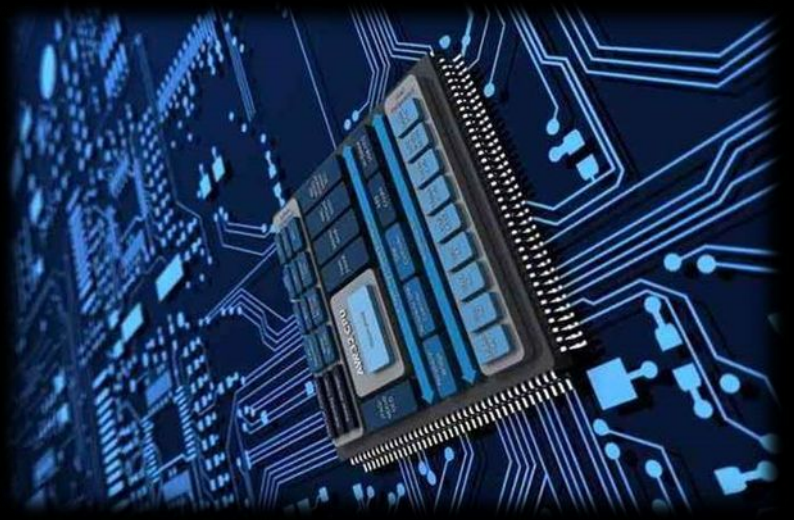
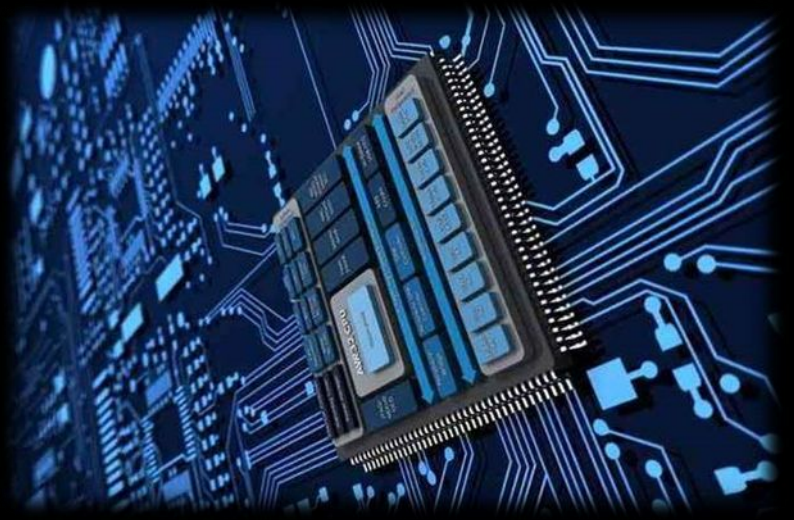


CPU的结构与功能

大连理工大学 赖晓晨



控制器概述

大连理工大学 赖晓晨

ARM公司

- ARM前身ACORN计算机公司，曾向Intel公司索要80286芯片的设计资料，遭到拒绝，于是被迫自行研发；
- ARM公司成立于1990年，最初只有12人，办公室只是一间仓库，现在有1700名员工；
- ARM公司的经营模式。
- 2012年，ARM授权处理器全球出货87亿颗。在手机、pad等领域占据绝对统治地位。

控制器的功能

- 取指令
- 分析指令
- 执行指令，发出各种操作命令
- 总线管理
- 处理中断、异常等特殊情况

CPU的寄存器

1. 用户可见寄存器

(1) 通用寄存器　存放操作数

可作 某种寻址方式所需的 寄存器

(2) 数据寄存器　存放操作数（满足各种数据类型）

两个寄存器拼接存放双倍字长数据

(3) 地址寄存器　存放地址，其位数应满足最大地址范围

用于特殊的寻址方式　段基址 栈指针

(4) 条件码寄存器　存放条件码，可作程序分支的依据

如正、负、零、溢出、进位等

CPU的寄存器

2. 控制和状态寄存器

(1) 控制寄存器

PC → MAR → M → MDR → IR

控制 CPU 操作

其中 MAR、MDR、IR　用户不可见

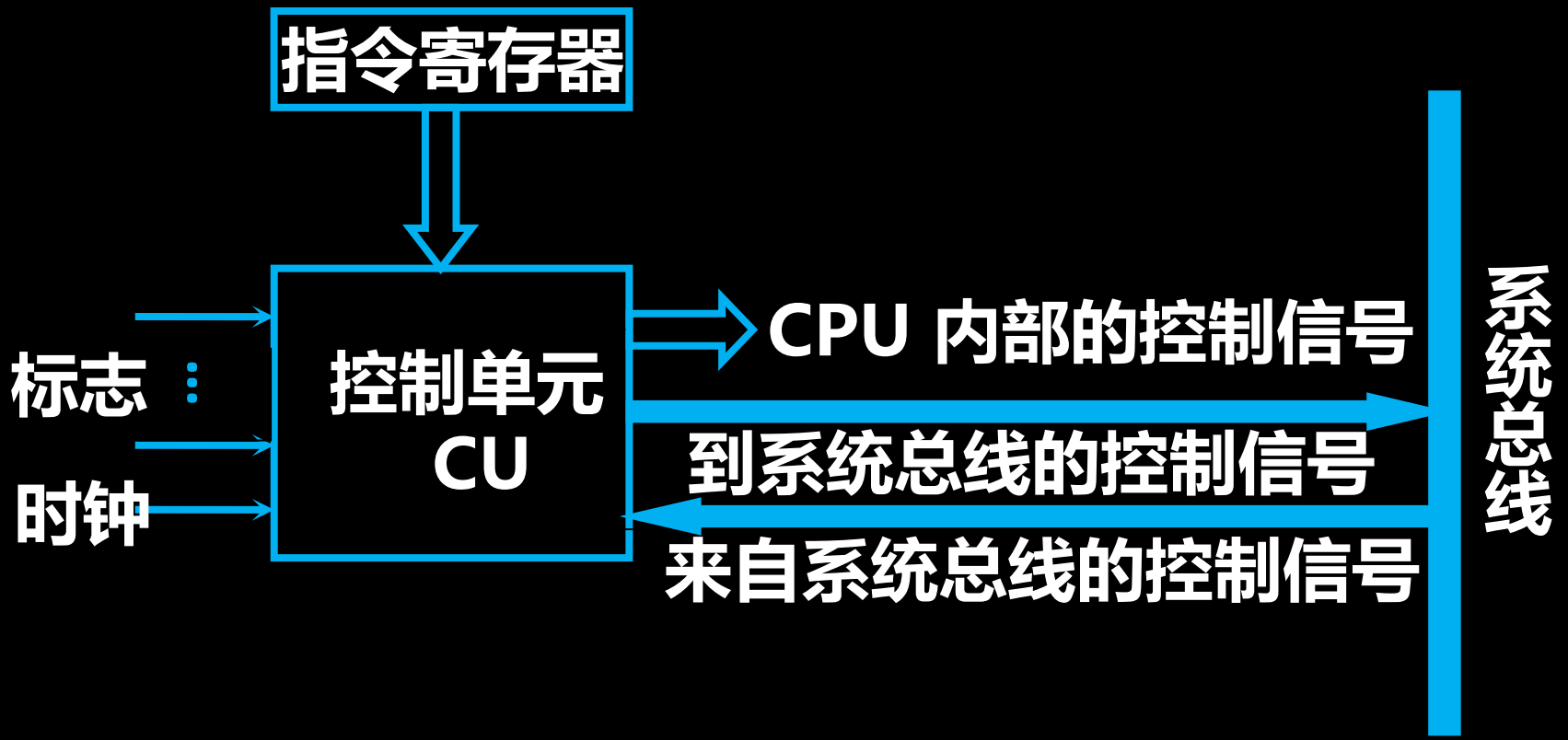
PC　用户可见

(2) 状态寄存器

PSW/flags　存放程序状态字

CPU的控制单元

控制单元的外特性



输入信号

(1) 时钟

CU受时钟控制

一个时钟脉冲

发一个操作命令或一组需同时执行的操作命令

(2) 指令寄存器

OP（IR）→ CU

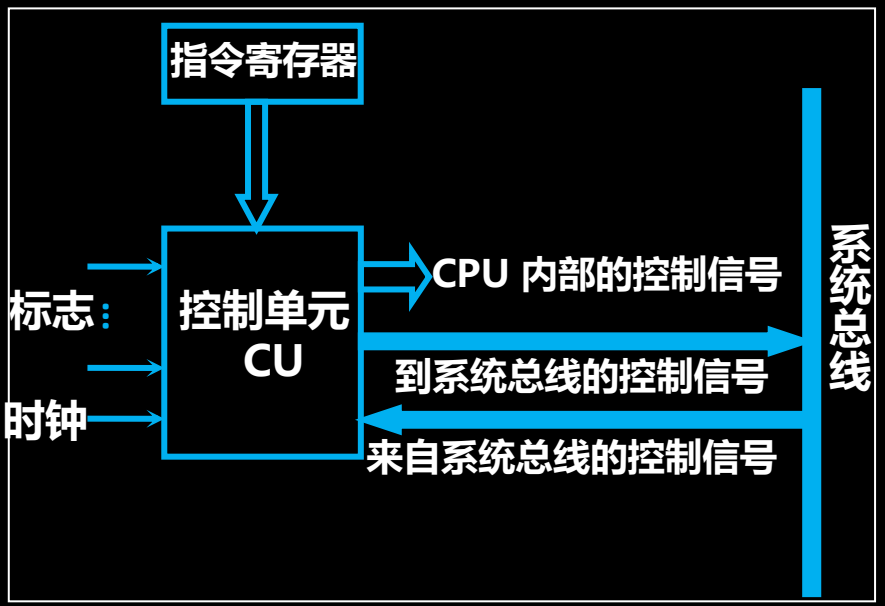
控制信号与操作码有关

(3) 标志 CU受标志控制

(4) 外来信号

如INTR中断请求

HRQ总线请求



输出信号

(1) CPU内的各种控制信号

$R_i \rightarrow R_j$

(PC) + 1 → PC

ALU　+、-、与、或 ……

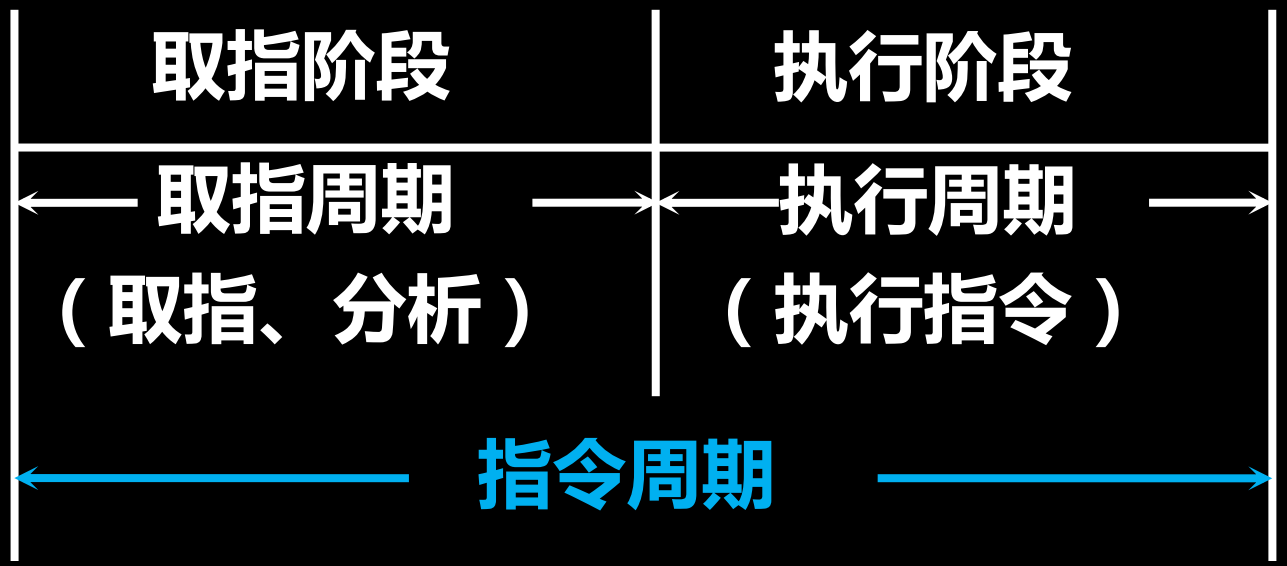
(2) 送至控制总线的信号

$\overline{\text{MREQ}}$	访存控制信号
$\overline{\text{IO}}/\text{M}$	访 IO/ 存储器的控制信号
$\overline{\text{RD}}$	读命令
$\overline{\text{WR}}$	写命令
INTA	中断响应信号
HLDA	总线响应信号

指令周期的基本概念

取出并执行一条指令所需的全部时间

完成一条指令 { 取指、分析 — **取指周期**
执行 — **执行周期**



指令周期的长度

1. 只有取指周期的指令周期

NOP

2. 有取指和执行周期的指令周期

ADD mem

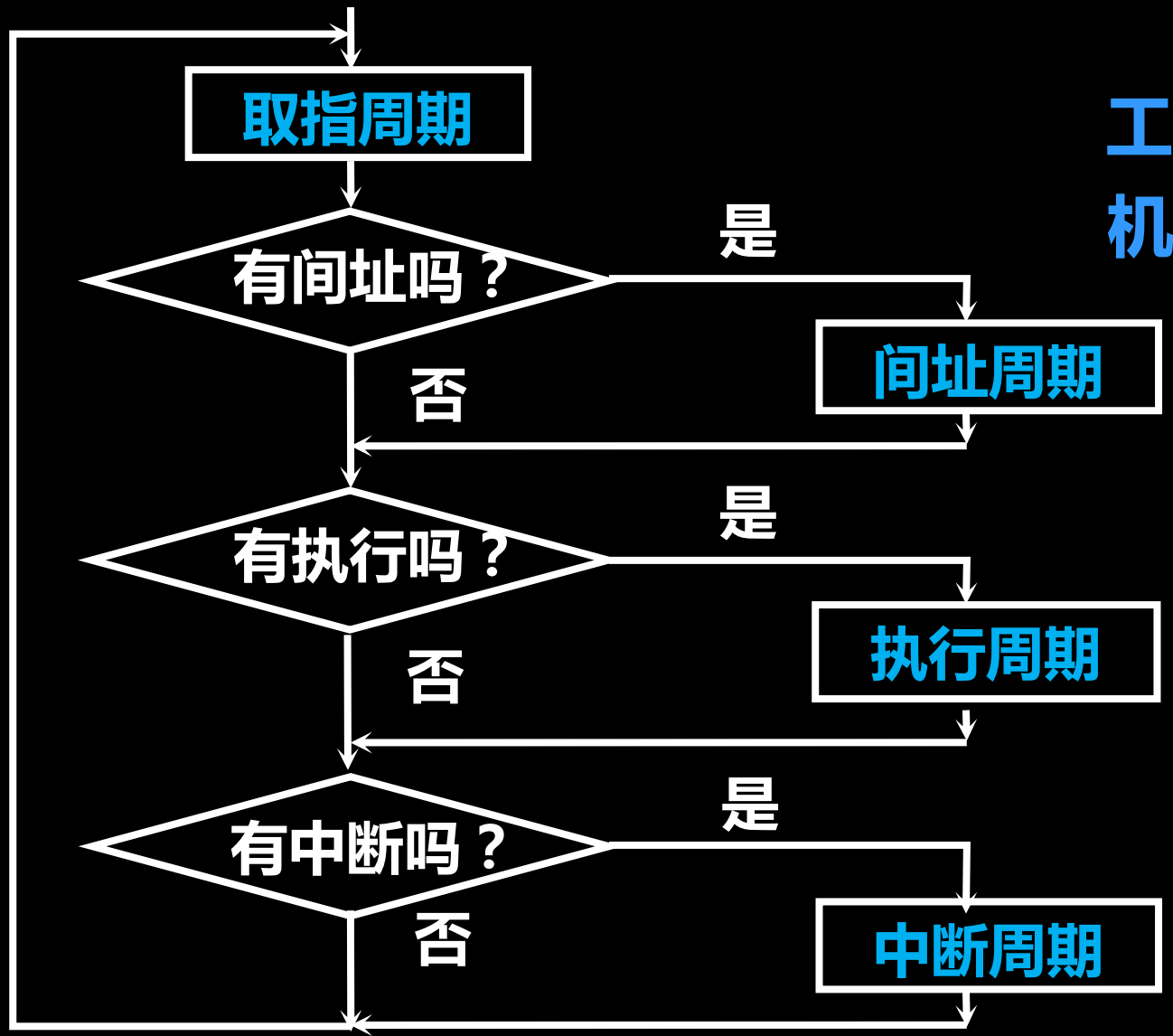
MUL mem

指令周期的长度

3. 具有间接寻址的指令周期

4. 带有中断周期的指令周期

指令周期流程



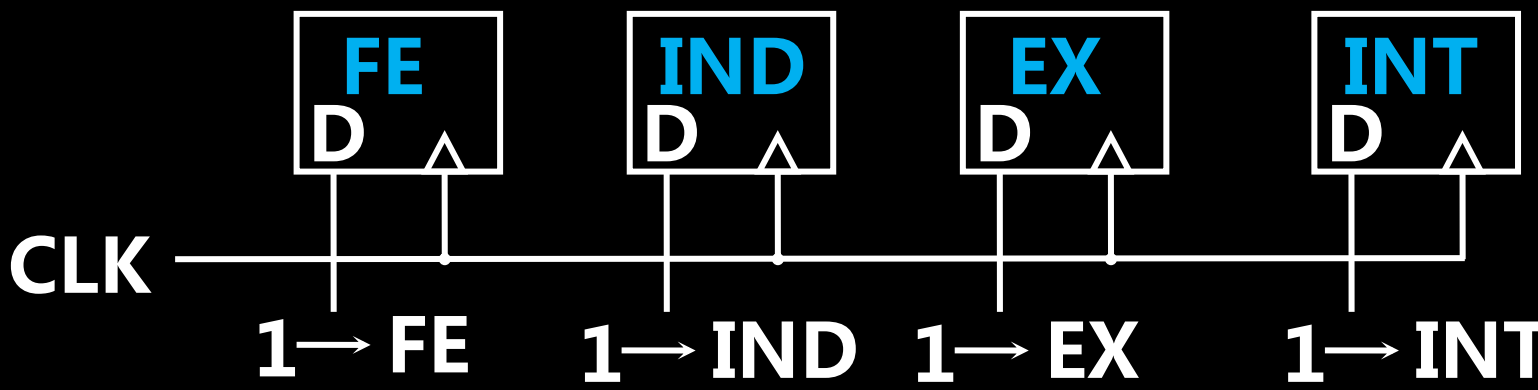
工作周期

机器周期

CPU工作周期的标志

CPU的4个工作周期

取指周期
间址周期
执行周期
中断周期



1. 机器周期

(1) 机器周期的概念

- 指令执行的每个阶段，称为一个机器周期

机器周期(工作周期)：取指、间址、执行、中断周期

(2) 确定机器周期需考虑的因素

- 每条指令的执行步骤。
- 每一步骤所需的时间。

(3)（统一）机器周期的确定

- 以完成最复杂、最慢指令功能的时间为准
- 以访问一次存储器的时间为基准

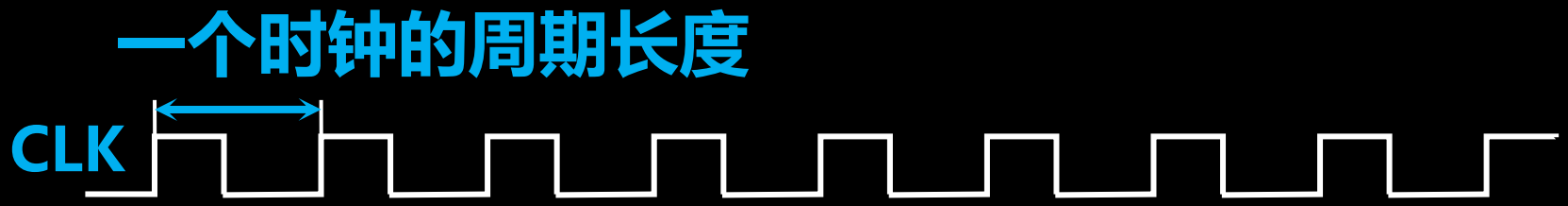
若指令字长=存储字长　　访存周期=机器周期

2. 时钟周期

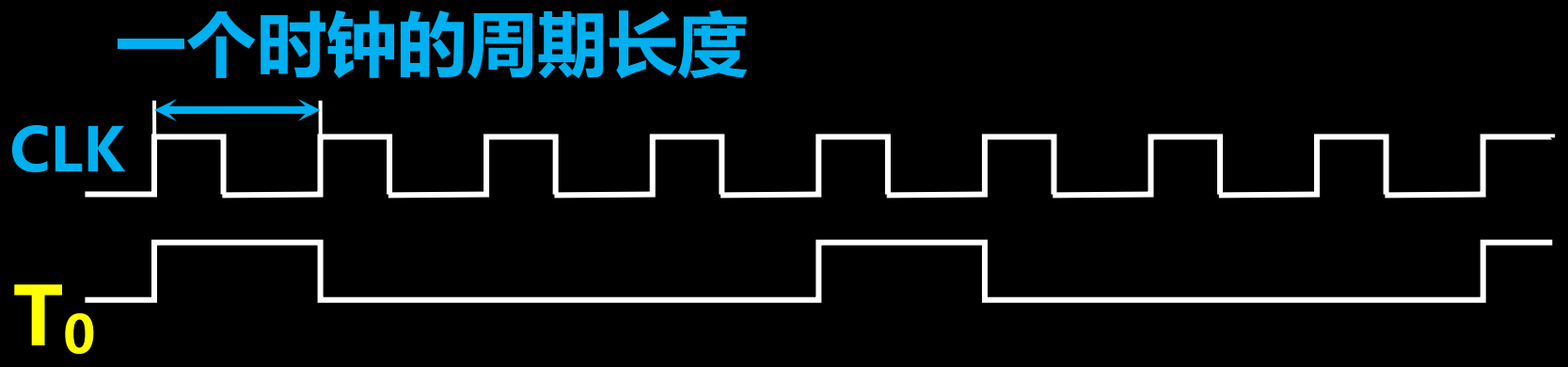
- 将一个机器周期分成若干个时间相等的时间段，每段称为一个时钟周期（节拍、状态）。
- 时钟周期是控制计算机操作的最小单位时间。
- 用时钟周期控制微操作命令，每个时钟周期产生一个或几个微操作命令。

T_0 T_1 T_2 T_3

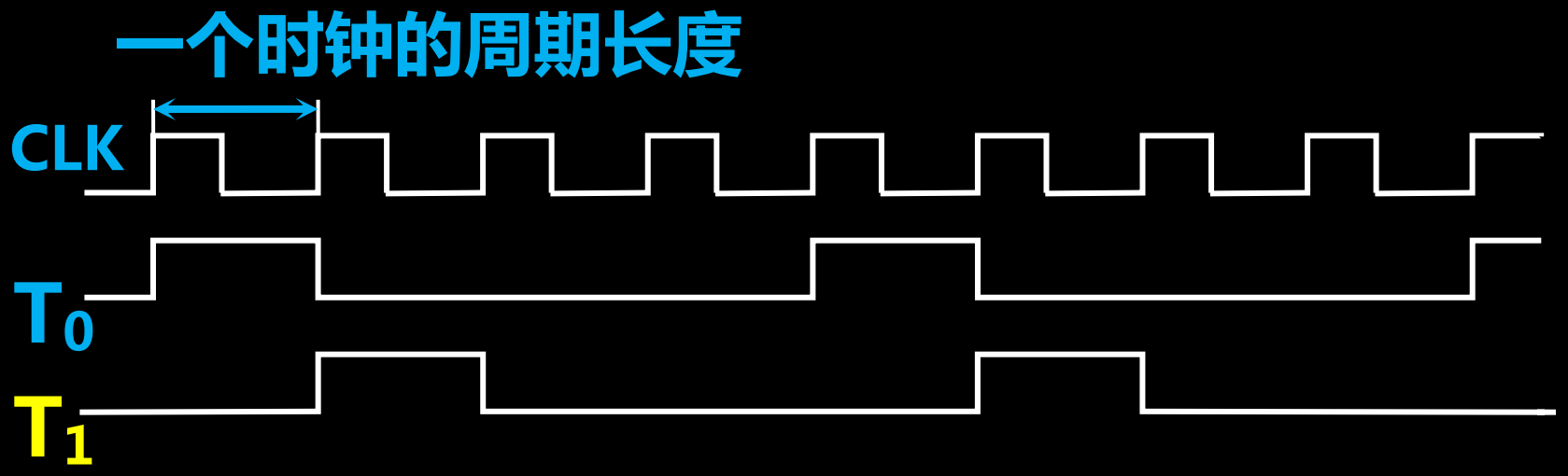
时钟周期的产生方法



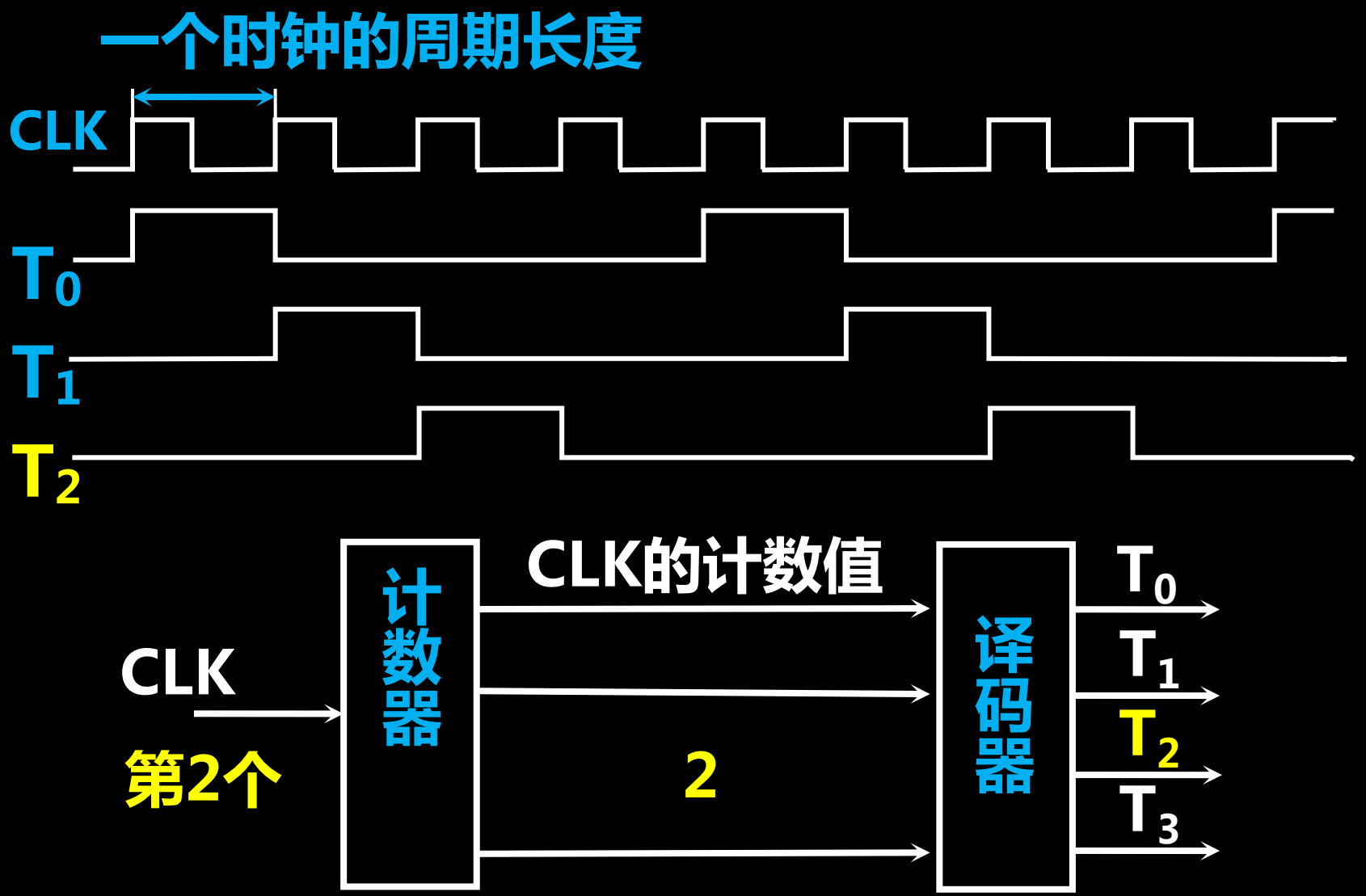
时钟周期的产生方法



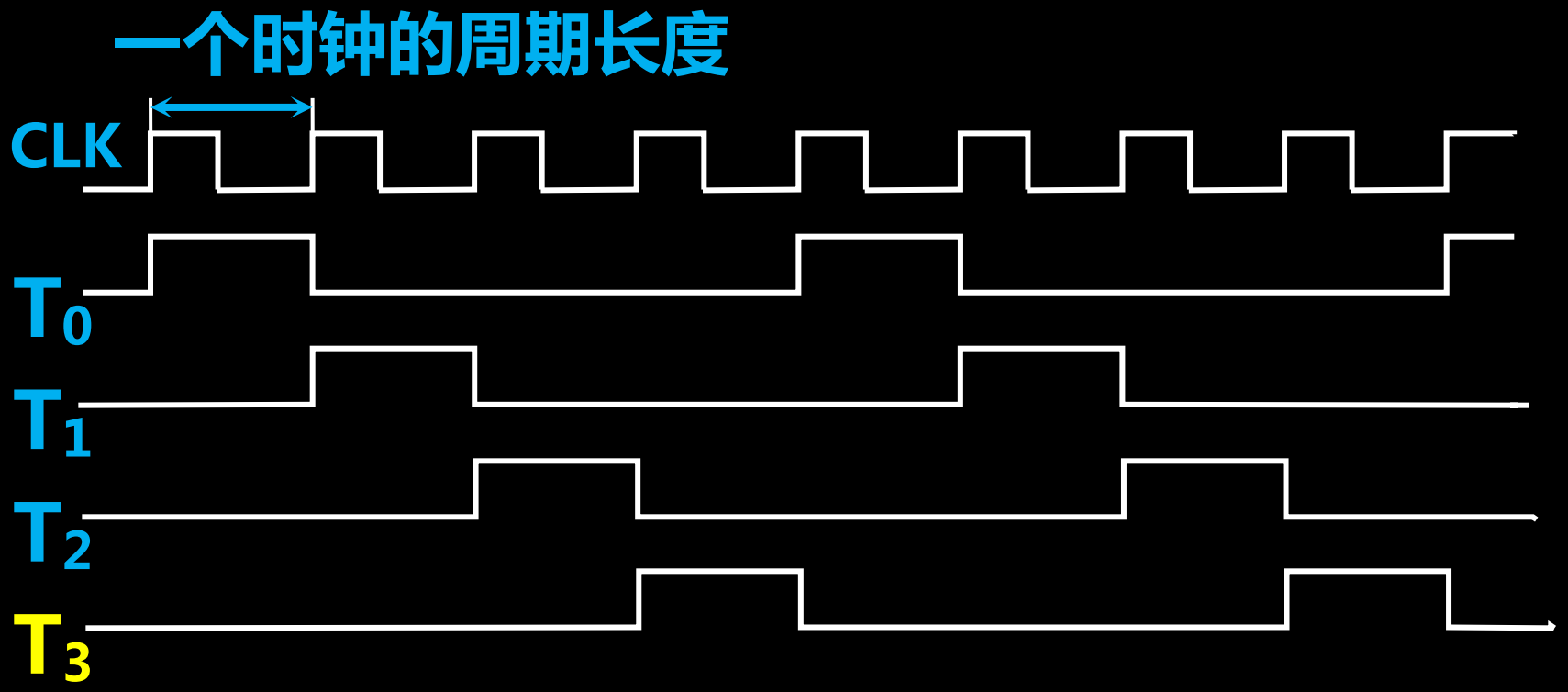
时钟周期的产生方法



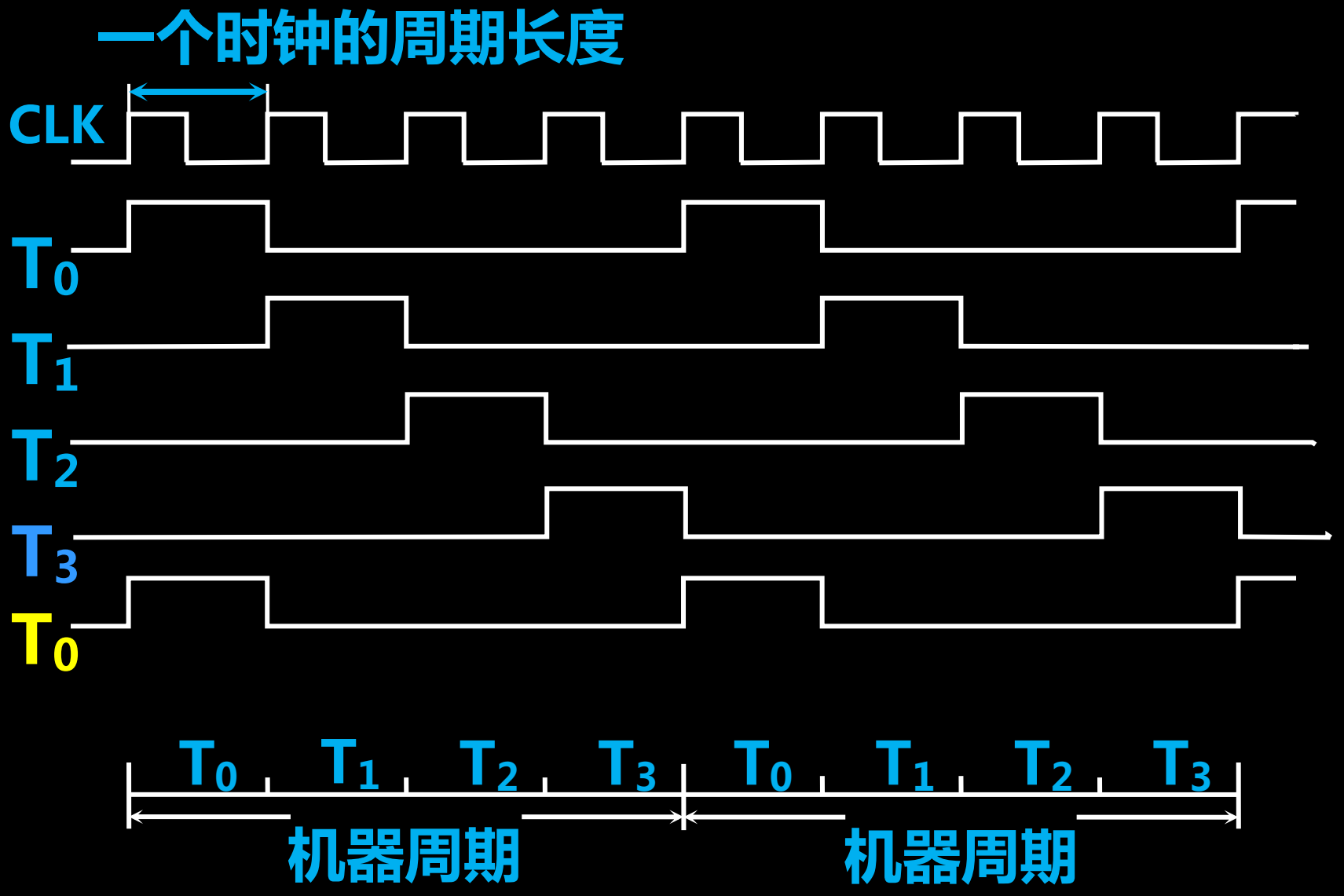
时钟周期的产生方法



时钟周期的产生方法

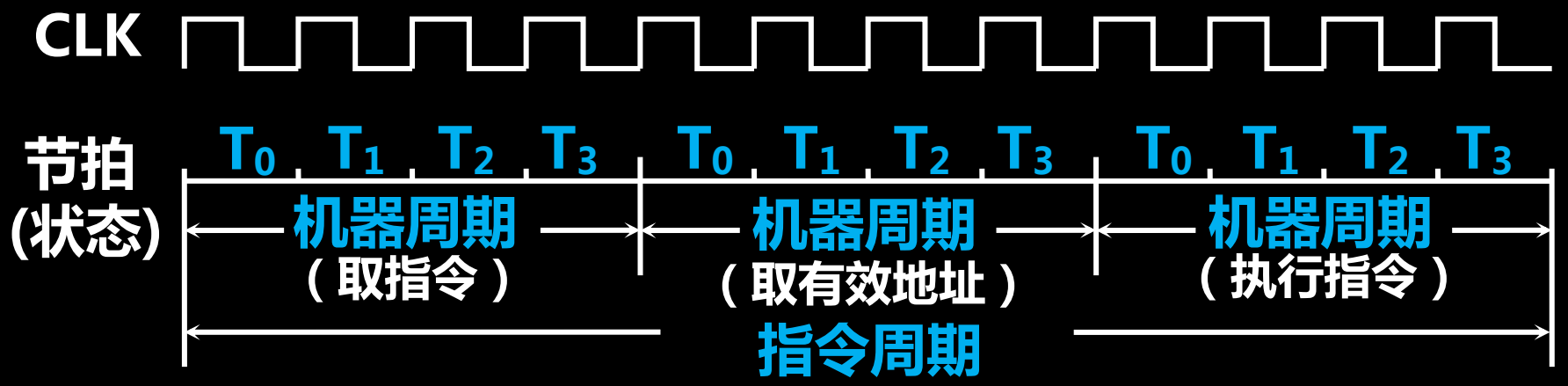


时钟周期的产生方法



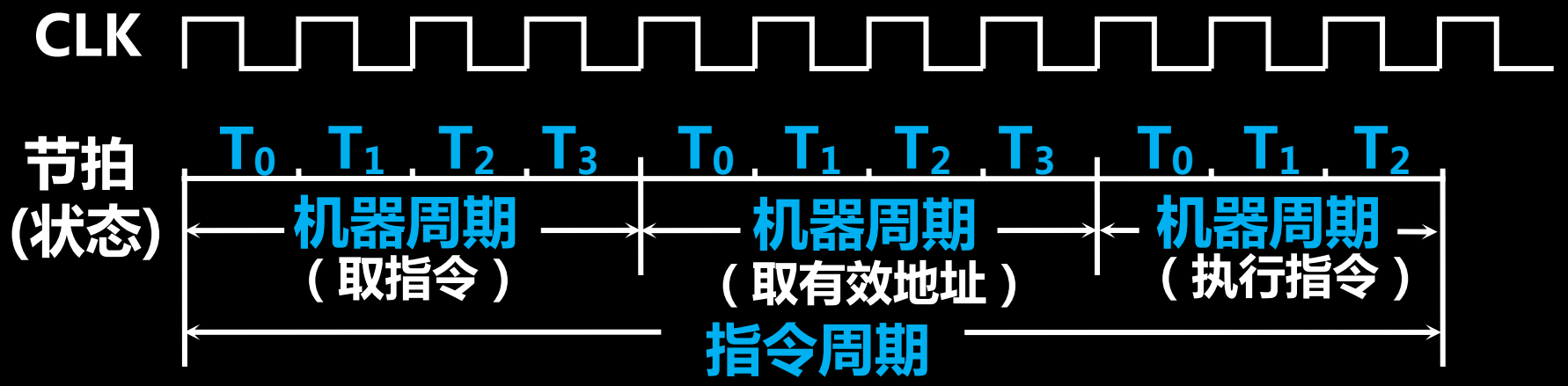
机器周期的类型

1. 定长机器周期



机器周期的类型

2. 不定长机器周期



多级时序系统

指令周期、机器周期、时钟周期（节拍、状态）组成多级（三级）时序系统

- 一个指令周期包含若干个机器周期
- 一个机器周期包含若干个时钟周期

T_i：时钟周期

超线程技术

- 超线程（Hyper-Threading）技术是Intel公司提出的一种提高CPU性能的技术，可将一个物理CPU当做两个逻辑CPU使用，同时执行多个线程，从而提高效率。
- 超线程处理器内部的两个逻辑处理器共享一组处理器执行单元，运算能力提升30%。
- 超线程技术可将处理器内部的闲置资源利用起来，当并行执行两个线程时，负责处理第二个线程的逻辑处理器，仅使用运行第一个线程的处理器不使用的资源。

超线程技术的实现

- 实现超线程技术的前提是需要五大支持：CPU、主板芯片组、主板BIOS、操作系统、应用软件。
- 执行单个线程时，该技术反而下降，因为打开超线程后，处理器内部缓存被划分为几个区域，互相共享内部资源，单个子系统性能反而下降。
- 超线程技术最早出现在2002年的pentium4处理器，以及后续2009的Core i系列处理器

双核处理器

- 双核处理器是在一个处理器芯片上集成了两个“物理的”运算核心。
- 1989年，Intel工程师首先提出了双核的概念，1995年首先发布了双核产品，2005年首先提供主流价位的双核处理器。
- 双核处理器并不能达到同频率两个单核处理器的计算能力，IBM公司曾对比过AMD和Intel双核处理器的性能，在多线程任务环境下，大约比单核提高60%。

超线程与双核处理器的区别

- 开启了超线程技术的单核处理器与双核处理器在操作系统中都被识别为两个处理器，二者的区别在于，前者是两个“逻辑”处理器，后者是两个“物理”处理器。
- 超线程的逻辑处理器并没有独立的执行单元、整数单元、寄存器甚至缓存，它们在运行时仍需要共用执行单元、缓存和总线接口。
- 执行多线程时，超线程的两个处理器交替工作，当争用某资源时，一个线程必须暂停并让出资源。

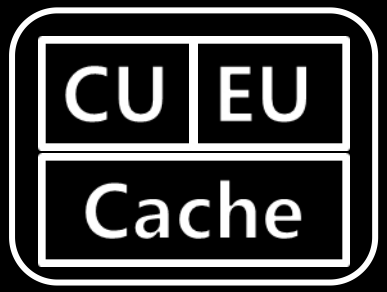
多核多线程技术

- 处理器有多个物理内核，每个内核采用超线程技术。
- Pentium：单核单线程；Pentium4：单核多线程；Pentium D：多核单线程；Pentium EE：多核多线程。
- 下面插图，如放不下4个图，后面再加一页

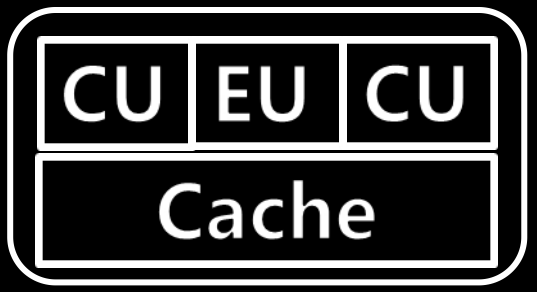
图形处理单元

- 随着图形处理和游戏产业的发展，图形处理器的地位不断上升，产生了图形处理单元（Graphics Processing Unit，GPU）
- GPU是显卡的大脑，是补充CPU的加速器，将CPU从显示任务中解脱出来，可同时执行上千个线程，高度并行
- nVidia、AMD(ATI)
- GPU的编程接口为高层次引用编程接口：OpenGL、DirectX
- 直接在GPU上编程：Brook，C语言（CUDA）

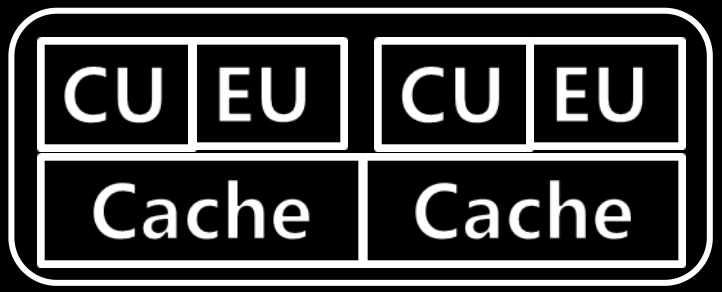
几种微处理器的内部结构



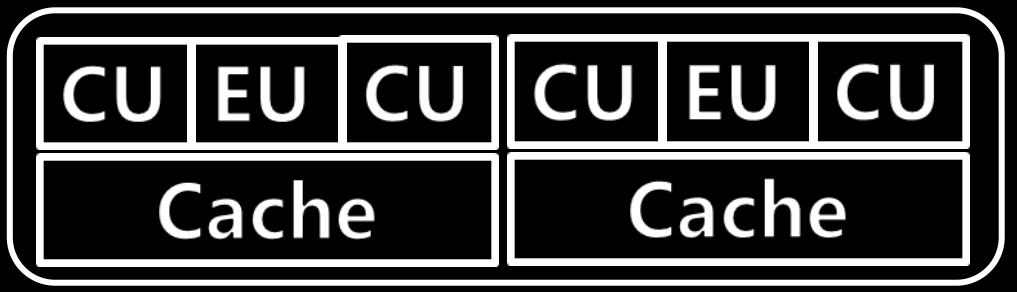
1. 单核单线程处理器



2. 单核多线程处理器



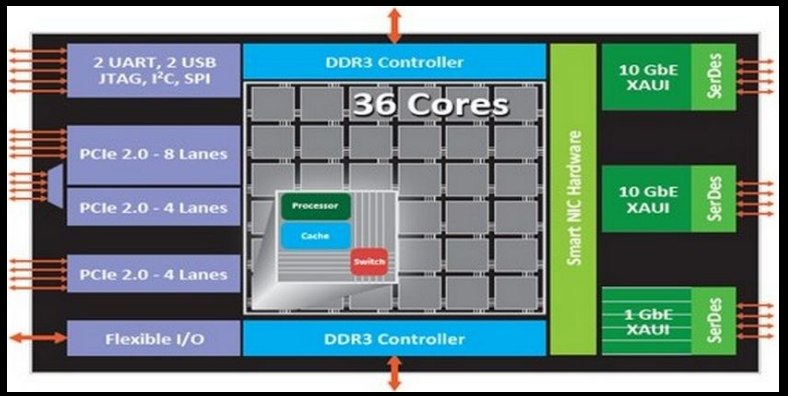
3. 多核单线程处理器

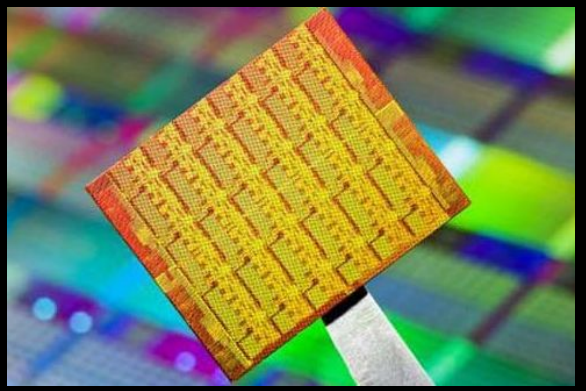


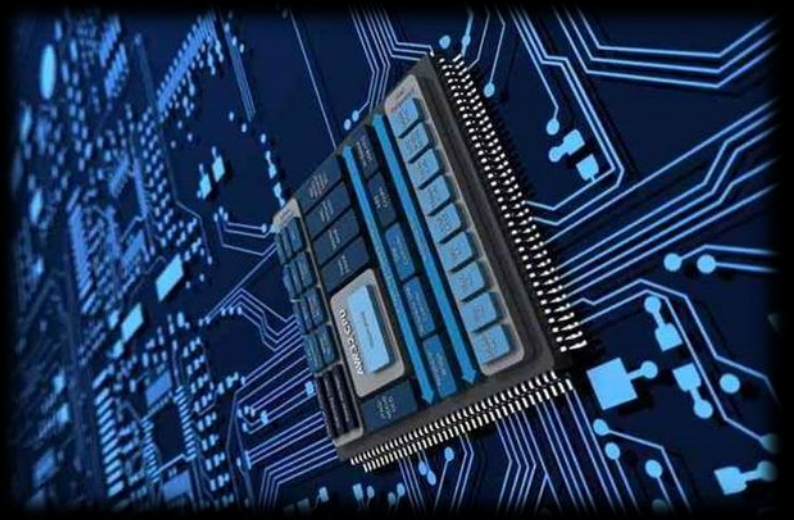
4. 多核多线程处理器

推荐阅读：众核、千核处理器

- 众核处理器比多核处理器中的处理内核数量还要多，计算能力更强大。
- 千核处理器是指内置1000个核心的CPU处理器。在测试中，FPGA芯片每秒能处理5GB的数据，处理速度大概相当于当前台式机的20倍。但能耗却相当低。这种处理器还只是初期概念验证研究，用于实际还为时尚早。





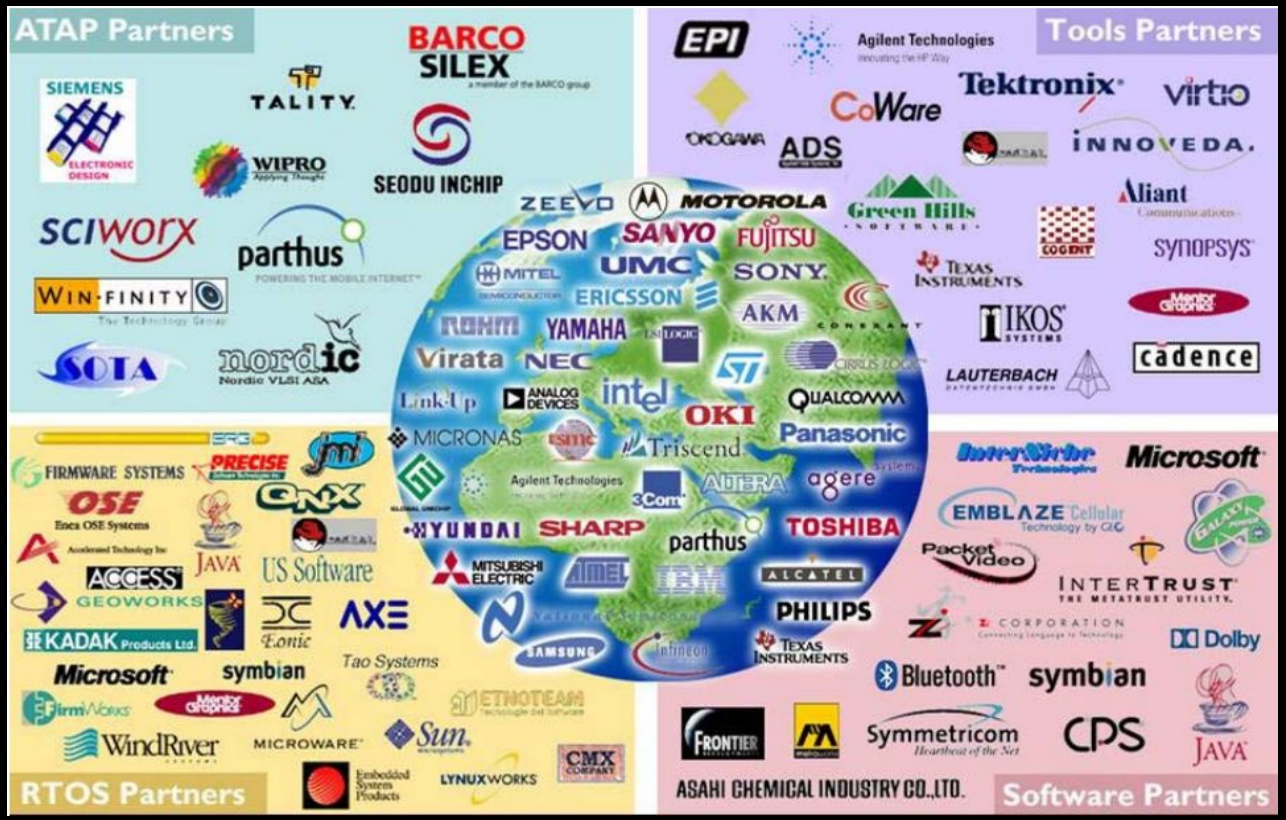


指令周期的数据流

大连理工大学 赖晓晨

ARM公司的小伙伴

ARM的全球技术网络包括领先的半导体和系统伙伴、ARM技术共享计划伙伴、实时操作系统（RTOS）、开发工具伙伴、应用软件伙伴。



取指周期的数据流

1. 取指周期

PC→MAR→地址线

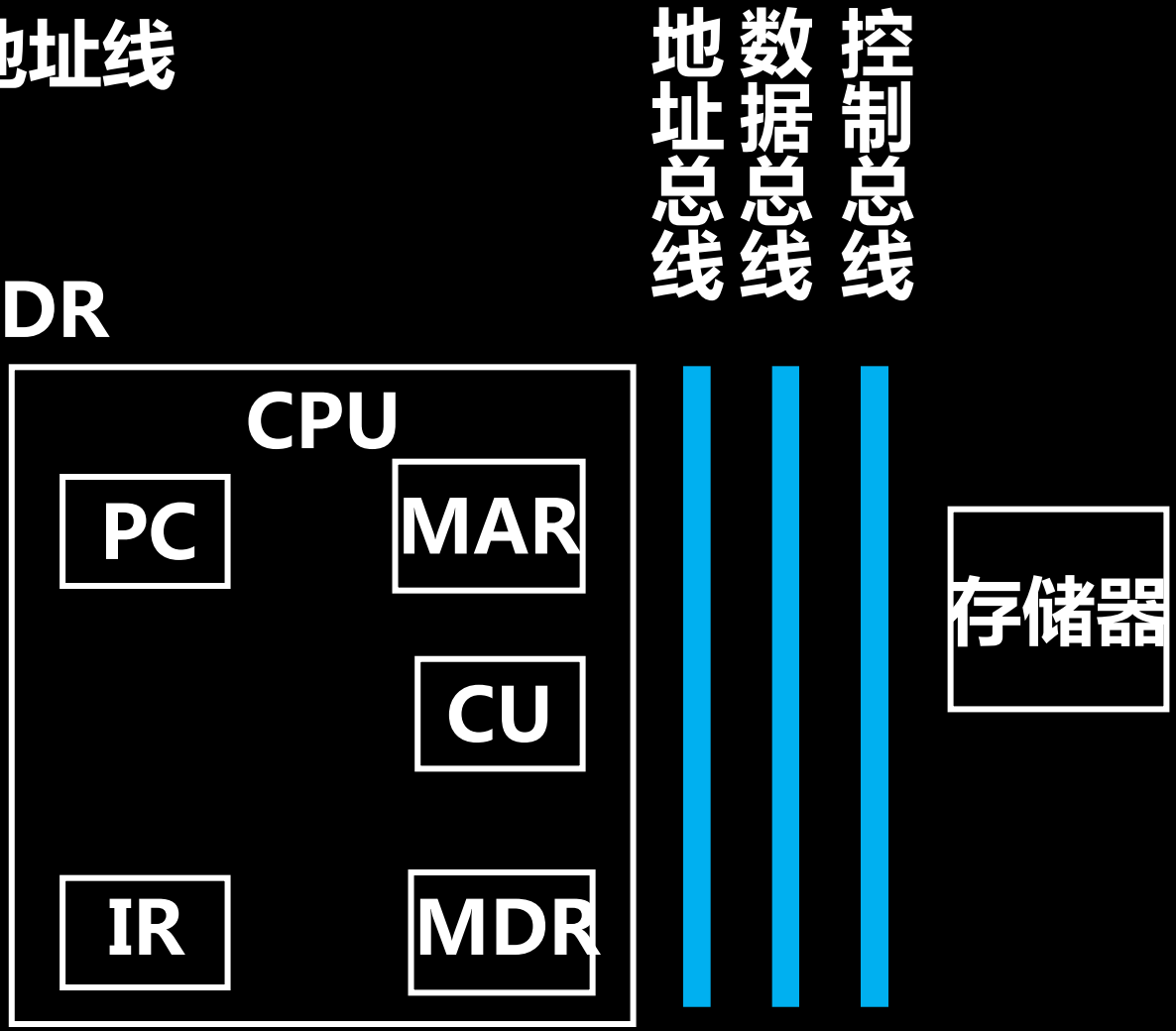
1 → R

M(MAR)→MDR

MDR→IR

OP(IR)→CU

(PC)+1→PC



取指周期的数据流

1. 取指周期

PC→MAR→地址线

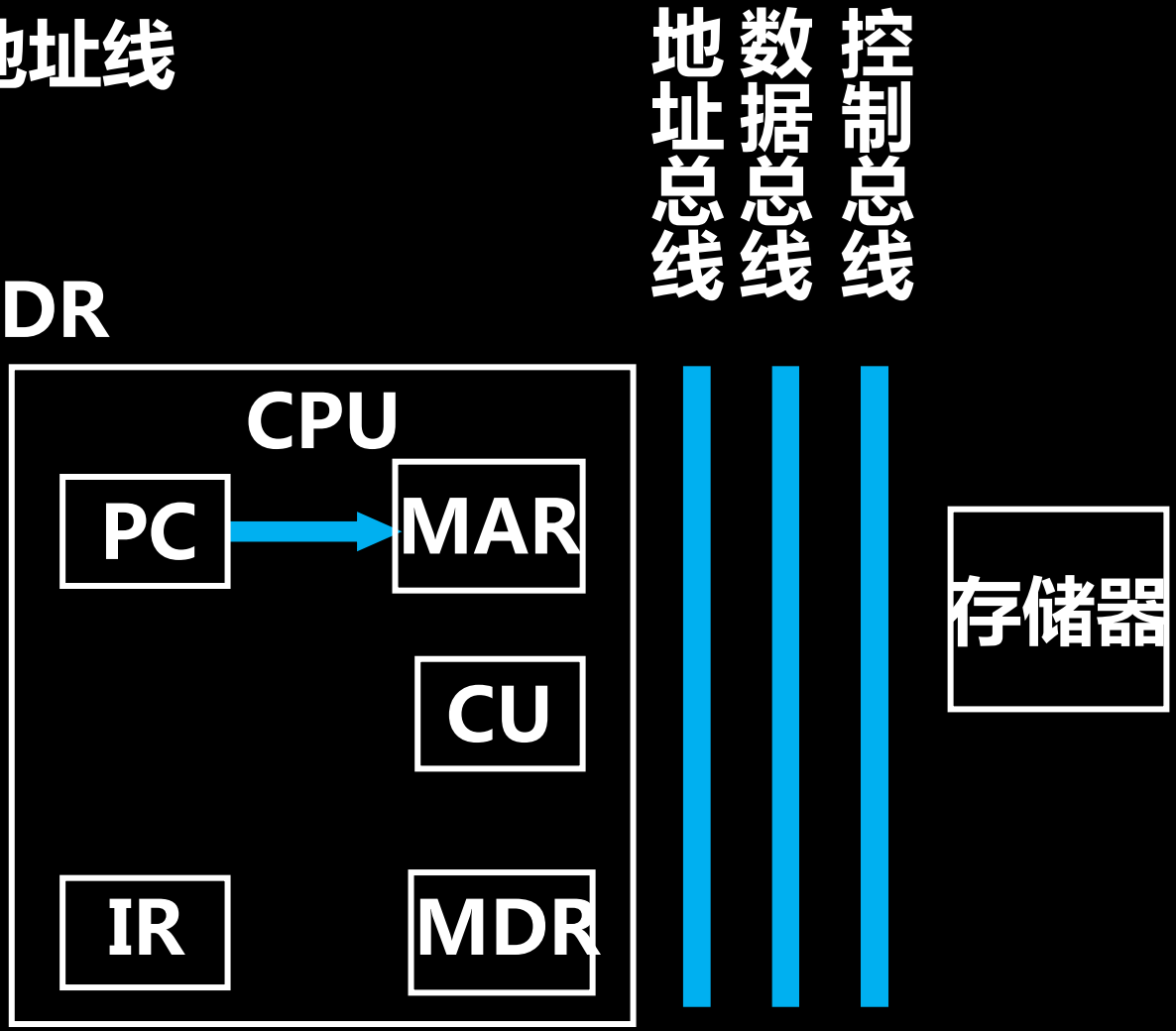
1 → R

M(MAR)→MDR

MDR→IR

OP(IR)→CU

(PC)+1→PC



取指周期的数据流

1. 取指周期

PC→MAR→地址线

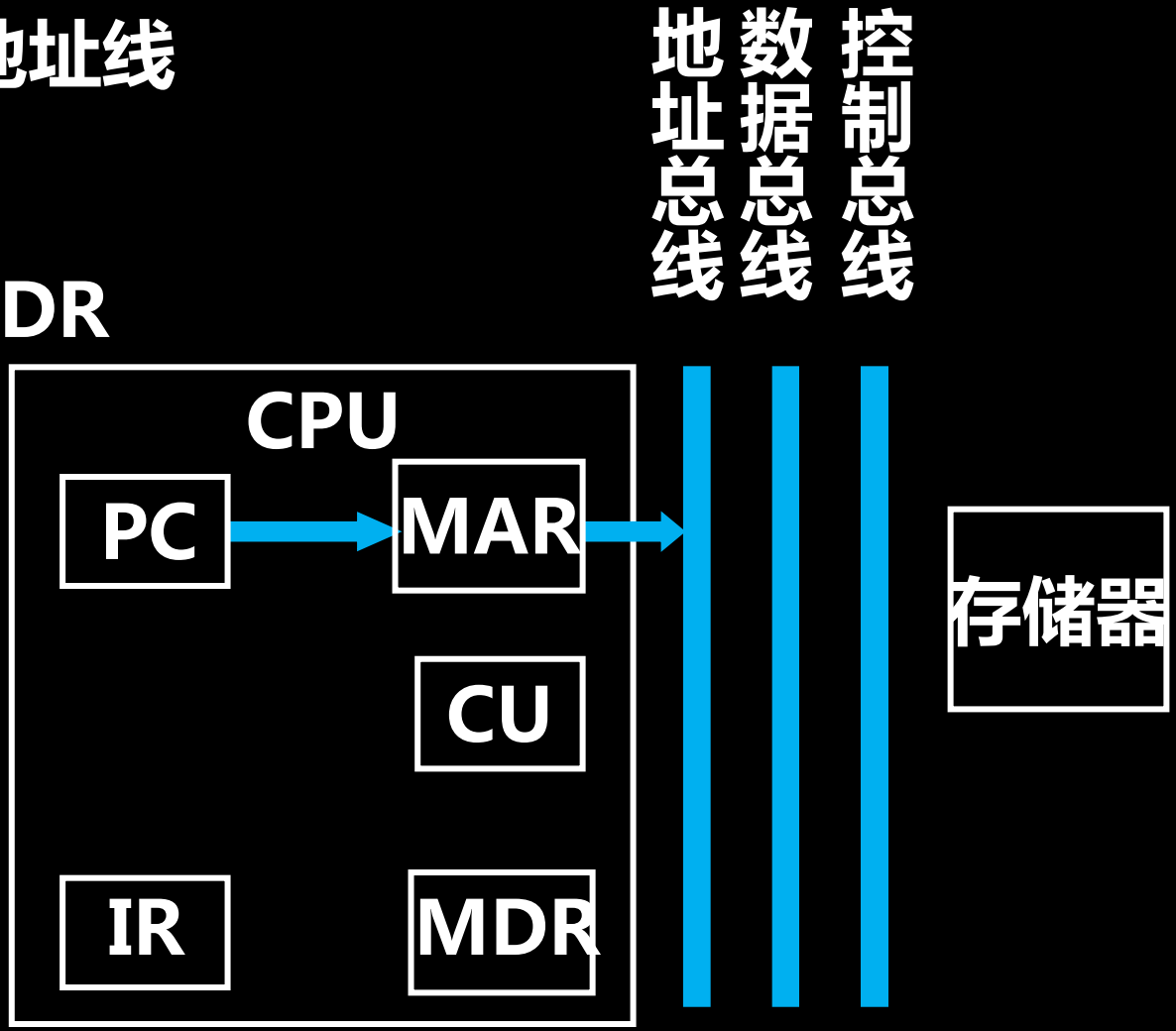
1 → R

M(MAR)→MDR

MDR→IR

OP(IR)→CU

(PC)+1→PC



取指周期的数据流

1. 取指周期

PC→MAR→地址线

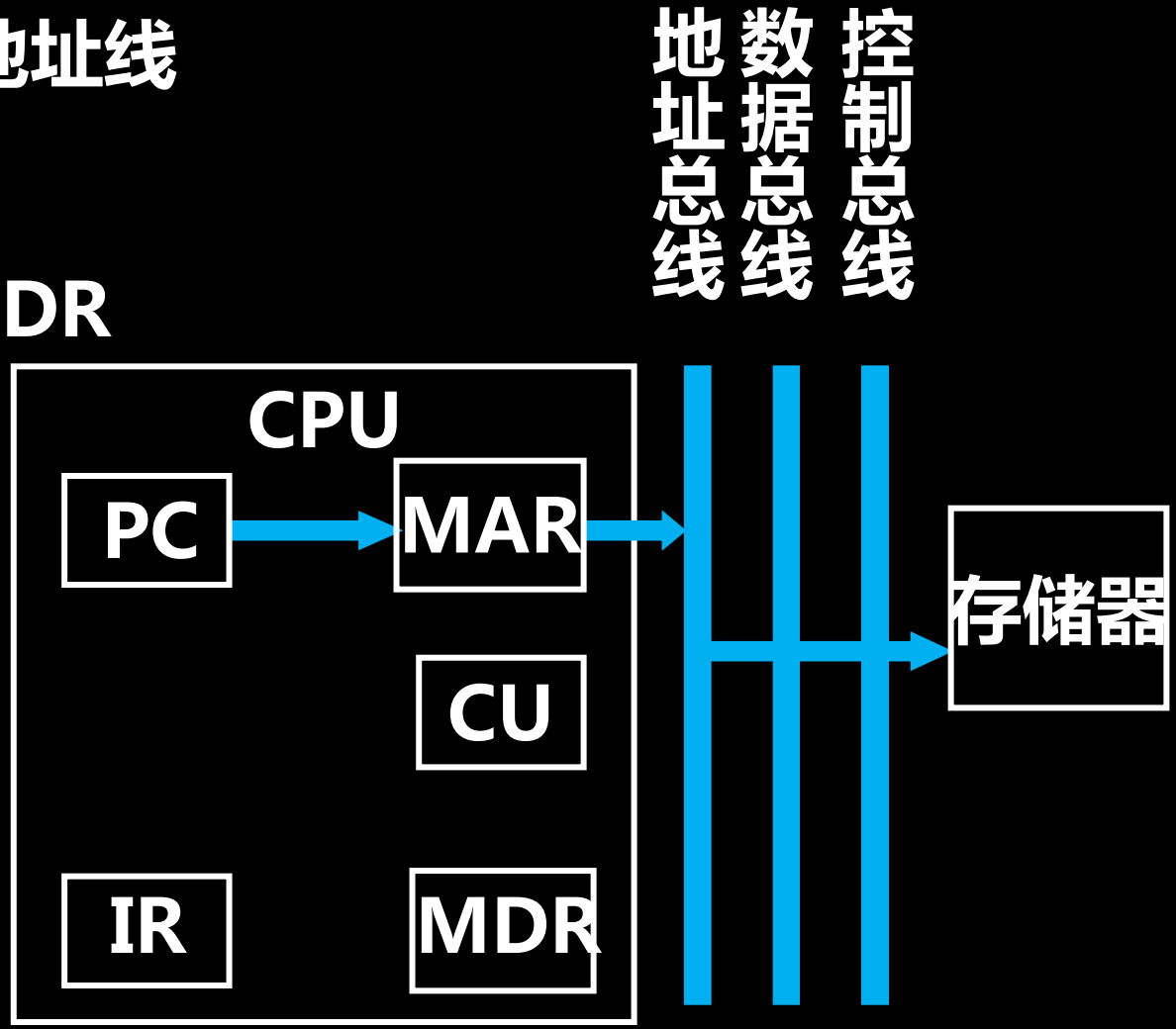
1 → R

M(MAR)→MDR

MDR→IR

OP(IR)→CU

(PC)+1→PC



取指周期的数据流

1. 取指周期

PC→MAR→地址线

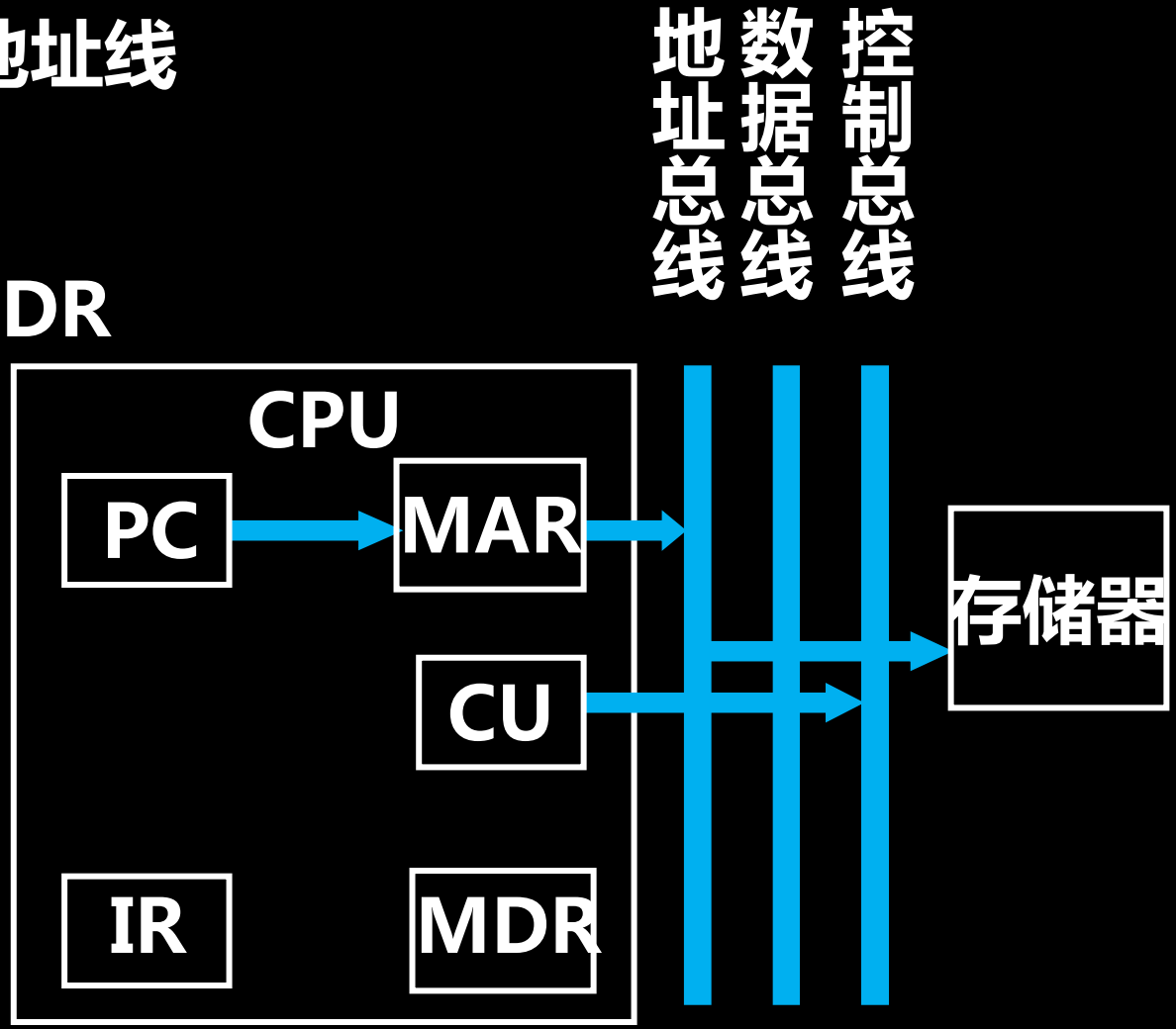
1 → R

M(MAR)→MDR

MDR→IR

OP(IR)→CU

(PC)+1→PC



取指周期的数据流

1. 取指周期

PC→MAR→地址线

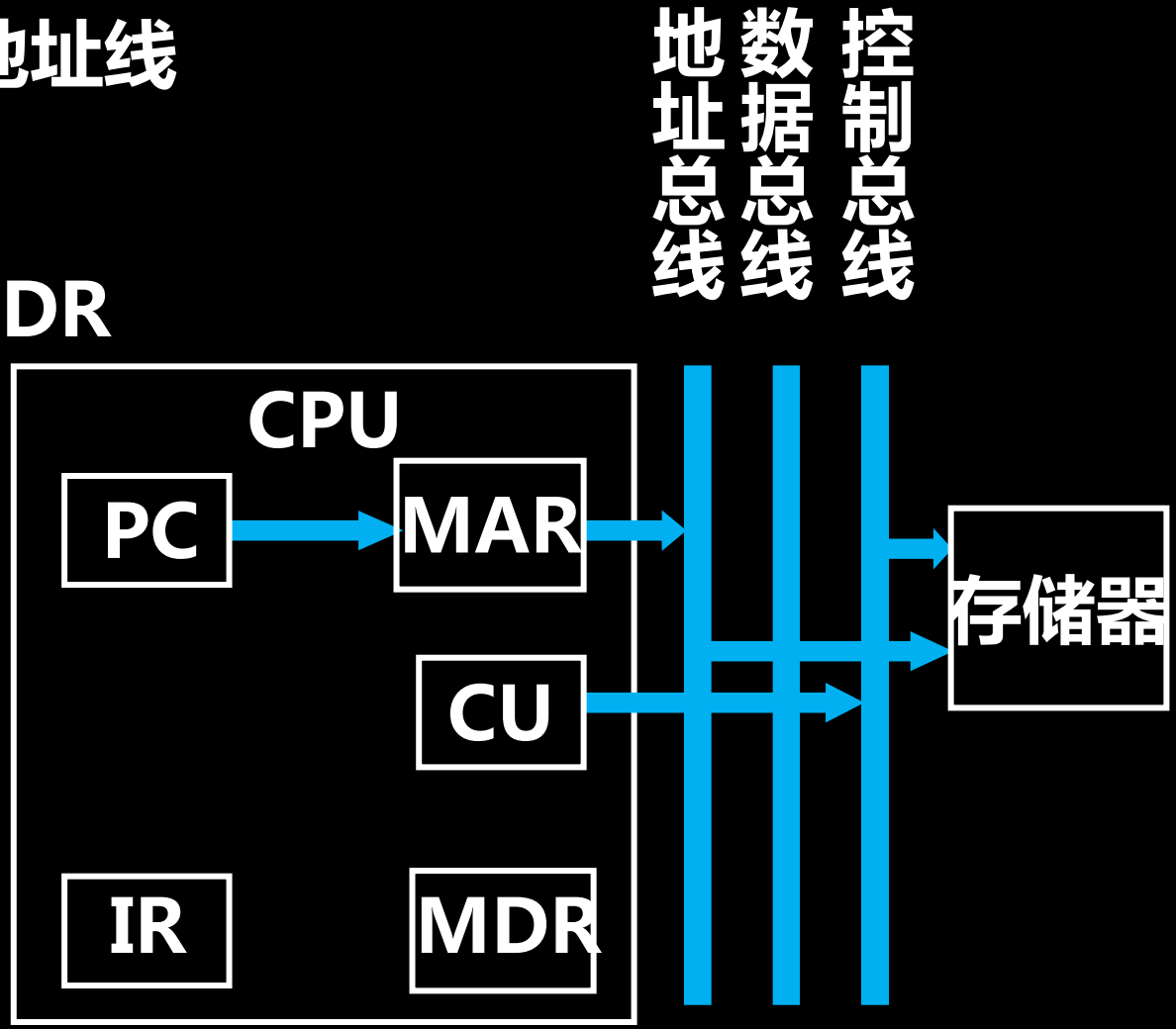
1 → R

M(MAR)→MDR

MDR→IR

OP(IR)→CU

(PC)+1→PC



取指周期的数据流

1. 取指周期

PC→MAR→地址线

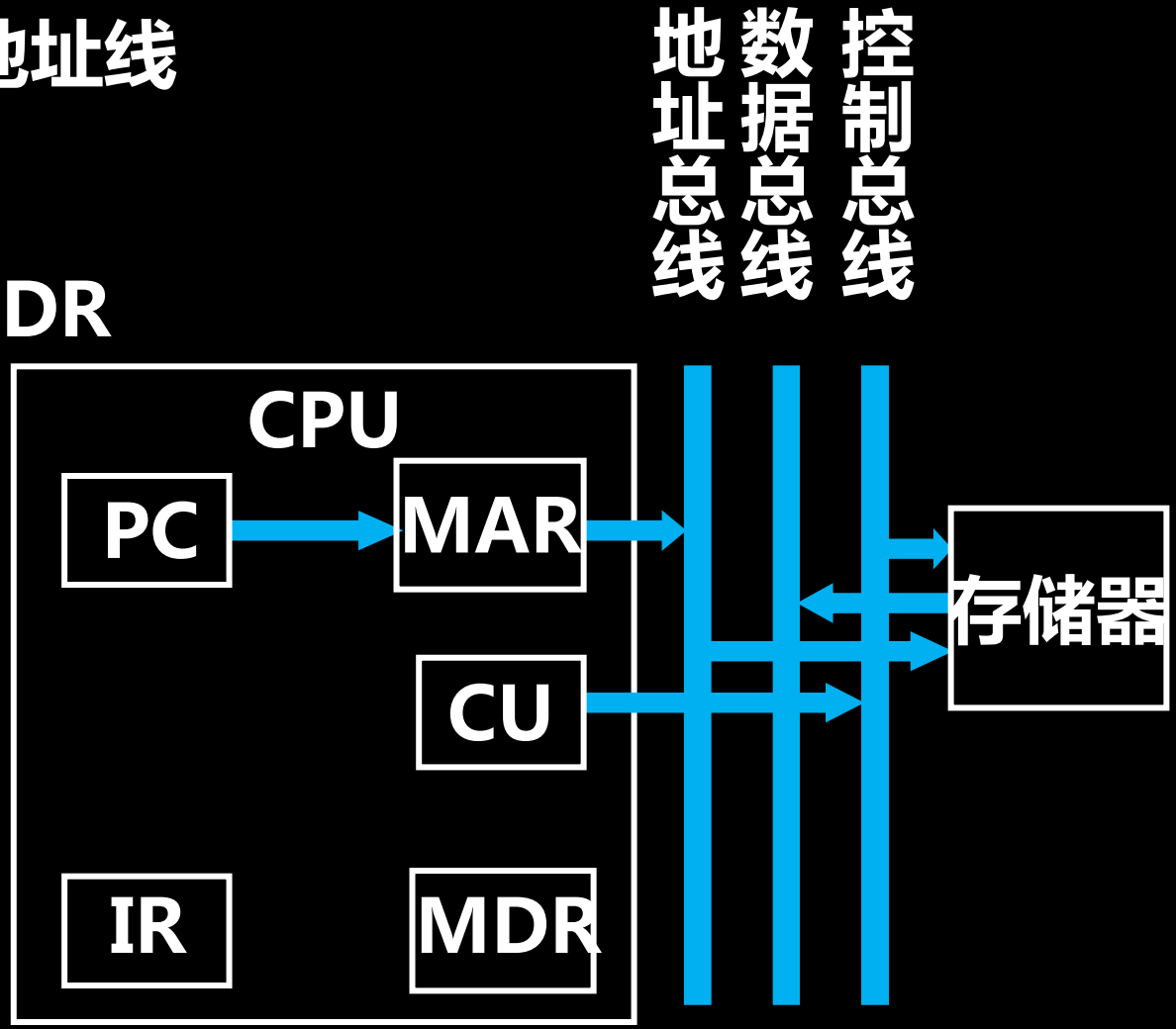
1 → R

M(MAR)→MDR

MDR→IR

OP(IR)→CU

(PC)+1→PC



取指周期的数据流

1. 取指周期

PC→MAR→地址线

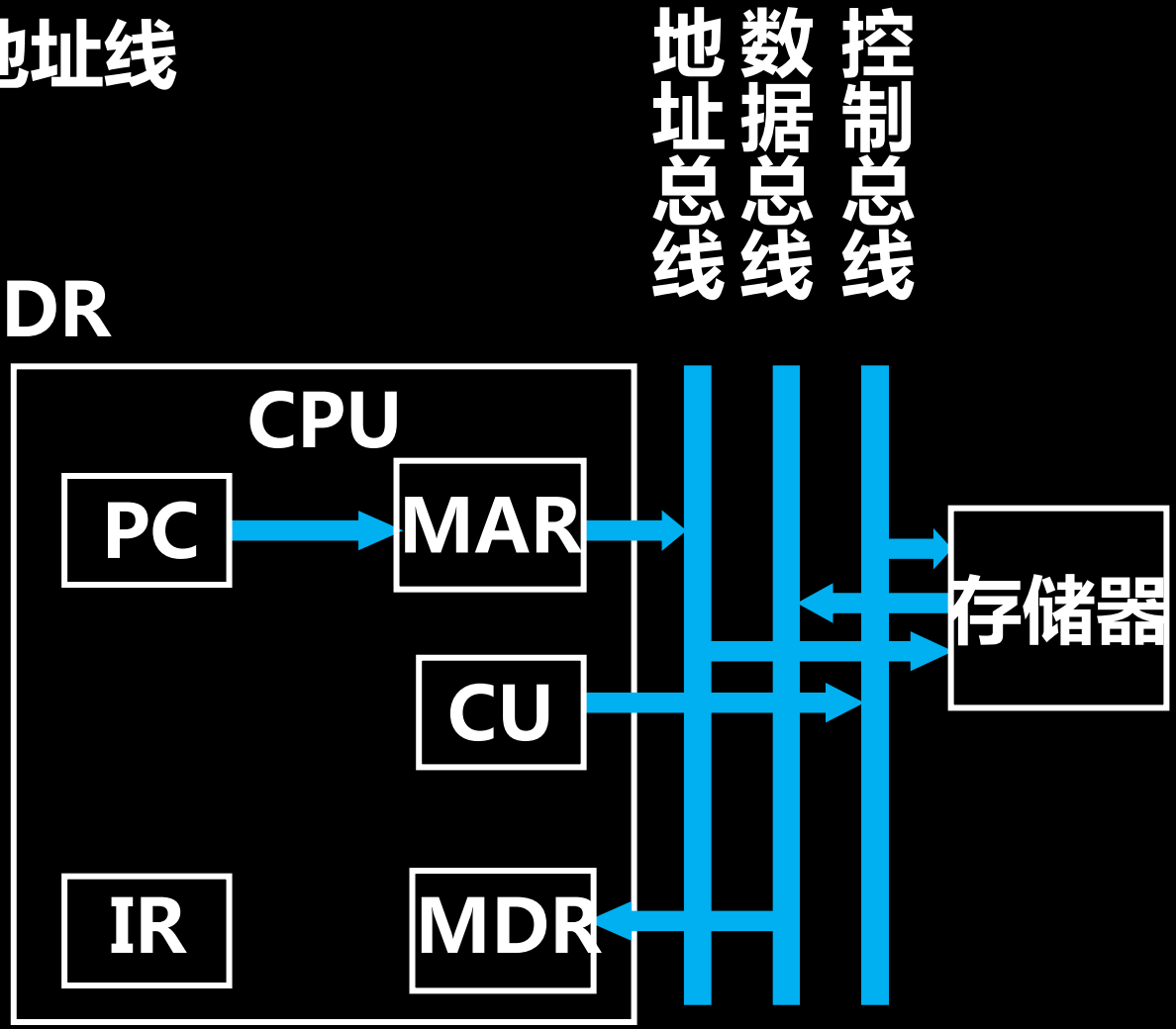
1 → R

M(MAR)→MDR

MDR→IR

OP(IR)→CU

(PC)+1→PC



取指周期的数据流

1. 取指周期

PC→MAR→地址线

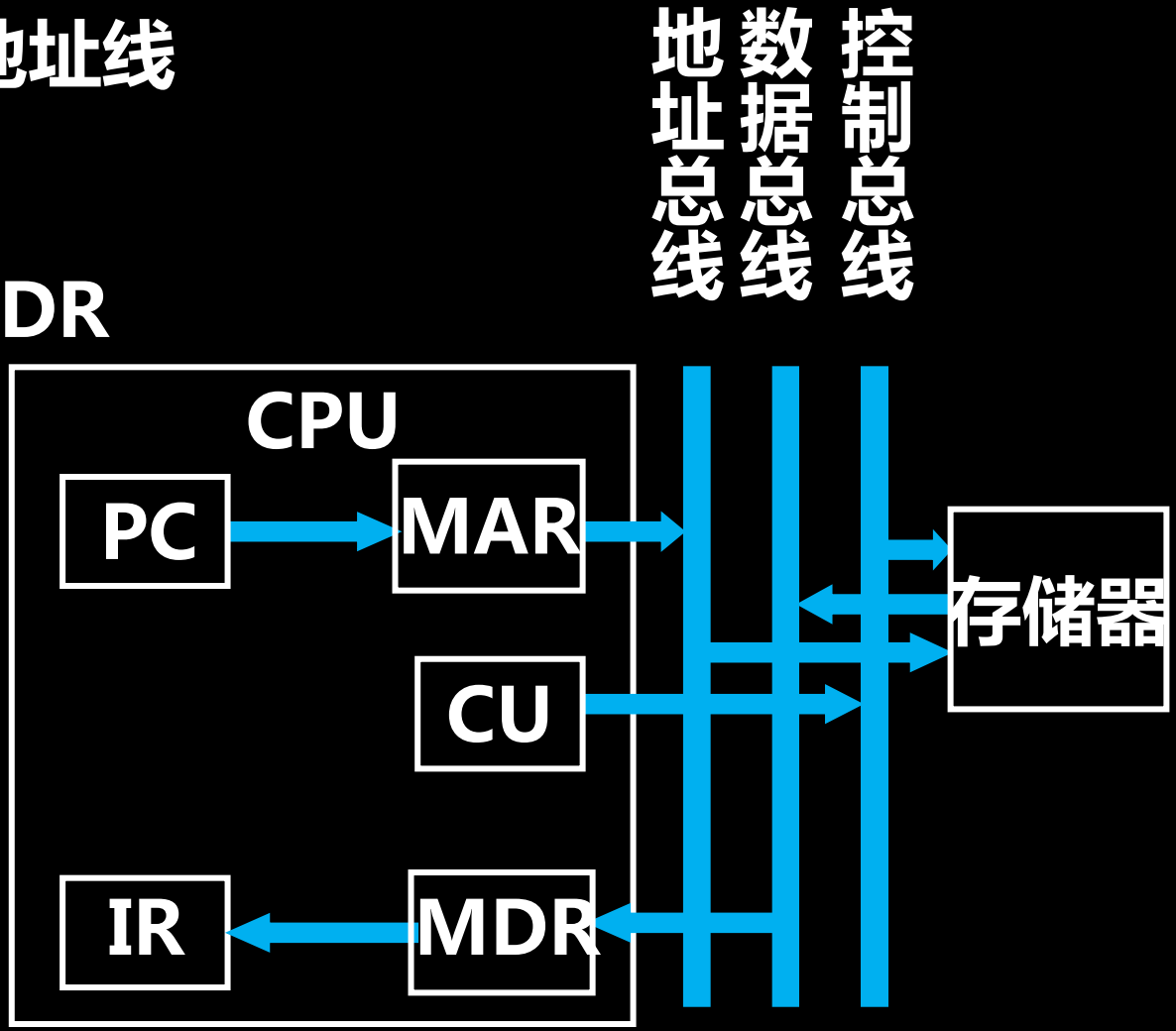
1 → R

M(MAR)→MDR

MDR→IR

OP(IR)→CU

(PC)+1→PC



取指周期的数据流

1. 取指周期

PC→MAR→地址线

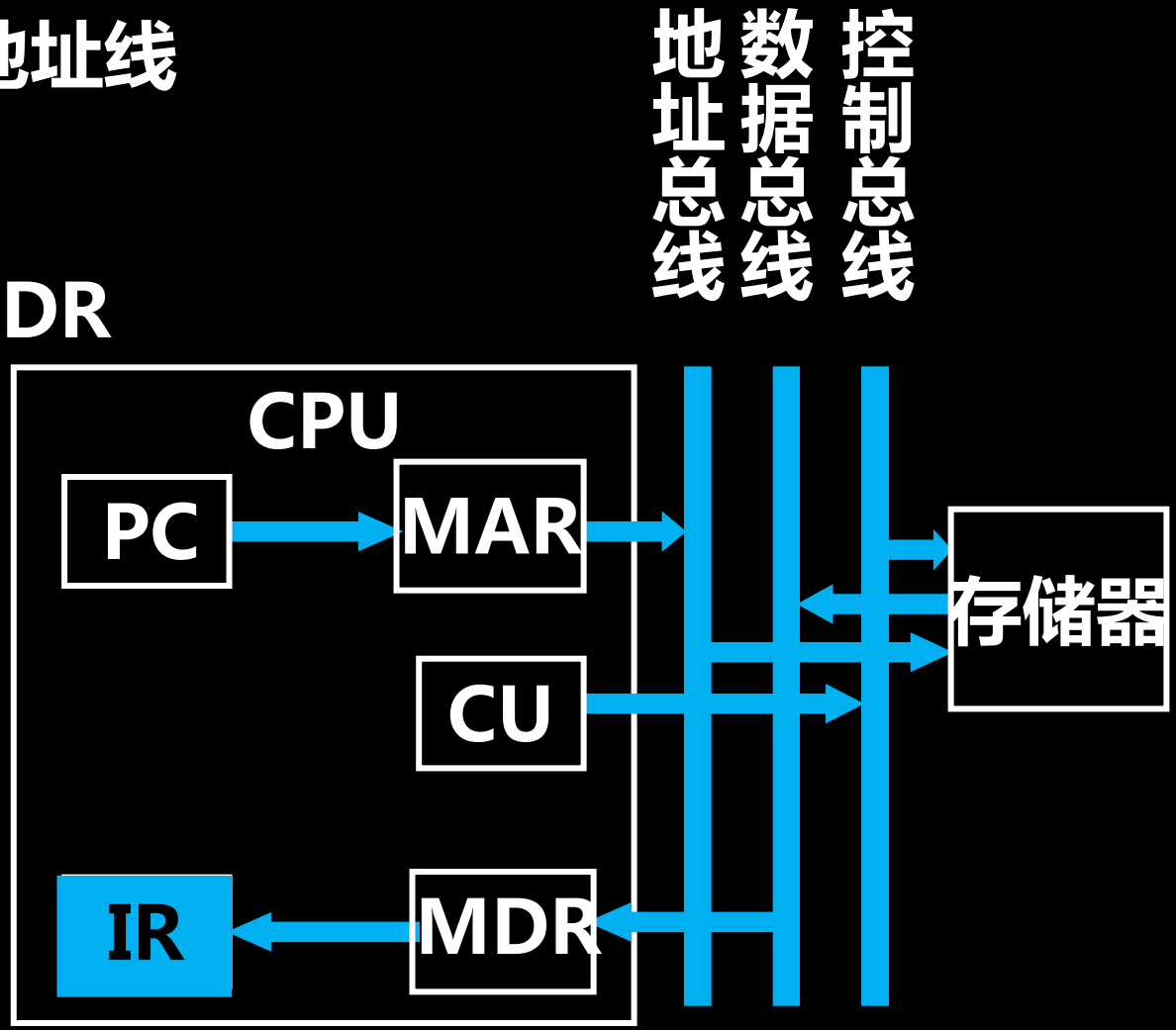
1 → R

M(MAR)→MDR

MDR→IR

OP(IR)→CU

(PC)+1→PC



取指周期的数据流

1. 取指周期

PC→MAR→地址线

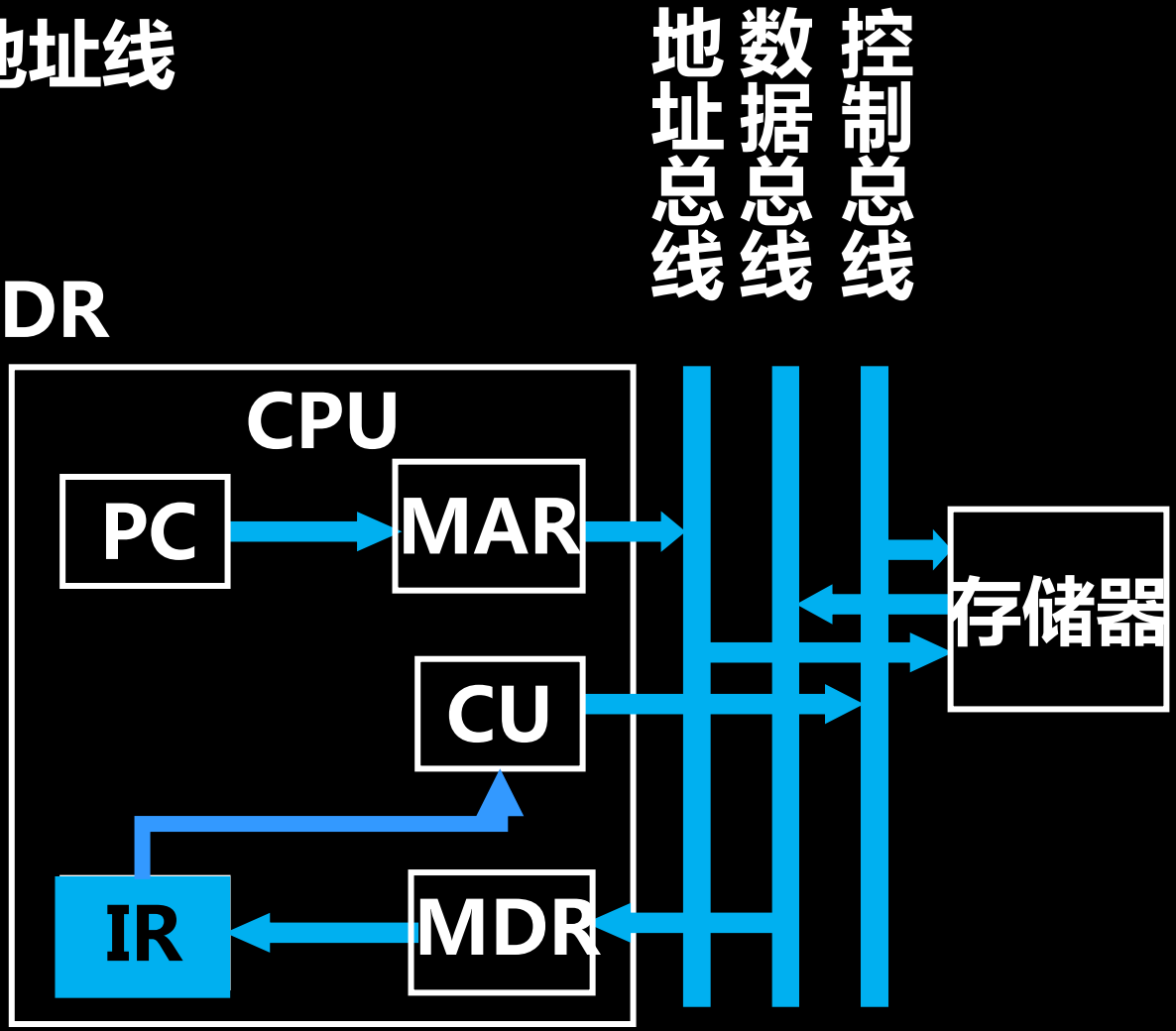
1 → R

M(MAR)→MDR

MDR→IR

OP(IR)→CU

(PC)+1→PC



取指周期的数据流

1. 取指周期

PC→MAR→地址线

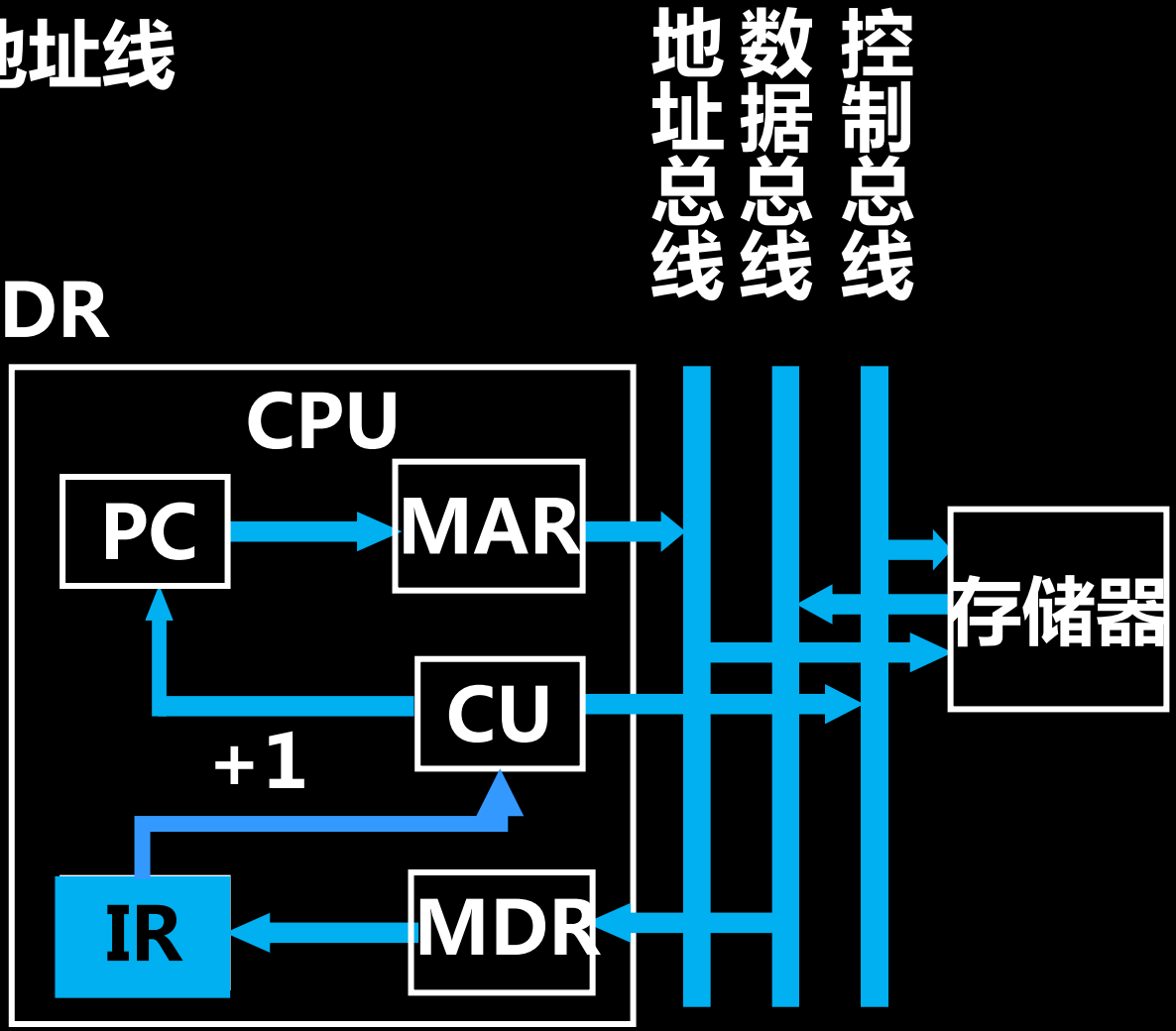
1 → R

M(MAR)→MDR

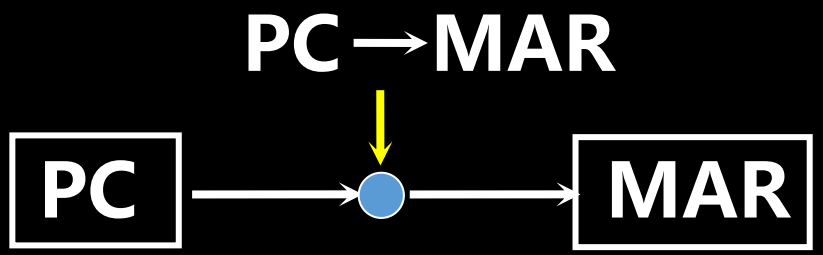
MDR→IR

OP(IR)→CU

(PC)+1→PC

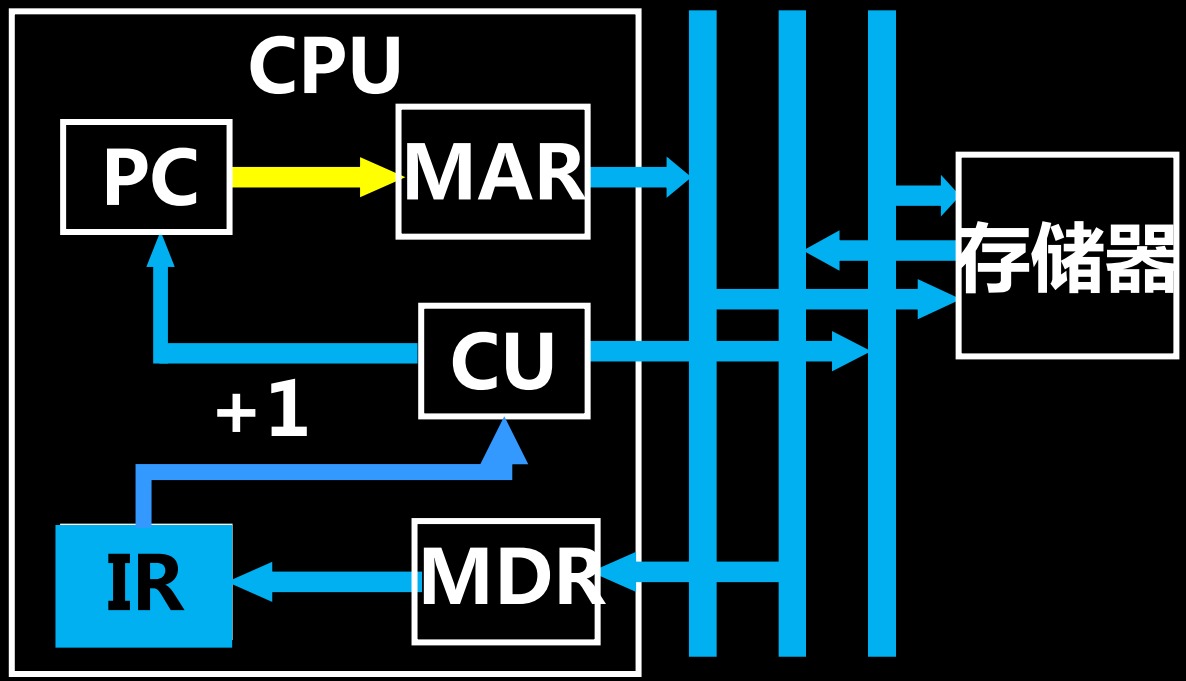


取指周期的操作命令



地址总线 数据总线 控制总线

操作命令
（微操作命令）
控制信号



间址周期的数据流

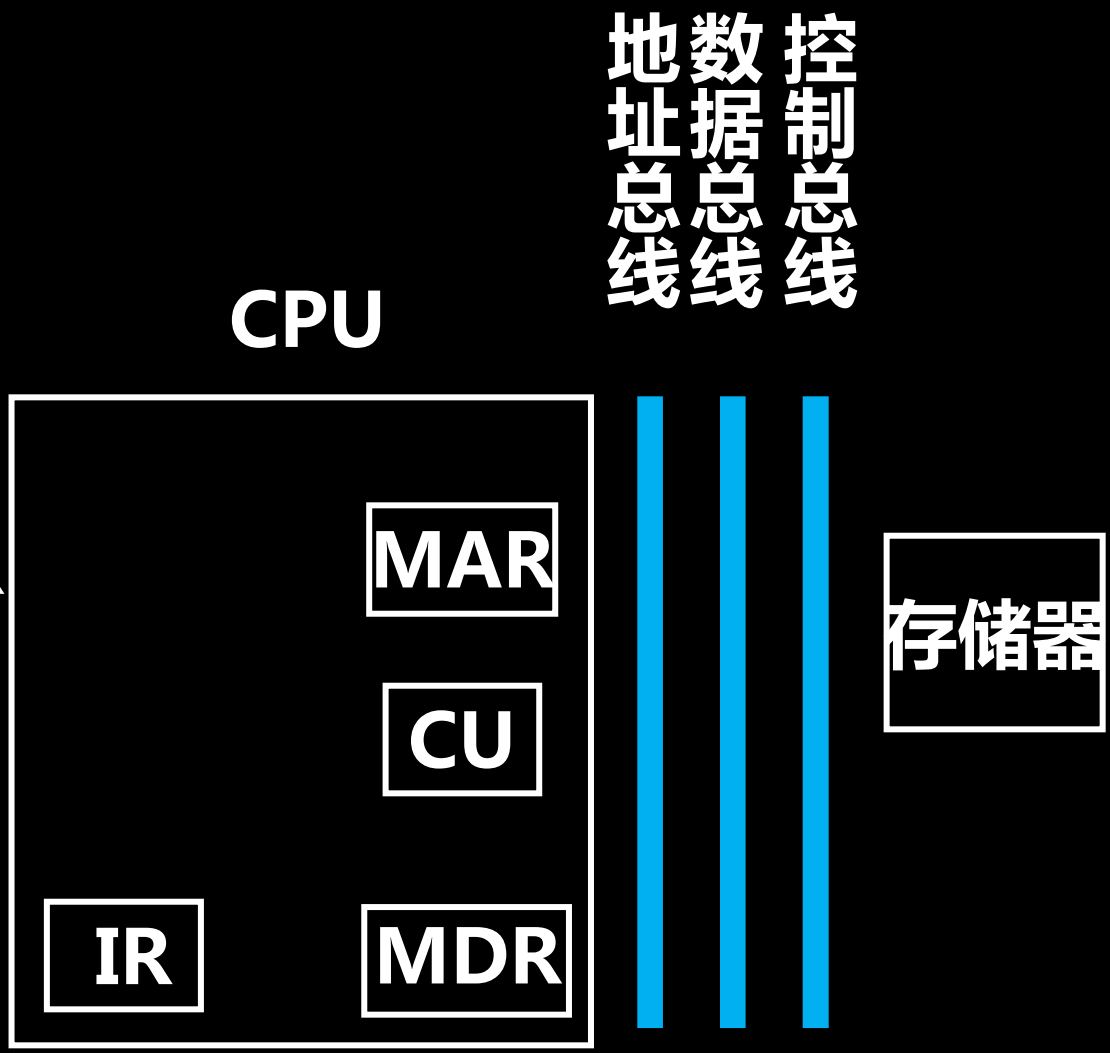
2. 间址周期

Ad(IR)→MAR

1→R

M(MAR)→MDR

MDR→ Ad(IR)



间址周期的数据流

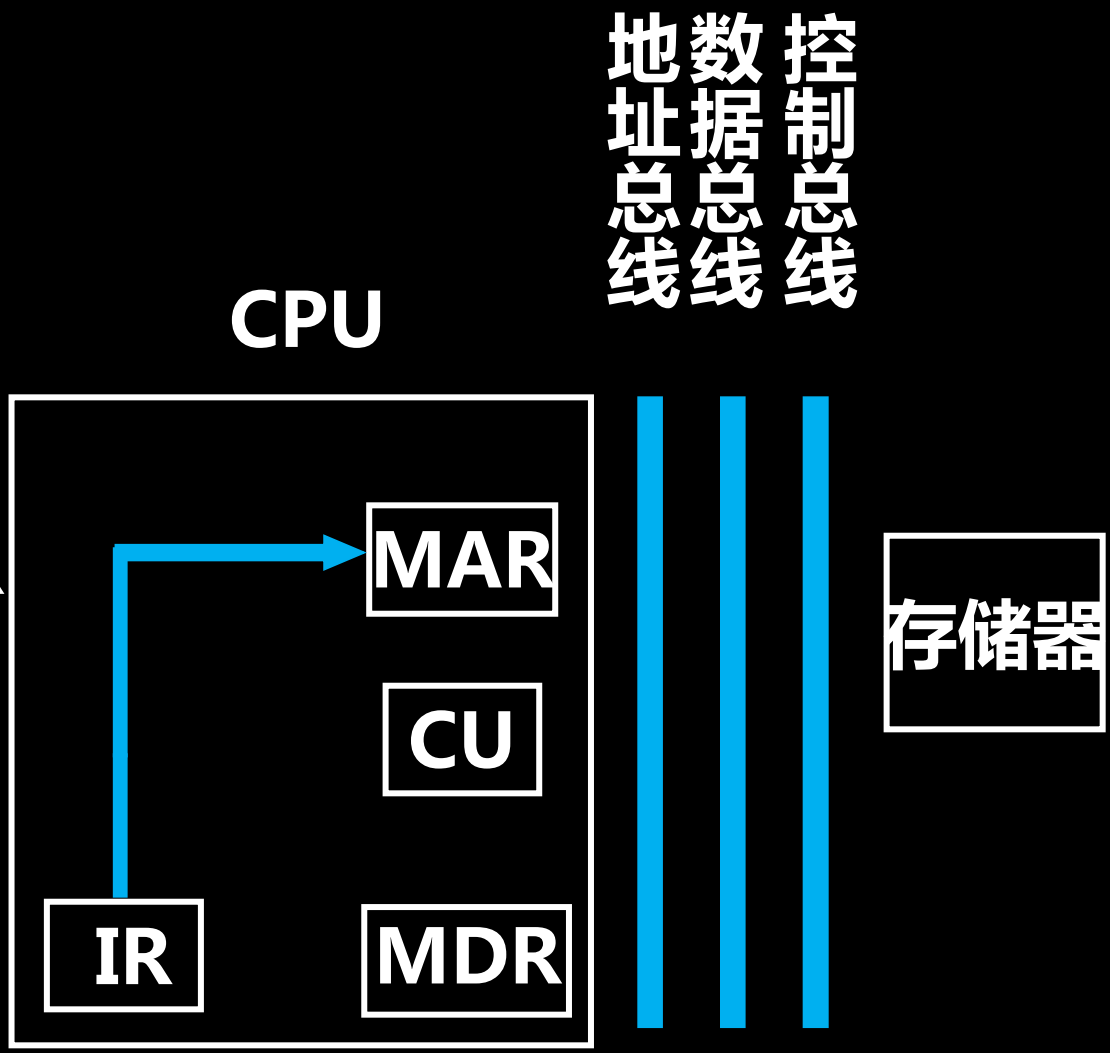
2. 间址周期

Ad(IR)→MAR

1→R

M(MAR)→MDR

MDR→ Ad(IR)



