回顾第18次课【指令】内容

- add x5，x6，x7
- slti x5，x6，30
- beq x5，x6，40
- jal x5，40
- lw x5，100（x6）
- sw x5，100（x6）

第8章 中央处理器（一）

第一讲 中央处理器概述

第二讲 单周期数据通路的设计

第三讲 单周期控制器的设计

第四讲 多周期处理器的设计

第五讲 流水线处理器设计

第六讲 流水线冒险及其处理

第七讲 高级流水线技术

第一讲 中央处理器概述

主 要 内 容

- 指令执行过程
- CPU的基本组成
  - 操作元件（组合逻辑元件）
  - 状态 / 存储元件（时序逻辑元件）
- 数据通路与时序控制
- 计算机性能与CPU时间

CPU执行程序和指令的过程

- CPU执行一条指令的过程
  - 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
  - 以上每步都需检测异常

  } 译码和执行阶段

  - 若有异常，则自动切换到异常处理程序
  - 硬件检测是否有“中断”请求，有则转中断处理
- RTL

问题：

“取指令”一定在最开始做吗?

“PC+1”一定在译码前做吗?

“译码”须在指令执行前做吗?

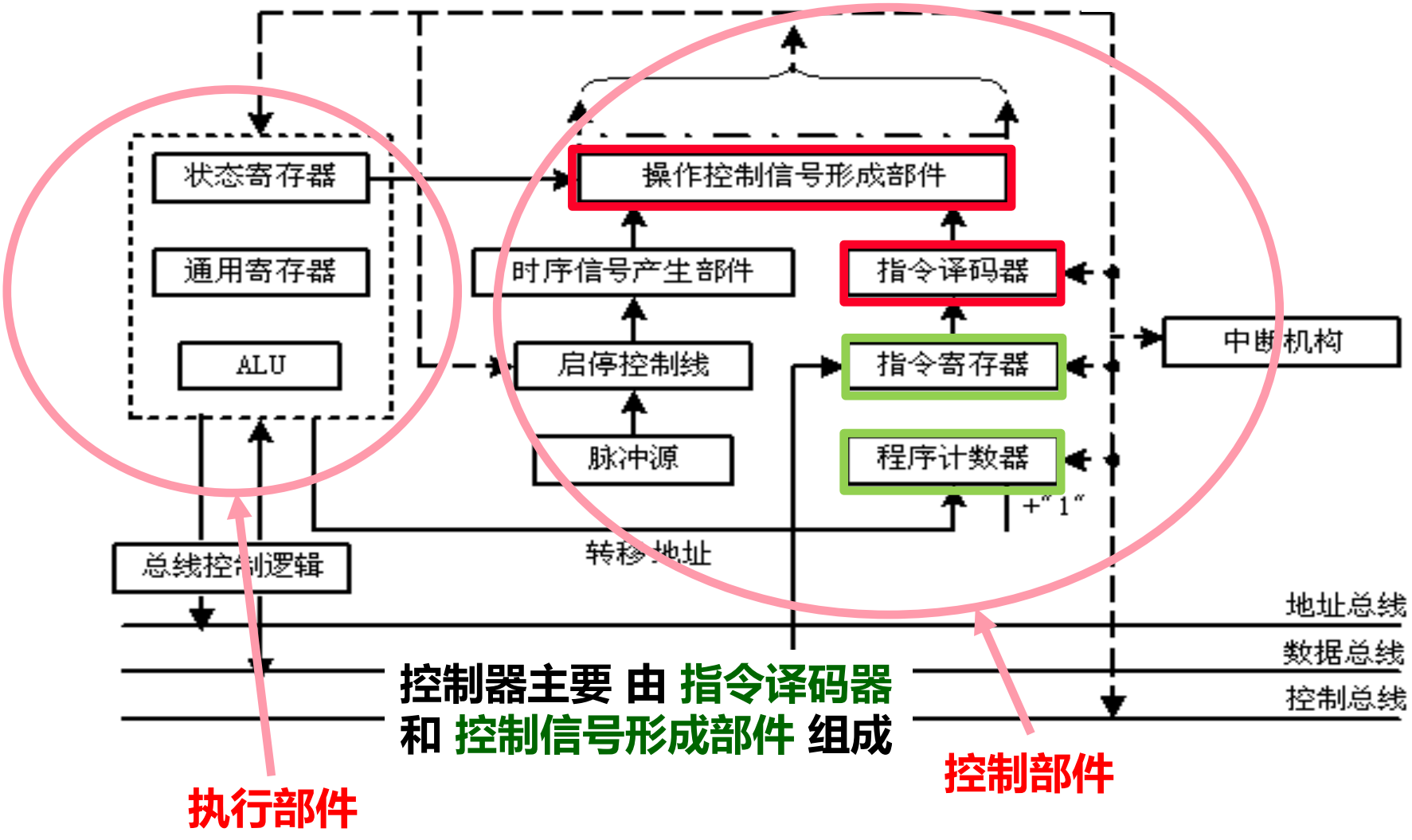
异常和中断的差别是什么?

异常是在CPU内部发生的

中断是由外部事件引起的

CPU基本组成原理图

指令寄存器----IR
程序计数器---PC



控制器主要 由 指令译码器
和 控制信号形成部件 组成

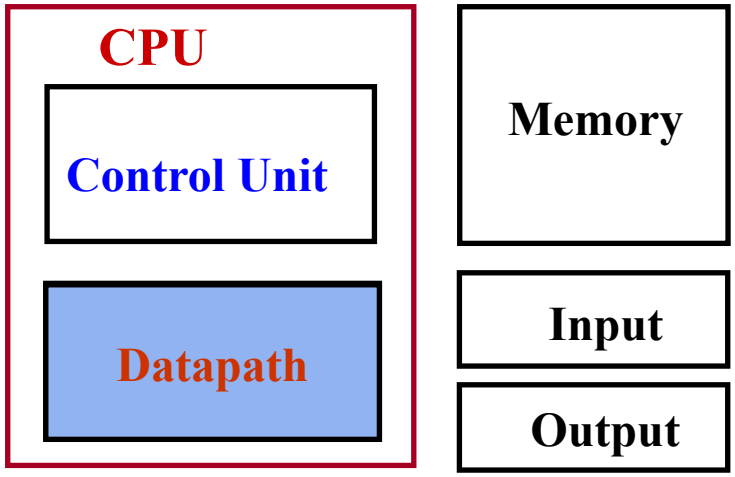
执行部件

控制部件

CPU 由 执行部件 和 控制部件组成

CPU的基本结构

- 计算机的五大组成部分：



- 执行部件（数据通路）
  - 操作元件（如ALU）
  - 存储（状态）元件
    - 寄存器
- 控制部件（控制器）
  - 译码部件
  - 控制信号生成部件
  - 存储（状态）元件
    - 寄存器

