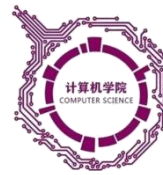


计算机系统核心课程群

《计算机硬件基础》

第八章 计算机组成原理初探



本章知识要点

01

数据通路

02

操作元件

03

状态元件

04

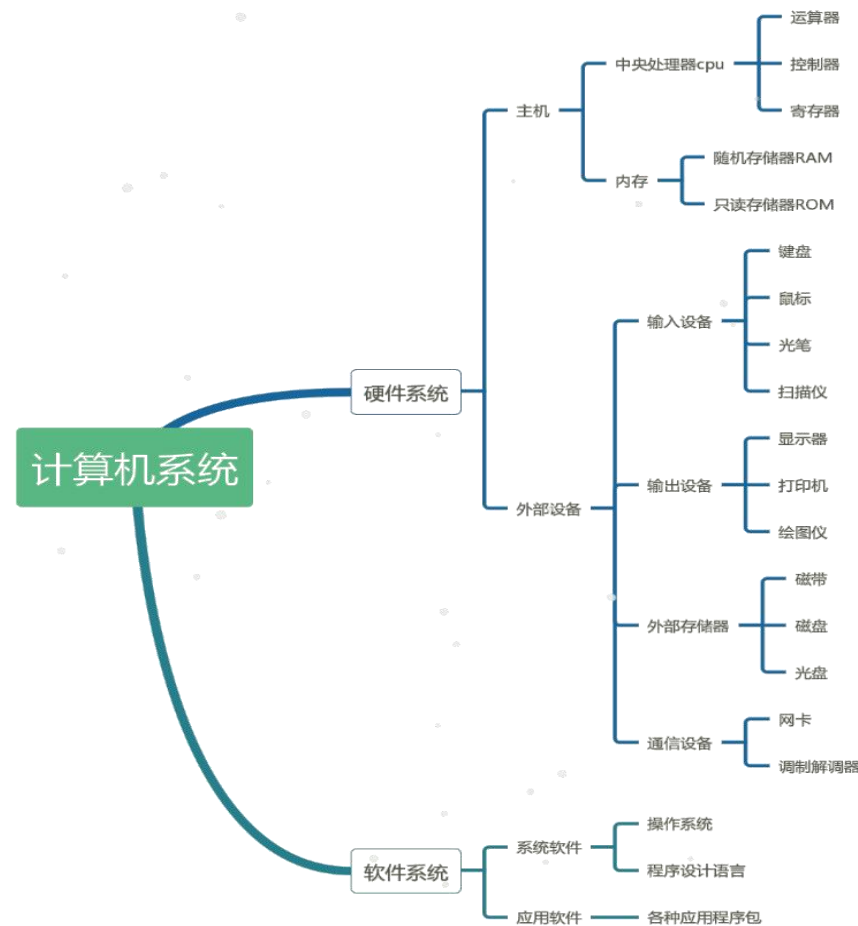
存储元件

8.1 计算机组成基础

一、系统组成框架

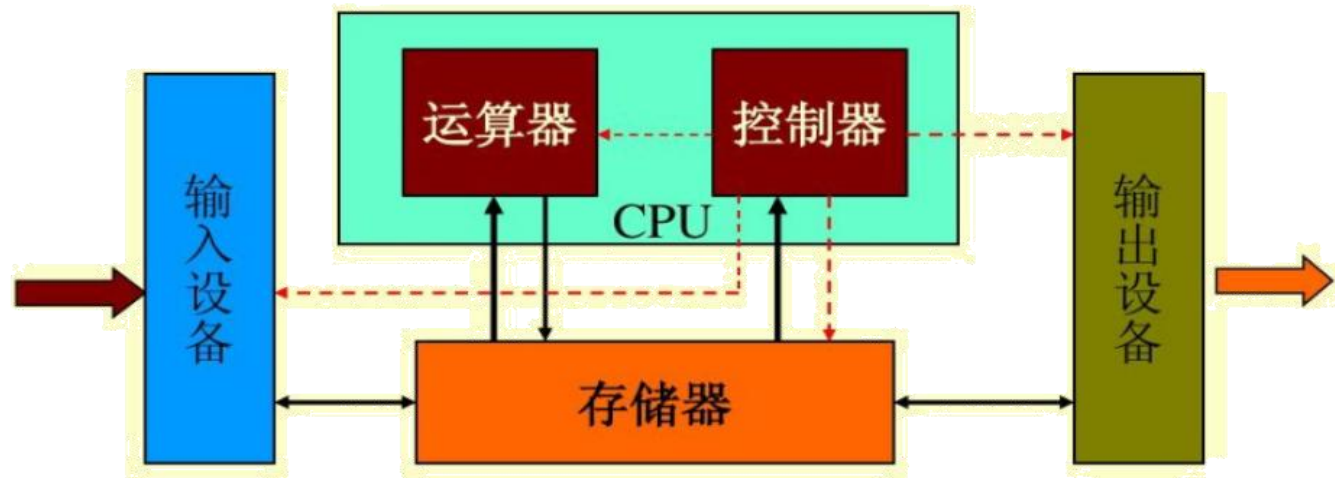
计算机系统是由**硬件系统**和**软件系统**组成的有机整体。

- **硬件系统**：系统赖以工作的实体，包括**CPU**、**存储器**、**输入输出设备**等。
- **软件系统**：各种**程序**和**文件**，包括系统软件、应用软件等，

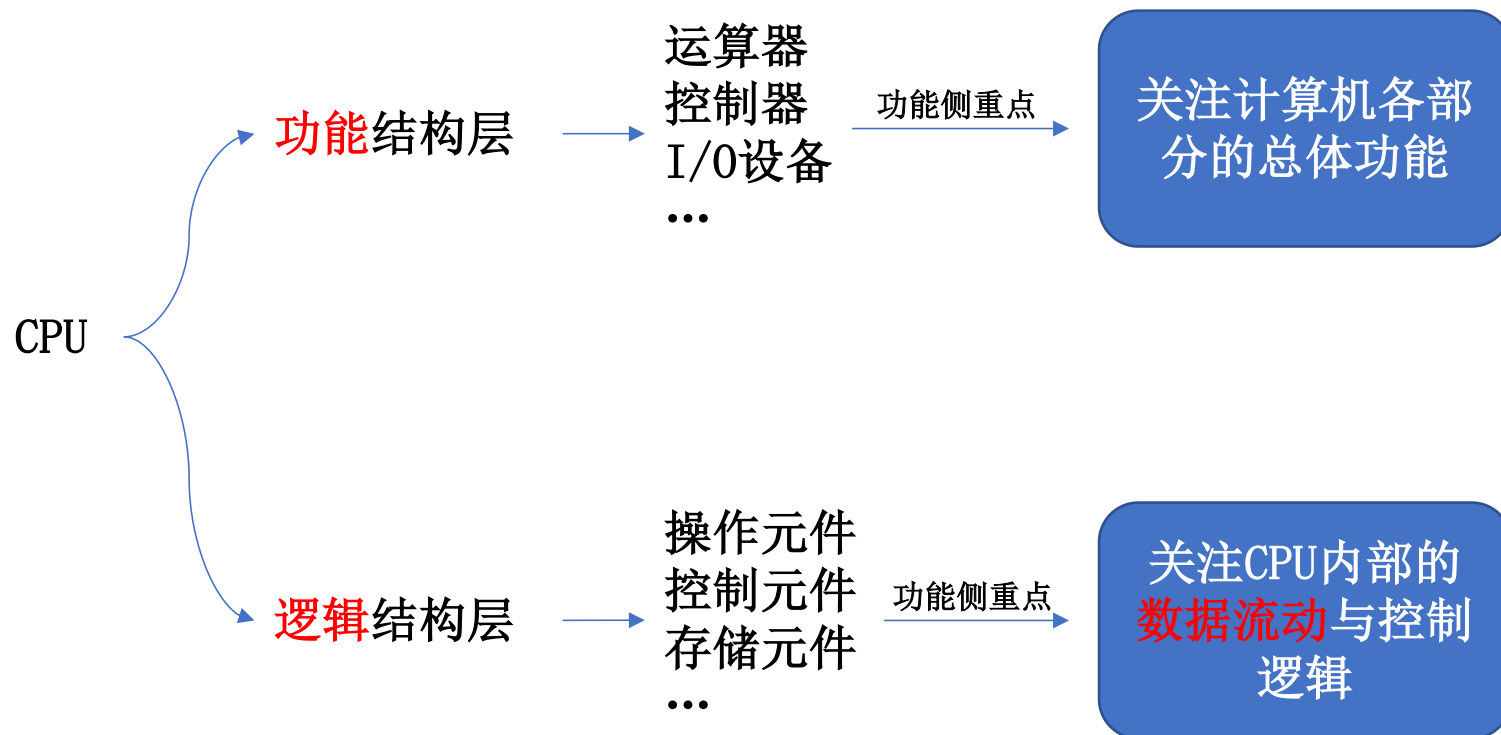


二、系统部件

- **控制器**：计算机的神经中枢，指挥着计算机中各个部件自动协调工作。
- **运算器**：计算机中执行各种**算术**和**逻辑**运算操作的部件。
- **存储器**：计算机用来存放所有**数据**和**程序**的记忆部件。它的基本功能是按指定的地址写入或者读出信息。
- **输入设备**：向计算机中输入信息的设备。
- **输出设备**：计算机数据的输出显示、打印、声音、控制外围设备等。

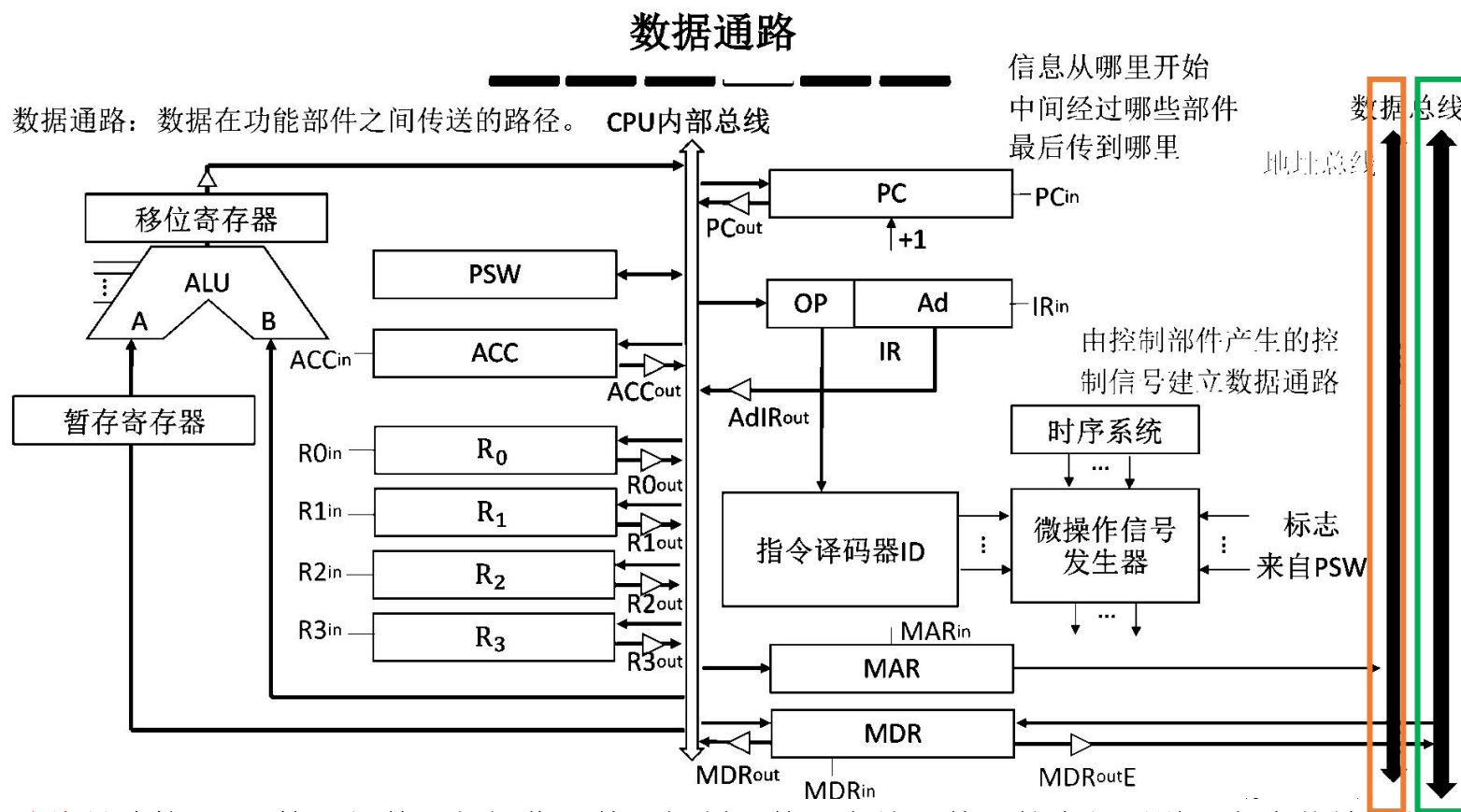


三、中央处理器 (CPU)