间址周期的数据流

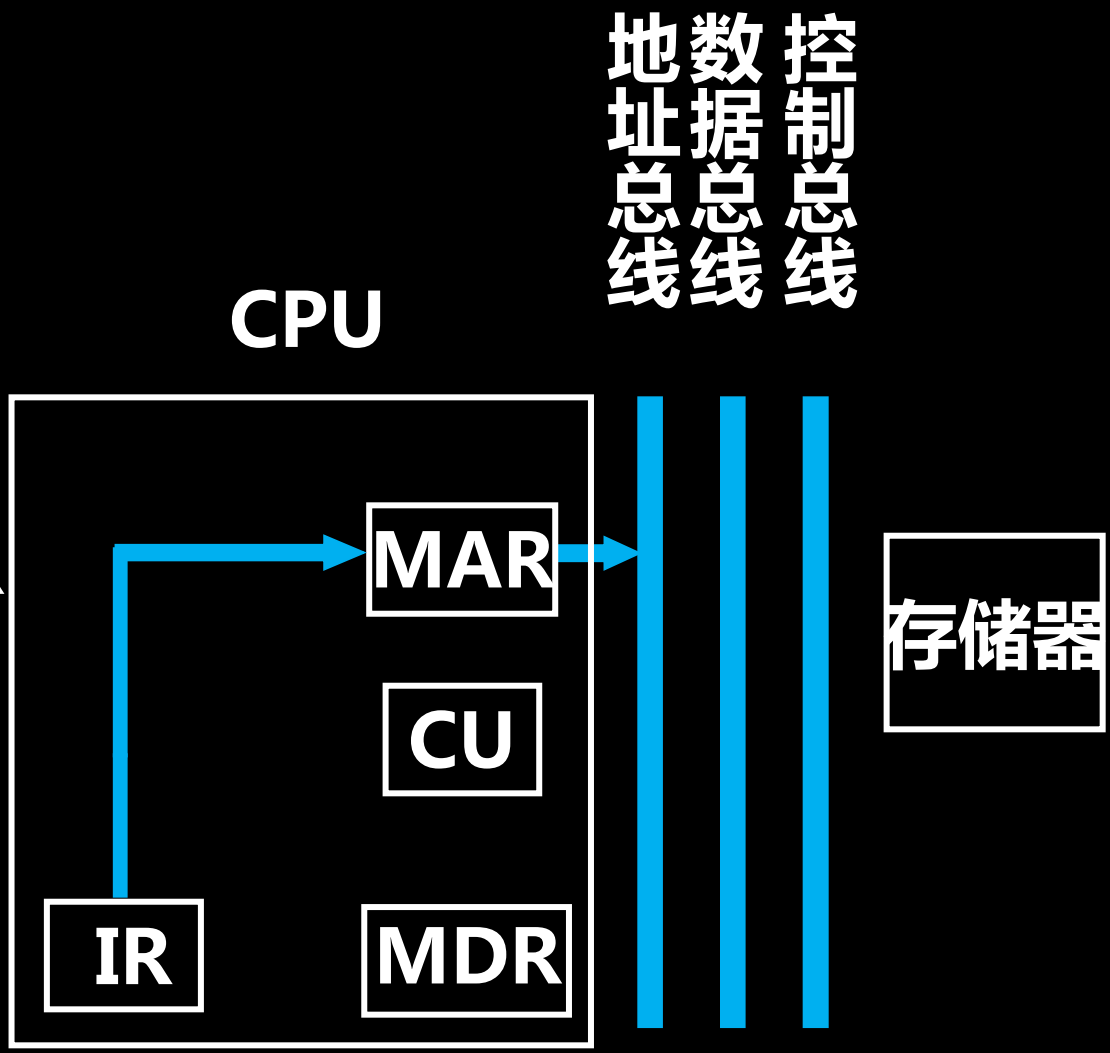
2. 间址周期

Ad(IR)→MAR

1→R

M(MAR)→MDR

MDR→ Ad(IR)



间址周期的数据流

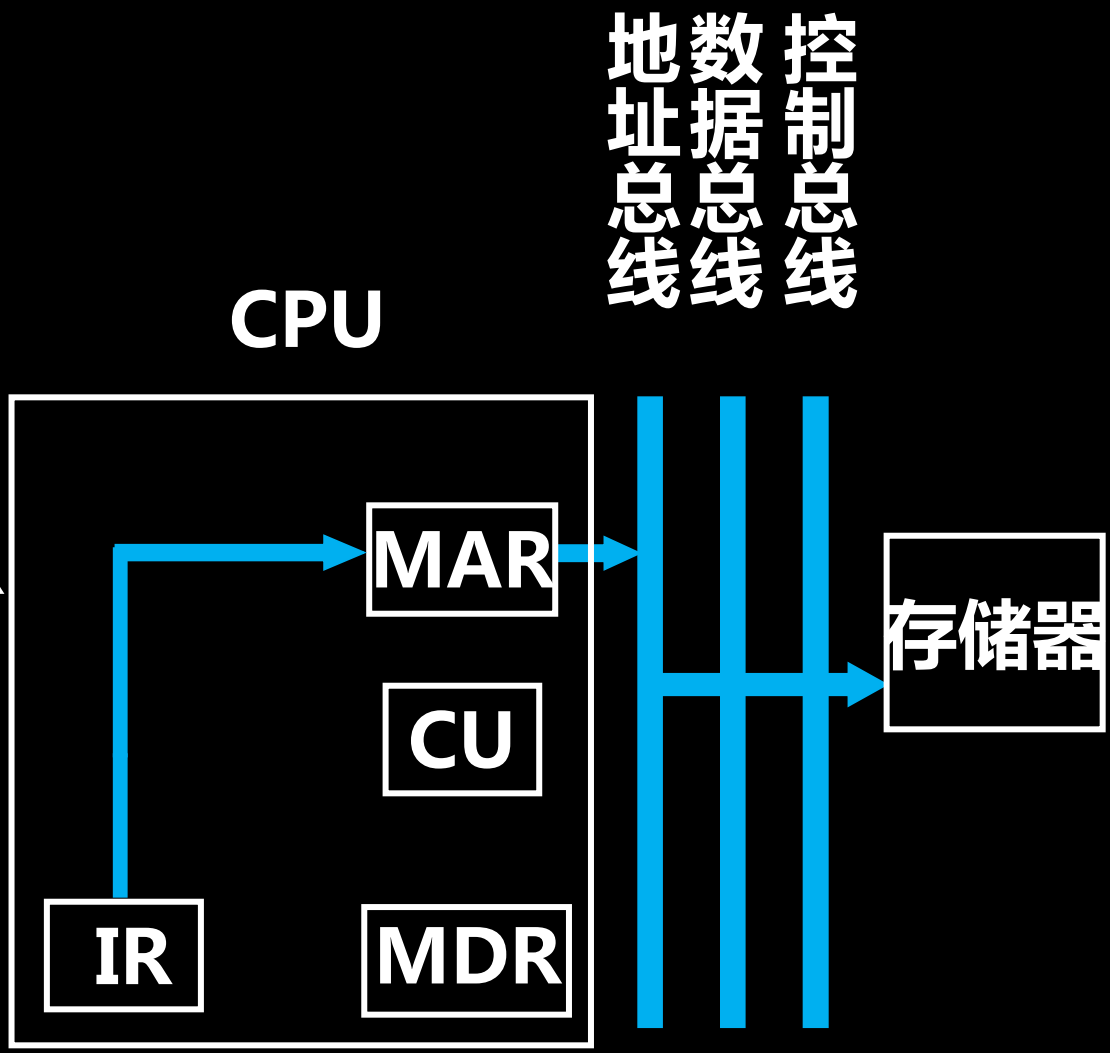
2. 间址周期

Ad(IR)→MAR

1→R

M(MAR)→MDR

MDR→ Ad(IR)



间址周期的数据流

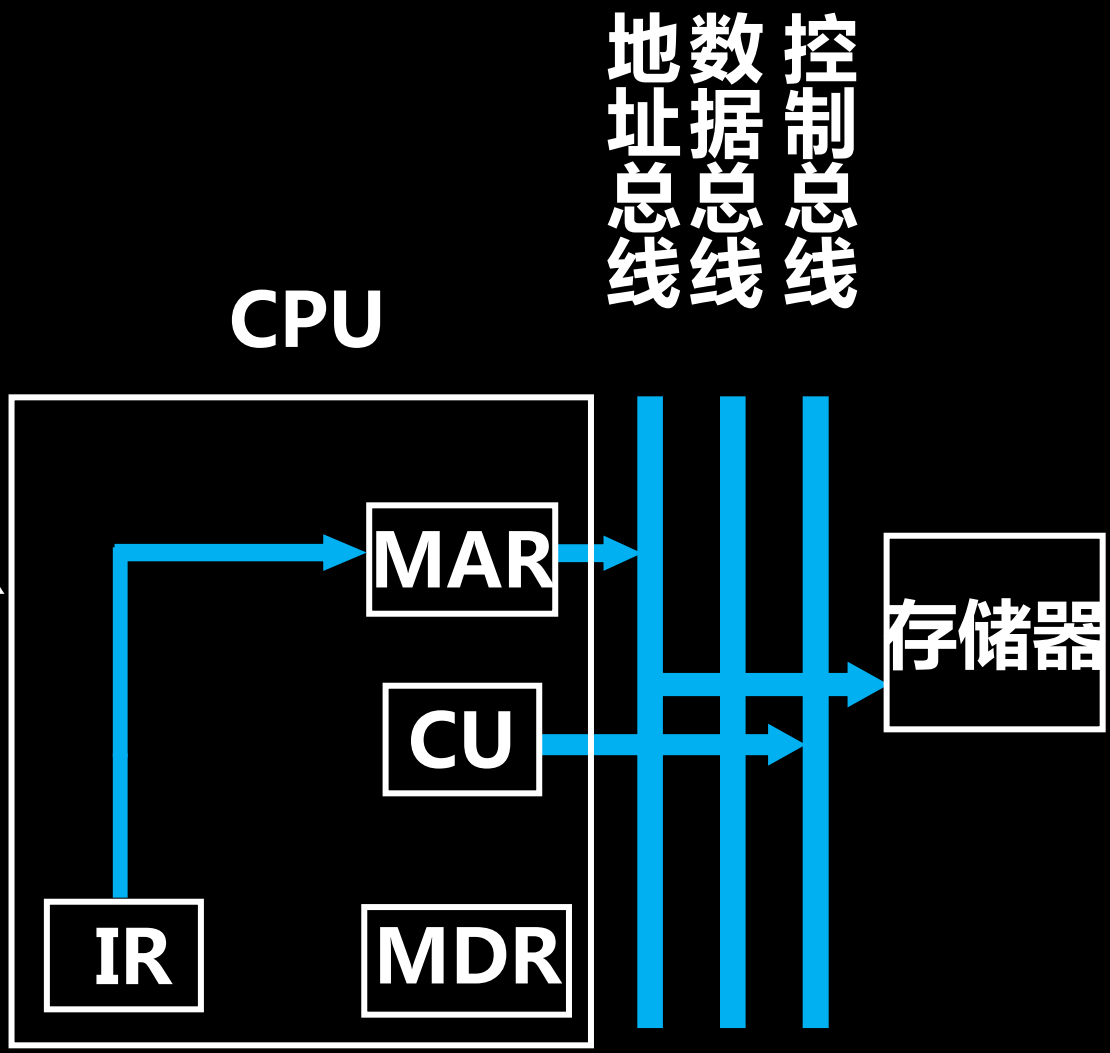
2. 间址周期

Ad(IR)→MAR

1→R

M(MAR)→MDR

MDR→ Ad(IR)



间址周期的数据流

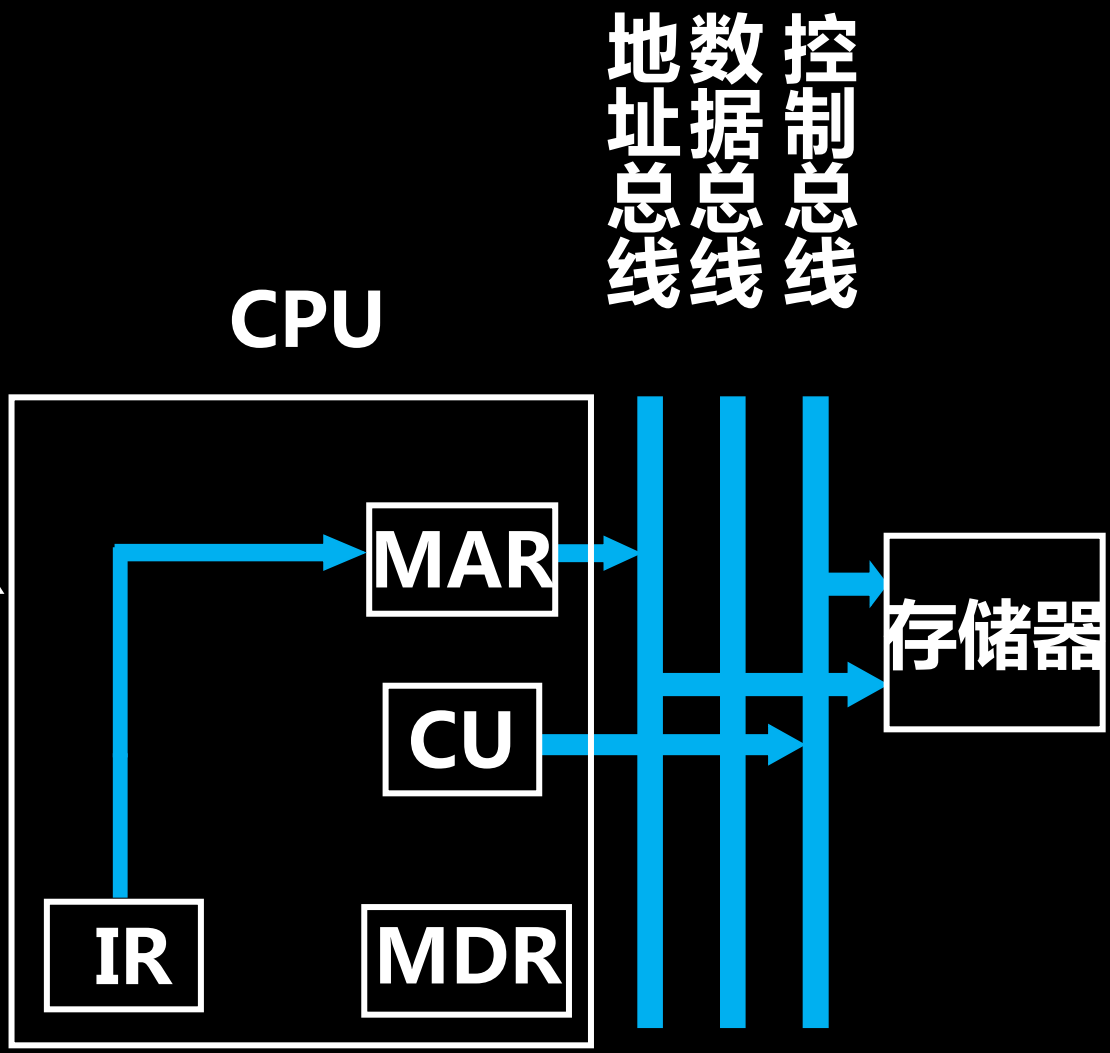
2. 间址周期

Ad(IR)→MAR

1→R

M(MAR)→MDR

MDR→ Ad(IR)



间址周期的数据流

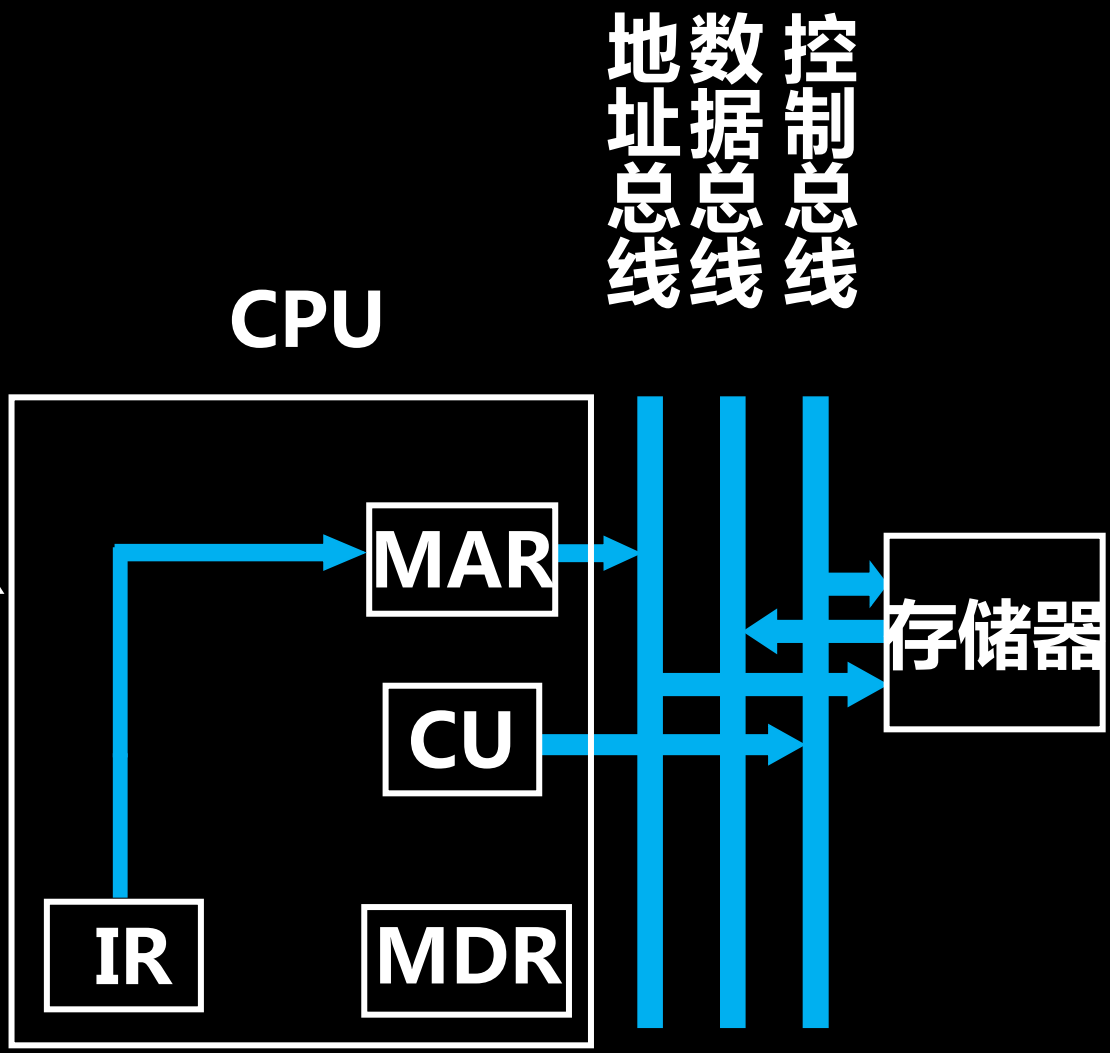
2. 间址周期

Ad(IR)→MAR

1→R

M(MAR)→MDR

MDR→ Ad(IR)



间址周期的数据流

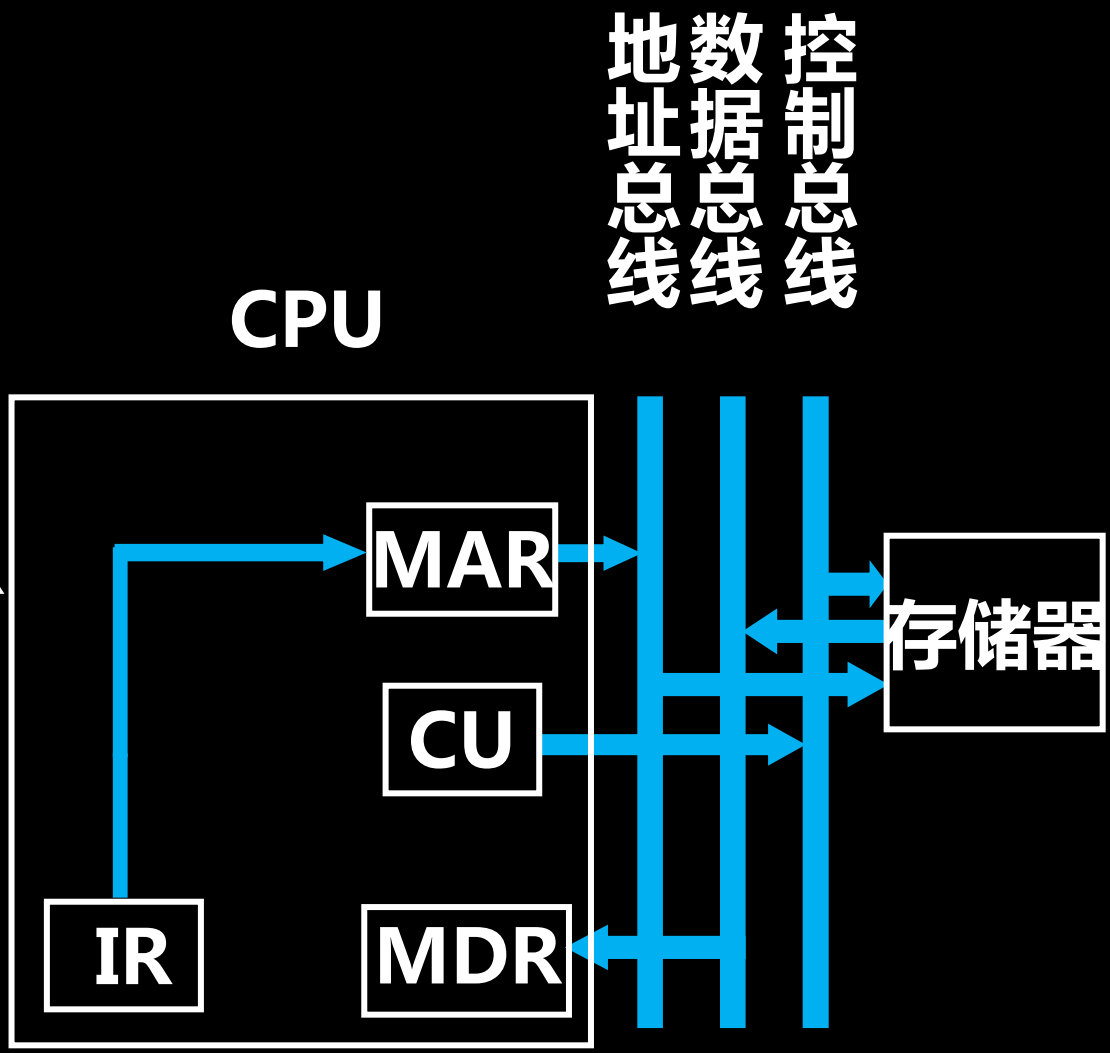
2. 间址周期

Ad(IR)→MAR

1→R

M(MAR)→MDR

MDR→ Ad(IR)



间址周期的数据流

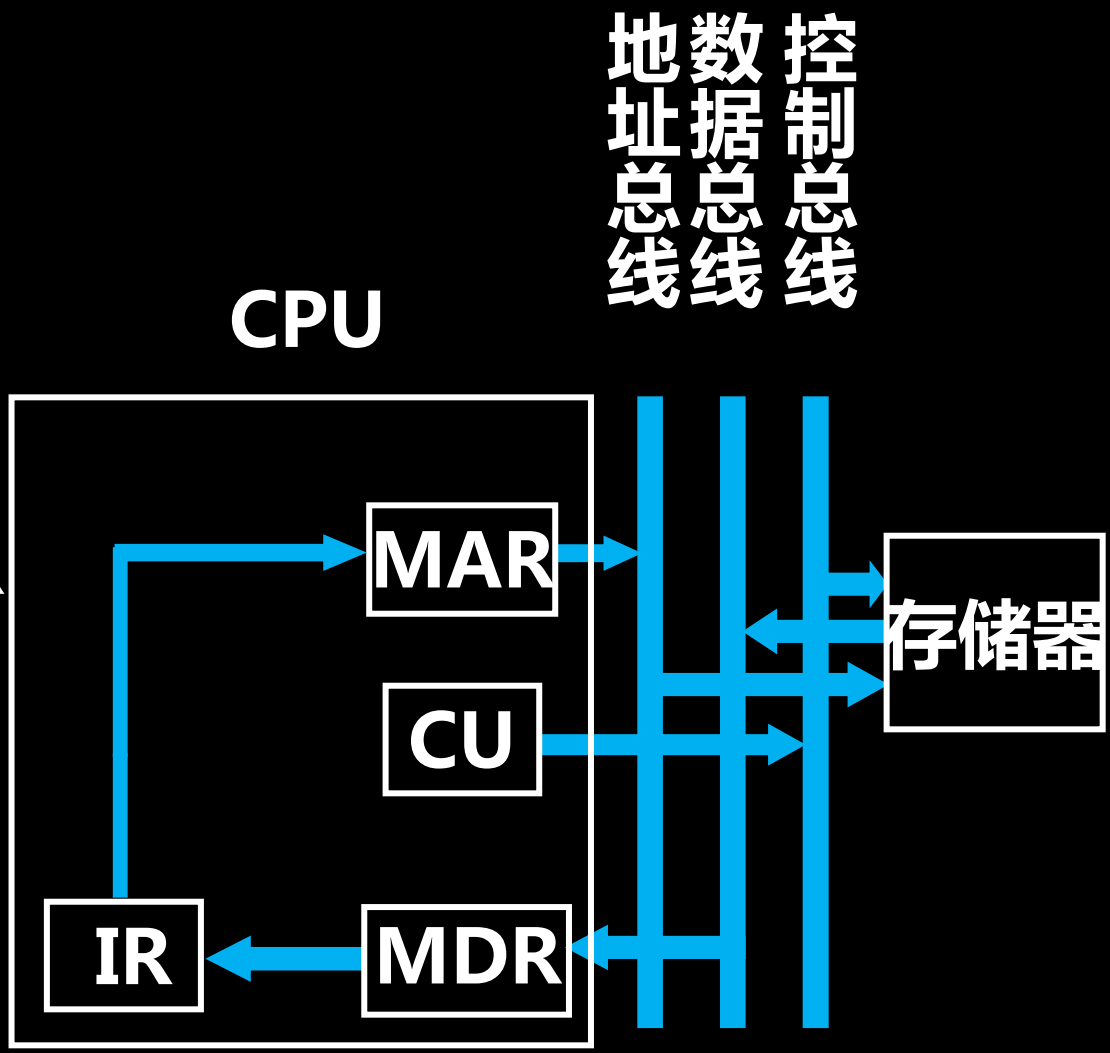
2. 间址周期

Ad(IR)→MAR

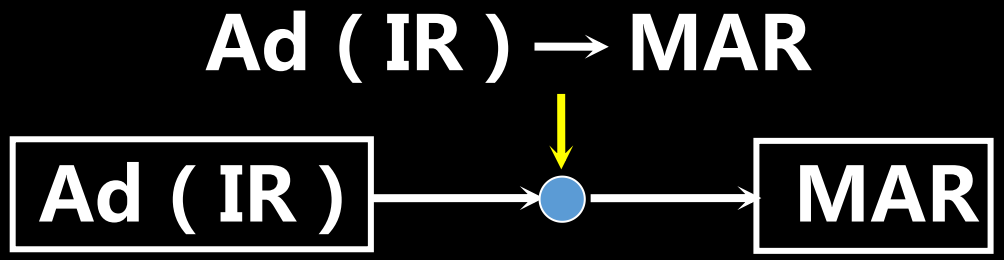
1→R

M(MAR)→MDR

MDR→ Ad(IR)



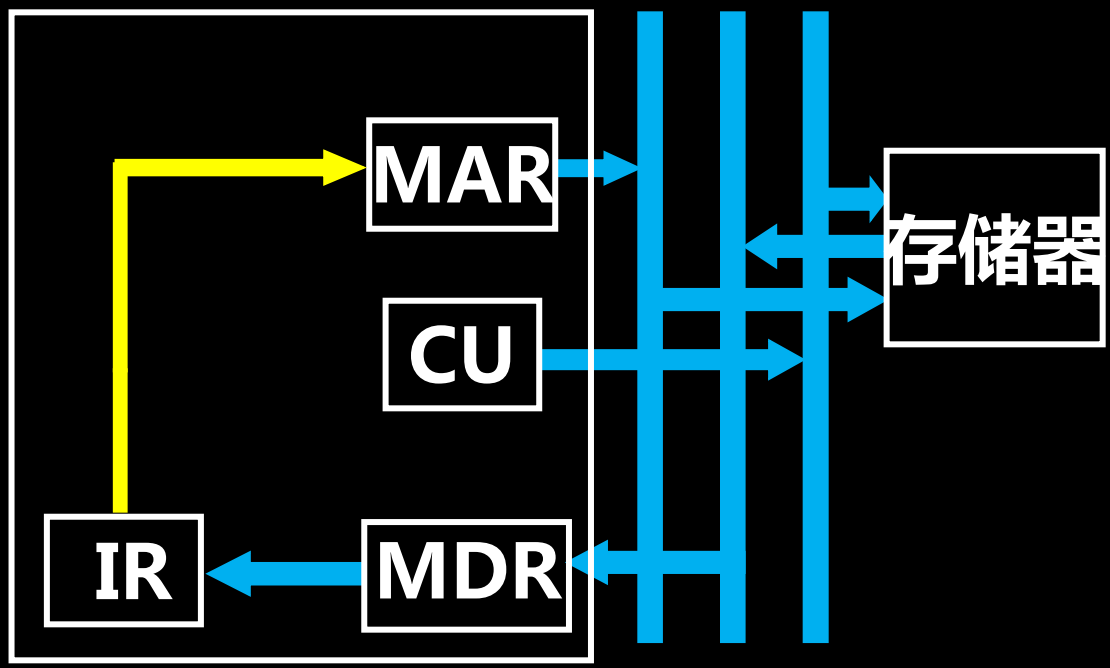
间址周期的操作命令



地址总线 数据总线 控制总线

CPU

操作命令
（微操作命令）
控制信号



执行周期的操作命令分析

3. 执行周期

(1) 非访存指令

① CLA　清A　　0 → ACC

② COM　取反　　$\overline{ACC}$ → ACC

③ SHR　算术右移　L(ACC)→ R(ACC), ACC_0→ ACC_0

④ CSL　循环左移　R(ACC)→ L(ACC)，ACC_0→ACC_n

⑤ STP　停机指令　0 → G

执行周期的操作命令分析

(2) 访存指令

① 加法指令　ADD　X

Ad(IR) ⟶ MAR

1 ⟶ R

M(MAR) ⟶MDR

(ACC) + (MDR) ⟶ ACC

② 存数指令　STA　X

Ad(IR) ⟶MAR

1 ⟶ W

ACC ⟶MDR

MDR ⟶M(MAR)

执行周期的操作命令分析

③ 取数指令　　LDA　X

Ad (IR) →　MAR

1 →　R

M (MAR) → MDR

MDR → ACC

(3) 转移指令

① 无条件转　　JMP　X

Ad (IR) →　PC

② 条件转移　　BAN　X　（负则转）

$A_0 \cdot Ad(IR) + \bar{A}_0(PC) \rightarrow PC$

指令周期长度

(4) 三类指令的指令周期

非访存 指令周期 | 取指周期 | 执行周期

直接访存 指令周期 | 取指周期 | 执行周期

间接访存 指令周期 | 取指周期 | 间址周期 | 执行周期

转移 指令周期 | 取指周期 | 执行周期

间接转移 指令周期 | 取指周期 | 间址周期 | 执行周期

中断周期的数据流

0 → MAR

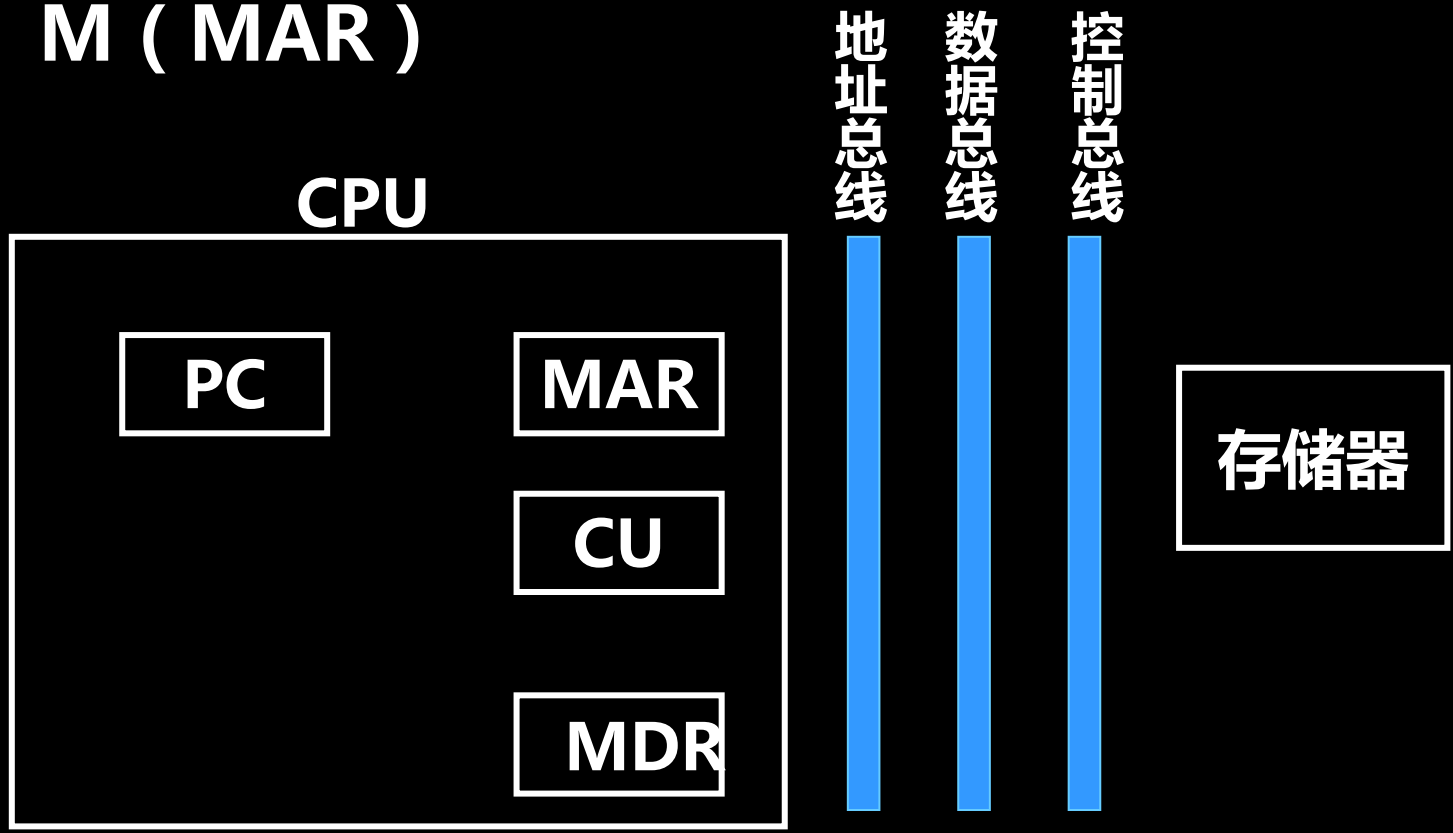
1 → W

PC → MDR

MDR → M (MAR)

向量地址 → PC

0 → EINT (置“0”)



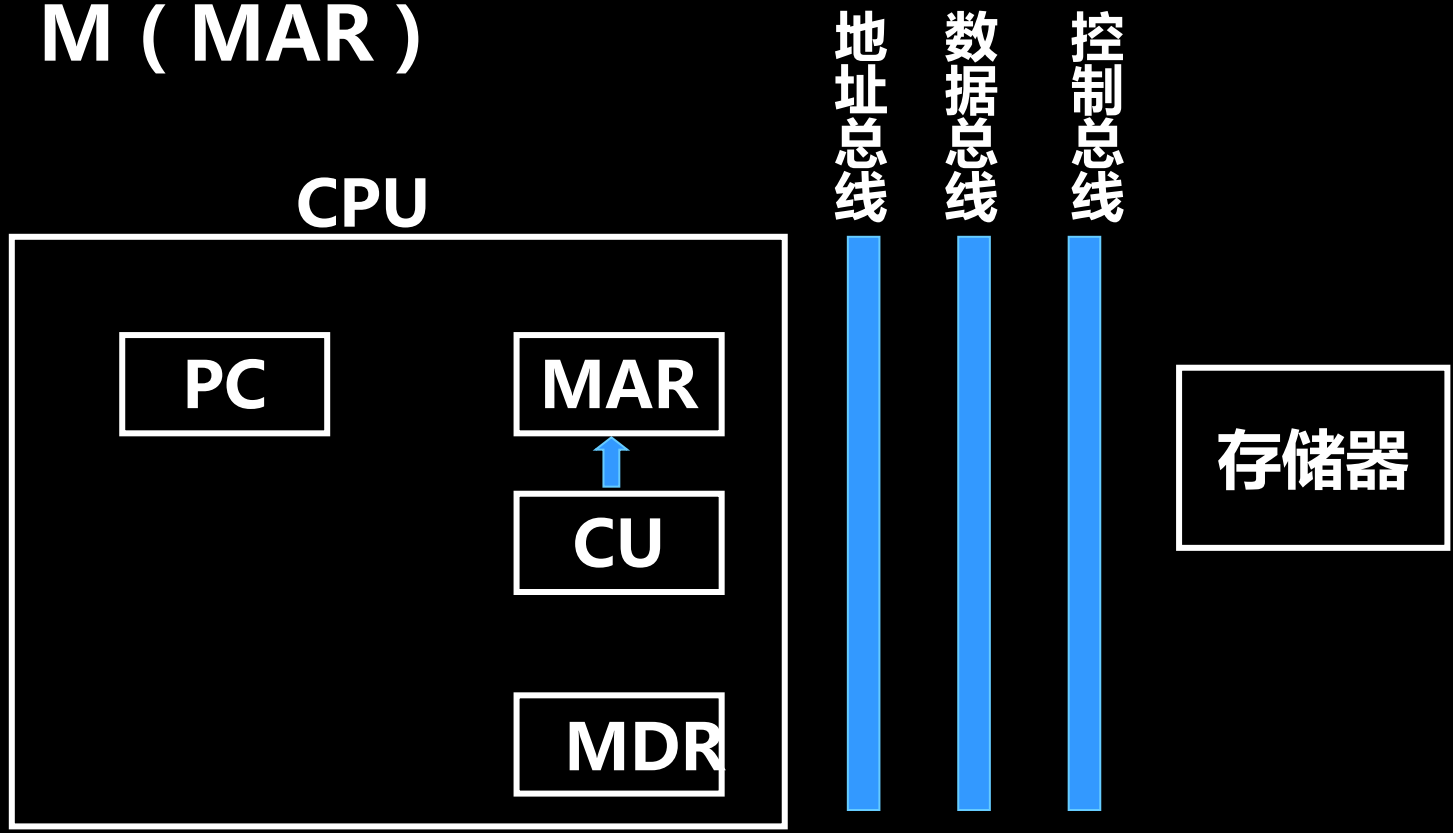
中断周期的数据流

0 → MAR　　向量地址 → PC

1 → W　　0 → EINT（置“0”）

PC → MDR

MDR → M（MAR）



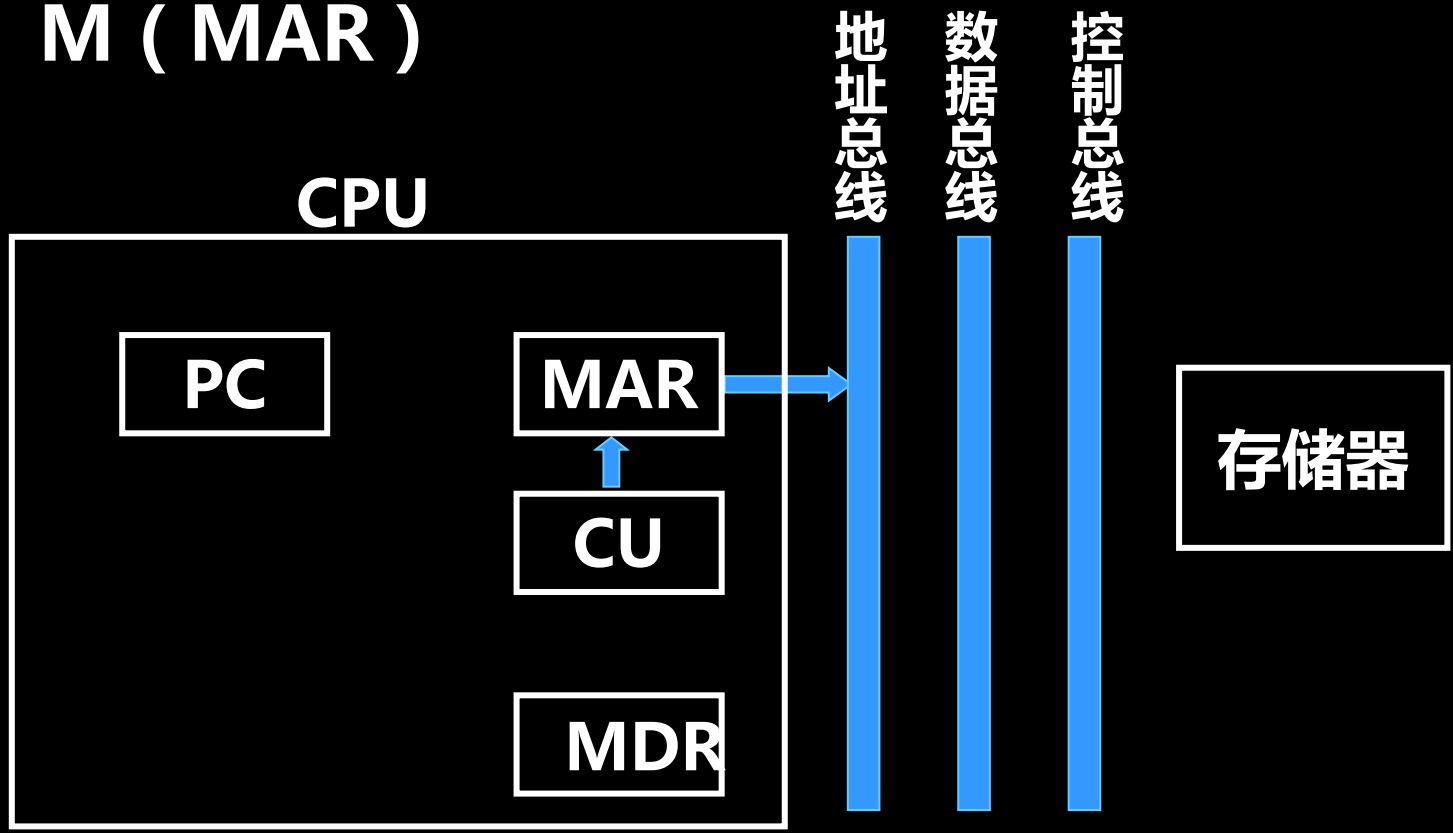
中断周期的数据流

0 → MAR　　向量地址 → PC

1 → W　　0 → EINT（置“0”）

PC → MDR

MDR → M（MAR）



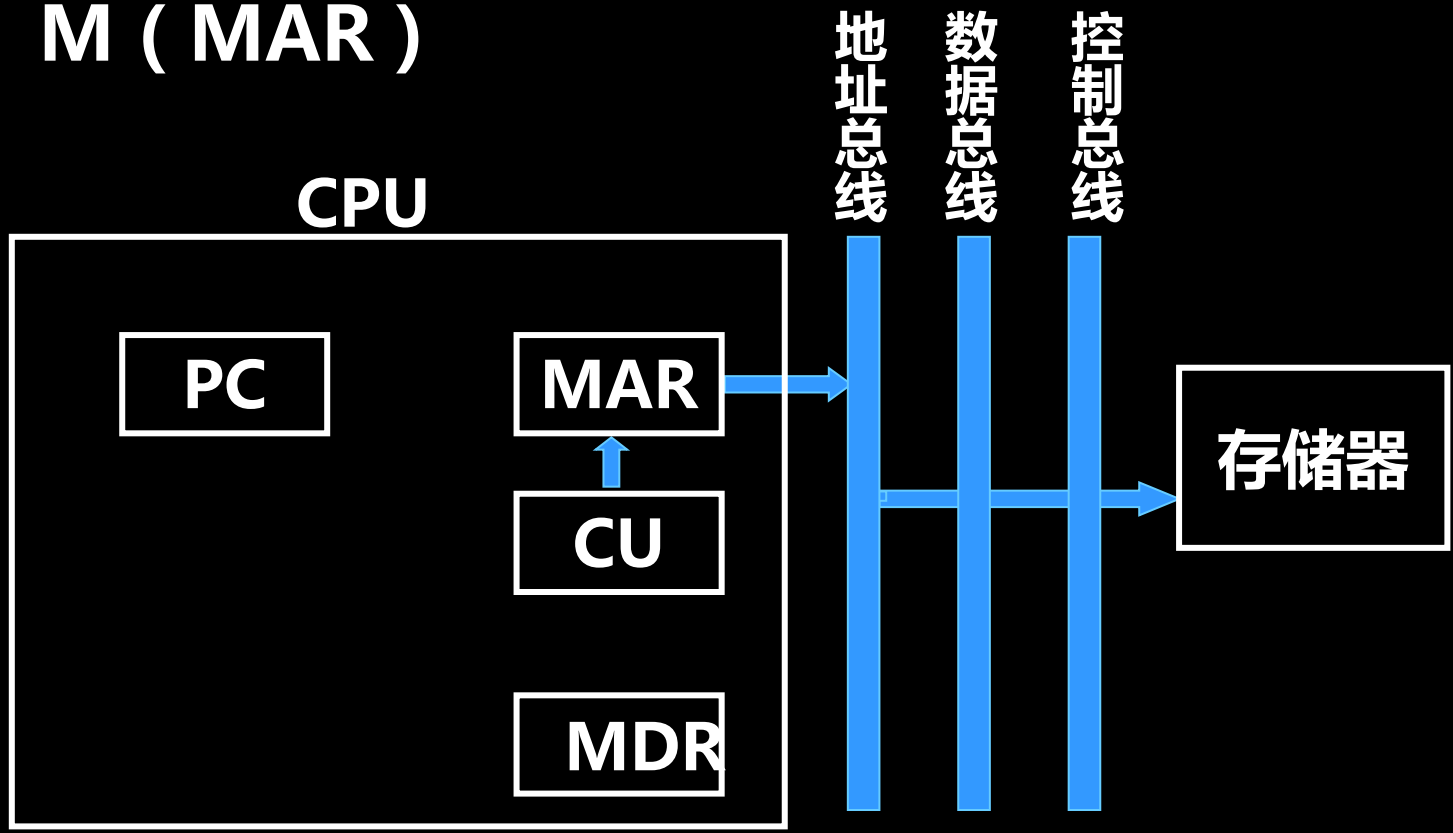
中断周期的数据流

0 → MAR　　向量地址 → PC

1 → W　　0 → EINT（置“0”）

PC → MDR

MDR → M（MAR）



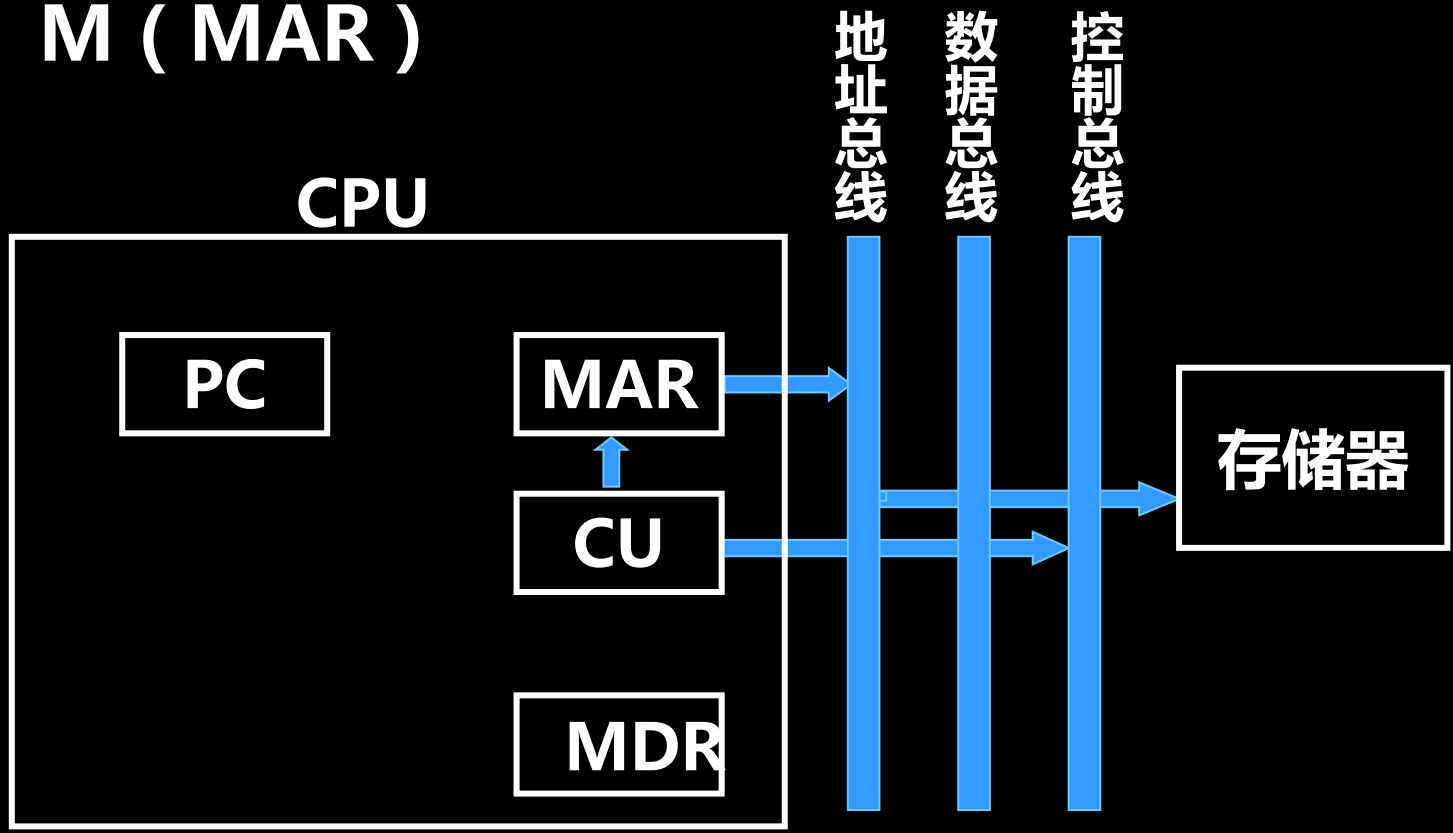
中断周期的数据流

0 → MAR　　向量地址 → PC

1 → W　　0 → EINT（置“0”）

PC → MDR

MDR → M（MAR）



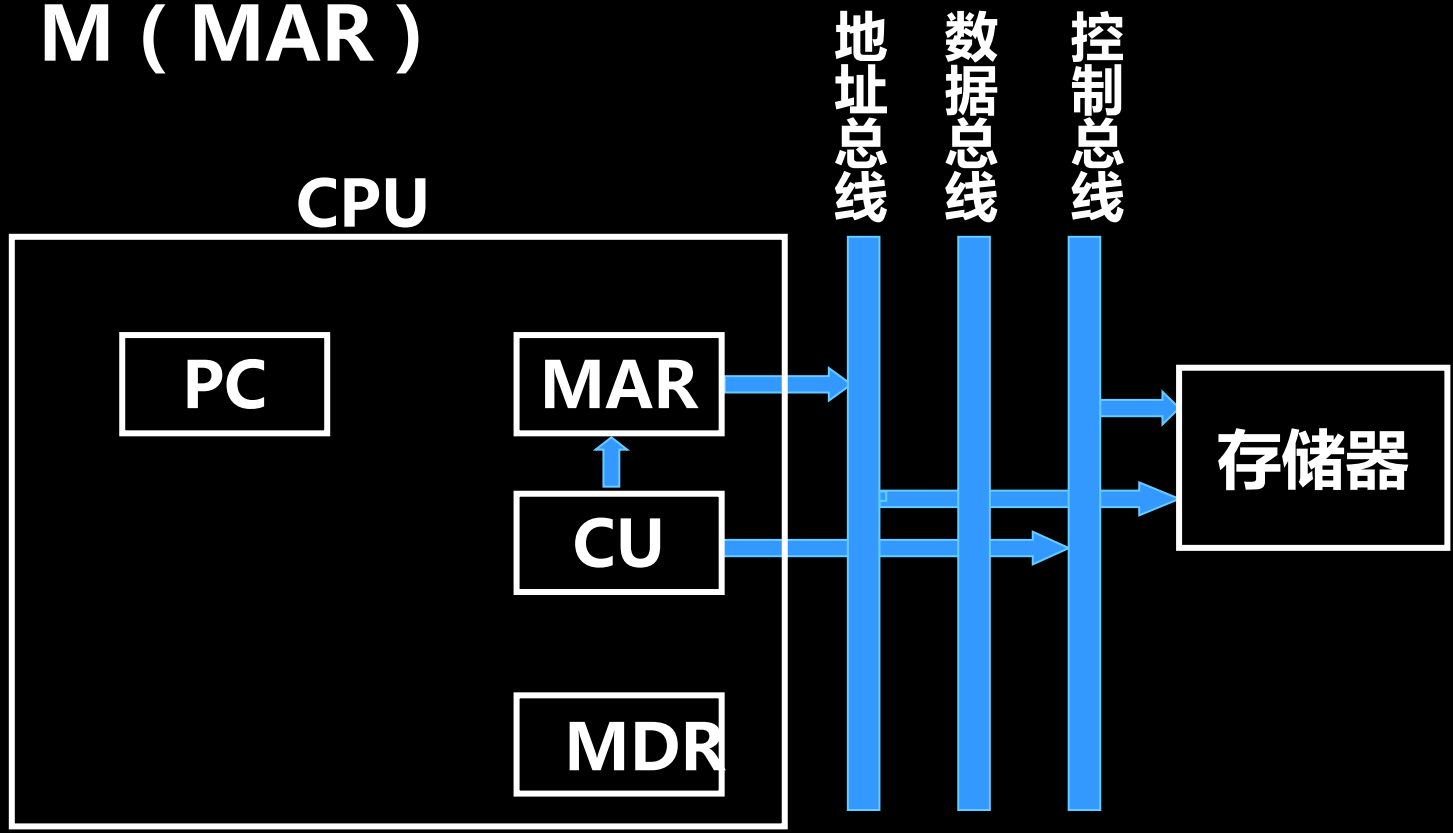
中断周期的数据流

0 → MAR　　向量地址 → PC

1 → W　　0 → EINT（置“0”）

PC → MDR

MDR → M（MAR）



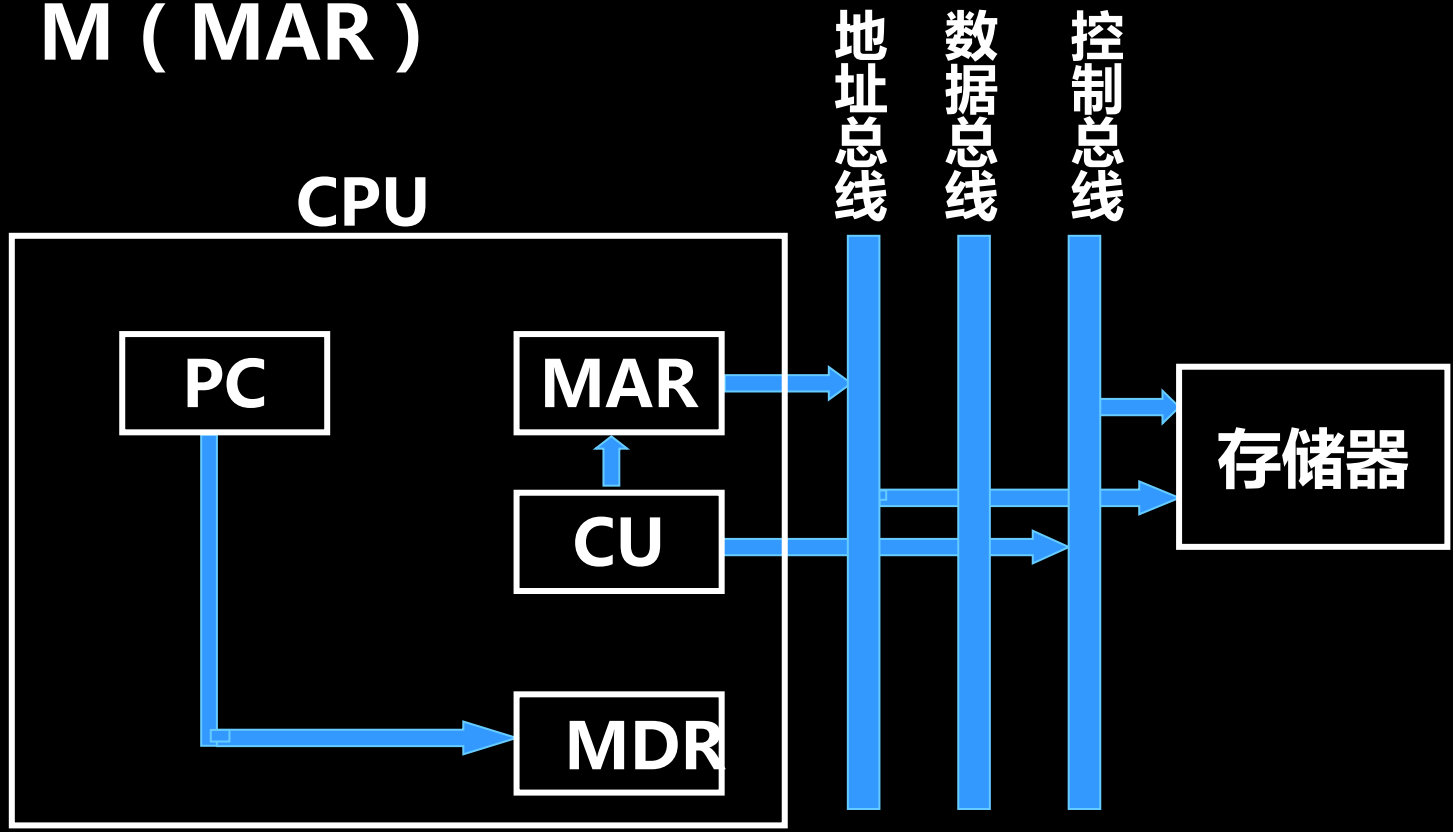
中断周期的数据流

0 → MAR　　向量地址 → PC

1 → W　　0 → EINT（置“0”）

PC → MDR

MDR → M（MAR）



中断周期的数据流

0 → MAR

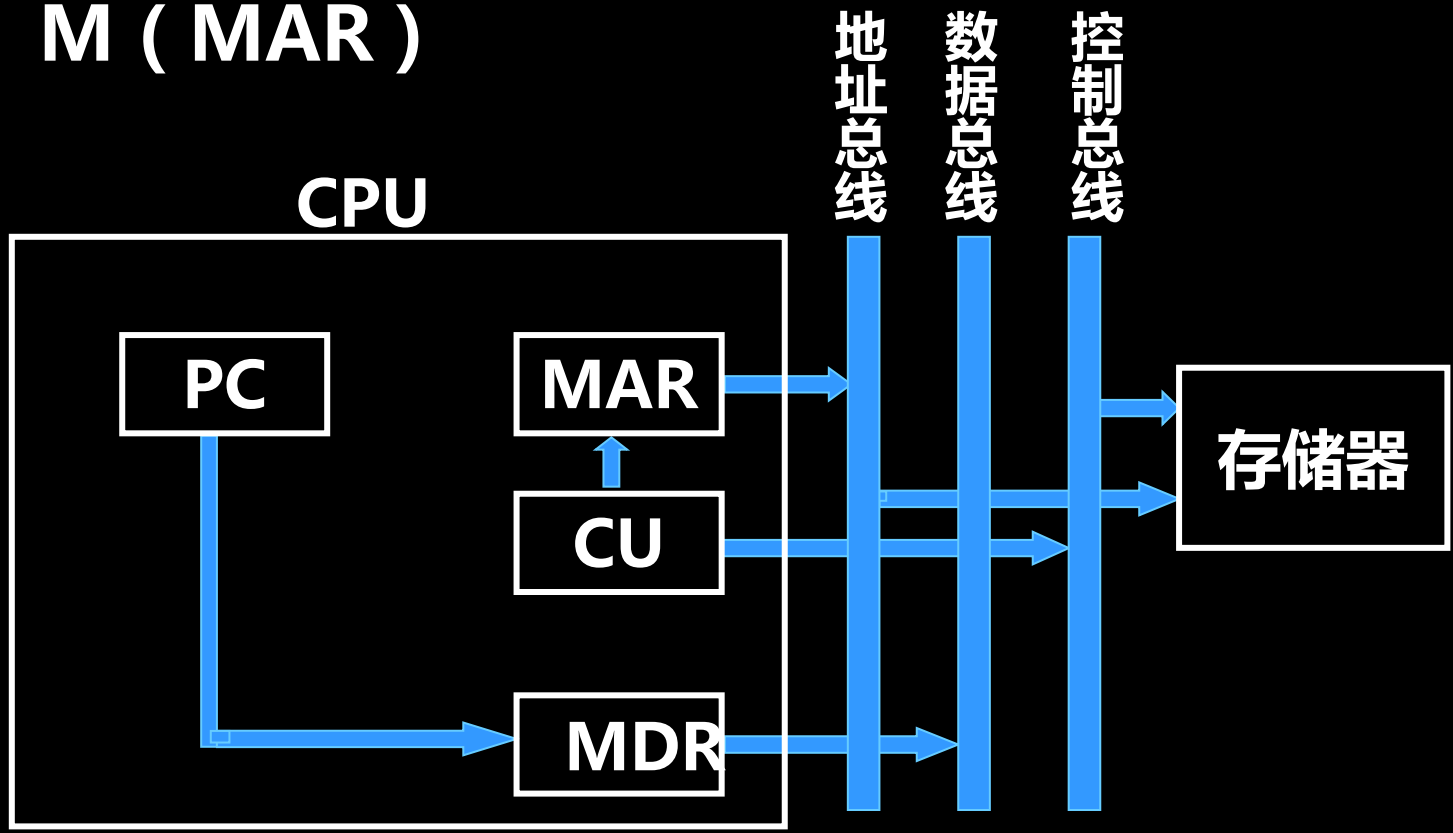
1 → W

PC → MDR

MDR → M (MAR)

向量地址 → PC

0 → EINT (置“0”)



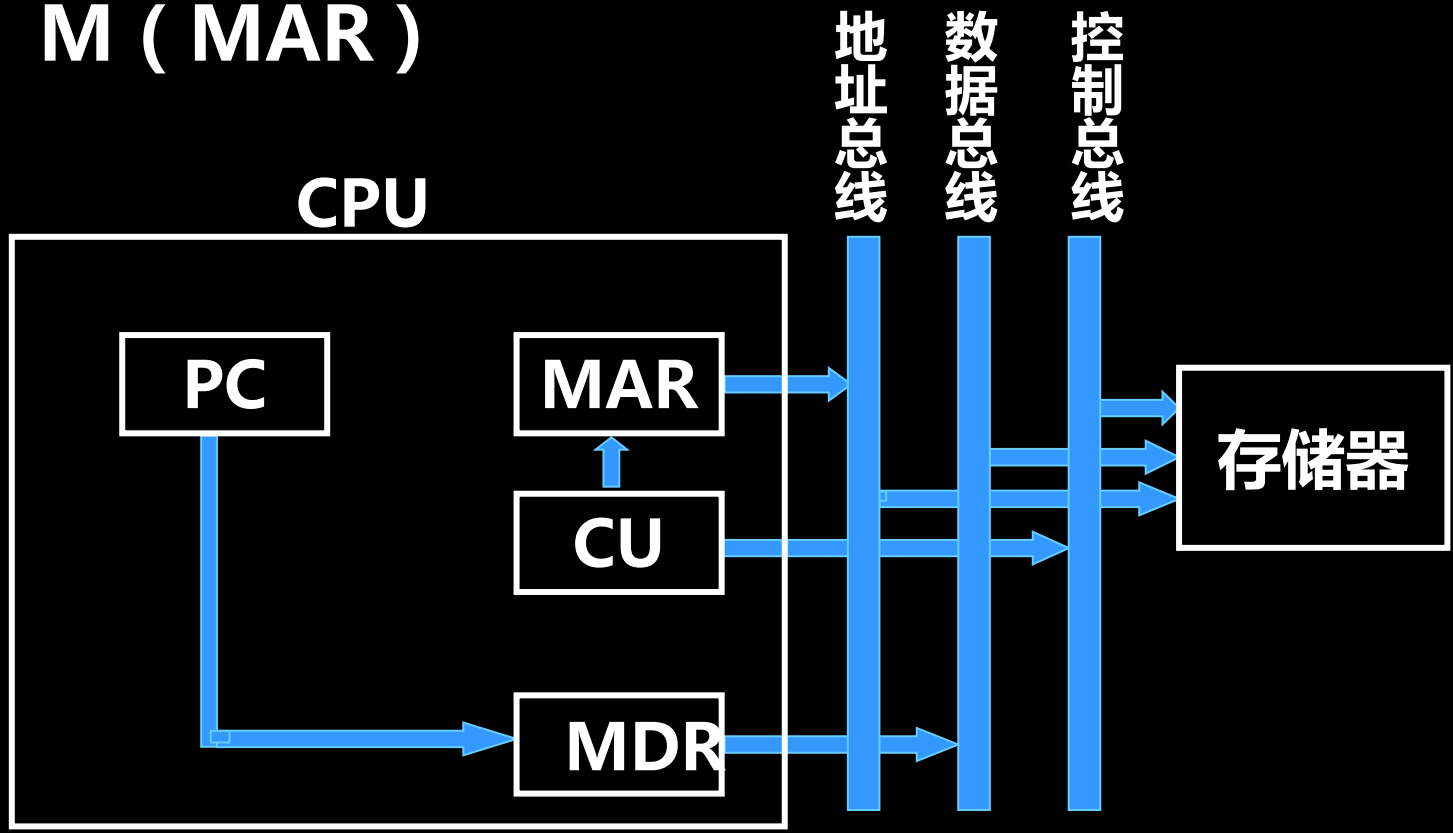
中断周期的数据流

0 → MAR　　向量地址 → PC

1 → W　　0 → EINT（置“0”）

PC → MDR

MDR → M（MAR）



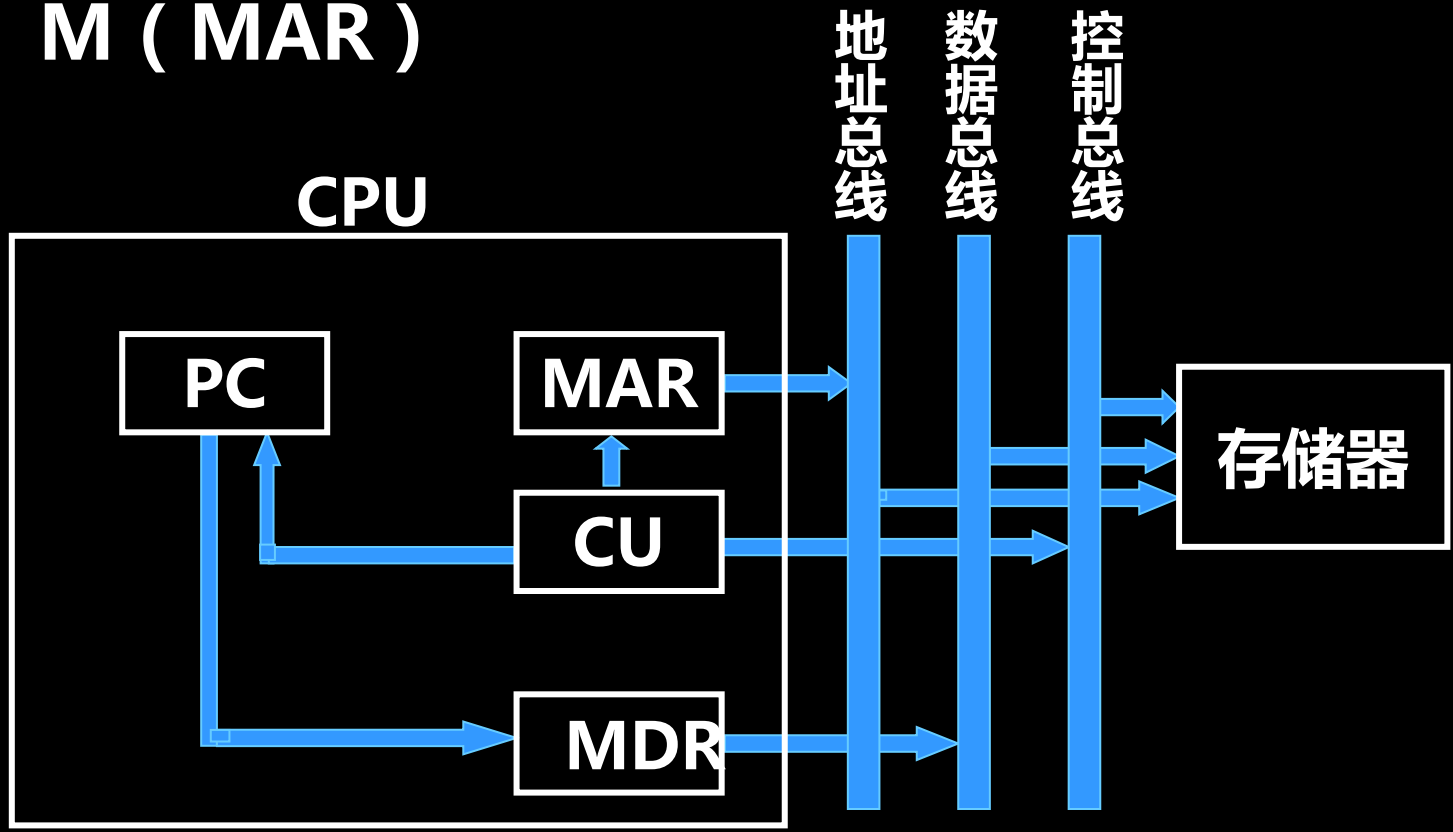
中断周期的数据流

0 → MAR　　向量地址 → PC

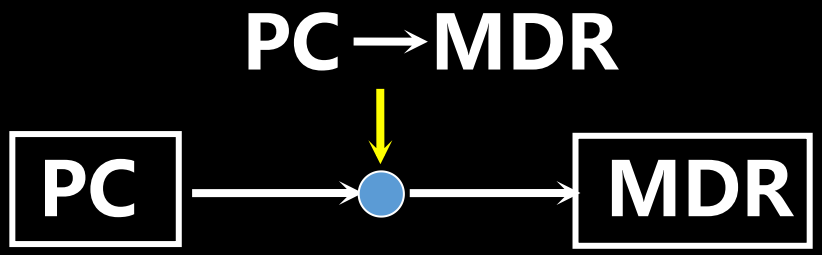
1 → W　　0 → EINT（置“0”）

PC → MDR

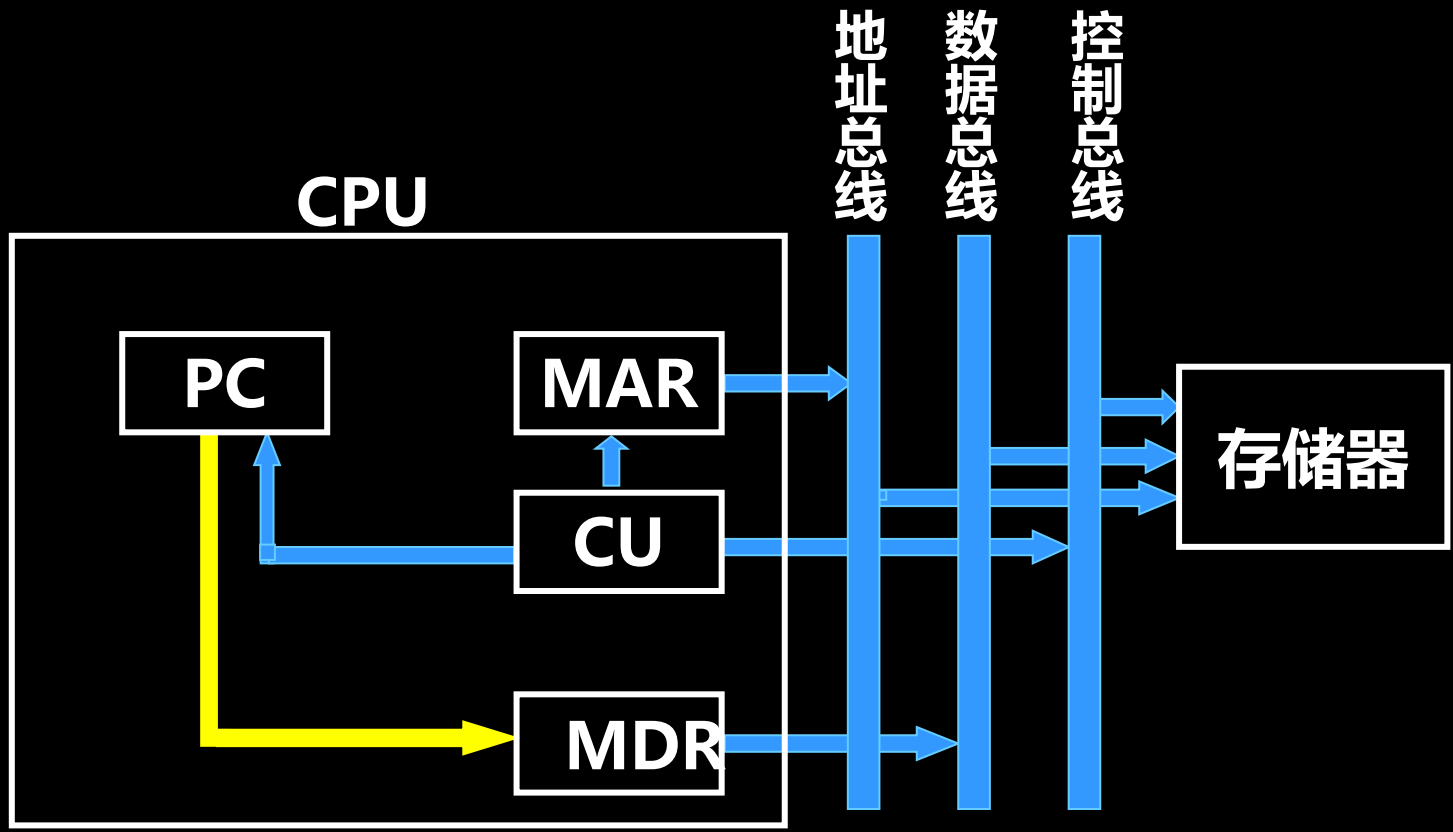
MDR → M（MAR）



中断周期的操作命令分析



操作命令（微操作命令）
控制信号



断点的保存位置

发出的内存地址有如下两种形式：

1. 程序断点存入“0”地址

0 → MAR

2. 程序断点进栈

(SP) – 1 → MAR

中断服务程序入口识别

识别中断服务程序入口，有如下两种形式：

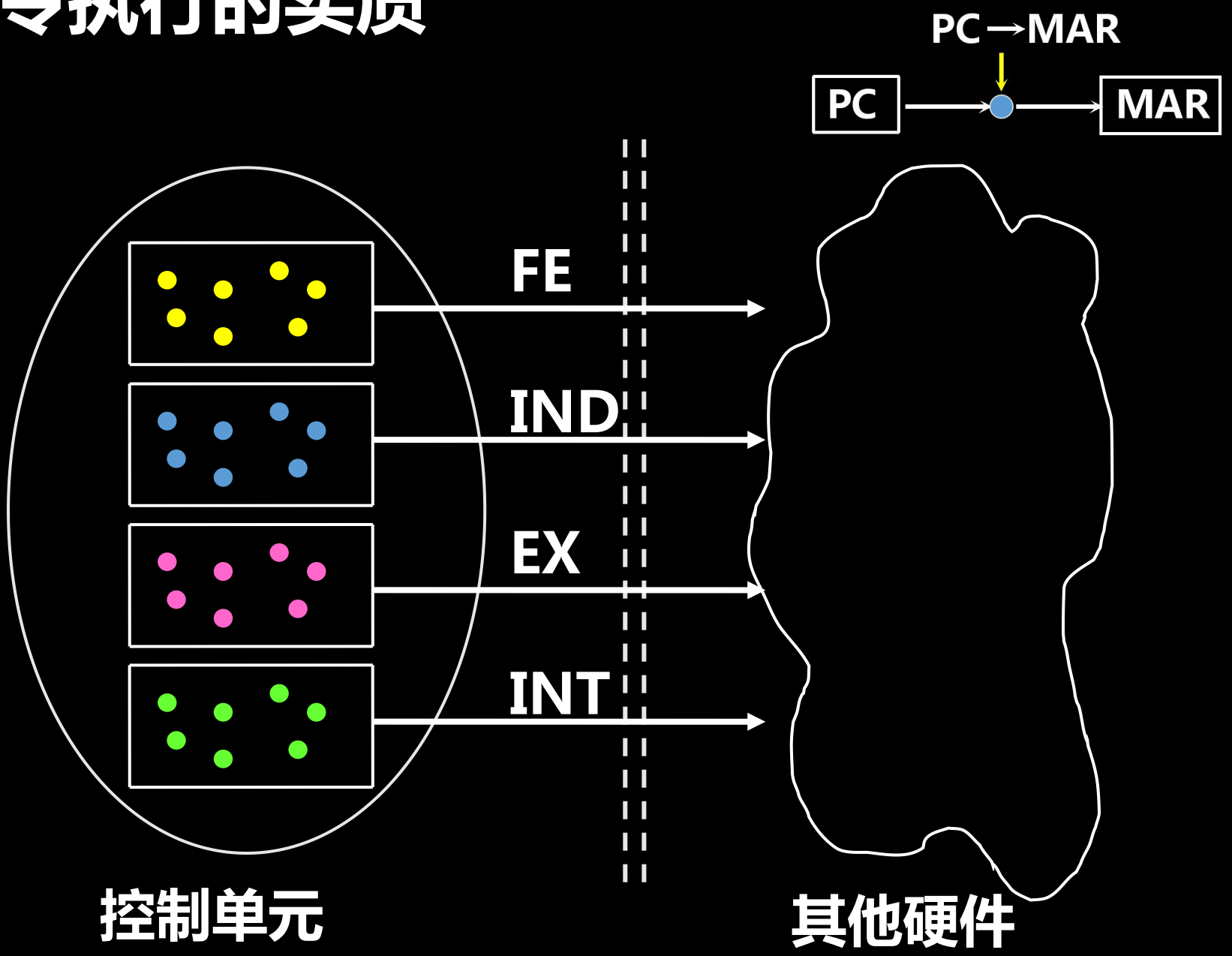
1. 硬件实现方法

   向量地址 → PC

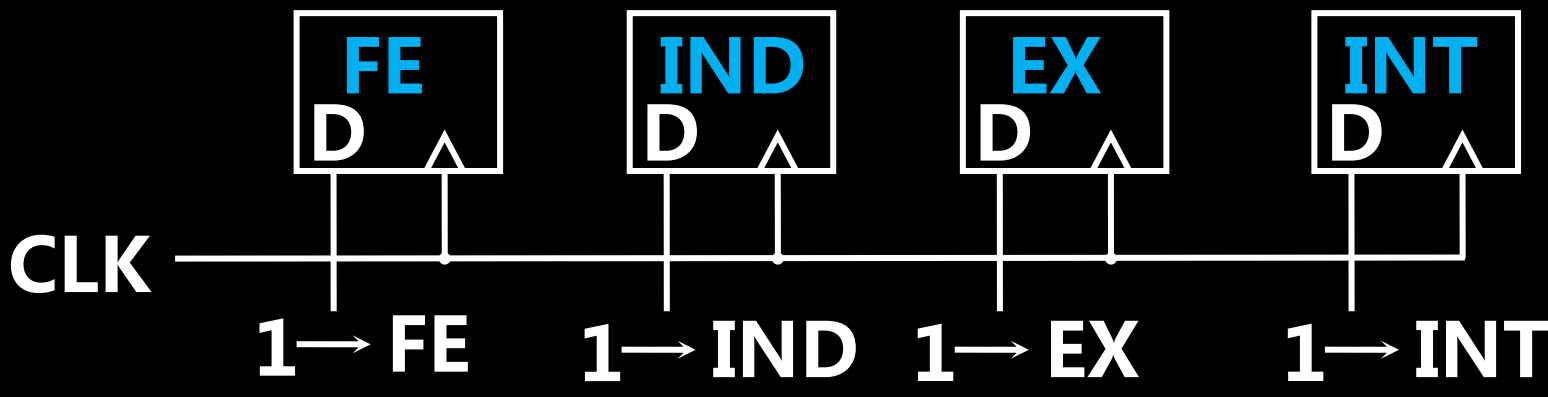
2. 软件实现方法

   中断识别程序入口地址 M → PC

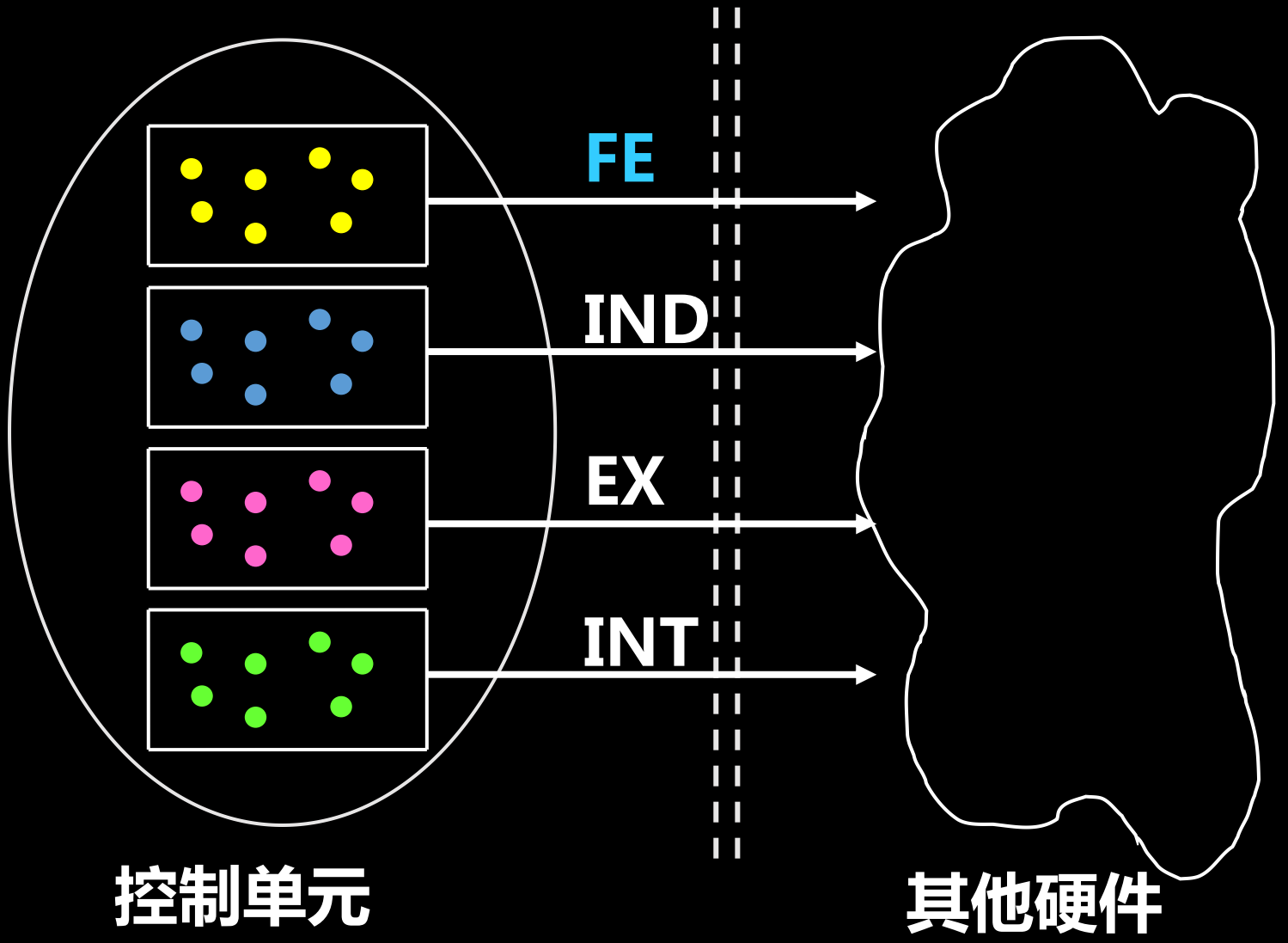
指令执行的实质



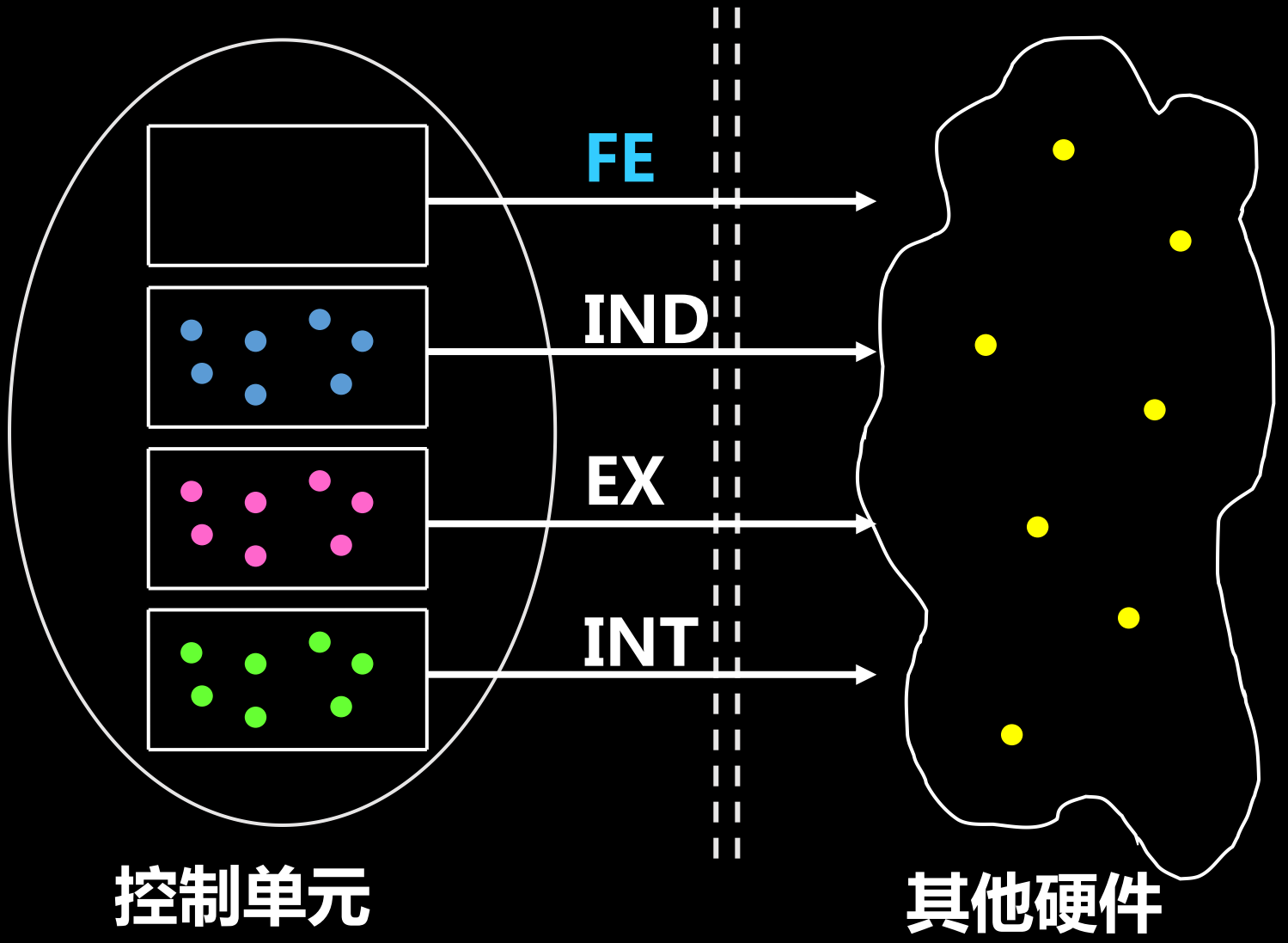
CPU工作周期的标志



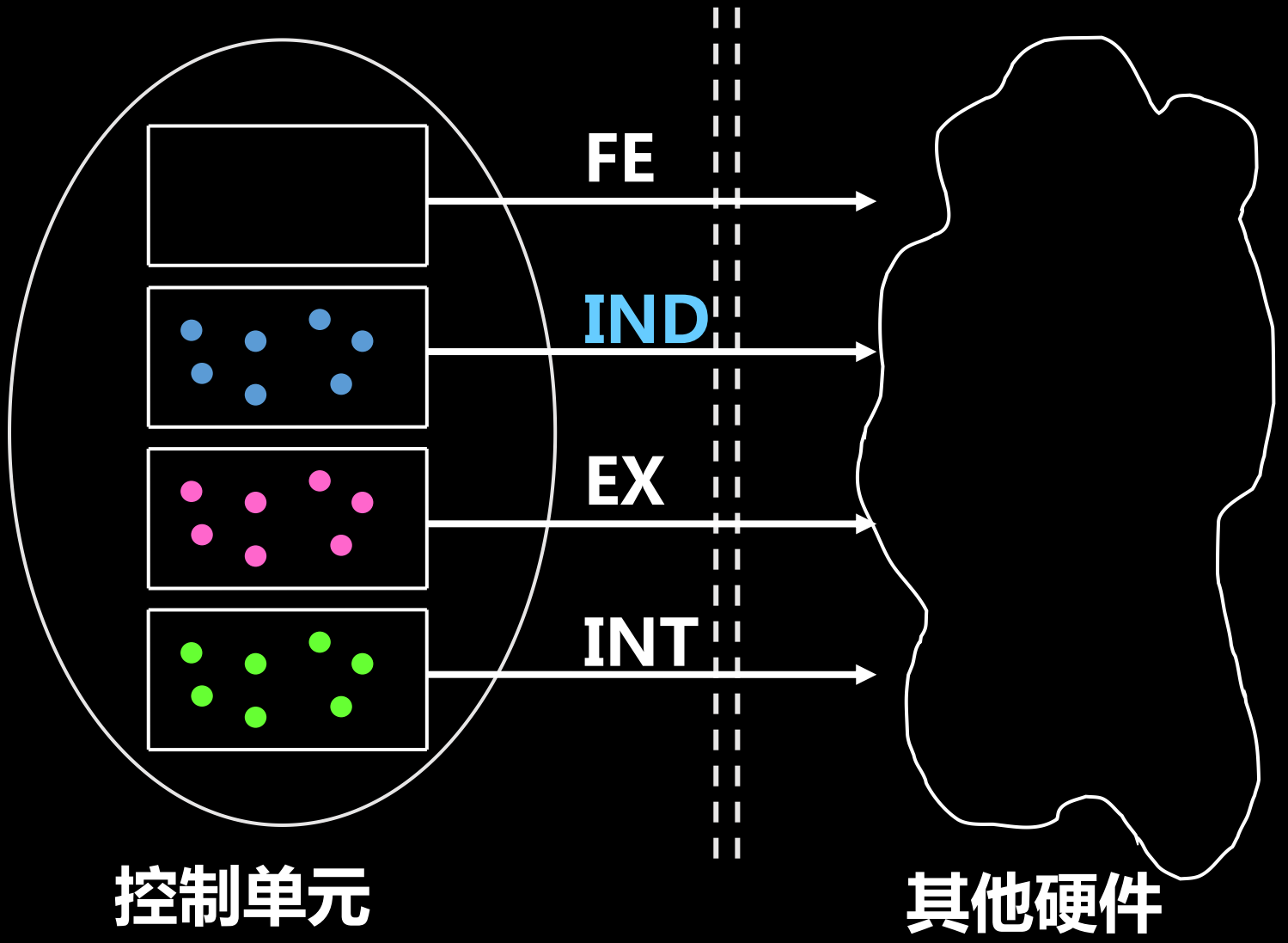
操作命令的作用



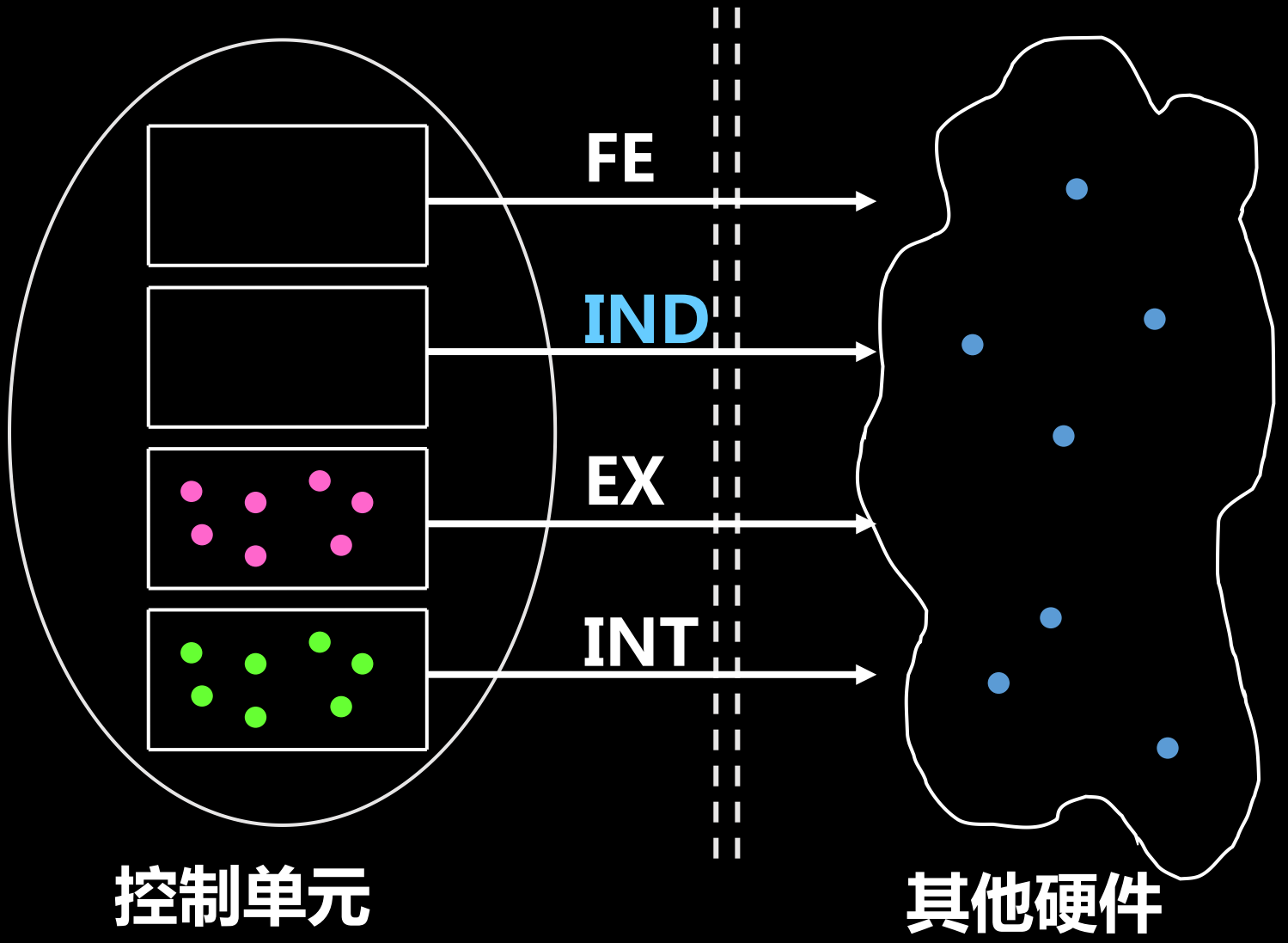
操作命令的作用



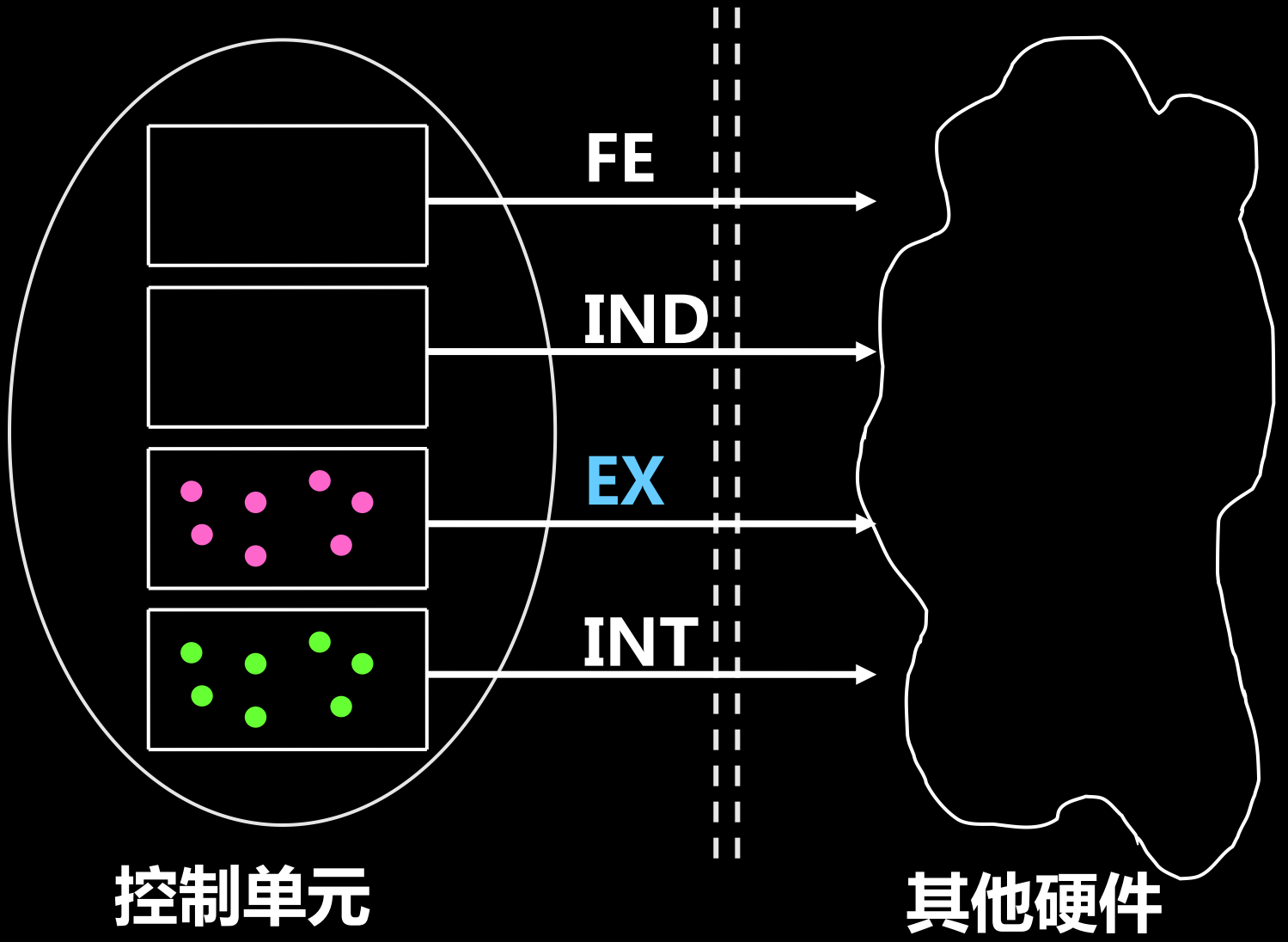
操作命令的作用



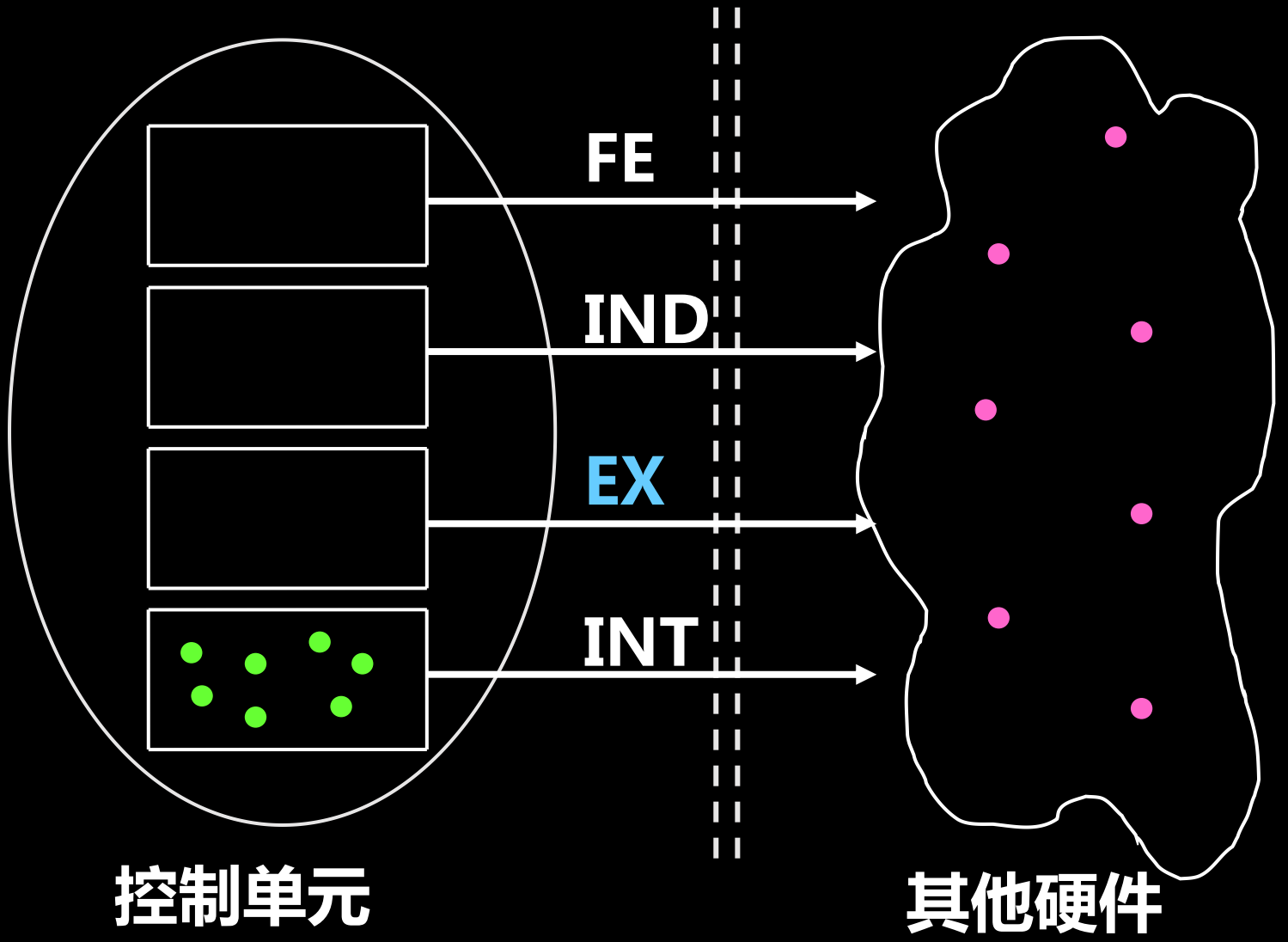
操作命令的作用



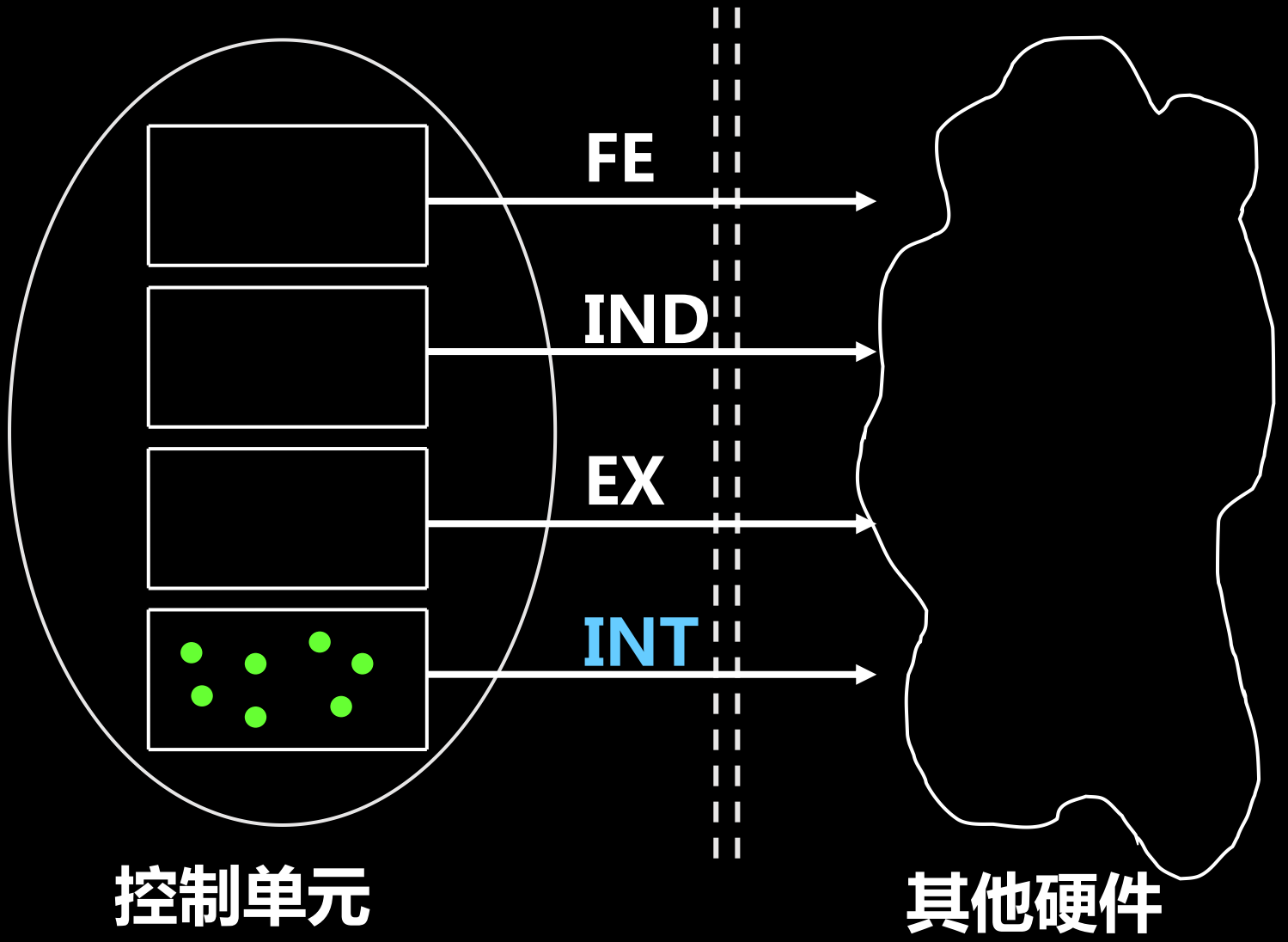
操作命令的作用



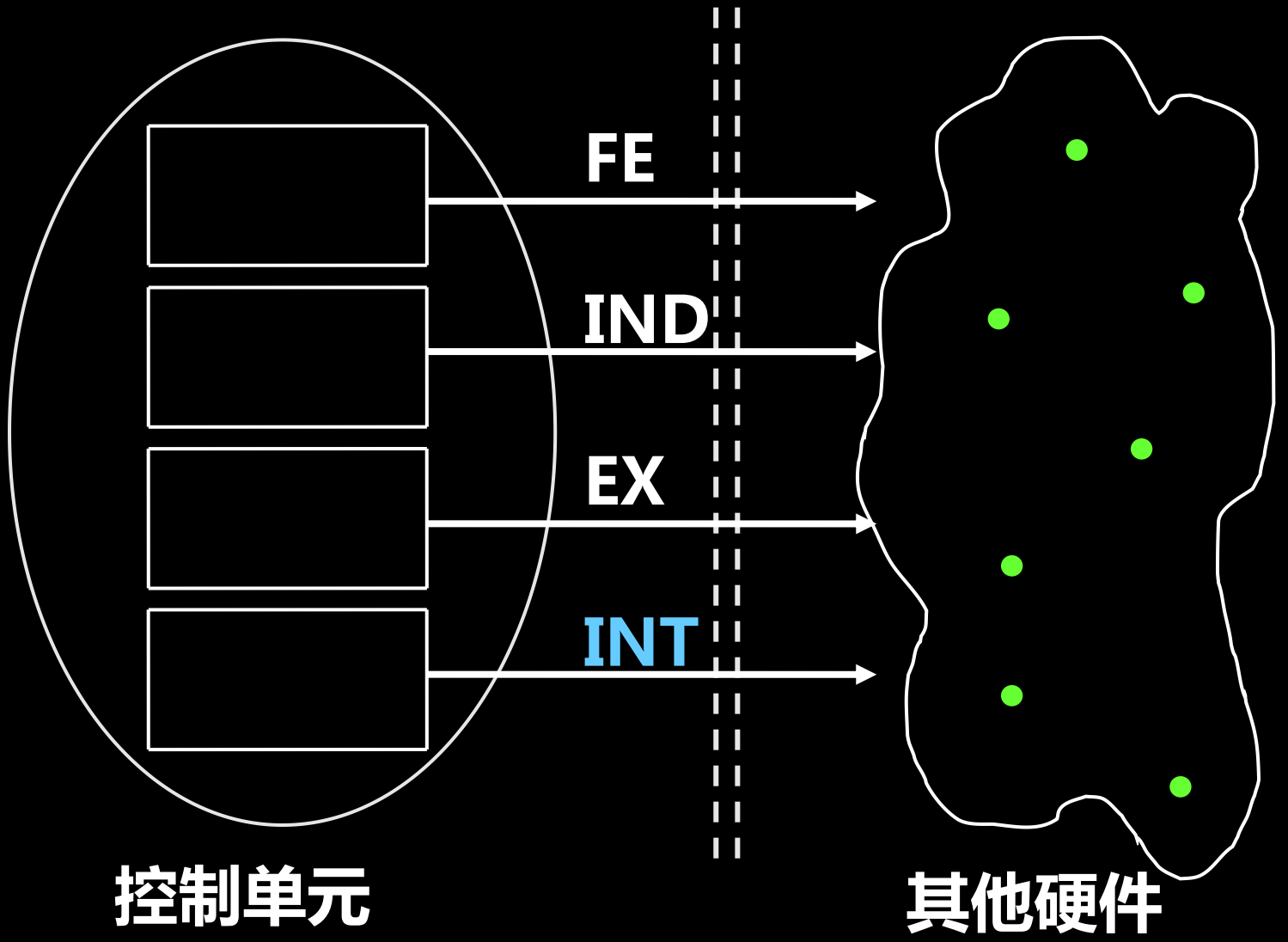
操作命令的作用



操作命令的作用

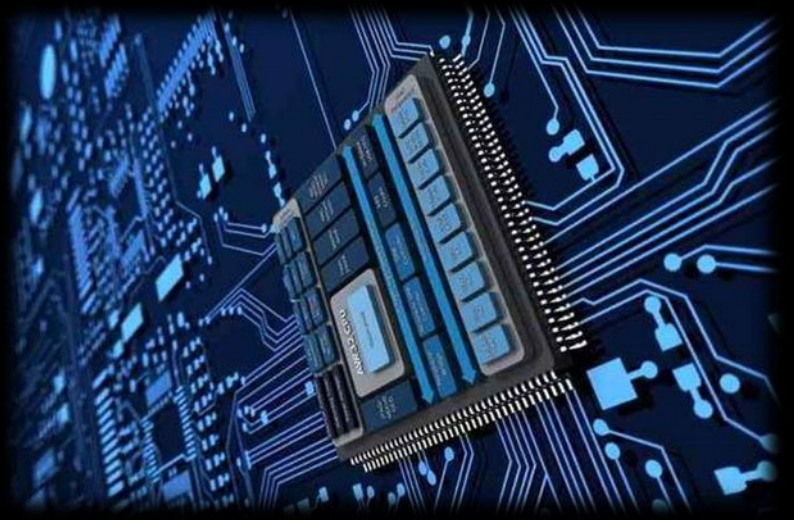


操作命令的作用



推荐阅读：如何测量程序执行的时间

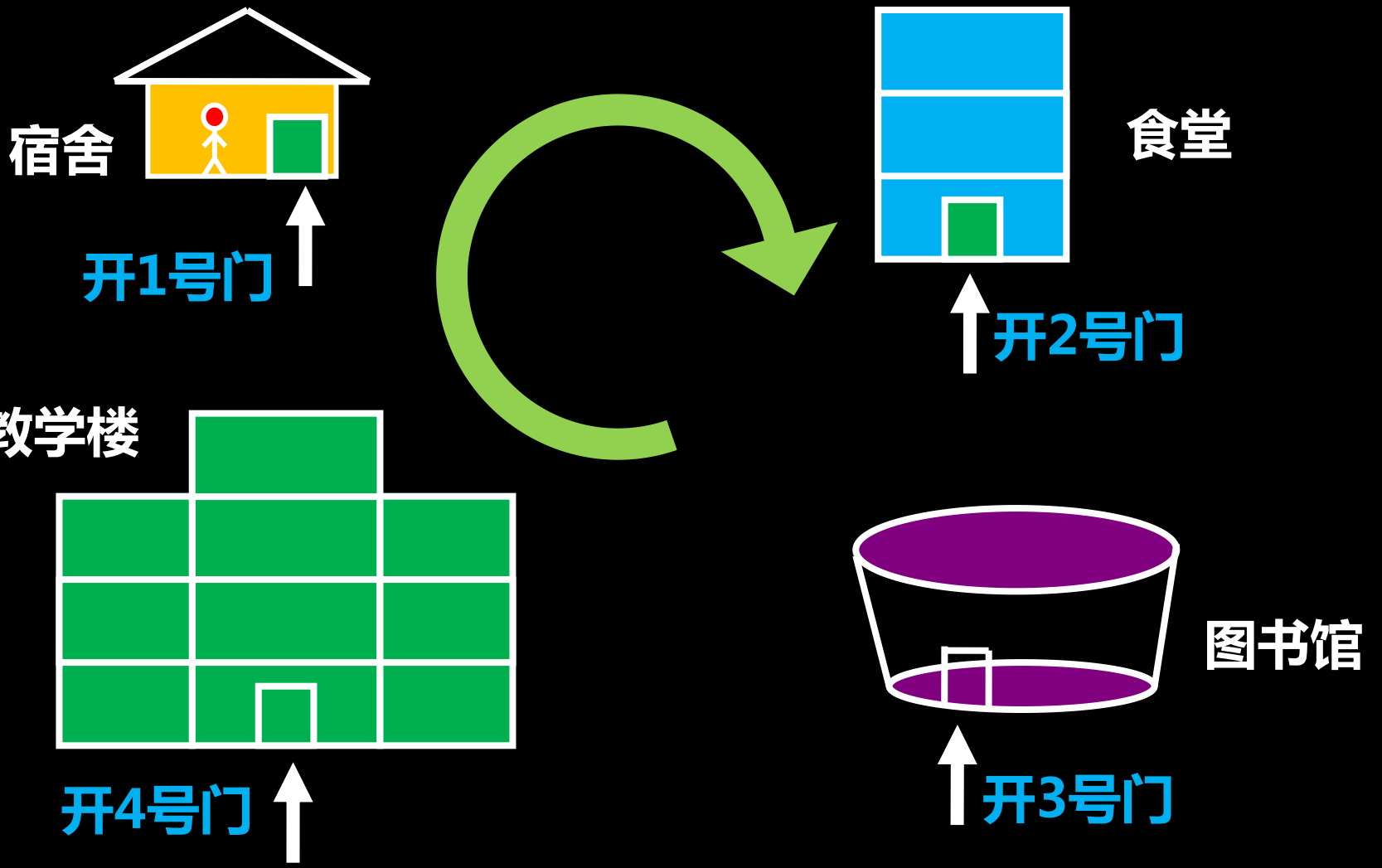
- 计算机是在微观和宏观两个不同尺度工作的；
- 多进程机制对程序执行时间的影响；
- 通过间隔计数（interval counting）可得到程序执行的粗略时间；
- 周期计数器（cycle counter）可得到更精确的程序执行时间，但是受上下文切换影响很大；
- 高速缓存和转移预测导致程序每次执行的时间都不同；
- 有一系列库函数支持程序时间测量



