



3. 指令系统设计与CPU运行控制

3.4 I/O系统

船说：计算机基础

3. 指令系统设计与CPU运行控制

4 I/O系统

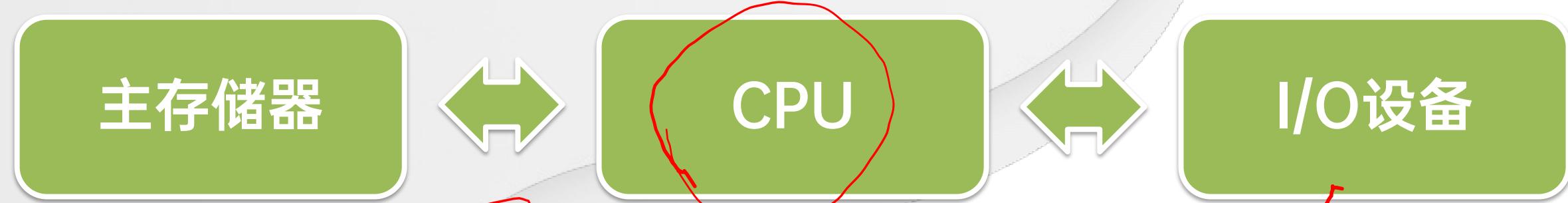
1. I/O系统概念与分类
2. I/O接口
3. I/O系统控制方式



3. 指令系统设计与CPU运行控制

3.4.1 I/O系统发展

分散连接

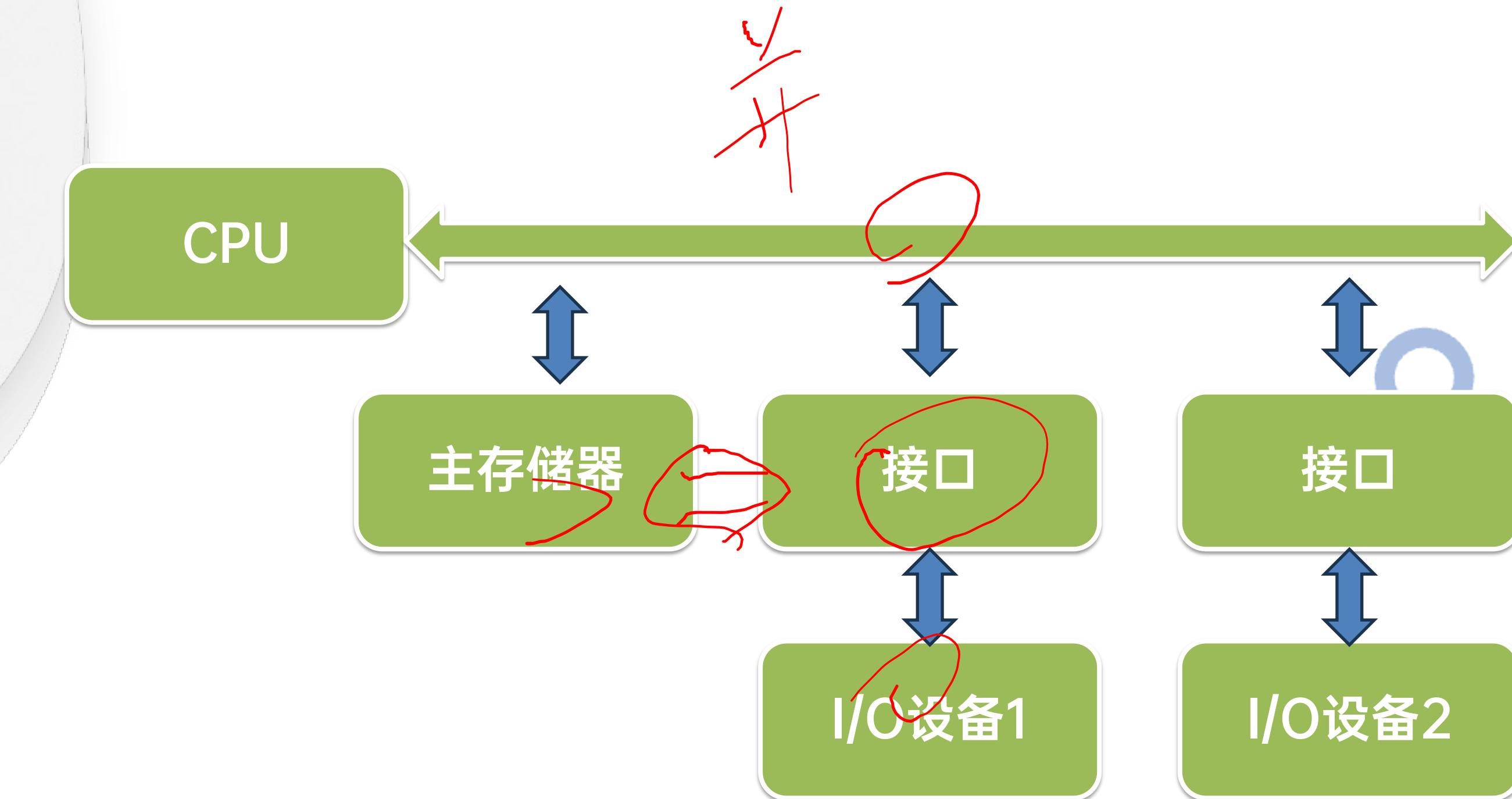
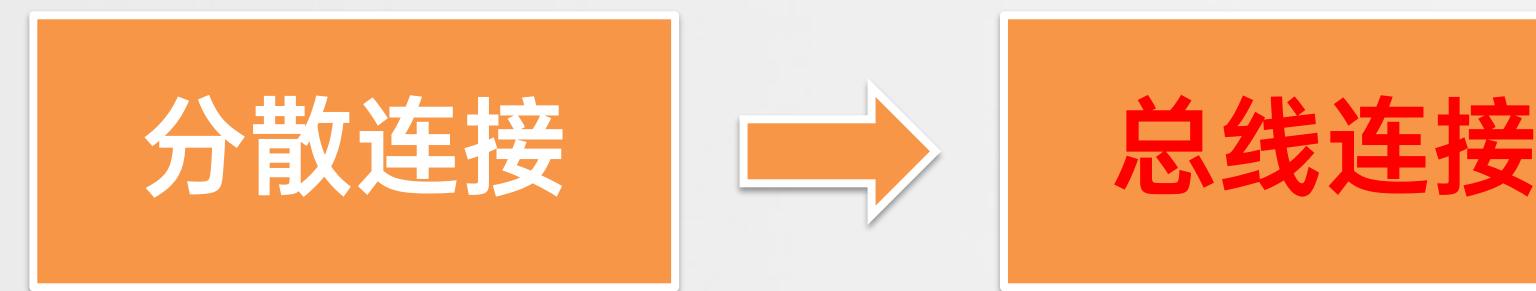


1. 线路散乱、庞杂
2. CPU查询状态，工作效率低
3. 硬件实现，可扩展性低



3. 指令系统设计与CPU运行控制

3.4.1 I/O系统发展



1. 接口有缓冲作用，可完成串-并变换
2. 接口处理中断事务
3. 接口可以使用多台设备分时占用总线，设备之间可并行工作
4. 设备和主存之间增加DMA



3. 指令系统设计与CPU运行控制

3.4.1 I/O系统发展



1. 通道负责管理I/O设备，并实现主存与I/O设备之间交换信息
2. 通道有专用指令，能独立执行用通道指令所编写的输入输出程序，
是从属于CPU的专用处理器
3. 设备工作时，CPU不直接参与管理



3. 指令系统设计与CPU运行控制

3.4.1 I/O系统发展

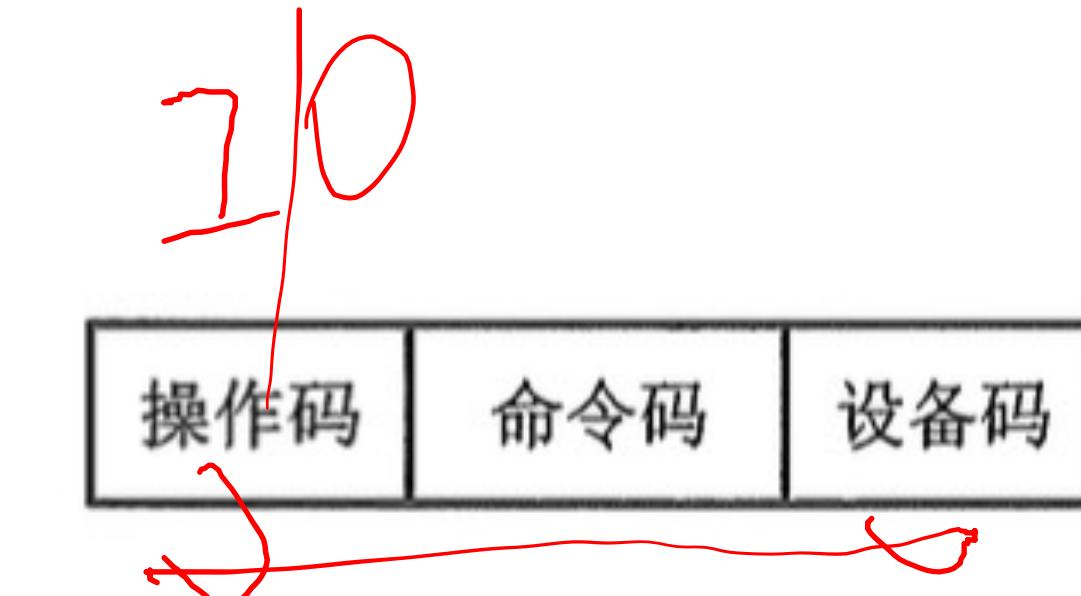
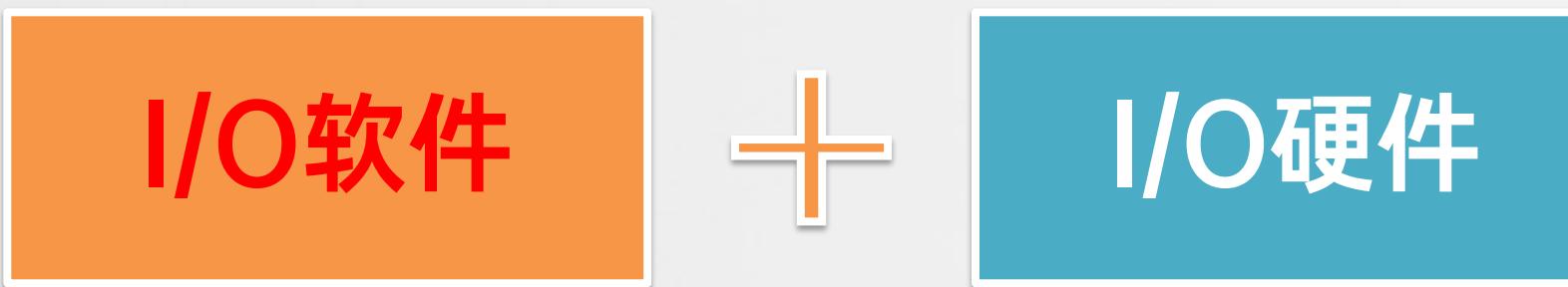


1. I/O处理机: Peripheral Processor (特殊功能CPU)
2. 基本独立于主机工作, 能完成I/O的控制, 还能完成码制变换、格式处理、数据块检错、纠错等
3. 设备工作并行性更高, 相对更独立



3. 指令系统设计与CPU运行控制

3.4.1 I/O系统组成



1. 将用户编制的程序（或数据）输入主机内
2. 将运算结果输送给用户
3. 实现输入输出系统与主机工作的协调
4. 主要由I/O指令、通道指令（通道控制字Channel Control Word）组成
5. 对于采用~~接口~~模块方式，必须要靠I/O指令完成协调工作；对于采用通道方式的，还需要通道指令

