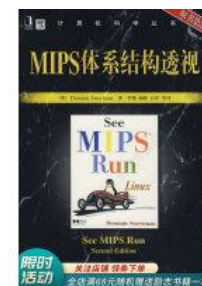
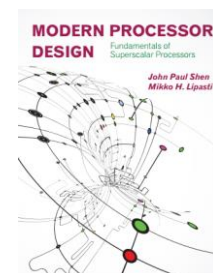
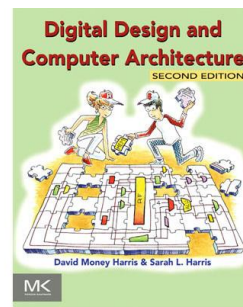


The RISC-V Processor Implementation: Pipeline——ILP

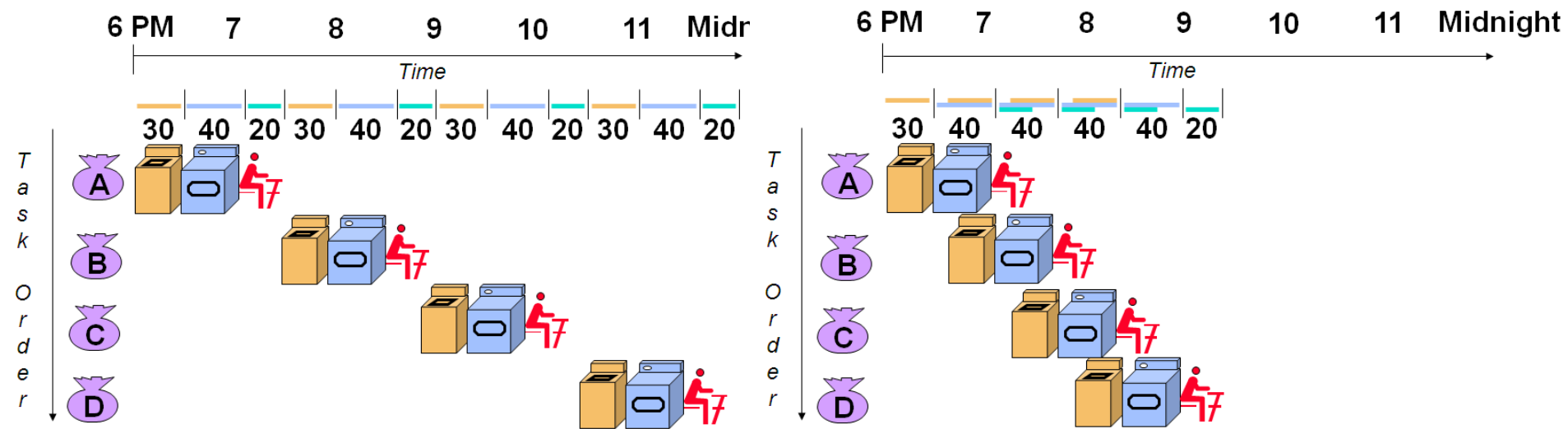
**“Computer Organization & Design ”
David Patterson, John Hennessy**

内容提要

- 流水线技术原理：4.6
- RV的五级流水线实现：4.6.1, 4.7
 - Verilog行为级定义：4.14
- Hazard问题：4.6.2
 - 结构冲突：哈佛结构
 - 数据依赖：4.8
 - 编译技术：插入nop, 指令重排, 寄存器重命名
 - forwarding技术：RAW
 - Interlock技术：Stall
 - 控制相关：4.9
 - 编译技术：延迟分支
 - 硬件优化：提前完成, 投机, 预测
 - 异常：4.10
- 多发射技术：4.11
- 硬件多线程：6.4
- 多核：6.5

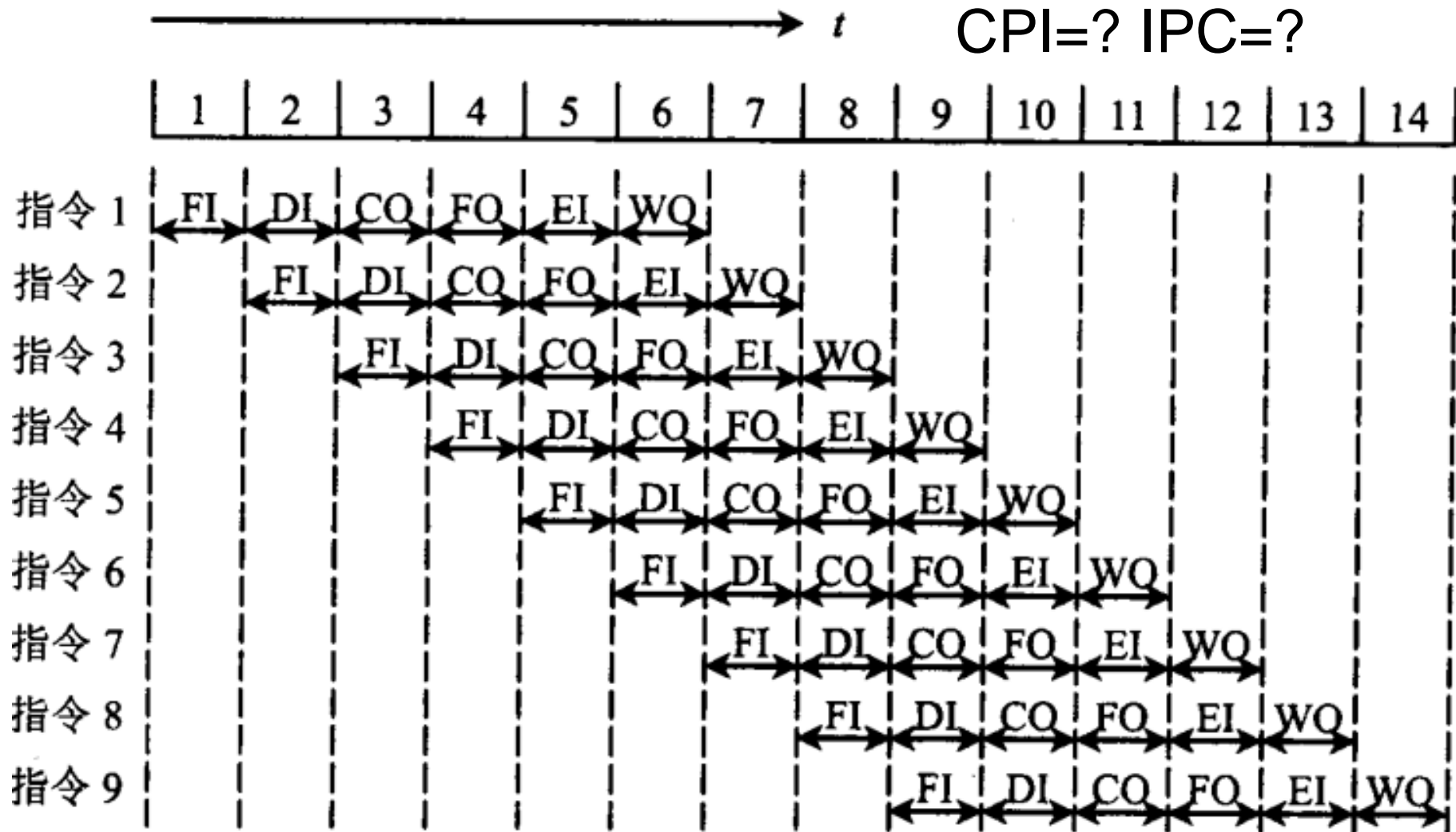


Laundry Example: 类图4-23



- 4 袋衣服: **Wash (30)** , **dry (40)** , **fold (20)**
 - Sequential Laundry: 需要 6 小时: $4 \times (30+40+20)=360$ 分钟 😞
 - Pipelined Laundry: 需 3.5 小时: $30+4 \times 40+20=210$ 分钟 😊
 - **前提:** 1) 过程可分解, 2) 设备独立 (单缸/滚筒, 双缸?), 3) 作业等时
 - **贪心:** Start work ASAP!
- ppl目标: 提升“吞吐率”, 缩短“程序完成时间”。IPC? **CPI?**
 - 铁律: 一袋衣服的完成时间? n袋衣服的完成时间?
- CISC顺序模型, RISC指令级并行模型 (ILP)

六级指令流水线时空图 (Gantt chart)



理想流水线性能

- CPI (μA的效率), IPC (吞吐率), 加速比
 - 单周期, 多周期, ppl?

$$\text{speedup} = \frac{\text{非流水线执行时间}}{\text{流水线执行时间}} = \lim_{n \rightarrow \infty} \frac{n * k * \Delta t}{k * \Delta t + (n - 1) * \Delta t} = k$$

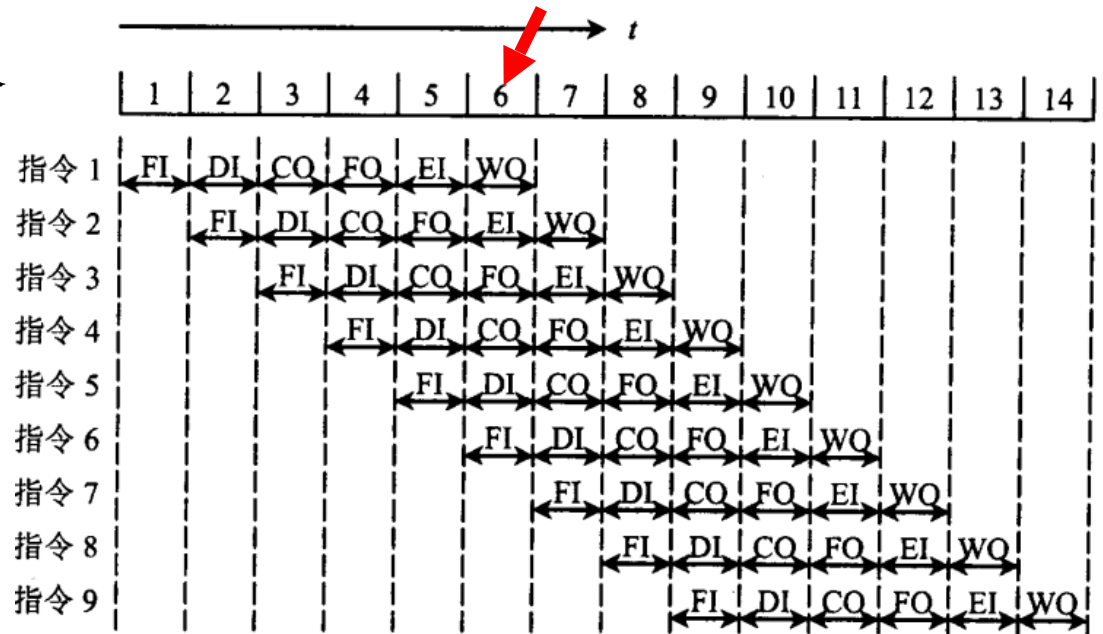
理想流水线: 从CC6开始, 每个周期流出一条指令

因此, $\text{CPI} = \text{IPC} \approx 1$ 。

- 每条指令的指令周期没变
- 整体性能 (程序执行时间) 提升

最坏情况? 最好情况? $\text{IPC} \geq 1$?

PPL越深越好?



流水线分类

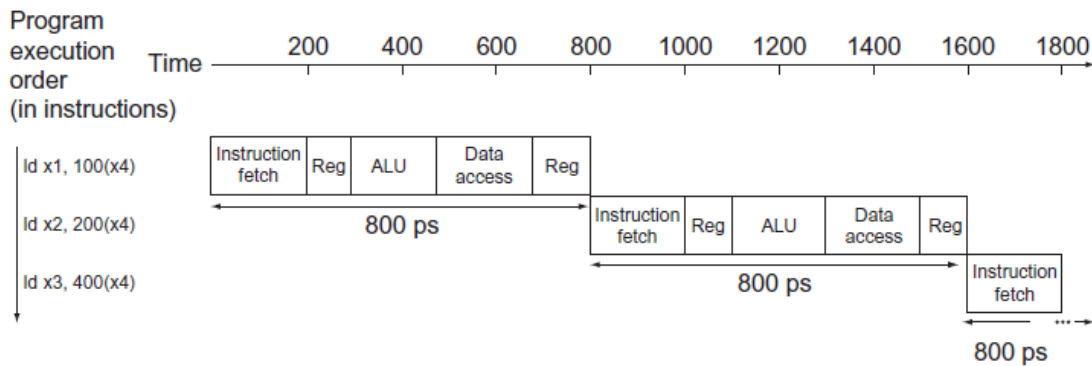
- 单功能流水线：只能完成一种功能的流水线，如浮点加法流水线。
- 多功能流水线：流水线的各段可以进行不同的连接，从而使流水线在不同的时间完成不同的功能。
- 静态流水线：在某一时间段内，流水线的各段只能按同一种功能的连接方式工作，即只有当输入是一串相同性质的操作时其性能才能得到发挥。
- 动态流水线：在某一段时间内，某些段正在实现某类操作（定点乘），其他段却在实现另一类操作（浮点加）。
- 顺序流水线（In-order）：流水线的流出顺序与其流入顺序相同。
- 乱序流水线（OoO）：流水线的流出顺序与其流入顺序不同。
- 静态调度流水线：不对指令执行顺序进行重排序。
- 动态调度流水线：对指令执行顺序进行重排序（“指令调度”）。

RV指令流水线技术

图4-24

Instruction class	Instruction fetch	Register read	ALU operation	Data access	Register write	Total time
Load doubleword (ld)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps
Store doubleword (sd)	200 ps	100 ps	200 ps	200 ps		700 ps
R-format (add, sub, and, or)	200 ps	100 ps	200 ps		100 ps	600 ps
Branch (beq)	200 ps	100 ps	200 ps			500 ps

- 分解指令执行步骤
 - 流水线“级”/“段”
 - 每个流水段所需的时间
 - 一个时钟周期(机器周期)
 - 流水线深度：段数
 - 深好浅好？



• Pipelining Idealism

- 各段均衡 (latency)
 - 执行时间长的流水段将成为瓶颈，造成block/stall
- 重复同样的指令
 - 单功能流水线
- 指令流无依赖

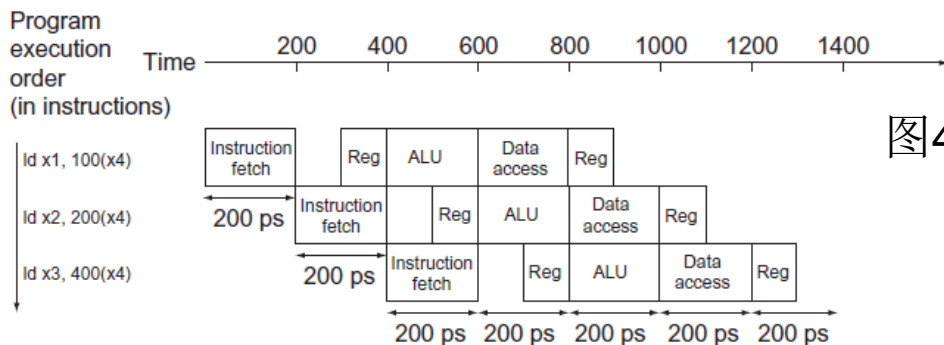


图4-25

面向流水线的ISA: RV的4特点 (\$4.6.1)

