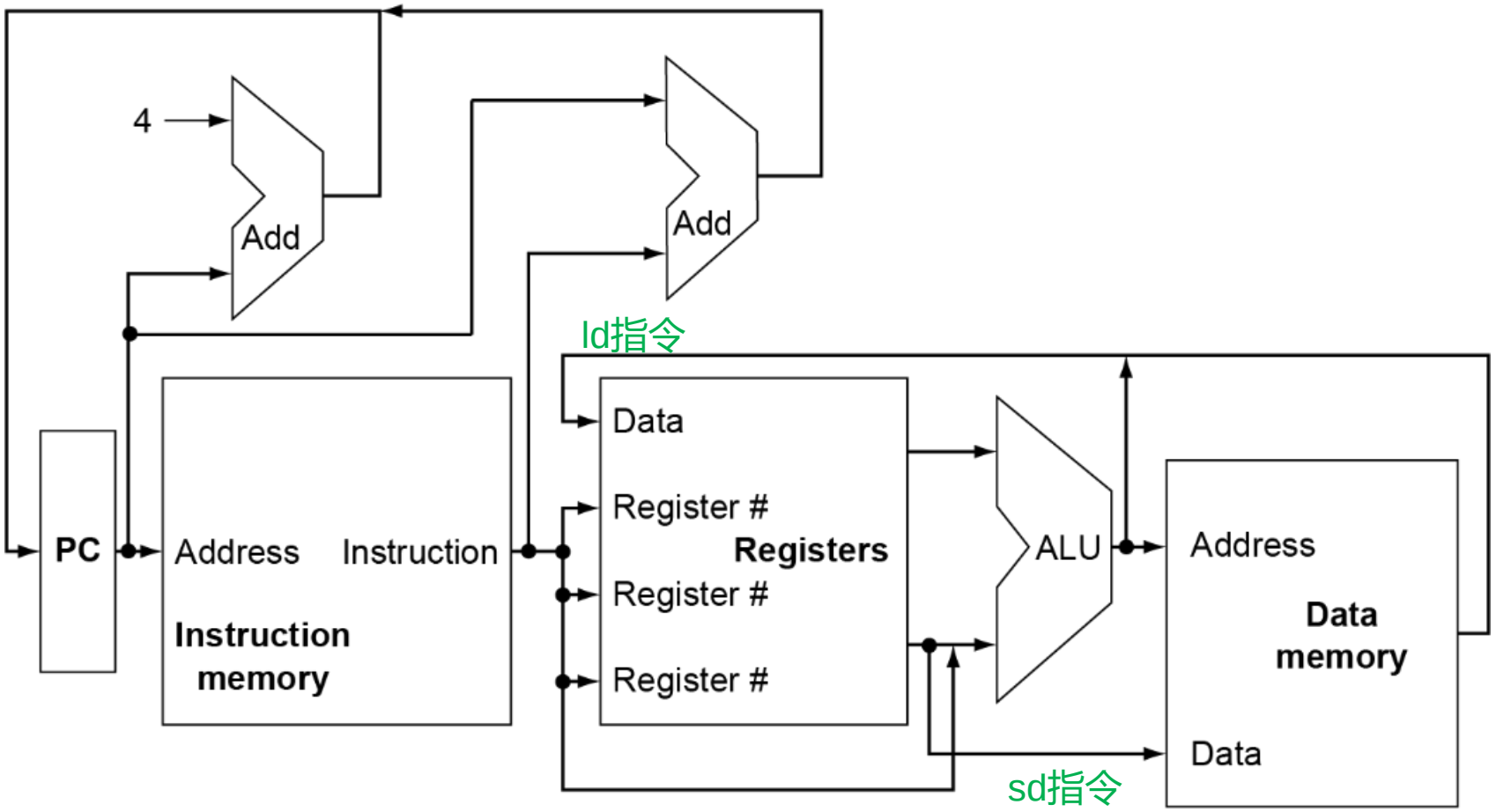
Chapter 4

处理器——指令集的实现

Outline

- **CPU预览**
- 组合逻辑
  - ALU
- 状态逻辑
  - Register寄存器
  - Memory内存

CPU Overview



逻辑设计基础

- 信息用二进制进行编码
  - 低电压表示 0, 高电压表示 1
  - 每条线传输一个bit
  - 多bit数据，用多条线构成的总线传输
- 组合逻辑
  - 输出只是输入的函数
- 状态逻辑(时序逻辑)
  - 可以存储信息

Outline

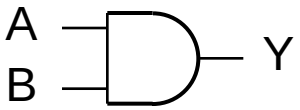
- **CPU 预览**
- **组合逻辑**
  - **ALU**
- 状态逻辑
  - Registers
  - Memory

阅读

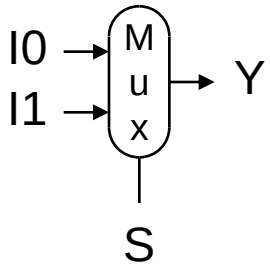
- COD 5th edition
  - 第4章
  - 附录A 逻辑设计基础
  - 附录C Mapping Control to Hardware C.2 C.3

组合逻辑

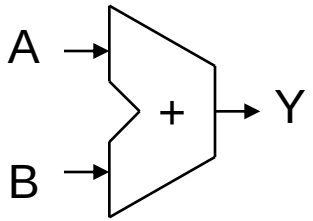
- AND-门
  - Y = A & B



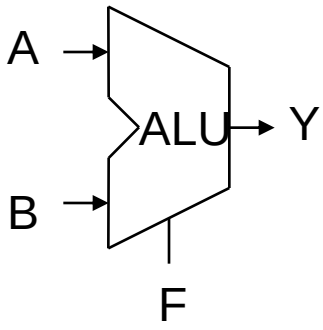
- 选择器
  - Y = S ? I1 : I0



- 加法器
  - Y = A + B



- 算术逻辑单元
  - Y = F(A, B)



基本逻辑门

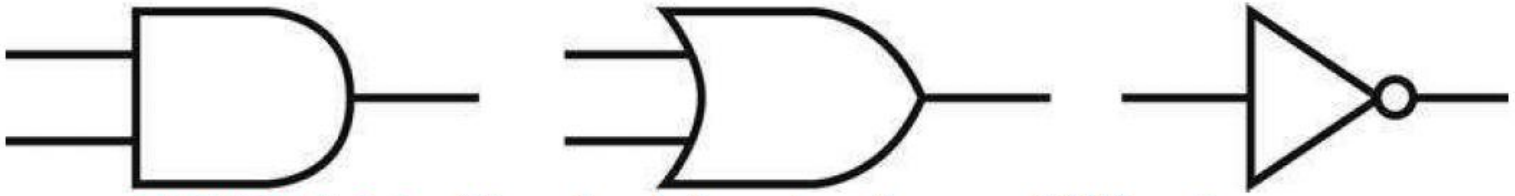


FIGURE A.2.1 Standard drawing for an AND gate, OR gate, and an inverter, shown from left to right.