3. 指令系统设计与CPU运行控制

3.4.1 I/O系统组成

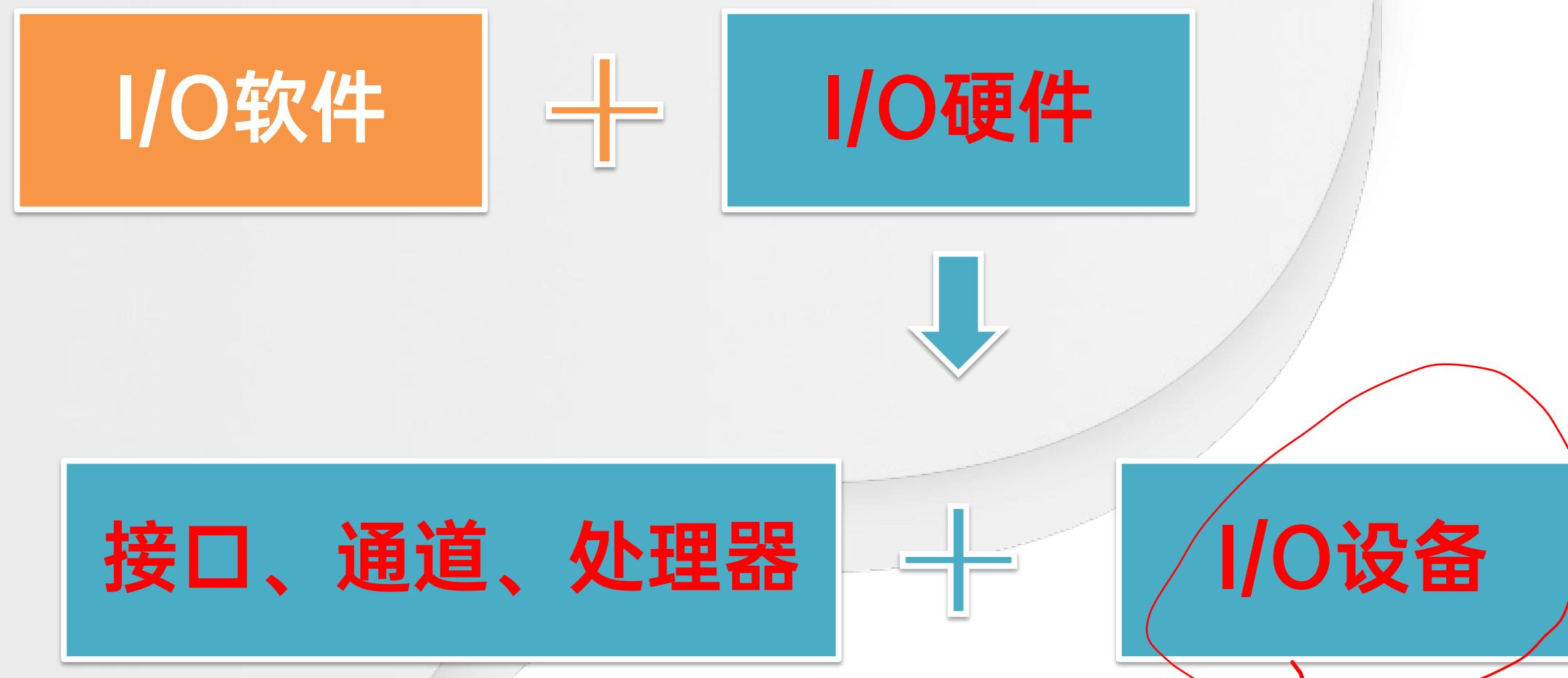
I/O软件

+

I/O硬件

3. 指令系统设计与CPU运行控制

3.4.1 I/O系统组成





3. 指令系统设计与CPU运行控制

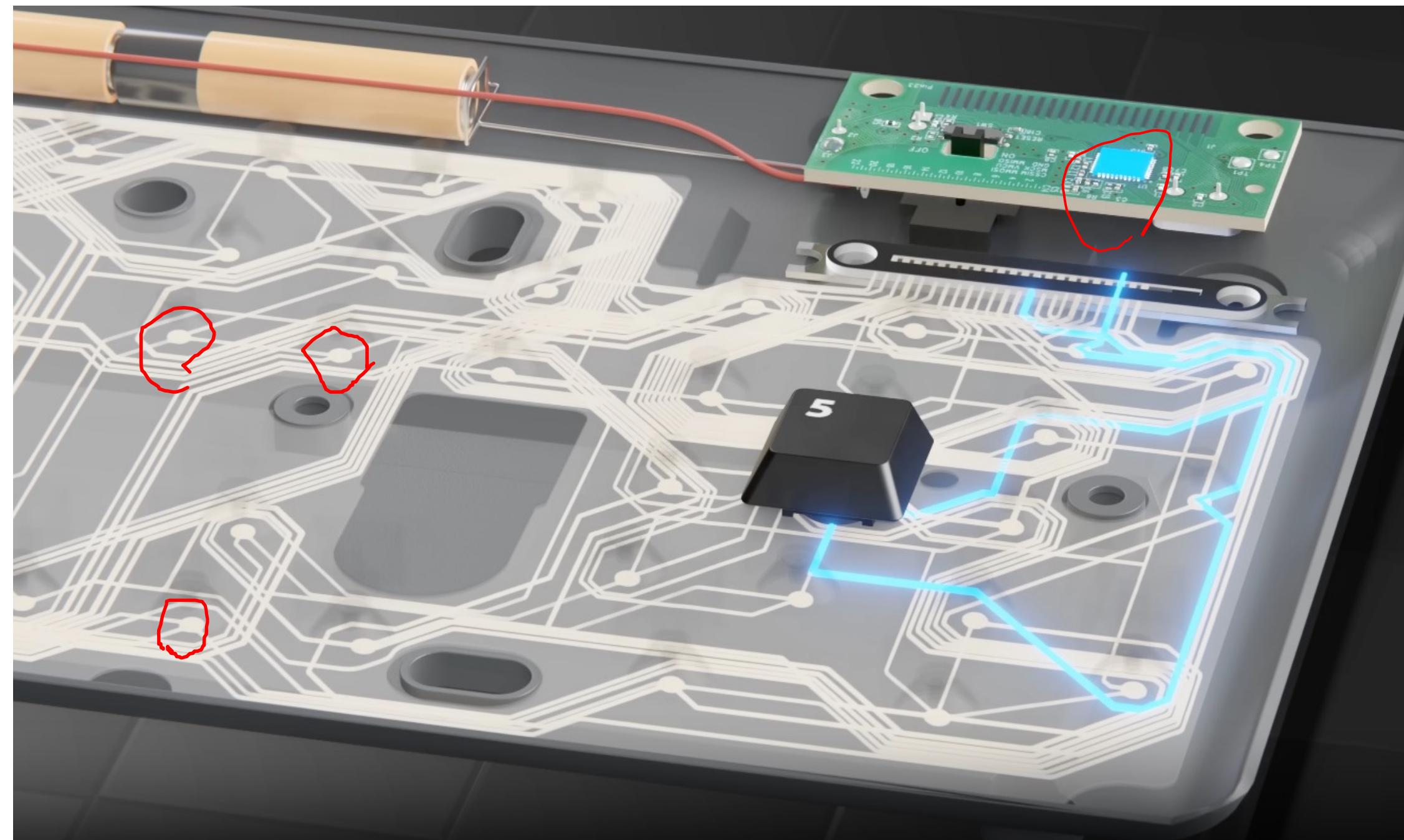
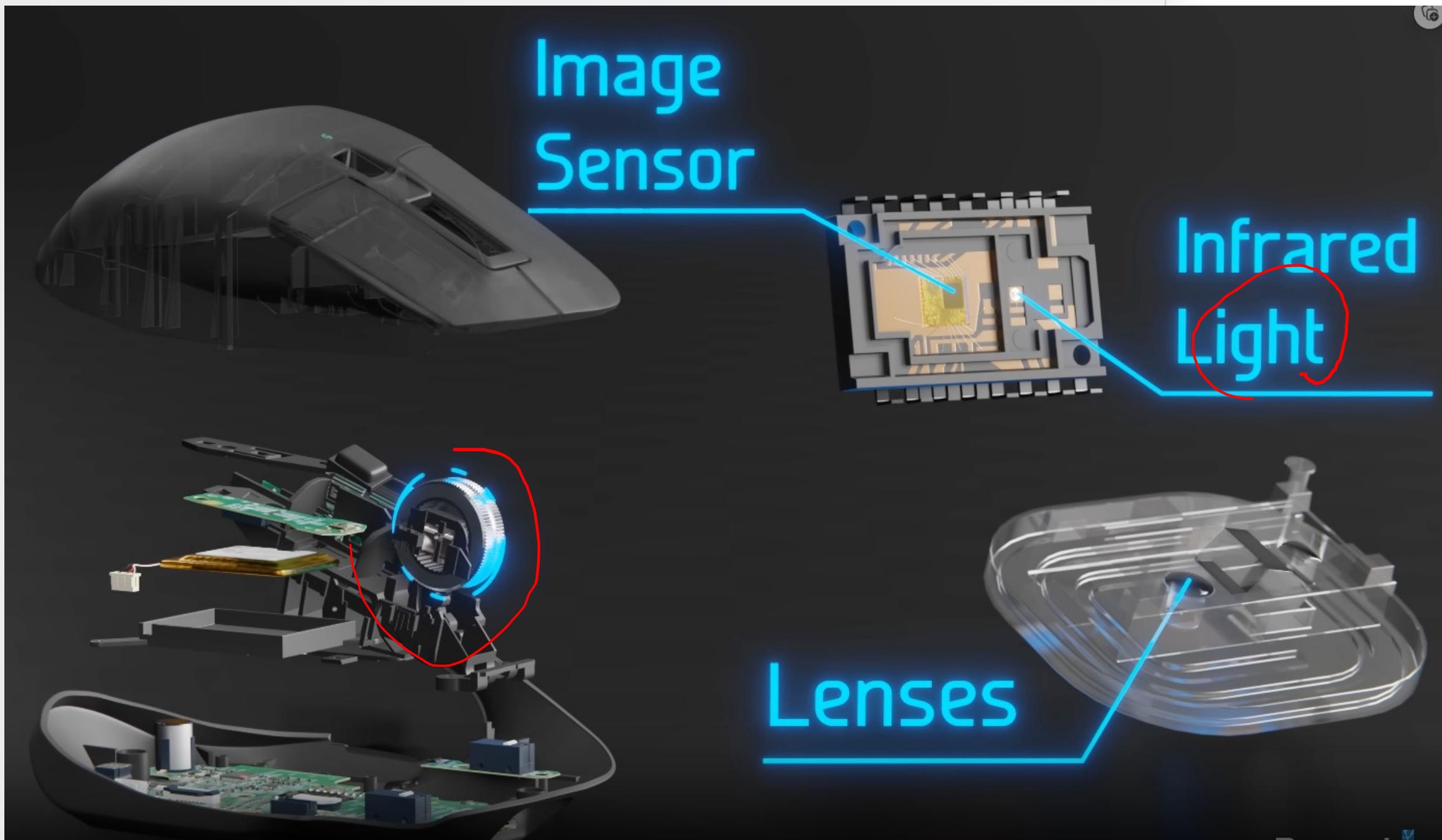
3.4.1 外部设备分类

1. 人机交互设备: 键盘、鼠标、打印机、显示器、写字绘画板、触摸屏等
2. 计算机信息存储设备: 硬盘 (HD、SSD) 、光盘、磁带等
3. 机-机通信设备: 调制解调器 (光猫) , RS232, IIC, SPI, CAN-Bus等



3. 指令系统设计与CPU运行控制

3.4.1 输入设备





3. 指令系统设计与CPU运行控制

3.4.1 输出设备-显示器

1. CRT: 光栅扫描和随机扫描, 分辨率和灰度级, 刷新和刷新存储器
2. LCD
3. 等离子
4. OLED
5. miniOLED

8. 16. 32

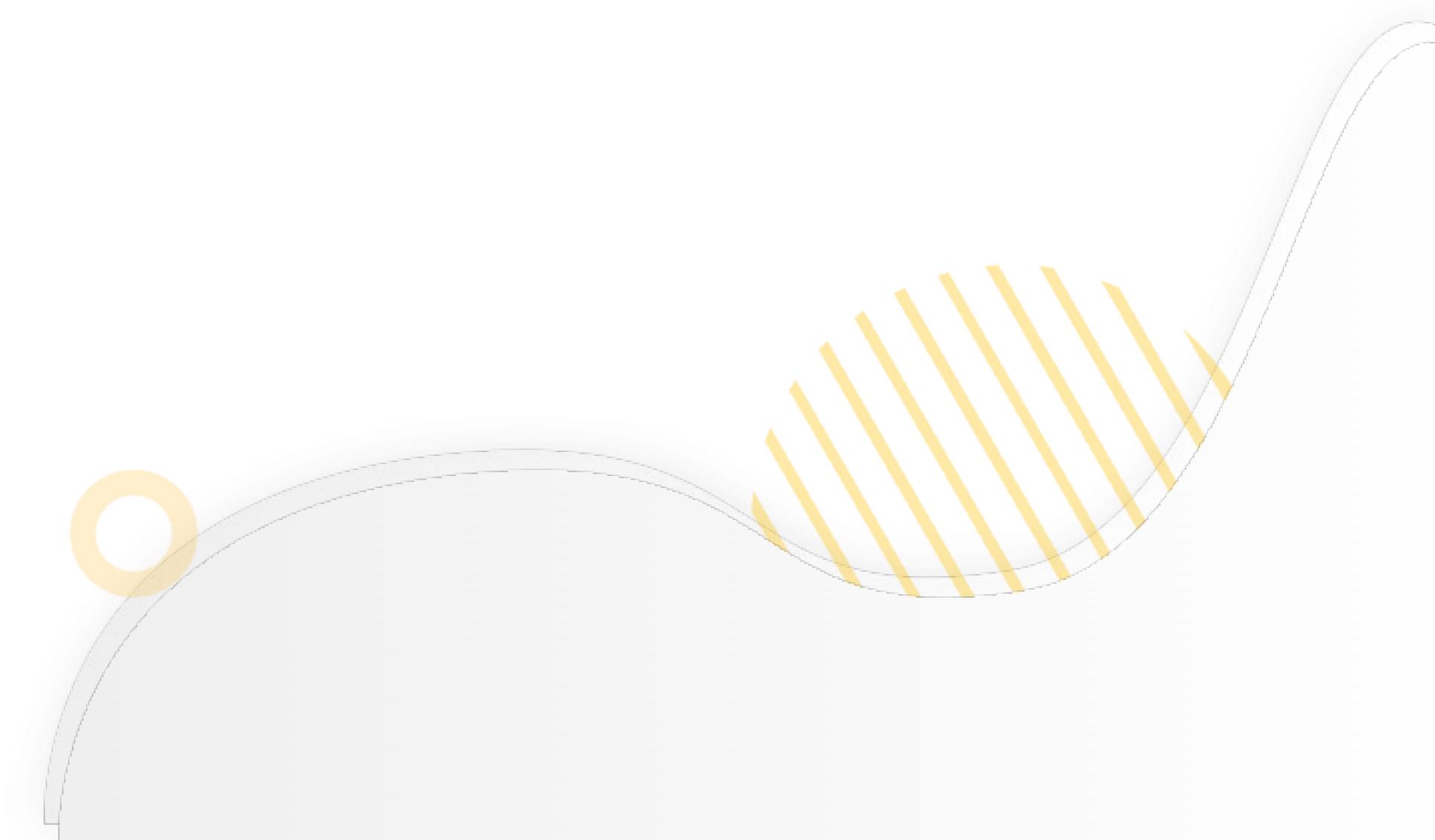


3. 指令系统设计与CPU运行控制

3.4.1 输出设备-打印机

1. 针式打印机
2. 喷墨打印机
3. 激光打印机
4. 3D打印机

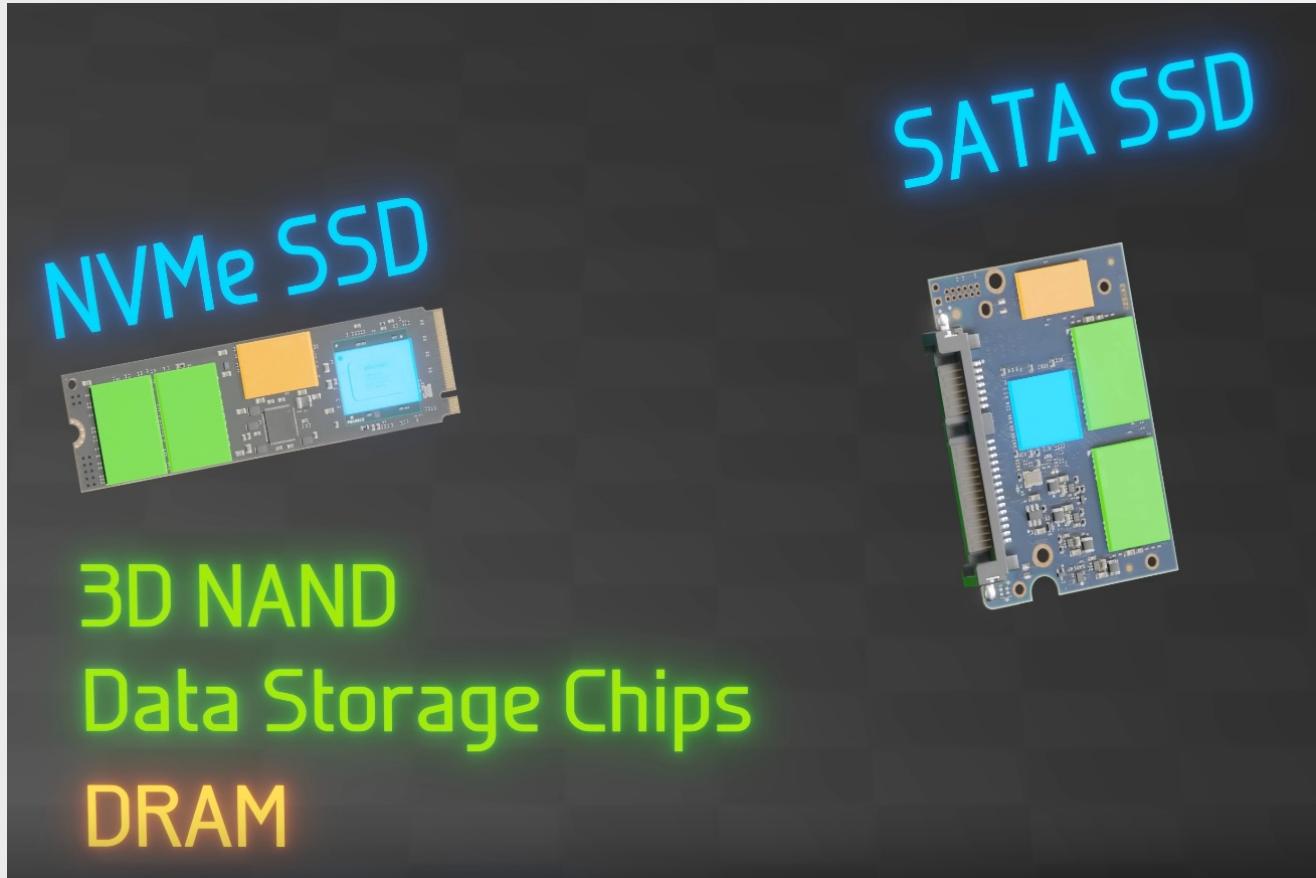
384





3. 指令系统设计与CPU运行控制

3.4.1 信息存储设备





3. 指令系统设计与CPU运行控制

3.4.1 常用I/O设备

输入输出设备	输入设备	键盘
		图形输入设备(鼠标、图形板、跟踪球、操纵杆、光笔)
		图像输入设备(摄像机、扫描仪、传真机)
		条形码阅读器
		光学字符识别
输入输出设备	输出设备	语音和文字输入设备
		显示器(字符、汉字、图形、图像)
		打印设备(点阵式打印机、激光打印机、喷墨打印机)
		绘图仪(平板式、滚筒式)
		音箱
		终端设备(键盘+显示器)
		汉字处理设备
		A/D、D/A 转换设备
		多媒体设备
		脱机输入输出设备(软磁盘数据站)

