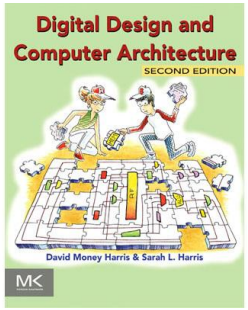
# The RISC-V Processor Implementation: Pipeline——ILP

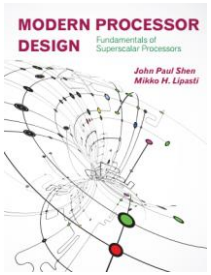
**“Computer Organization & Design ”**

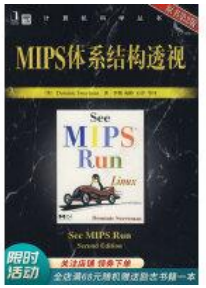
**David Patterson, John Hennessy**

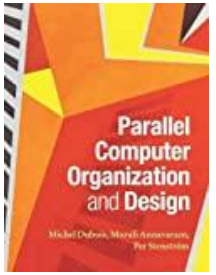
# 内容提要

- 流水线技术原理：4.5
- RV的五级流水线实现：4.6
  - Verilog行为级定义：4.13
- Hazard问题：4.5.2
  - 结构冲突：哈佛结构
  - 数据依赖：4.7
    - 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：4.8
    - 编译技术：延迟分支
    - 硬件优化：提前完成，投机，预测
- 多发射技术：4.10
- 硬件多线程：6.4

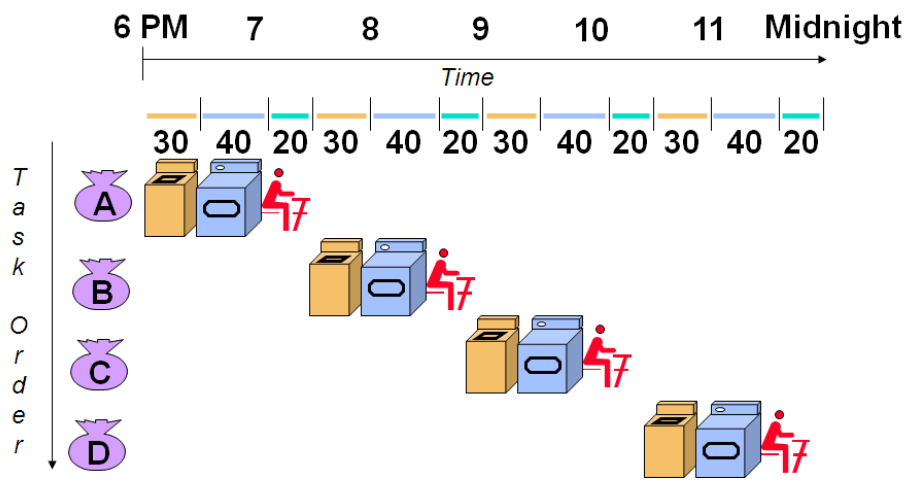


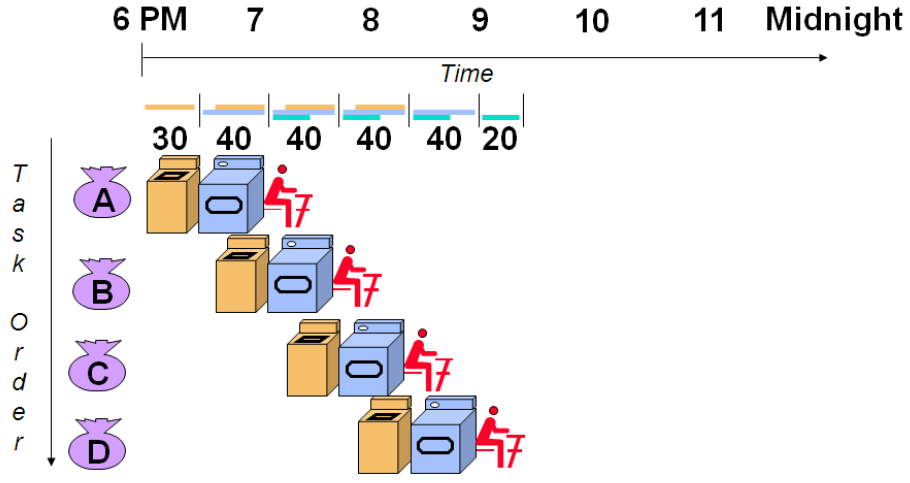



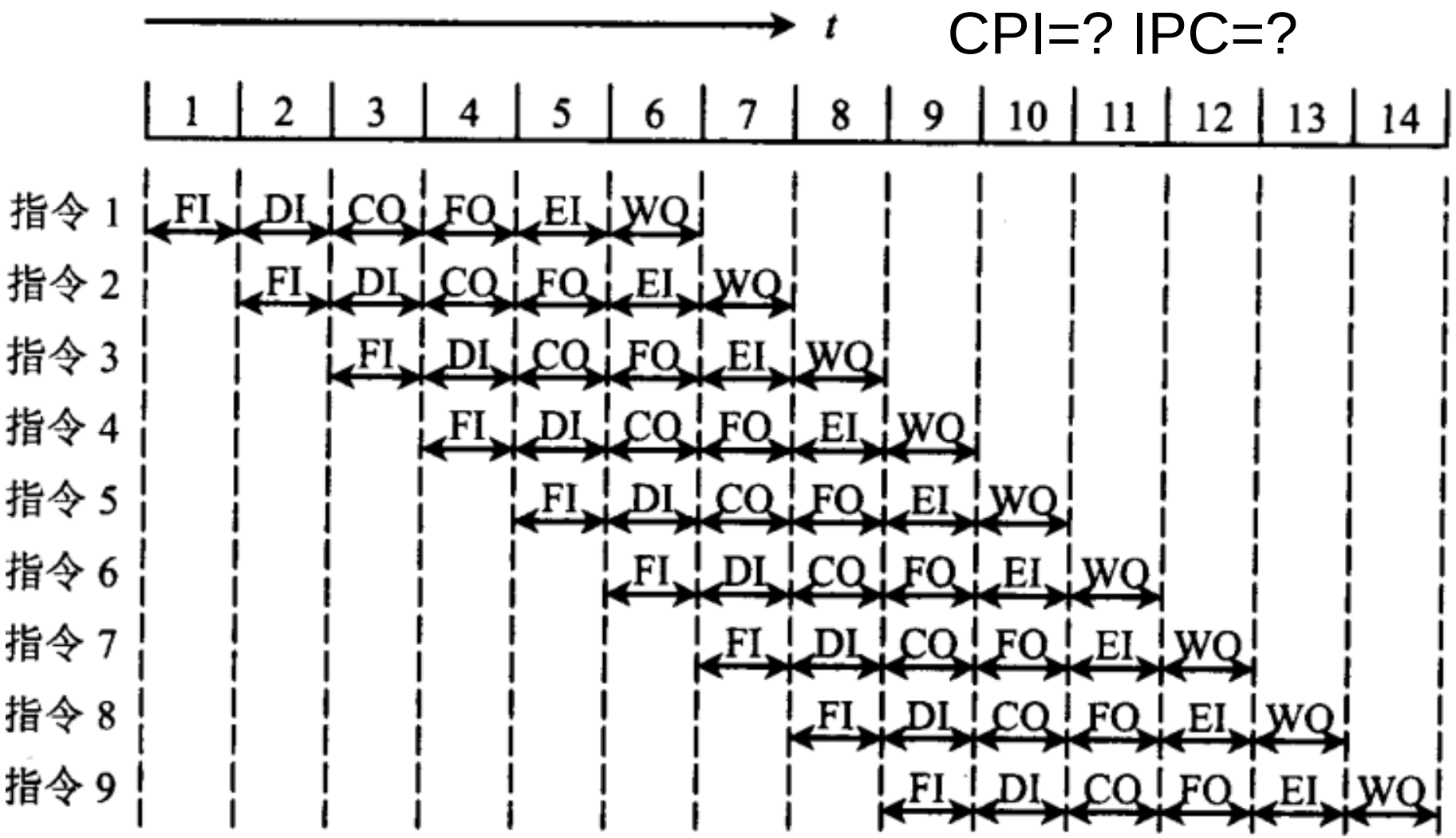
# Laundry Example




- 4 袋衣服：Wash（30）, dry（40）, fold（20）
  - Sequential Laundry：需要 6 小时：4x(30+40+20)=360分钟☹
  - Pipelined Laundry：需3.5小时：30+4x40+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！一袋衣服？
- 流水线目标：提升“吞吐率”，缩短“完成时间”。IPC？CPI?
- CISC顺序模型，RISC指令级并行模型（ILP）

# 指令流水线时空图（Gantt chart）：六级



# 流水线性能：Pipelining Idealism

- CPI，IPC，加速比（speedup）
  - 单周期，多周期，ppl？

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

从CC6开始，每个周期流出一条指令，因此，CPI = IPC ≈ 1。

- 每条指令的指令周期没变
- 整体性能提升

最坏情况？最好情况？IPC≥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**：流水线的流出顺序与其流入顺序相同。
- **乱序流水线**：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

- 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
  - 重复相同的指令
    - 单功能流水线
  - 指令流无依赖

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 |

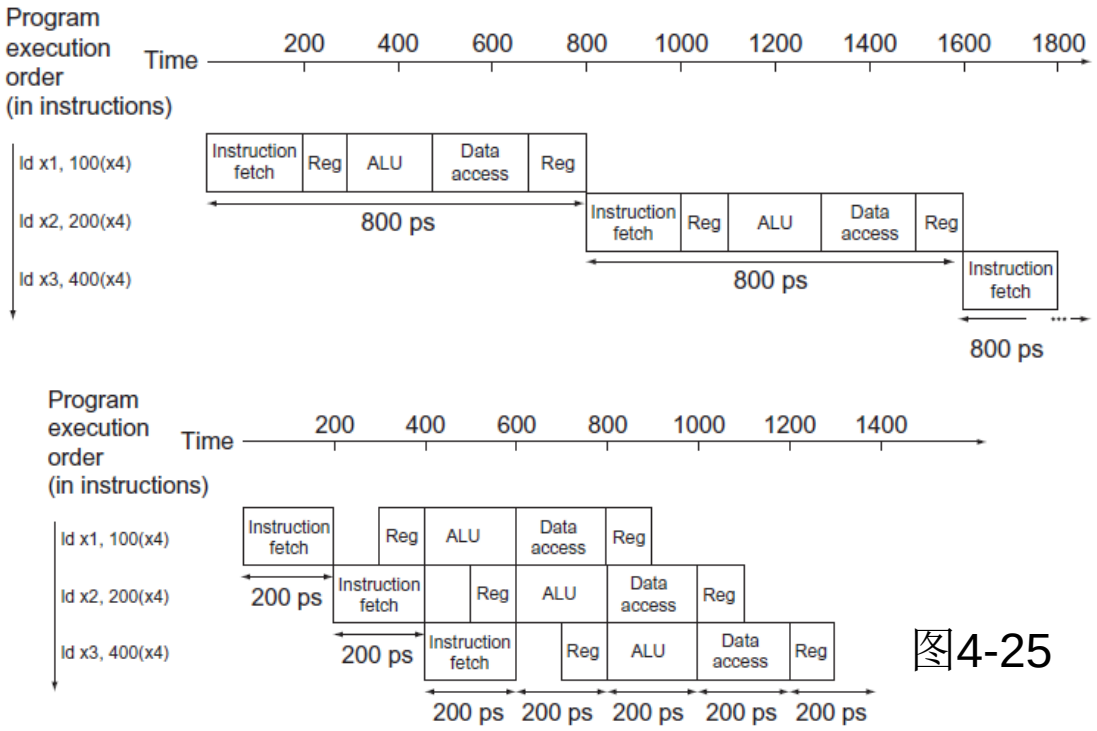



图4-25

# 面向流水线的ISA：4特点（$4.5.1）

| 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 |

- 指令字等长
  - 适于取指和译码
- 指令格式简单规则
  - 源操作数位置固定，因此可以在译码的同时取操作数（读寄存器堆），否则要增加一级流水线段。
- 只有load/store访存
  - 意味着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，则变为取指、译码、地址计算、访存、执行、写回等几个阶段
- 每条指令最多只写一个结果，且在最后一级进行
  - 利于前推（$4.5.2）
- *操作数在内存中“字对齐”——MIPS版，RV/ARM无？*
  - *因此在取操作数时不需要访问存储器多次*
- *寻址方式简单？*

# RISC-V指令格式与寻址方式

图2-19

| Name (Field size) | Field 7 bits | 5 bits | 5 bits | 3 bits | 5 bits | 7 bits | Comments |
|---|---|---|---|---|---|---|---|
| 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 |

| funct6 | immed | rs1 | funct3 | rd | opcode |
|---|---|---|---|---|---|

- 指令格式：6种
  - 基本：R/I/S/U
    - 7种：4+2+“1”
    - IS-type：funct6，立即数移位
  - 规整：Reg和Imm位置固定
    - B/J-type的立即数域？
  - op码与类型绑定
- 寻址方式：4种
  - 本质：Imm，Reg，Base
  - 指令寻址方式
    - PC相对寻址：beq，jal
    - 间接跳转：jalr x0，100(x1)