- 指令字等长
 - 适于取指和译码
- 指令格式简单规则
 - 操作数位置固定: 可以在译码的同时取opr, 否则要增加一级流水段?
- 只有load/store访存
 - 可以在执行阶段计算访存地址, 然后在下一阶段访存。
 - 如果R-type指令也可访存?
- 每条指令最多只写一个结果, 且在最后一级进行 (WB)
 - 利于前推 (\$4.6.2), load-Rtype竞争 (race, 结构冲突)
 - 利于异常处理 (\$5.7.9): 缺页异常须指令可重启
- 寻址方式简单: 间址寻址、堆栈寻址?
- 操作数在内存中“字对齐”——COD MIPS版, RV/ARM无?
 - 因此在取操作数时不需要访问存储器多次

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

RISC-V指令格式与寻址方式

图4-16

Name (Field size)	Field						Comments
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

- 指令字等长
- 指令格式规整：6种
 - Reg和Imm位置固定
 - 指令类型与格式绑定
 - 寻址方式：3种
 - Imm, Reg, Base
- load/store架构

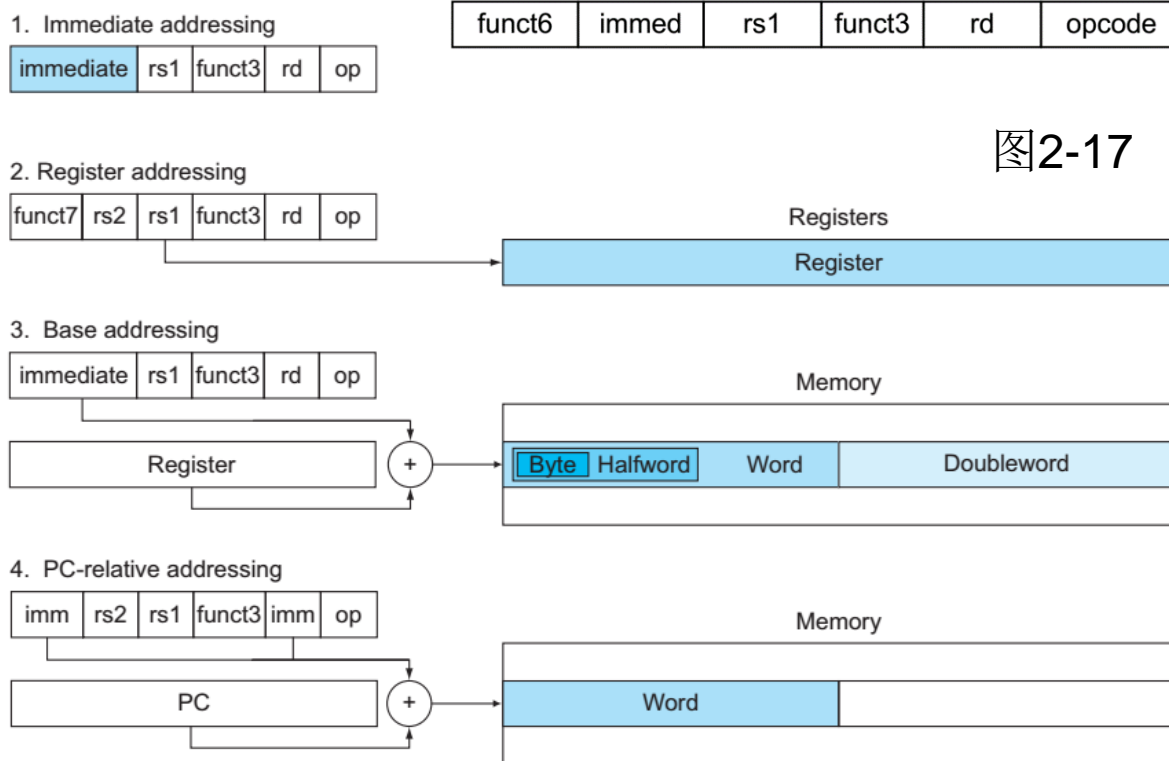
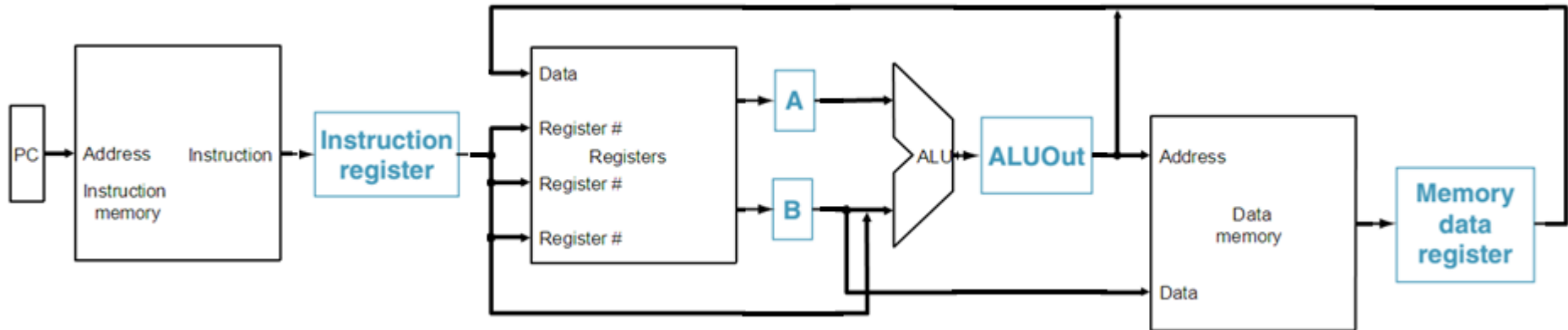
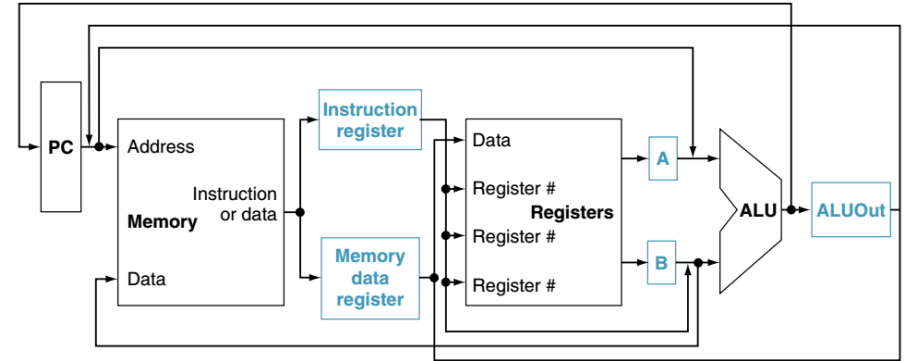
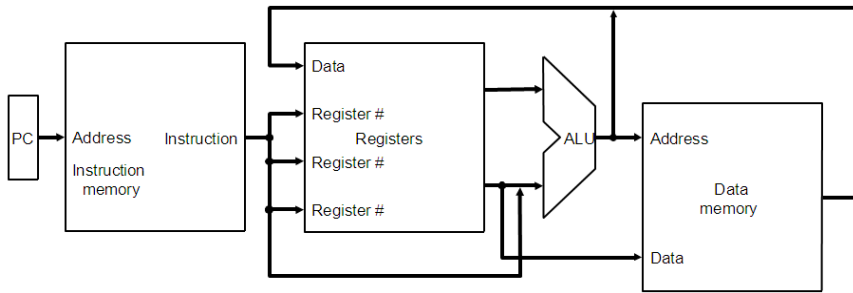


图2-17

流水线数据通路？

Instruction number	1	2	3	4	5	6
Instruction i	IF	ID	EX	MEM	WB	
Instruction $i+1$		IF	ID	EX	MEM	WB
Instruction $i+2$			IF	ID	EX	MEM
Instruction $i+3$				IF	ID	EX
Instruction $i+4$					IF	ID



