Lecture 21: CPU - Datapath and Control

中央处理器：数据通路和控制器

单周期数据通路的设计

主 要 内 容

- CPU的功能及其与计算机性能的关系
- 数据通路的位置
- 单周期数据通路的设计
  - 数据通路的功能和实现
    - 操作元件（组合逻辑部件）
    - 状态 / 存储元件（时序逻辑部件）
  - 数据通路的定时
- 选择MIPS指令集的一个子集作为CPU的实现目标
  - 下条指令地址计算与取指令部件
  - R型指令的数据通路
  - 访存指令的数据通路
  - 立即数运算指令的数据通路
  - 分支和跳转指令的数据通路
- 综合所有指令的数据通路

CPU功能及其与计算机性能的关系

- CPU执行指令的过程：
  - 取指令
  - PC+1送PC

  （取指阶段）
  - 指令译码
  - 进行主存地址运算
  - 取操作数
  - 进行算术 / 逻辑运算
  - 存结果
  - 判断和检测“异常”事件

  （译码和执行阶段）
  - 若有异常，则自动切换到异常处理程序
  - 检测是否有“中断”请求，有则转中断处理

  （指令执行过程）
- CPU的实现与计算机性能的关系
  - 计算机性能(程序执行快慢)由三个关键因素决定：
    - 指令数目、CPI、时钟周期
      - 指令数目由编译器和指令集决定
      - 时钟周期和CPI由CPU的实现来决定

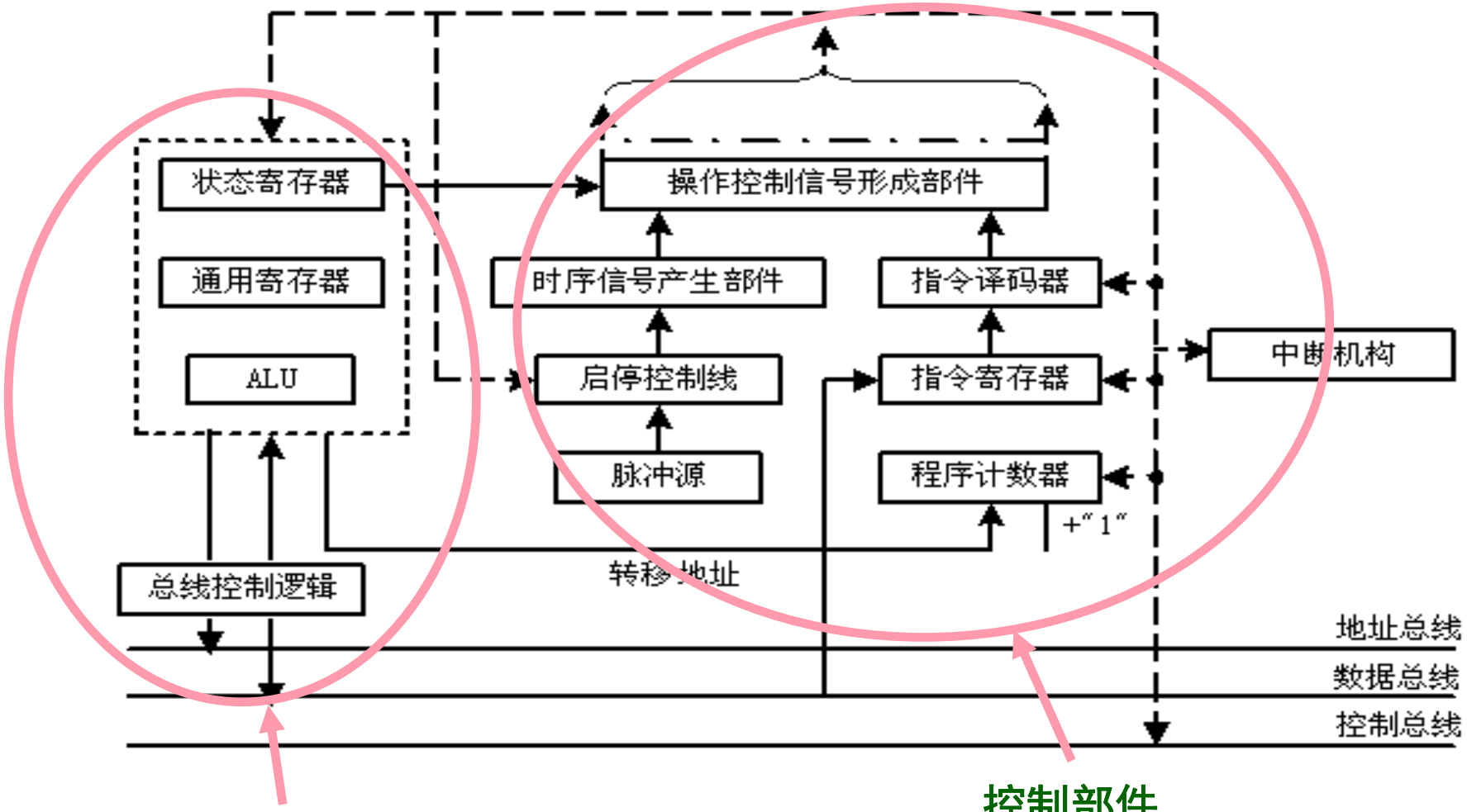
因此，CPU的设计与实现非常重要！它直接影响计算机的性能。

组成指令功能的四种基本操作

- 每条指令的功能总是由以下四种基本操作来实现：
  （1）读取某一主存单元的内容，并将其装入某个寄存器；
  （2）把一个数据从某个寄存器存入给定的主存单元中；
  （3）把一个数据从某个寄存器送到另一个寄存器或者ALU；
  （4 进行某种算术运算或逻辑运算，将结果送入某个寄存器。
- 操作功能可形式化描述
  - 描述语言称为寄存器传送语言RTL (Register Transfer Language)
  - 本章所用的RTL规定如下：
    （1）用R[r]表示寄存器r的内容；
    （2）用M[addr]表示读取主存单元addr的内容；
    （3）传送方向用“←”表示，传送源在右，传送目的在左；
    （4）程序计数器PC直接用PC表示其内容；
    （5）用OP[data]表示对数据data进行OP操作。