本章节主要从**逻辑结构层**对CPU进行讲解！

CPU各功能部件之间的正常工作，依赖于**数据通路**。



- **CPU内部总线**是连接 CPU 核心组件（如操作元件、控制元件、存储元件）的内部通道，负责传输内部指令和数据。
- **地址总线**是 CPU 向内存 / 外设传输存储单元地址的**单向总线**，决定系统可访问的最大内存容量。
- **数据总线**是 CPU 与内存 / 外设**双向传输**实际数据的通道，宽度直接影响数据传输速率。

8.2 数据通路

一、基本概念

1. 定义

- 数据通路：数据在各功能部件之间传送的路径。
- CPU内部的数据通路：运算器与各寄存器之间的数据传送路径。

2. 特点

- 数据在数据通路中的传送操作需要在控制信号的控制下进行。
- 同一指令在不同周期内可走不同的数据通路。
- 例：ADD R1, [内存地址]：把某个内存单元的值加到R1上
 - 取指阶段→取数阶段→执行阶段→写回阶段

3. 作用

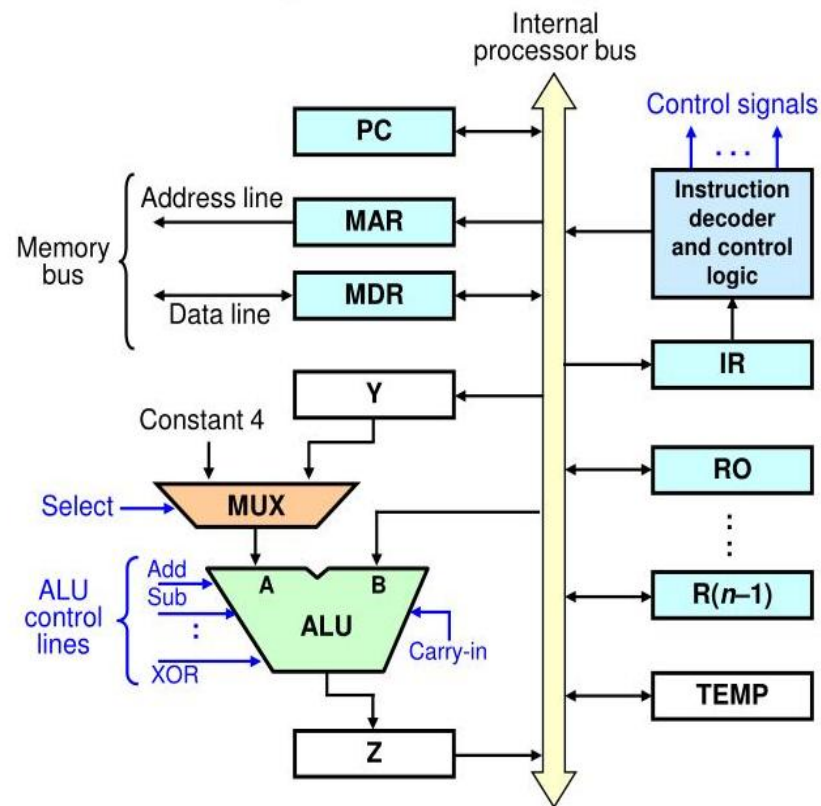
数据通路结构影响：

- CPU内部各种数据的**传送路径**
- 可执行的**微操作类型**
- 控制器设计的**复杂度与效率**

4. 基本结构

- 总线型数据通路
- 专用通路型数据通路

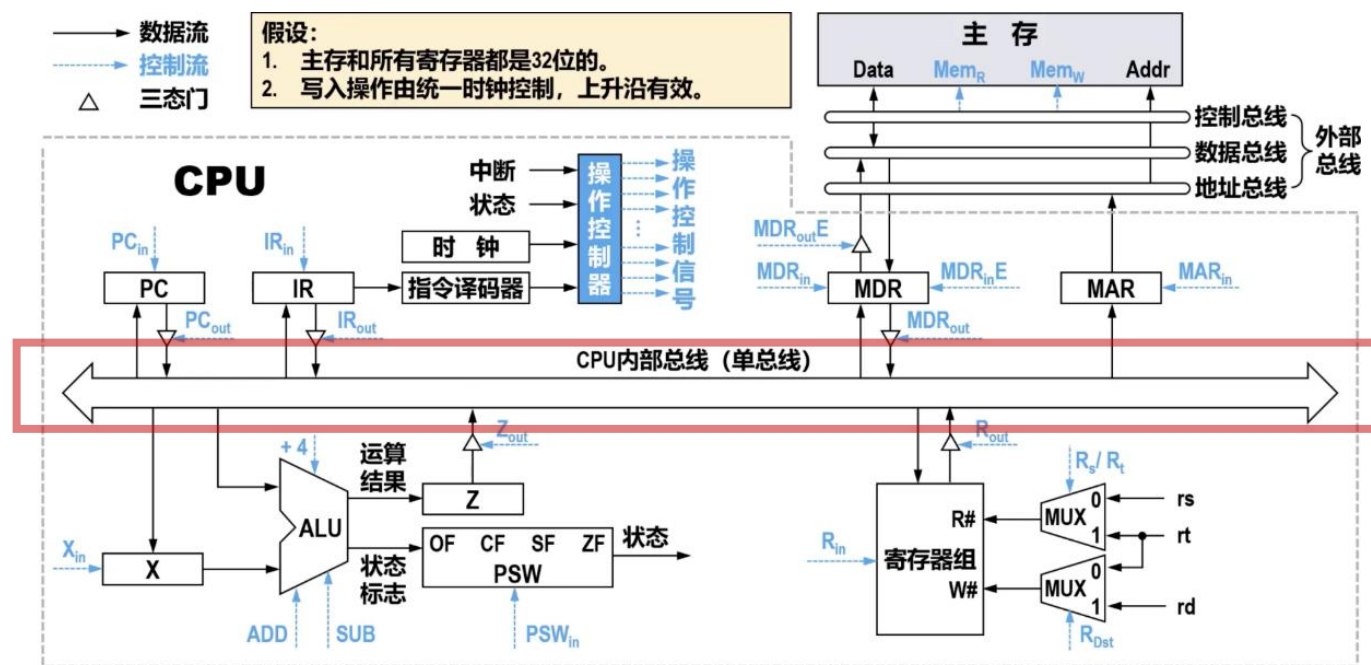
Single-bus Organization



二、单总线型数据通路

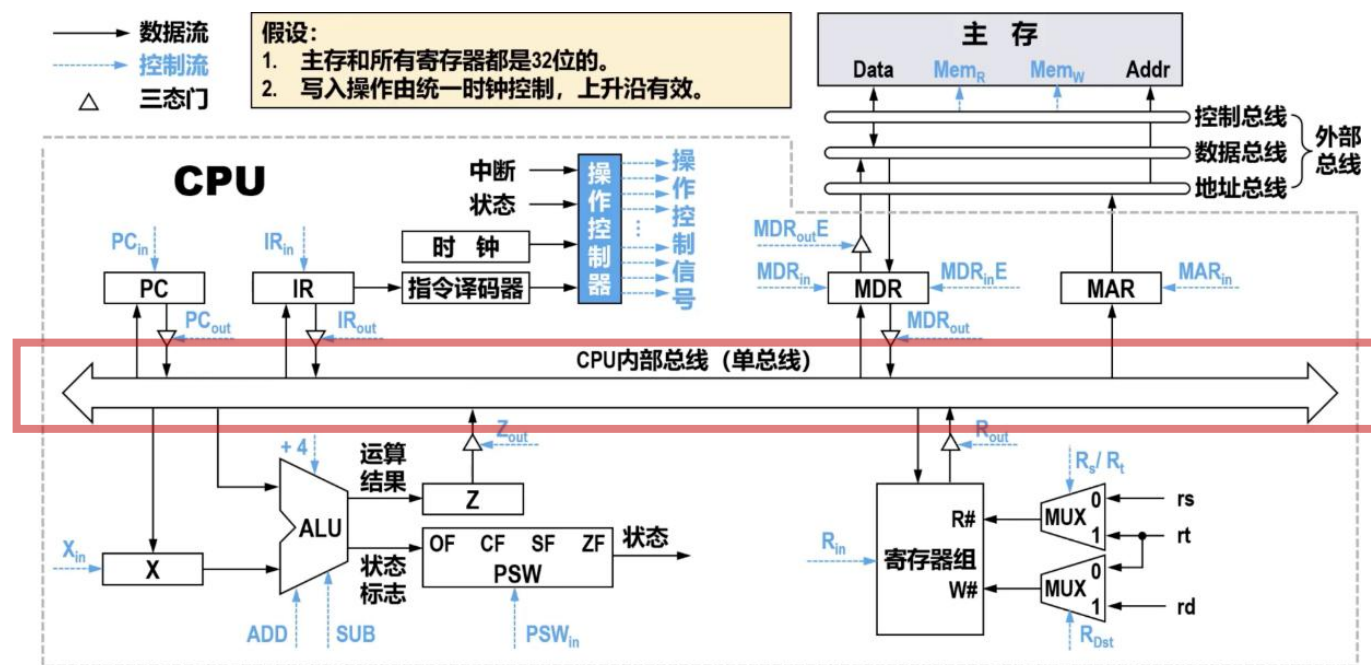
1. CPU内部总线

CPU内部总线连接算数逻辑单元（ALU）、通用寄存器组（GPR）、程序计数器（PC）、指令寄存器（IR）、内存地址寄存器（MAR）、内存数据寄存器（MDR）等寄存器，构成CPU内部的单总线结构数据通路。



运算器、控制器和各类寄存器共享一条内部公共总线，各部件可以**同时**从总线读数据，但**同一时刻只能有一个部件向总线写数据**，其输出端通过**三态门**按需接入或断开总线以避免冲突。