- 暂存，复用？

指令数据通路划分：执行时间平衡

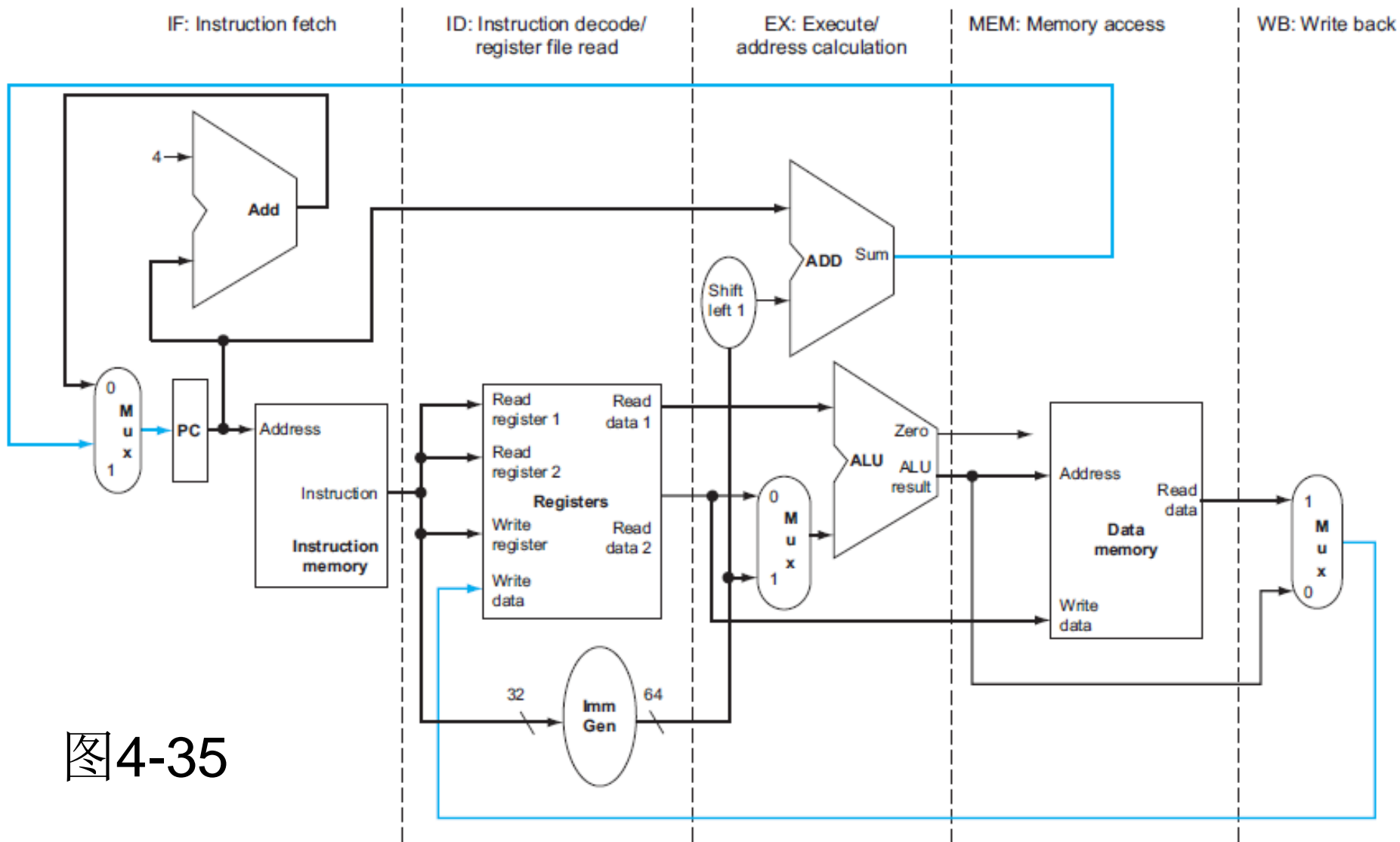


图4-35

流水线段寄存器

- 隔离流水段，暂存中间结果

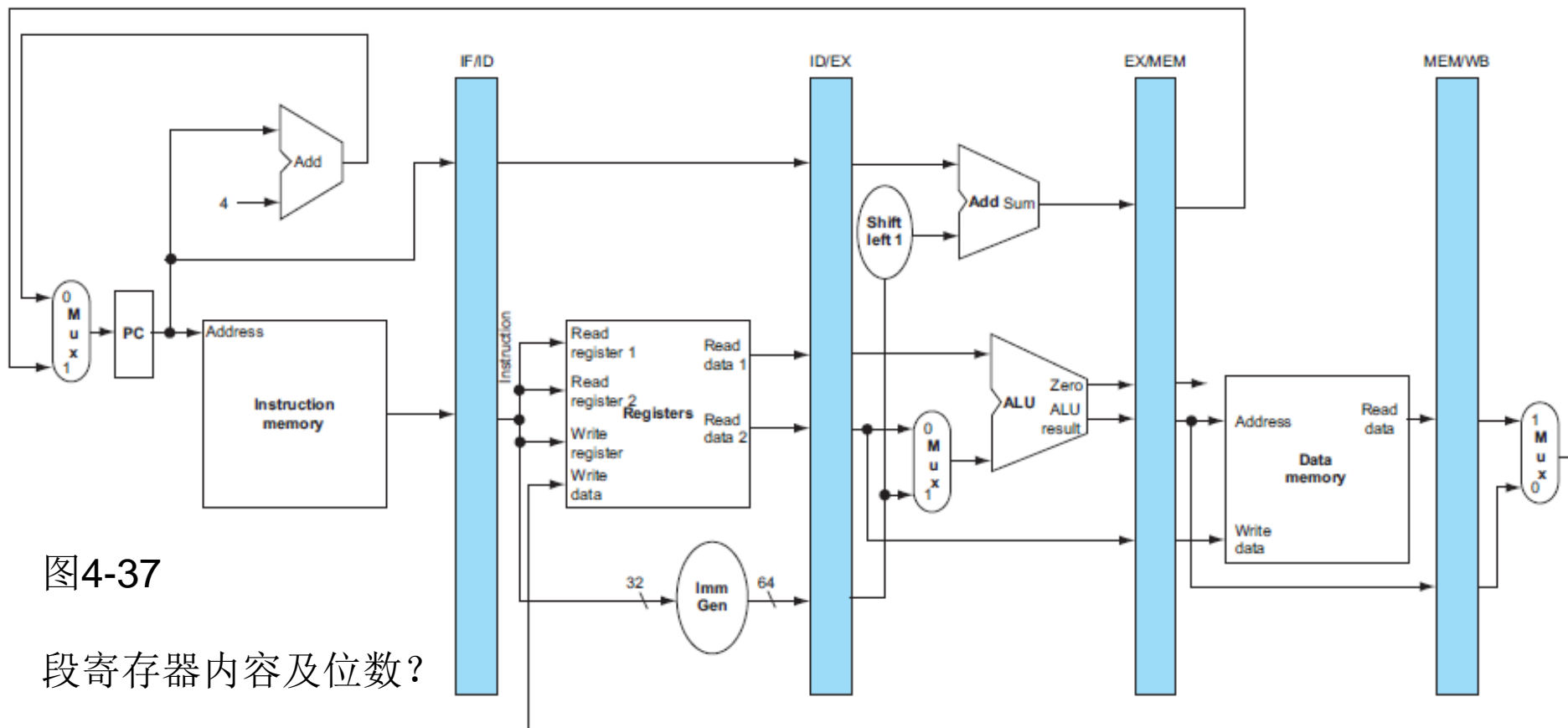
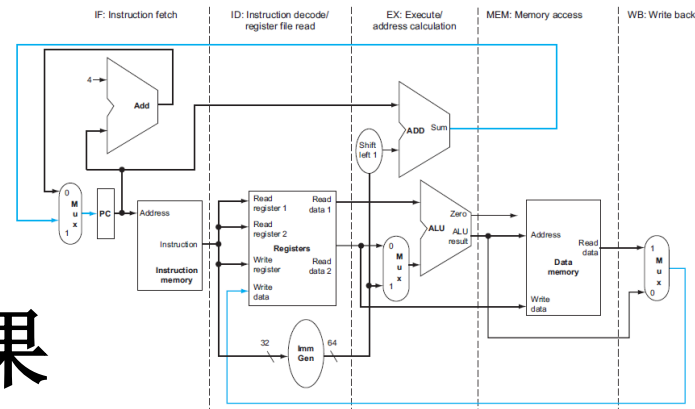
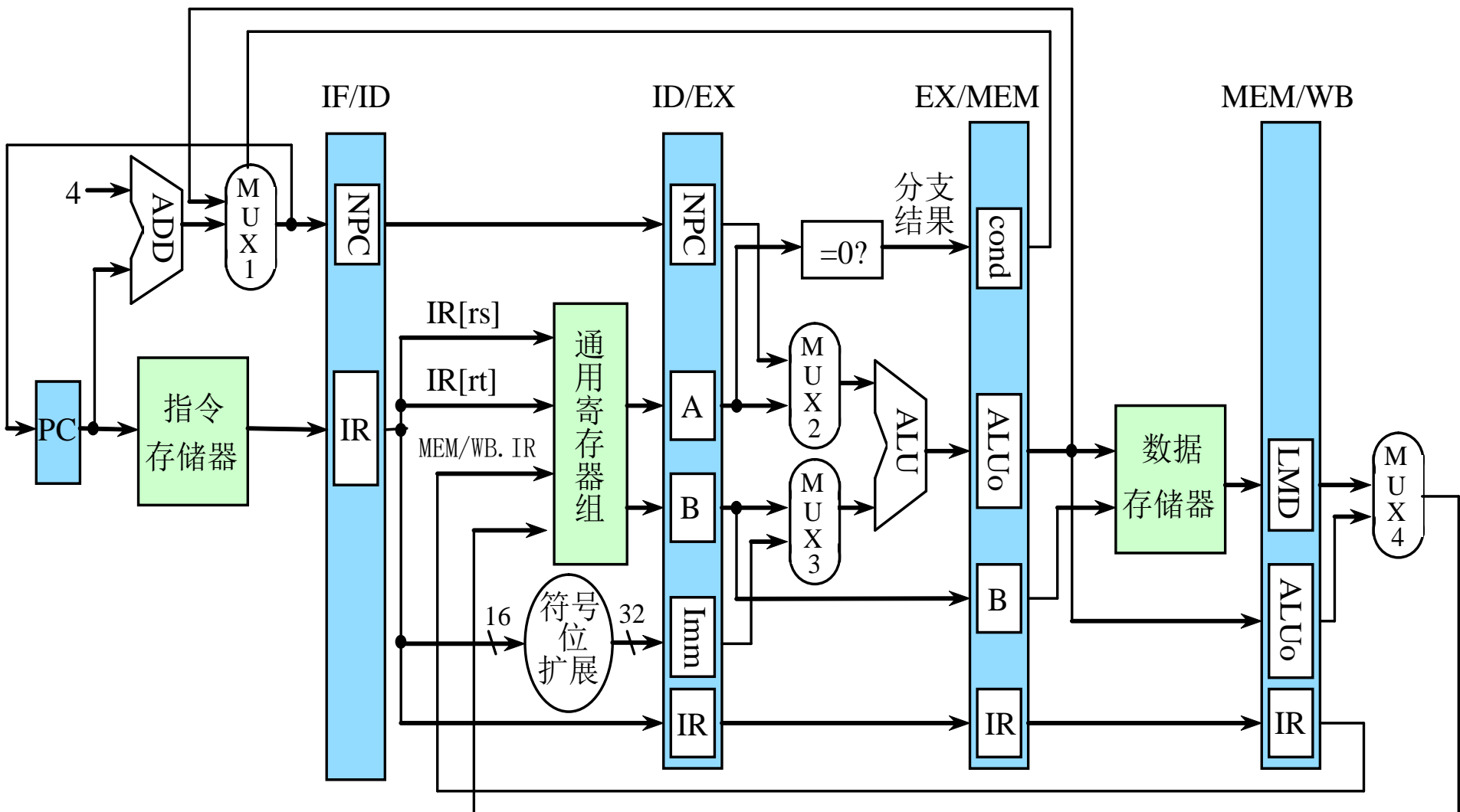
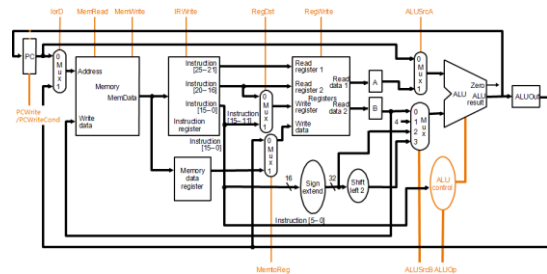


图4-37

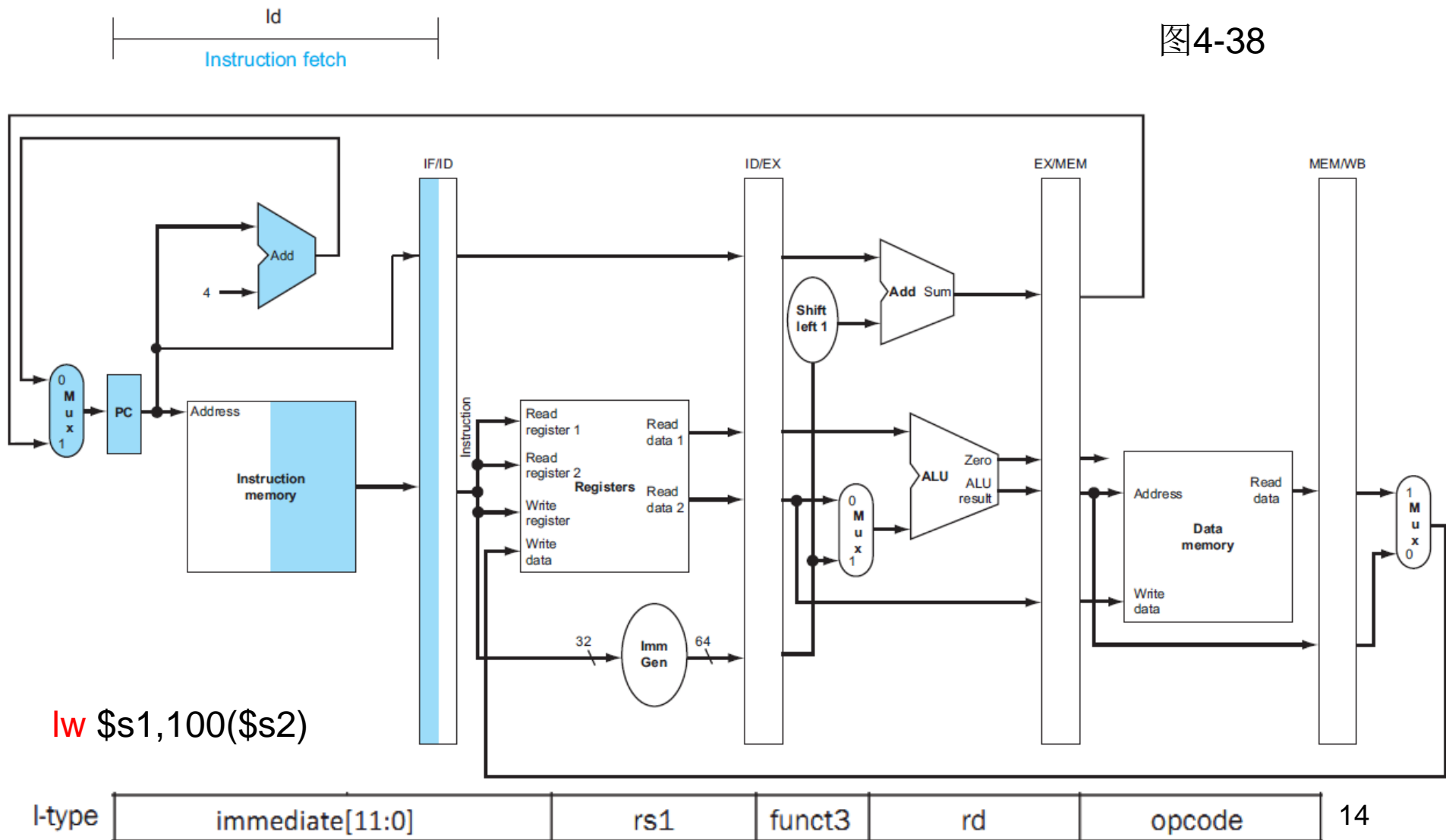
段寄存器内容及位数?

DLX的流水段寄存器



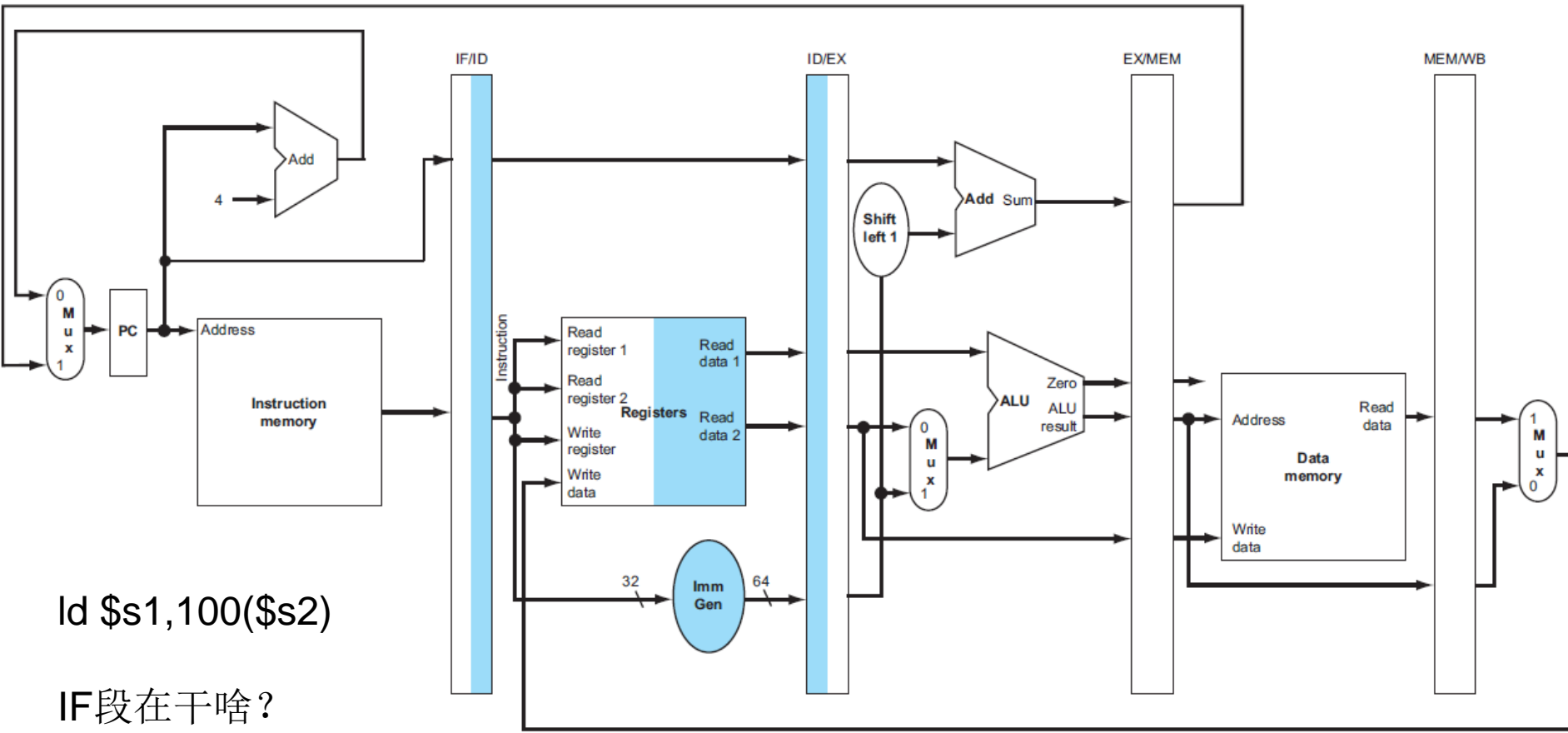
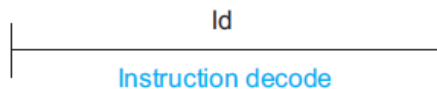
load指令的取指阶段

图4-38



load指令的译码阶段

图4-38



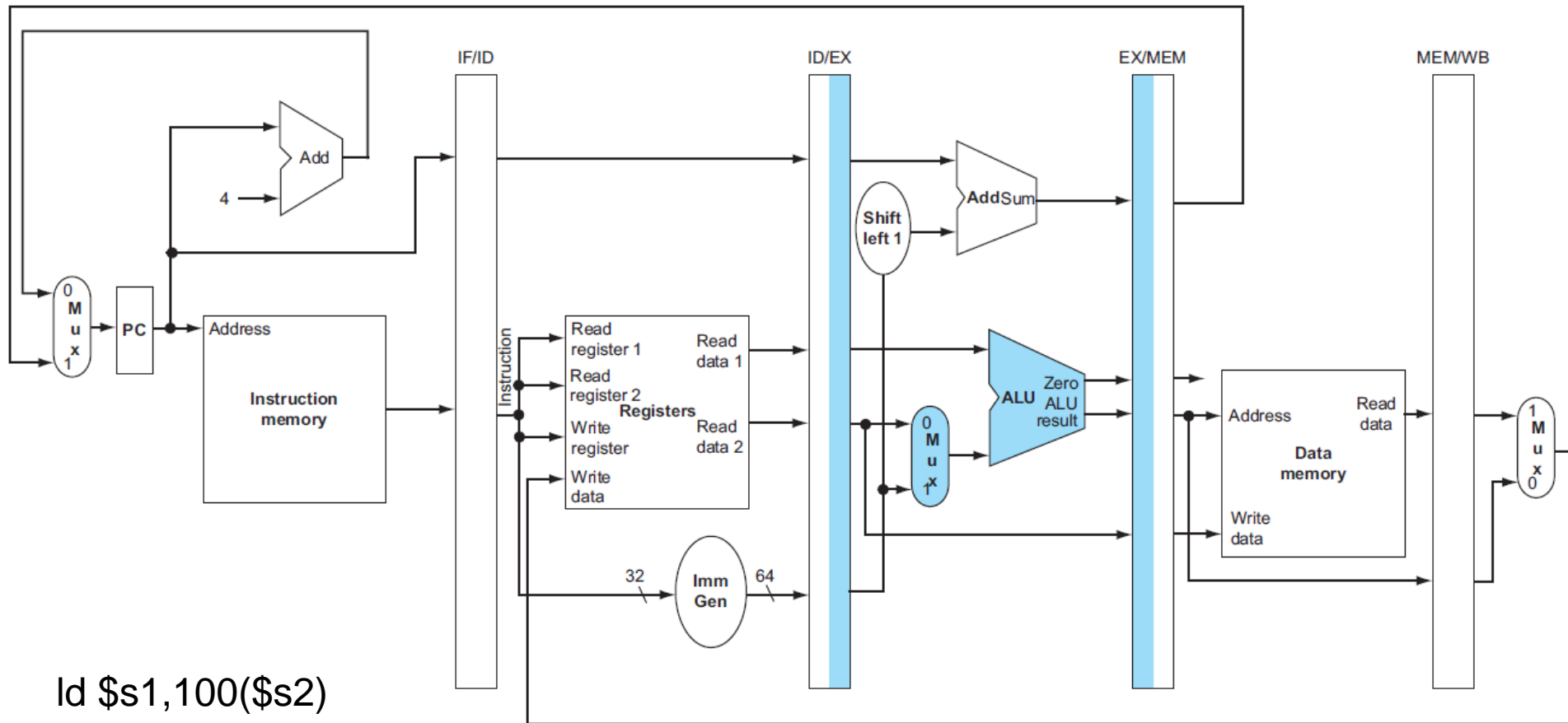
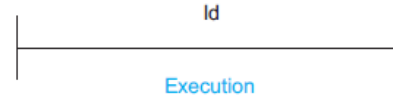
ld \$s1,100(\$s2)

