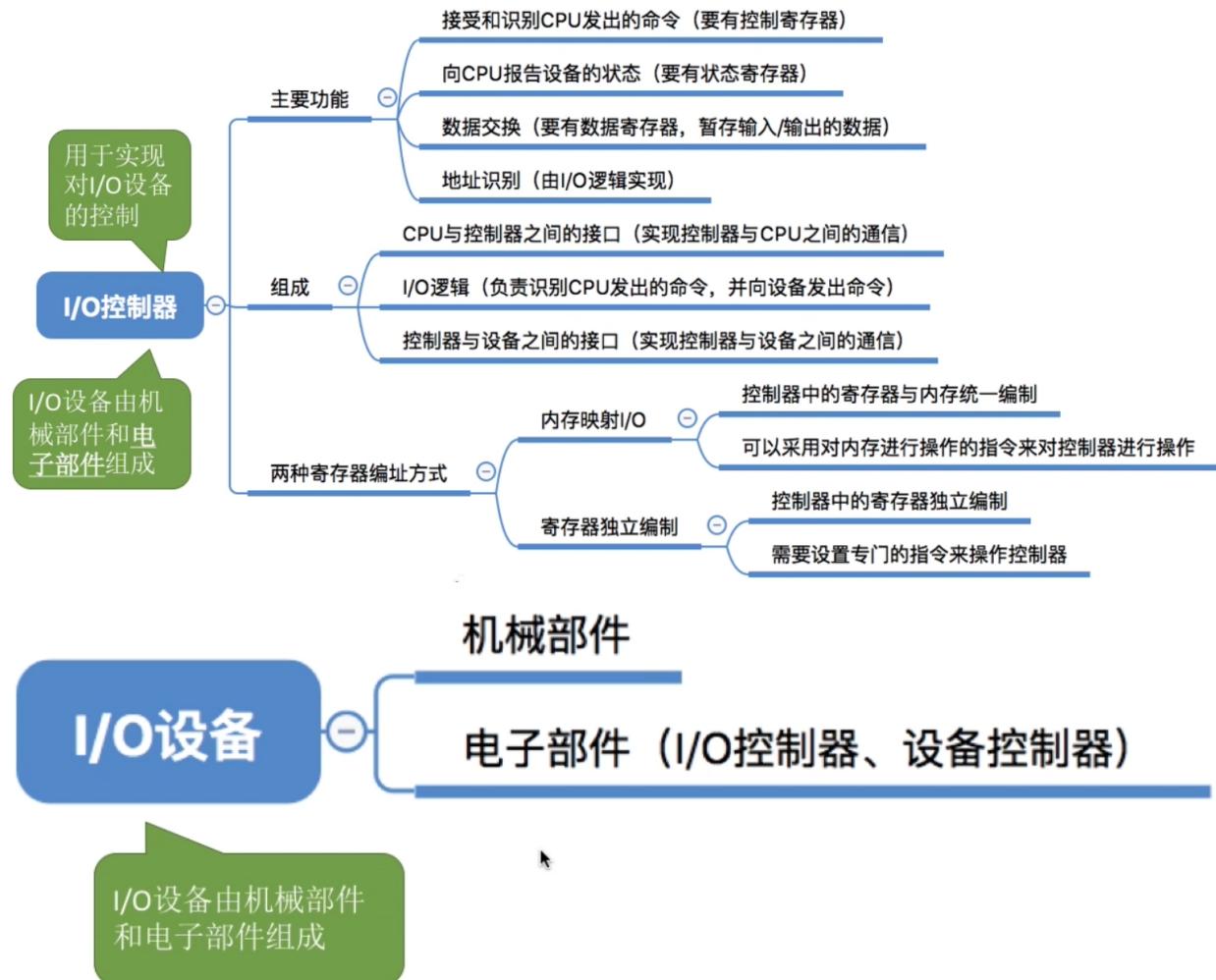


I/O控制方式

I/O控制器



I/O设备的机械部件主要用来执行具体I/O操作。

如我们看得见摸得着的鼠标/键盘的按钮；显示器的LED屏；移动硬盘的磁臂、磁盘盘面。

I/O设备的电子部件通常是一块插入主板扩充槽的印刷电路板。

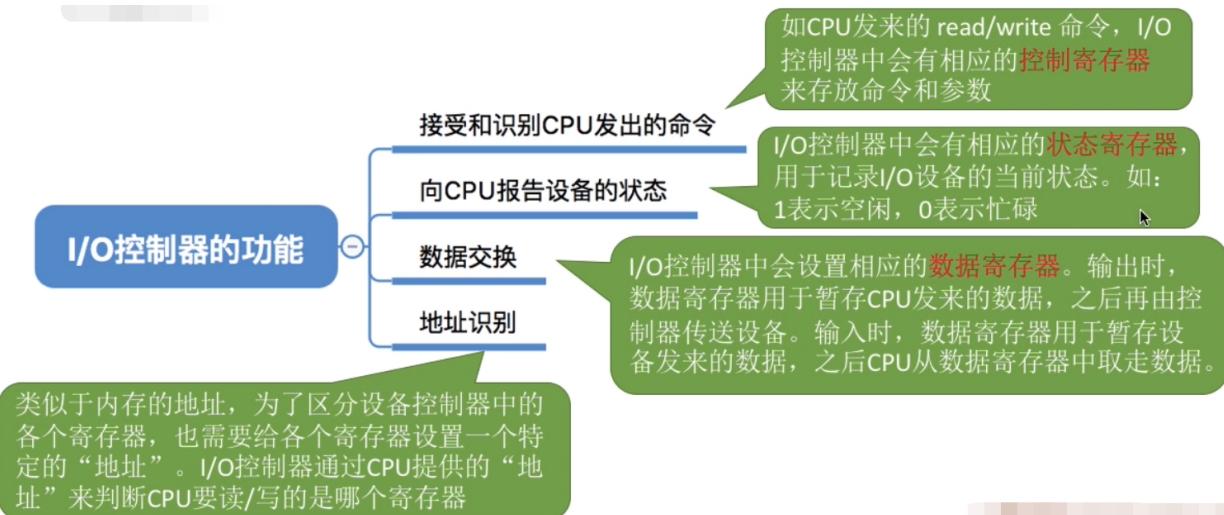