控制信号

大连理工大学 赖晓晨

指令执行的实质：一天的生活

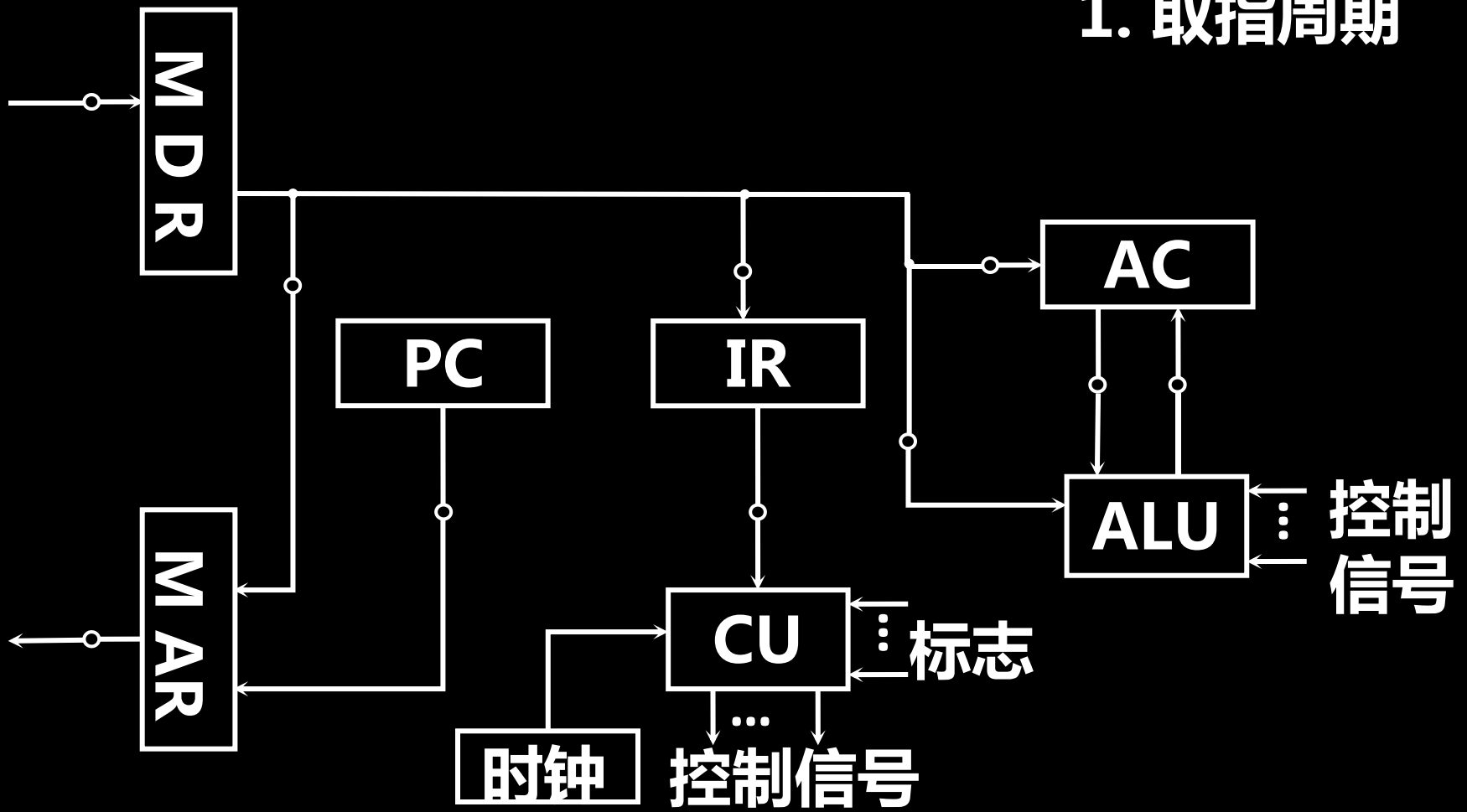


指令执行的实质：一天的生活

控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

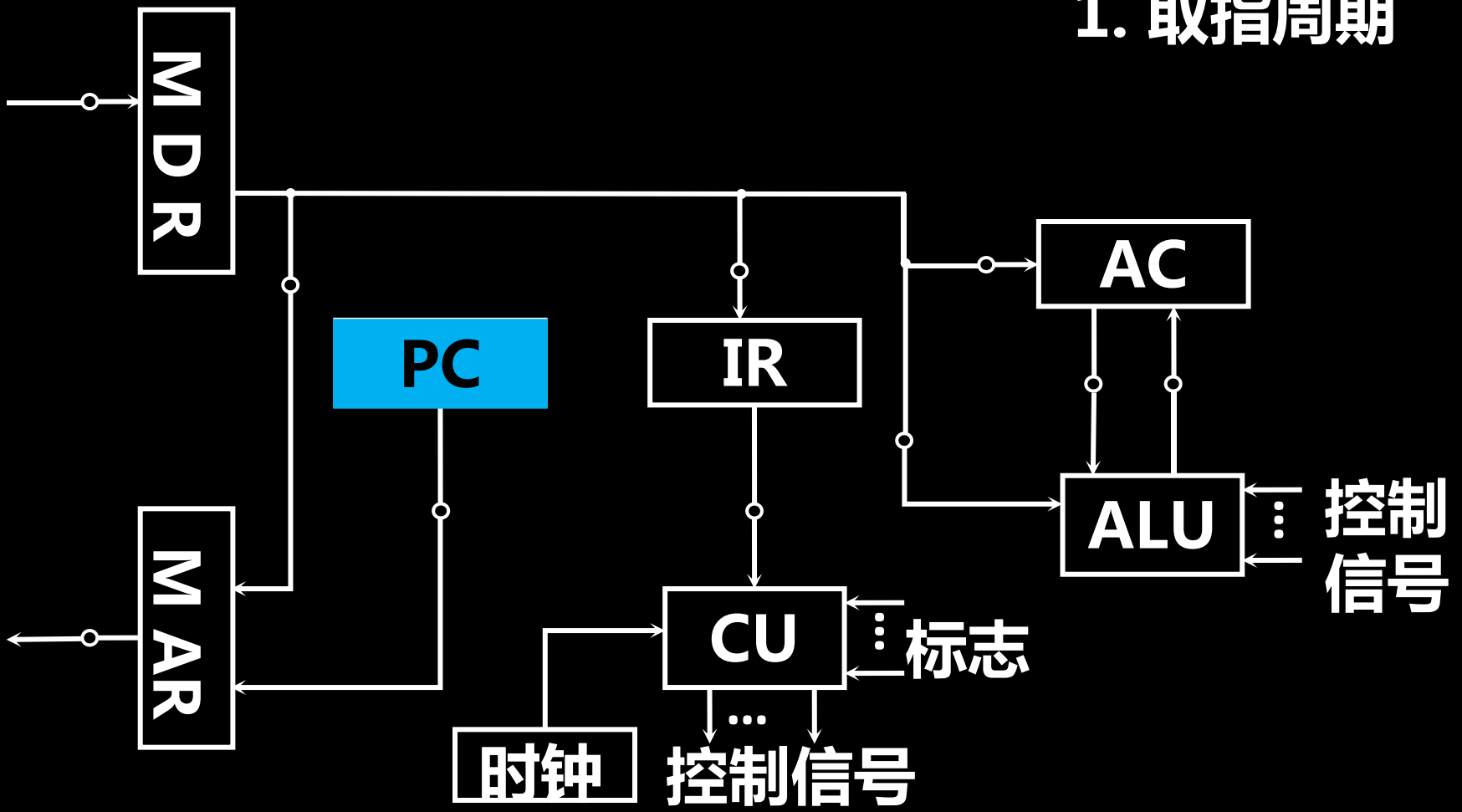
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

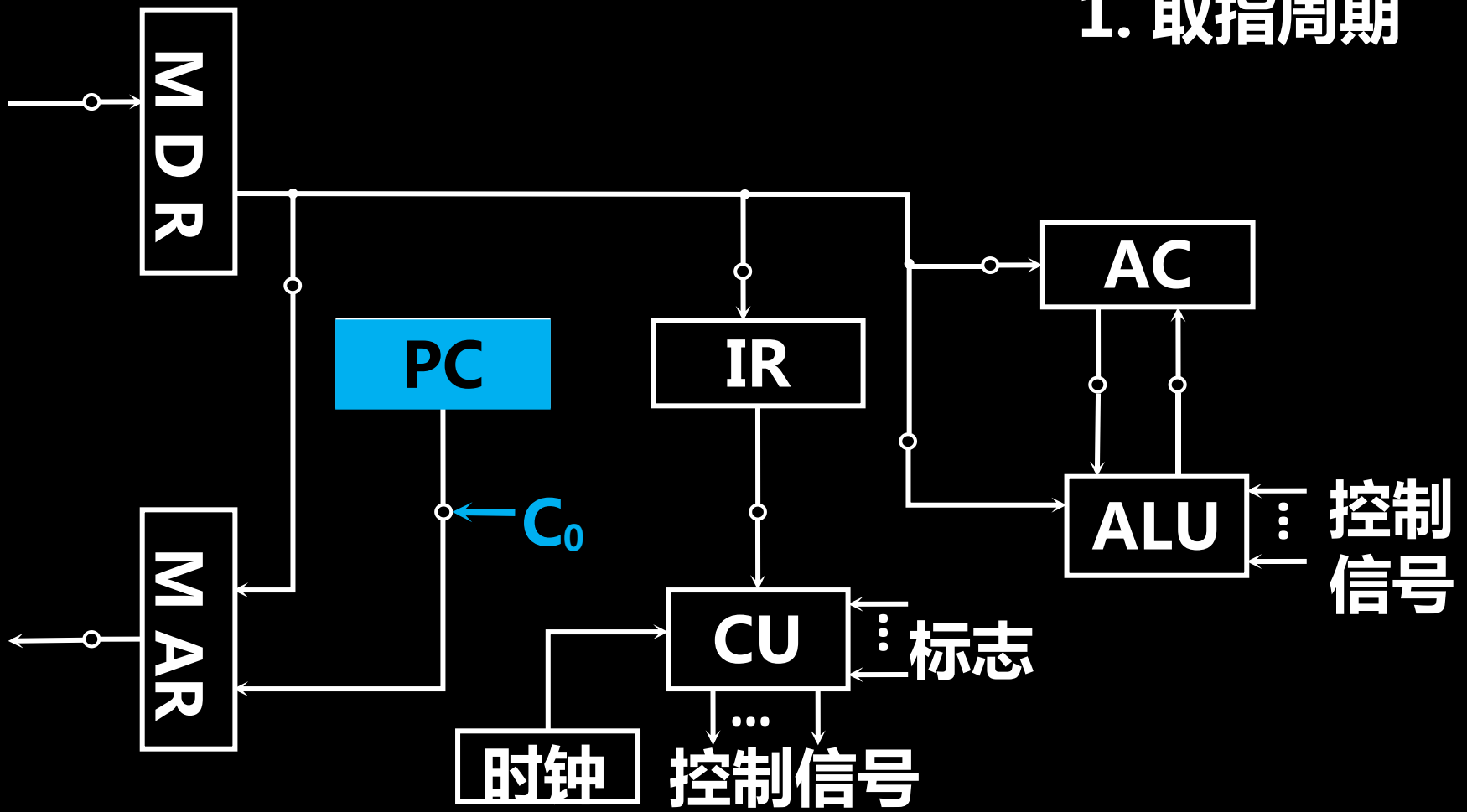
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

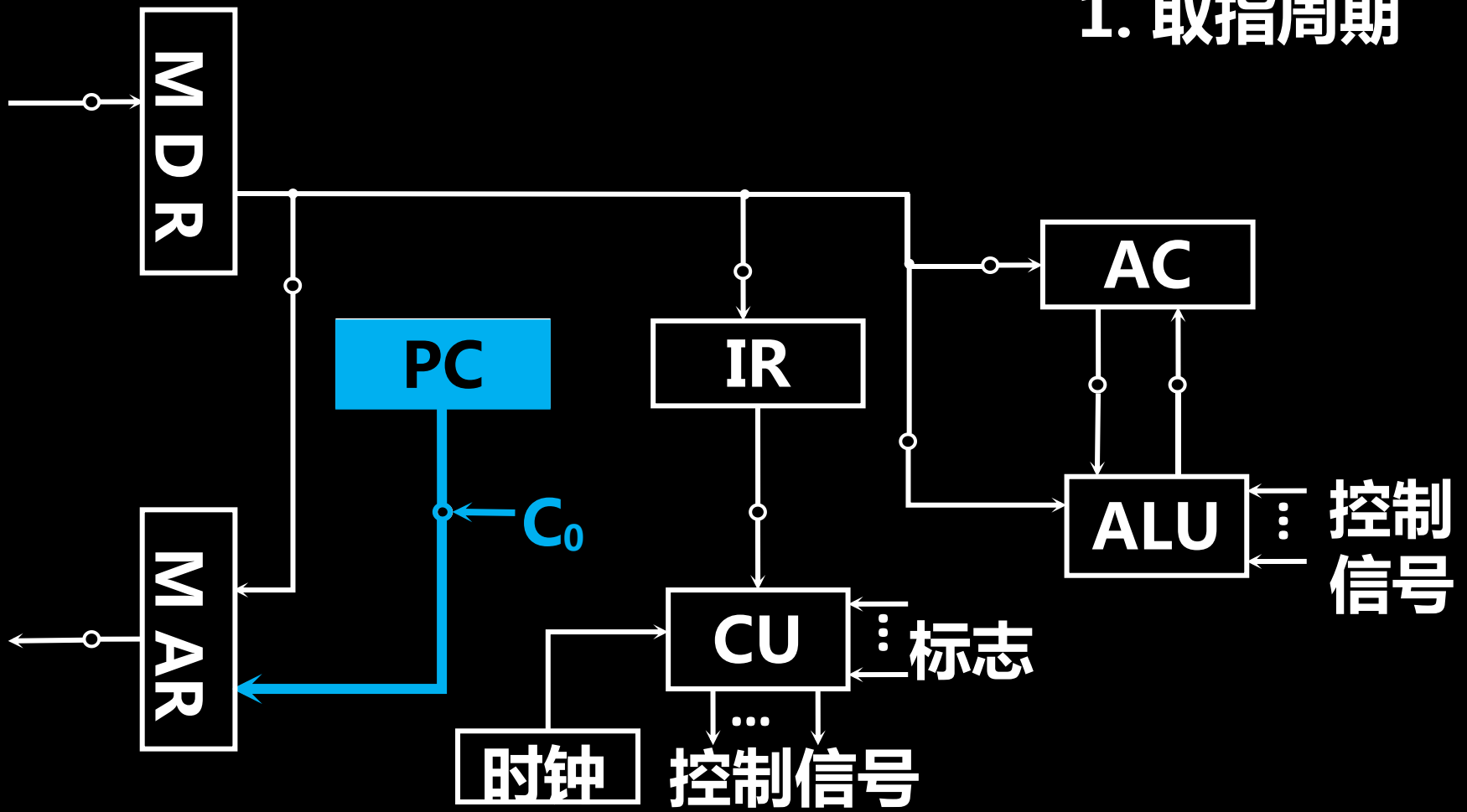
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

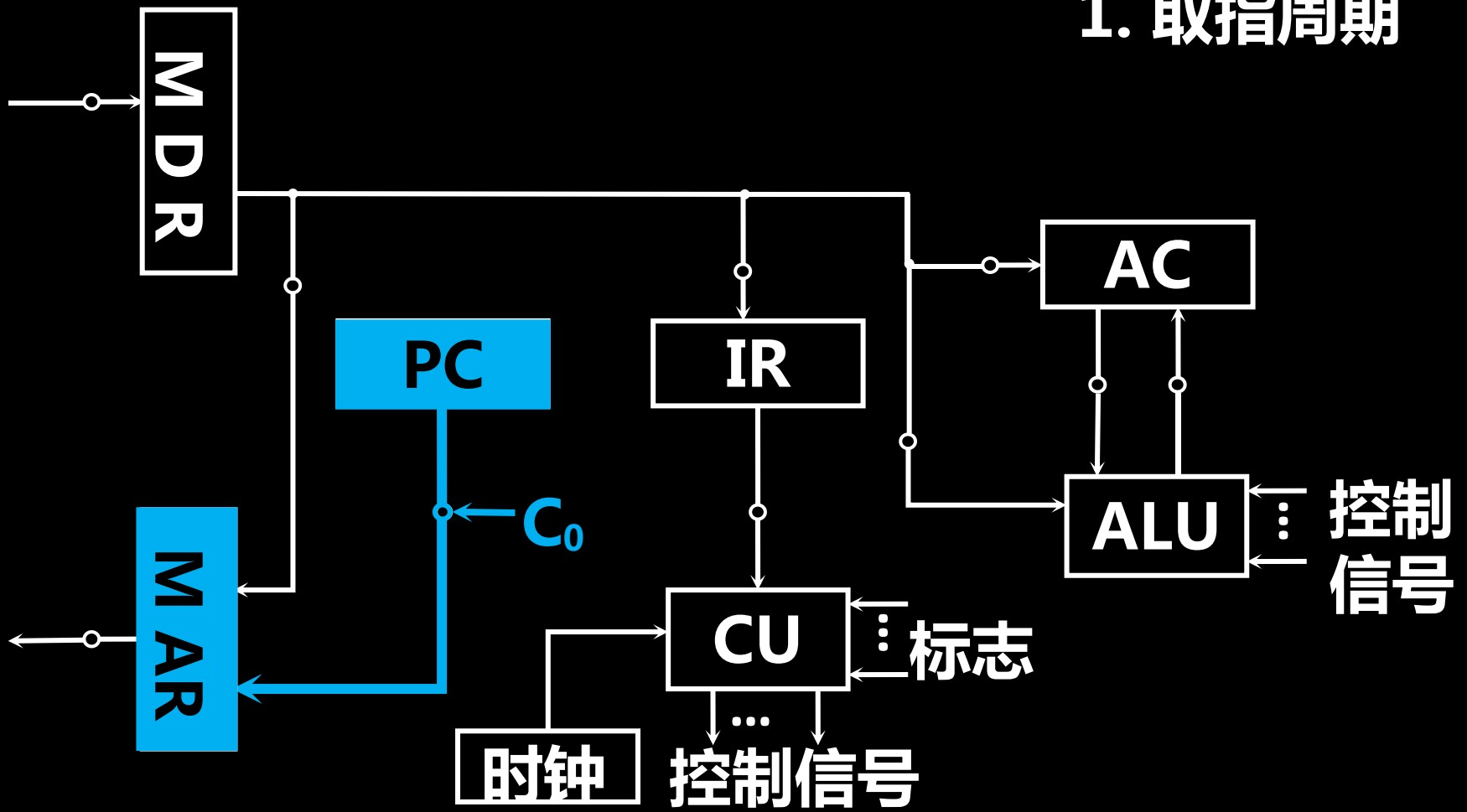
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

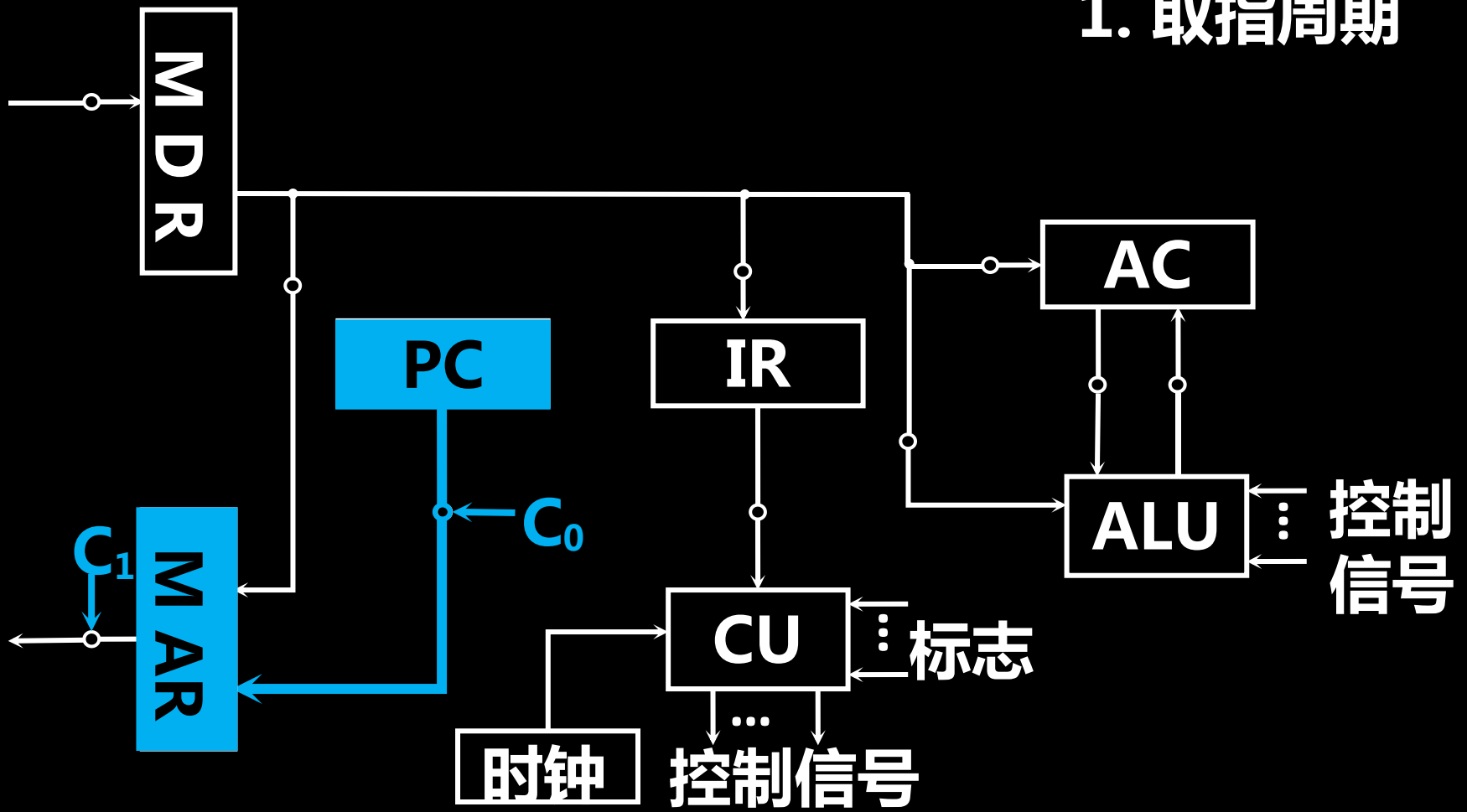
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

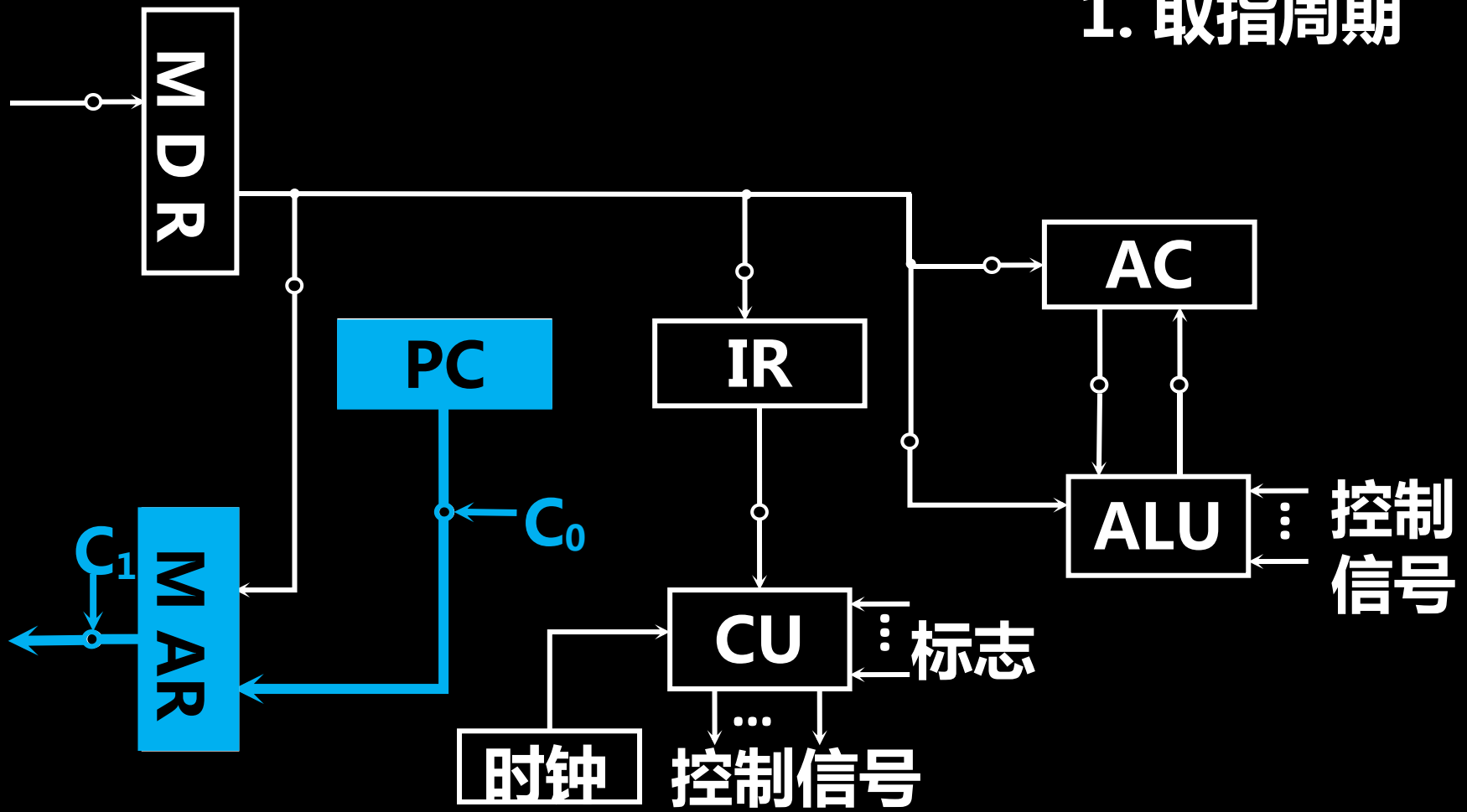
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

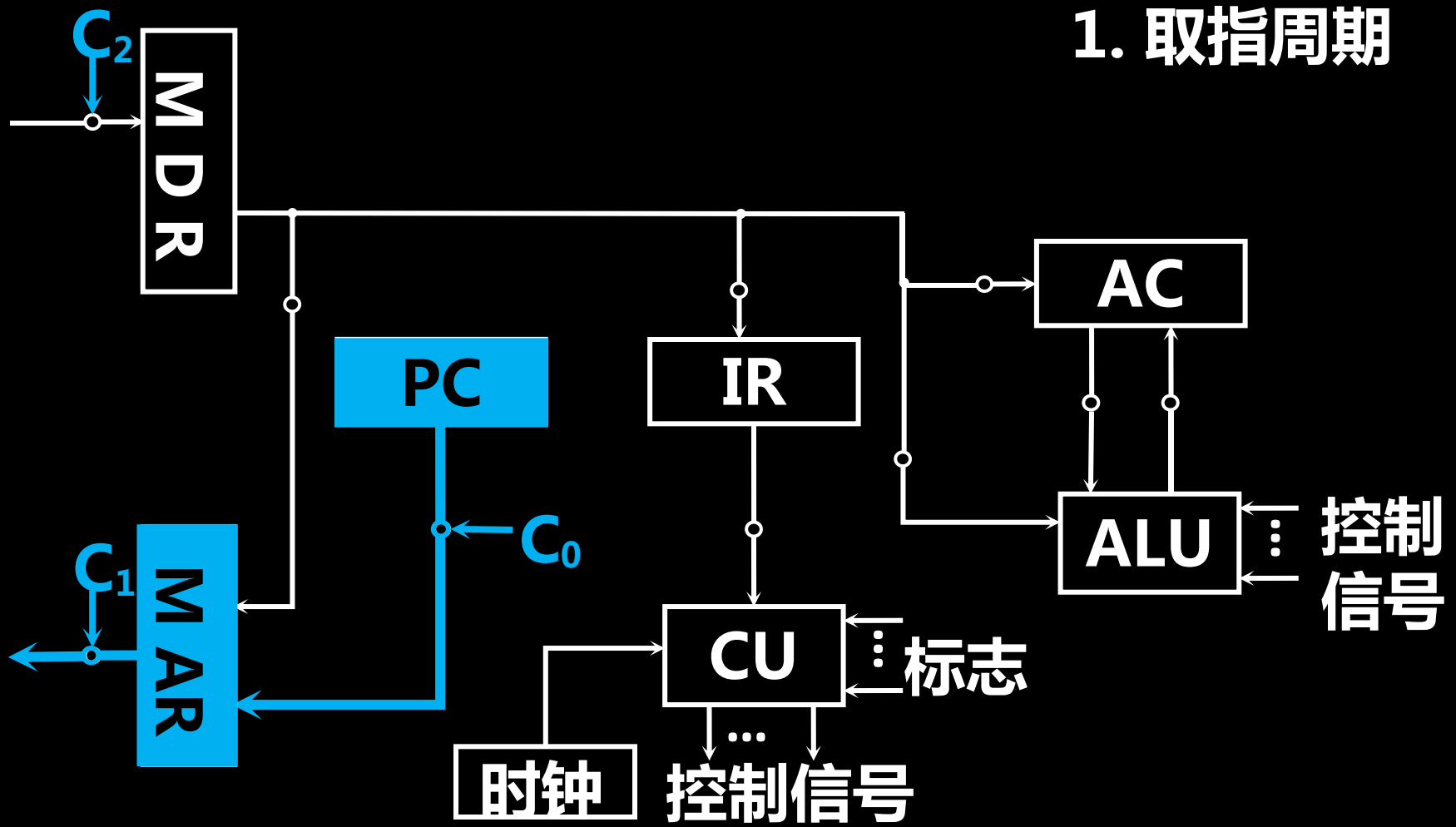
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

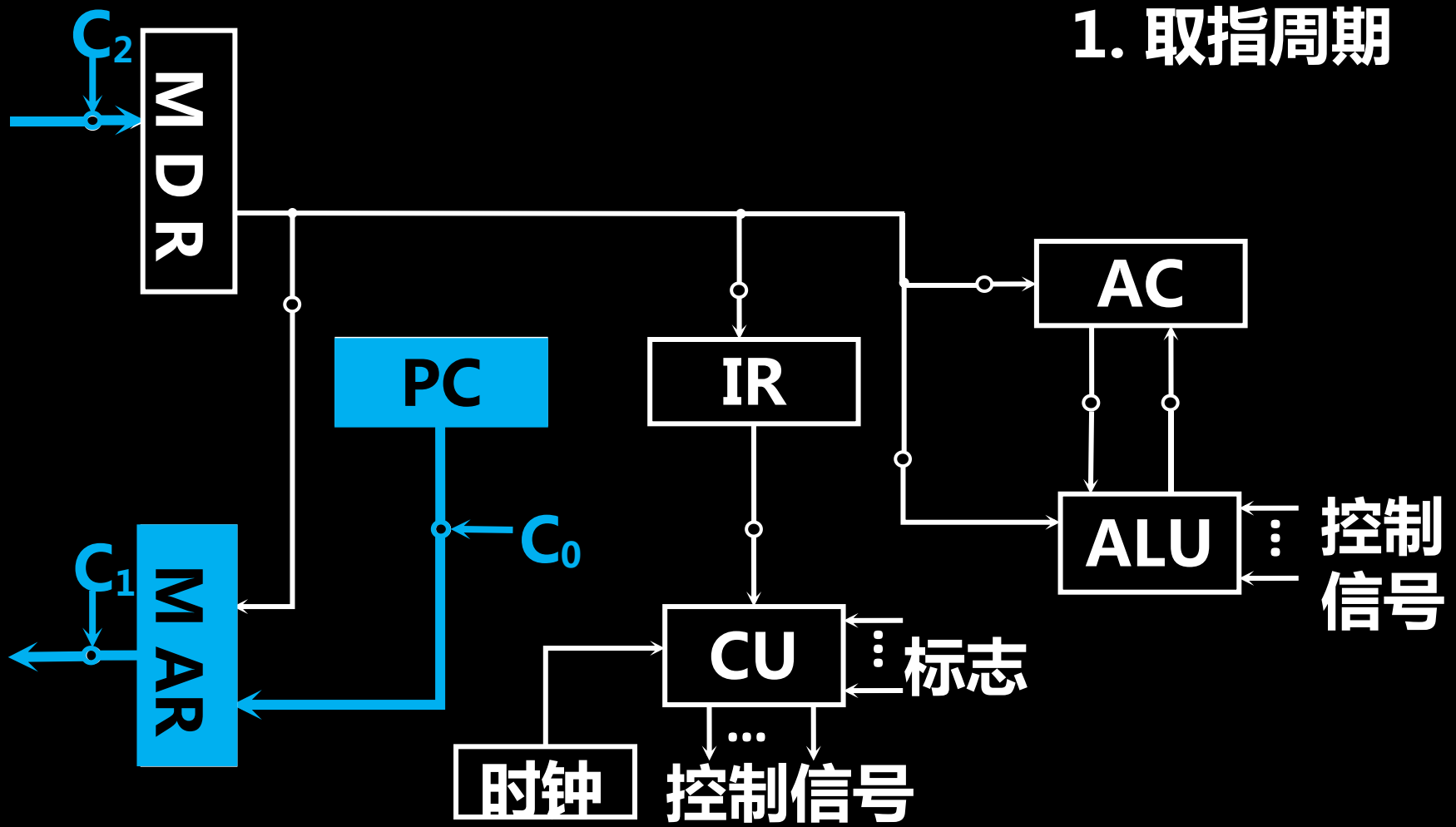
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

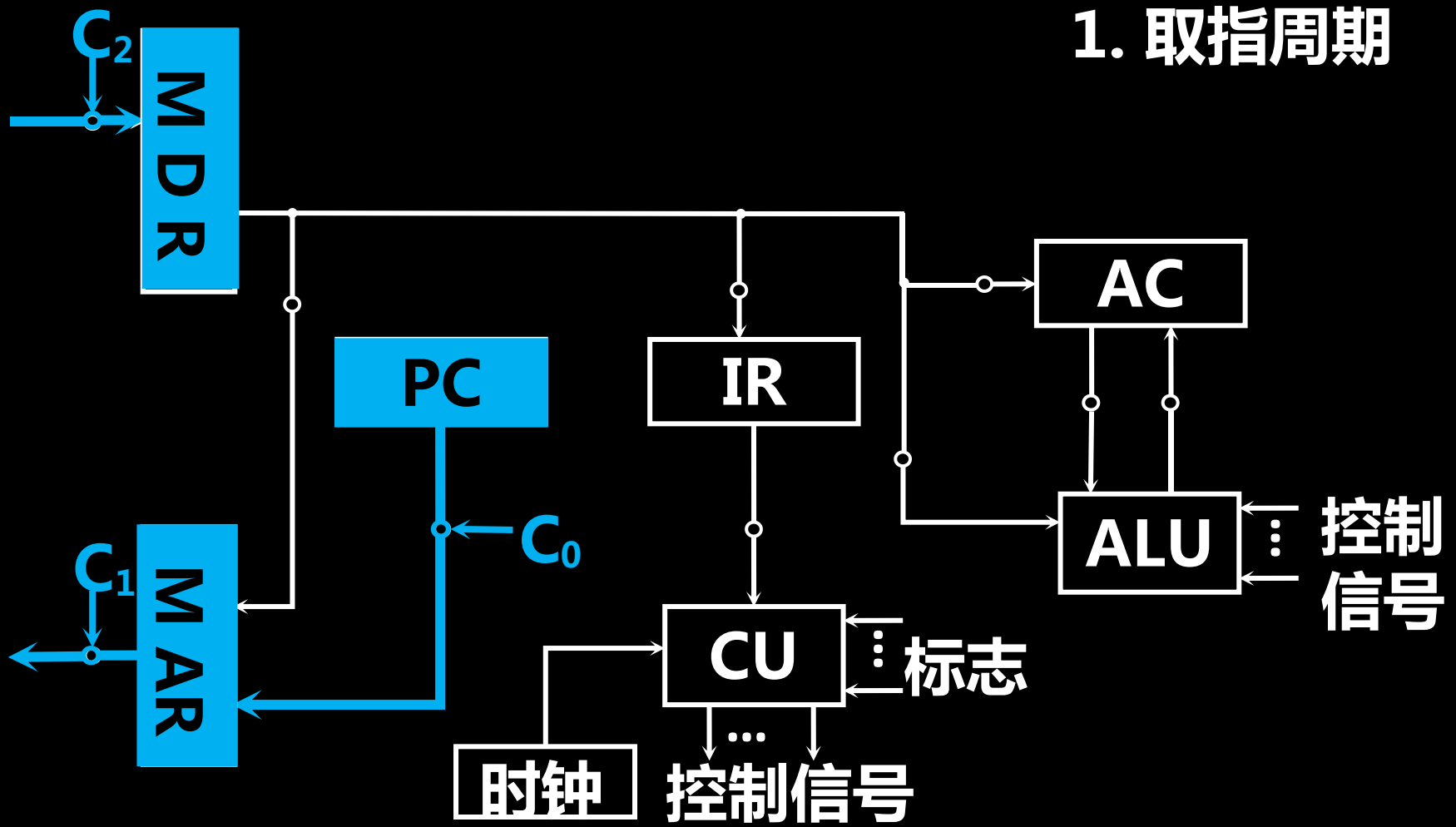
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

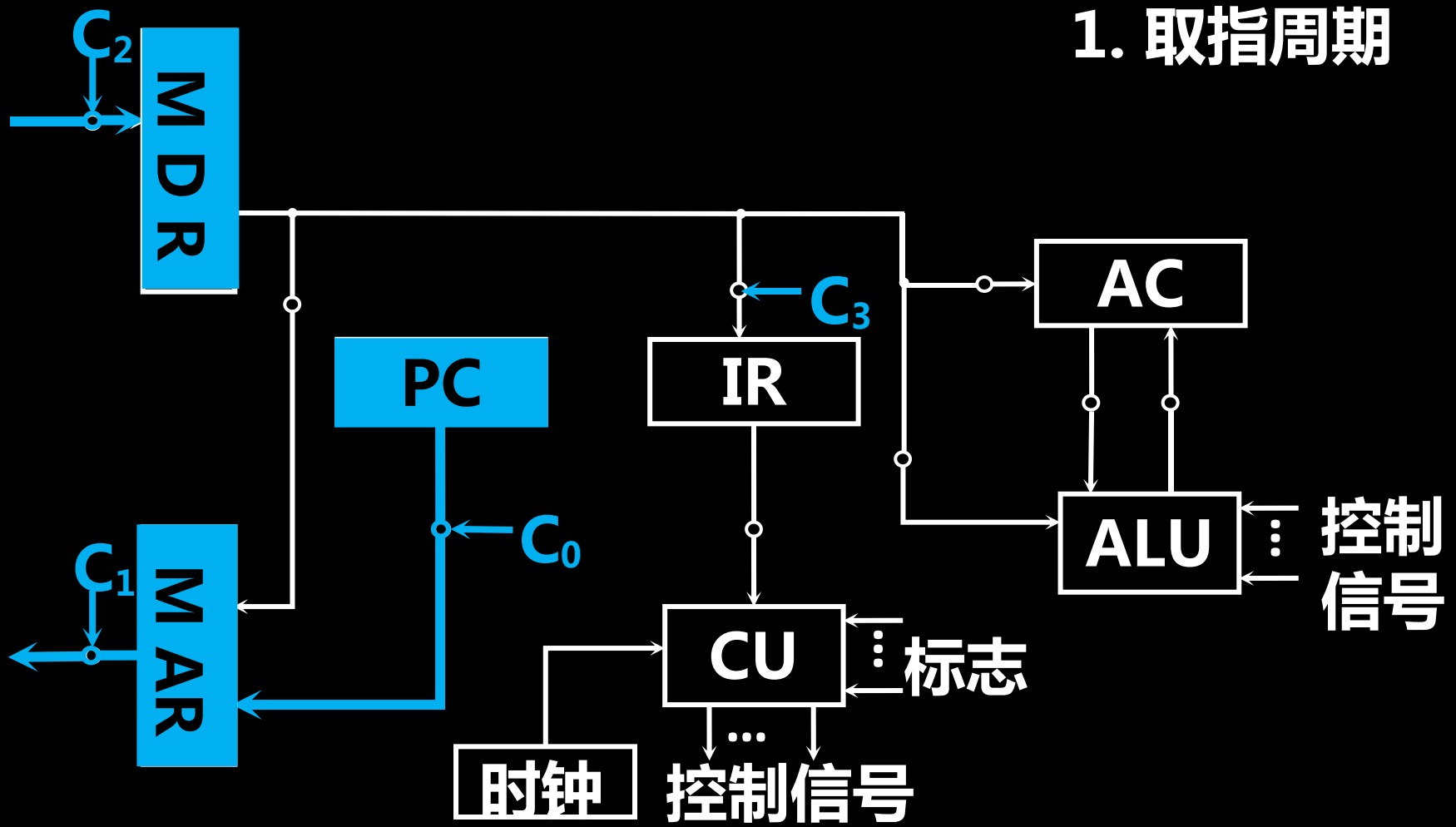
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

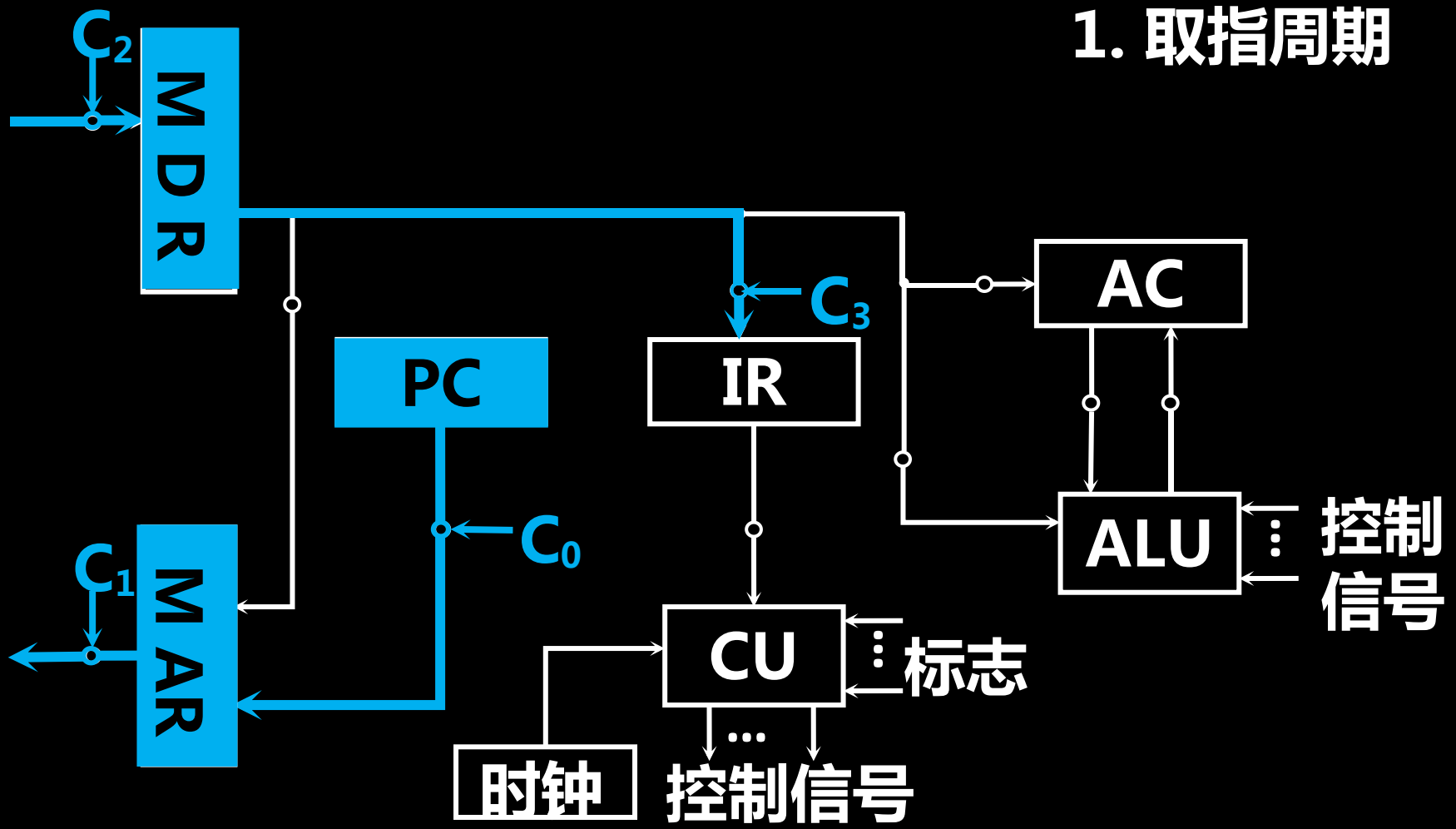
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

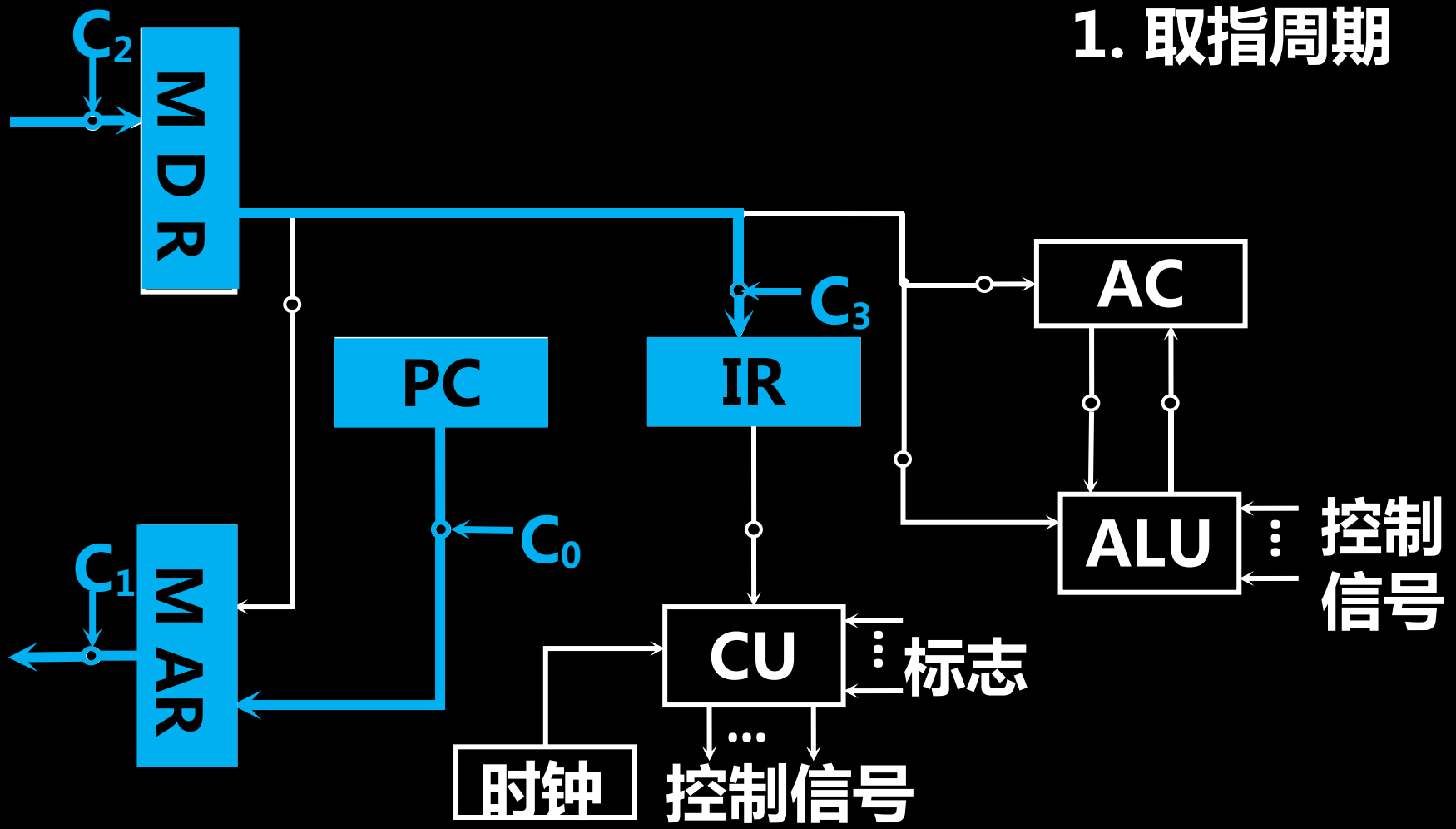
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

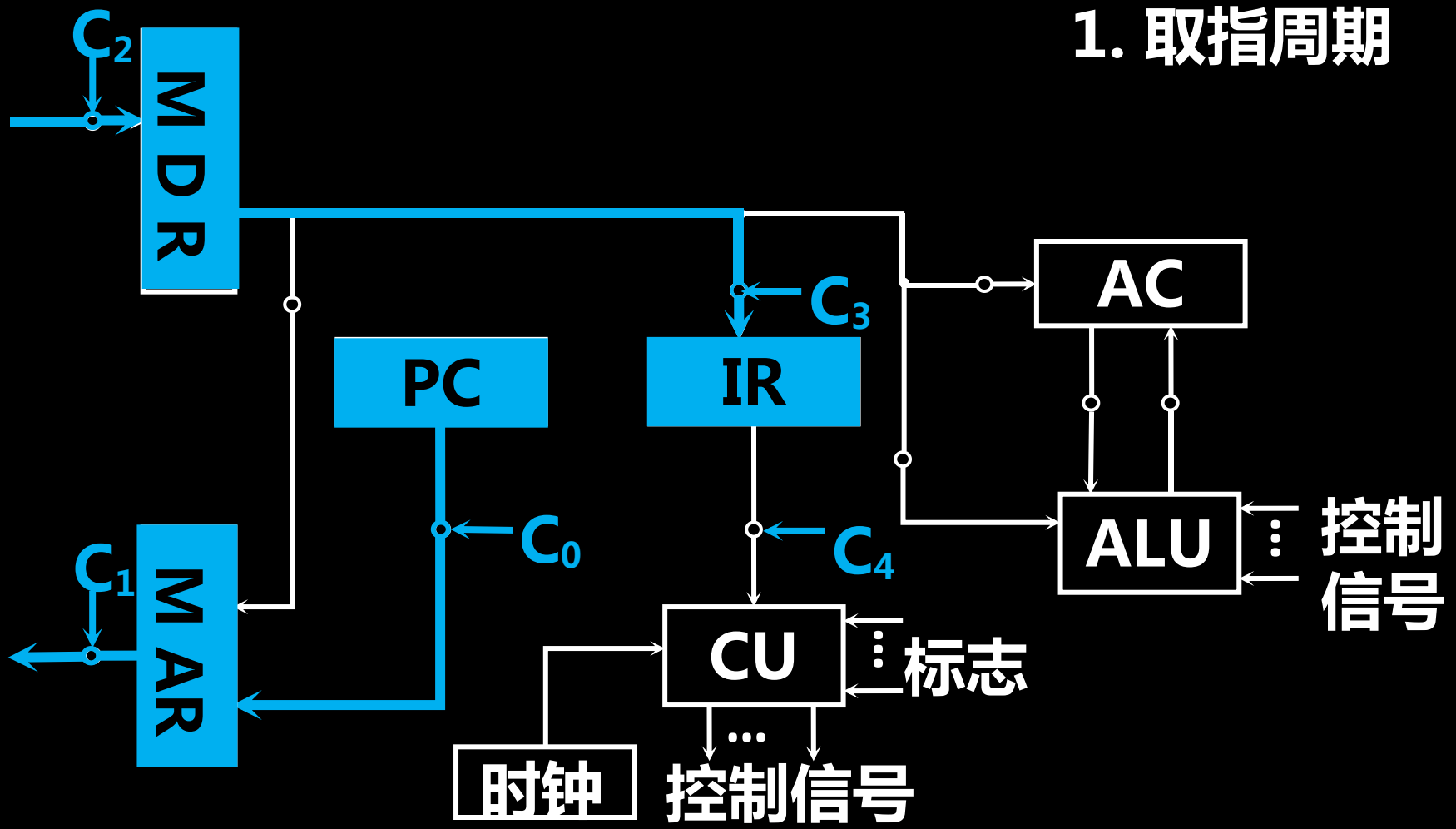
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

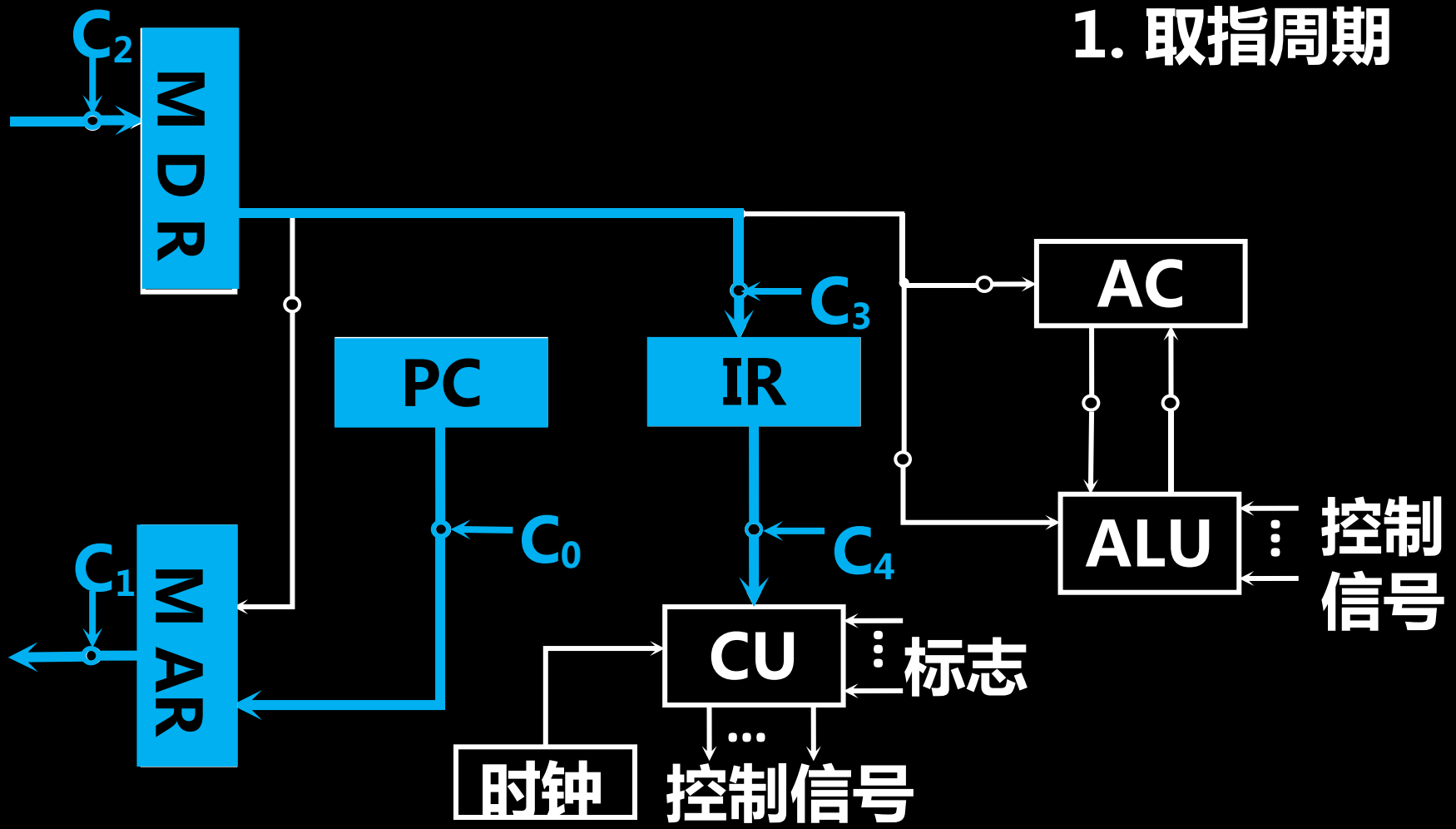
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