IF段在干啥？

段寄存器：后写！！

load指令的执行阶段

图4-39。 adder在干啥？

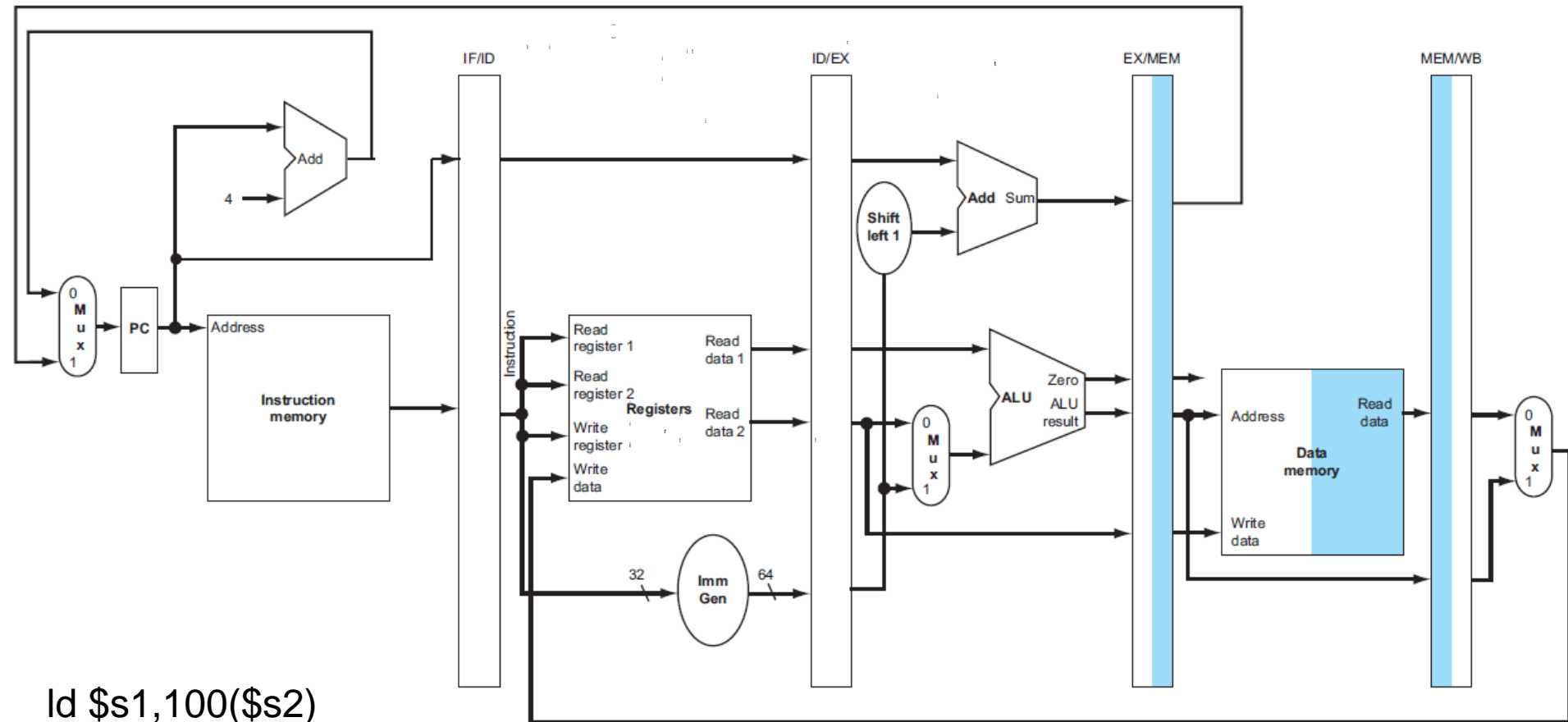


`ld $s1, 100($s2)`

load指令的访存阶段

图4-40

Id
Memory

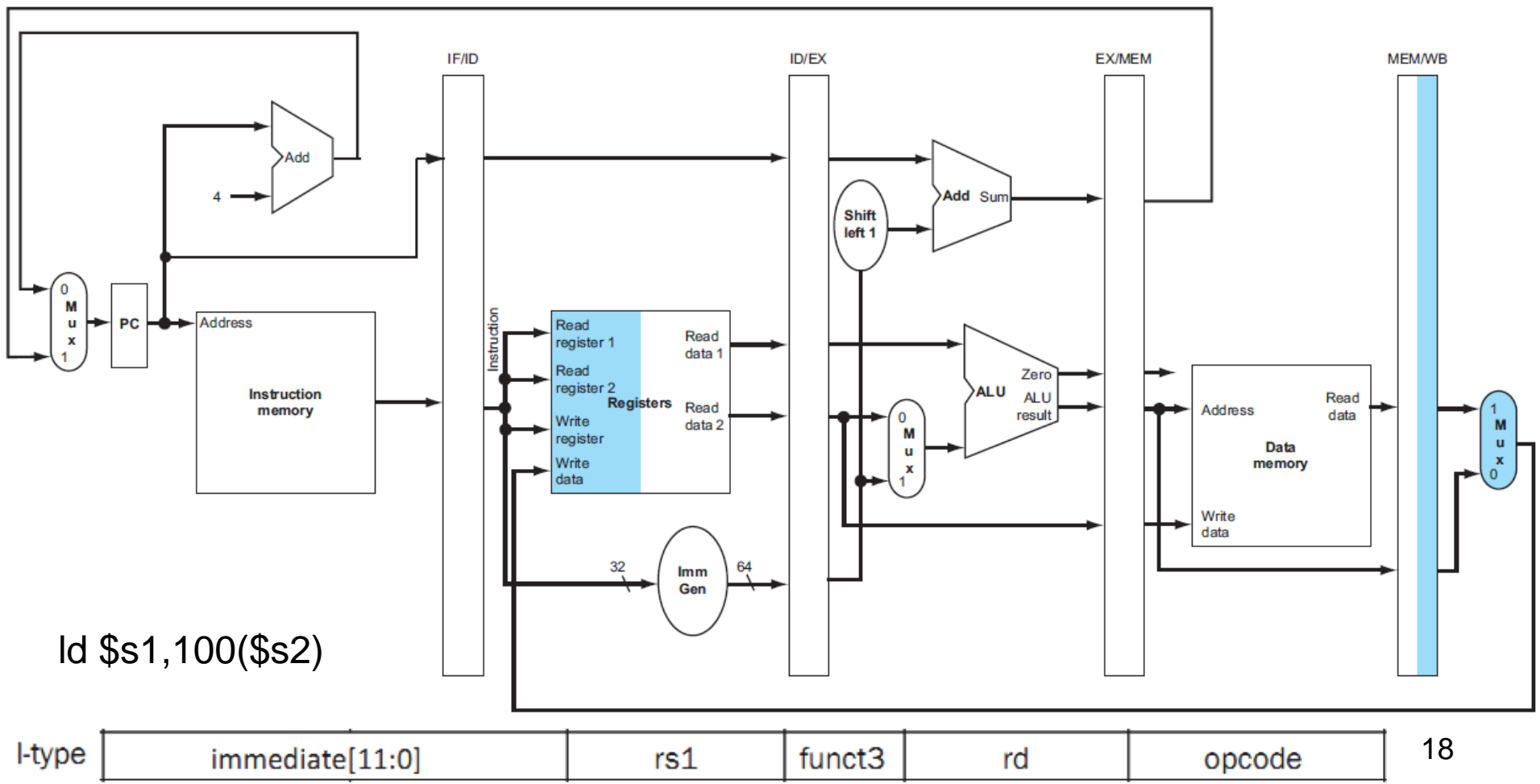


ld \$s1, 100(\$s2)

load的写回阶段: bug?

图4-40

Id
Write-back



ld \$s1, 100(\$s2)

I-type immediate[11:0] rs1 funct3 rd opcode 18

流水线改正版

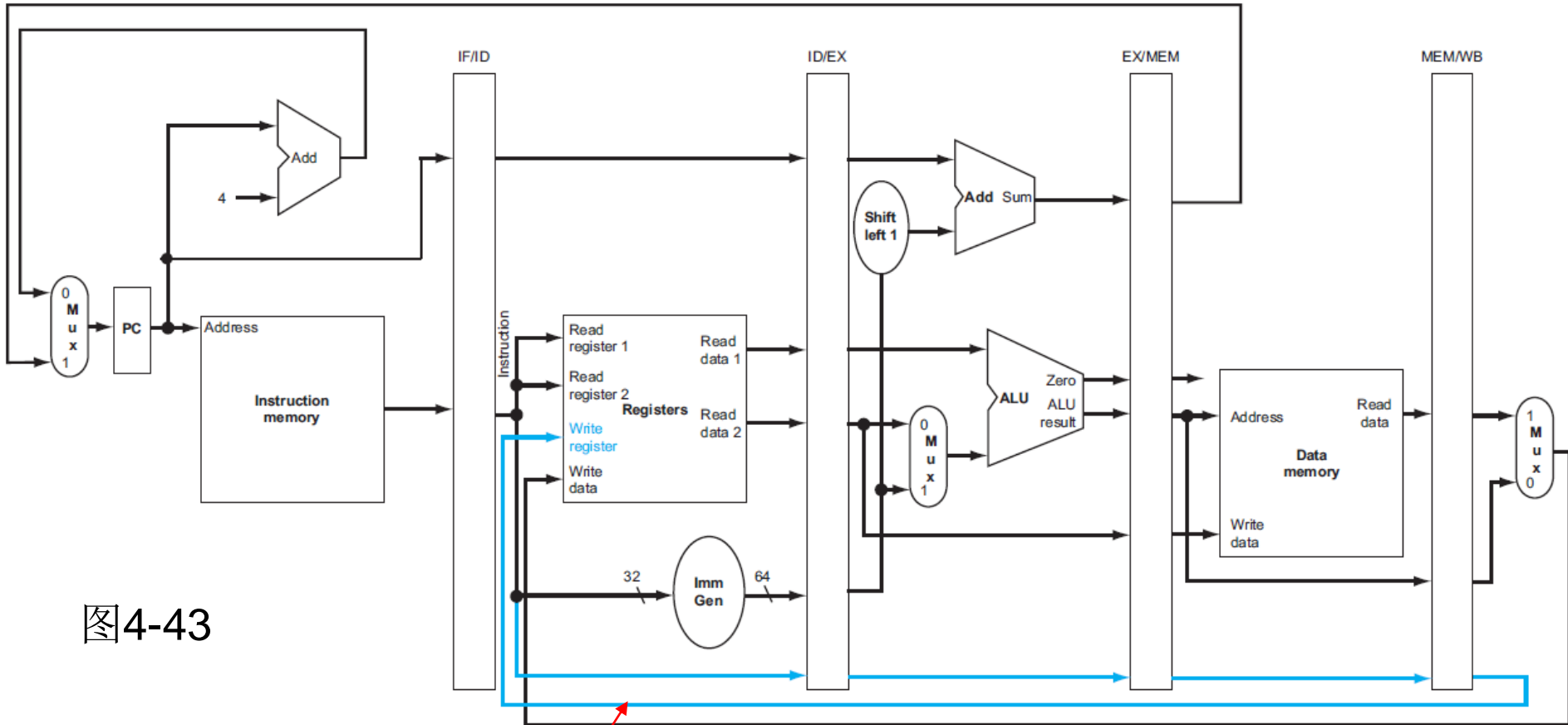
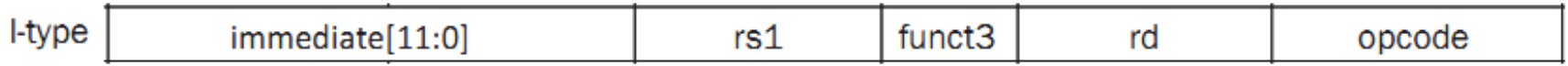
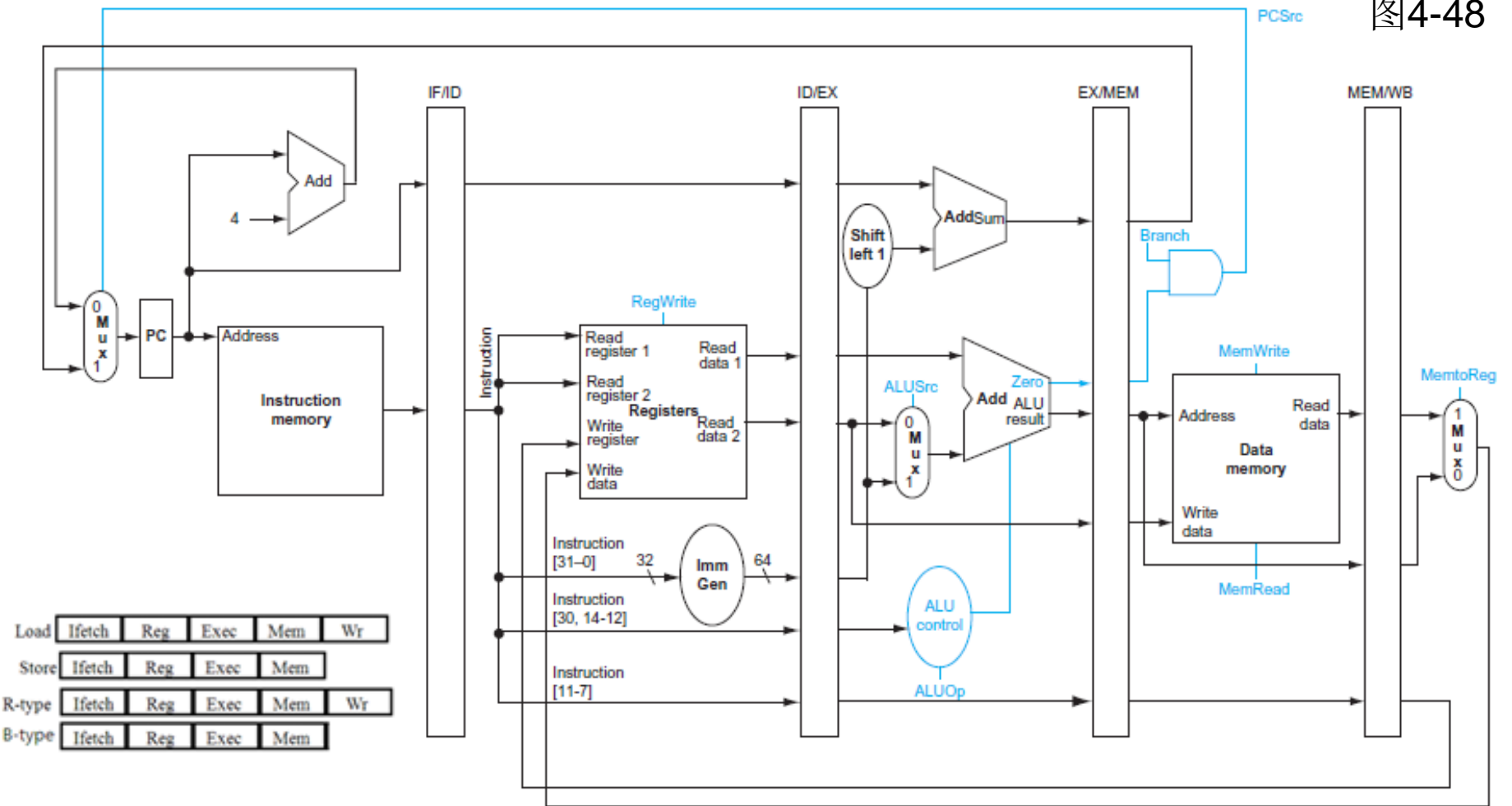


图4-43

R-type? beq?