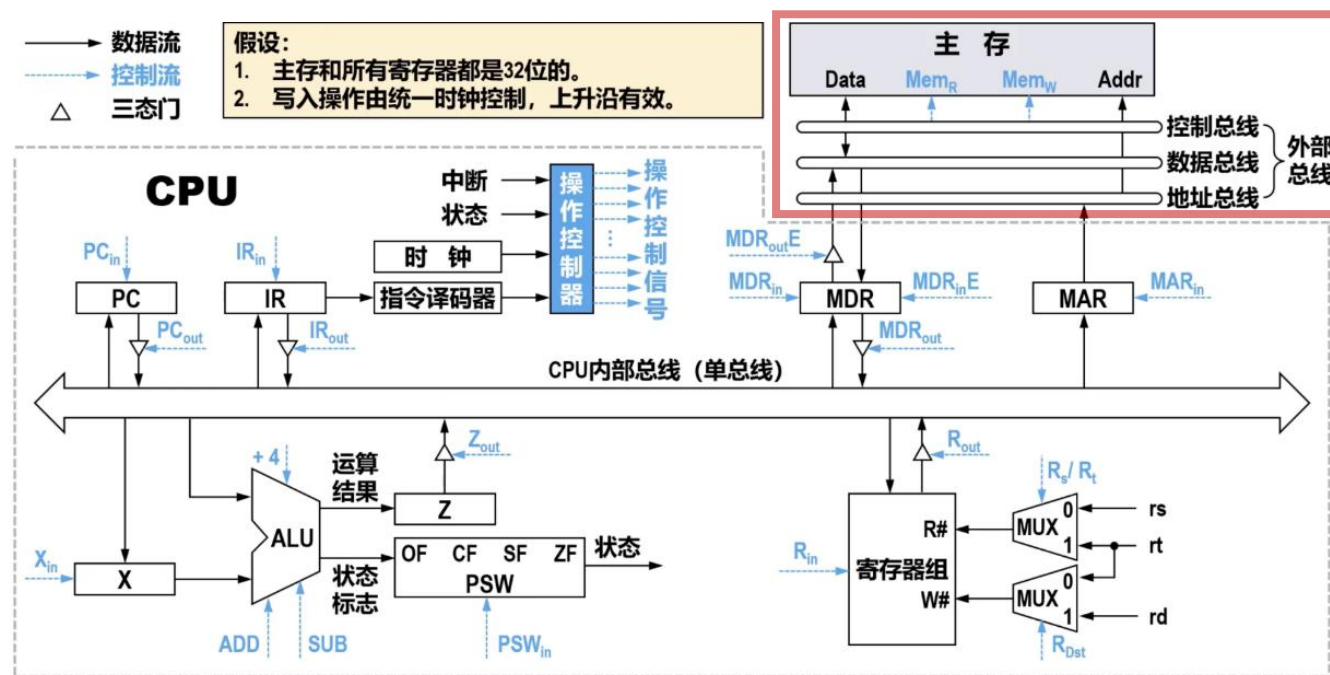
单总线是指各部件共享的一条**逻辑公共总线**，而不是一根物理导线。



2、CPU外部总线

在CPU外部，包含控制总线、数据总线和地址总线等CPU外部总线，用于连接CPU与主存及各类I/O设备，也称为系统总线或外部总线。

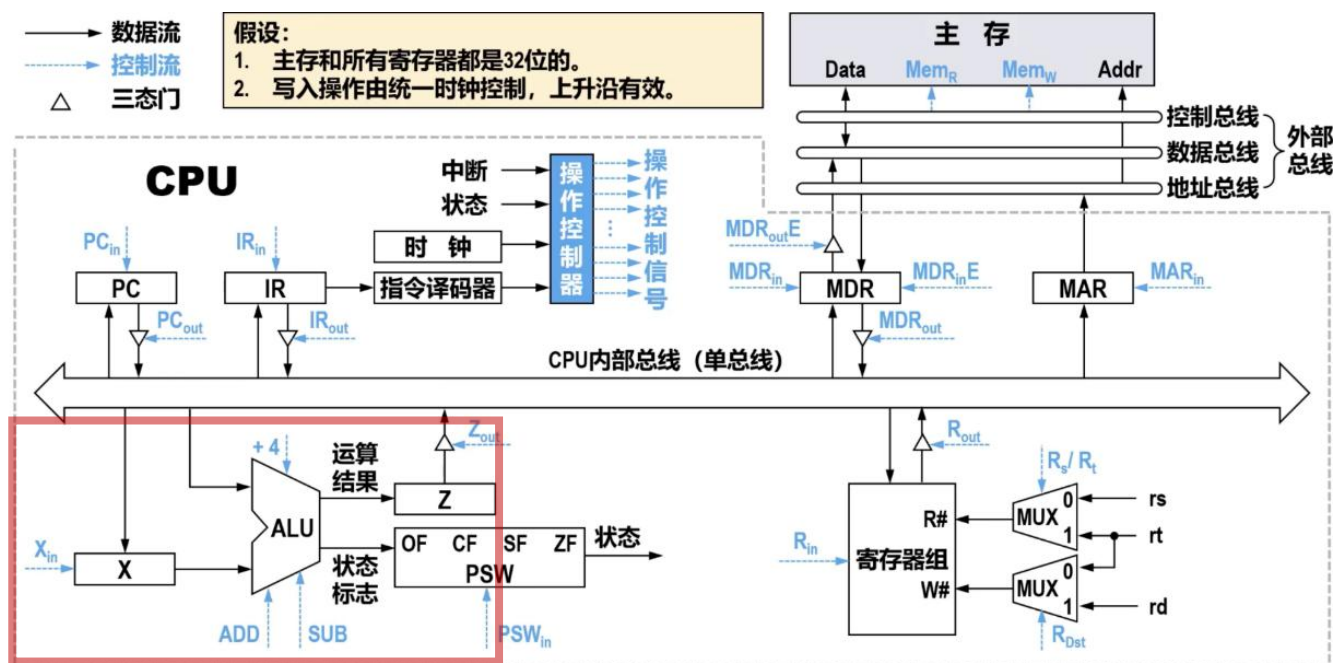
通过内部总线和外部总线，CPU内部各部件与主存、I/O之间的数据都能在统一的总线结构上有序传送。



3、算术逻辑单元

算术逻辑单元 (ALU) 用于完成各种**算术和逻辑运算**。

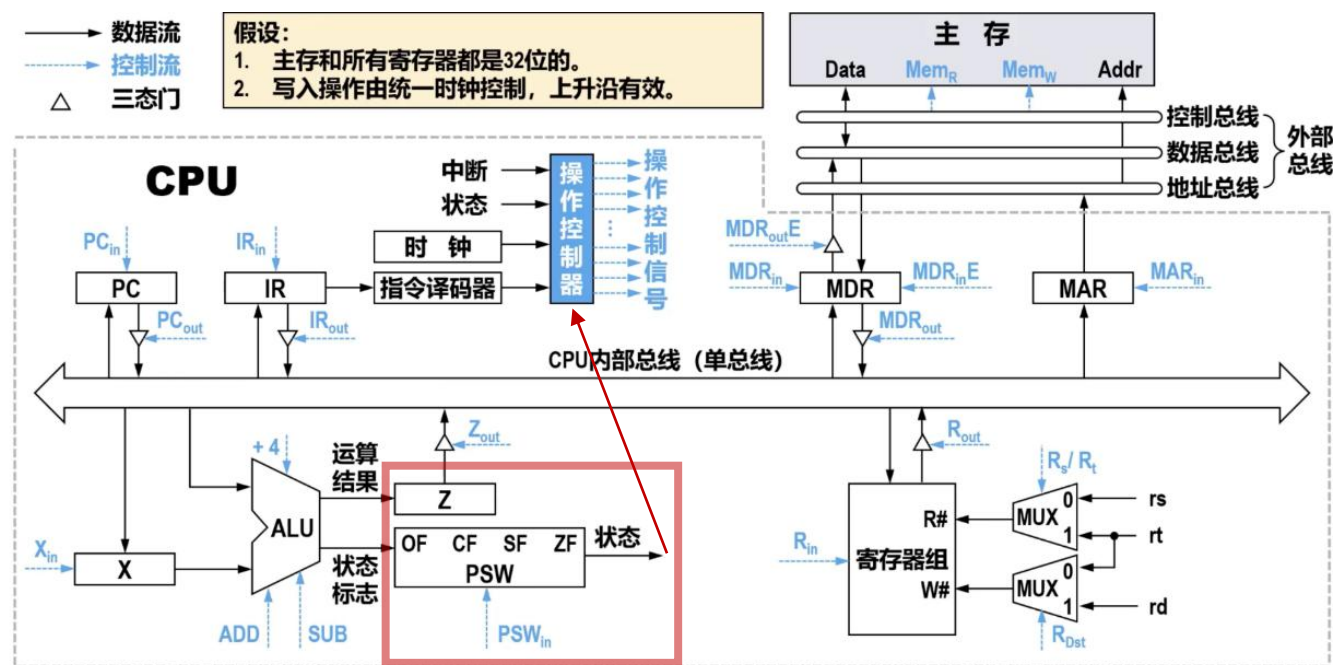
ALU的两个操作数，一个直接来自内部总线，一个由暂存寄存器X在内部总线上取得。运算结果先送入暂存寄存器Z，再通过内部总线写回寄存器组等部件。



4、程序状态字寄存器

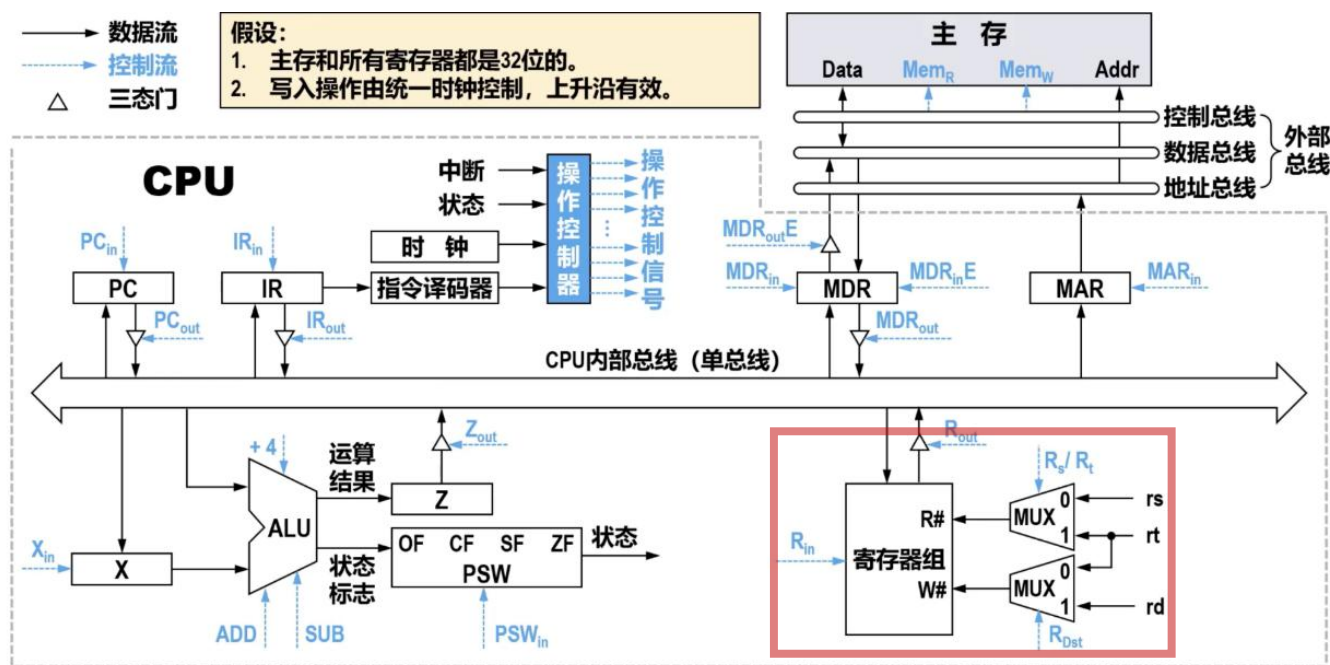
程序状态字寄存器 (PSW) 用于保存ALU运算产生的**状态标志**。

其中，**OF**为溢出标志、**CF**为进位标志、**SF**为符号标志、**ZF**为零标志，这些标志会送入**操作控制器**，影响后续指令的执行（例如条件转移）。



5、通用寄存器组/寄存器堆

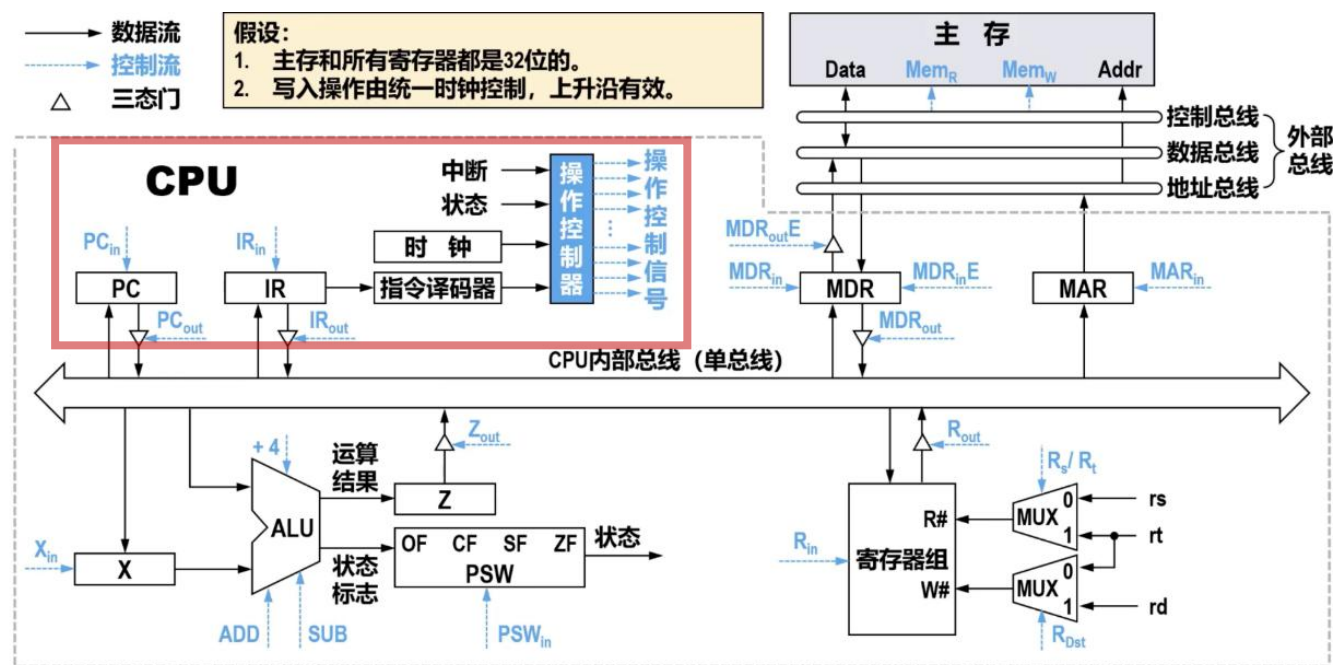
通用寄存器组，即寄存器堆，内部包含一组按编号访问的通用寄存器。通过R#端口选择要读取的寄存器、W#端口选择要写入的寄存器。数据经由CPU内部总线传输，从而在寄存器之间实现快速、灵活的数据读写。