CPU基本组成原理图

指令寄存器----IR
程序计数器---PC



执行部件

控制部件

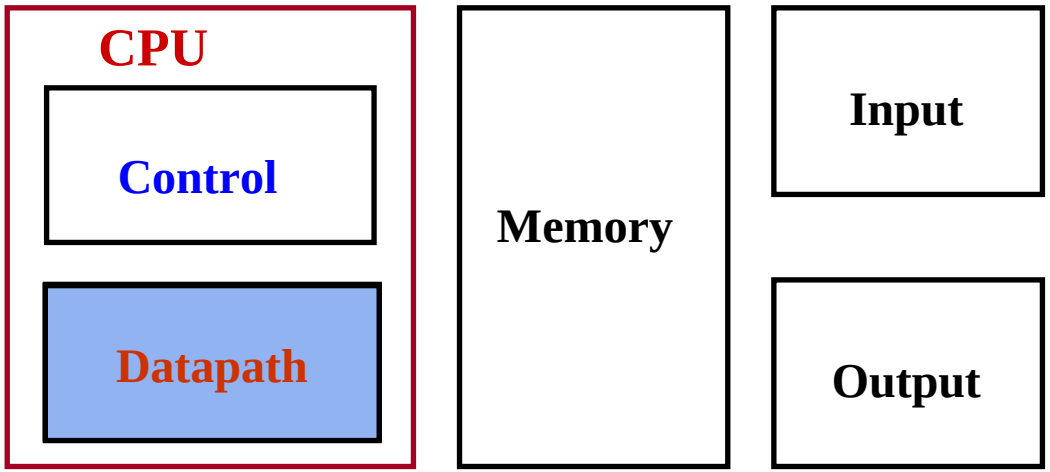
CPU 由 执行部件 和 控制部件组成

CPU 包含 数据通路 和 控制器

控制器 由 指令译码器 和 控制信号形成部件 组成

数据通路的位置

- 计算机的五大组成部分：



- 什么是数据通路（DataPath）？
  - 指令执行过程中，数据所经过的路径，包括路径中的部件。它是指令的执行部件。
- 控制器（Control）的功能是什么？
  - 对指令进行译码，生成指令对应的控制信号，控制数据通路的动作。能对执行部件发出控制信号，是指令的控制部件。

数据通路的基本结构

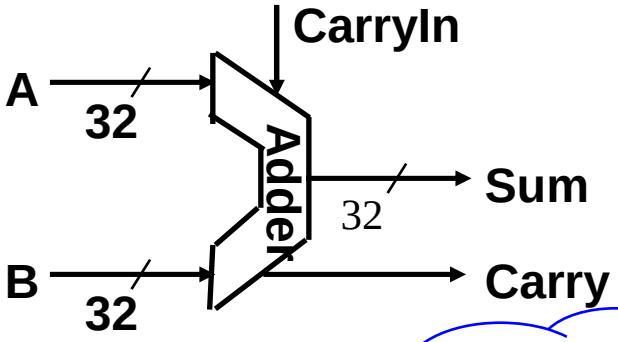
- 数据通路由两类部件组成
  - 组合逻辑元件（也称操作元件）
  - 存储元件（也称状态元件）
- 元件间的连接方式
  - 总线连接方式
  - 分散连接方式
- 数据通路如何构成？
  - 由“操作元件”和“存储元件”通过总线方式或分散方式连接而成
- 数据通路的功能是什么？
  - 进行数据存储、处理、传送

因此，数据通路是由操作元件和存储元件通过总线方式或分散方式连接而成的进行数据存储、处理、传送的路径。

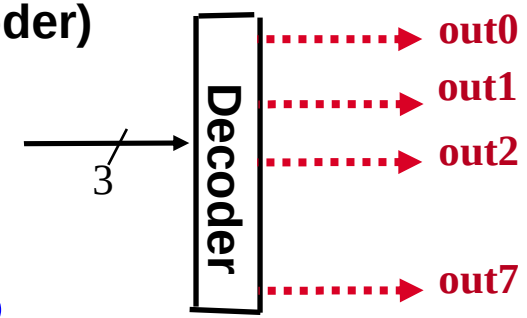
操作元件：组合逻辑电路

控制信号

- 加法器 (Adder)



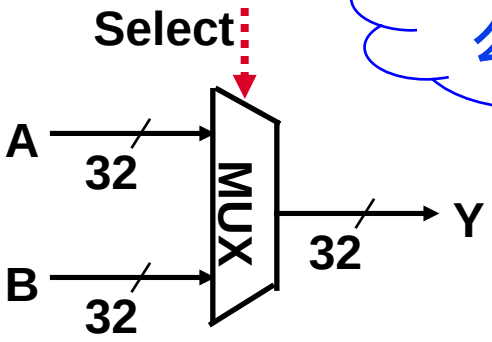
- 译码器 (Decoder)



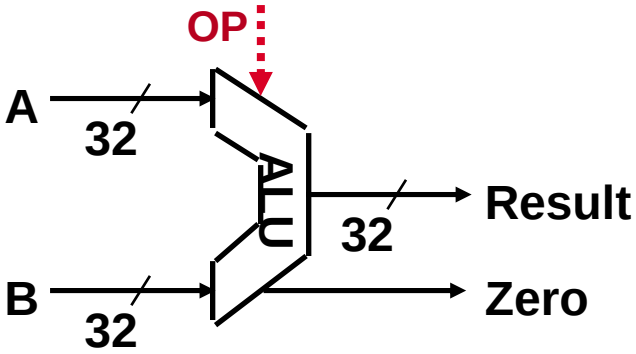
加法器需要什么控制信号?