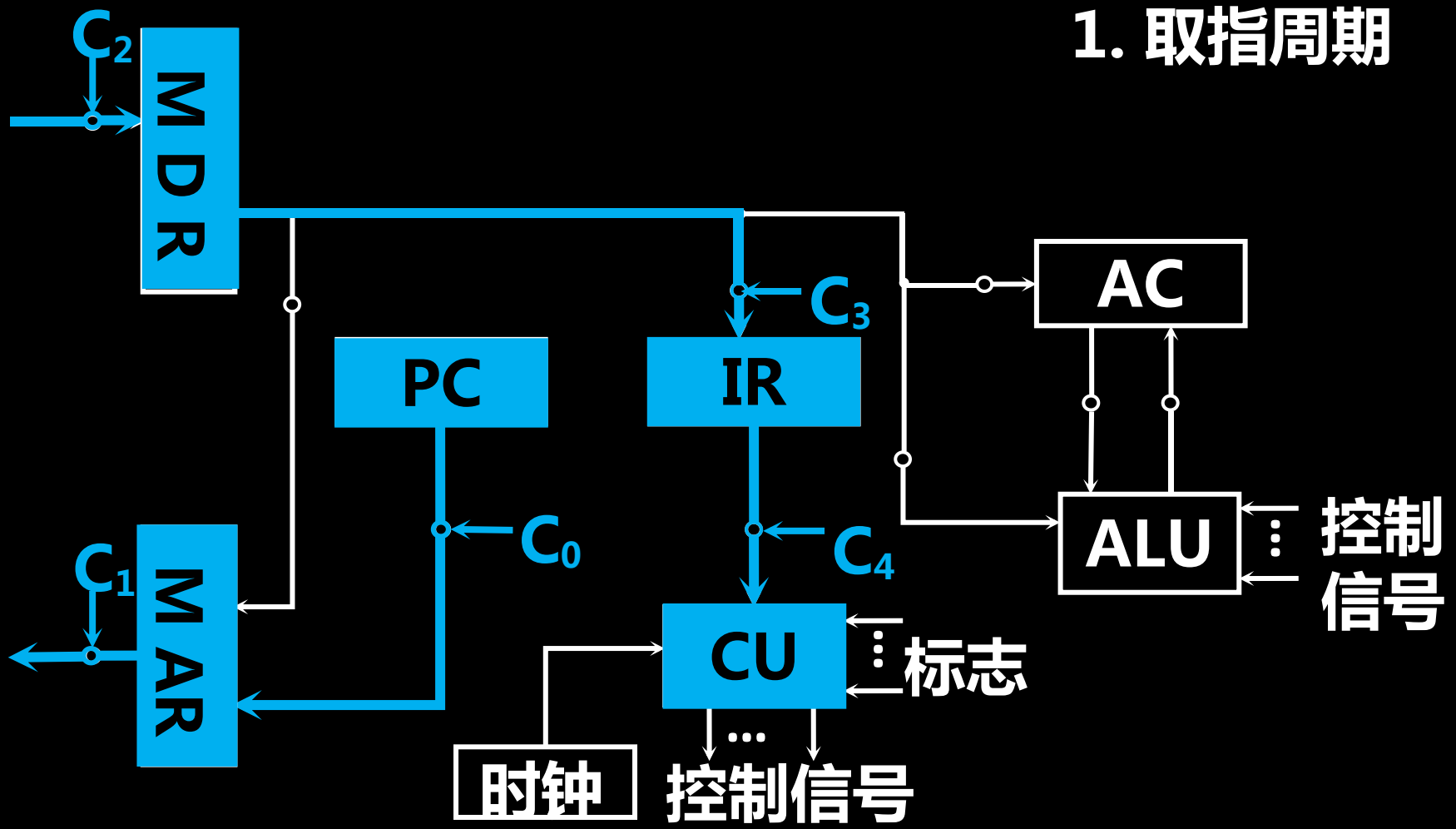
1. 取指周期



控制信号举例

不采用CPU内部总线的方式　以ADD @X 为例

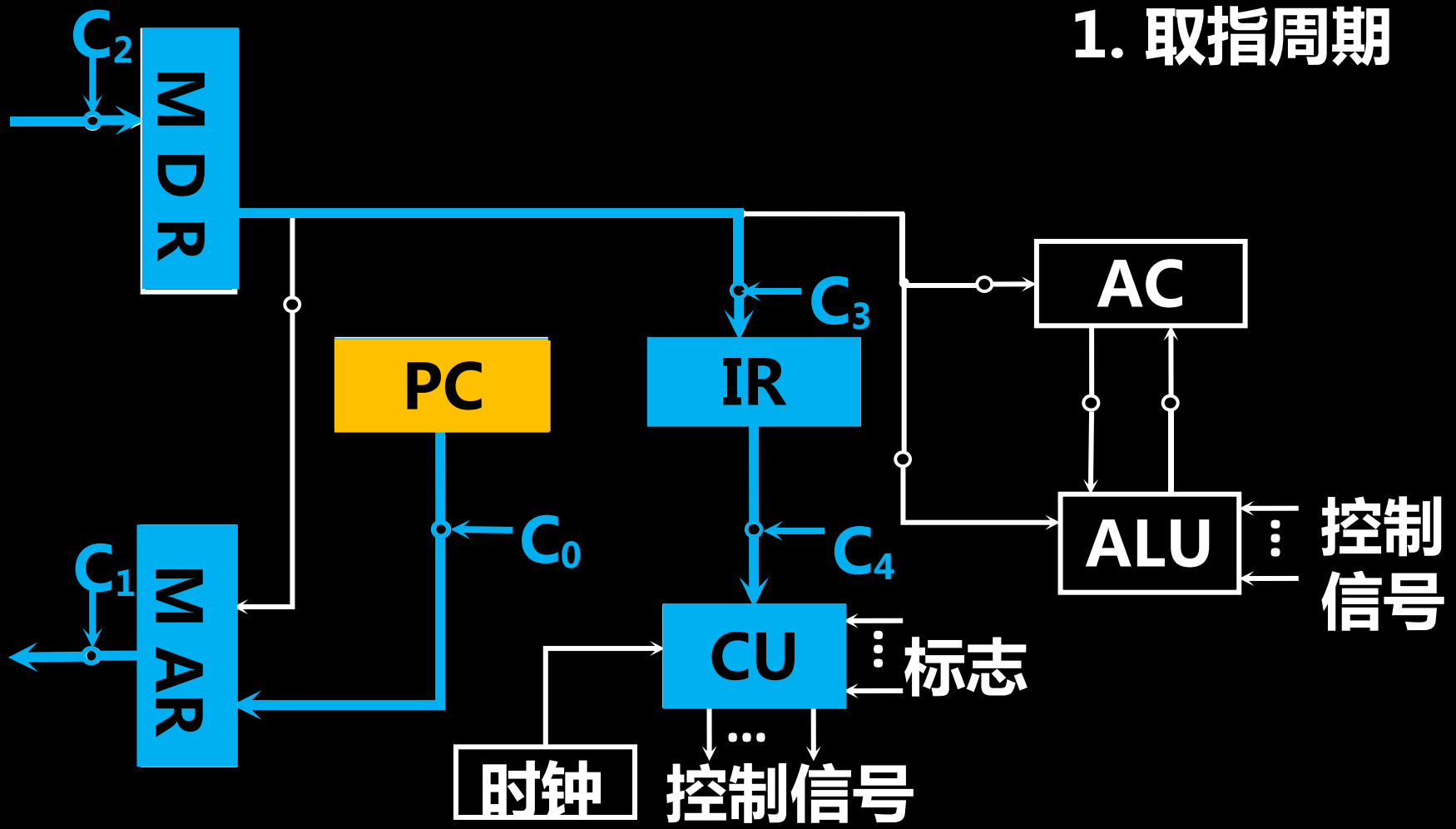
1. 取指周期



控制信号举例

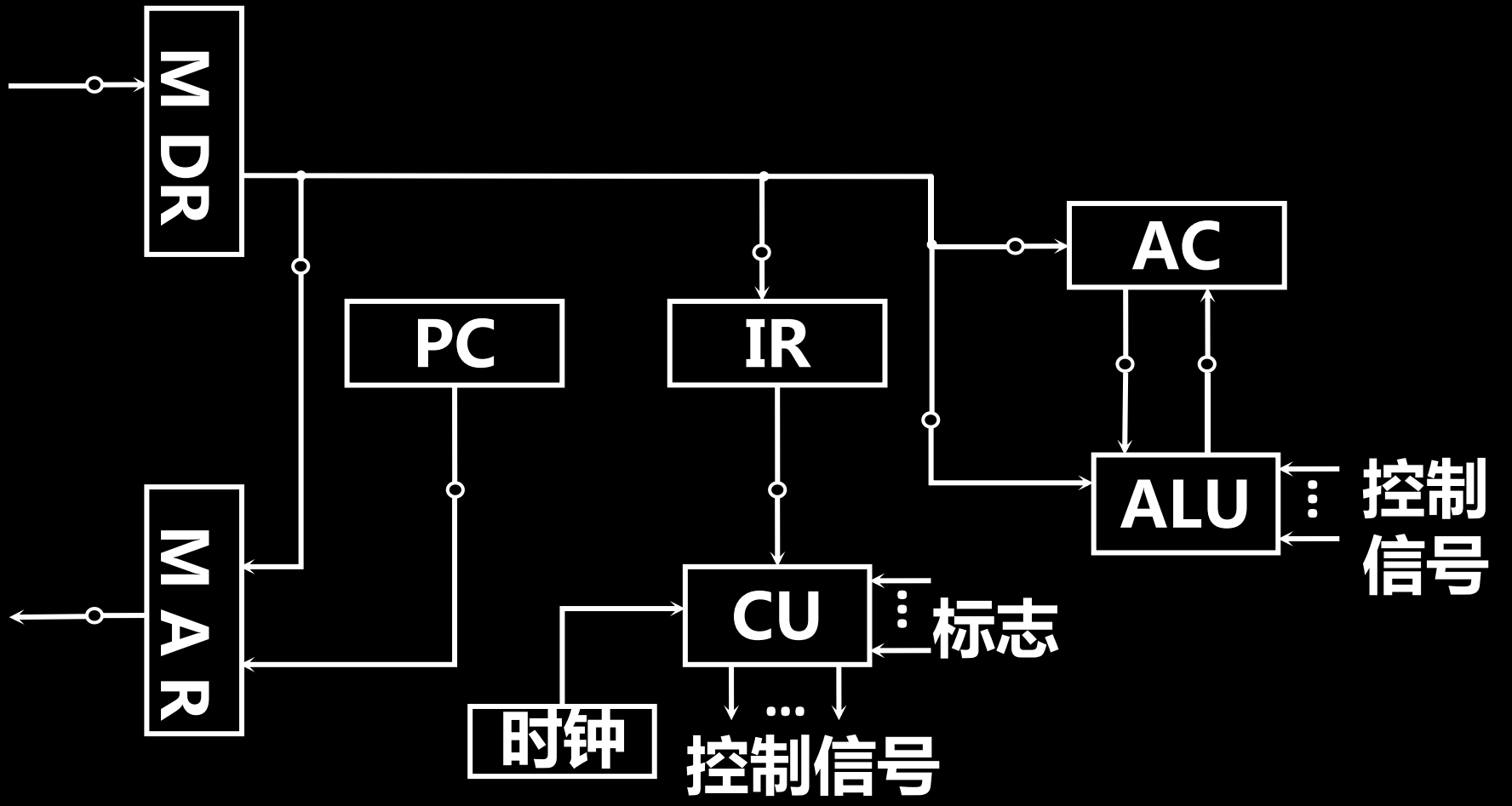
不采用CPU内部总线的方式　以ADD @X 为例

1. 取指周期



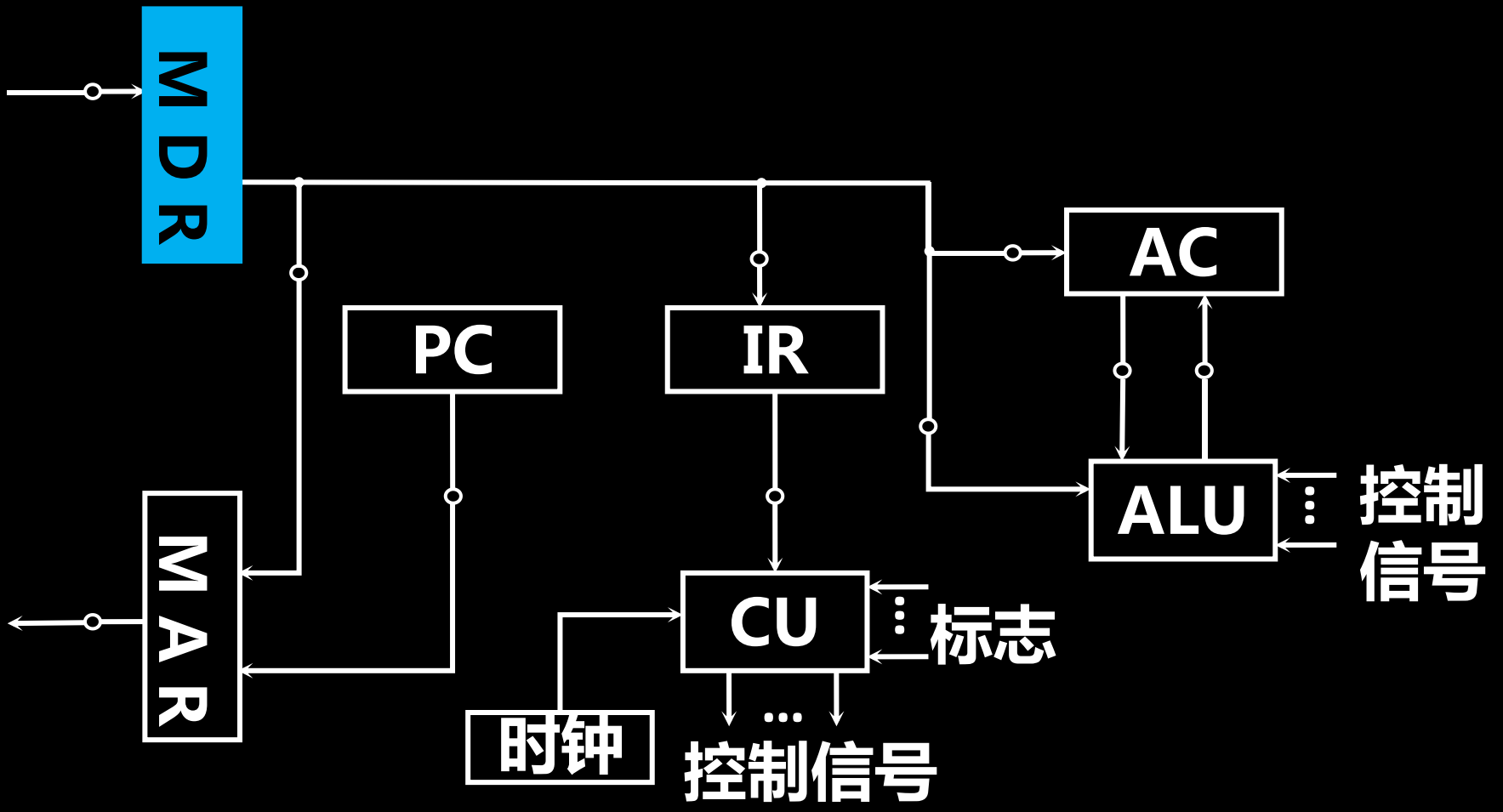
控制信号举例

2. 间址周期



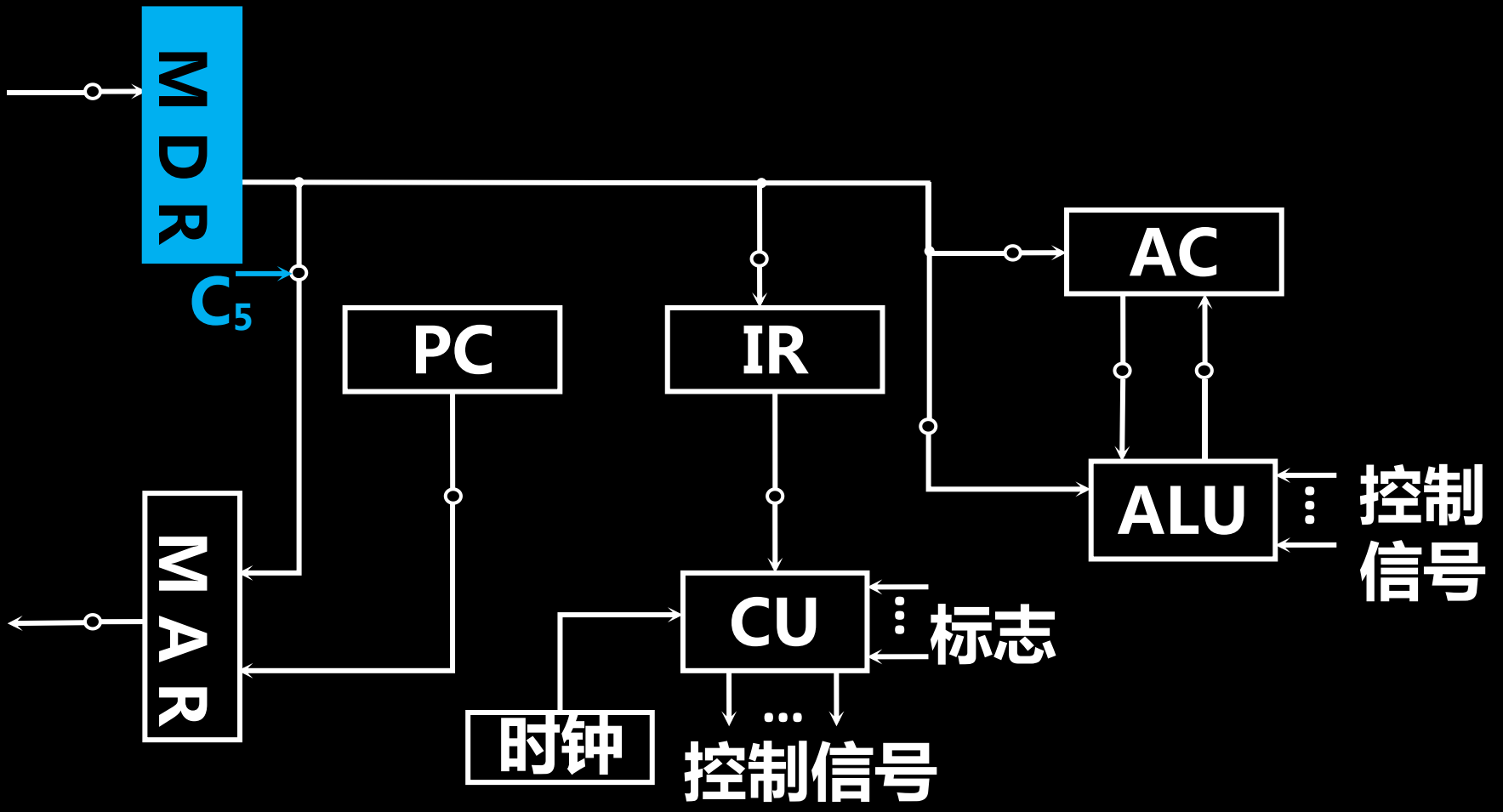
控制信号举例

2. 间址周期



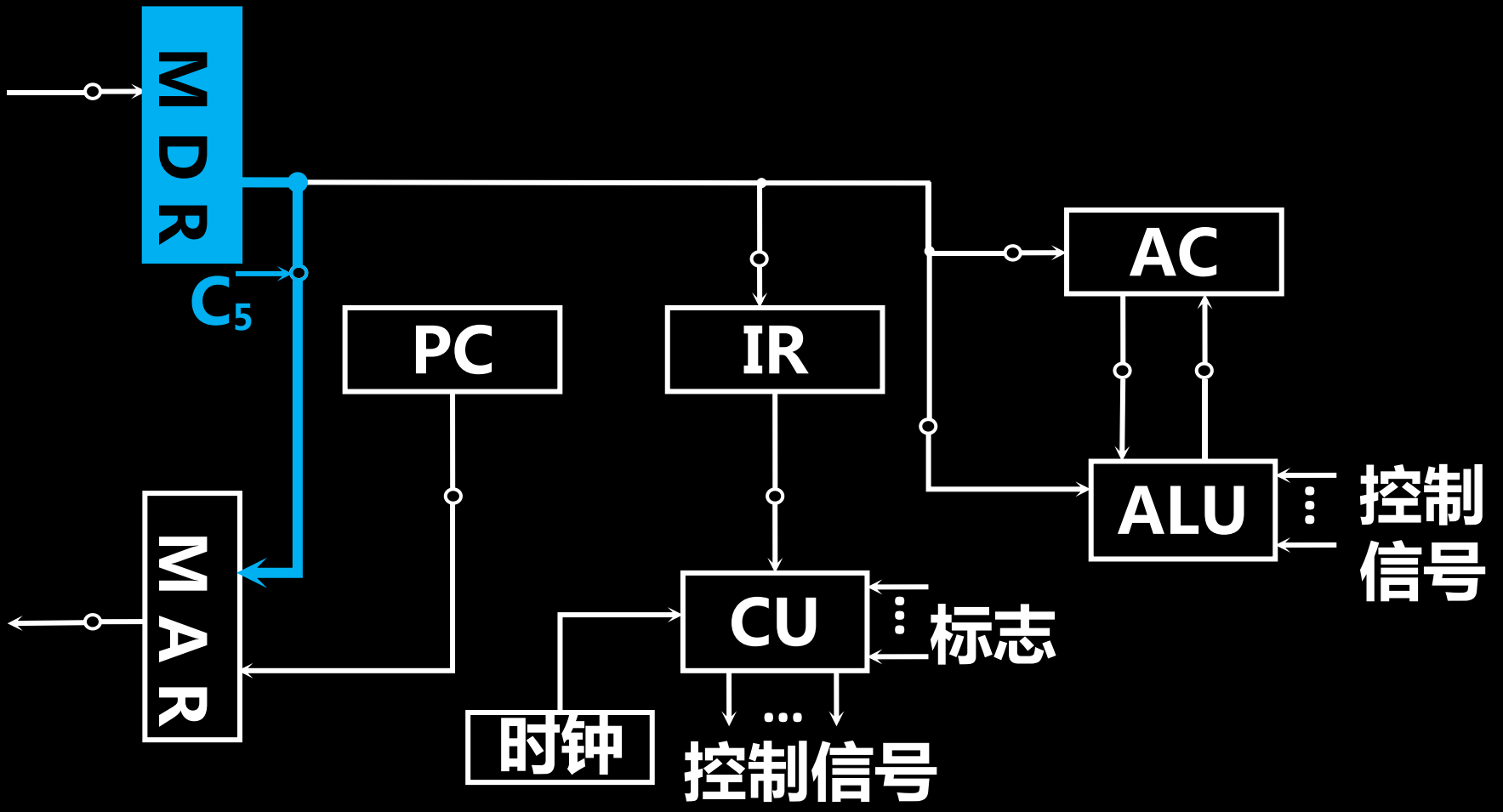
控制信号举例

2. 间址周期



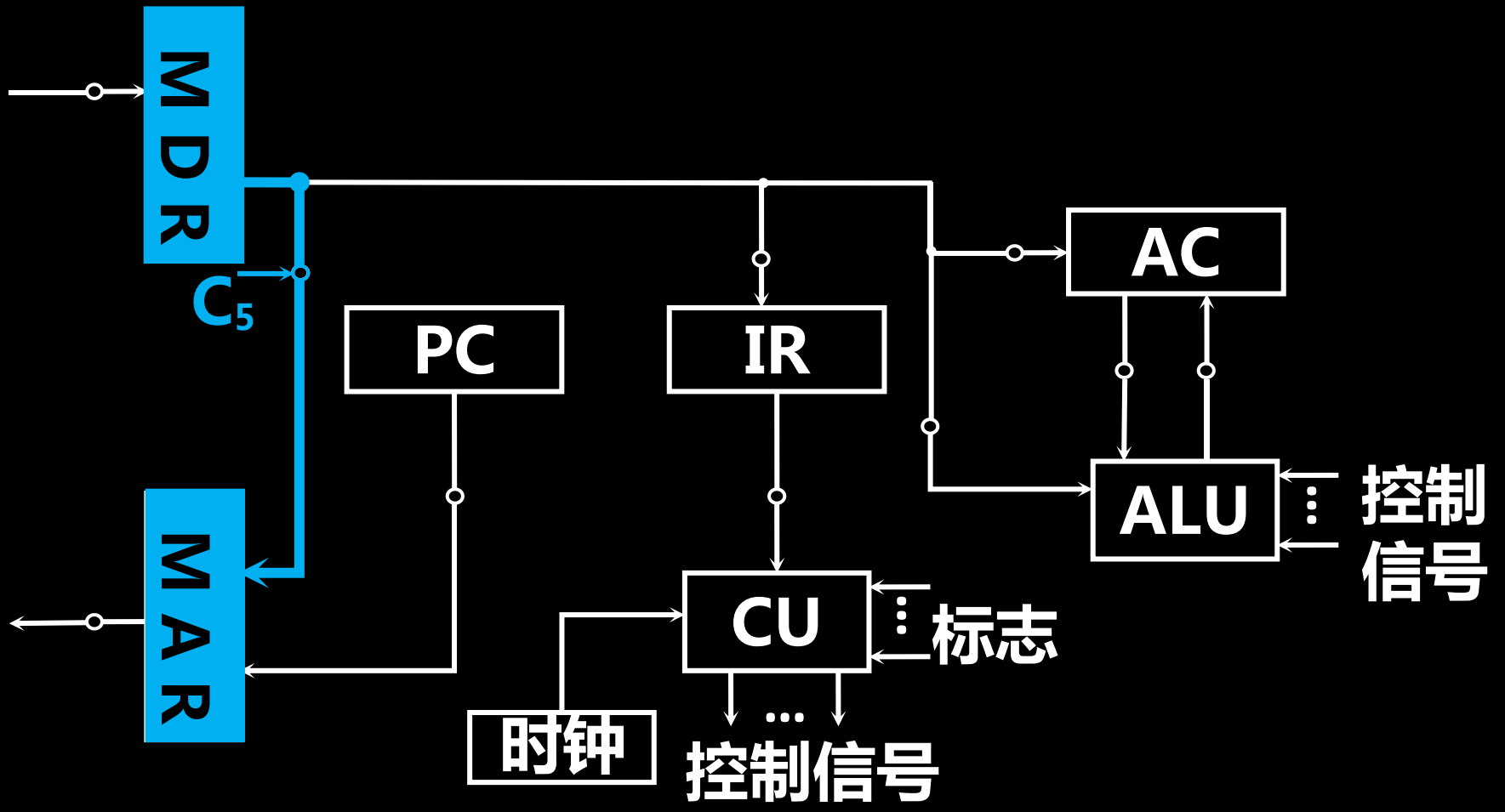
控制信号举例

2. 间址周期



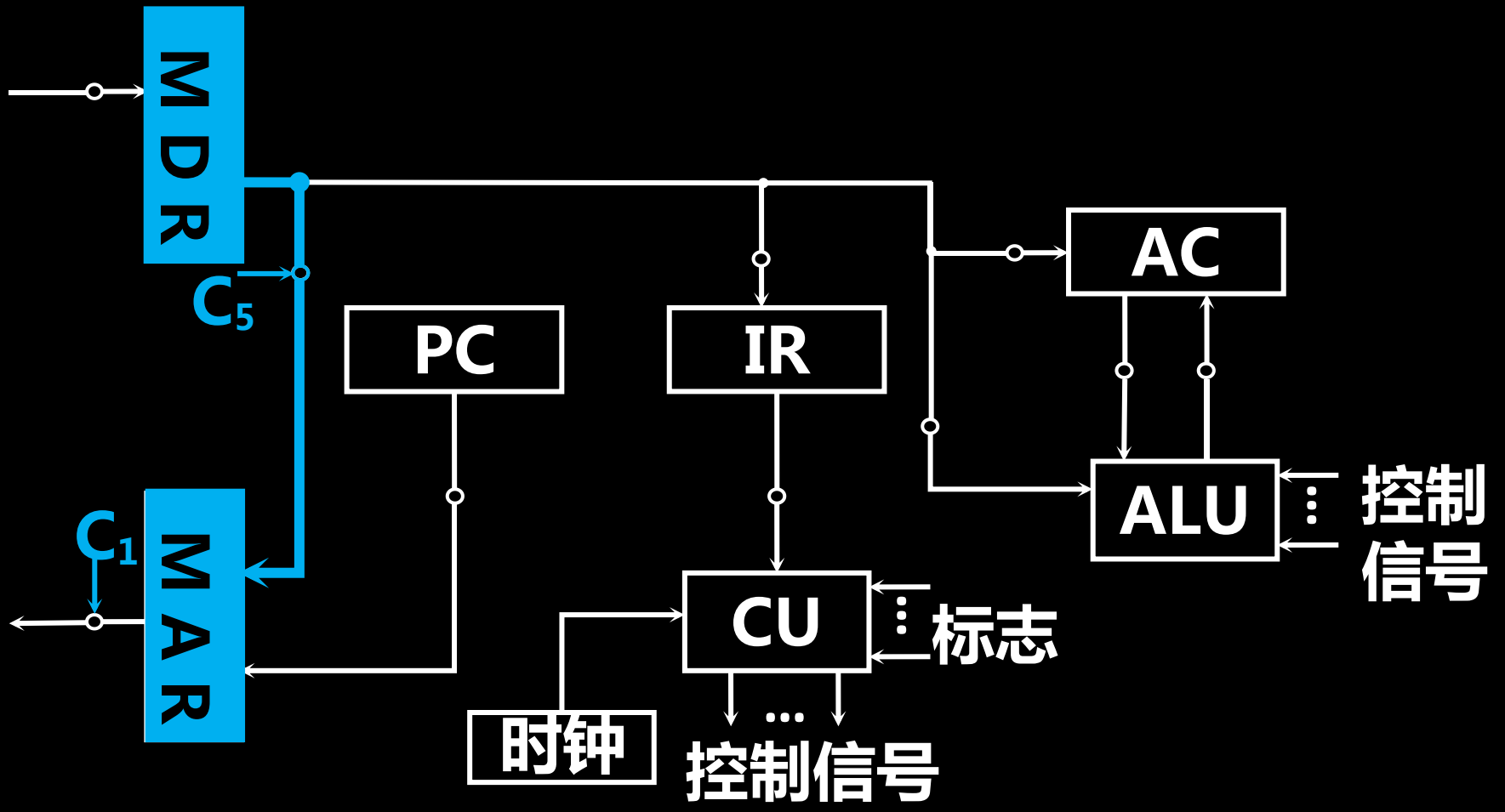
控制信号举例

2. 间址周期



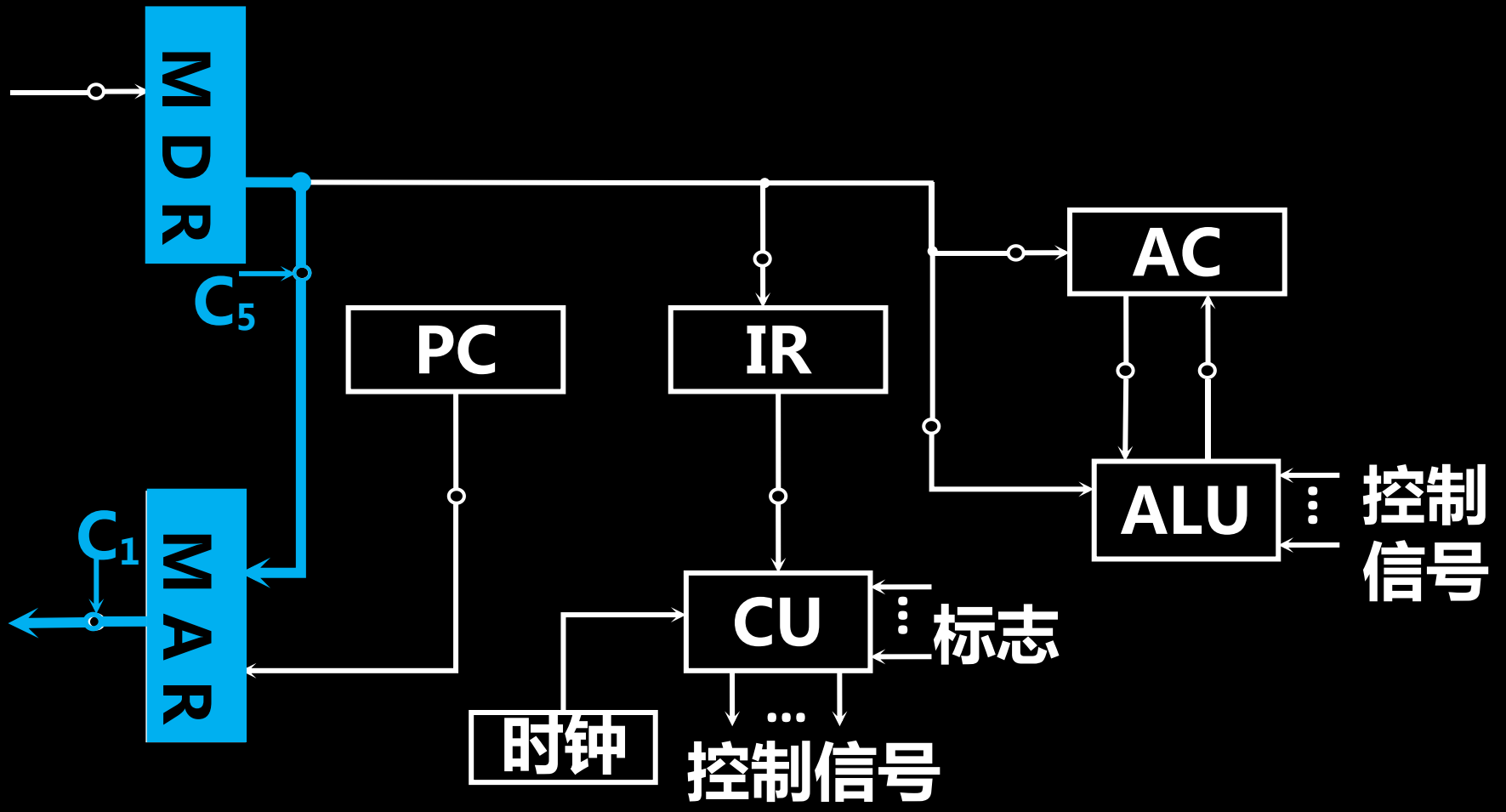
控制信号举例

2. 间址周期



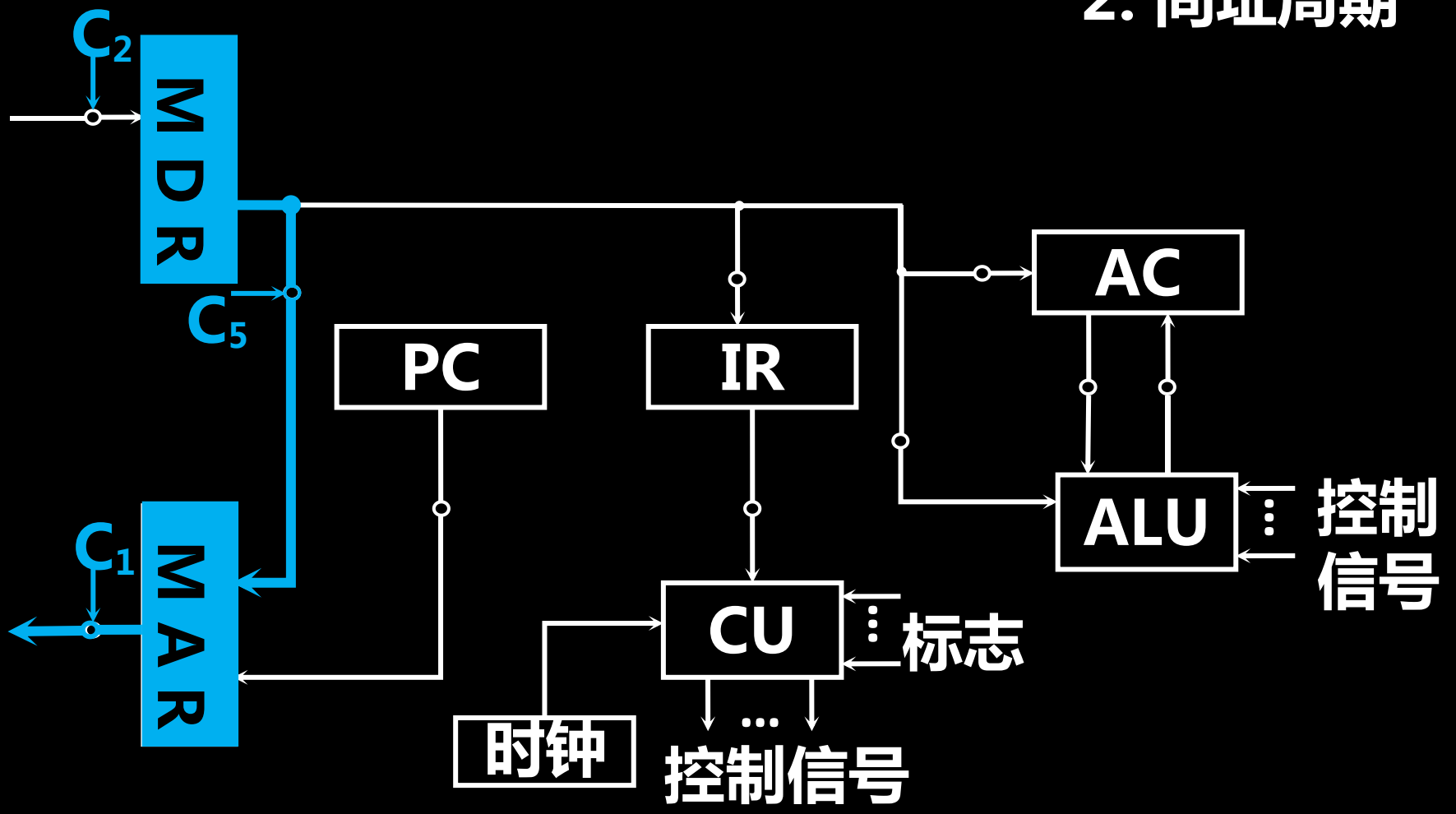
控制信号举例

2. 间址周期

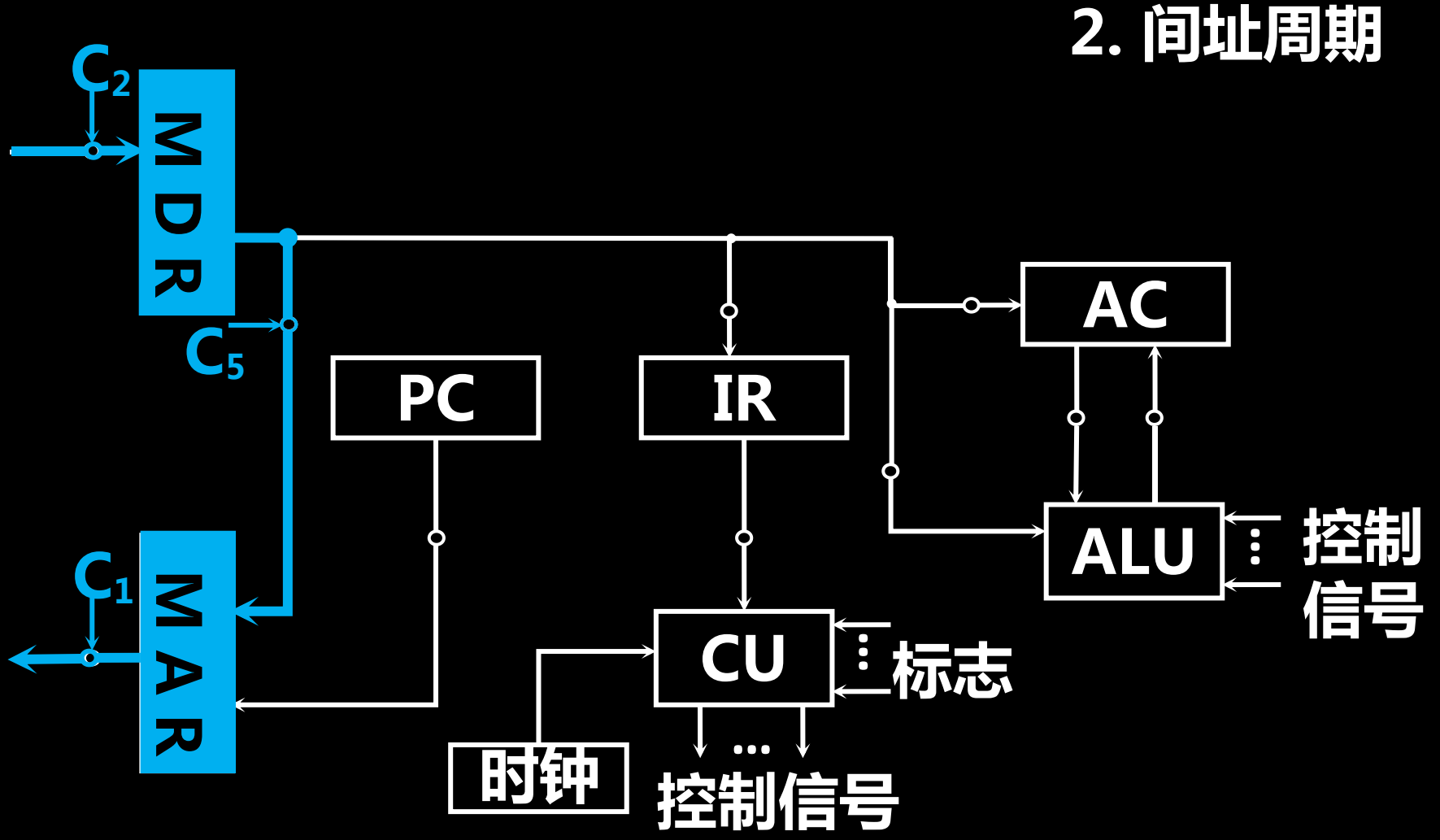


控制信号举例

2. 间址周期

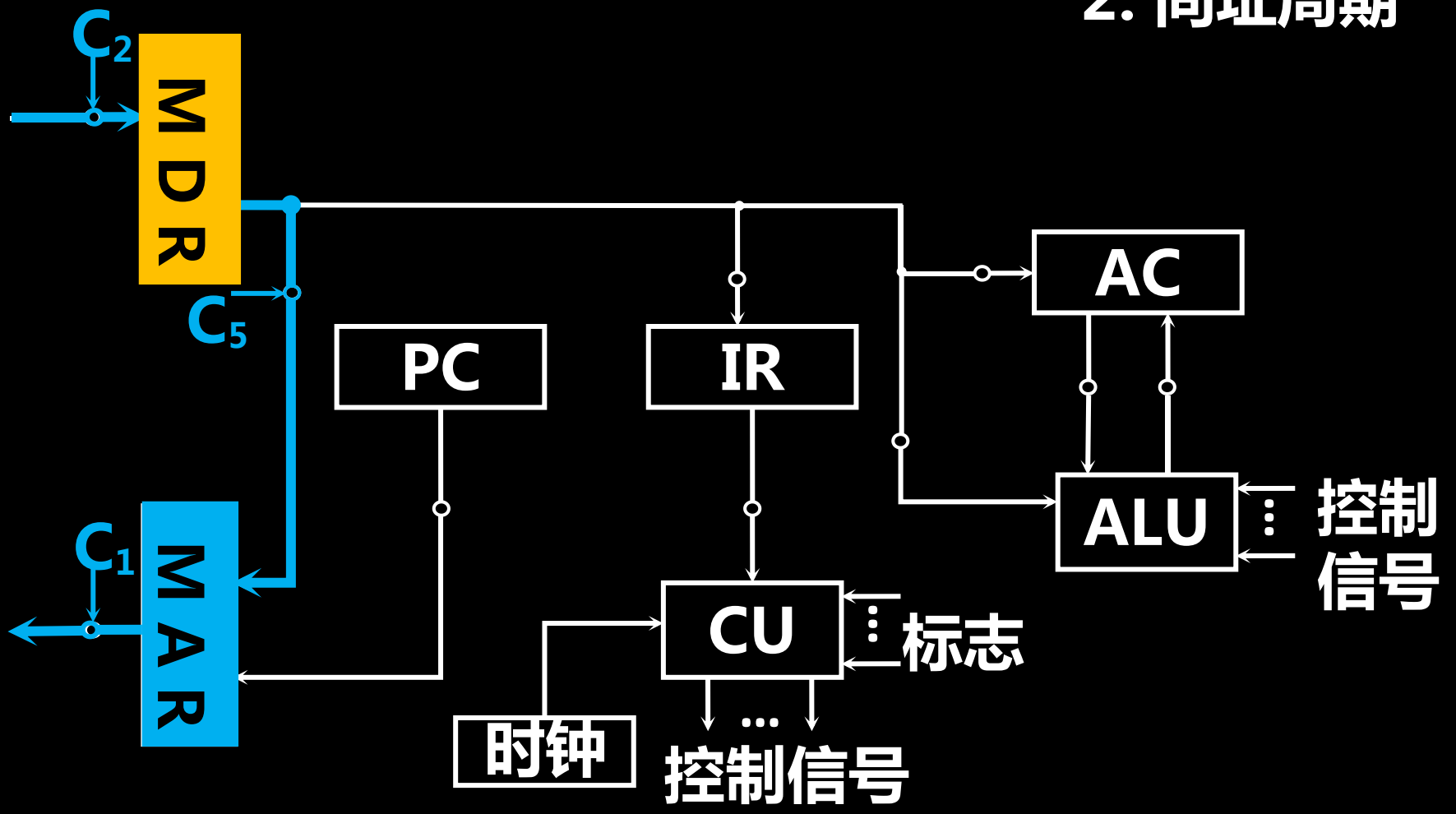


控制信号举例



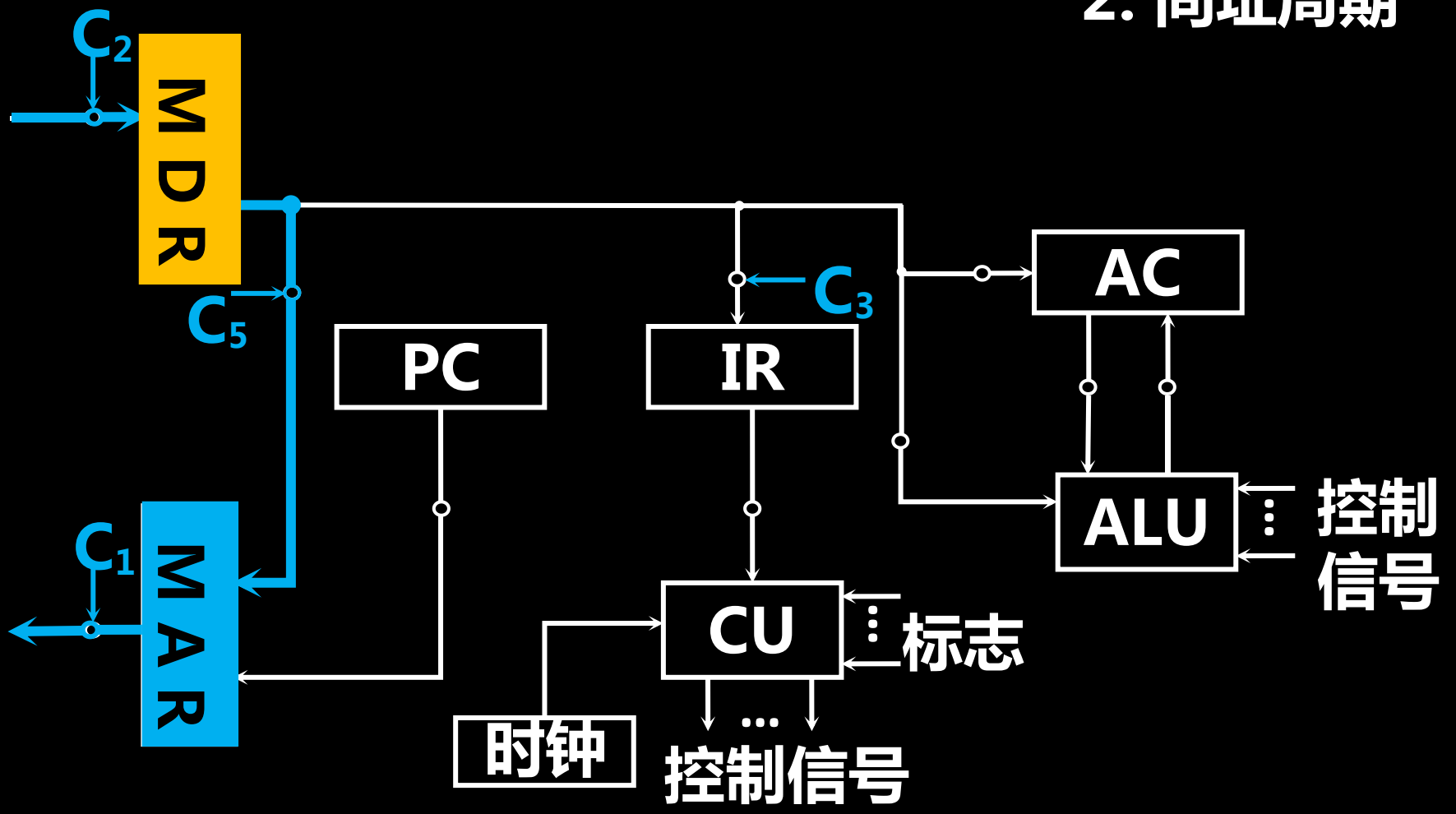
控制信号举例

2. 间址周期

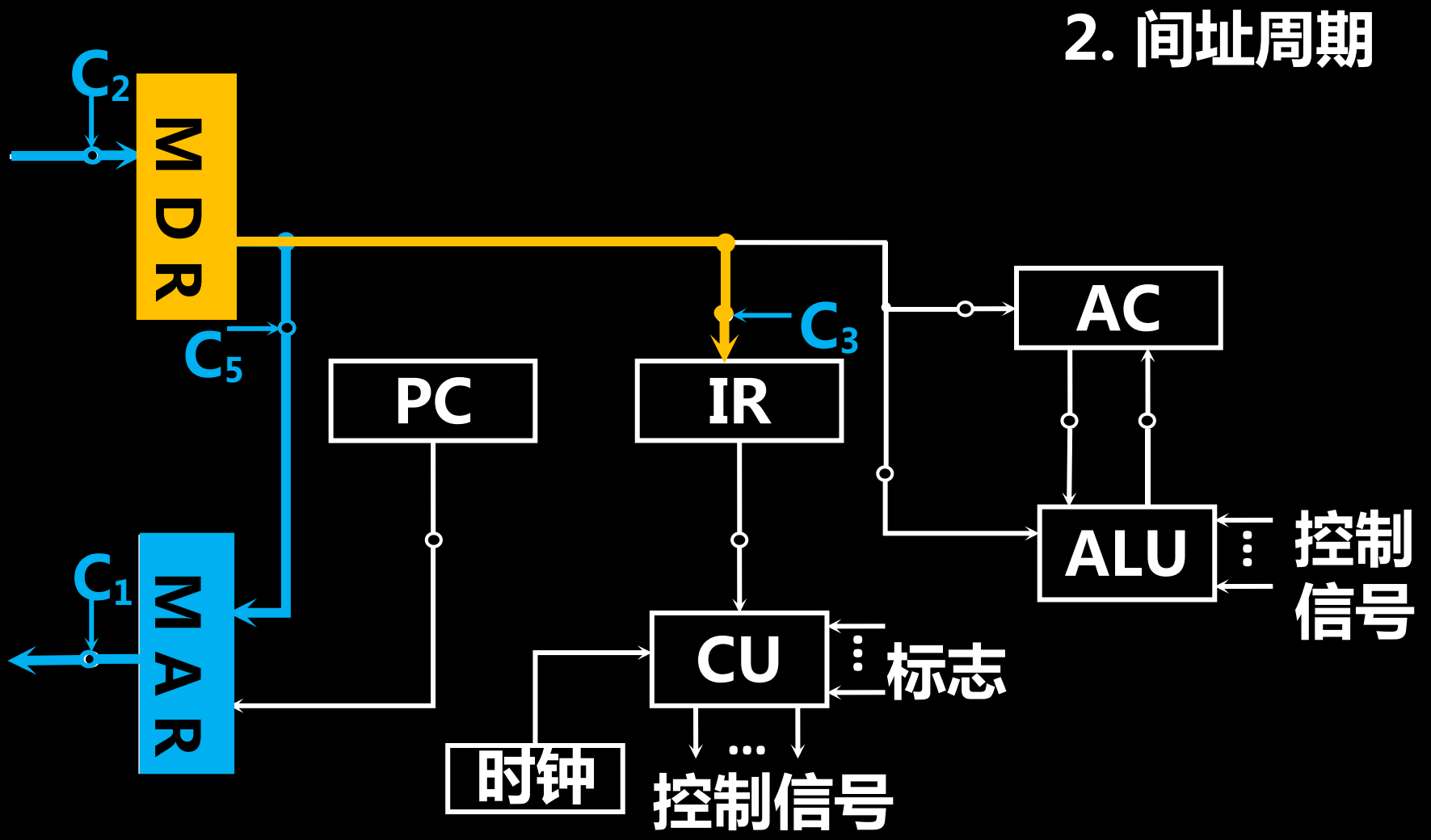


控制信号举例

2. 间址周期

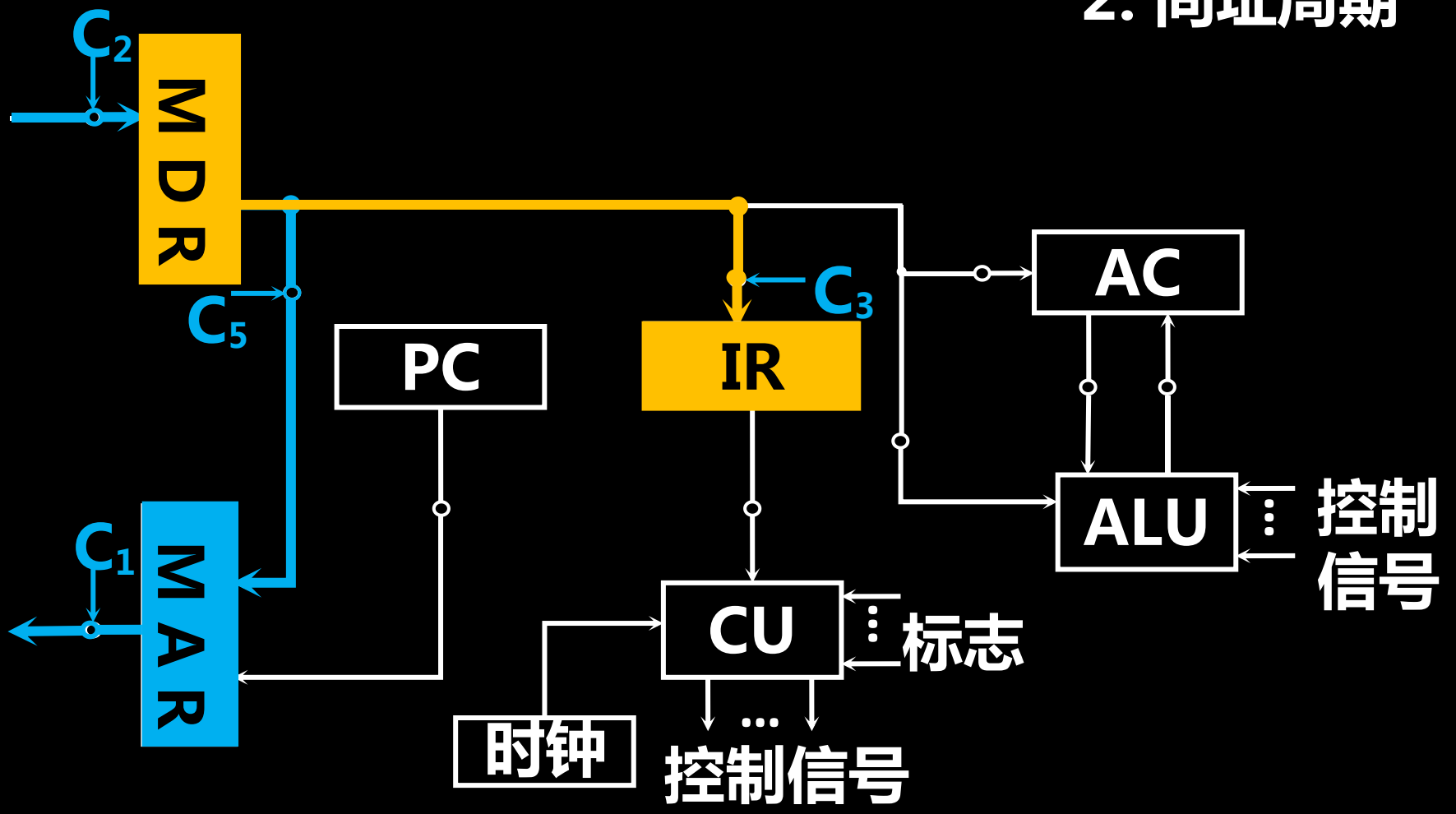


控制信号举例



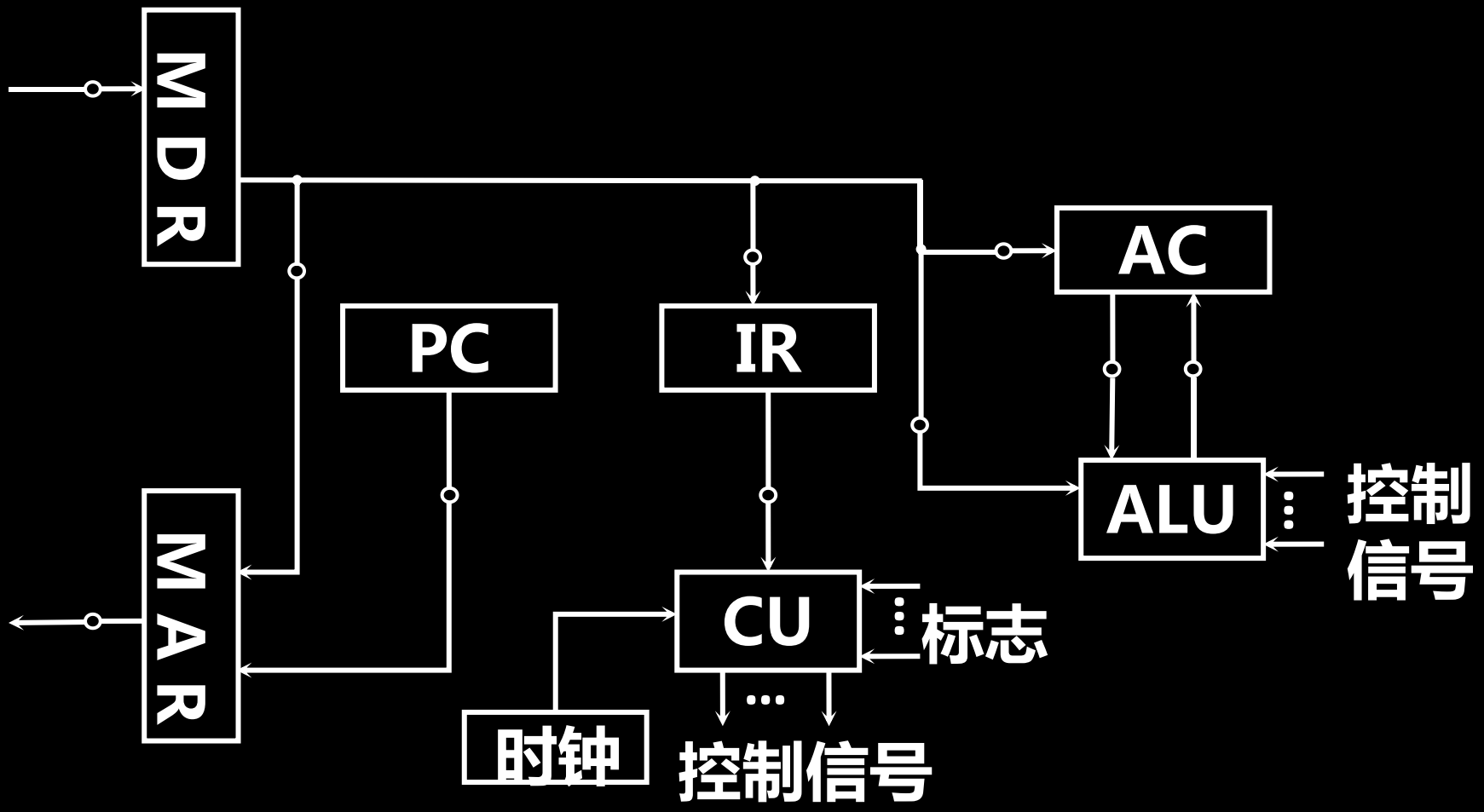
控制信号举例

2. 间址周期



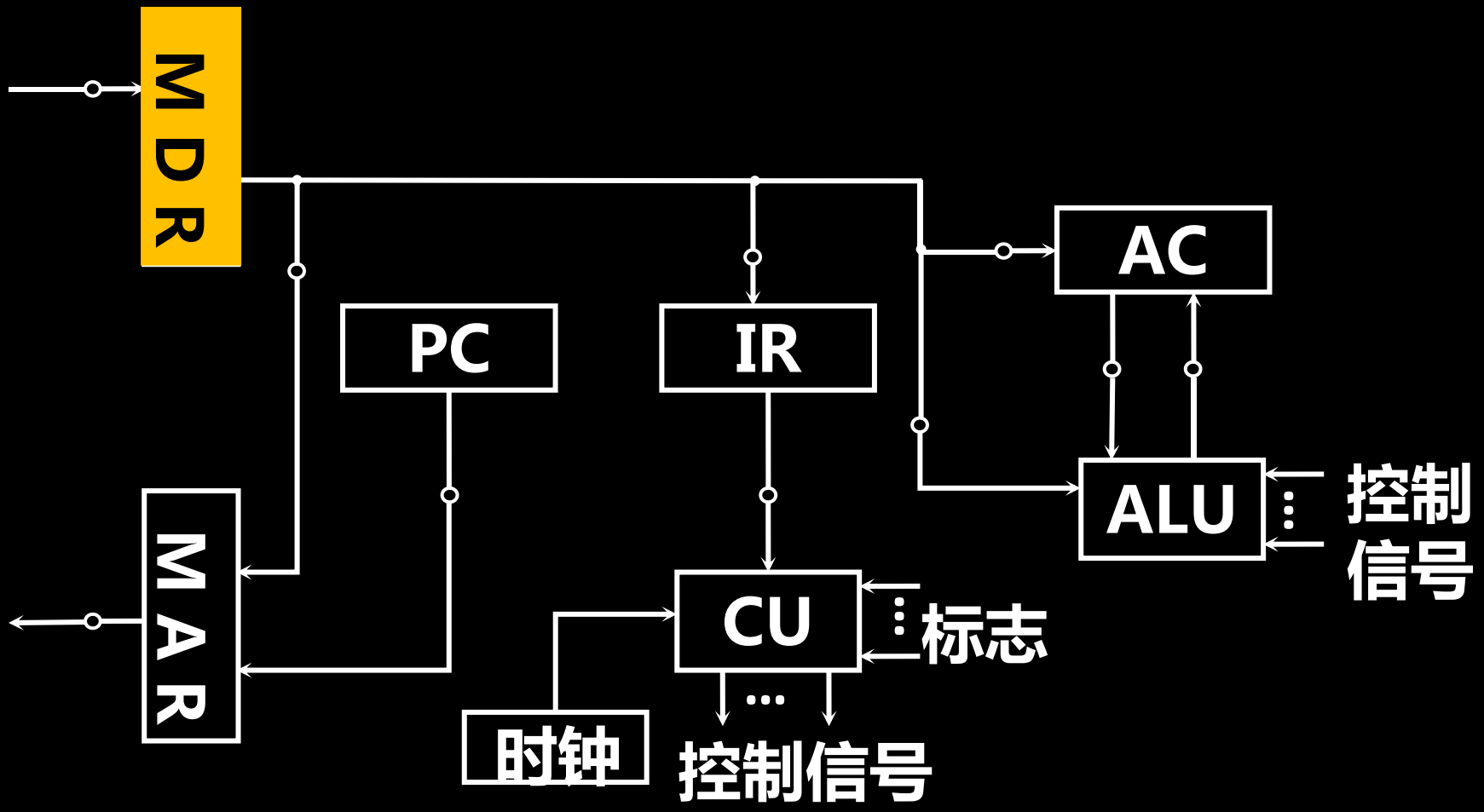
控制信号举例

3. 执行周期



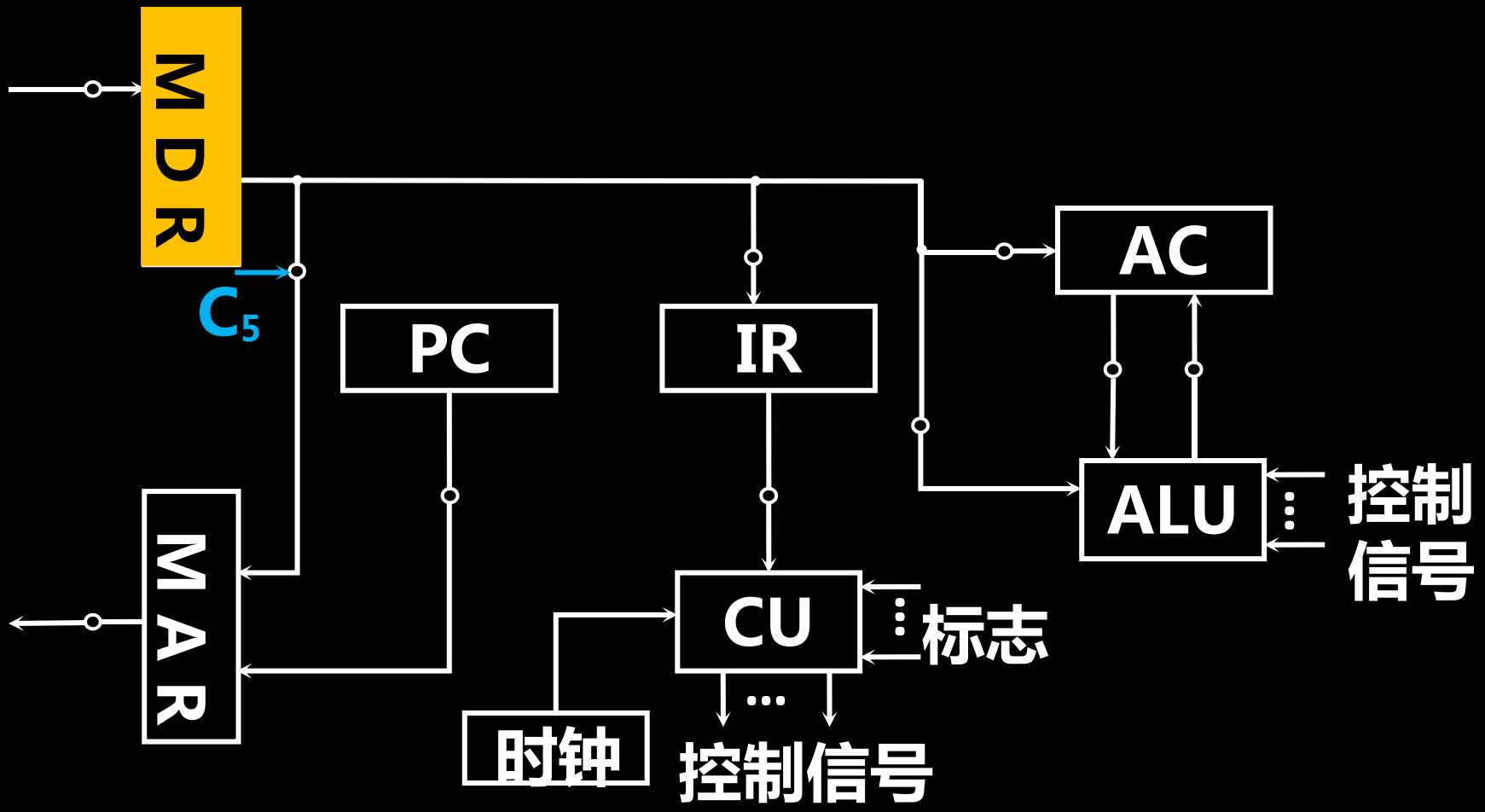
控制信号举例

3. 执行周期



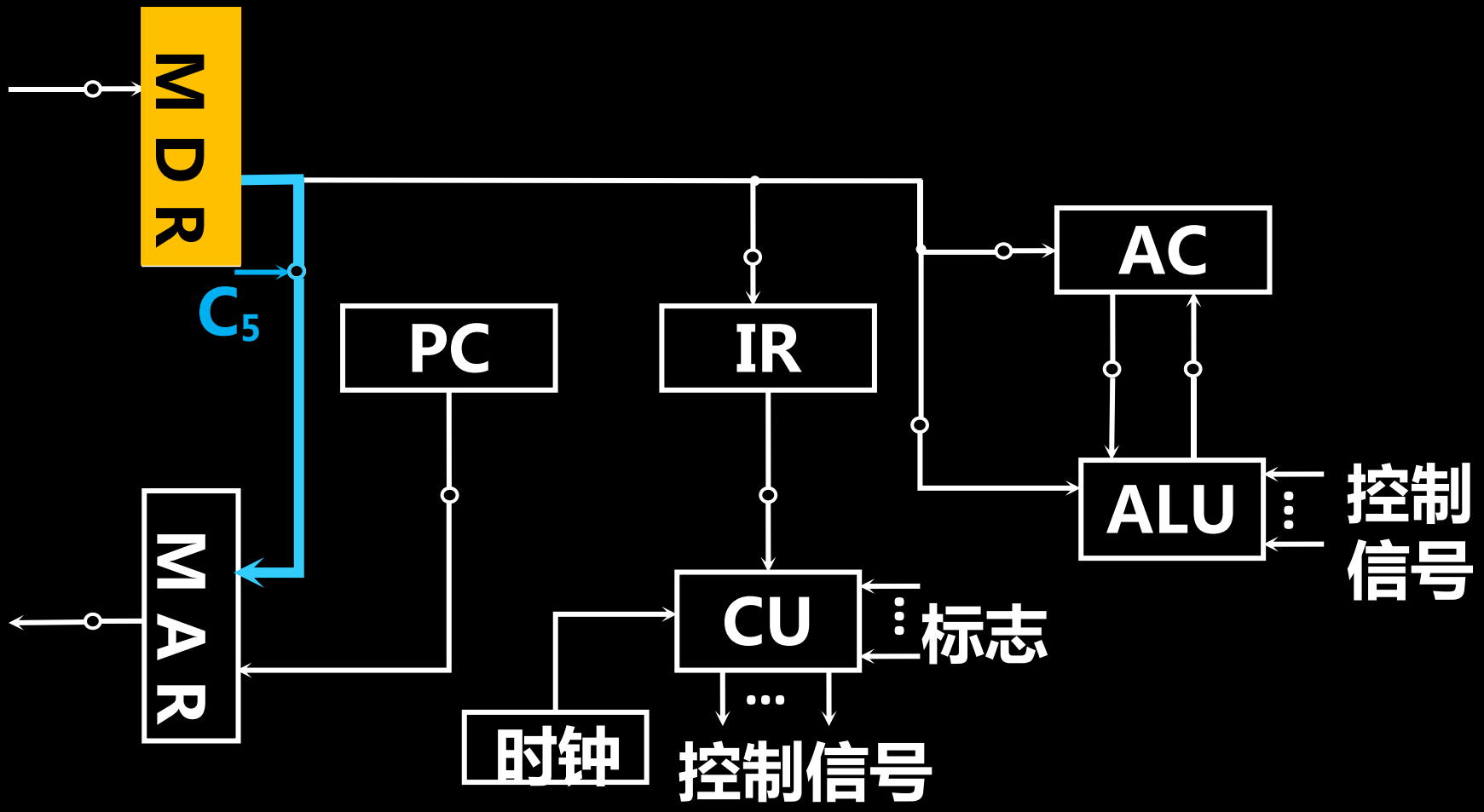
控制信号举例

3. 执行周期



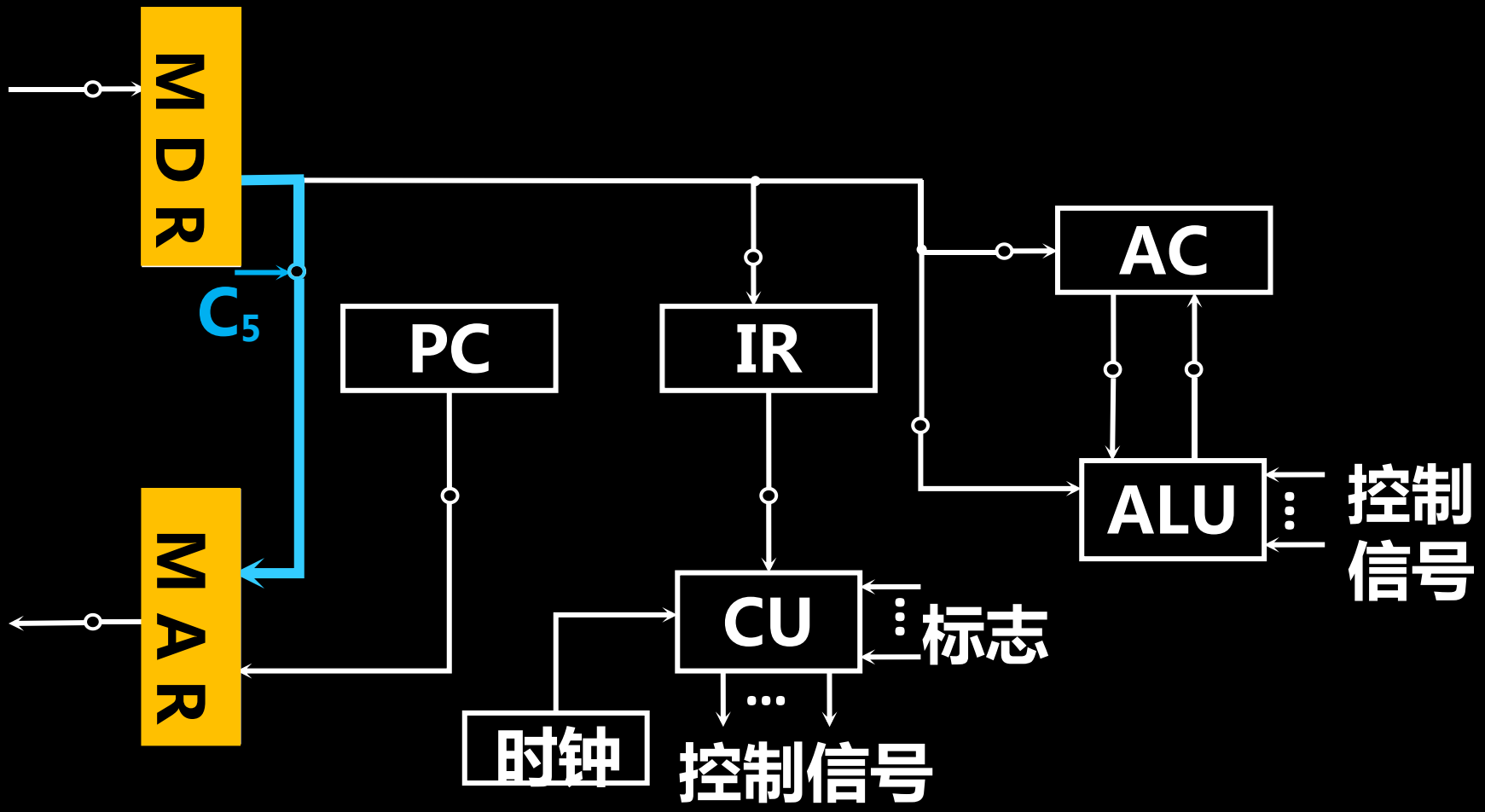
控制信号举例

3. 执行周期



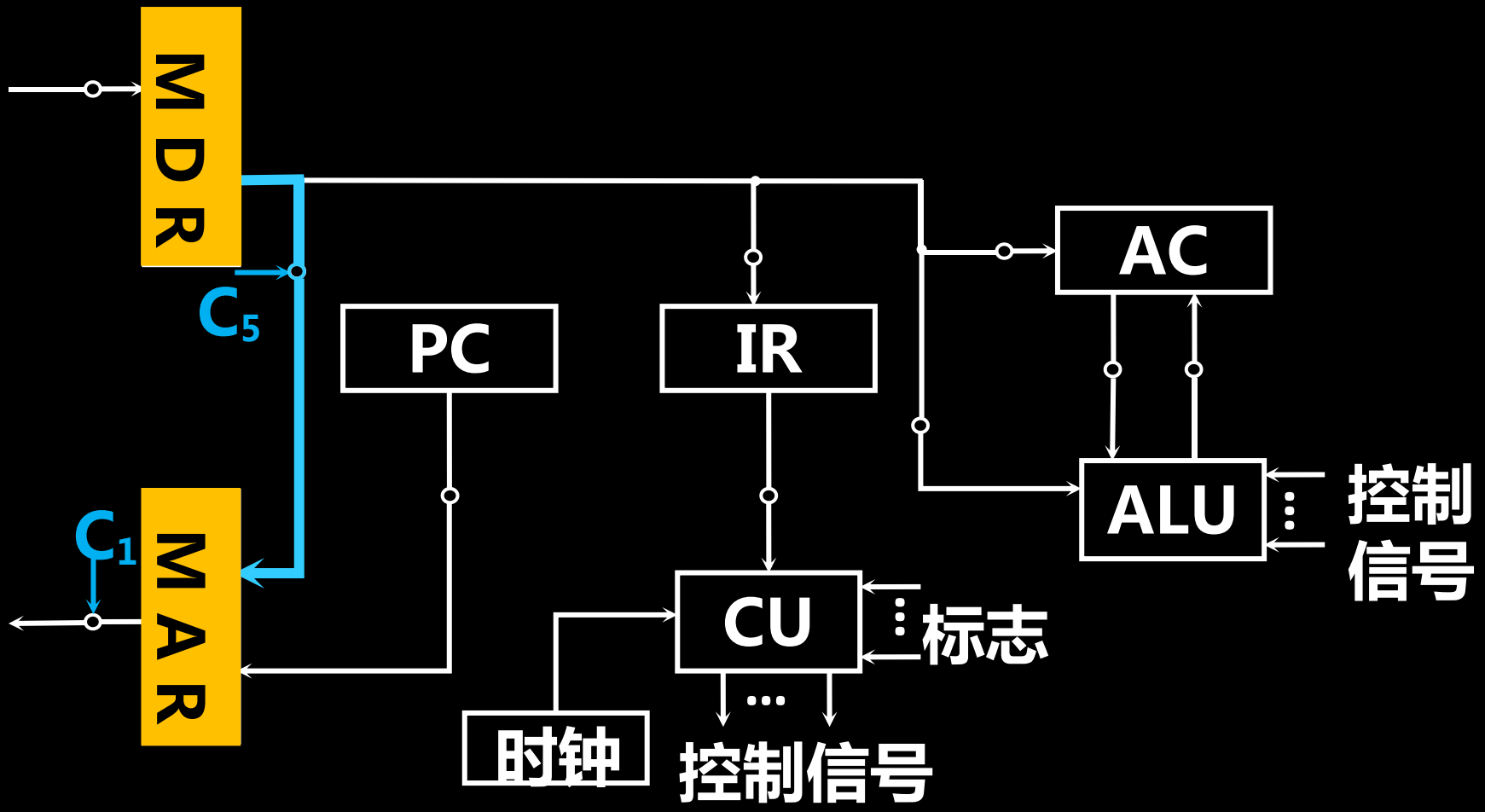
控制信号举例

3. 执行周期



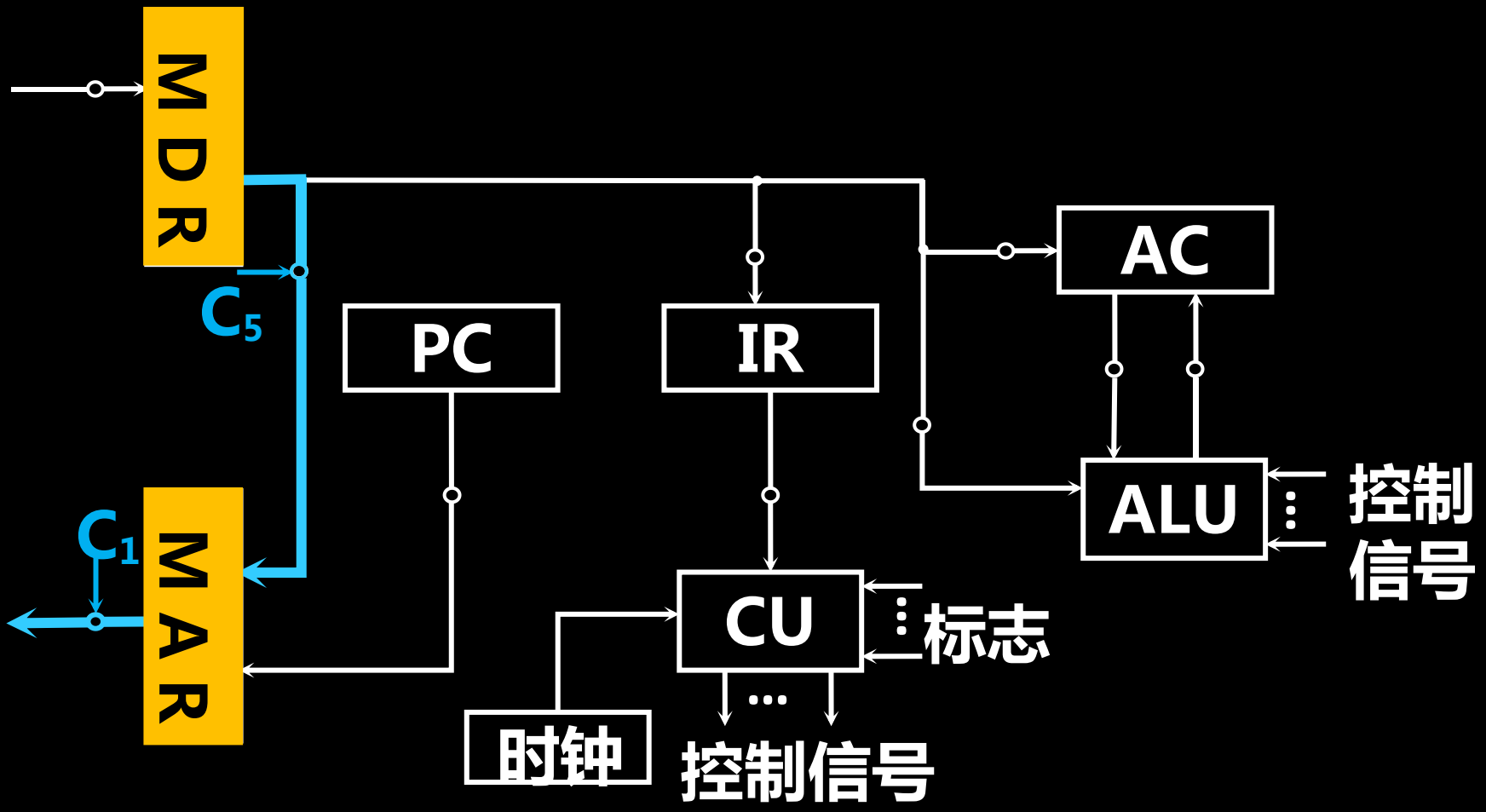
控制信号举例

3. 执行周期

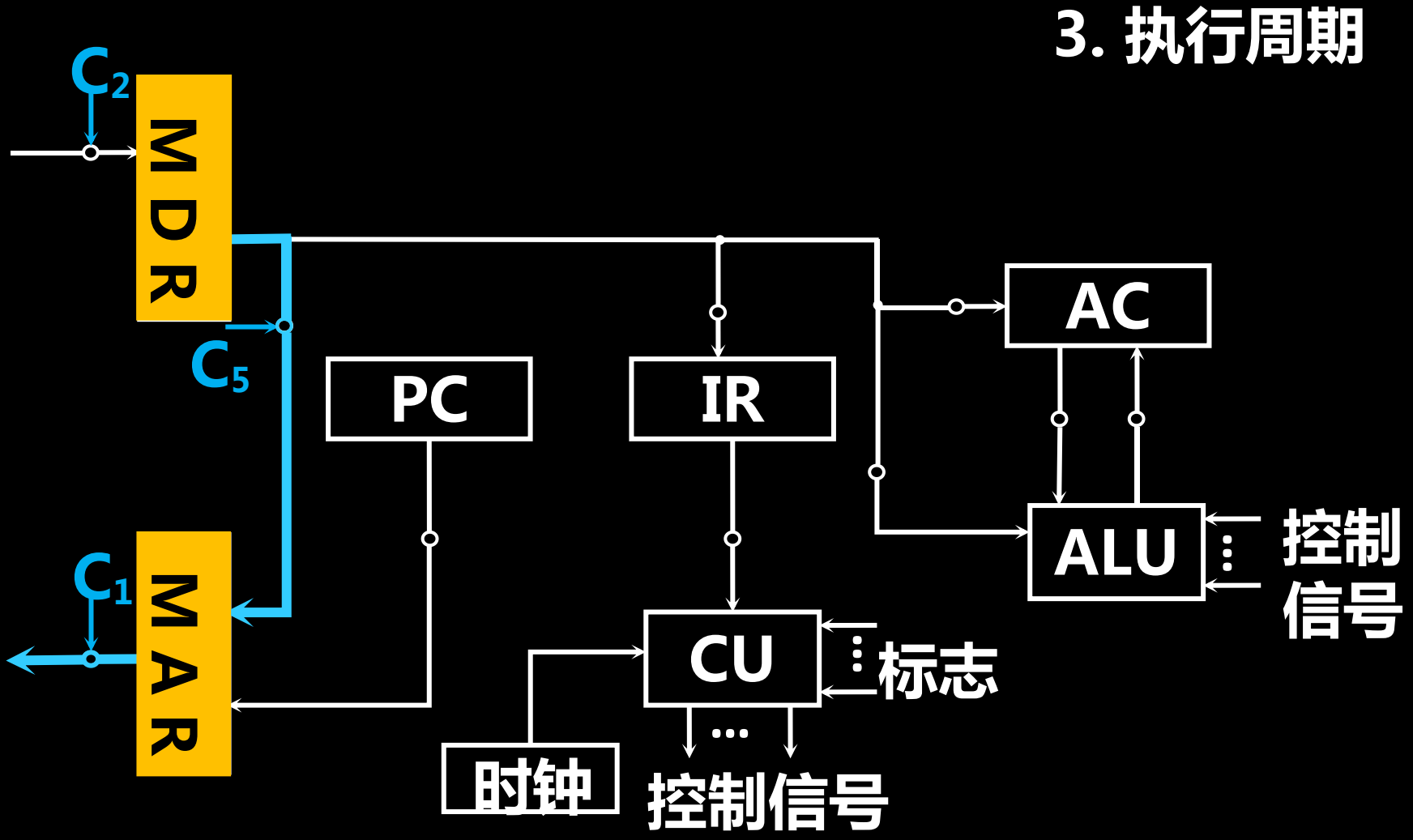


控制信号举例

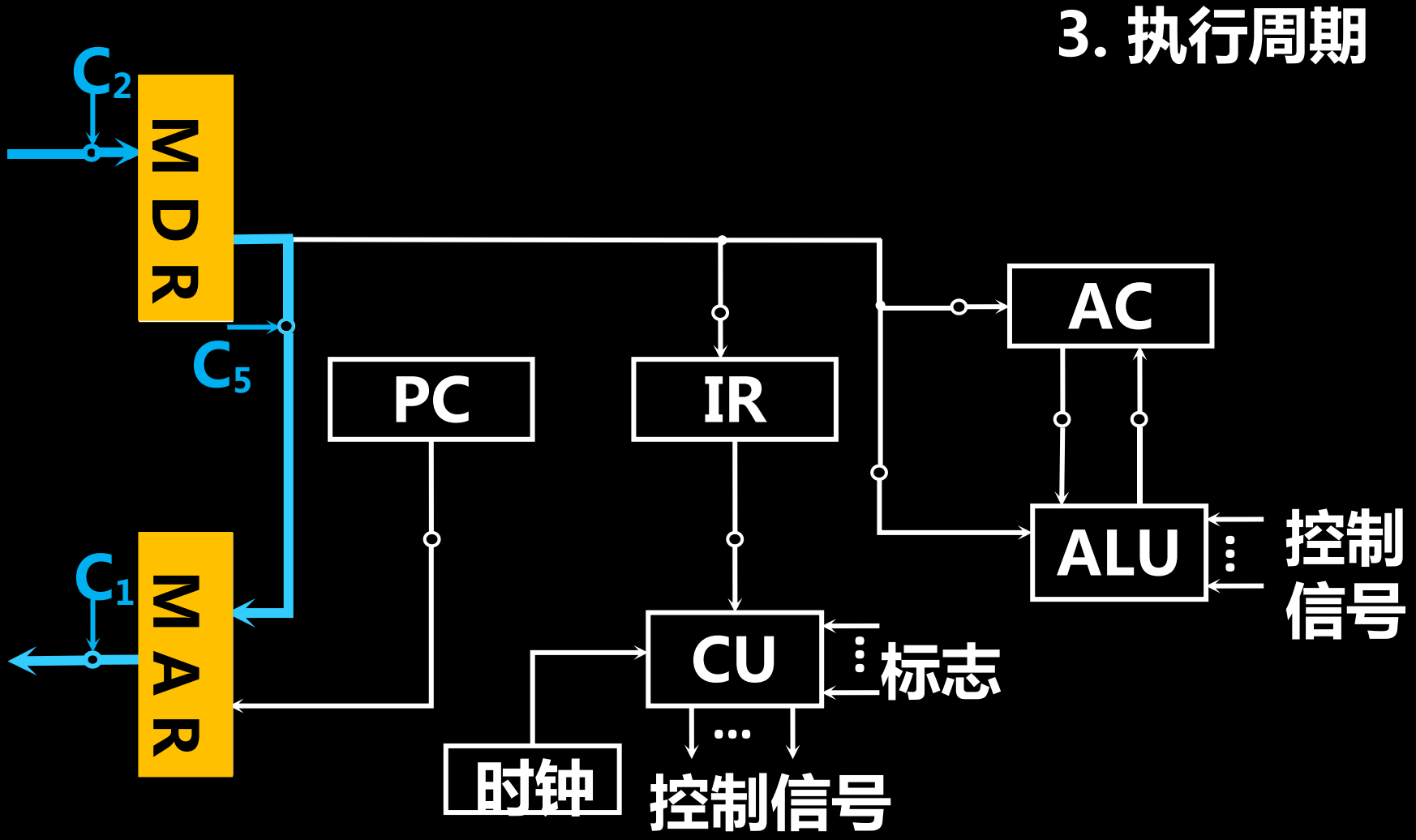
3. 执行周期



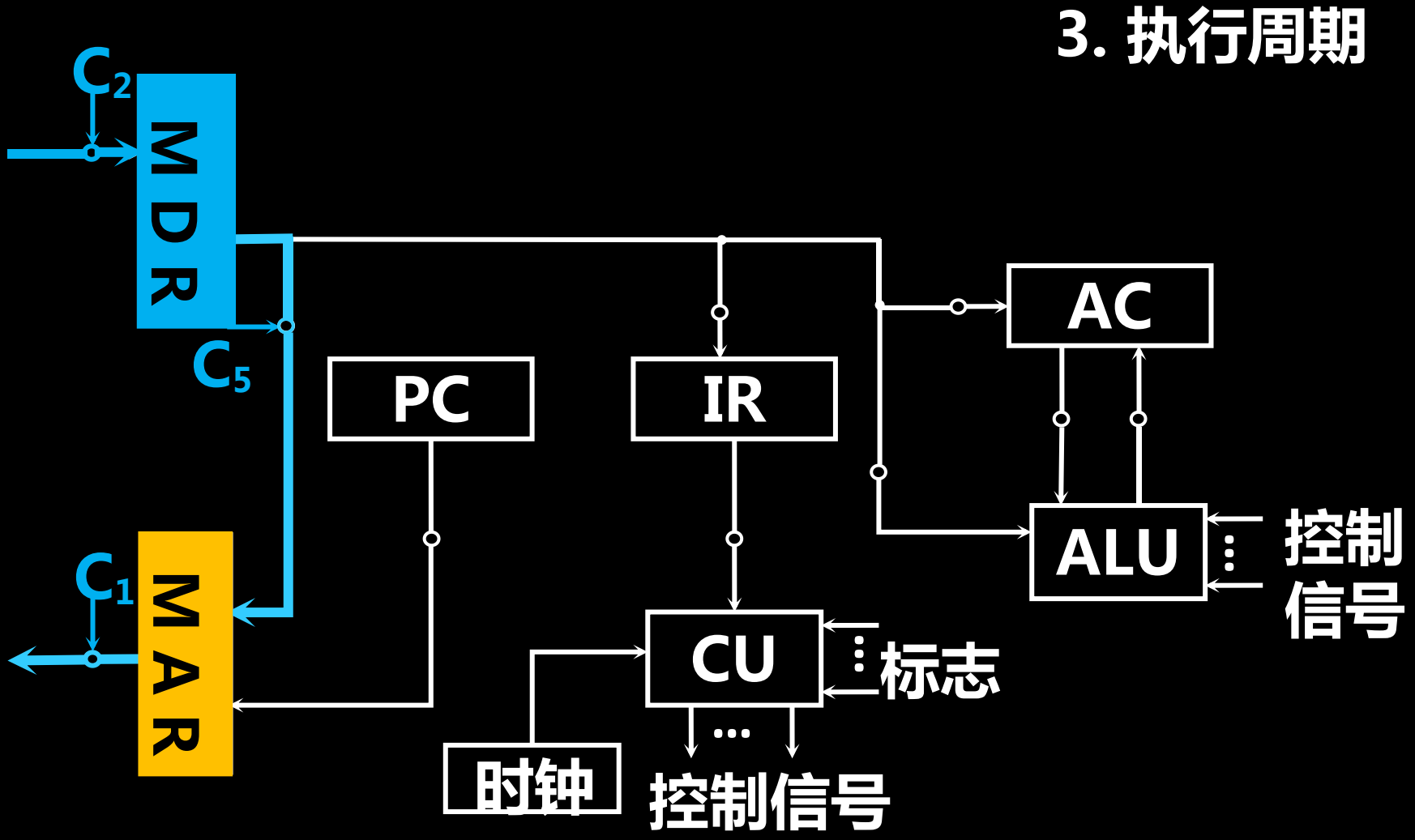
控制信号举例



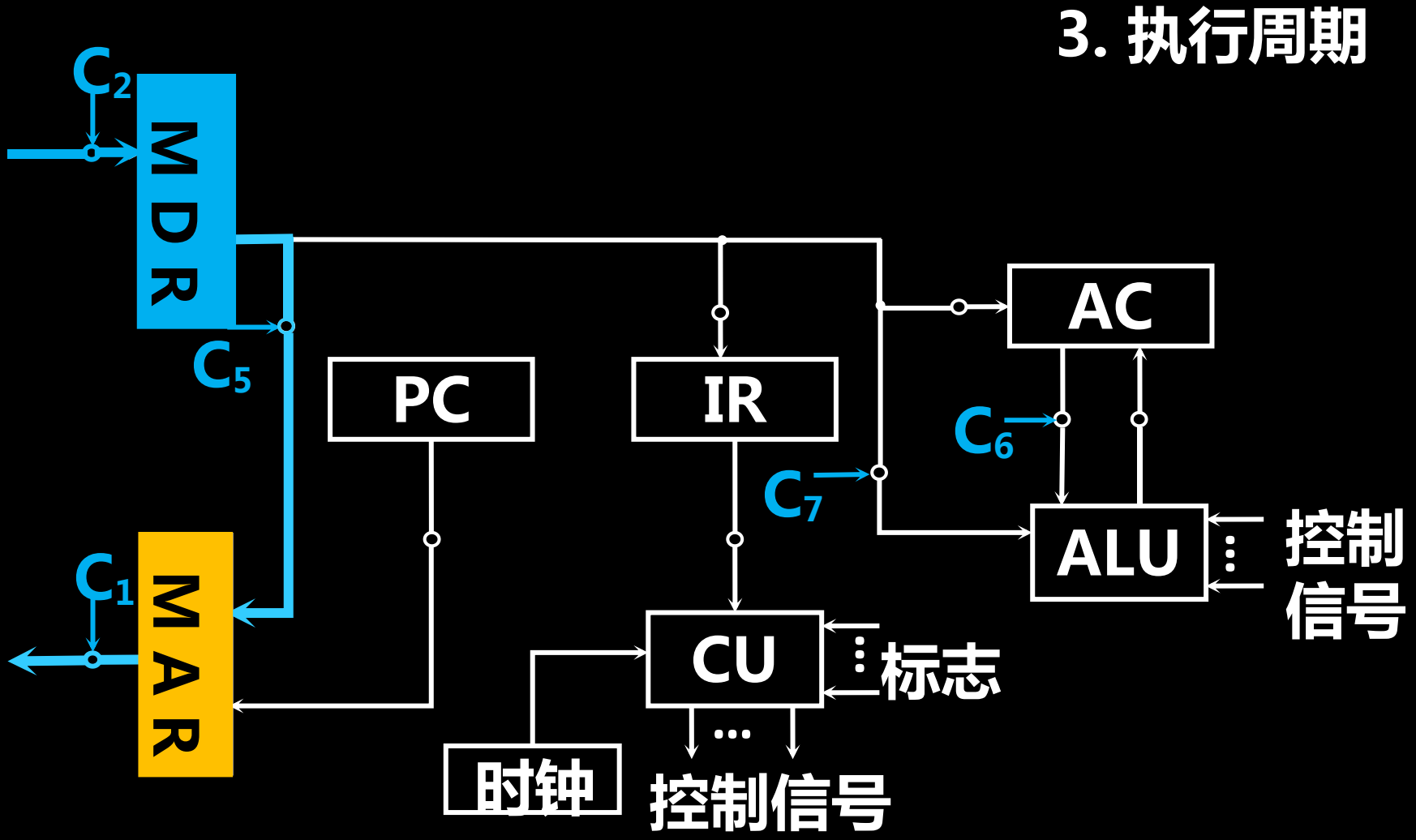
控制信号举例



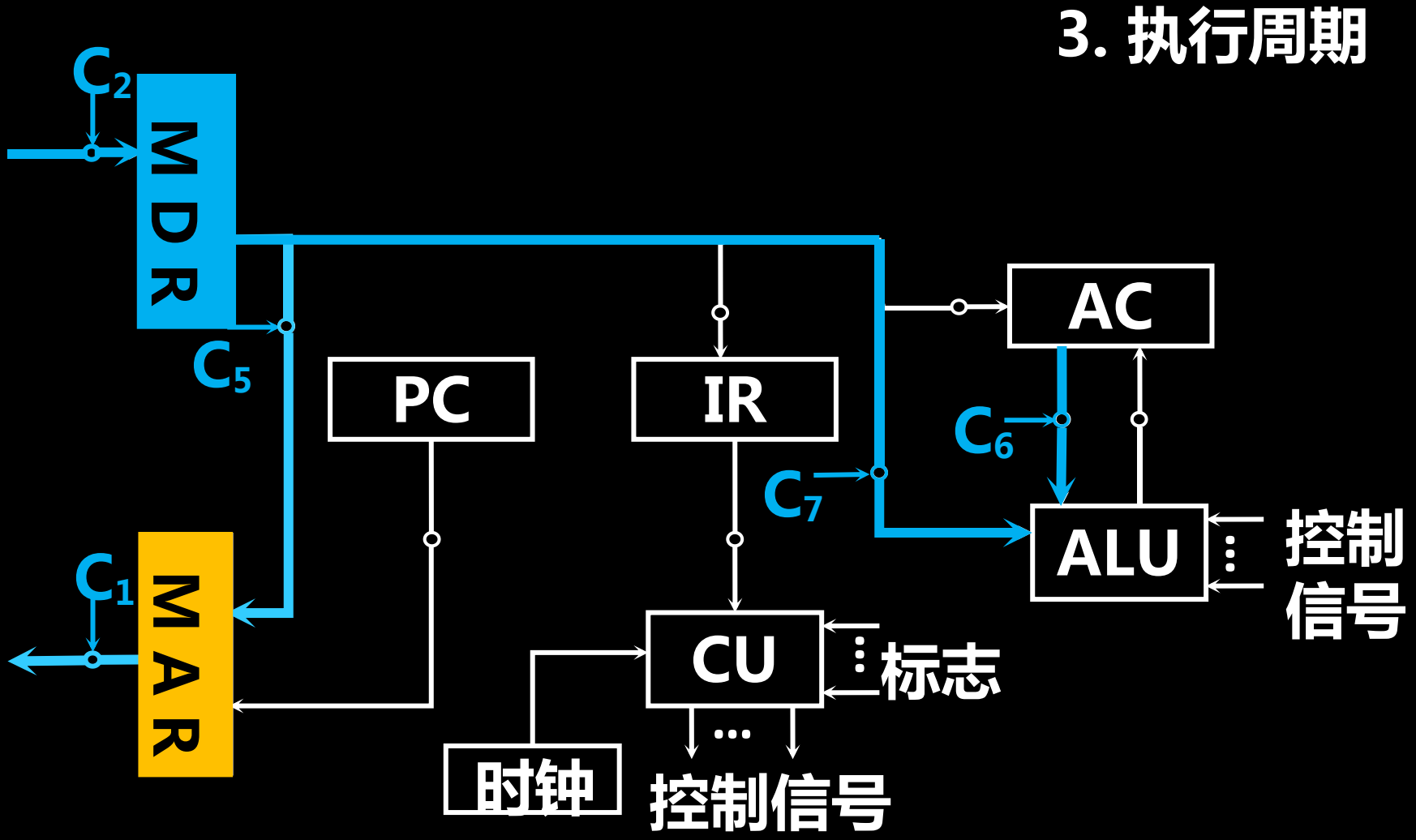
控制信号举例



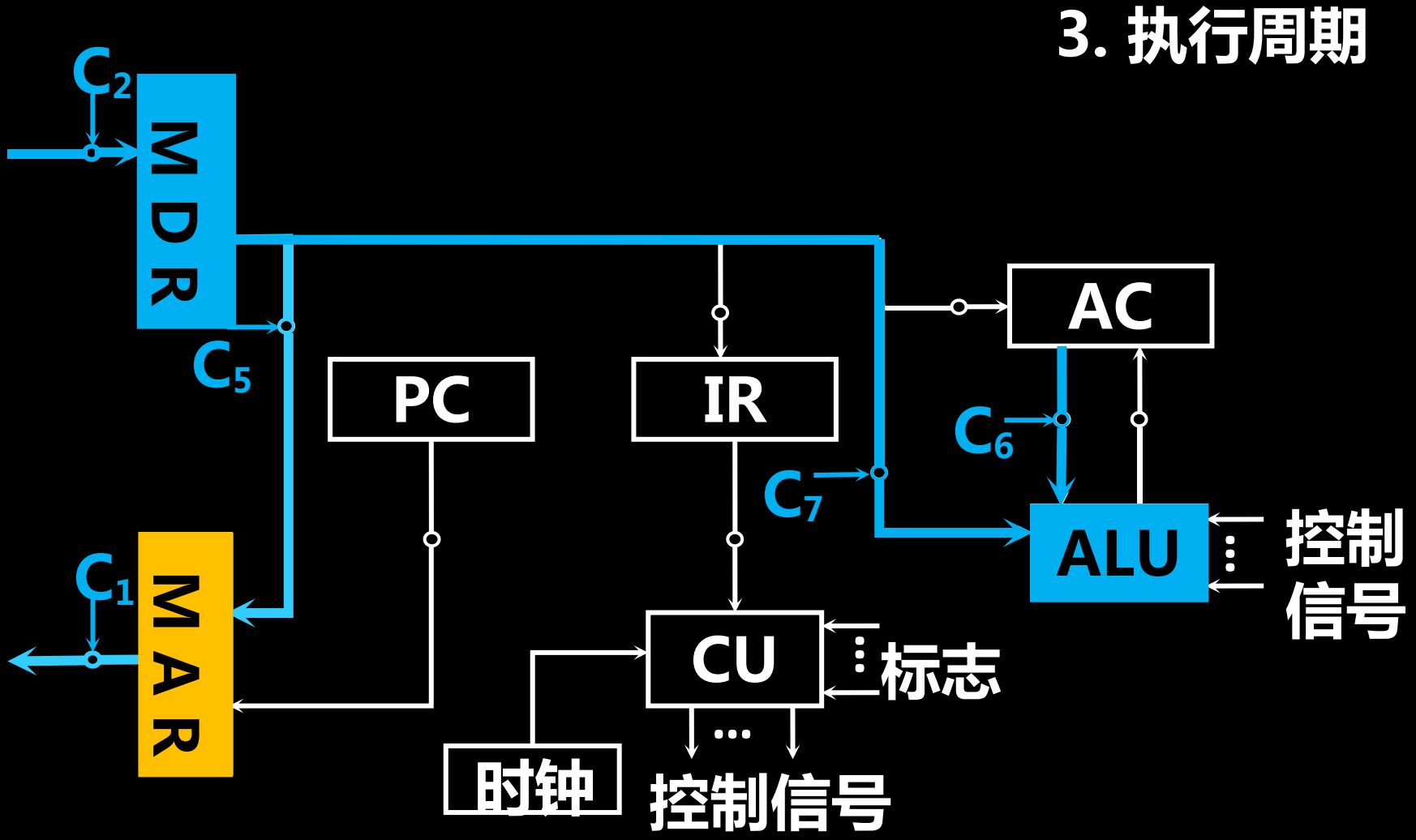
控制信号举例



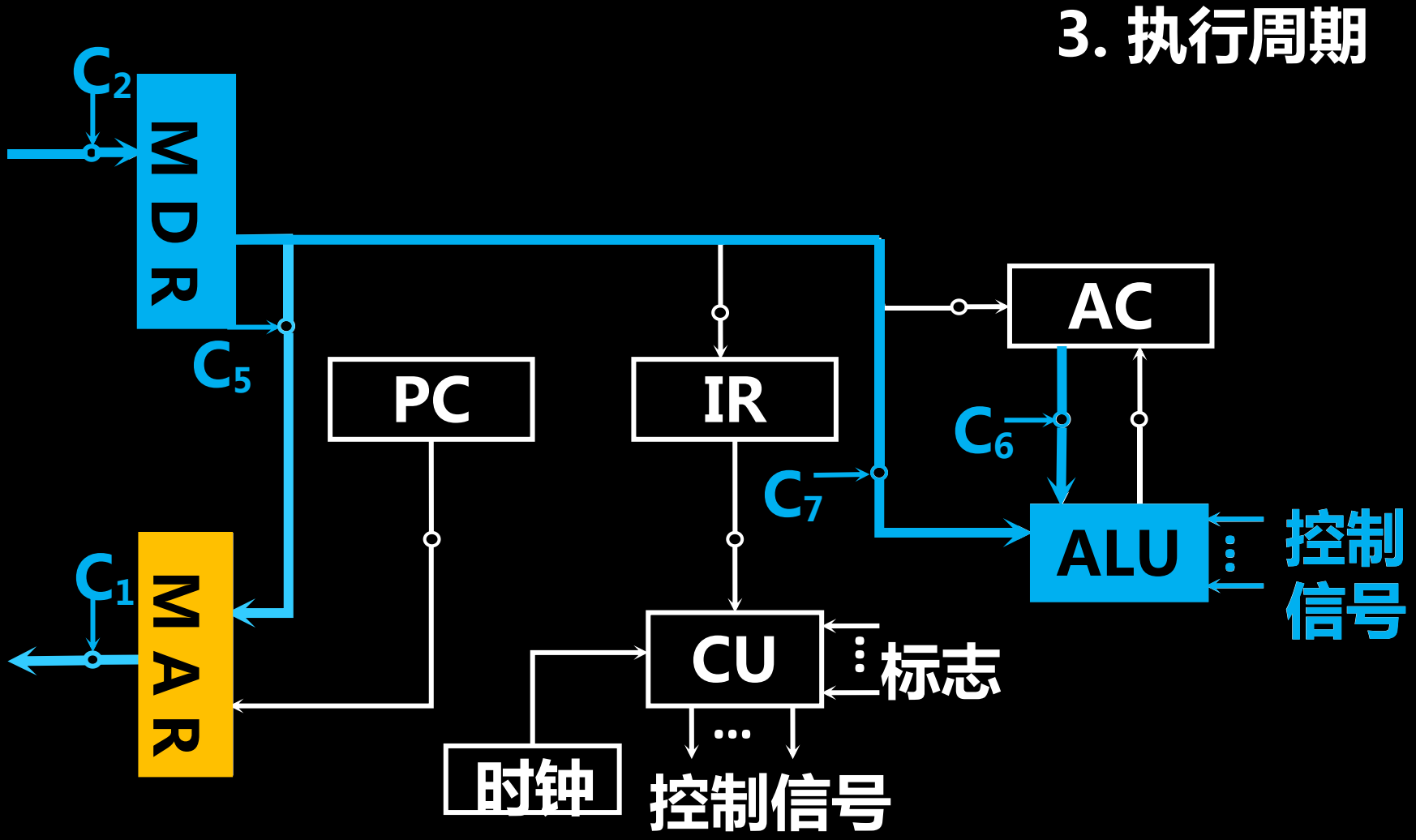
控制信号举例



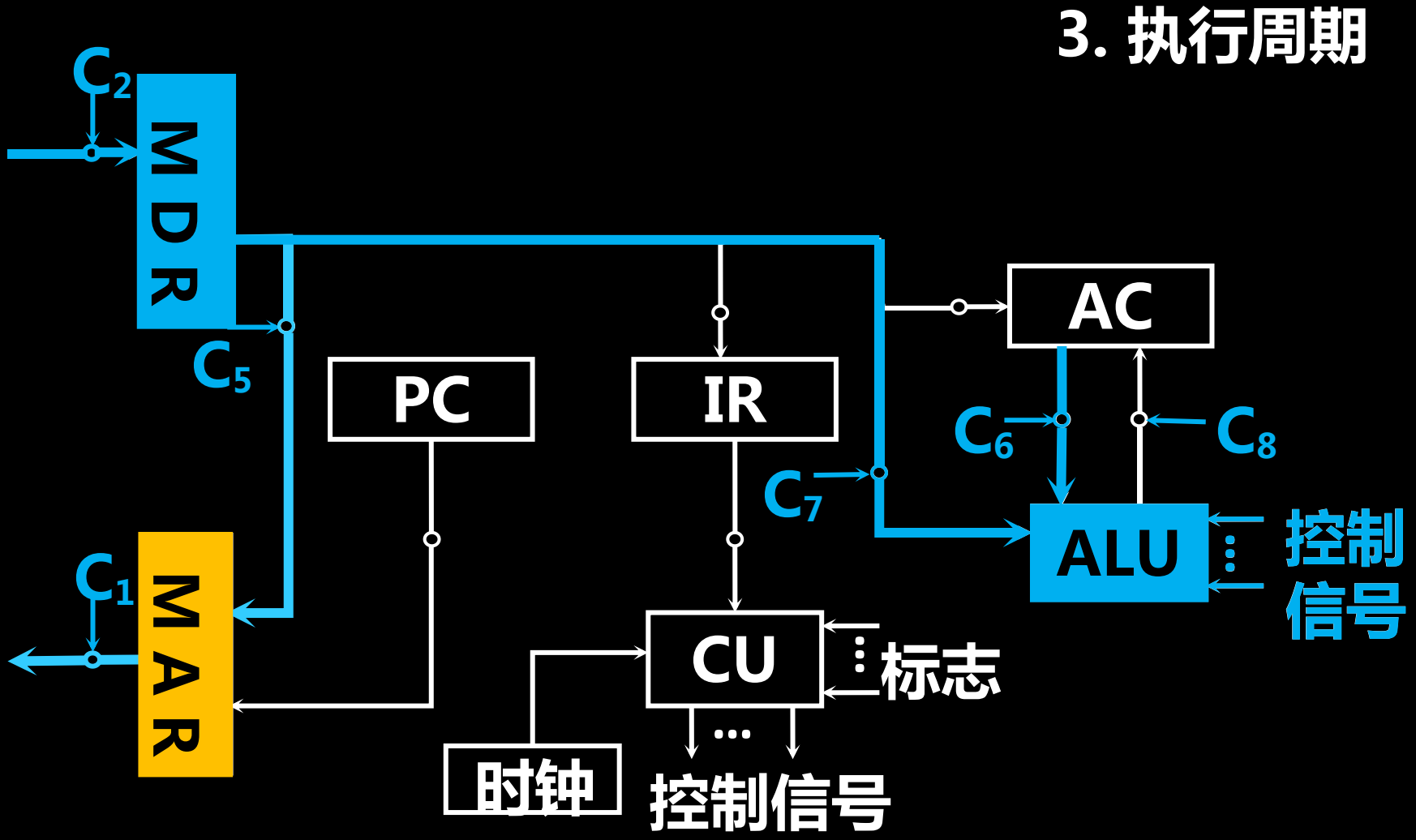
控制信号举例



控制信号举例

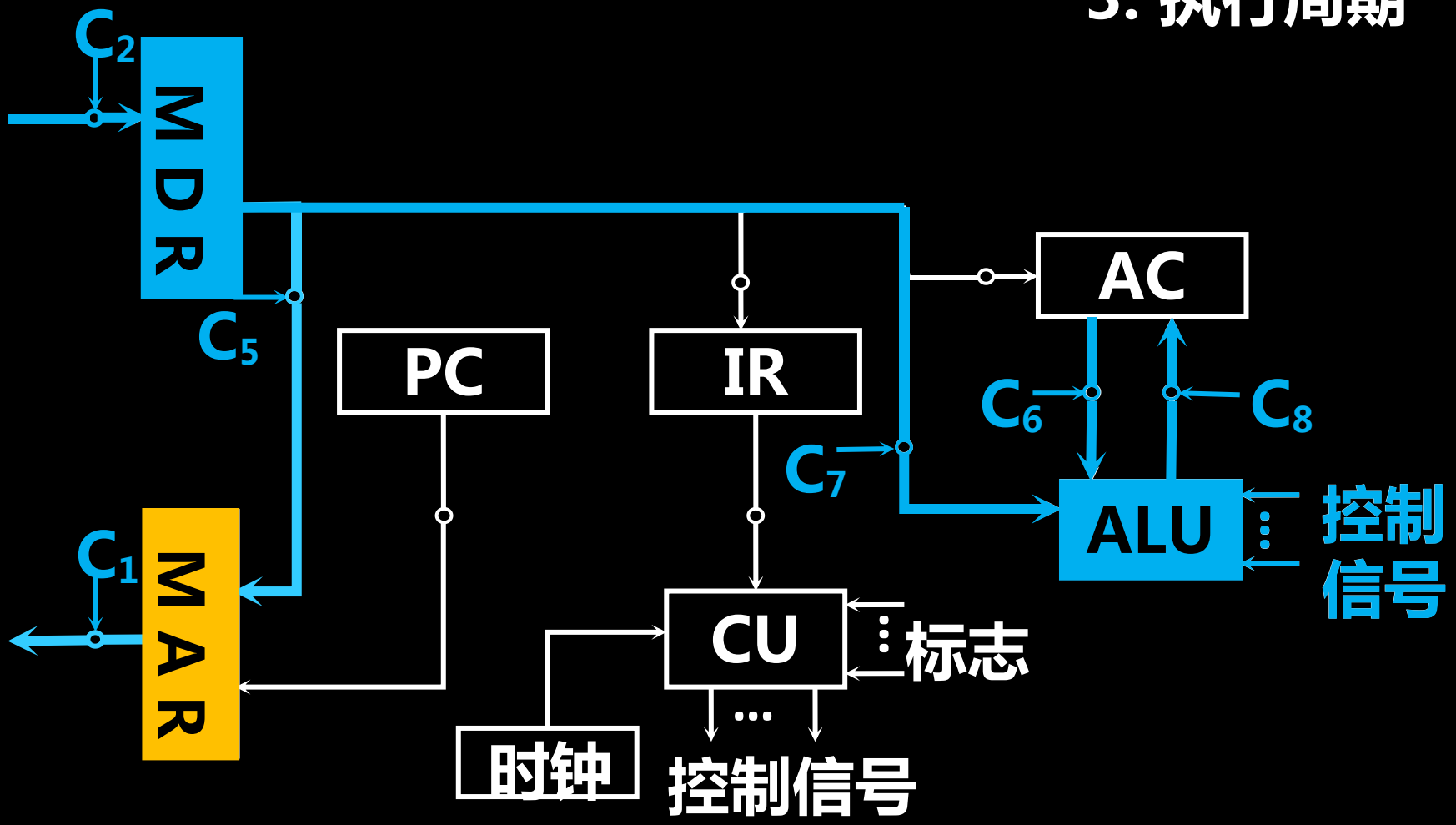


控制信号举例



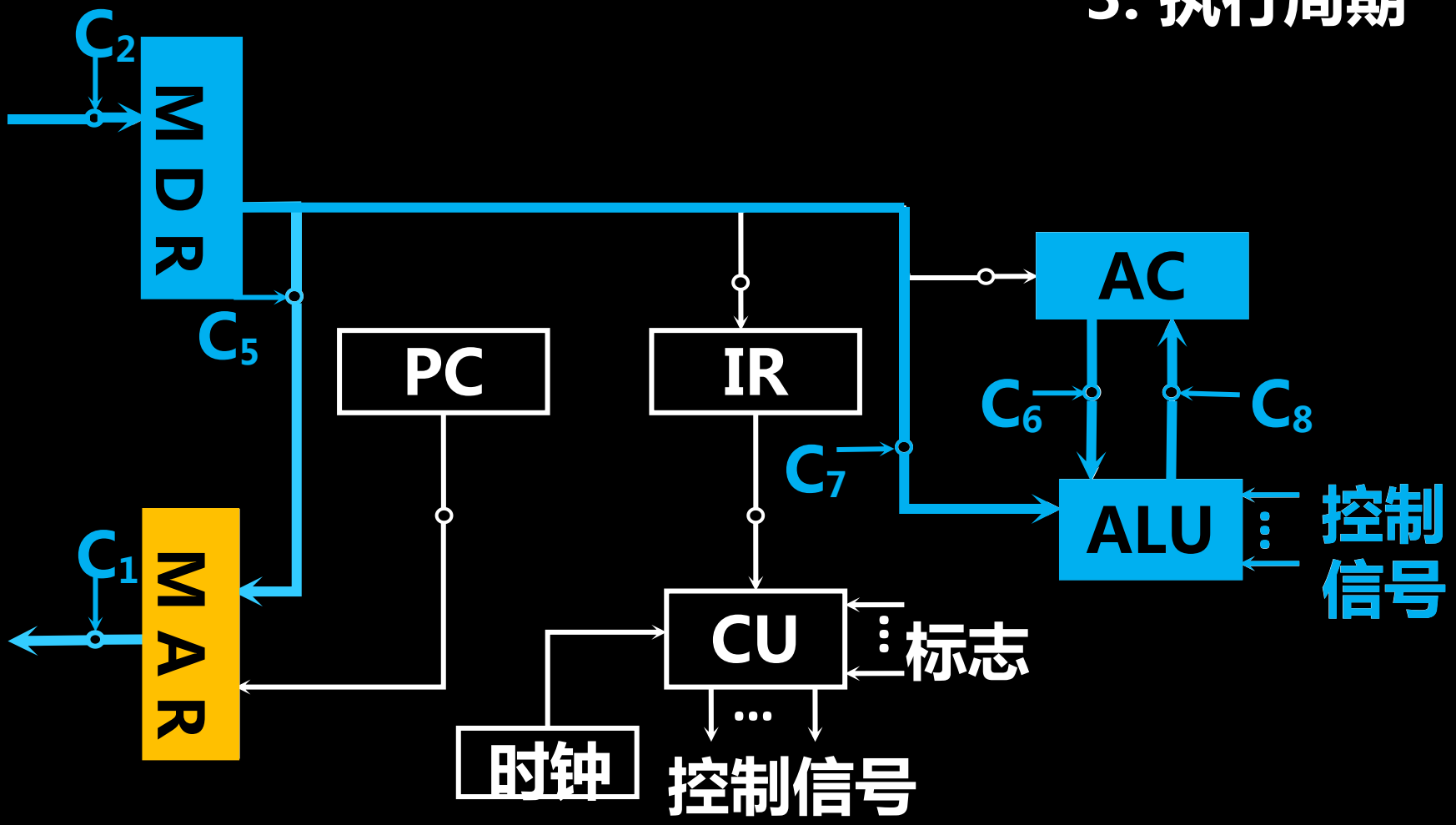
控制信号举例

3. 执行周期

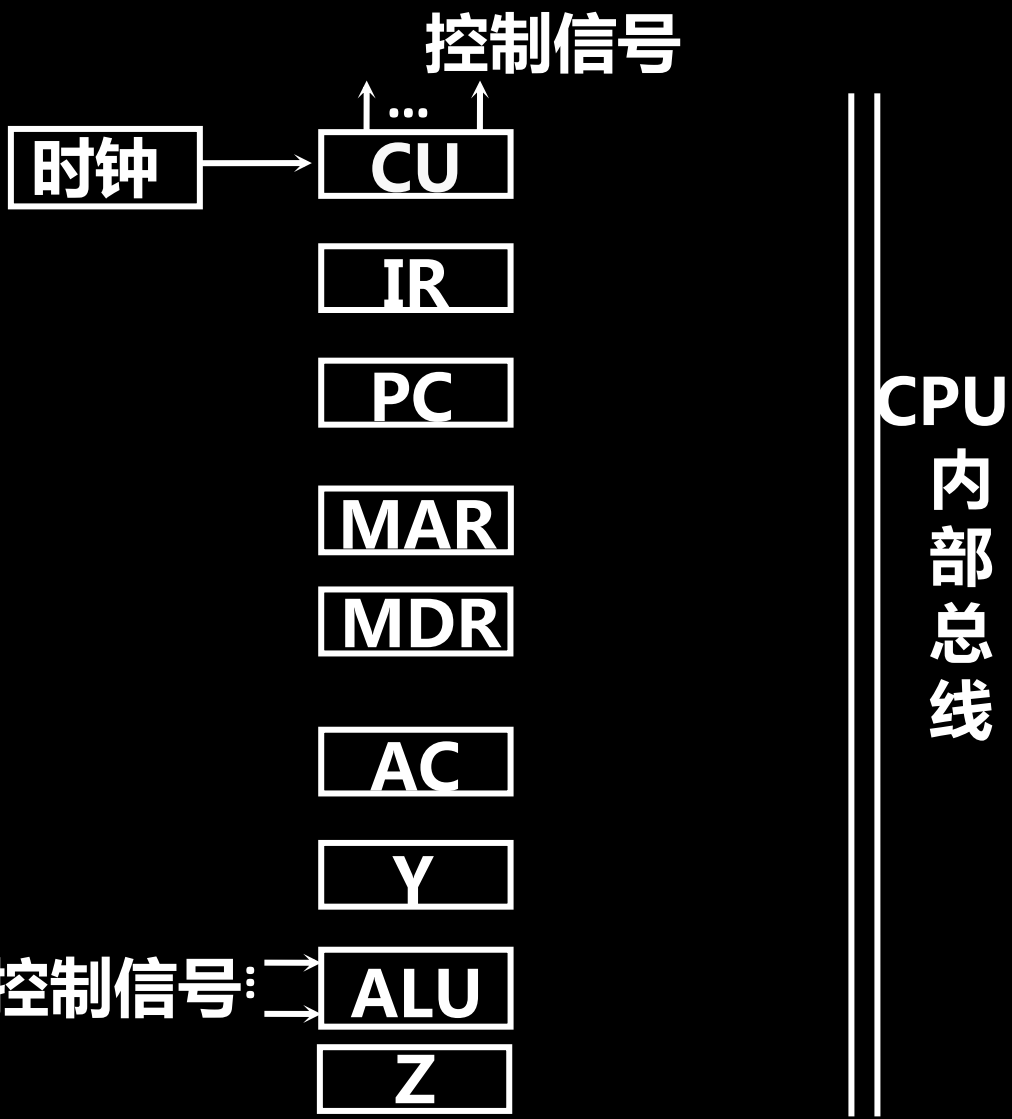


控制信号举例

3. 执行周期

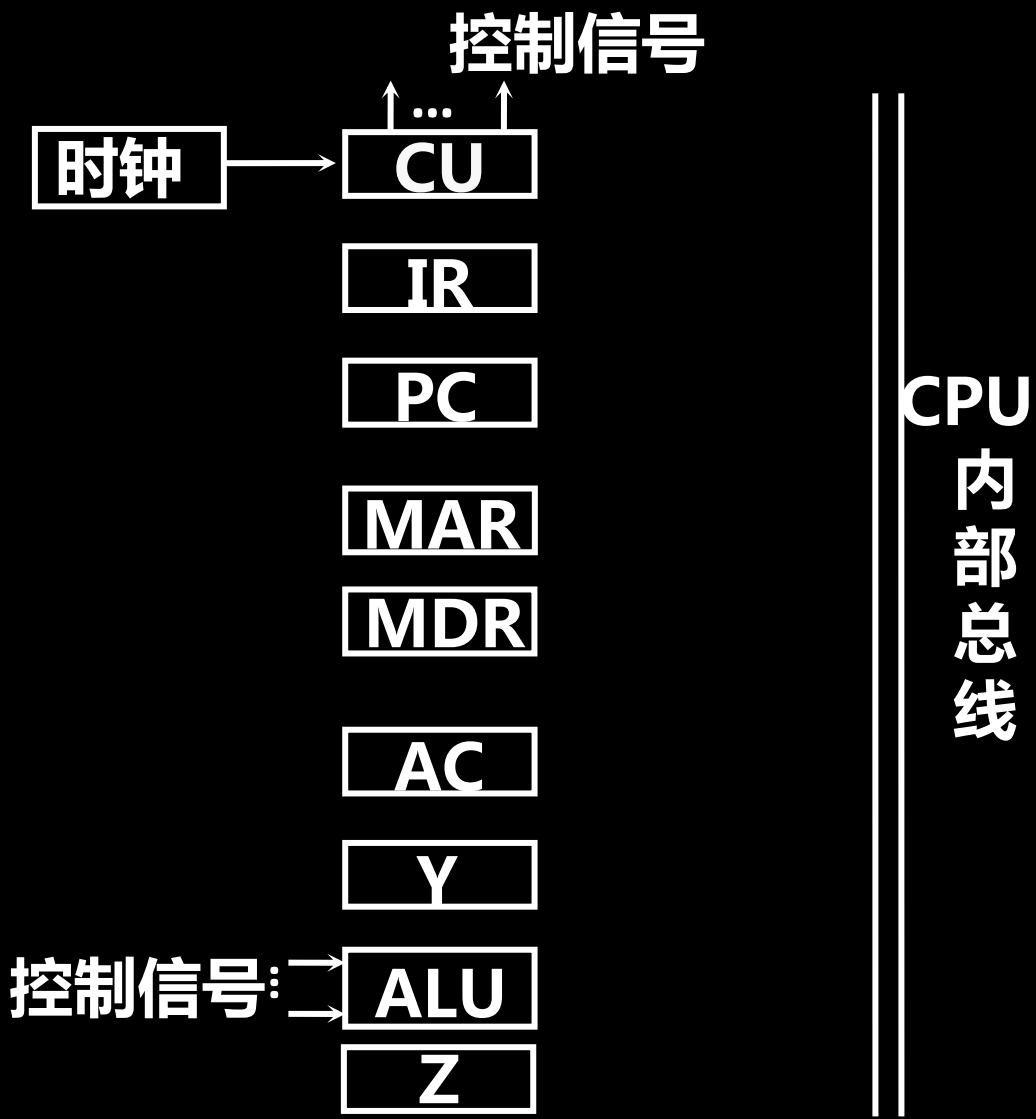


采用CPU内部总线方式



采用CPU内部总线方式

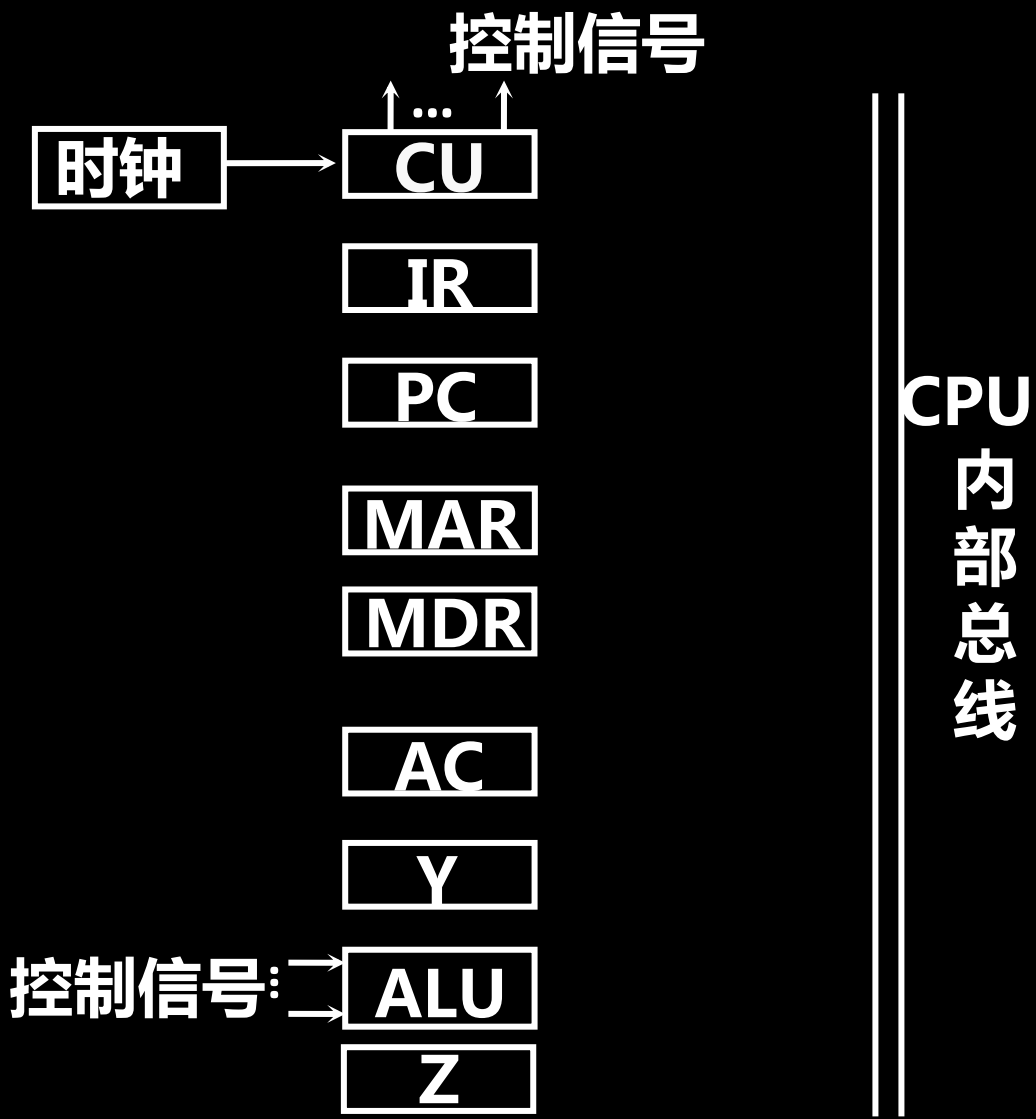
(1) ADD @ X 取指周期



采用CPU内部总线方式

(1) ADD @ X 取指周期

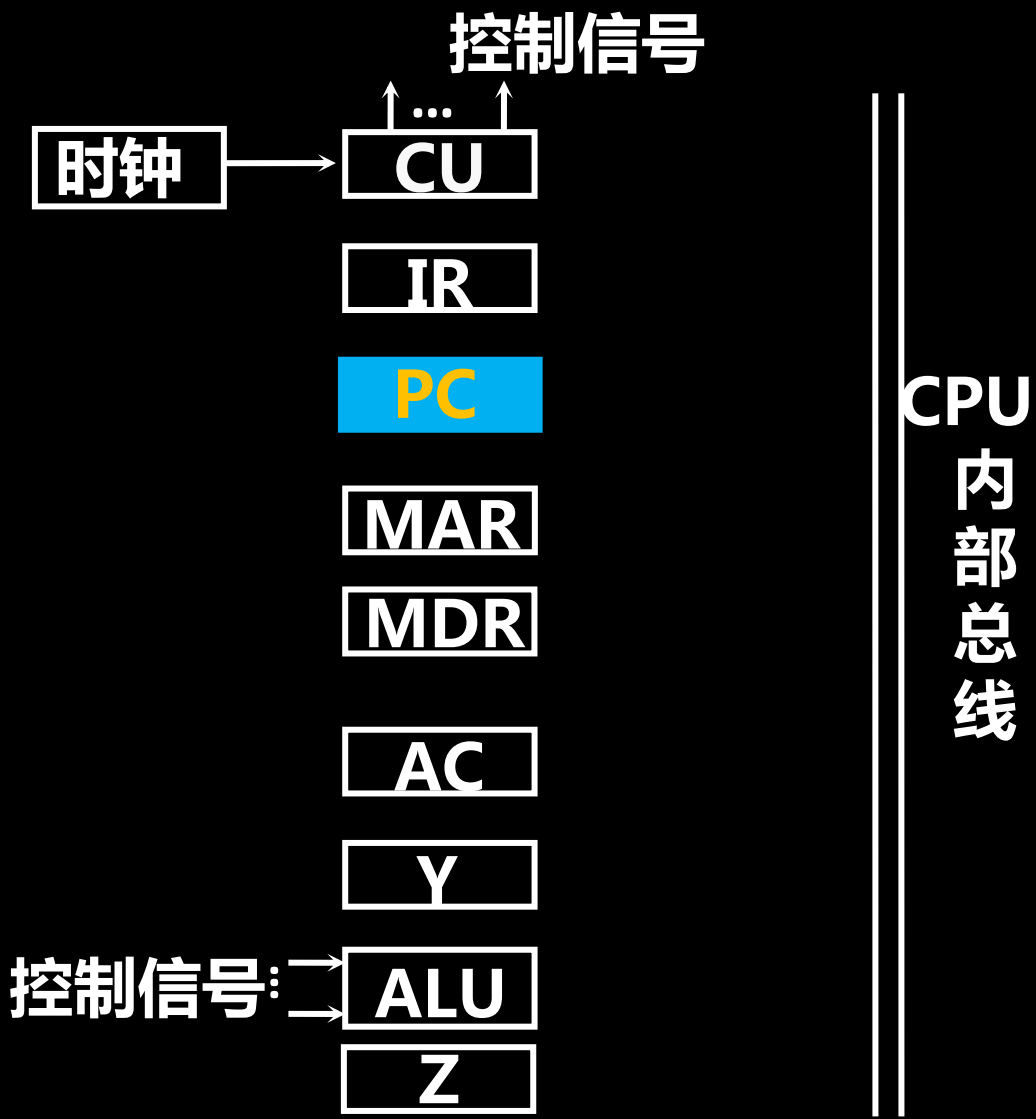
PC



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC

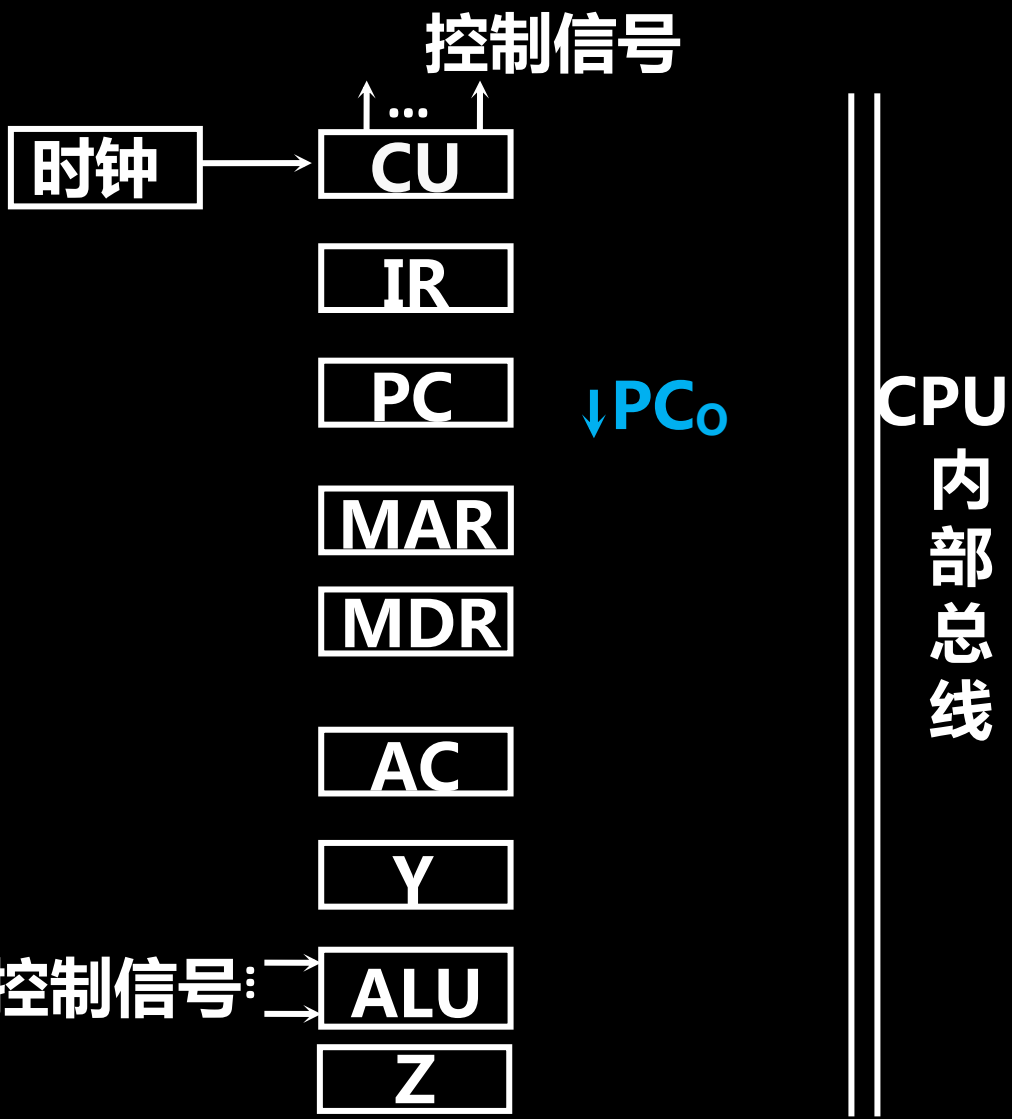


采用CPU内部总线方式

(1) ADD @ X 取指周期

PC

PC_O



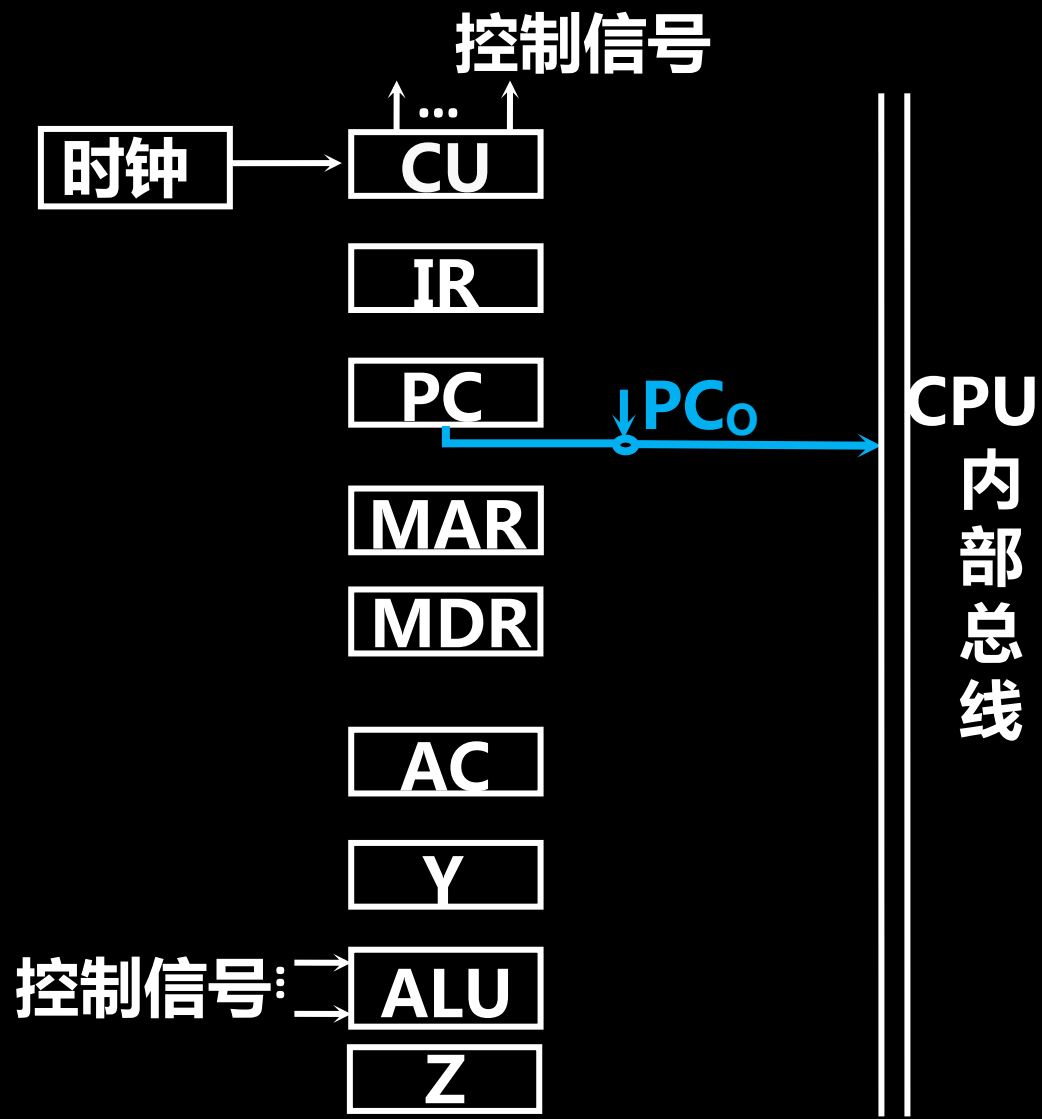
采用CPU内部总线方式

采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶

PC_O

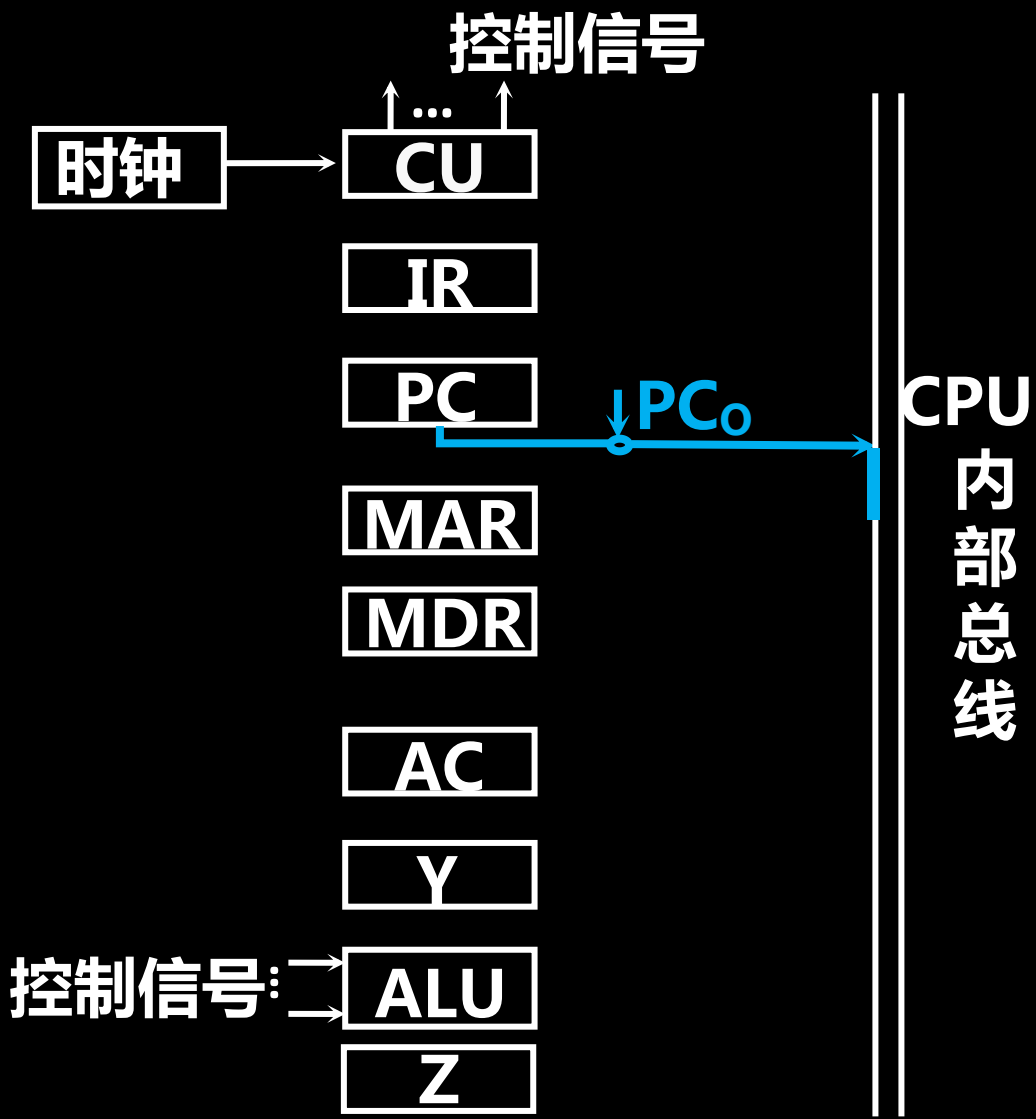


采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶

PC_O

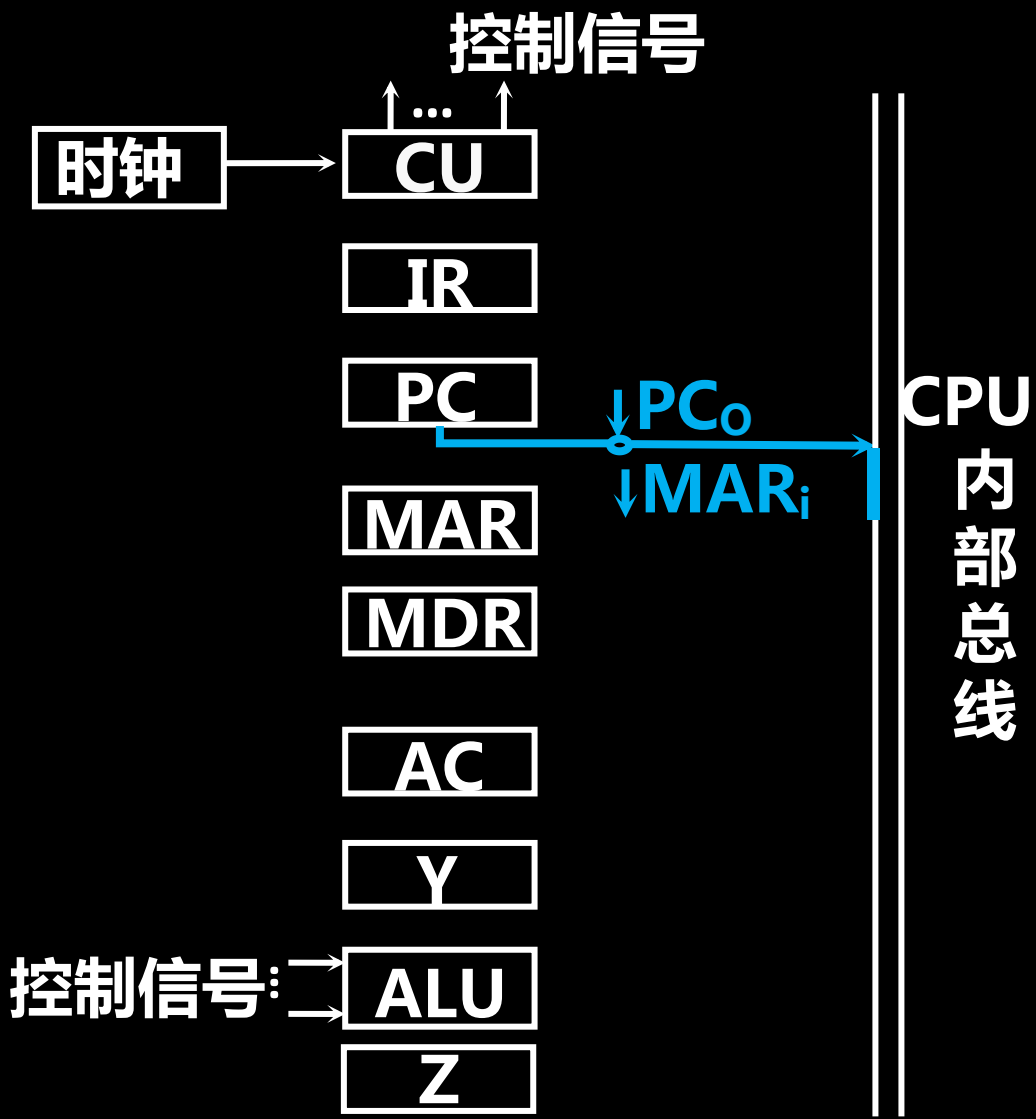


采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶

PC_O MAR_i

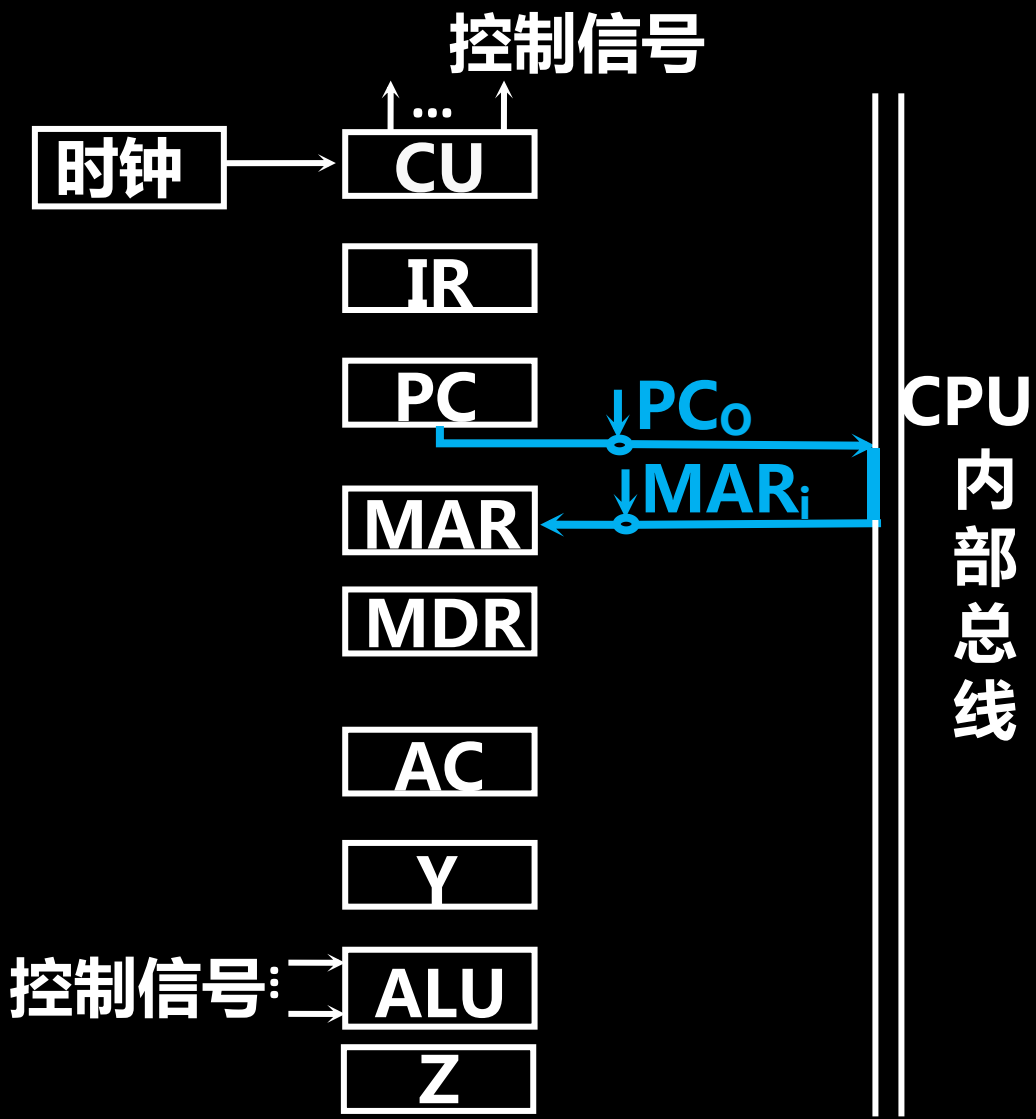


采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶

PC_O MAR_i

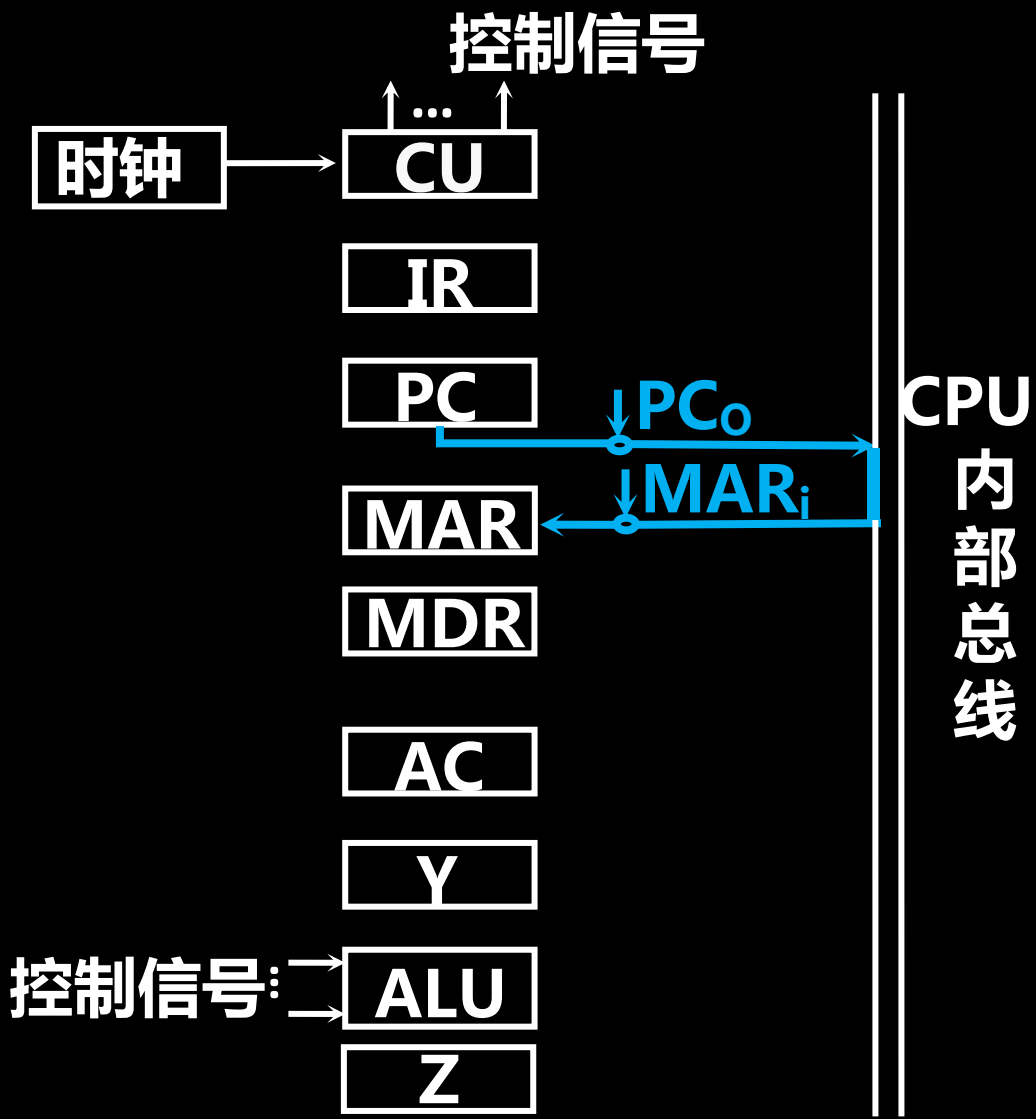


采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR

PC_O MAR_i

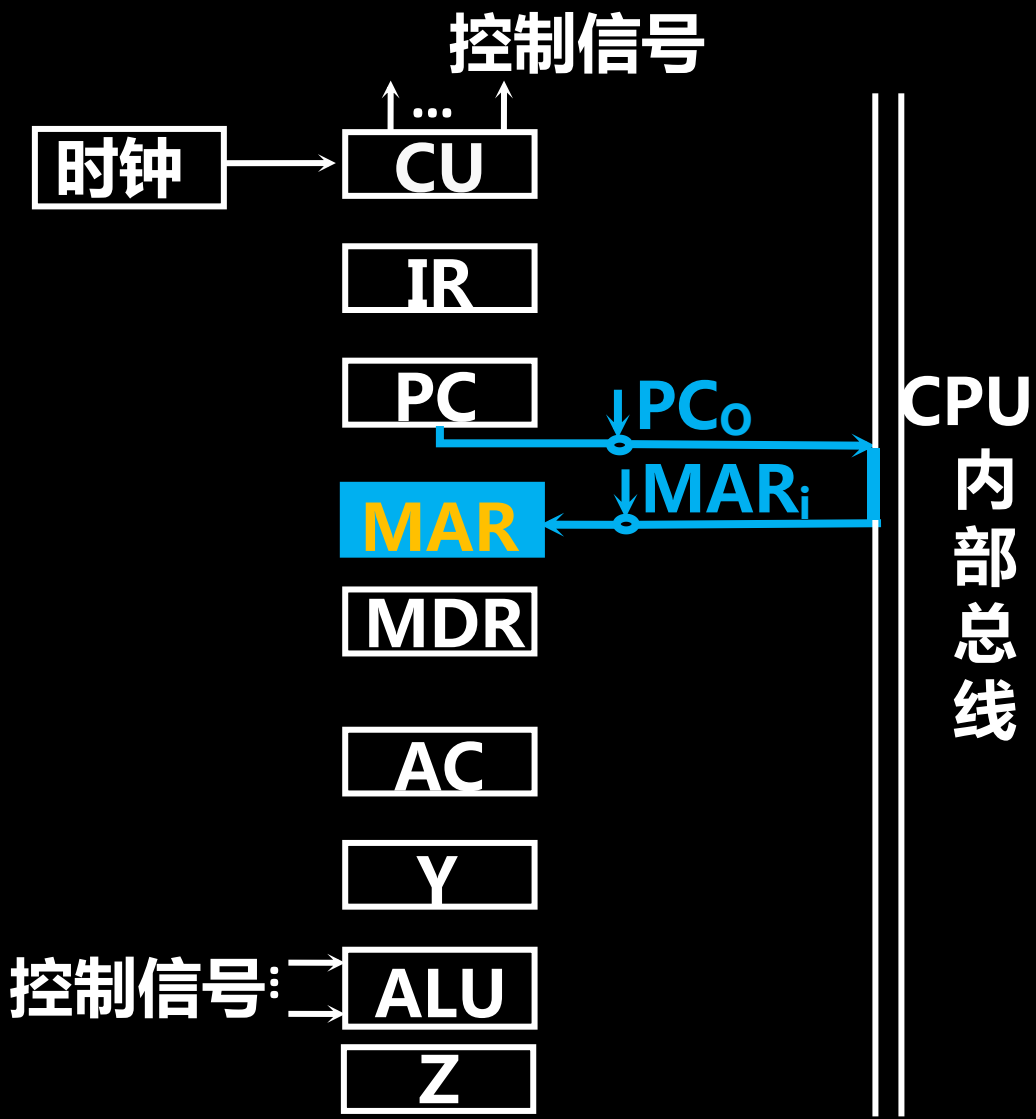


采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR

PC_O MAR_i

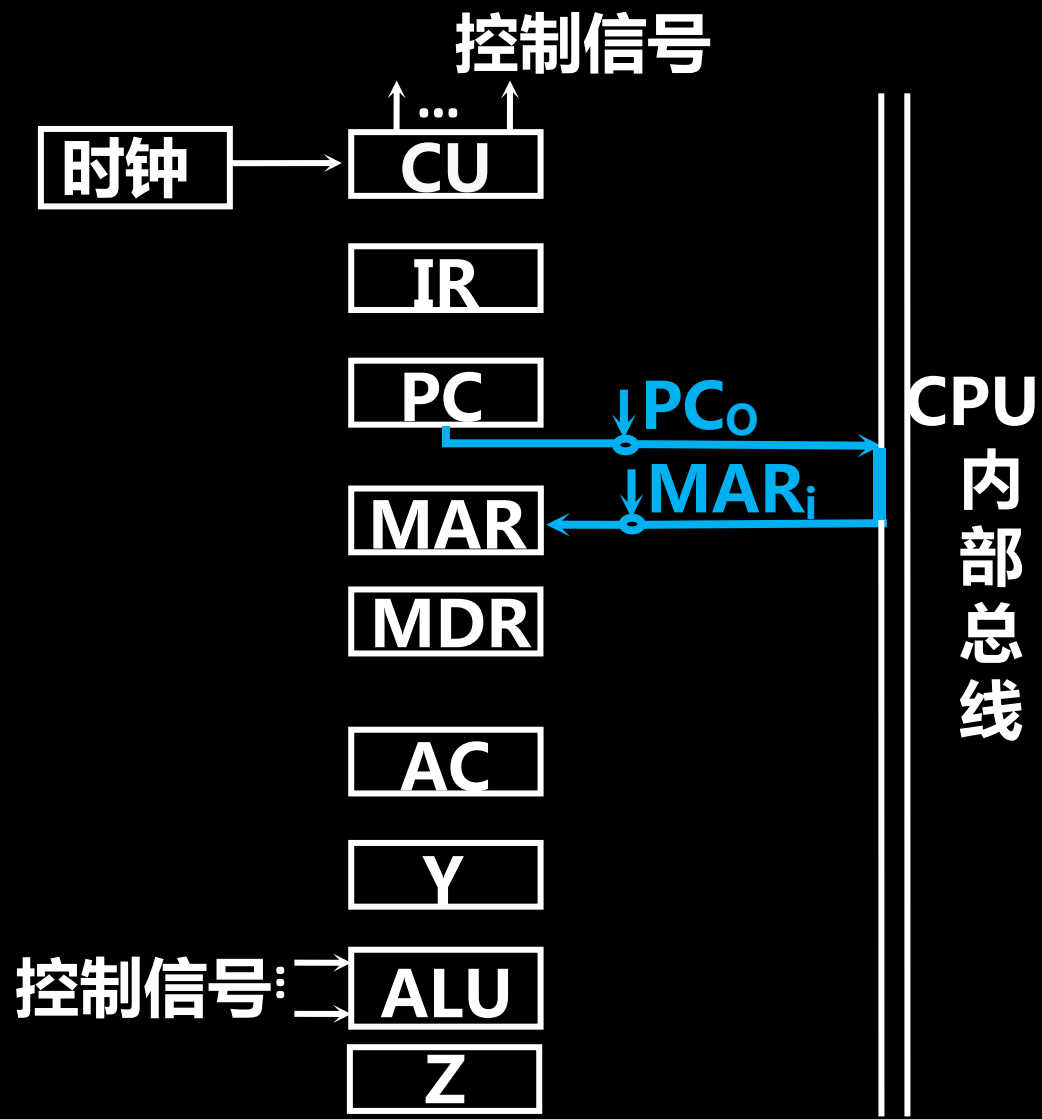


采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶

PC_O MAR_i

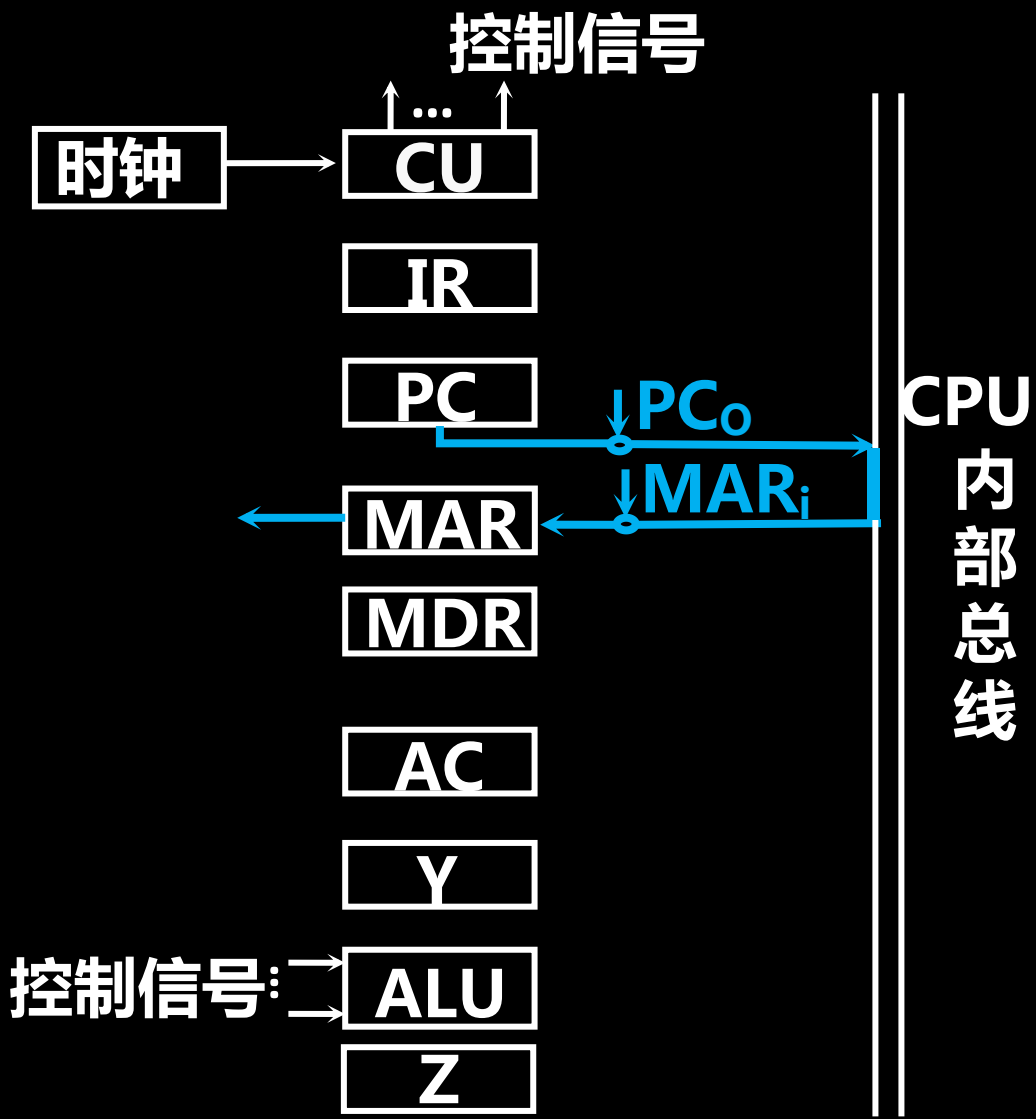


采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶

PC_O MAR_i

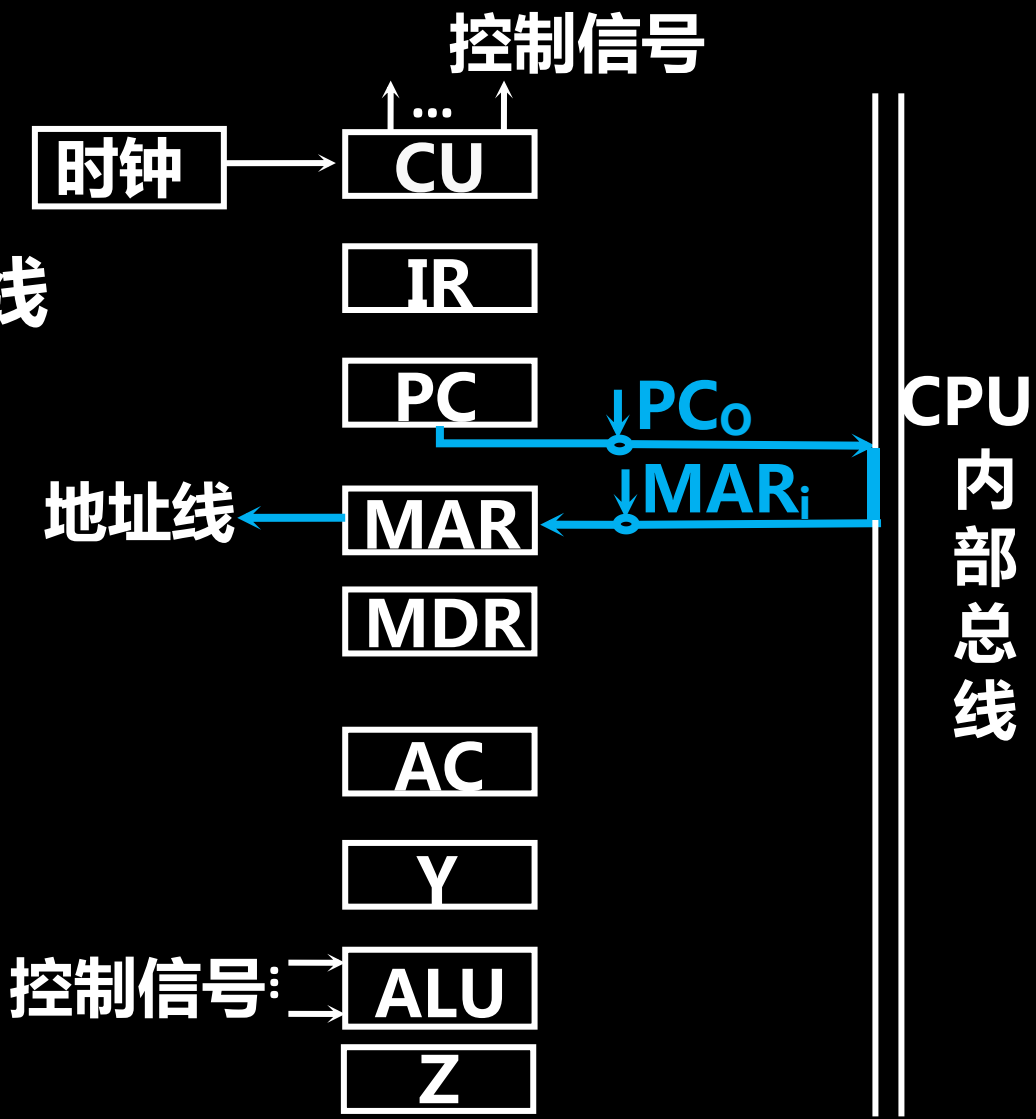


采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i



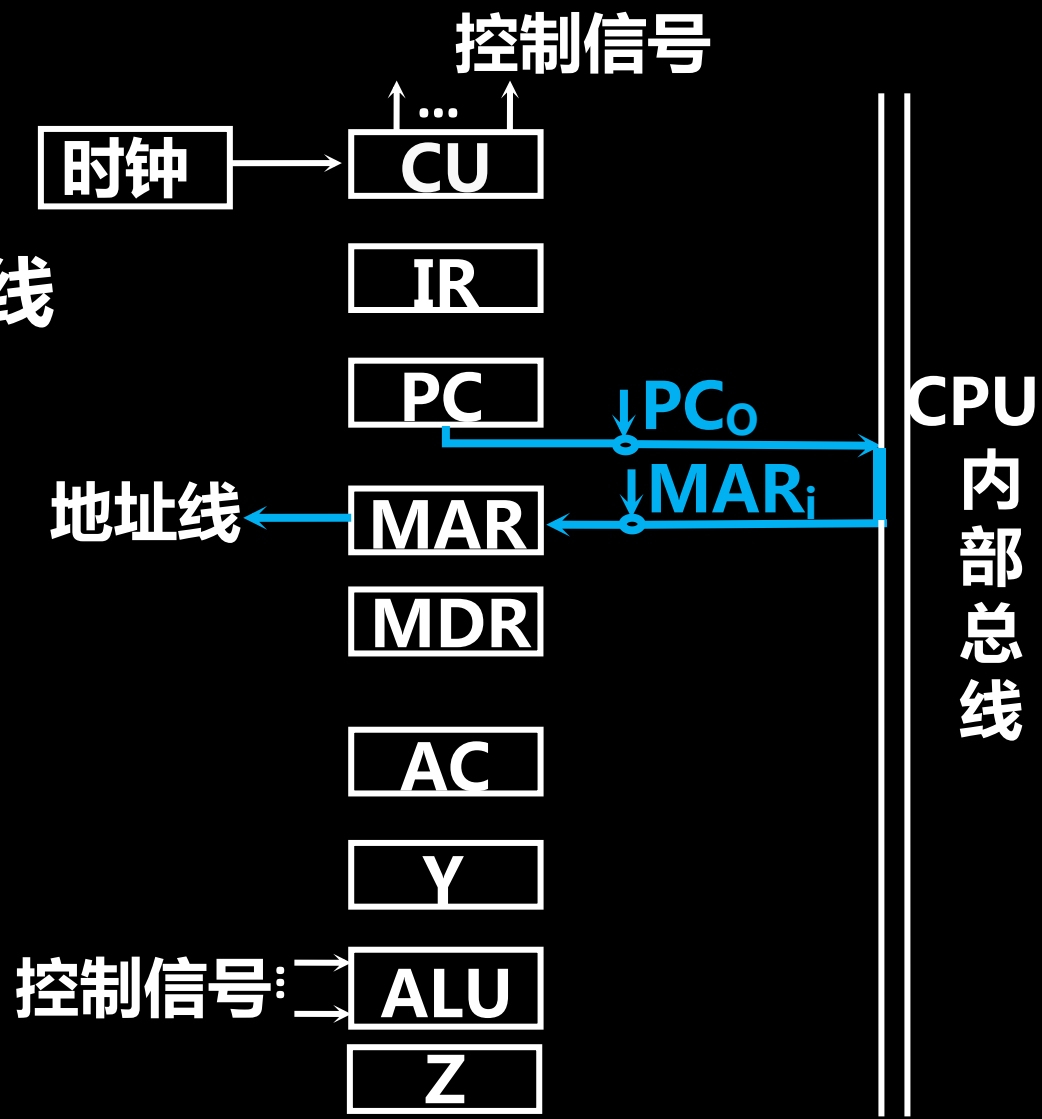
采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