6、程序计数器、指令寄存器、操作控制器

程序计数器(PC)用于保存下一条指令的地址，指令寄存器(IR)暂存当前指令，指令经译码后送入操作控制器。

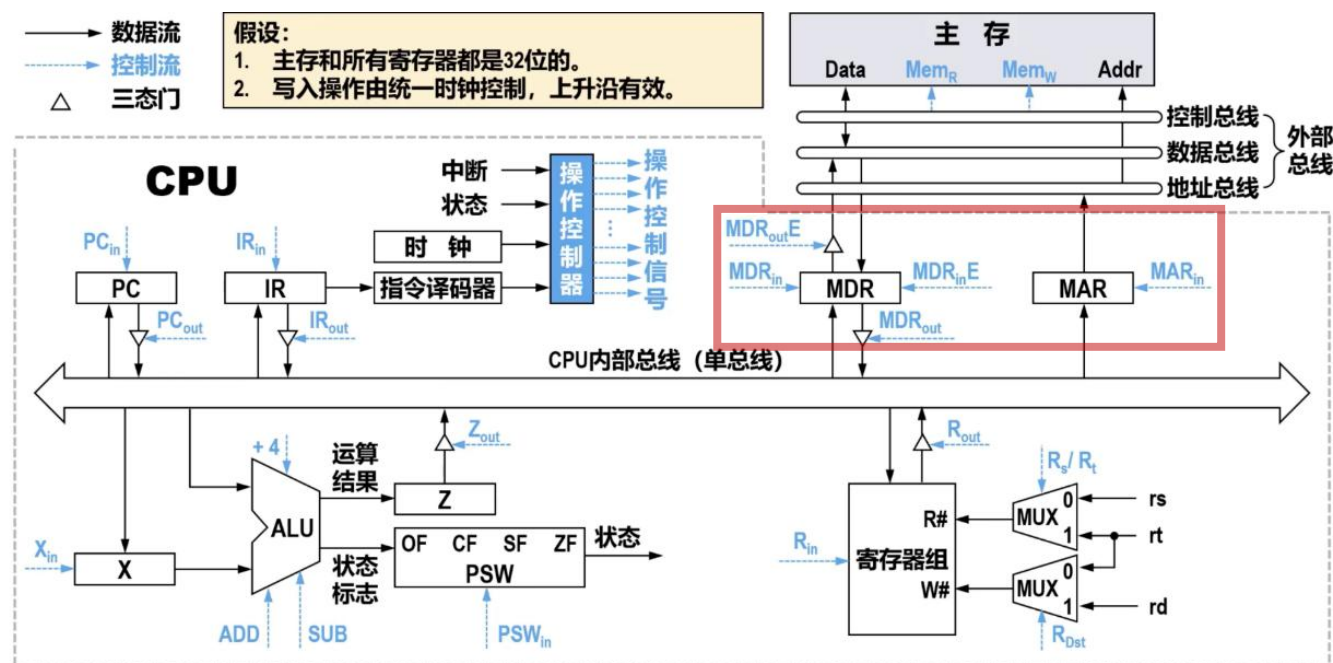
操作控制器产生控制信号，既协调CPU内部各寄存器、ALU和内部总线的工作，又通过控制总线来控制主存和I/O的读写。



7、地址寄存器、数据寄存器

存储器地址寄存器 (MAR) 暂存要访问的**主存地址**，存储器数据寄存器 (MDR) 暂存**要写入或读出的数据**。

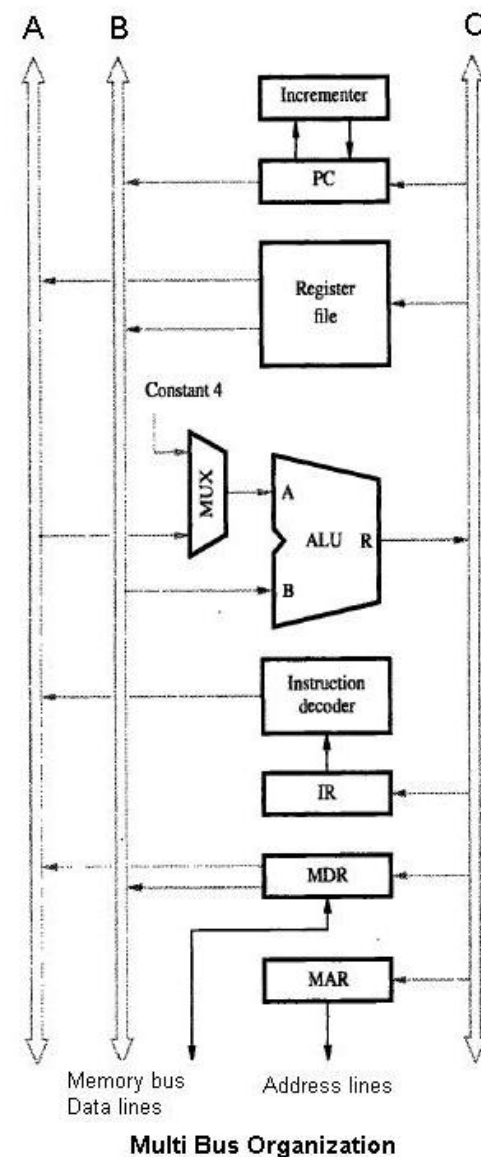
二者一方面通过CPU内部总线与寄存器、ALU交换地址和数据，另一方面通过外部地址总线、数据总线与主存和I/O进行读写。



四、多总线型数据通路

多总线结构在 CPU 内部设置**多条内部总线**(如**A总线**、**B总线**、**C总线**)，将寄存器组、ALU、PC/IR、MAR/MDR等部件分布在**不同总线之间**。

寄存器组通常做成**多端口寄存器堆**，可同时在**两条源操作数总线**(**A、B 总线**)上输出寄存器内容，并在**结果总线**(**C总线**)上写回运算结果。

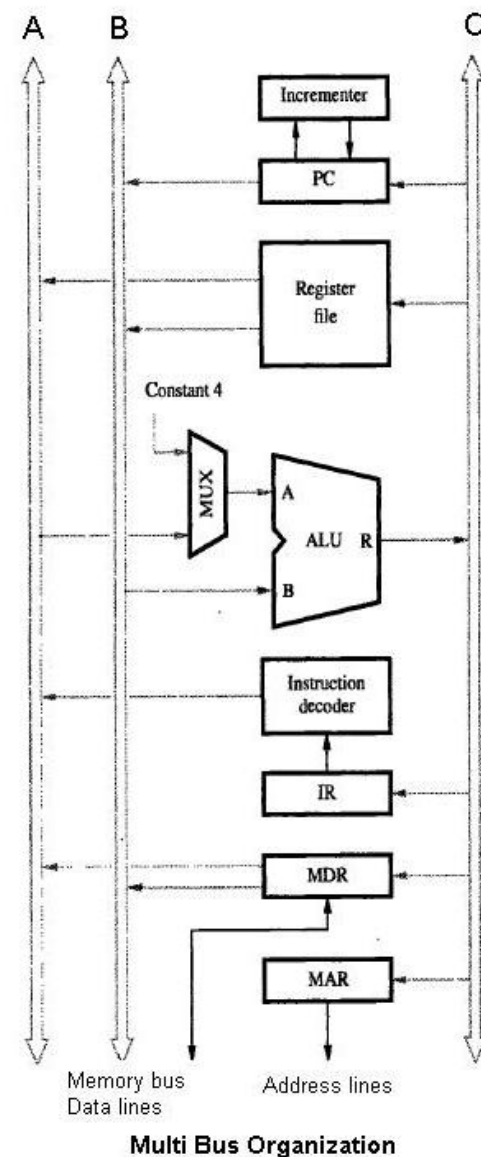


数据通路



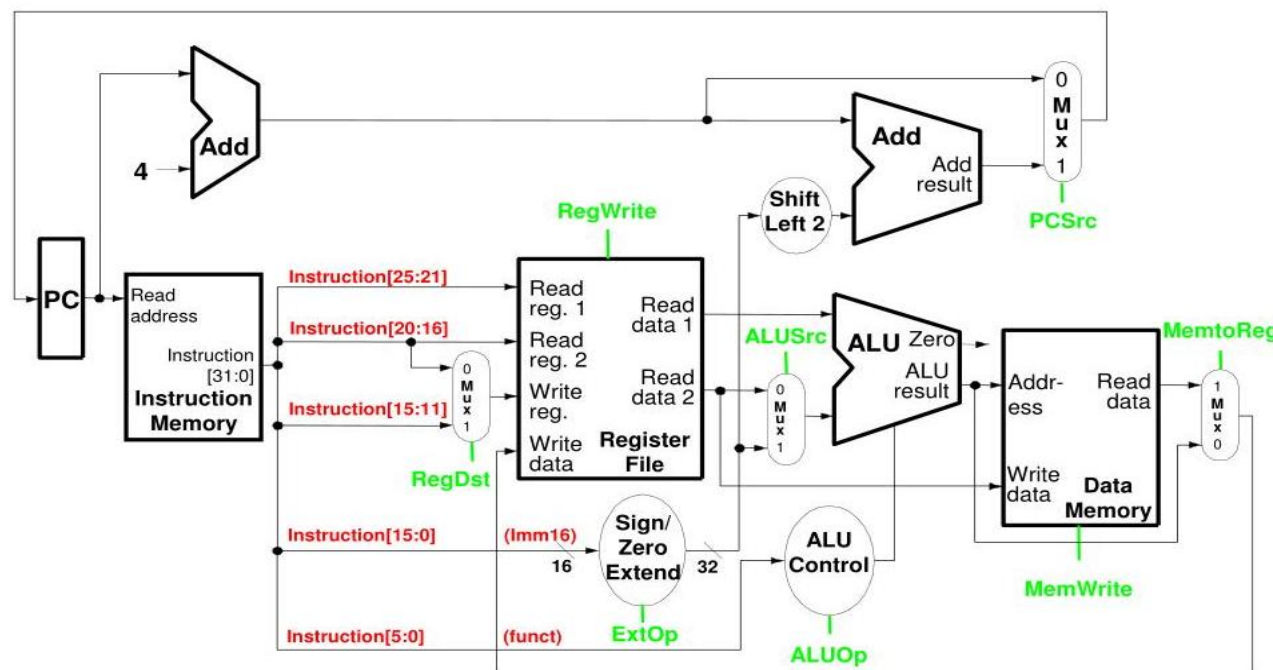
多总线结构可以在一个时钟周期内**并行完成多组数据传送**，减少总线竞争、缩短控制步骤，相比单总线结构**提高了指令执行速度**。

代价是需要更多的**总线、选择器和控制信号**，硬件成本和控制复杂度都高于单总线结构。



四、专用通路型数据通路

专用通路型数据通路在CPU内部为寄存器组、ALU、PC、存储器等部件单独铺设点对点连线，而不是共享一条总线，以**减少竞争、提高速度**，但是结构较为复杂。下图为一周期CPU中的专用通路结构。



8.3 操作元件

一、操作元件定义

操作元件是数字系统中用于执行特定算术运算或逻辑运算的功能单元。操作元件接收二进制数据作为输入，在控制信号的指示下，完成特定的算术、逻辑或移位操作，并输出结果。

二、操作元件与组合逻辑电路的关系

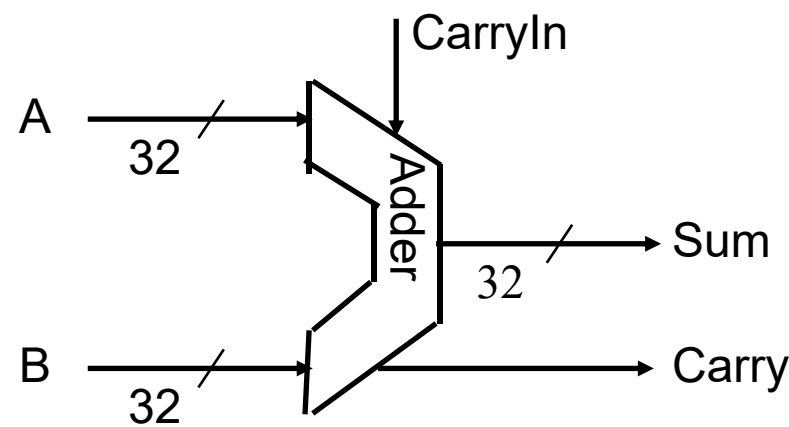
操作元件是由**组合逻辑电路**构建而成的物理实现。



三、典型操作元件

1. 加法器(Adder)

加法器是一种实现二进制加法运算的组合逻辑电路，是计算机算术逻辑单元(ALU)的基本组成部分。



信号名称	位宽	类型	说明
A	32位	输入	第一个加数
B	32位	输入	第二个加数
CarryIn	1位	输入	来自低位加法器的进位输入
Sum	32位	输出	加法结果
Carry	1位	输出	加法后的进位输出

- 应用场景举例：程序计数器更新

CPU需要顺序执行指令。程序计数器（PC）存储着下一条要执行指令的地址。绝大多数情况下，指令是顺序存放的，下一条指令的地址就是当前地址加上一个固定偏移量（通常是4字节）。