何时要用到adder, ALU, MUX or Decoder?

多路选择器 (MUX)



算逻部件 (ALU)



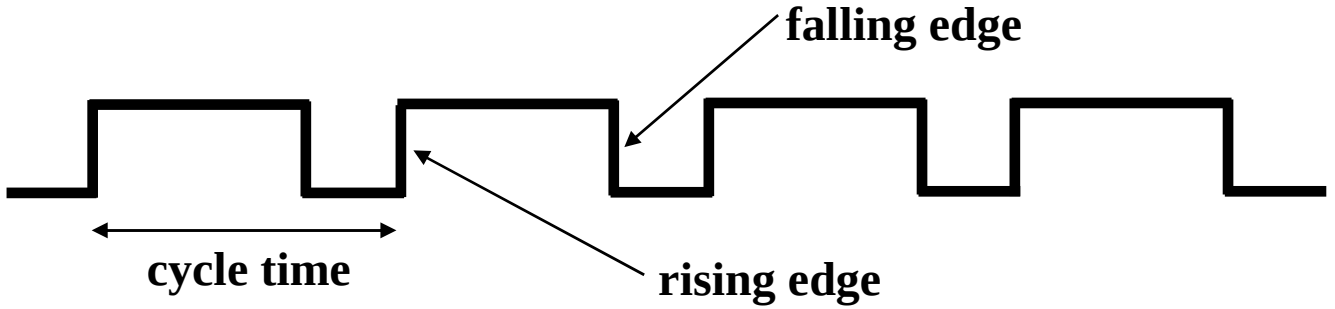
组合逻辑元件的特点：

其输出只取决于当前的输入。即：输入一样，其输出也一样

定时：所有输入到达后，经过一定的逻辑门延时，输出端改变，并保持到下次改变，不需要时钟信号来定时

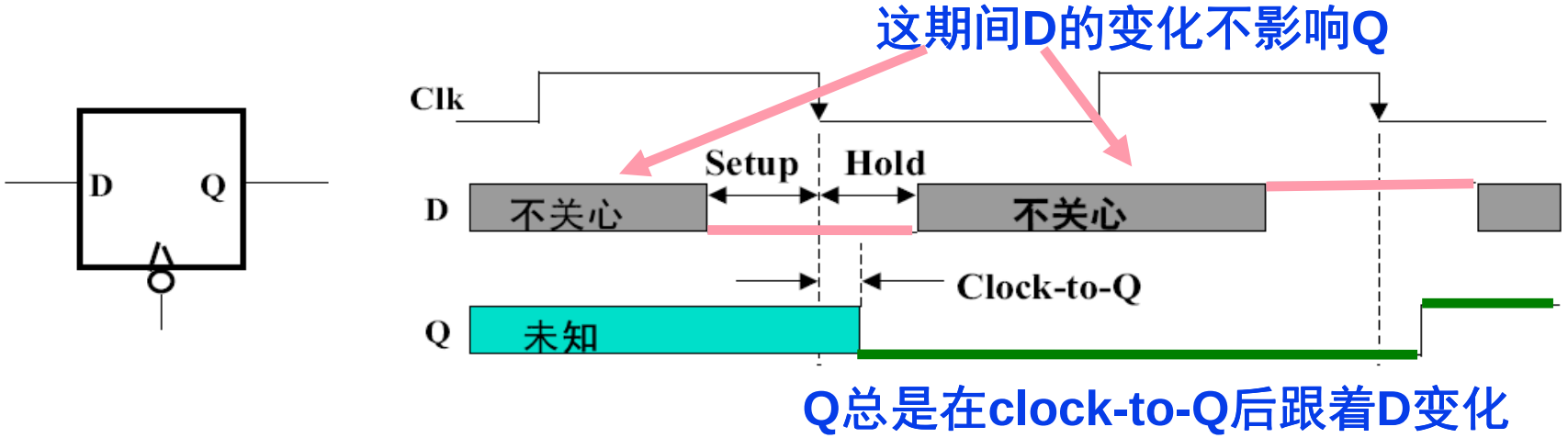
状态元件：时序逻辑电路

- 状态（存储）元件的特点：
  - 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
- 定时方式：规定信号何时写入状态元件或何时从状态元件读出
  - 边沿触发（edge-triggered）方式：
    - 状态单元中的值只在时钟边沿改变。每个时钟周期改变一次。
      - 上升沿（rising edge） 触发：在时钟正跳变时进行读/写。
      - 下降沿（falling edge）触发：在时钟负跳变时进行读/写。



- 最简单的状态单元（回顾：数字逻辑电路课程内容）：
  - D触发器：一个时钟输入、一个状态输入、一个状态输出

存储元件中何时状态被改变？



- 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: (Latch Prop - 锁存延迟)**
  - 在触发时钟边沿，输出并不能立即变化