- 数据通路（DataPath）

  ——指令的执行部件

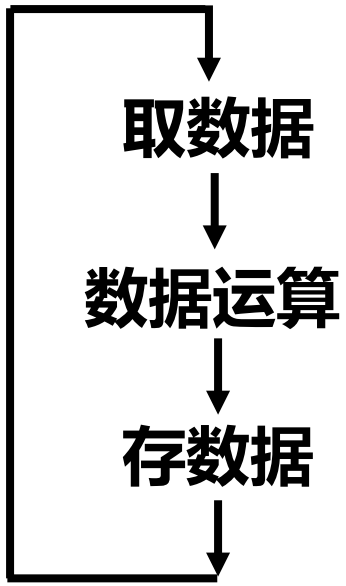
- 控制器（Control Unit）

  ——指令的控制部件

- 控制器负责对执行部件（数据通路）发出控制信号。

数据通路的基本结构

- 数据通路由两类元件组成
  - 组合逻辑元件（也称操作元件）
  - 时序逻辑元件（也称状态元件，存储元件）
- 元件间的连接方式
  - 总线连接方式
  - 分散连接方式
- 数据通路的具体工作是什么?
  - 进行数据存储、处理、传送



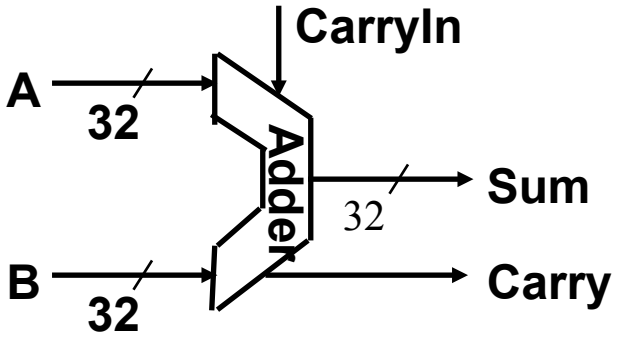
数据通路是由操作元件和存储元件通过总线方式或分散方式连接而成的进行数据存储、处理、传送的路径。

（回忆之前的“运算部件”）

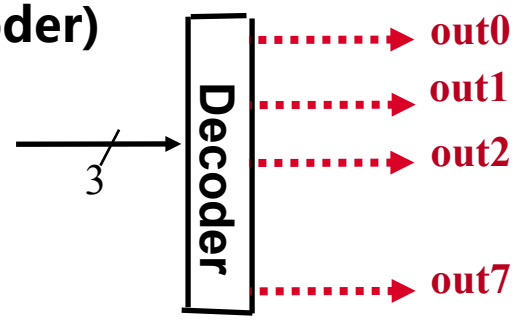
操作元件：组合逻辑电路

控制信号

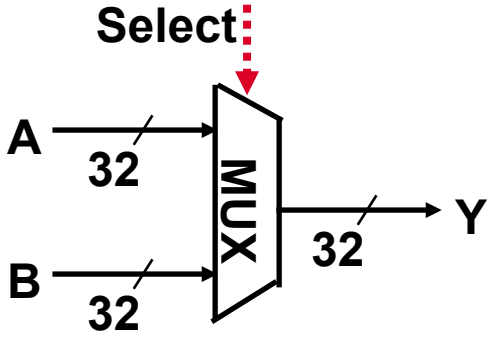
- 加法器 (Adder)



- 译码器 (Decoder)

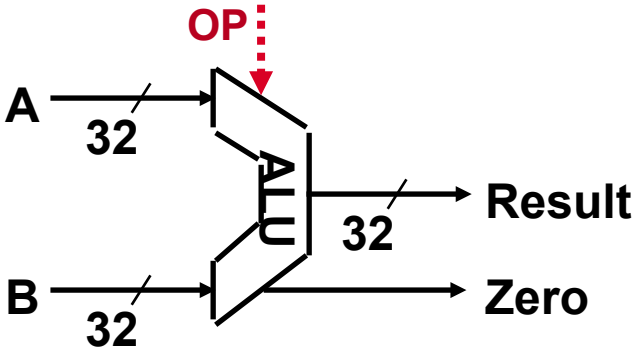


多路选择器 (MUX)



何时要用到adder, ALU, MUX or Decoder?

算逻部件 (ALU)