- 加法器在其中的作用：

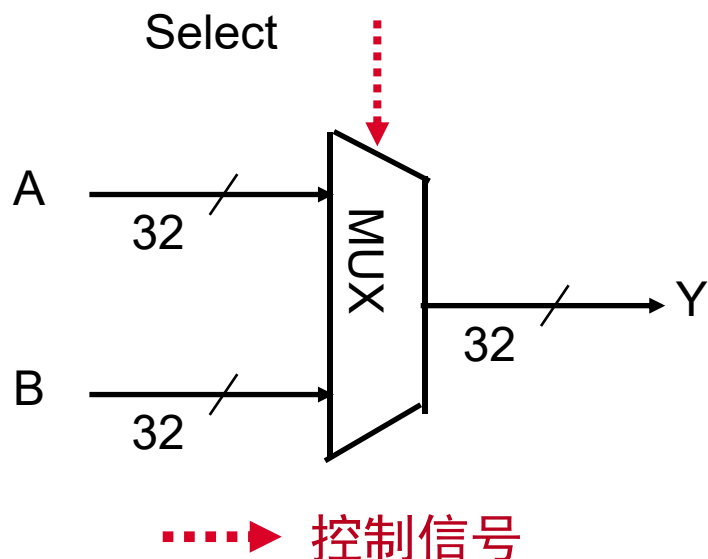
一个专用的加法器持续计算 $PC + 4$ ，并将其结果写回PC寄存器。这确保了指令流能够自动、连续地向前推进。

在地址计算与后续的算术运算实现中，加法器都是极度重要的组件。

2. 多路选择器(MUX)

多路选择器从多个输入信号中，根据选择信号（Select），**选择其中一路输出**。

举例：2:1 多路选择器



Select	Y (输出)
0	A
1	B

- 应用场景举例：ALU数据源选择

ALU可以执行多种运算，其输入数据可能来自不同的地方。例如，对于 `add $t0, $t1, $t2`（寄存器相加，即 $\$t0 = \$t1 + \$t2$ ）和 `addi $t0, $t1, 100`（寄存器与立即数相加，即 $\$t0 = \$t1 + 100$ ），ALU的第二个输入源是不同的。

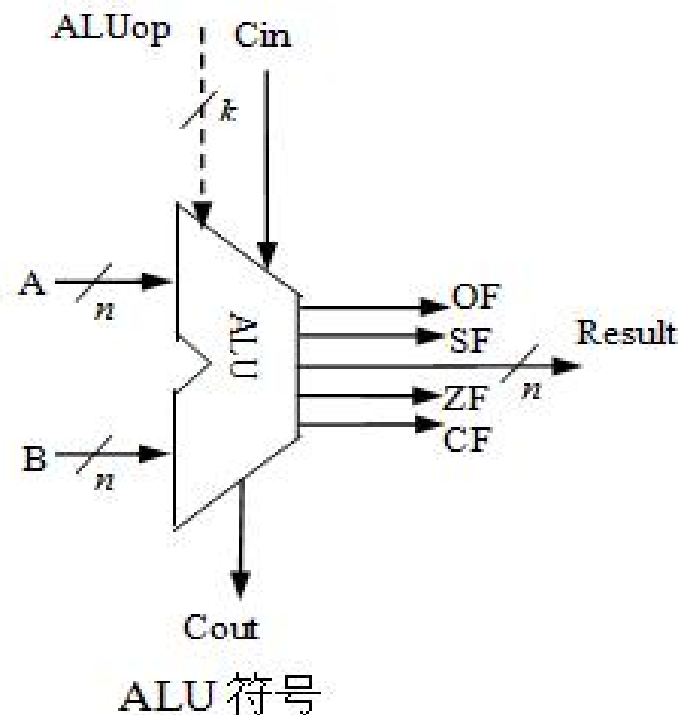
- 多路选择器在其中的作用：

一个多路选择器被放置在ALU的B输入端前。它的一个数据端连接寄存器文件，另一个连接符号扩展后的立即数。控制器根据指令操作码产生“选择信号”，决定将哪一路数据送入ALU。

多路选择器控制“数据从哪里来，到哪里去”。

3. 算术逻辑单元 (ALU)

- 进行**基本**算术运算与逻辑运算
 - 无符号整数加、减
 - 带符号整数加、减
 - 与、或、非、异或等逻辑运算
- 核心电路是**整数加/减**运算部件
- 输出运算结果与标志信息
- 有一个**操作控制端** (ALUop)，用来决定ALU所执行的处理功能。

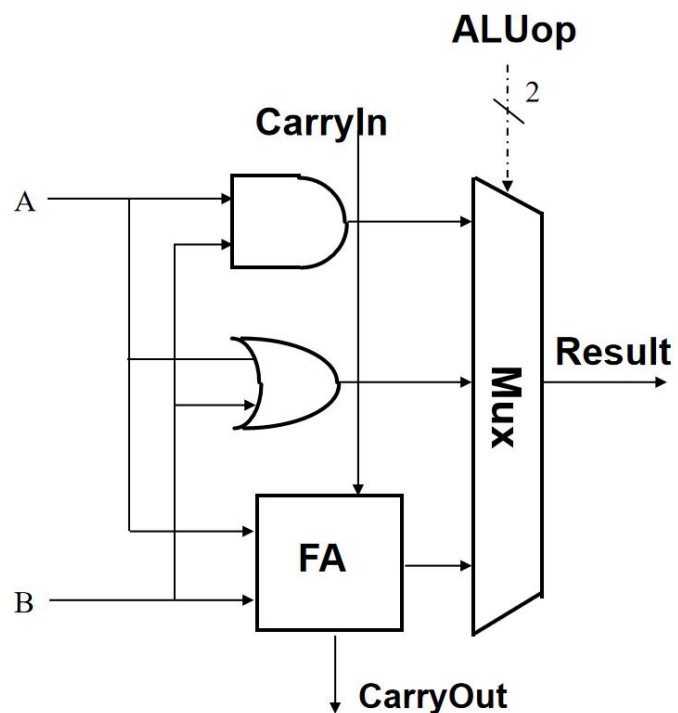


ALUop的位数 k 决定了操作的种类，例如，当位数 k 为3时，ALU最多只有 $2^3=8$ 种操作。

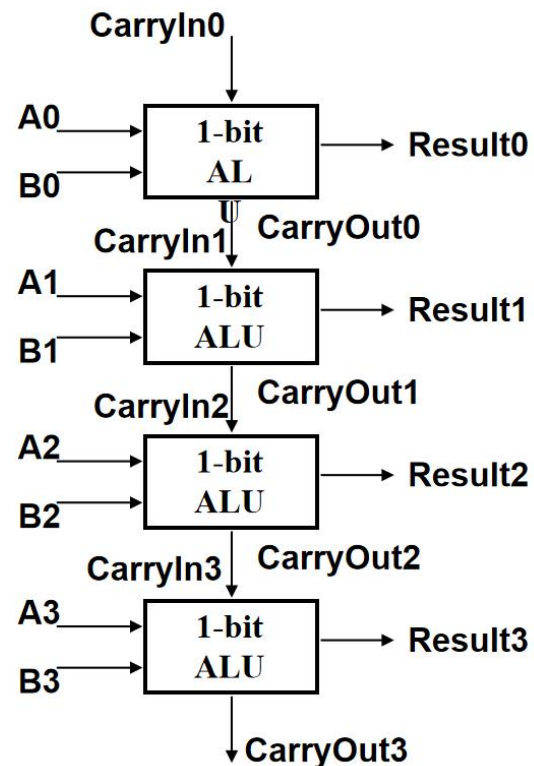
典型的三位控制信号ALU，常见于MIPS、RISC-V

ALUop	Result	ALUop	Result	ALUop	Result	ALUop	Result
0 0 0	A加B	0 1 0	A与B	1 0 0	A取反	1 1 0	A
0 0 1	A减B	0 1 1	A或B	1 0 1	$A \oplus B$	1 1 1	未用

1-bit ALU



4-bit串行ALU



重要认识： 计算机中所有算术运算都基于加法器实现！

- 应用场景举例：条件判断与分支决策

执行 `beq $t1, $t2, label` 指令（如果 `$t1` 等于 `$t2`，则跳转到 `label` 标签处执行；否则继续顺序执行下一条指令）时，需要判断 `$t1` 和 `$t2` 是否相等。

- ALU在其中的作用：

ALU执行一个减法操作 $\$t1 - \$t2$ 。虽然我们不关心减法的具体结果，但ALU会生成一个至关重要的“零标志位”。如果结果为0，则零标志位为1，表示两数相等。

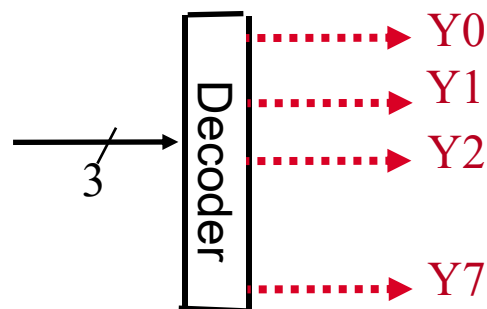
ALU不仅提供强大的计算功能，更是通过标志位参与决策，驱动程序做出不同分支。

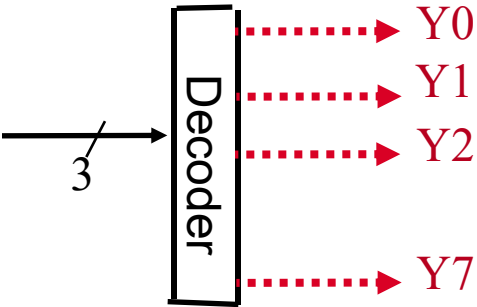
4. 译码器(Decoder)

译码器是将二进制编码信号转换为唯一输出信号的数字逻辑电路。

举例：3-to-8 译码器的功能是：

输入 3 位二进制信号 (A_2 、 A_1 、 A_0) ， 输出 8 条信号线 ($Y_0 \sim Y_7$) 中只有一条为高电平 (1) ， 其余为低电平 (0) 。





$$\begin{aligned} Y_0 &= \overline{A_2} \overline{A_1} \overline{A_0} \\ Y_1 &= \overline{A_2} \overline{A_1} A_0 \\ Y_2 &= \overline{A_2} A_1 \overline{A_0} \\ Y_3 &= \overline{A_2} A_1 A_0 \\ Y_4 &= A_2 \overline{A_1} \overline{A_0} \\ Y_5 &= A_2 \overline{A_1} A_0 \\ Y_6 &= A_2 A_1 \overline{A_0} \\ Y_7 &= A_2 A_1 A_0 \end{aligned}$$

输入	Y ₀	Y ₁	Y ₂	Y ₃	Y ₄	Y ₅	Y ₆	Y ₇
000	1	0	0	0	0	0	0	0
001	0	1	0	0	0	0	0	0
010	0	0	1	0	0	0	0	0
011	0	0	0	1	0	0	0	0
100	0	0	0	0	1	0	0	0
101	0	0	0	0	0	1	0	0
110	0	0	0	0	0	0	1	0
111	0	0	0	0	0	0	0	1

- **应用场景举例：**寄存器选择与内存地址译码
一条指令需要读写特定的寄存器。寄存器文件有32个寄存器，需要根据5位的寄存器编号选中其中一个。
- **译码器在其中的作用：**
在寄存器文件内部，有地址译码器。它将5位的寄存器编号（如 10000）作为输入，转换为32根“独热码”选择线之一（如第16根线为高电平），从而激活对应的寄存器进行读写。

译码器将抽象的二进制代码（指令、地址）翻译为具体的、物理级的动作信号。

四、元件关键关联说明

1. 译码器 → 多路选择器/ALU

- 控制单元中的译码器产生选择信号 (ALUSrc) 控制多路选择器
- 控制单元产生ALUOp信号指导ALU执行特定操作

2. 多路选择器 → ALU

- 多路选择器为ALU筛选和提供正确的操作数
- 多路选择器是数据流向ALU的“必经之路”

3. ALU → 多路选择器

- ALU的计算结果作为多路选择器的输入选项之一
- 实现运算结果的后续路由选择

4. 加法器作为ALU的核心

- 加法器是ALU内部的基础运算单元
- 其他复杂运算(减法、乘法)都基于加法器构建



五、元件功能与特点总结

元件类型	核心功能	输入特性	输出特性	依赖关系	配合对象
译码器	地址 / 编码 转换	N位编码输入	2 ^N 位独热码输出	依赖指令寄存器或地址寄存器	为寄存器文件、内存控制器提供选择信号
多路选择器	数据路由选择	多路数据输入 + 选择信号	单路数据输出	依赖控制单元的选择信号	为ALU筛选操作数，为寄存器选择写入源
ALU	算术逻辑运算	操作数 A / B + 操作码	运算结果 + 标志位	依赖多路选择器提供操作数	执行控制单元指定的运算，输出供后续使用
加法器	专用加法运算	操作数 A / B + 进位输入	和+进位输出	作为ALU的核心组件	为ALU提供基础加法能力，独立用于地址计算