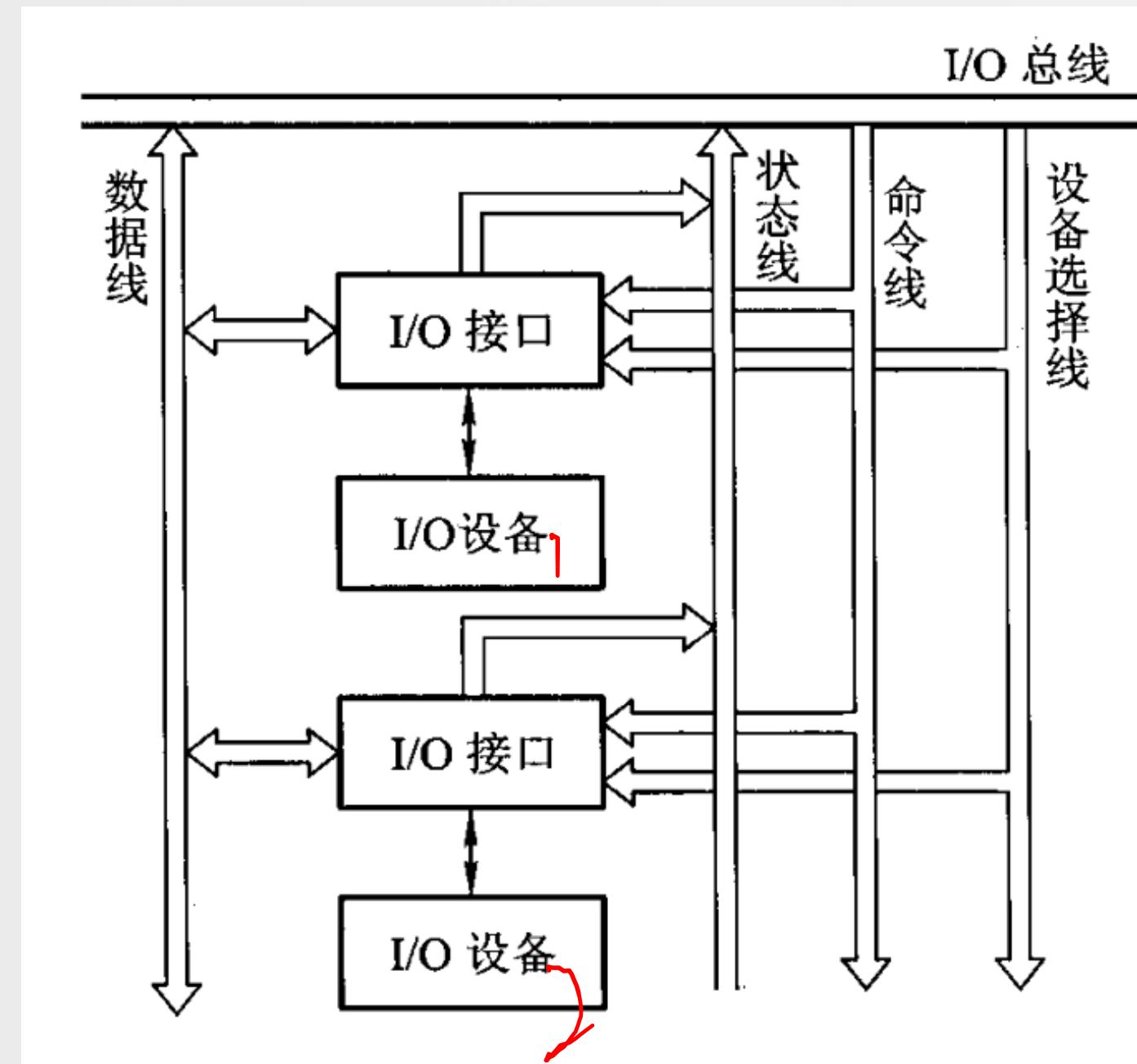


3. 指令系统设计与CPU运行控制

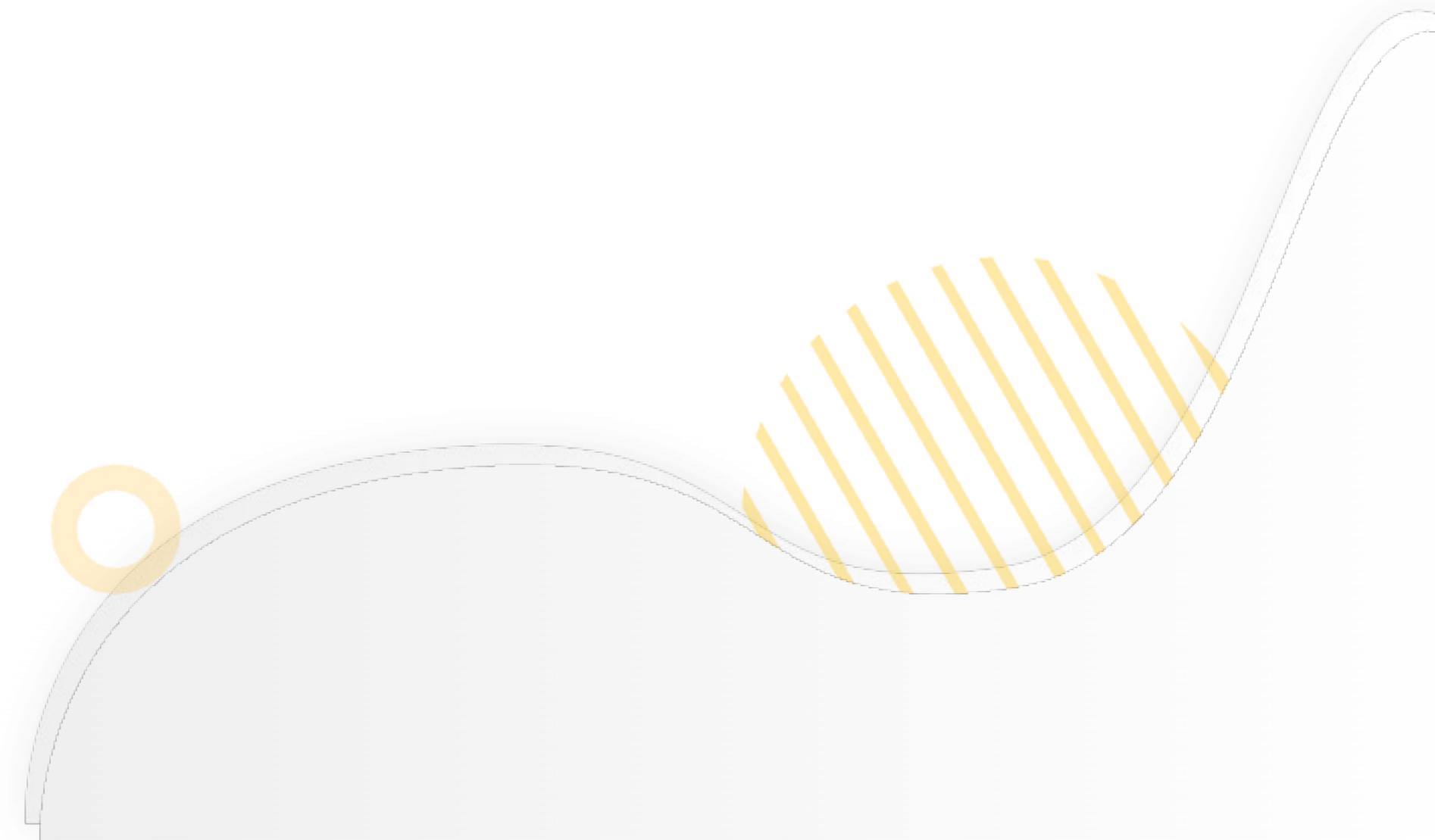
3.4.2 I/O 接口功能作用



MCU



1. 不同速度设备通过接口进行数据缓冲
2. 多个相同类型的设备
3. 通信方式转换，串行通信变成并行传送到CPU



3. 指令系统设计与CPU运行控制

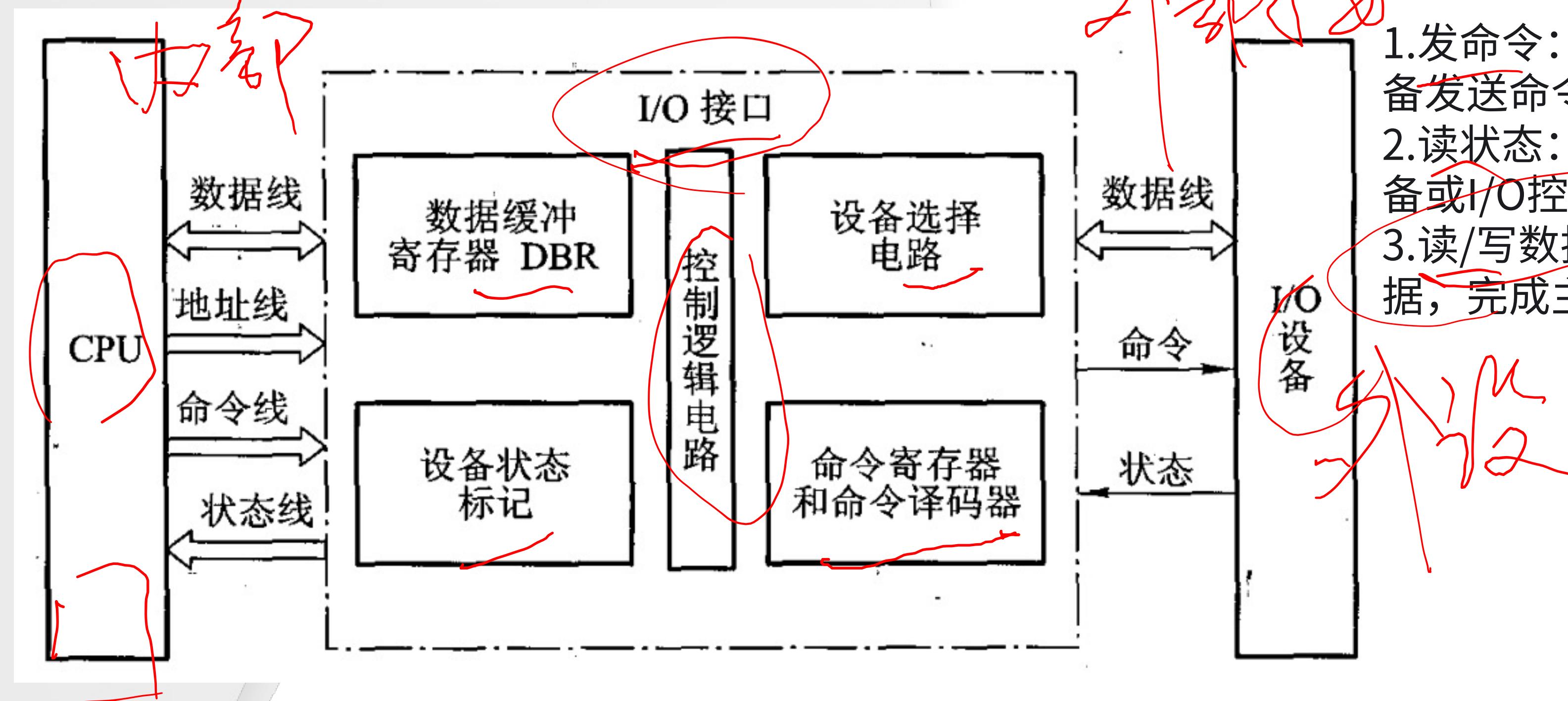
3.4.2 I/O 接口功能

1. 选址功能：当设备选择线上的设备码与本设备码相符时，应发出设备选中信号SEL；
2. 传送命令的功能：通常在I/O接口中设有存放命令的命令寄存器以及命令译码器，来对CPU的命令进行响应。
3. 传送数据的功能：接口中通常设有数据缓冲寄存器(Data Buffer Register, DBR)，它用来暂存I/O设备与主机准备交换的信息，与I/O总线中的数据线是相连的。
4. 反映I/O设备工作状态的功能：为了使CPU能及时了解各I/O设备的工作状态，接口内必须设置一些反映设备工作状态的触发器。



3. 指令系统设计与CPU运行控制

3.4.2 I/O 接口结构组成



1. 发命令：发送命令字到I/O控制寄存器，向设备发送命令（需要驱动程序的协助）
2. 读状态：从状态寄存器读取状态字，获得设备或I/O控制器的状态信息
3. 读/写数据：从数据缓冲寄存器发送或读取数据，完成主机与外设的数据交换

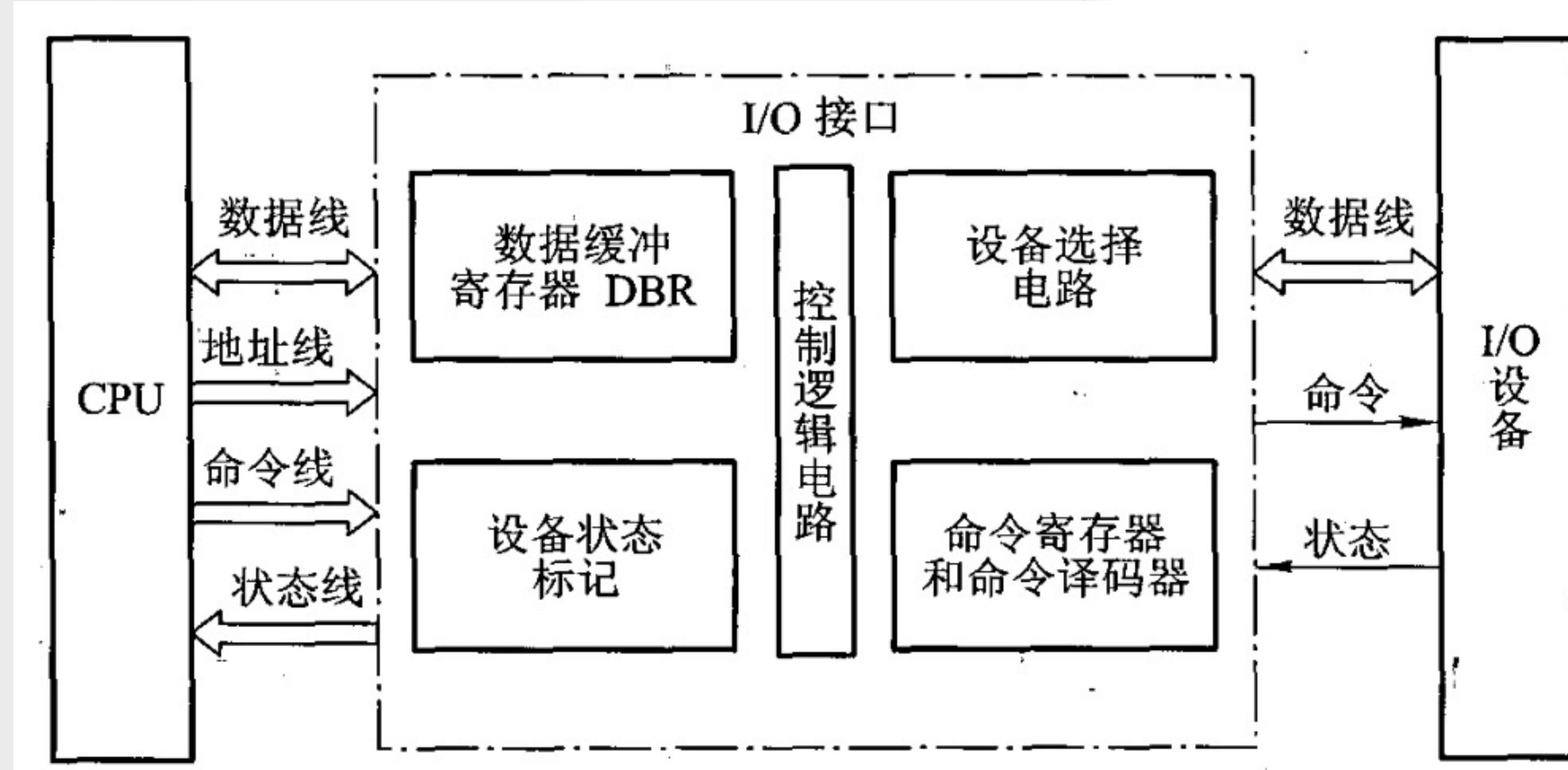
外部接口：外部接口通过接口电缆与外设相连，外部接口的数据传输可能是串行方式，因此I/O接口需具有串/并转换功能。
内部接口：内部接口与系统总线相连，实质上是与内存、CPU相连。

CHIPS



3. 指令系统设计与CPU运行控制

3.4.2 I/O 接口工作过程



- 1.发命令：发送命令字到I/O控制寄存器，向设备发送命令（需要驱动程序的协助）
- 2.读状态：从状态寄存器读取状态字，获得设备或I/O控制器的状态信息
- 3.读/写数据：从数据缓冲寄存器发送或读取数据，完成主机与外设的数据交换



3. 指令系统设计与CPU运行控制

3.4.2 I/O 接口类型与分类

85+326+

• 按数据传送方式分类：

- 并行接口：将一个字节（或一个字）的所有位同时传送；
- 串行接口：在设备与接口间一位一位传送，由于接口与主机之间是按字节或字并行传送，因此对串行接口而言，其内部还必须设有串—并转换装置。

串口 SPI

• 按功能选择的灵活性分类：

- 可编程接口：其功能及操作方式可用程序来改变或选择；
- 不可编程接口：不能由程序来改变其功能，但可通过硬连线逻辑来实现不同的功能。

• 按通用性分类：

- 通用接口：可供多种I/O设备使用；
- 专用接口：为某类外设或某种用途专门设计的。

• 按数据传送的控制方式分类：

- 程序型接口：用于连接速度较慢的I/O设备，如显示终端、键盘、打印机等；
- DMA型接口：用于连接高速I/O设备，如磁盘、磁带等。

XIP

• 按主机访问I/O设备的方式分类：

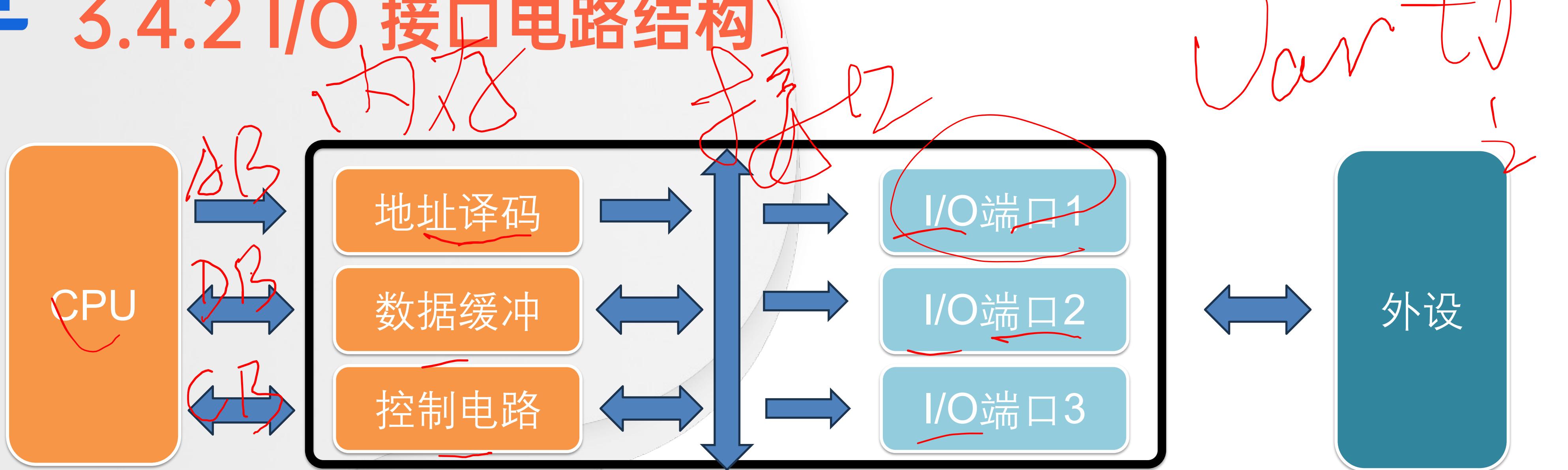
- 查询接口：用查询方式处理；
- 中断接口：用中断方式处理；
- DMA接口：用DMA方式处理；

O



3. 指令系统设计与CPU运行控制

3.4.2 I/O 接口电路结构



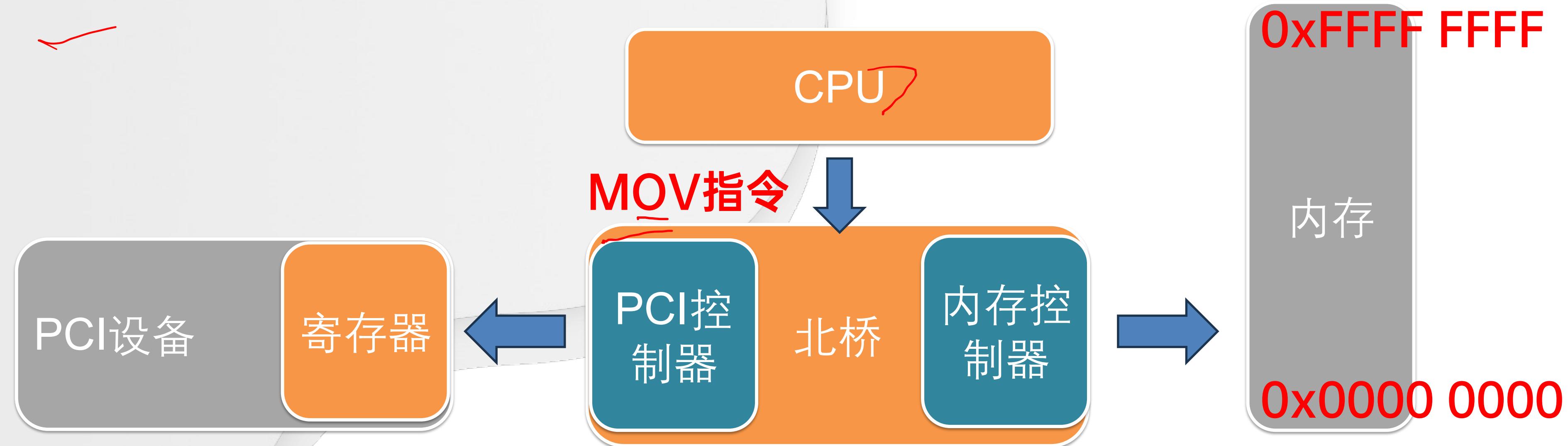
1. I/O端口：可以由CPU进行读或写的寄存器，这些寄存器用来存放数据、控制信息、状态信息。若干个端口加上相应的控制逻辑电路组成接口。
2. I/O端口的编址：统一编址和不统一编址