各指令完成时间

图4-48

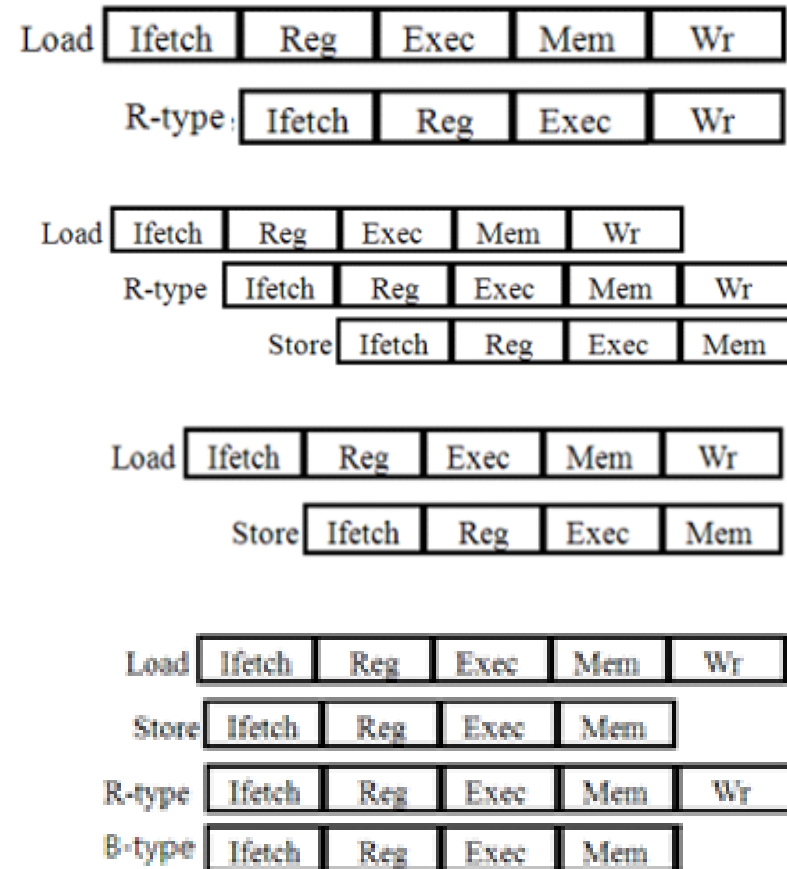


“指令周期定长”？每个周期流出几条指令？ **CPI? IPC?**

中文版p211：“an instruction passes through a stage even if there is **nothing to do**”？

每个CC完成的动作：结构冲突？

- PC：一读一写
 - 先读后写
 - beq, jal/jalr：一次读两次写（实现？）
 - 分支预测器：使PC“一读一写”？
- RF：两读一写
 - 先读后写？
 - 注意：单周期版为先读后写
 - 先写后读：“三有四无”
- Imem：一读
- Dmem：一读/一写
- 段寄存器：一读一写
 - 数据通路寄存器
 - 控制信号寄存器
 - 先读后写，down读up写

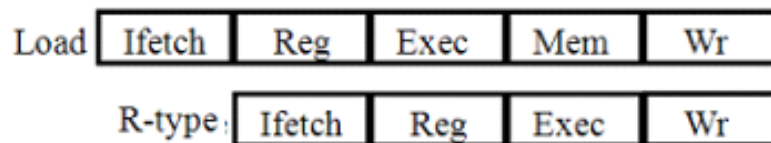


R-type指令完成时间：WB段

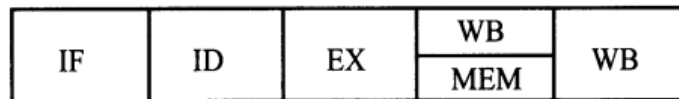
- R-type指令4个周期完成？
 - 例：当前指令为lw，下一条指令为R-type。CC5？

FIGURE e4.5.6 多周期架构的指令完成时间

Action for R-type instructions	Action for memory reference instructions	Action for branches
	IR \leftarrow Memory[PC] PC \leftarrow PC + 4	
	A \leftarrow Reg [IR[19:15]] B \leftarrow Reg [IR[24:20]] ALUOut \leftarrow PC + immediate	
ALUOut \leftarrow A op B	ALUOut \leftarrow A + immediate	if (A == B) PC \leftarrow ALUOut
Reg [IR[11:7]] \leftarrow ALUOut	Load: MDR \leftarrow Memory[ALUOut] or Store: Memory [ALUOut] \leftarrow B	
	Load: Reg[IR[11:7]] \leftarrow MDR	

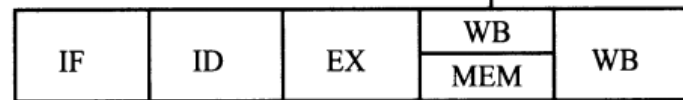


load Reg2,A



Register file

mul Reg3,Reg4,Reg5



cycle time

time

RF “两读一写”，RF写端口冲突！【不是写地址】

方案1: ppl停顿 (stall) /阻塞

方案2: “每条指令最多只写一个结果，且在最后一级进行”。【4特点 (\$4.6.1)】

节省指令完成时间：有意义？

- R-type指令能否4个周期完成？
- S-type指令需要在WB段完成？

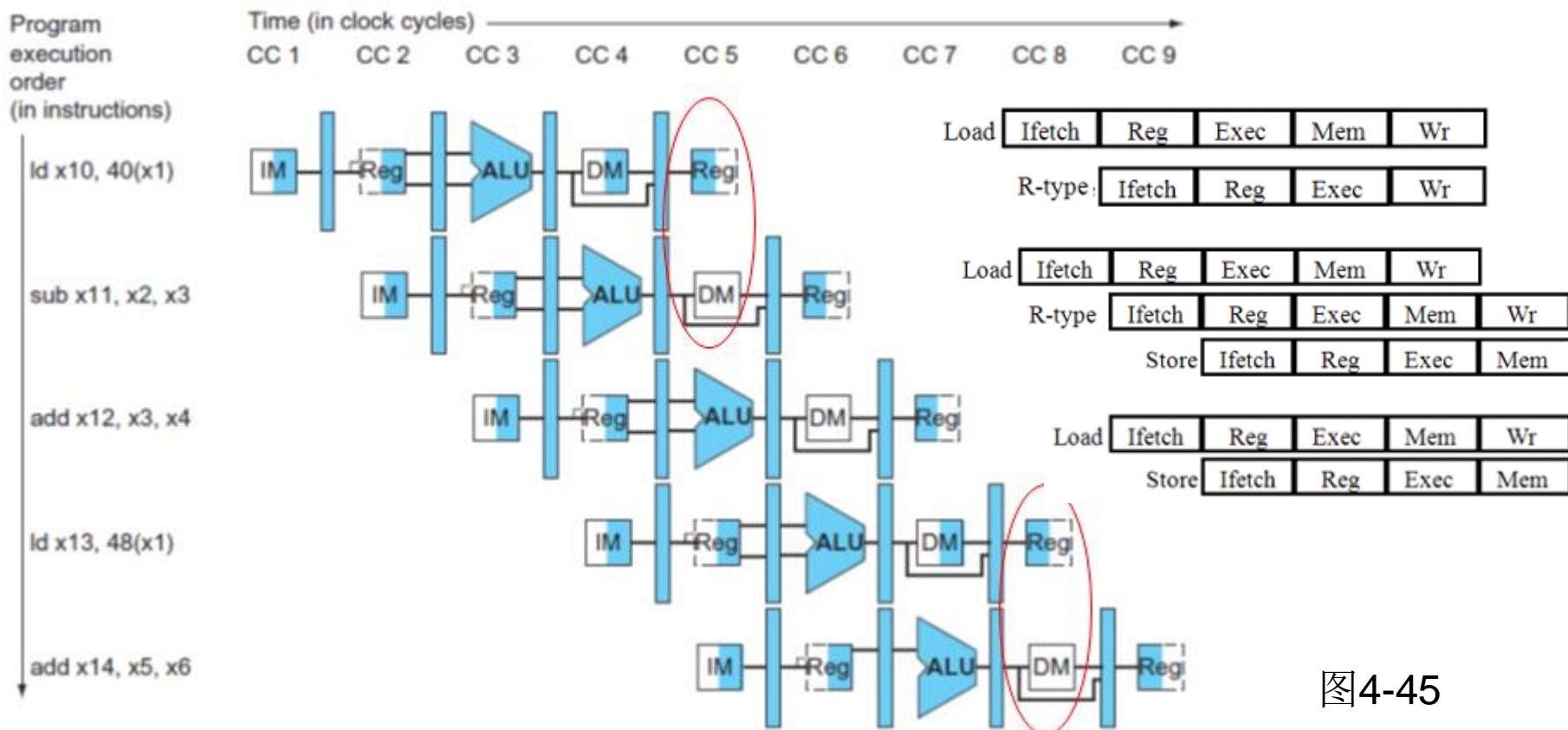
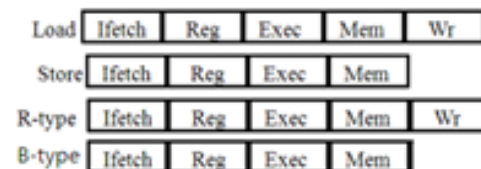


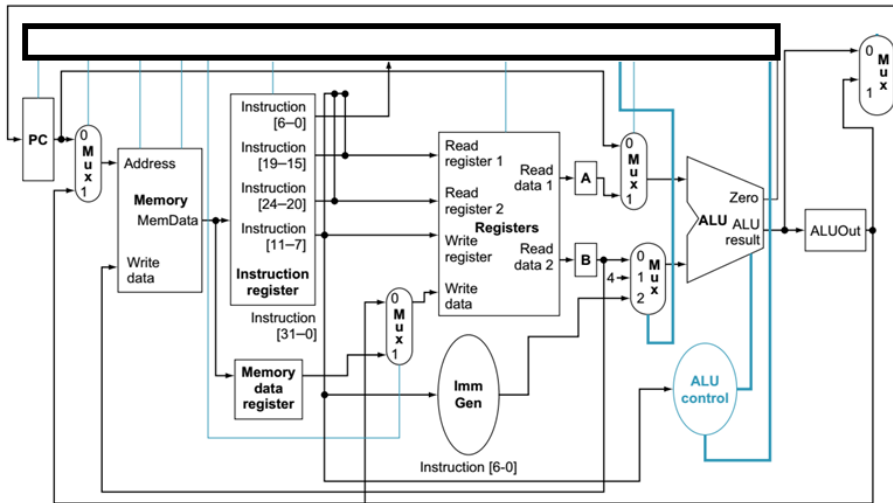
图4-45

beq指令完成时间?

FIGURE e4.5.6

- 分支以哪一条指令为基址?
 - 单周期
 - 多周期
 - ppl

FIGURE e4.5.4



Action for R-type instructions	Action for memory reference instructions	Action for branches
	IR \leftarrow Memory[PC] PC \leftarrow PC + 4	
	A \leftarrow Reg [IR[19:15]] B \leftarrow Reg [IR[24:20]] ALUOut \leftarrow PC + immediate	
ALUOut \leftarrow A op B	ALUOut \leftarrow A + immediate	if (A == B) PC \leftarrow ALUOut
Reg [IR[11:7]] \leftarrow ALUOut	Load: MDR \leftarrow Memory[ALUOut] or Store: Memory [ALUOut] \leftarrow B	
	Load: Reg[IR[11:7]] \leftarrow MDR	