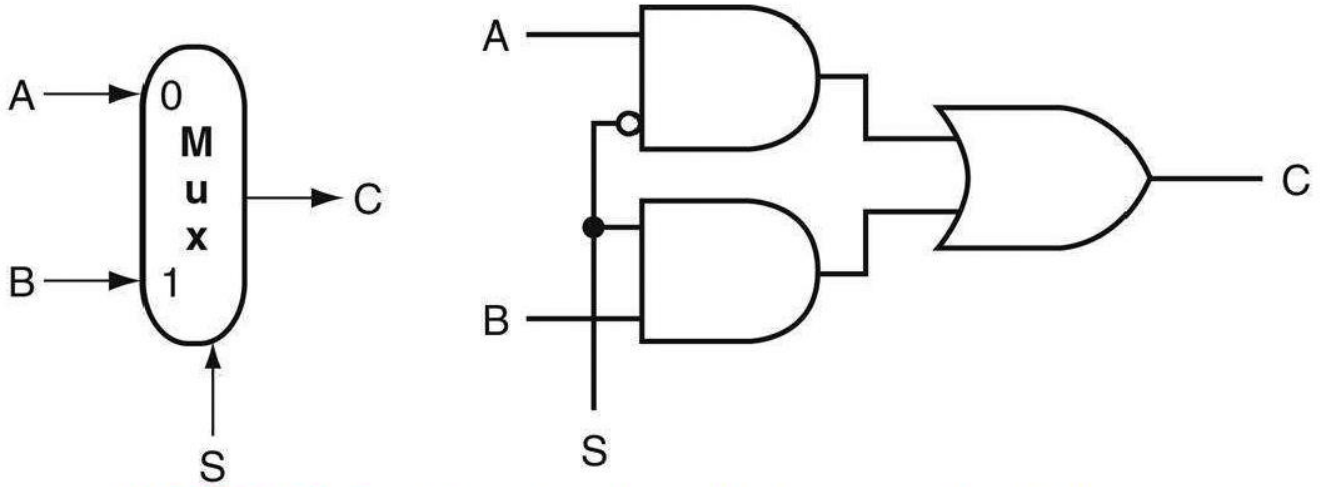
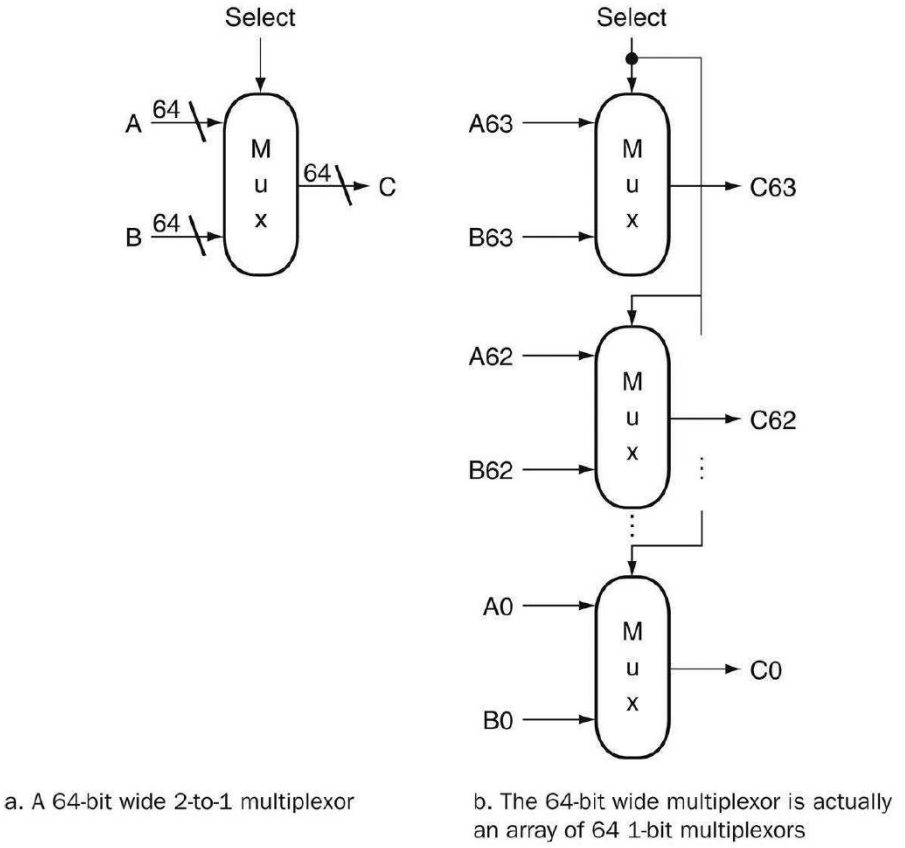


FIGURE A.3.2 A two-input multiplexor on the left and its implementation with gates on the right.

多bit 门——并行处理

- 并行的64位选择器



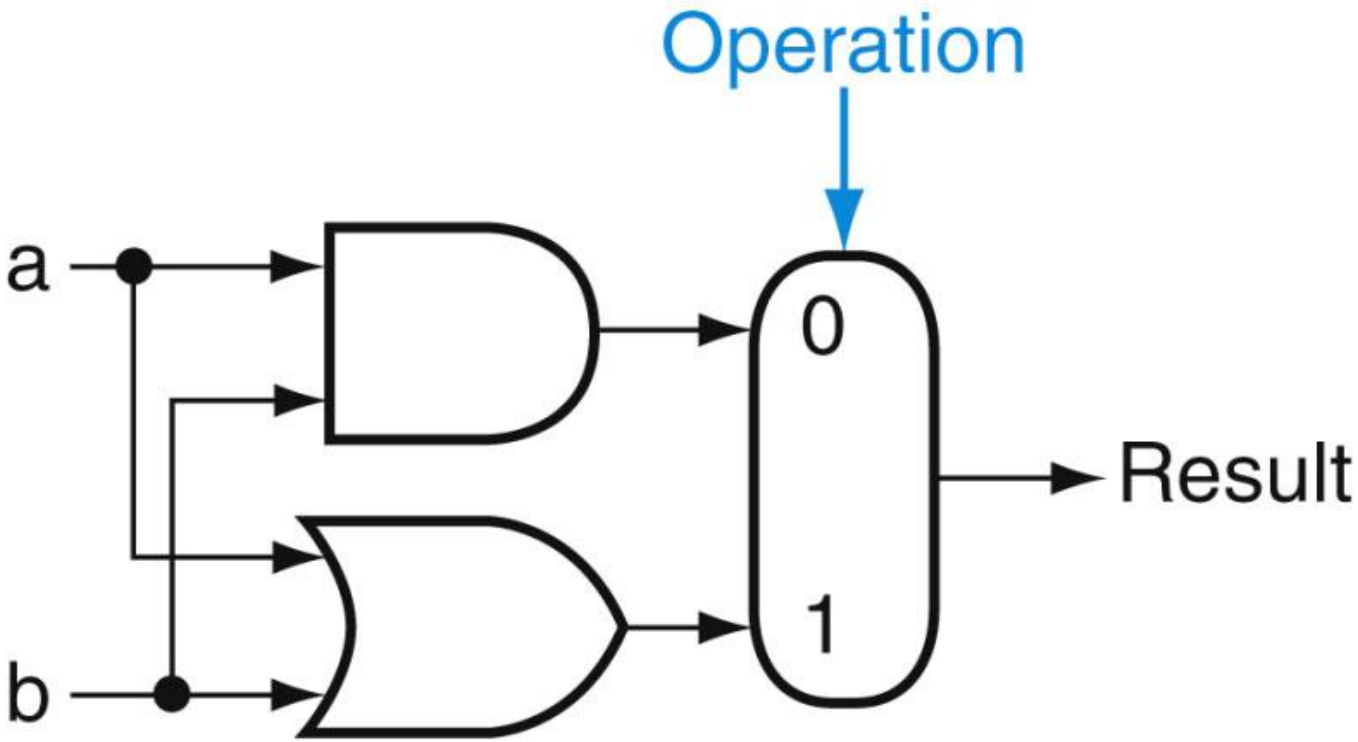
a. A 64-bit wide 2-to-1 multiplexor

b. The 64-bit wide multiplexor is actually an array of 64 1-bit multiplexors

FIGURE A.3.6 A multiplexor is arrayed 64 times to perform a selection between two 64-bit inputs.