3. 指令系统设计与CPU运行控制

3.4.2 I/O 端口编址方式

X86系统的32位机地址空间：



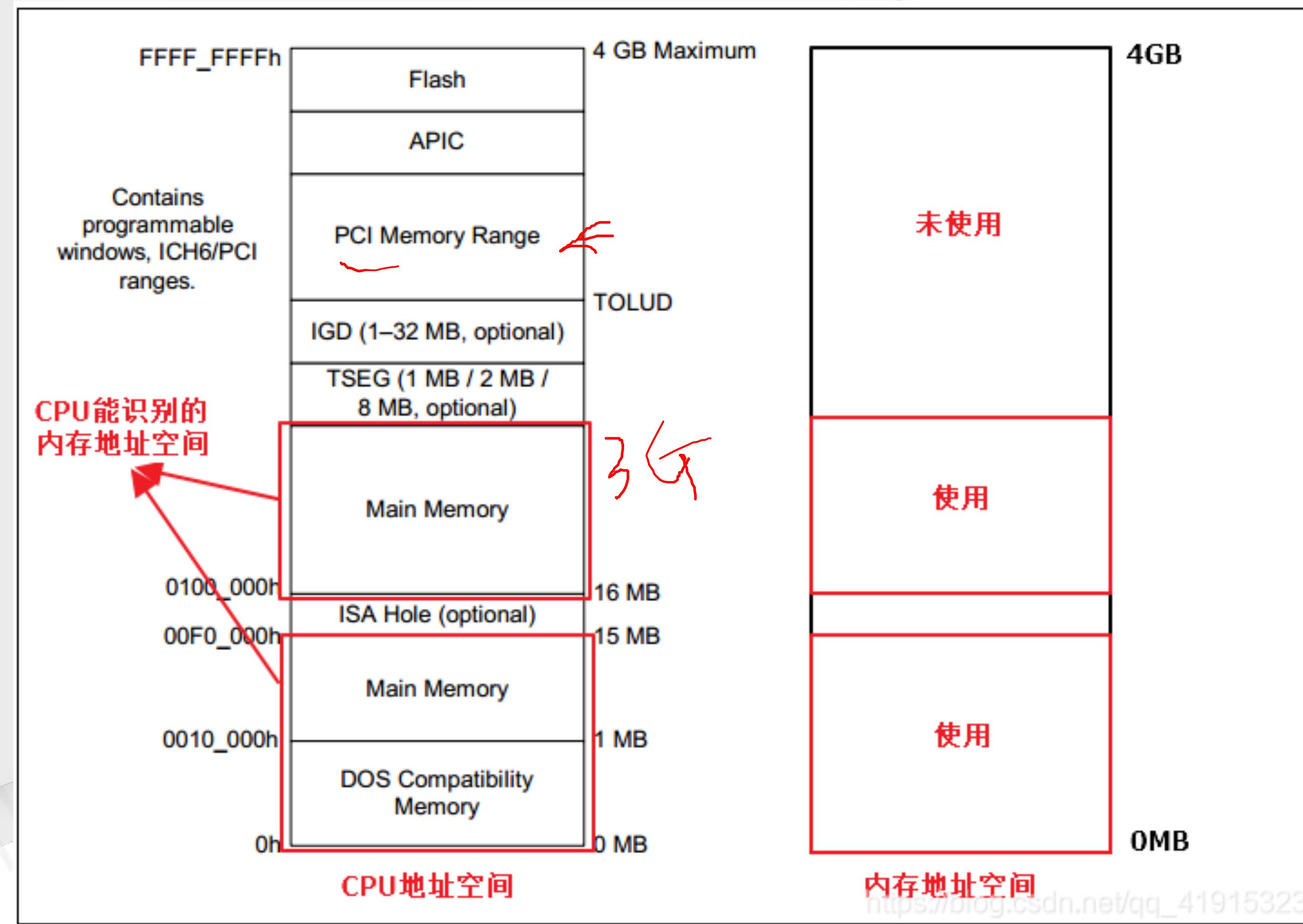
readl, writel 指令也是mov指令的封装



3. 指令系统设计与CPU运行控制

3.4.2 I/O 端口编址方式

X86系统的32位机地址空间：



readl, writel 指令也是
mov指令的封装

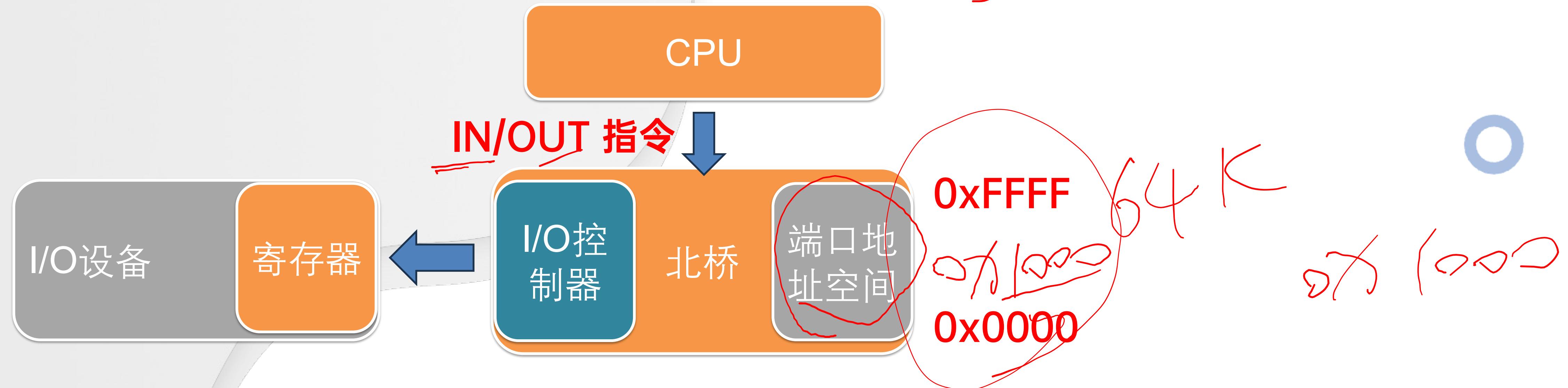


3. 指令系统设计与CPU运行控制

3.4.2 I/O 端口编址方式

X86系统的32位机地址空间：

in8, out8 指令也是in/out指令的封装



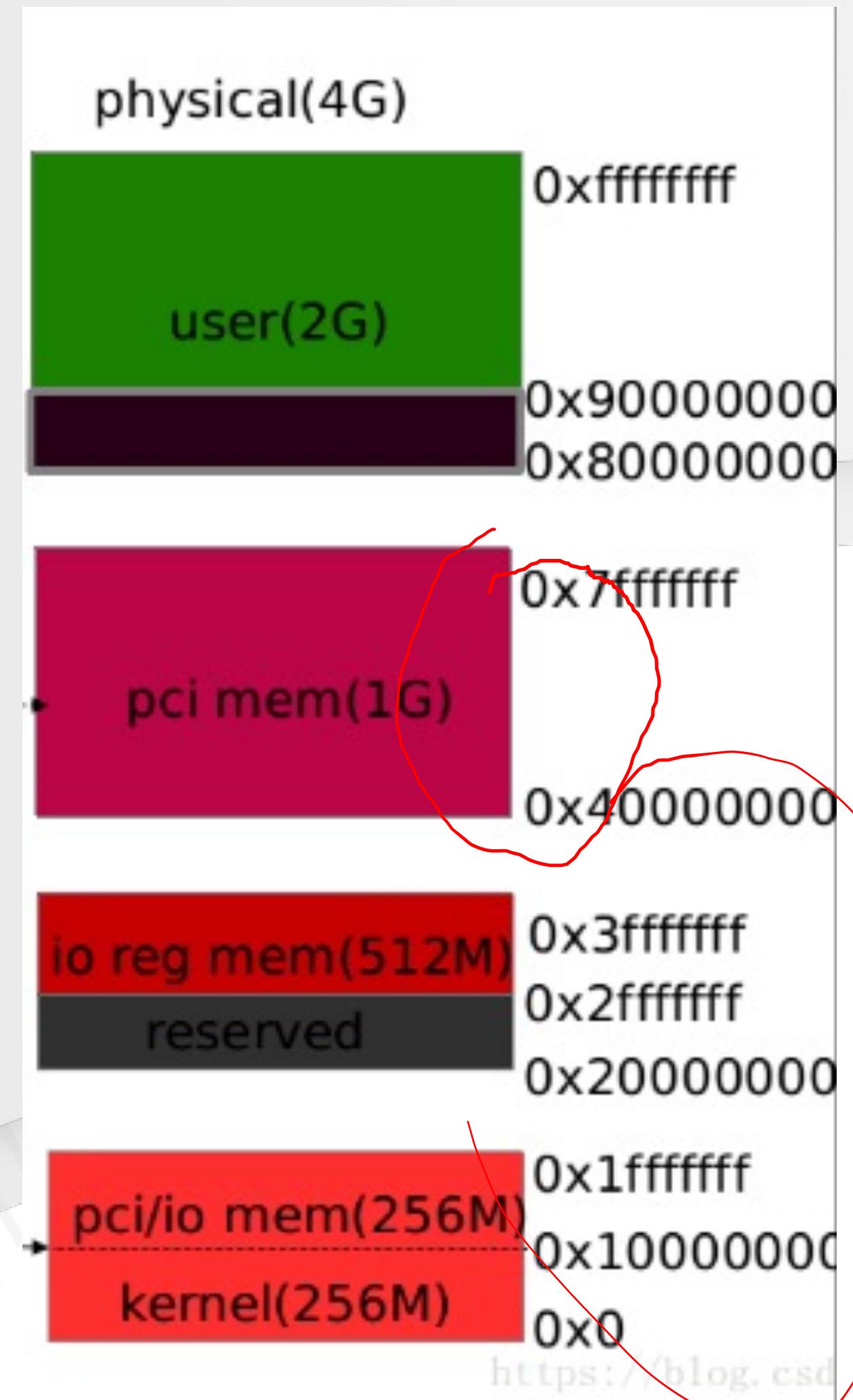
独立编址优点: 又称I/O映射方式, 独立的指令, 程序编制清晰

独立编址缺点: 输入输出指令少, 只对端口操作

3. 指令系统设计与CPU运行控制

3.4.2 I/O 端口编址方式

ARM 芯片 32位机地址空间：



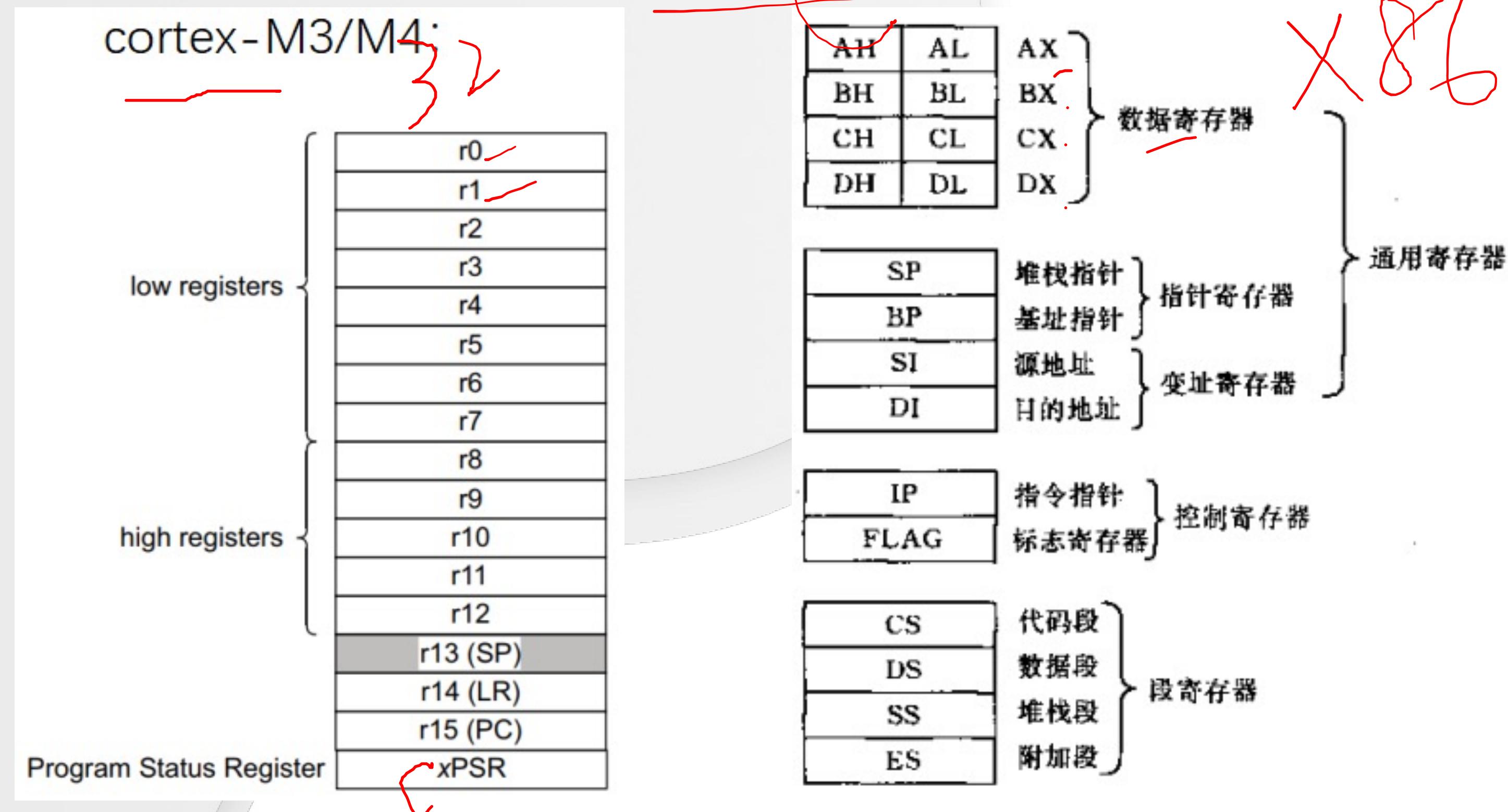
LDR, STR, LDM, STM 指令访存

统一编址优点：无需独立指令，操作灵活方便，端口空间扩展性好
 独立编址缺点：占用主存地址，执行速度相对偏慢

3. 指令系统设计与CPU运行控制



3.4.2 CPU内的通用寄存器和I/O端口控制寄存器





3. 指令系统设计与CPU运行控制

3.4.2 练习

例1 【2012真题】：下列选项中，在I/O总线的数据线上传输的信息包括①。

- ① I/O接口中的命令字 ② I/O接口中的状态字 ③ 中断类型号
- A. 仅①和② B. 仅①和③ C. 仅②和③ D. ①、②和③

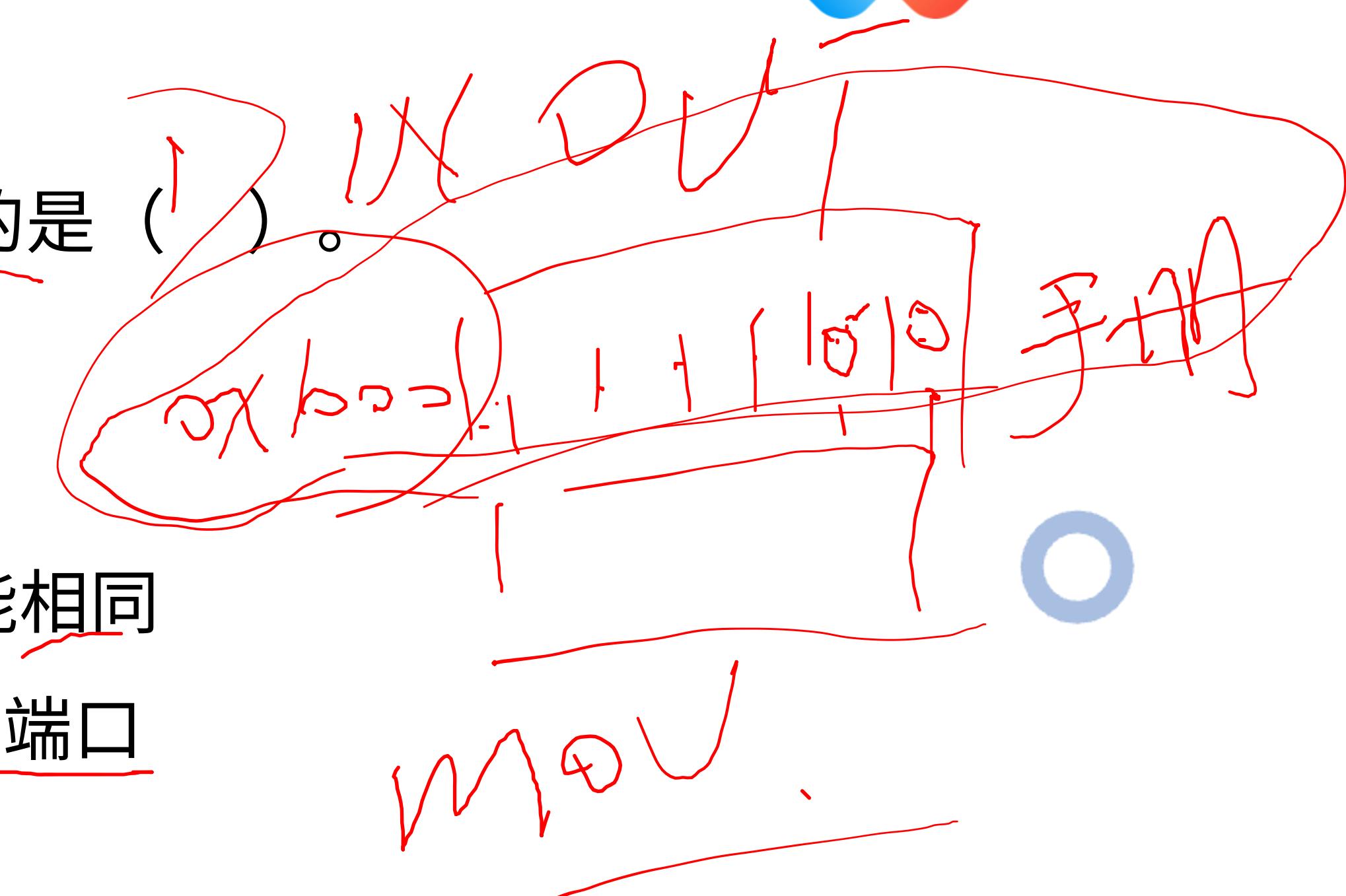


3. 指令系统设计与CPU运行控制

3.4.2 练习

例2 【2014真题】：下列有关I/O接口的叙述中，错误的是

- A. 状态端口和控制端口可以合用同一个寄存器
- B. I/O接口中CPU可访问的寄存器称为I/O端口
- C. 采用独立编址方式时，I/O端口地址和主存地址可能相同
- D. 采用统一编址方式时，CPU不能用访存指令访问I/O端口