CU 发读命令 1 ⟶ R



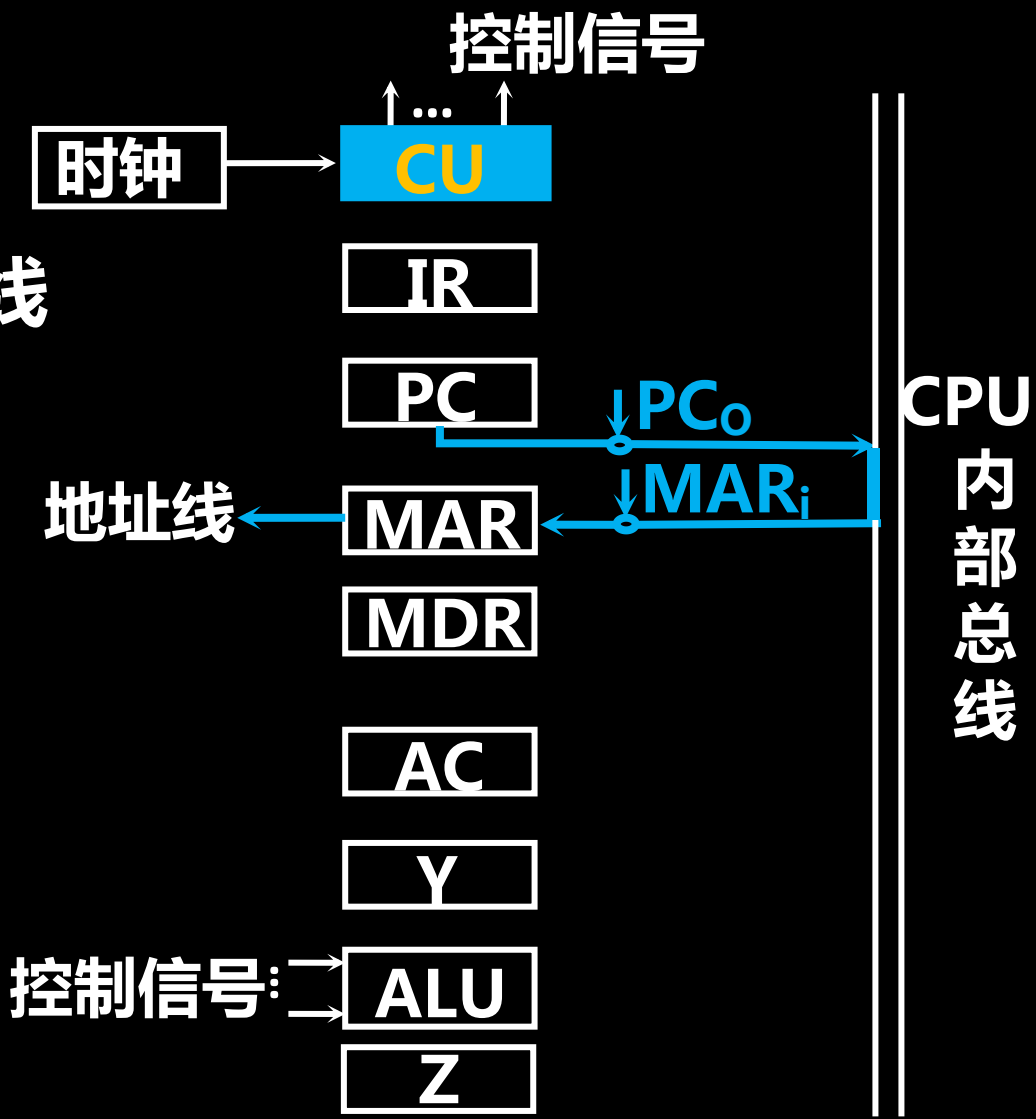
采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

CU 发读命令 1 ⟶ R



采用CPU内部总线方式

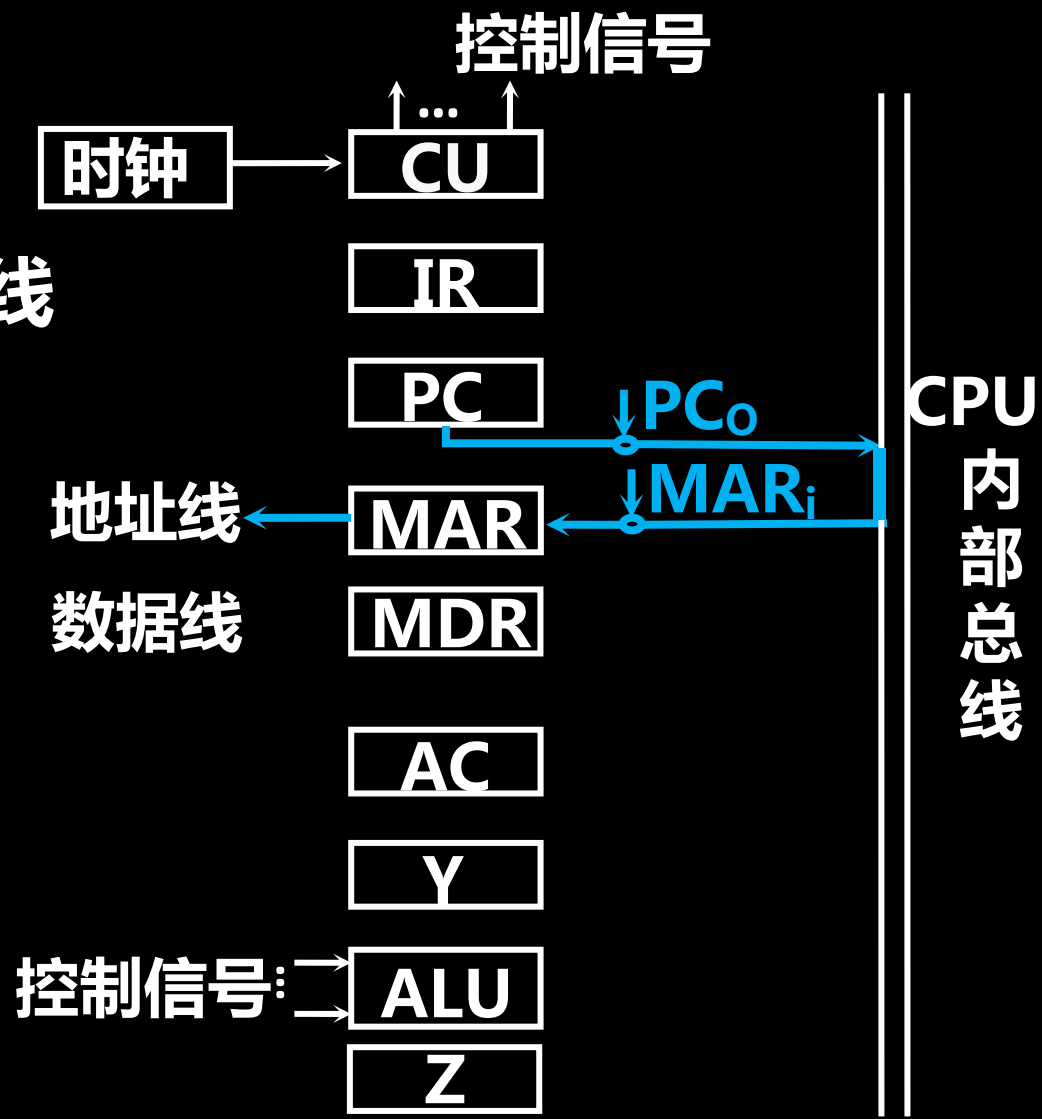
(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线



采用CPU内部总线方式

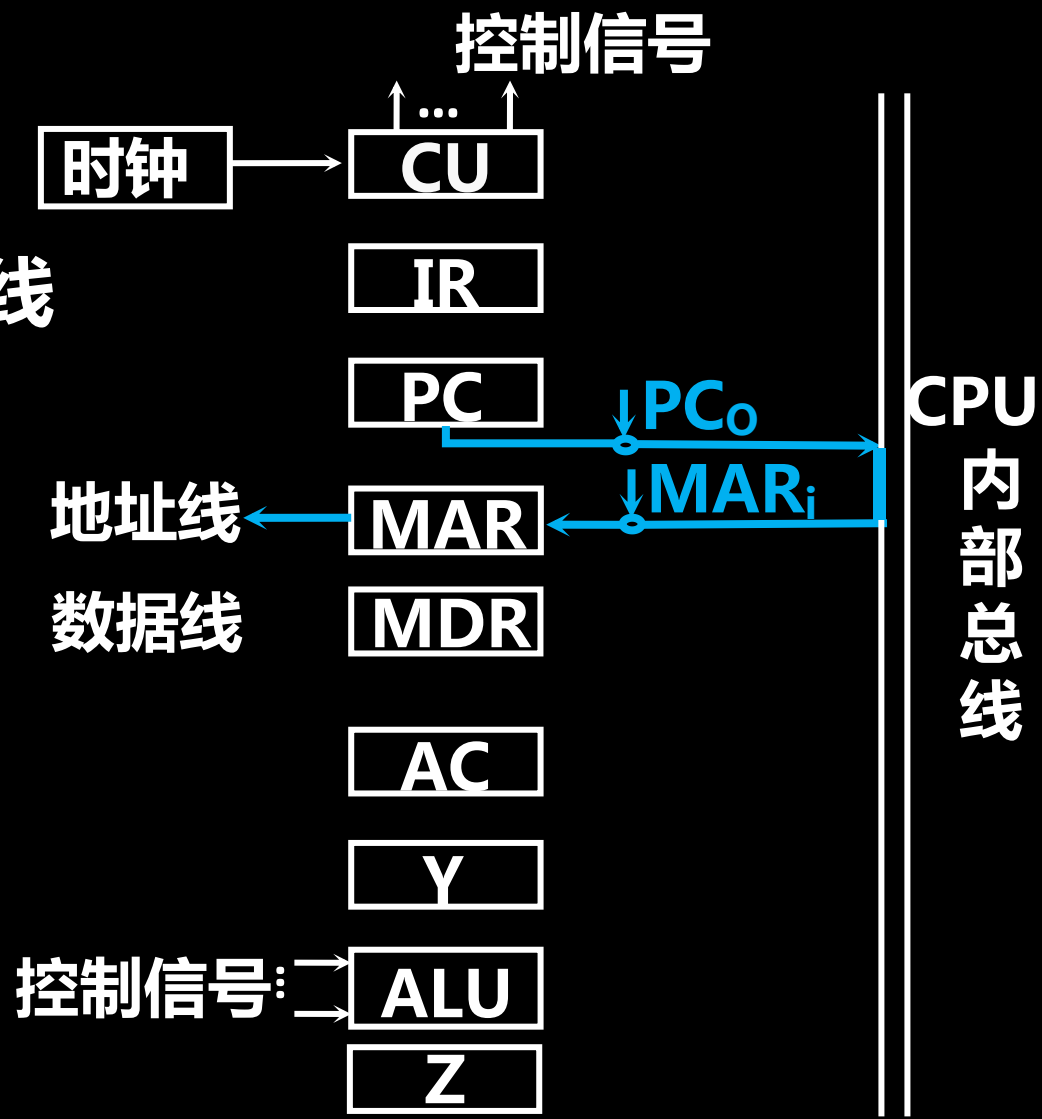
(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶



采用CPU内部总线方式

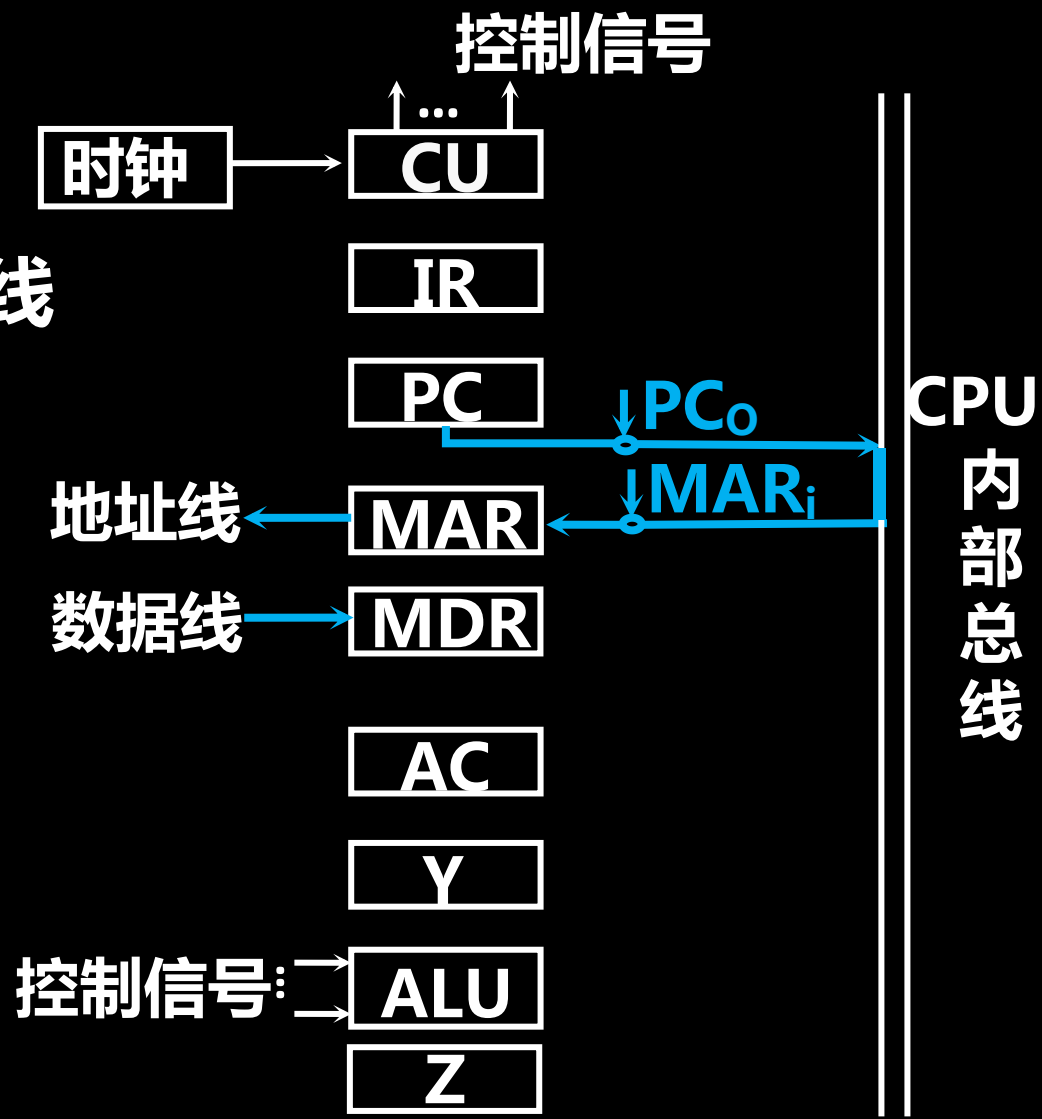
(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶



采用CPU内部总线方式

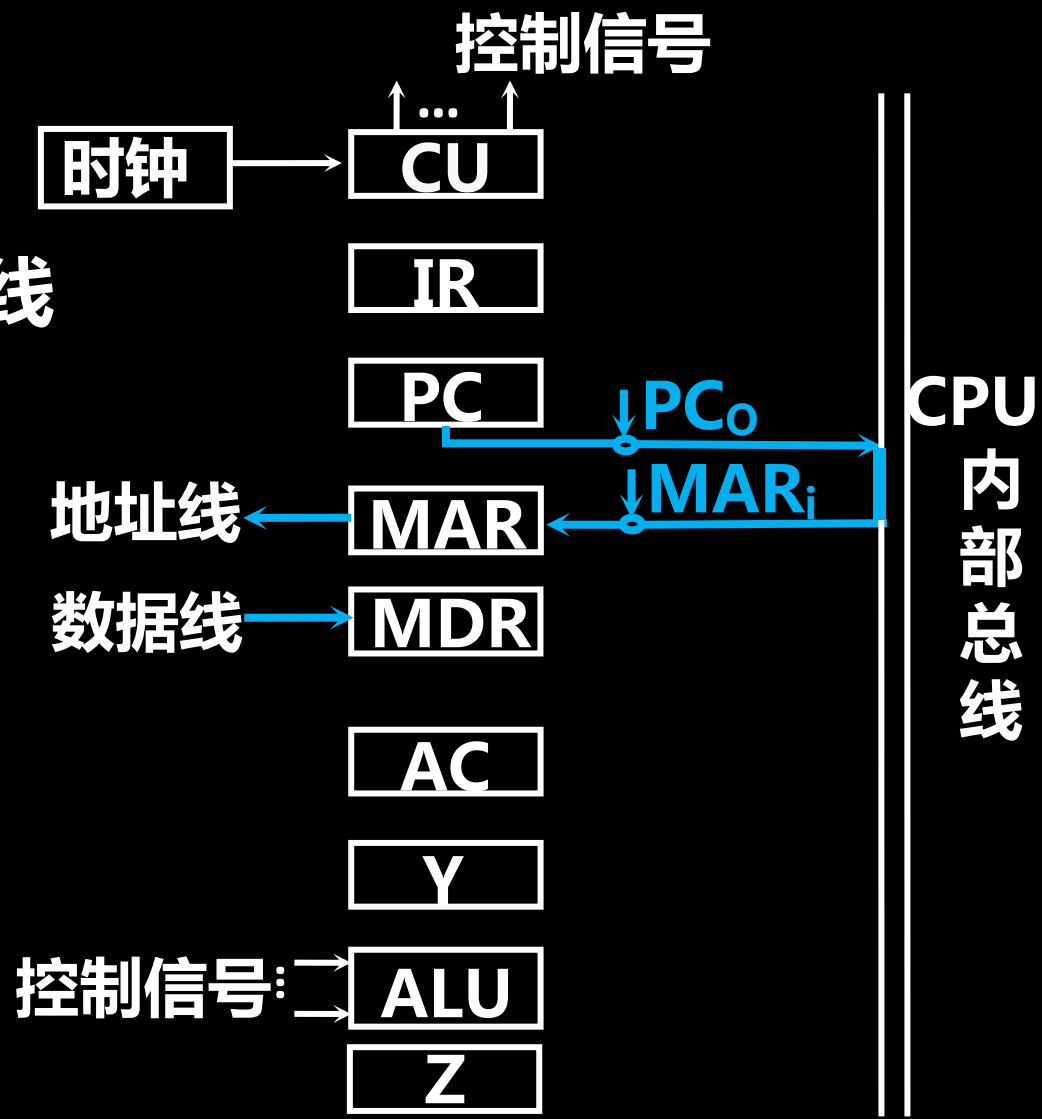
(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶ MDR



采用CPU内部总线方式

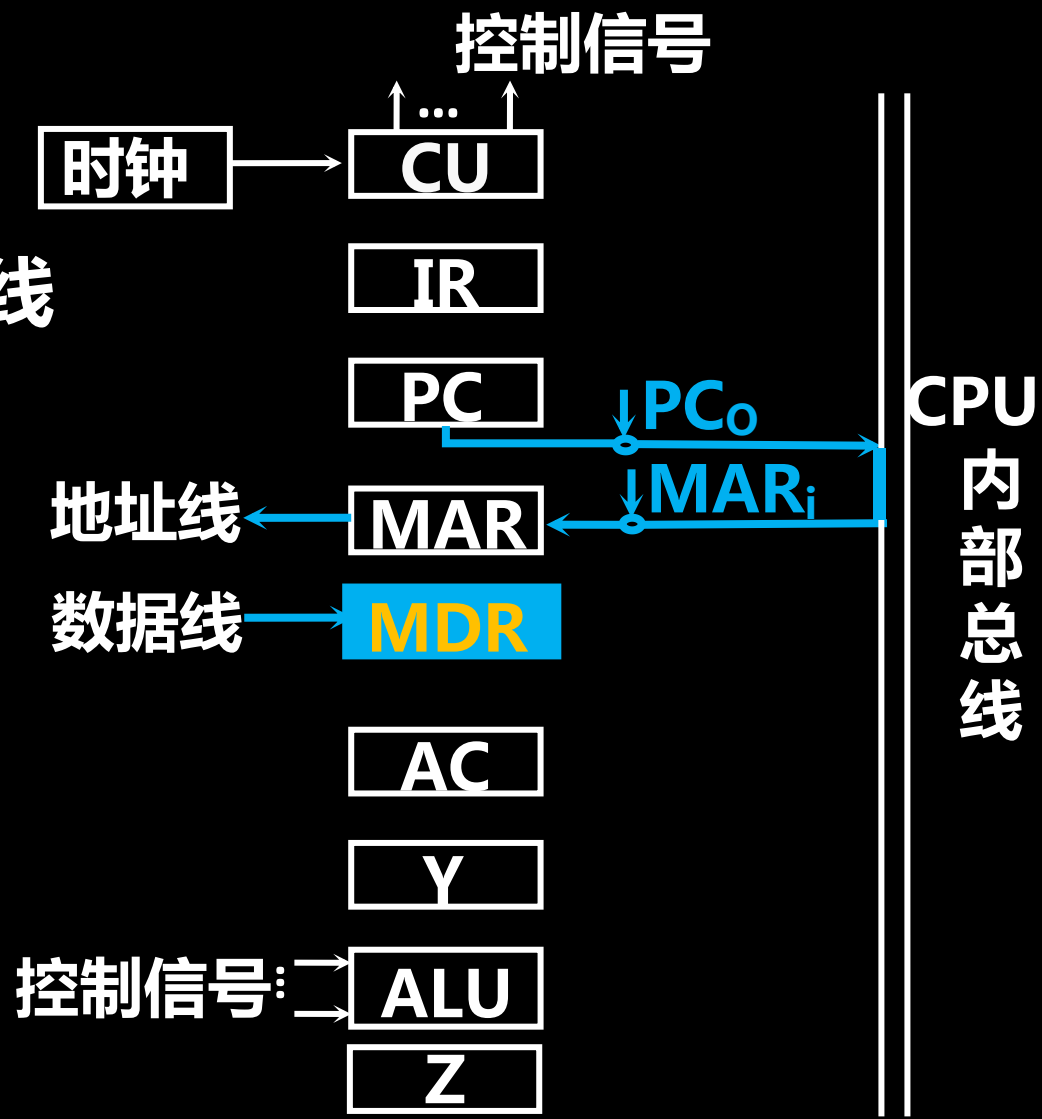
(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶ MDR



采用CPU内部总线方式

(1) ADD @ X 取指周期

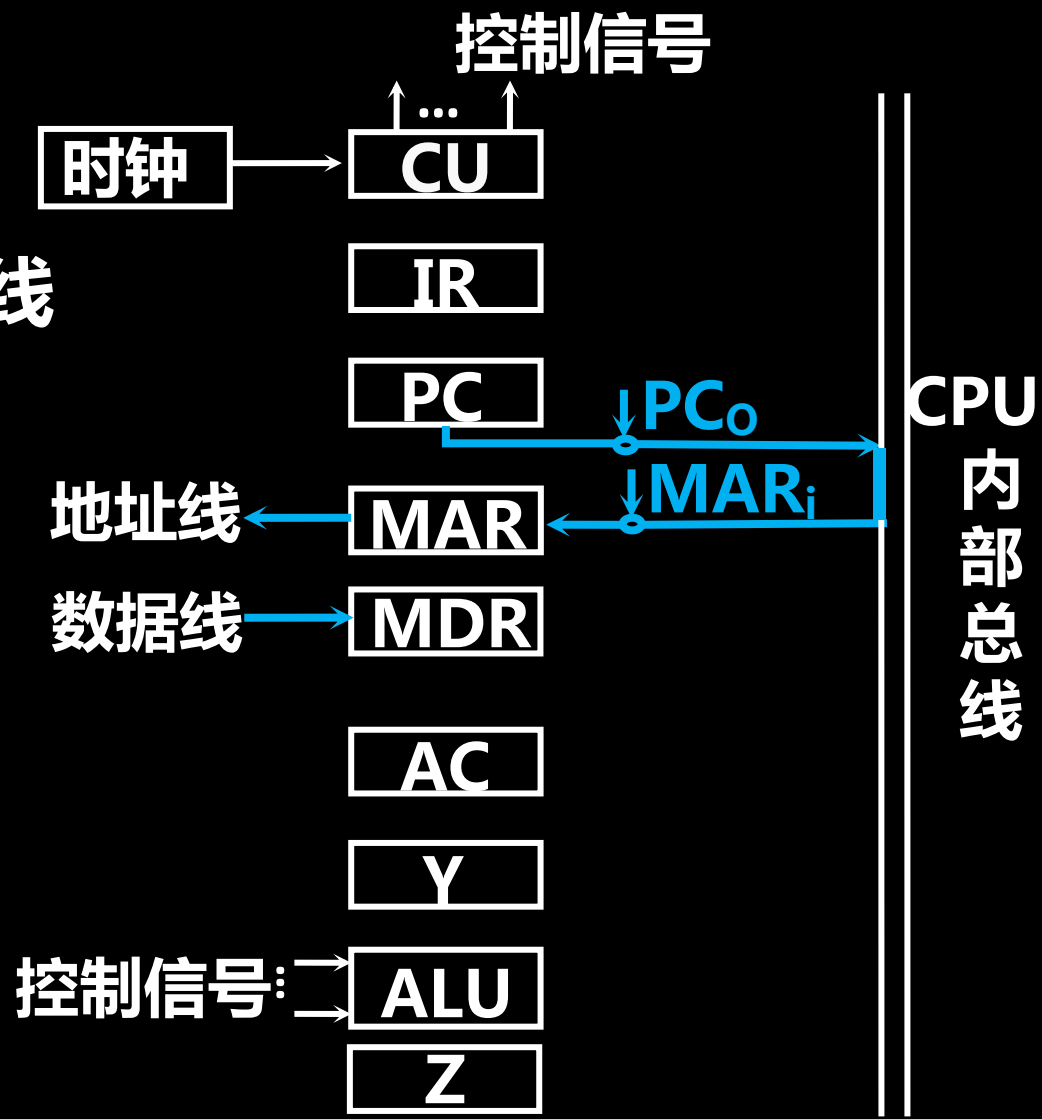
PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR



采用CPU内部总线方式

(1) ADD @ X 取指周期

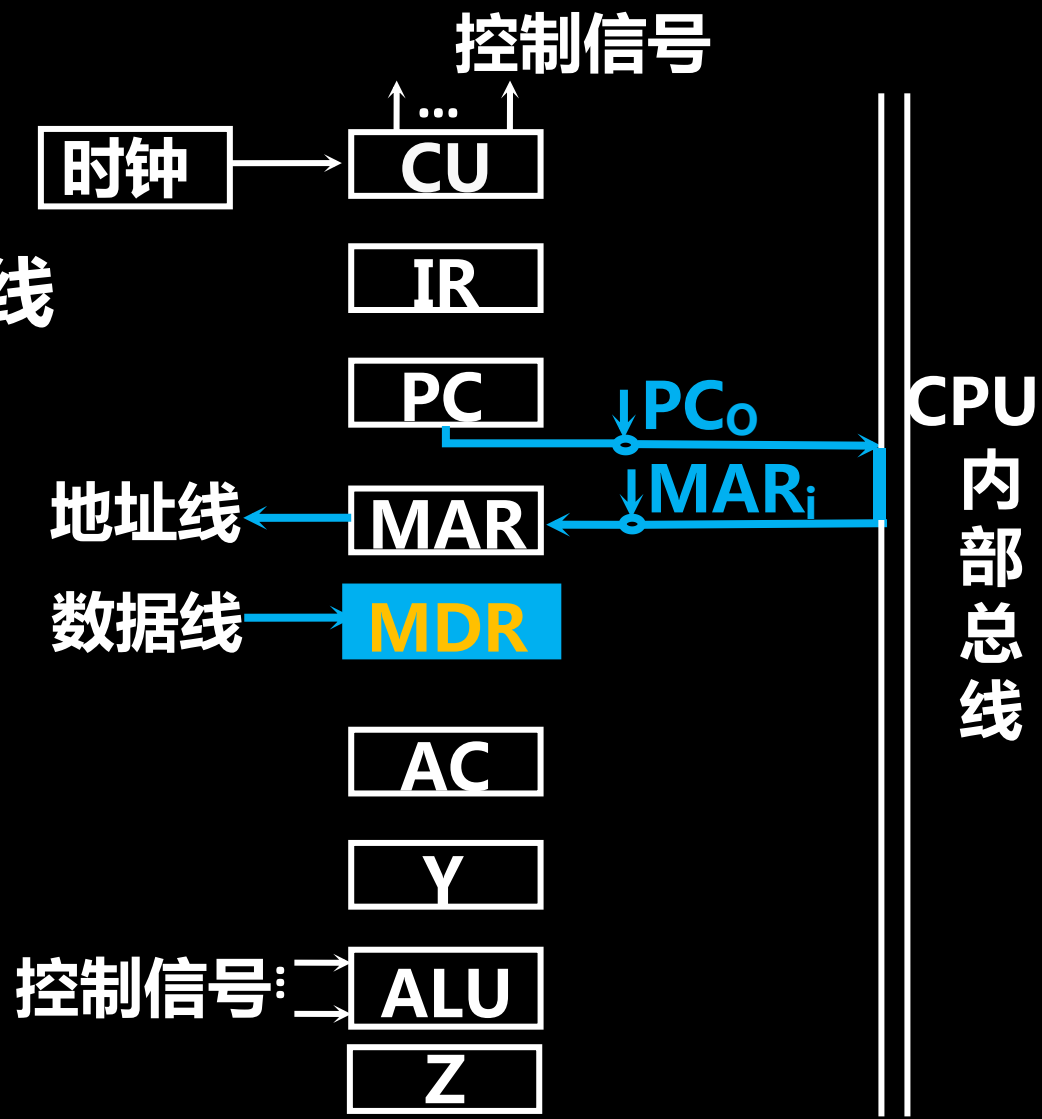
PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

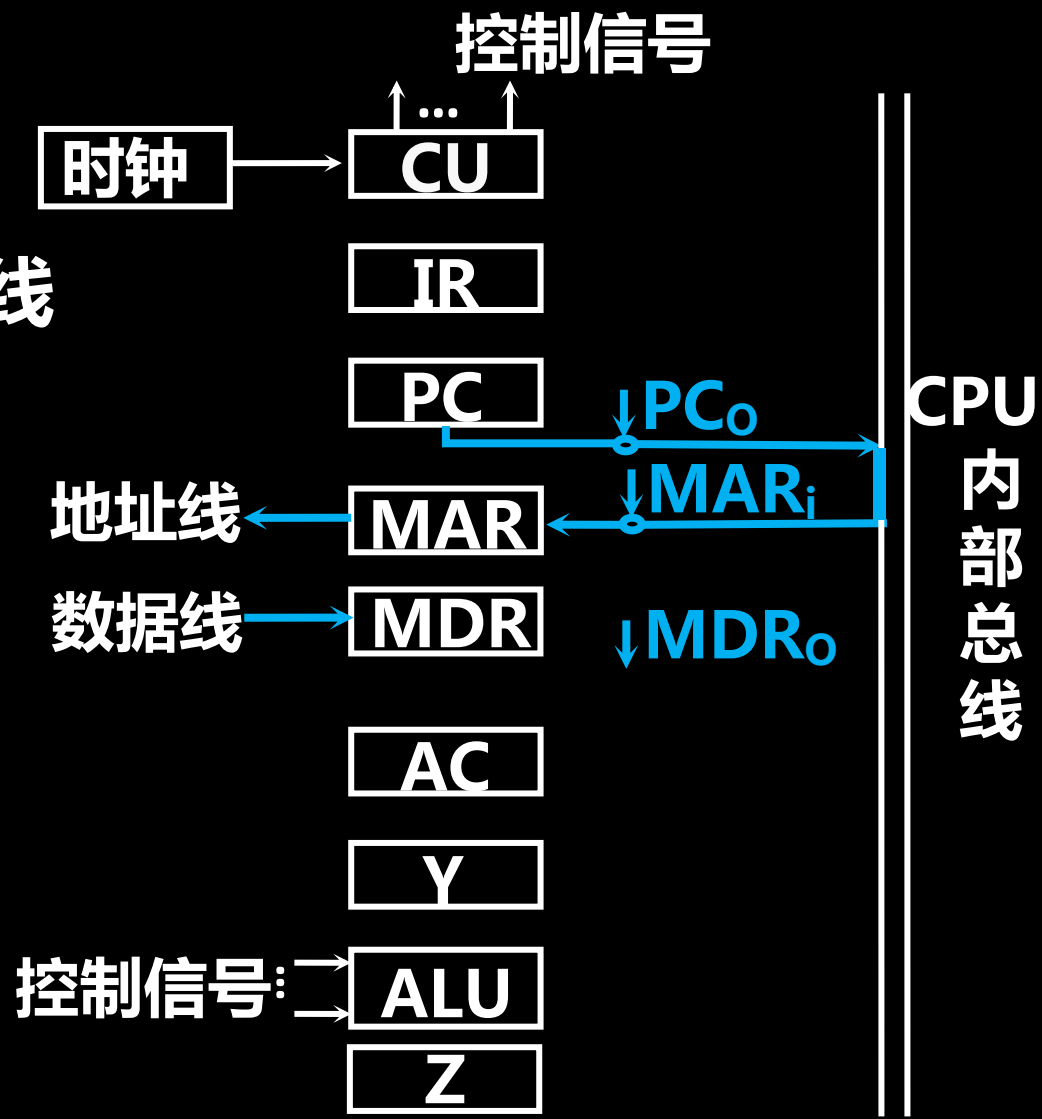
PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR

MDR_O



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

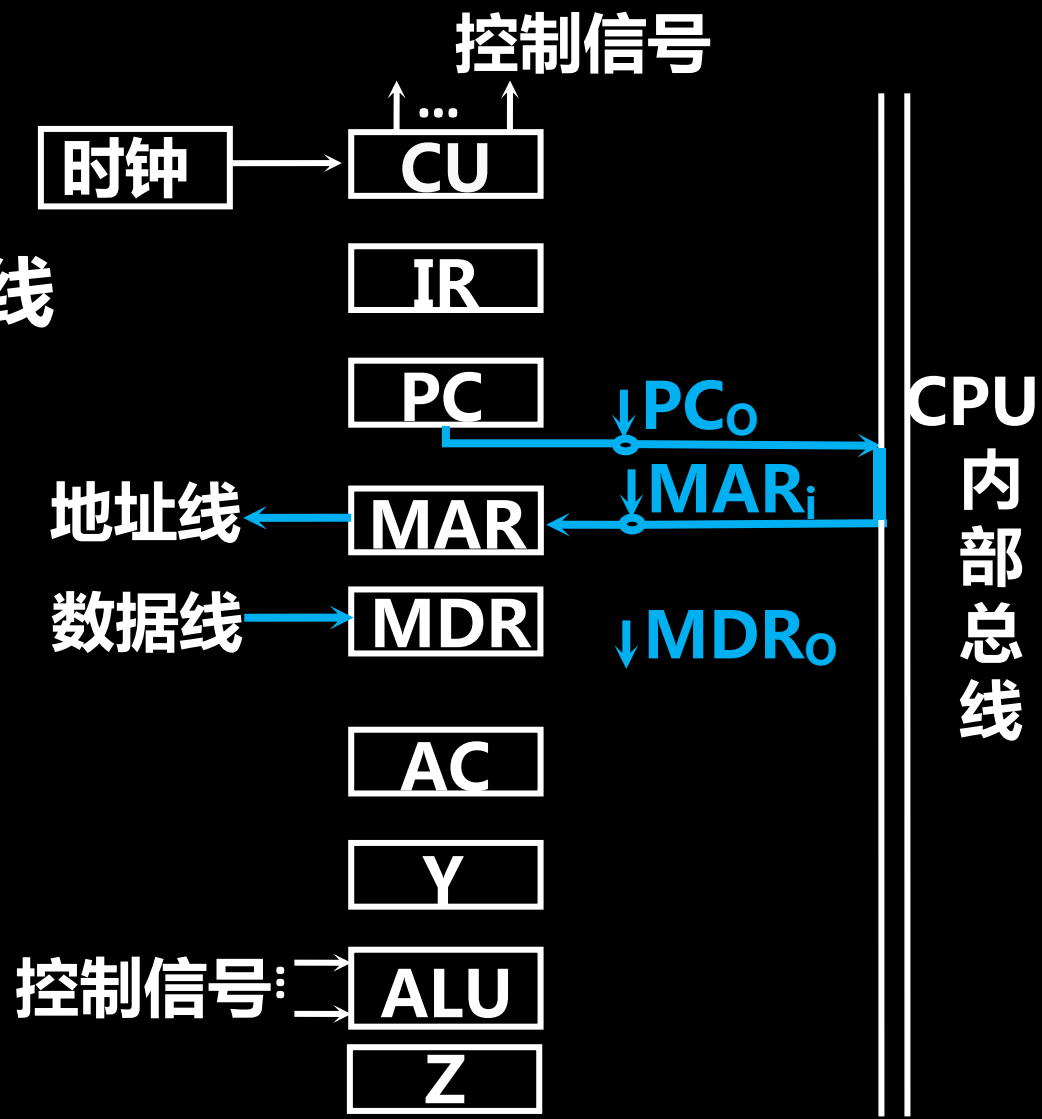
PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR ⟶

MDR_O



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

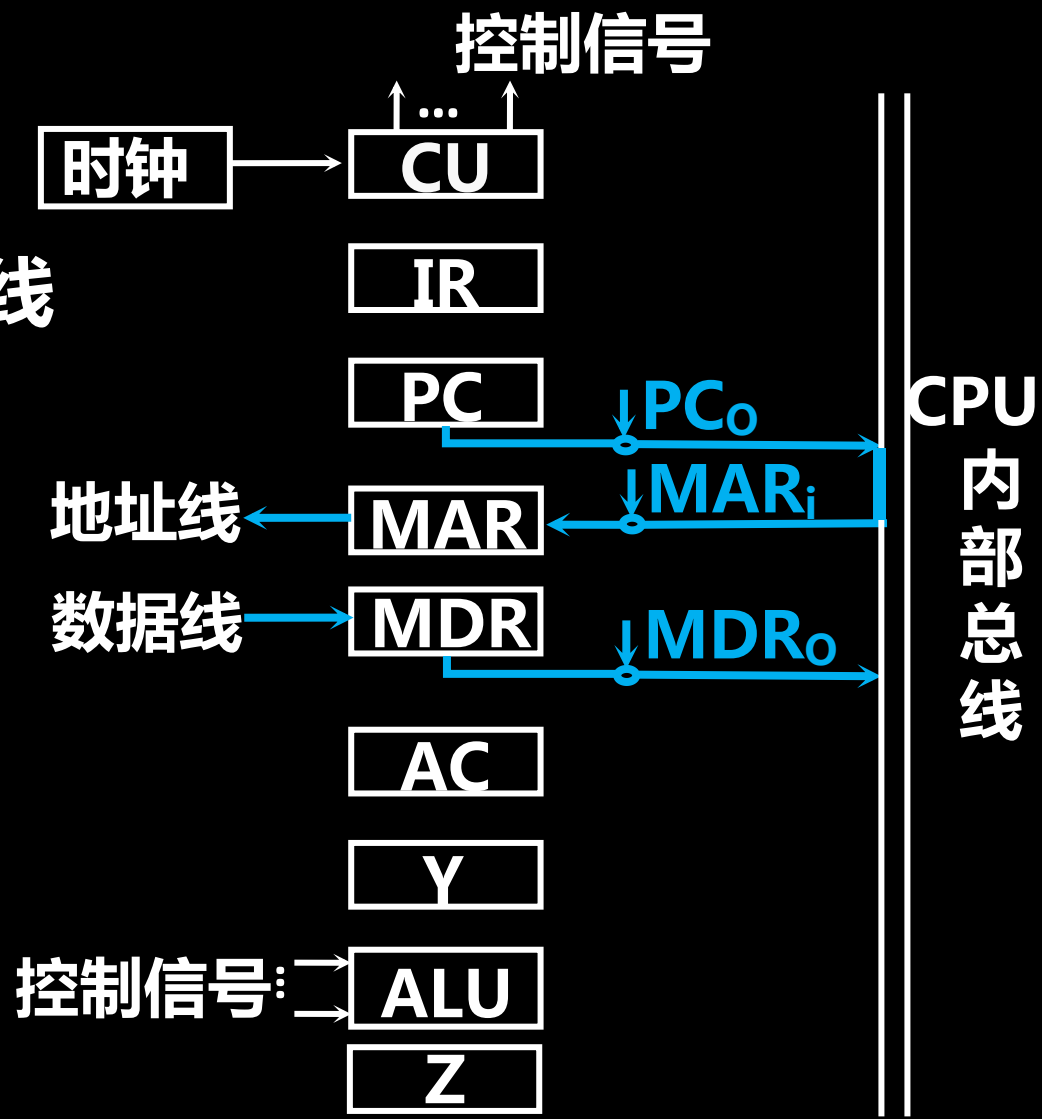
PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR ⟶

MDR_O



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

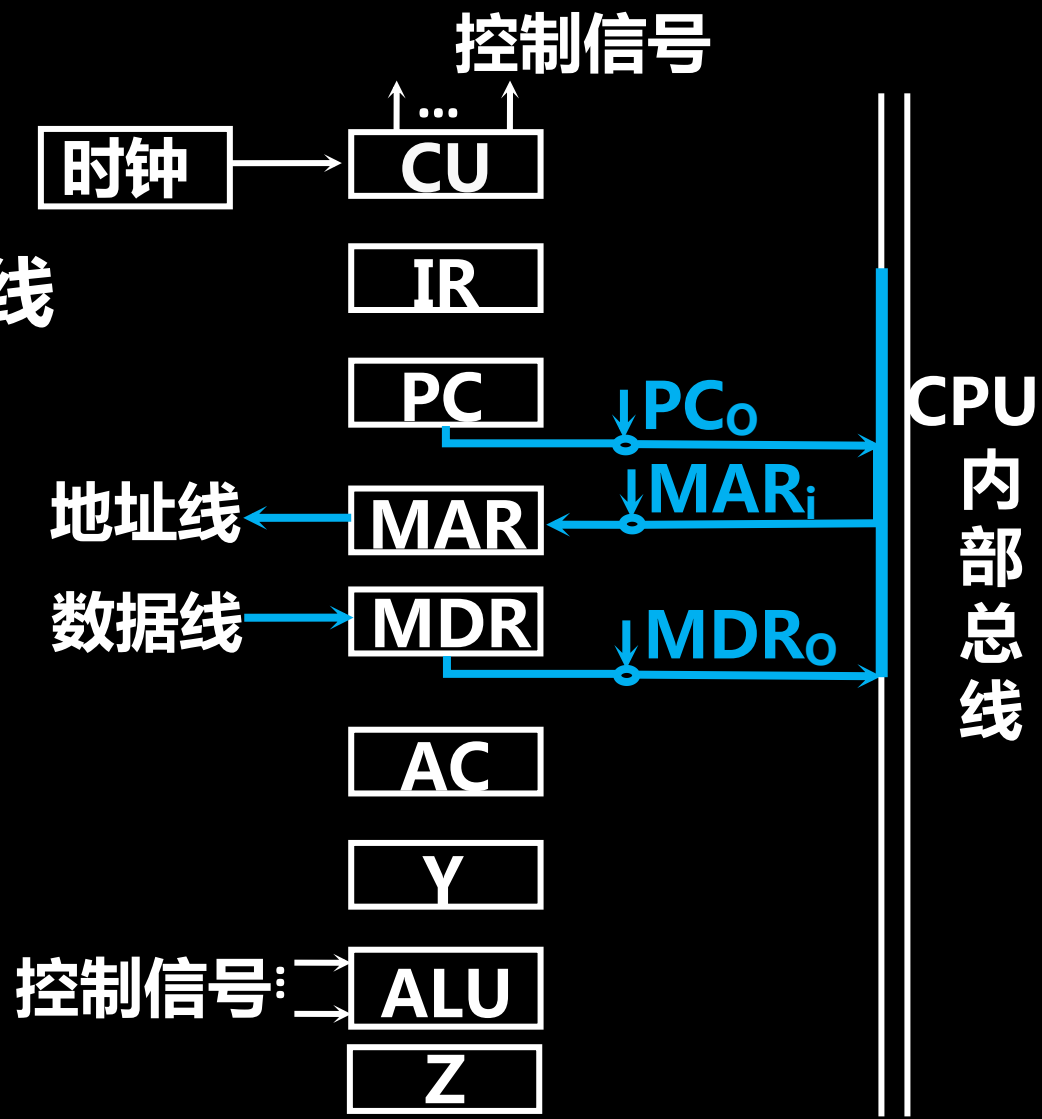
PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR ⟶

MDR_O



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

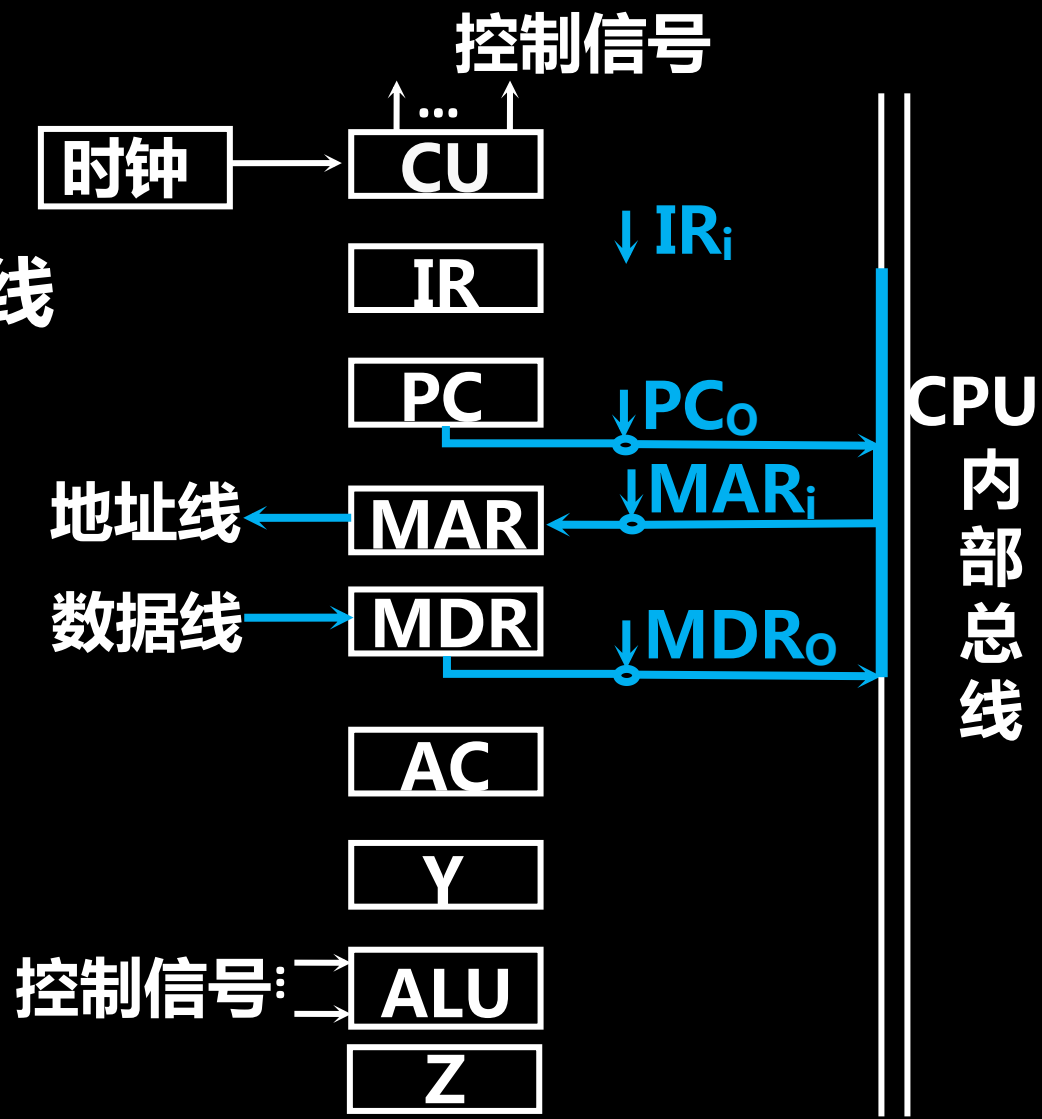
PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR ⟶

MDR_O IR_i



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

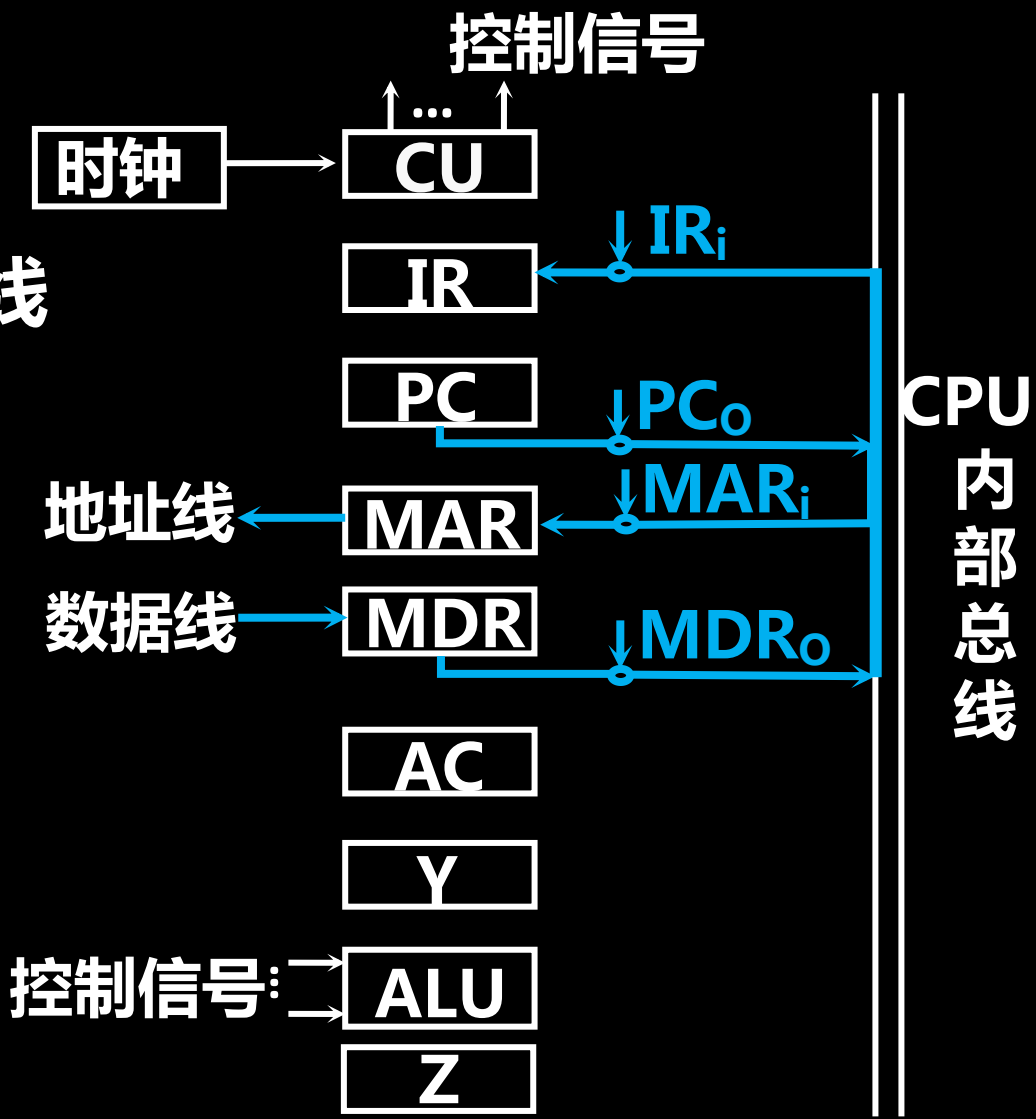
PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR ⟶

MDR_O IR_i



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

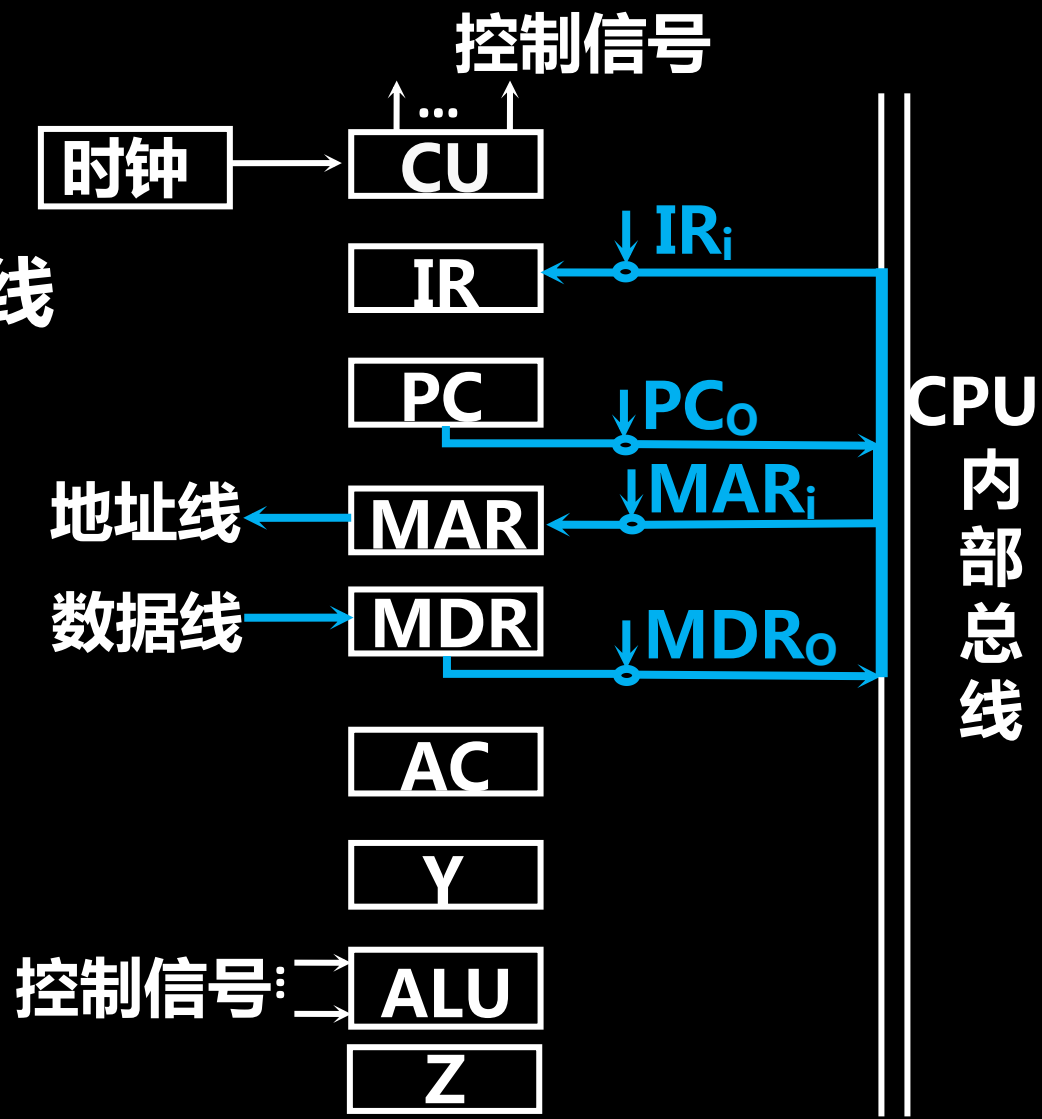
PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR ⟶ IR

MDR_O IR_i



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

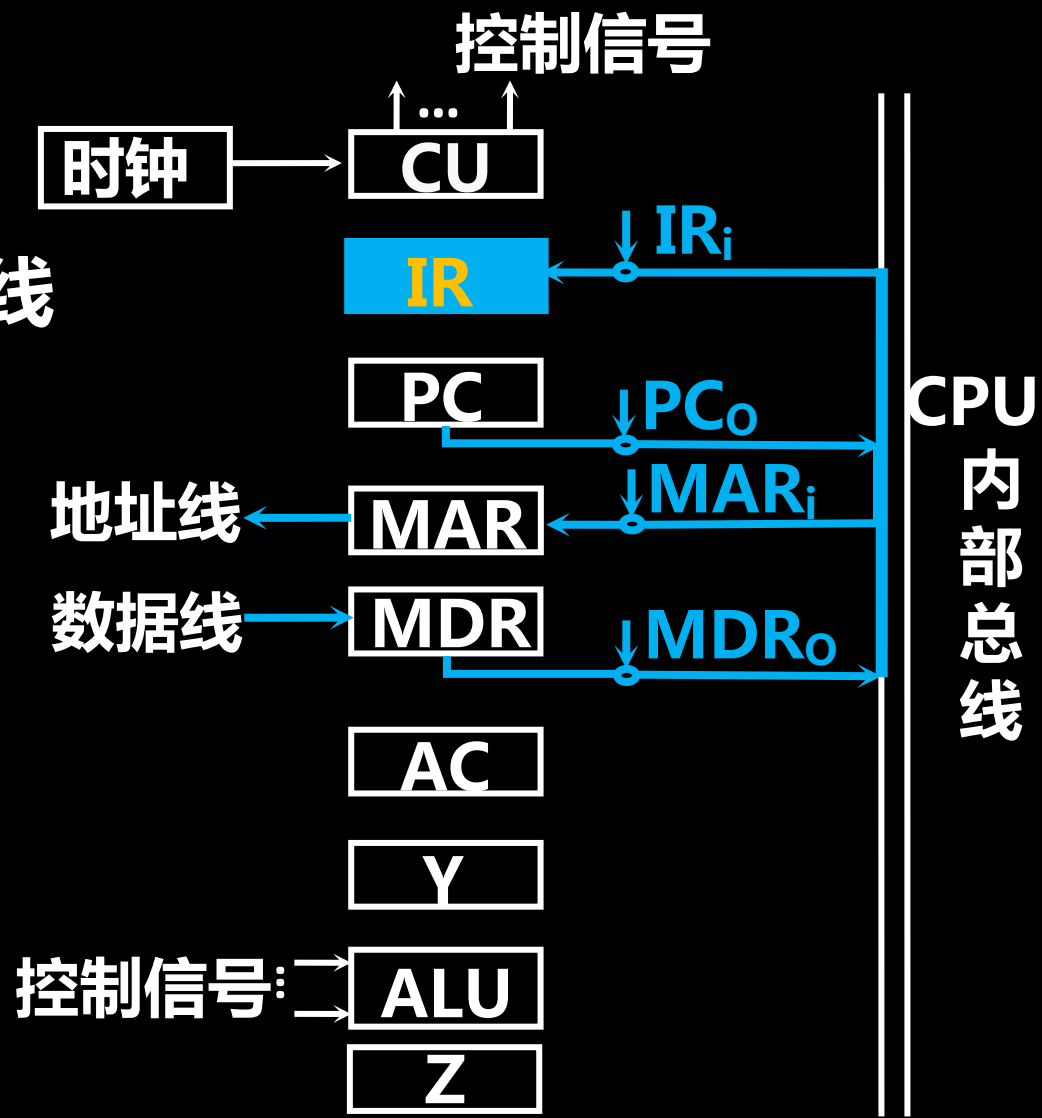
PC_O MAR_i

CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR ⟶ IR

MDR_O IR_i



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

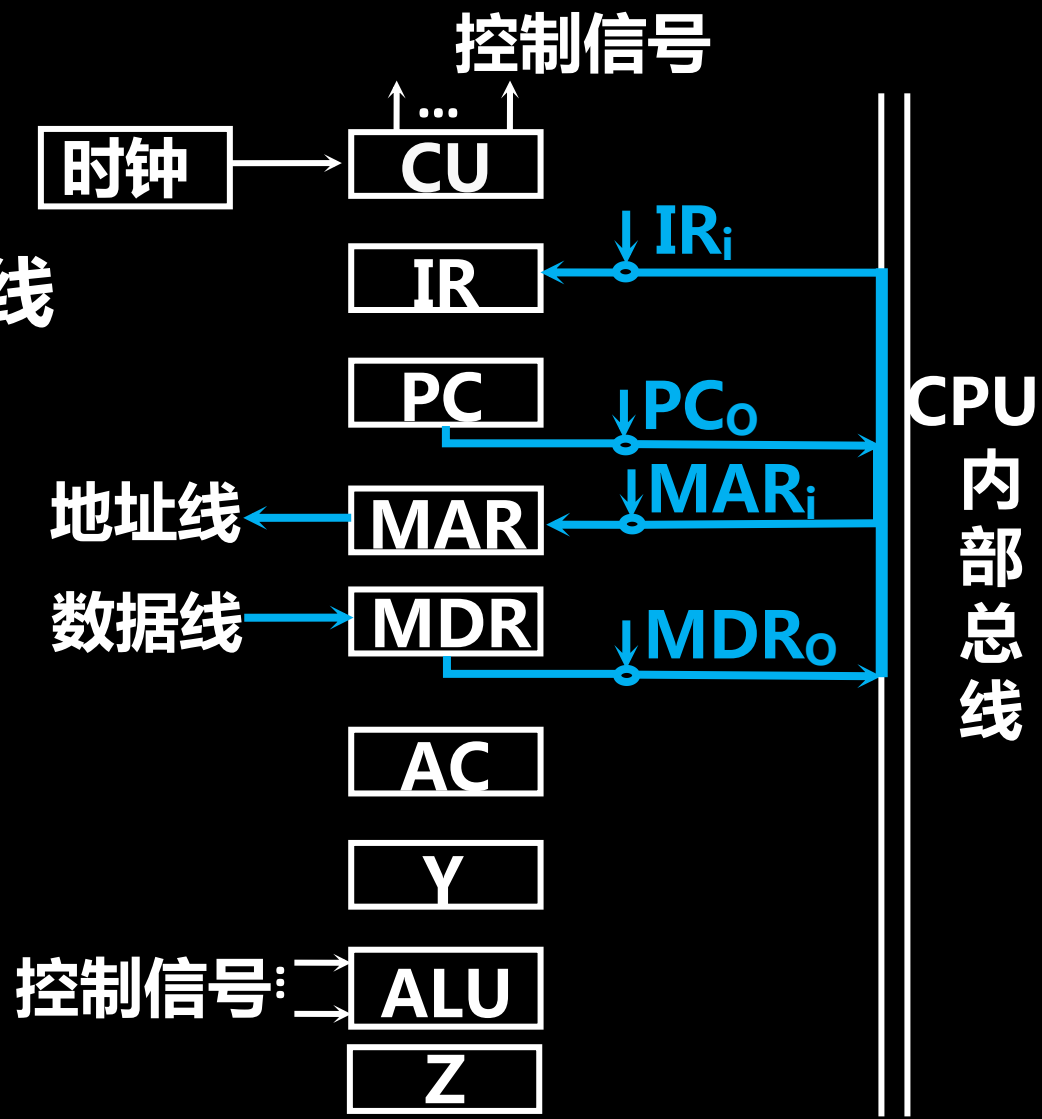
CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR ⟶ IR

MDR_O IR_i

OP(IR)



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

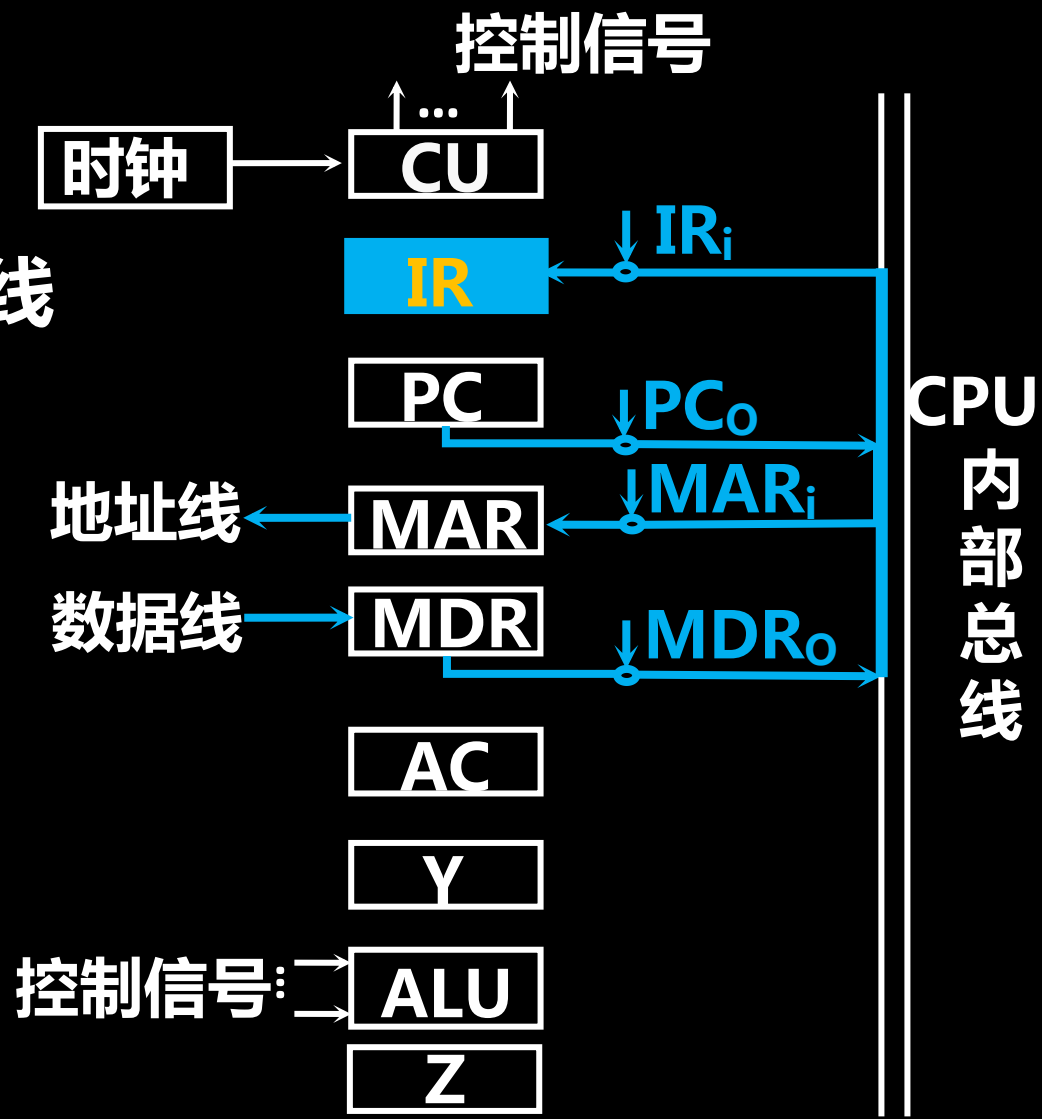
CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR ⟶ IR

MDR_O IR_i

OP(IR)



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

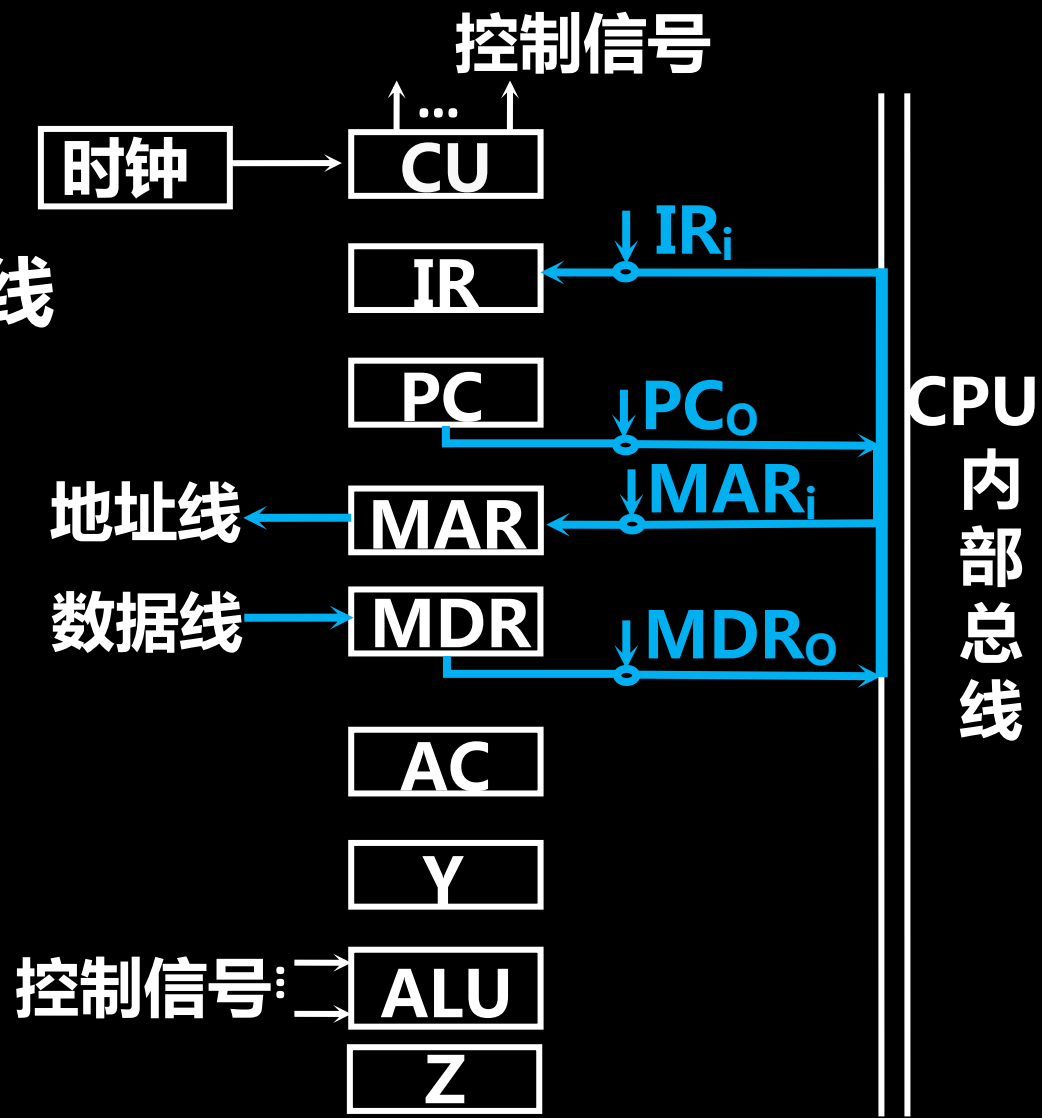
CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR ⟶ IR

MDR_O IR_i

OP(IR) ⟶



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

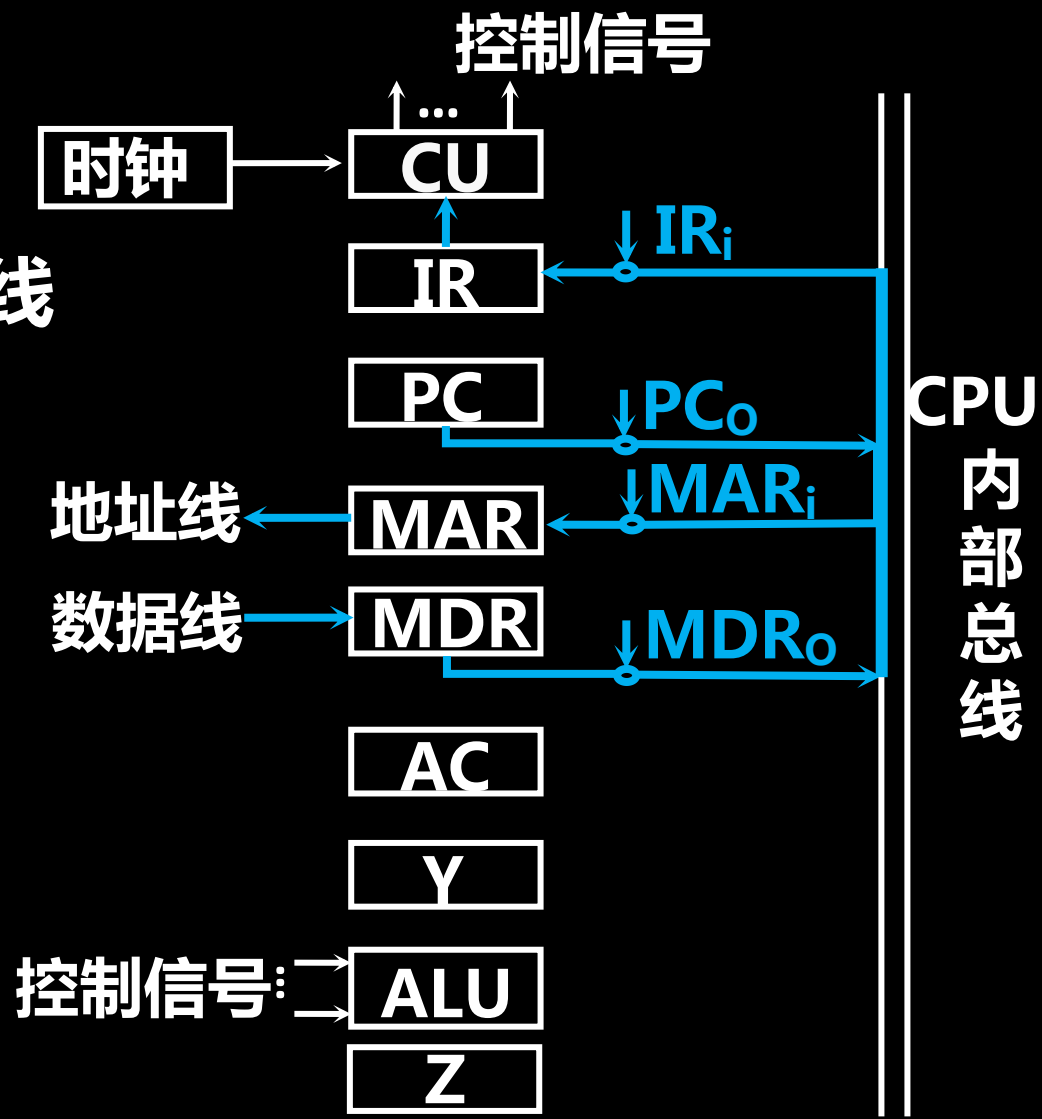
CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR ⟶ IR

MDR_O IR_i

OP(IR) ⟶



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

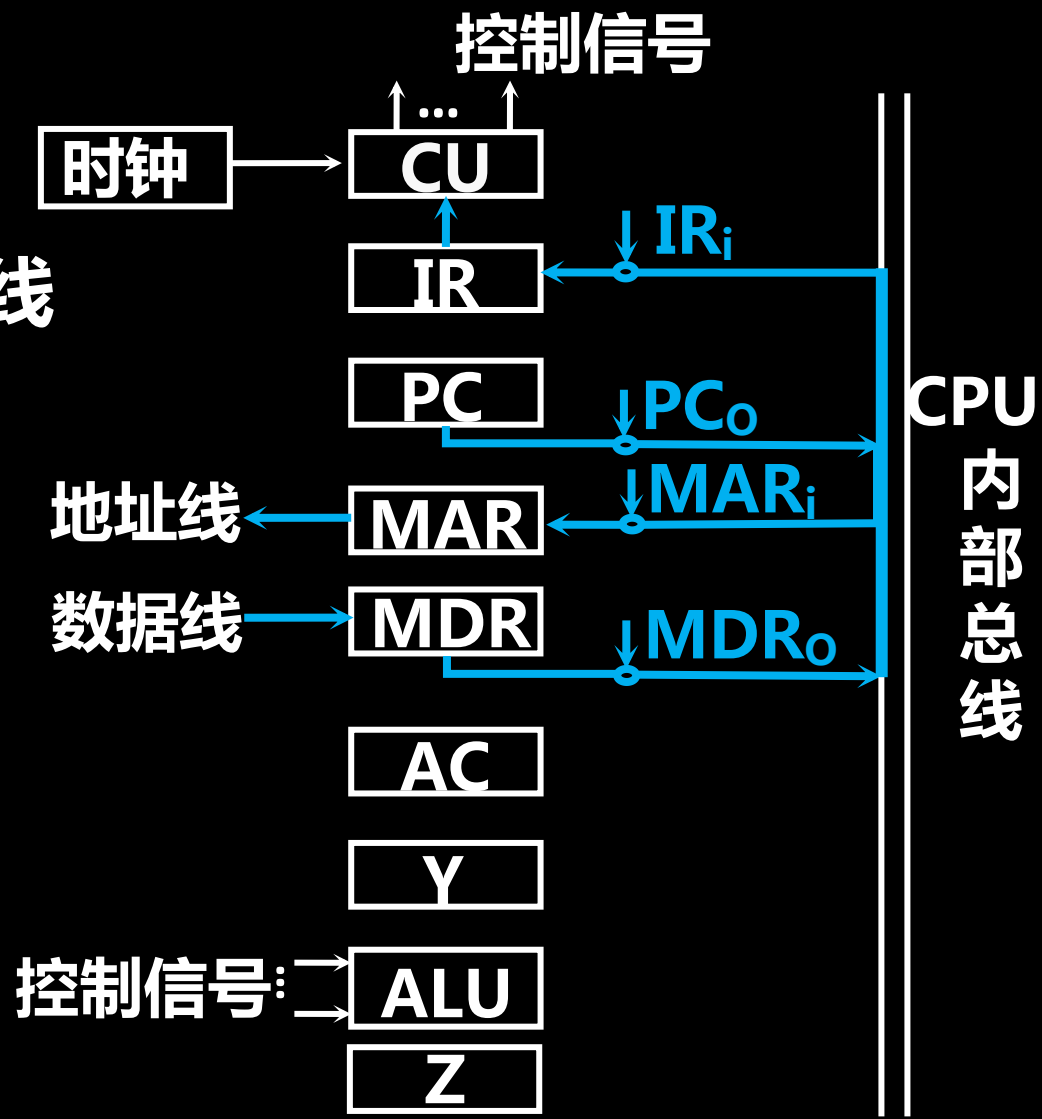
CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR ⟶ IR

MDR_O IR_i

OP(IR) ⟶ CU



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

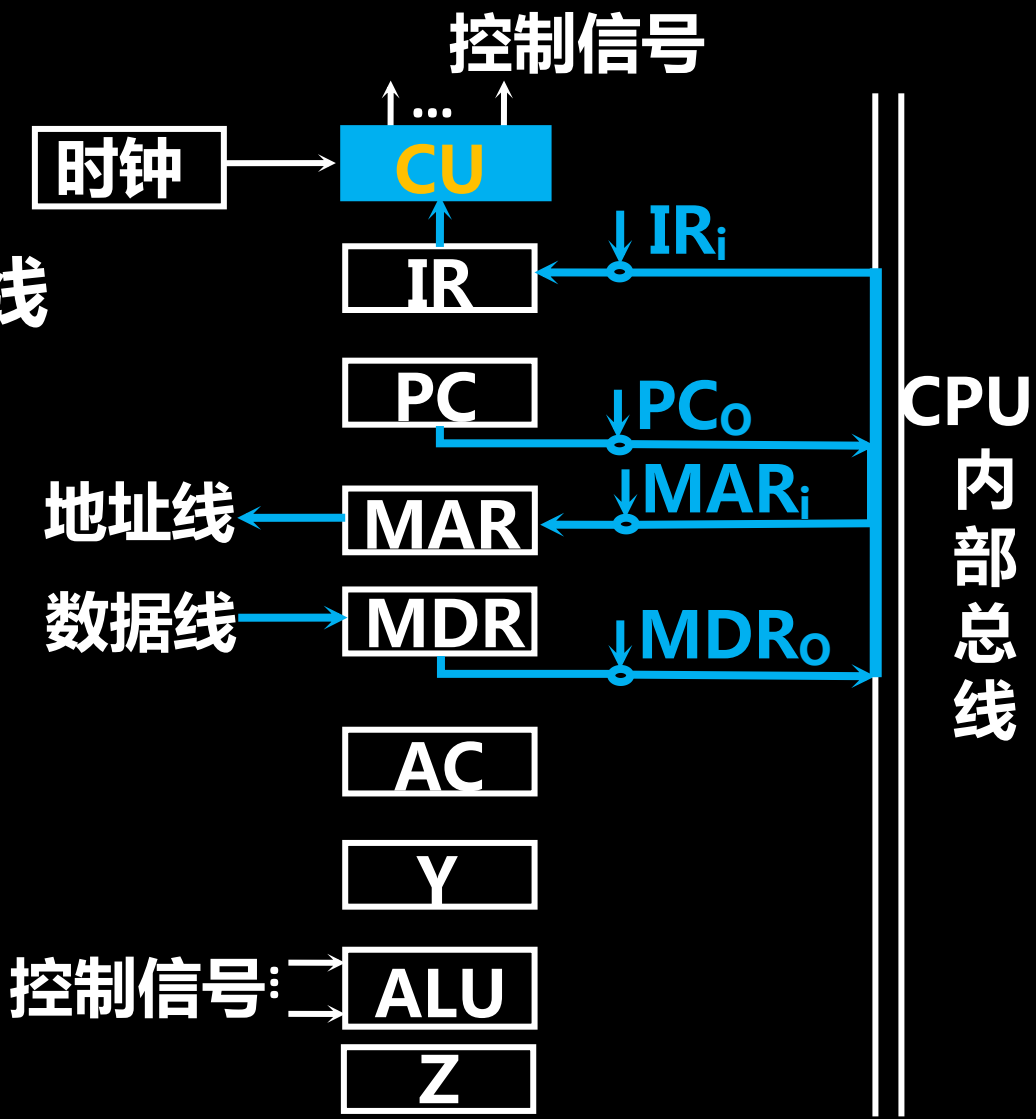
CU 发读命令 1 ⟶ R

数据线 ⟶ MDR

MDR ⟶ IR

MDR_O IR_i

OP(IR) ⟶ CU



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

CU 发读命令 1 ⟶ R

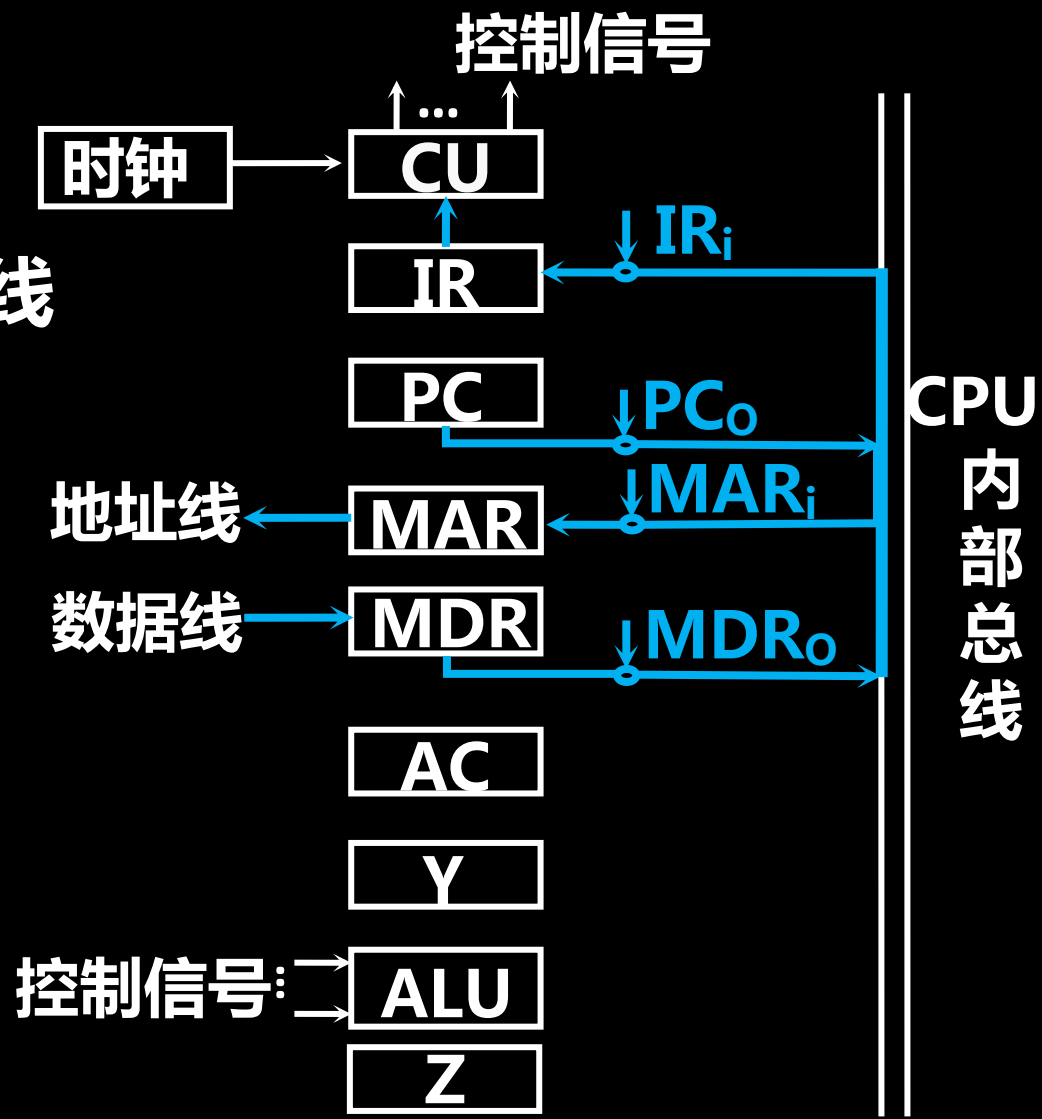
数据线 ⟶ MDR

MDR ⟶ IR

MDR_O IR_i

OP(IR) ⟶ CU

(PC)+1 ⟶ PC



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

CU 发读命令 1 ⟶ R

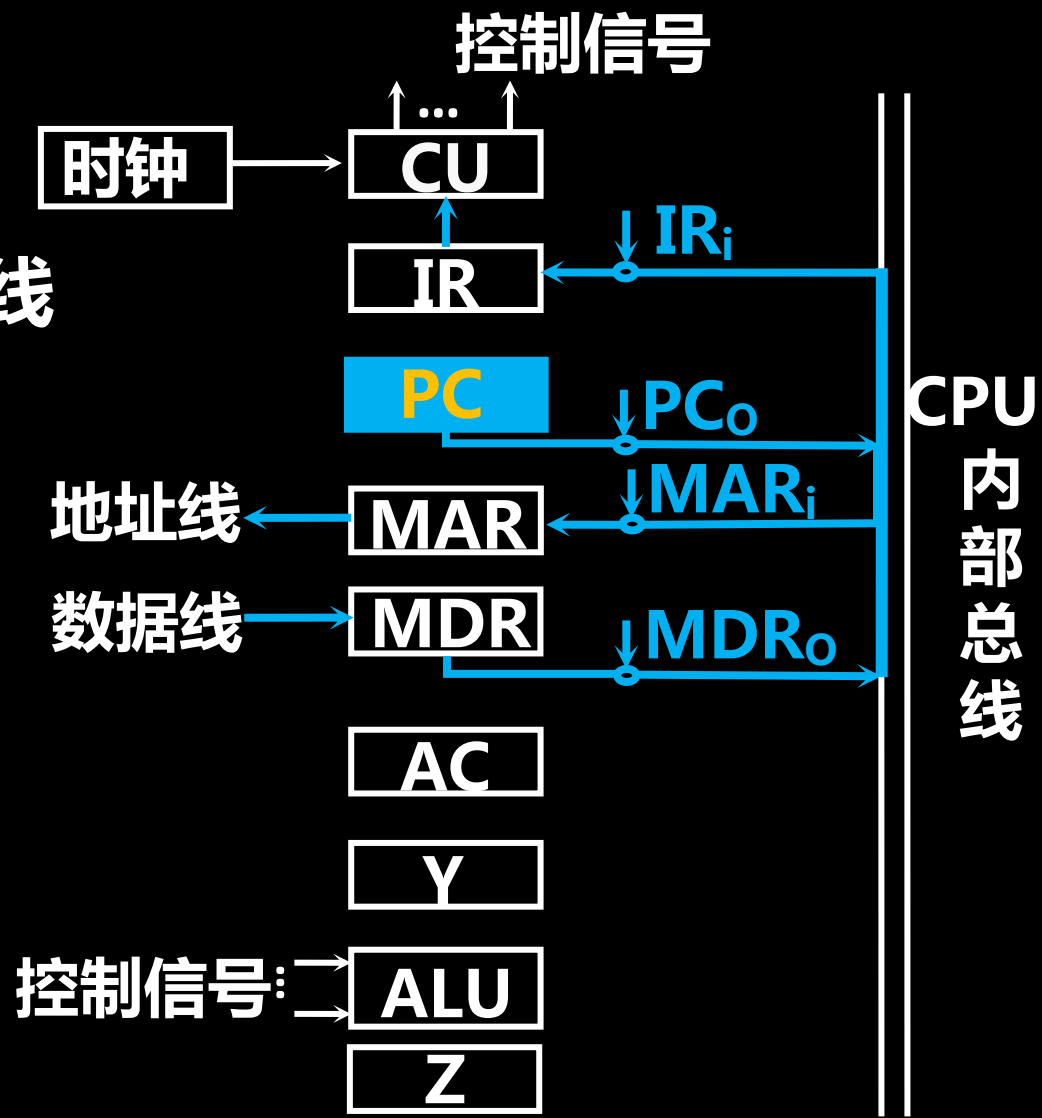
数据线 ⟶ MDR

MDR ⟶ IR

MDR_O IR_i

OP(IR) ⟶ CU

(PC)+1 ⟶ PC



采用CPU内部总线方式

(1) ADD @ X 取指周期

PC ⟶ MAR ⟶ 地址线

PC_O MAR_i

CU 发读命令 1 ⟶ R

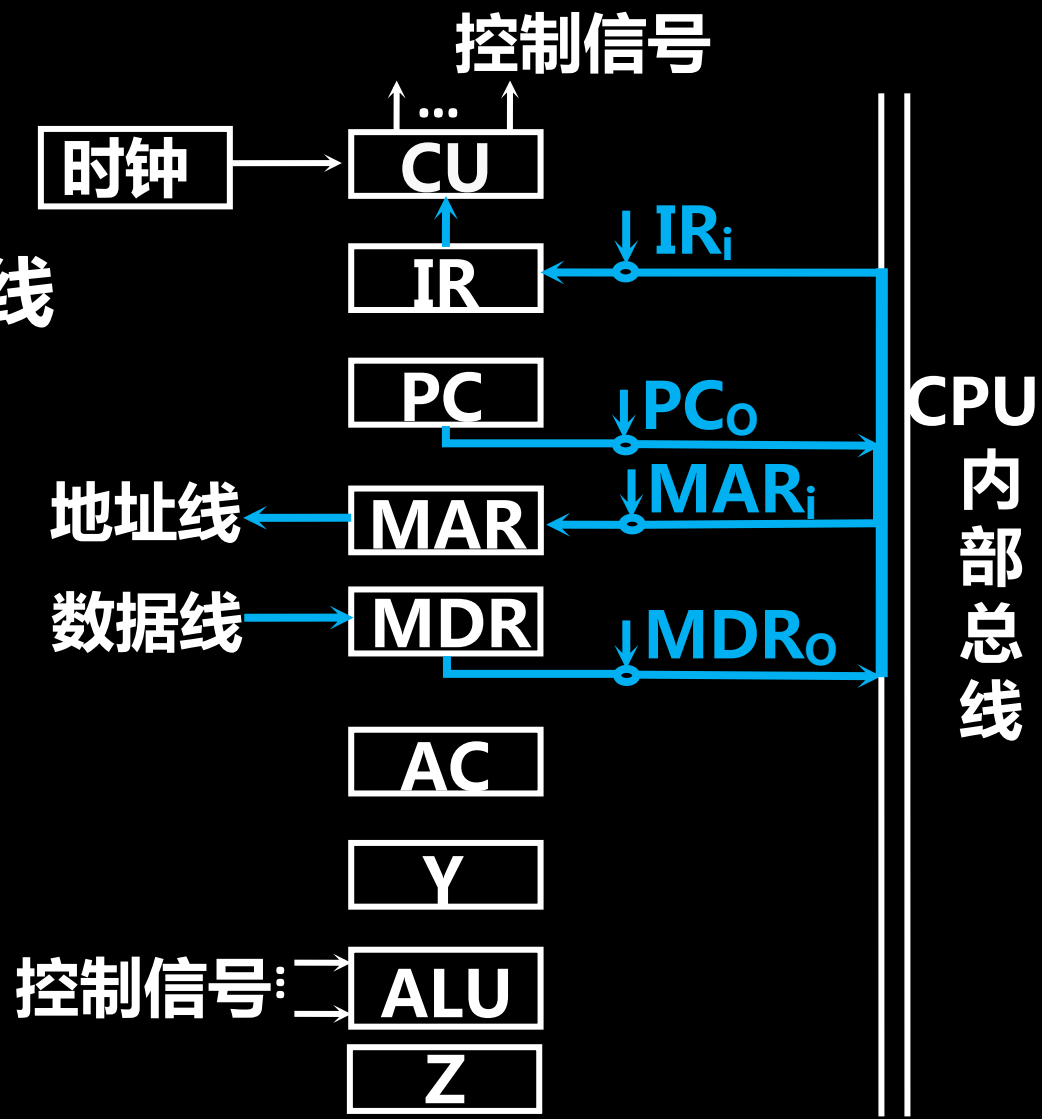
数据线 ⟶ MDR

MDR ⟶ IR

MDR_O IR_i

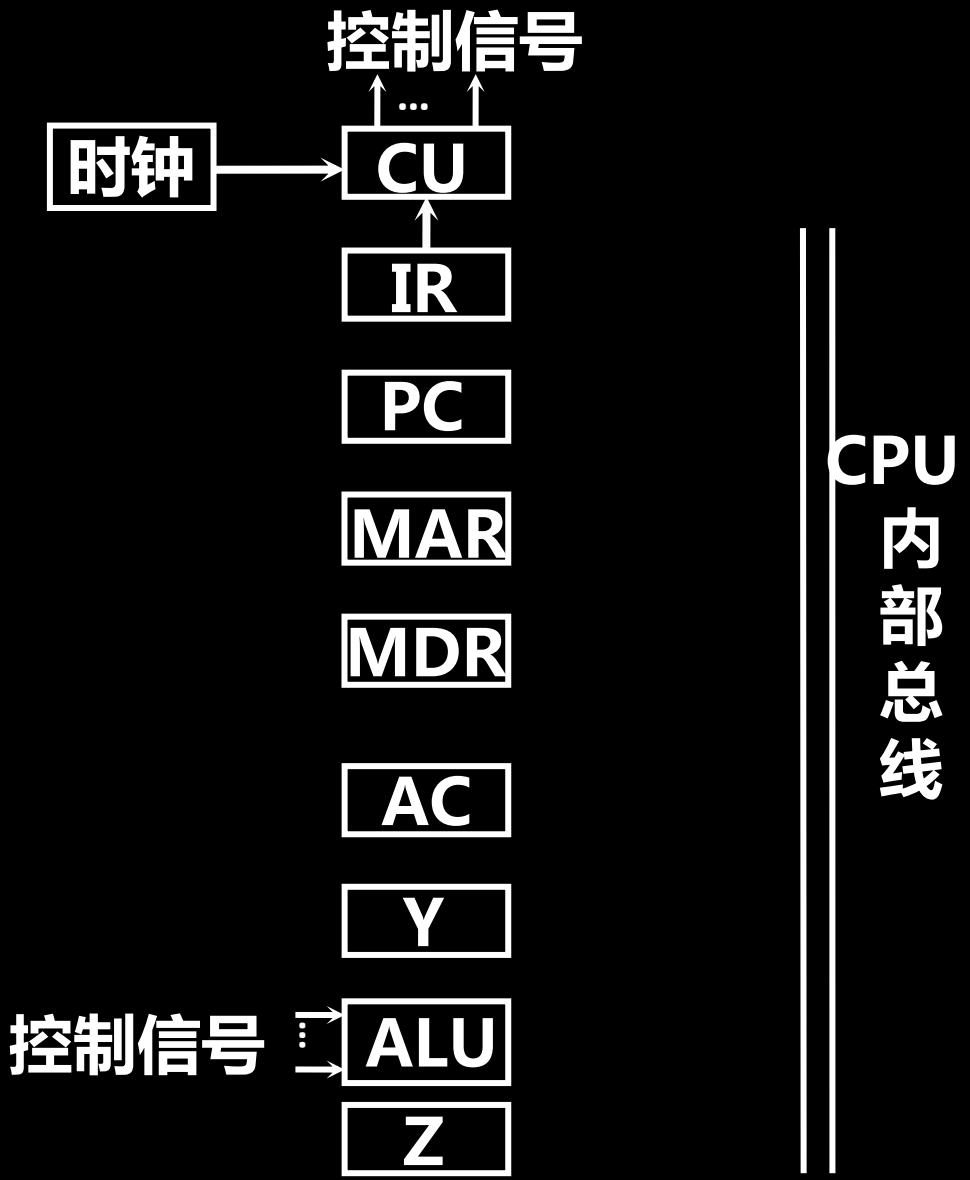
OP(IR) ⟶ CU

(PC)+1 ⟶ PC



采用CPU内部总线方式

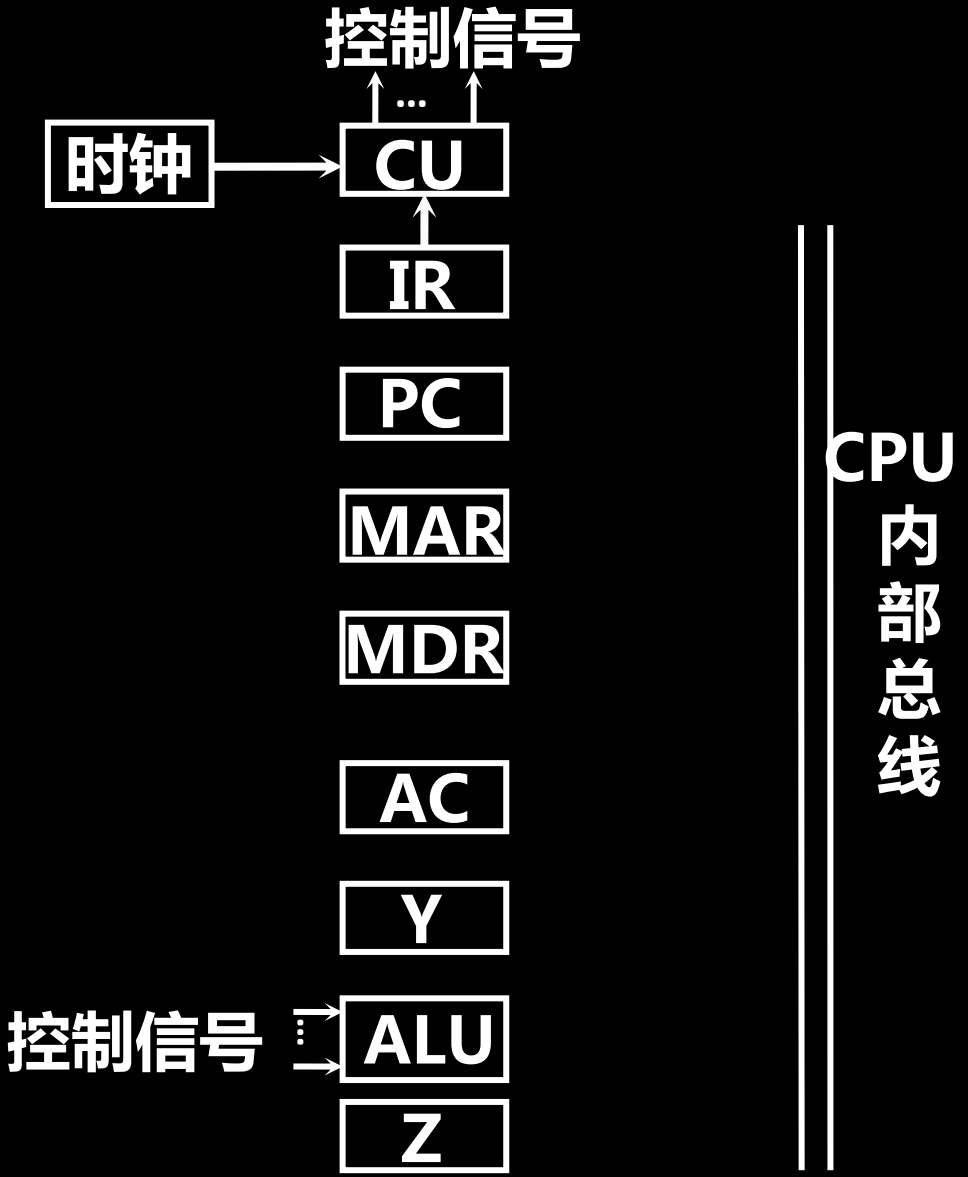
(2) ADD @ X 间址周期



采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

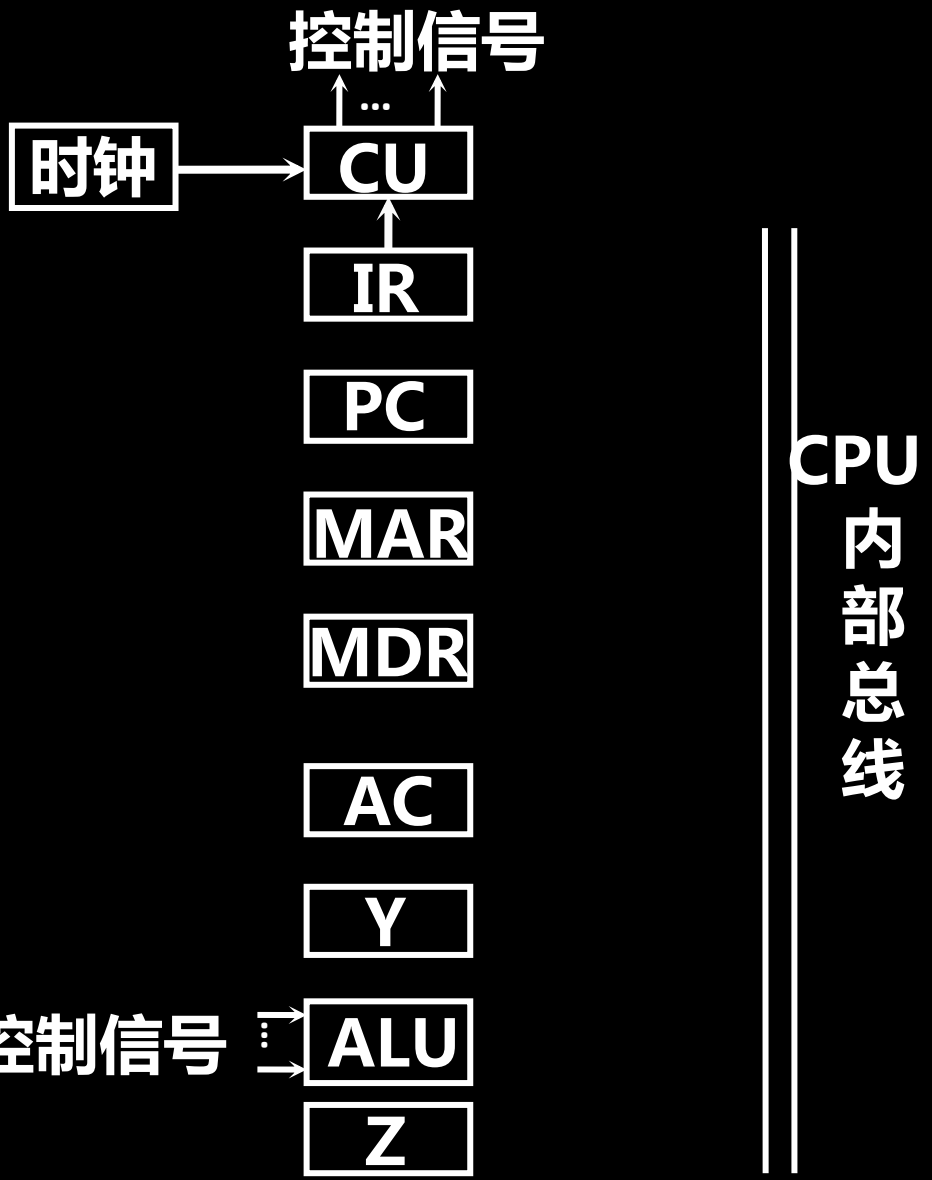


采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR

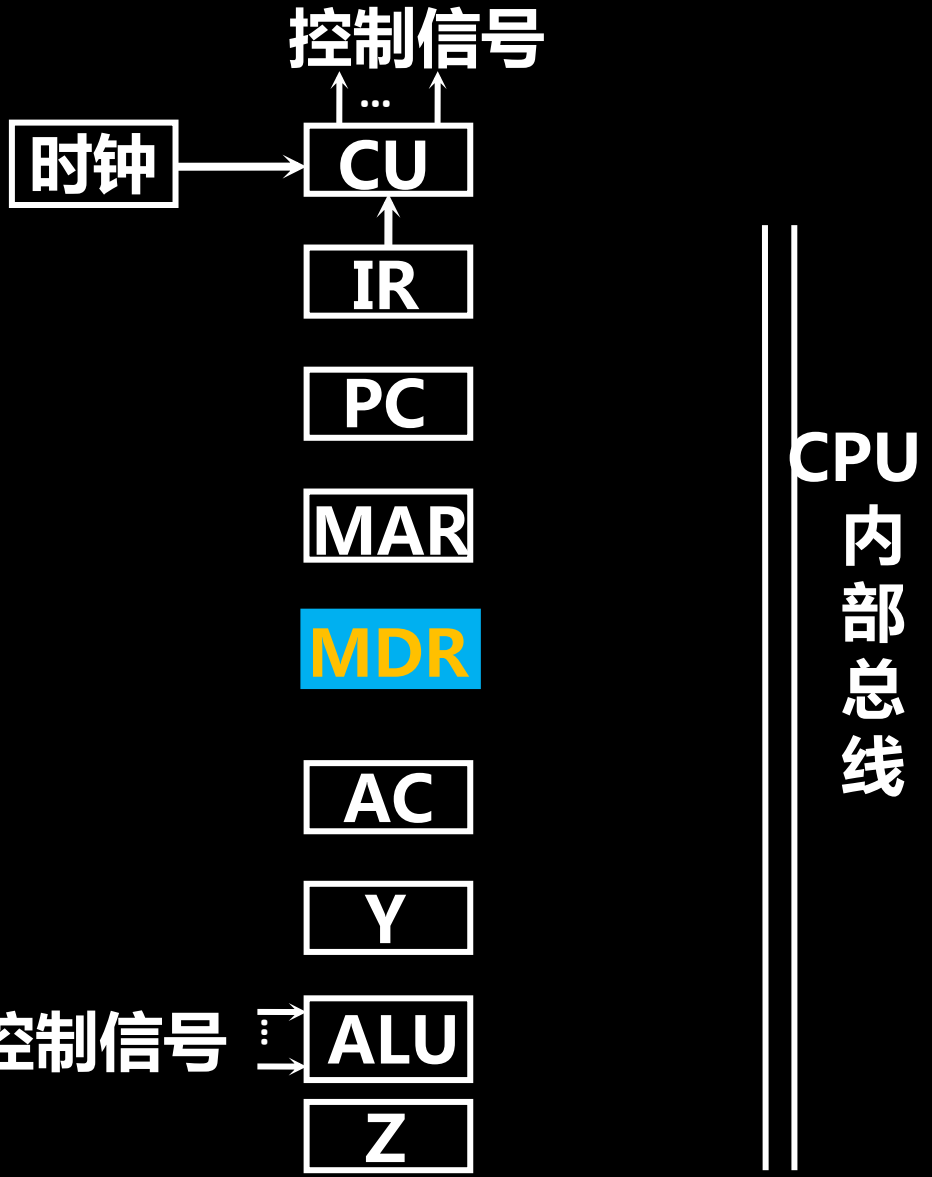


采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR



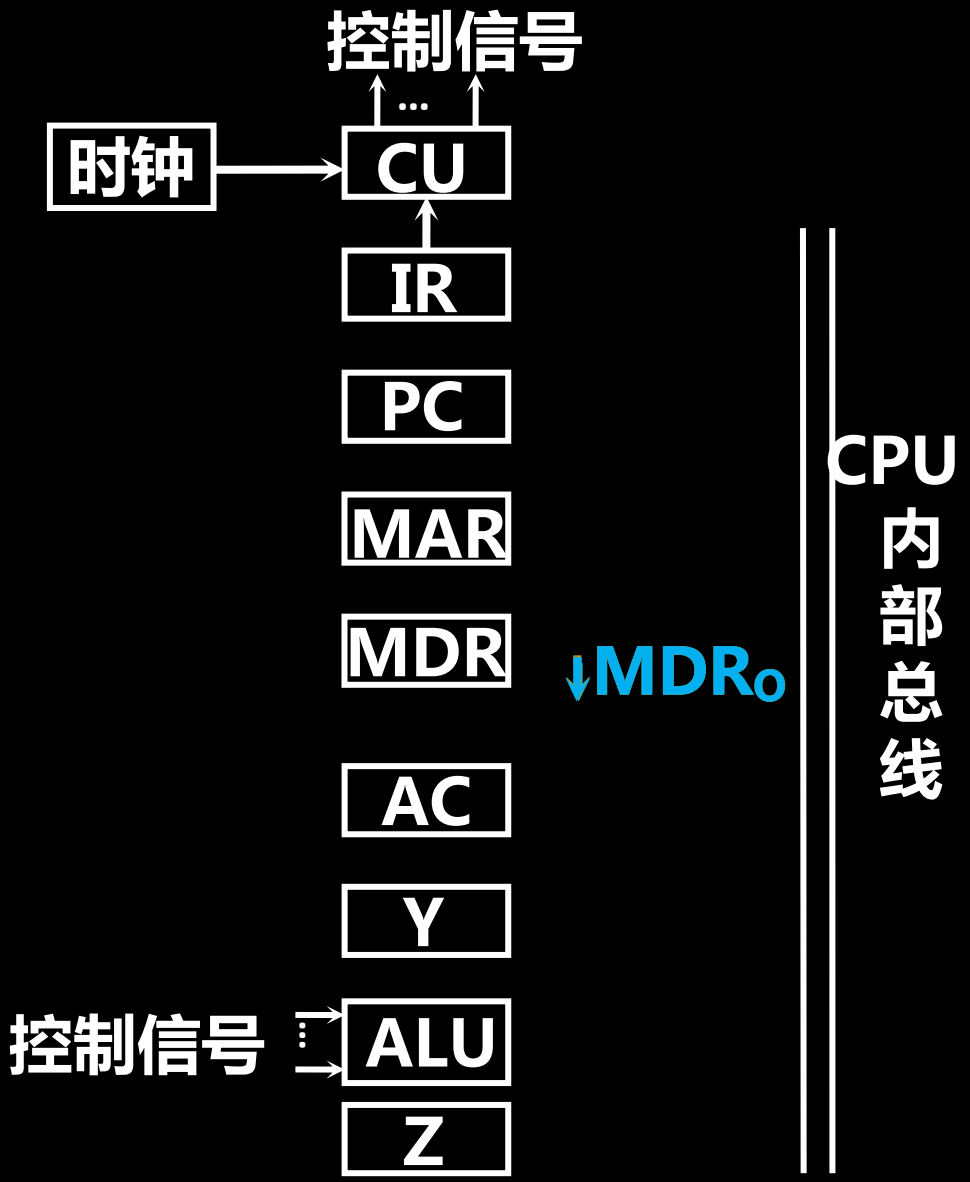
采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR

MDR_o



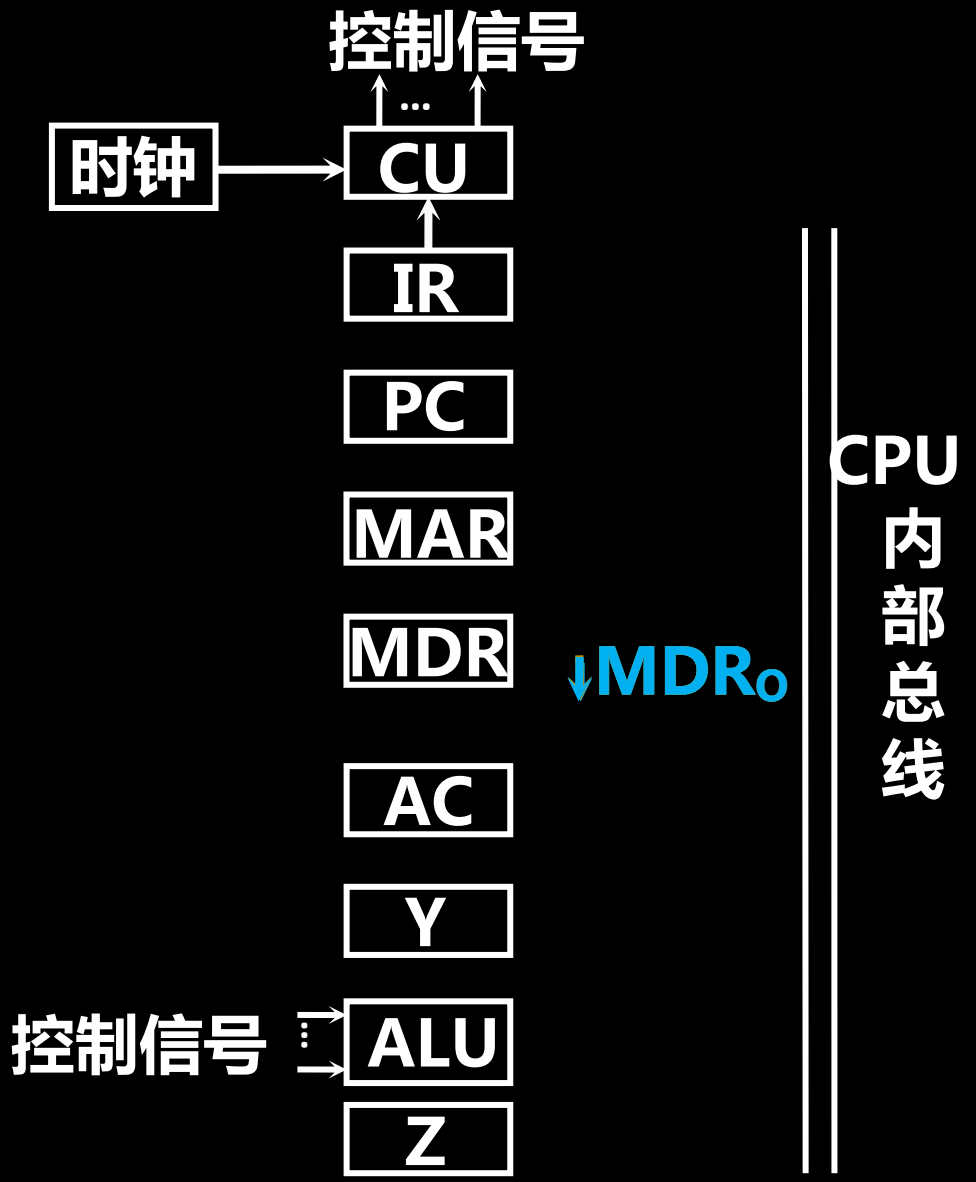
采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶

MDR_o



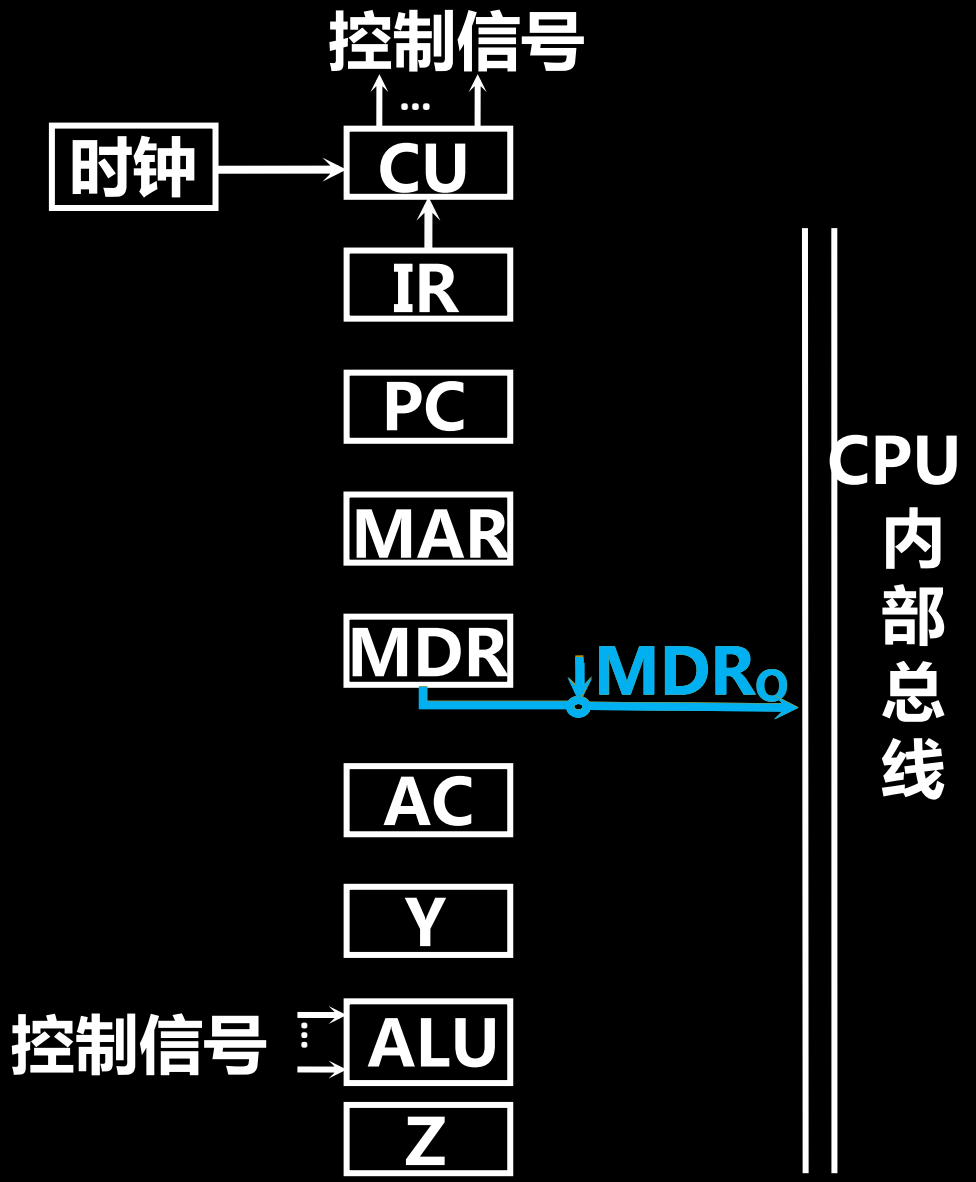
采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶

MDR_o



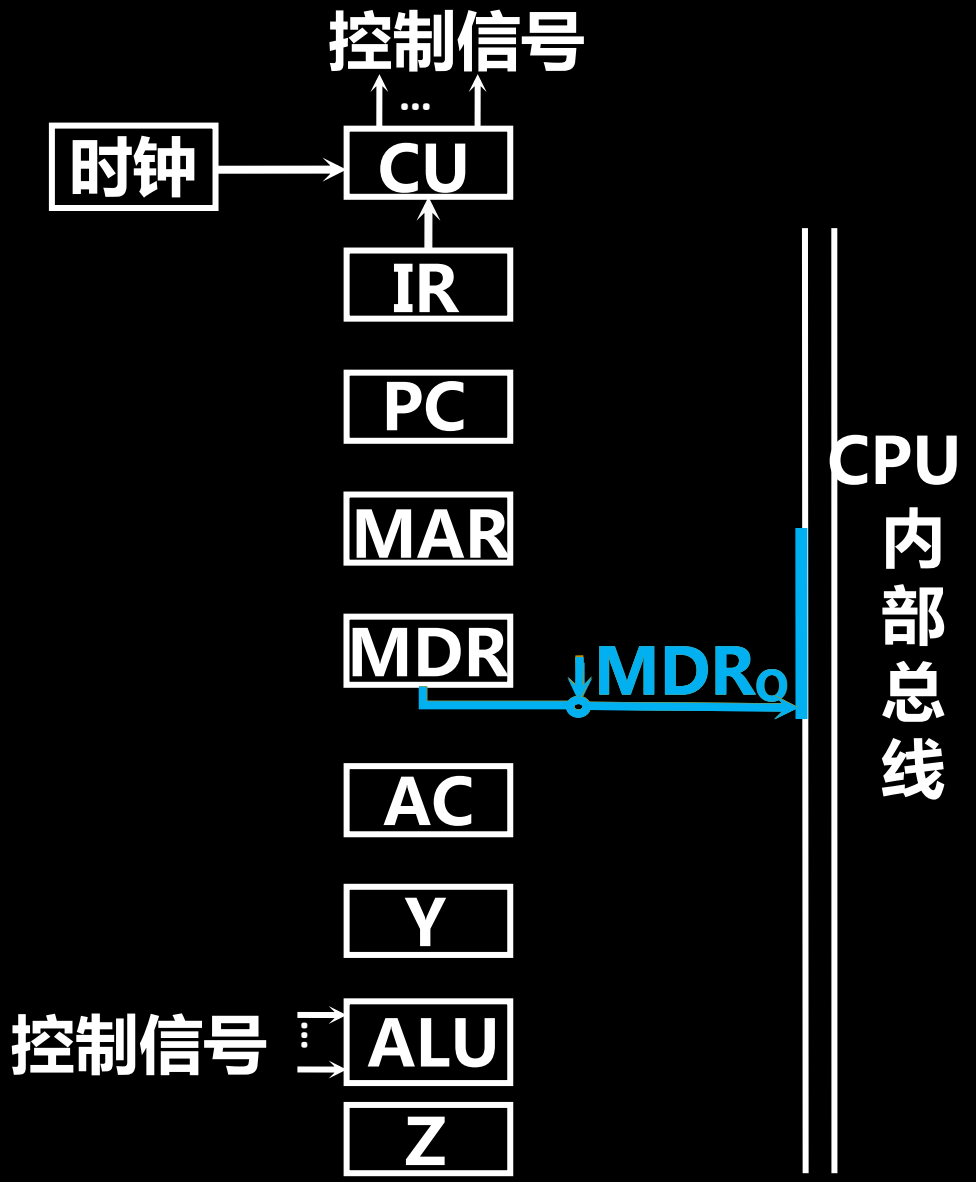
采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶

MDR_o



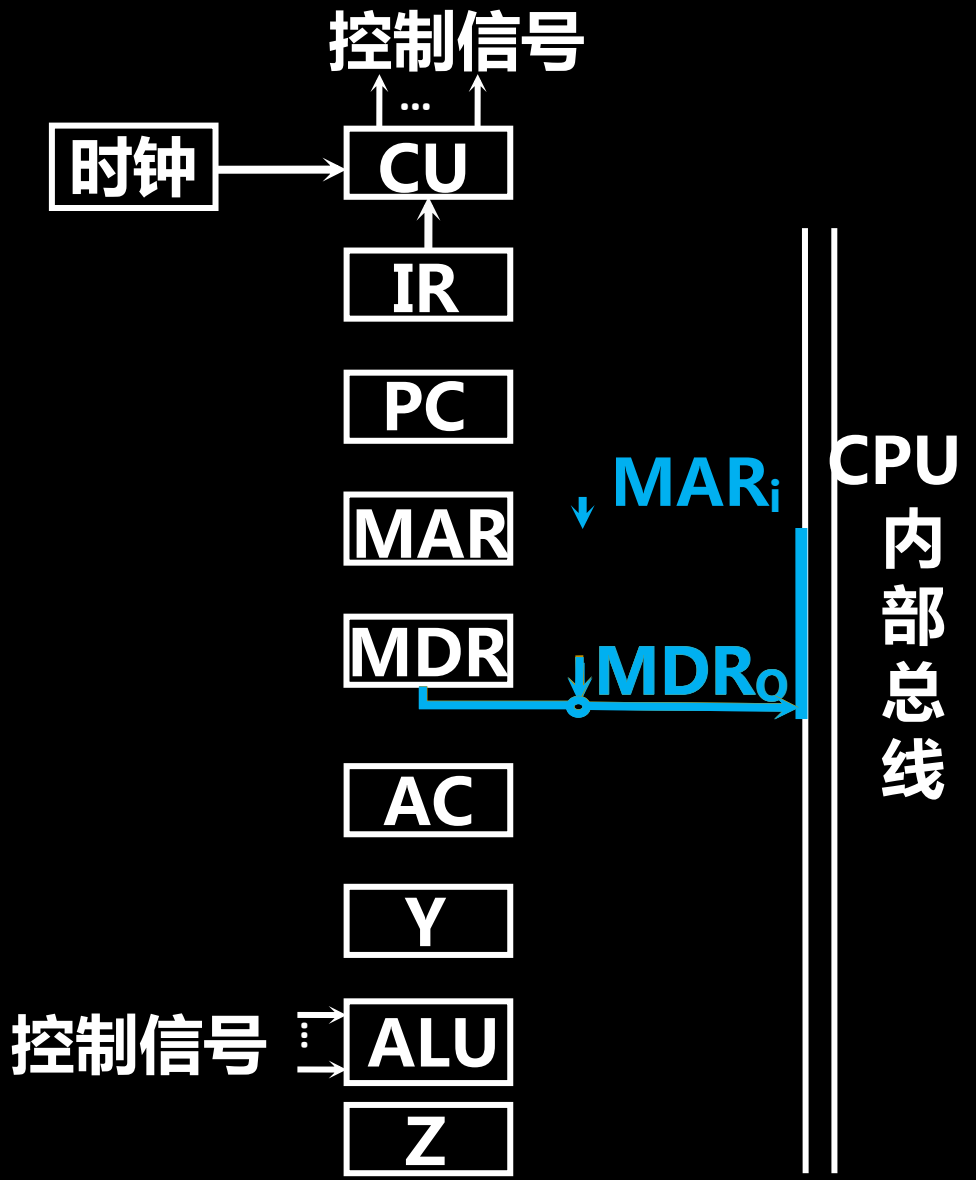
采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶

MDR_O MAR_i



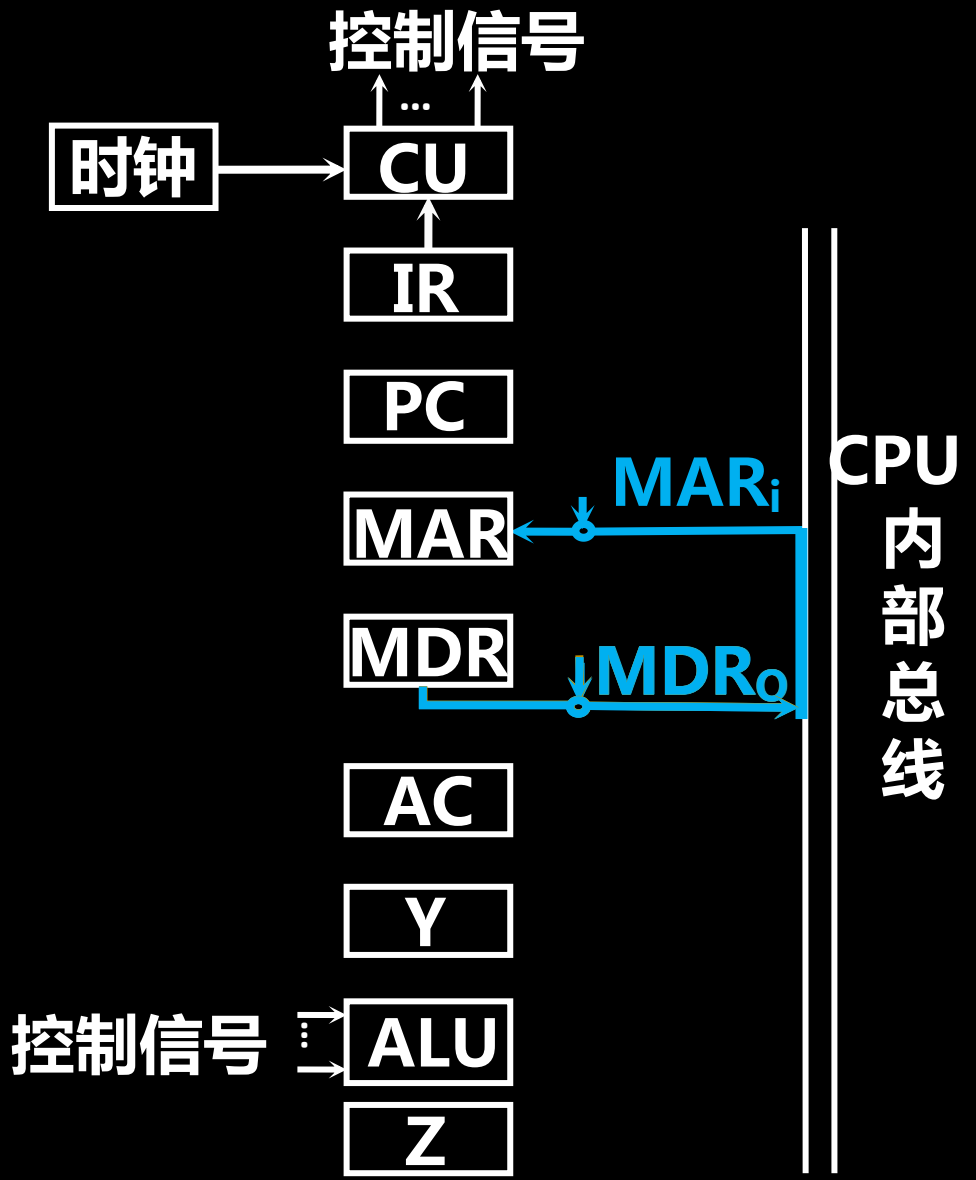
采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶

MDR_O MAR_i



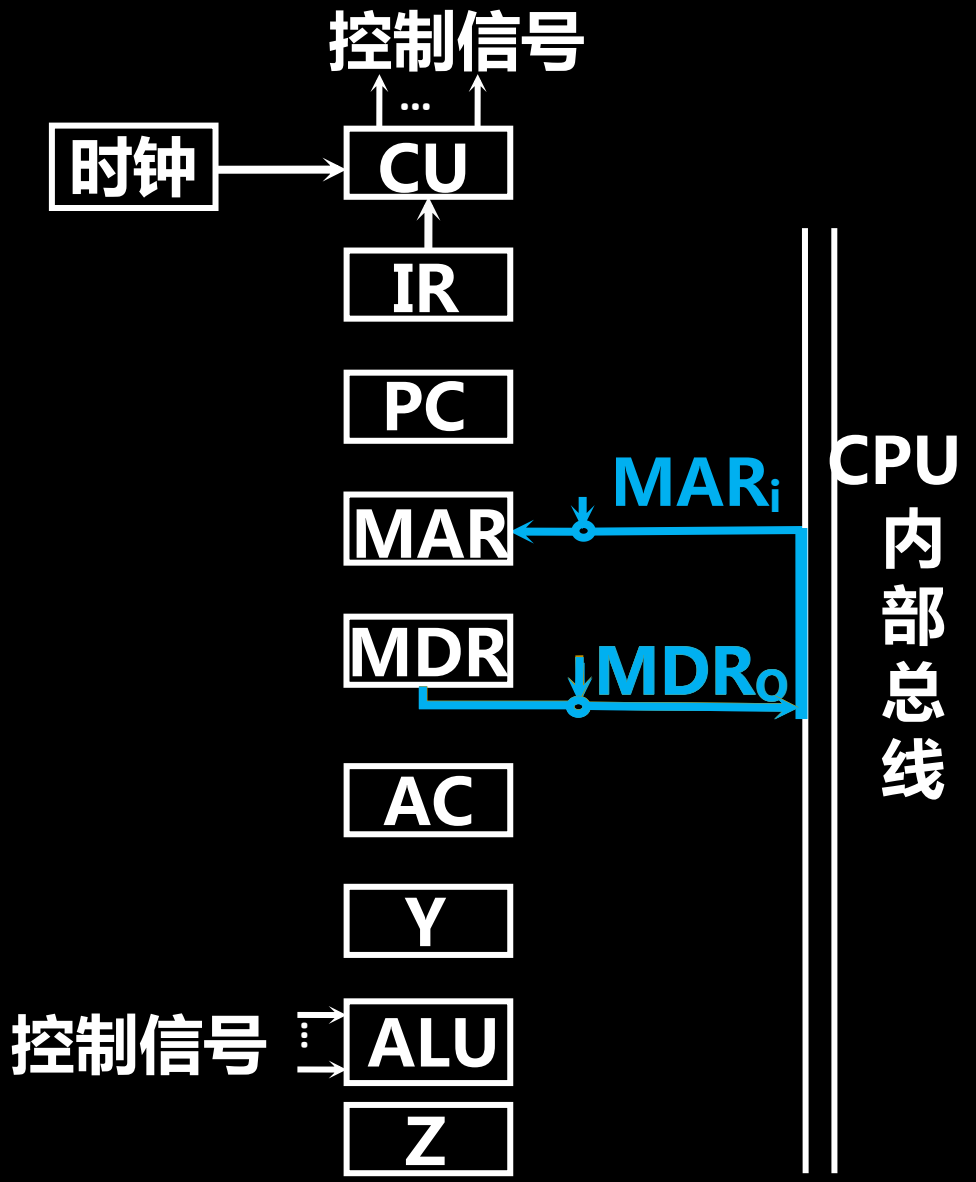
采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR

MDR_O MAR_i



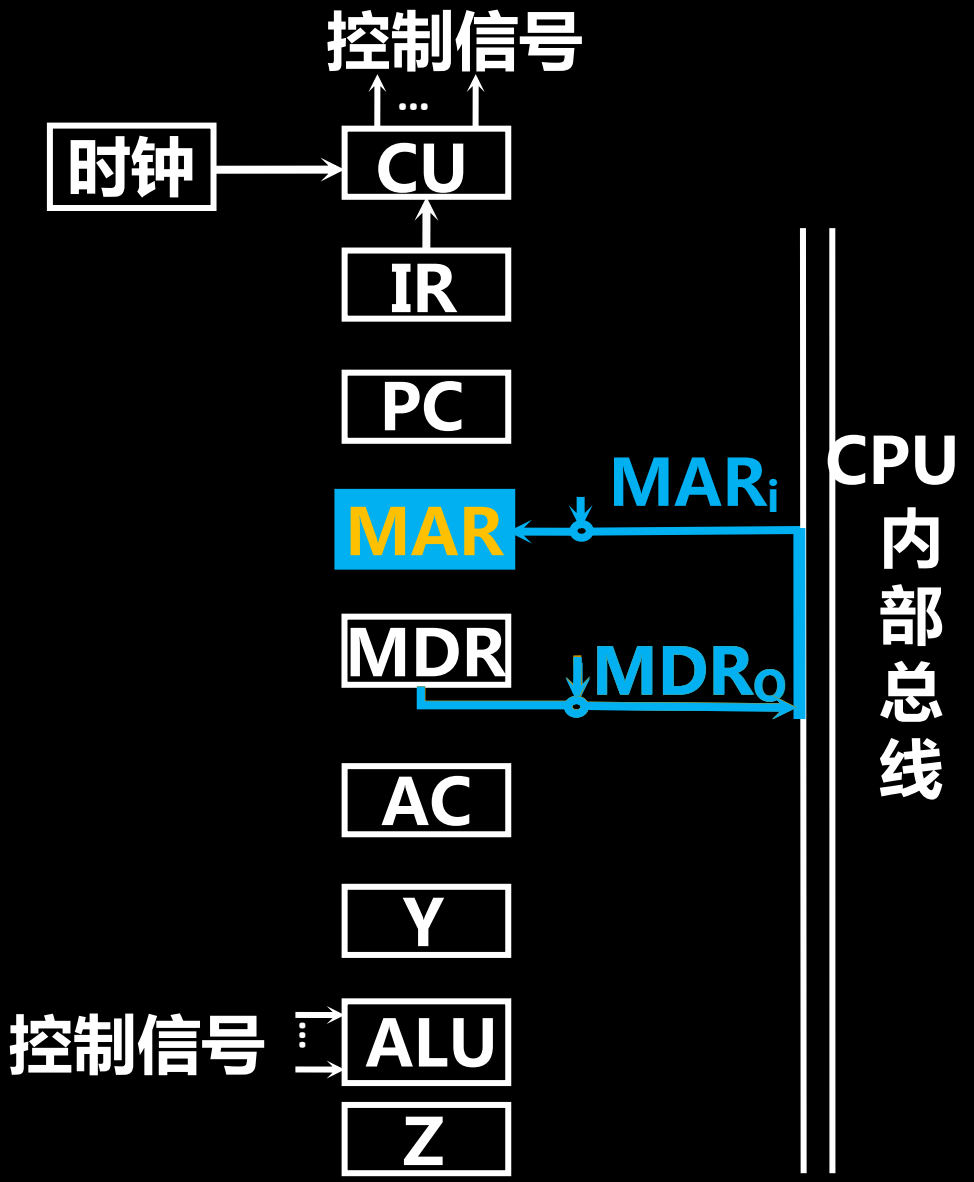
采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR

MDR_o MAR_i



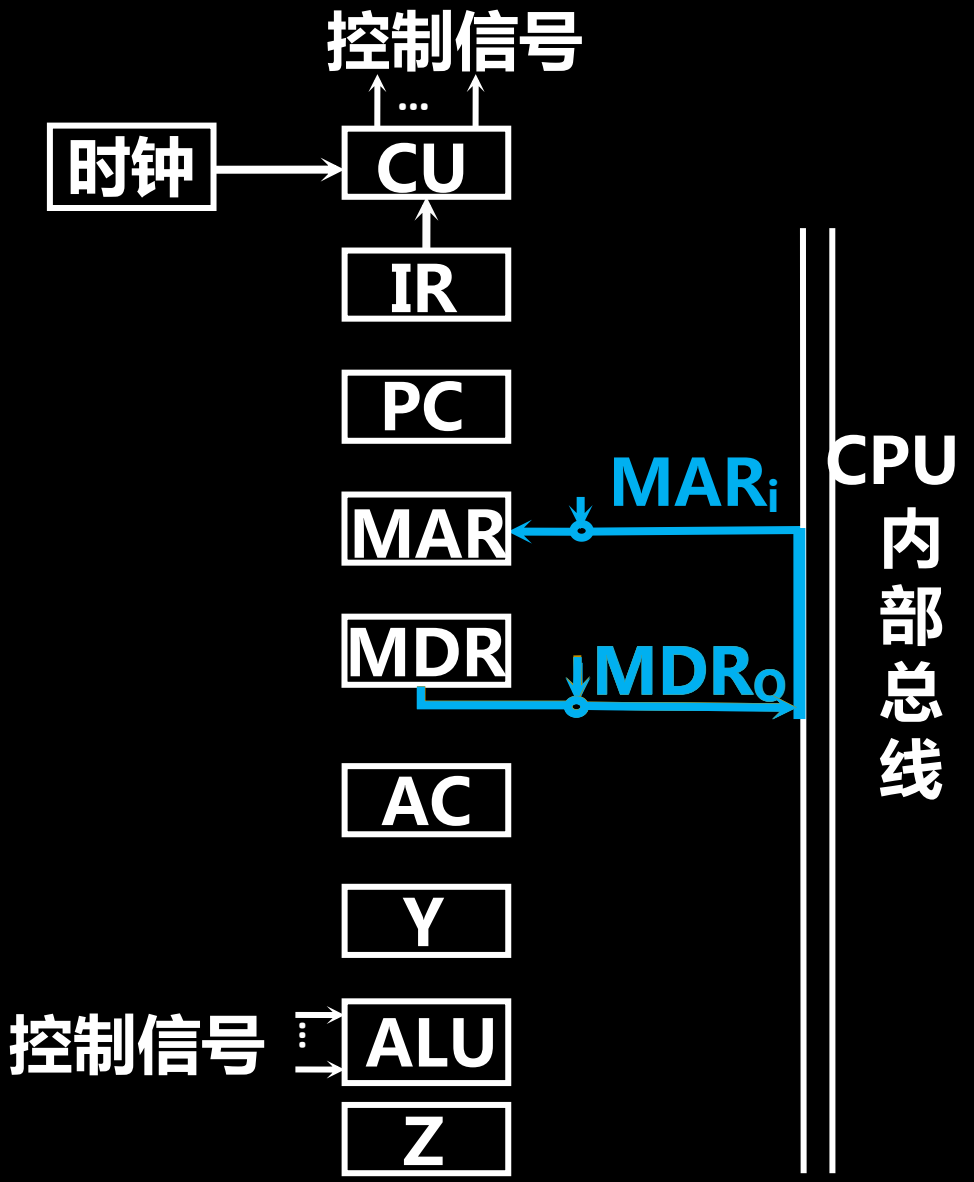
采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR⟶

MDR_o MAR_i



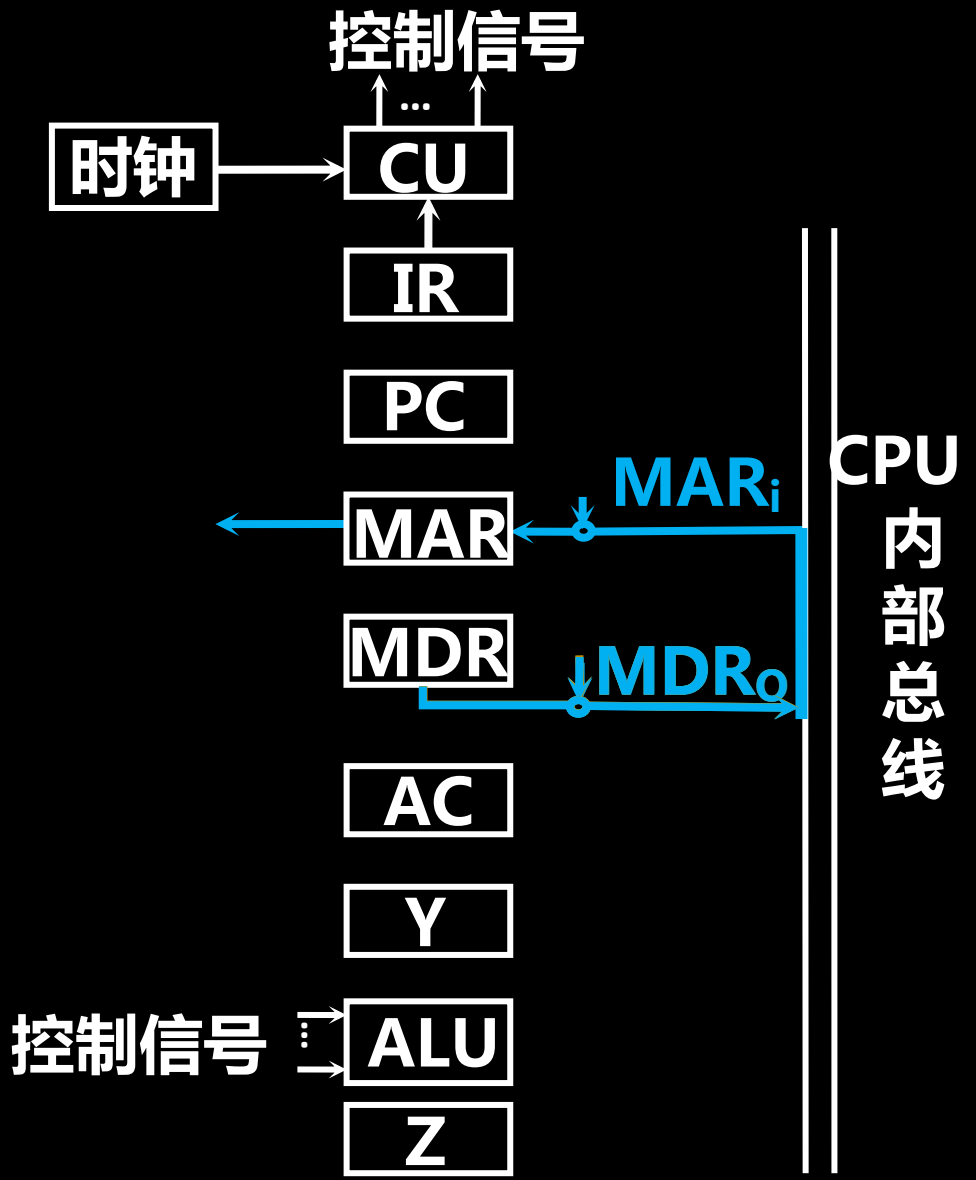
采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR⟶

MDR_o MAR_i



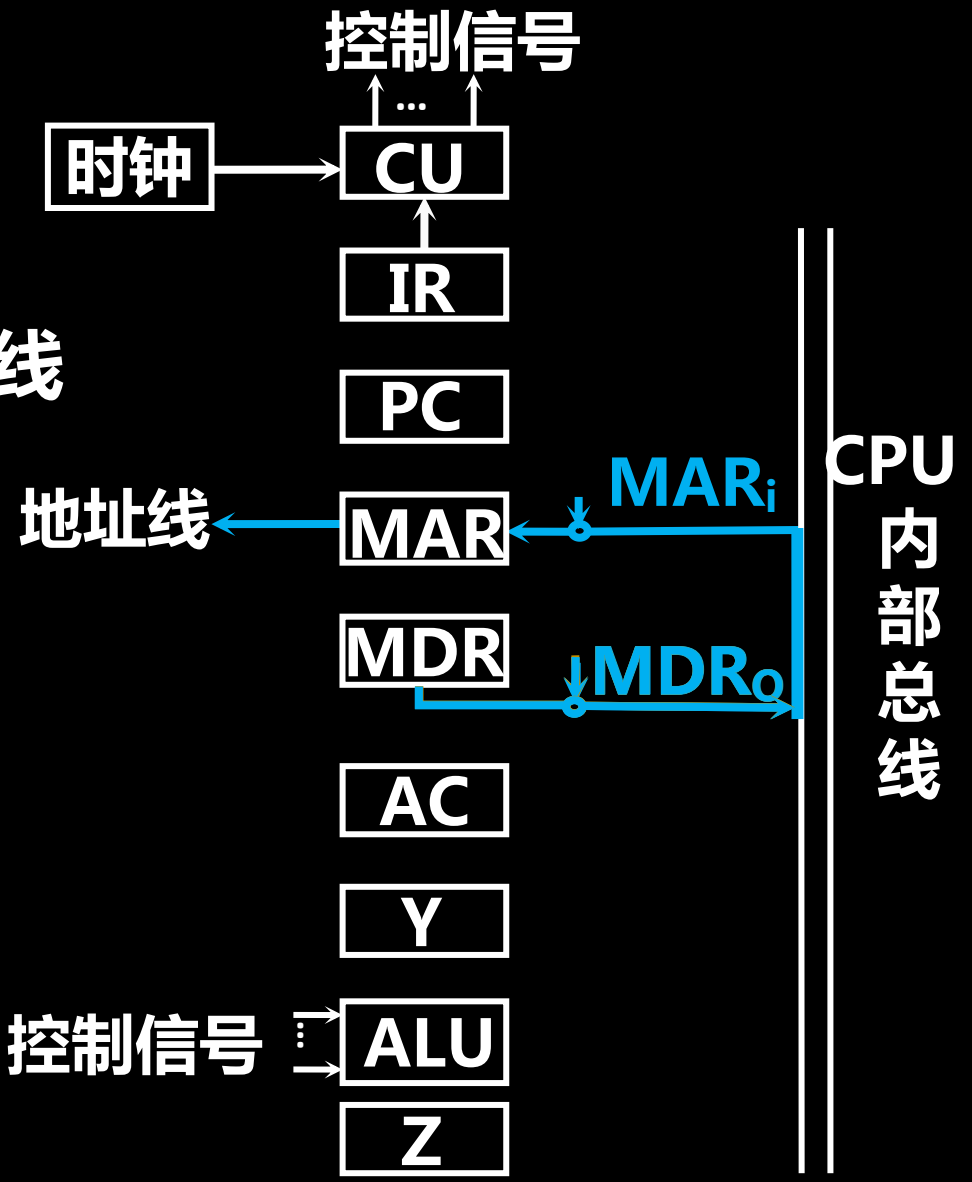
采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

MDR_o MAR_i



采用CPU内部总线方式

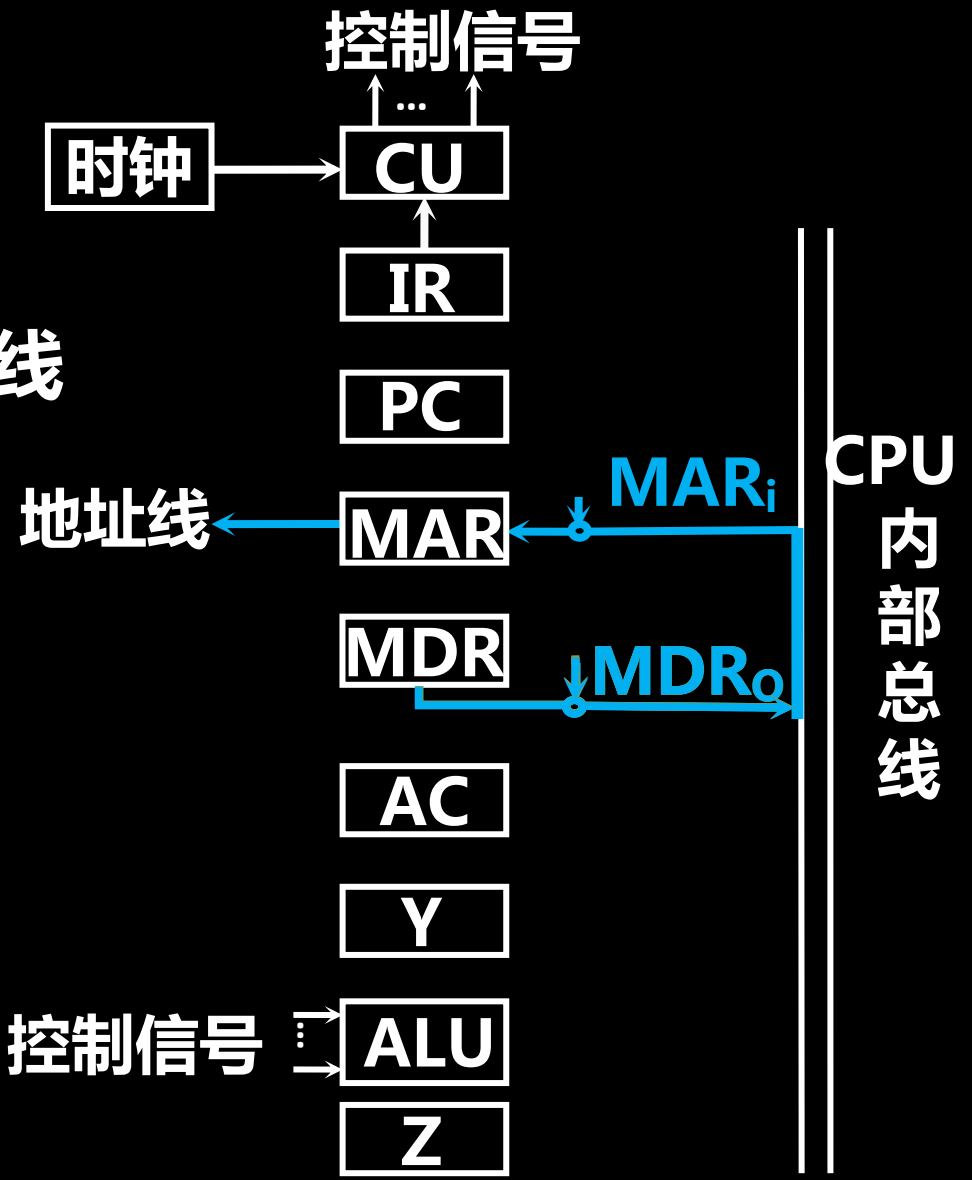
(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

MDR_O MAR_i

1 ⟶ R



采用CPU内部总线方式

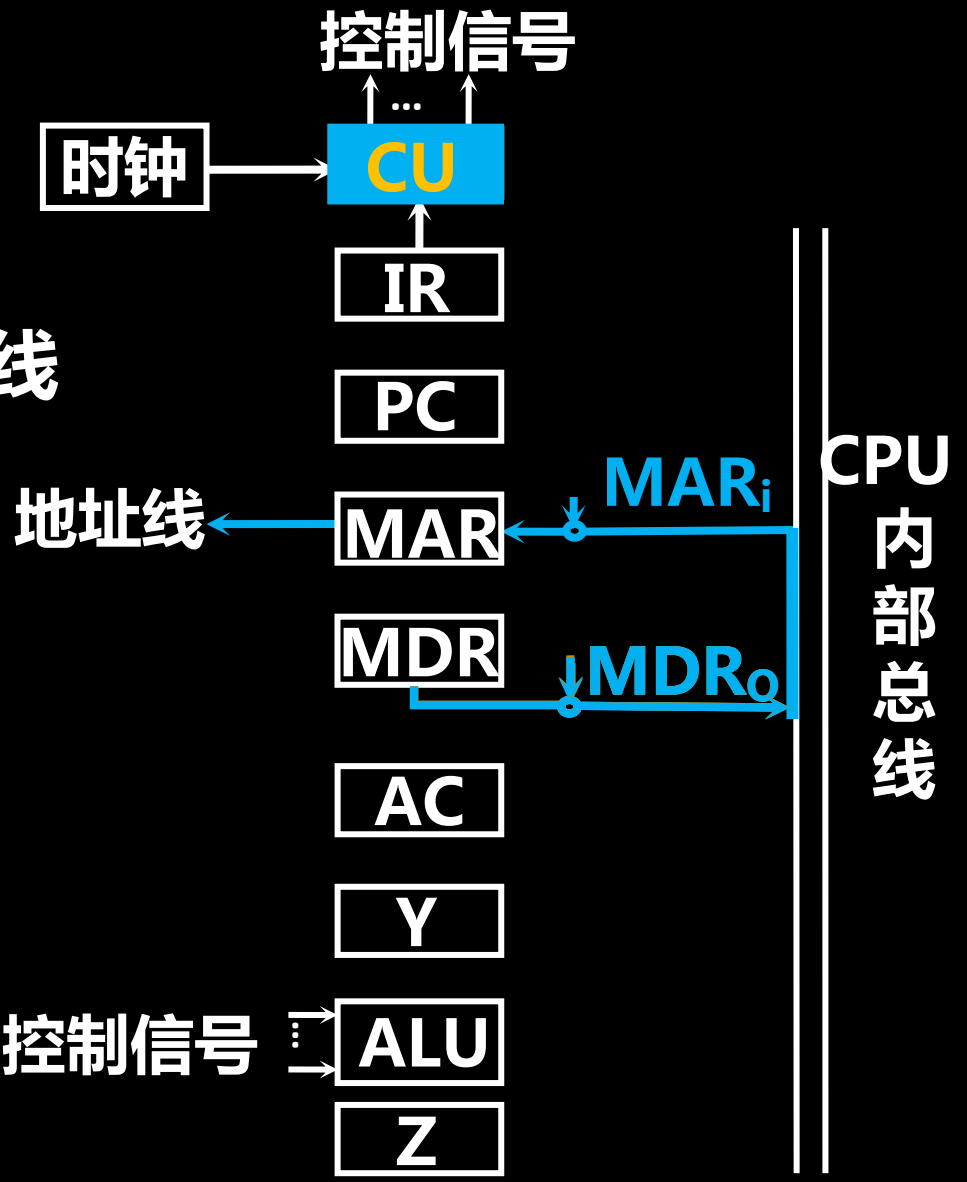
(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

MDR_O MAR_i

1 ⟶ R



采用CPU内部总线方式

(2) ADD @ X 间址周期

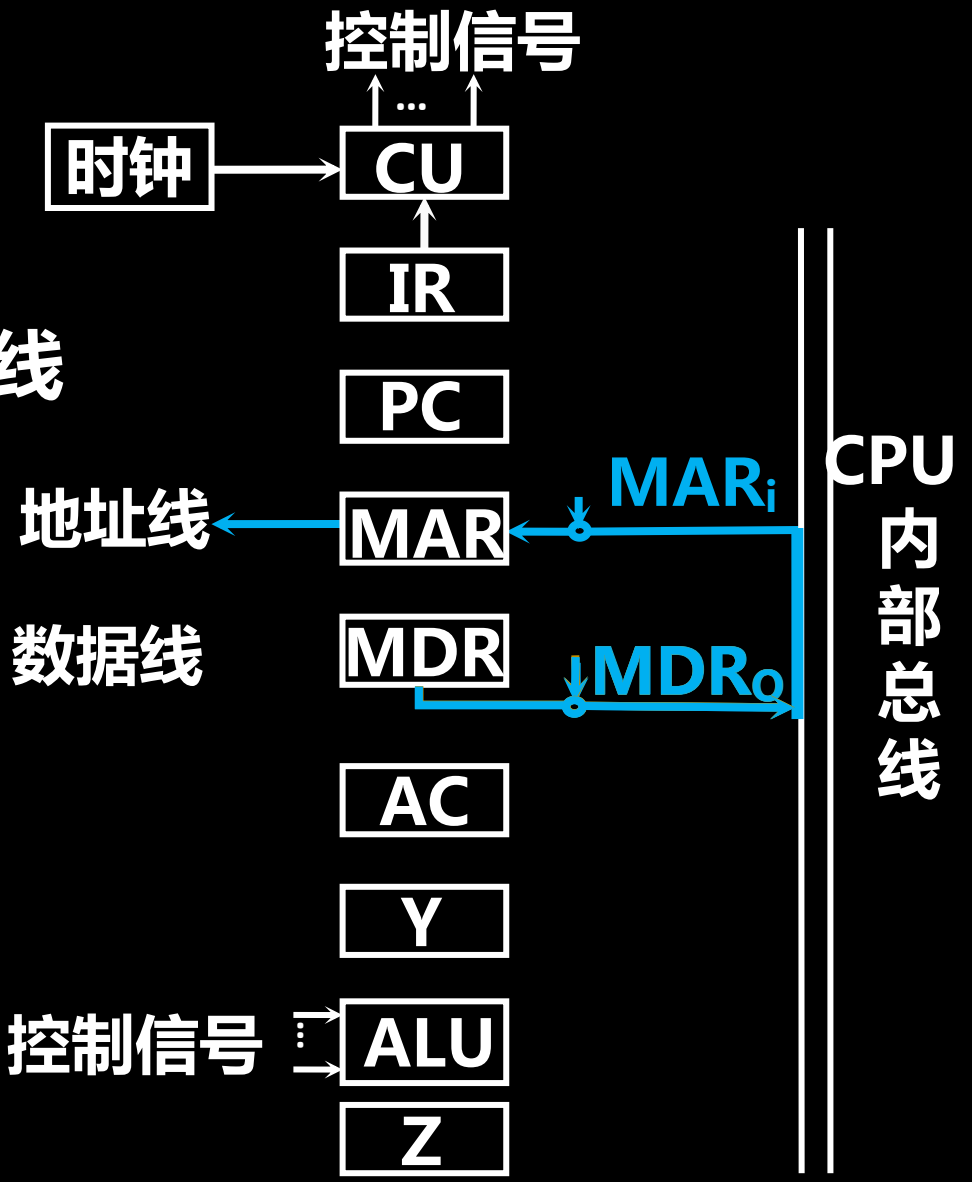
形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

MDR_o MAR_i

1 ⟶ R

数据线



采用CPU内部总线方式

(2) ADD @ X 间址周期

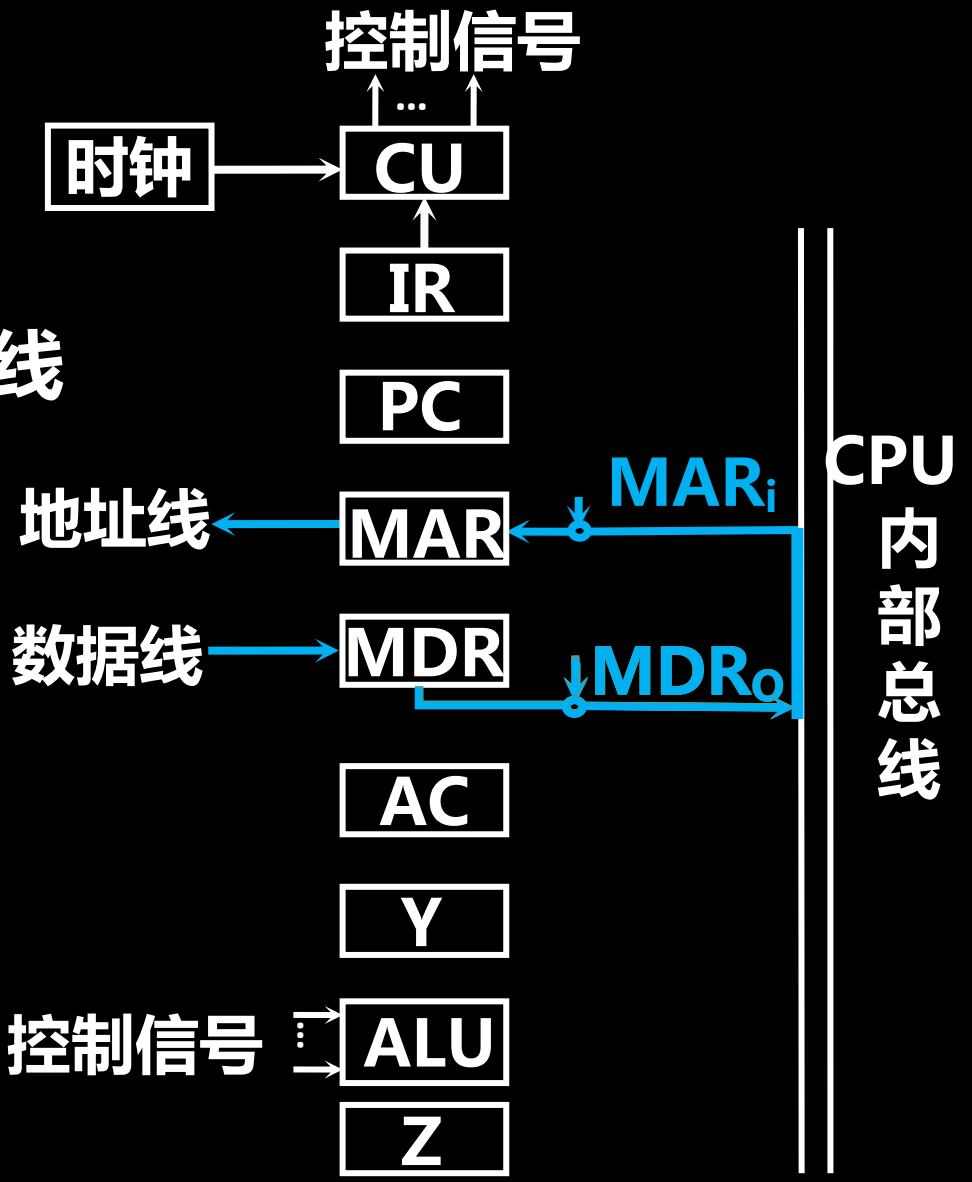
形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

MDR_o MAR_i

1 ⟶ R

数据线 ⟶



采用CPU内部总线方式

(2) ADD @ X 间址周期

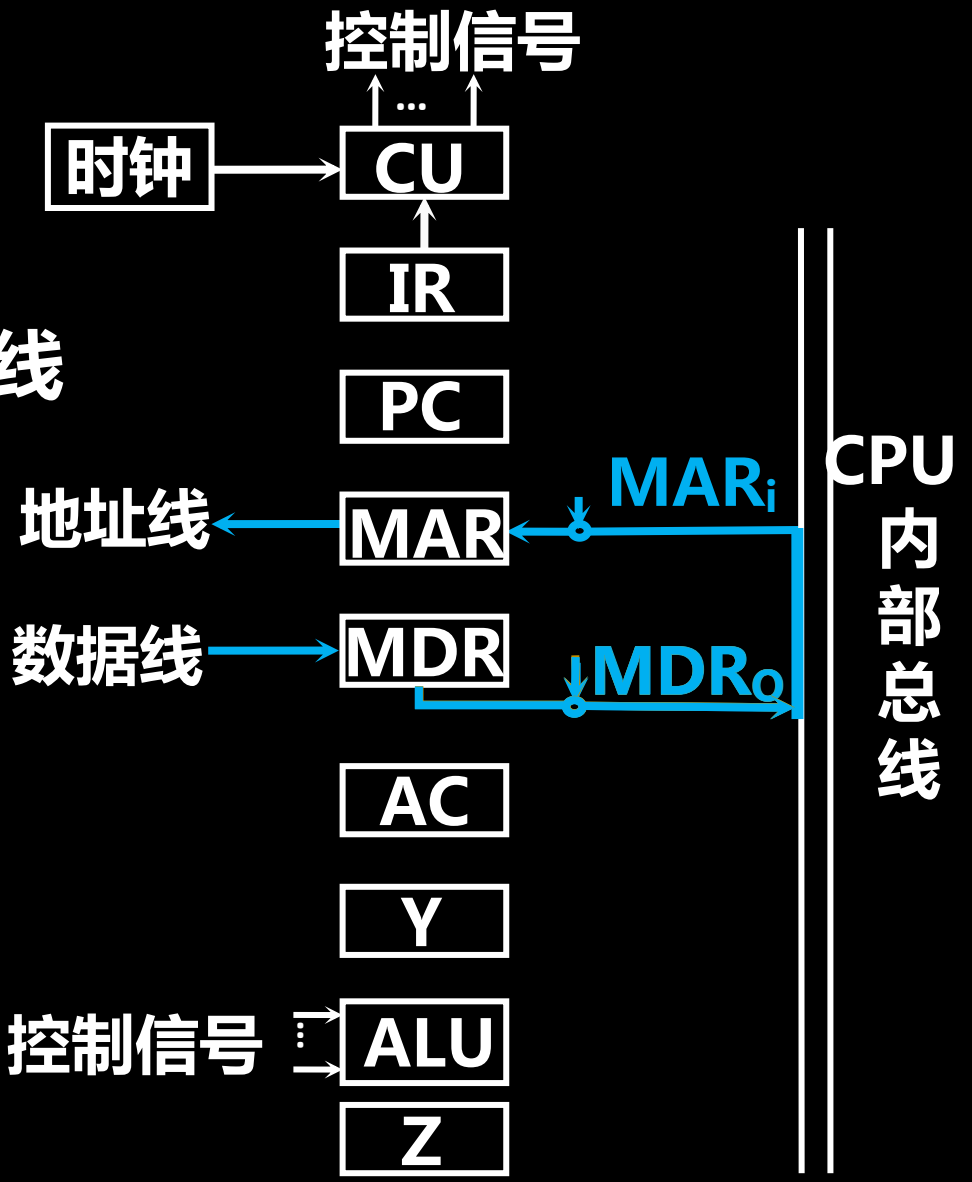
形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

MDR_o MAR_i

1 ⟶ R

数据线 ⟶ MDR



采用CPU内部总线方式

(2) ADD @ X 间址周期

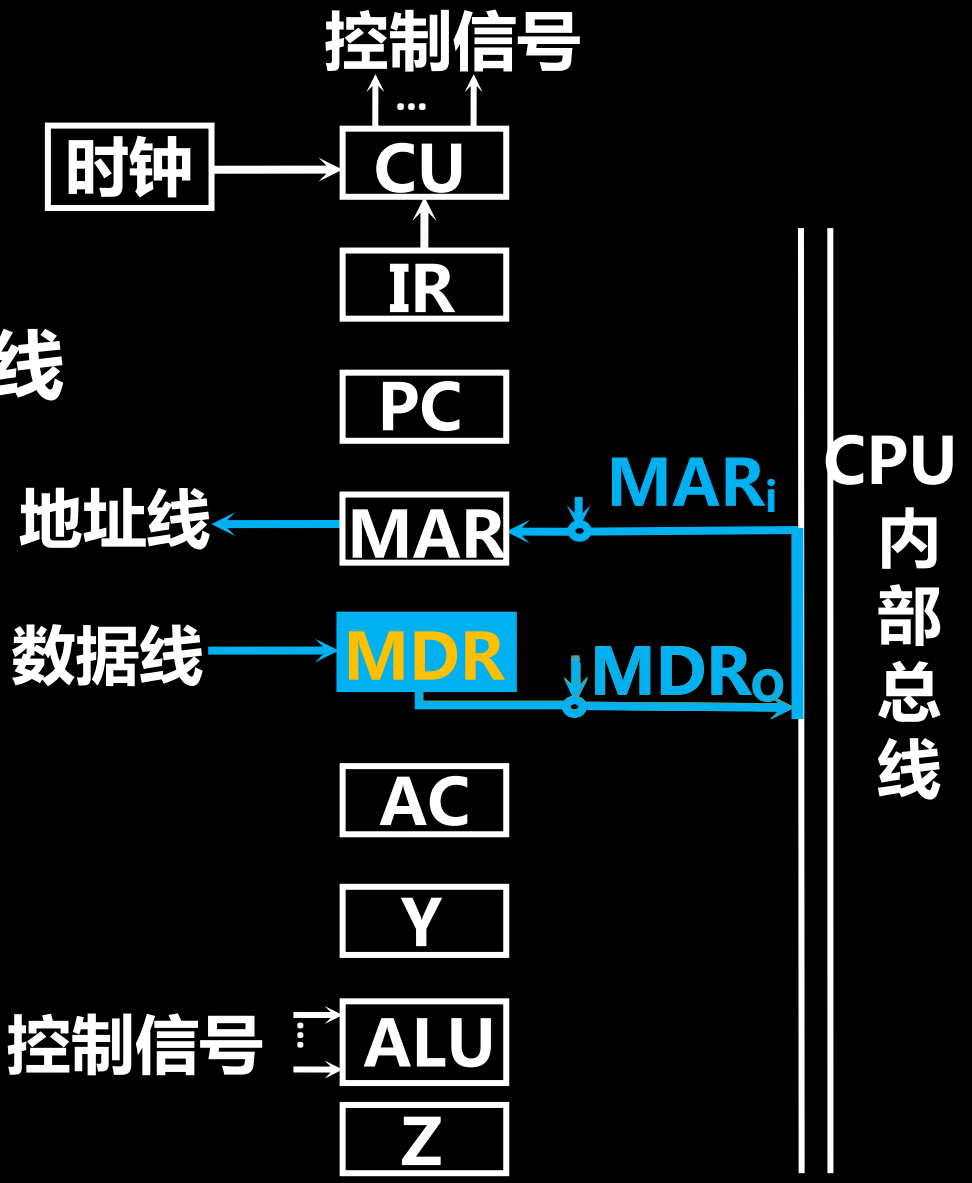
形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

