

FSE598 前沿计算技术

模块 1 计算思维

单元 1 计算机系统设计

第 4 讲 数据路径与控制单元

本讲座的英文版内容基于教材：

The English version of the lectures are partly based on the book:

Patterson and Hennessy, Computer Organization and Design: The Hardware Software Interface

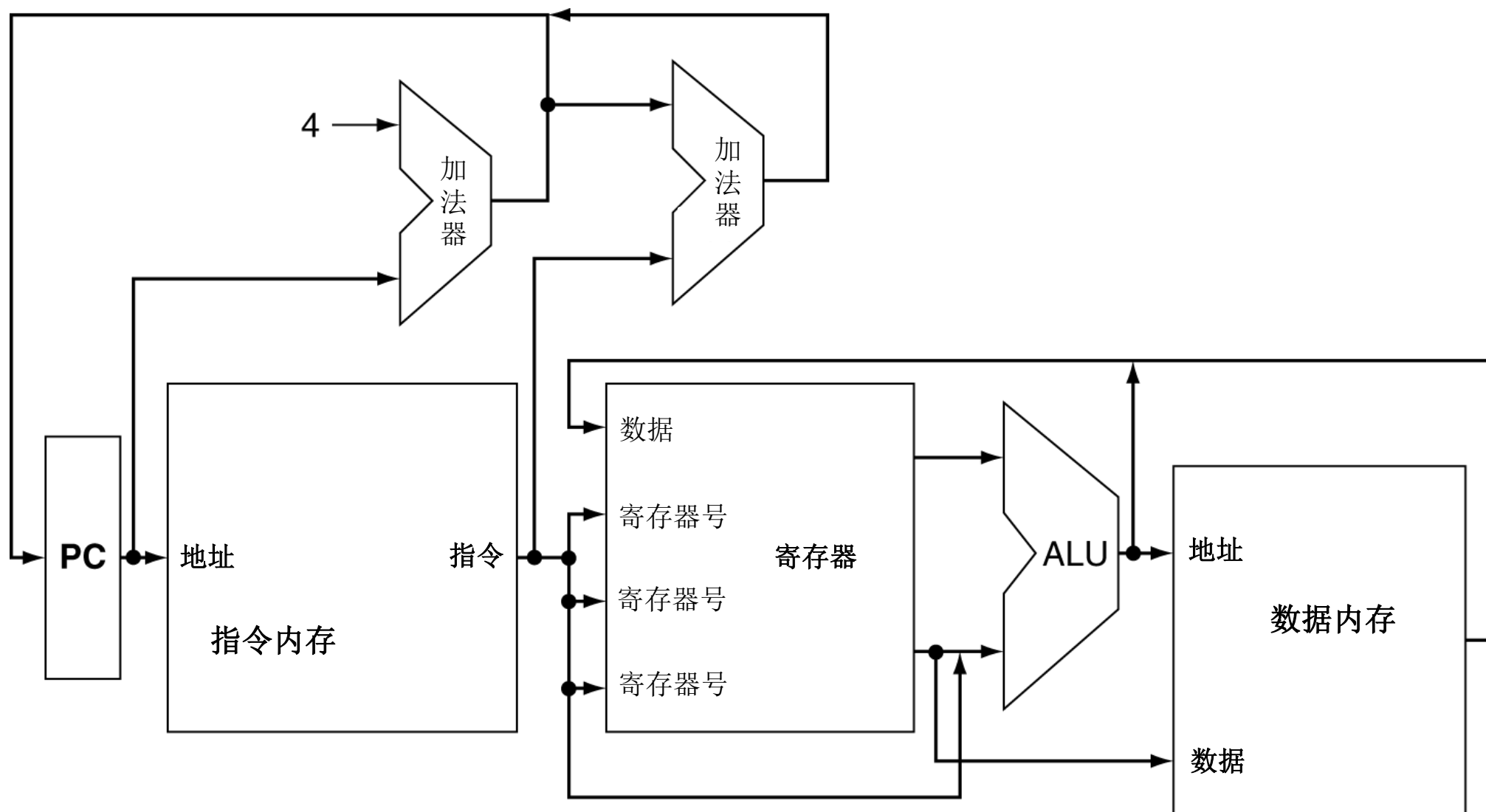
学习

- ❑ 指令执行：五个步骤
- ❑ 处理器概念模型
- ❑ 数据路径与内存
- ❑ 不同指令的执行
- ❑ 控制信号

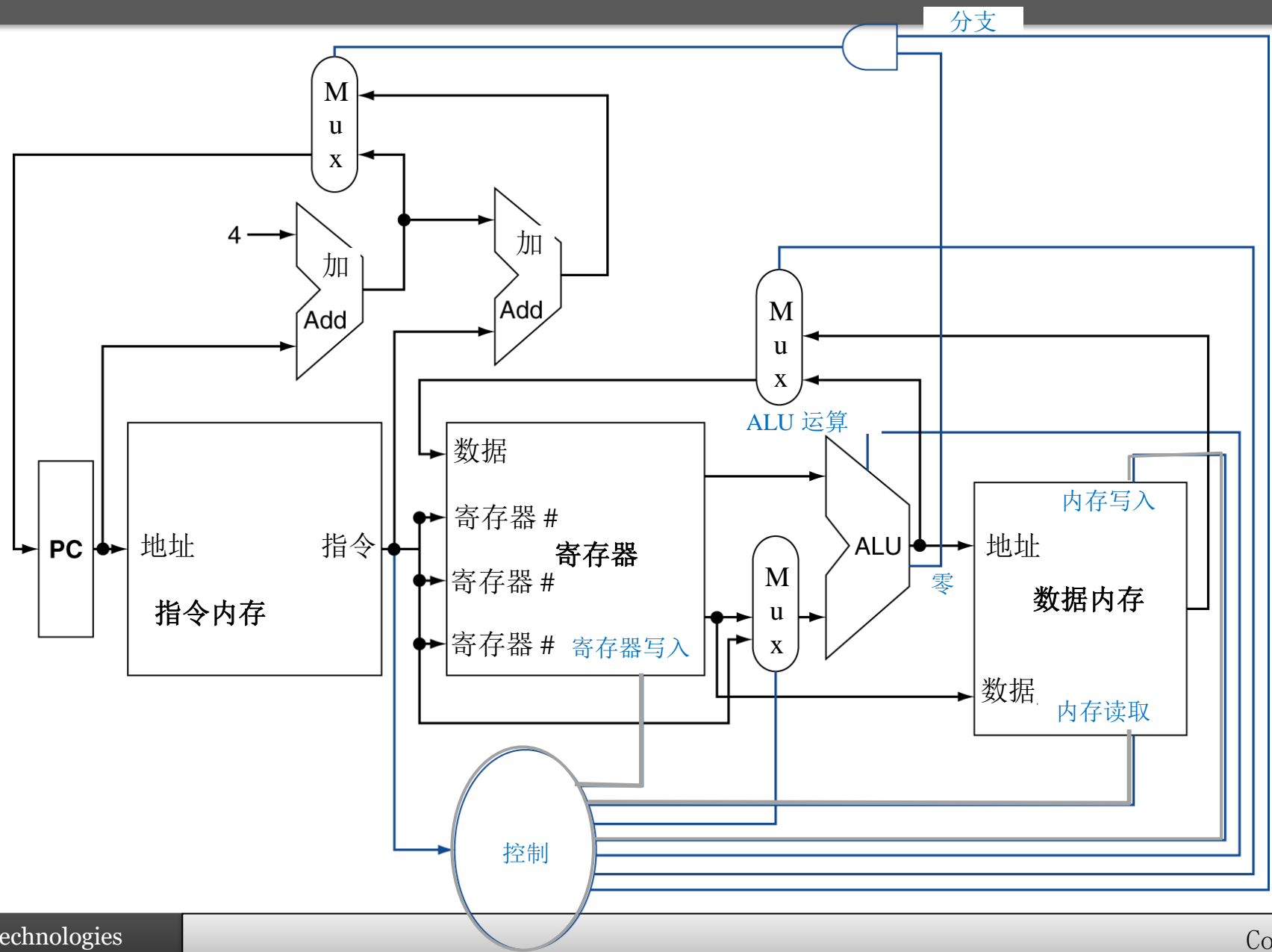
指令执行：五个步骤

1. $PC \rightarrow$ 指令内存, 获取指令,
和 $PC \leftarrow PC + 4$
2. 寄存器编号 \rightarrow 寄存器文件, 读取寄存器
3. 取决于指令类型
 - 使用 ALU 执行
 - 算术结果
 - 加载/存储的内存地址
 - 分支目标地址
4. 访问加载/存储的数据内存
5. 将结果写回寄存器文件

CPU 概念模型：数据路径与内存



包含多路转接器 (Mux) 和控制单元



逻辑设计基础：回顾

□ 二进制编码的信息

- 低压 = 0, 高压 = 1
- 每个位连一条线 —————
- 在多线总线上编码的多位数据



□ 组合元件和电路

- 对数据进行操作
- 输出是输入的函数

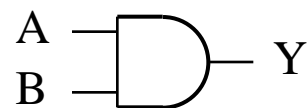
□ 状态元件和时序电路

- 将信息存储在状态（内存）中
- 输出是输入和状态的函数

组合元件示例

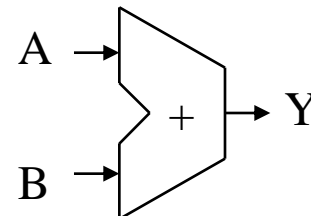
□ 与门 (and-gate)