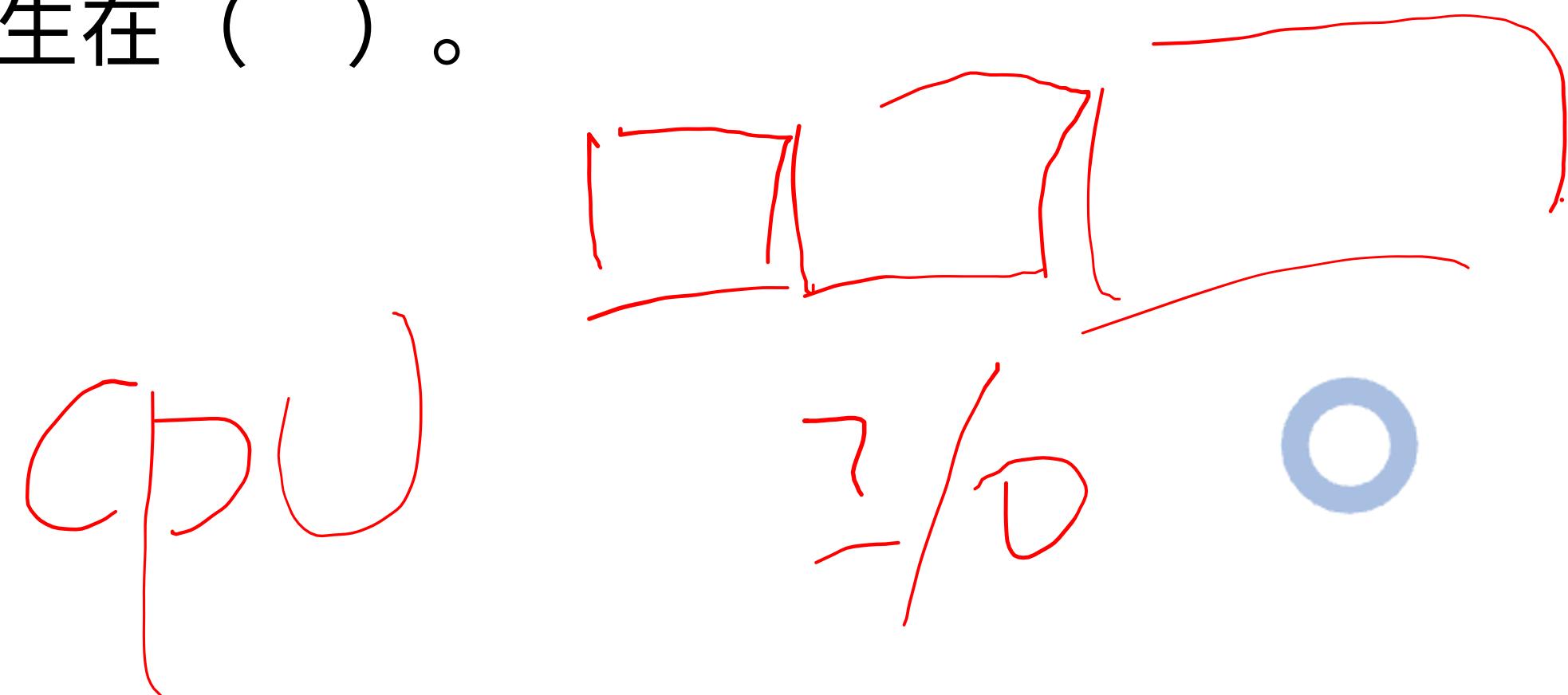


3. 指令系统设计与CPU运行控制

3.4.2 练习

例3 【2017真题】：I/O指令实现的数据传送通道通常发生在（ ）。

- A. I/O设备和I/O端口之间
- B. 通用寄存器和I/O设备之间
- C. I/O端口和I/O端口之间
- D. 通用寄存器和I/O端口 之间





3. 指令系统设计与CPU运行控制

3.4.2 练习

例4 【2021真题】：下列选项中，不属于I/O接口的是 (A)。

- A. 磁盘驱动器
- B. 打印机适配器
- C. 网络控制器
- D. 可编程中断控制器



3. 指令系统设计与CPU运行控制

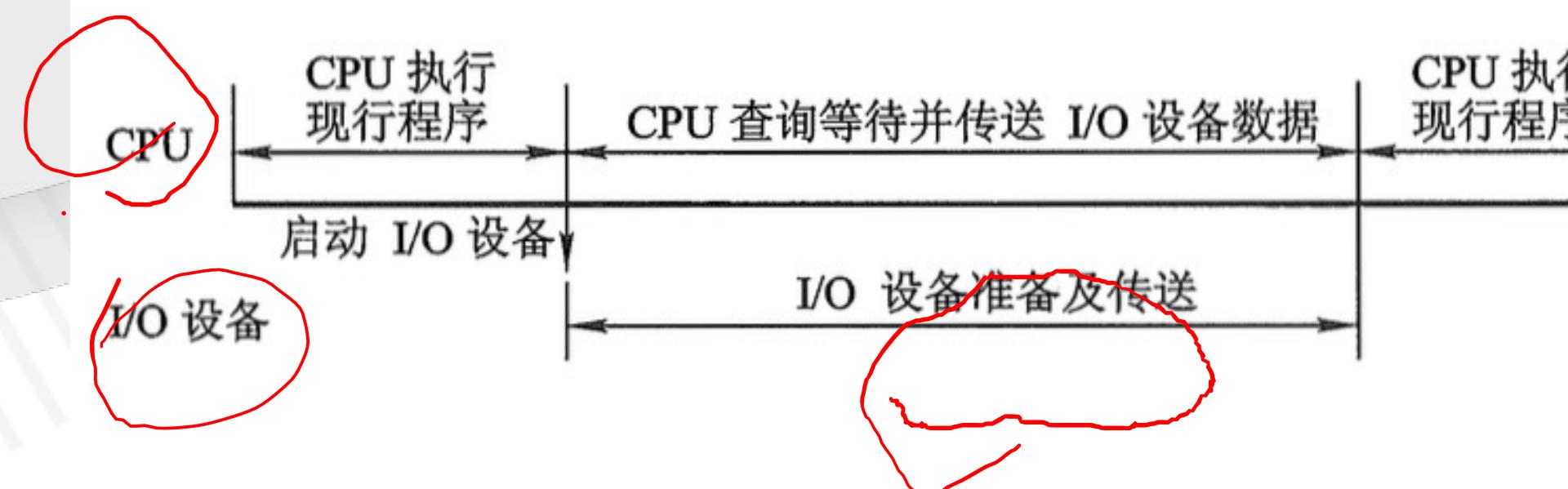
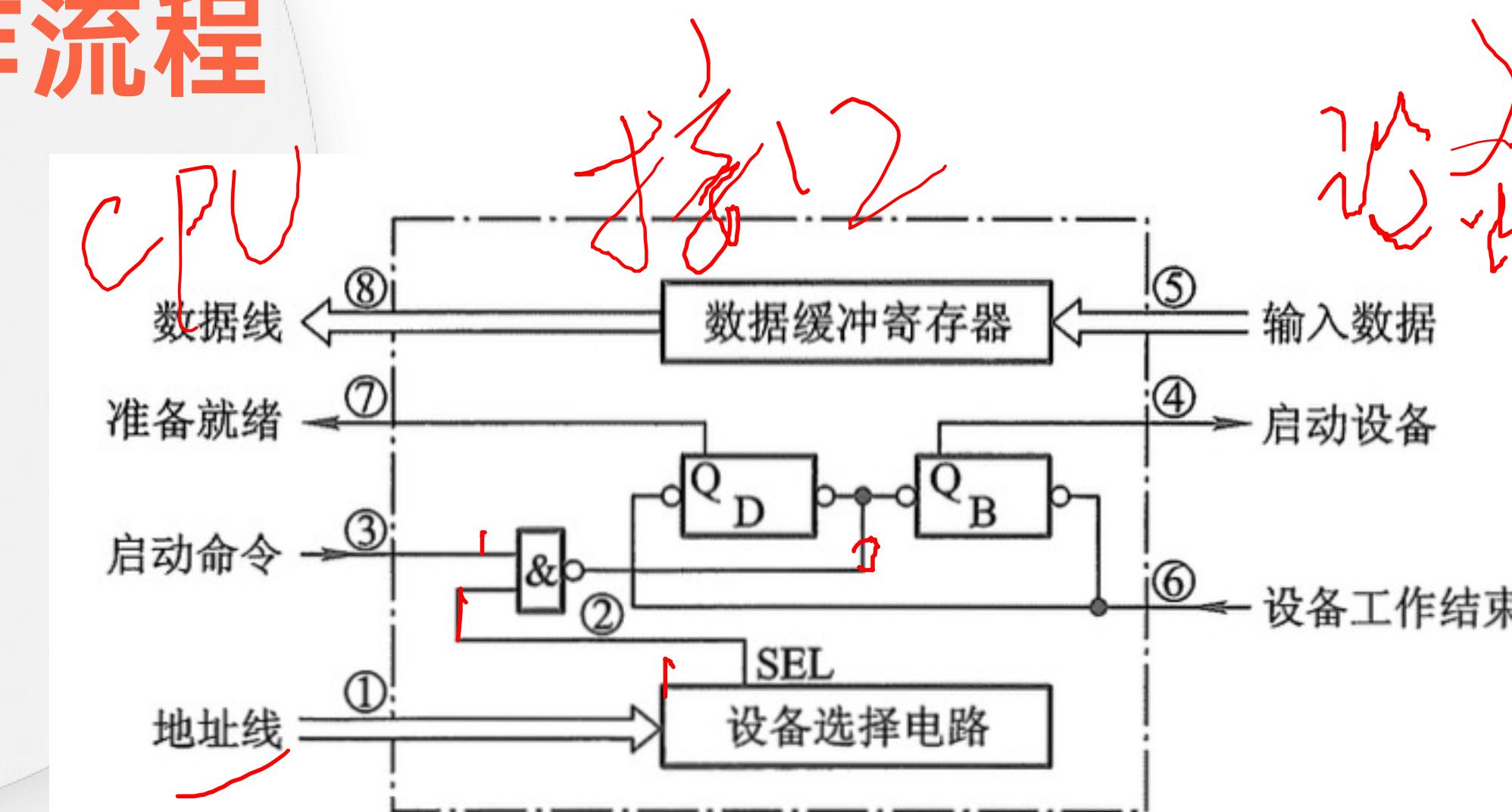
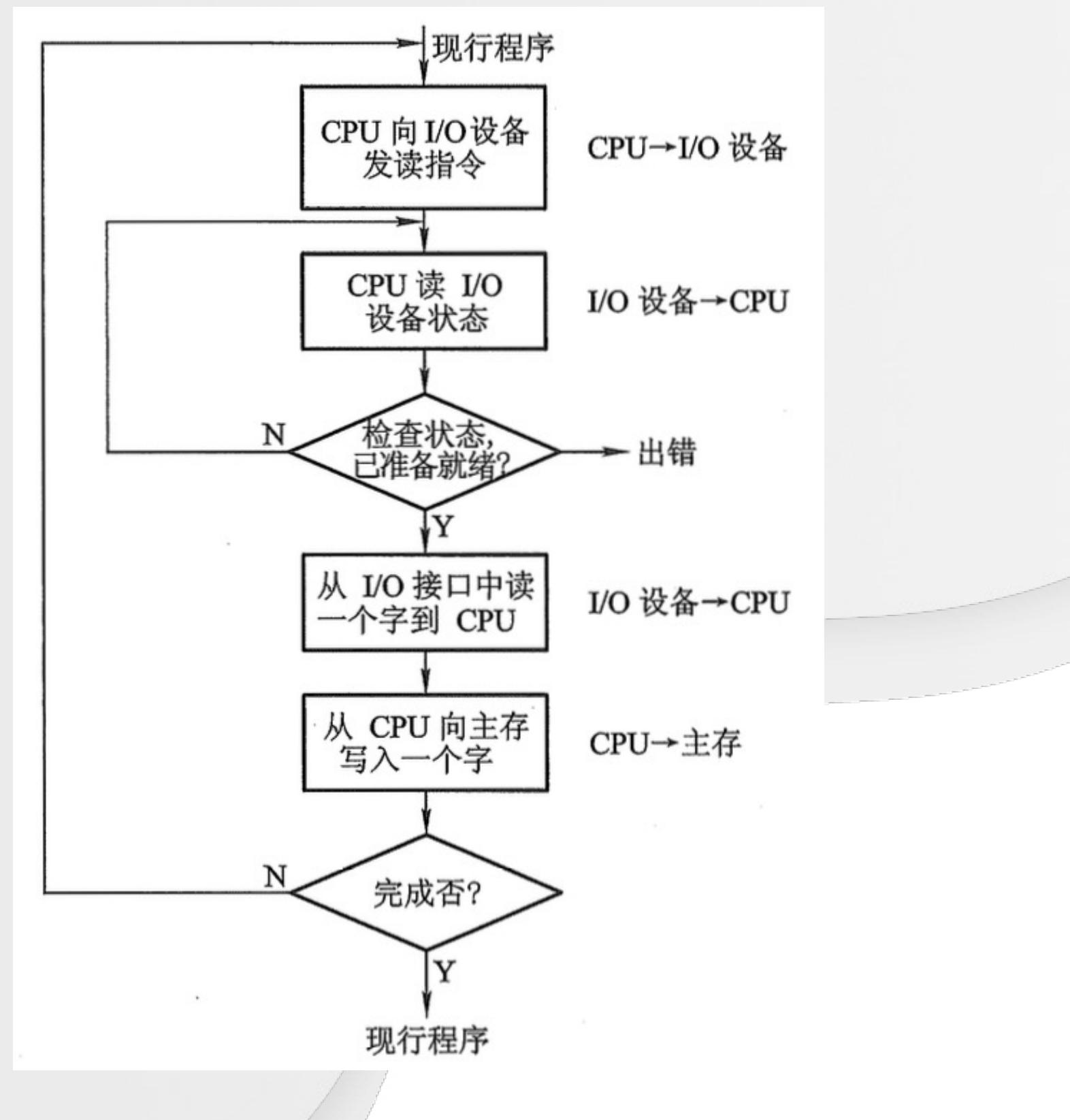
3.4.3 I/O设备与主机信息传递的控制方式 ★ ★ ★

1. 程序查询方式
2. 程序中断方式
3. 直接存储器存取 (DMA)
4. I/O通道方式
5. I/O处理机方式



3. 指令系统设计与CPU运行控制

3.4.3 查询方式工作流程





3. 指令系统设计与CPU运行控制

3.4.3 STM32的串口收发接口

S3C2440全套中文手册.PDF

S3C2440A RISC 微处理器

UART

UART 发送缓冲寄存器 (保持寄存器和 FIFO 寄存器)

有 3 个 UART 发送缓冲状态寄存器，在 UART 模块中包含了 UTXH0 , UTXH1 和 UTXH2。UTXHn 发送数据为 8 位数据。

寄存器	地址	R/W	描述	复位值
UTXH0	0x50000020(L) 0x50000023(B)	W (字节)	UART 通道 0 发送缓冲寄存器	-
UTXH1	0x50004020(L) 0x50004023(B)	W (字节)	UART 通道 1 发送缓冲寄存器	-
UTXH2	0x50008020(L) 0x50008023(B)	W (字节)	UART 通道 2 发送缓冲寄存器	-

UTXHn	位	描述	初始状态
TXDATA _n	[7:0]	UART _n 要发送的数据	-

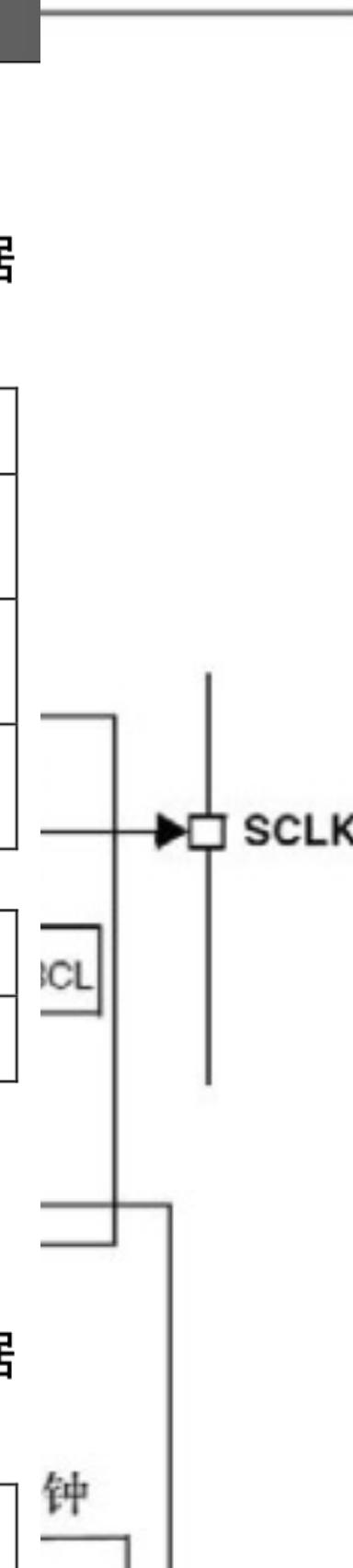
注意：(L)：小端模式；(B)：大端模式。

UART 接收缓冲寄存器 (保持寄存器和 FIFO 寄存器)

有 3 个 UART 接收缓冲状态寄存器，在 UART 模块中包含了 URXH0 , URXH1 和 URXH2。URXHn 发送数据为 8 位数据。

寄存器	地址	R/W	描述	复位值
URXH0	0x50000024(L) 0x50000027(B)	W (字节)	UART 通道 0 接收缓冲寄存器	-
URXH1	0x50004024(L) 0x50004027(B)	W (字节)	UART 通道 1 接收缓冲寄存器	-
URXH2	0x50008024(L) 0x50008027(B)	W (字节)	UART 通道 2 接收缓冲寄存器	-

URXHn	位	描述	初始状态
RXDATA _n	[7:0]	UART _n 接收到的数据	-





3. 指令系统设计与CPU运行控制

3.4.3 查询方式程序

```
4 void USART2_Init(void) {
5     // 1. Enable the clock for USART2 and GPIOA
6     RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
7     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);
8
9     // 2. Configure PA2 for USART2 TX, PA3 for USART2 RX
10    GPIO_InitTypeDef GPIO_InitStruct;
11    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_2; // TX
12    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;
13    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
14    GPIO_Init(GPIOA, &GPIO_InitStruct);
15
16    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_3; // RX
17    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
18    GPIO_Init(GPIOA, &GPIO_InitStruct);
```

```
20 // 3. Configure the USART2
21 USART_InitTypeDef USART_InitStruct;
22 USART_InitStructUSART_BaudRate = 9600;
23 USART_InitStructUSART_WordLength = USART_WordLength_8b;
24 USART_InitStructUSART_StopBits = USART_StopBits_1;
25 USART_InitStructUSART_Parity = USART_Parity_No;
26 USART_InitStructUSART_HardwareFlowControl = USART_HardwareFlowControl_None;
27 USART_InitStructUSART_Mode = USART_Mode_Rx | USART_Mode_Tx;
28 USART_Init(USART2, &USART_InitStruct);
29
30 // 4. Enable the USART2
31 USART_Cmd(USART2, ENABLE);
32 }
```

```
34 uint8_t USART2_Read(void) {
35     // Wait until data is received
36     while (!USART_GetFlagStatus(USART2, USART_FLAG_RXNE));
37     // Read and return the received data
38     return USART_ReceiveData(USART2);
39 }
40
41 int main(void) {
42     USART2_Init();
43
44     while (1) {
45         uint8_t data = USART2_Read();
46         // Do something with the received data
47     }
48 }
```

