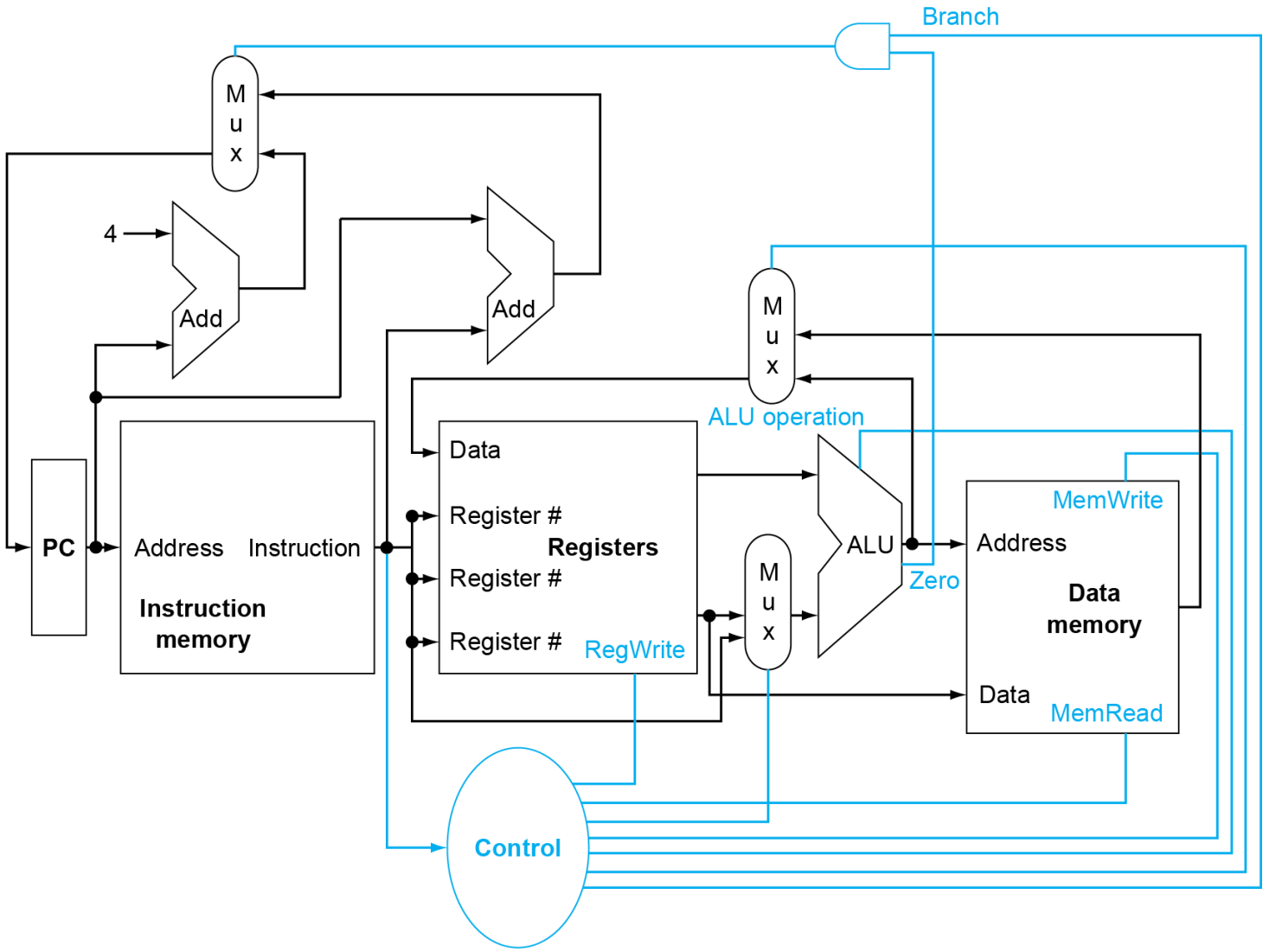
- PC提供指令地址 → 从**IMEM**中读取指令
- 指令提供寄存器编号 →从**寄存器文件**读取指定寄存器
- 根据指令提供的操作码，分别
  - 使用 ALU 进行计算，得到
    - 算术运算的结果
    - load指令/store指令需要访问的内存地址
    - 分支比较的结果
  - load/store指令读/写访问DMEM
  - 对于load指令和R型指令，将结果**更新**到目的寄存器
- **更新**PC ← 分支指令的目标地址 或者 PC + 4

CPU预览(数据通路+选择器)

- 多条输入线，需要选择器



CPU预览（控制器）


Branch
M
u
x
4
Add
Add
M
u
x
ALU operation
PC
Address
Instruction
Instruction
memory
Data
Register #
Registers
Register #
Register #
RegWrite
M
u
x
ALU
Zero
MemWrite
Address
Data
memory
Data
MemRead
Control

Outline

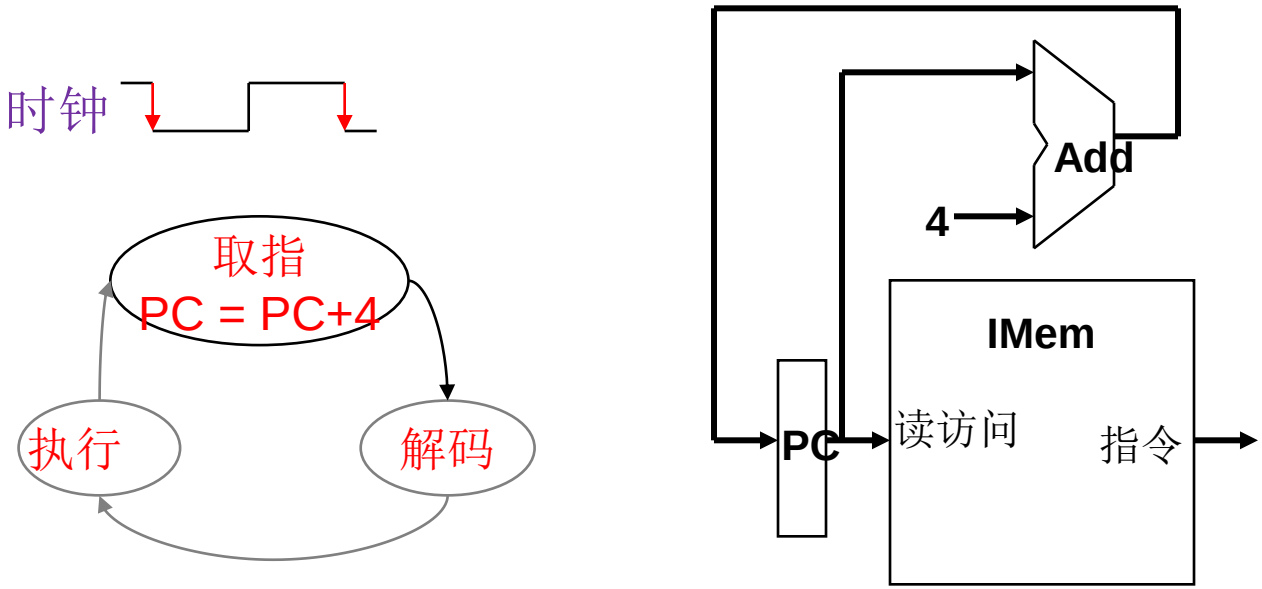
- **CPU的组件**
- **数据通路**
- The controller

构建数据通路

- 数据通路
  - CPU中用来处理数据和地址的逻辑的集合
    - 寄存器, ALU, 选择器, 内存, …
- 一步步构建RISC-V的数据通路

Fetching Instructions取指（图4-6）

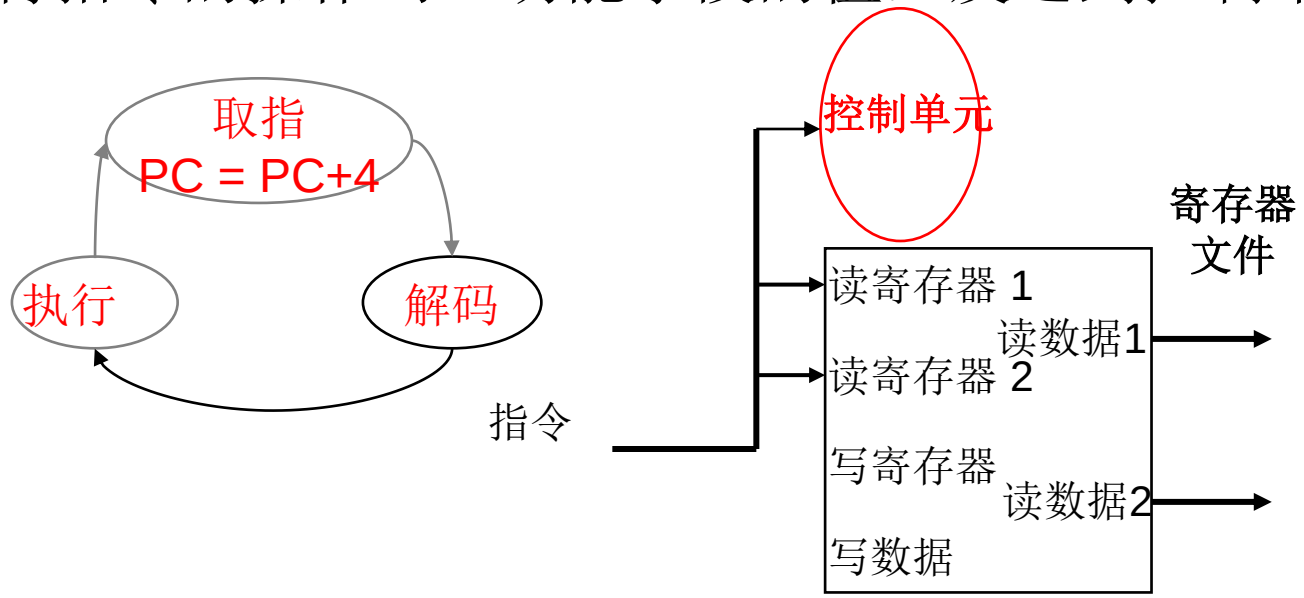
- 取指包括
  - 从IMem中读取指令
  - 更新PC：修改程下一条（顺序）指令的地址



  - 每个时钟都会更新PC寄存器, 因此 PC寄存器不需要写控制信号, 只需要时钟信号
  - 从IMem读取指令是组合逻辑行为（不需要更新状态）, 因此不需要读控制信号

Decoding Instructions 解码

- 解码包括
  - 解码：理解指令的含义
  - 将指令的操作码、功能字段的值，发送到控制单元

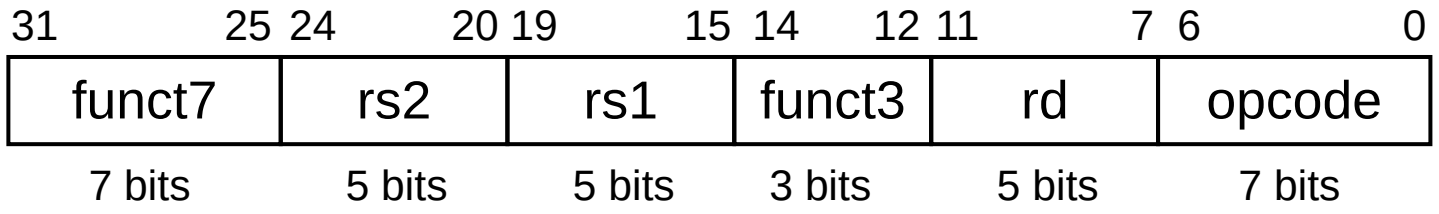


  - 从寄存器文件读取两个寄存器
    - 指令内部包含这两个寄存器的编号
    - 不必等解码完成？

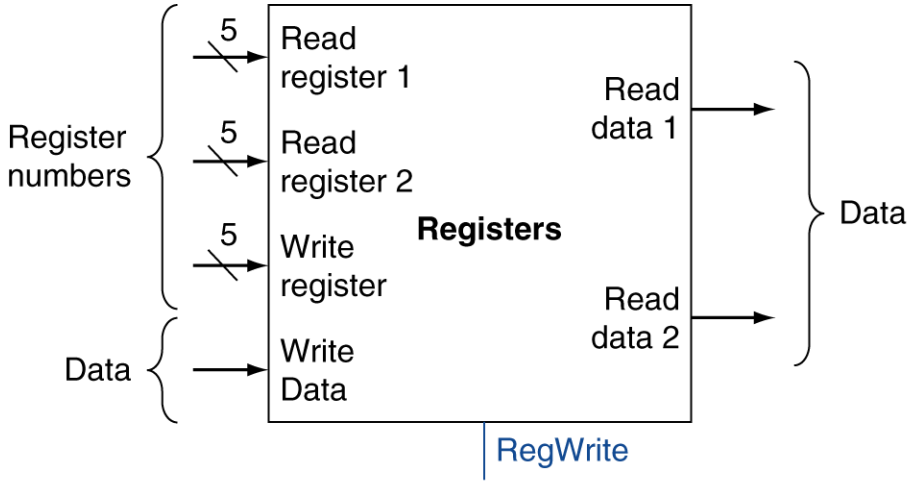
读取寄存器

- 注意：对于所有指令，在解码时寄存器文件的2个读端口都是活跃的（分别使用rs1和rs2指定寄存器）
  - 但是，此时指令还没有解码完成，CPU压根不知道指令的类别！
  - 选择提前读取2个寄存器，以防万一该指令真的会使用这些值
    - Forgiveness is better than asking for permit!