- $Y = A \& B$



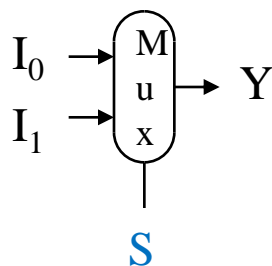
■ 加法器

- $Y = A + B$



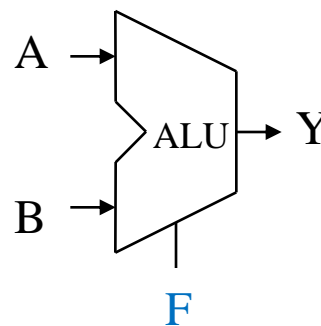
■ 多路转接器

- $Y = (S == 0) ? I_0 : I_1$



■ 算术/逻辑单元

- $Y = \text{fun}(A, B, F)$



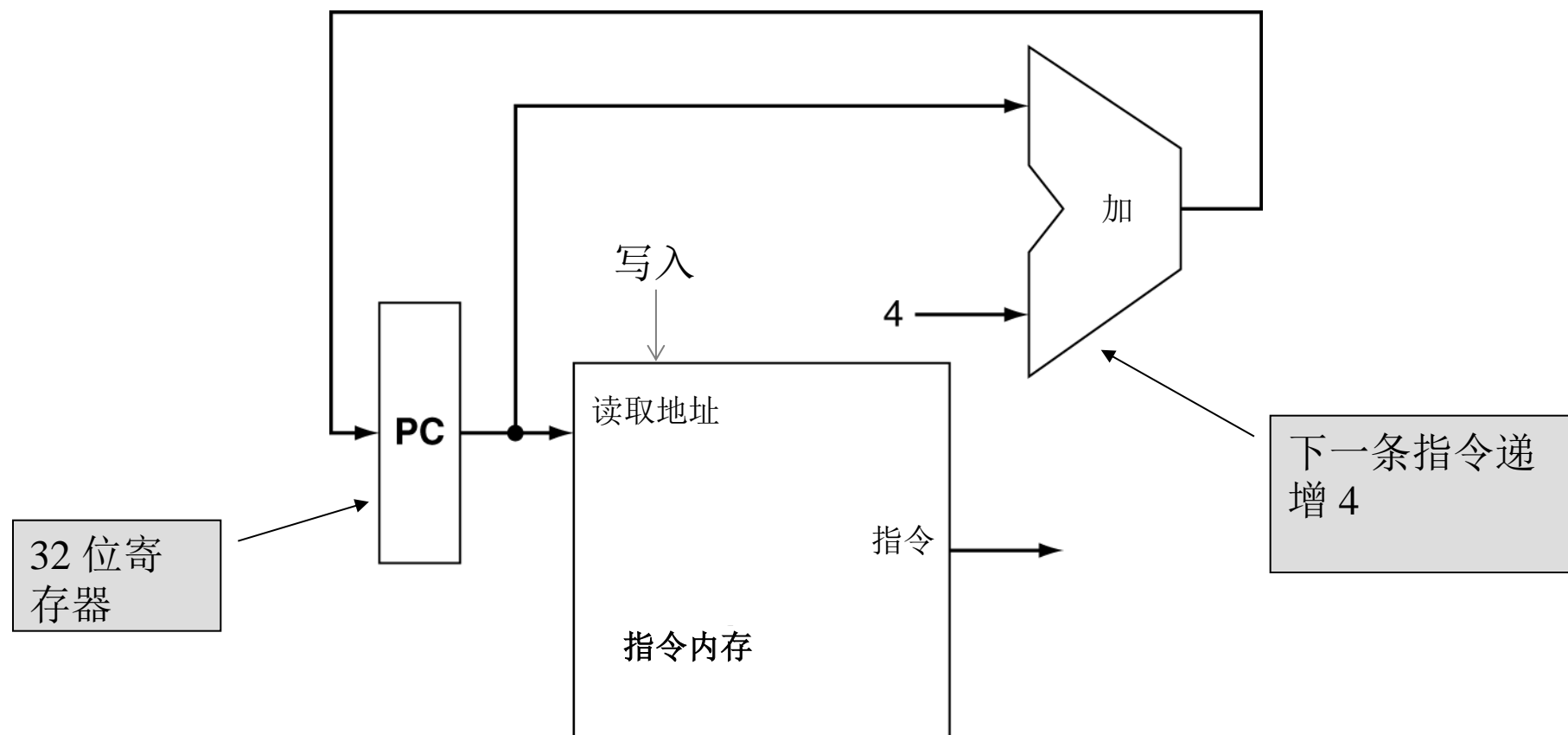
□ 数据路径定义：

- CPU 中处理数据和地址的元件
 - 寄存器
 - ALU、加法器、多路转接器、