切记：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”时才被写入到单元中，此时的输出才反映新的状态值

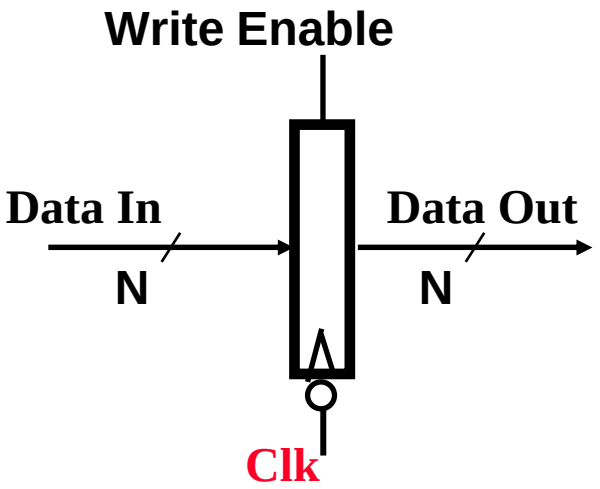
数据通路中的状态元件有两种：寄存器(组) + 存储器

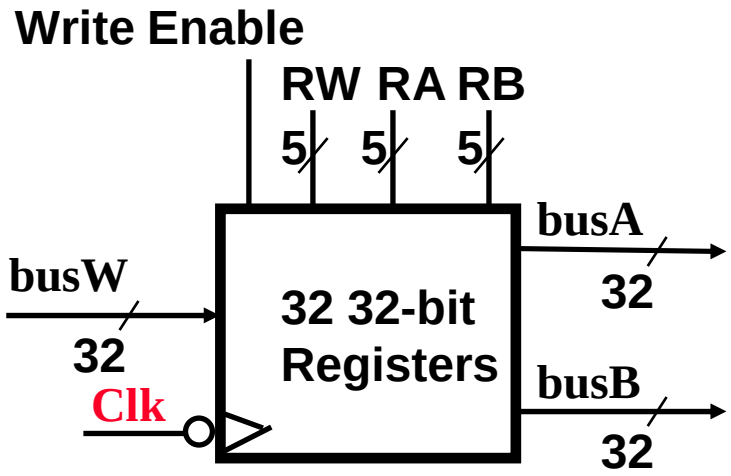
存储元件：寄存器和寄存器组

- 寄存器（Register）
  - 有一个写使能（Write Enable-WE）信号

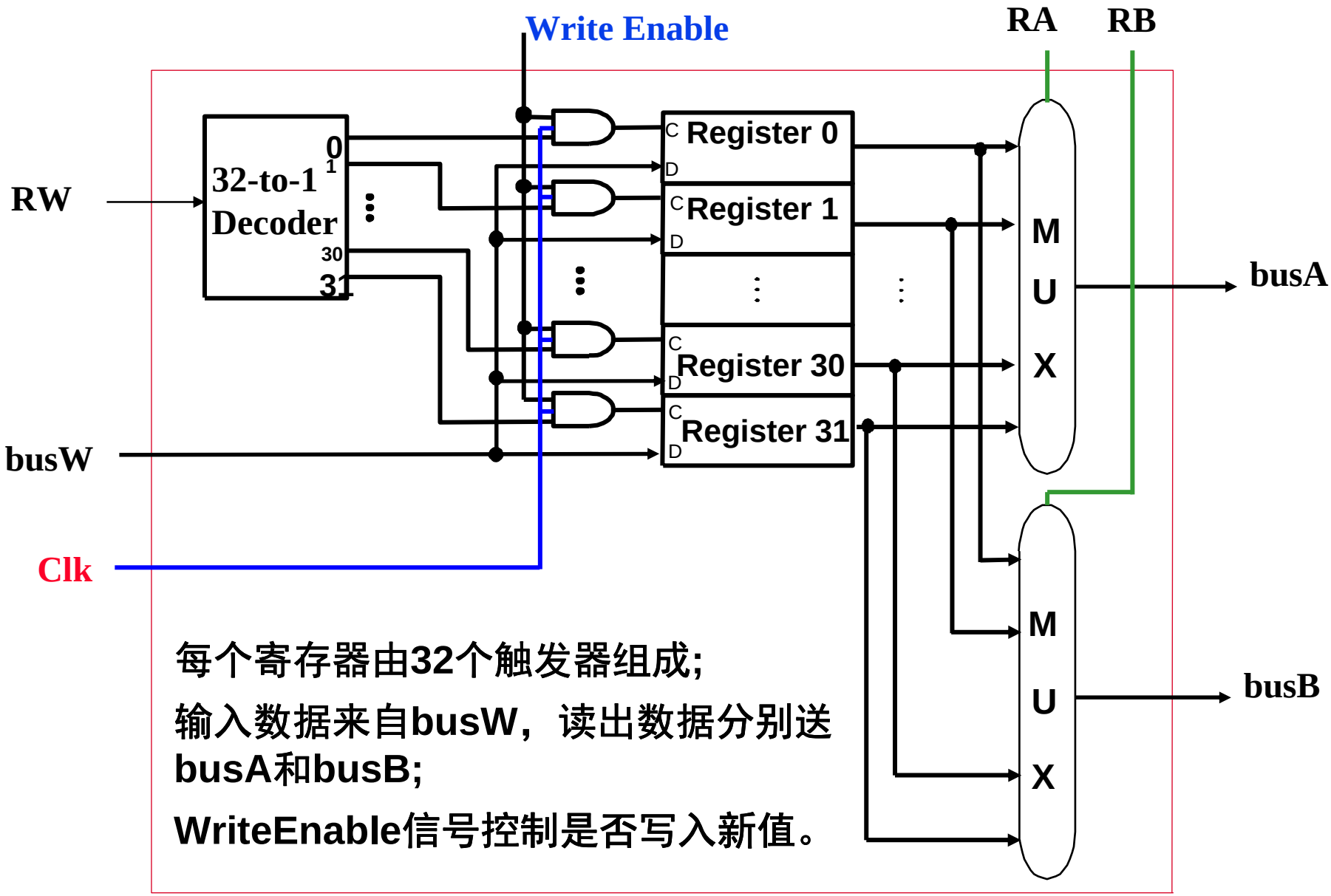
    0: 时钟边沿到来时，输出不变

    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
- 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。





寄存器组的内部结构



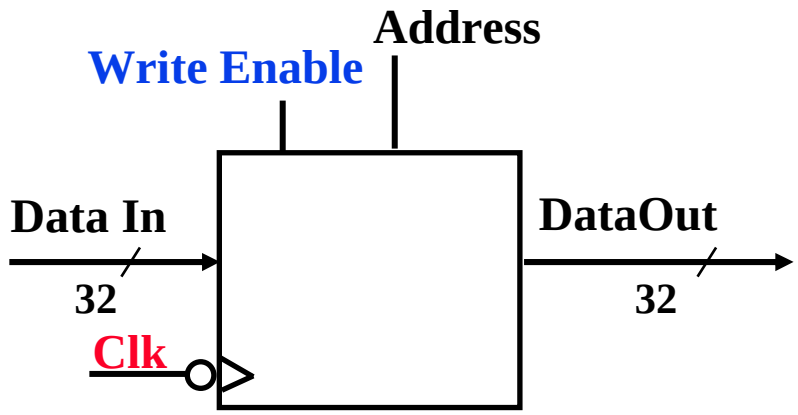
每个寄存器由32个触发器组成;