ALU——And运算 + Or运算

- 1-bit ALU: for AND and OR.
- 利用一个选择器
  - 决定 a AND b 还是a OR b

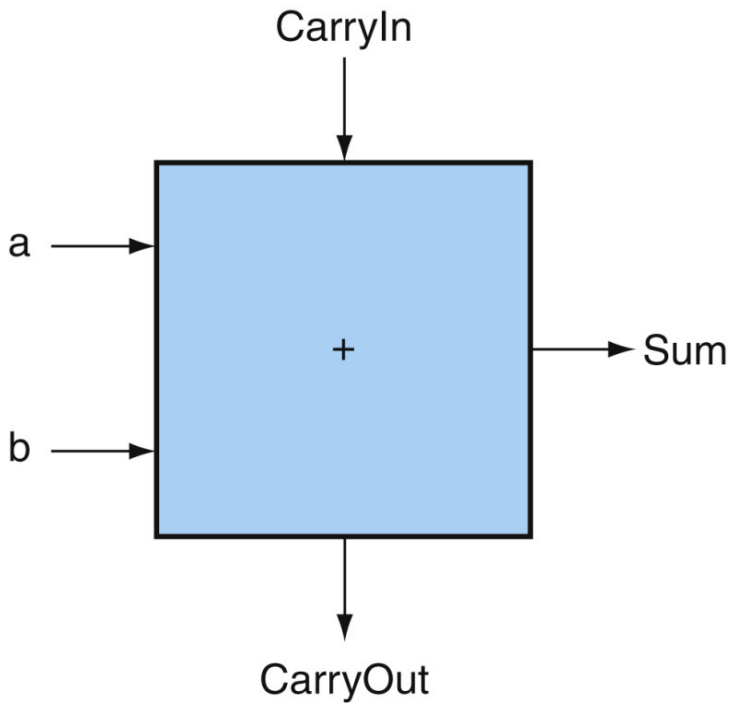


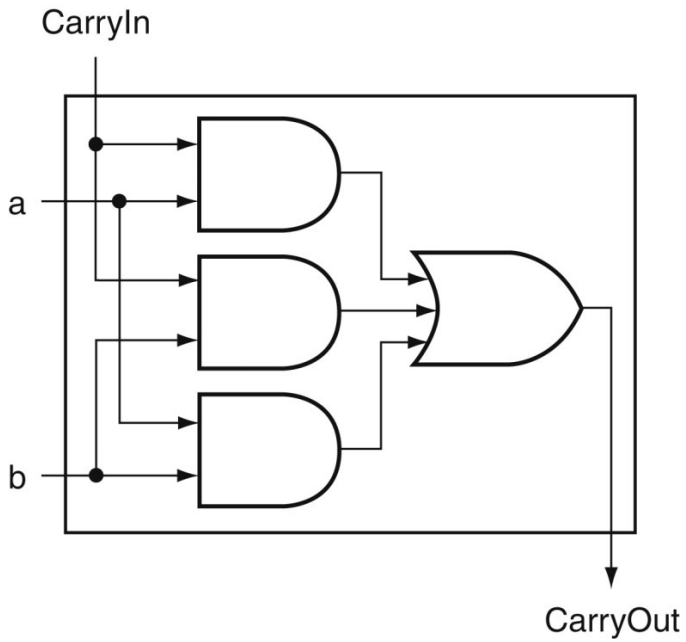
ALU——加法器

- 1-bit adder, a full adder, a (3,2) adder

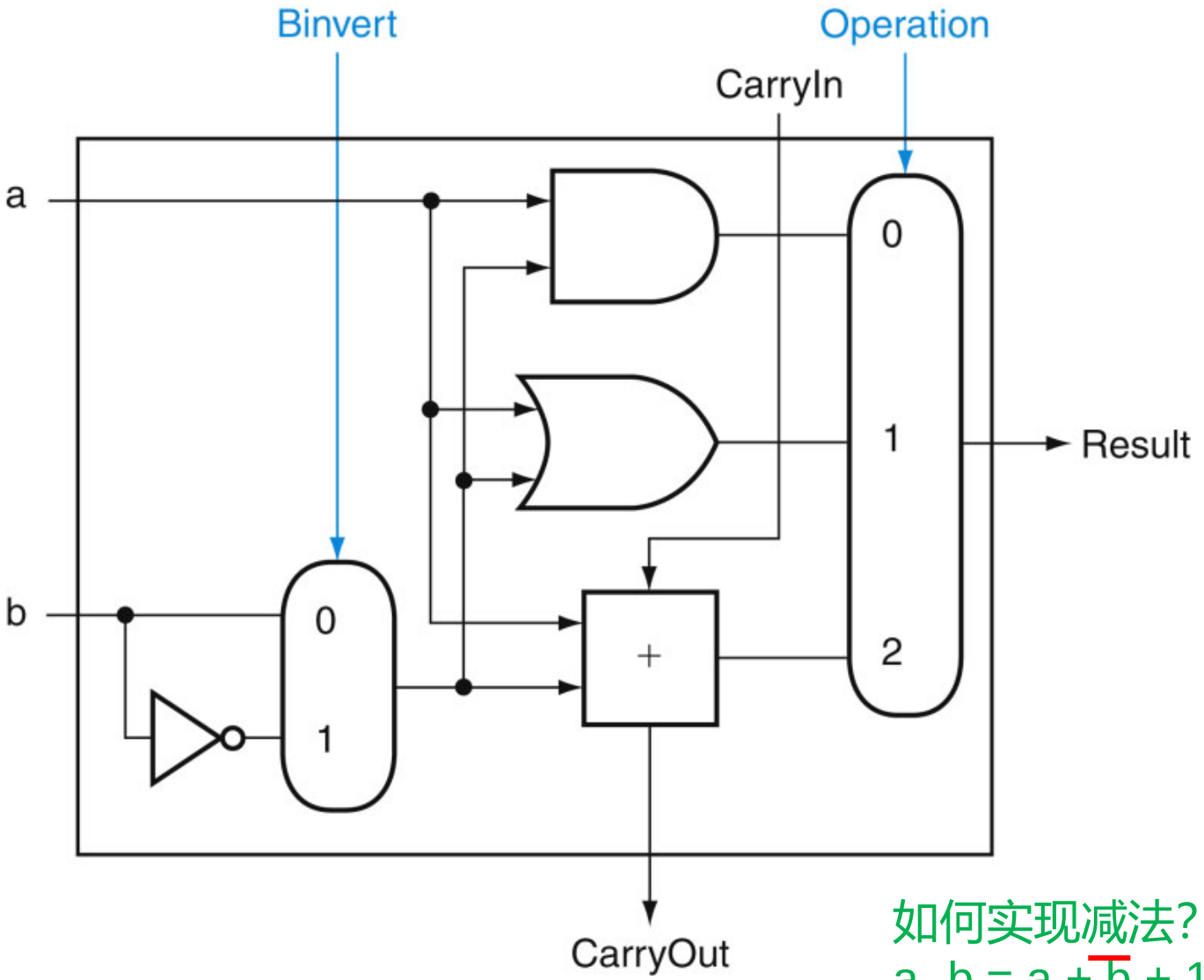
$$\text{CarryOut} = (b \cdot \text{CarryIn}) + (a \cdot \text{CarryIn}) + (a \cdot b)$$

$$\text{Sum} = (a \cdot \bar{b} \cdot \overline{\text{CarryIn}}) + (\bar{a} \cdot b \cdot \overline{\text{CarryIn}}) + (\bar{a} \cdot \bar{b} \cdot \text{CarryIn}) + (a \cdot b \cdot \text{CarryIn})$$