组合逻辑元件的特点：

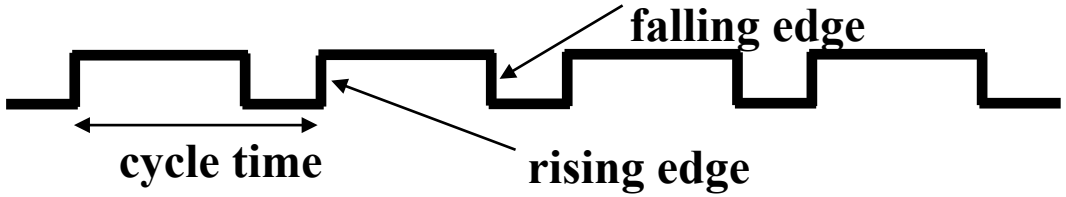
其输出只取决于当前的输入。即：若输入一样，则其输出也一样

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

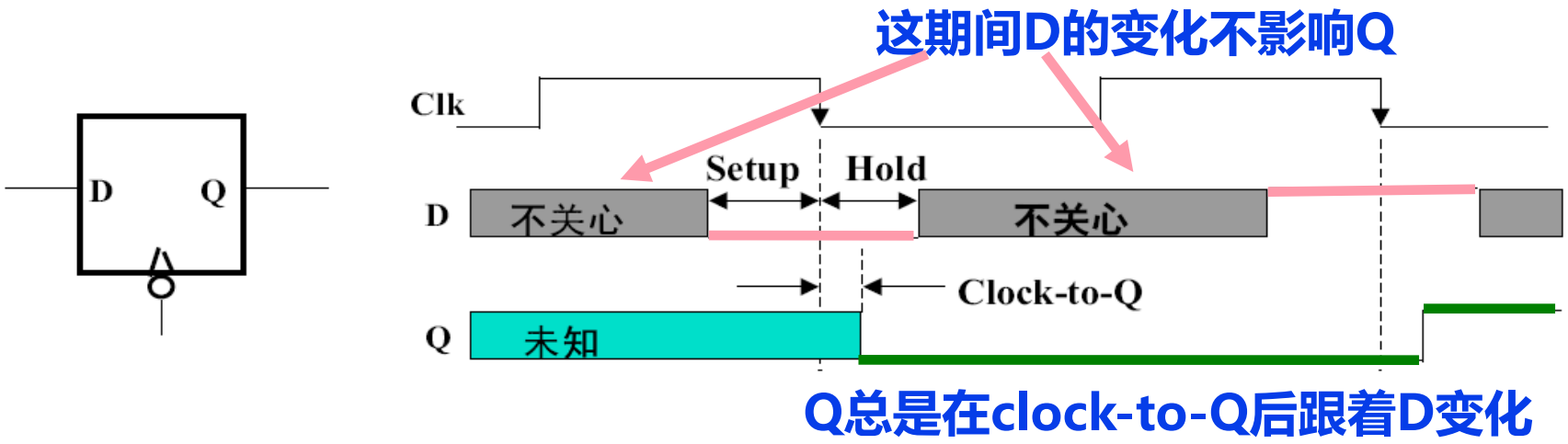
- 状态（存储）元件的特点：
  - 具有存储功能，在时钟控制下输入被写到电路中，直到下个时钟到达
  - 输入端状态由时钟决定何时被写入，输出端状态随时可以读出

- 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：
  - 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

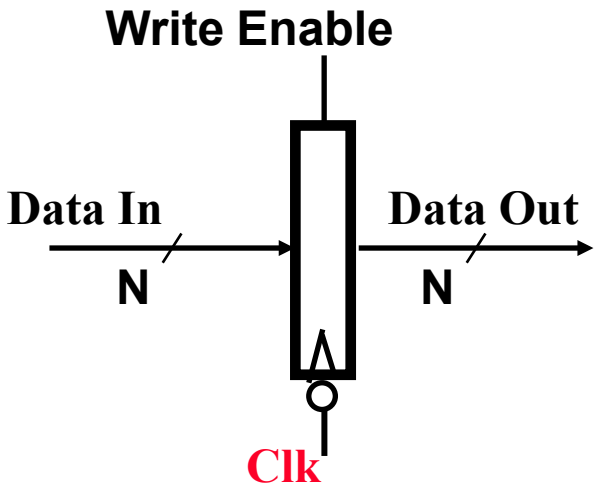
数据通路中的状态元件有两种：寄存器(组) + 理想存储器

存储元件：寄存器和寄存器组

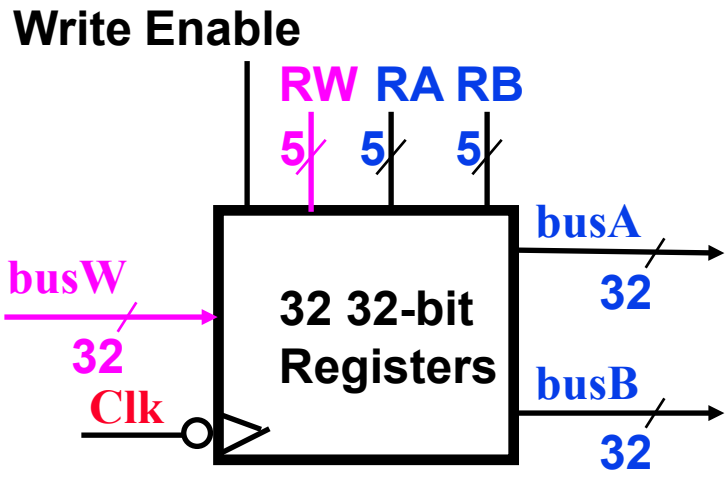
- 寄存器（Register）
  - 有一个写使能（Write Enable-WE）信号

    0: 时钟边沿到来时，输出不变

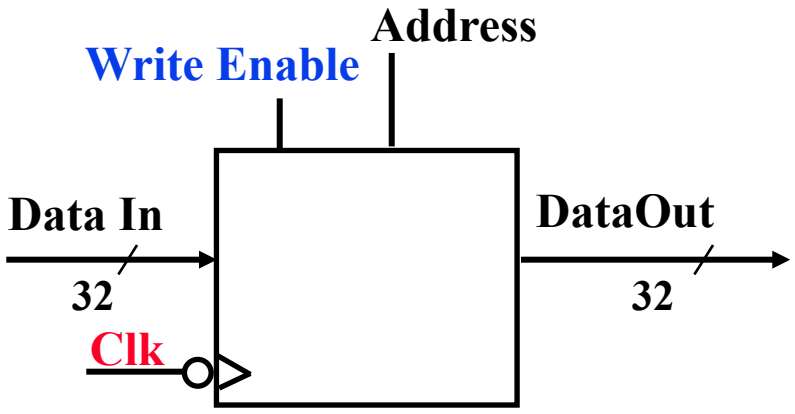
    1: 时钟边沿到来时，输出开始变为输入
  - 一定需要写使能信号吗？



- 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。



额外假设的状态元件: 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（组合逻辑操作）：地址Address有效后，经一个“取数时间AccessTime”，Data Out上数据有效。
- 写操作（时序逻辑操作）：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。并非真实存在于CPU中！