8.4 状态元件

一、状态元件的特点

数据通路中的**时序逻辑元件**作为**状态/存储元件**，与组合逻辑元件共同负责**处理和传递数据**。

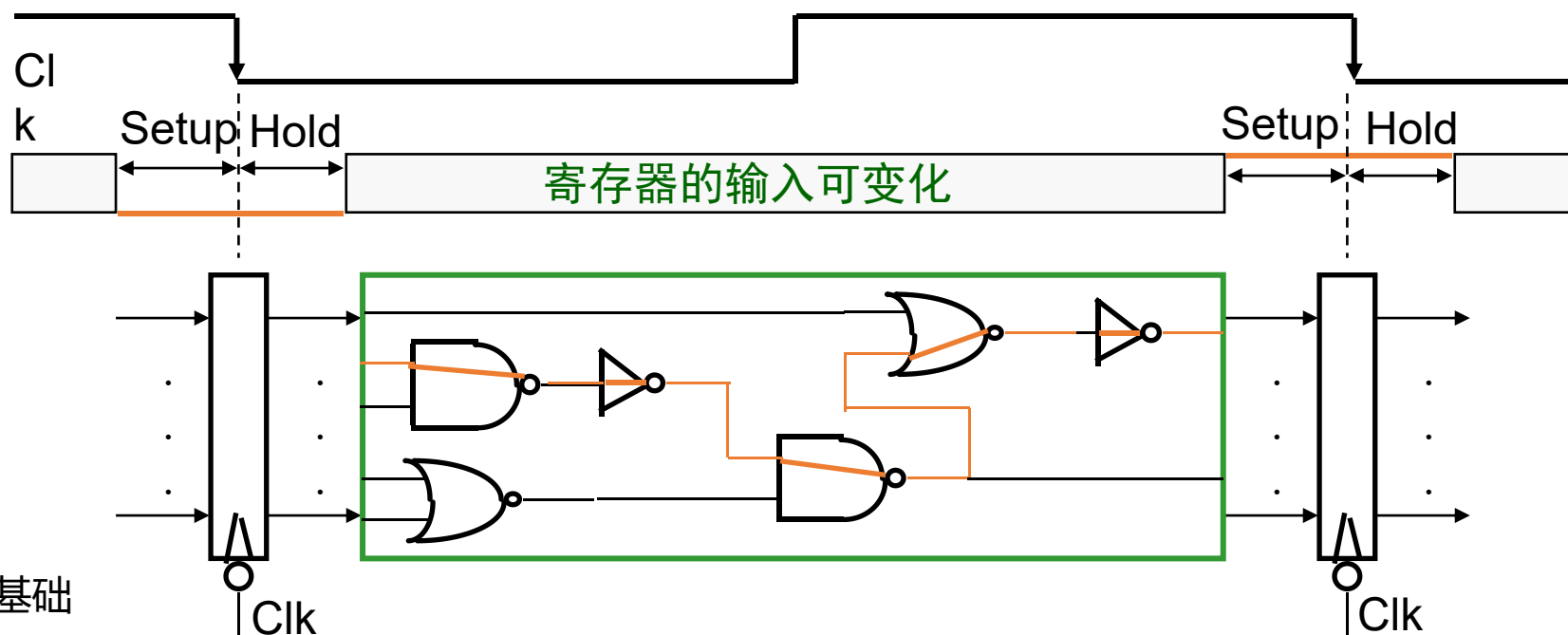
状态元件具有**存储功能**，通常在**时钟控制**下输入被写到电路中，直到下个时钟到达。输入端状态由时钟决定何时被写入，输出端状态随时可以读出。

特性维度	组合逻辑元件	时序逻辑元件
功能	执行数据变换	存储数据，如指令地址、中间结果等
输出依赖性	仅取决于当前输入。输入不变，输出不变	输出取决于时钟信号控制的先前输入和当前输入
时序控制	无时钟信号控制，输入变化经延迟后直接影响输出	受时钟信号同步控制，通常在时钟边沿更新状态
代表性部件	ALU、多路选择器、译码器	程序计数器、通用寄存器组、指令寄存器、数据寄存器等

数据通路由 “... + 状态元件 + 操作元件(组合电路) + 状态元件 + ...” 组成

- Longest Delay: 最长组合逻辑传播延迟
- 时钟偏移: 时钟信号到达不同寄存器的时间
- 建立时间: 在下一个时钟沿到达前, 数据必须提前稳定在目的寄存器D输入端的时间
- Clk-to-Q时间: 时钟沿到达后, 寄存器输出Q端从旧值变为新值并稳定所需的时间

Cycle Time = Clk-to-Q时间 + Longest Delay + 建立时间 + 时钟偏移



二、状态元件的作用

- 既有组合逻辑，为何还需状态元件？若无状态元件，CPU能否执行指令？

组合逻辑只能实现数据运算与逻辑处理，若CPU仅含组合逻辑，**无法保存执行的状态**并且**无法连续执行指令**。

- 状态元件的作用：

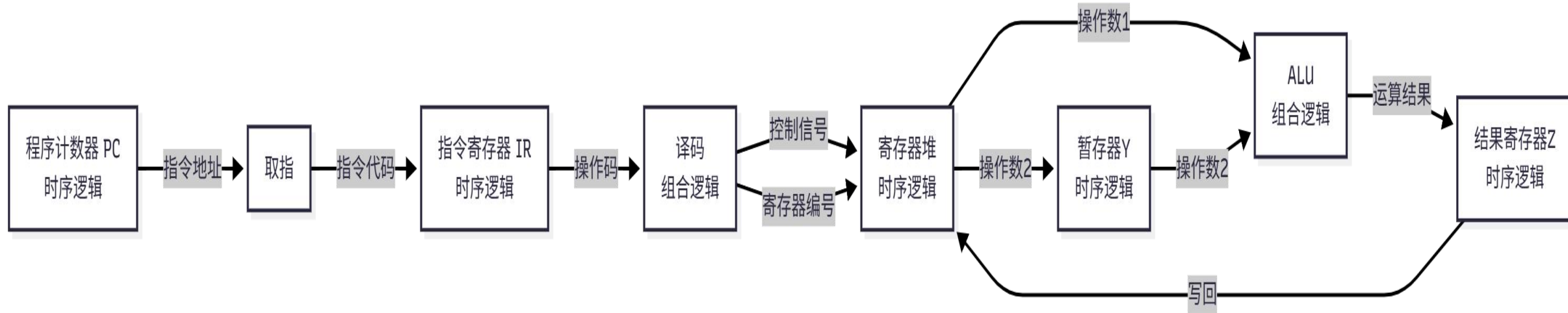
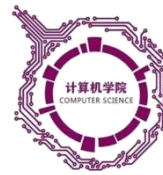
1. **保存中间结果**：维持系统状态，保存PC、寄存器内容及中间计算结果，维持运算连续性。

2. **时序同步**：确保操作按序执行，在时钟信号控制下更新输出，驱动数据在各周期间有序流动，确保CPU内部操作同步、有节奏。

CPU之所以能“执行指令”，是因为**状态元件赋予其记忆能力与时序控制能力**。没有状态元件，CPU无法保存“当前执行状态”，永远停留在“瞬时运算”，无法形成连续的程序执行过程。

功能	状态元件作用	无状态元件后果
程序计数器PC	保存当前指令地址	永远执行同一条指令
寄存器文件	保存操作数与结果	结果瞬间消失
流水线寄存器	保持阶段数据	信号紊乱，无法同步
条件跳转	依赖上次比较结果	无法跳转/循环

状态元件



上图为**ADD指令示意图**的数据通路，其中程序计数器、指令寄存器、寄存器堆、暂存器Y、结果寄存器Z等构成了主要的状态元件。

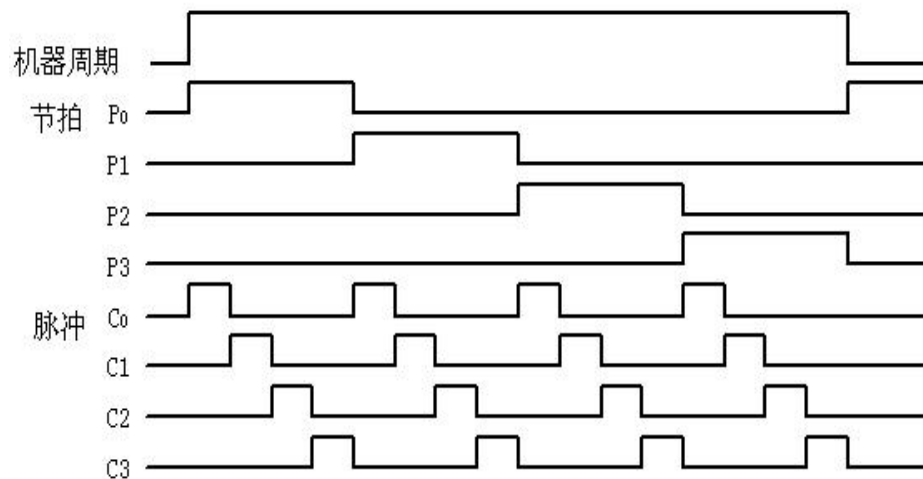
各个状态元件在**时钟控制**下实现了指令执行的**阶段划分与数据传递**。

状态元件将组合逻辑之间的**信号隔离**，使每个阶段的输出能够在下一个时钟周期稳定作为输入。

三、时钟

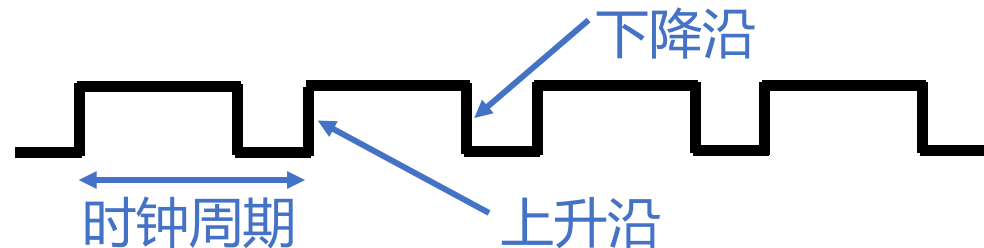
时钟为系统提供统一时间参考，是全系统时间基准，确保系统协调工作。保证系统同步运行，避免操作混乱或冲突。

早期计算机用三级时序系统控指令执行。后CPU结构变复杂，为减延迟、提主频，简化为统一时钟周期控制。



四、触发方式

规定信号何时写入状态元件
或何时从状态元件读出。



1. **边沿触发**：仅在时钟跳变瞬间采样输入，其余时段输出恒定。

优点：抗干扰强，系统同步优，规避竞态。

缺点：硬件构造稍显复杂。

应用：几乎所有寄存器、状态机等时序逻辑均采用此模式。

2. **电平触发**：时钟处于高（或低）电平时，输入实时作用于输出。

优点：电路简洁，响应迅速。

缺点：易引发竞态与毛刺，时序欠稳。

应用：多见于锁存器。

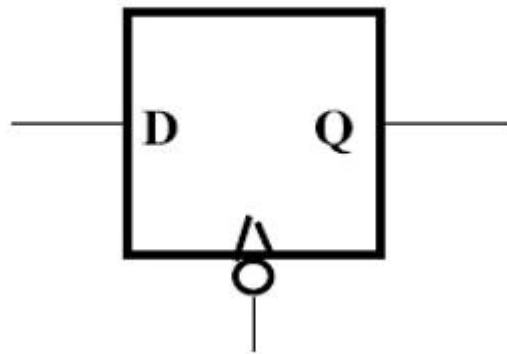
3. 其它触发方式还有：**主从式触发、异步触发、双边沿触发等。**

五、典型状态元件

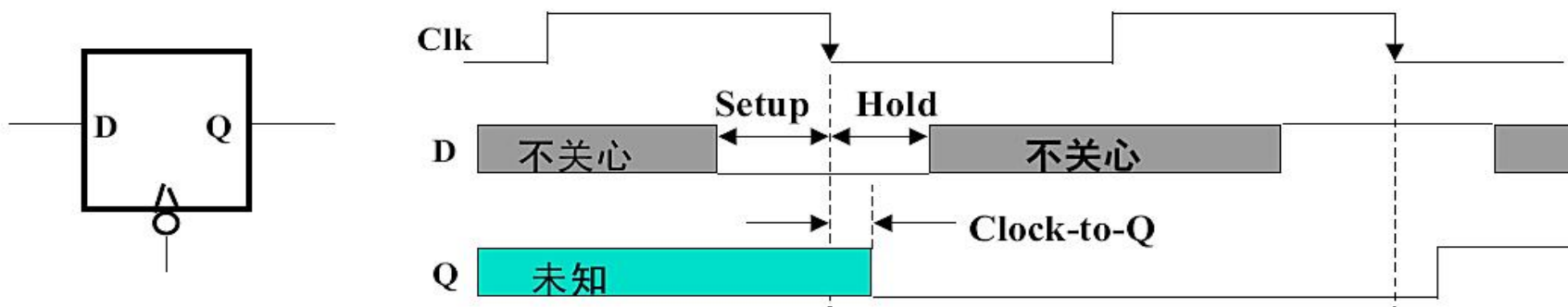
1. D触发器

最简单的状态单元：一个时钟输入、一个状态输入、一个状态输出。

D触发器在触发边沿到来时，将输入端的值存入其中，并且这个值与当前存储的值无关。



状态单元的输入信息总是在一个时钟边沿到达后的“Clk-to-Q”时间才被写入，此时的输出才反映新的状态值。



- 建立时间（**Setup Time**）：在触发时钟边沿 **之前** 输入必须稳定
- 保持时间（**Hold Time**）：在触发时钟边沿 **之后** 输入必须保持
- **Clock-to-Q time**:
 - 在触发时钟边沿，输出并不能立即变化