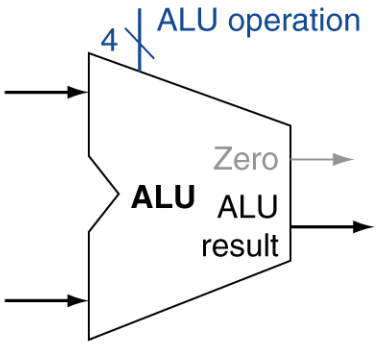
R-型 指令（图4-7）



- 读两个寄存器
- 进行 算术逻辑运算
- 写寄存器的结果

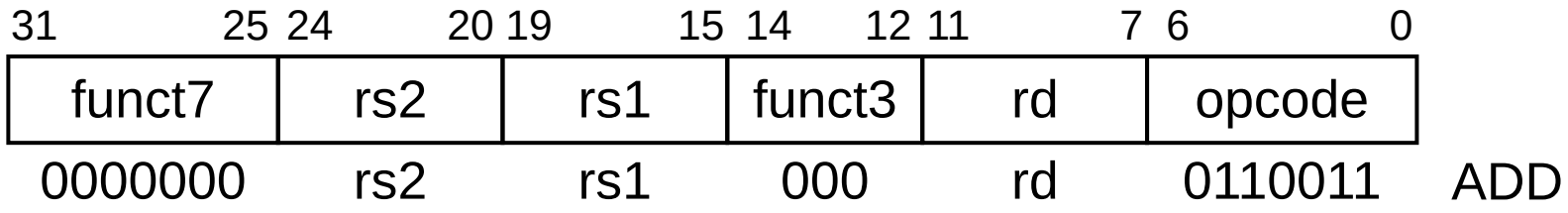


a. Registers

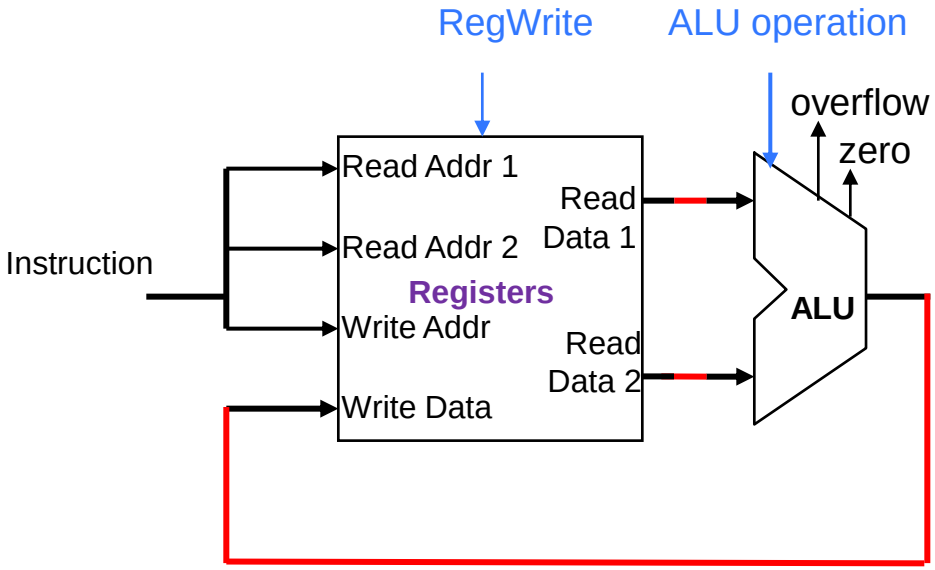


b. ALU

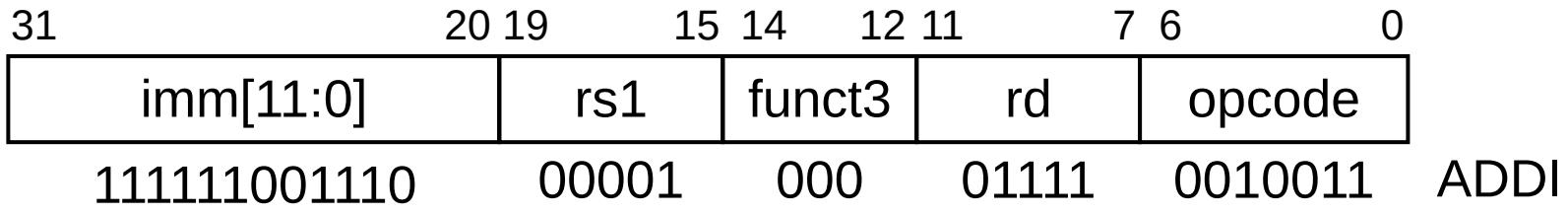
实现add指令



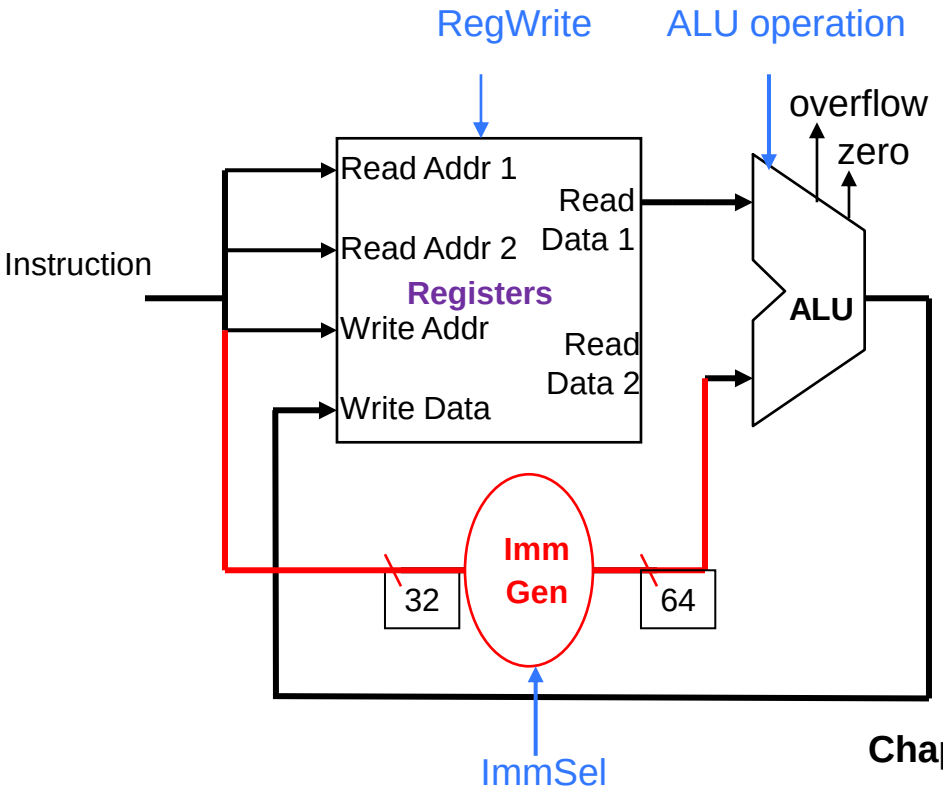
add rd, rs1, rs2



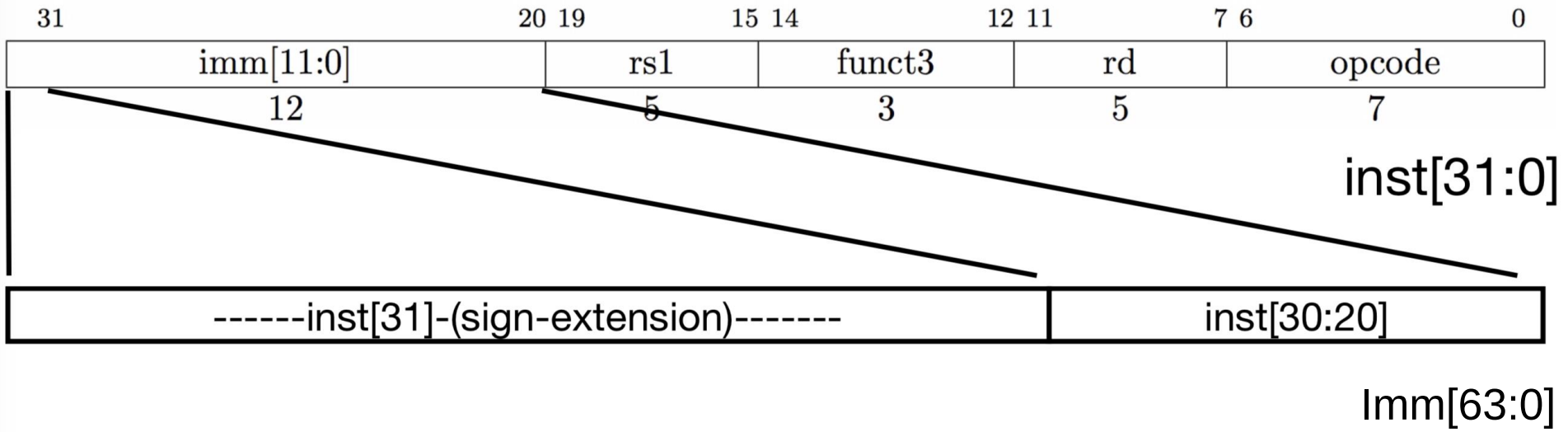
实现addi指令

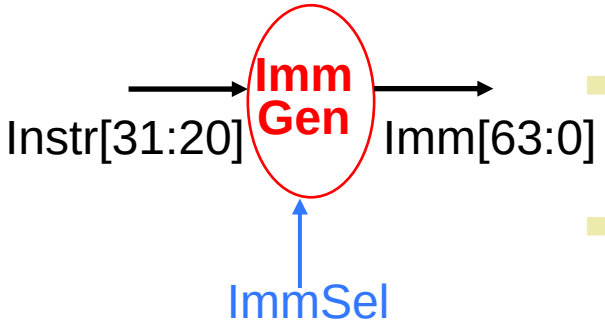


addi x15, x1, -50



I-型指令中的 immediate（图4-8）

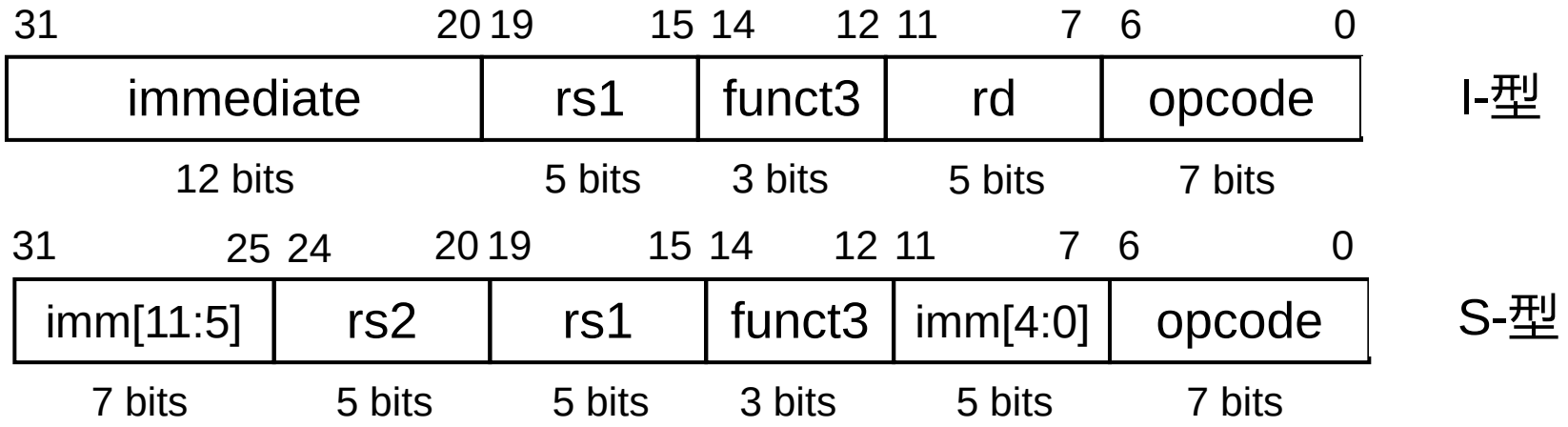




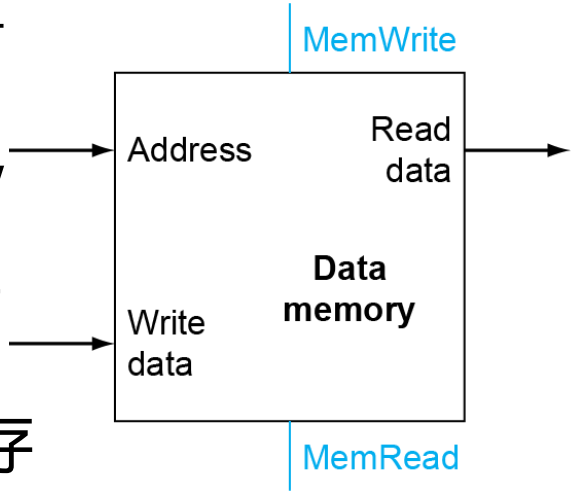
- 指令的高12位 (inst[31:20]) 拷贝到 immediate 的低12位 (imm[11:0])
- 符号扩展：将符号位inst[31]填充到immediate 的高52位 (imm[63:12])

所有其他的 I-型 算术指令 (slti, sltiu, andi, ori, xori, slli, srli, srai)类似，只需要修改 ALU的选择信号op