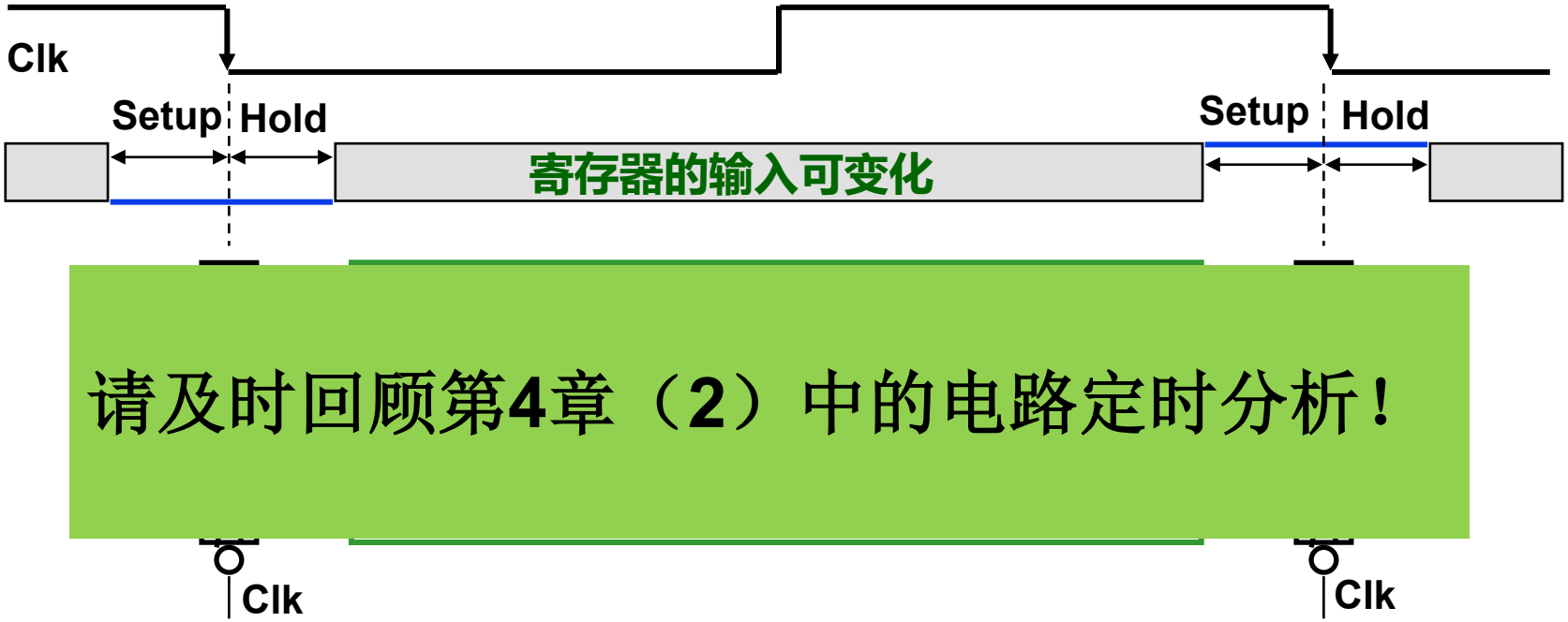
数据通路与时序控制

- 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，由时序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?
  - 早期计算机的三级时序系统（自行了解）
  - 用于进行同步控制的定时信号——时钟信号
- 什么叫指令周期?
  - CPU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？ ——不一定

- 时钟周期又是什么呢？

数据通路与时序控制　　现代计算机的时钟周期

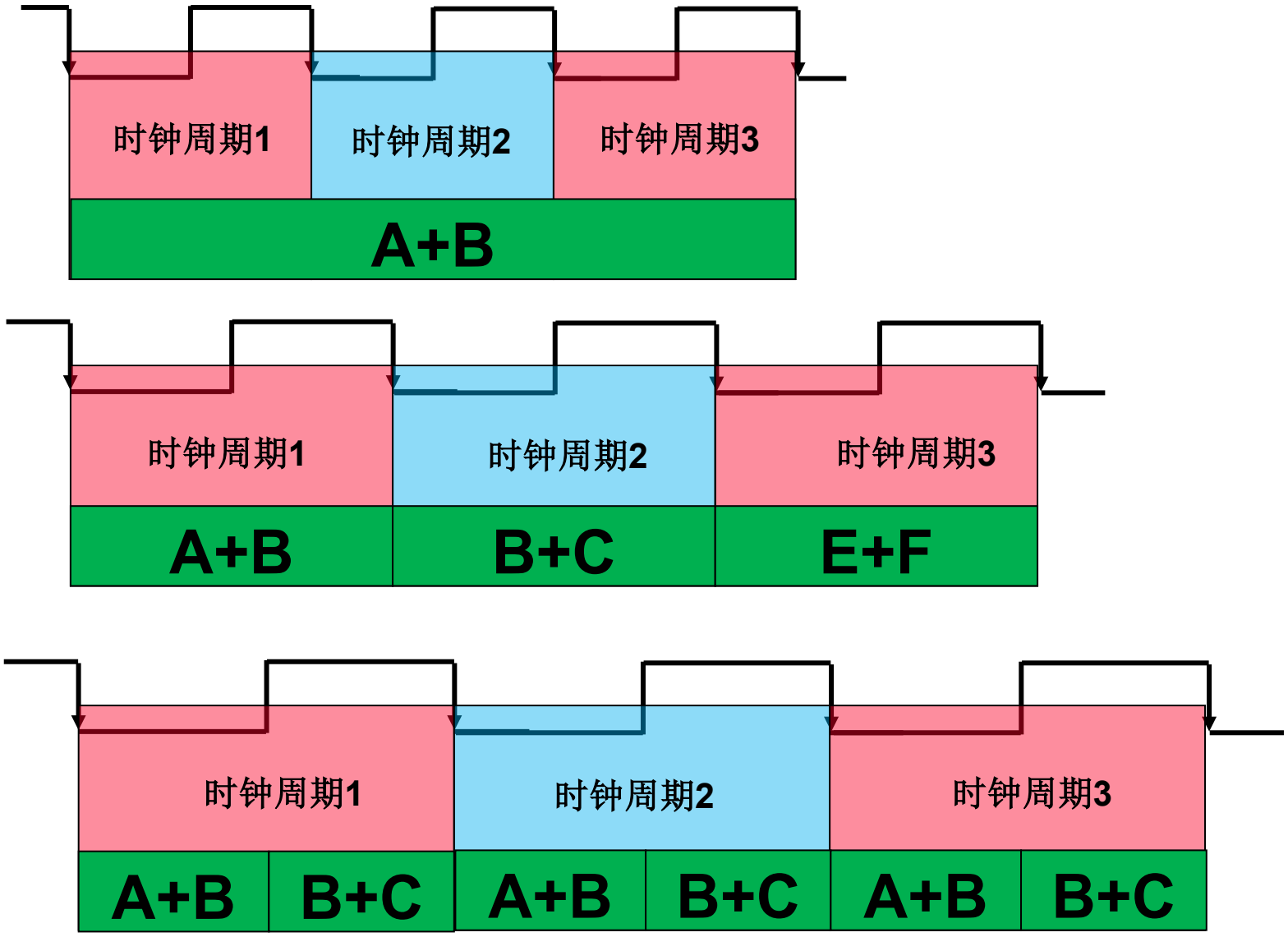


数据通路由 “ … + 状态元件 + 操作元件(组合电路) + 状态元件 + …” 组成

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
- 约束条件：(Latch Prop + Shortest Delay Path - Clock Skew) > Hold Time