D触发器是最基础的时序逻辑单元，它在时钟边沿把输入数据D锁存为稳定的状态，因此被广泛用于构建各种数字系统的状态存储与时序控制功能：

- 构成寄存器与寄存器堆，构建流水线寄存器
- 构成有限状态机（FSM）
- 数据同步与跨时钟域传输
- 分频器、计数器等时序电路的基础单元
- 作为延时单元，一个D触发器就能实现对信号延迟一个时钟周期，是时序对齐、数据暂存的基本手段。

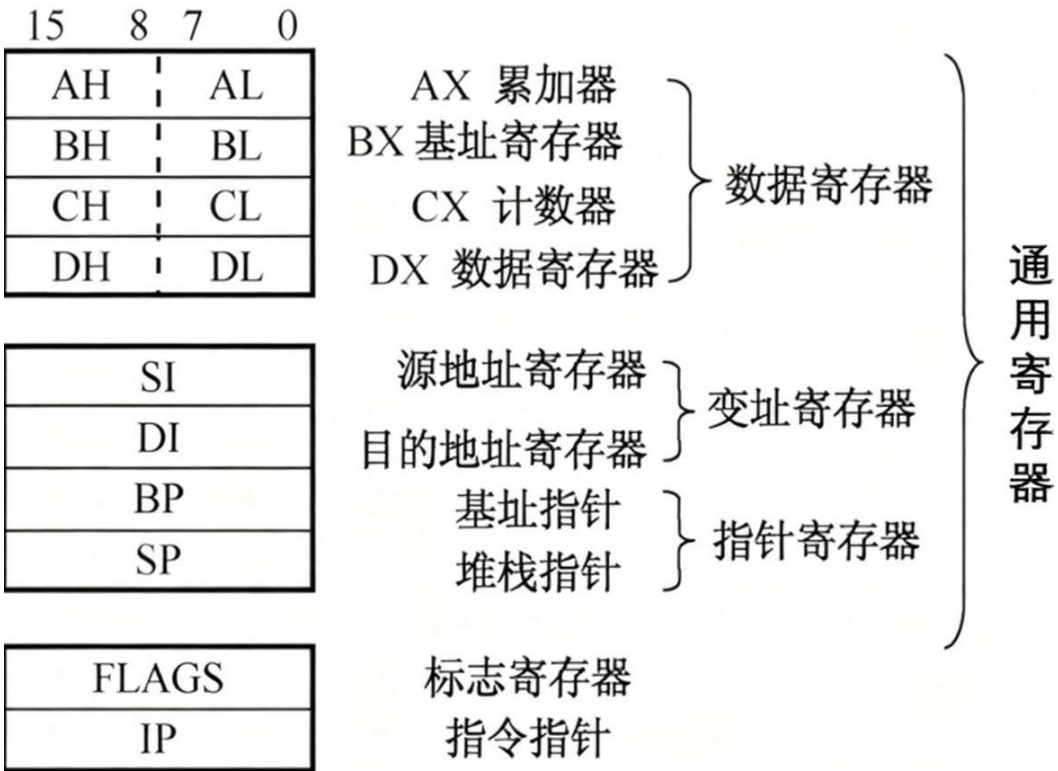


2. 通用寄存器

通用寄存器暂存运算数据等关键信息，在计算机硬件中占据核心地位。

通用寄存器的种类和数量，反映处理器的复杂度和功能多样性，是衡量计算机性能的重要指标。

体系结构	通用寄存器数量	特点
x86	8 → 16 → 32	多用途但部分有历史功能限制（AX、BX等）
MIPS	32 个	简洁统一，每个寄存器功能对等
ARM	16/32 个	支持条件执行与多模式寄存器组
RISC-V	32 个（x0~x31）	x0恒为0，简化硬件设计

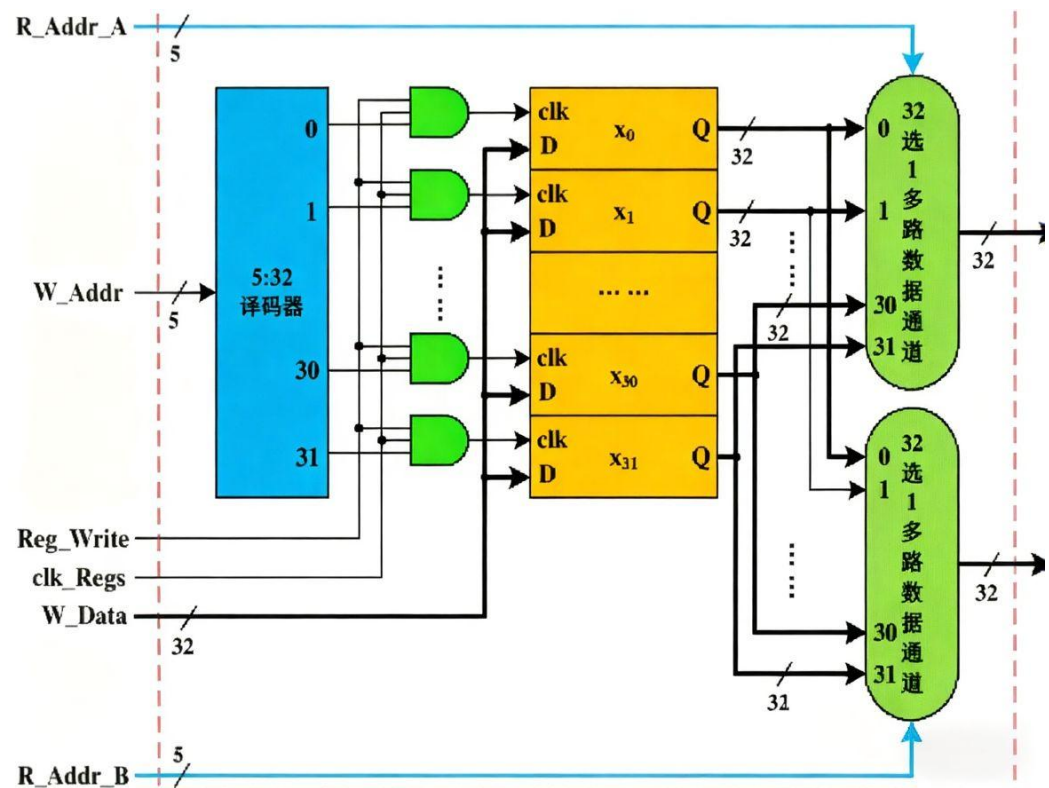


3. 寄存器堆

寄存器堆通常由**多个通用寄存器组成**，并支持多端口并行读写，这样在流水线执行过程中，可以同时读取多个操作数或写回计算结果。

在RISC-V、MIPS等精简指令集架构中，寄存器堆承担着**减少对主存访问、加速指令执行**的重要角色。

寄存器堆配合寄存器转发机制，有效解决数据冒险问题，确保指令依赖关系得到正确处理。



4. 专用寄存器

专用寄存器是处理器中用于**存放特定功能信息**的寄存器，其与通用寄存器不同，不直接参与普通算术或逻辑运算，而是承担控制、状态或系统管理等特殊任务。专用寄存器通常由指令集架构**提供专门访问方式**，并在CPU的控制逻辑和异常处理机制中起关键作用。

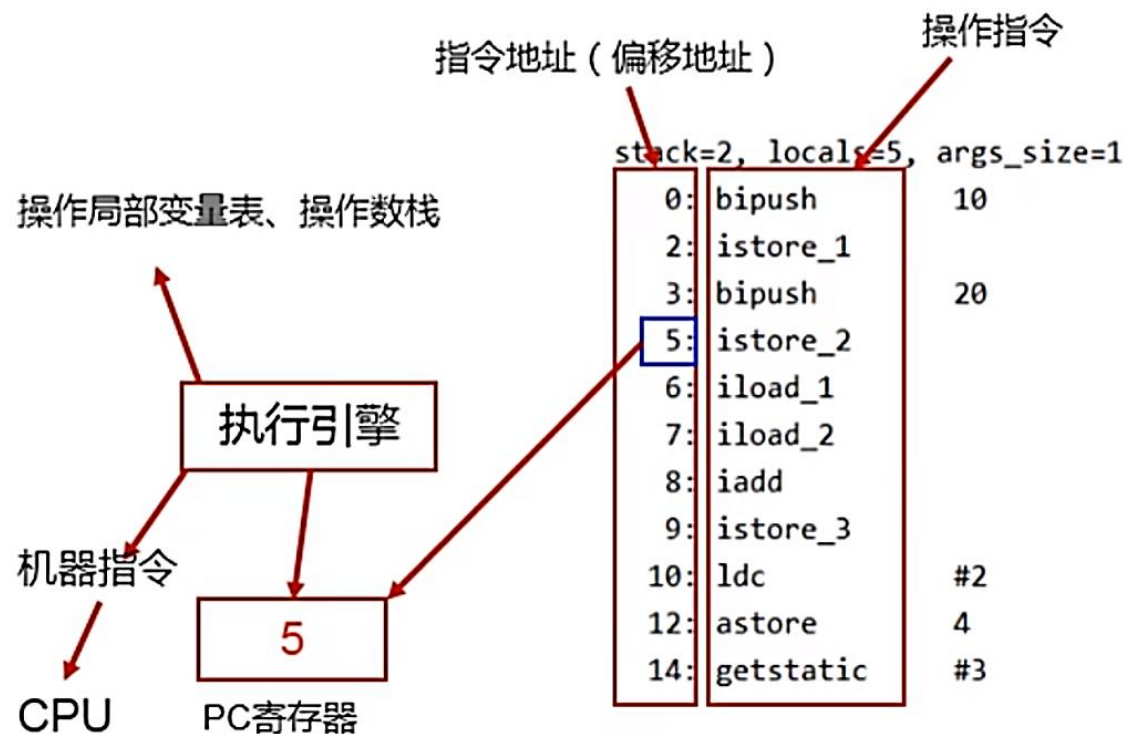
部分专用寄存器包括：

- 堆栈指针指示当前堆栈顶位置。
- 控制寄存器用于配置缓存、内存映射或特权级别。
- 地址寄存器保存当前要访问的内存地址。
- 标志寄存器存放程序运行状态信息，运算结果的符号、进位、零、溢出等。
- **程序计数器（PC，x86系统有时称为IP）**，负责存储下一条指令的地址，确保指令流的连续性。

状态元件



专用寄存器在数据通路中既是控制信息的存储中心，也是指令执行和状态管理的枢纽，它与通用寄存器、ALU、存储器和控制单元紧密协作，共同保证处理器的高效运行。





六、总结

状态元件的核心特征为有记忆性，输出同时依赖当前输入与历史状态；通常由时钟信号同步控制状态变化。

状态元件的主要功能是存储信息与保持状态，是计算机具备“记忆”能力的基础，用于暂存数据、记录指令地址、长期存储数据等。

状态元件	主要功能	核心特点	典型应用
触发器	存储1位二进制数据	时钟边沿触发 建立/保持时间要求	D触发器
寄存器	暂存多位数据 并行读写	多触发器组成 快速数据暂存	通用寄存器 标志寄存器
专用寄存器	承担特定控制 管理功能	功能专一 系统级控制	堆栈指针控制寄存器
通用寄存器	暂存运算数据 等关键信息	计算机核心元件	数据寄存器 变址寄存器
寄存器堆	提供多端口并行读写能力	支持同时读写多个操作数	RISC-V/MIPS 寄存器文件

8.5 存储元件

一、特点和作用

在数据通路中，**时序逻辑元件 = 状态元件 = 存储元件**。
区别在于语境：

- 从**电路特性**看：称为 **时序逻辑元件**
- 从**功能角色**看：称为 **状态元件**
- 从**存储作用**看：称为 **存储元件**

因此，存储元件具有和状态元件一样的特点和作用，即：

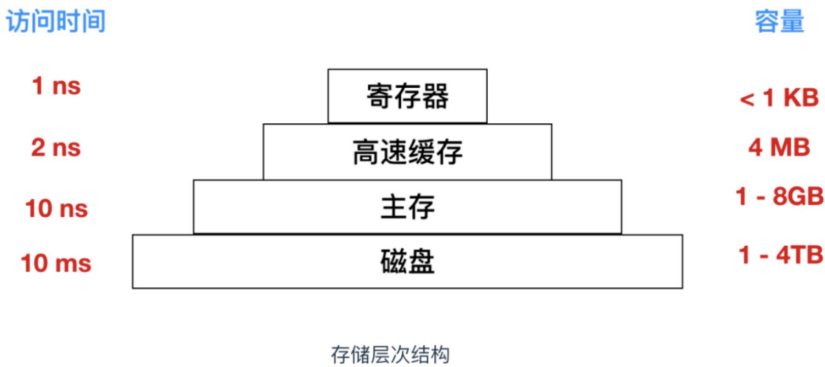
- 具有存储功能，在时钟控制下输入被写到电路中，直到下个时钟到达
- 输入端状态由时钟决定何时被写入，输出端状态随时可以读出
- 与组合逻辑元件一起，在数据通路中共同负责处理和传递数据