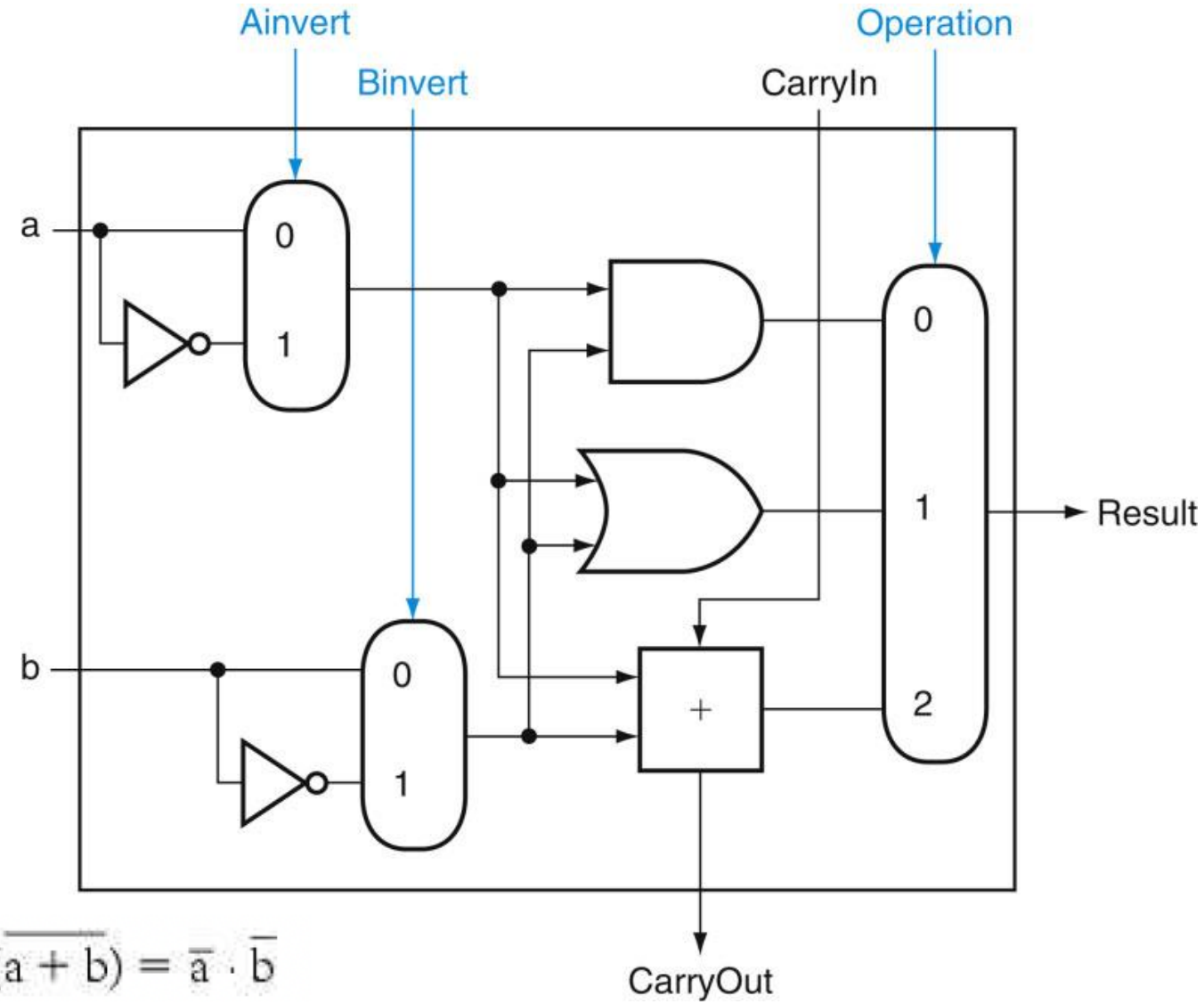
1-bit ALU with AND, OR, a+b, or a+$\overline{b}$



如何实现减法?