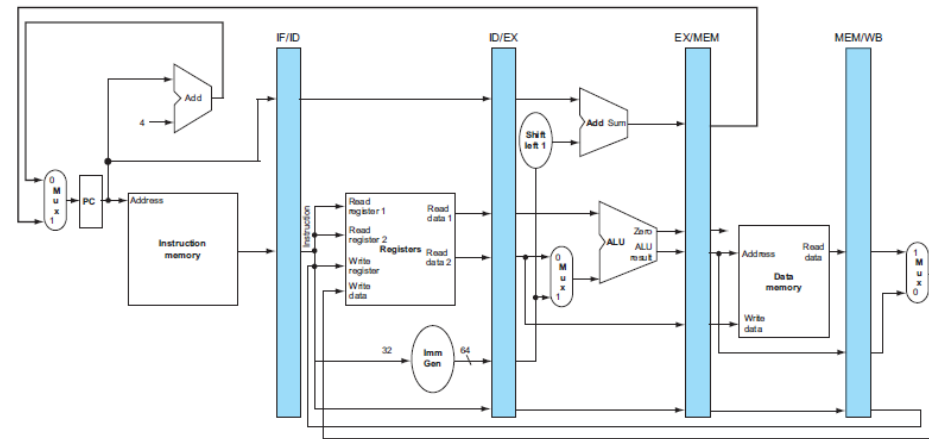


图4-33

beq数据通路与完成时间：MM段

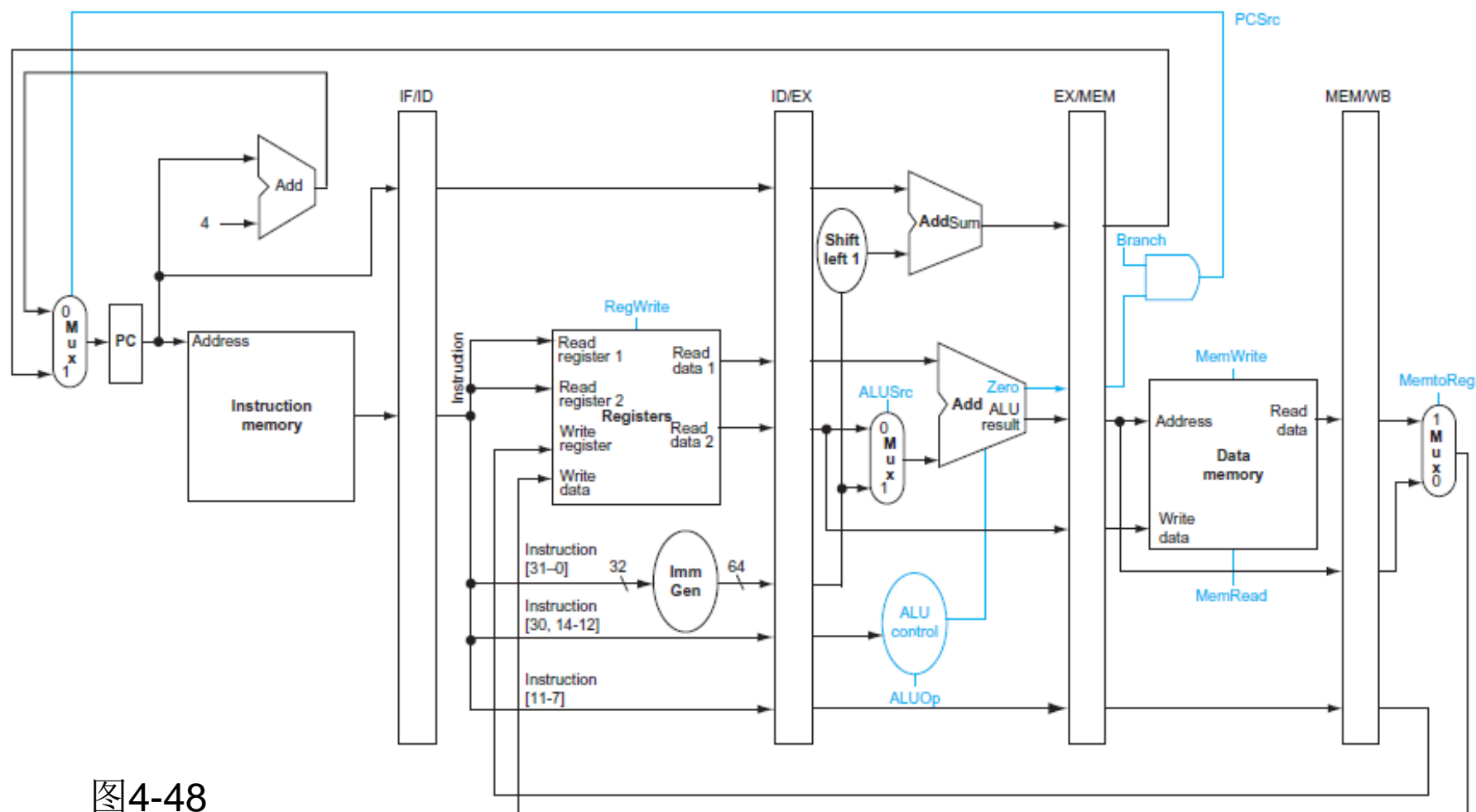


图4-48

Activity of each instruction in each stage of the pipeline: MIPS

Instruction	I-fetch (IF)	I-decode (ID)	Execute (EX)	Memory (ME)	Write back (WB)
LW R1,#20(R2)	fetch; PC+=4	decode; fetch R2	compute address – (R2)+20	read	write in R1
SW R1,#20(R2)	fetch; PC+=4	decode; fetch R1 and R2	compute address – (R2)+20	write	–
ADD R1,R2,R3	fetch; PC+=4	decode; fetch R2 and R3	compute (R2)+(R3)	–	write in R1
ADDI R1,R2,imm.	fetch; PC+=4	decode; fetch R2	compute (R2)+imm	–	write in R1
BEQ R1,R2,offset	fetch; PC+=4	decode; fetch R1 and R2 compute target – address (PC)+offset	subtract (R1) and (R2); take branch if zero	–	–
J target	fetch; PC+=4	decode; take jump	–	–	–

RV? DIY! ! !

Multiple-clock-cycle Pipeline Diagram, §4.7.1

Time (in clock cycles) →
 CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9

Program execution order (in instructions)

ld x10, 40(x1)
 sub x11, x2, x3
 add x12, x3, x4
 ld x13, 48(x1)
 add x14, x5, x6

阴影：各CC占用的资源

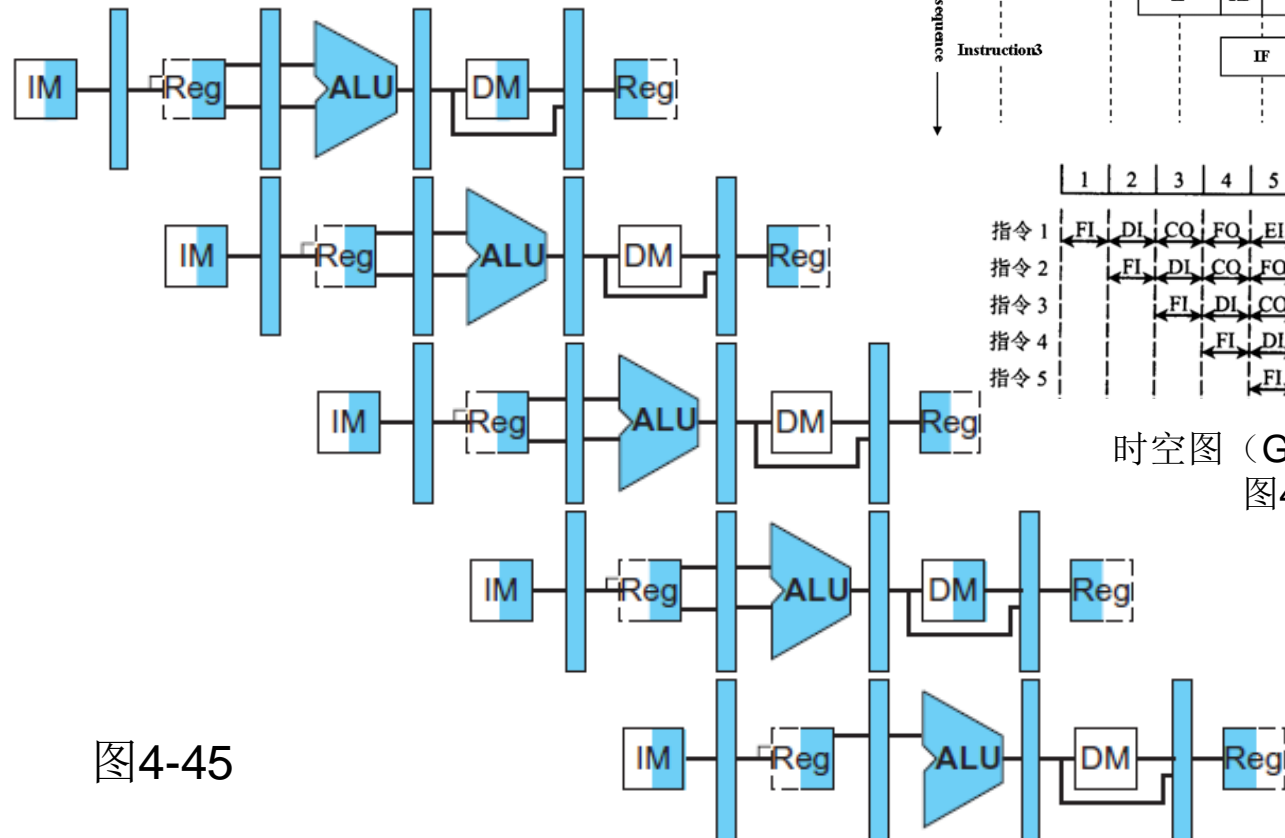
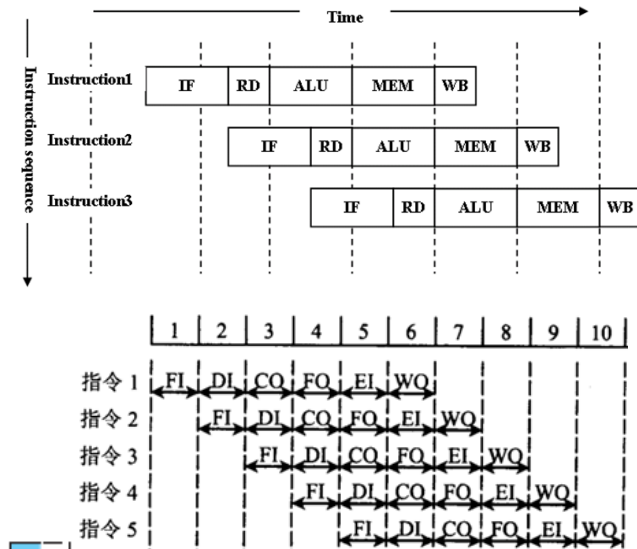


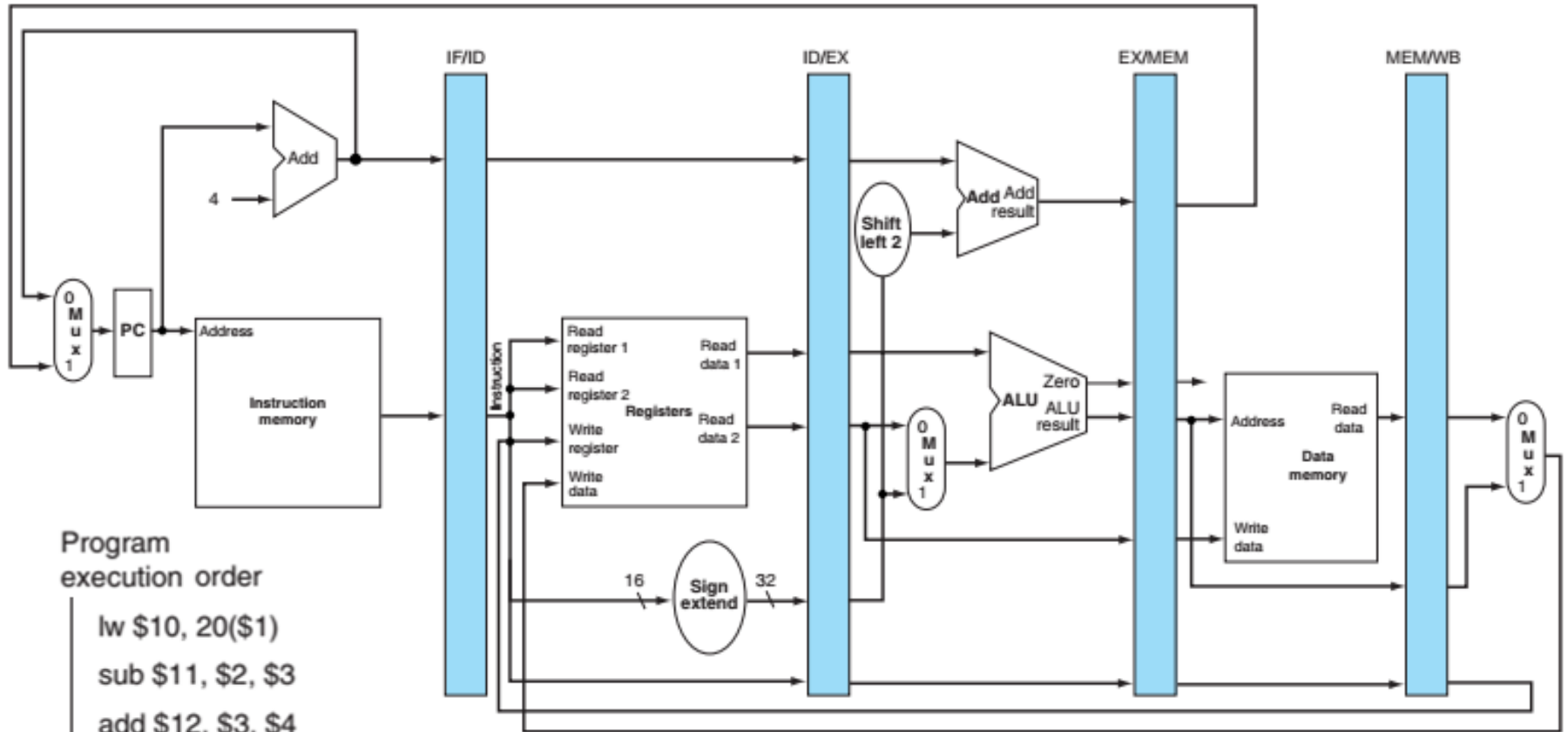
图4-45



时空图 (Gantt chart)

图4-33

Single-clock-cycle Diagram: CC5, §4.7.1

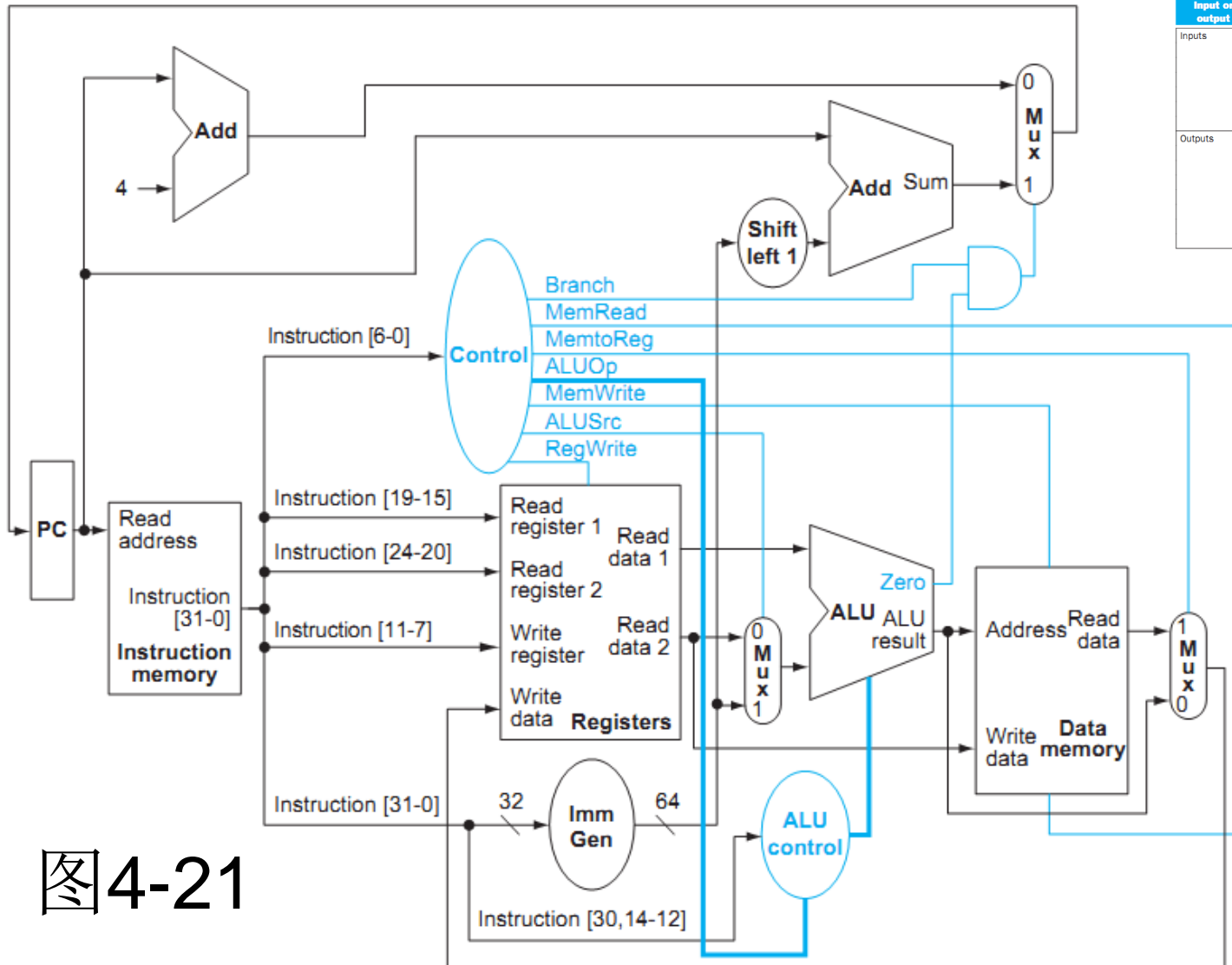


Program execution order

- lw \$10, 20(\$1)
- sub \$11, \$2, \$3
- add \$12, \$3, \$4
- lw \$13, 24(\$1)
- add \$14, \$5, \$6

图 4-47

流水线控制信号？



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
Outputs	I[0]	1	1	1	1
	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

单周期真值表

图4-21

流水线控制信号?