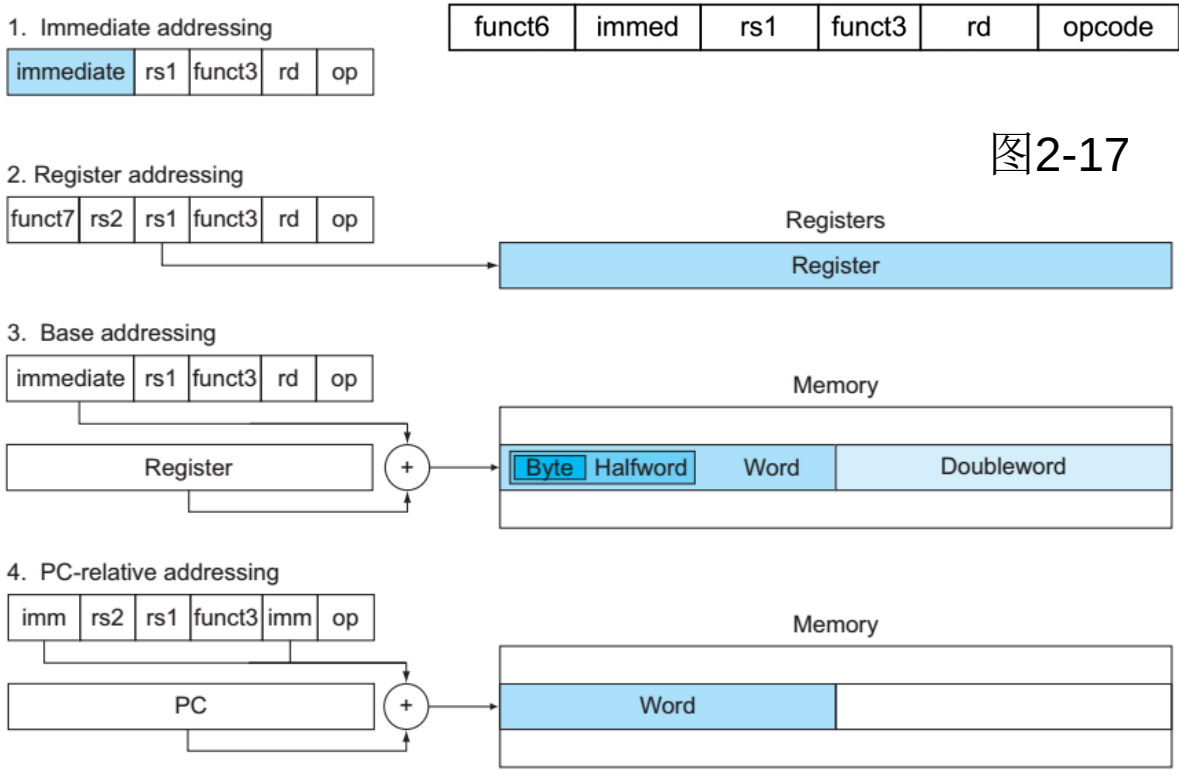
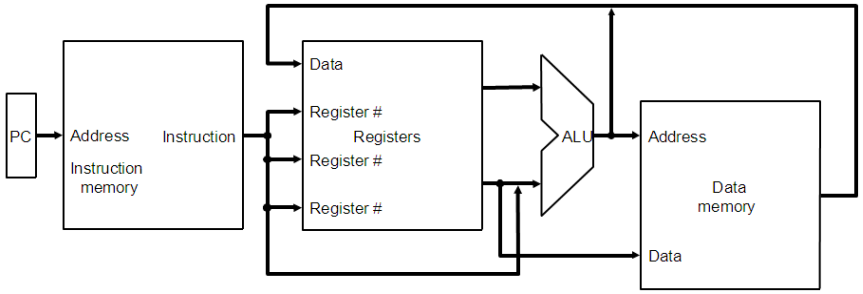
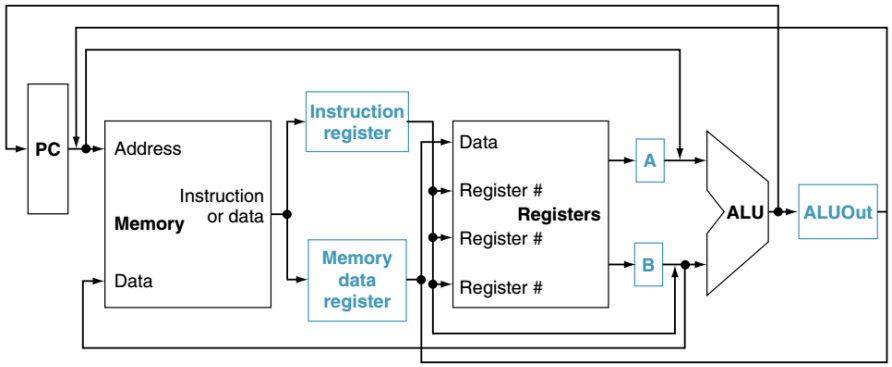


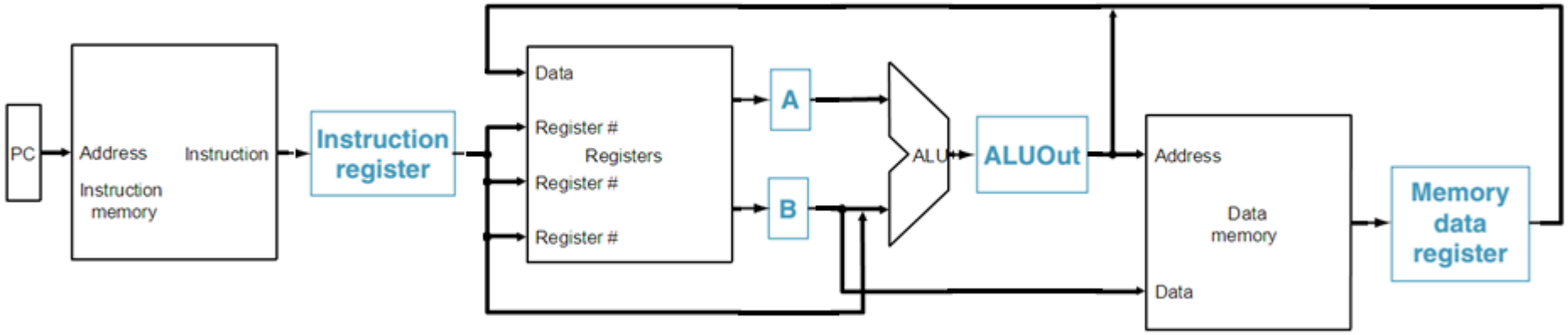

图2-17

# 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

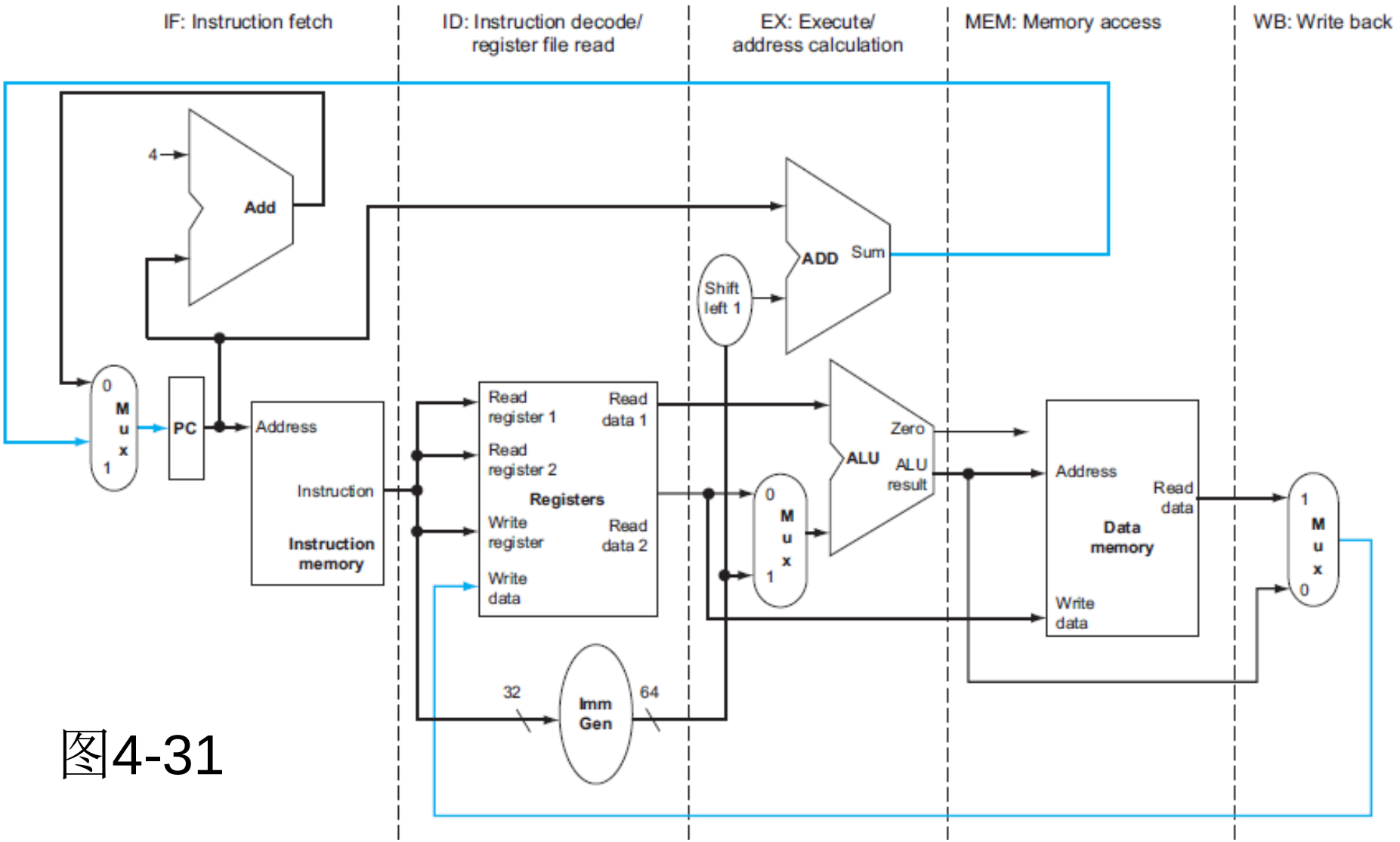
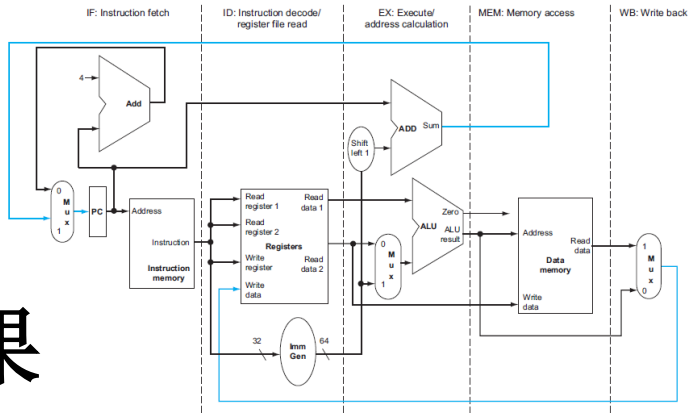