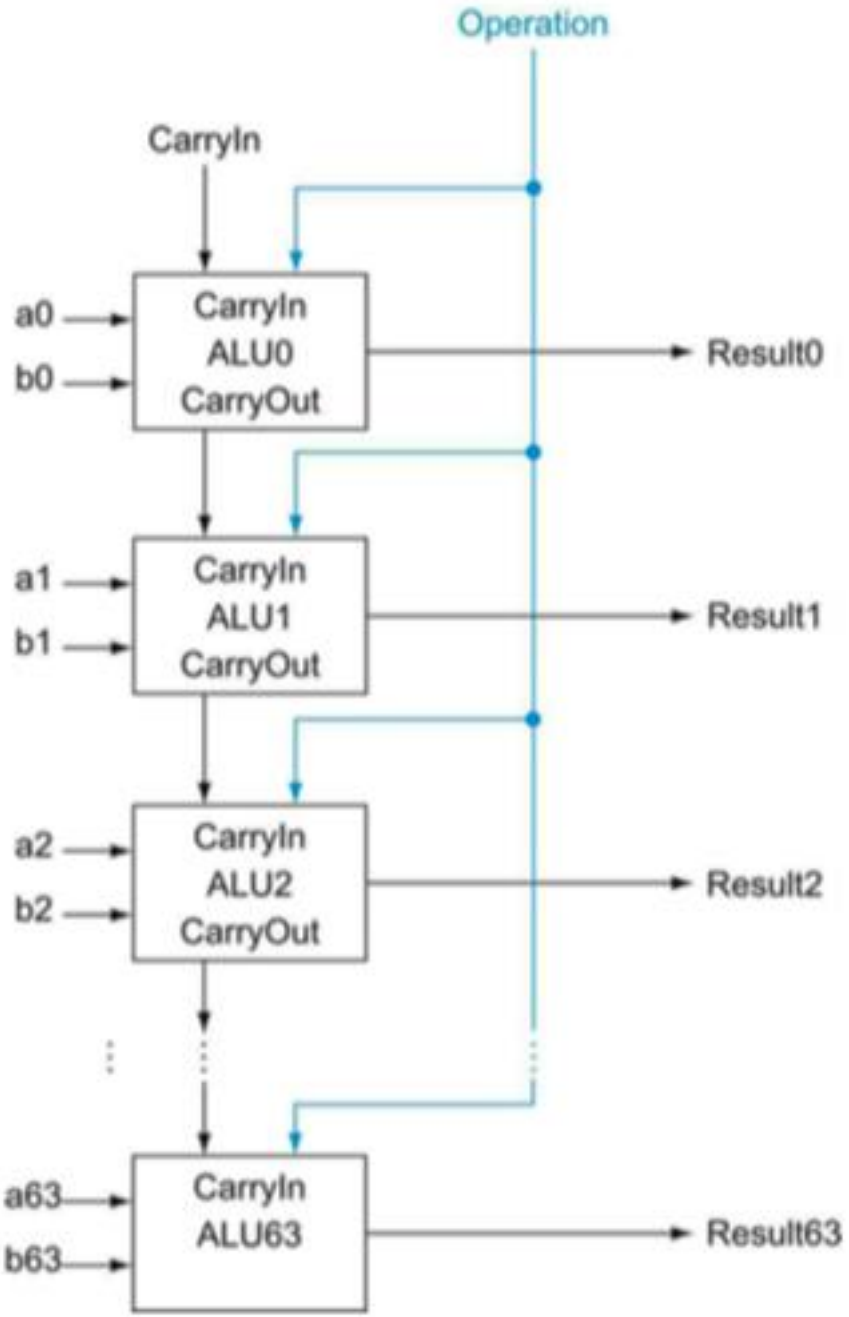
a -b = a + $\overline{b}$ + 1

1-bit ALU with NOR

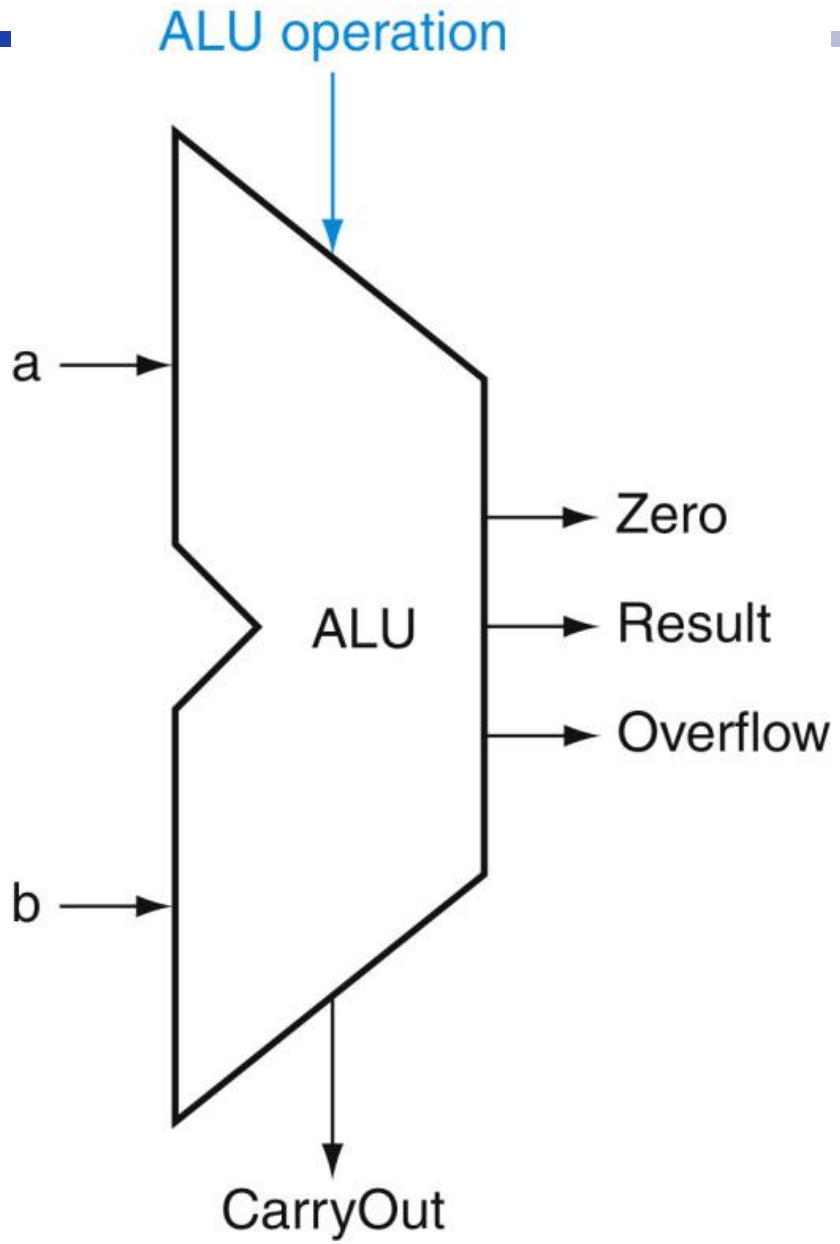


$$\overline{(a + b)} = \bar{a} \cdot \bar{b}$$

64-bit ALU


Operation
CarryIn
a0
b0
CarryIn
ALU0
CarryOut
Result0
a1
b1
CarryIn
ALU1
CarryOut
Result1
a2
b2
CarryIn
ALU2
CarryOut
Result2
a63
b63
CarryIn
ALU63
Result63

ALU的符号



ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract
0111	set less than
1100	NOR

练习

- Suppose you wanted to add the operation NOT (a AND b), called NAND. How could the ALU change to support it?

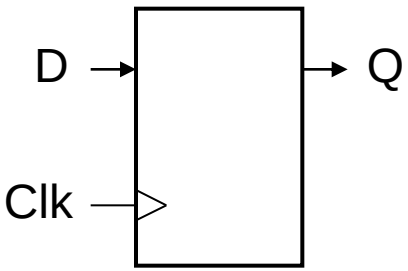
1. No change. You can calculate NAND quickly using the current ALU since $\overline{(a \cdot b)} = \overline{a} + \overline{b}$ and we already have NOT a, NOT b, and OR.
2. You must expand the big multiplexor to add another input, and then add new logic to calculate NAND.

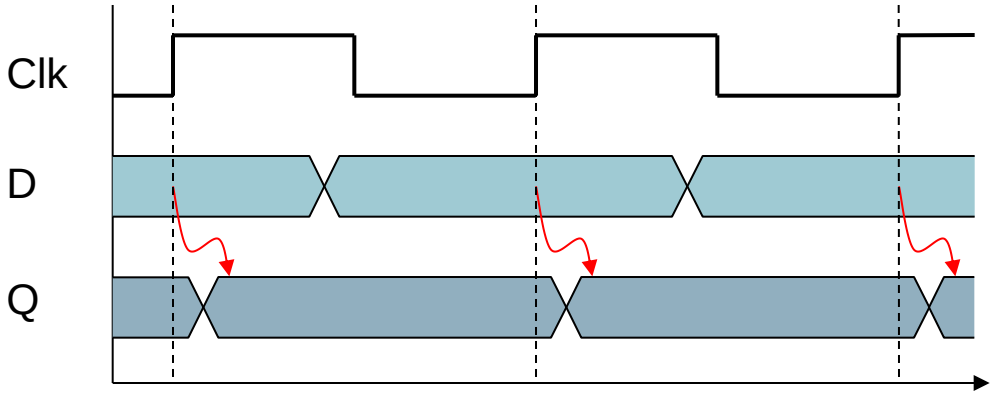
Outline

- CPU preview
- 组合逻辑
  - ALU
- 状态逻辑
  - Registers寄存器
  - Memory内存

状态逻辑

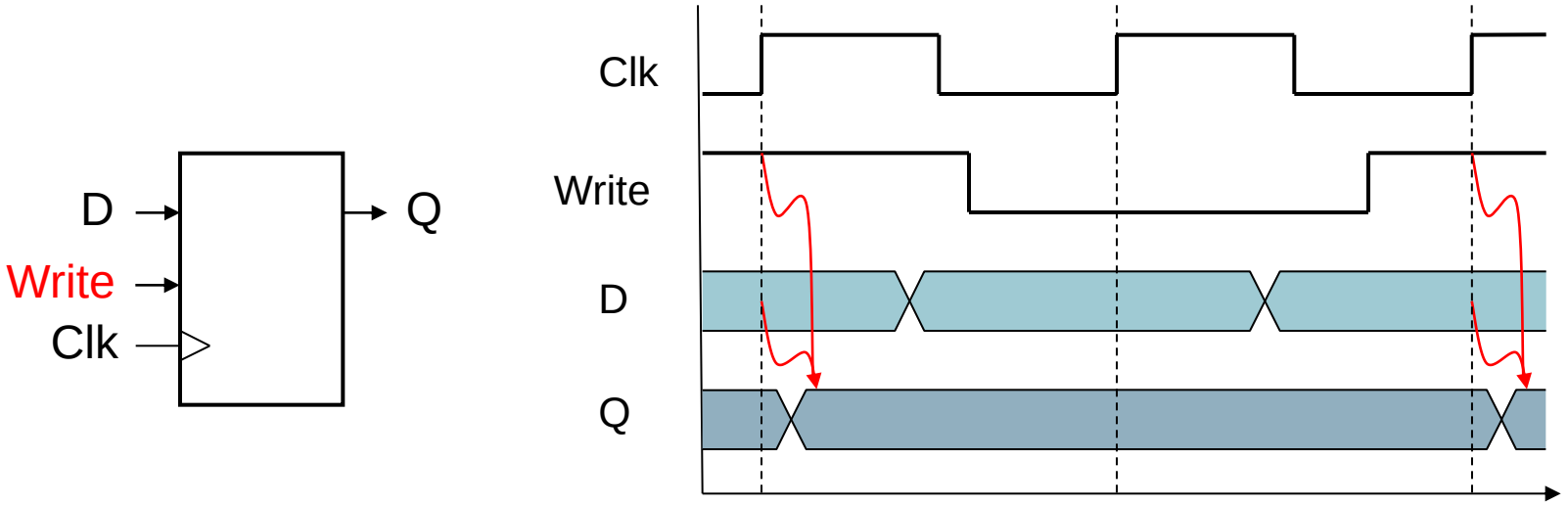
- 寄存器: 在电路中存储数据
  - 使用一个 时钟信号 确定何时更新存储的数据
  - 边缘触发: 当时钟从0 变成 1时触发





状态逻辑

- 有写控制信号的Register
  - 在时钟边缘，且只有**写控制**为1时，才更新数据
  - 用于这样的场景：后续需要使用存储的数据



状态逻辑

- D锁存器，电平触发

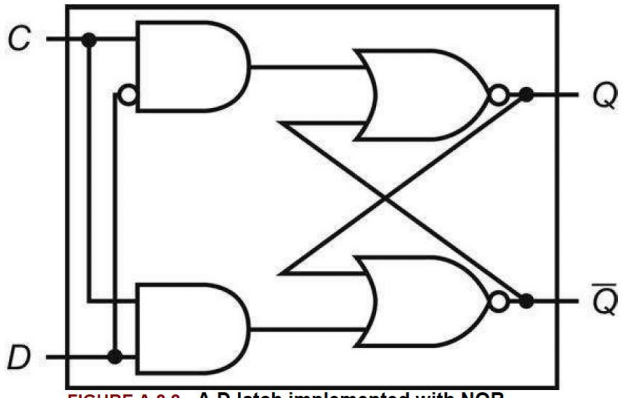


FIGURE A.8.2 A D latch implemented with NOR gates.