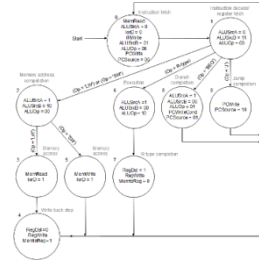
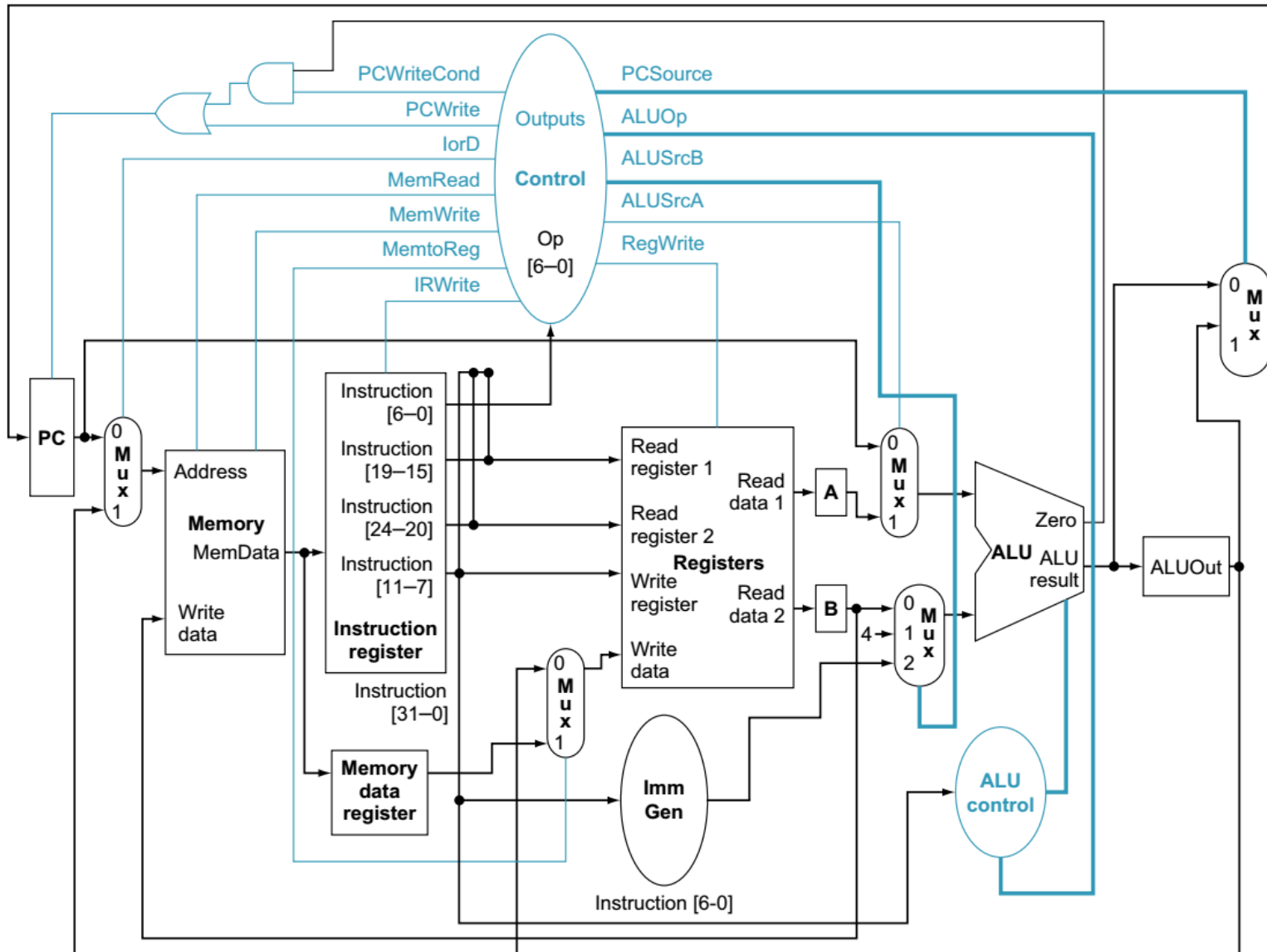


FIGURE e4.5.4

流水线所需控制信号

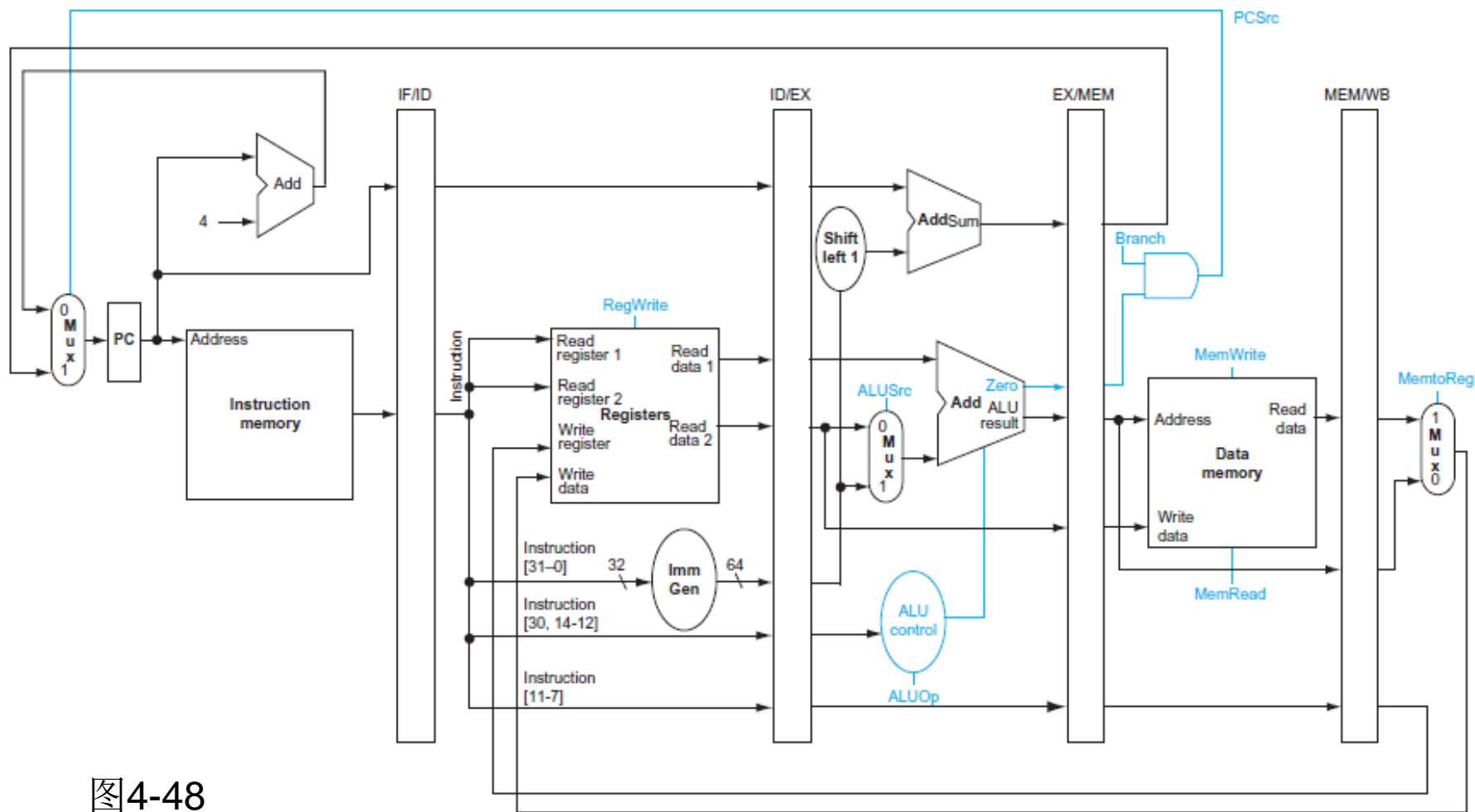
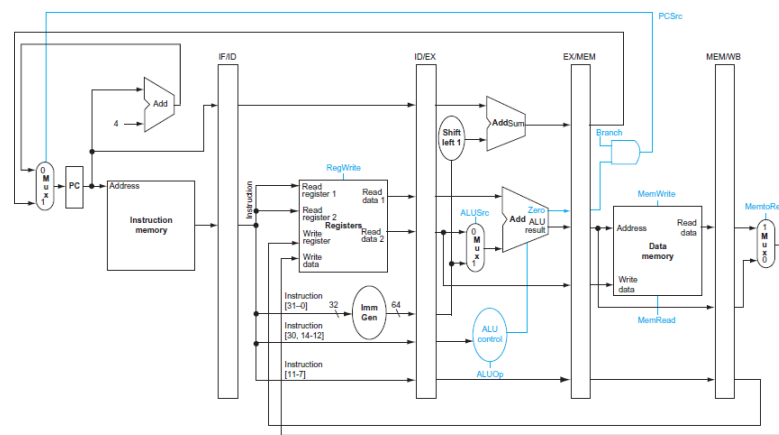


图4-48

流水线控制信号：RV2 \$4.7.2

- 所有控制信号名及其功能与单周期版类似：7+1
 - 取指：读IM，写PC（每个周期写入一次，假设不需写控制信号）
 - 译码/寄存器堆读：没有控制信号
 - 执行/地址计算：ALUOp, ALUSrc
 - 访存：Branch, MemRead, MemWrite
 - PCSrc = Branch && Zero \approx Branch
 - 写回：MemtoReg, RegWrite
- 流水线段寄存器
 - 每个周期写入一次，假设不需写控制
- 需要将控制分配给不同的流水线段：与各段的当前指令绑定！！！！



Instruction	Execution/address calculation stage control lines		Memory access stage control lines			Write-back stage control lines	
	ALUOp	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	10	0	0	0	0	1	0
lw	00	1	0	1	0	1	1
sw	00	1	0	0	1	0	X
beq	01	0	1	0	0	0	X

图4-51

控制的传递

Instruction	Execution/address calculation stage control lines		Memory access stage control lines			Write-back stage control lines	
	ALUOp	ALUSrc	Branch	Mem-Read	Mem-Write	Reg-Write	Memto-Reg
R-format	10	0	0	0	0	1	0
lw	00	1	0	1	0	1	1
sw	00	1	0	0	1	0	X
beq	01	0	1	0	0	0	X