 - 内存的地址计算逻辑（不包括内存和内存中的寻址逻辑）
 -

□ 我们将逐步构建数据路径

- 从总体设计逐步完善

指令提取：指令执行的第 1 步

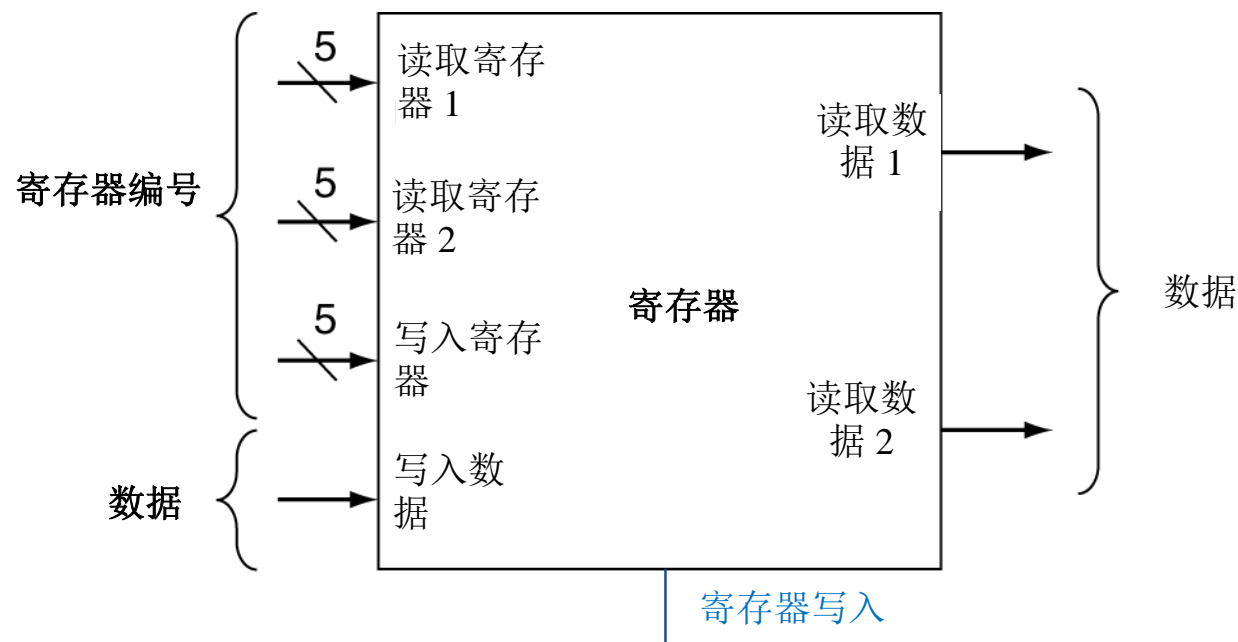


`add $10, $11, $12 # $10 ← $11 + $12`

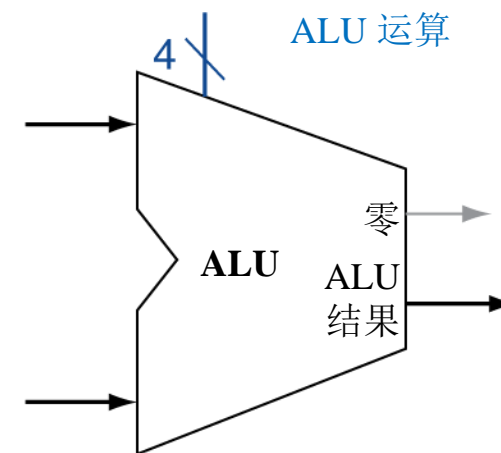
R 格式指令

- ❑ 读取两个寄存器运算数
- ❑ 执行算术/逻辑运算
- ❑ 写入寄存器结果

`add $10, $11, $12 # $10 ← $11 + $12`

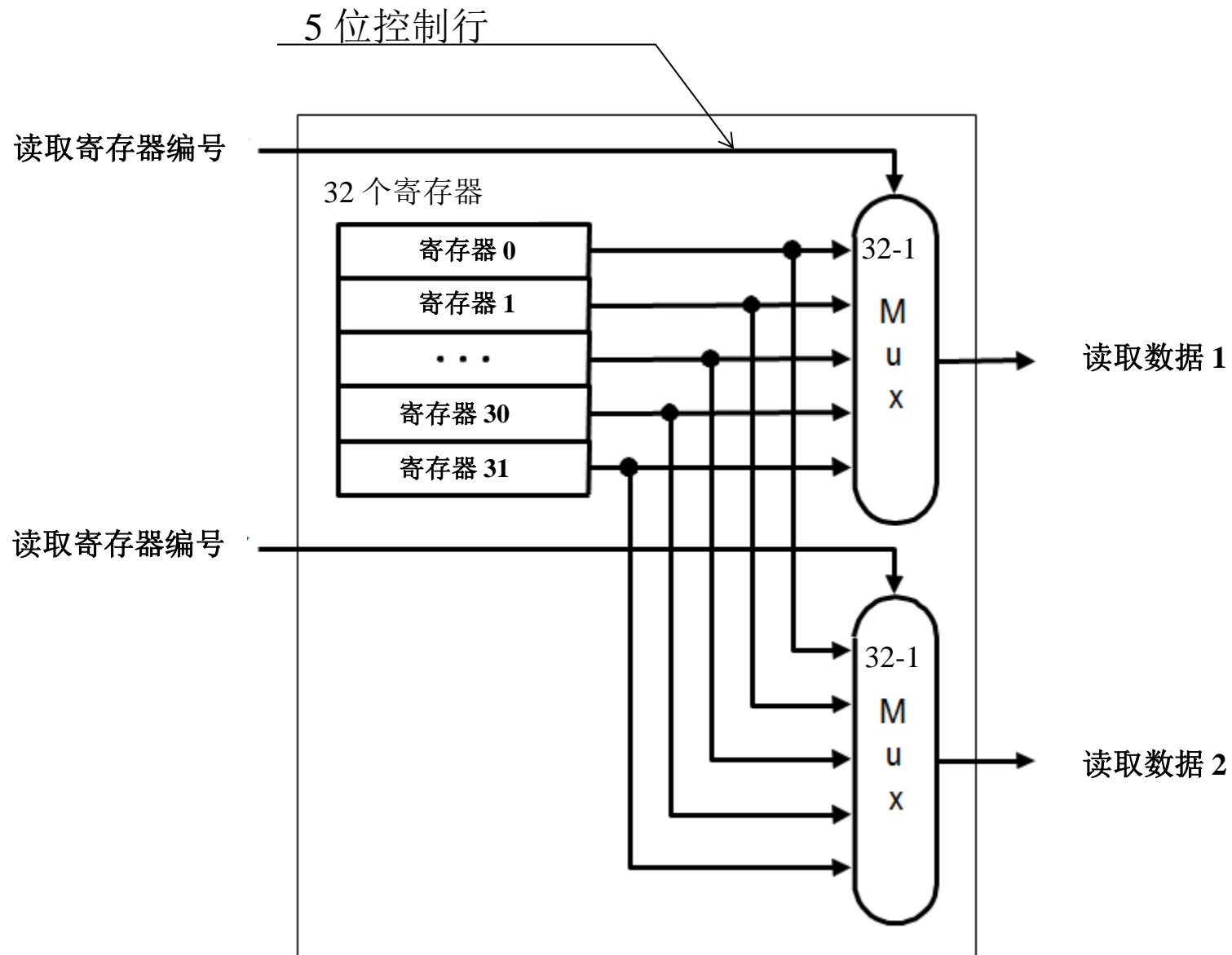