- D触发器，边缘触发

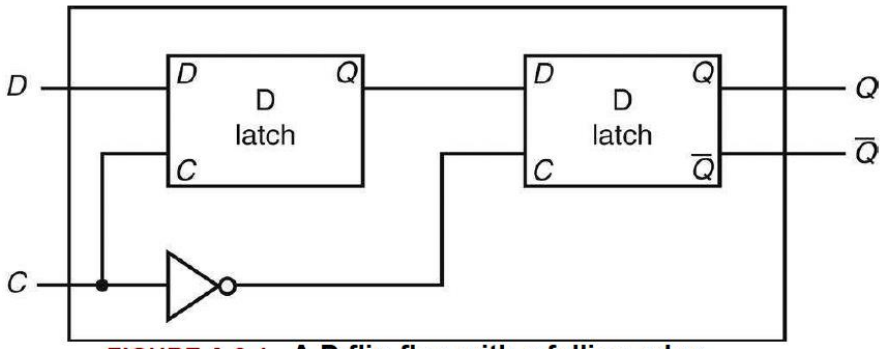


FIGURE A.8.4 A D flip-flop with a falling-edge trigger.

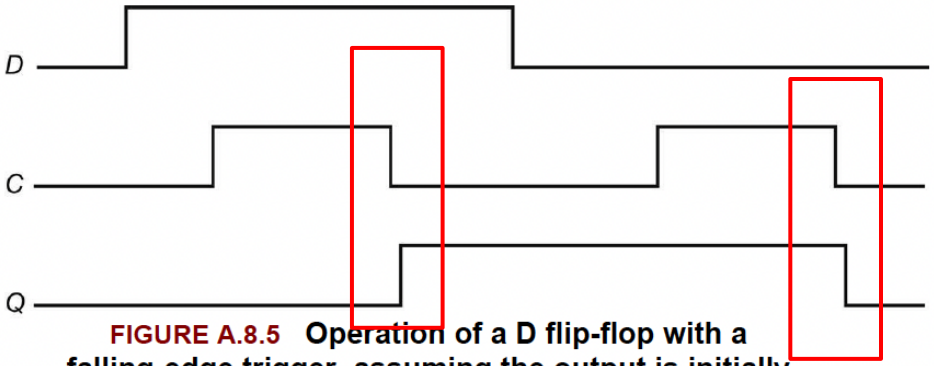


FIGURE A.8.5 Operation of a D flip-flop with a falling-edge trigger, assuming the output is initially deasserted.

下降沿边缘触发

寄存器文件: 写端口

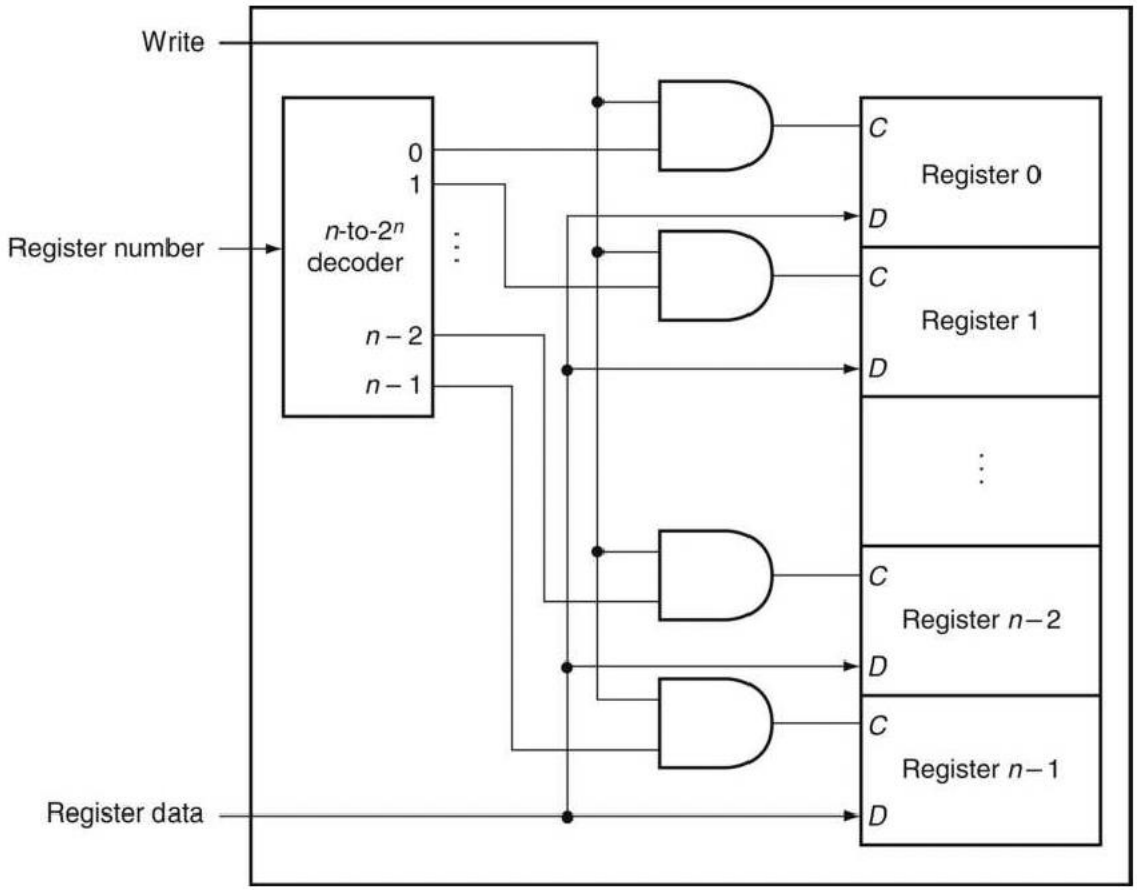


FIGURE A.8.9 The write port for a register file is implemented with a decoder that is used with the write signal to generate the C input to the registers.