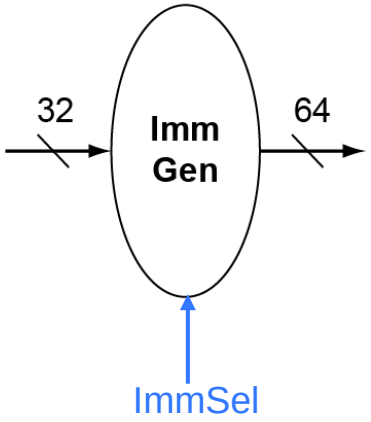
load指令/Store 指令（图4-8）



- 利用rs1和12位偏移，计算地址
  - 先对偏移imm进行符号扩展，再使用ALU
- load指令: 读内存，更新寄存器
- Store: 将寄存器值写入内存

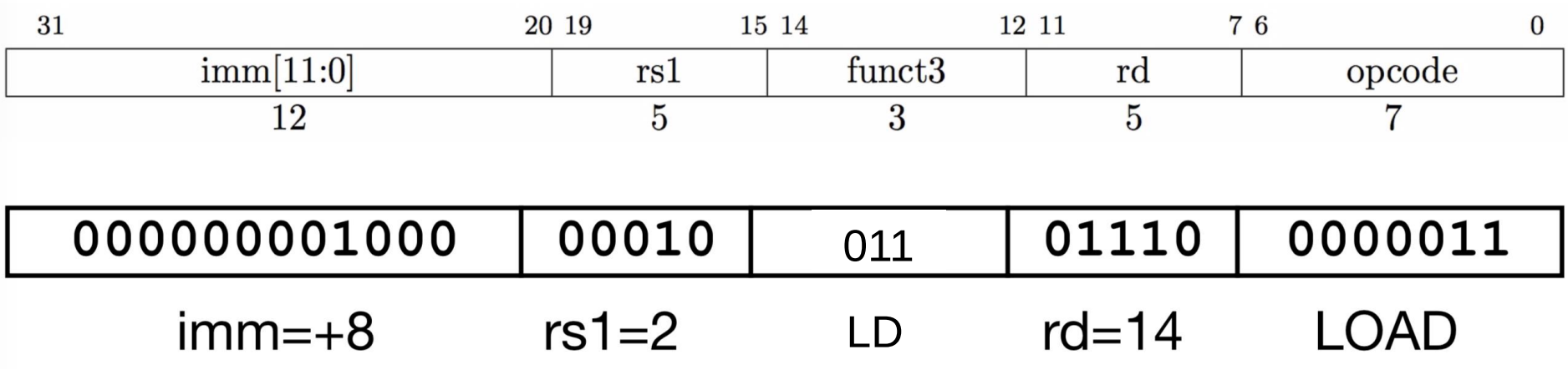


a. Data memory unit



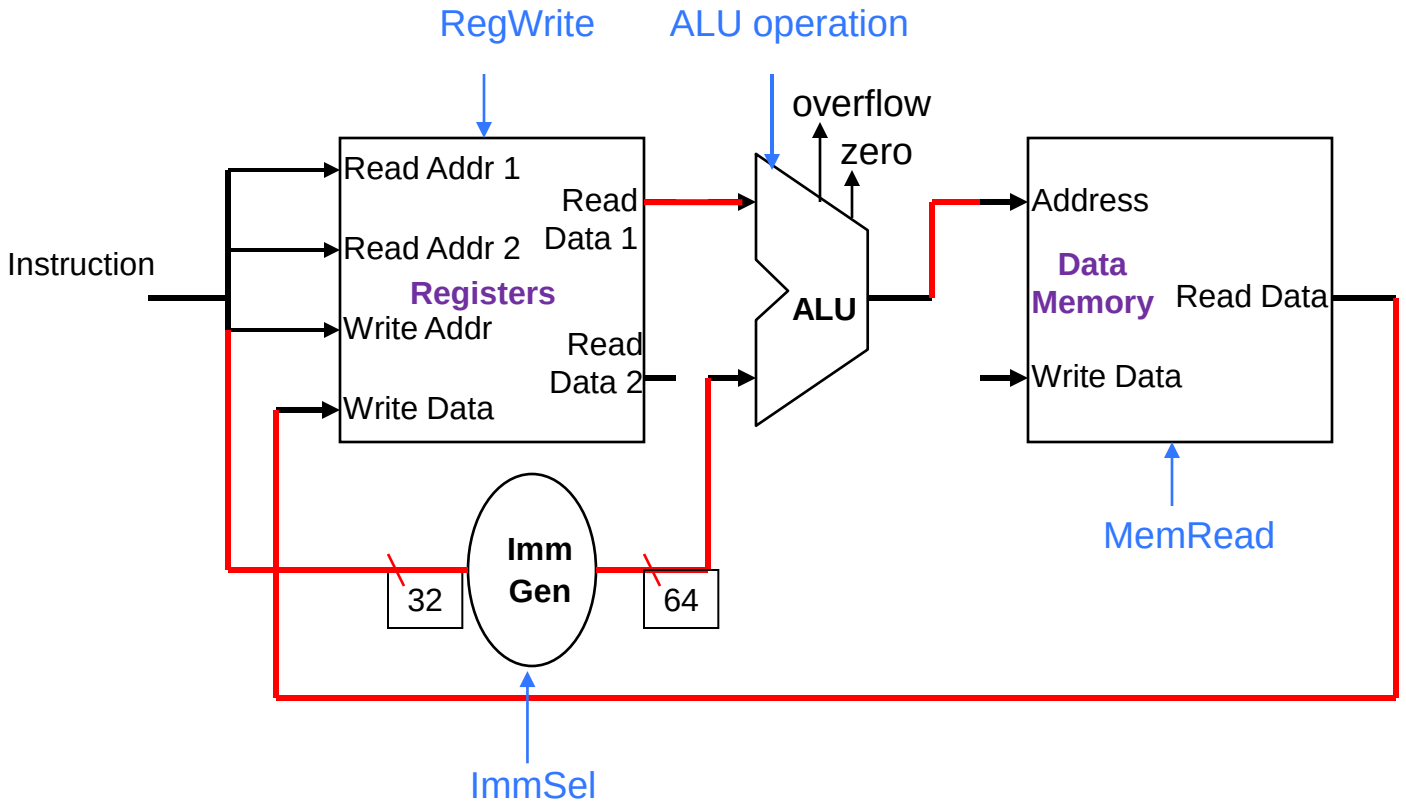
b. Immediate generation unit

实现load指令



LD x14, 8(x2)

执行 load指令的过程

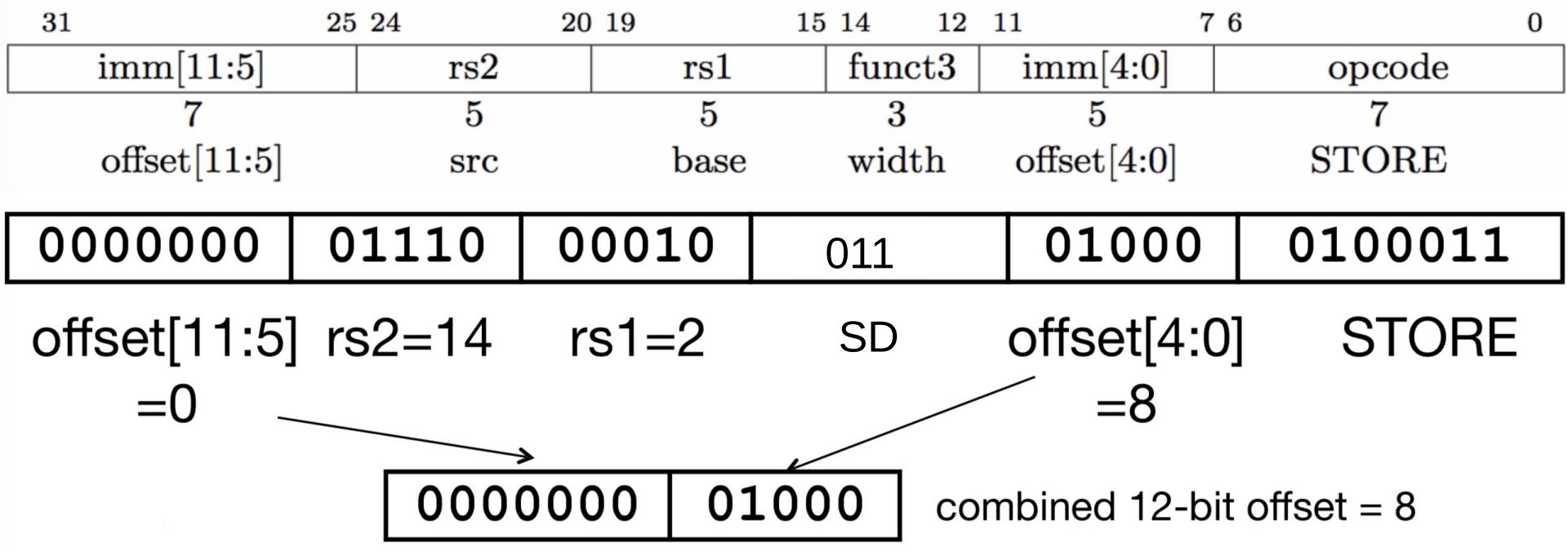


所有的RV64 load指令指令

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
imm[11:0]	rs1	110	rd	0000011	LWU
imm[11:0]	rs1	011	rd	0000011	LD

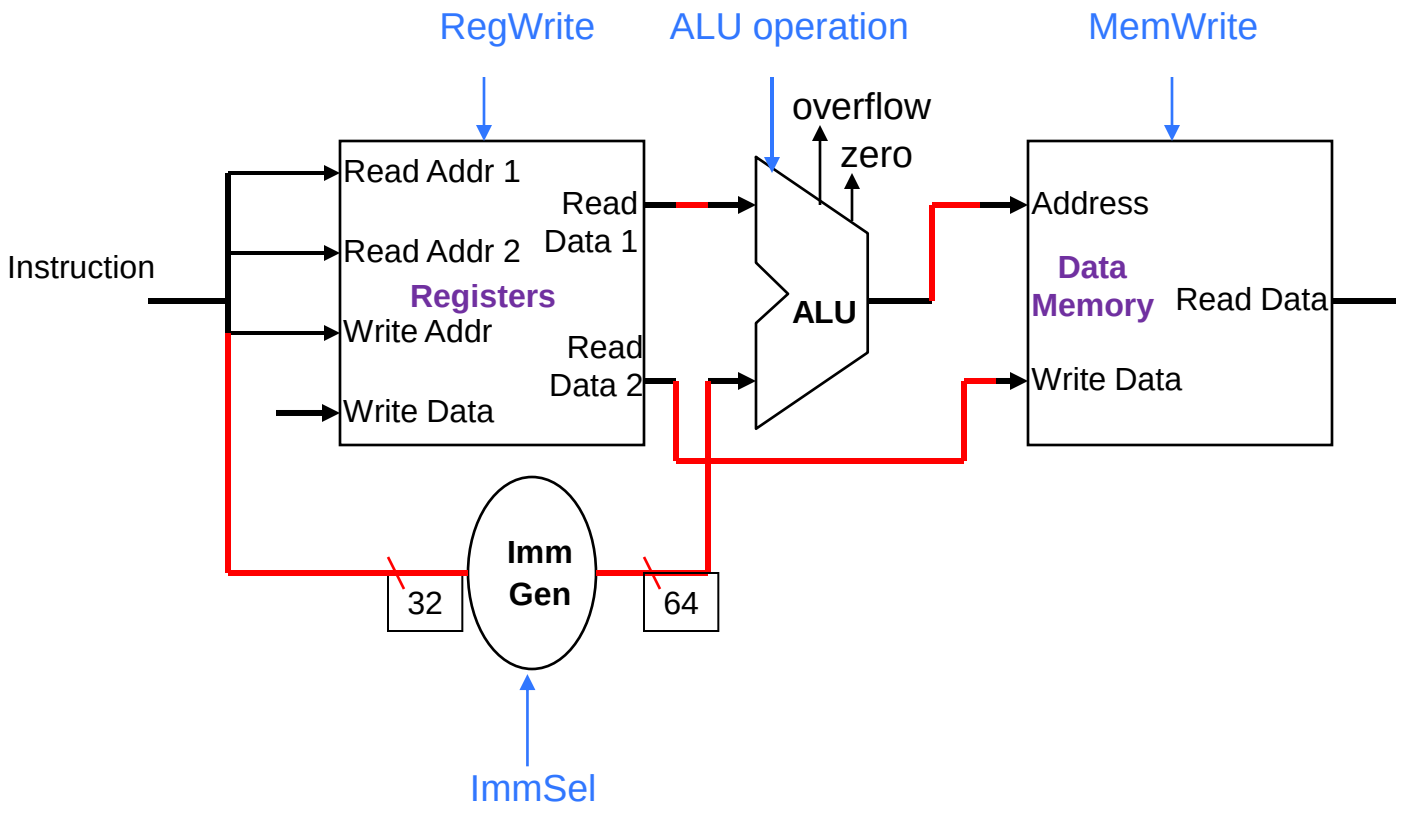
funct3 编码load指令的宽度、是否符号扩展
为什么只有7种情况?