哪一种更好呢？——不能简单下结论



计算机（系统）性能

- 计算机有两种不同的性能
  - Time to do the task
    - 响应时间（response time）
    - 执行时间（execution time）
    - 等待时间或时延（latency）
  - Tasks per day, hour, sec, ns. ..
    - 吞吐率（throughput）
    - 带宽（bandwidth）

不同应用场合用户关心的性能不同：

-要求吞吐率高的场合，例如：

多媒体应用（音/视频播放要流畅）

-要求响应时间短的场合：例如：

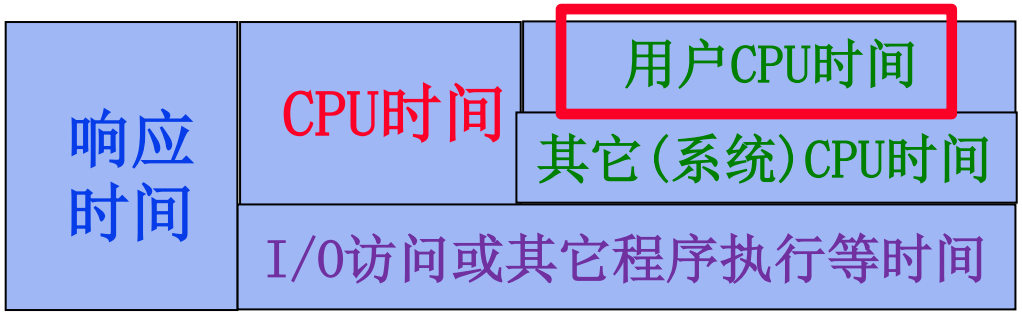
事务处理系统（存/取款速度要快）

-要求吞吐率高且响应时间短的场合：

文件服务器、Web服务器等

系统性能 和 CPU性能

CPU总是被多个程序（OS、多个应用软件等）共享使用，所以：



程序由指令构成。

CPU性能(CPU performance)：用户CPU时间 → 就是执行用户程序中各条指令的总时间

系统性能(System performance)：

一般指没有其它负载时的响应时间

吞吐率 = 单位时间内运行的作业（指令）数（有或无负载/干扰）

“机器X的速度（性能）是Y的n倍”的含义：

$$\frac{\text{ExTime(Y)}}{\text{ExTime(X)}} = \frac{\text{Performance(X)}}{\text{Performance(Y)}} = n$$

CPU执行时间（用户CPU时间，下同）的计算

CPU 执行时间 = CPU时钟周期数 / 程序 ÷ 时钟频率

= CPU时钟周期数 / 程序 × 时钟周期

= 指令条数 / 程序 × CPI × 时钟周期

CPI：Cycles Per Instruction

对于某一条特定的指令而言，其CPI是一个确定值——与CPU设计有关。无需计算。

但是，对于某一个程序或一台机器而言，其CPI是一个平均值，表示该程序或该机器指令集中的1条指令执行时平均需要多少时钟周期。

如何计算平均CPI?

假定CPI_i和C_i分别为第i类指令的CPI和指令条数，则程序的总时钟数为：

$$\text{总时钟数} = \sum_{i=1}^{n} CPI_i \times C_i$$

所以，

$$\text{CPU时间} = \text{时钟周期} \times \sum_{i=1}^{n} CPI_i \times C_i$$

（1）一种计算程序综合CPI的方法:

$$\text{CPI} = \text{总时钟周期数} / \text{指令条数} = (\text{CPU 时间} \times \text{时钟频率}) / \text{指令条数}$$

（2）另一种计算程序综合CPI的方法：假定CPI_i、F_i是各指令CPI和在程序中的出现频率（所有频率总和为1）：

$$\text{CPI} = \sum_{i=1}^{n} CPI_i \times F_i$$

$$F_i = \frac{C_i}{\text{程序的总指令条数}}$$

一台机器的CPI：考虑该机器的指令集（每类指令的CPI 不同），也需要考虑该机器上运行何种程序（用到的指令频率会不同），最终进行平均

CPI和CPU性能的关系？

由上页可知（以下均针对“一个程序”来计算）：

——程序CPI = CPU时钟周期数 ÷指令条数

有了程序CPI，反过来可以计算：

——CPU 执行时间 = 指令条数 × CPI × 时钟周期

CPI 常用来衡量以下各方面的综合结果

- Instruction Set Architecture（ISA）
- Implementation of the architecture
  （Organization & Technology）
- Program（Compiler、Algorithm）

但是，单靠CPI的大小，并不能绝对准确的反映CPU性能！

影响CPU性能的各个方面

指令条数 / 程序 × CPI × 时钟周期

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	instr. count	CPI	clock rate
Programming	X	X	
Compiler	X	(X)	
Instr. Set Arch.	X	X	
Organization		X	X
Technology			X

计算机硬件性能与ISA、计算机组织、计算机实现技术息息相关

Marketing Metrics （产品宣称指标）

MIPS = Instruction Count / Second x1/10^6

= Clock Rate / CPI x 1/10^6

Million Instructions Per Second （定点指令执行速度）

因为每条指令执行时间不同，所以MIPS总是一个平均值。

MFLOPS = FP Operations / Second x 1/10^6

Million Floating-point Operations Per Second（浮点操作速度）

都有各自的局限。

- 一个GFLOPS（gigaFLOPS）等於每秒拾亿（=10^9）次的浮点运算，
- 一个TFLOPS（teraFLOPS）等於每秒万亿（=10^12）次的浮点运算，
- 一个PFLOPS（petaFLOPS）等於每秒千万亿（=10^15）次的浮点运算，
- 一个EFLOPS（exaFLOPS）等於每秒百亿亿（=10^18）次的浮点运算

基准测试程序

基准测试程序——专门用来进行性能评价的一组程序

浮点运算实际上包括了所有涉及小数的运算，在某类应用软件中常常出现，比整数运算更费时间。现今大部分的处理器中都有浮点运算器。因此每秒浮点运算次数所量测的实际上就是浮点运算器的执行速度。

最常用来测量每秒浮点运算次数的基准程序（benchmark）之一，就是Linpack。

基准测试程序开发：SPEC (Systems Performance Evaluation Committee) http://www.spec.org

例子1

程序P在机器A上运行需10 s， 机器A的时钟频率为400MHz。现在要设计一台机器B，希望该程序在B上运行只需6 s.