二、寄存器（组）

在CPU的数据通路中，存储元件一般指寄存器（组）。而像高速缓存，主存（Memory）等虽然也是广义上的“存储元件”，但通常不被归入数据通路内部，而是被视为外部存储资源。

- 寄存器（组）具有如下特点和作用：
- **高速存储**：寄存器与CPU核心逻辑集成在**同一芯片**上，访问速度比主内存快得多
 - **暂存数据**：寄存器用于临时存储正在执行的指令、参与运算的数据、地址以及CPU自身的状态信息



层级	位置	存储数据类型
寄存器	CPU内部	指令操作数、中间结果
高速缓存	CPU与主存之间	最近使用的主存数据
主存	外部（RAM）	程序和运行时的数据
磁盘	外部（硬盘）	文件、程序、数据库

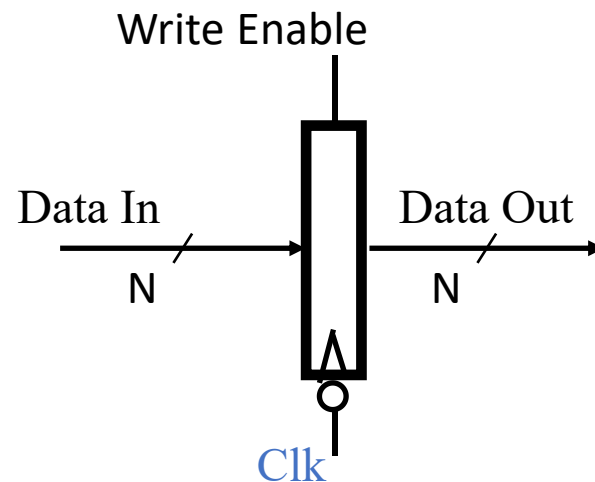
1. 寄存器 (Register)

有一个写使能 (Write Enable-WE) 信号。

0: 时钟边沿到来时, 输出不变

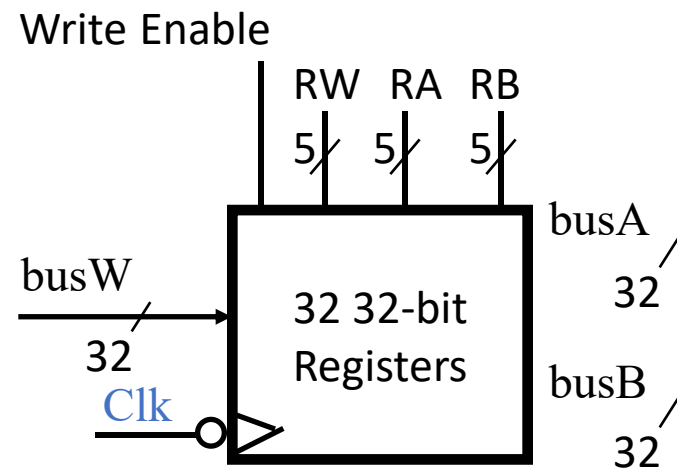
1: 时钟边沿到来时, 输出开始变为输入

若每个时钟边沿都写入, 则不需WE信号。



2. 寄存器组 (Register File)

- **两个读口（组合逻辑操作）**：busA和busB分别由RA和RB给出地址。地址RA或RB有效后，经一个“取数时间 (AccessTime)”，busA和busB有效。
- **一个写口（时序逻辑操作）**：写使能为1的情况下，时钟边沿到来时，busW传来的值开始被写入RW指定的寄存器中。



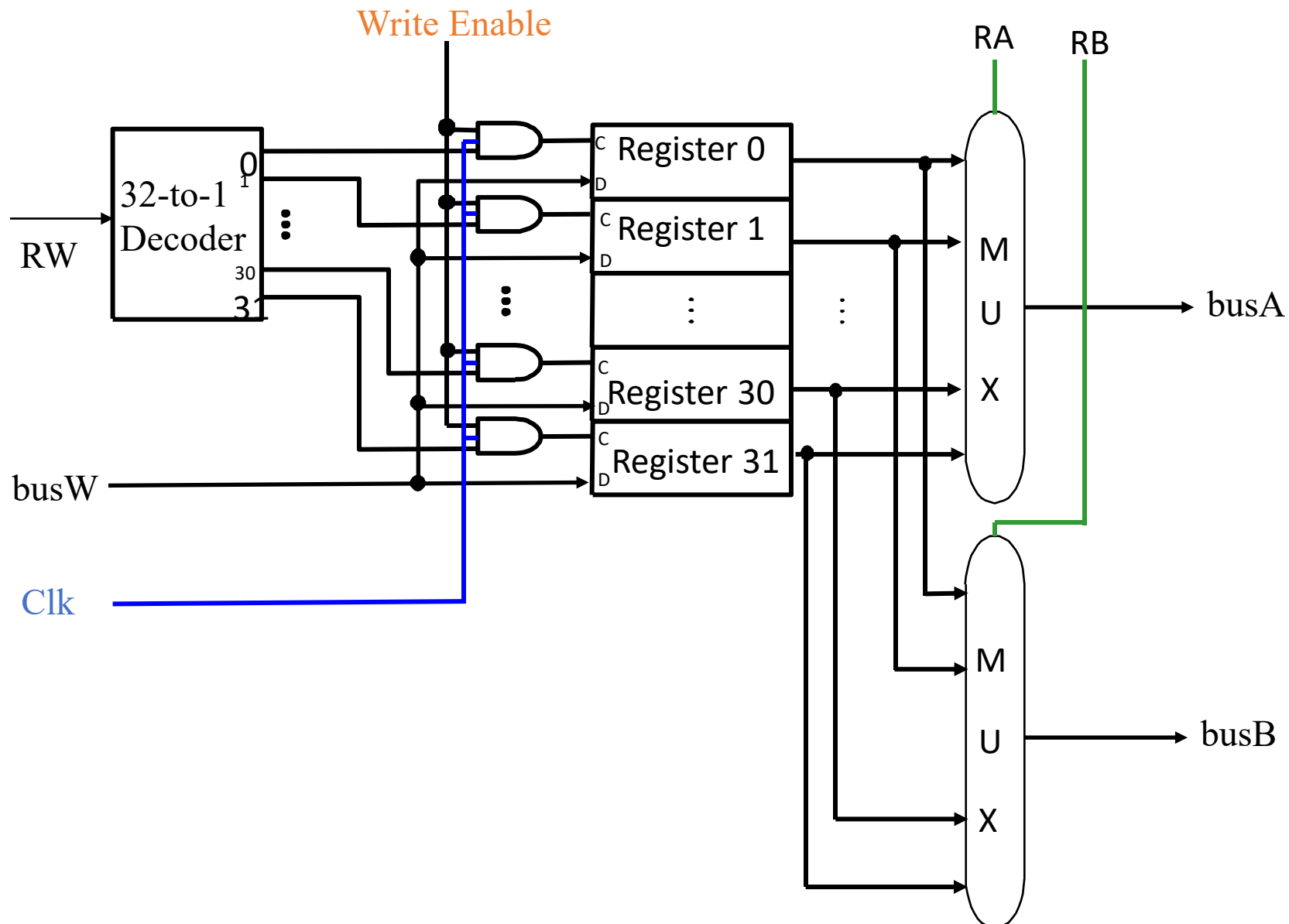
取数时间：从提供地址（RA、RB）到寄存器数据在busA、busB上稳定有效所需的时间。

这段延迟主要由以下因素决定：

- 寄存器组内部译码逻辑的延迟
- 多路选择器（MUX）的传播延迟
- 寄存器输出缓冲的延迟

3. 寄存器组的内部结构

- 每个寄存器由**32个触发器**组成
- 输入数据来自busW, 读出数据分别送busA和busB
- WriteEnable信号控制是否写入新值



三、寄存器的主要类型

- 通用寄存器

作用：用于暂存元算数据、中间结果或操作数

特点：可以由程序员自由使用

- 专用寄存器

特点：这些寄存器由特定用途，一般不能直接使用

例子：程序计数器（PC），指令寄存器（IR），标志寄存器（PSW）……

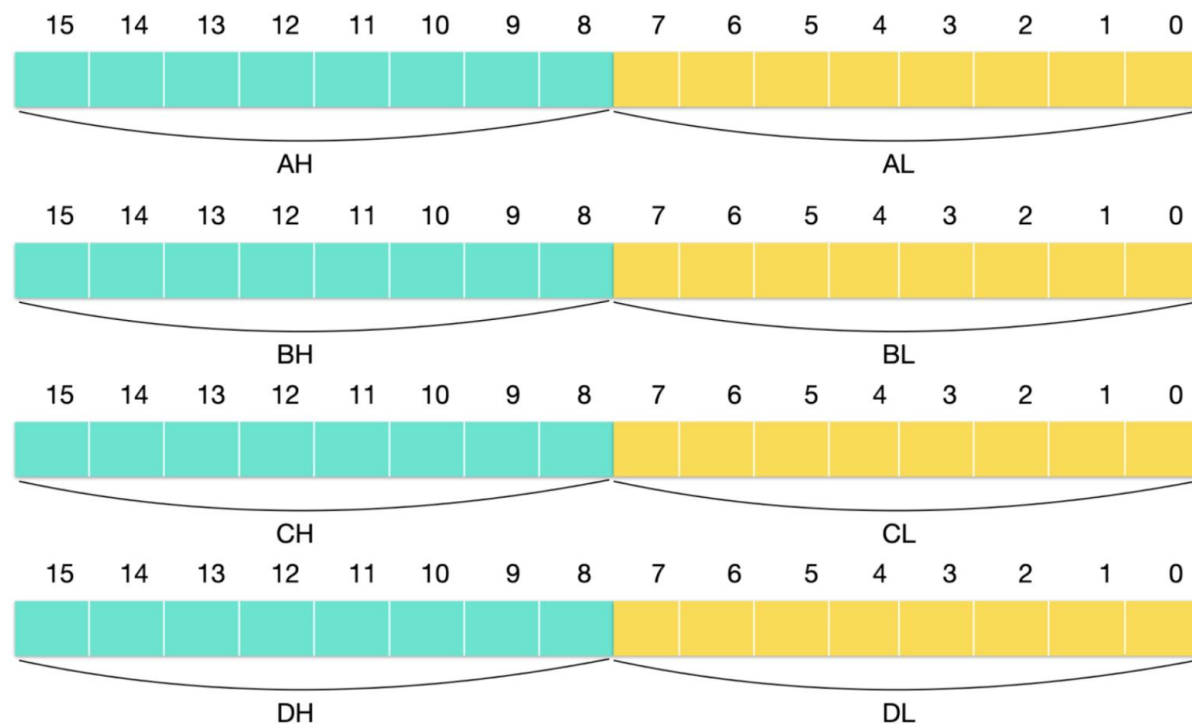
1. 通用寄存器（同P46）

以8086架构的CPU寄存器为例，8086中有四个通用16位寄存器AX、BX、CX、DX。

- AX (Accumulator Register): **累加寄存器**，它主要用于输入/输出和大规模的指令运算
- BX (Base Register): **基址寄存器**，用来存储基础访问地址
- CX (Count Register): **计数寄存器**，CX寄存器在迭代的操作中会循环计数
- DX (data Register): **数据寄存器**，它也用于输入/输出操作。它还与AX寄存器以及DX一起使用，用于涉及大数值的乘法和除法运算

AX、BX、CX、DX 都可以独立作为两个8位寄存器来使用。

它们的存储方式是先存储低位，如果低位满足不了就存储高位，如果低位能够满足，高位用0补全，在其他低位能满足的情况下，其余位也用0补全。



2. 常见专用寄存器

程序计数器 (Program Counter, PC)

- 作用：存放下一条要执行的指令地址。
- 每执行一条指令后，PC自动增加或被跳转指令修改。

指令寄存器 (Instruction Register, IR)

- 作用：存放当前正在执行的指令。
- 控制器通过分析IR的内容生成控制信号。

地址寄存器 (Address Register, AR)

- 作用：保存当前要访问的内存地址。

栈指针寄存器 (Stack Pointer, SP)

- 作用：指向当前栈顶的内存地址，用于函数调用、返回、局部变量等。

标志寄存器

- 作用：存放程序运行状态信息，比如运算结果的符号、进位、零、溢出等。
- 典型标志位：
- ZF (Zero Flag)：结果是否为0
- CF (Carry Flag)：是否有进位或借位
- SF (Sign Flag)：结果符号
- OF (Overflow Flag)：有无溢出
- IF (Interrupt Flag)：中断是否允许