MDR_o MAR_i

1 ⟶ R

数据线 ⟶ MDR



采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

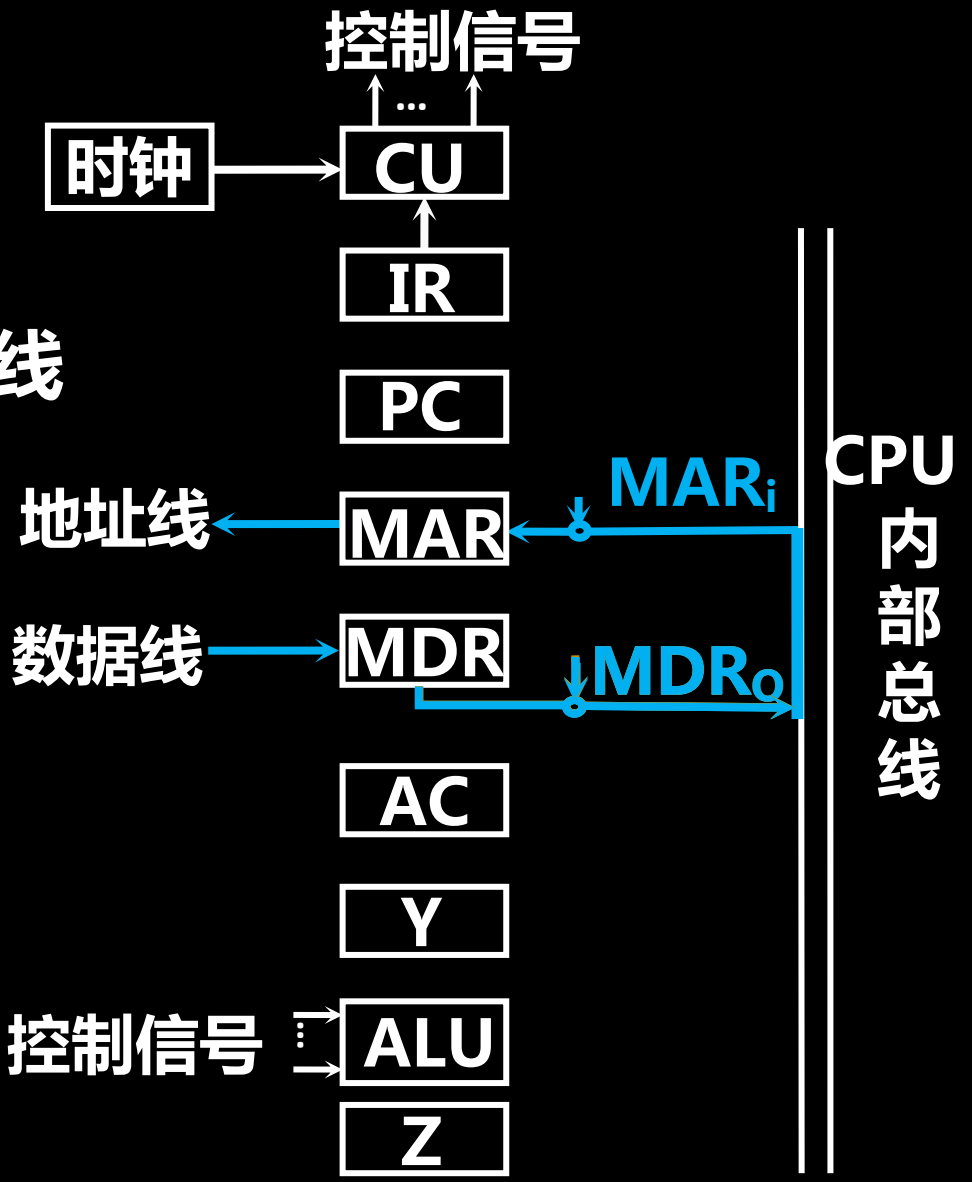
MDR ⟶ MAR ⟶ 地址线

MDR_o MAR_i

1 ⟶ R

数据线 ⟶ MDR

MDR



采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

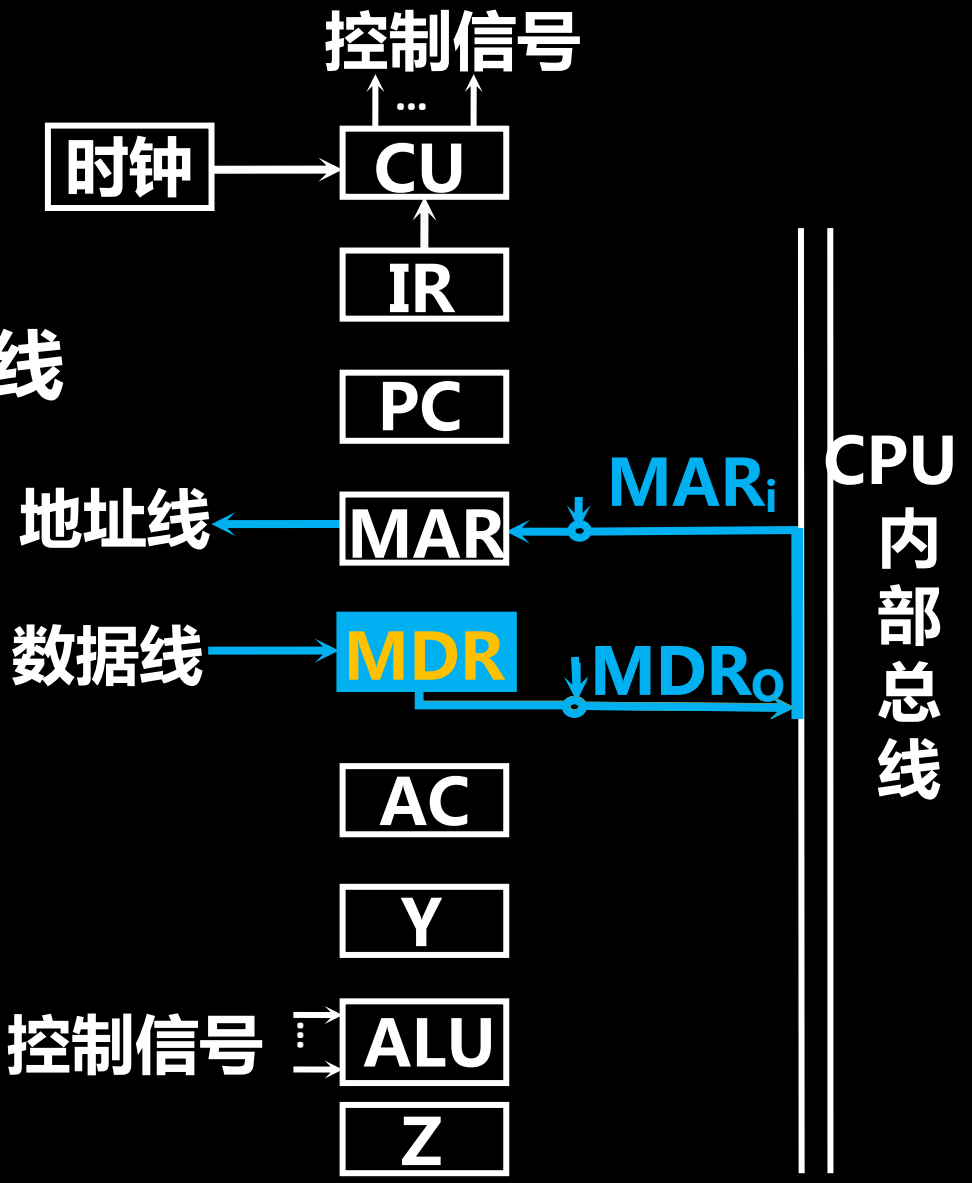
MDR ⟶ MAR ⟶ 地址线

MDR_o MAR_i

1 ⟶ R

数据线 ⟶ MDR

MDR



采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

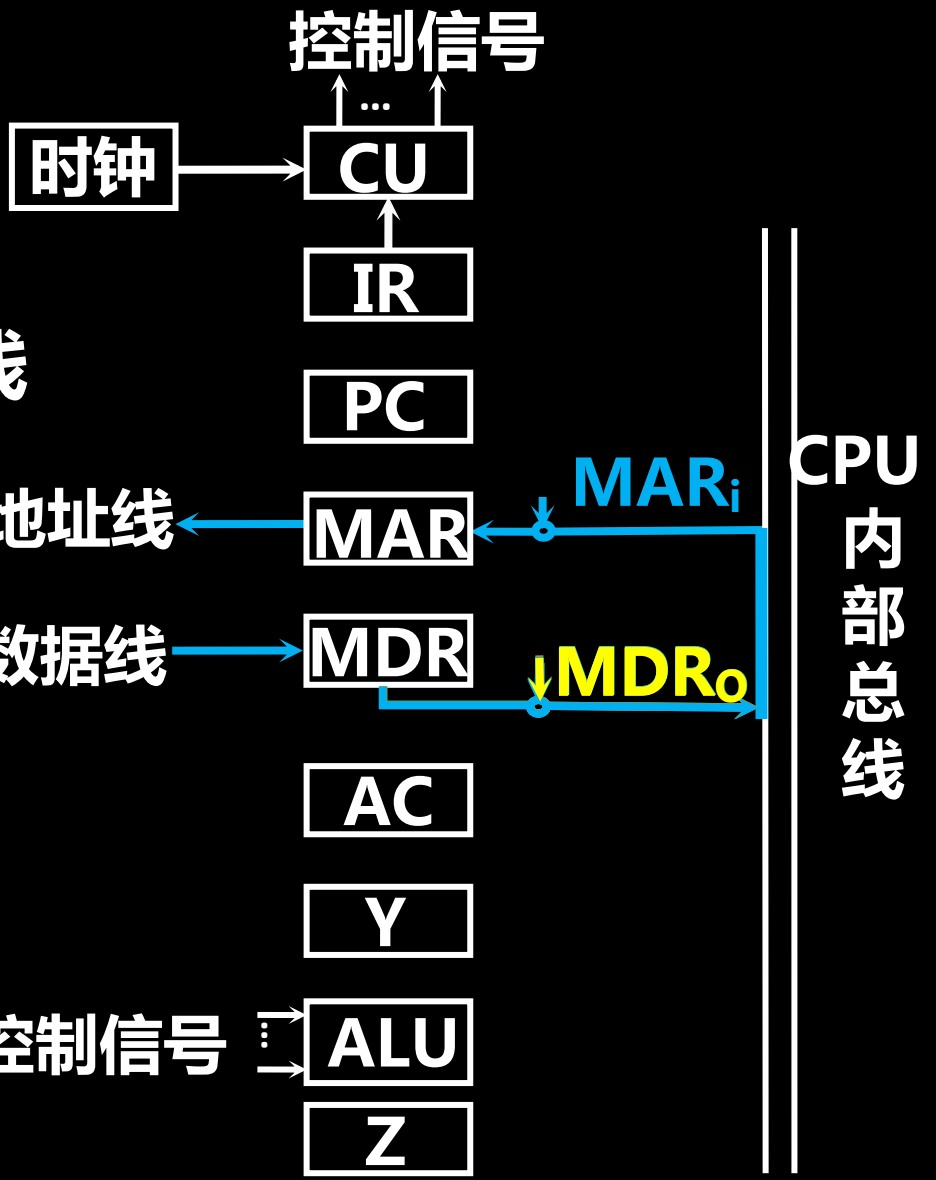
MDR_o MAR_i

1 ⟶ R

数据线 ⟶ MDR

MDR

MDR_o



采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

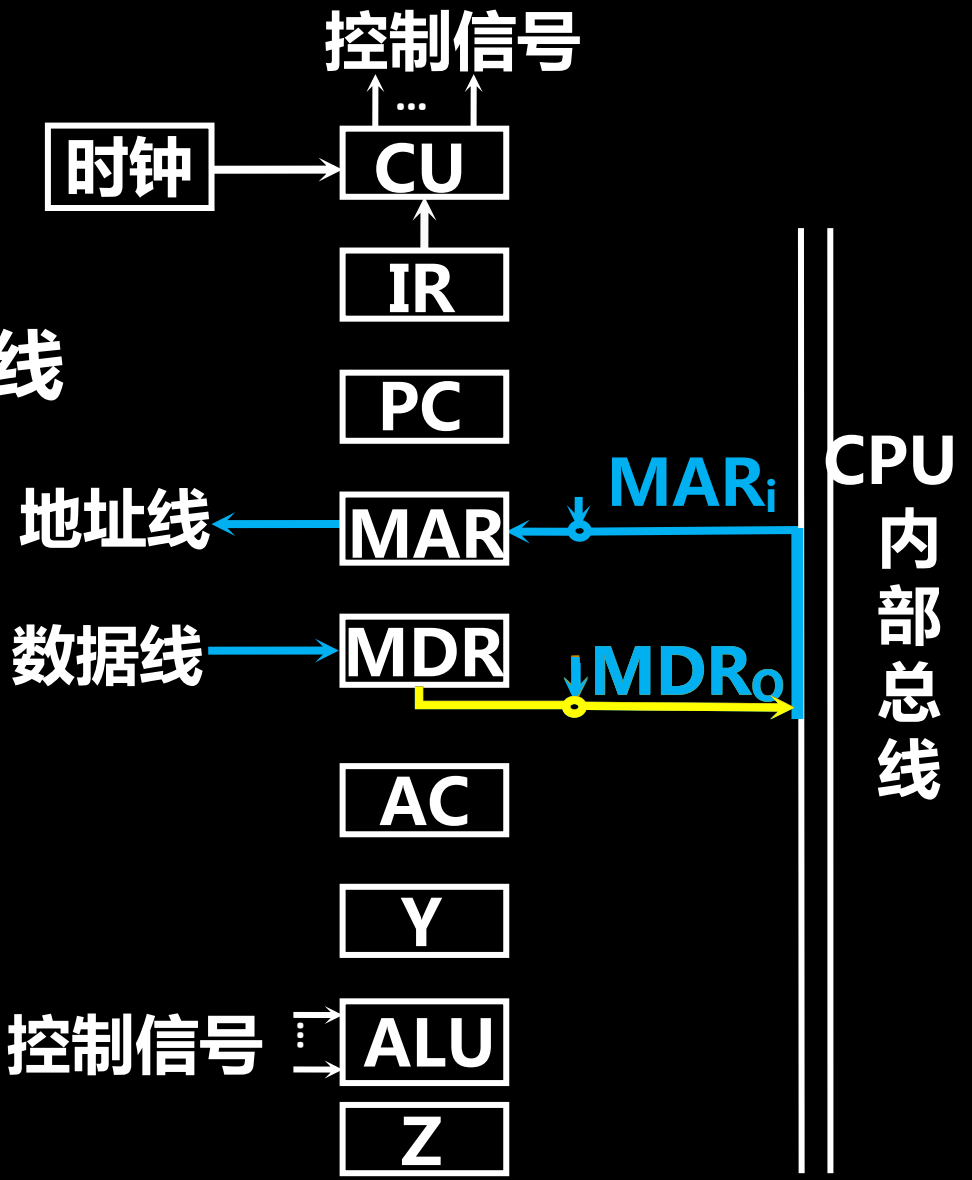
MDR_o MAR_i

1 ⟶ R

数据线 ⟶ MDR

MDR ⟶

MDR_o



采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

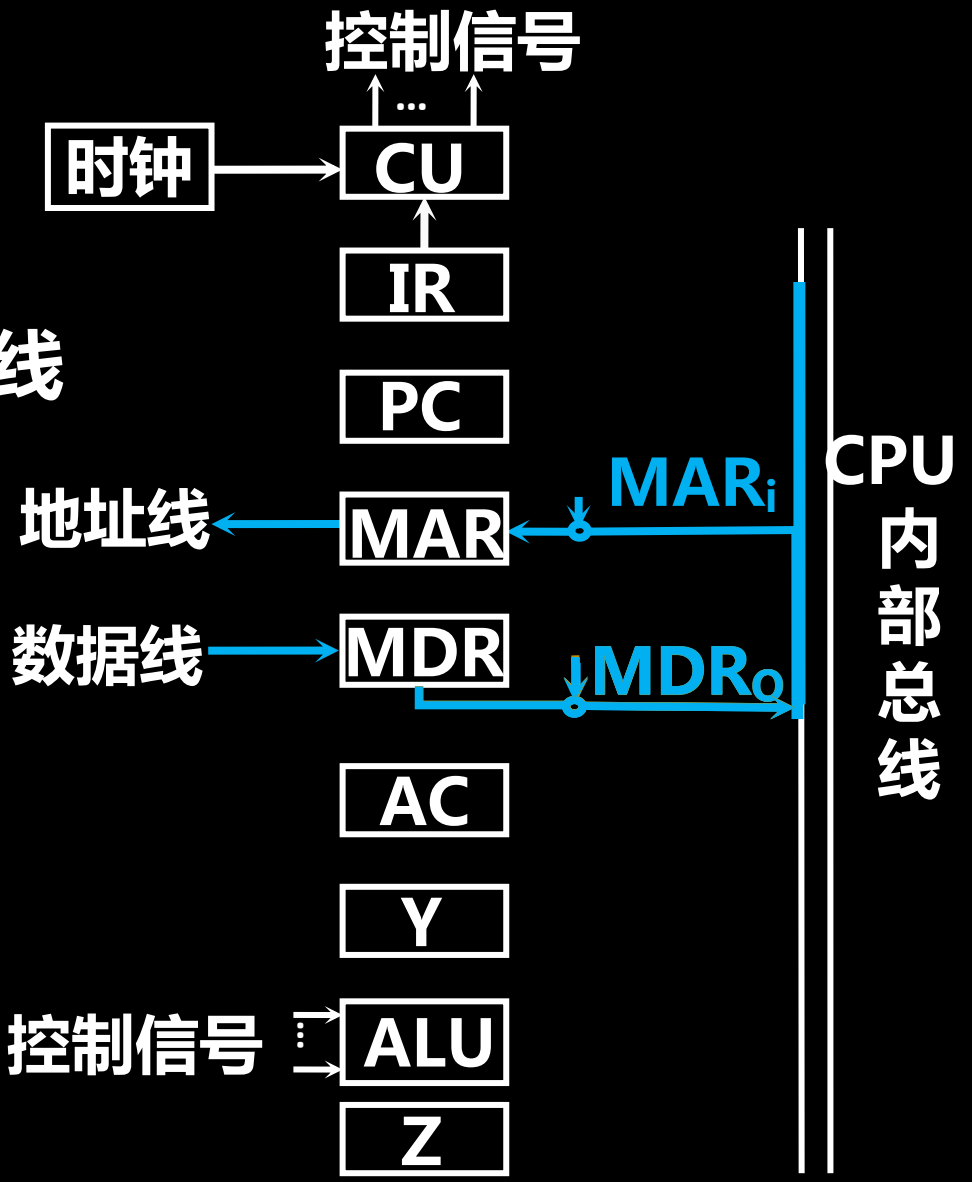
MDR_o MAR_i

1 ⟶ R

数据线 ⟶ MDR

MDR ⟶

MDR_o



采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

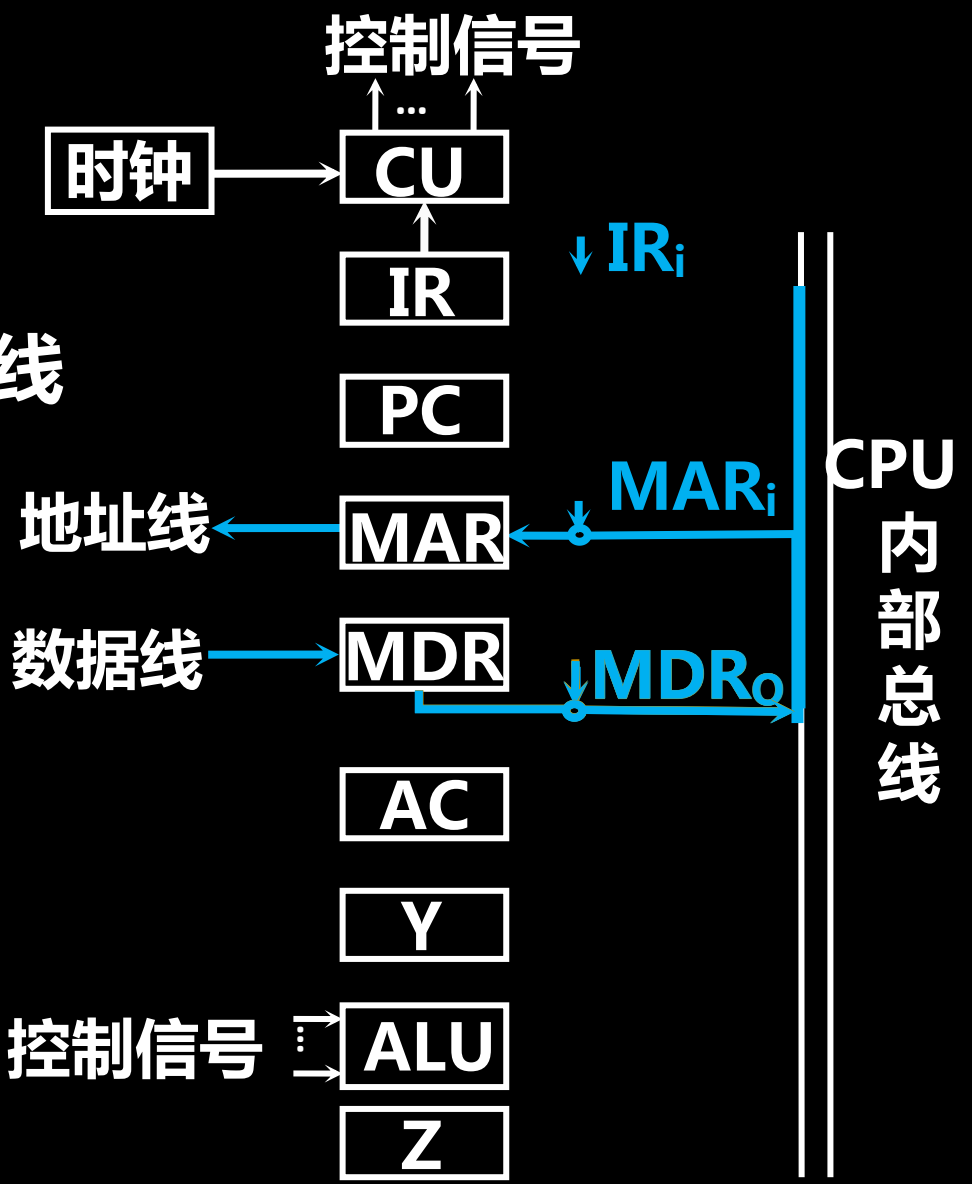
MDR_o MAR_i

1 ⟶ R

数据线 ⟶ MDR

MDR ⟶

MDR_o IR_i



采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

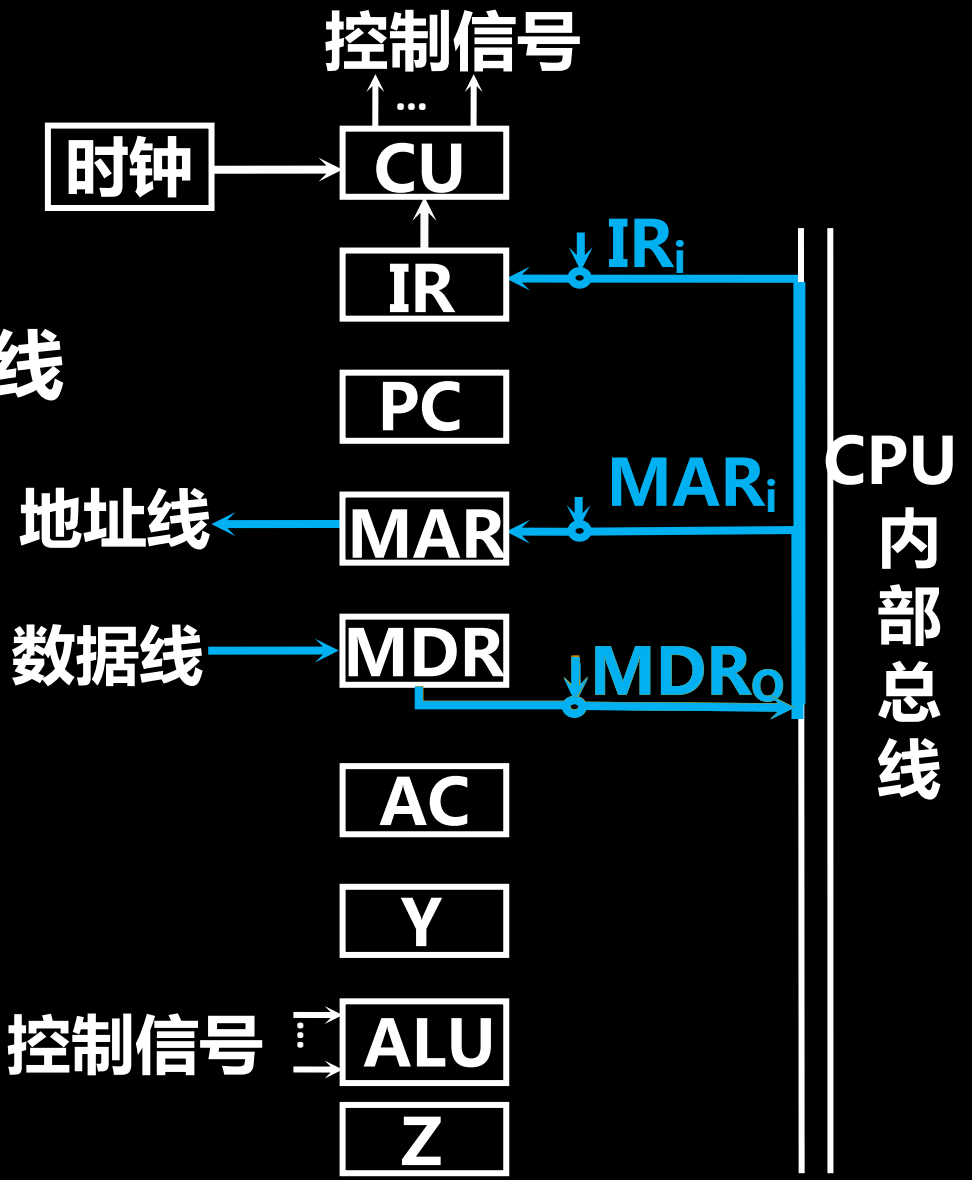
MDR_o MAR_i

1 ⟶ R

数据线 ⟶ MDR

MDR ⟶

MDR_o IR_i



采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

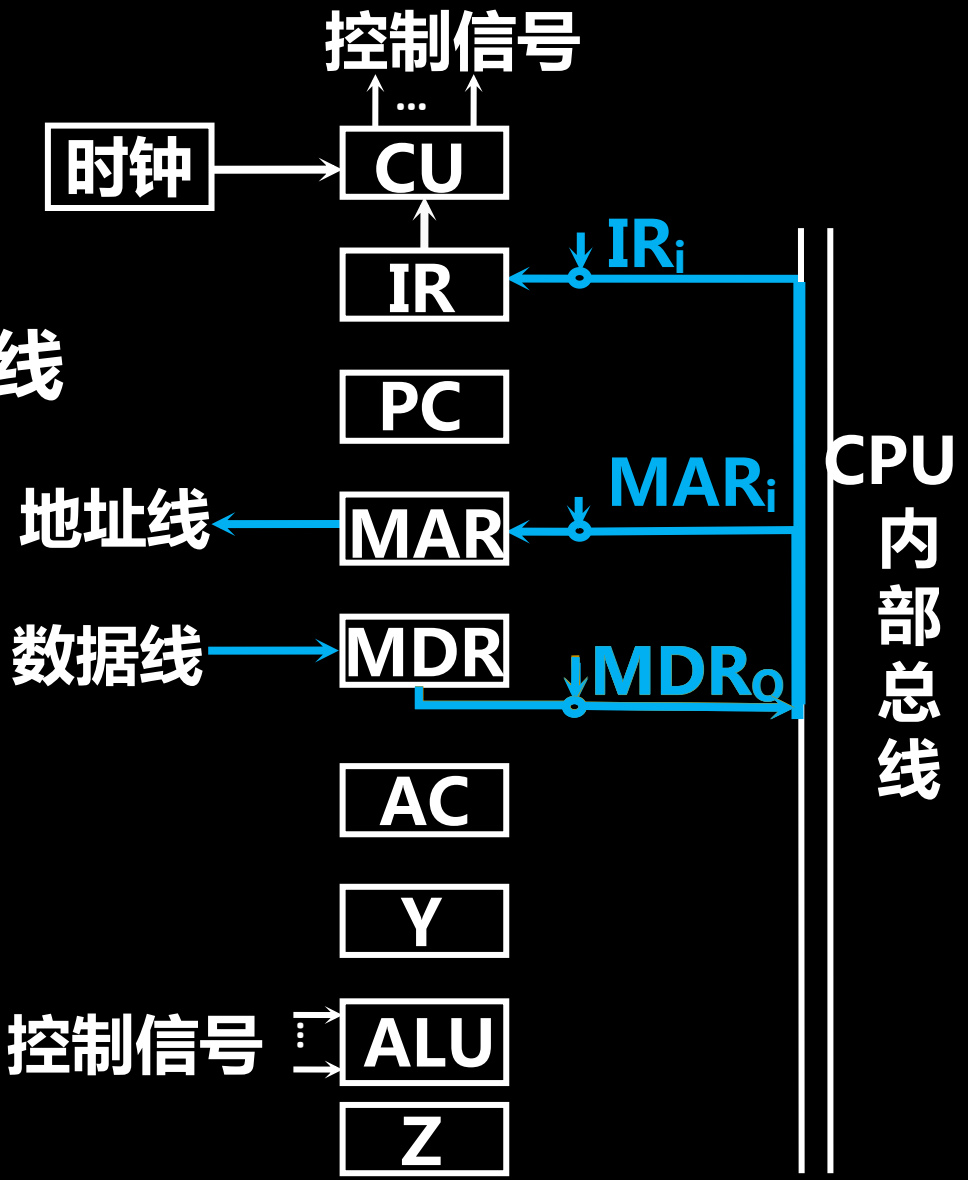
MDR_o MAR_i

1 ⟶ R

数据线 ⟶ MDR

MDR ⟶ IR

MDR_o IR_i



采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

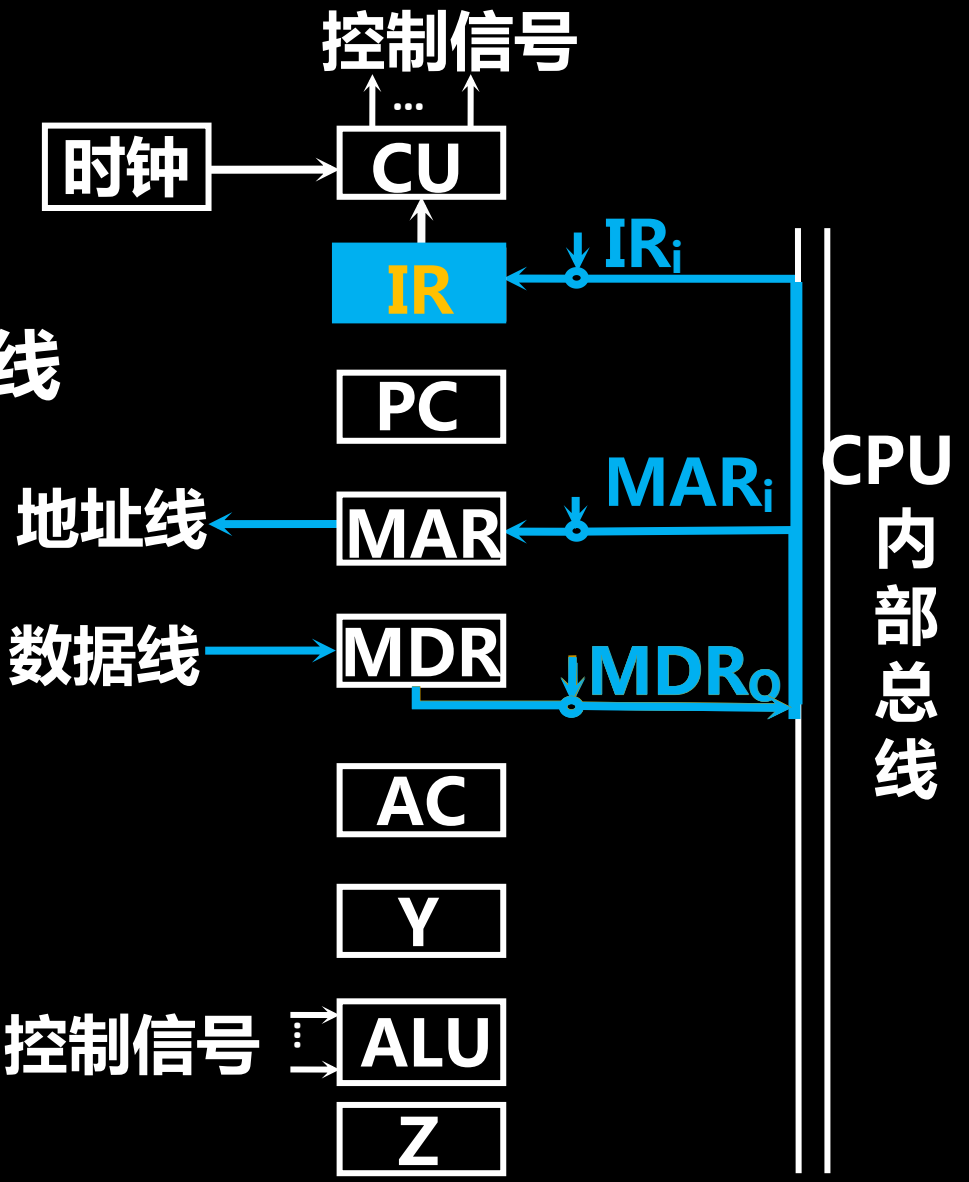
MDR_o MAR_i

1 ⟶ R

数据线 ⟶ MDR

MDR ⟶ IR

MDR_o IR_i



采用CPU内部总线方式

(2) ADD @ X 间址周期

形式地址 ⟶ MAR

MDR ⟶ MAR ⟶ 地址线

MDR_o MAR_i

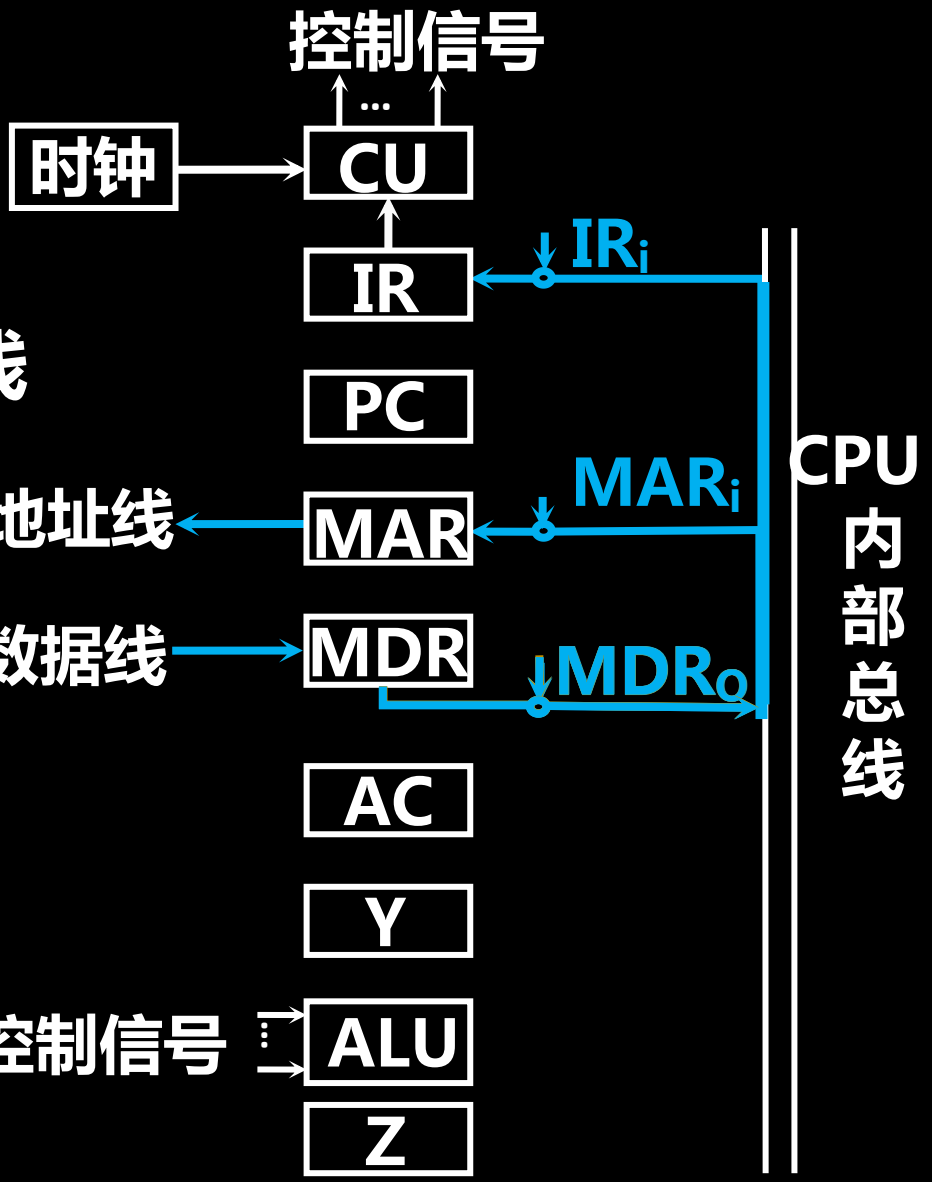
1 ⟶ R

数据线 ⟶ MDR

MDR ⟶ IR

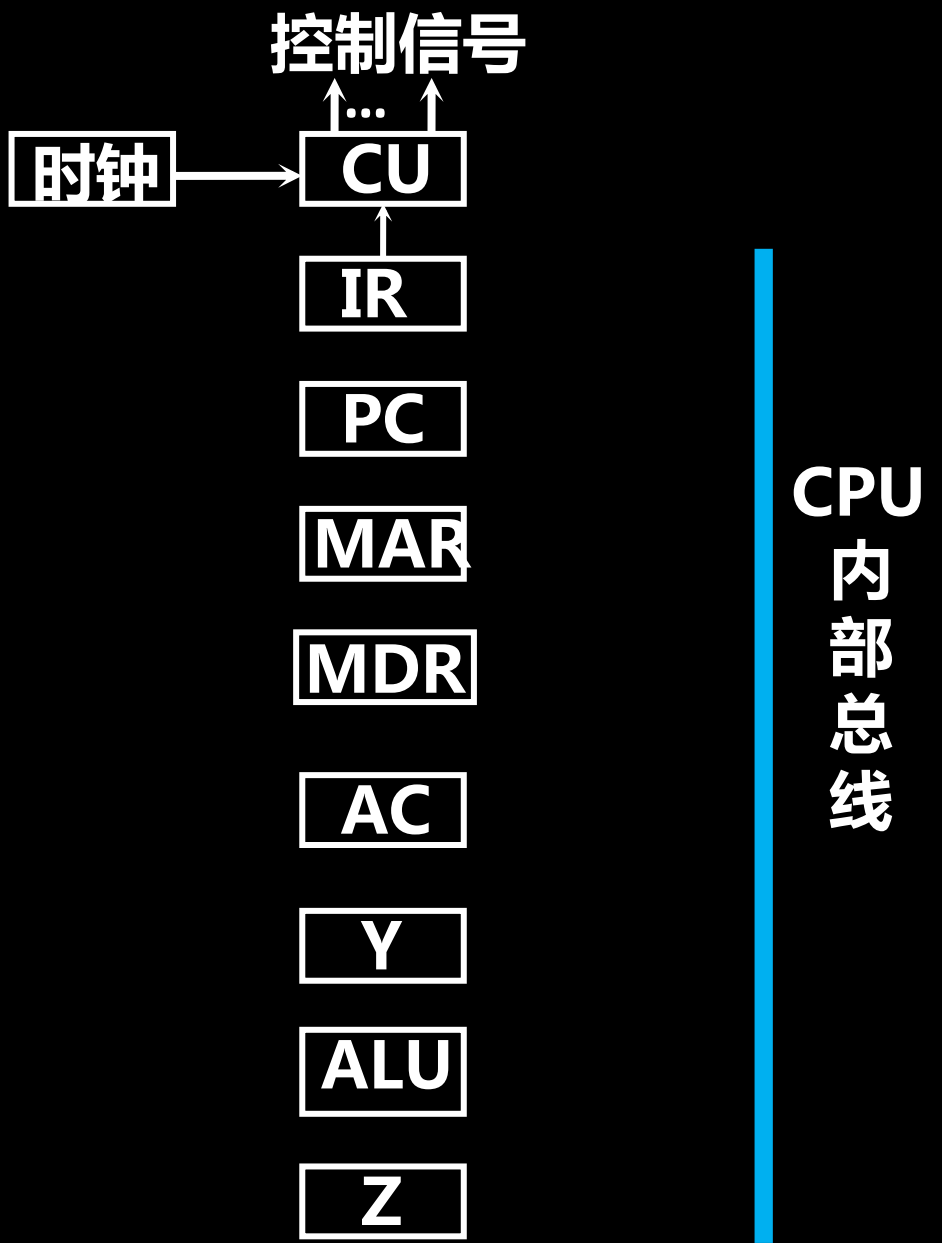
MDR_o IR_i

有效地址 ⟶ Ad（IR）



采用CPU内部总线方式

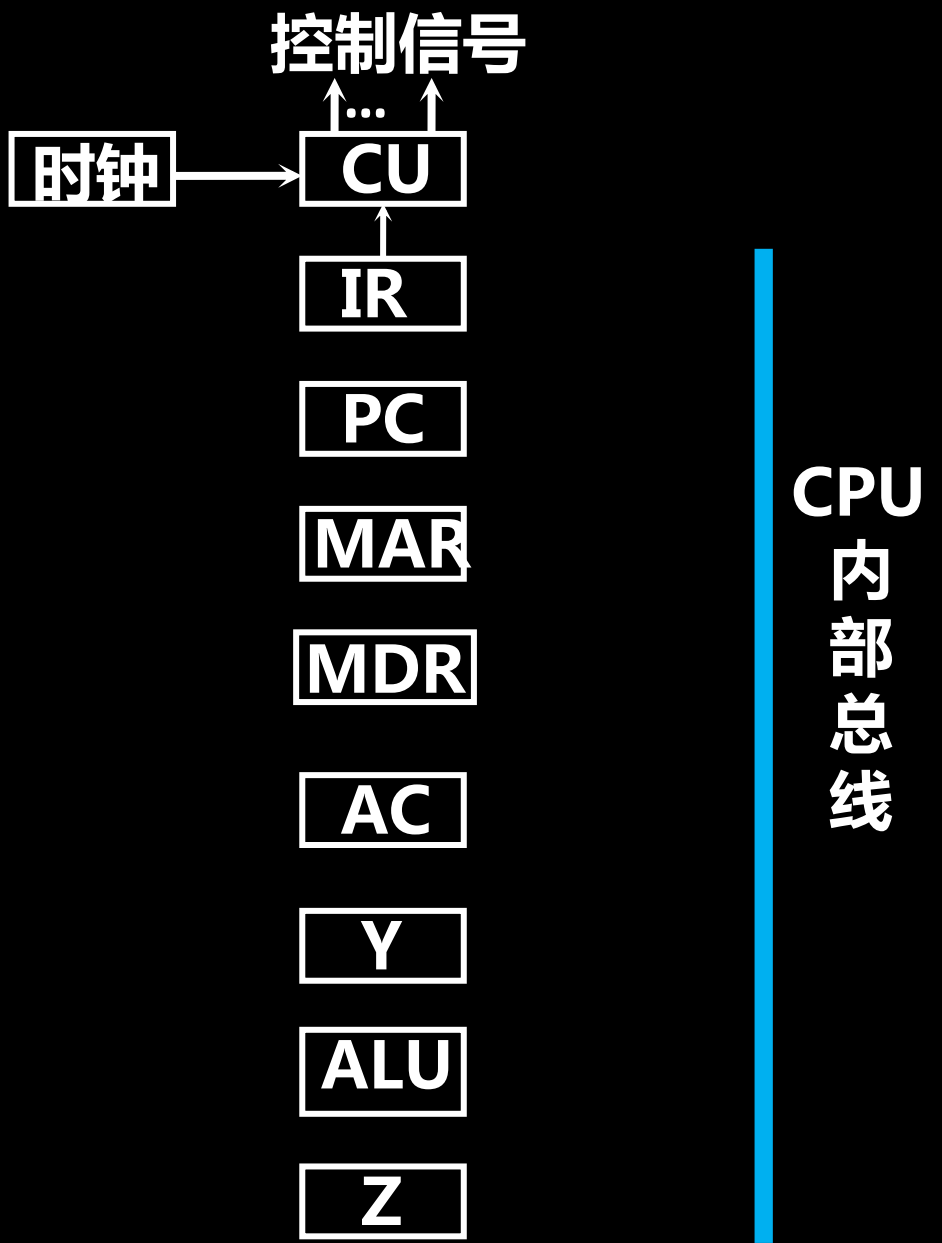
(3) ADD @ X 执行周期



采用CPU内部总线方式

(3) ADD @ X 执行周期

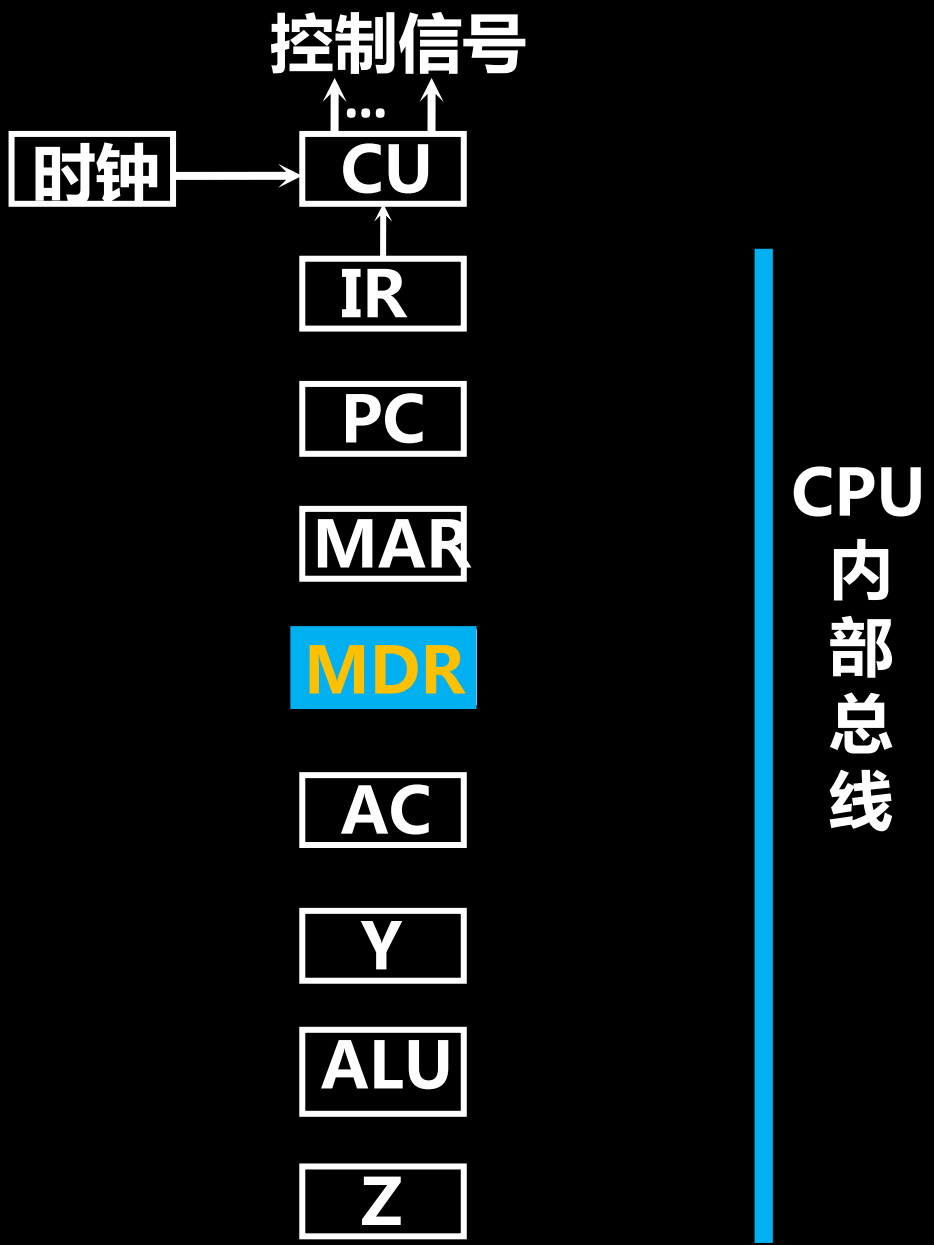
MDR



采用CPU内部总线方式

(3) ADD @ X 执行周期

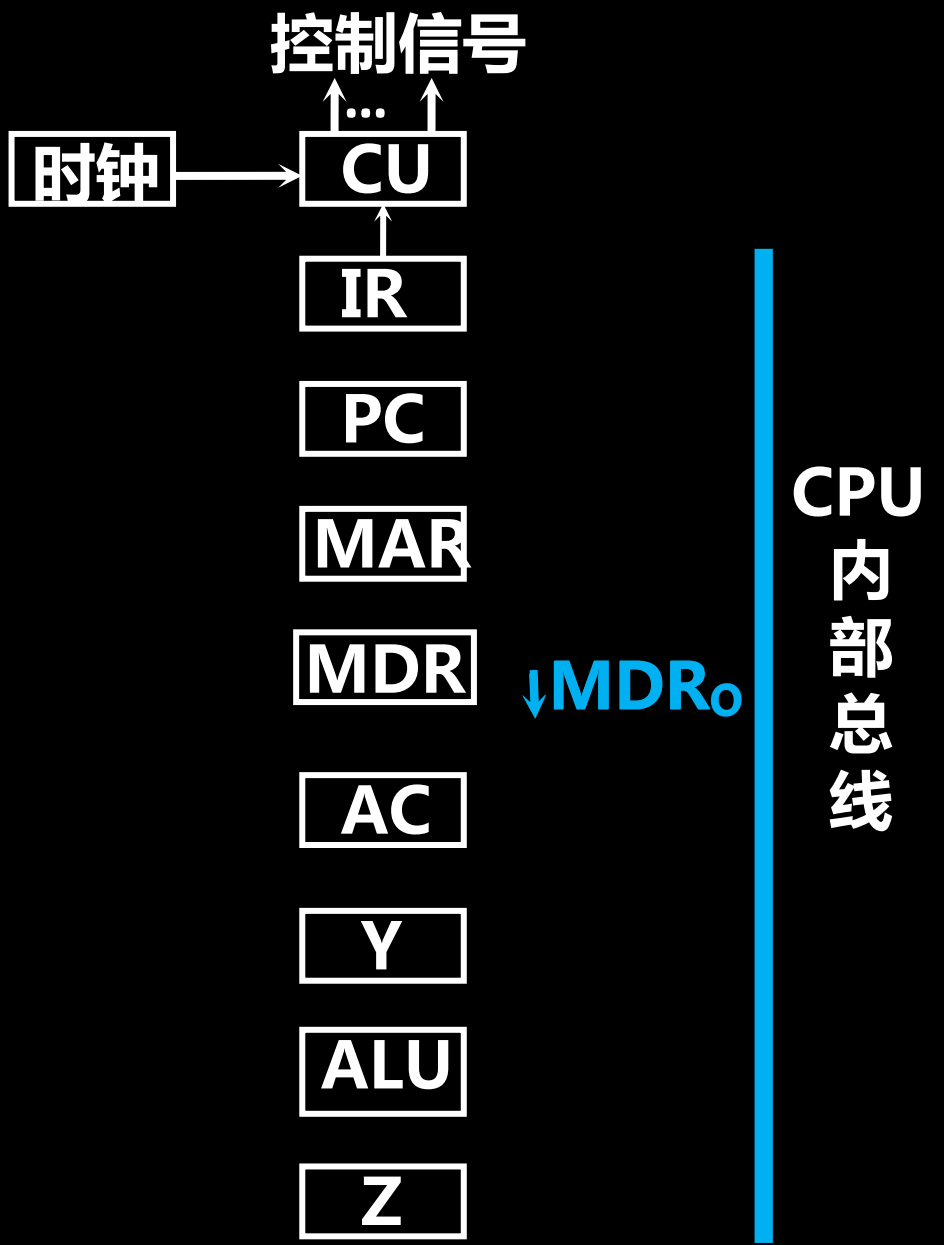
MDR



采用CPU内部总线方式

(3) ADD @ X 执行周期

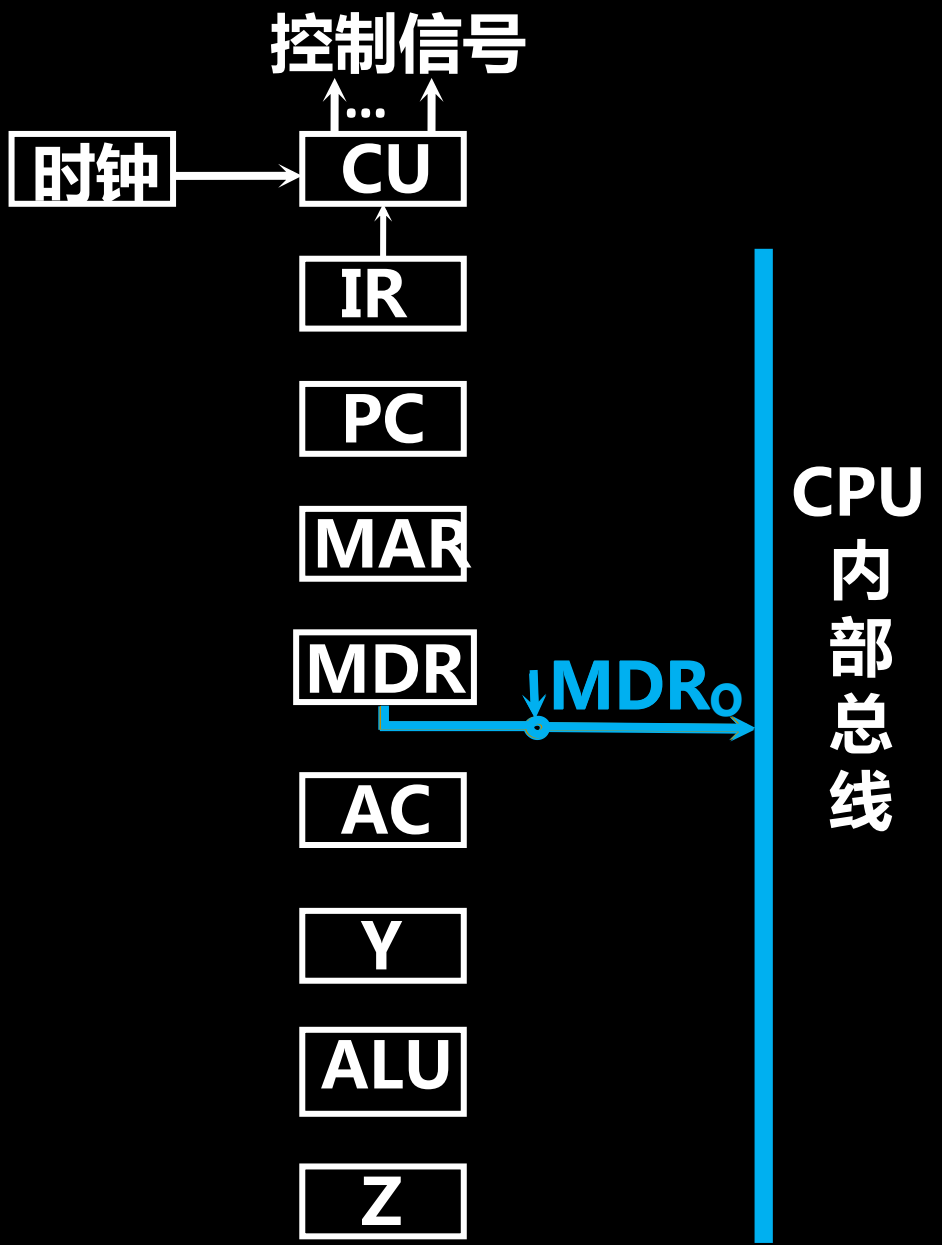
MDR
MDR_o



采用CPU内部总线方式

(3) ADD @ X 执行周期

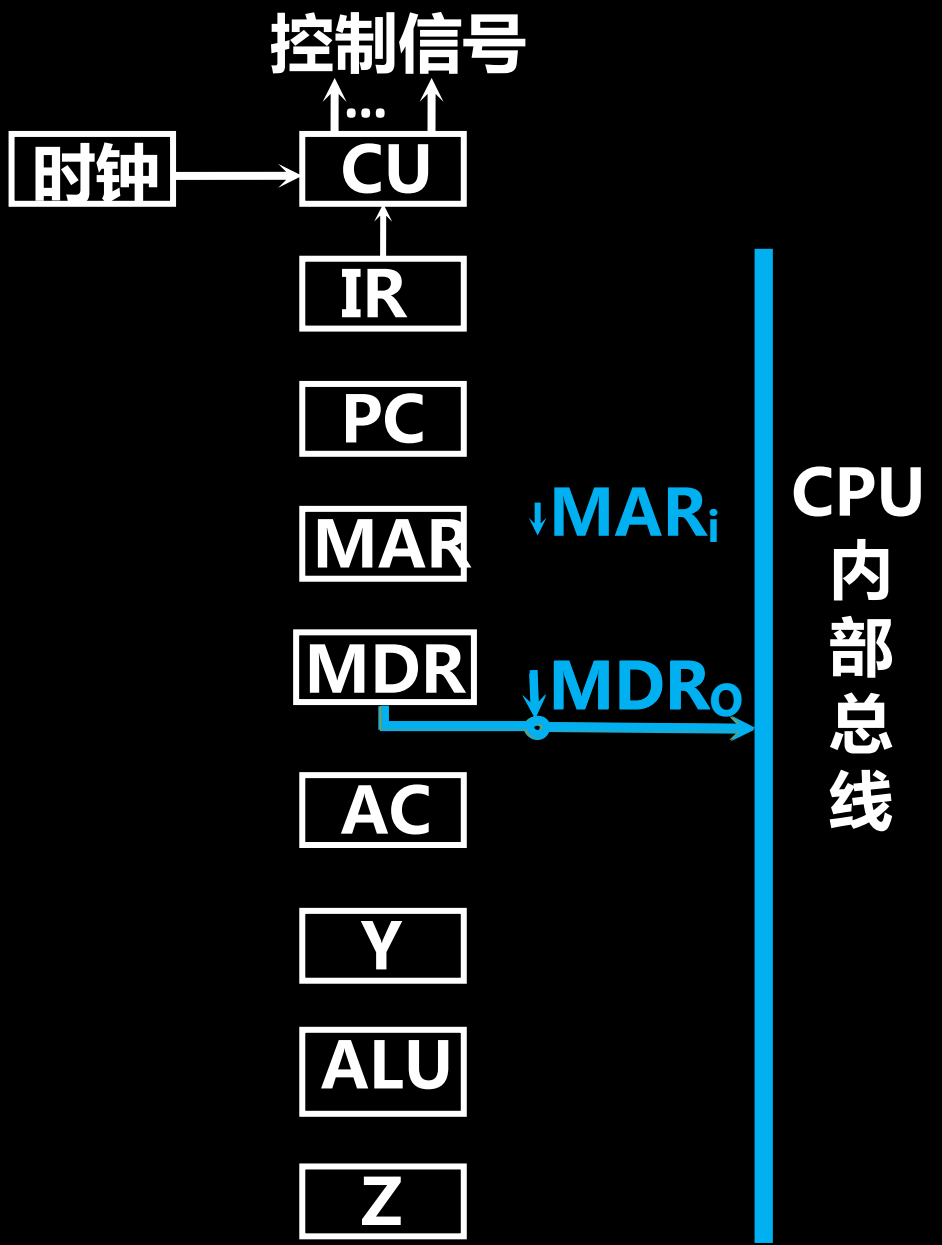
MDR⟶
MDR_O



采用CPU内部总线方式

(3) ADD @ X 执行周期

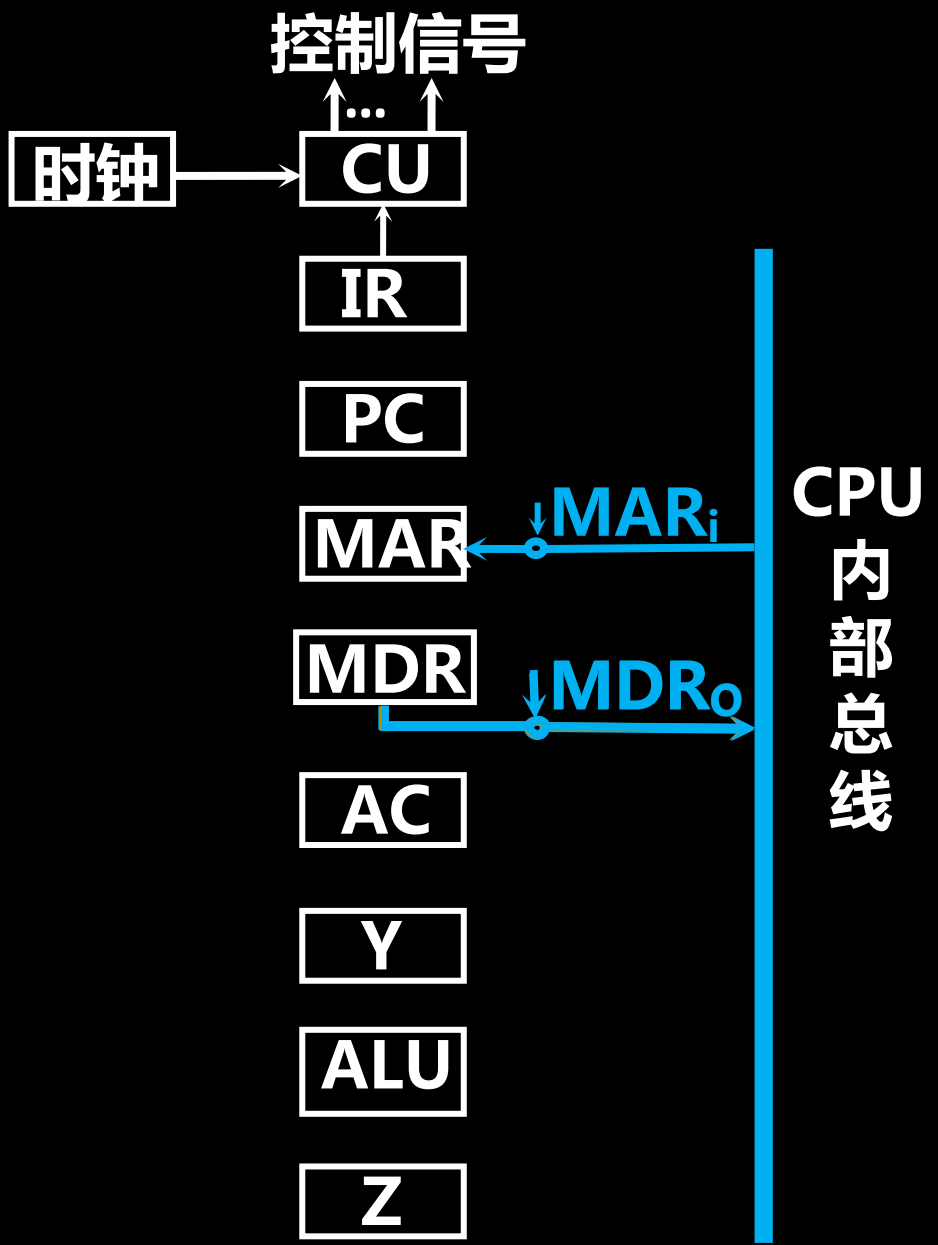
MDR⟶
MDR_o MAR_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶
MDR_o MAR_i

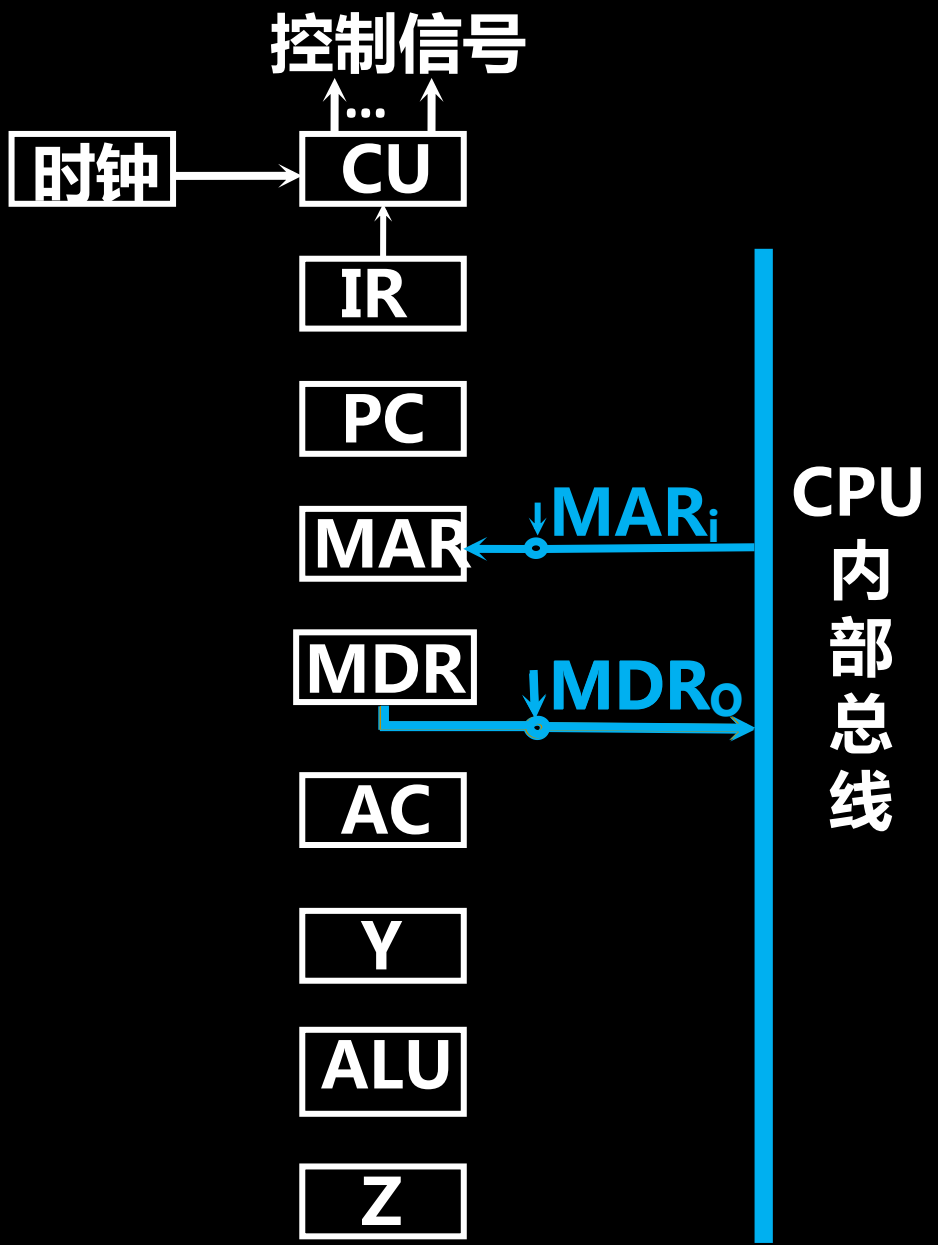


采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR

MDR_o MAR_i

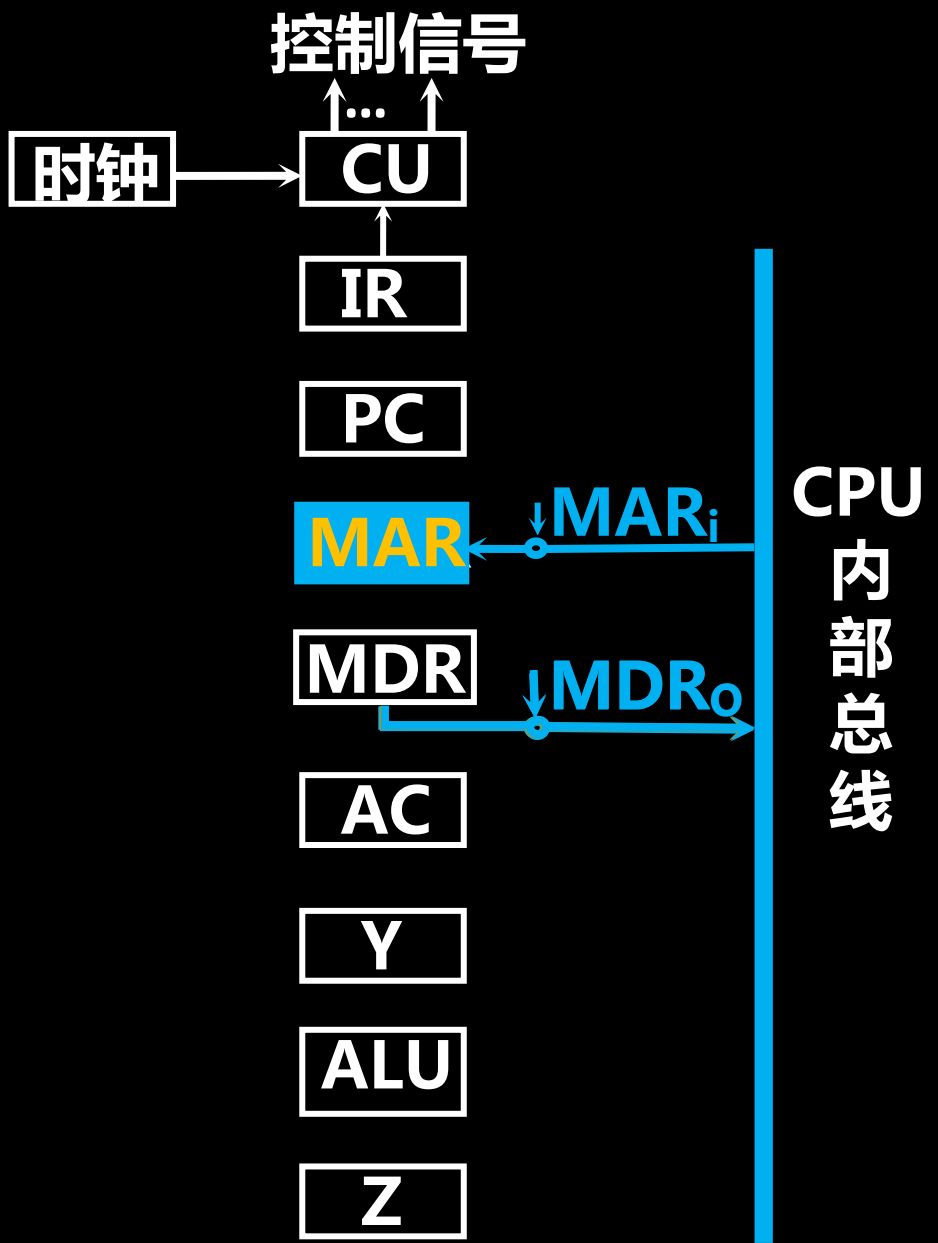


采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR

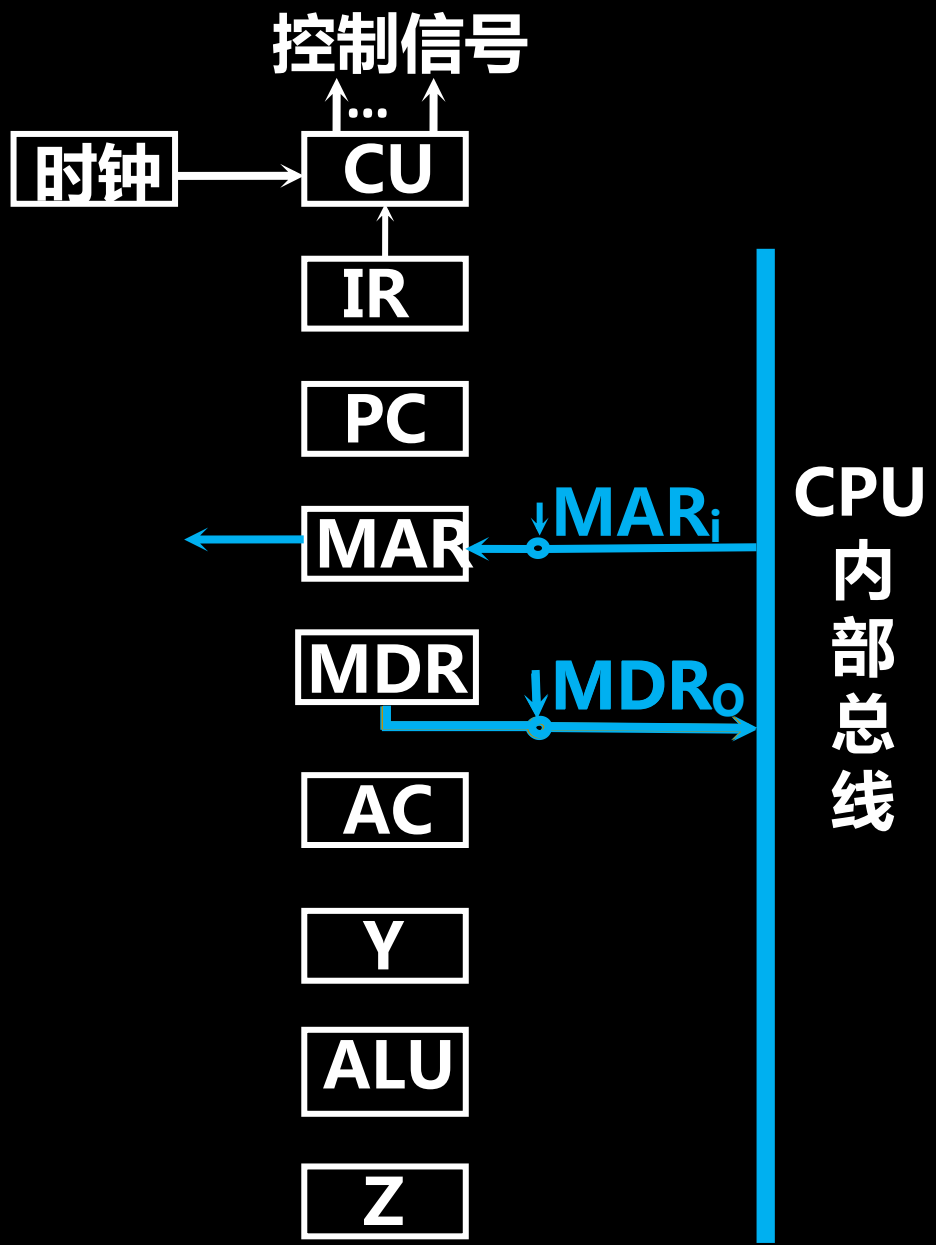
MDR_o MAR_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶
MDR_o MAR_i

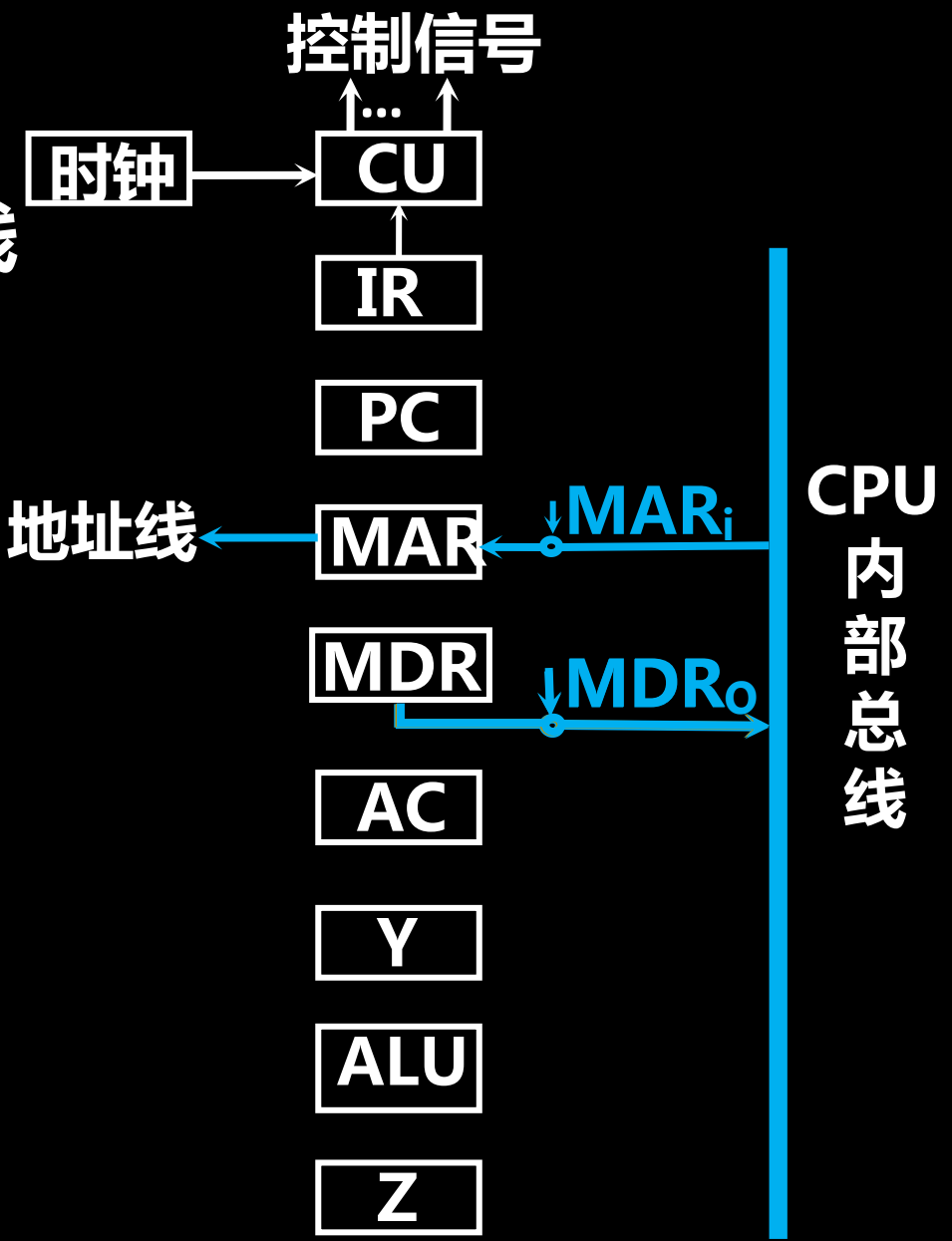


采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i



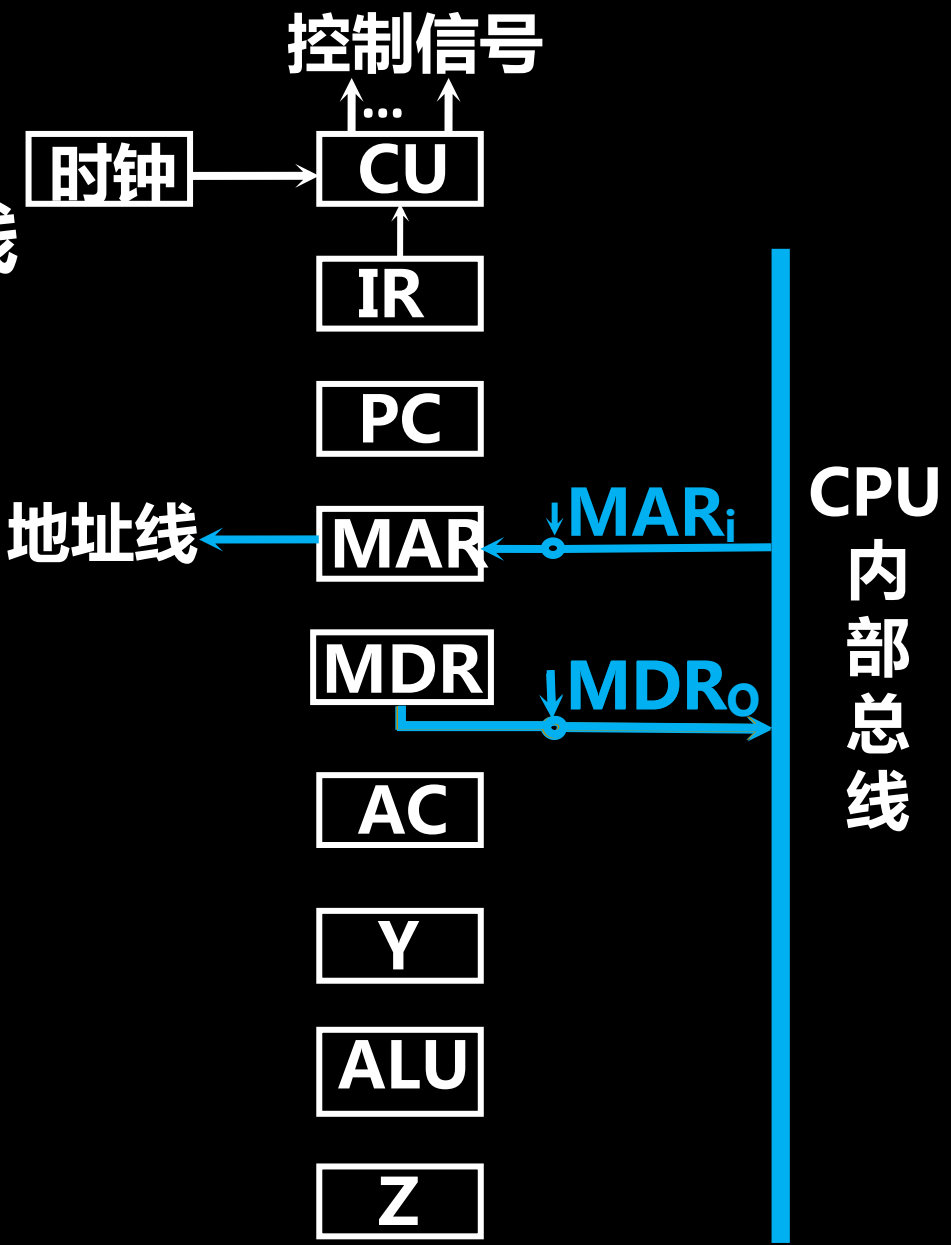
采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R



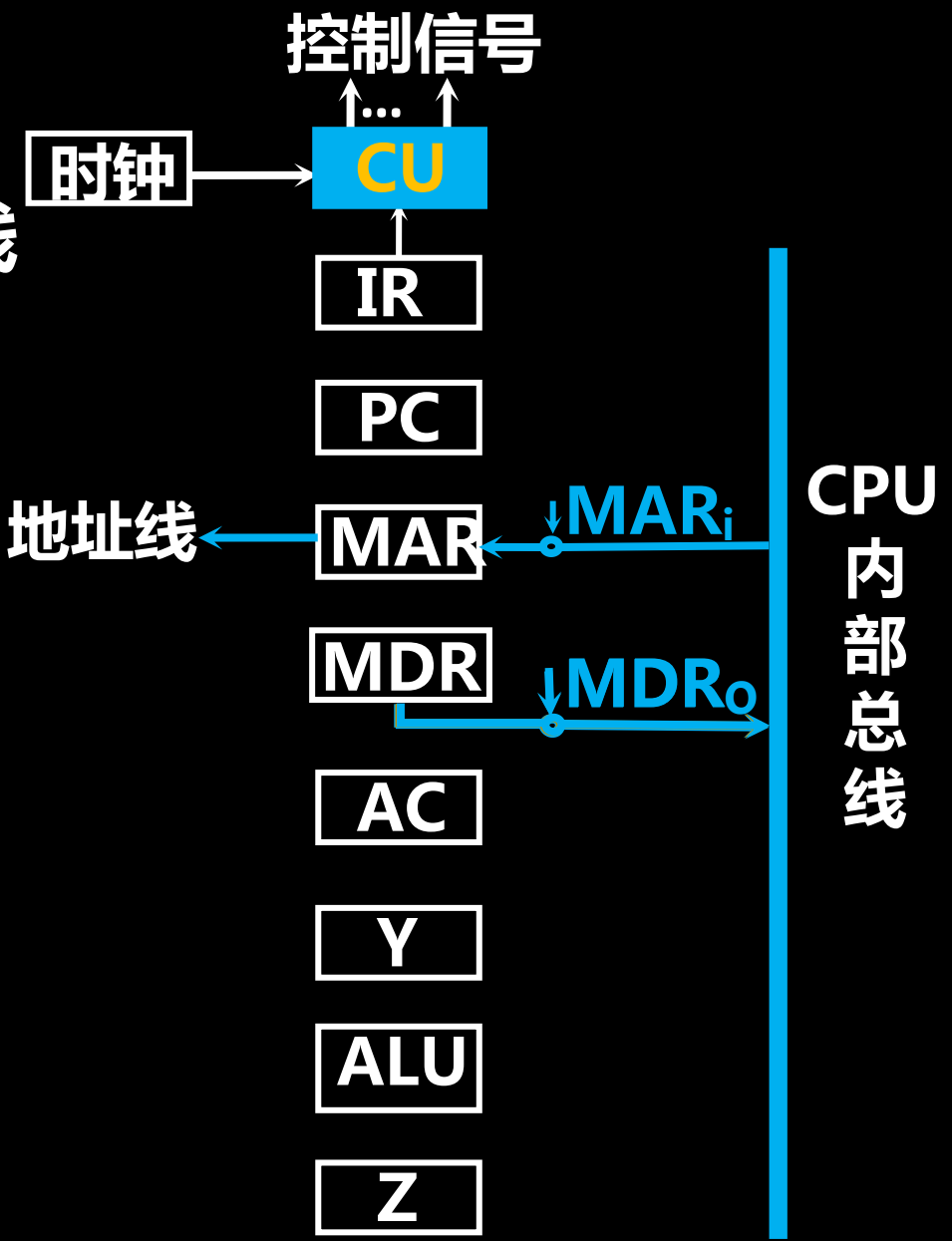
采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R



采用CPU内部总线方式

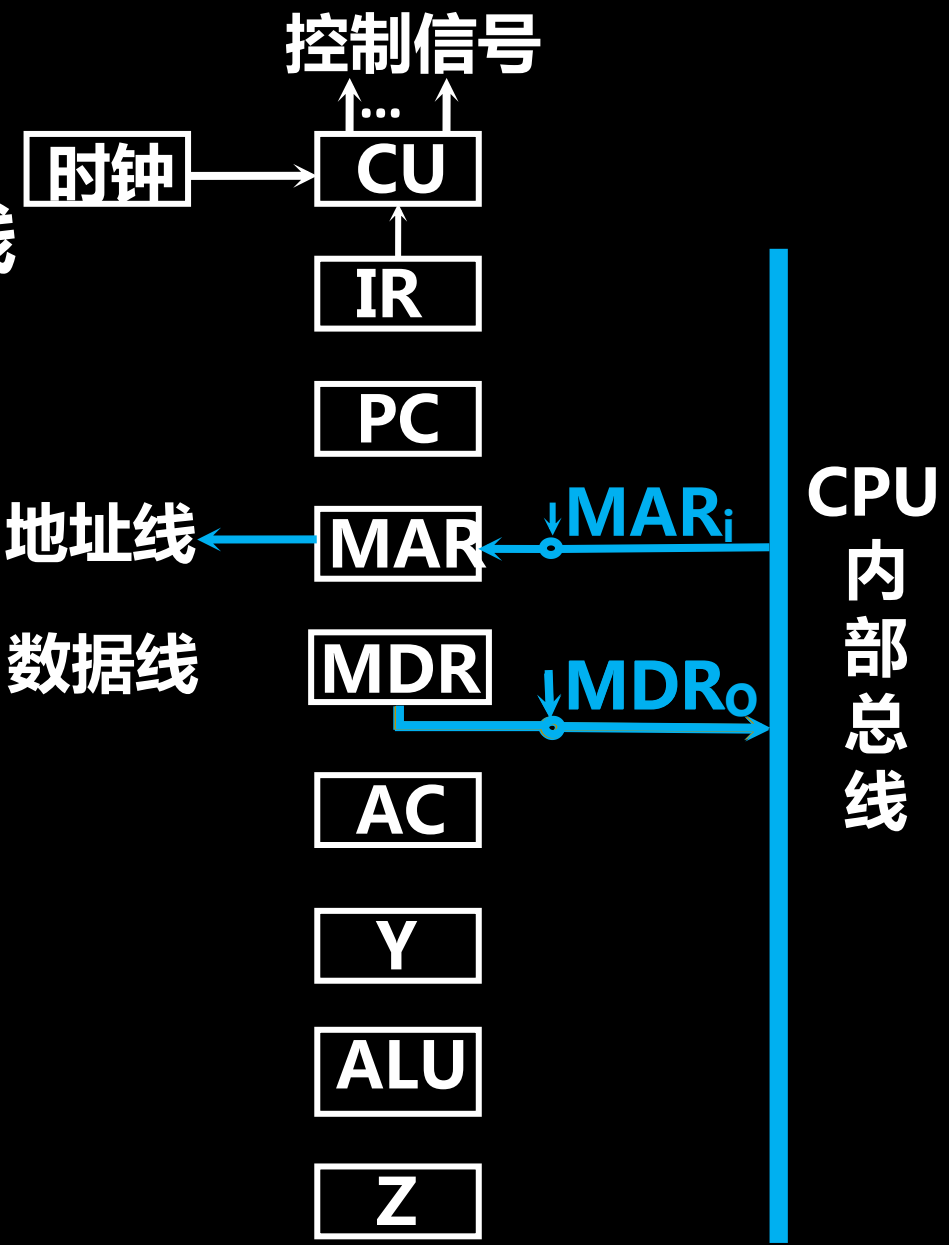
(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线



采用CPU内部总线方式

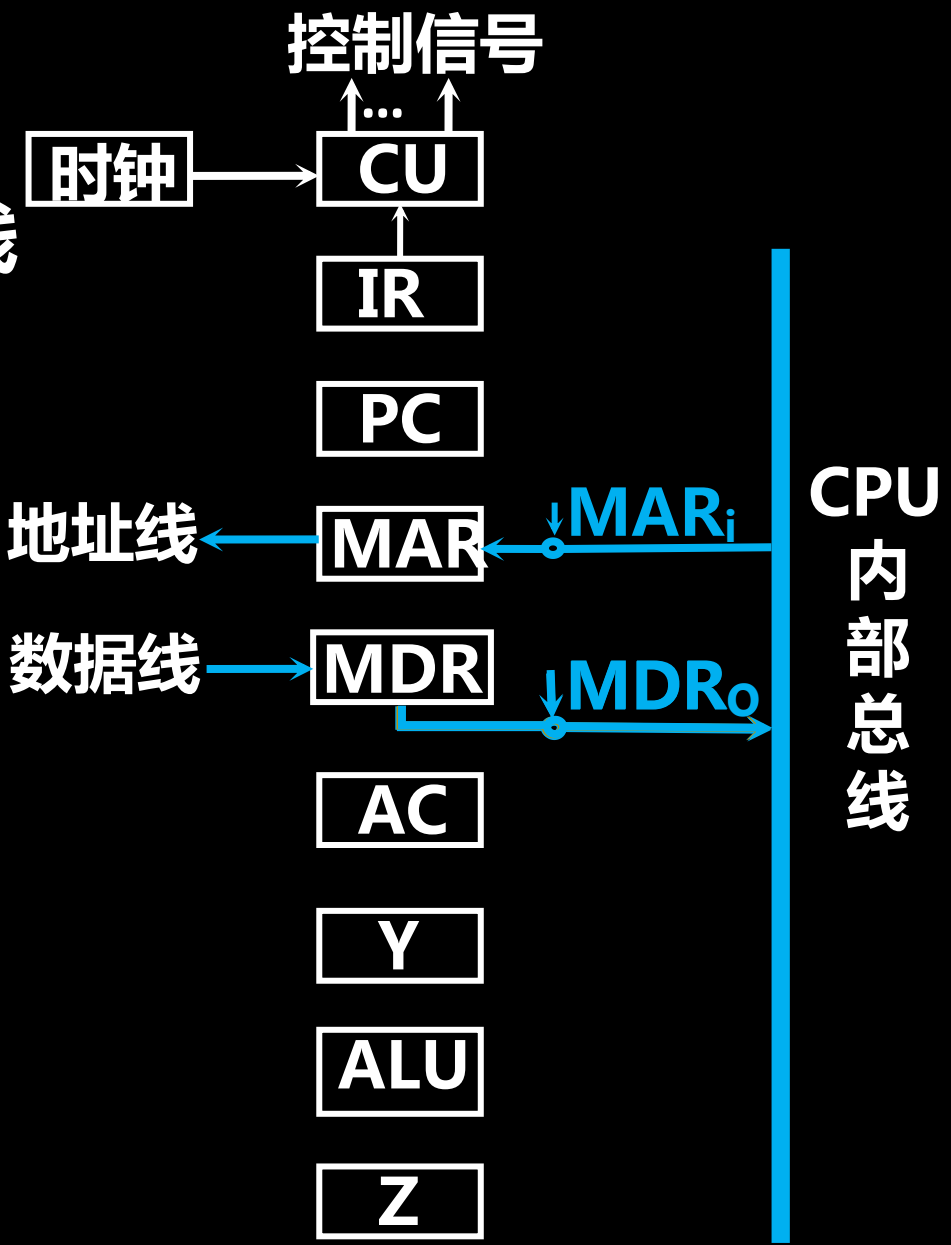
(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线⟶



采用CPU内部总线方式

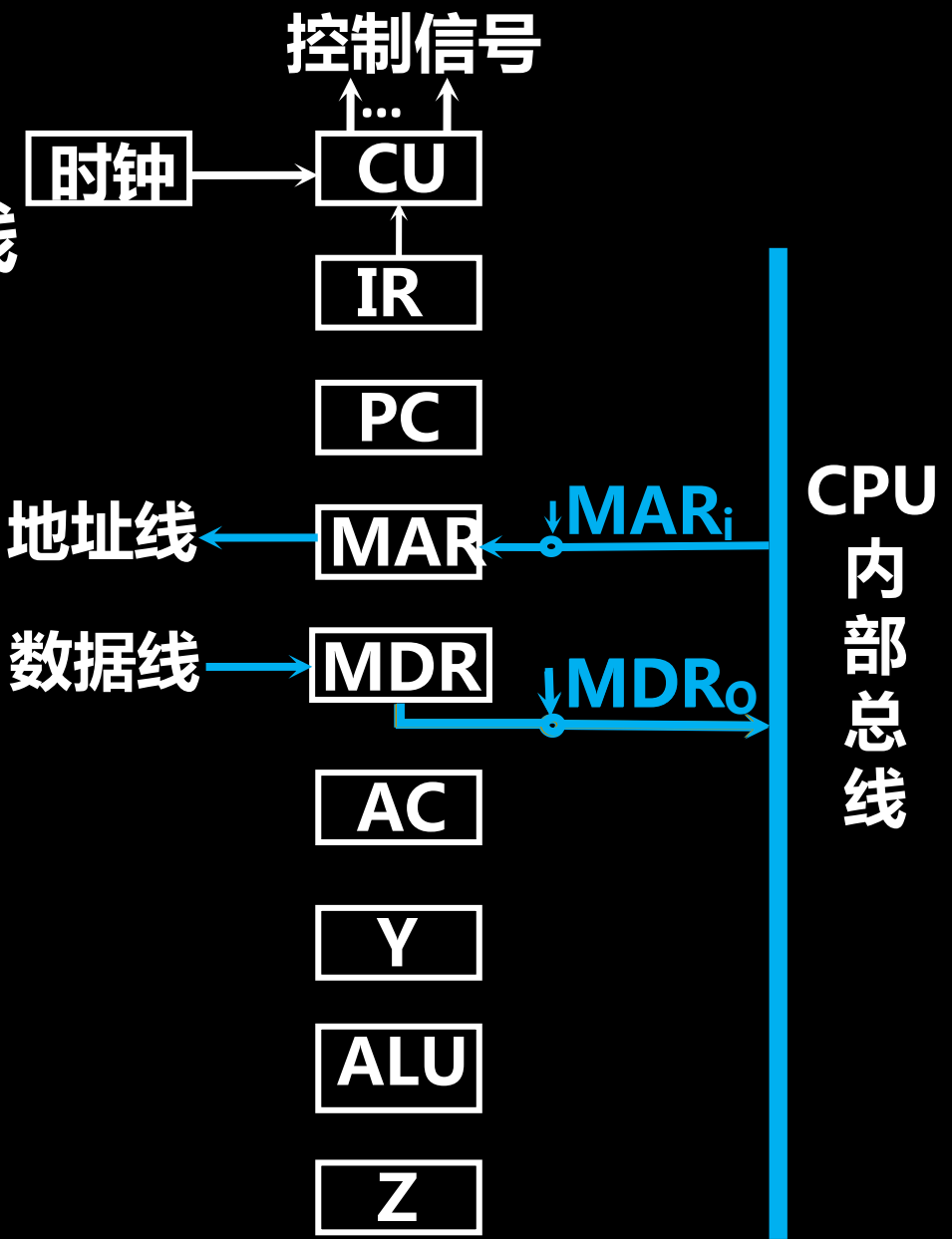
(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线⟶MDR



采用CPU内部总线方式

(3) ADD @ X 执行周期

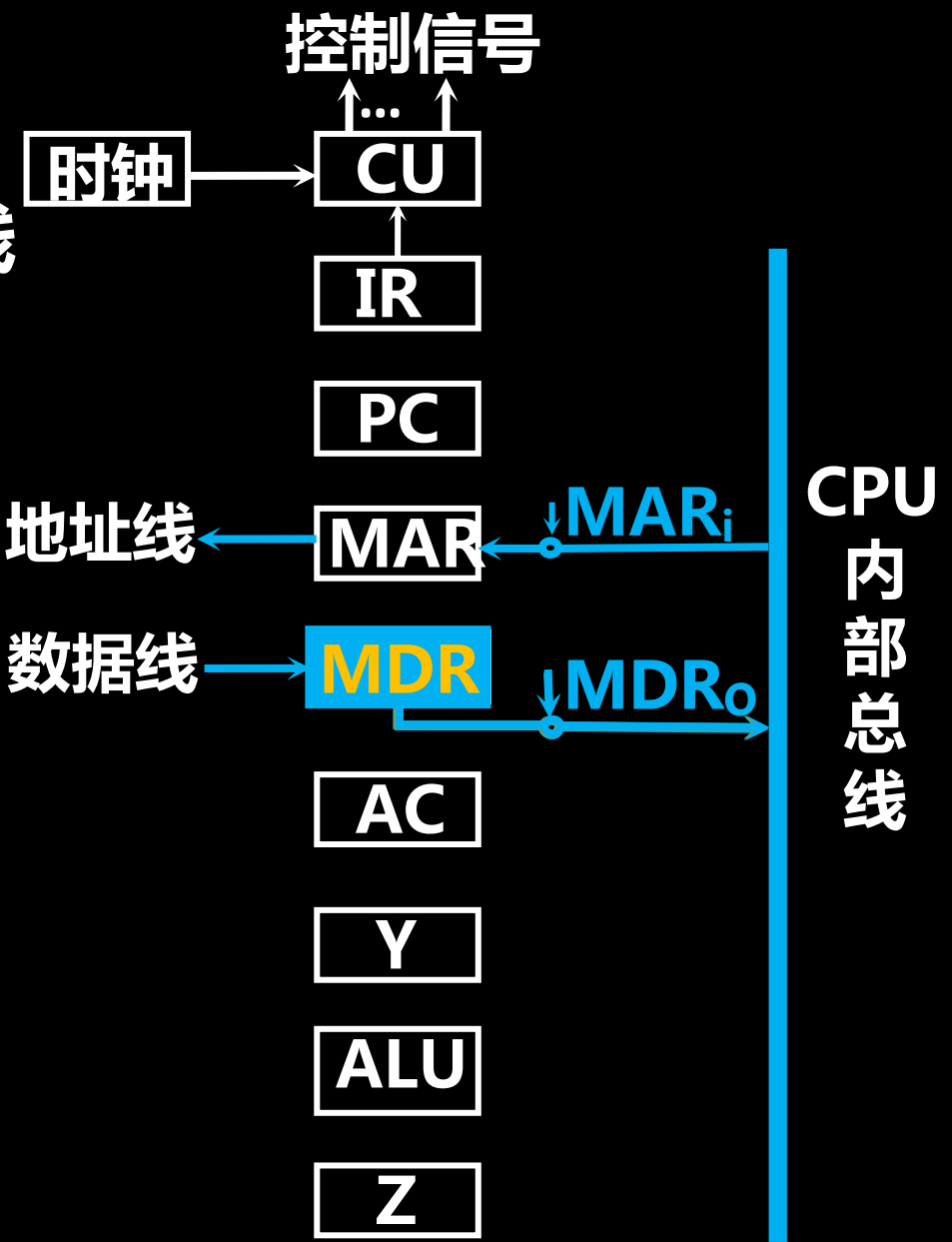
MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

MDR



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

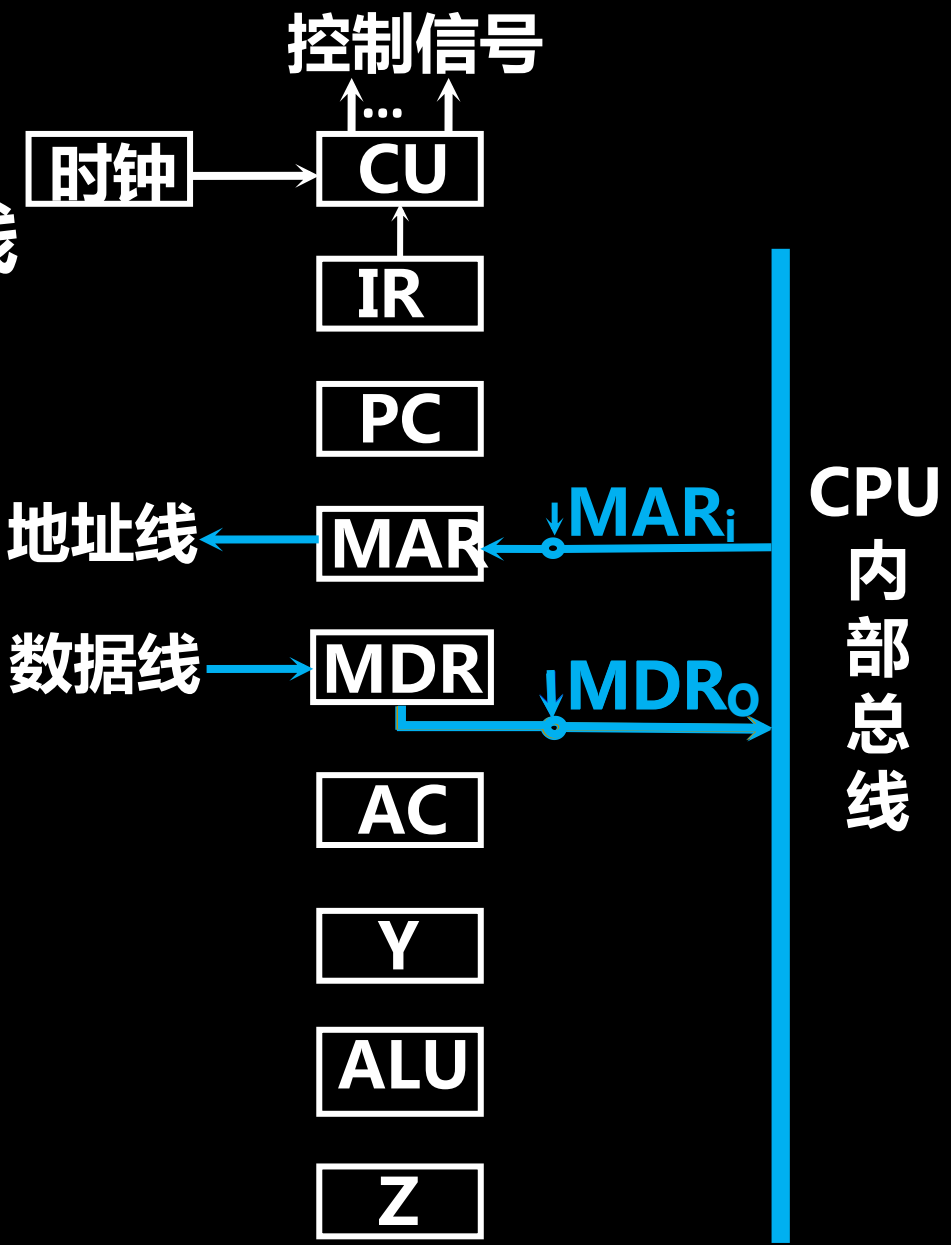
MDR_O MAR_i

1 ⟶ R

数据线⟶MDR

MDR

MDR_O



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

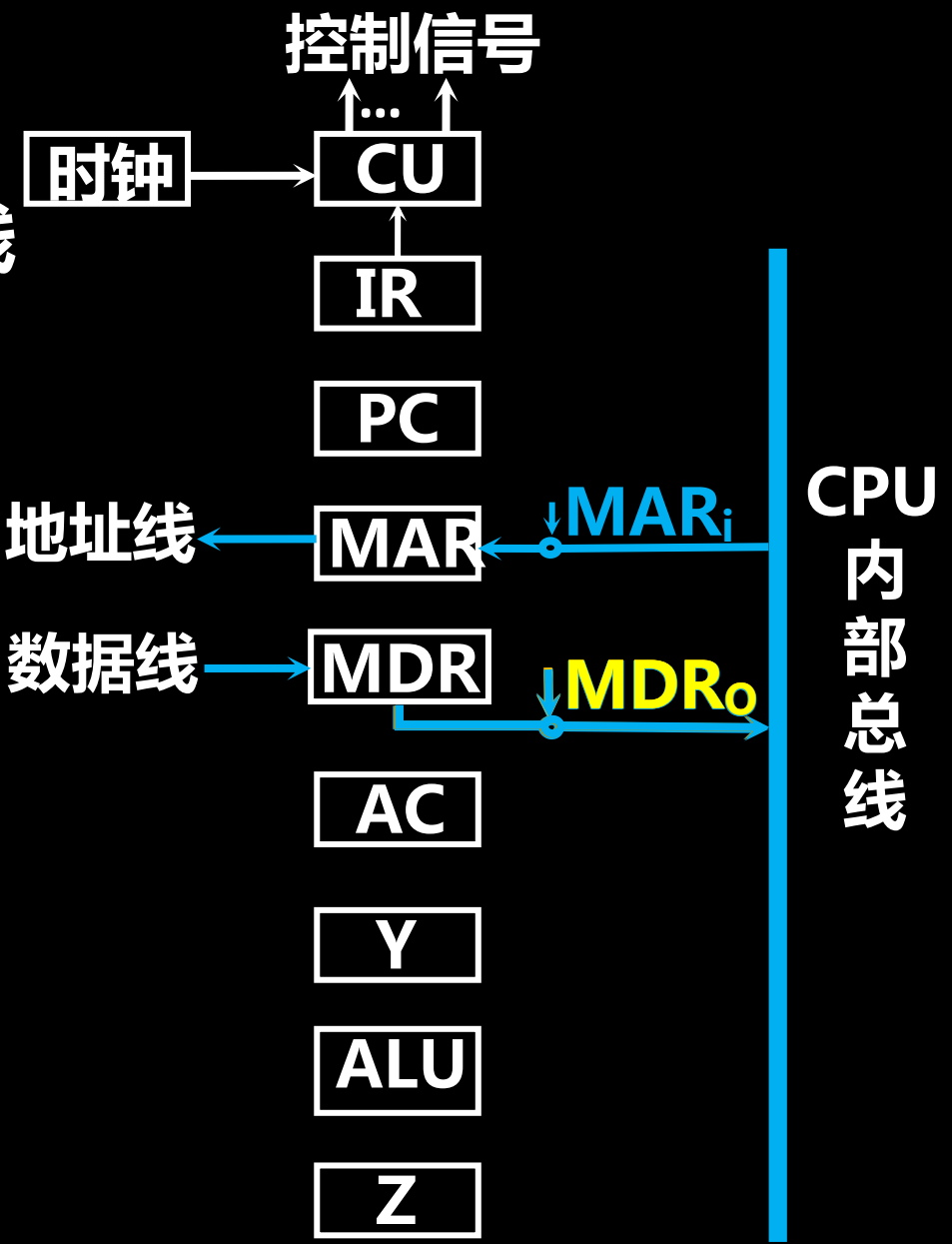
MDR_O MAR_i

1 ⟶ R

数据线⟶MDR

MDR

MDR_O



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

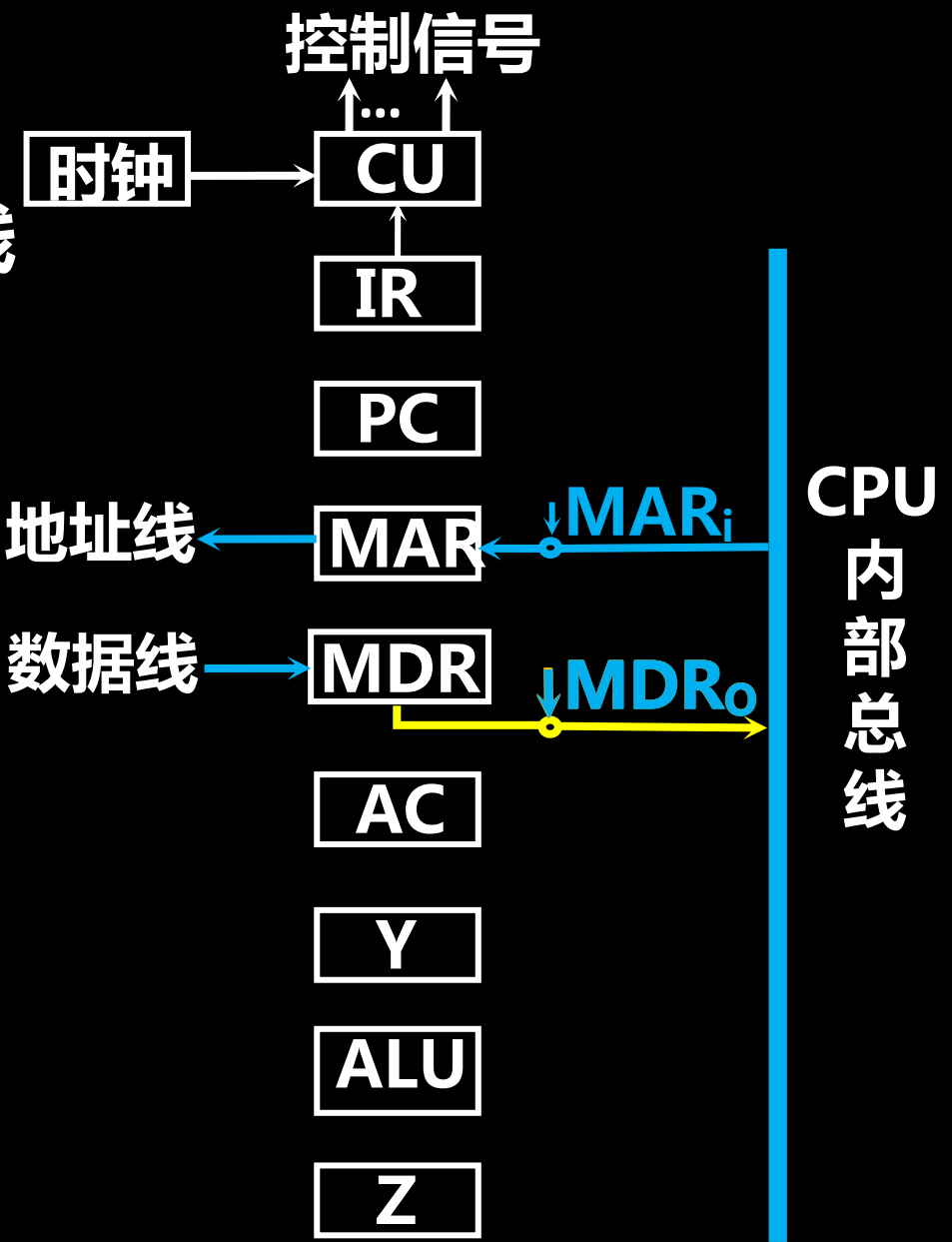
MDR_O MAR_i

1 ⟶ R

数据线⟶MDR

MDR⟶

MDR_O



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

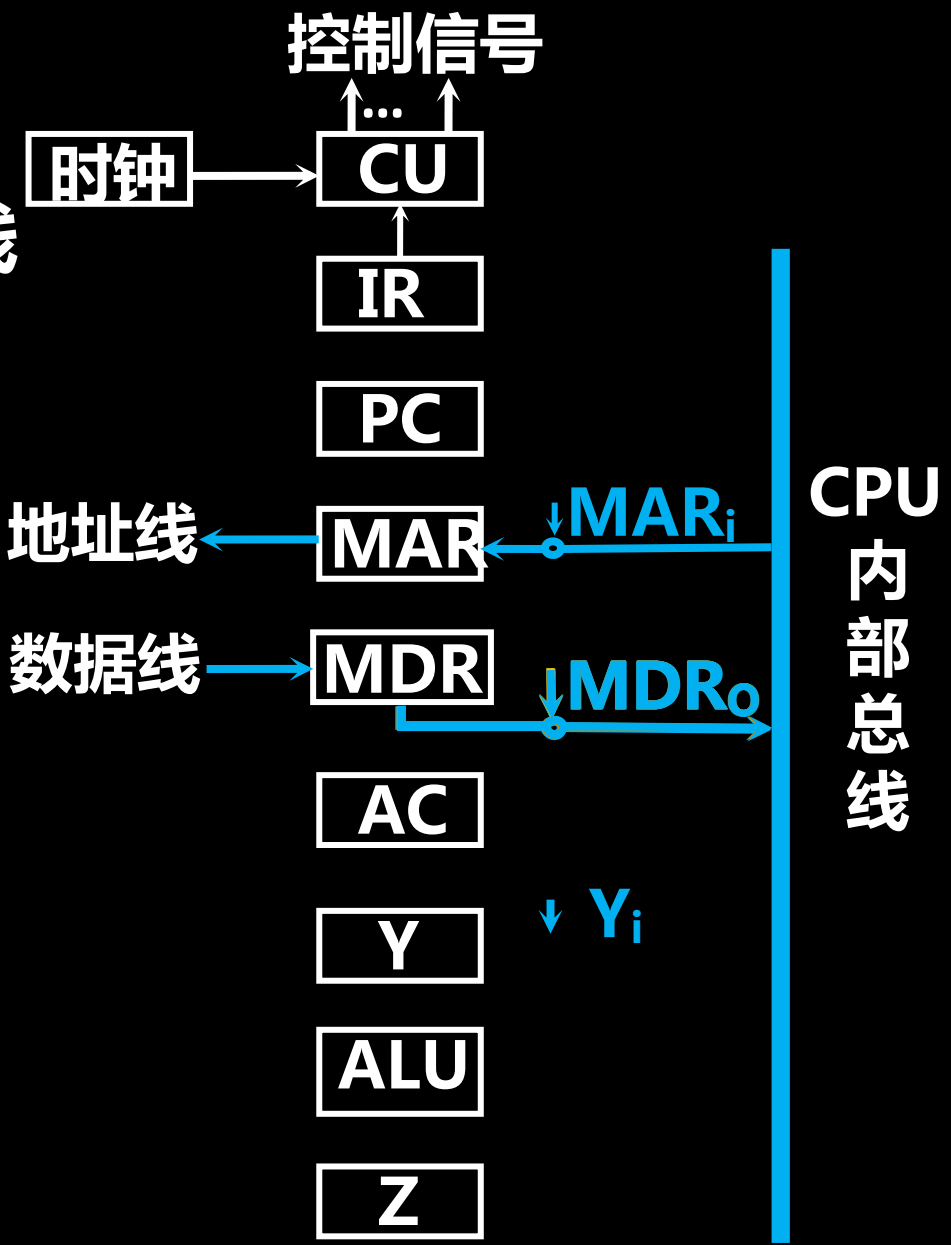
MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

MDR⟶

MDR_o Y_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

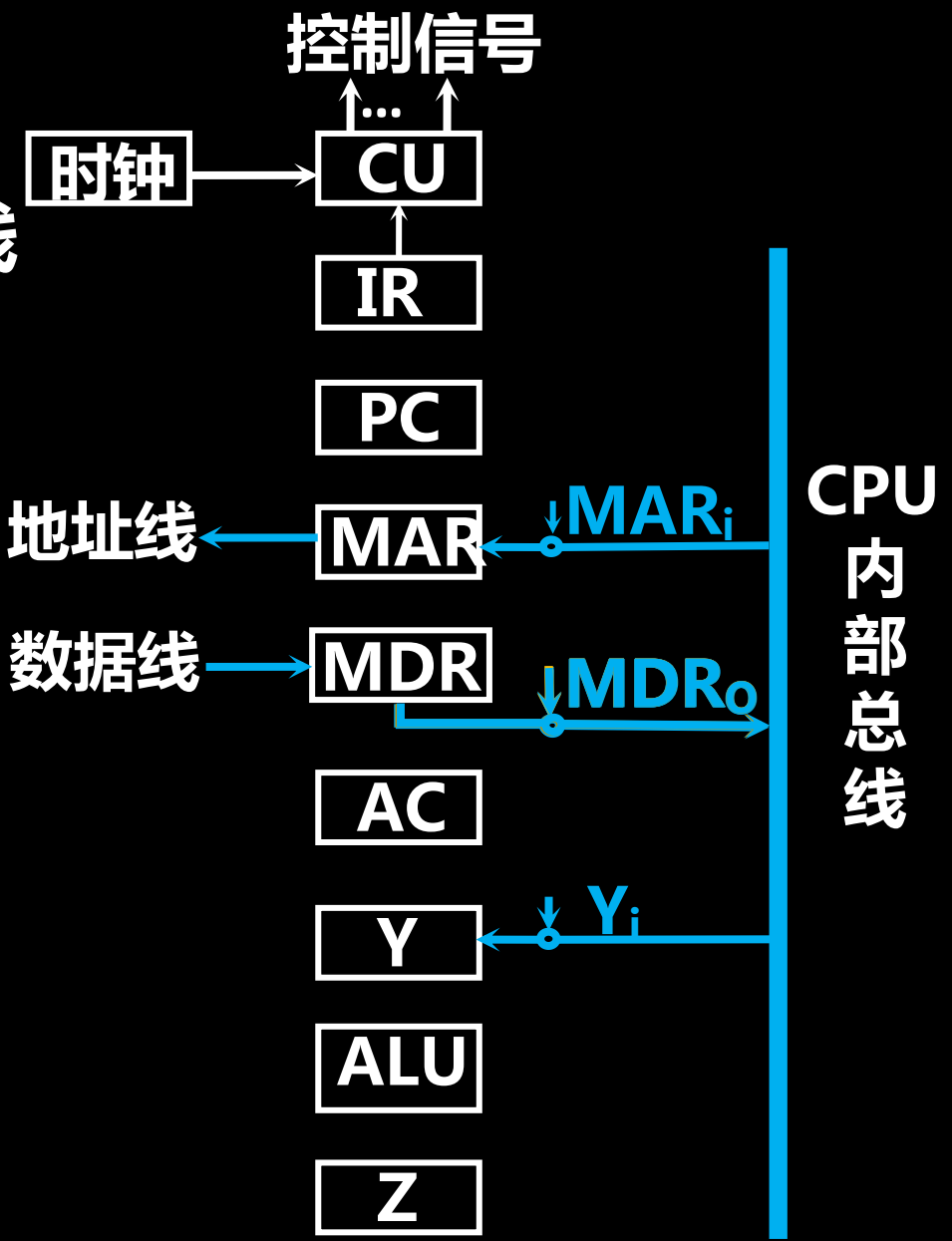
MDR_O MAR_i

1 ⟶ R

数据线⟶MDR

MDR⟶

MDR_O Y_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

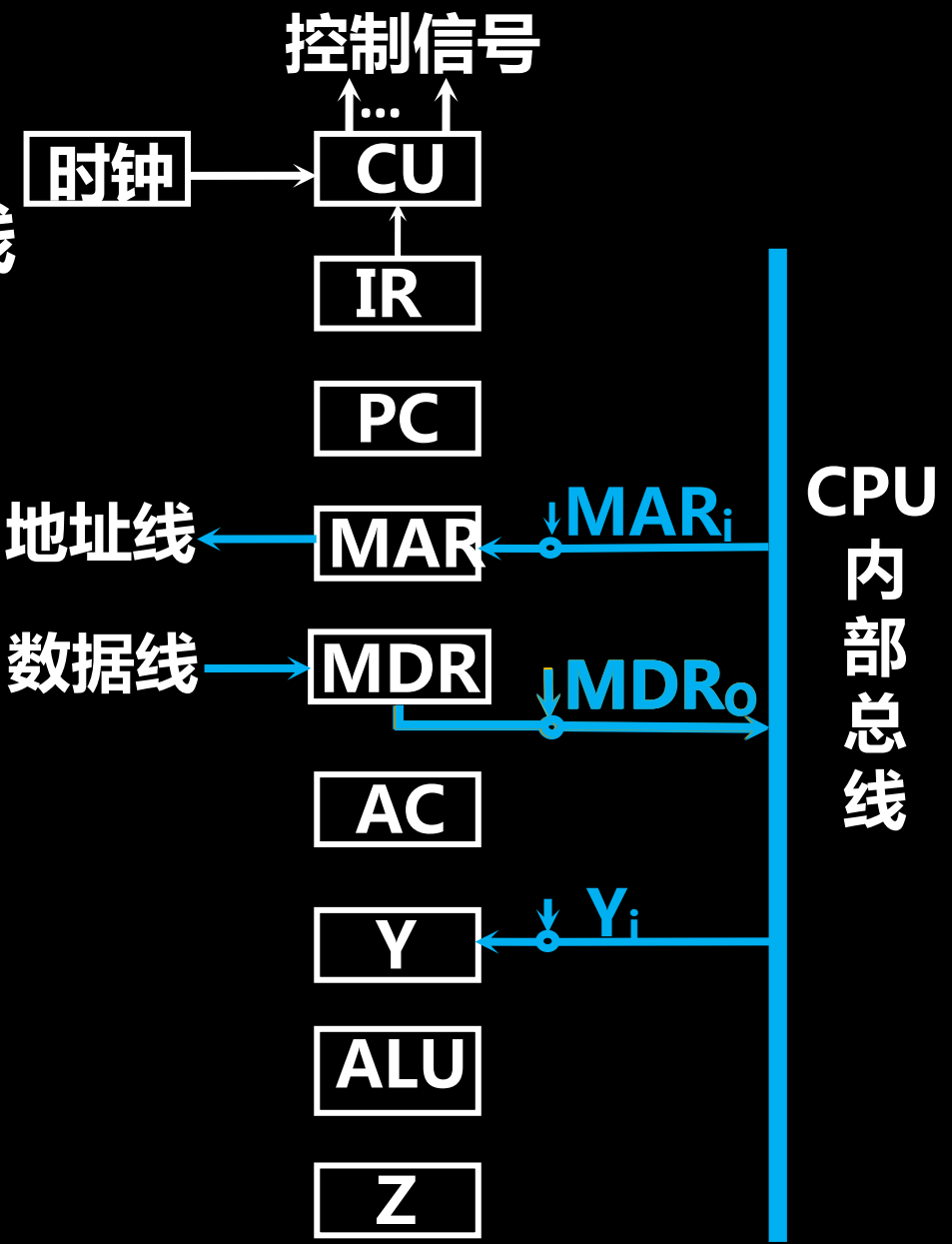
MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

MDR⟶Y

MDR_o Y_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

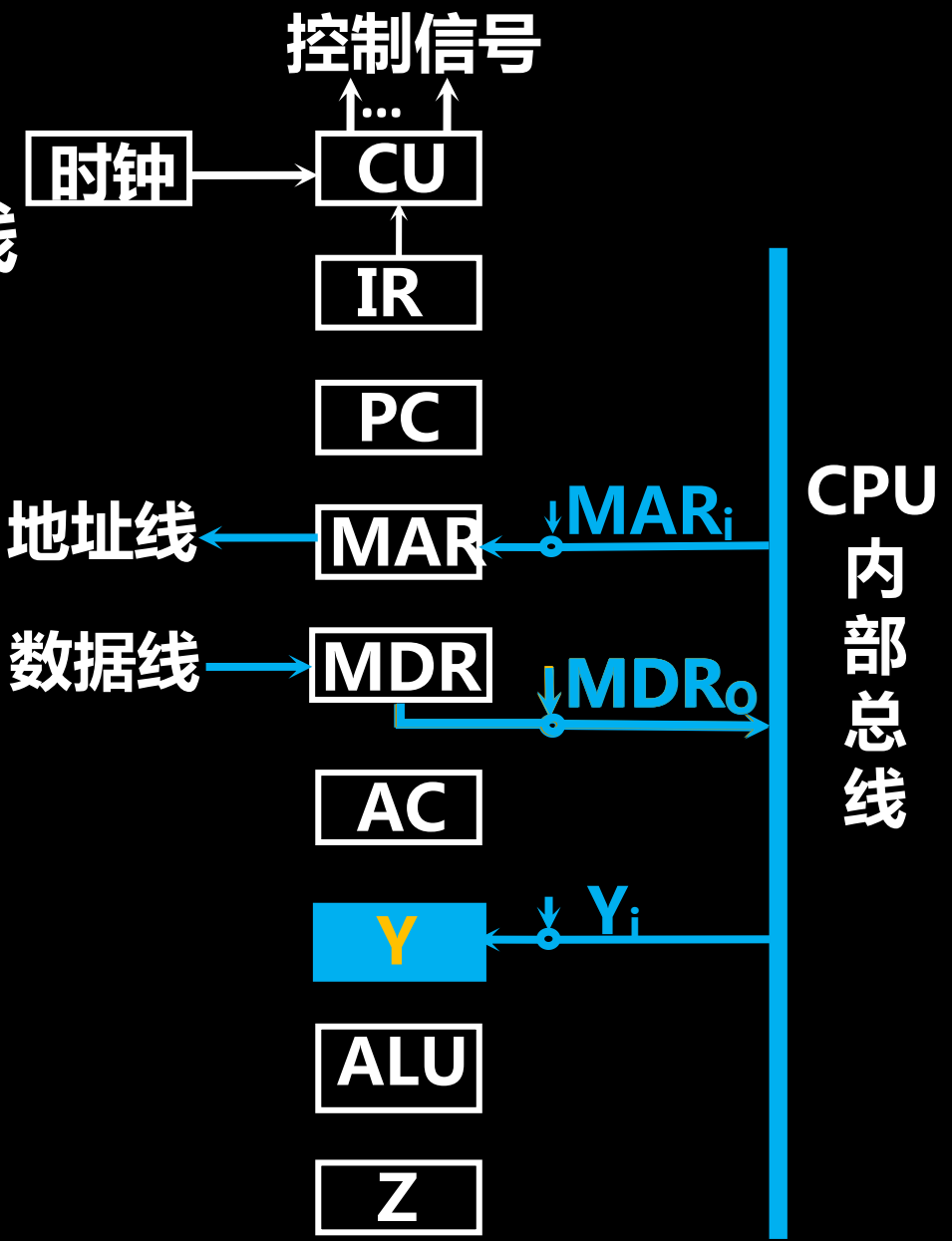
MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

MDR⟶Y

MDR_o Y_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

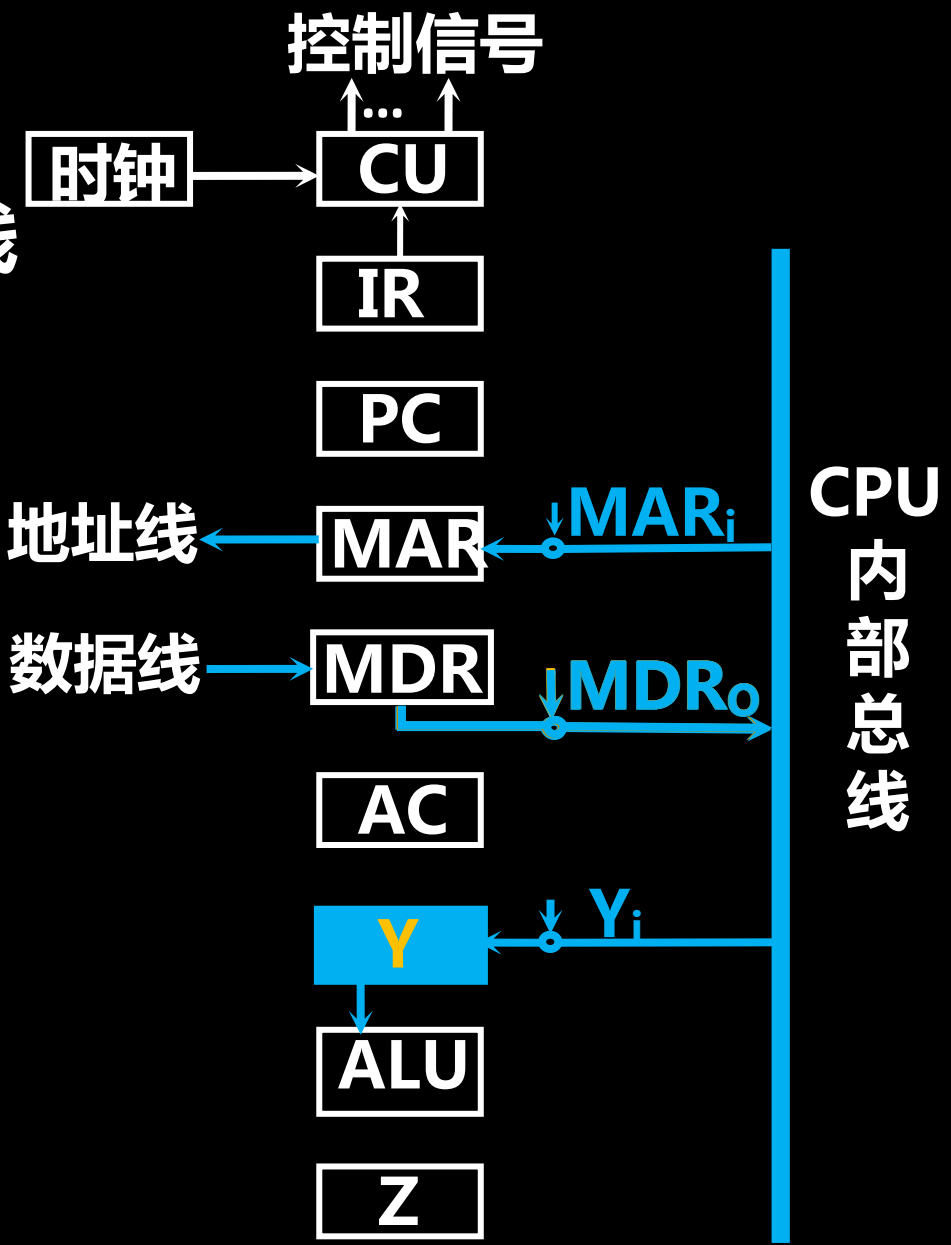
MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

MDR⟶Y⟶

MDR_o Y_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

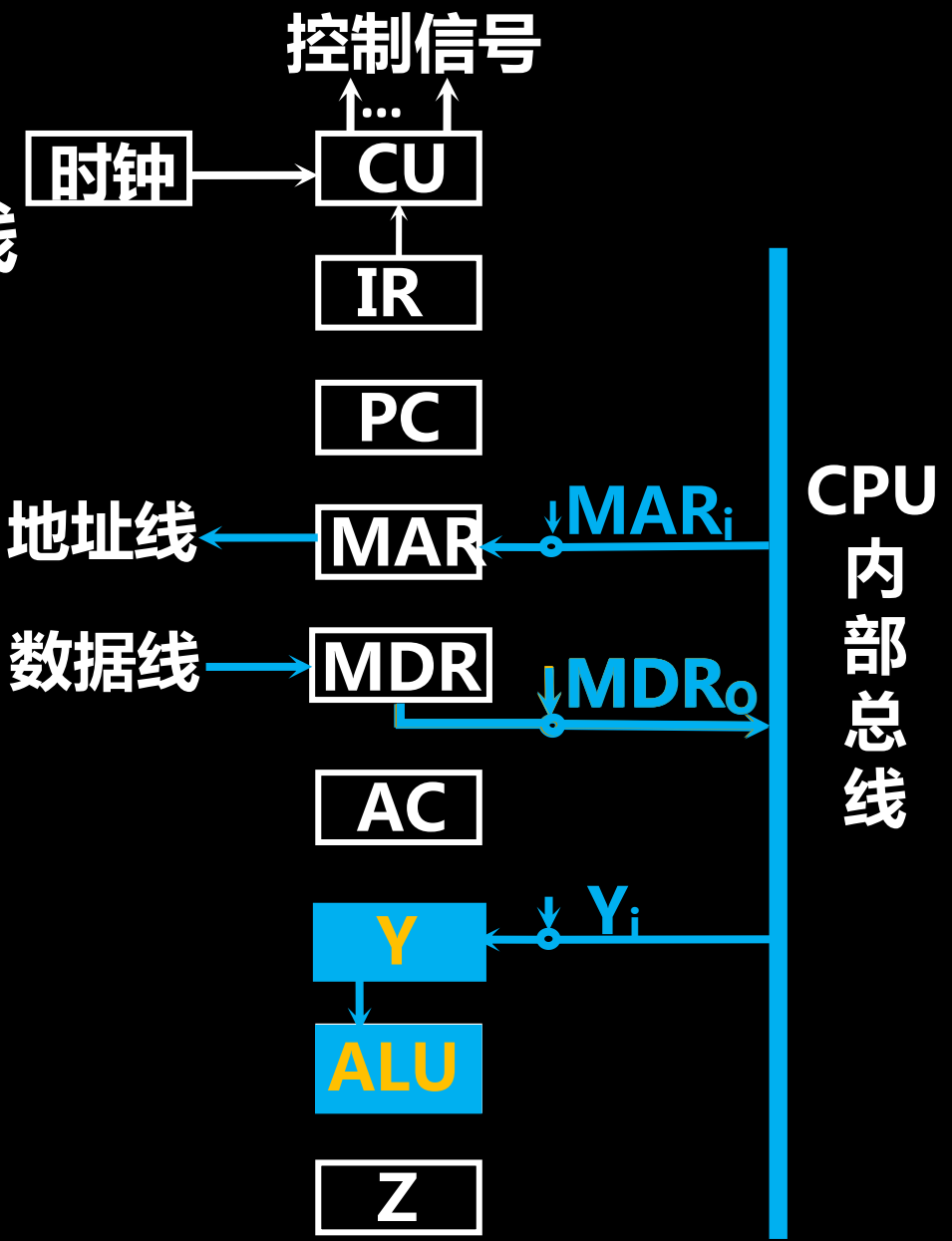
MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