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。

存储元件：理想存储器

- 理想存储器（idealized memory）
  - Data Out：32位读出数据
  - Data In： 32位写入数据
  - Address：读写公用一个32位地址



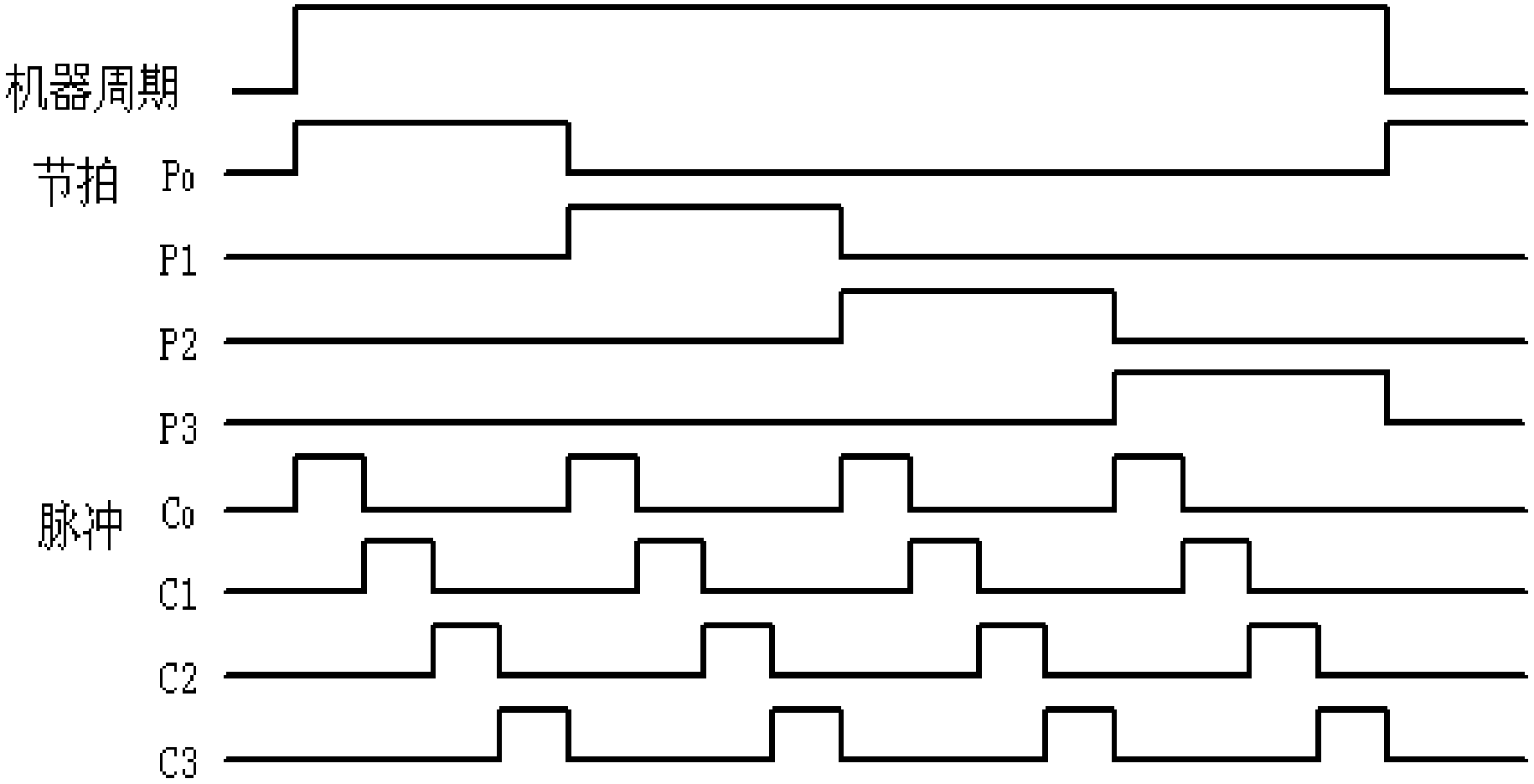
- 读操作：地址Address有效后，经一个“取数时间AccessTime”，Data Out上数据有效。
- 写操作 ：写使能为1的情况下，时钟Clk边沿到来时，Data In传来的值开始被写入Address指定的存储单元中。

为简化数据通路操作说明，把存储器简化为带时钟信号Clk的理想模型。

数据通路与时序控制

- 同步系统(Synchronous system)
  - 所有动作有专门时序信号来定时
  - 由时序信号规定何时发出什么动作
    例如，指令执行过程每一步都有控制信号控制，由定时信号确定控制信号何时发出、作用时间多长
- 什么是时序信号？
  - 同步系统用于同步控制的定时信号，如时钟信号
- 什么叫指令周期？
  - 取并执行一条指令的时间
  - 每条指令的指令周期肯定一样吗？
- 早期计算机的三级时序系统
  - 机器周期 - 节拍 - 脉冲
  - 指令周期可分为取指令、读操作数、执行并写结果等多个基本工作周期，称为机器周期。
  - 机器周期有取指令、存储器读、存储器写、中断响应等不同类型