图4-31

# 流水线段寄存器




- 隔离流水段，暂存中间结果

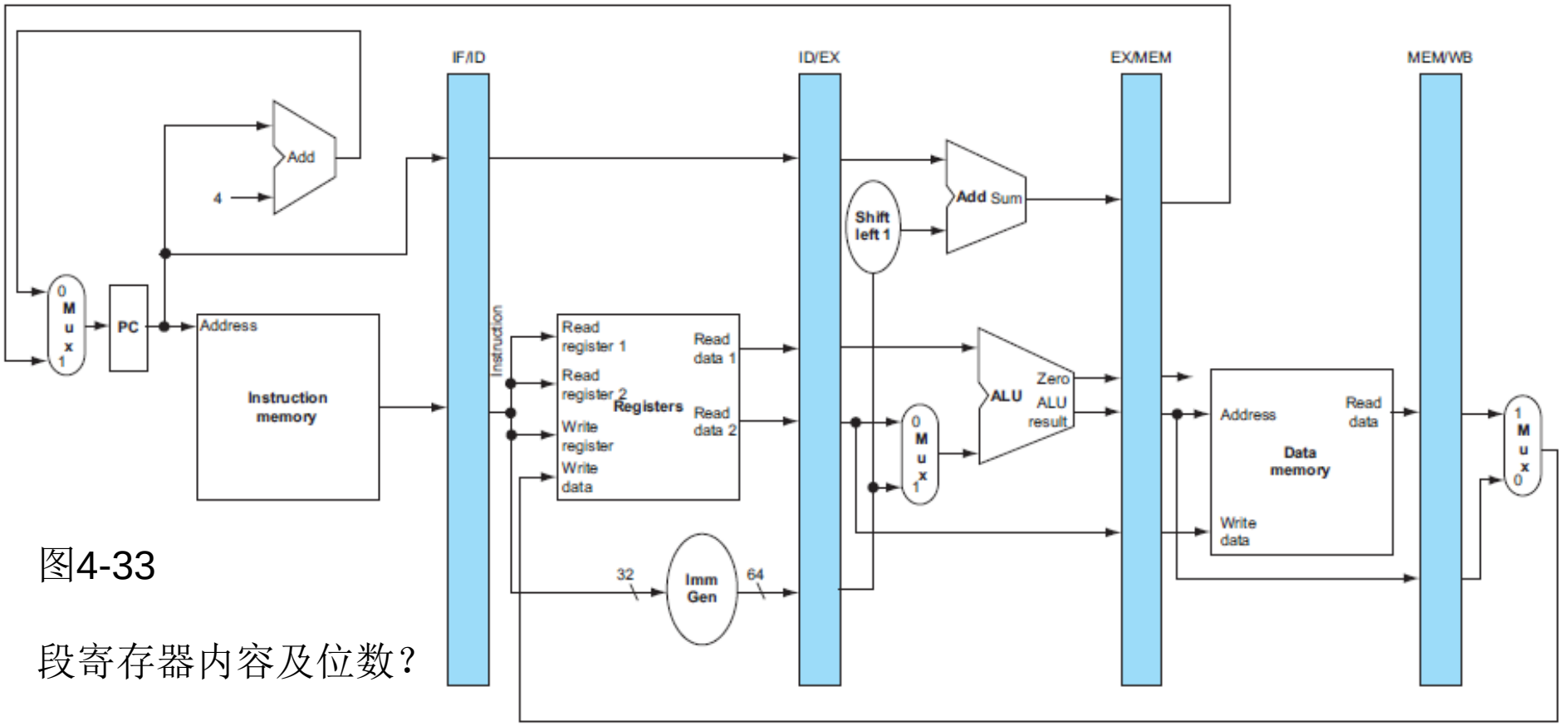



图4-33

段寄存器内容及位数？

# load指令的取指阶段

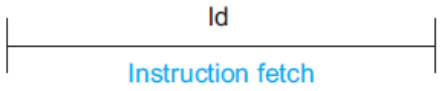
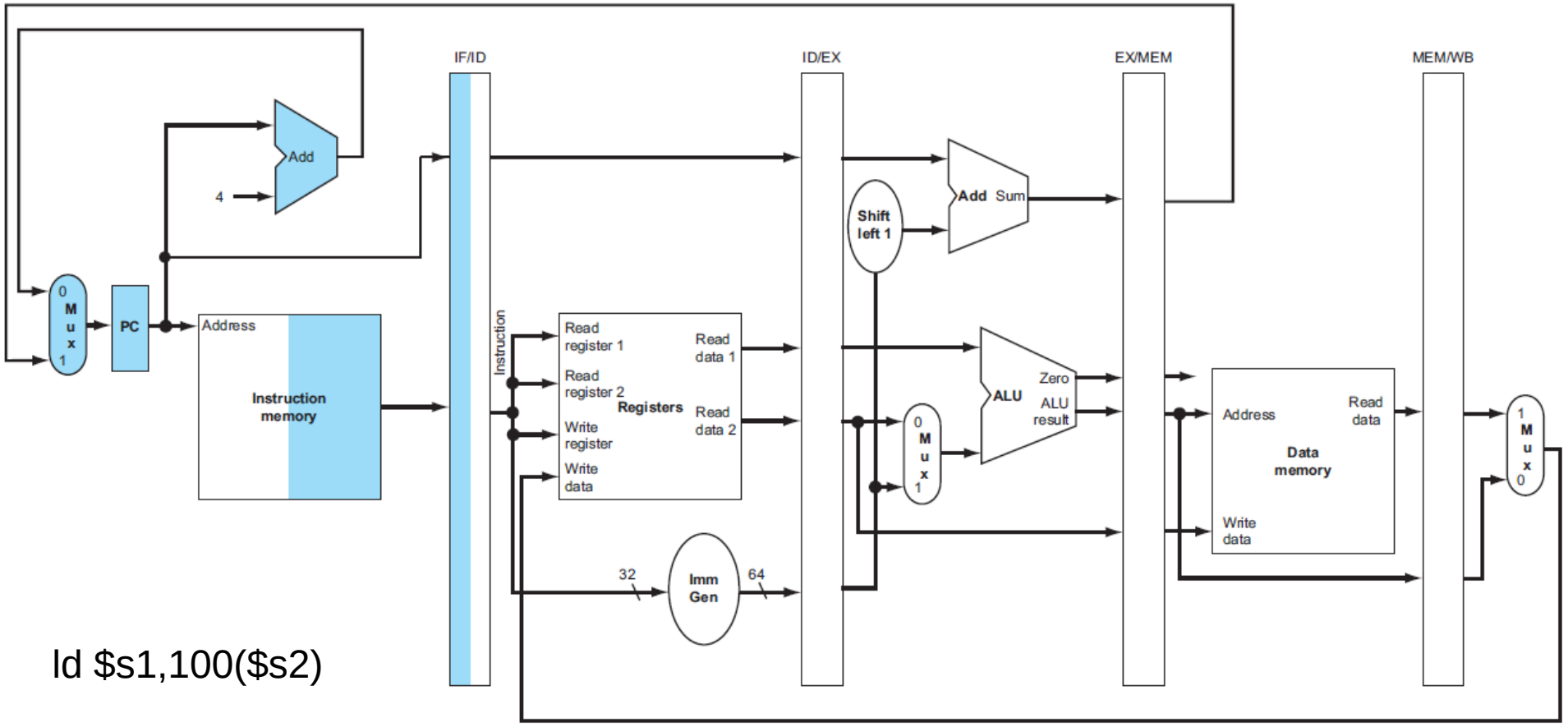
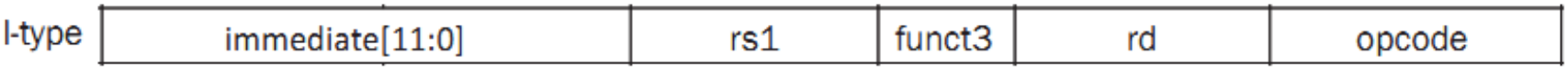



图4-34




ld $s1,100($s2)



# load指令的译码阶段

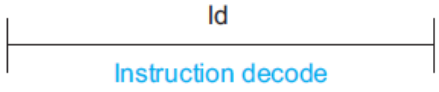
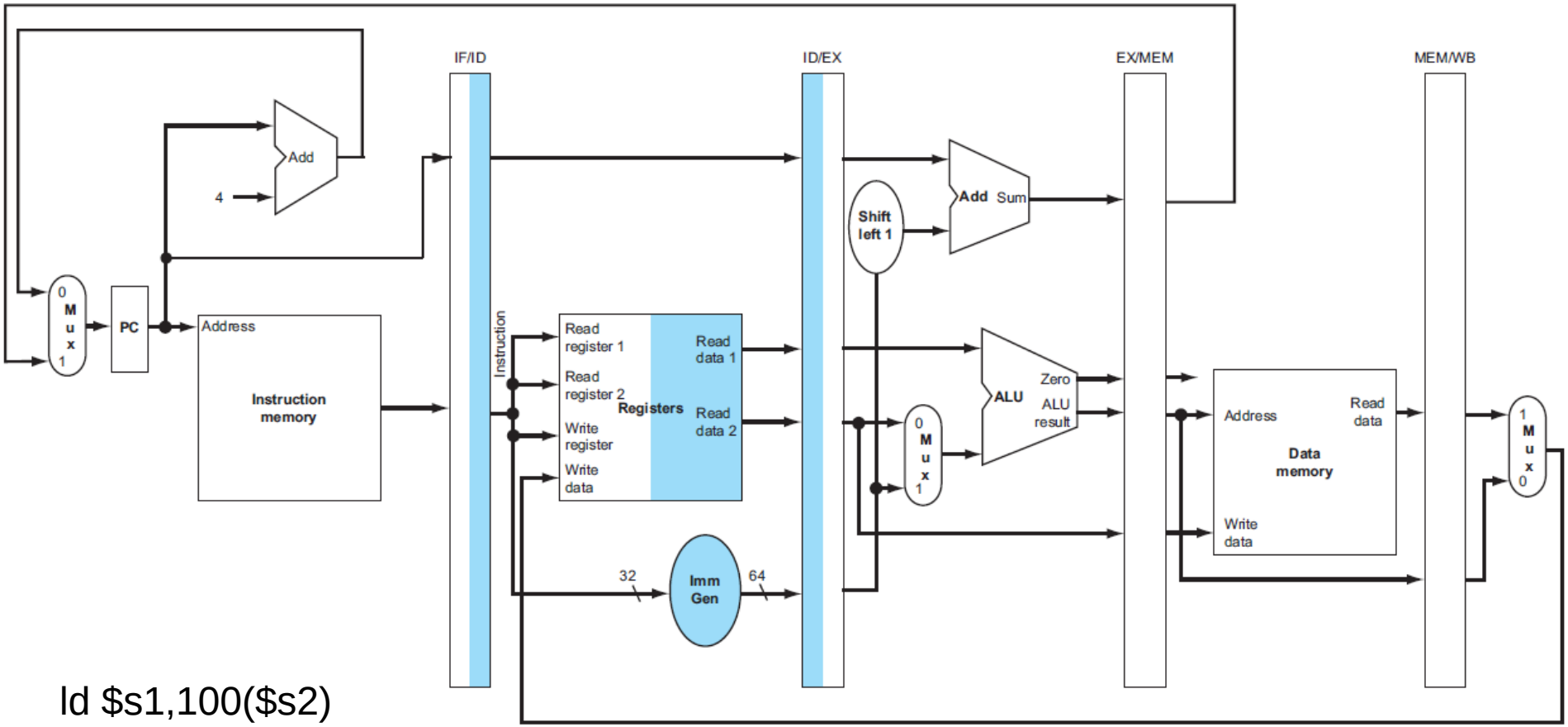


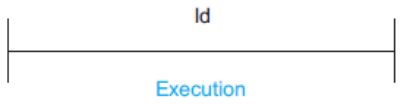

图4-34

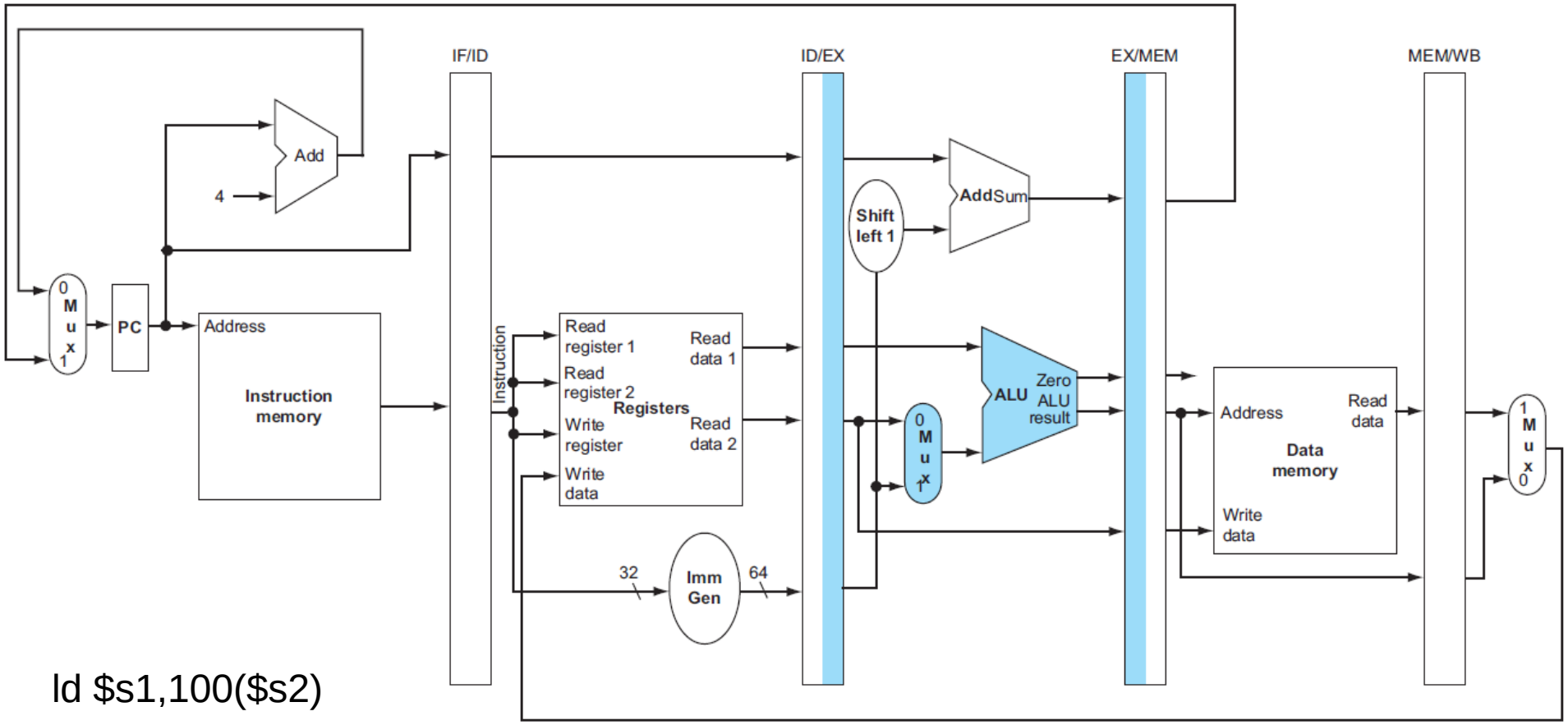



ld $s1,100($s2)