实现 Store指令

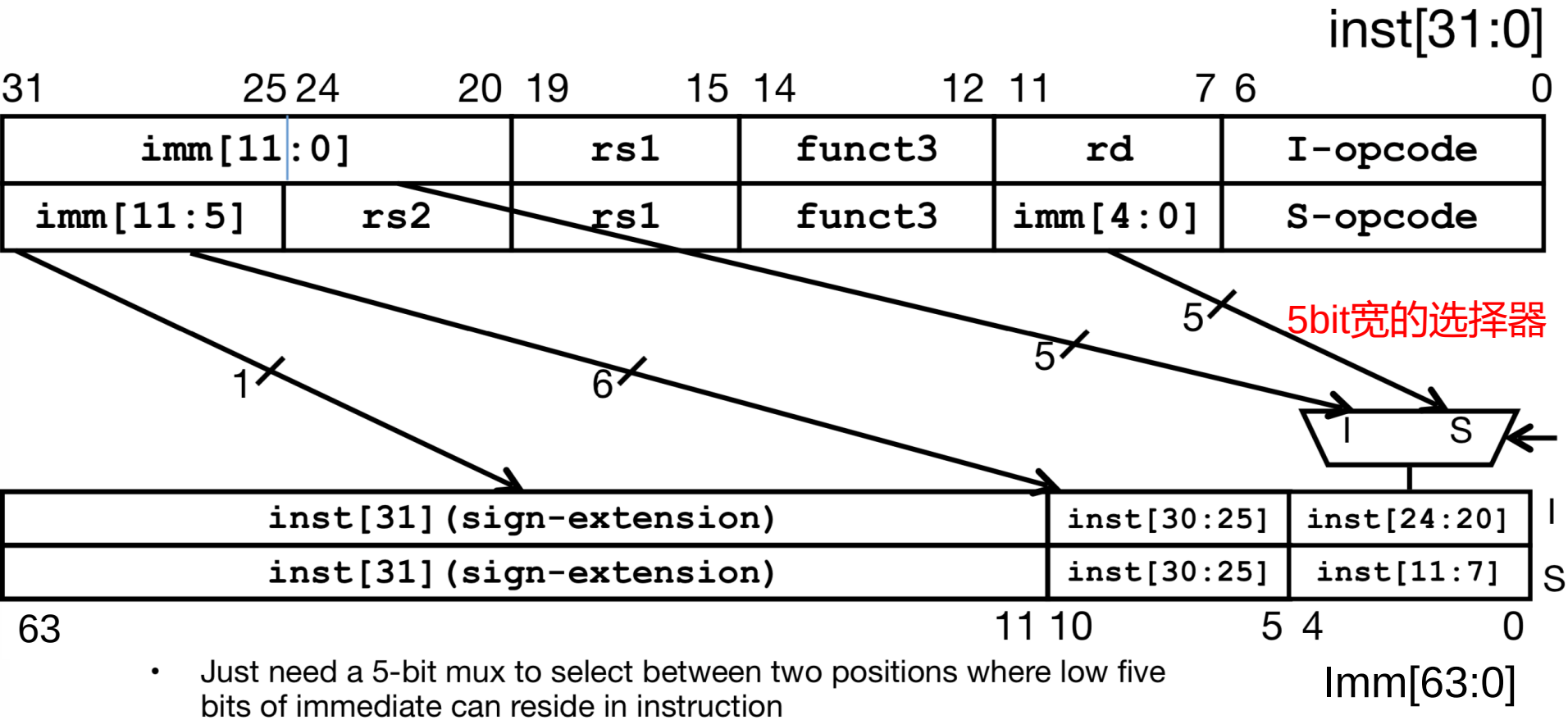


SD x14, 8(x2)

执行Store指令的过程



I型和S型指令的Immediate Generator

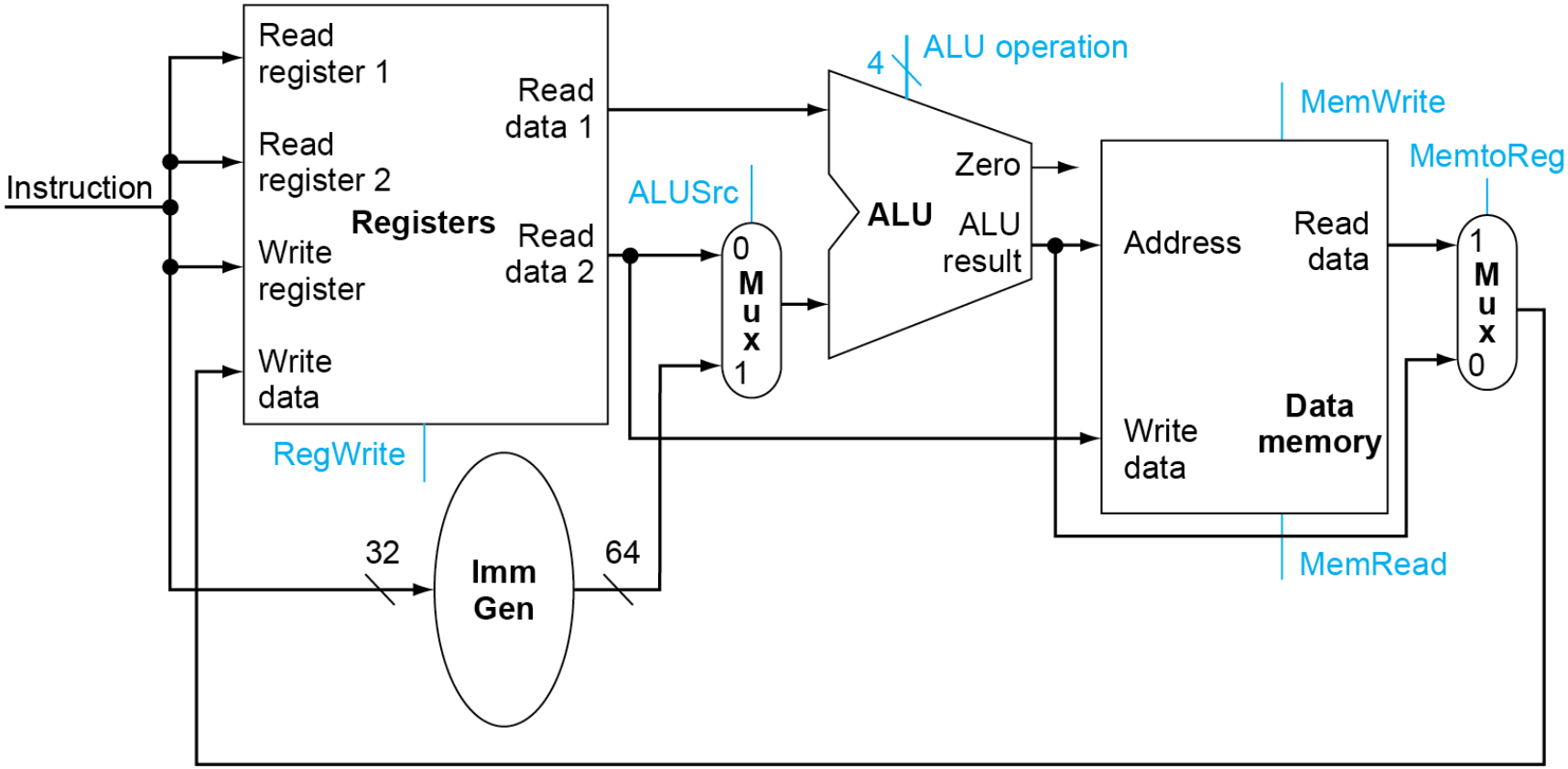


- Just need a 5-bit mux to select between two positions where low five bits of immediate can reside in instruction
- Other bits in immediate are wired to fixed positions in instruction

集成在一起

- 简化版的数据通路，在一个时钟周期内完成一条指令
  - 单个时钟内，每个通路逻辑的只完成一个功能
    - 数据流上没有回边
    - 需要分开的IMem和DMem
  - 当多条指令导致多个数据来源时，需要引入**选择器**

R-型/load/Store 指令的数据通路（图4-10）



练习 1

下列关于load指令的描述，是否正确? 参考 图4.10.

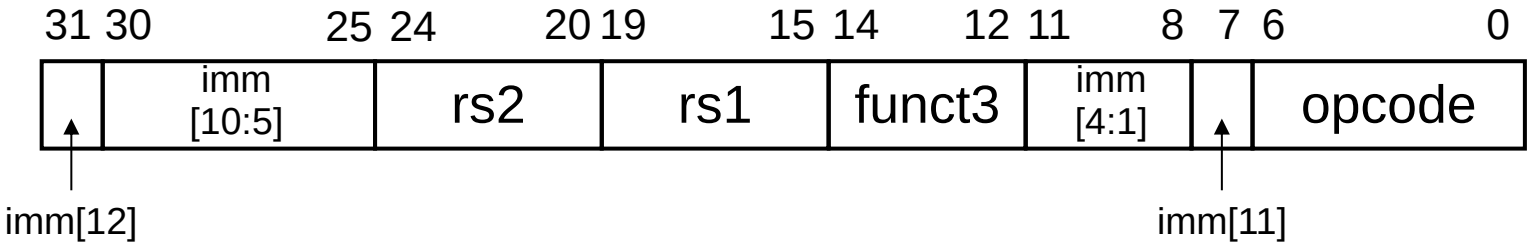
- a. 应该设置MemtoReg，来选择DMem中的数据写回到寄存器
- b. 应该设置MemtoReg，来选择正确的目标寄存器，以便写回
- c. load指令不关心 MemtoReg 控制信号.

练习 2

本节讨论的单周期的CPU数据通路**必须**有分开的IMem和DMem，是因为

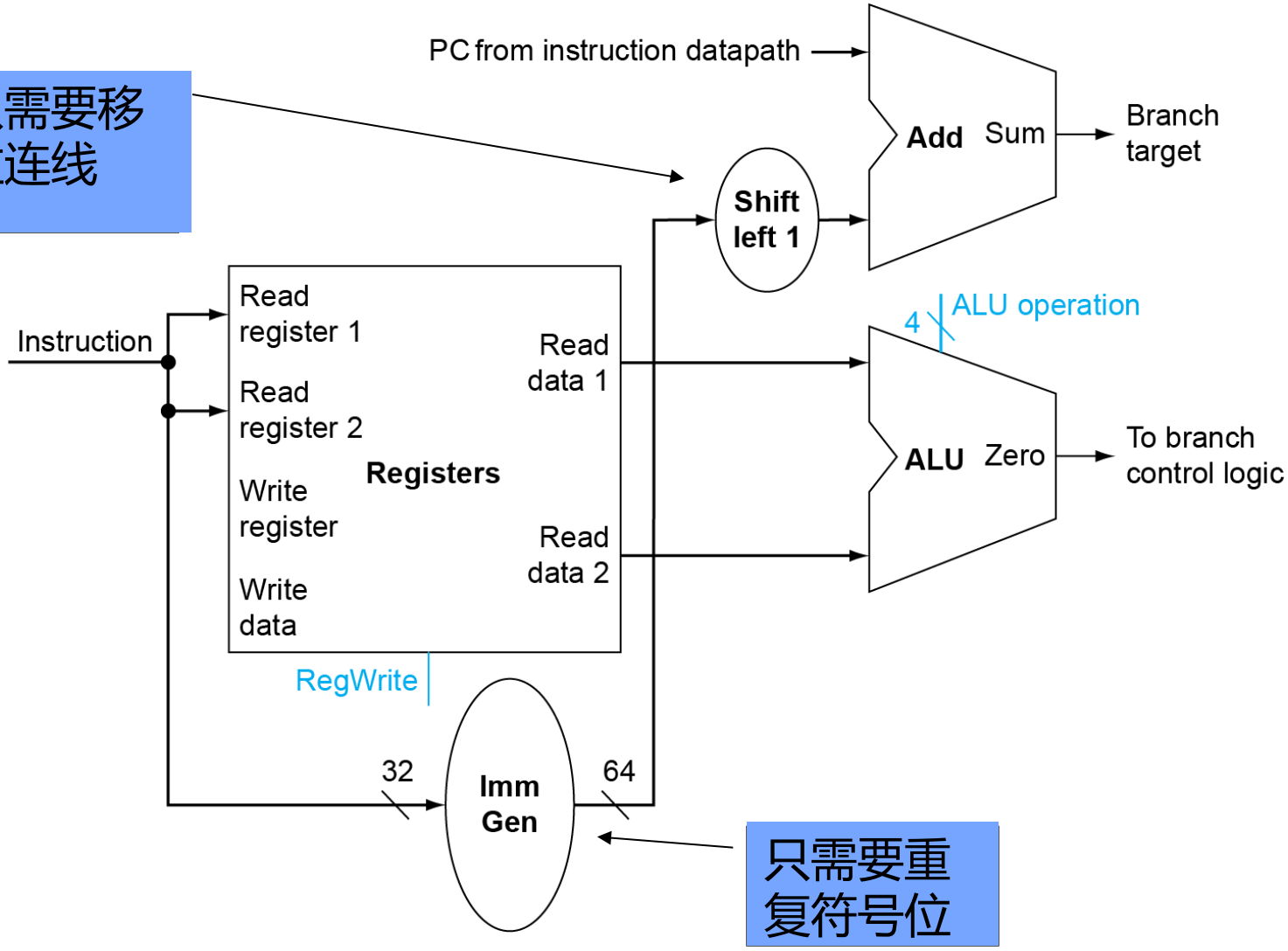
- a. RISC-V中的数据和指令的编码格式不一样，所以需要不同的内存
- b. 使用分开的内存经济上更划算
- c. 在同一个周期内，CPU不能使用同一个（单端口）的内存同时进行两种不同的访问