机器B时钟频率的提高导致了其CPI的增加，使得程序P在机器B上时钟周期数是在机器A上的1.2倍。机器B的时钟频率达到A的多少倍才能使程序P在B上执行速度是A上的10/6=1.67倍？

分析:

CPU时间A （ 10s ）= 时钟周期数A / 时钟频率A

所以，时钟周期数A = 10 sec x 400MHz = 4000M个

时钟频率B = 时钟周期数B / CPU时间B

= 1.2 x 4000M / 6 sec = 800 MHz

机器B的频率是A的两倍，但机器B的速度并不是A的两倍！

例子2：如何给出综合评价结果？

问题：如果用一组基准程序在不同机器上测出了运行时间，那么如何综合评价机器的性能呢？

先看一个例子：

程序 1: A上运行1秒，　　B上运行10秒

程序2: A上运行1000秒，　　B上运行100秒

- A 速度是B的10倍 ？ （如果看程序1）
- B速度是A的10倍 ？ （如果看程序2）

可以计算总时间作为综合度量值：

B上运行110秒，A上运行1001秒（B是 A 的9.1倍）

可考虑每个程序在作业中的使用频度，即加权平均

综合评价（续）

- 可用以下两种平均值来评价：
  - Arithmetic mean(算术平均)：求和后除n
  - Geometric mean(几何平均)：求积后开根号n
- 根据算术平均执行时间能得到总平均执行时间
- 根据几何平均执行时间不能得到程序总的执行时间
- ----------
- 执行时间的规格化（测试机器相对于参考机器的性能）：
  - 参考机器上执行时间÷ 待测机器上执行时间
- 平均规格化执行时间不能用算术平均计算，而应该用几何平均
  - 程序1 从2秒变成1秒 等价于 程序2 从2000秒变成1000秒.
    （算术平均值不能反映这一点）

综上所述，算术平均和几何平均各有长处，可灵活使用

第8章 中央处理器（一）

第一讲 中央处理器概述

第二讲 单周期数据通路的设计

第三讲 单周期控制器的设计

第四讲 多周期处理器的设计

第五讲 流水线处理器设计

第六讲 流水线冒险及其处理

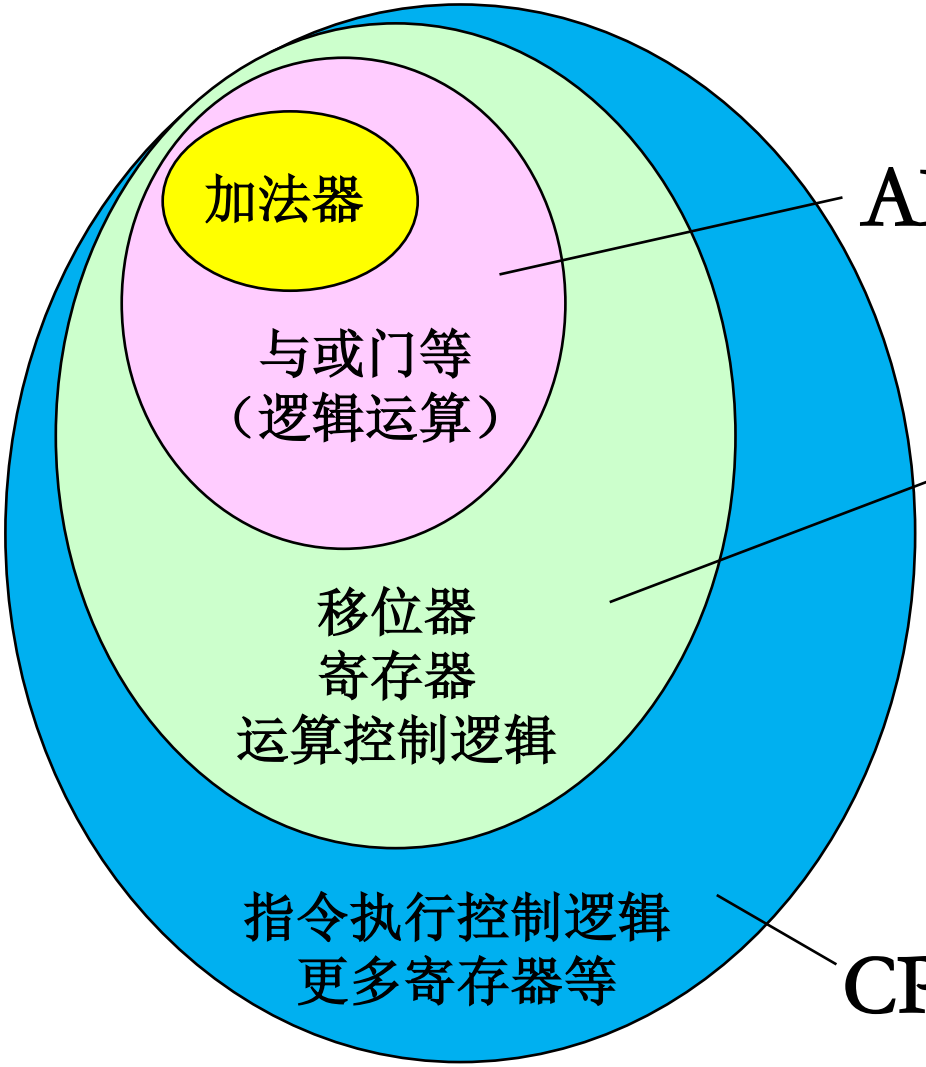
第七讲 高级流水线技术

第二讲 单周期数据通路设计

主 要 内 容

- 实现目标（RV32I中的9条指令）概述
- 扩展器部件的设计
- 算术逻辑部件的设计
- 取指令部件的设计
- R-型指令的数据通路
- I-型指令的数据通路
- U-型指令的数据通路
- Load/Store指令的数据通路
- B-型指令的数据通路
- J-型指令的数据通路
- 完整的单周期数据通路

回顾： CPU功能及其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（回顾）
  - **指令数目、CPI、时钟周期**
    - 指令数目由编译器和ISA决定
    - CPI由ISA和CPU的实现来决定
    - 时钟周期由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。