# load指令的执行阶段

图4-35。 adder在干啥？

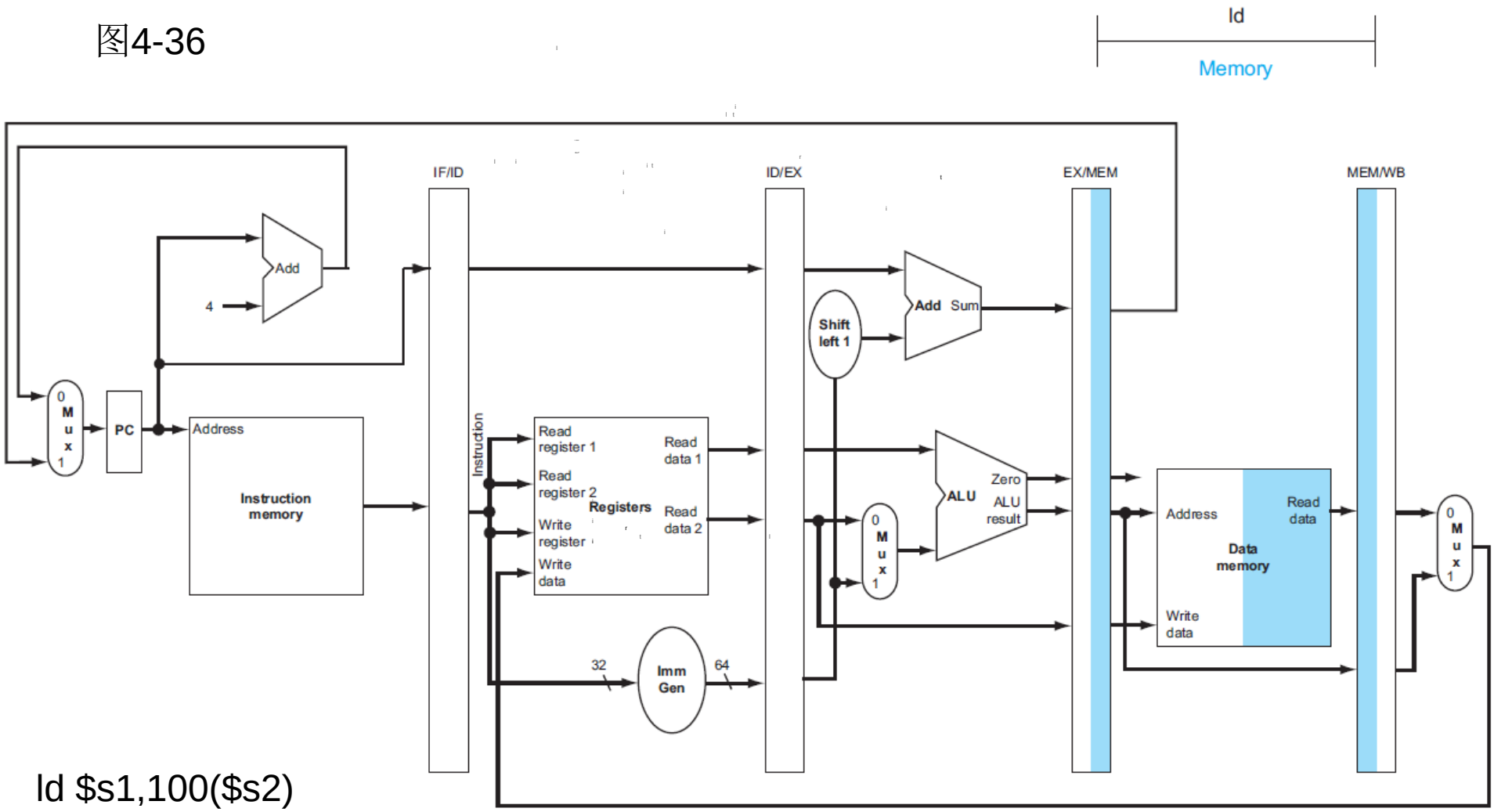



ld $s1,100($s2)

# load指令的访存阶段

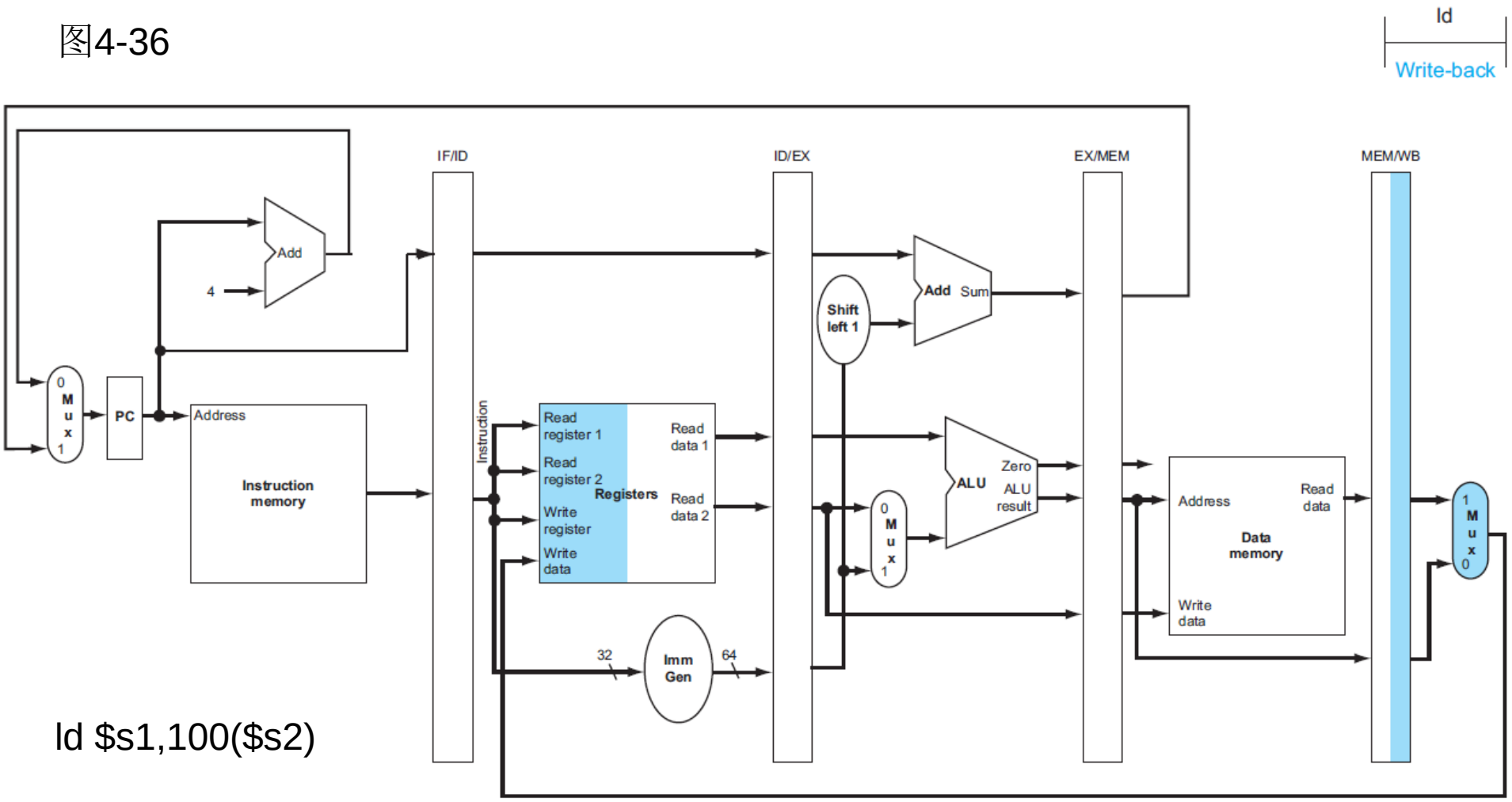
图4-36




ld $s1,100($s2)

# load的写回阶段：bug?

图4-36




ld $s1,100($s2)

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

# 流水线改正版

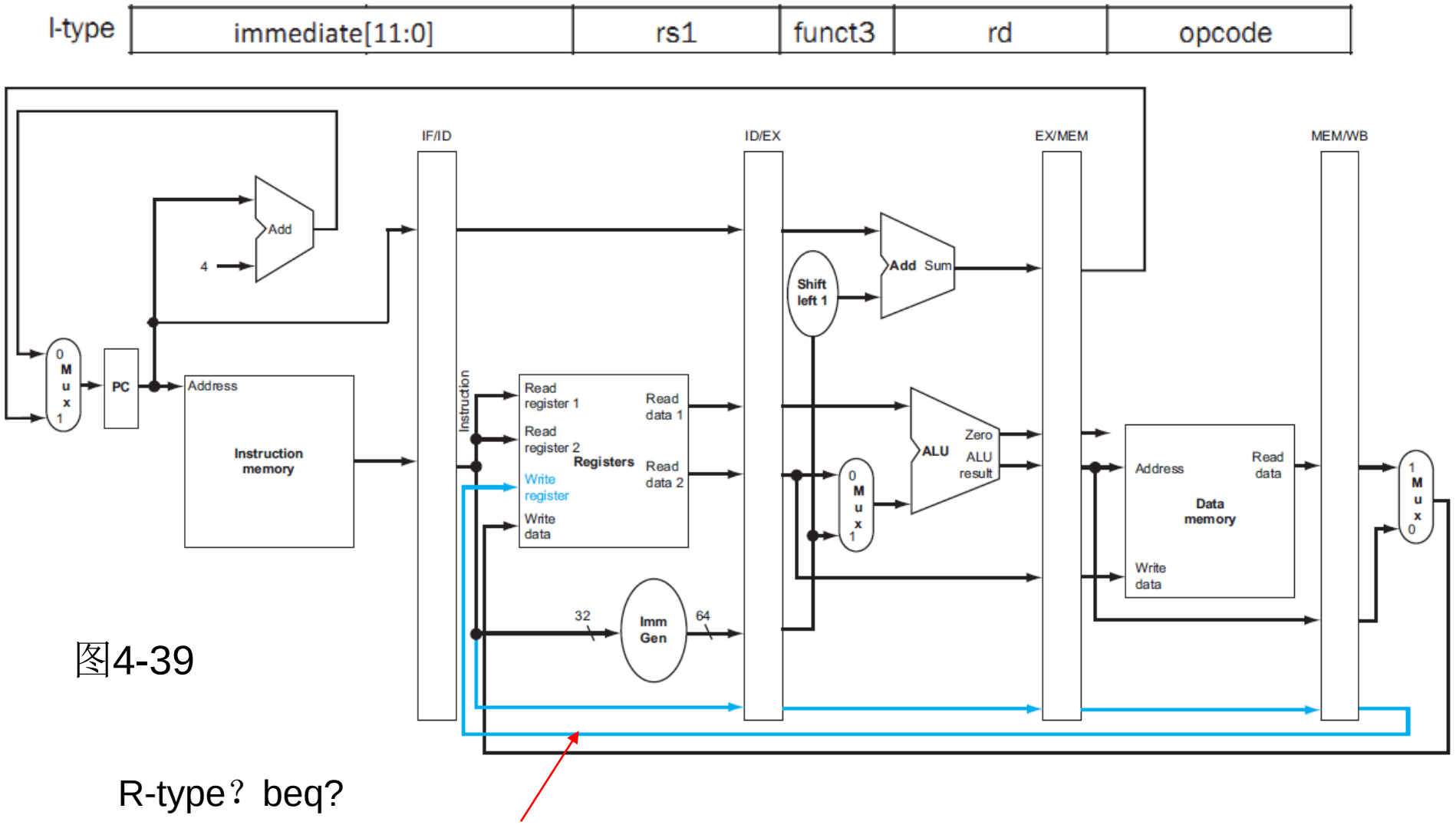
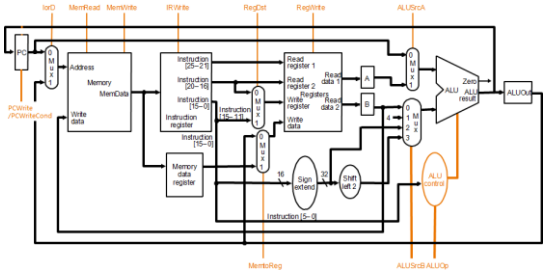
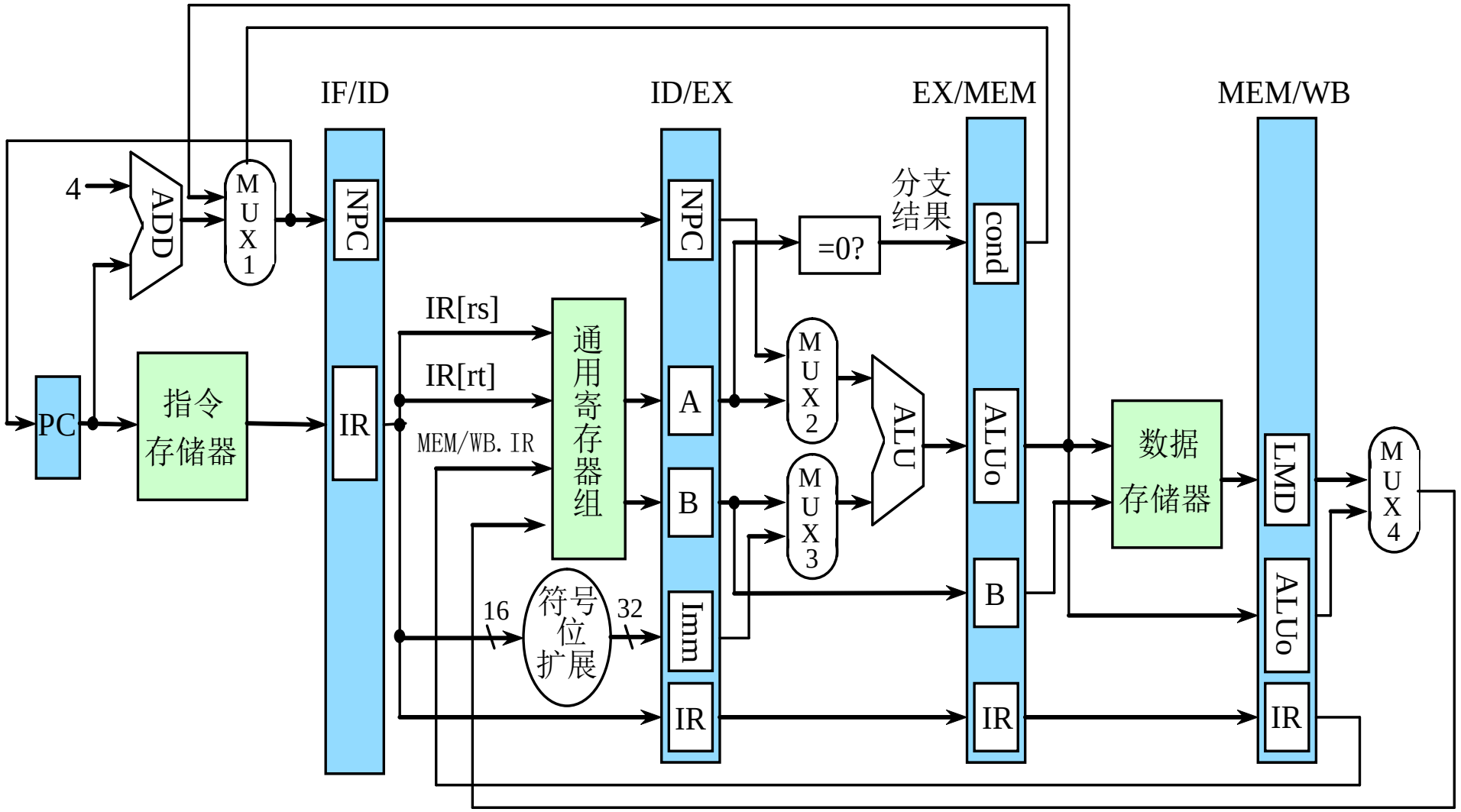



图4-39

R-type？beq?