a. 寄存器

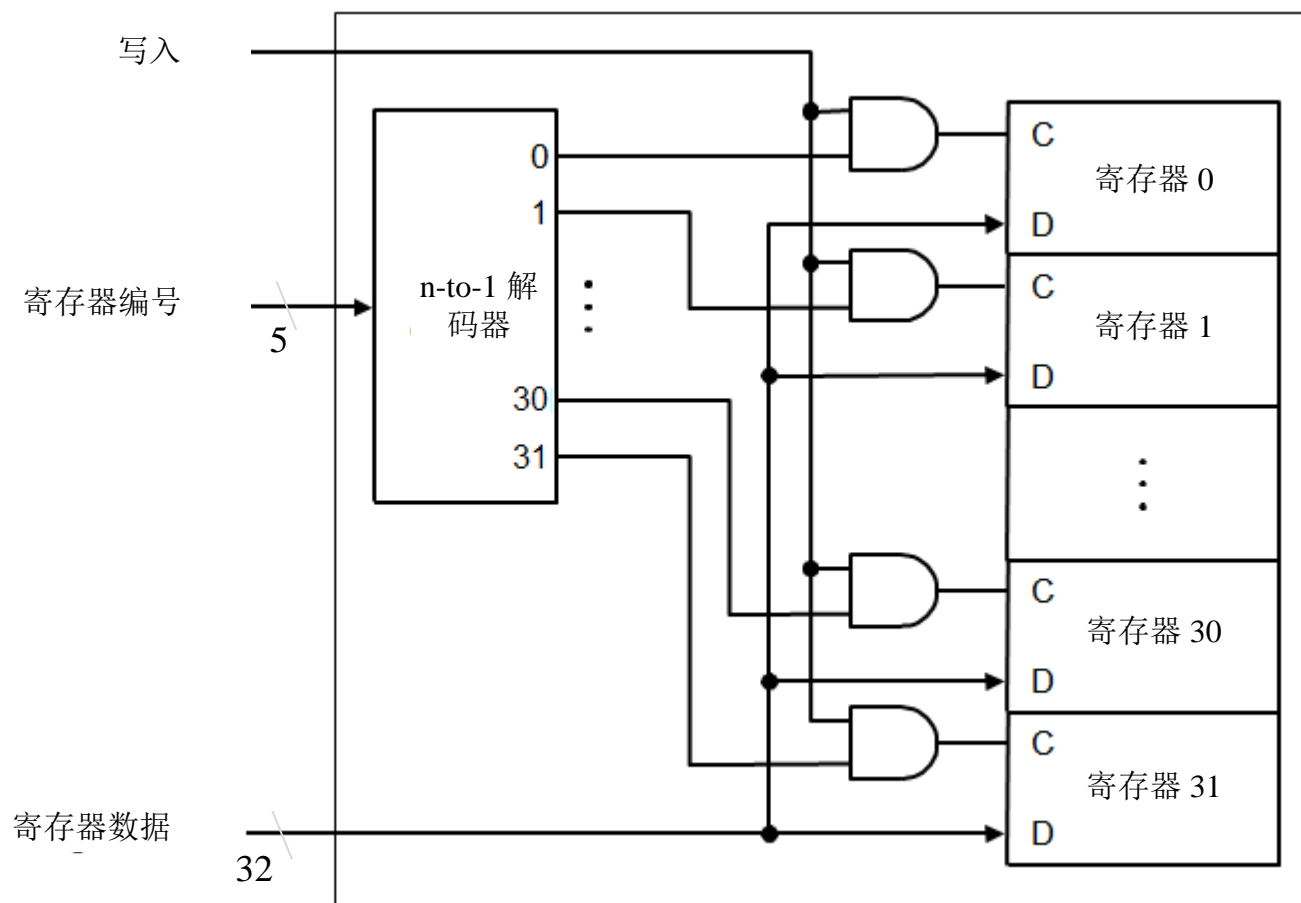


b. ALU

读取寄存器文件及其选择逻辑



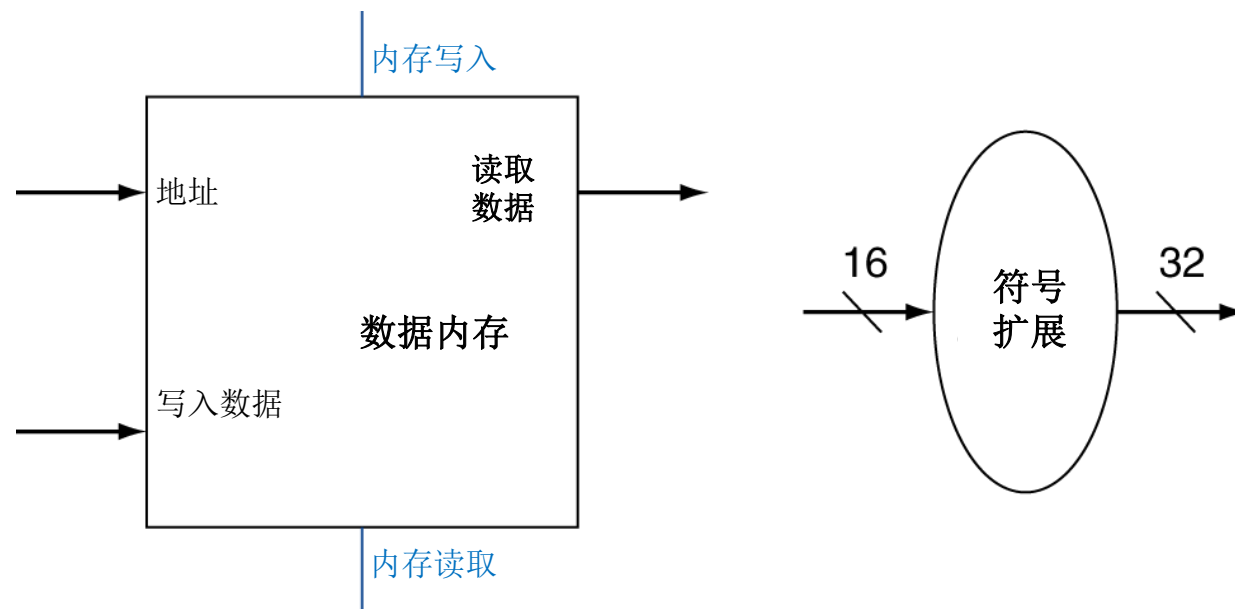
写入寄存器文件及其解码逻辑



注意：读取和写入内存位置类似于读取和写入寄存器。实际上，寄存器文件只是一个有 32 个位置的小内存。

加载/存储指令

- ❑ 读取寄存器运算数
- ❑ 使用 16 位偏移量计算地址
 - 使用 ALU，但对偏移进行符号扩展
- ❑ 加载：读取内存和更新寄存器
- ❑ 存储：将寄存器值写入内存