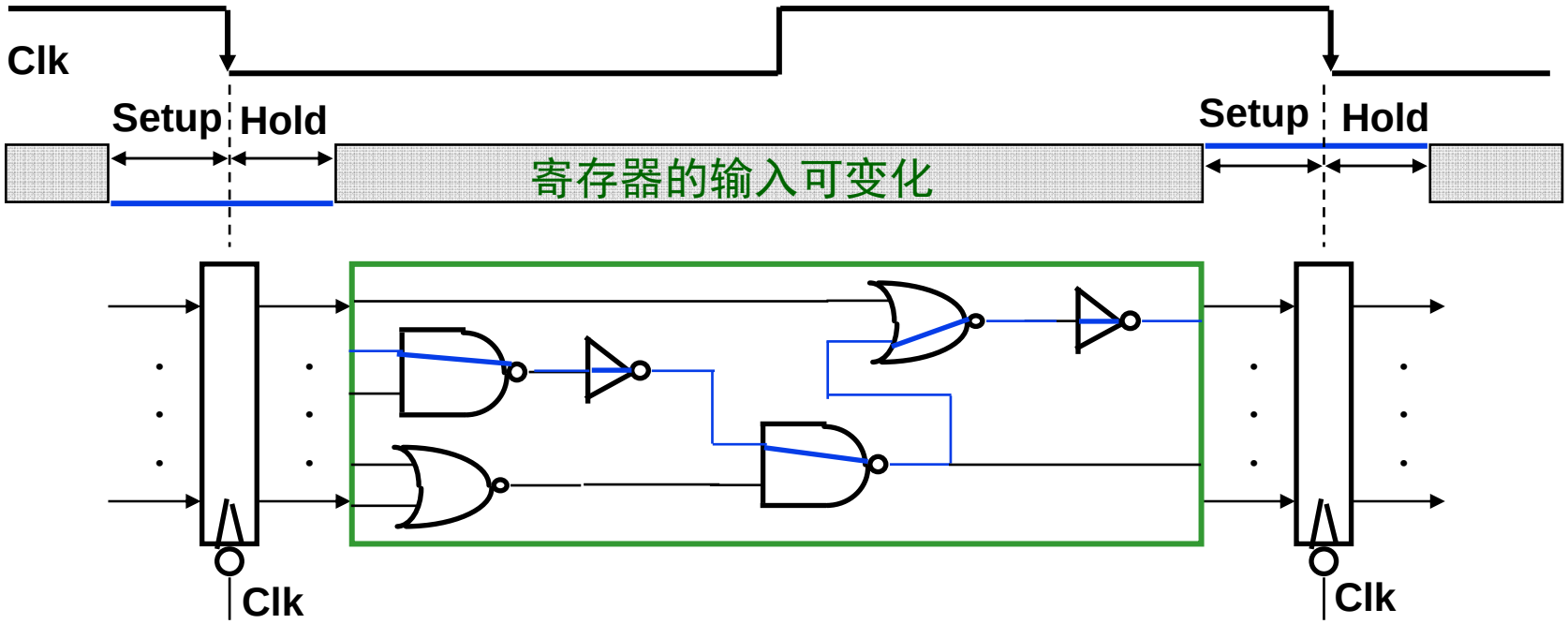
数据通路与时序控制



现代计算机已不再采用三级时序系统，机器周期的概念已逐渐消失。

整个数据通路中的定时信号就是时钟，一个时钟周期就是一个节拍。

现代计算机的时钟周期



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

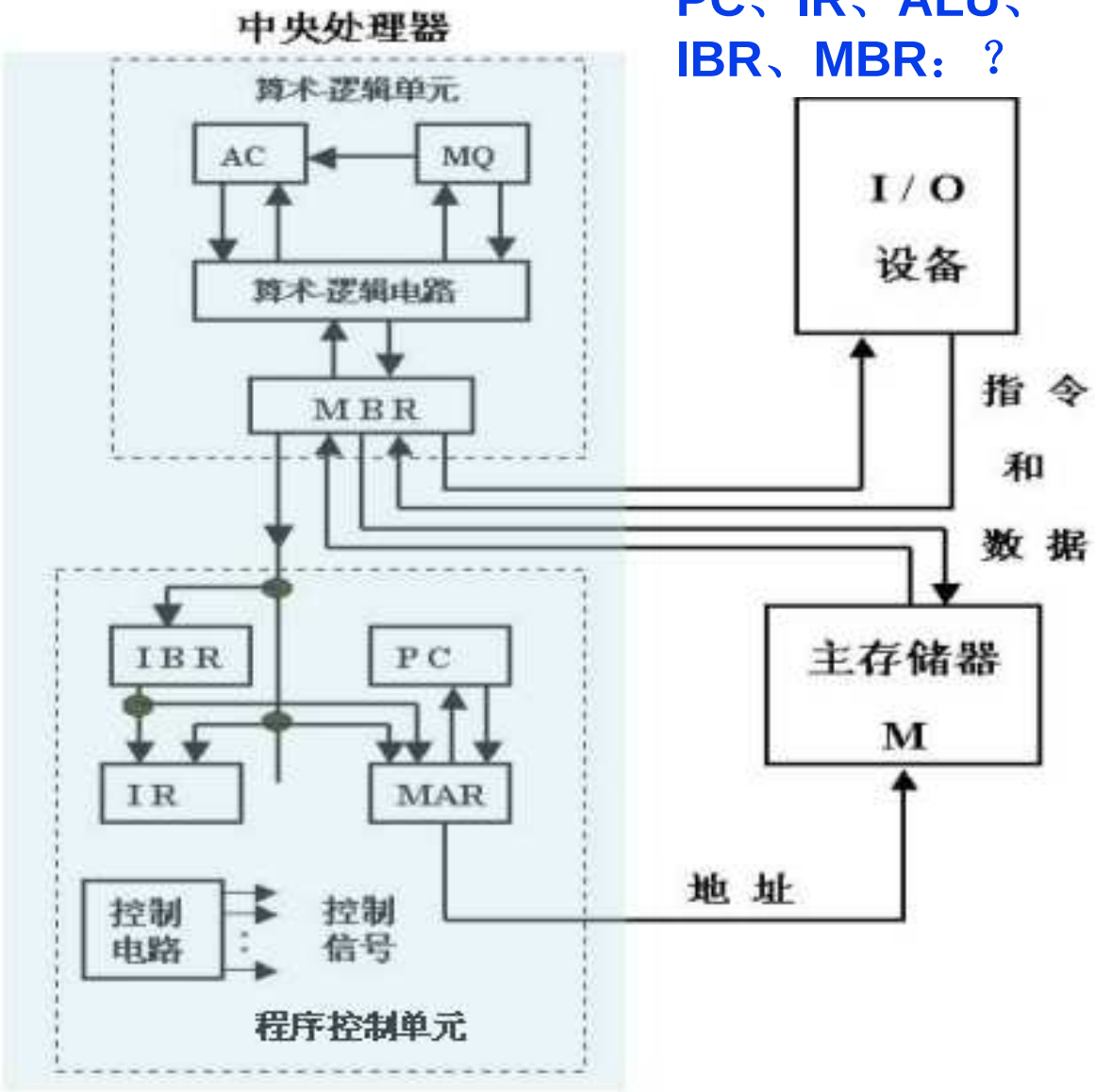
只有状态元件能存储信息，所有操作元件都须从状态单元接收输入，并将输出写入状态单元中。其输入为前一时钟生成的数据，输出为当前时钟所用的数据