寄存器文件: 读端口

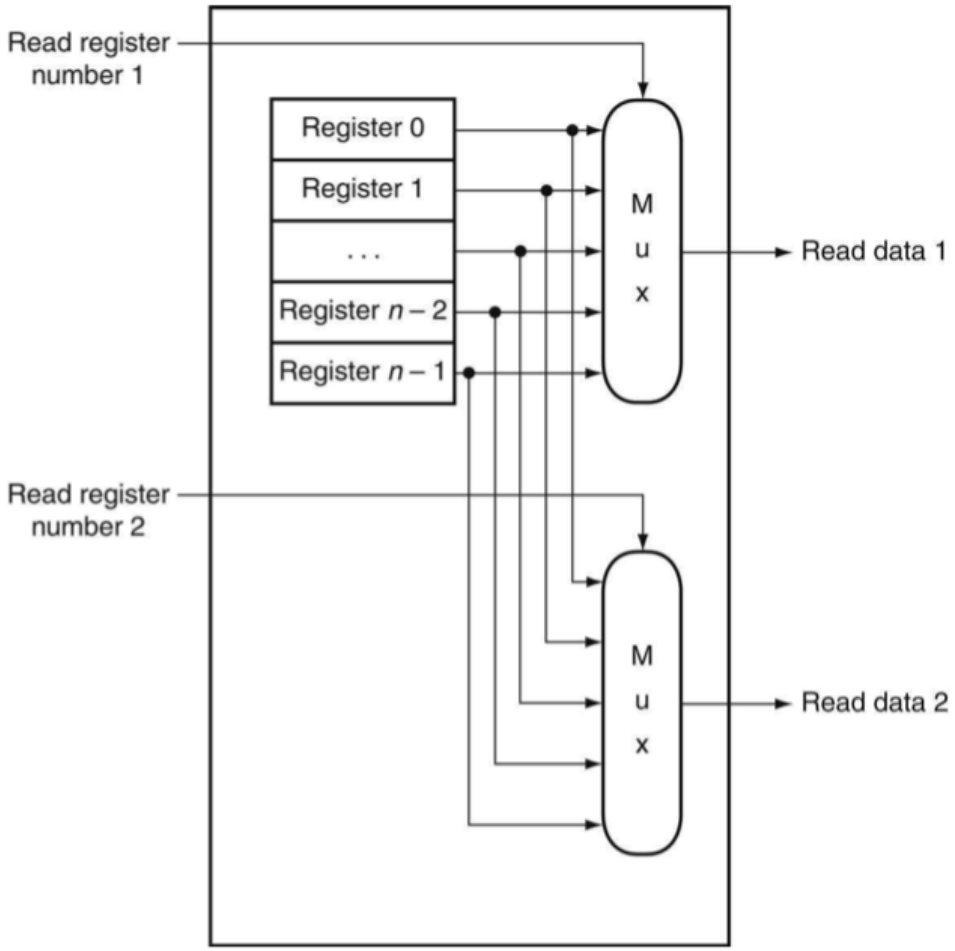
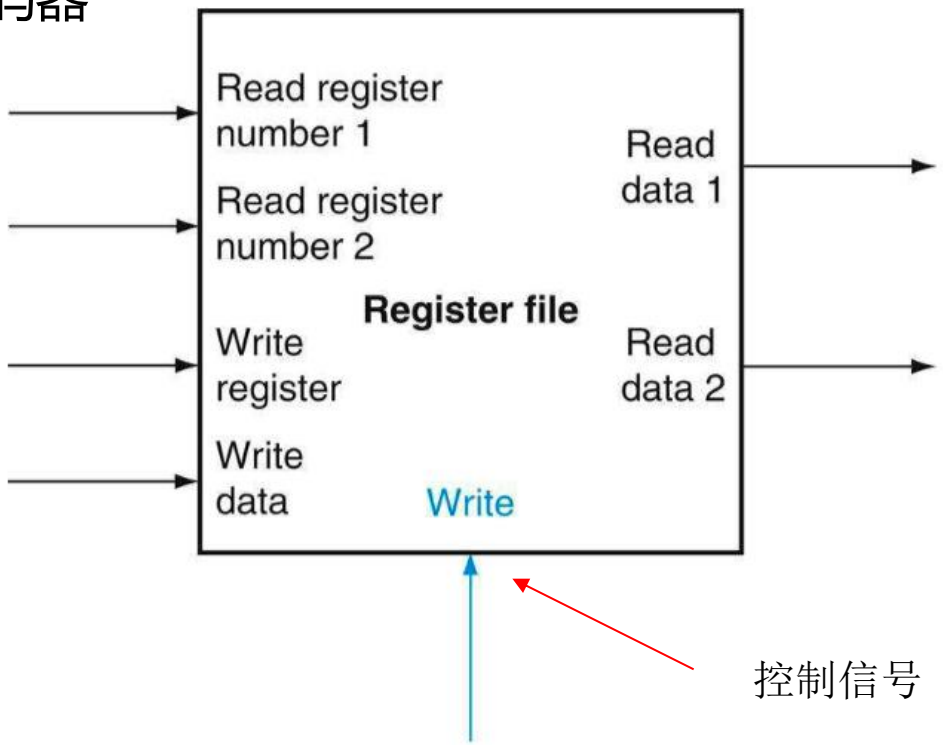


FIGURE A.8.8 The implementation of two read ports for a register file with *n* registers can be done with a pair of *n*-to-1 multiplexors, each 64 bits wide.

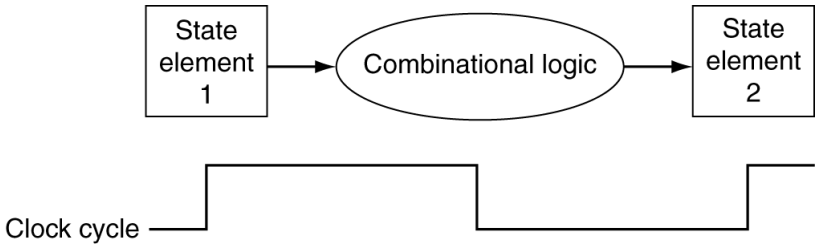
寄存器文件

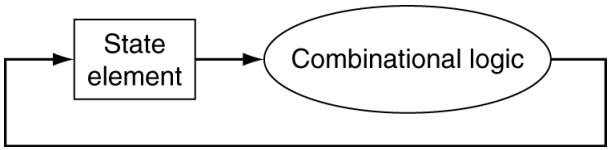
- 一个寄存器文件 (D 触发器构成) 支持读、写操作
  - 寄存器文件：由D 触发器构成的寄存器数组
  - 寄存器号用来索引具体的寄存器
  - 每个读端口需要一个选择器
  - 每个写端口需要一个译码器



时钟方法

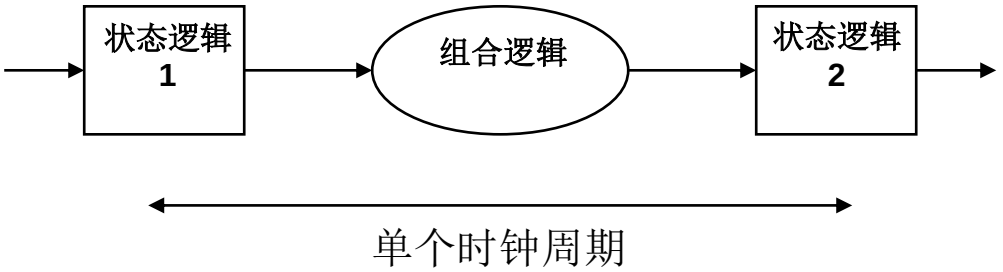
- 组合逻辑需要完成运算和数据传输
  - 在每个时钟内（两个时钟边缘之间）
  - 从一个状态逻辑获取输入，并将输出写入下一个状态逻辑
  - Longest delay决定了时钟周期
    - 需要确保时钟周期的长度，足够每个组合逻辑完成：信息读写和计算过程，且读取到的信息是上周期更新后的、稳定的状态；而不是正处于变化过程中的不稳定状态





实现

- 一种上升沿边缘触发的方法
  - **在时钟上升沿**，将计算结果写入状态逻辑1和2
  - **持续**读取状态逻辑1的内容(读过程无需时钟信号的控制)
  - **持续**计算读取的内容（组合逻辑完成计算）
    - t时钟开始上升沿，前面组合逻辑的输出写入状态逻辑 1 ，作为当前逻辑的**输入**
    - 时钟周期前半段，**持续**读取的输入和**持续**计算的值可能不稳定
    - t+1时钟开始上升沿，当前组合逻辑的**输出**值开始写入状态逻辑 2
    - 所以，必须保证在时钟周期内（t+1时钟开始之前），能够完成：状态逻辑1**上升沿**写入+状态逻辑1**持续**读取+计算