a. 数据内存单元

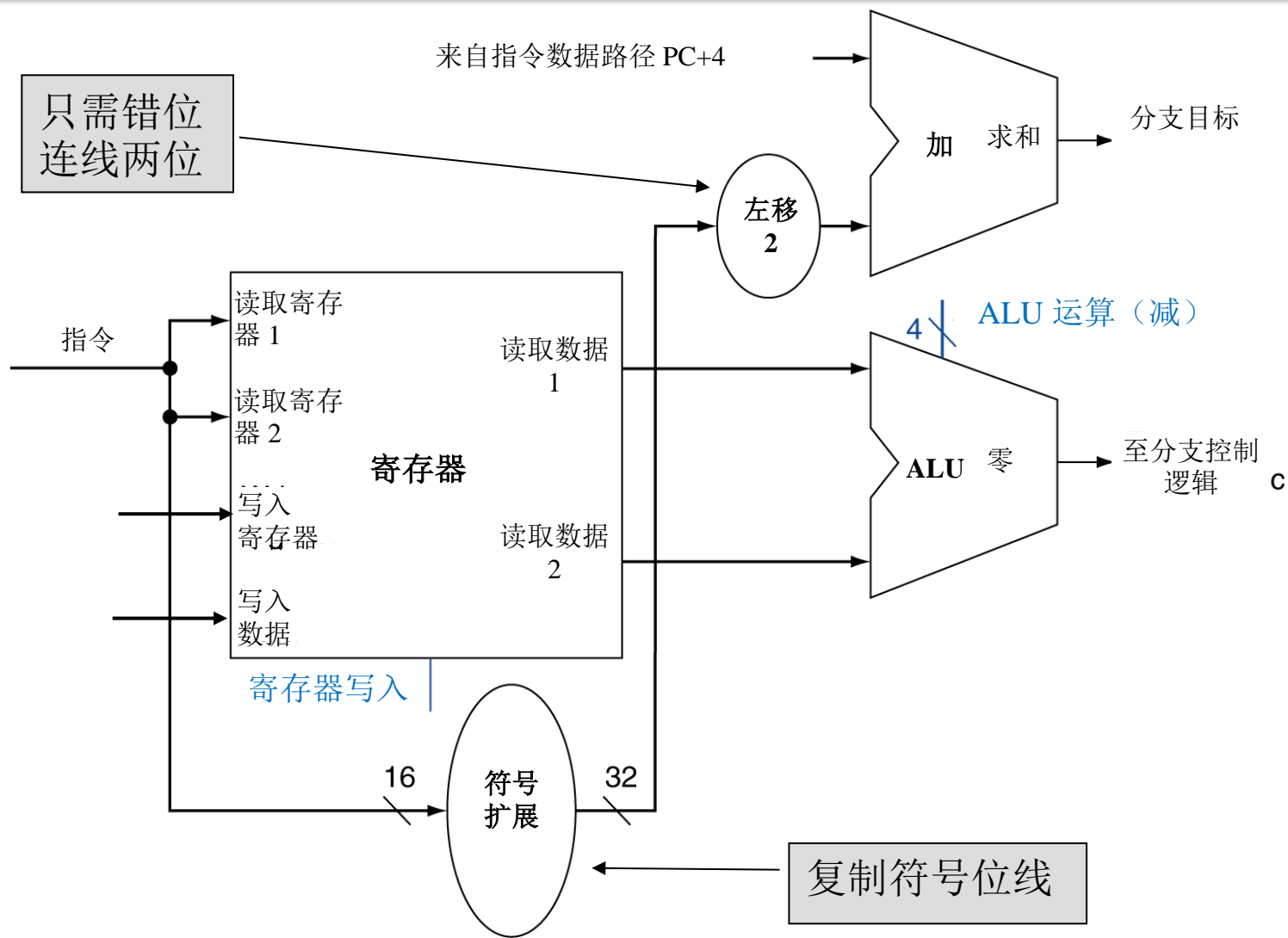
b. 符号扩展单元

分支指令

- 读取寄存器运算数
- 比较运算数
 - 使用 ALU: 减去并生成零输出
- 计算目标地址
 - 符号扩展位移
 - 左移 2 个位置（词位移）
 - 加至 $PC + 4$
 - 已经通过指令提取计算

```
beq $14, $15, L # if ($14 == $15) go to L
```