# DLX的流水段寄存器



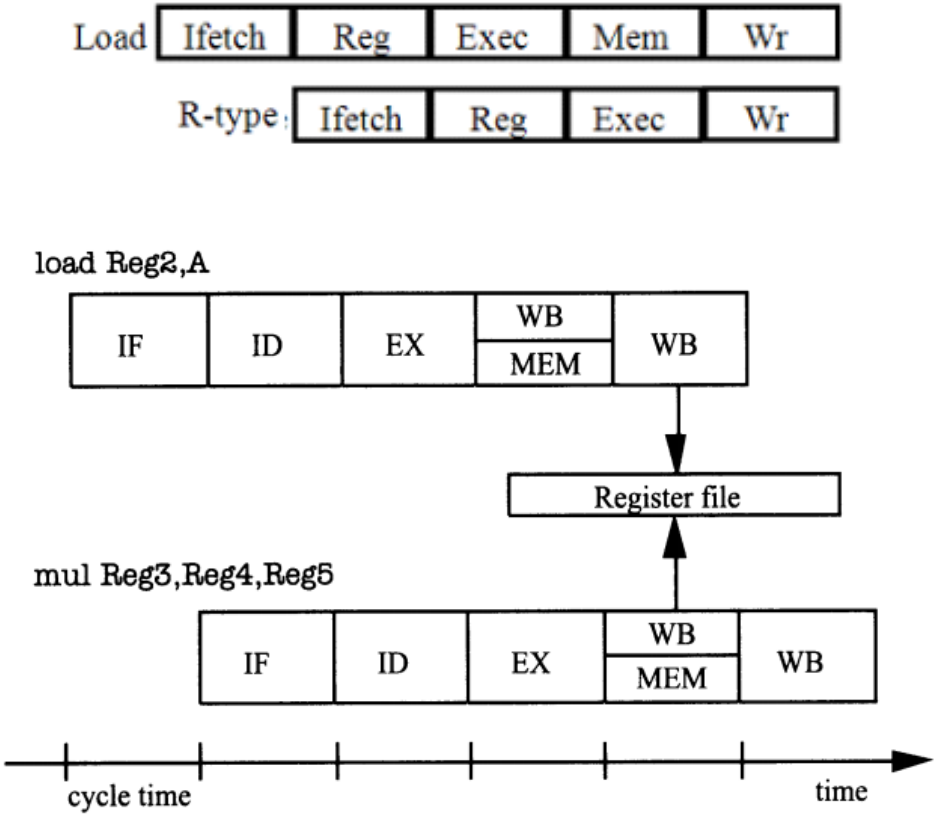

IF/ID
ID/EX
EX/MEM
MEM/WB
4
ADD
MUX 1
NPC
IR
PC
指令存储器
IR[rs]
IR[rt]
MEM/WB. IR
通用寄存器组
16
符号位扩展
32
NPC
A
B
Imm
IR
=0?
分支结果
MUX 2
MUX 3
ALU
cond
ALUo
B
IR
数据存储器
LMD
ALUo
IR
MUX 4

# R-type指令完成时间

- **R-type指令4个周期完成？**
  - **例：当前指令为ld，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 <= Memory[PC]<br>PC <= PC + 4 | | |
| A<= Reg [IR[19:15]]<br>B <= Reg [IR[24:20]]<br>ALUOut <= PC + immediate | | |
| ALUOut <= A op B | ALUOut <= A + immediate | if (A == B)<br>PC <= ALUOut |
| Reg [IR[11:7]] <= ALUOut | Load: MDR <= Memory[ALUOut]<br>or<br>Store: Memory [ALUOut] <= B | |
| | Load: Reg[IR[11:7]] <= MDR | |