- 假定采用下降沿触发（负跳变）方式（也可以是上升沿方式）
  - 所有状态单元在下降沿写入信息，经过Latch Prop (clk-to-Q) 后输出有效
  - Cycle Time = Latch Prop + Longest Delay Path + Setup + Clock Skew(最大偏移)
- 约束条件：(Latch Prop + Shortest Delay Path - Clock Skew) > Hold Time

早期累加器型指令系统数据通路

AC：累加器
MQ：乘商寄存器
PC、IR、ALU、
IBR、MBR：？

- 最简单的数据通路结构
- 取指令数据路径为：
  PC→MAR,
  Read M,
  M→MBR→IBR→IR
- 取操作数、运算、送结果的数据路径为：
  操作数地址→MAR,
  Read M,
  M→MBR→ALU输入端,
  AC→ALU输入端,
  ALU操作,
  ALU结果→MBR,
  Write M

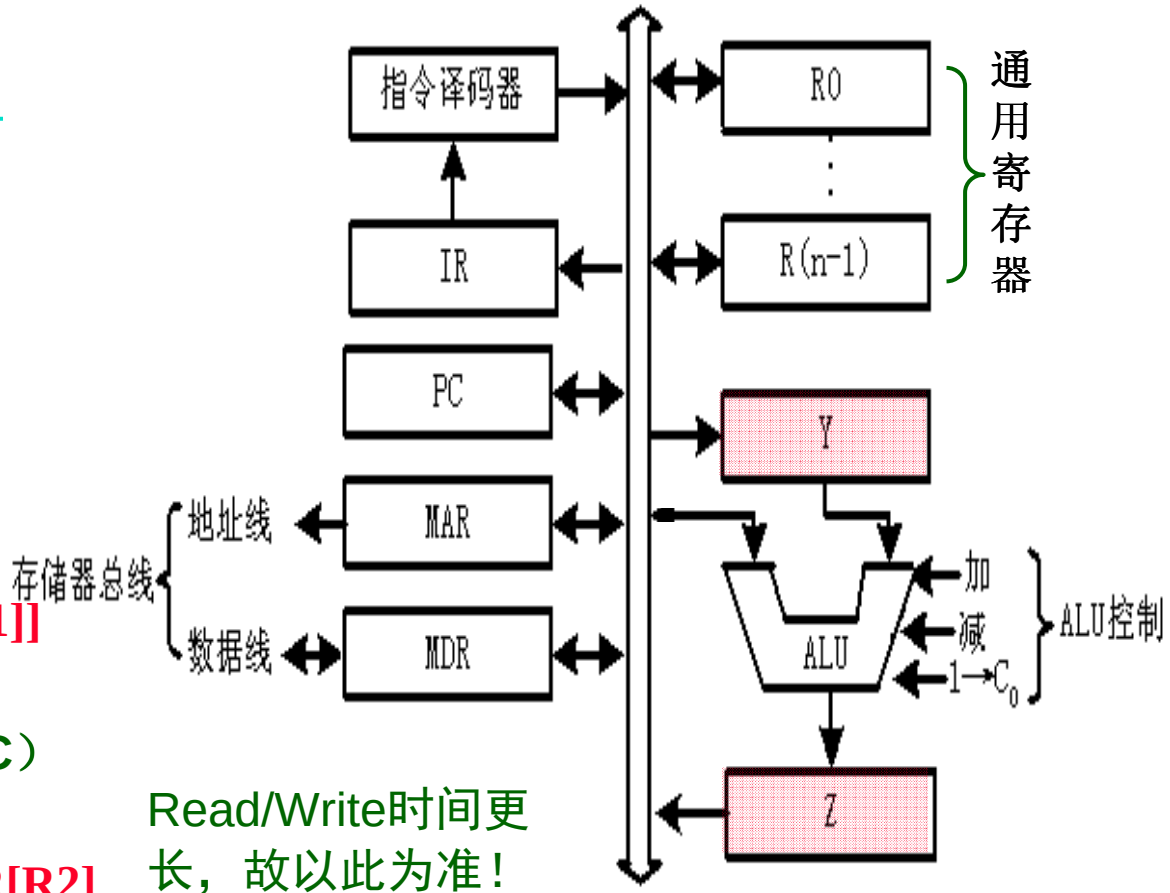


IAS计算机（冯.诺依曼等设计）是现代计算机的原型

单总线数据通路

四种基本操作的时序

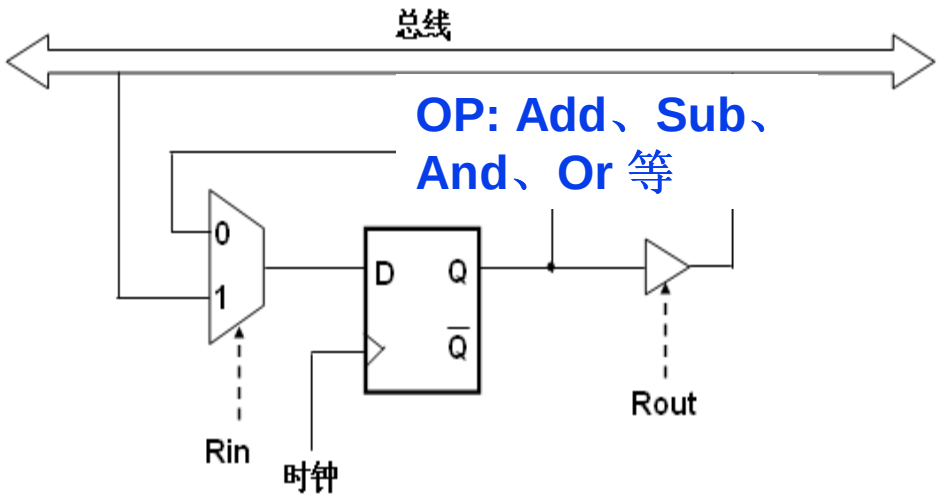
- 在通用寄存器之间传送数据
  R0out，Yin
- 完成算术、逻辑运算
  R1out，Yin
  R2out，Add，Zin