实现

- 边缘触发方法
  - 允许一个状态逻辑在同一个时钟周期内完成写和读，而不会导致不稳定的状态

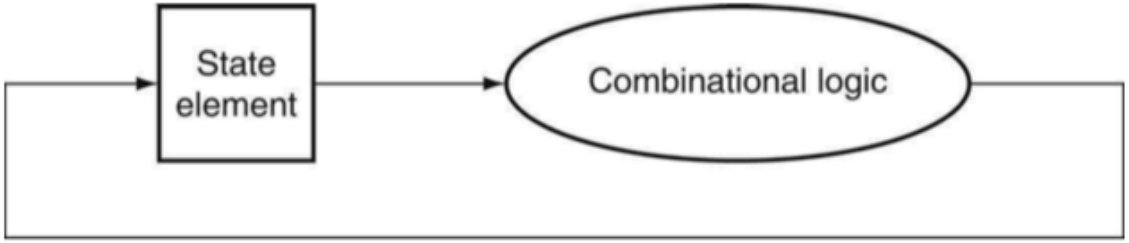


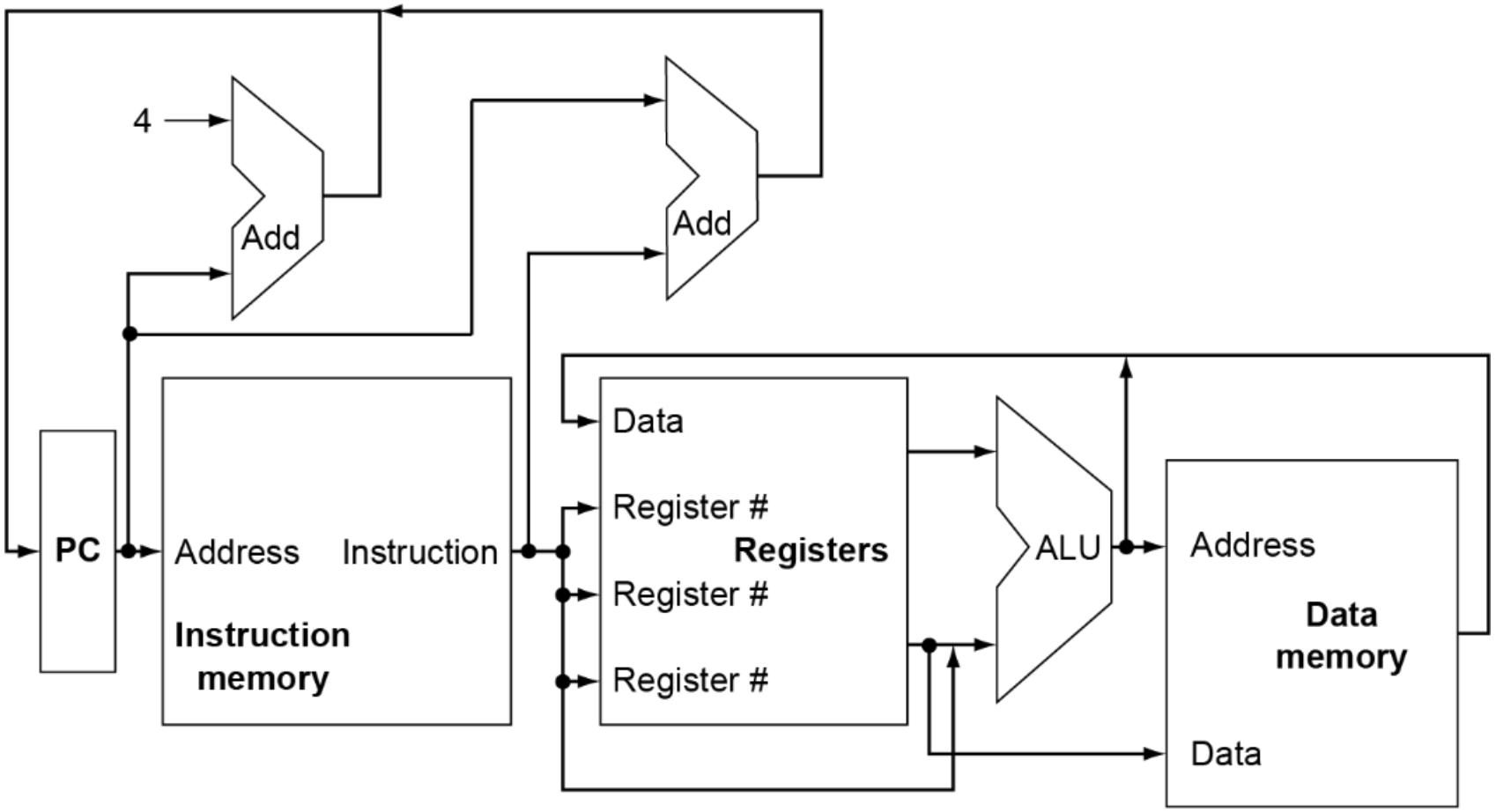
FIGURE A.7.3 An edge-triggered methodology allows a state element to be read and written in the same clock cycle without creating a race that could lead to undetermined data values.

- 默认每个时钟都会更新状态逻辑; 否则，需要额外的写控制信号
  - 这样，只有写控制为1，且时钟边缘达到时，才更新

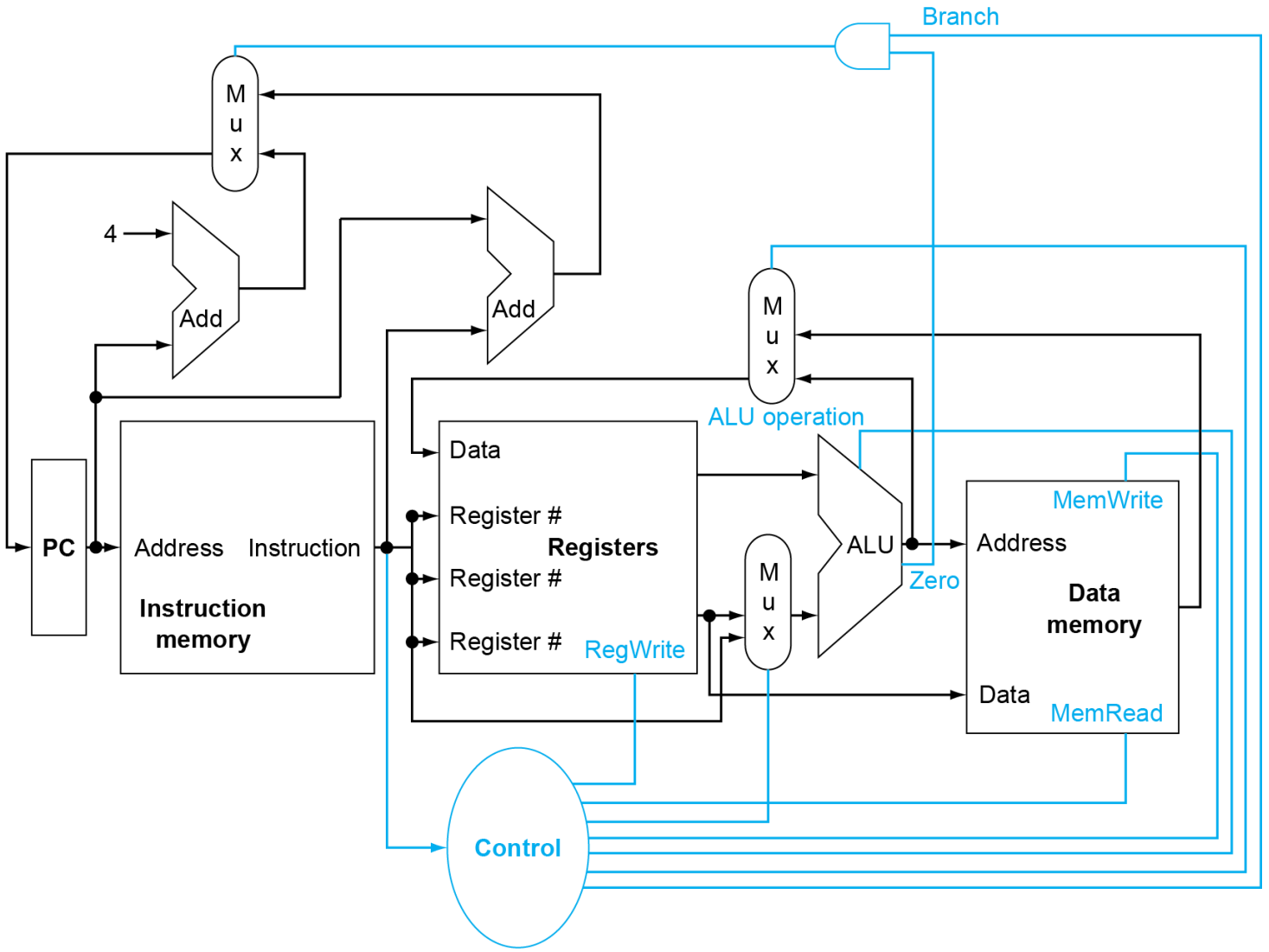
Outline

- CPU preview
- 组合逻辑
  - ALU
- 状态逻辑
  - Registers

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

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