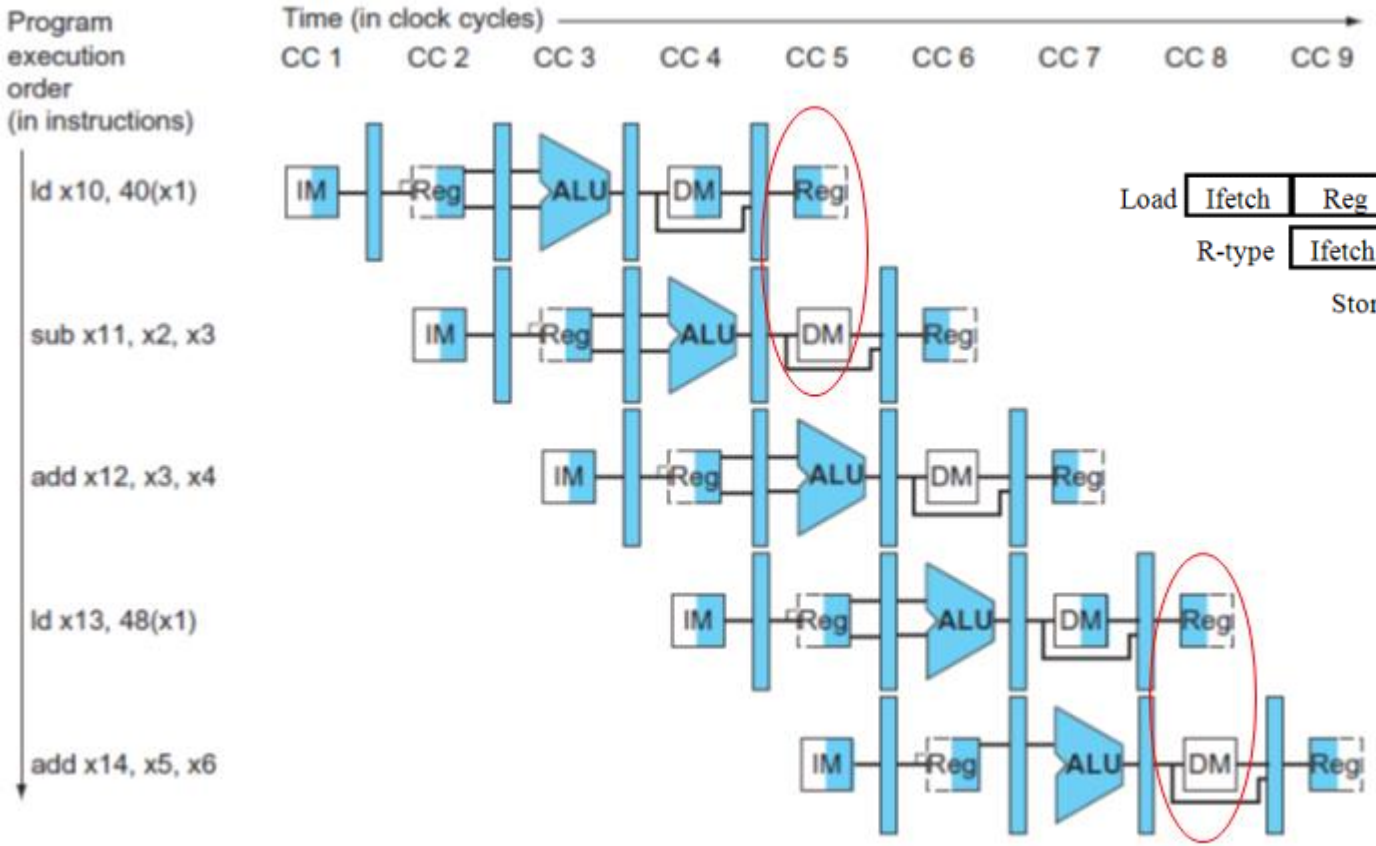

RF“两读一写”，RF写端口冲突！

方案1：

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（$4.5.1）

# R-type指令完成时间

- **R-type指令能否4个周期完成，节省WB段？有意义？**
  - **例：当前指令为ld，下一条指令为R-type。CC5?**



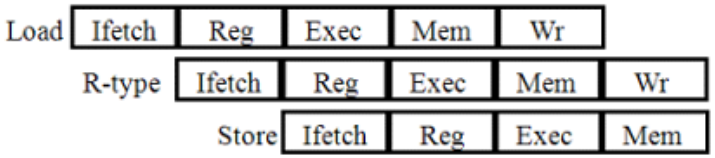




图4-41

# beq指令完成时间？

- 分支以哪一条指令为基址？
  - 单周期
  - 多周期
  - ppl

FIGURE e4.5.4

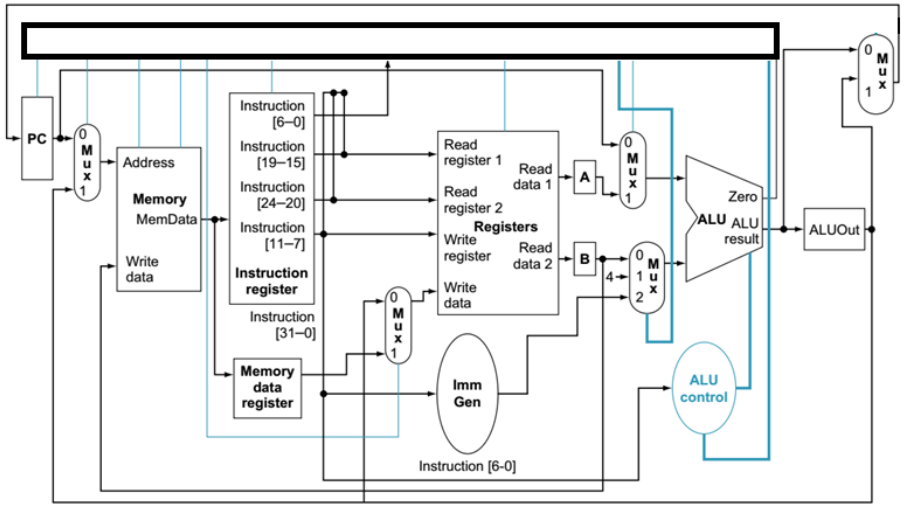



FIGURE e4.5.6

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

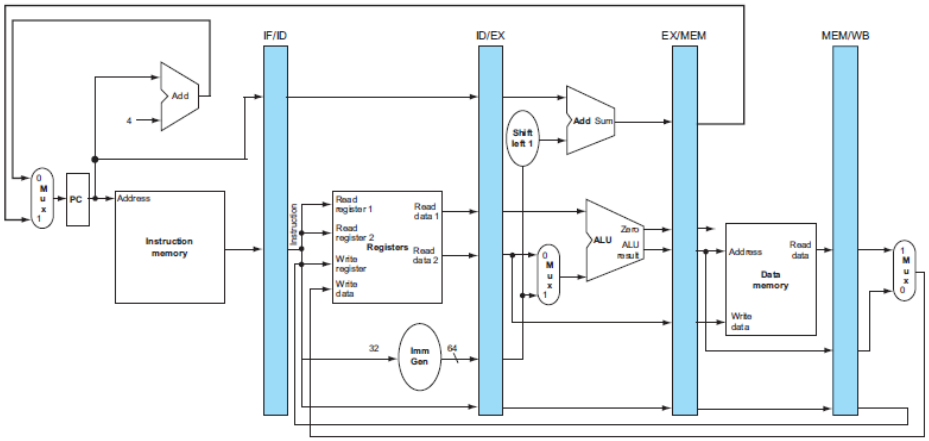



图4-39

# beq指令数据通路与完成时间，$4.8

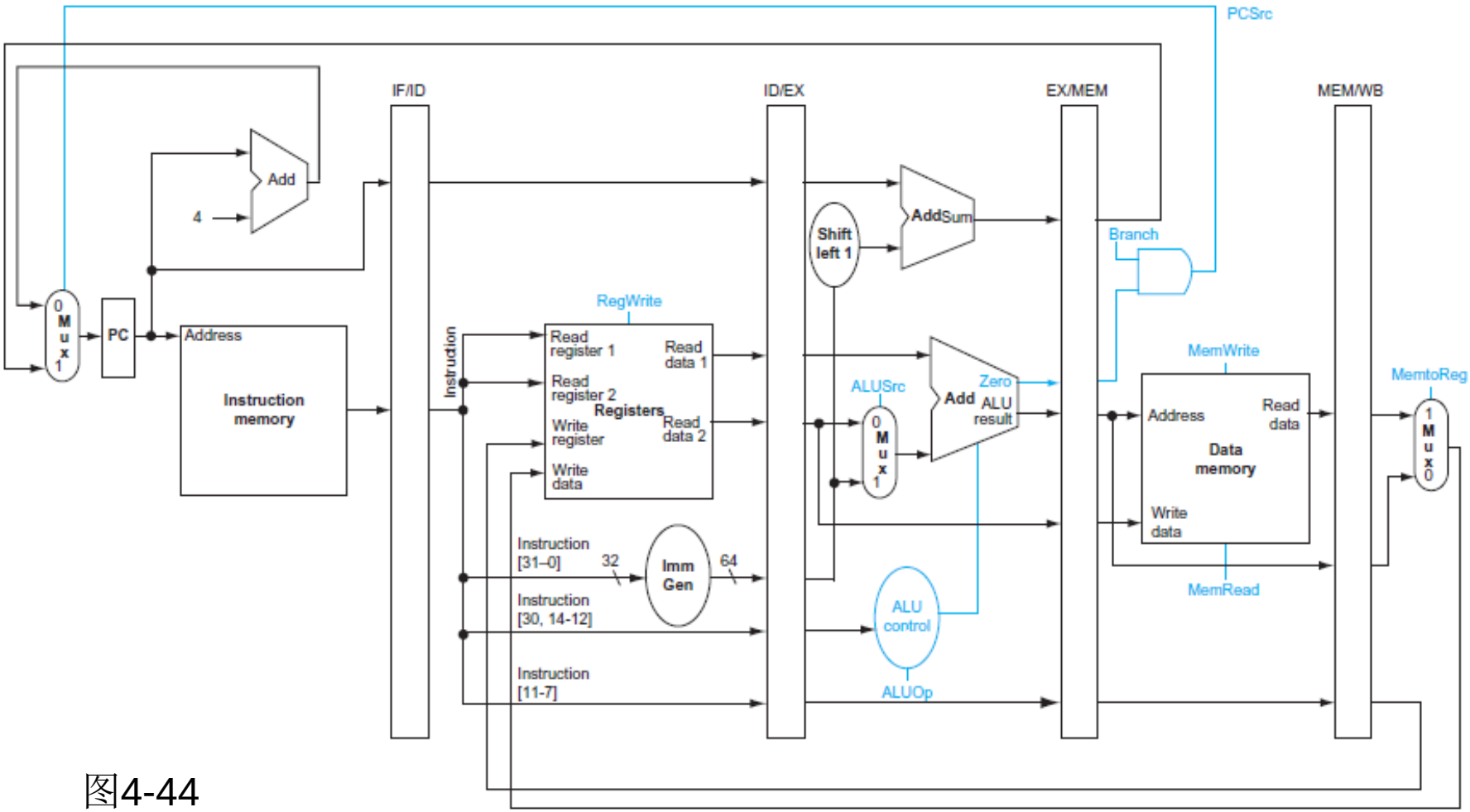
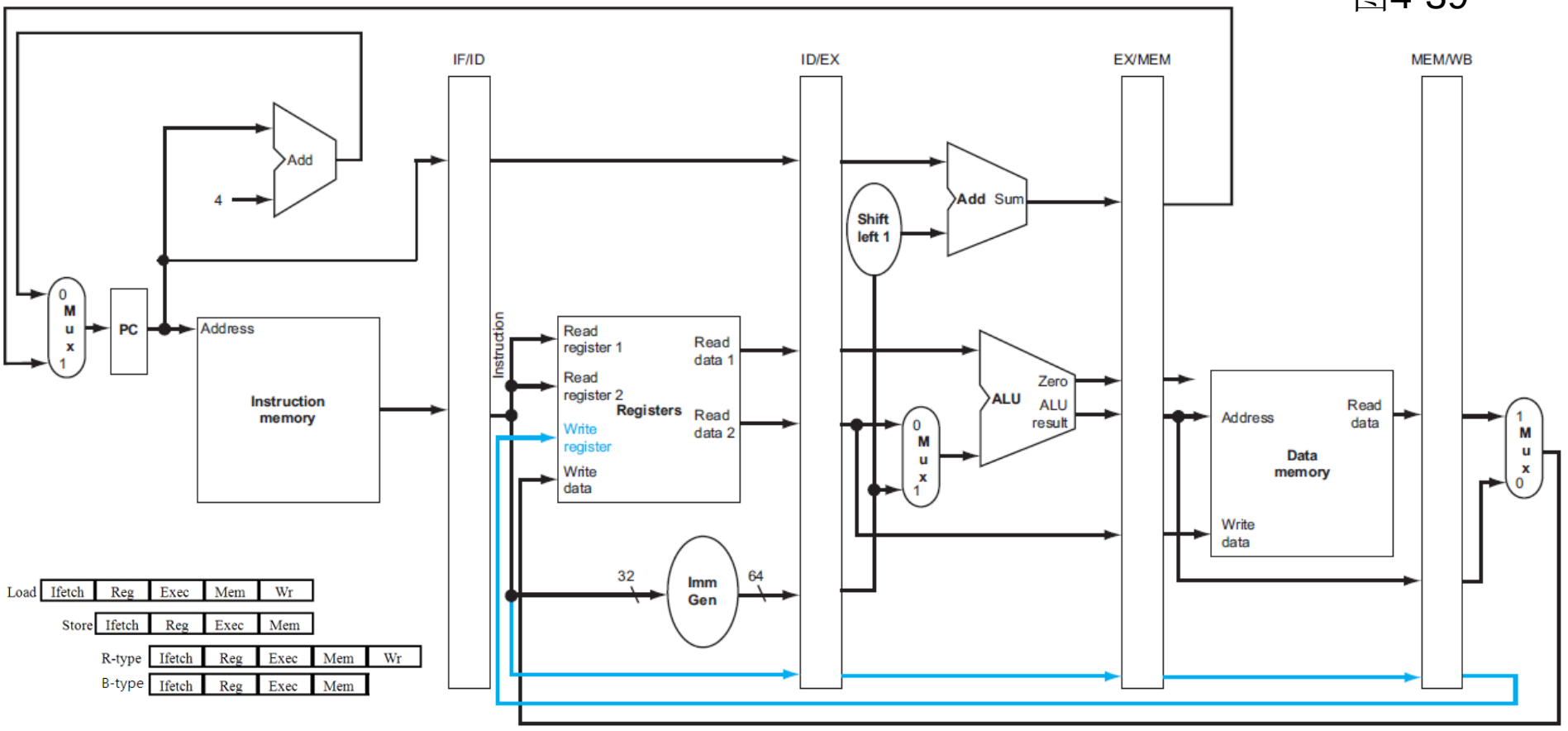



图4-44

# 各指令完成时间

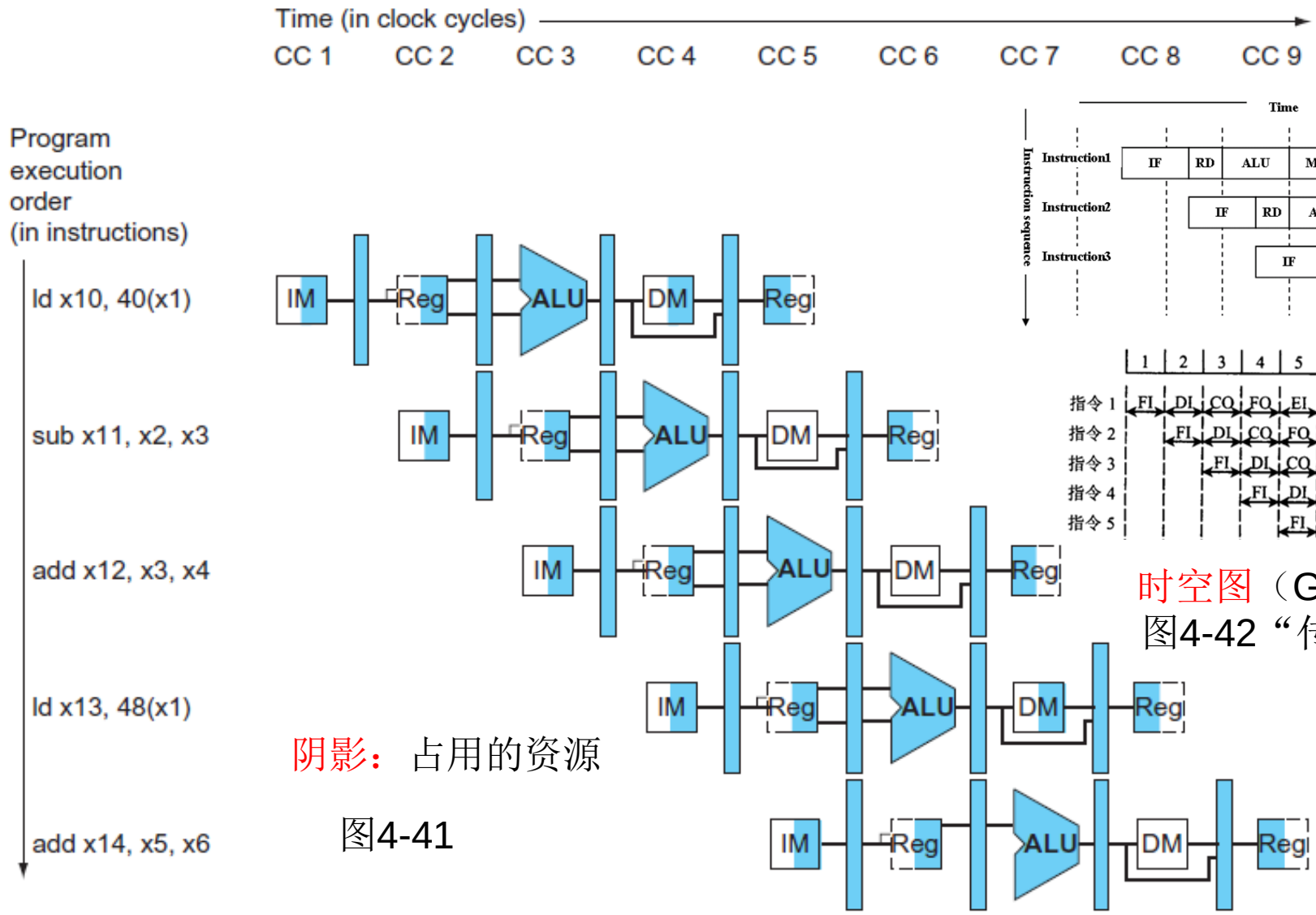
图4-39




“指令周期定长”？每个周期流出一条指令？ CPI？IPC？

中文版p200/原版p284：“an instruction passes through a stage even if there is nothing to do”？

# Multiple-clock-cycle Pipeline Diagram，$4.6.1




阴影：占用的资源

图4-41




时空图（Gantt chart）

图4-42“传统表示”

# Single-clock-cycle Diagram: CC5，$4.6.1

| add $14, $5, $6 | lw $13, 24 ($1) | add $12, $3, $4, $11 | sub $11, $2, $3 | lw$10, 20($1) |
|---|---|---|---|---|
| Instruction fetch | Instruction decode | Execution | Memory | Write back |