分支指令

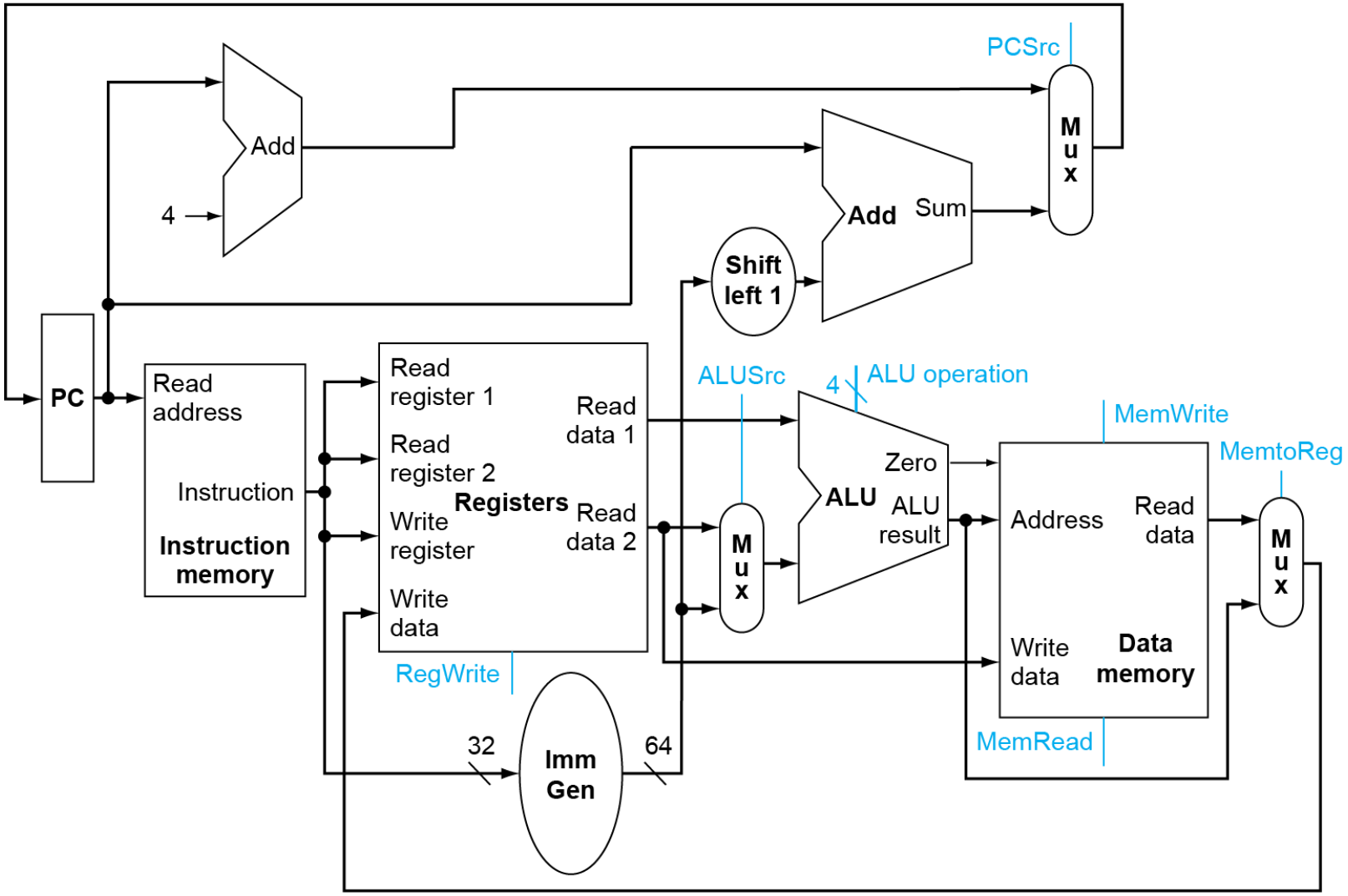


- B-型指令 机会跟 S-型指令一样, 都有两个来源寄存器 (rs1/rs2) 和一个 12-bit 立即数
- 对两个操作数进行比较运算
  - 使用ALU, 先减法运算，再检测 Zero 输出
- 立即数表示范围： -4096 to +4094 （步长为2字节）
- 计算目标地址
  - 先对立即数进行**有符号扩展**
  - 然后立即数左移1位 (恢复成单字节寻址)
  - 最后跟PC值相加

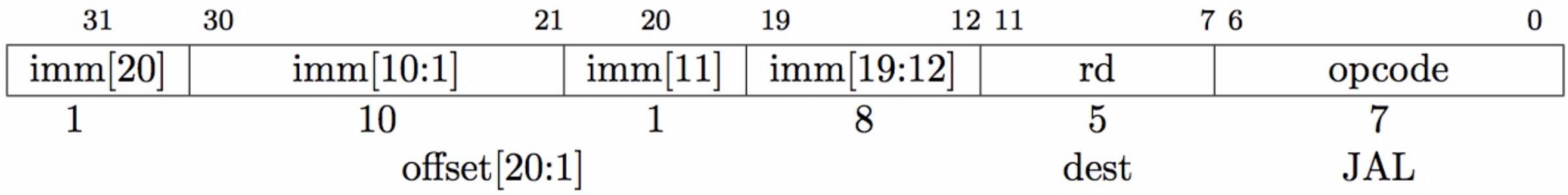
实现beq 指令（图4.9）


只需要移位连线
PC from instruction datapath
Add
Sum
Branch target
Shift left 1
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
4
ALU operation
ALU
Zero
To branch control logic
RegWrite
32
Imm Gen
64
只需要重复符号位

完整的数据通路（图4-11）

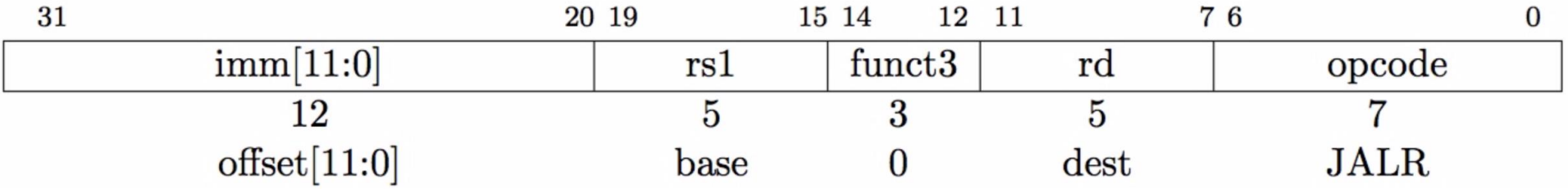


实现JAL 指令(I-型)



- JAL 将PC+4 保存到rd寄存器 (返回地址)
- 将PC 更新为：PC + offset (PC-相对跳转)
  - 跳转目标的范围为：当前地址 $\pm 2^{19}$ 个2-字节地址
  - $\pm 2^{18}$ 条32位宽的指令
  - 跟分支指令beg类似，立即数的编码节省了一个bit位

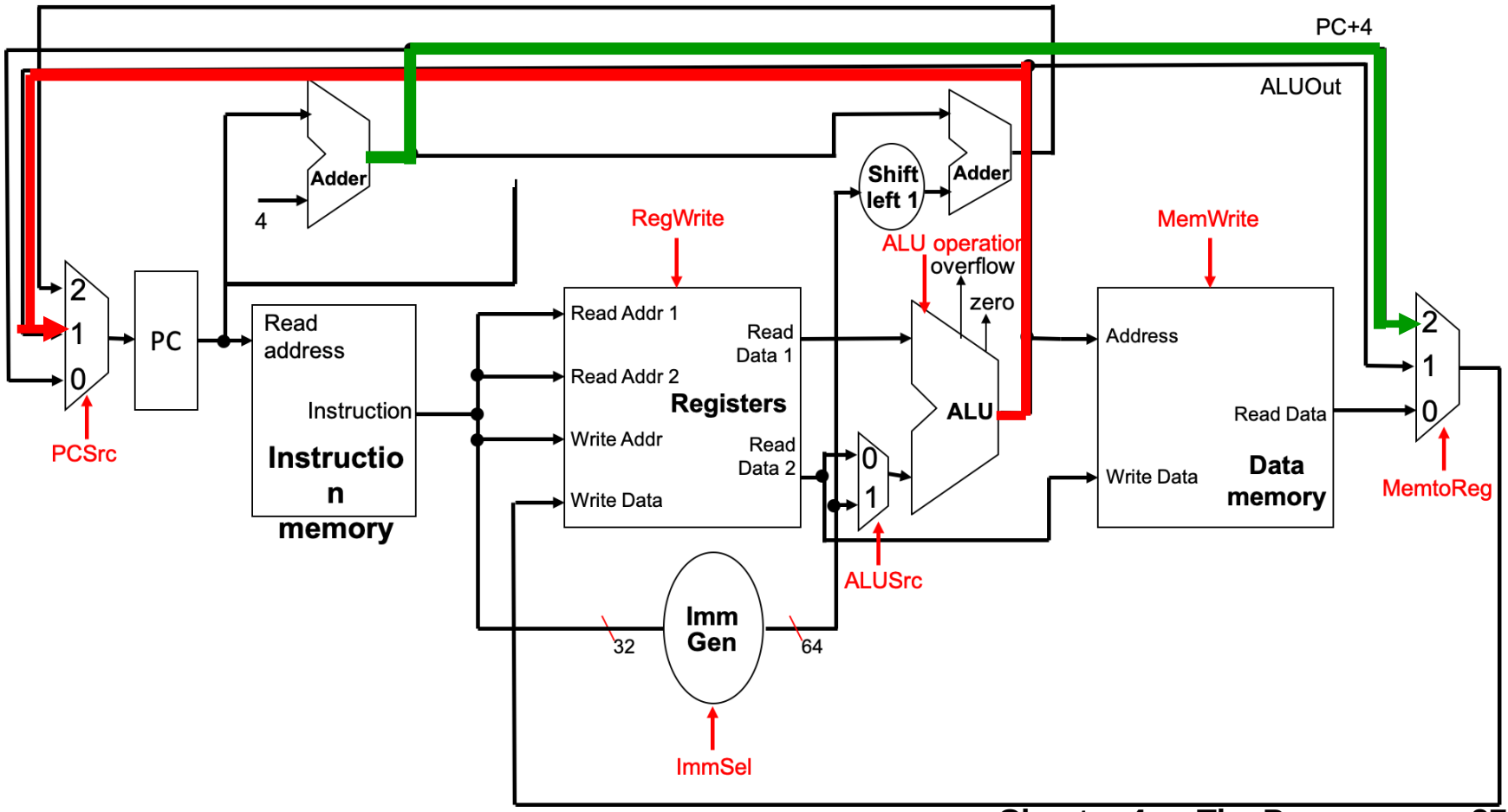
实现JALR 指令(I-型)



- 将PC+4 保存到rd寄存器 (返回地址) ——如果rd不是x0
- 将PC更新为： rs1寄存器的值 + immediate
- 跟算术指令、load指令指令类似，立即数参与运算时
  - ***不需要*** **乘以2**（跟分支指令不同）

单周期 RISC-V RV64I 数据通路

- 可以从Reg更新PC（间接跳转）——PC寄存器第3个来源
- PC+4 可以被保存到一个寄存器——寄存器文件第3个来源



小结

- 统一的数据通路
  - 能够在一个周期内完成任意一条RISC-V指令
  - 每条指令只会用到通路中的特定的部分
- 指令的执行包括5个阶段
  - IF, ID, EX, MEM, WB
  - 大部分指令不会在5个阶段都活跃
- 控制器来规定指令的执行

Outline

- CPU的构成
- 数据通路
- **控制单元**

控制单元

- ALU的控制: 选择一种运算
- 选择器的控制: 选择一个输入
- 状态逻辑的 r/w 控制: 使能状态逻辑
  - 寄存器的 **写** 使能（enable）信号
  - 内存的 **读/写** 使能信号