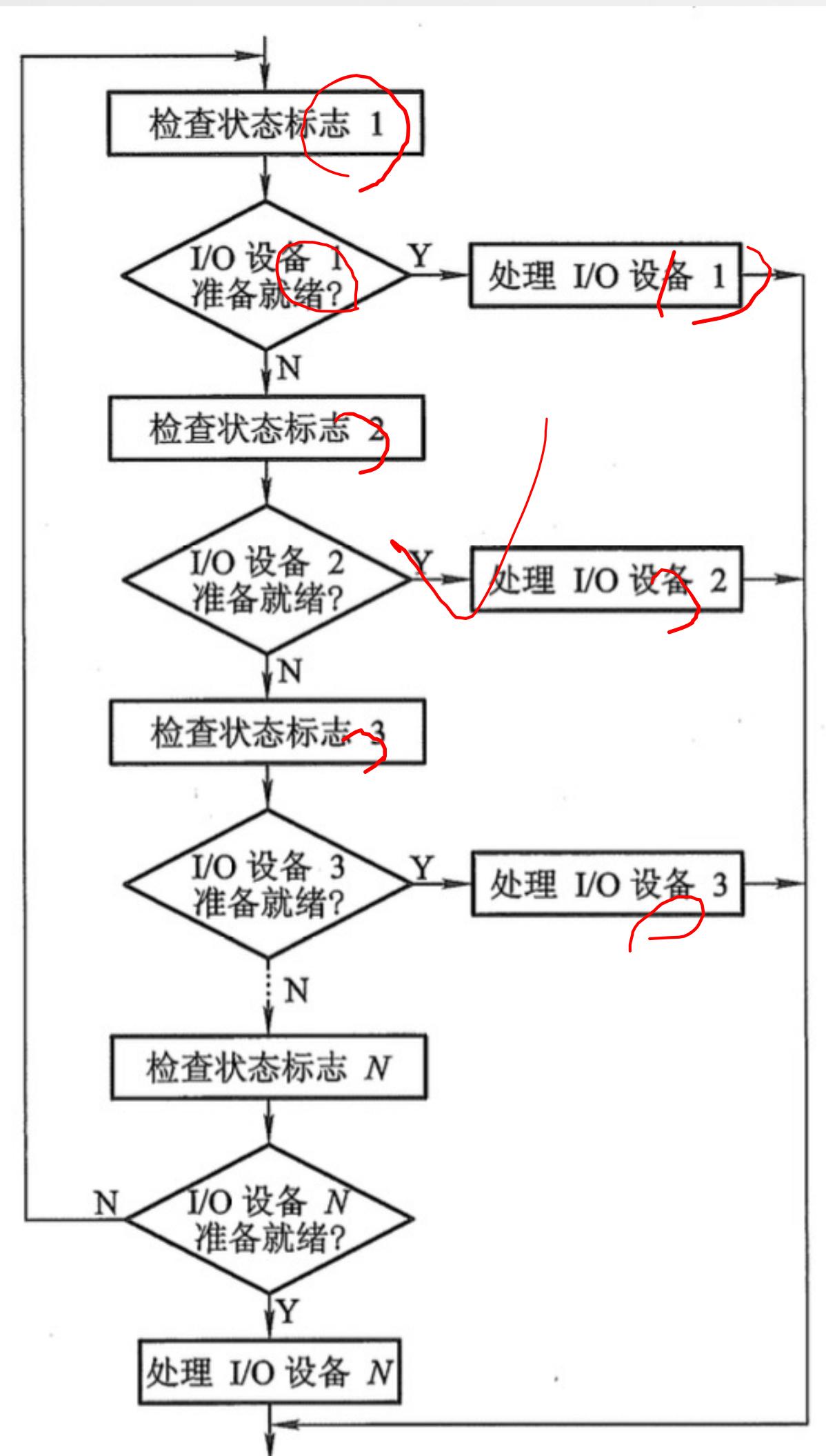




3. 指令系统设计与CPU运行控制

3.4.3 多个设备同时使用查询方式

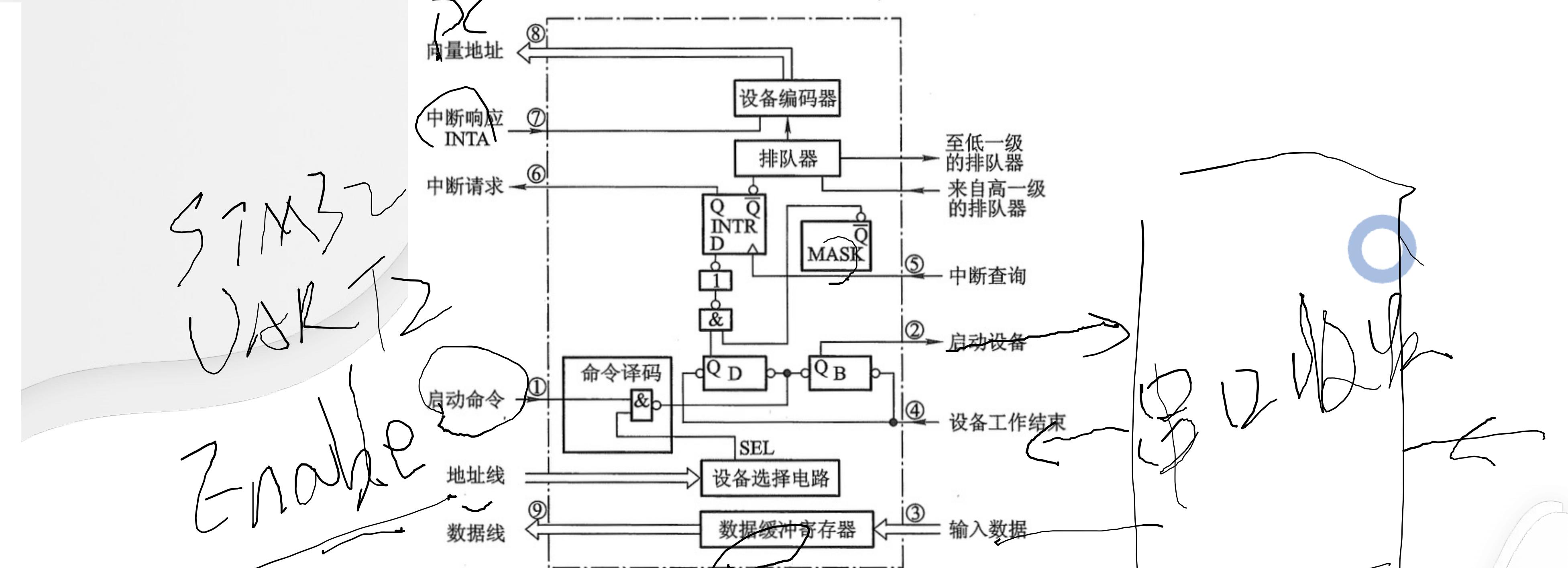
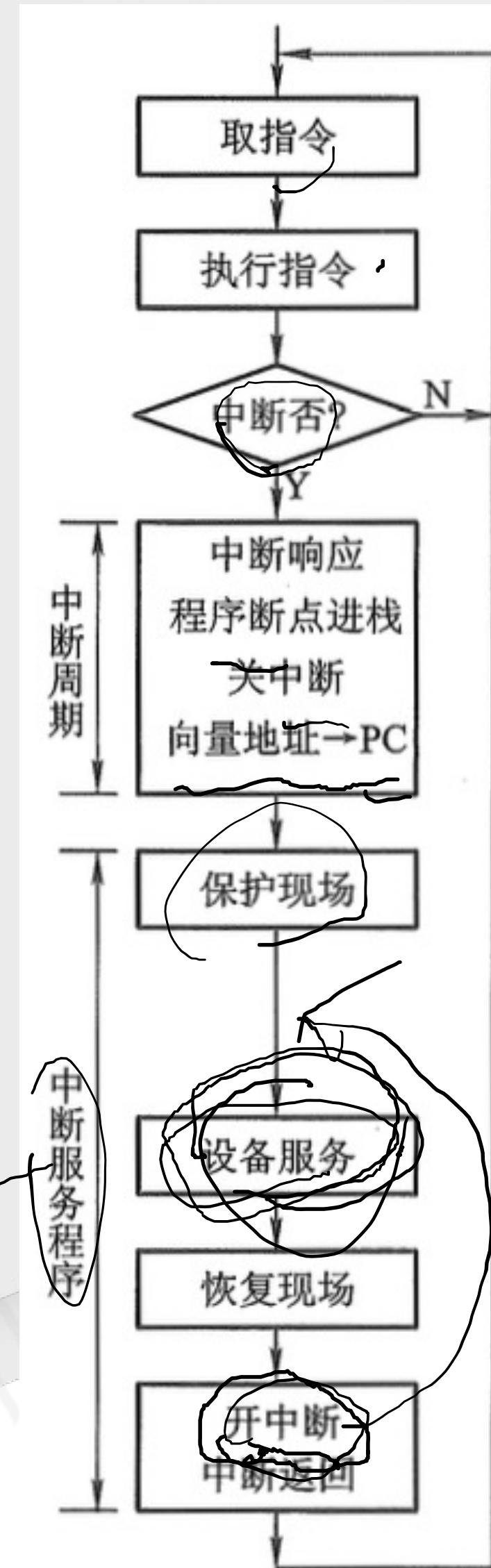
VJRT2





3. 指令系统设计与CPU运行控制

3.4.3 中断方式工作流程



3. 指令系统设计与CPU运行控制

3.4.3 STM32中的串口中断处理程序

```

void USART2_Init(void) {
    // 1. Enable the clock for USART2 and GPIOA
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA, ENABLE);

    // 2. Configure PA2 for USART2 TX, PA3 for USART2 RX
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_2; // TX
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOA, &GPIO_InitStruct);

    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_3; // RX
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_InitStruct);

    // 3. Configure the USART2
    USART_InitTypeDef USART_InitStruct;
    USART_InitStructUSART_BaudRate = 9600;
    USART_InitStructUSART_WordLength = USART_WordLength_8b;
    USART_InitStructUSART_StopBits = USART_StopBits_1;
    USART_InitStructUSART_Parity = USART_Parity_No;
    USART_InitStructUSART_HardwareFlowControl = USART_HardwareFlowControlNone;
    USART_InitStructUSART_Mode = USART_Mode_Rx | USART_Mode_Tx;
    USART_Init(USART2, &USART_InitStruct);
}

```

```

    // 4. Enable the USART2 receive interrupt
    USART_ITConfig(USART2, USART_IT_RXNE, ENABLE);

    // 5. Configure and enable the USART2 interrupt in NVIC
    NVIC_InitTypeDef NVIC_InitStruct;
    NVIC_InitStruct.NVIC_IRQChannel = USART2_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);

    // 6. Enable the USART2
    USART_Cmd(USART2, ENABLE);
}

```

```

void USART2_IRQHandler(void) {
    // Check if the USART2 receive interrupt flag was set
    if(USART_GetITStatus(USART2, USART_IT_RXNE) != RESET){
        uint8_t data = USART_ReceiveData(USART2);
        // Do something with the received data, e.g., store it in a buffer

        // (Optional) Clear the USART2 receive interrupt flag
        USART_ClearITPendingBit(USART2, USART_IT_RXNE);
    }
}

int main(void) {
    USART2_Init();

    while (1) {
        // Your main loop code here
        // The received data will be handled in the USART2_IRQHandler
    }
}

```



3.指令系统设计与CPU运行控制

3.4.3 STM32中的串口中断处理程序

```
1 #define USART2_BASE (0x40004400UL) // USART2 base address
2 #define GPIOA_BASE (0x40010800UL) // GPIOA base address
3 #define RCC_BASE (0x40021000UL) // RCC base address
4
5 #define USART_SR (*volatile uint32_t*)(USART2_BASE + 0x00) // Status register
6 #define USART_DR (*volatile uint32_t*)(USART2_BASE + 0x04) // Data register
7 #define USART_BRR (*volatile uint32_t*)(USART2_BASE + 0x08) // Baud rate register
8 #define USART_CR1 (*volatile uint32_t*)(USART2_BASE + 0x0C) // Control register 1
9 #define USART_CR2 (*volatile uint32_t*)(USART2_BASE + 0x10) // Control register 2
10 #define USART_CR3 (*volatile uint32_t*)(USART2_BASE + 0x14) // Control register 3
11
12 #define GPIOA_CRL (*volatile uint32_t*)(GPIOA_BASE + 0x00) // Port configuration register low
13 #define GPIOA_CRH (*volatile uint32_t*)(GPIOA_BASE + 0x04) // Port configuration register high
14
15 #define RCC_APB2ENR (*(volatile uint32_t*)(RCC_BASE + 0x18)) // APB2 peripheral clock enable register
16 #define RCC_APB1ENR (*(volatile uint32_t*)(RCC_BASE + 0x1C)) // APB1 peripheral clock enable register
```

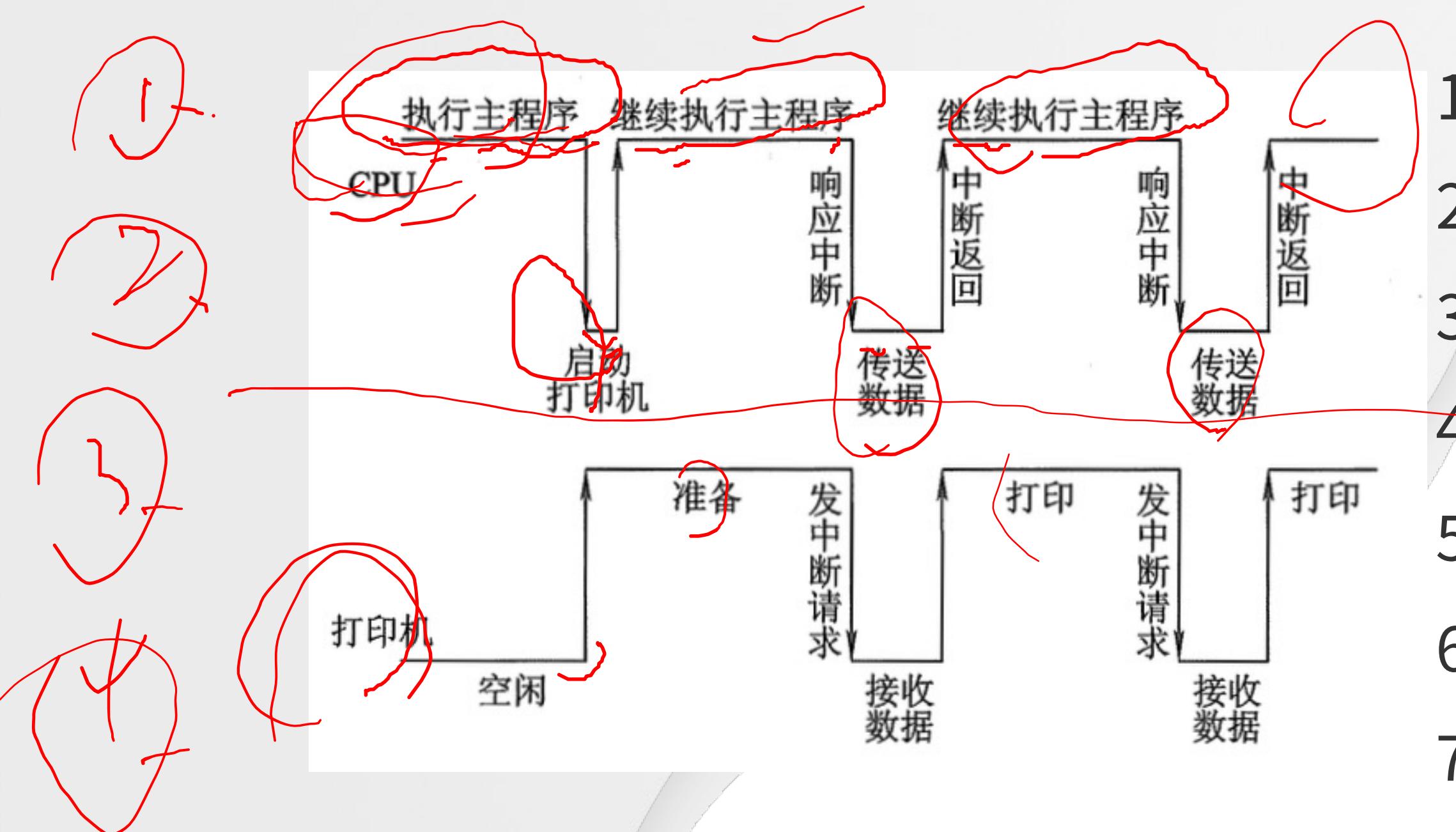
```
18 void USART2_Init(void) {
19     // Enable clock for GPIOA and USART2
20     RCC_APB2ENR |= (1 << 2); // Enable clock for GPIOA
21     RCC_APB1ENR |= (1 << 17); // Enable clock for USART2
22
23     // Configure PA2 as USART2 TX (Alternate Function Push-Pull)
24     // Configure PA3 as USART2 RX (Input Floating)
25     GPIOA_CRL &= ~(0xFF << 8); // Clear PA2 and PA3 settings
26     GPIOA_CRL |= (0xB << 8); // Set PA2 as AFPP and PA3 as Input Floating
27
28     // Configure USART2
29     USART2->BRR = 0x1D4C; // Set baud rate (assuming 36MHz clock, 9600 baud)
30     USART_CR1 |= (1 << 13); // USART enable
31     USART_CR1 |= (1 << 2); // Receiver enable
32     USART_CR1 |= (1 << 3); // Transmitter enable
33     USART_CR1 |= (1 << 5); // RXNE interrupt enable
34
35     // Enable USART2 interrupt in NVIC
36     NVIC_EnableIRQ(USART2_IRQn);
37 }
```

```
39 void USART2_IRQHandler(void) {
40     if (USART_SR & (1 << 5)) { // Check if RXNE is set
41         uint8_t data = USART_DR; // Read received data
42         // Handle the received data
43     }
44 }
45
46 int main(void) {
47     USART2_Init();
48     while (1) {
49         // Main loop
50     }
51 }
52 }
```



3. 指令系统设计与CPU运行控制

3.4.3 中断主要功能



1. 实现CPU与I/O设备的并行工作。
2. 处理硬件故障和软件错误
3. 实现人机交互，用户干预机器需要用到中断系统
4. 实现多道程序、分时操作、多道程序的切换需借助于中断系统。
5. 实时处理需要借助中断系统来实现快速响应
6. 实现应用程序和操作系统（内核程序）的切换，称为“软中断”
7. 多处理器系统中各处理器之间的信息交流和任务切换。