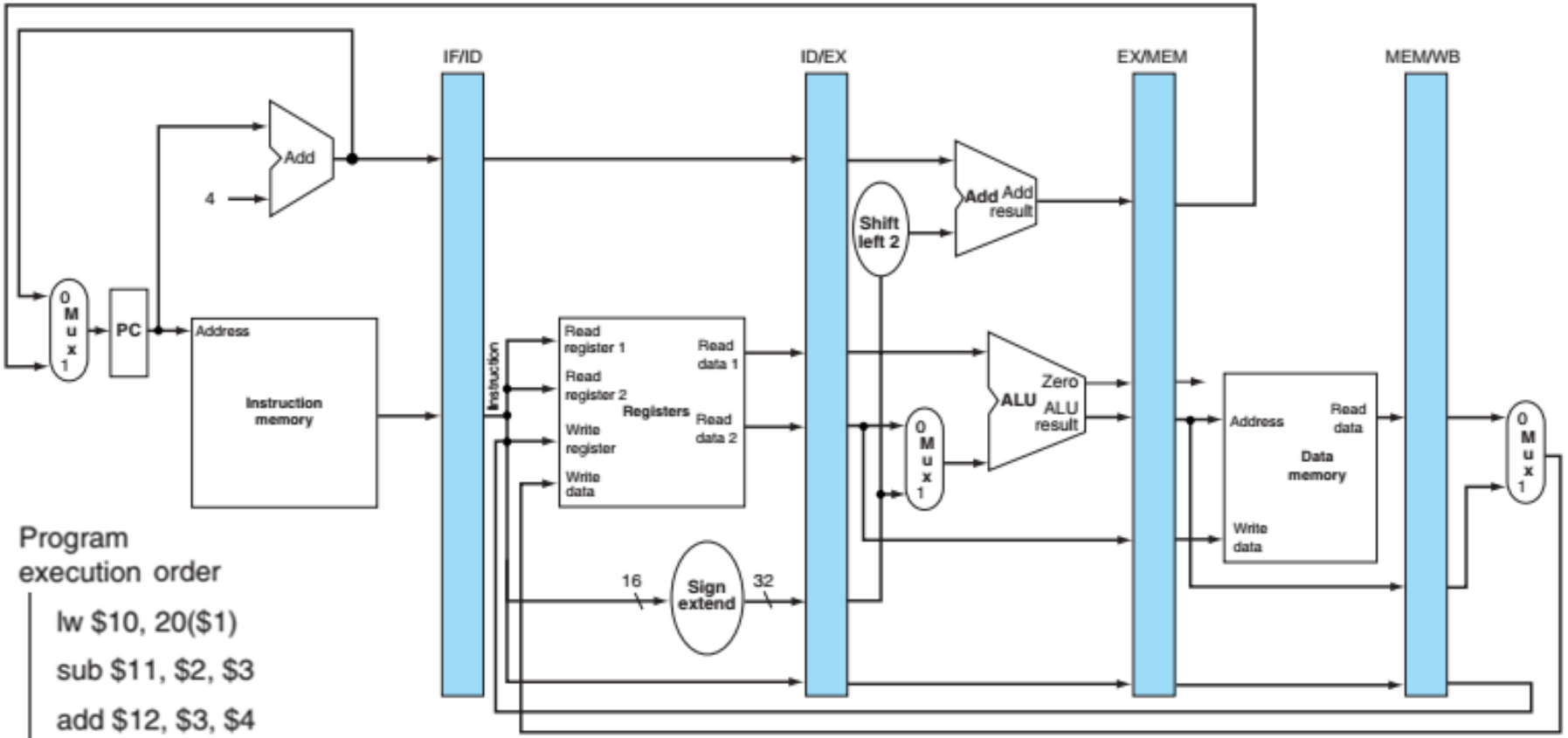



图4-43

# 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 |
| | 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 |

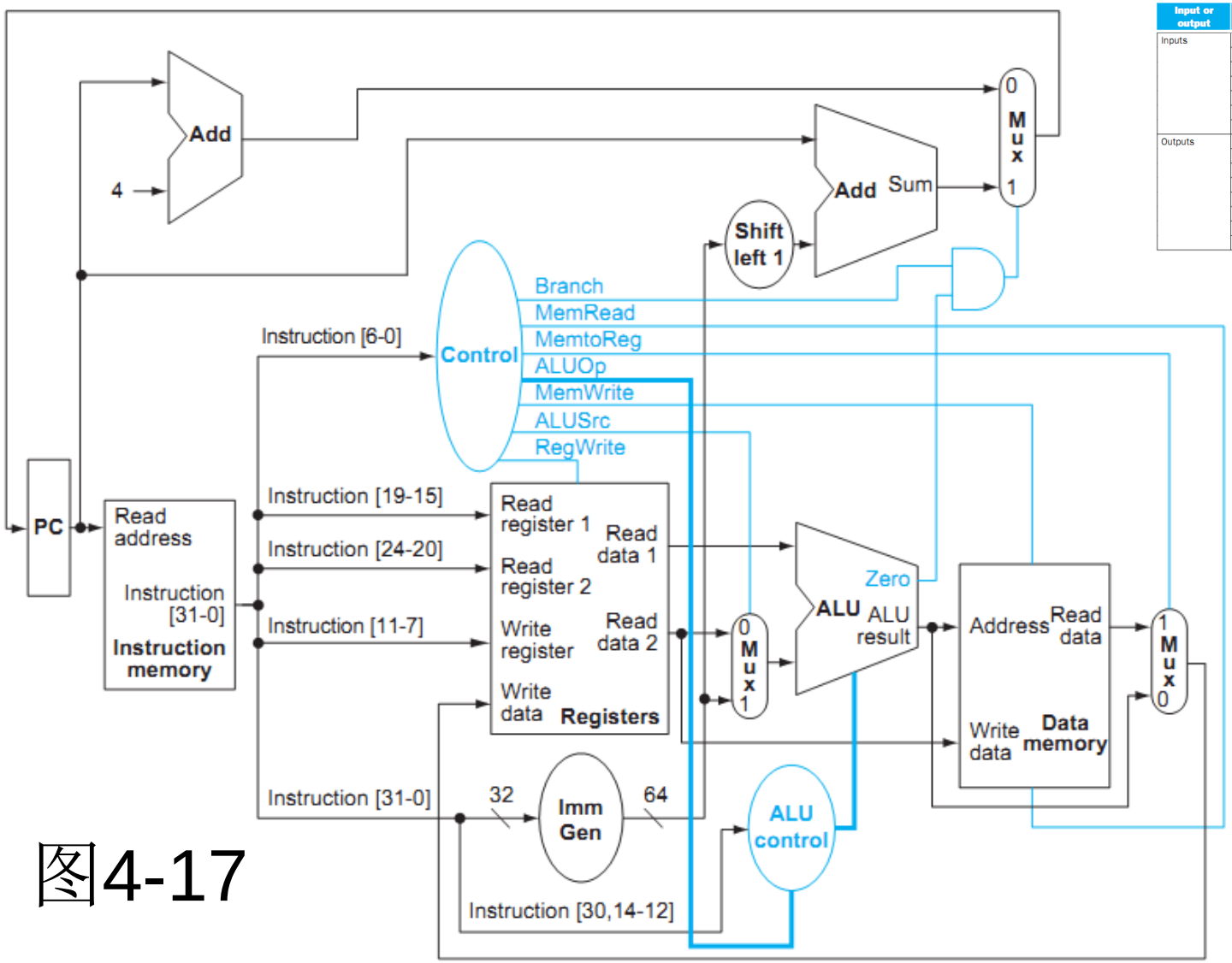
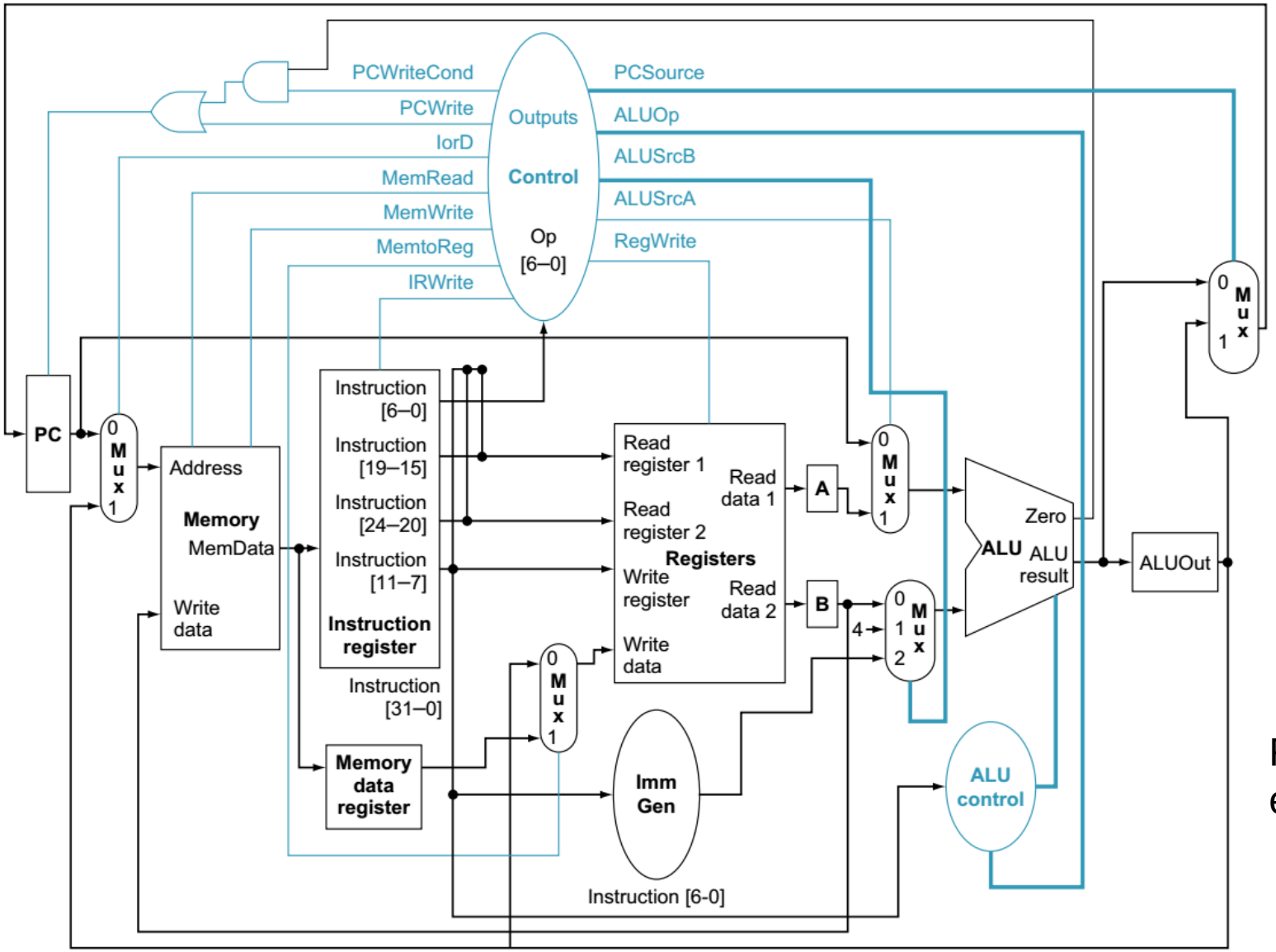



图4-17

# 流水线控制信号？



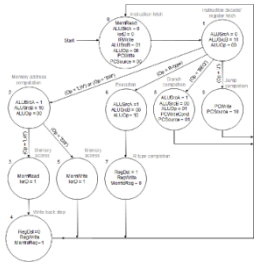



FIGURE
e4.5.4

# 流水线所需控制信号

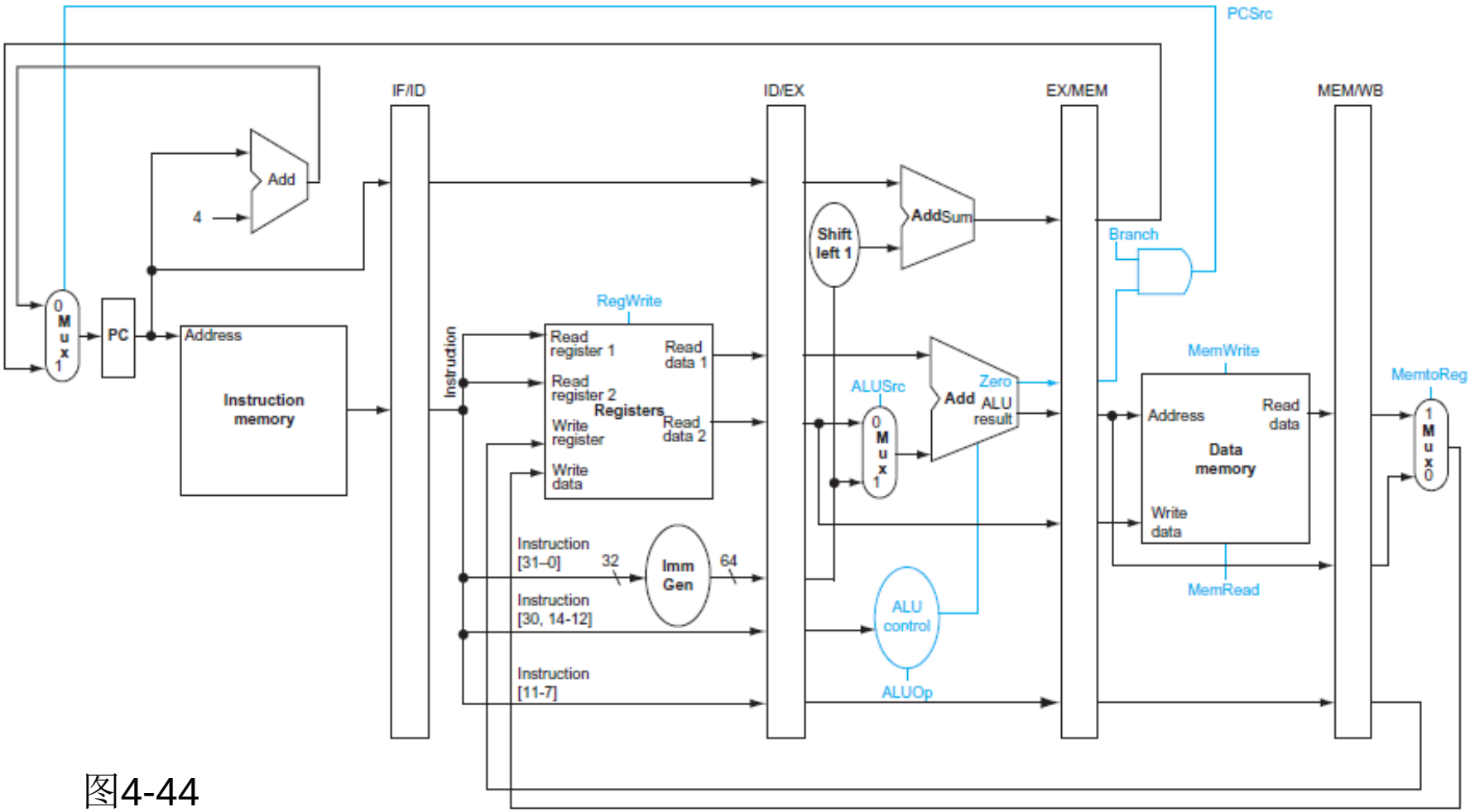
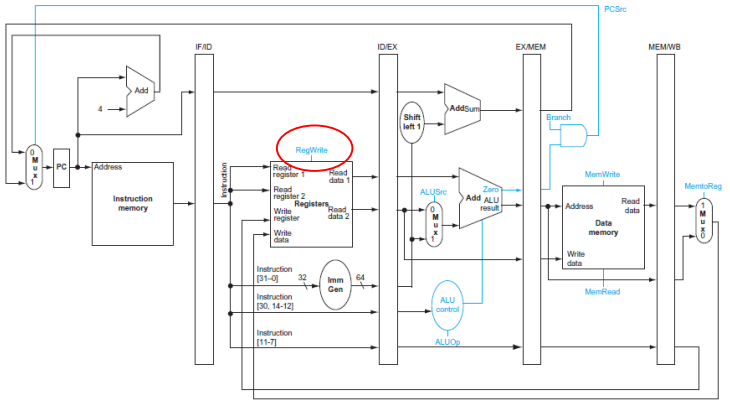