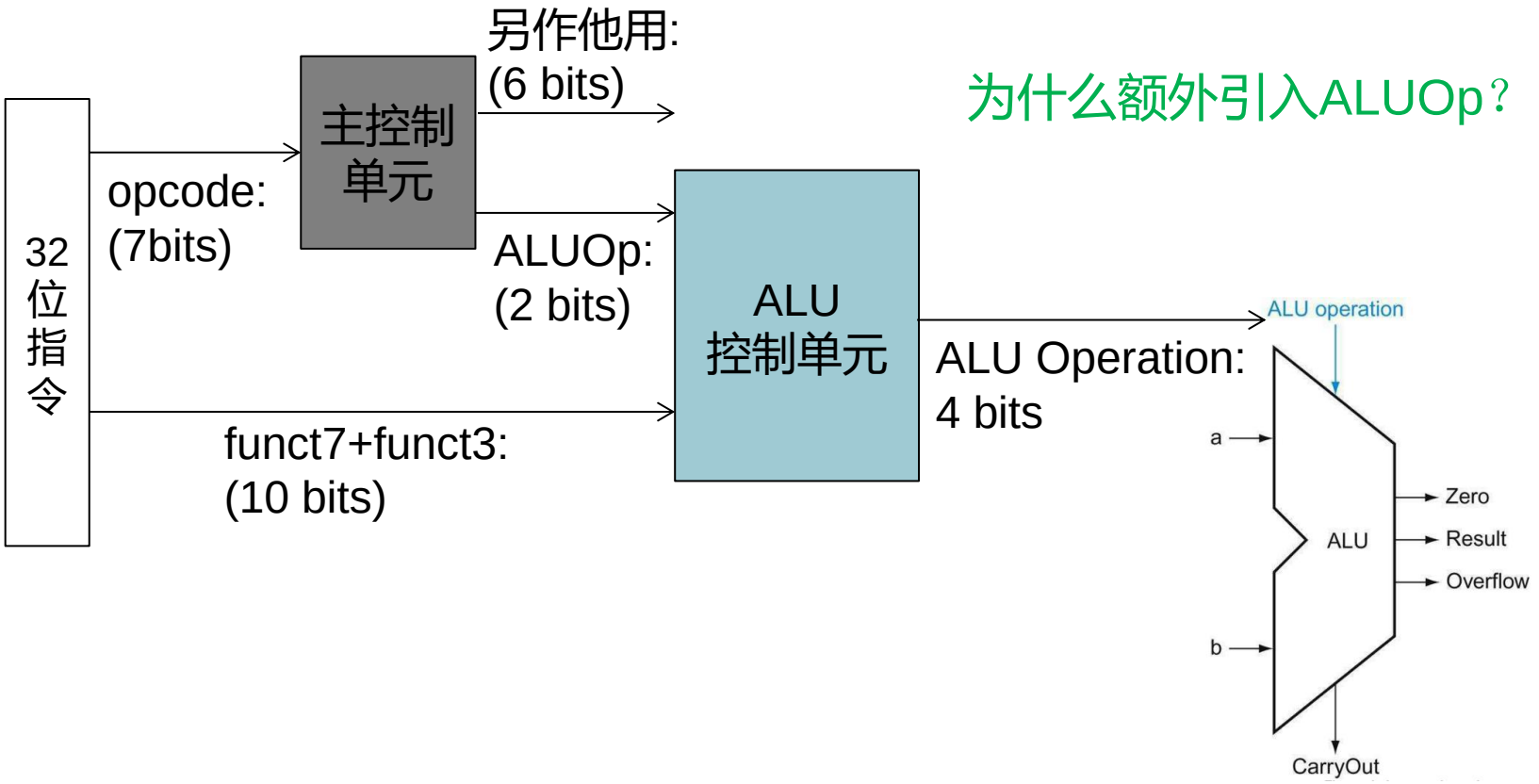
一种简化版的实现方案

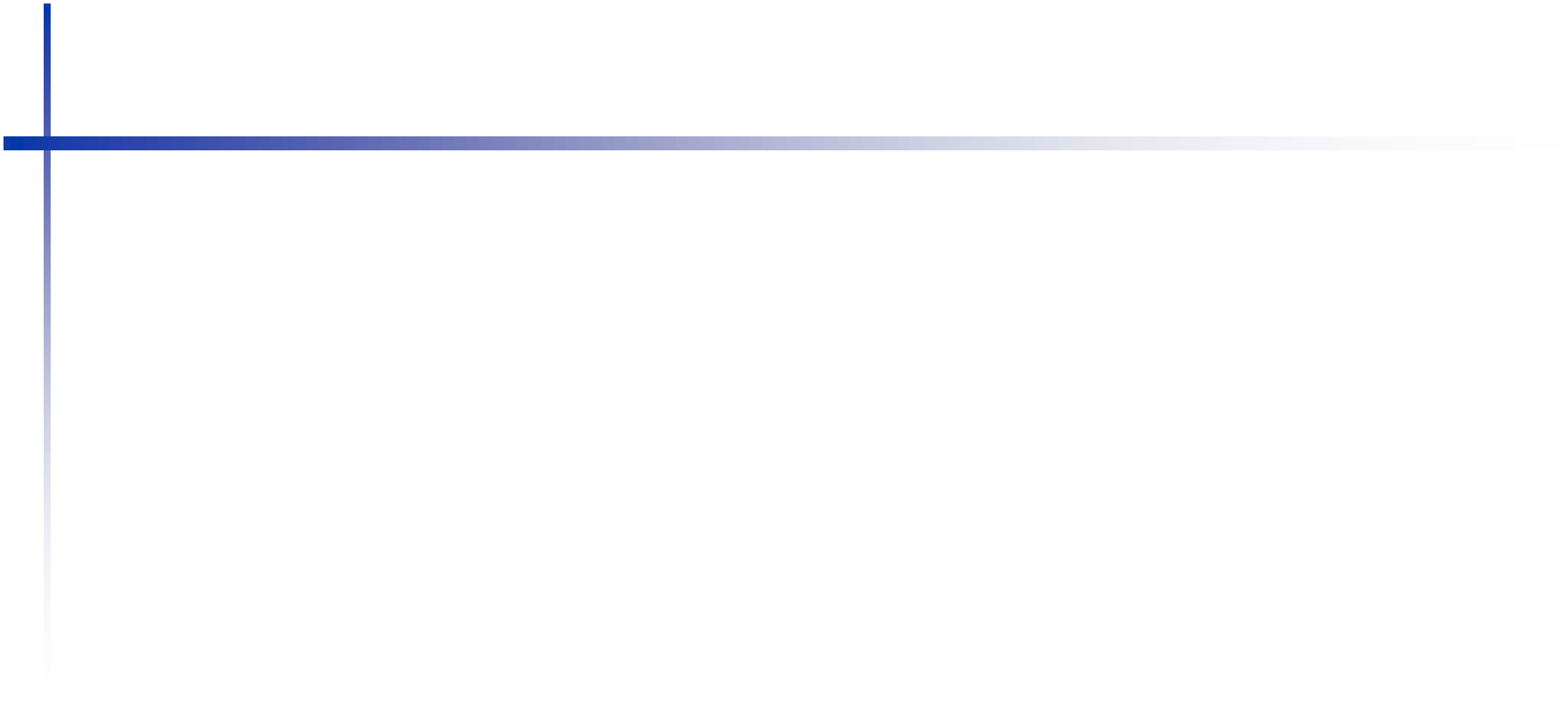
该方案支持以下指令:

- load指令 (`ld`),
- store指令(`sd`)
- 分支指令 (`beq`)
- 算术逻辑指令
  - `add`, `sub`, `and`, and `or`

ALU的控制

- 多级译码：主控制单元（产生ALUOp）和ALU 控制单元
- 降低主控制单元的大小，也可能降低控制单元的延迟。
- 这很重要，因为控制单元的延迟是决定时钟周期的关键因素。





ALU的控制

- 4个bit的ALU控制信号，选择ALU的功能：
  - R-型指令: F由操作码决定
  - load/Store指令: F = add
  - 分支指令: F = subtract

不同指令的操作码 ⇩

ALU operation	Function
0000	AND
0001	OR
0010	add
0110	subtract

load/Store指令 ⇨ 0010

分支指令 ⇨ 0110

ALU的控制（图4.12）

- 从操作码生成 2-bit **ALUOp** 信号
  - 基于ALUOp、指令中的Funct7（bits 31:25) 和Funct3（bits 14:12) 构造组合逻辑，生成 ALU的控制信号
    - **只有R型指令的Funct7 和Funct3影响ALU控制信号**
  - 对ALU Control来说：**蓝色**是输入，**红色**是输出

ld/sd的输入输出完全一样，本质都是寄存器+立即数的add

opcode	ALUOp	Operation	Funct7 field	Funct3 field	ALU function	ALU control
ld	**00**	load register	XXXXXXX	XXX	add	0010
sd	**00**	store register	XXXXXXX	XXX	add	0010
beq	**01**	branch if equal	XXXXXXX	XXX	subtract	0110
R-type	**10**	add	0000000	000	add	0010
		subtract	0100000	000	subtract	0110
		AND	0000000	111	AND	0000
		OR	0000000	110	OR	0001

R-型指令: F由funct7+funct3决定

从真值表到布尔表达式，及无关项优化

实验要做但是考试不考

输入A	输入B	输入C	输出D
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$$D = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$$

基于输入无关项的优化（X代表**无关项**）

输入A	输入B	输入C	输出D
0	X	0	0
0	X	1	1
1	X	X	1

$$D = \bar{A}C + A$$

- 五个或逻辑变成一个

ALU控制单元

- 控制信号由指令编码决定（funct7、funct3字段）

Name (Bit position)	Fields 31:25	24:20	19:15	14:12	11:7	6:0
(a) R-type	funct7	rs2	rs1	funct3	rd	opcode
(b) I-type	immediate[11:0]		rs1	funct3	rd	opcode
(c) S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
(d) SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode

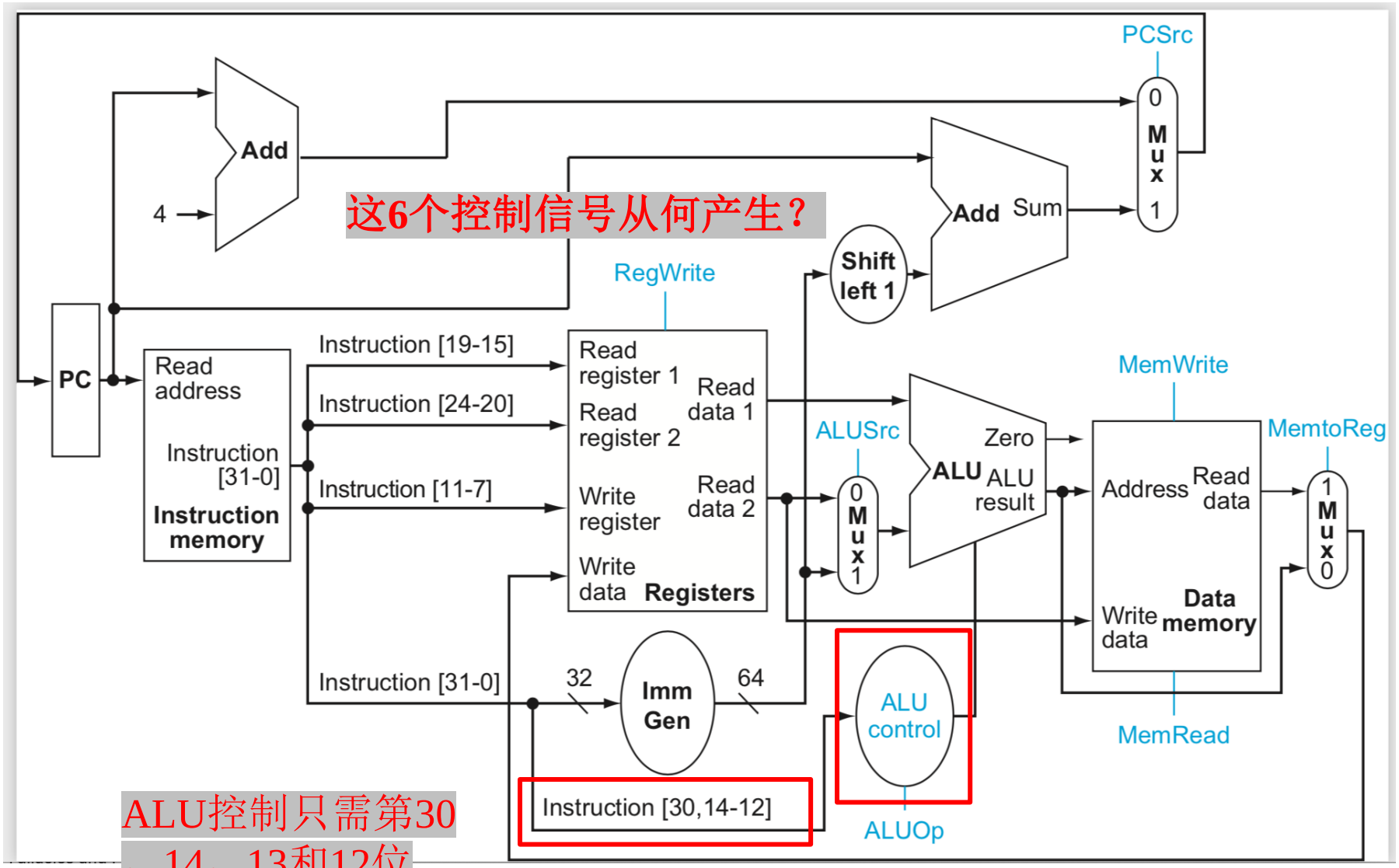
优化1：虽然有10位funct，但四种R型指令，只需要第30，14，13和12位

ld/sd合并

ALUOp		Funct7 field							Funct3 field			
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	Operation
0	0	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	0110
1	X	0	0	0	0	0	0	0	0	0	0	0010
1	X	0	1	0	0	0	0	0	0	0	0	0110
1	X	0	0	0	0	0	0	0	1	1	1	0000
1	X	0	0	0	0	0	0	0	1	1	0	0001