ALU以加法器为核心

运算部件以ALU/加法器为核心

（大家都需要：多路选择器和传输线路等）

单总线数据通路

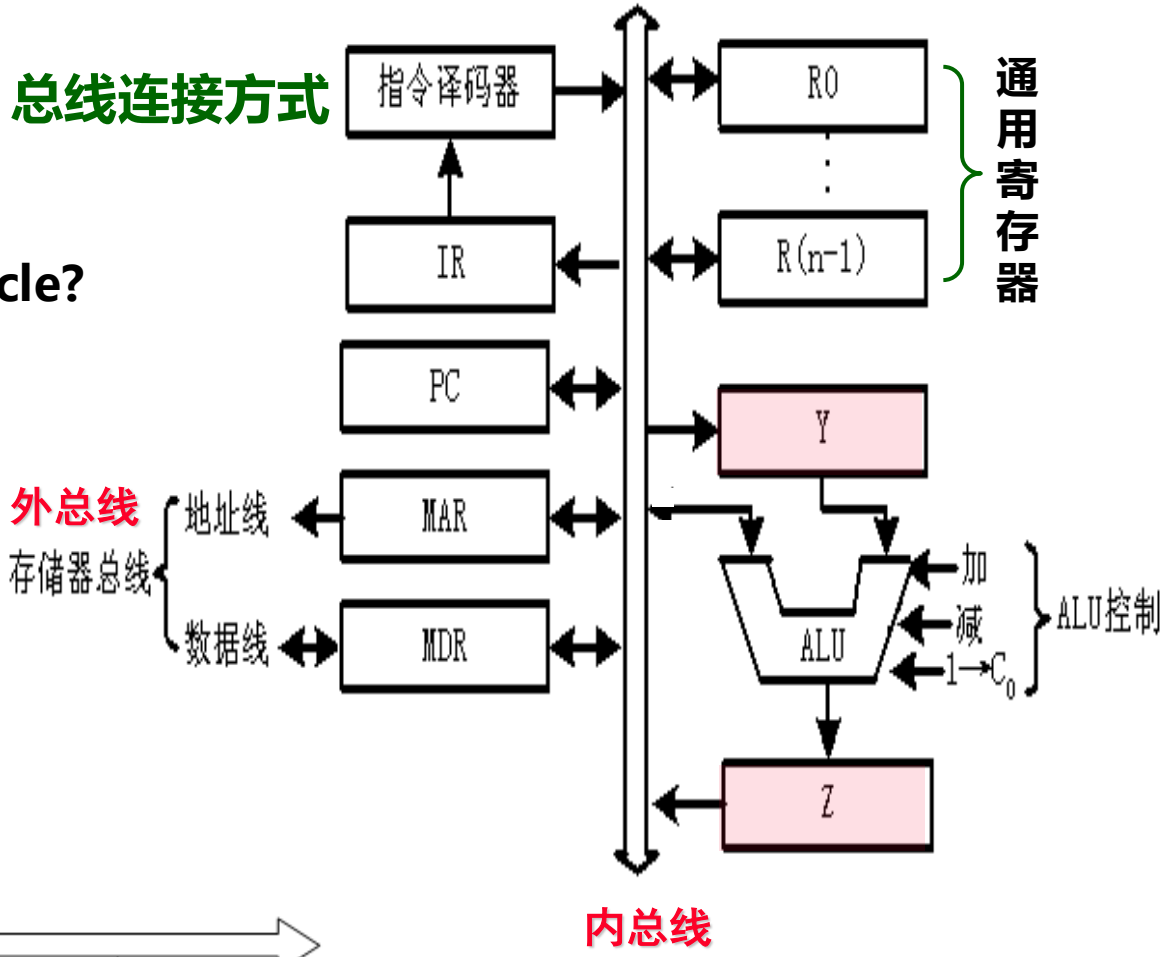
四种基本操作的时序

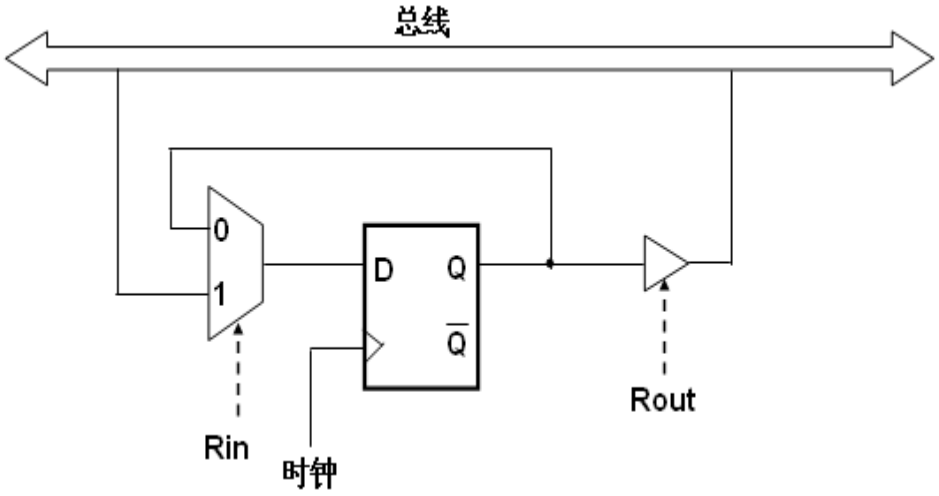
- **在寄存器之间传送数据**
  R0out，Yin
- **完成算术、逻辑运算**
  R1out，Yin
  R2out，Add，Zin
  Zout，R3in

1Cycle?

3Cycles?