3. 指令系统设计与CPU运行控制



3.4.3 练习

例6：当有中断源发出请求时，CPU可执行相应的中断服务程序，可以提出中断的有

(1)。

①外部事件 ②Cache ③虚拟存储器失效 ④浮点数运算下溢 ⑤浮点数运算上溢

- A. ① ③ ④
- B. ① ⑤
- C. ① ② ⑤
- D. ① ③ ⑤

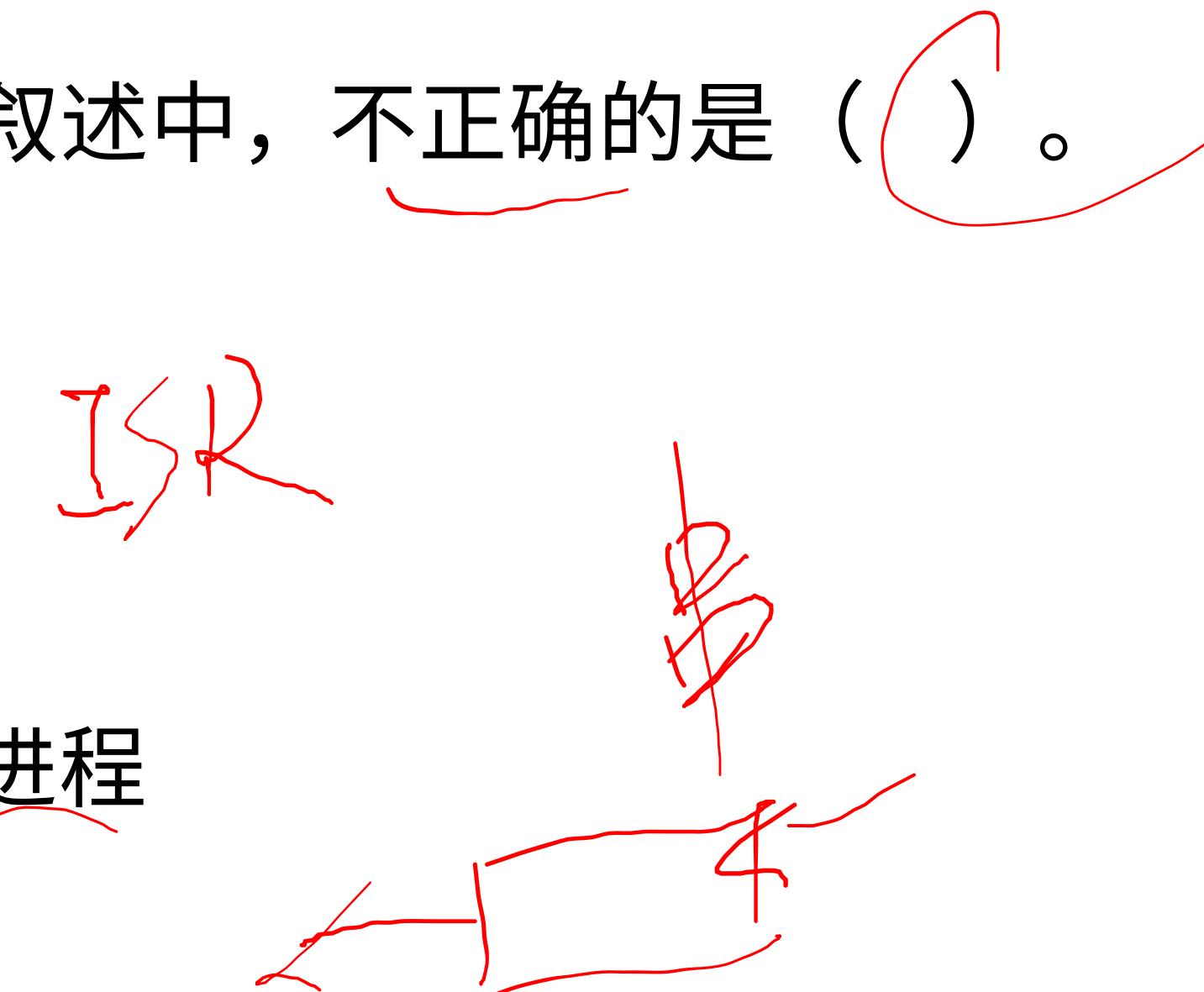


3. 指令系统设计与CPU运行控制

3.4.3 练习

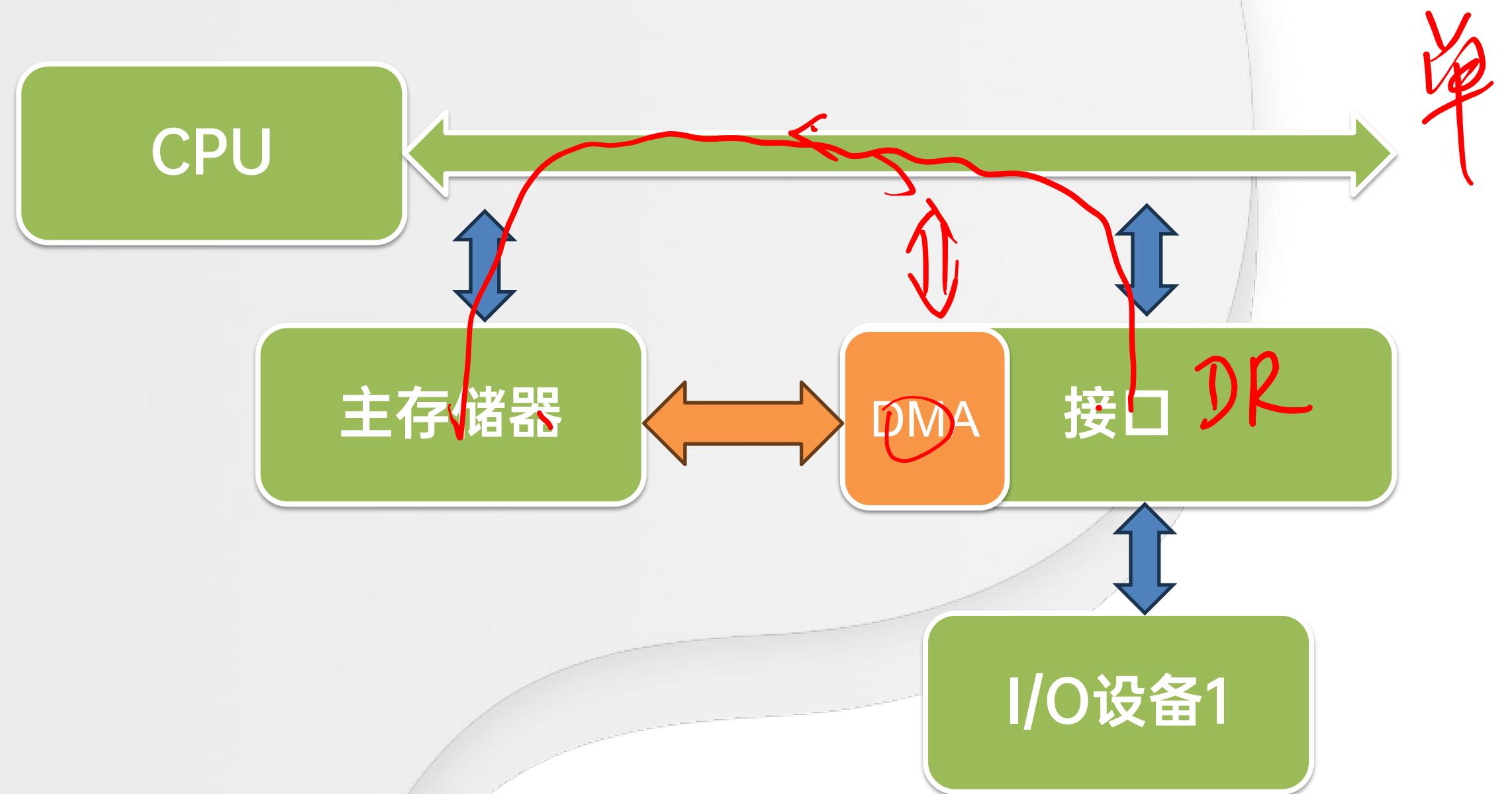
例7 【2022真题】：下列关于中断I/O方式的叙述中，不正确的是 ()。

- A. 适用于键盘、针式打印机等字符型设备
- B. 外设各主机之间的数据传送通过软件完成
- C. 外设准备数据的时间应小于中断处理时间
- D. 外设为某进程准备数据时CPU可运行其他进程



3. 指令系统设计与CPU运行控制

3.4.3 DMA方式





3. 指令系统设计与CPU运行控制

3.4.3 DMA方式实例阅读

S3C2440A RISC 微处理器

直接存储器存取 DMA

8

直接存储器存取 DMA

概述

S3C2440A 支持 4 通道处于系统总线和外设总线间的 DMA 控制器。DMA 控制器的每个通道都可以无限制的执行系统总线与/或外设总线之间设备的数据移动。换句话说，每个通道都可以处理以下 4 种情况：

- 1) 源和目标都在系统总线上
- 2) 当目标在外设总线上时源在系统总线上
- 3) 当目标在系统总线上时源在外设总线上
- 4) 源和目标都在外设总线上

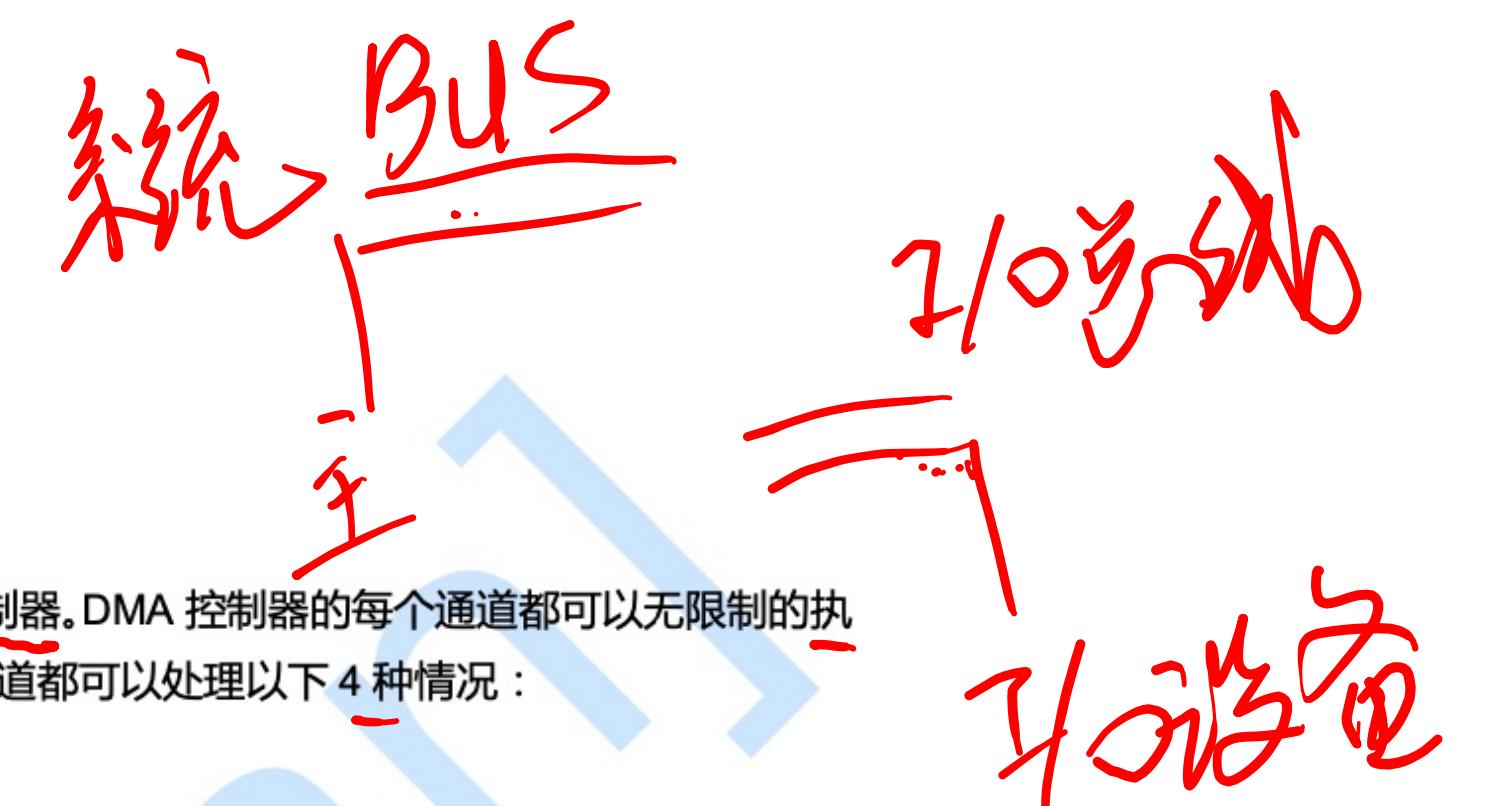
DMA 的主要优点是在无 CPU 干预的情况下能进行数据传输。DMA 的运行可以由软件开始，也可以来自内部外设或外部请求引脚的请求。

DMA 请求源

如果由 DCON 寄存器选择了硬件 DMA 请求模式，则 DMA 控制器的每个通道都可以在 4 个 DMA 源中选择 DMA 请求源的其中之一。（注意如果选择了软件请求模式，此 DMA 请求源没有一点意义）表 8-1 显示了每个通道的 4 个 DMA 源。

	源 0	源 1	源 2	源 3	源 4	源 5	源 6
通道 0	nXDREQ0	UART0	SDI	定时器	USB 设备 EP1	I2SSDO	PCMINT
通道 1	nXDREQ1	UART1	I2SSDI	SPI0	USB 设备 EP2	PCMOUT	SDI
通道 2	I2SSDO	I2SSDI	SDI	定时器	USB 设备 EP3	PCMINT	MICIN
通道 3	UART2	SDI	SPI1	定时器	USB 设备 EP4	MICIN	PCMOUT

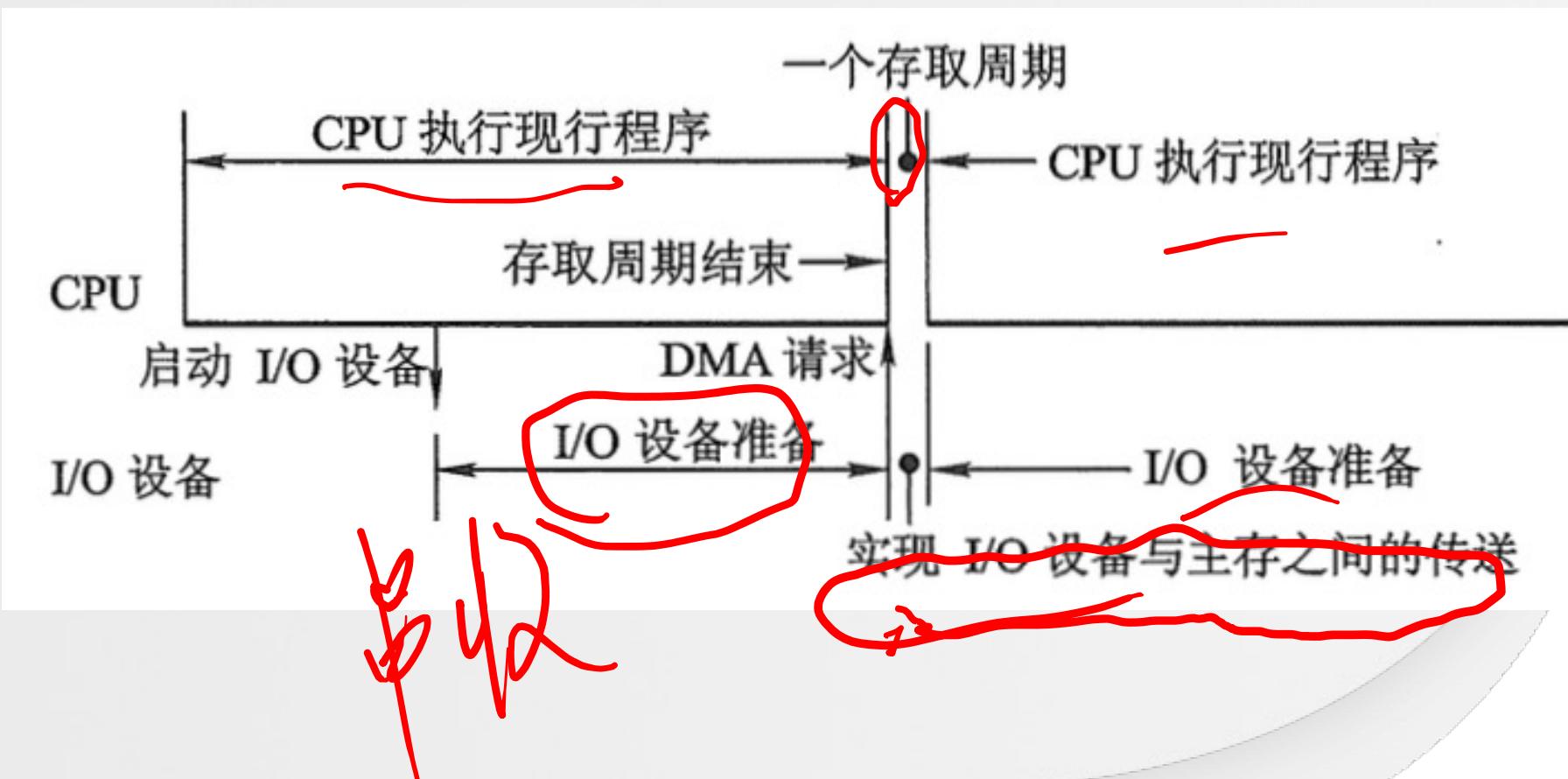
此处的 nXDREQ0 和 nXDREQ1 代表两个外部源（外部设备），I2SSDO 和 I2SSDI 分别代表 IIS 的传送和接收。





3. 指令系统设计与CPU运行控制

3.4.3 DMA方式

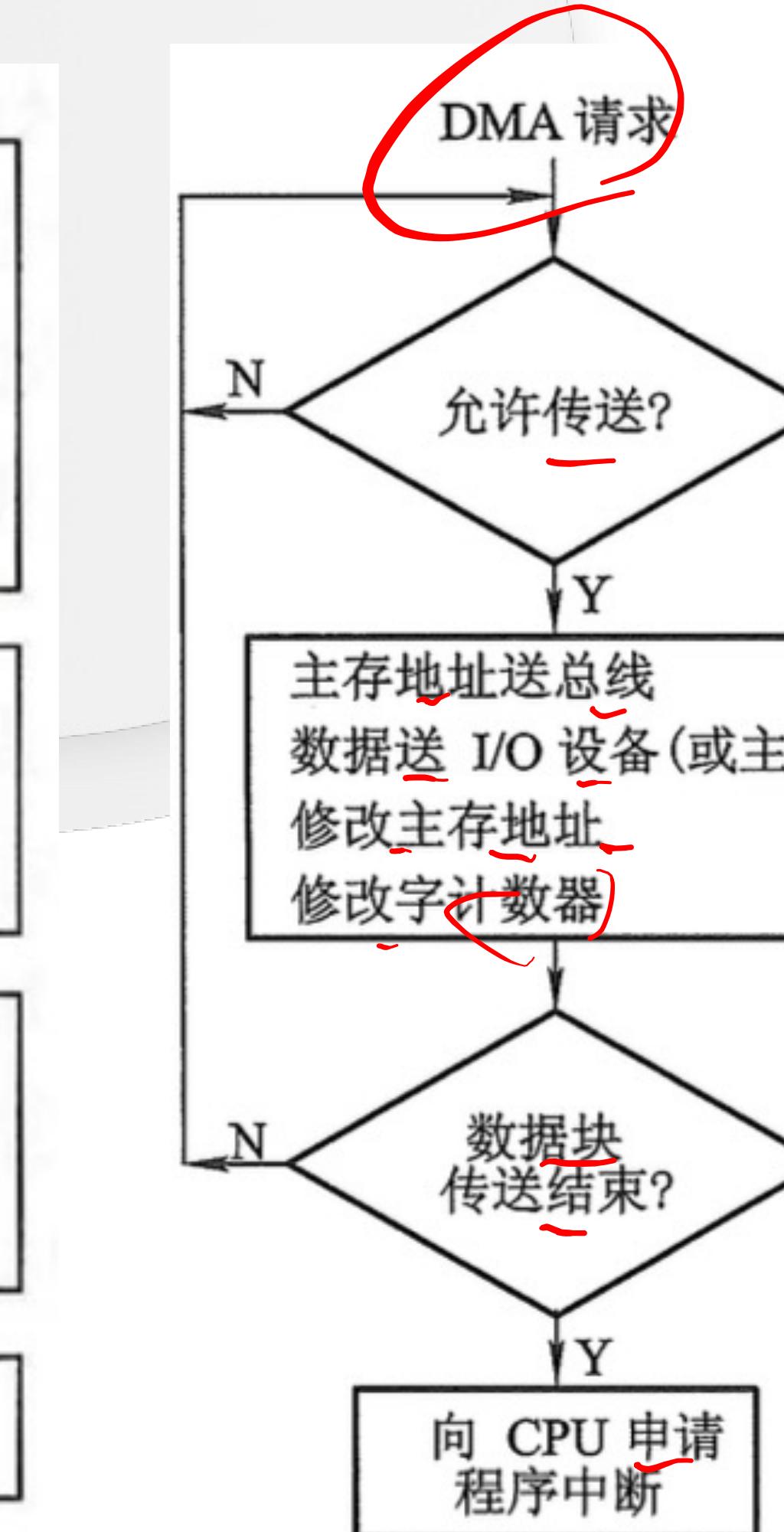
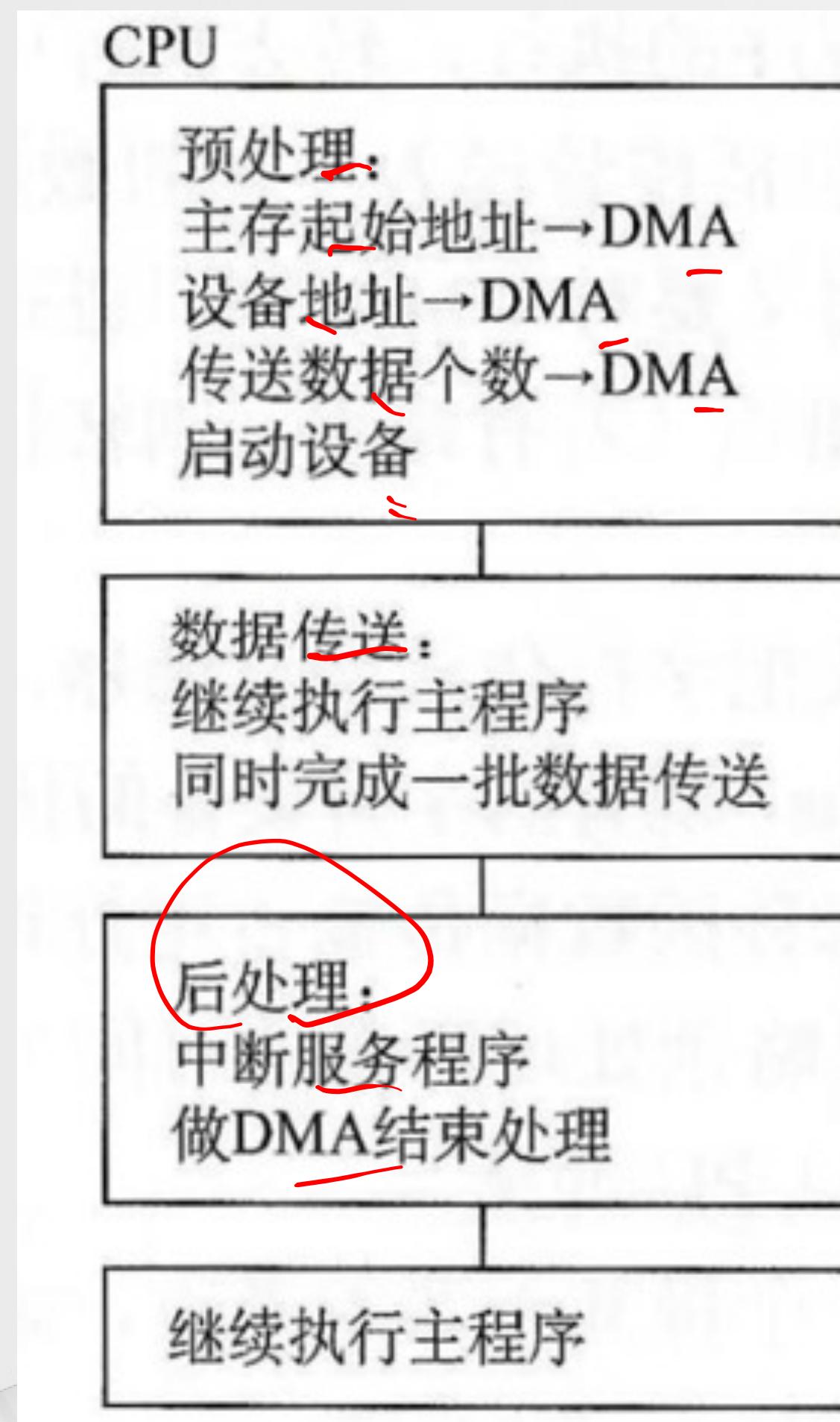