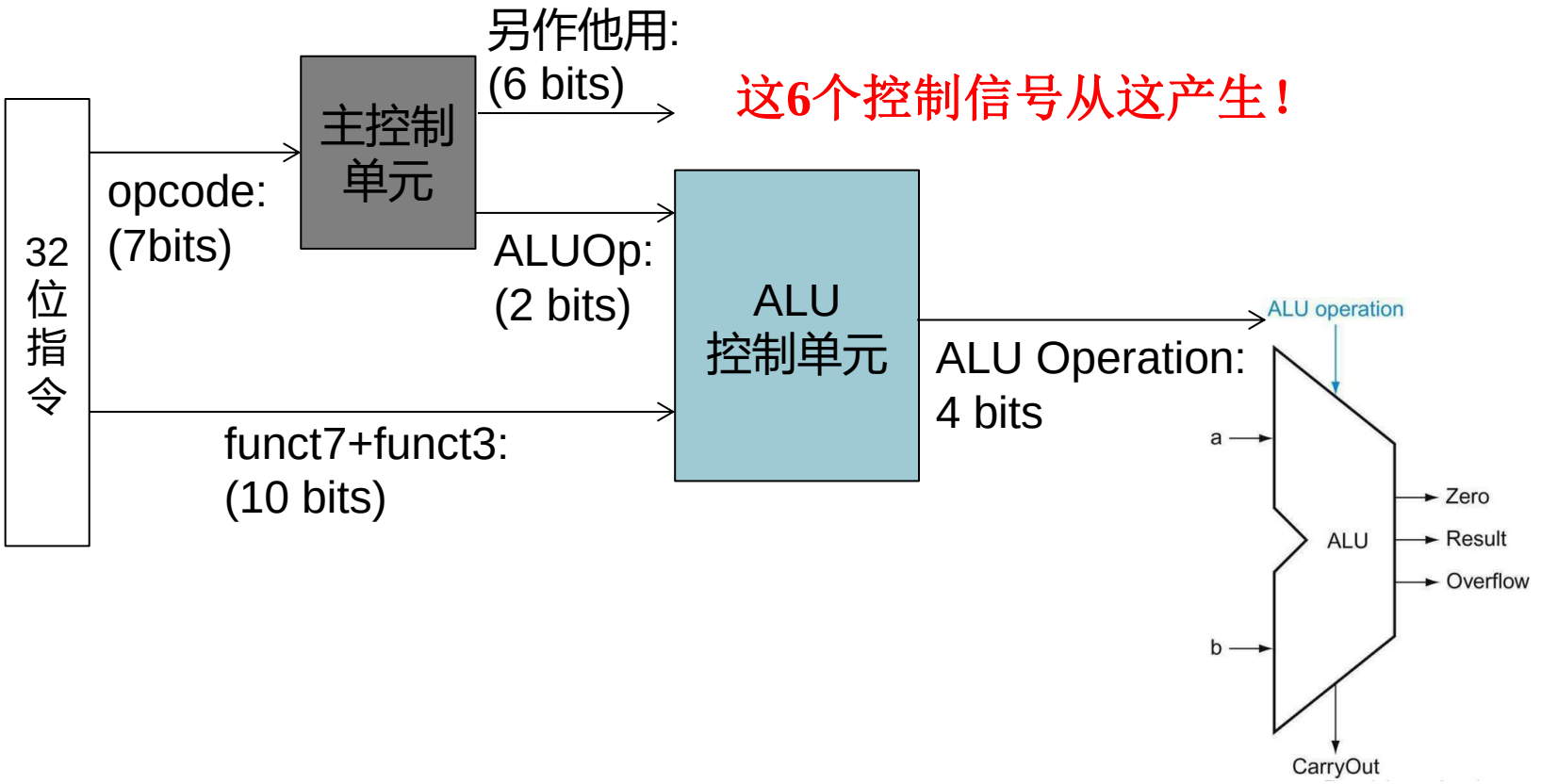
优化2：AULOp中编码11没有被使用过，可以跟01优化成X1，跟10优化成1X

数据通路（包括所有选择器和控制线，图4.19）



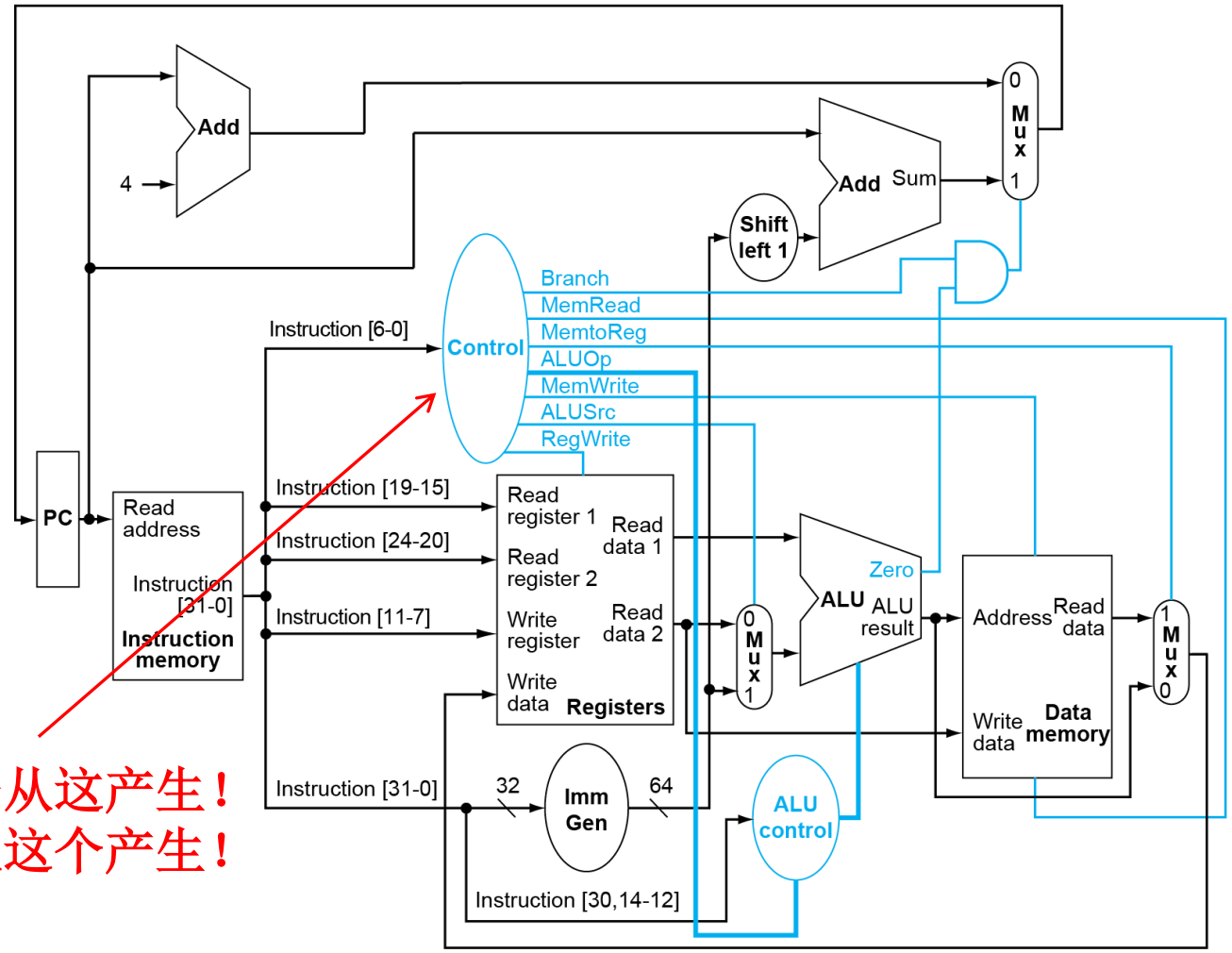
ALU的控制

- 多级译码：主控制单元（产生ALUOp）和ALU 控制单元
- 降低主控制单元的大小，也可能降低控制单元的延迟。
- 这很重要，因为控制单元的延迟是决定时钟周期的关键因素。



带控制线的数据通路

控制线的值只取决于操作码



6个信号从这产生！
ALUOp从这个产生！

6种控制信号的功能（图4-20）

	低电平（或0）时的 功能	高电平（或1）时的 功能
Signal name	**Effect when deasserted**	**Effect when asserted**
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, 12 bits of the instruction.
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

注意：所有状态逻辑都有一个隐含的控制信号——时钟

控制线的值

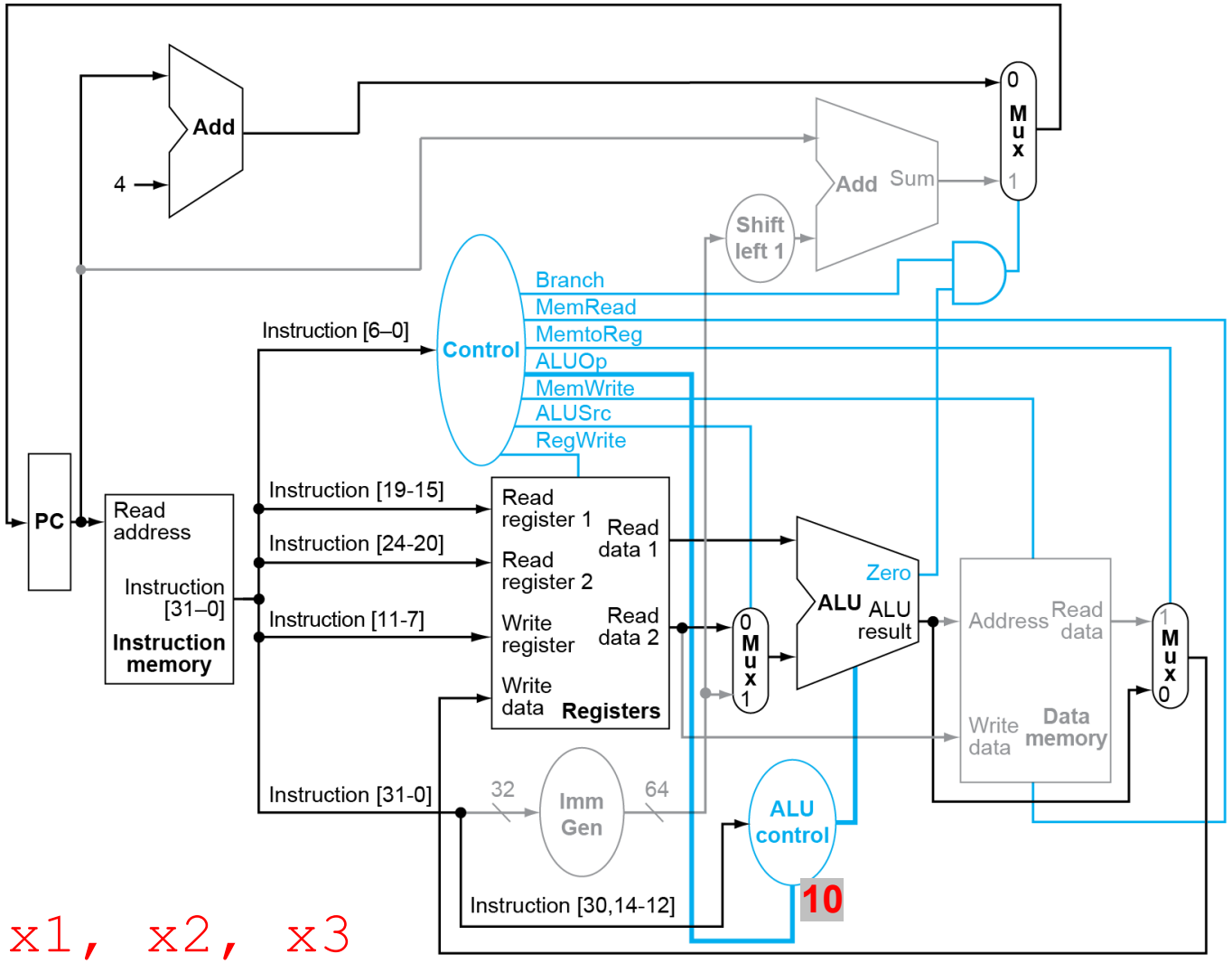
控制线的值只取决于操作码

Instruction	ALUSrc	Memto-Reg	Reg-Write	Mem-Read	Mem-Write	Branch	ALUOp1	ALUOp0
R-format	0	0	1	0	0	0	1	0
ld	1	1	1	1	0	0	0	0
sd	1	X	0	0	1	0	0	0
beq	0	X	0	0	0	1	0	1

控制函数的真值表（图4.26）

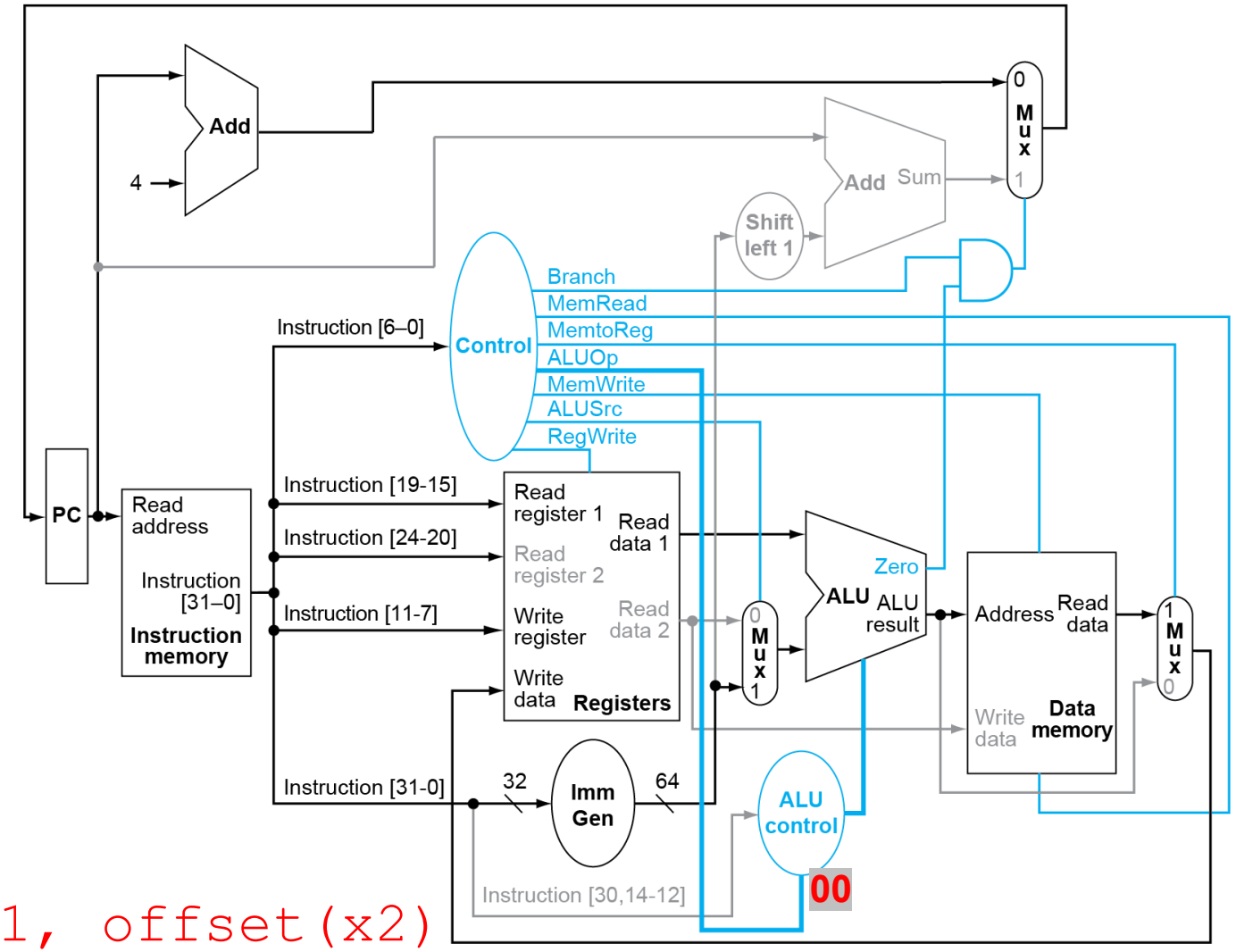
Input or output	Signal name	R-format	ld	sd	beq
Inputs	I[6]	0	0	0	1
	I[5]	1	0	1	1
	I[4]	1	0	0	0
	I[3]	0	0	0	0
	I[2]	0	0	0	0
	I[1]	1	1	1	1
	I[0]	1	1	1	1
Outputs	ALUSrc	0	1	1	0
	MemtoReg	0	1	X	X
	RegWrite	1	1	0	0
	MemRead	0	1	0	0
	MemWrite	0	0	1	0
	Branch	0	0	0	1
	ALUOp1	1	0	0	0
	ALUOp0	0	0	0	1