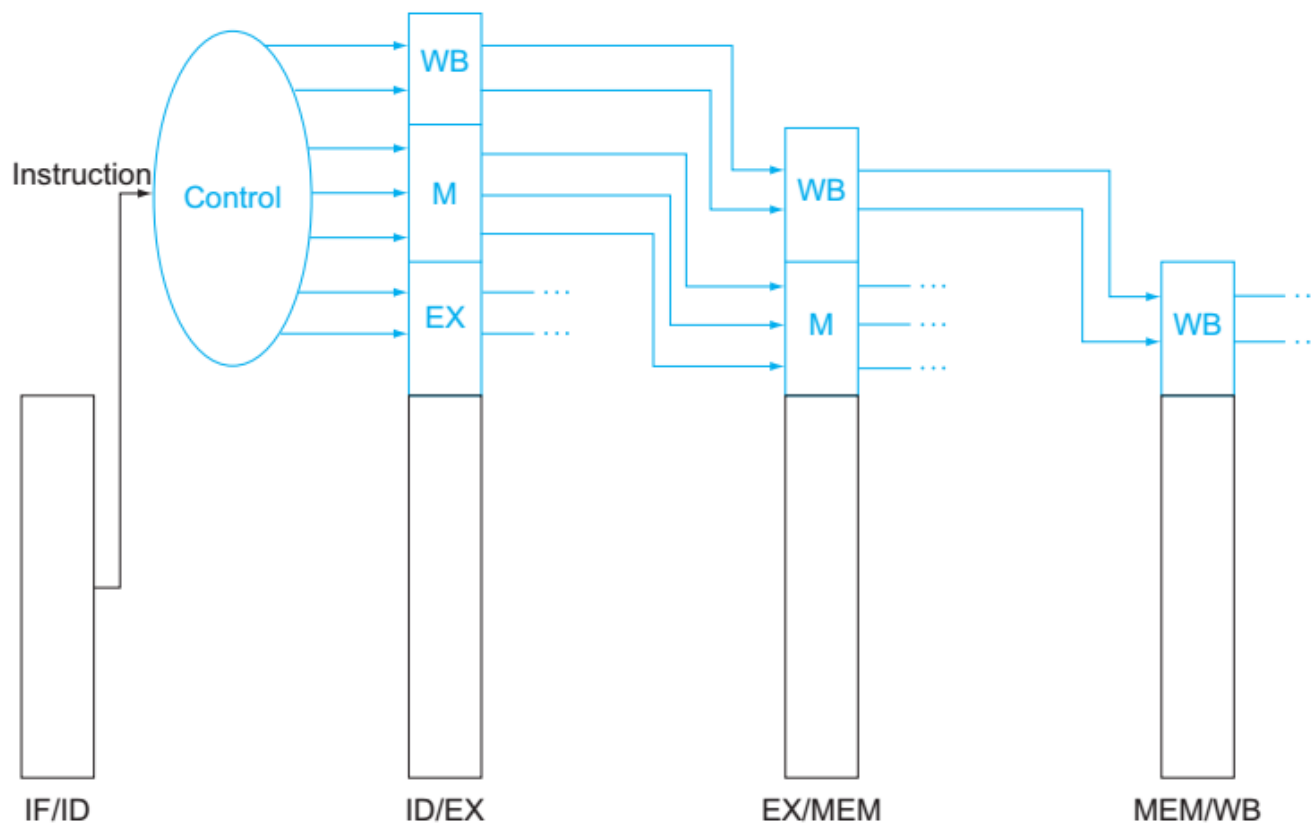


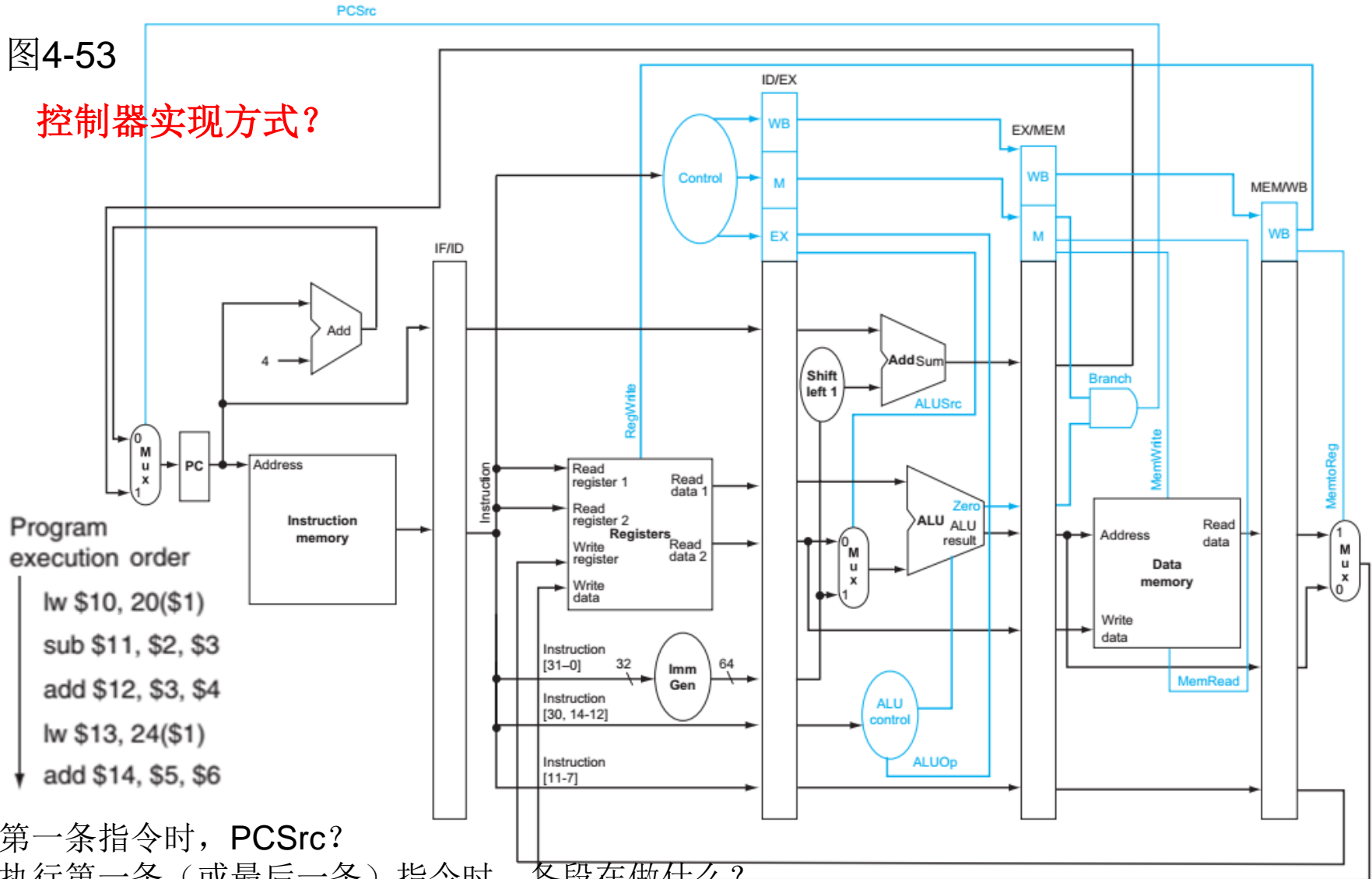
图4-52

- 需要**控制寄存器**：分段（分周期）使用**7**个控制信号

the control of the pipeline register

图4-53

控制器实现方式?

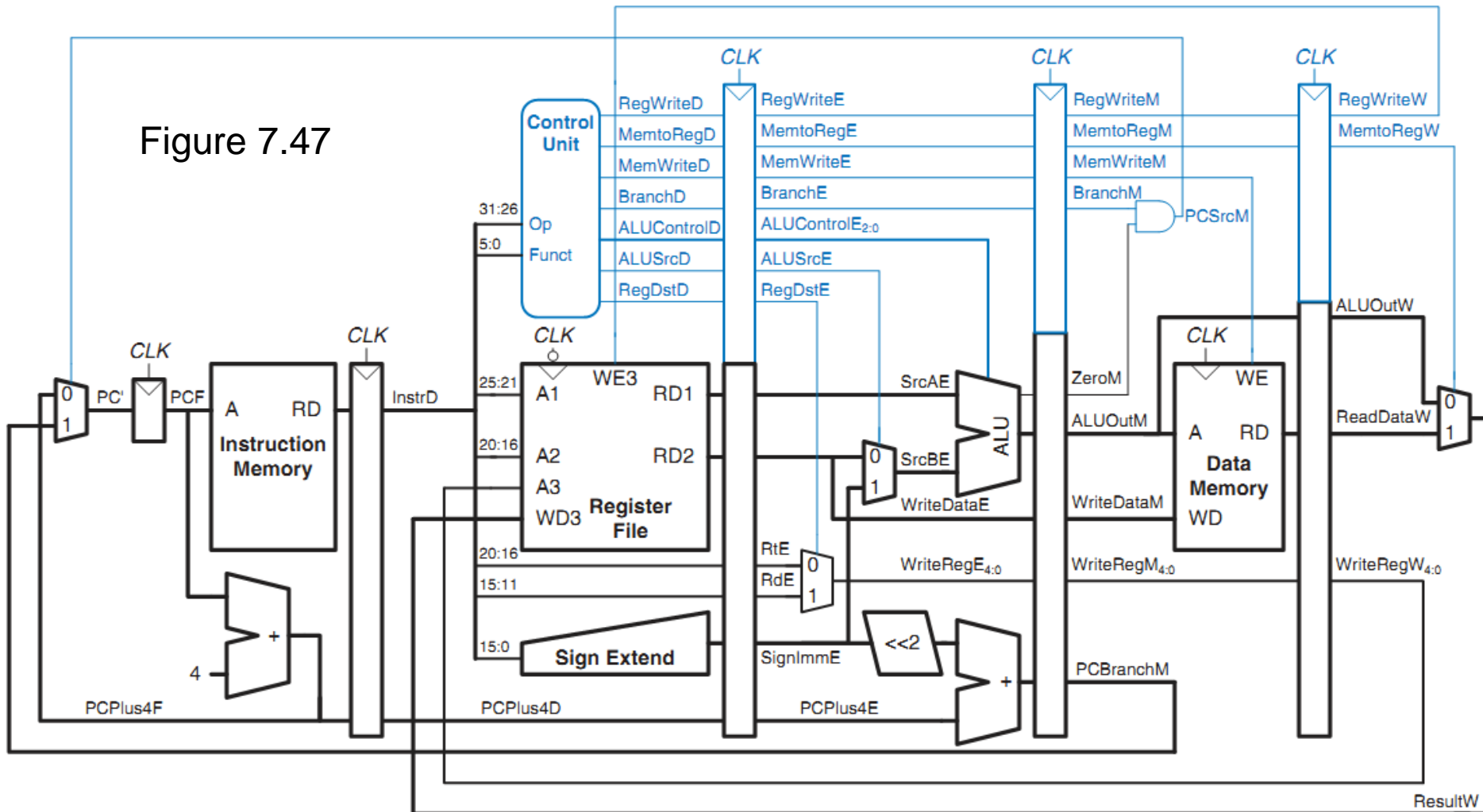


第一条指令时, PCSrc?

执行第一条 (或最后一条) 指令时, 各段在做什么?

Pipelined processor with **clock**

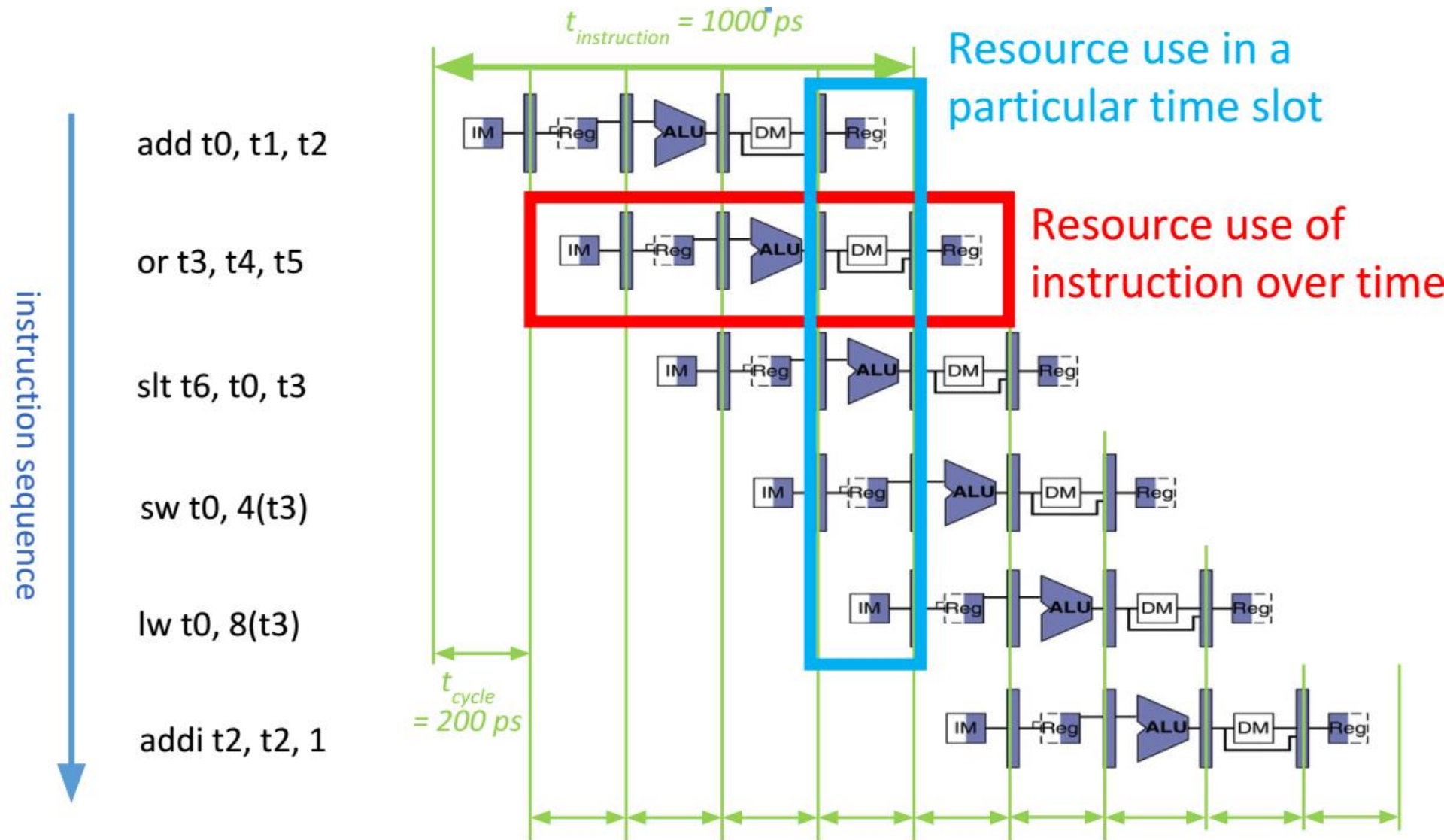
Figure 7.47



单周期实现 vs. 流水线性能: 类p196

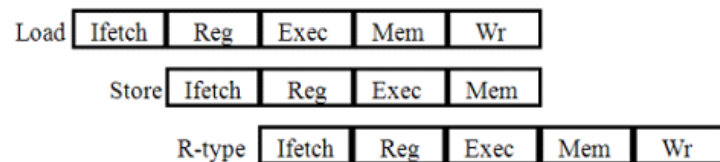
- 例: 设MEM=2ns, ALU和加法器=2ns, Reg=1ns, 其他部件没有延时。单周期模式与五段流水线分别执行3条指令的时间?
 - 单周期: 按执行时间最长的指令定时, 8ns
 - 执行3条指令所需的时间为 $8\text{ns} \times 3 = 24\text{ns}$
 - 第4条指令与第1条指令之间的时间间隔= $8\text{ns} \times 3 = 24\text{ns}$
 - 流水线段, 按操作时间最长的阶段定义CC, 2ns
 - 执行3条指令所需的时间为 $2\text{ns} \times 5 + 2\text{ns} + 2\text{ns} = 14\text{ns}$
 - 第4条指令与第1条指令之间的时间间隔= $2\text{ns} \times 3 = 6\text{ns}$
 - 性能提升
 - 按照3条指令的总执行时间, $24/14 = 2$ 倍
 - 按照时间间隔, $24/6 = 4$ 倍
 - 加速比?

RV 5-stages流水线: 资源



Summary

- 程序性能与CPU性能：铁律
- ISA的uArch实现模式：假设一个机器周期=1个时钟周期
 - 单周期：指令周期（定长）由一个时钟周期组成。
 - 多周期：指令周期（变长）由多个时钟周期组成。
 - 功能部件可以在不同周期中复用；中间结果需要暂存
 - 流水线：指令周期（变长）由多个流水级组成，ILP
 - 组织：部件独立，等时，贪心（ASAP）
 - 理想流水线三要素：“均衡，重复，无依赖”
 - 提升了吞吐率：CPI，IPC，加速比
 - 最好情况，最坏情况？
- 流水线RV的**四特征**
 - R-type指令是否可以不要访存段？
 - 是否会出现同一cycle流出多条指令？
- CPU设计步骤：ISA，数据通路，控制器，时序
 - uArch 4步法：分解微操作，数据通路，控制器，综合



作业

- 思考
 - 为何RV只有load/store指令访存？
 - 为何单周期、多周期的控制信号无须buffer，而流水线的控制信号需要buffer？
 - 流水线控制器的实现方式？
 - 影响流水线性能发挥的因素有哪些？
- 实验：RV指令级仿真器？
- 作业
 - 4.16.1~3, 4.13, 4.23

Thank You