  Zout，R3in
- 从主存取字 **R[R2]←M[R[R1]]**
  R1out，MARin
  Read, WMFC （等待MFC）
  MDRout，R2in
- 写字到主存 **M[R[R1]] ← R[R2]**
  R1out，MARin
  R2out，MDRin，
  Write, WMFC



Read/Write时间更长，故以此为准！

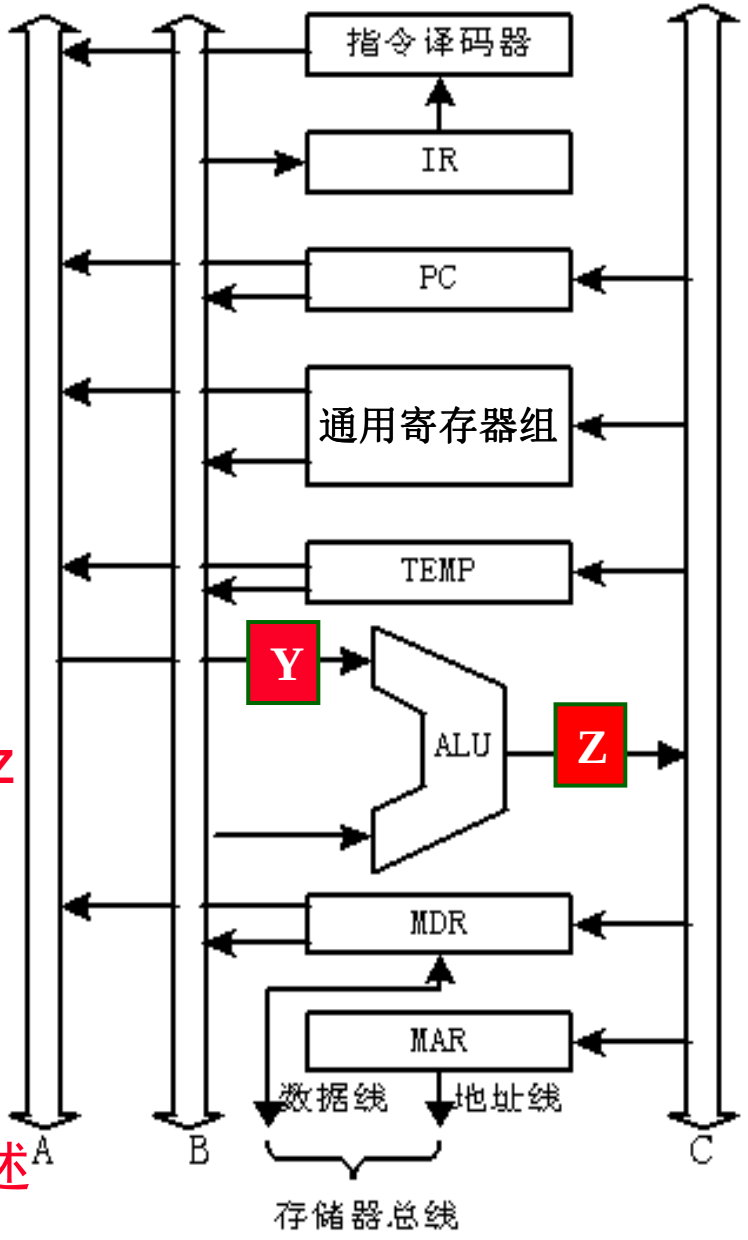
CPU访存有两种通信方式

早期：直接访问MM，“异步”方式，用MFC应答信号；现在：先Cache后MM，“同步”方式，无需应答信号。



三总线数据通路

- 单总线中一个时钟内只允许传一个数据，因而指令执行效率很低
- 可采用多总线方式，同时在多个总线上传送不同数据，提高效率
- 例如：三总线数据通路
  - 总线A、B分别传送两个源操作数，总线C传送结果
  - 单总线中的暂存器Y和Z在此可取消，Why？
    三个总线各自传不同数据，不会发生冲突，故无需Y和Z
  - 采用双口通用寄存器组
  - 如何实现： R[R3] ←R[R1] op R[R2]
    R1outA，R2outB，op，R3inC
    只要一个时钟周期（节拍）即可！



目前，计算机大都采用流水线方式执行指令，上述单总线或三总线数据通路很难实现指令流水执行。

以下以**MIPS**指令系统为例介绍非总线式**CPU**的设计。