- 使主存与CPU的固定联系脱钩，主存既可被CPU访问，又可被外设访问
- 在数据块传递时，主存地址的确定和传送数据的计数都由硬件直接实现
- 主存中需要专用缓冲区，及时供给和接收数据
- DMA传递速度快，CPU和外设并行工作
- DMA在传送数据前要通过程序进行预处理，结束后要通过中断方式进行后处理。



3. 指令系统设计与CPU运行控制

3.4.3 DMA传送过程

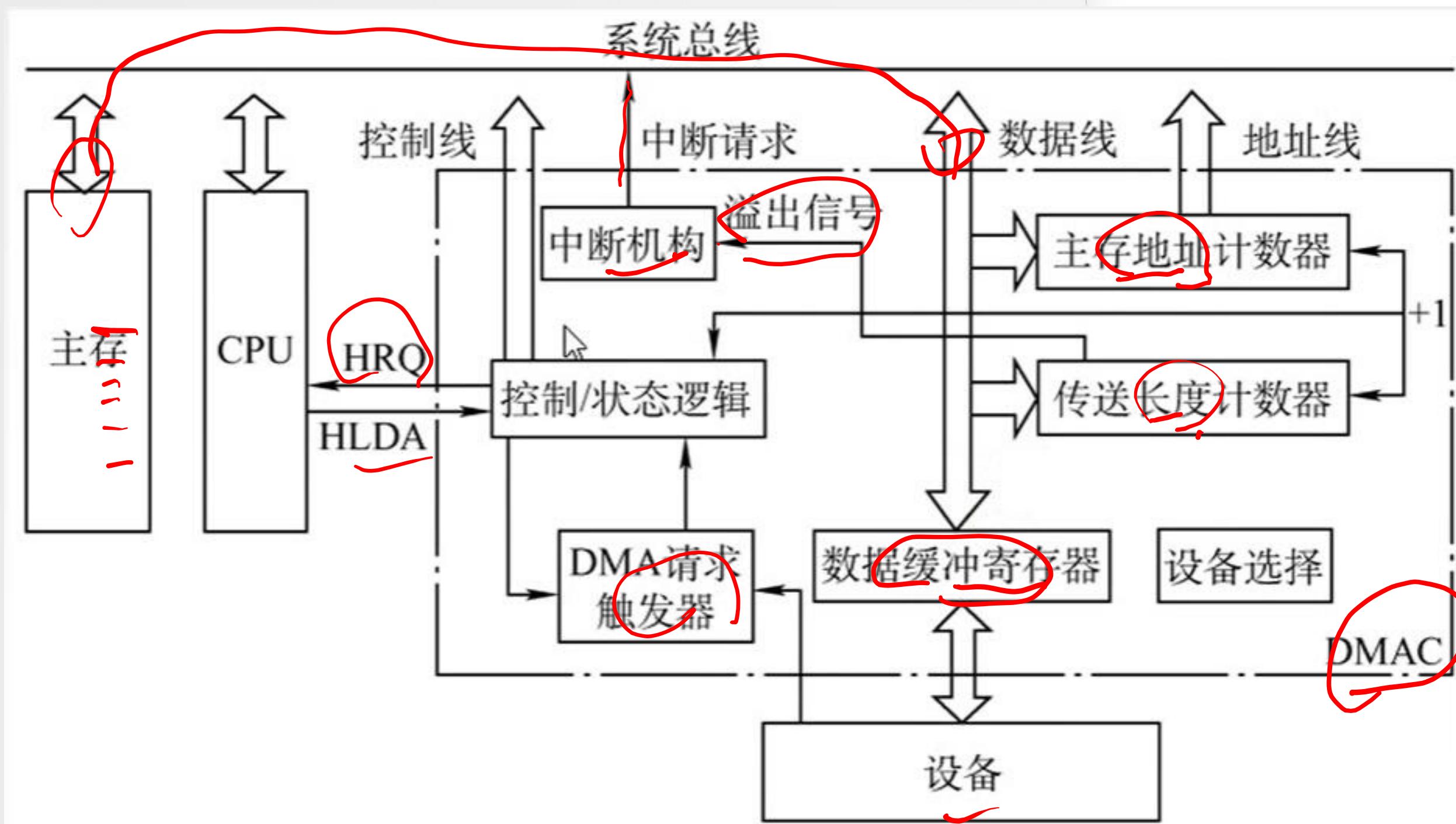


1. 数据传送是以数据块为单位
2. 首先由外部设备向DMA控制器发出请求，然后由DMA控制器向CPU发出总线请求
3. 当字计数器溢出（全0）时，表示一批数据交换完毕，由溢出信号通过中断机构向CPU提出中断请求。



3. 指令系统设计与CPU运行控制

3.4.3 DMA控制器的组成

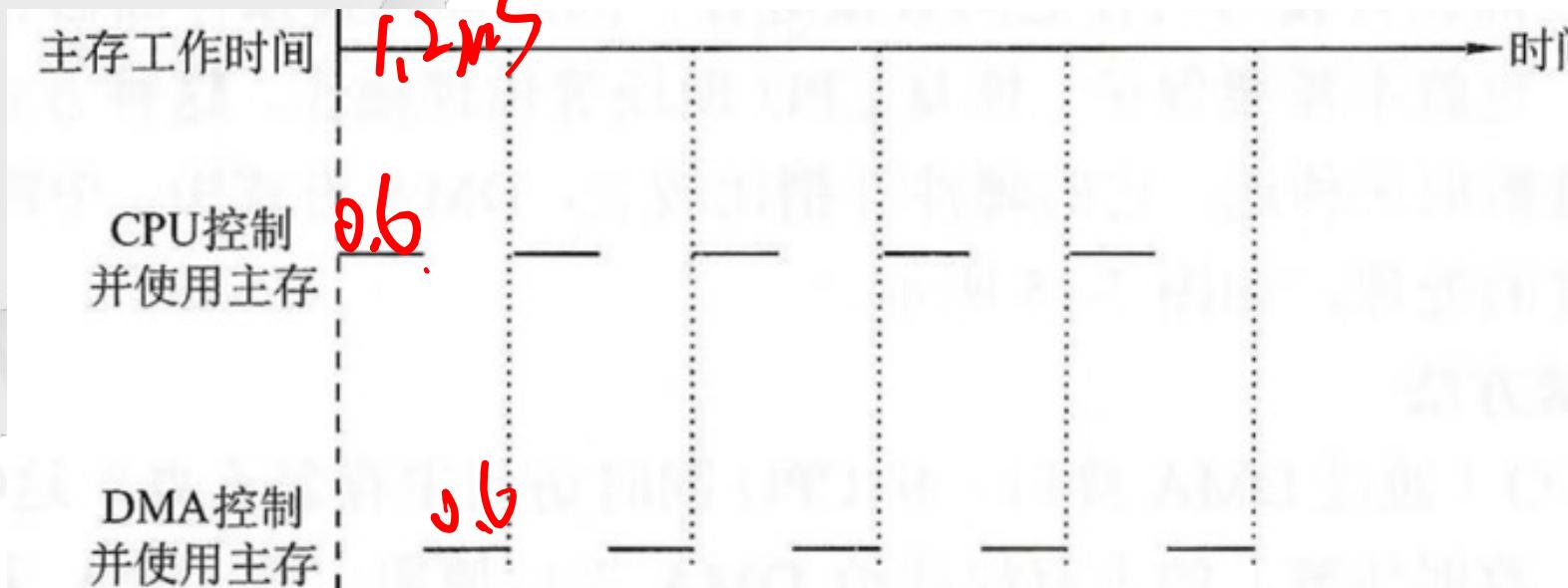
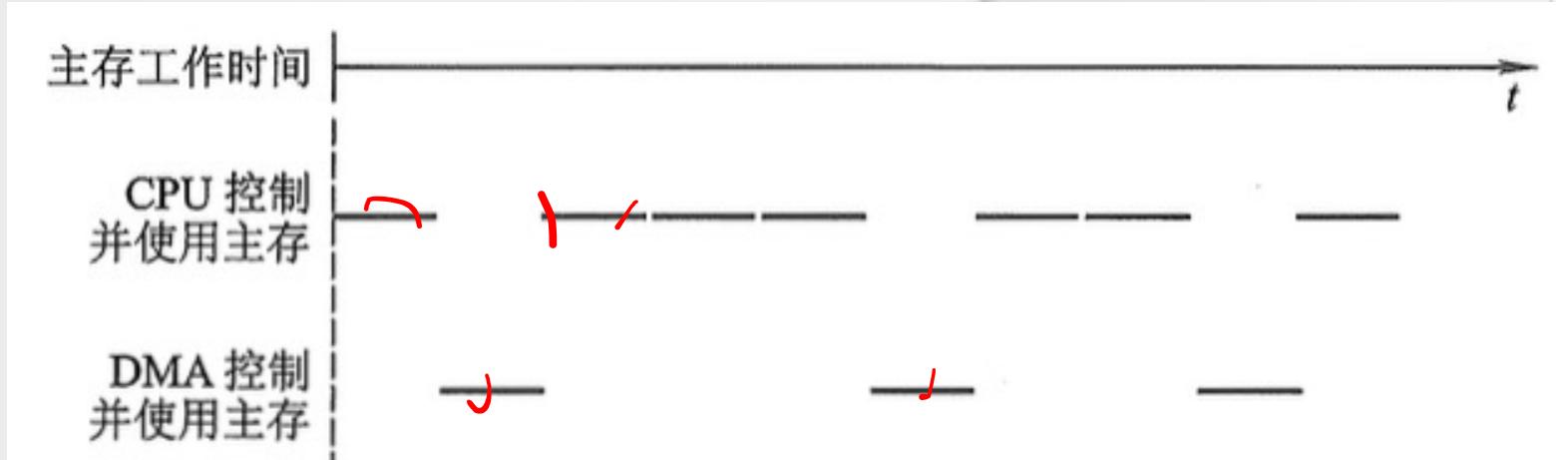




3. 指令系统设计与CPU运行控制

3.4.3 DMA传递数据的方式

- 停止CPU访问主存，DMA工作
- 周期挪用：I/O和CPU同时请求访存，出现冲突，CPU暂时放弃总线占有权，让I/O挪用一个或几个存储周期。
- DMA与CPU交替访问：适用于CPU的工作周期比主存存取周期长的情况





3. 指令系统设计与CPU运行控制



3.4.3 练习

例8 【2020真题】：若设备采用周期挪用DMA方式进行输入和输出，每次DMA传送的数据块大小为512字节，相应的I/O接口中有一个32位的数据缓冲寄存器。对于数据输入过程，下列叙述中，错误的是（ ）。

- A. 每准备好32位数据， DMA控制器就发出一次总线请求
- B. 相对于CPU， DMA控制器的总线使用权的优先级更高
- C. 在整个数据块的传送过程中， CPU不可以访问主存储器 X
- D. 数据块传送结束时，会产生“DMA传送结束”中断请求



3. 指令系统设计与CPU运行控制



3.4.3 中断方式和DMA方式主要区别

ISR PC

PC

1. 中断方式是程序的切换；DMA方式不中断现行程序
2. 中断请求的响应只能发生在执行结束，而对DMA的响应可以生在做任意一个机器周期结束时（取指、间址、执行周期后）CU
3. 中断传送数据需要CPU处理，DMA由硬件完成
4. DMA请求的优先级高于中断
5. 中断能处理异常，DMA只是进行大批量数据传送



3. 指令系统设计与CPU运行控制

3.4.3 控制方式对比

单
JR
32位



	处理过程	CPU参与度	处理单位	数据流向
查询	CPU发出I/O命令后要不断轮询状态	极高	字	设备→CPU→内存 内存→CPU→设备
中断	CPU发出I/O命令后继续完成原来的程序运行，本次I/O完成后设备会发中断请求	高	字	设备→CPU→内存 内存→CPU→设备
DMA	CPU发出I/O命令后继续完成原来的程序运行，本次I/O完成后DMA控制器发出中断信号	中	块	设备→内存 内存→设备
通道控制	CPU发出I/O命令后继续完成原来的程序运行，通道会执行通道程序完成I/O，完成后通道向CPU发出中断信号	低	一组块	设备→内存 内存→设备



3.指令系统设计与CPU运行控制



3.4.3 练习

例9 【2009真题】：某计算机的CPU主频为500MHz，CPI为5（即执行每条指令平均需5个时钟周期）。假定某外设的数据传输率为0.5MB/s，采用中断方式与主机进行数据传送，以32位为传输单位，对应的中断服务程序包含18条指令，中断服务的其他开销相当于2条指令的执行时间。回答下列问题，要求给出计算过程。

- 1) 在中断方式下，CPU用于该外设I/O的时间占整个CPU时间的百分比是多少？
- 2) 当该外设的数据传输率达到5MB/s时，改用DMA方式传输数据。假定每次DMA传送块大小为5000B，且DMA预处理和后处理的总开销为500个时钟周期，则CPU用于该外设I/O的时间占整个CPU时间的百分比是多少（假设DMA与CPU之间没有访存冲突）？



3. 指令系统设计与CPU运行控制

3.4.3 练习

例9【2009真题】：某计算机的CPU主频为500MHz，CPI为5（即执行每条指令平均需5个时钟周期）。假定某外设的数据传输率为0.5MB/s，采用中断方式与主机进行数据传送，以32位为传输单位，对应的中断服务程序包含18条指令，中断服务的其他开销相当于2条指令的执行时间。回答下列问题，要求给出计算过程。

1) 在中断方式下，CPU用于该外设I/O的时间占整个CPU时间的百分比是多少？

多少次中断.

$$20 \text{ 条} \times 5 = 100 \text{ 个时钟周期}.$$

每个中断消耗.

$$0.5 \text{ MB} / 4 \text{ B} = 125000.$$

1/s 处理1同时.

$$125000 \times 100 = 12.5 \text{ 从} 1 \text{ 时钟周期}.$$

$$12.5 / 500 = 2.5\%.$$



3. 指令系统设计与CPU运行控制

3.4.3 练习

例9 【2009真题】：某计算机的CPU主频为500MHz，CPI为5（即执行每条指令平均需5个时钟周期）。假定某外设的数据传输率为0.5MB/s，采用中断方式与主机进行数据传送，以32位为传输单位，对应的中断服务程序包含18条指令，中断服务的其他开销相当于2条指令的执行时间。回答下列问题，要求给出计算过程。

2) 当该外设的数据传输率达到 $\underline{5\text{MB/s}}$ 时，改用DMA方式传输数据。假定每次DMA传送块大小为 $\underline{5000\text{B}}$ ，且DMA预处理和后处理的总开销为 $\underline{500}$ 个时钟周期，则CPU用于该外设I/O的时间占整个CPU时间的百分比是多少（假设DMA与CPU之间没有访存冲突）？

$$\text{每秒DMA次数} = \frac{5\text{MB}}{5000\text{B}} = 1000,$$

$$\text{总I/O时间} = 500 \times 1000 = 0.5\text{M}$$

$$0.5\text{M} / 500\text{M} = 0.1\%.$$



3. 指令系统设计与CPU运行控制

3.4 本节总结

1. I/O设备通过接口、通道或处理机的方式接入系统，具备更强的并行处理能力和可扩展性
2. 有了I/O接口或通道，设备的驱动和控制主要是针对接口或通道进行编程，接口和通道内有相应的寄存器可以访问，这些寄存器都有对应的地址，编址方式在不同CPU架构上有所区别
3. CPU在和I/O设备通信时，可以采用查询（轮询），中断，DMA处理方式，后两种的效率相对高一些，大部分设备都会使用到中断处理方式，DMA处理适用于大数据量突出性传送需求。

欢迎参与学习

WELCOME FOR YOUR JOINING

船说：计算机基础