R-型 指令使用的数据通路及控制信号



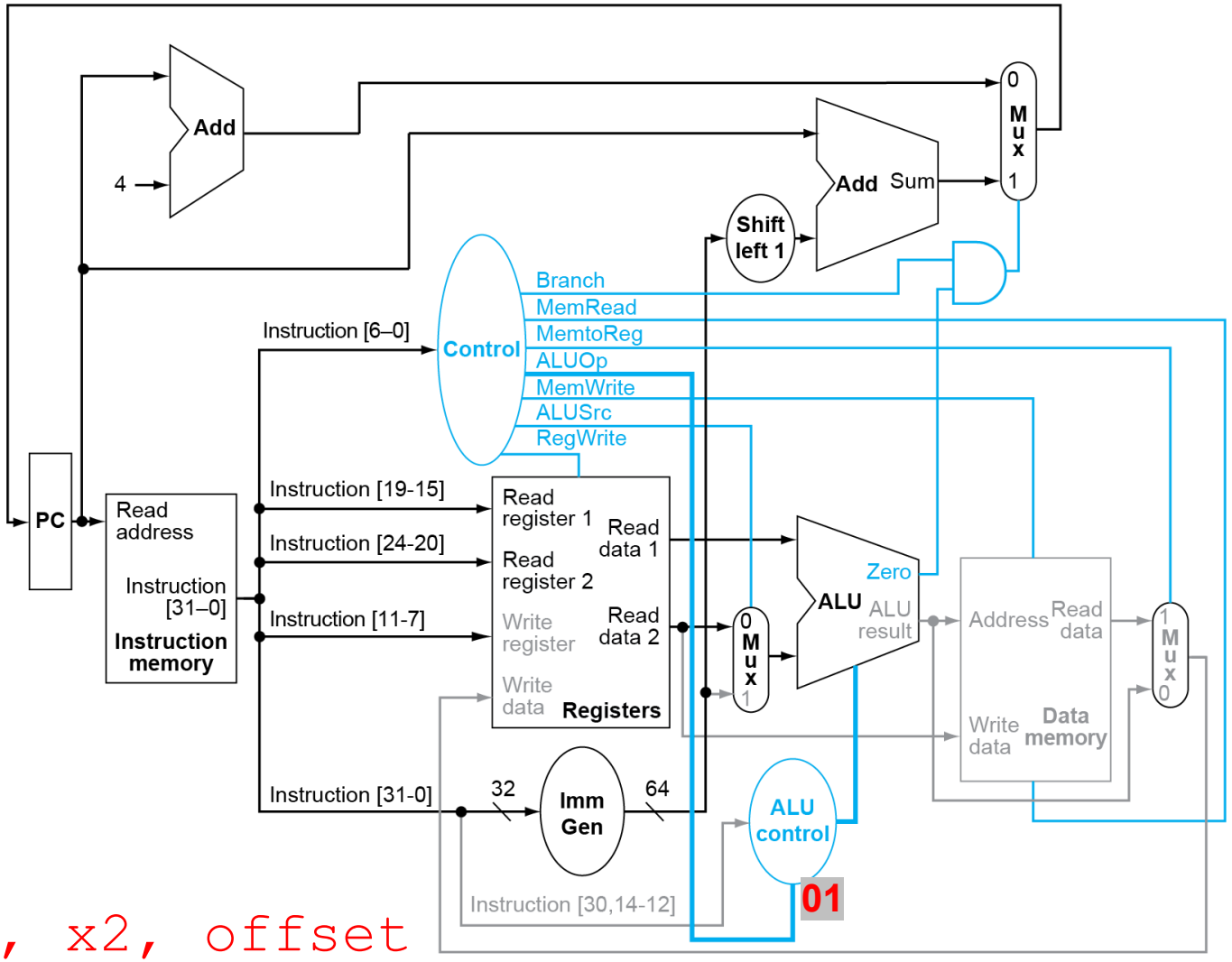

```
add x1, x2, x3
```

load指令 使用的数据通路及控制信号




```
ld x1, offset(x2)
```

BEQ 指令 使用的数据通路及控制信号




```
beq x1, x2, offset
```

本课程需要掌握

- 清楚数据通路的所有部件
- 清楚所有控制信号的用途
- 清楚不同指令的执行过程中
  - 使用到的部件
  - 控制信号的取值

例子：不同指令对CPU通路的使用，及延迟

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

- 最大时钟频率？ （由最慢指令决定）
  fmax = 1/800ps = 1.25 GHz
- CPU通路的利用率如何?
  - 60% ～100%?

性能问题

- Longest delay 决定时钟周期
  - 关键路径: load指令指令，会使用所有5个阶段
  - Instruction memory → register file → ALU → data memory → register file
  - 不同指令使用不同的时钟，在物理实现上是不可行的
- 但是，不同延迟的指令共用时钟周期，是一个次优的方案
  - 那些快速的指令会浪费时间去等待
  - 此时，CPU的某些阶段会空闲
- 如何提升性能?
  - 如果你是一个工厂的负责人，发现每天晚上机器在空闲，你会怎么办?
  - 那么，如何确保ALU 一直处于忙碌状态?

例子：不同指令对CPU通路的使用，及延迟

			执行		Write Back	
Instr	IF=200 ps	ID=100 ps	ALU=200 ps	MEM=200 ps	WB=100 ps	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？ （由最慢指令决定）
  fmax = 1/800ps = 1.25 GHz
- CPU通路的利用率如何?
  - ~~60% ～100%？~~
  - No!!! More like 12%～20%

在大部分时间里，大部分逻辑是空闲的

数据通路	时间线									
	200	100	200	200	100	200	100	200	200	100
iF	load指令					load指令				
ID		load指令					load指令			
ALU			load指令					load指令		
MEM				load指令					load指令	
WB					load指令					load指令

利用率：20%

数据通路	时间线									
	200	100	200	200	100	200	100	200	200	100
iF	add					add				
ID		add					add			
ALU			add					add		
MEM										
WB					add					add

利用率：16%

如何让数据通路一直忙碌?——流水线

空闲的时间线

数据通路

	200	100	200	200	100	200	100	200	200	100
iF	load指令 IF					add指令 IF				
ID		load指令 ID					add指令 ID			
ALU			load指令 EX					add指令 EX		
MEM				load指令 MEM						
WB					load指令 WB					add指令 WB

利用率：18%

忙碌的时间线

数据通路

	200	200	200	200	200	200	200	200	200	200
iF	load指令 IF	add指令 IF				add指令 IF				
ID		load指令 ID	add指令 ID				add指令 ID			
ALU			load指令 EX	add指令 EX				add指令 EX		
MEM				load指令 MEM						
WB					load指令 WB	add指令 WB				add指令 WB

利用率？

一份小学生的学习计划表

学习计划表

时间段	周一至周五	周六	周日	备注
6：40	起床洗漱，换衣服，梳头	8:00 起床洗漱、吃饭	8:00 起床洗漱、吃饭	
6：50	吃早餐	9:00 写作业 2 小时	9:00 写作业至全部完成	
7：10	练习 10 分钟钢琴	11:00 休息 1 小时	11:00 休息 1 小时	
7：20	出发上学	12 点吃午饭	12 点吃午饭	
11：40	放学到家休息 10 分钟	1 点午休 2 小时	1 点午休 2 小时	
11：50	练习钢琴 30 分钟	3 点练习钢琴半小时	下午时间自由支配	
12：30	午餐	4 点钢琴课		
13：00	午睡	下课后带着玩		
14：20	起床上学	7 点晚饭时间		
17：30	放学到家休息 10 分钟	8 点写作业 30 分钟		
17：40	开始写家庭作业	8:30 以后自由时间		
18：20	晚饭	10:00 睡觉	9 点睡觉	
18：40	继续写作业，交家长检查			
20：30	完成作业，休息 10 分钟			
20：40	练习钢琴半小时			
21：10	休息 20 分钟			
21：30	洗漱上床，阅读半小时睡觉			

说明：1、以上所有的项目达标每天可奖励 1 元钱作为零花钱。2、如作业和练琴不能按时完成扣除零花钱一元。3、如果表现良好，不需要父母督促，所有的作业都尽力的去完成。主动提出问题，可得到适当的额外奖励。4、如一周内每天都达标，周末奖励一次看电影或游玩一次。5、如各项目可提前完成，节省的时间可以自由支配（看电视玩电脑）。

网传清华学霸的学习计划表

第　周计划	周一	周二	周三	周四	周五	周六	周日
06:00-06:40	起床、锻炼、早饭	起床、早饭	起床、早饭	起床、早饭	起床、早饭	班级组织春游	起床、早饭
06:40-08:00	预习几代 2、微 3	复习微 2，大物作业	复习微 2，微 3	微 2，努力钻研	复习、整理、打课件		大物自习
第一节	几代 2 6C101	读飘、背单词	微 3，一教 101	微 2	大物　三教 2302		大物自习
第二节	微 3 一教 101	微 2 西阶 0	大物　三教 2302	微 2，11:10 去吃饭	程设　四教 4203		大物自习，读飘
11:25-13:30	午饭 13:10 系馆王老师	打印课件，午休	回寝取东西，自习	11:30 去 6#312B 找张导	午饭，自习微 2，大物		午饭节俭，回寝社工
第三节	复习几代 2，完成作业	体育，东操	英语高级口语 6B204	班会新水 329、买练习册	自习		准备思源，英语 presen
第四节	复习微 3，完成作业	史纲　6A016	自习！自习！	微 3 习题集 4306	程设上机 东楼 9-224		思源笔试：15:30-17:30
第五节	微 3 习题	晚饭，复习微 2 作业	晚饭，自习	晚饭，自习大物	晚饭，自习大物	zj 园尹老师谈话	晚饭，自习
第六节	微 3 习题	小说　三教 2301	清女 6A103	微 2 习题课 4305	自习大物	自习大物完成	自习到 21:30 去跳绳
20:00-22:30	系统复习大物书作业、习题	微 2 作业，洗澡	校会文化部例会	自习大物（静心）	自习到 9：：3 回 zj 长绳	自习微 2，英语	听 CNN 英语
22:30-23:00	听 CNN 英语	听 CNN 英语	听 CNN 英语	听 CNN 英语	听 CNN 英语	社工：文化部（信息简报，会议记录）自习	
23:00-01:00	读飘，背单词社工	自习，追求高效	自习，坚决完成任务	自习，平心静气	自习，用电脑社工，sleeping	社工（女排 photos 今日 photos，女生节）	
01:00 以后	sleeping	sleeping	sleeping	sleeping	sleeping	准备思源，英语 presen	sleeping
计划完成情况	高效	加油！	高效、专注	微 2 大物完成	微 2 大物；全部社工	抓紧时间	收尾，规划学习
学习情况	认真对待大物	复习微 2，大物	微 2 要钻研	加油！	静心，思考	完成任务	思源加油！
社会情况	思源做毕，微 3 作业	美贺姐姐	例会，发春游稿	给素拓起好名	素拓宣传、校会、宣委	完成任务	我相信我是最棒的！
体育锻炼	不要求	认真上体育课	————	————	长绳跳好	————	跳好长绳
生活状态	昂扬，惜时	积极平和	积极，内敛	积极，平和	积极、平和	加油啊	A
修养品行	礼、德、才	外柔如水内刚似火	知我者，谓我心忧，	为青春喝彩！	多思、少言、必行	A	A

- 科技越发达，好像人越忙?
  - keep **machines** as busy as possible，and
  - keep **human beings** as busy as machines
- So what?