MDR⟶Y⟶ALU

MDR_o Y_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_O MAR_i

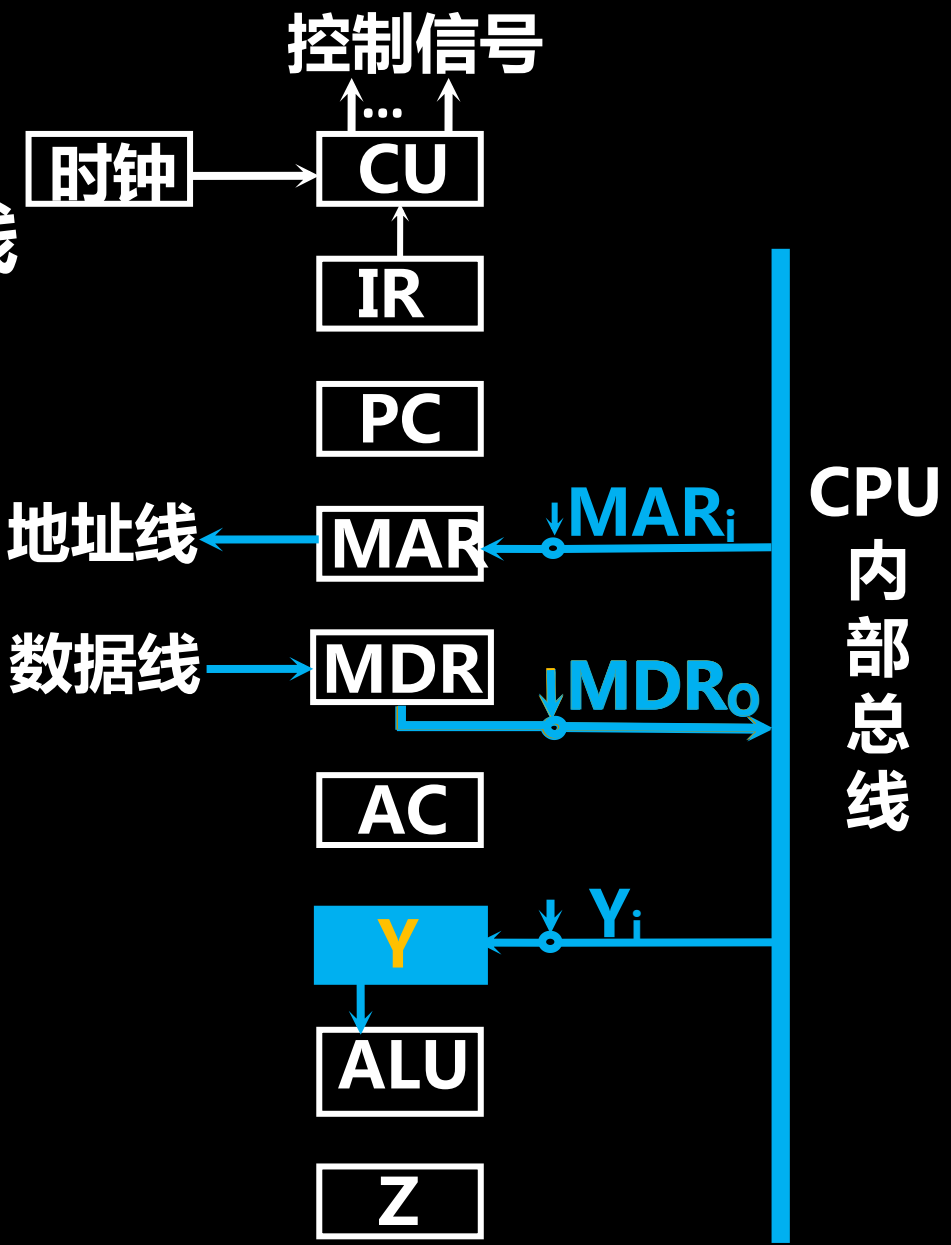
1 ⟶ R

数据线⟶MDR

MDR⟶Y⟶ALU

MDR_O Y_i

AC



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_O MAR_i

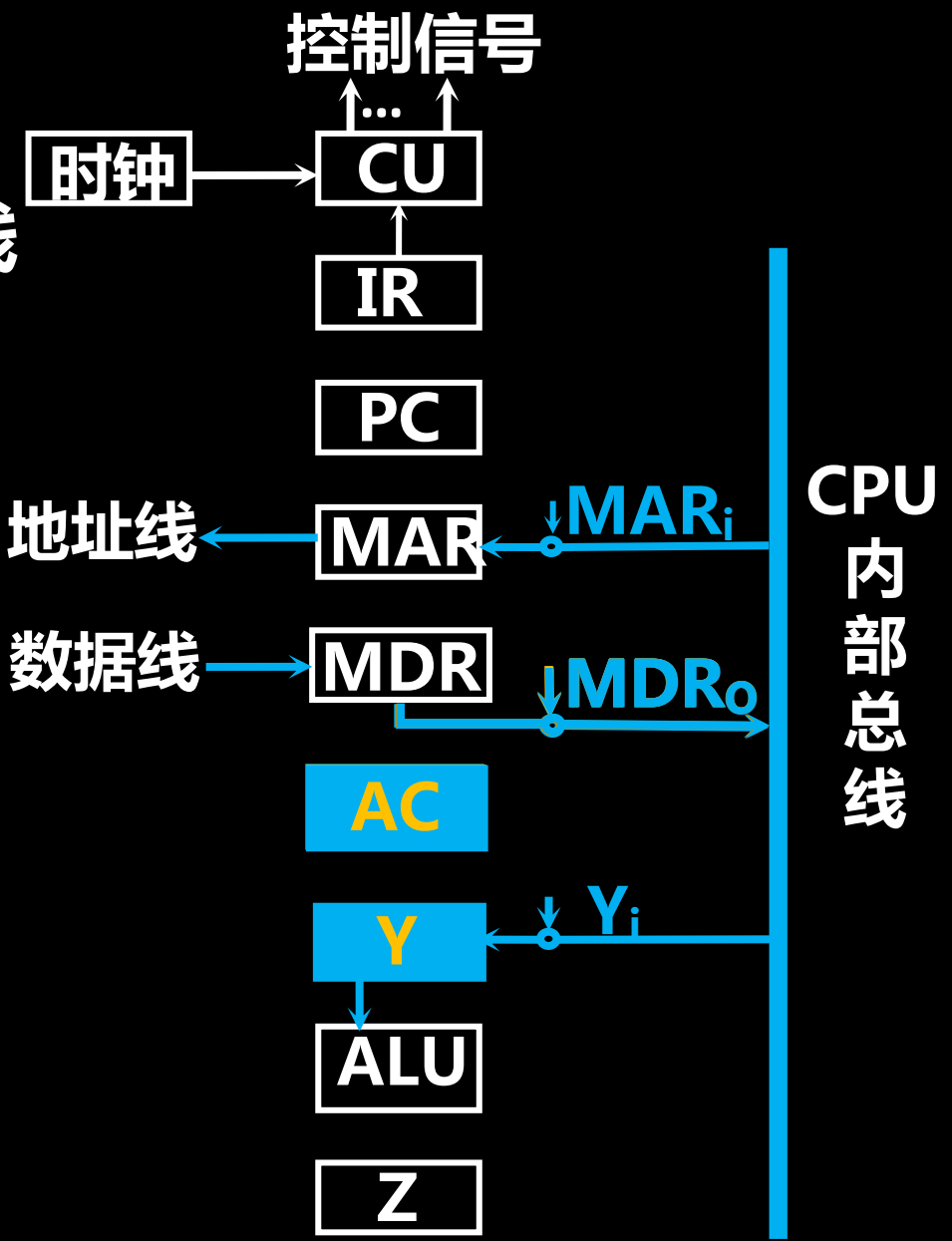
1 ⟶ R

数据线⟶MDR

MDR⟶Y⟶ALU

MDR_O Y_i

AC



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

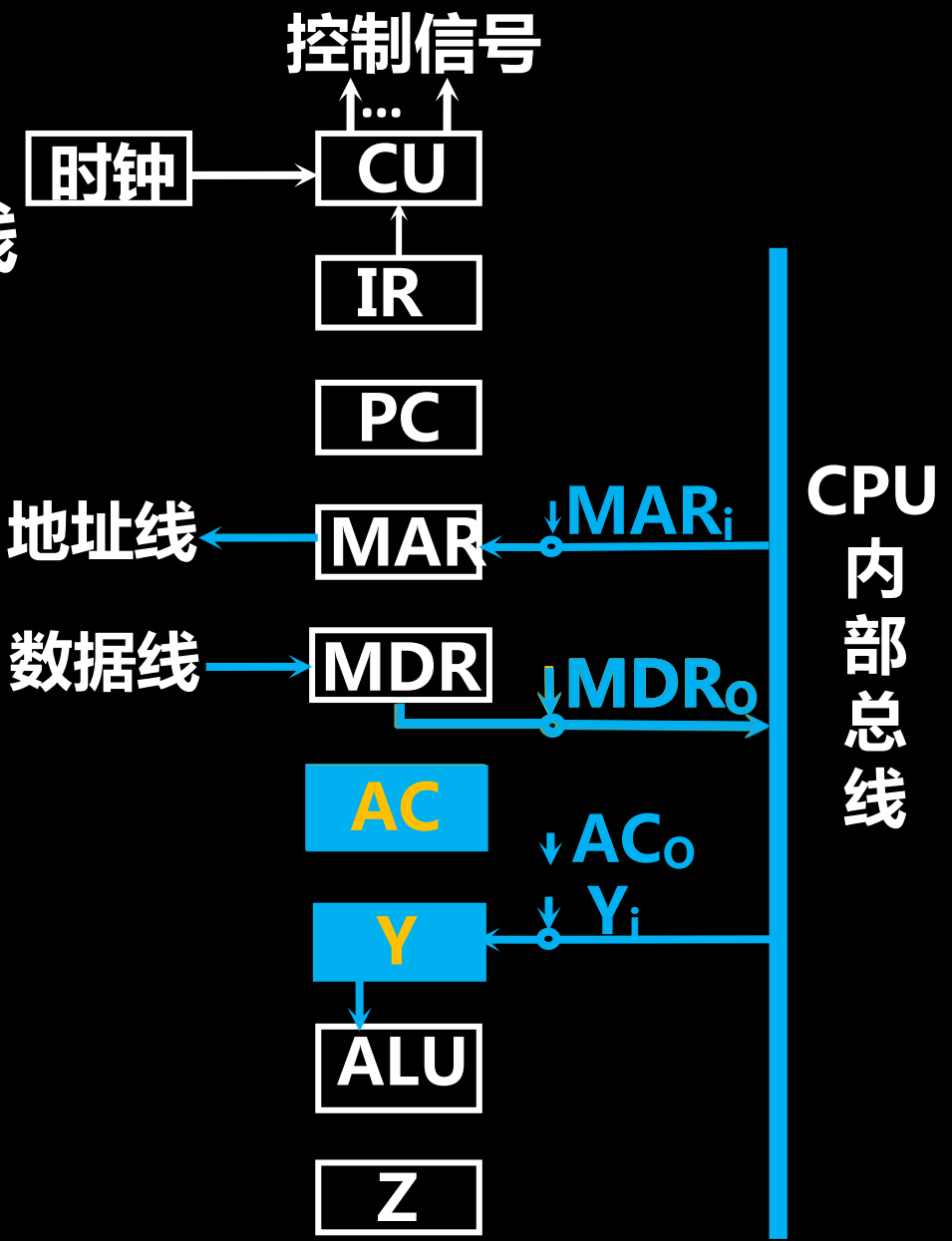
数据线⟶MDR

MDR⟶Y⟶ALU

MDR_o Y_i

AC

AC_o



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

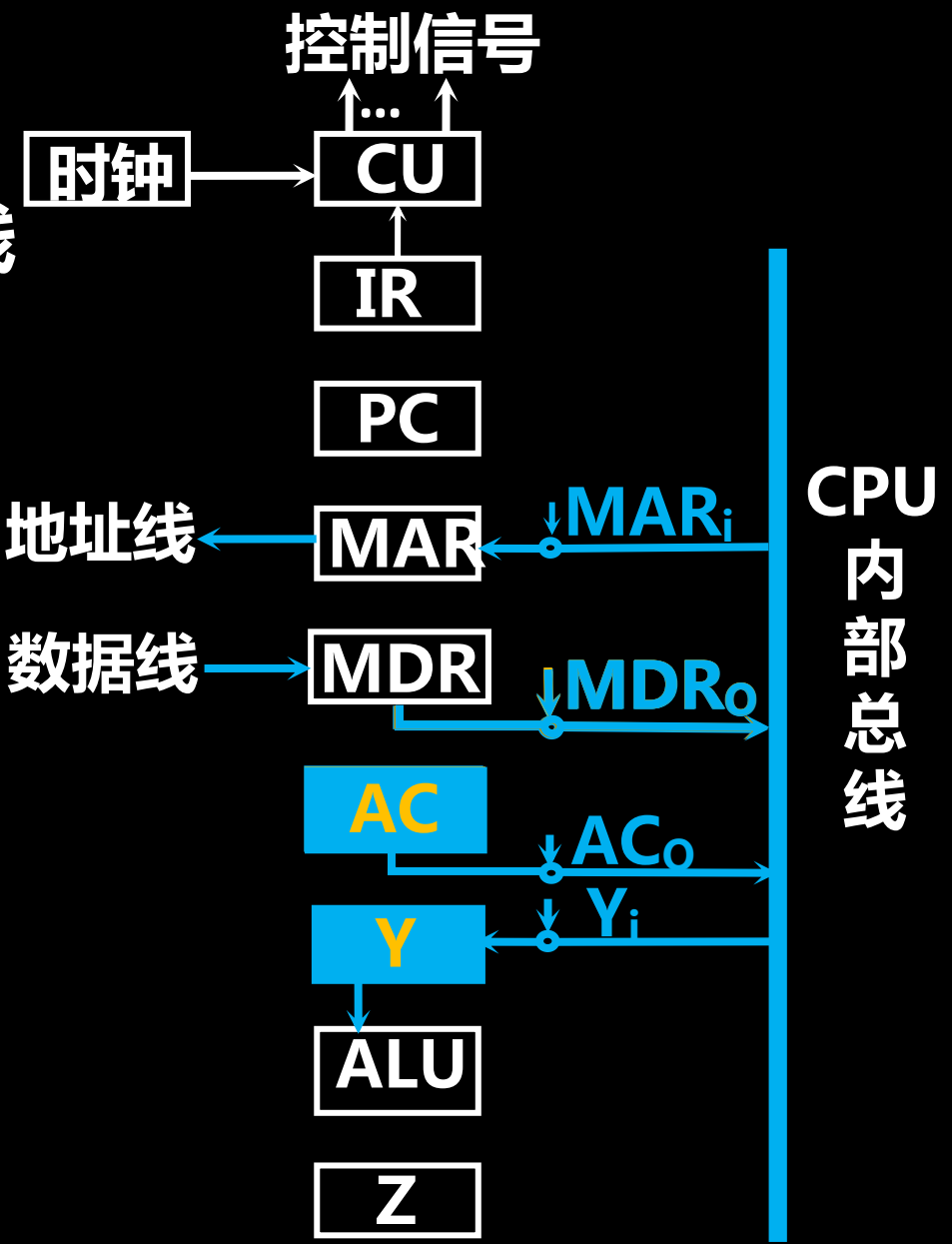
数据线⟶MDR

MDR⟶Y⟶ALU

MDR_o Y_i

AC ⟶

AC_o



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

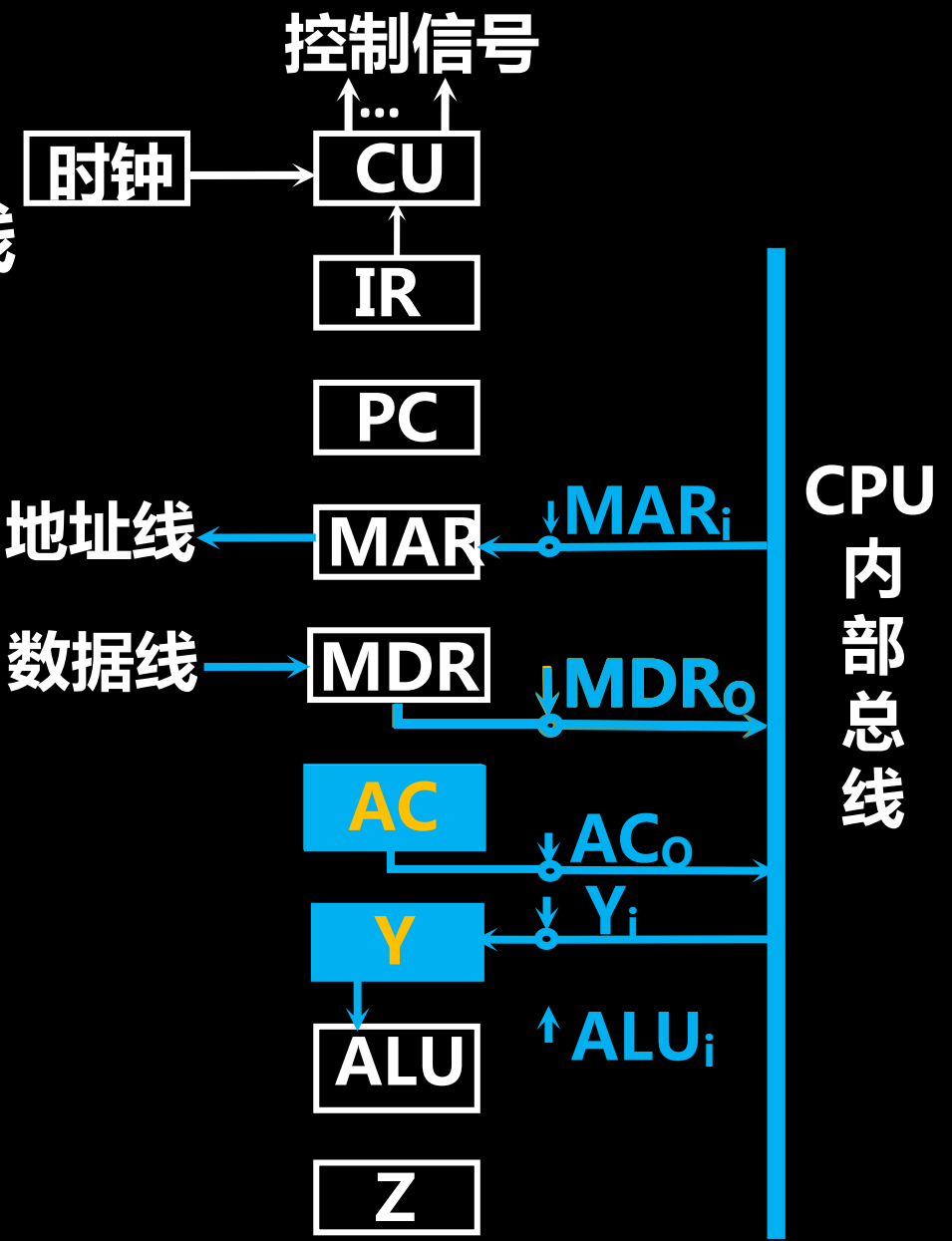
数据线⟶MDR

MDR⟶Y⟶ALU

MDR_o Y_i

AC ⟶

AC_o ALU_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

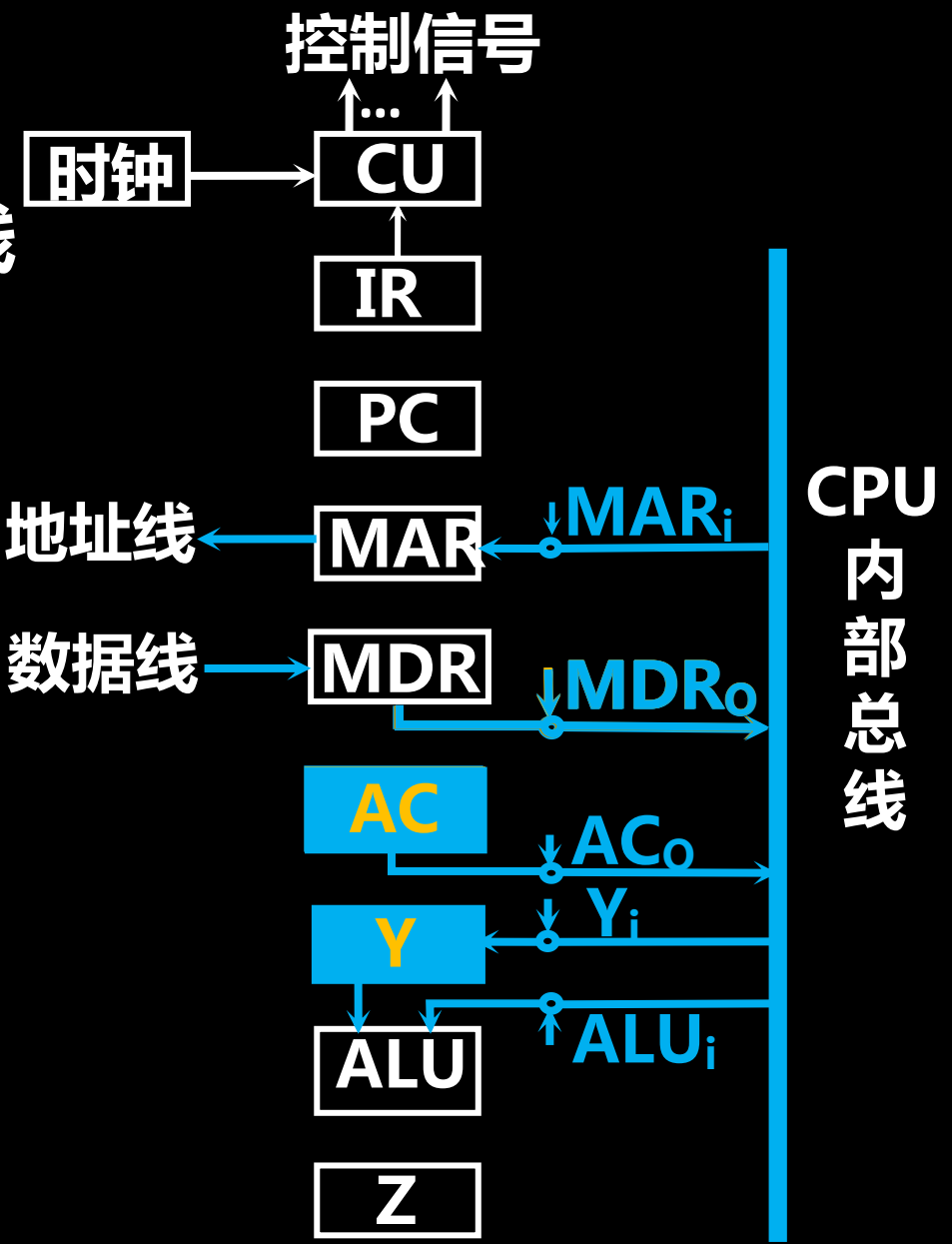
数据线⟶MDR

MDR⟶Y⟶ALU

MDR_o Y_i

AC ⟶

AC_o ALU_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

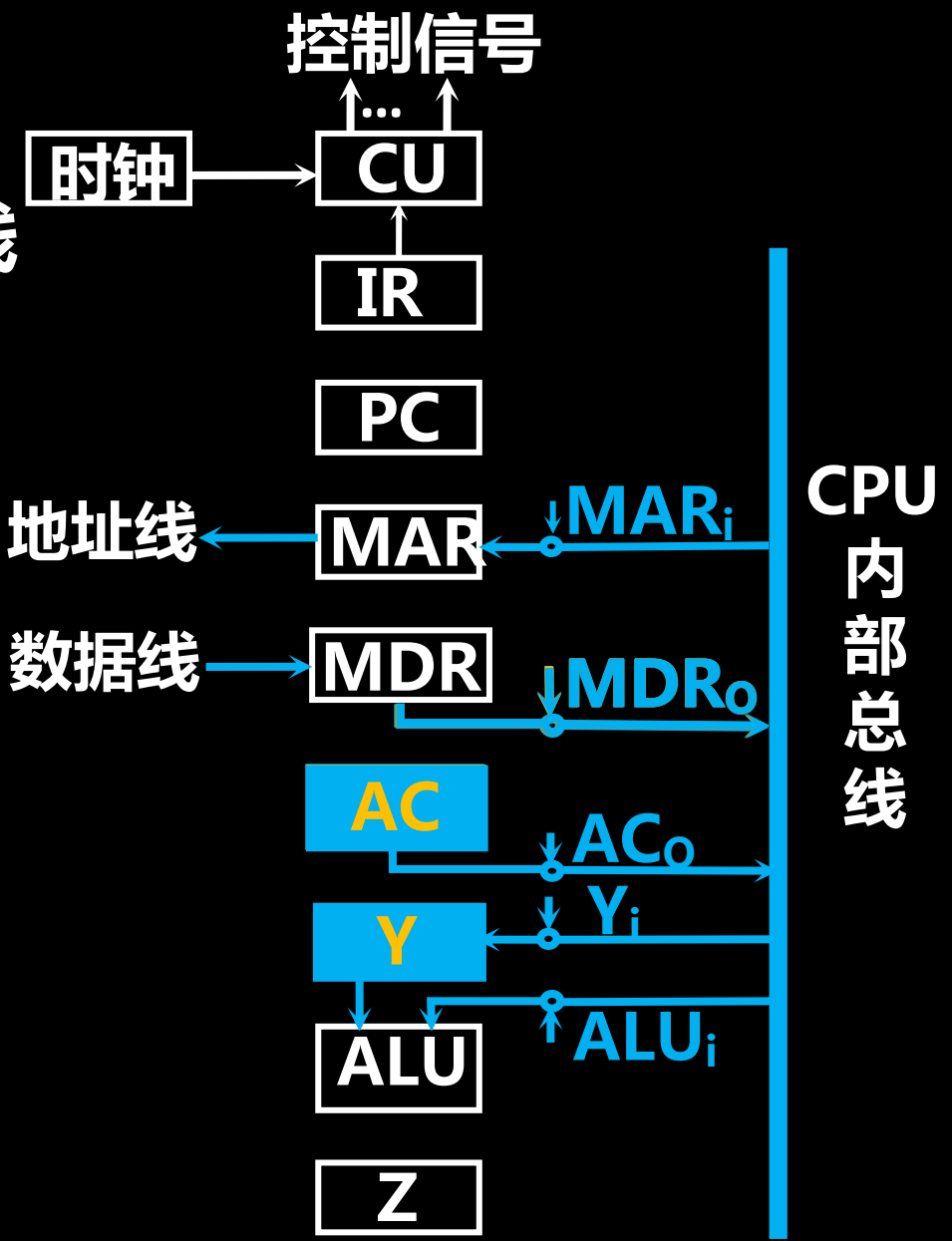
数据线⟶MDR

MDR⟶Y⟶ALU

MDR_o Y_i

AC ⟶ ALU

AC_o ALU_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

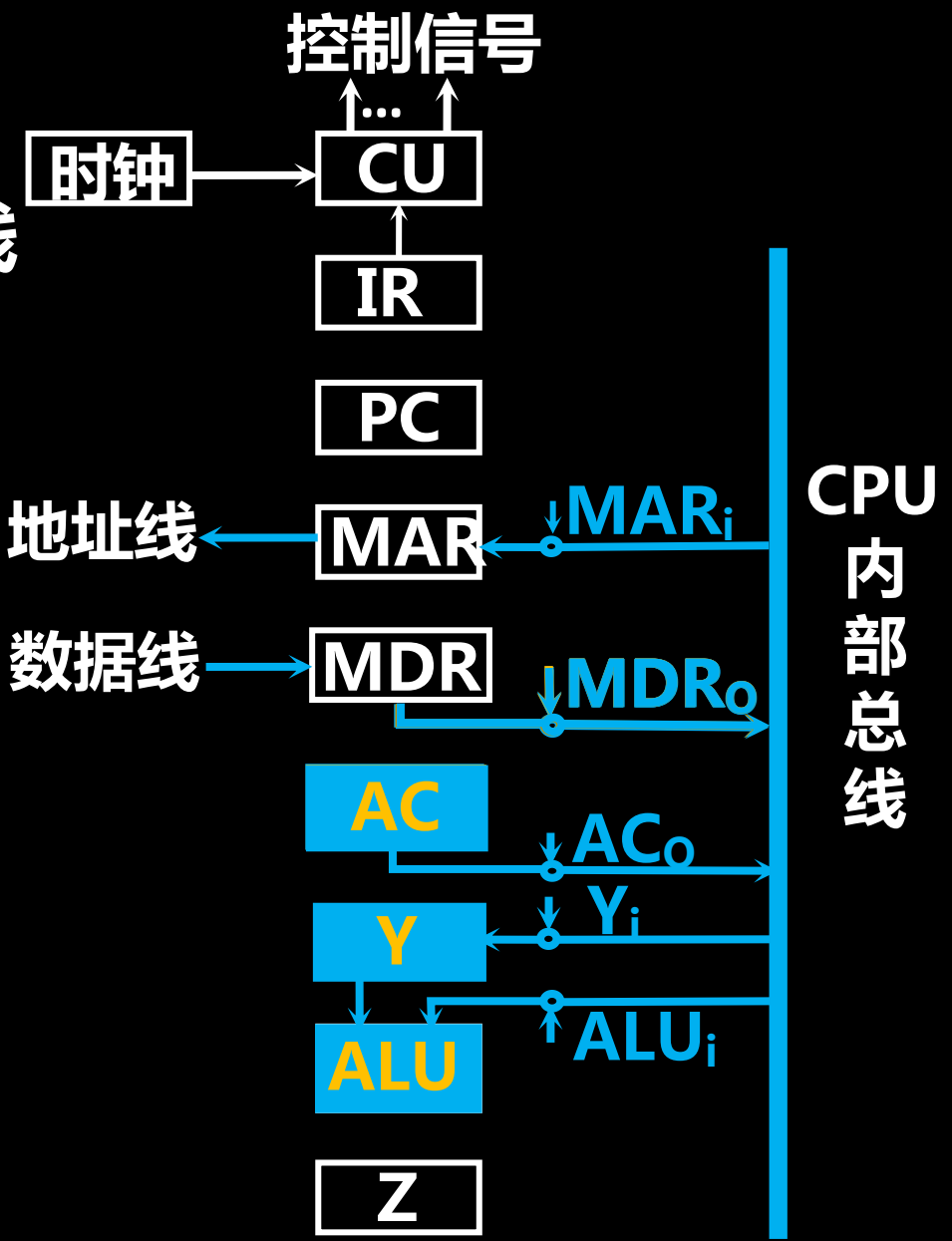
数据线⟶MDR

MDR⟶Y⟶ALU

MDR_o Y_i

AC ⟶ ALU

AC_o ALU_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

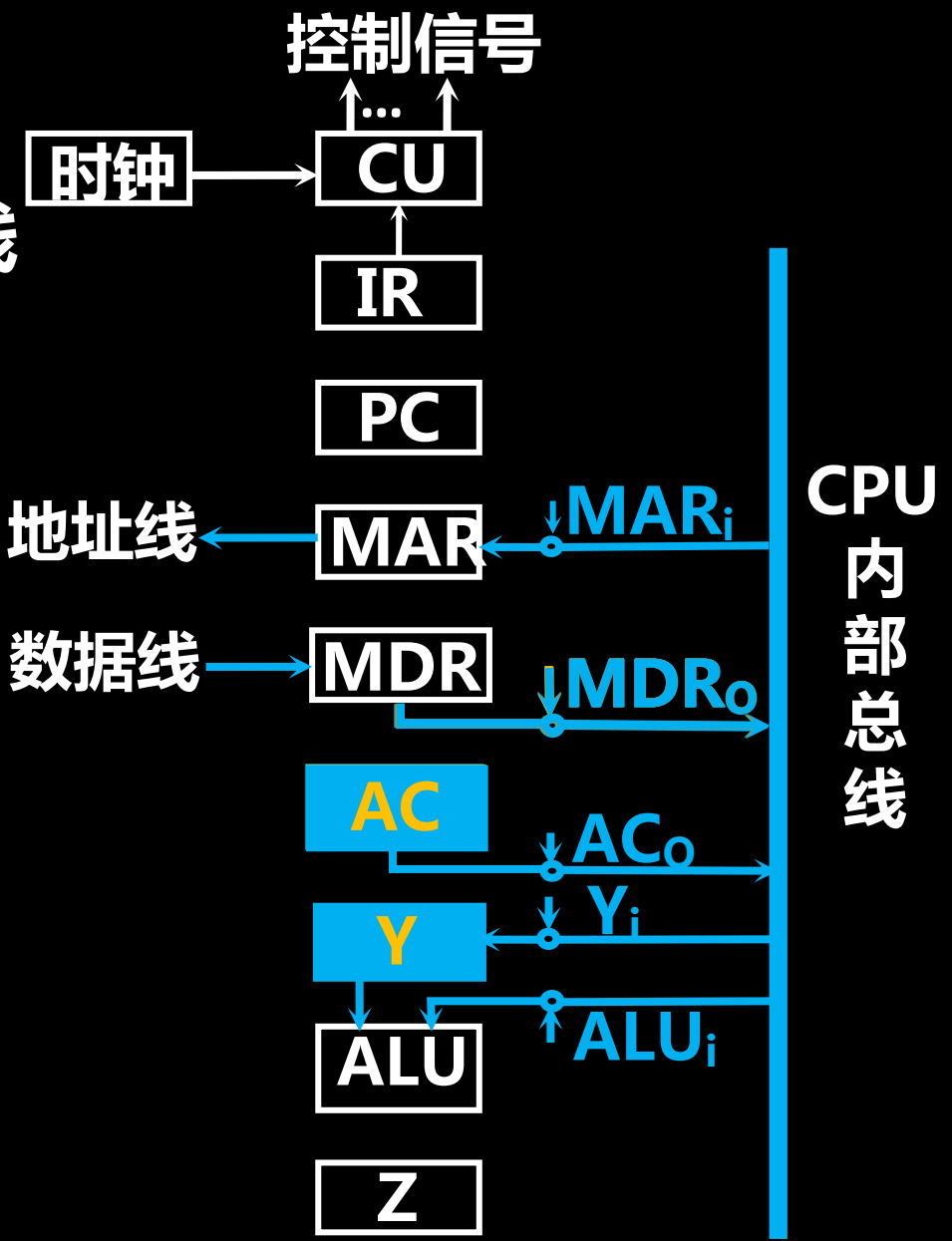
MDR⟶Y⟶ALU

MDR_o Y_i

AC ⟶ ALU

AC_o ALU_i

(AC)+(Y)



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

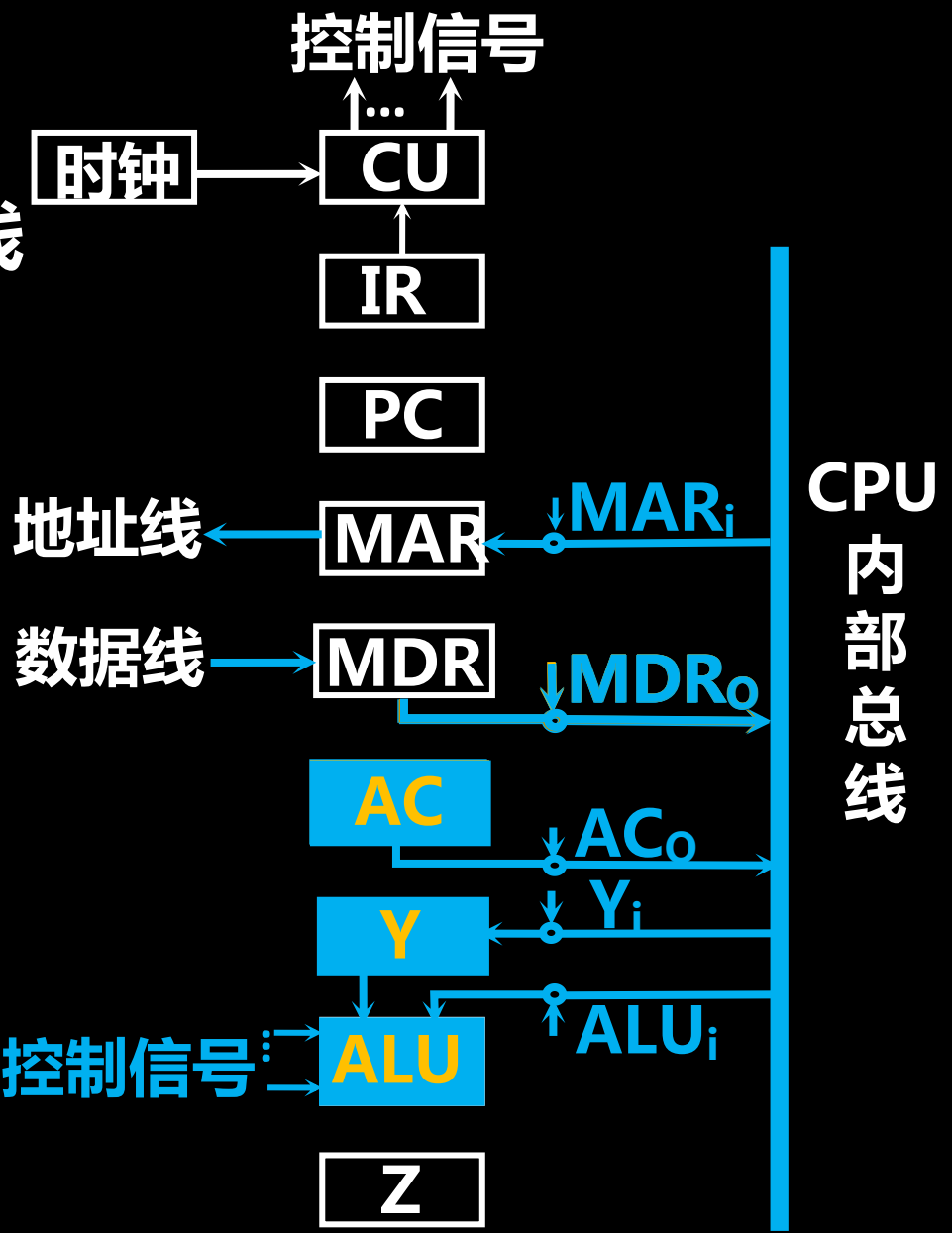
MDR⟶Y⟶ALU

MDR_o Y_i

AC ⟶ ALU

AC_o ALU_i

(AC)+(Y)



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

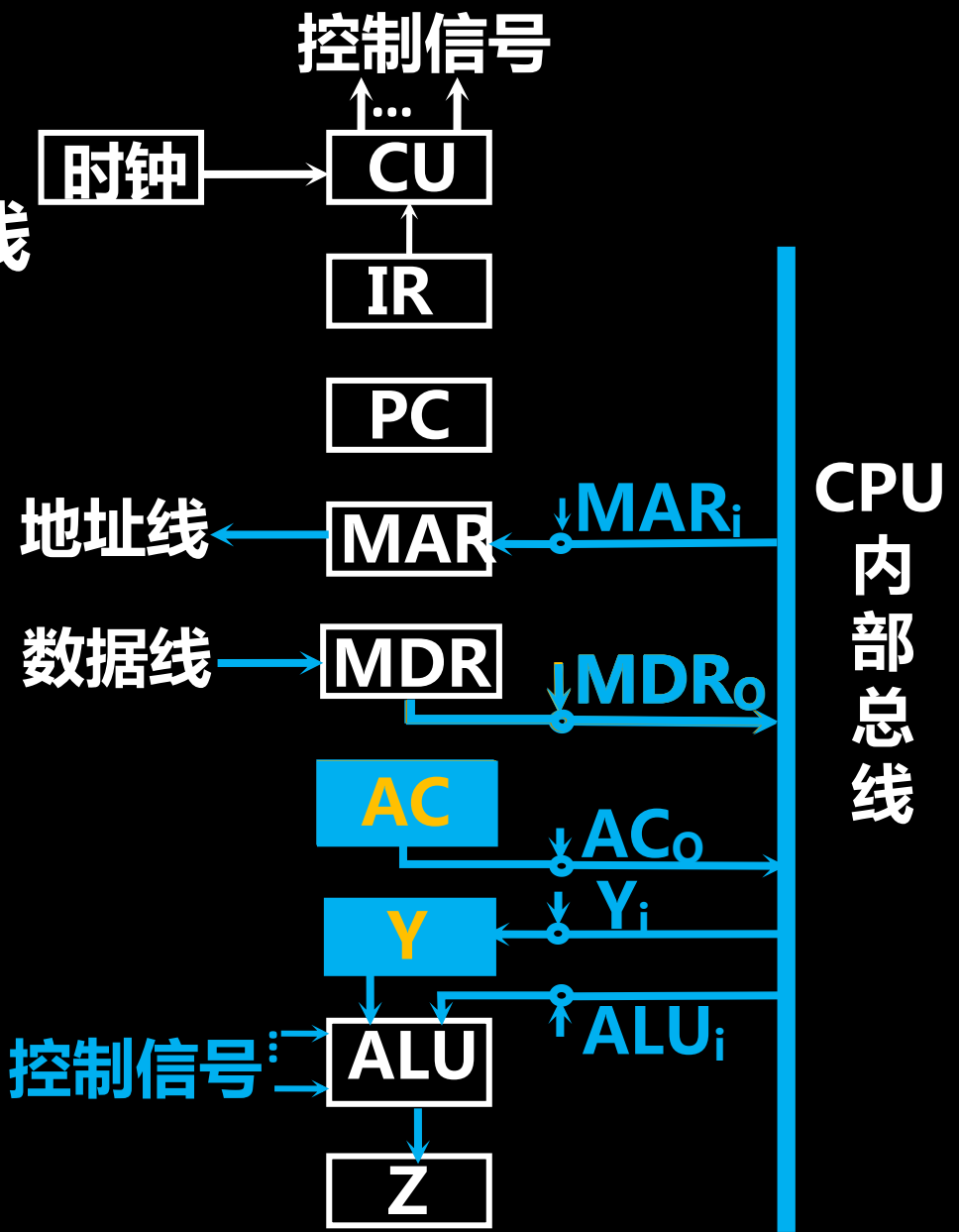
MDR⟶Y⟶ALU

MDR_o Y_i

AC ⟶ ALU

AC_o ALU_i

(AC)+(Y) ⟶



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

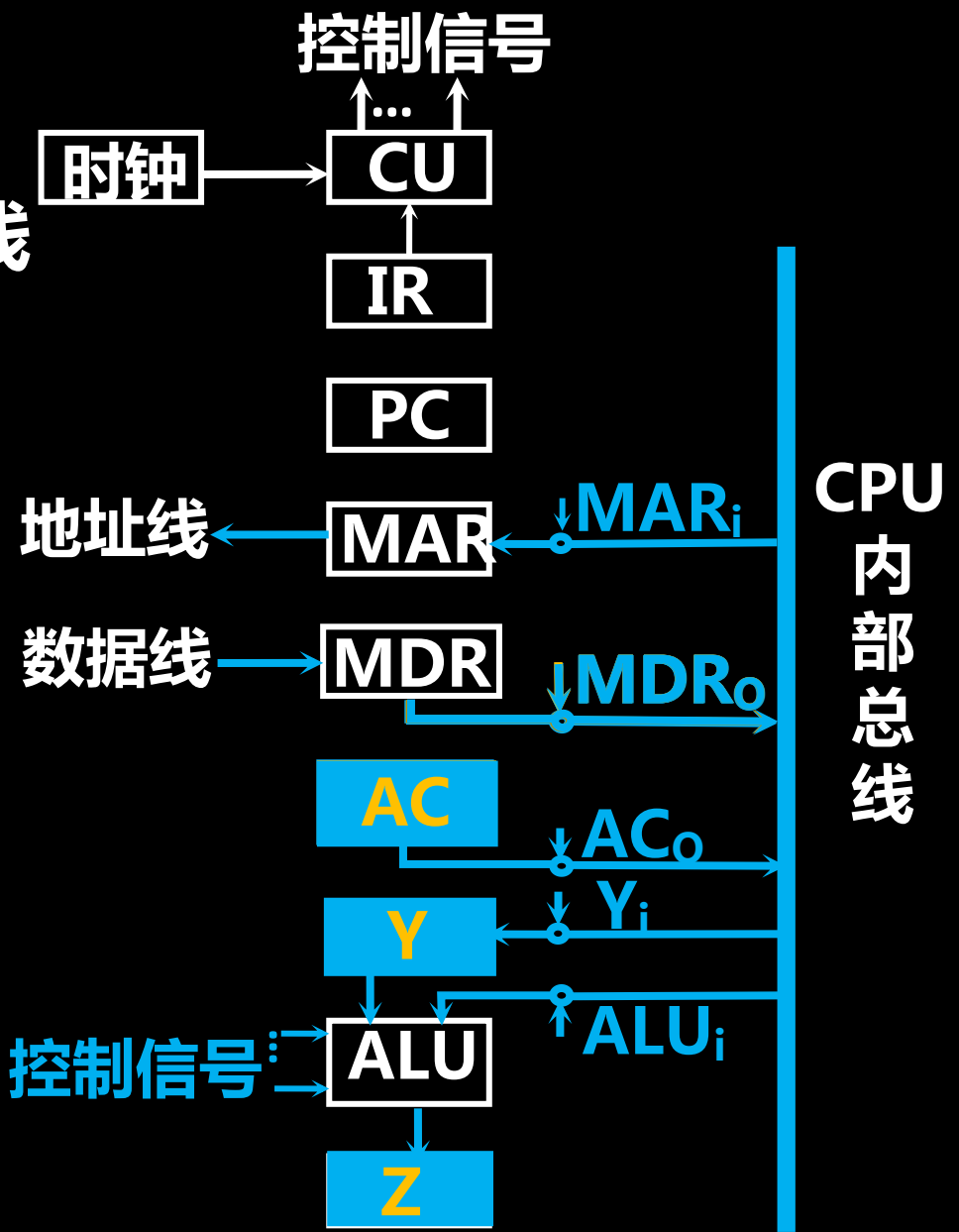
MDR⟶Y⟶ALU

MDR_o Y_i

AC ⟶ ALU

AC_o ALU_i

(AC)+(Y) ⟶ Z



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

MDR⟶Y⟶ALU

MDR_o Y_i

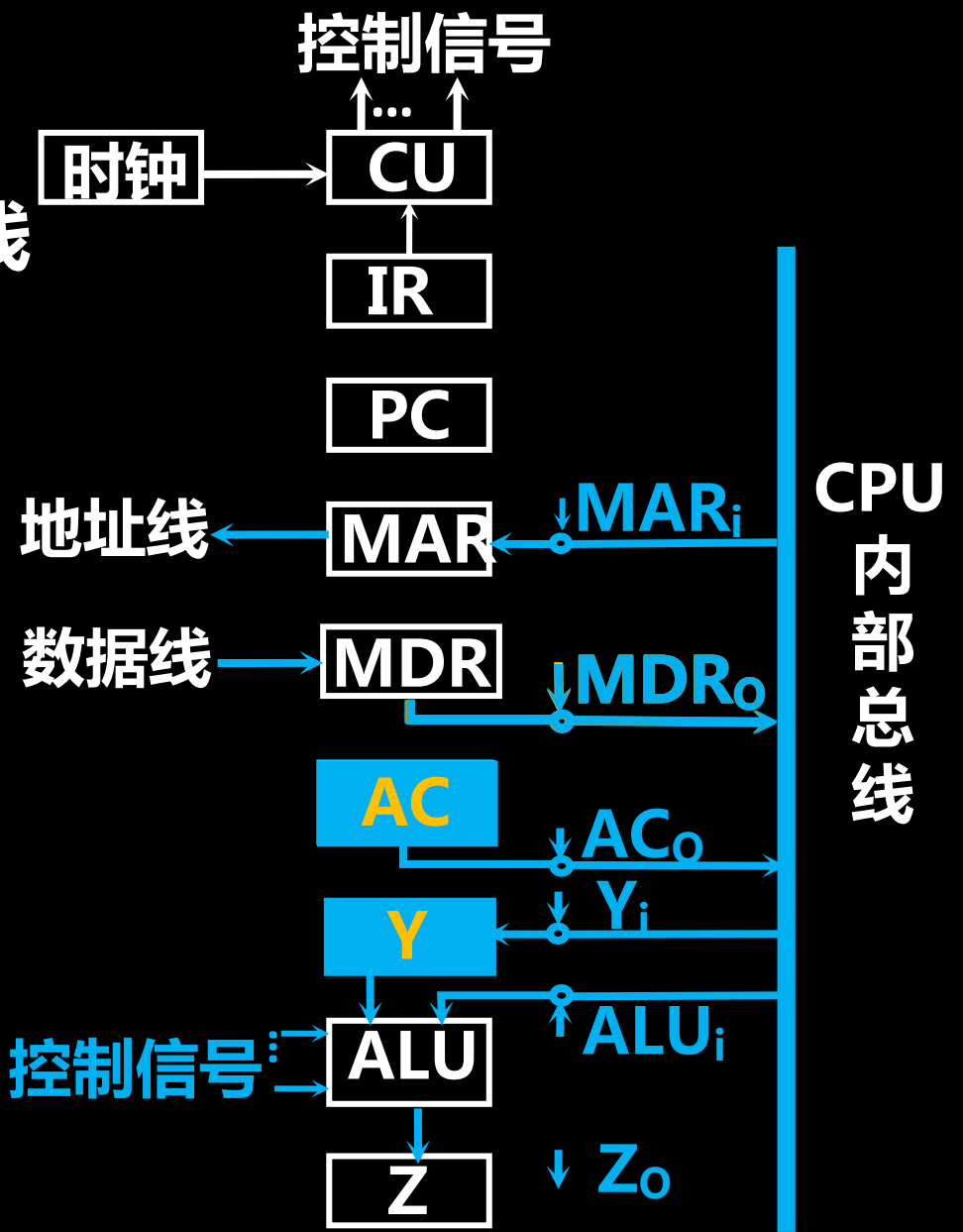
AC ⟶ ALU

AC_o ALU_i

(AC)+(Y) ⟶ Z

Z

Z_o



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

MDR⟶Y⟶ALU

MDR_o Y_i

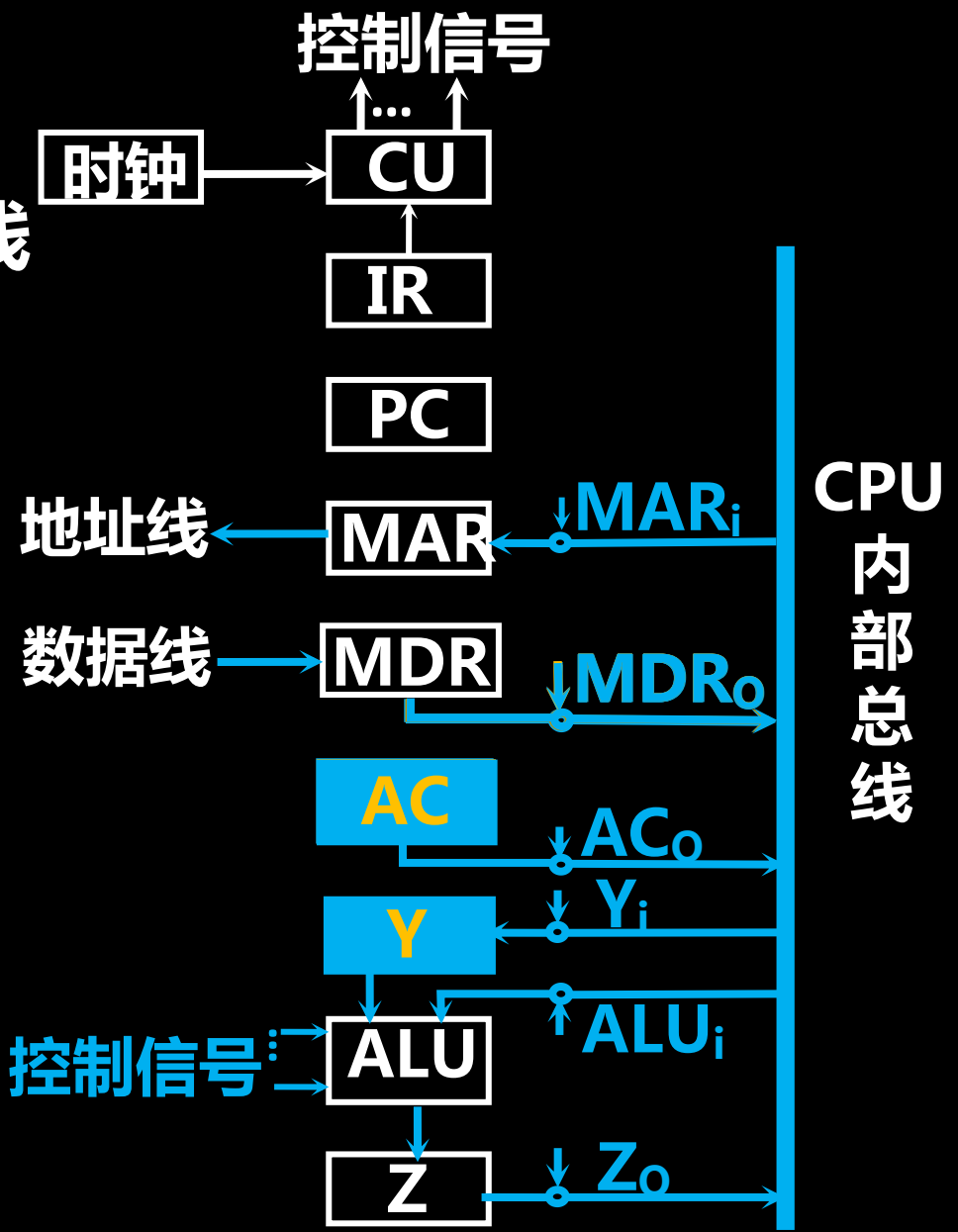
AC ⟶ ALU

AC_o ALU_i

(AC)+(Y) ⟶ Z

Z ⟶

Z_o



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

MDR⟶Y⟶ALU

MDR_o Y_i

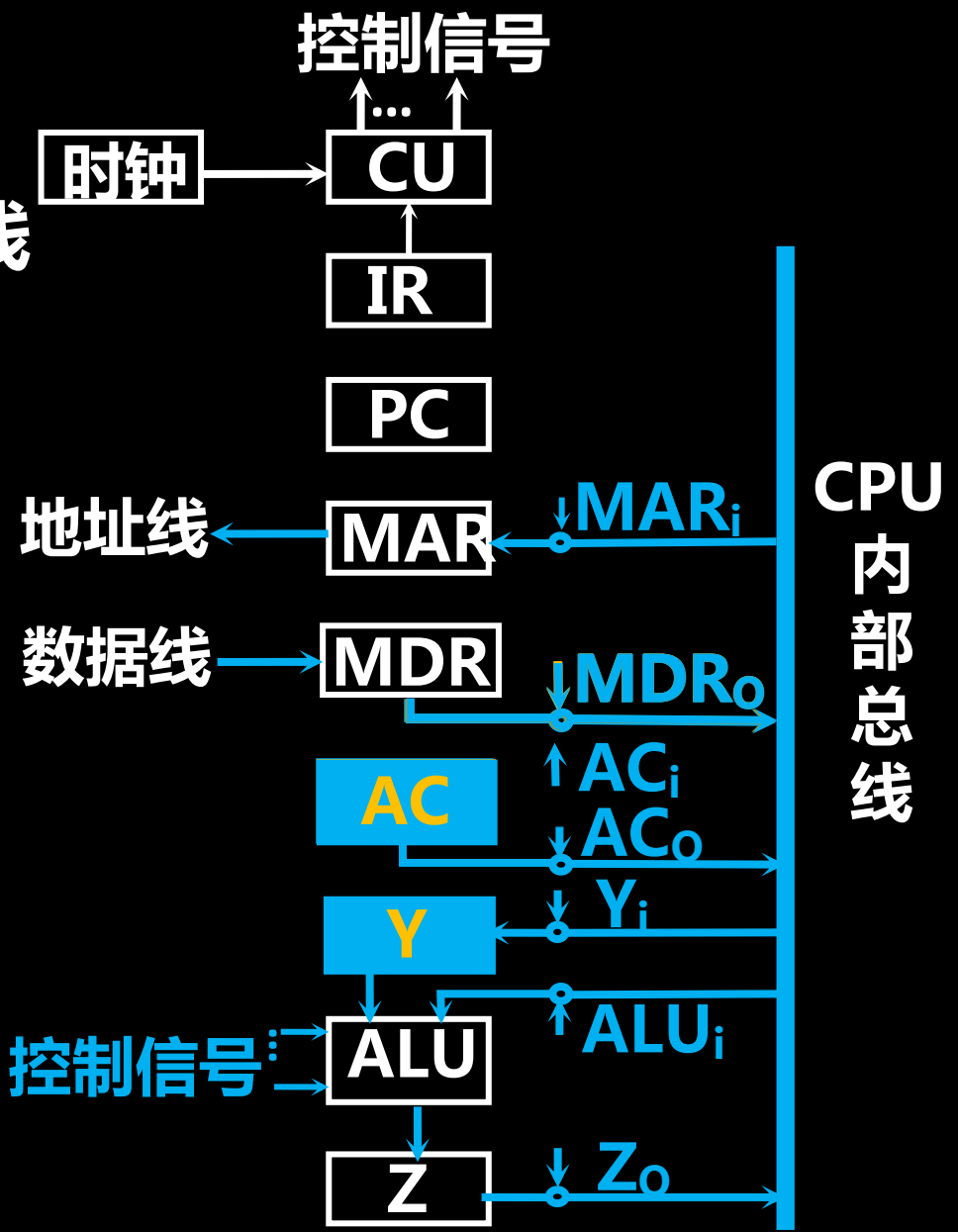
AC ⟶ ALU

AC_o ALU_i

(AC)+(Y) ⟶ Z

Z ⟶

Z_o AC_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

MDR⟶Y⟶ALU

MDR_o Y_i

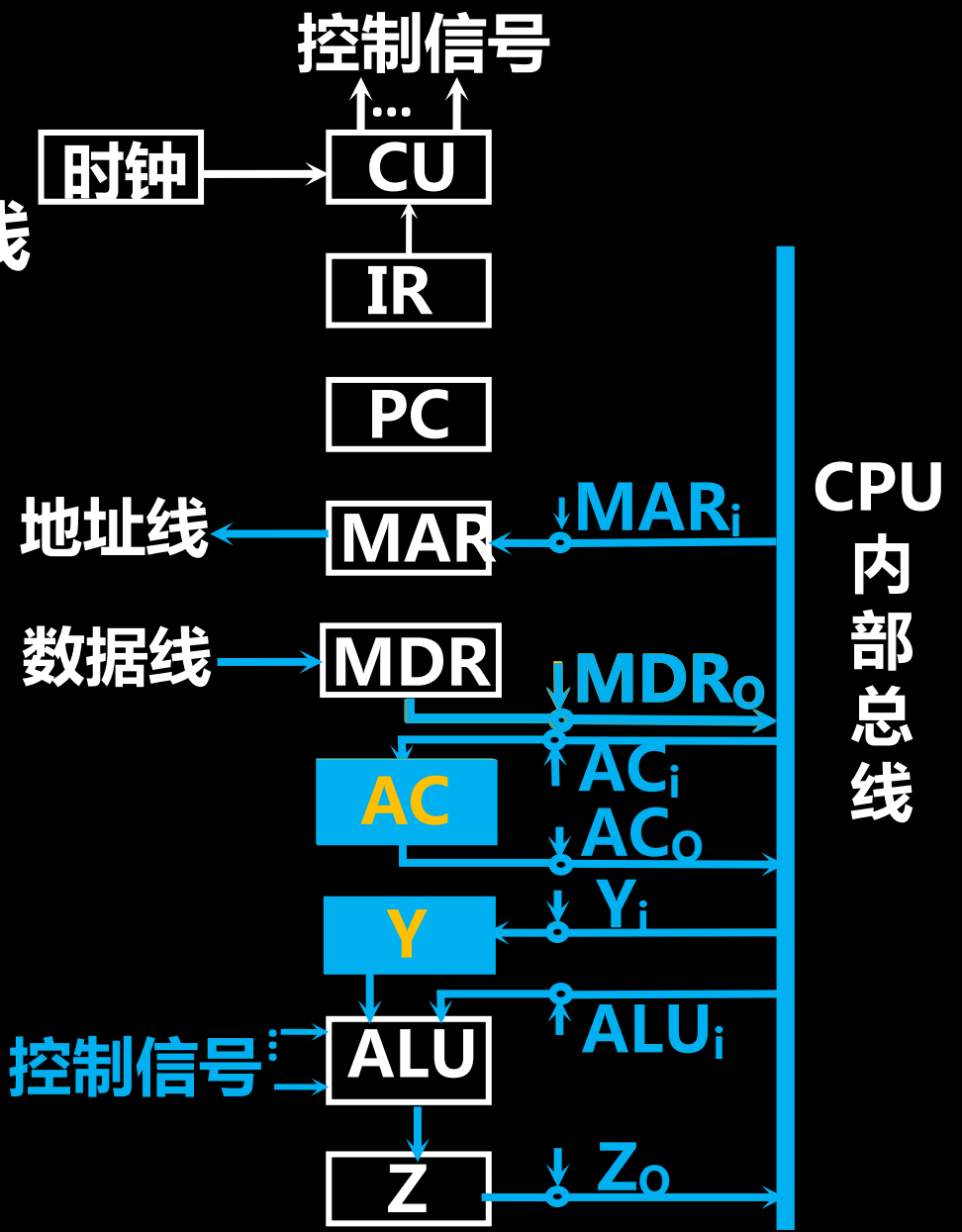
AC ⟶ ALU

AC_o ALU_i

(AC)+(Y) ⟶ Z

Z ⟶

Z_o AC_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

MDR⟶Y⟶ALU

MDR_o Y_i

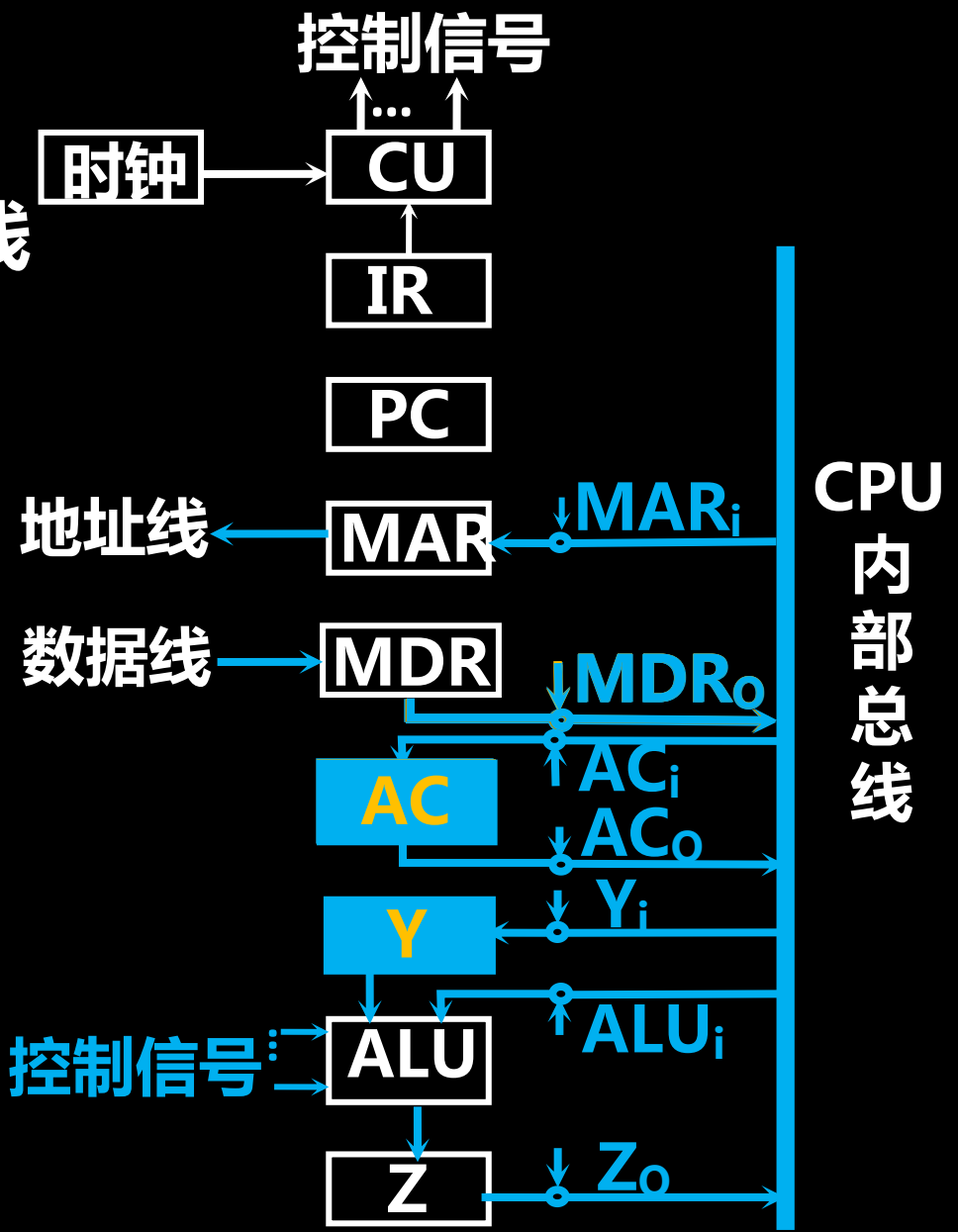
AC ⟶ ALU

AC_o ALU_i

(AC)+(Y) ⟶ Z

Z ⟶ AC

Z_o AC_i



采用CPU内部总线方式

(3) ADD @ X 执行周期

MDR⟶MAR⟶地址线

MDR_o MAR_i

1 ⟶ R

数据线⟶MDR

MDR⟶Y⟶ALU

MDR_o Y_i

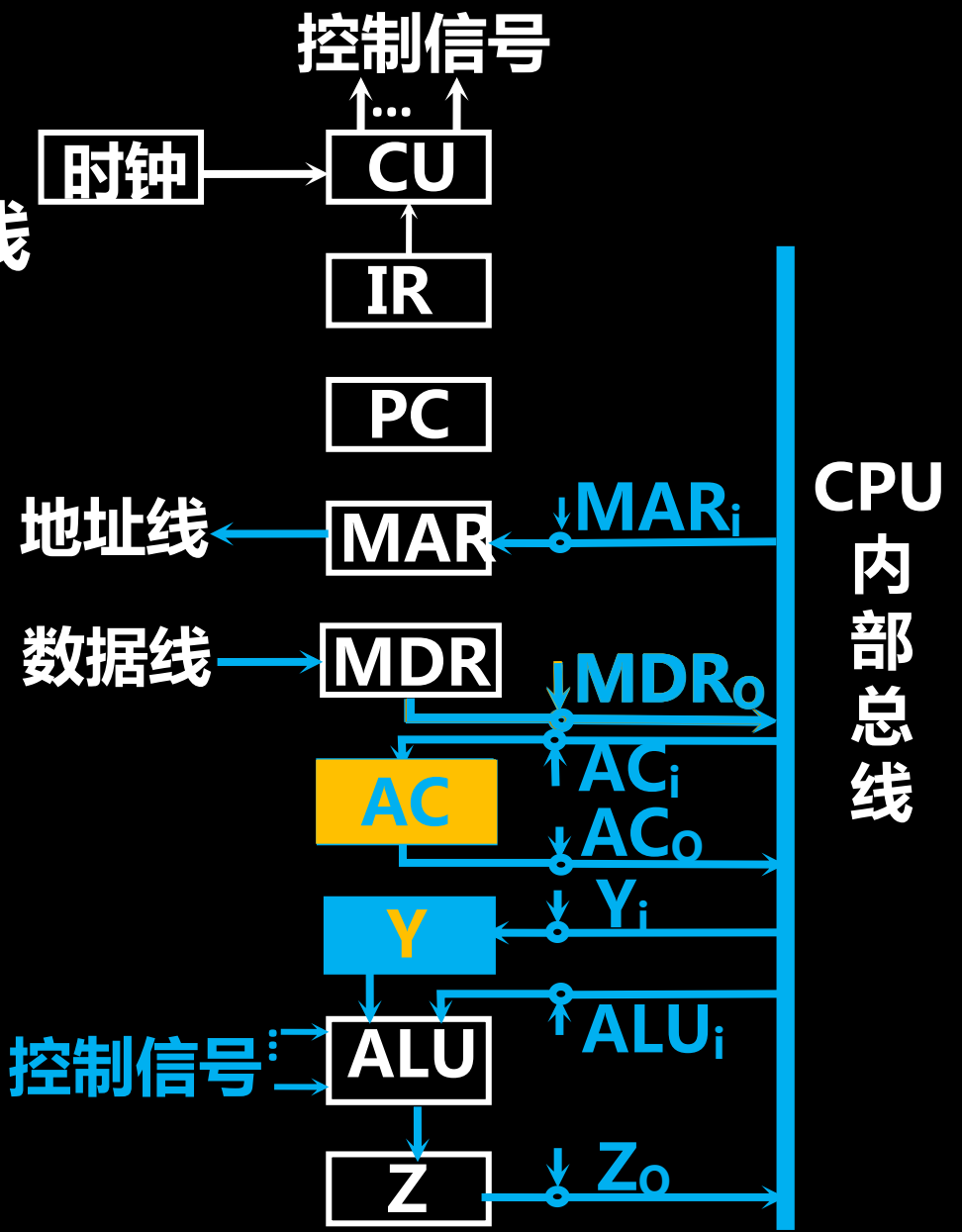
AC ⟶ ALU

AC_o ALU_i

(AC)+(Y) ⟶ Z

Z ⟶ AC

Z_o AC_i



控制方式

产生不同微操作命令序列所用的时序方式。

1. 同步控制方式

任一微操作均由**统一基准时标**的时序信号控制。

(1) 采用**定长**的机器周期

- 以**最长**的**微操作序列**和**最繁**的微操作作为**标准。**
- 机器周期内**节拍数相同。**

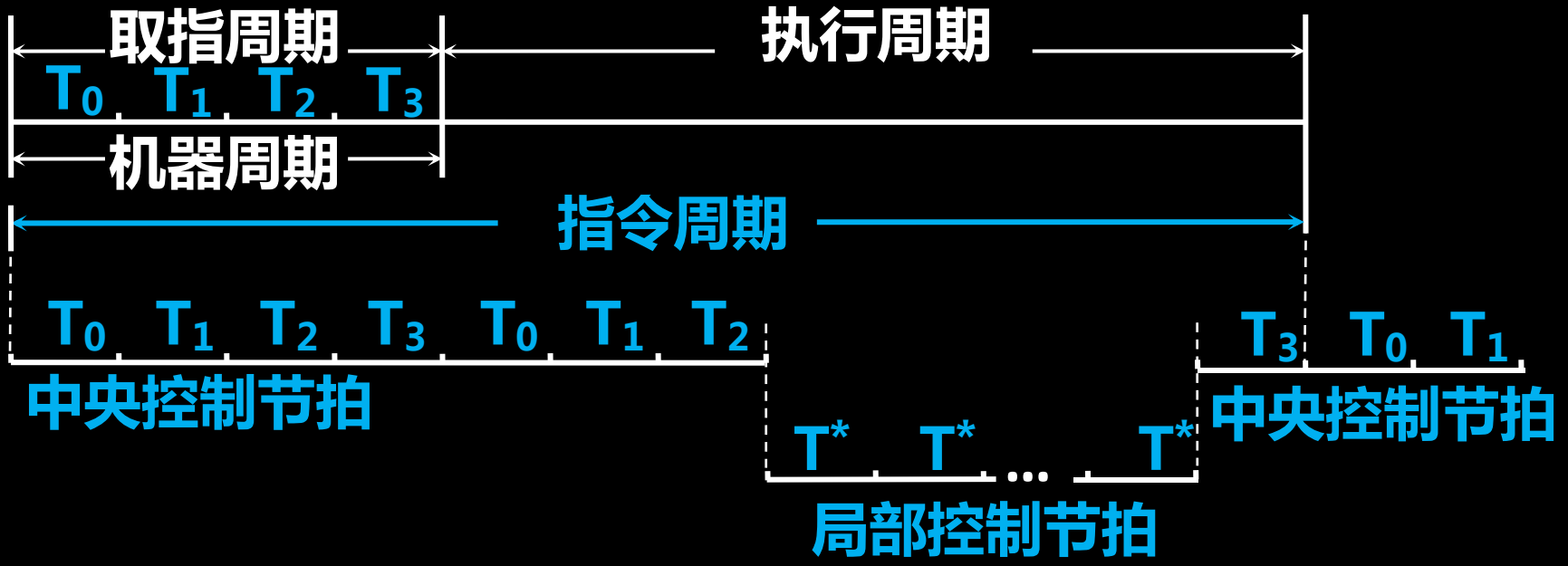
控制方式

(2) 采用不定长的机器周期

- 机器周期内**节拍数不等。**

控制方式

(3) 采用中央控制和局部控制相结合的方法。



局部控制的节拍宽度与
中央控制的节拍宽度一致

控制方式

2. 异步控制方式

- 无基准时标信号
- 无固定的周期节拍和严格的时钟同步
- 采用 应答方式

3. 联合控制方式

同步与异步结合

控制方式

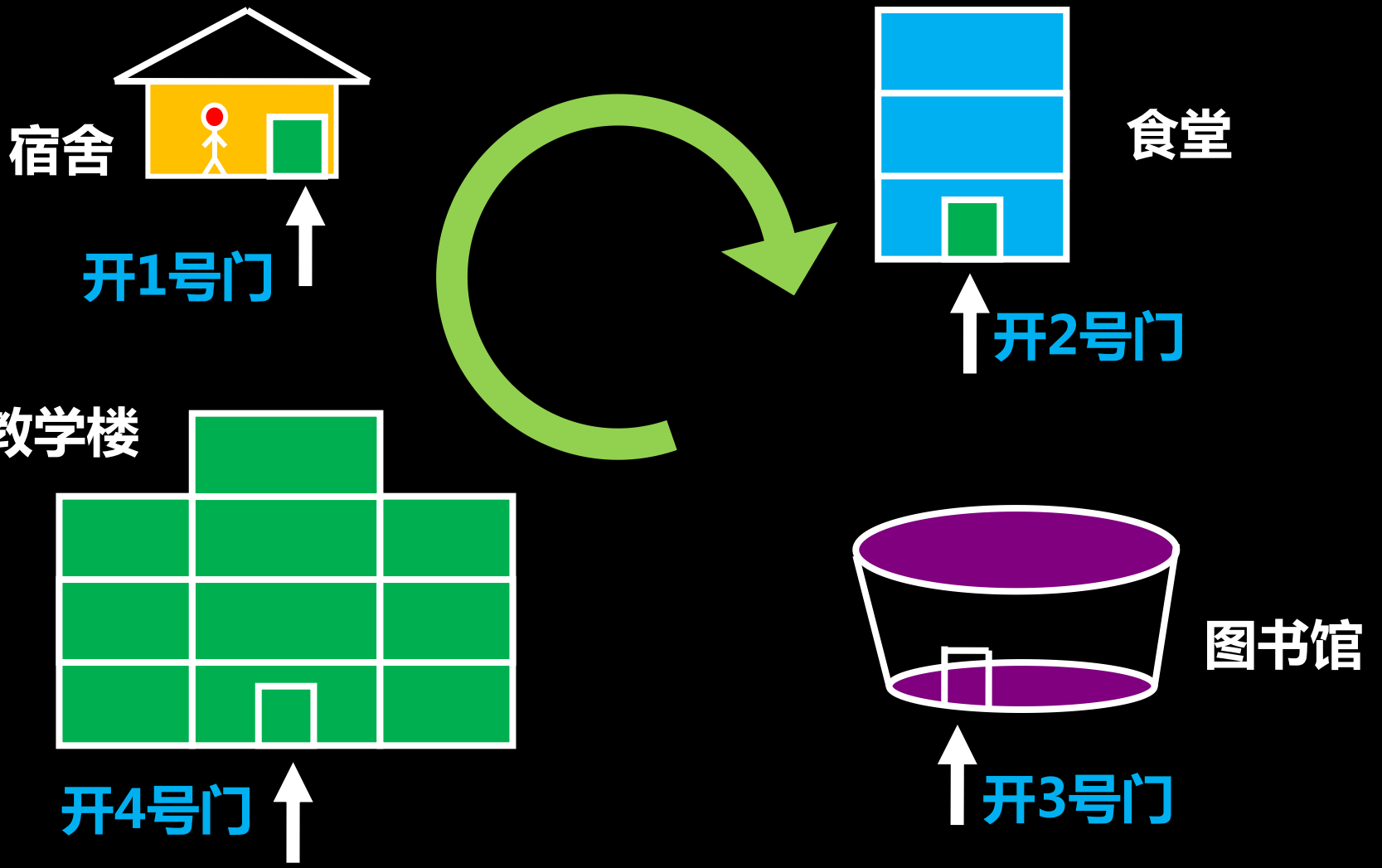
4. 人工控制方式

(1) Reset

(2) 连续和单条指令执行转换开关

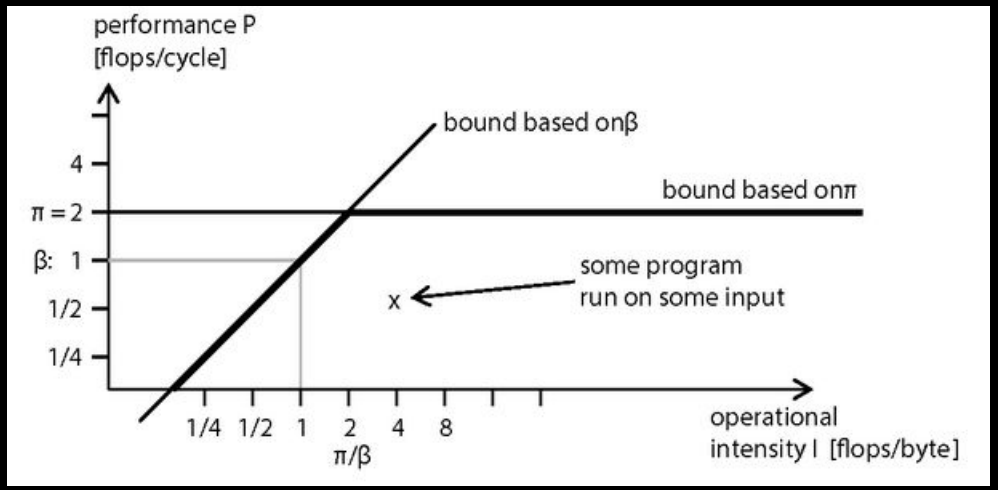
(3) 符合停机开关

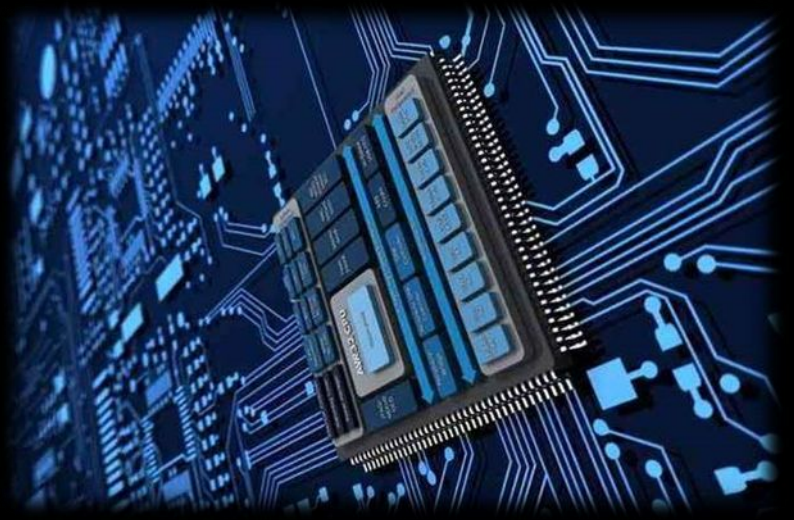
指令执行的实质：一天的生活



推荐阅读：屋顶模型

- Williams和Patterson于2008年发表的一篇论文中提出了屋顶模型（Roofline）
- 屋顶模型，可以将浮点性能、算术密度和存储性能联系在一个二维图中。
- 屋顶模型给出了处理器性能的上界，如果你的程序远低于该上界，那么应该如何优化呢？





流水线技术

大连理工大学 赖晓晨

流水线技术的由来

在英格兰北部的一个小镇里，艾薇开了一个小店，卖鱼和油煎土豆片。艾薇店里的油煎土豆片是小镇中最好的，在每个集市日中午的时候，长长的队伍都会排出商店。

后来他们把柜台加长，艾薇、伯特、狄俄尼索斯和玛丽站成一排。顾客进来的时候，艾薇先给他们一个盛着鱼的盘子，然后伯特给加上油煎土豆片，狄俄尼索斯再给盛上豌豆糊，最后玛丽倒茶并收钱。

顾客们不停的走动；当一个顾客拿到豌豆糊的同时，他后面的已经拿到了油煎土豆片，再后面的一个已经拿到了鱼。

提高机器速度的方法

如何提高机器速度

1. 提高访存速度

   高速芯片　　Cache　　多体并行

2. 提高I/O和主机之间的传送速度

   中断　　DMA　　通道　　I/O处理机

   多总线

3. 提高运算器速度

   高速芯片　　改进算法　　快速进位链

- 总之，提高整机处理能力

  高速器件　　改进系统结构，挖掘系统的并行性

系统的并行性

1. 并行的概念

并行
- **并发** 两个或两个以上事件在**同一时间段**发生
- **同时** 两个或两个以上事件在**同一时刻**发生

时间上互相重叠

2. 并行性的等级

过程级（进程）	**粗粒度**	软件实现
指令级（指令）	**细粒度**	硬件实现

指令流水原理

1. 指令的串行执行

取指令　　取指令部件　　完成

执行指令　　执行指令部件　　完成

总有一个部件 空闲

指令流水原理

2. 指令的二级流水

指令预取

若取指和执行阶段时间上完全重叠

指令周期减半　速度提高1倍

指令的六级流水

设指令的执行分为六个阶段：

(1) 取指

(2) 译码

(3) 计算操作数地址

(4) 取数

(5) 执行

(6) 写回

指令的六级流水

取指、译码、计算操作数地址、取数、执行，写回

六级流水，14个时间单位，完成9条指令，第6个时间单位之后，每个时间单位完成一条指令。

指令流水原理

3. 影响指令流水效率加倍的因素

(1) 执行时间 > 取指时间

(2) 条件转移指令对指令流水的影响

必须等上条指令执行结束，才能确定下条指令的地址，造成时间损失。

影响指令流水效率的因素主要是各种相关问题

流水线性能

1. 吞吐量

- 单位时间内流水线所完成指令或输出结果的数量
- 设 m 级的流水线，各段时间为Δt
  - 最大吞吐率

$$T_{pmax} = \frac{1}{\Delta t}$$

  - 实际吞吐率（未考虑相关问题）

连续处理 n 条指令的吞吐率为

$$T_p = \frac{n}{m \cdot \Delta t + (n-1) \cdot \Delta t}$$

流水线性能

2. 加速比 Sp

- m 段的流水线的速度与等功能的非流水线的速度之比
- 设流水线各段时间为 Δt

完成n条指令在m段流水线上共需

$$T = m \cdot \Delta t + (n-1) \cdot \Delta t$$

完成n条指令在等效的非流水线上共需

$$T' = n(m \cdot \Delta t)$$

则

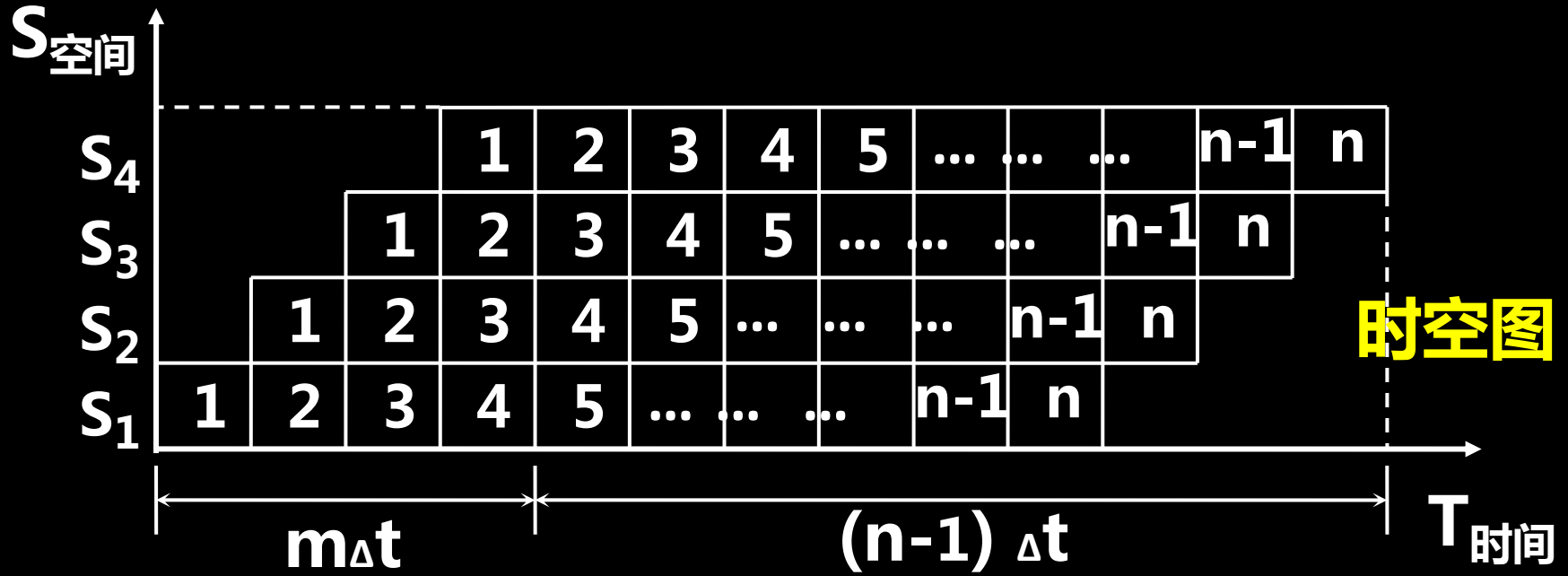
$$S_p = \frac{nm \cdot \Delta t}{m \cdot \Delta t + (n-1) \cdot \Delta t} = \frac{nm}{m + n - 1}$$

流水线性能

3. 效率

- 流水线中各功能段的**利用率**。
- 由于流水线有**建立时间**和**排空时间**。

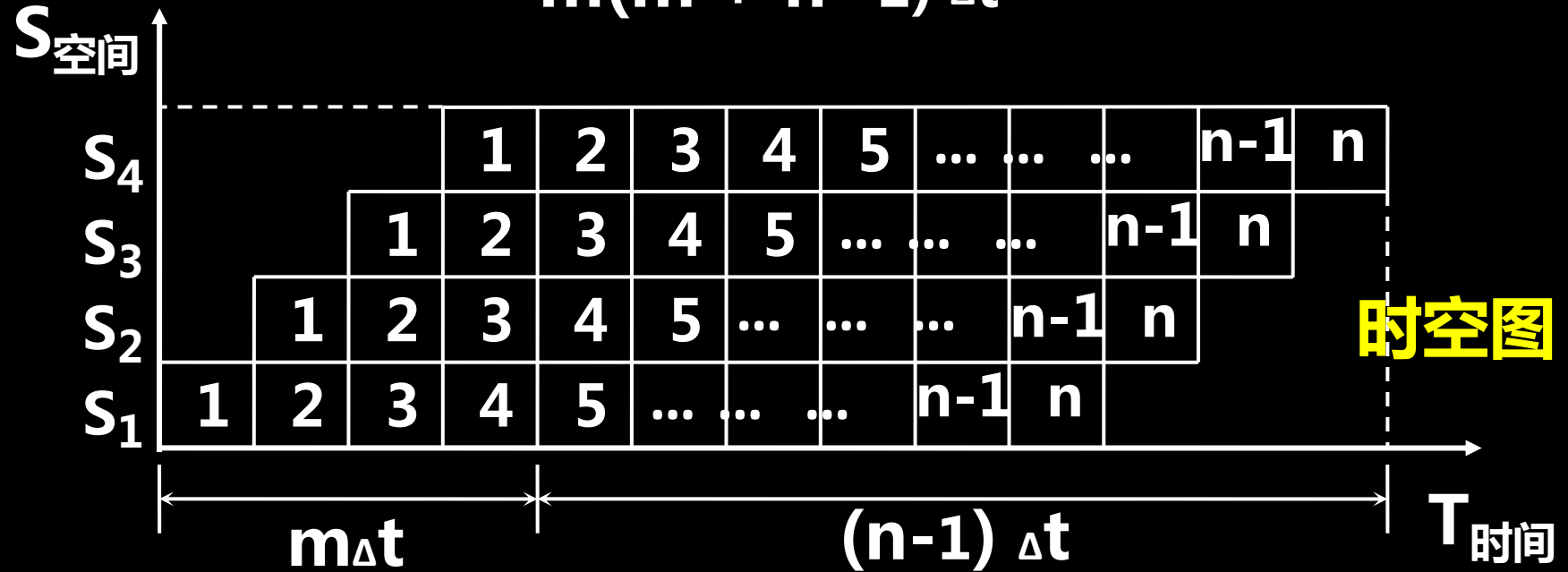
因此各功能段的设备**不可能一直**处于**工作**状态。



流水线性能

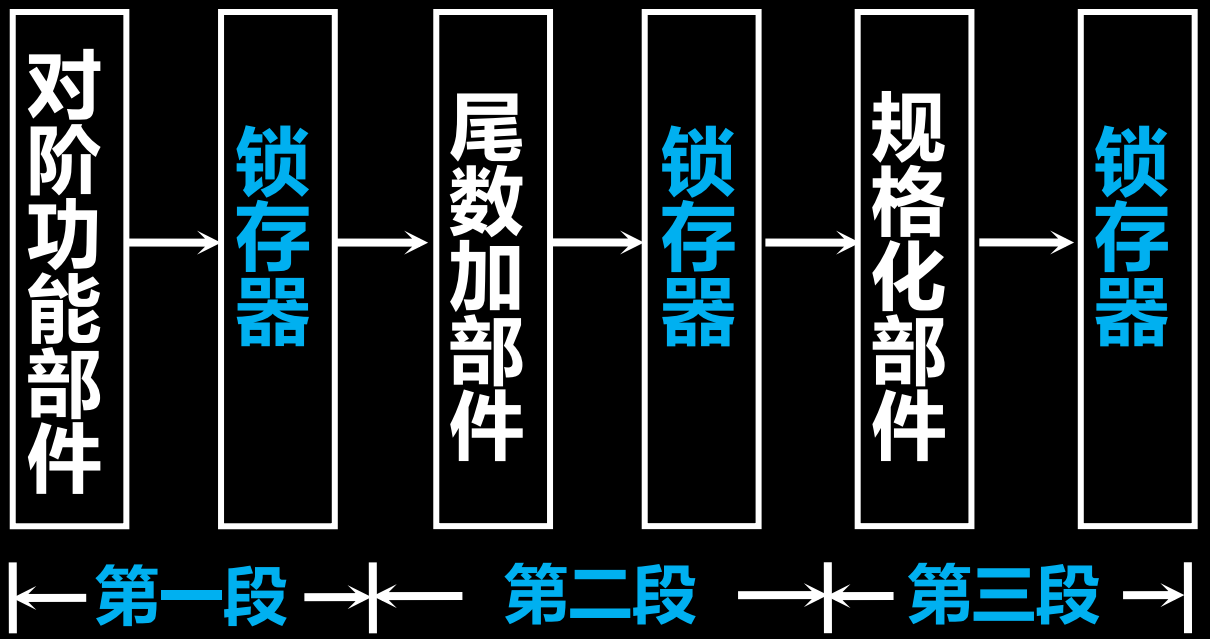
3. 效率

$$\text{效率} = \frac{\text{流水线各段处于工作时间的时空区}}{\text{流水线中各段总的时空区}} = \frac{mn\Delta t}{m(m+n-1)\Delta t}$$



运算流水线结构

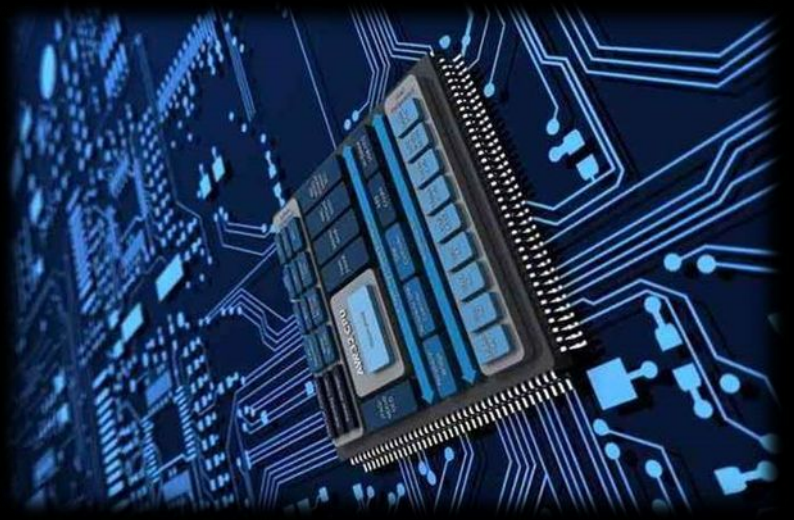
完成浮点加减运算可分对阶、尾数求和、规格化三段。



分段原则：每段操作时间尽量一致。

推荐阅读：指令静态调度

- 静态指令调度是指编译程序通过调整指令的顺序来减少流水线的停顿。
- 转移槽技术
- 更多内容。。。。



流水线的相关问题

大连理工大学 赖晓晨

“胶水”处理器——“真”双核 vs “假”双核

- 2005年，AMD提出“真”双核说法；
- AMD频频邀请Intel “决战双核”；
- 2006年，Intel反击并称自己首创双核概念；
- “假双核”指的是intel的Pentium D，由两个pentium 4组成，被网友称为“胶水”处理器；

指令的六级流水

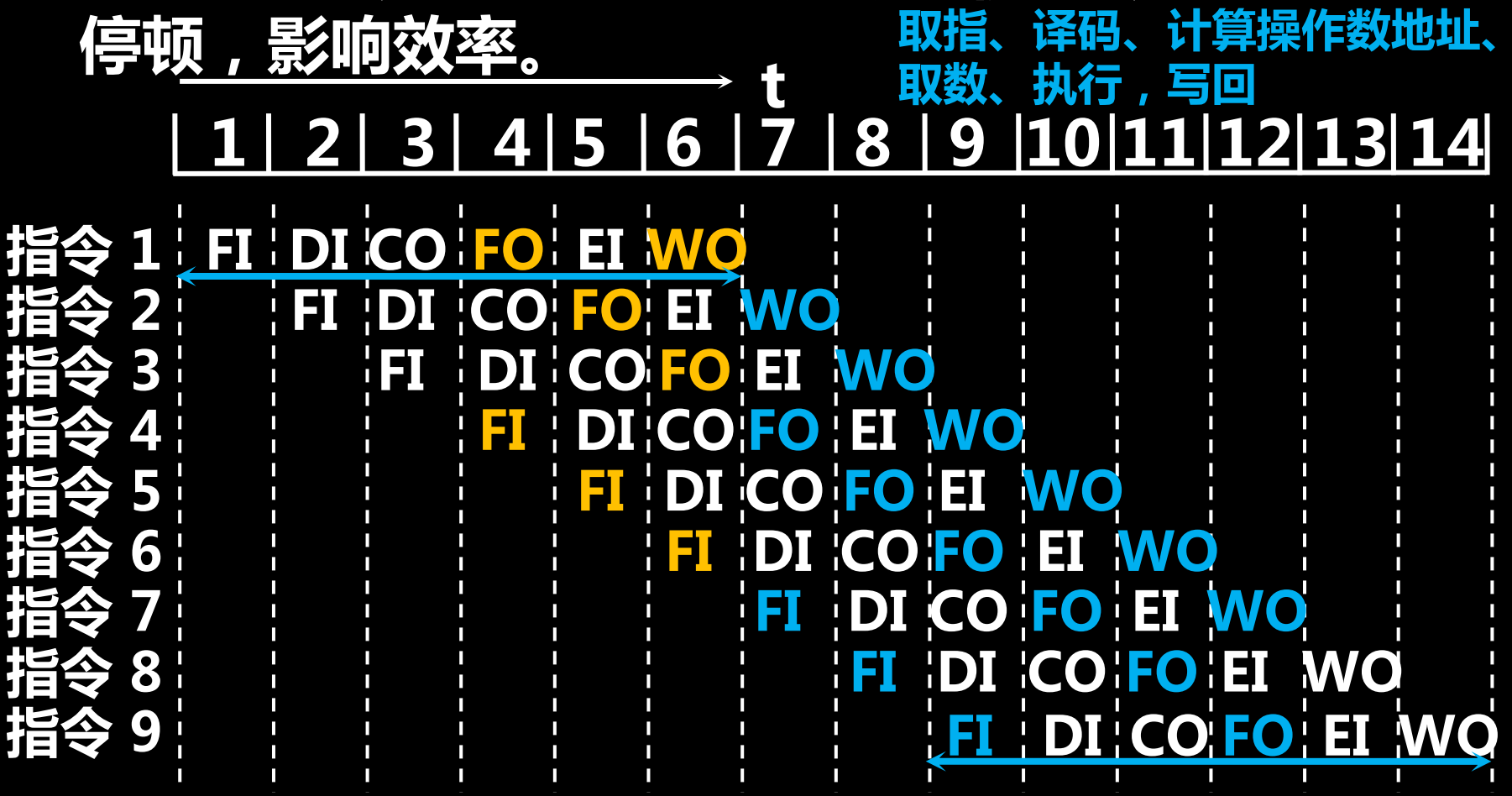
取指、译码、计算操作数地址、取数、执行，写回

完成 一条指令，6个时间单位；

六级流水，完成9条指令 14个时间单位；

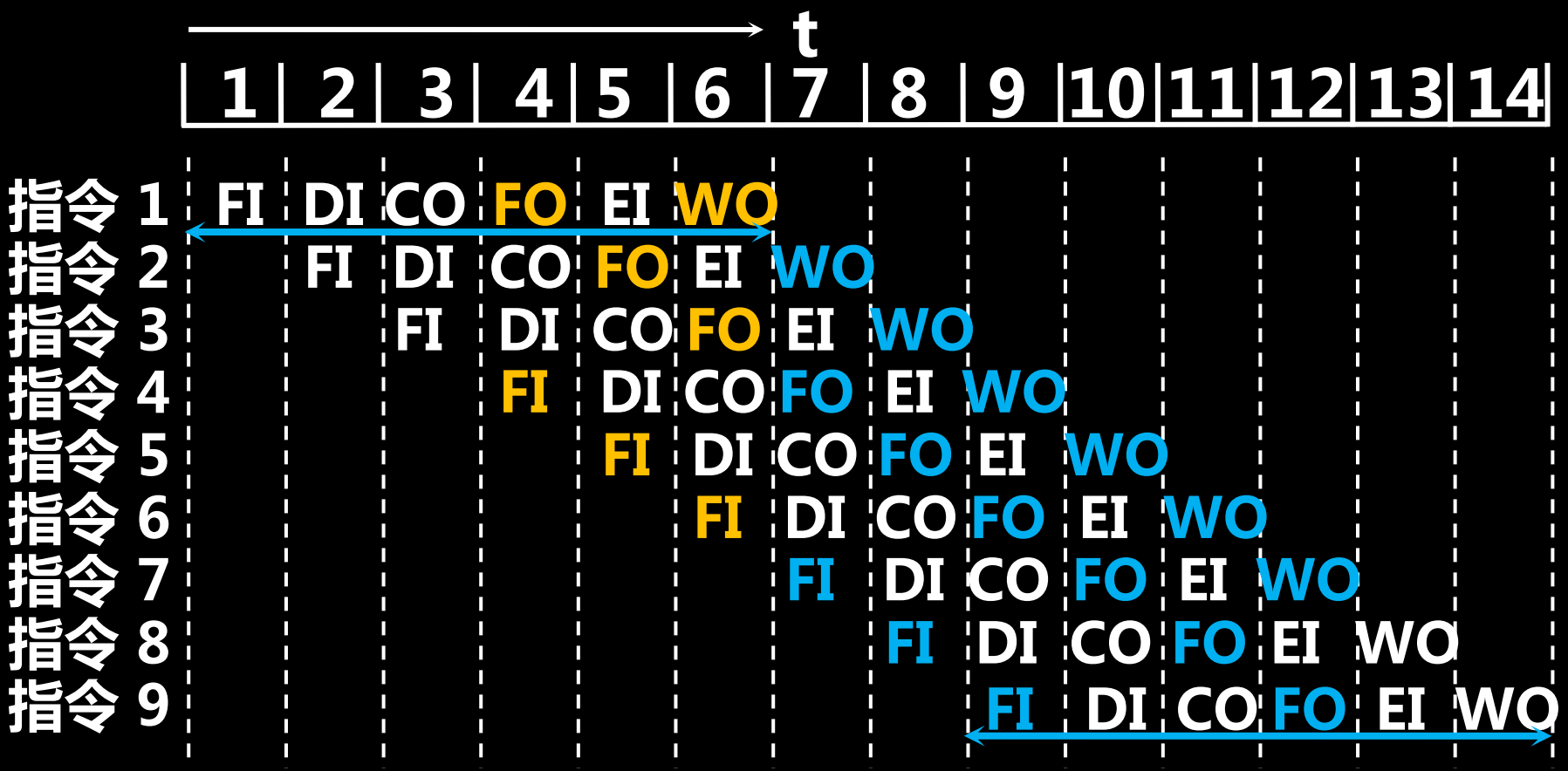
流水线的相关问题

相近指令之间出现某种关联，使指令流水出现停顿，影响效率。

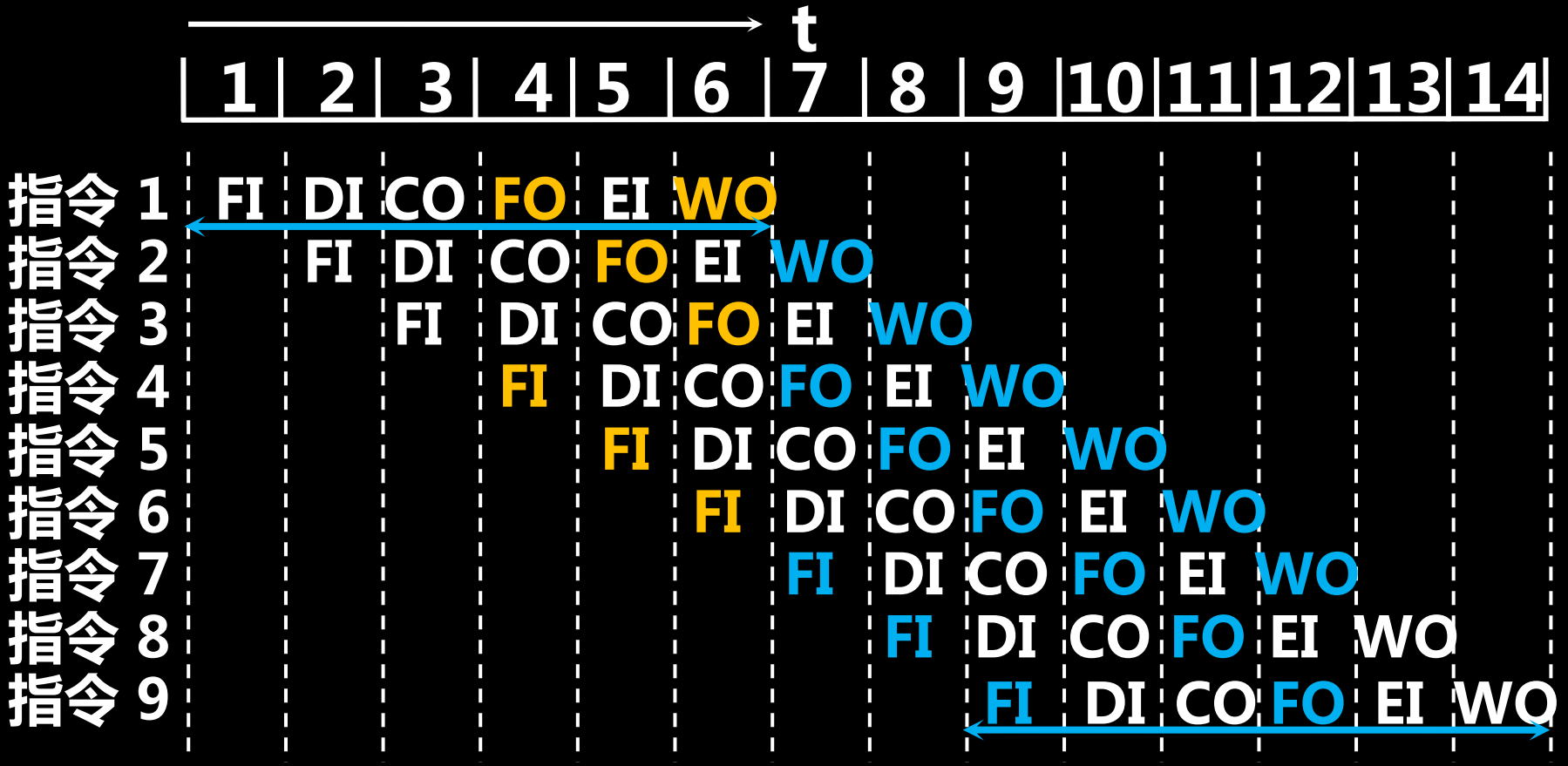


1. 结构相关

不同指令争用同一功能部件产生资源冲突。



结构相关的解决办法



解决办法

- 后续指令停顿
- 指令存储器和数据存储器分开，多部件
- 指令预取技术（适用于访存周期短的情况）

2. 数据相关

不同指令因重叠操作，可能改变操作数的读/写访问顺序，数据相关分为3种类型：

(1) 写后读相关（RAW）

SUB R_1，R_2，R_3 ；（R_2）–（R_3）→ R_1

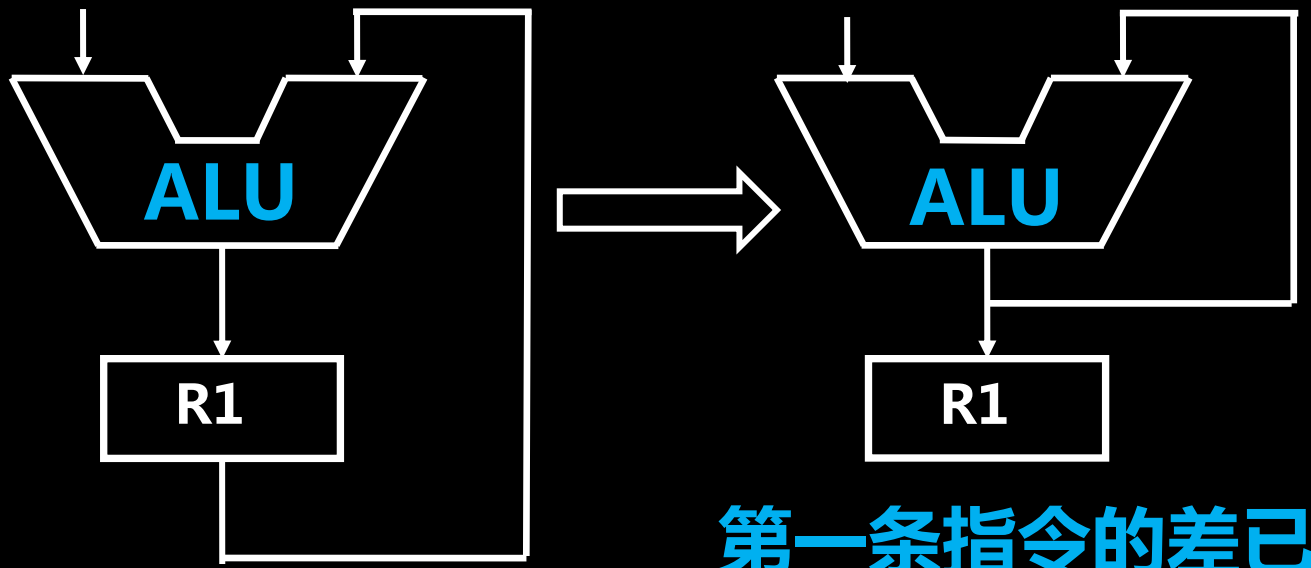
ADD R_4，R_5，R_1 ；（R_5）+（R_1）→ R_4

取指	译码	取数	执行	写回	
	取指	译码	取数	执行	写回

数据相关的解决办法

SUB R_1，R_2，R_3 ；（R_2）－（R_3）→ R_1

ADD R_4，R_5，R_1 ；（R_5）＋（R_1）→ R_4



旁路技术

第一条指令的差已经计算出，
从ALU的输出端口直接取数

取指	译码	取数	执行	写回	
	取指	译码	取数	执行	写回

其他两种数据相关

(2) 读后写相关（WAR）

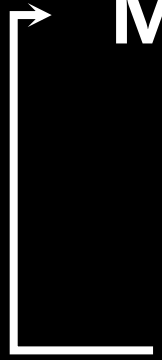
(3) 写后写相关（WAW）

指令乱序执行

3. 控制相关

由转移指令引起

```
        LDA     # 0
        LDX     # 0
M：     ADD     X, D
        INX
        CPX     # N
        BNE     M
        DIV     # N
        STA     ANS
```



DIV 指令必须等

BNE 指令的结果

才能确定是否做除法

控制相关分析

设指令3是条件转移指令，等到指令2执行完毕后才知道转移地址。

怎么办？

控制相关的解决办法

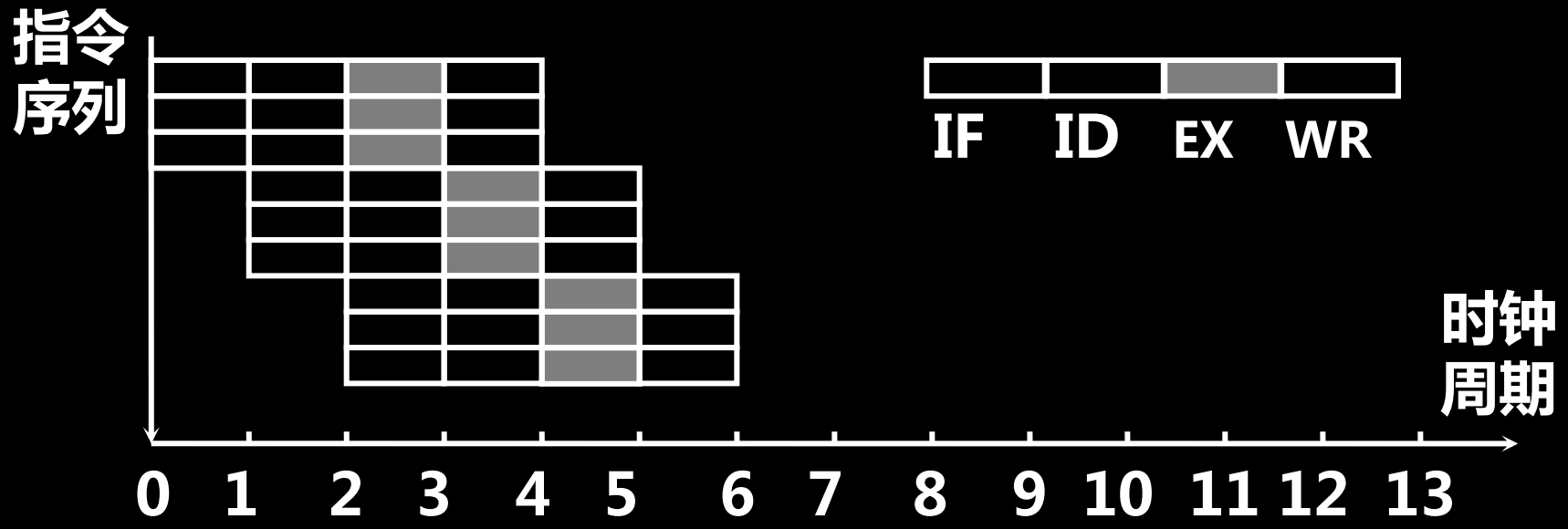
分支预测法

```
for(i=0; i<100; i++)
{
    .......
    .......
    .......
}
    .......
```

流水线的多发射技术

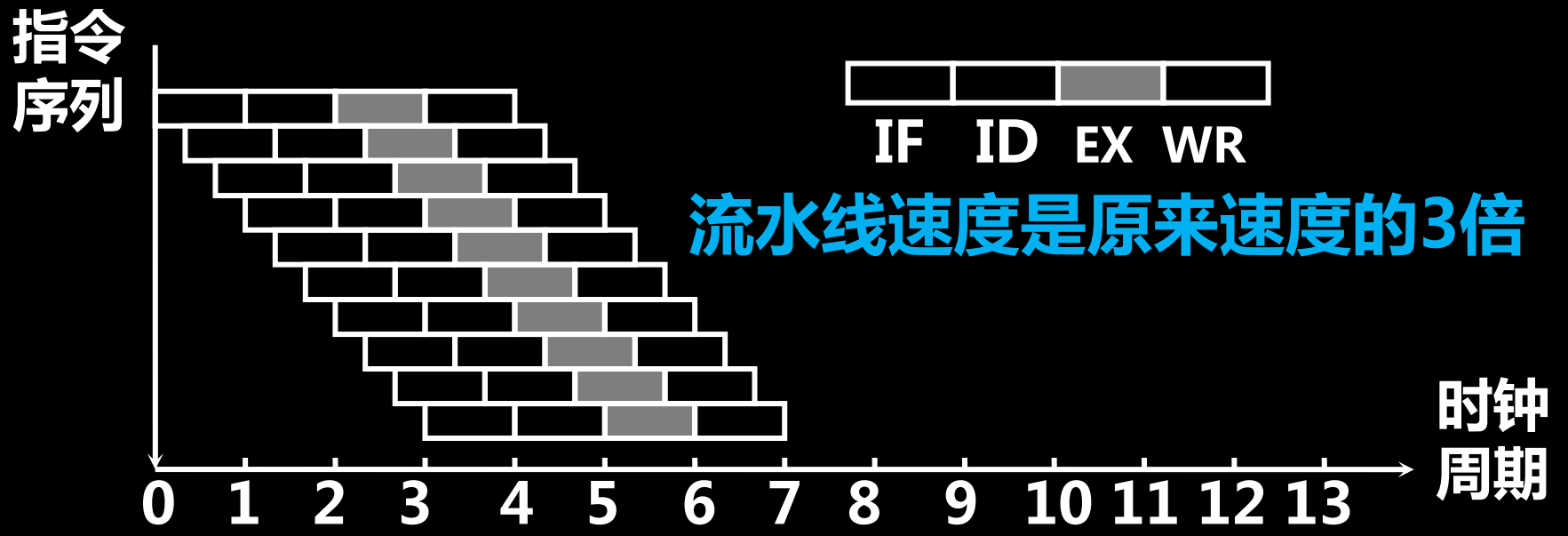
1. 超标量技术

- 每个时钟周期内可**并发发射多条独立指令**，配置多个功能部件。
- **不能调整**指令的**执行顺序**：通过编译优化技术，把可并行执行的指令搭配起来。



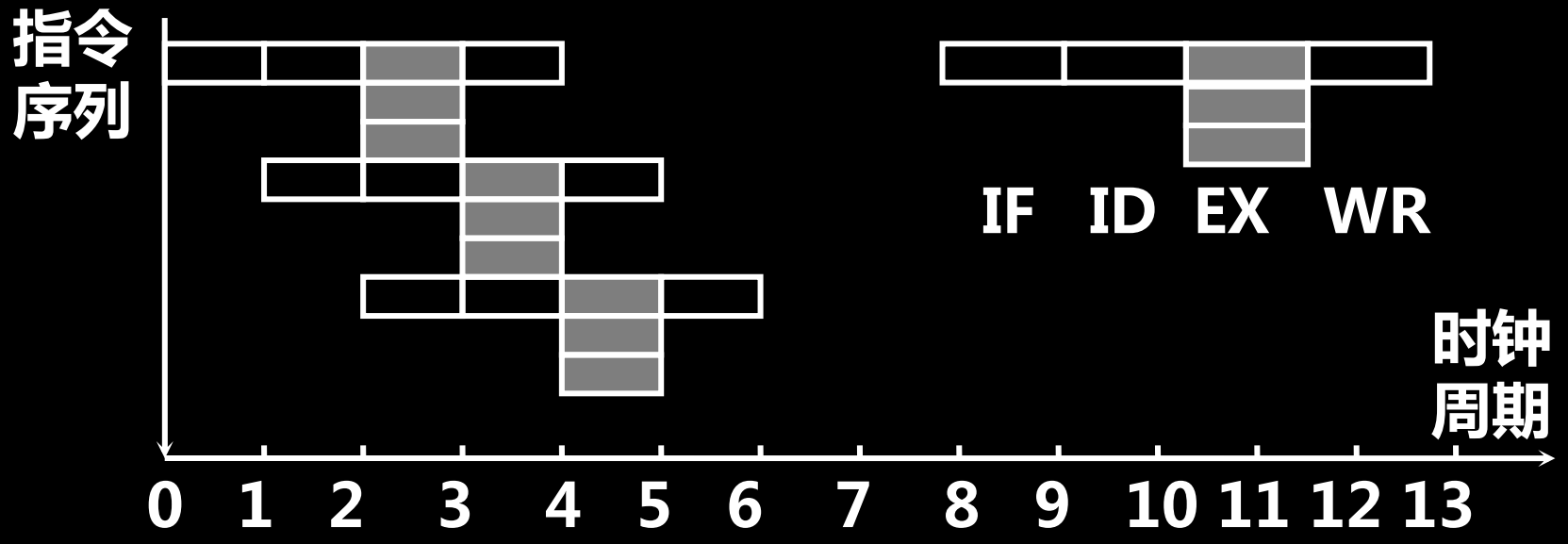
2. 超流水线技术

- 在一个时钟周期内再分段（3段）
  在一个时钟周期内一个功能部件使用多次（3 次）
- 不能调整指令的执行顺序：
  靠编译程序解决优化问题



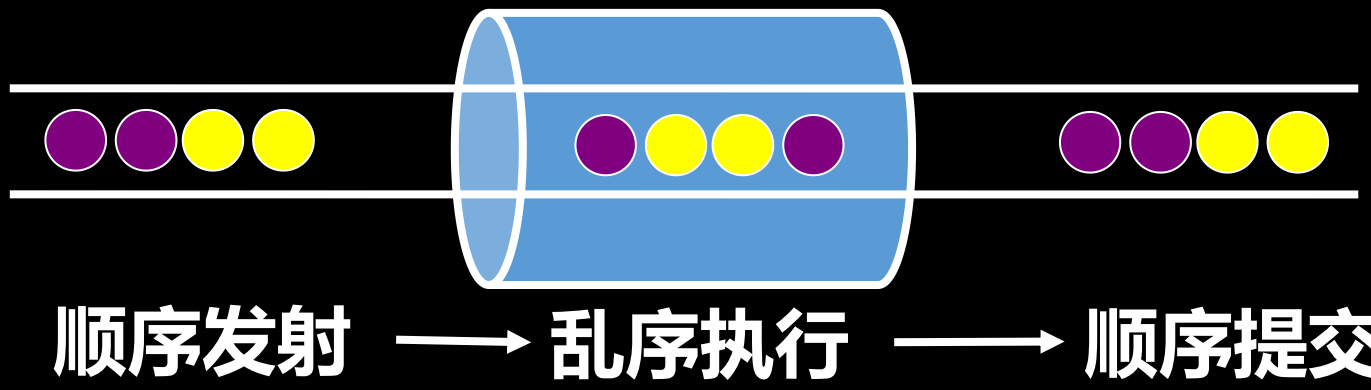
3. 超长指令字技术

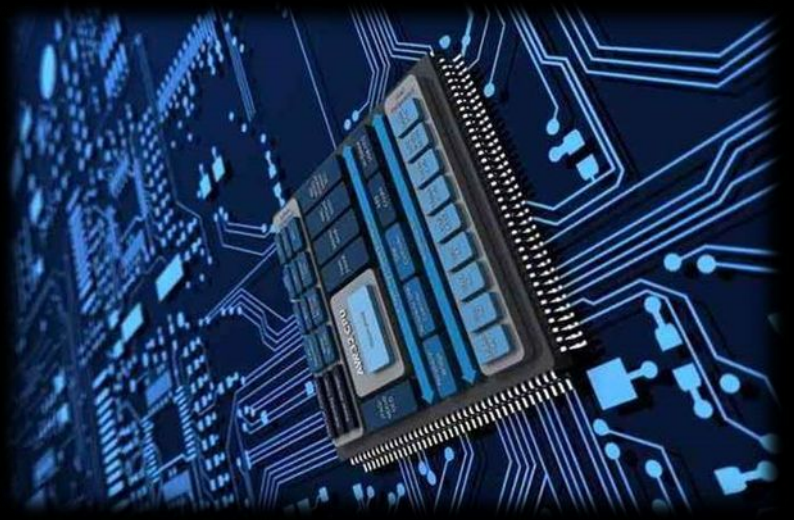
- 由编译程序挖掘出指令间潜在的并行性
  将多条能并行操作的指令组合成一条
- 具有多个操作码字段的超长指令字（可达几百位）
  采用多个处理部件



推荐阅读：指令动态调度

- 指令动态调度是指选择某个时钟周期内将要执行的指令，约束条件是尽量不产生相关。
- 记分牌法
- 乱序执行
- 更多内容。。。。





计算机组织与结构

大连理工大学 赖晓晨