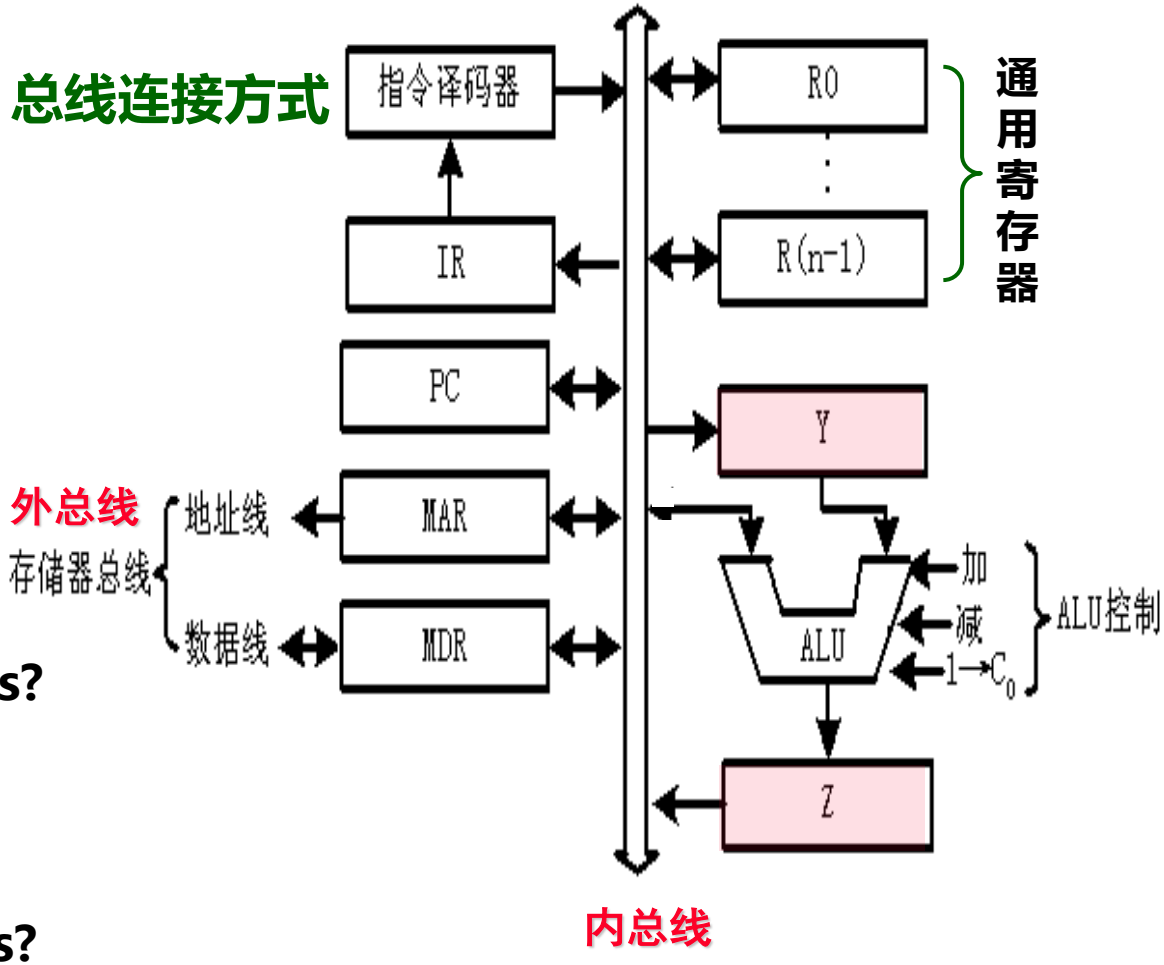
总线连接方式





单总线数据通路

四种基本操作的时序（续）



- 从主存取字 R[R2]←M[R[R1]]
  R1out，MARin　3+？ Cycles?
  Read, WMFC (等待MFC)
  MDRout，R2in
- 写字到主存 M[R[R1]] ← R[R2]
  R1out，MARin　3+？ Cycles?
  R2out，MDRin，
  Write, WMFC

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。

时钟周期的宽度如何确定？

以“Riout,OP（Add、Sub、And等）,Zin”还是以“Read/Write”所花时间来确定？

Read/Write时间长，机器周期以此为准！

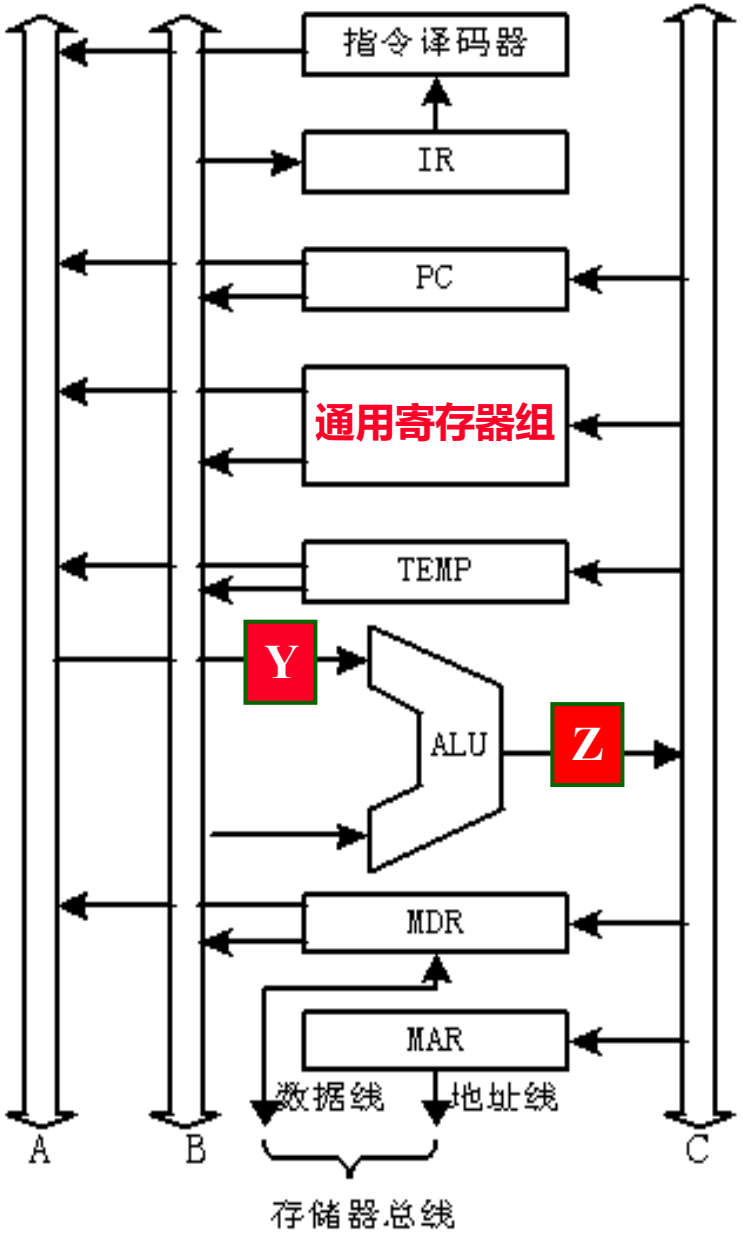
三总线数据通路（自学）

- 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

    三个总线各自传不同的数据，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！



目前大都采用流水线方式执行指令，单总线或三总线的总线式数据通路很难实现指令流水执行。