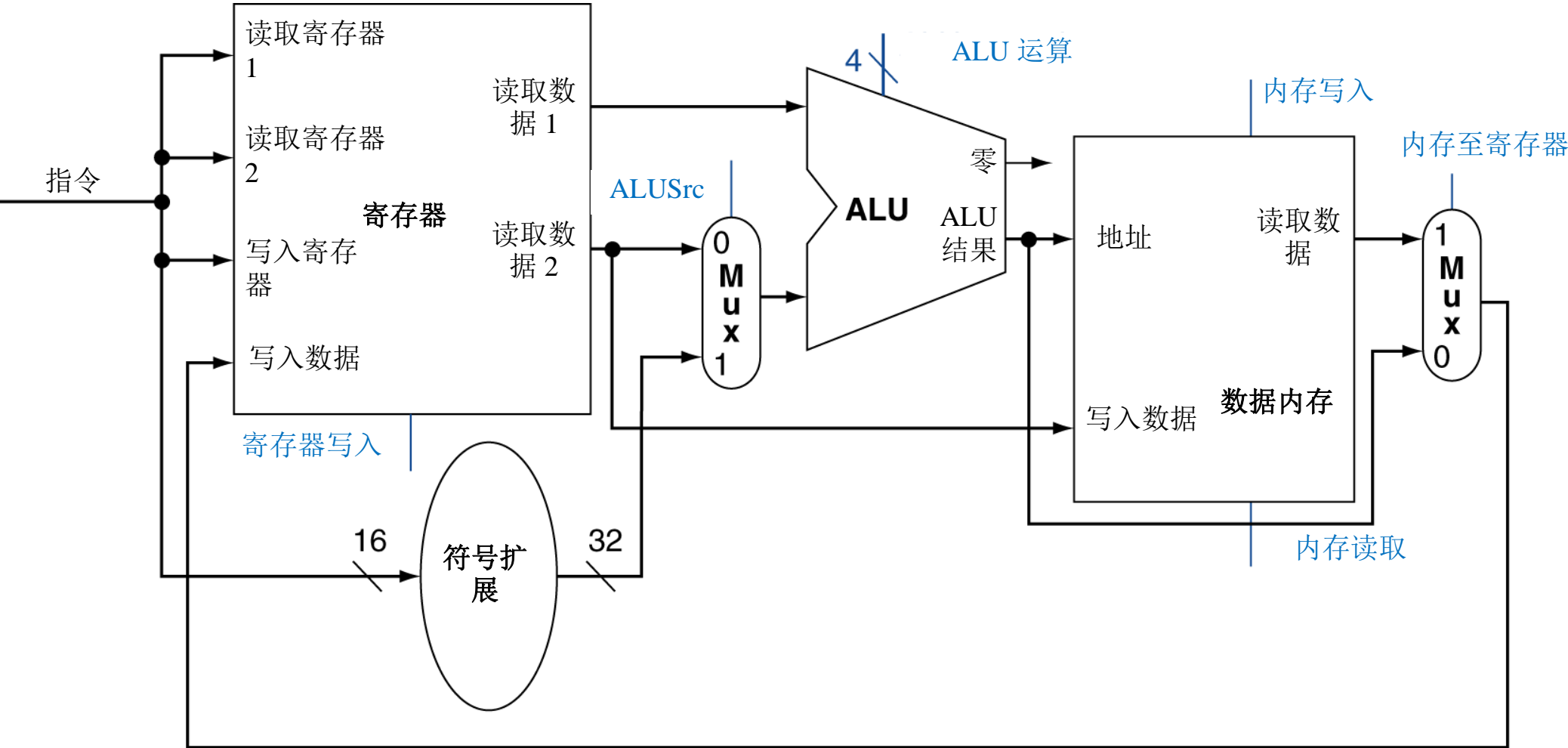
分支指令



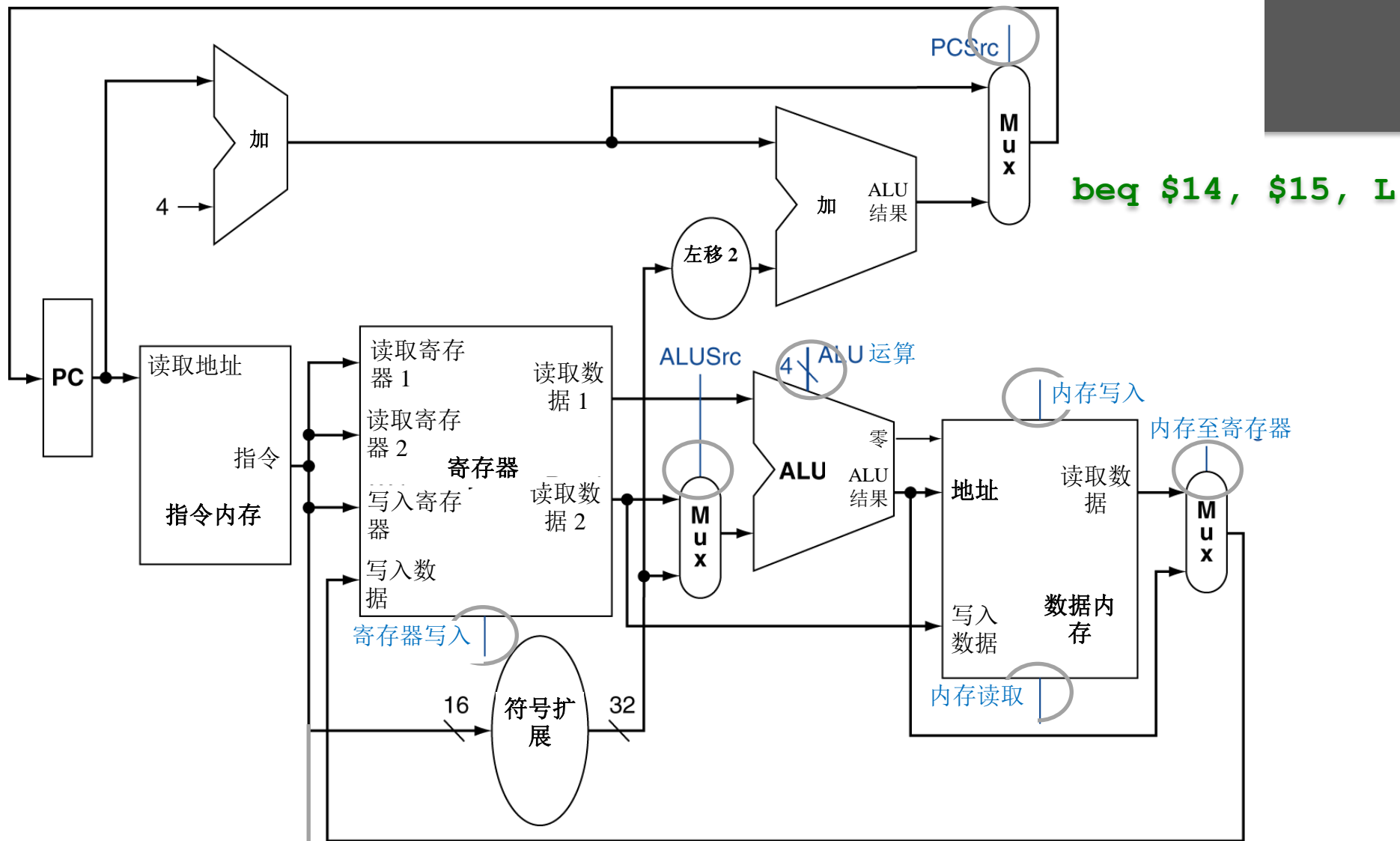
`beq $14, $15, L # if ($14 == $15) go to L`

- 我们设计数据路径以在**单个时钟周期**内执行指令
 - 每个数据路径元素一次只能执行一个函数
 - 因此，我们需要单独的指令和数据内存
 - 多个 ALU
- 使用多路转接器，可以将不同数据源导于不同的指令