图4-44

# 流水线控制信号：RV $4.6.2

- 所有控制信号名及其功能与单周期版相同：7个
  - 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
  - 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

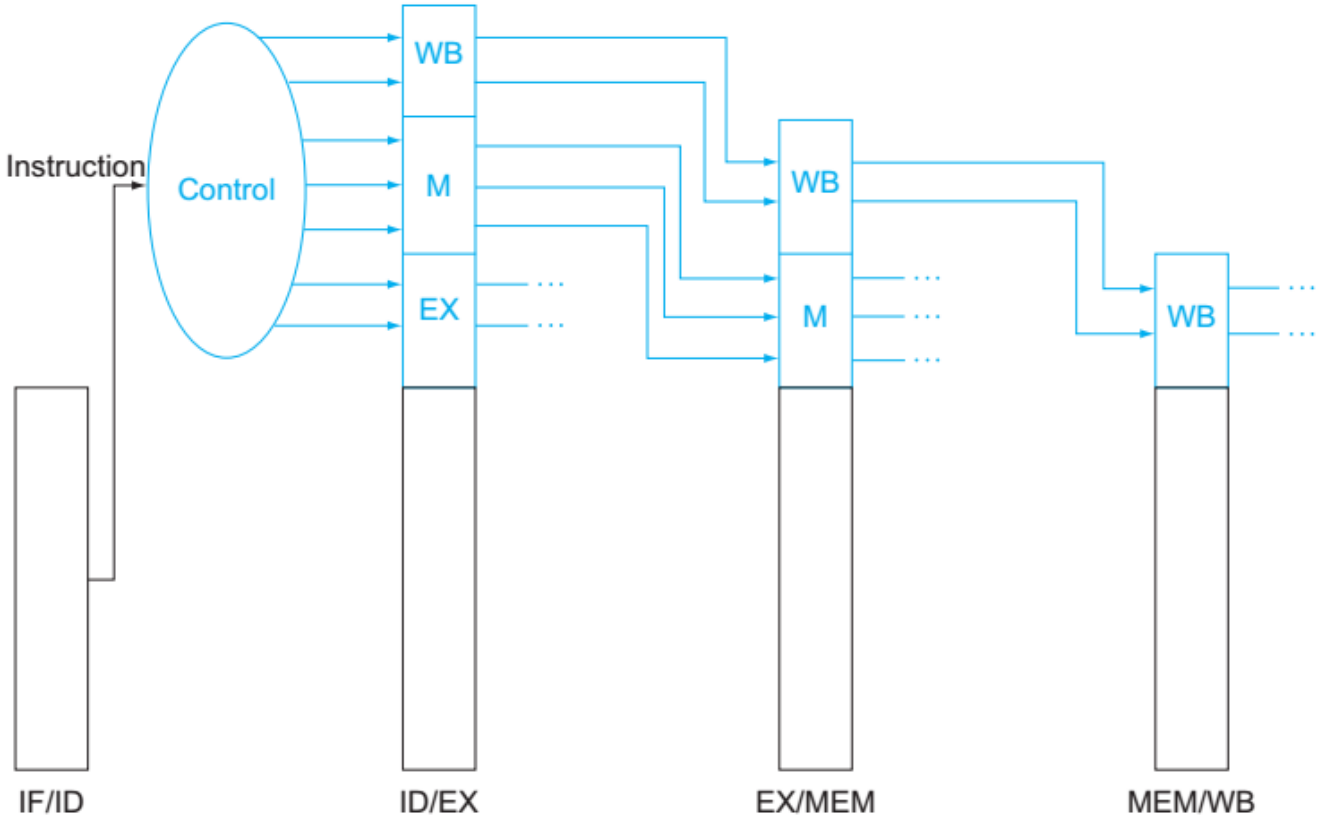



| Instruction | Execution/address calculation stage control lines | | Memory access stage control lines | | | Write-back stage control lines | |
|---|---|---|---|---|---|---|---|
| | ALUOp | ALUSrc | Branch | Mem-Read | Mem-Write | Reg-Write | Memto-Reg |
| R-format | 10 | 0 | 0 | 0 | 0 | 1 | 0 |
| ld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

图4-47

# 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 |
| ld | 00 | 1 | 0 | 1 | 0 | 1 | 1 |
| sd | 00 | 1 | 0 | 0 | 1 | 0 | X |
| beq | 01 | 0 | 1 | 0 | 0 | 0 | X |




- 需要控制缓冲：分段使用7个控制信号

# the control of the pipeline register

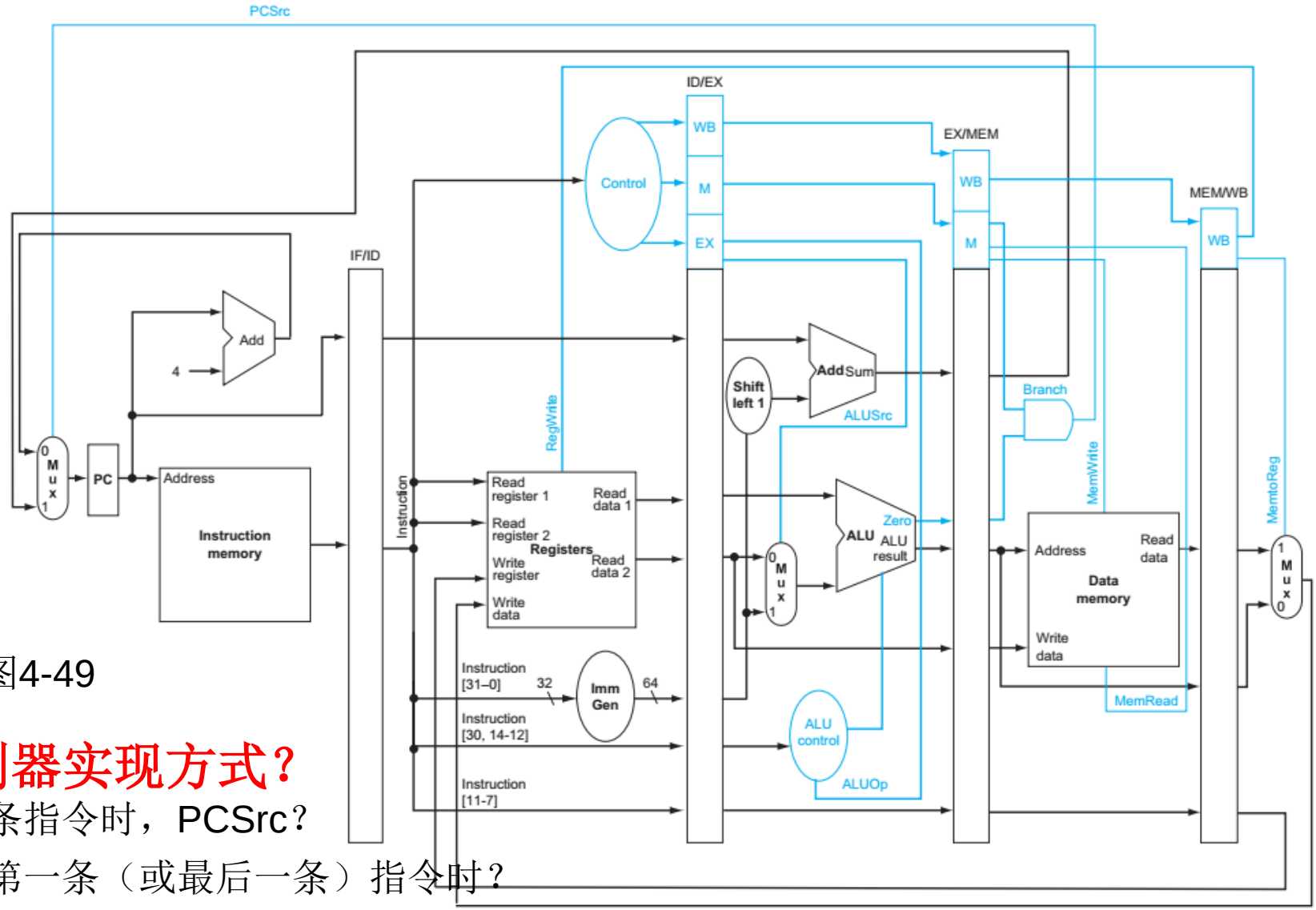



图4-49

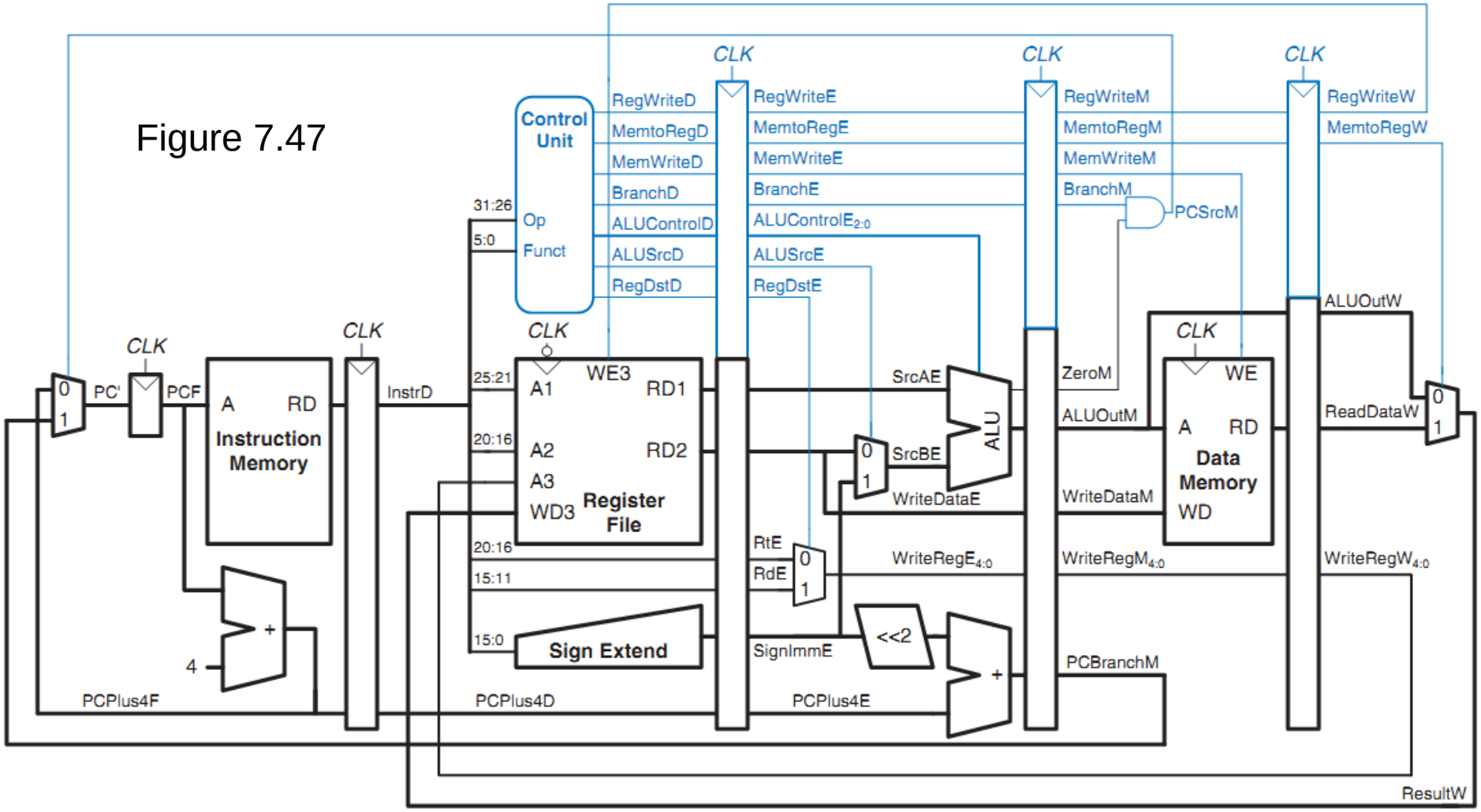
**控制器实现方式？**

第一条指令时，PCSrc？

执行第一条（或最后一条）指令时？

# Pipelined processor with clock

Figure 7.47



# 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？

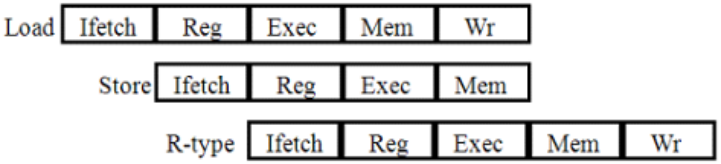
Michel Dubois, et al. Parallel Computer Organization and Design (2012)

# 单周期实现 vs. 流水线性能

- 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8ns×3=24ns
    - 第4条指令与第1条指令之间的时间间隔=8ns×3=24ns
  - 流水线段，按操作时间最长的阶段定义，2ns
    - 执行3条指令所需的时间为2ns×5＋2ns＋2ns=14ns
    - 第4条指令与第1条指令之间的时间间隔=2ns×3=6ns
  - 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？

# 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ISA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
  - 影响流水线性能发挥的因素有哪些?
  - 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？
- 作业
  - 4.16.1~3，4.23


Thank you