算数/加载/存储类型数据路径



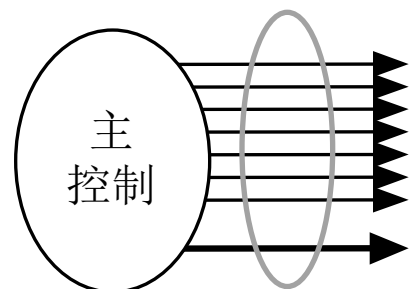
```
add $10, $11, $12    # $10 ← $11 + $12
```



beq \$14, \$15, L

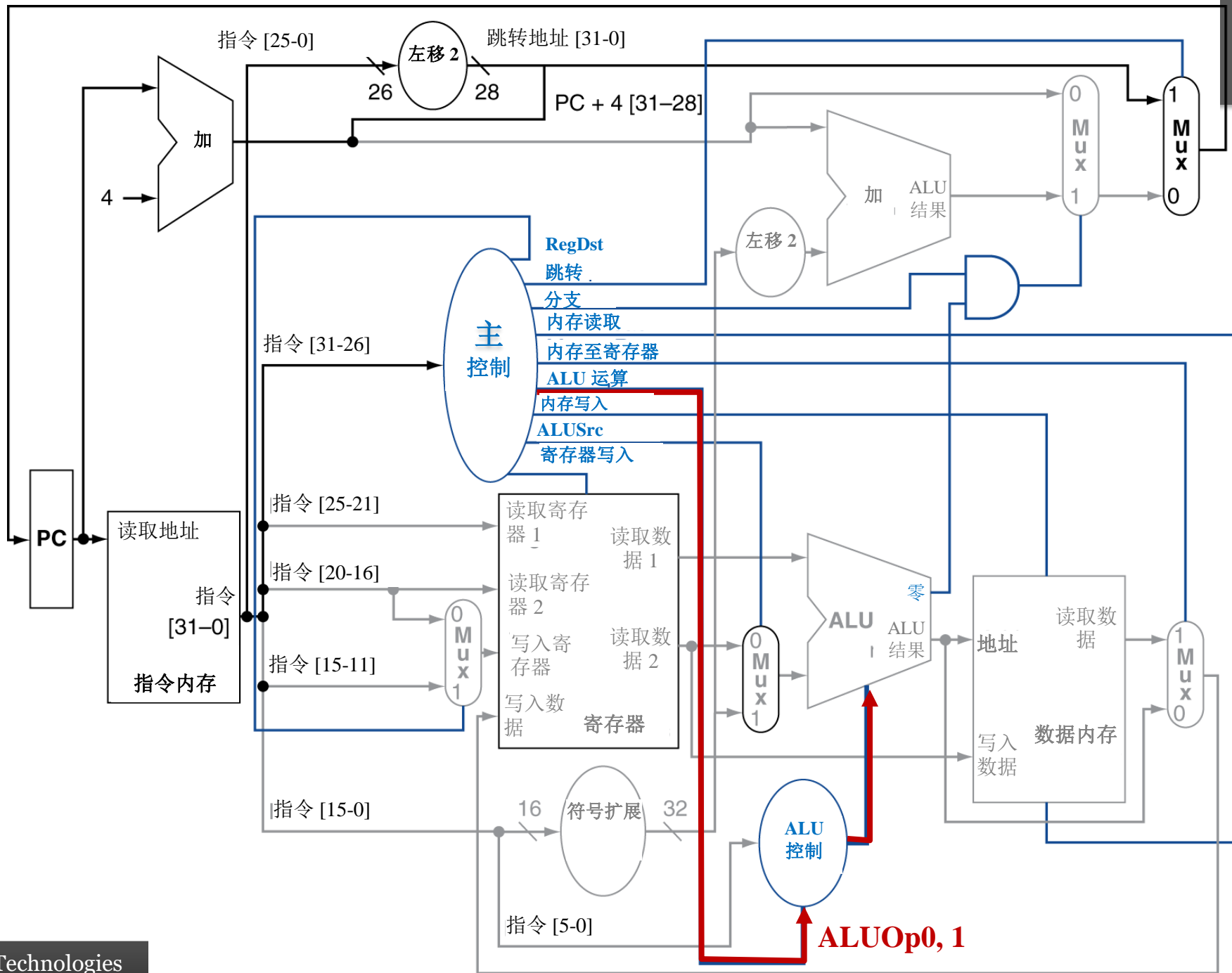
add \$10, \$11, \$12

指令
[31-26]



独立信号
— 不需要解码器

2 个组合信号: ALUOp0, 1
— 需要解码



我们如何设计主控制单元？



Additional Slides (Not To be Assessed)
额外幻灯片（不会评估）

RISC-V 计算机架构仿真器 (学习VIPLE后可以练习)

<https://venus.sod.asu.edu/VIPLE/RISCV/>

Unamed - VIPLE

File Edit Services Run Tools Language Help

Start

Start Unity Simulator

Start Unity Simulator 2

Start Unity Traffic Simulator

Start TORCS

Start Web 2D Simulator

Start Web 3D Simulator

Start RISC-V Simulator

127.0.0.1

:

8222

WebSocket

Open

Close

Animation

☐

Speed

10

Reset

Run

Step

Fetch

Decode

ALU

Compare

Mem/Reg

PC

存储器

动画展示

速度

程序计数器

寄存器

存储器地址

总线

将被执行的汇编语言程序

指令寄存器

比较器

Memory

C Code

Address

0

1

2

3

Instructions

00000000

93

00

00

02

addi x1, x0, 32

00000004

37

01

00

c0

lui x2, 0xc00000

00000008

83

c1

00

00

lbu x3, 0(x1)

0000000c

63

88

01

00

beq x3, x0, +16

00000010

23

00

31

00

sb x3, 0(x2)

00000014

93

80

10

00

addi x1, x1, 1

00000018

6f

f0

1f

ff

jal x0, -16

0000001c

6f

00

00

00

jal x0, 0

00000020

48

65

6c

6c

-

00000024

6f

00

00

00

jal x0, 0

00000028

00

00

00

00

-

0000002c

00

00

00

00

-

00000030

00

00

00

00

-

Text I/O

0

1

b0000000 (ctrl, data)

00

00

Program counter

pc

00000000

mepc

00000000

pc+4

00000004

Bus

addr

-

data

-

irq

false

Instruction reg.

instr

-

fn

-

rs1

-

rs2

-

rd

-

imm

-

ALU

op

-

a

-

b

-

r

-

Comparator

op

-

a

-

b

-

taken

-

General-purpose regs

x0

00000000

x1

00000000

x2

00000000

x3

00000000

x4

00000000

x5

00000000

x6

00000000

x7

00000000

x8

00000000

x9

00000000

x10

00000000

x11

00000000

x12

00000000

x13

00000000

x14

00000000

x15

00000000