I/O设备的机械部件



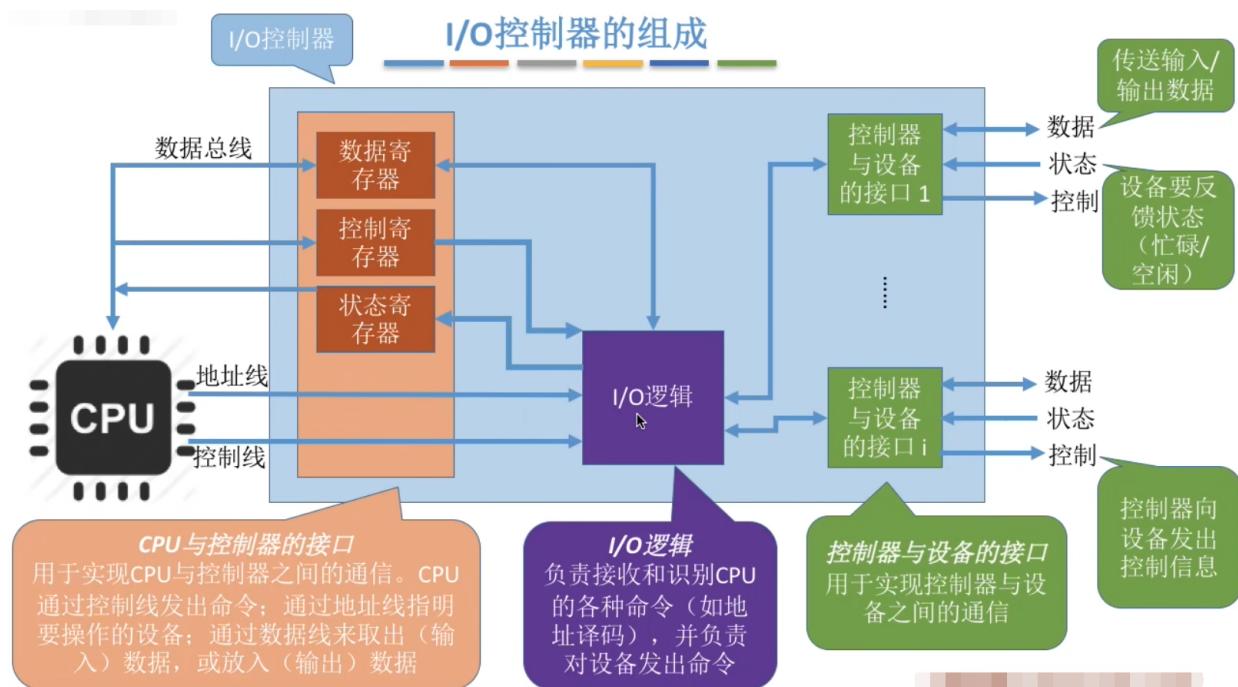
I/O设备的电子部件 (I/O控制器)

CPU无法直接控制I/O设备的机械部件，因此I/O设备还要有一个电子部件作为CPU和I/O设备机械部件之间的“中介”，用于实现CPU对设备的控制。

这个电子部件就是I/O控制器，又称设备控制器。CPU可控制I/O控制器，又由I/O控制器来控制设备的机械部件。

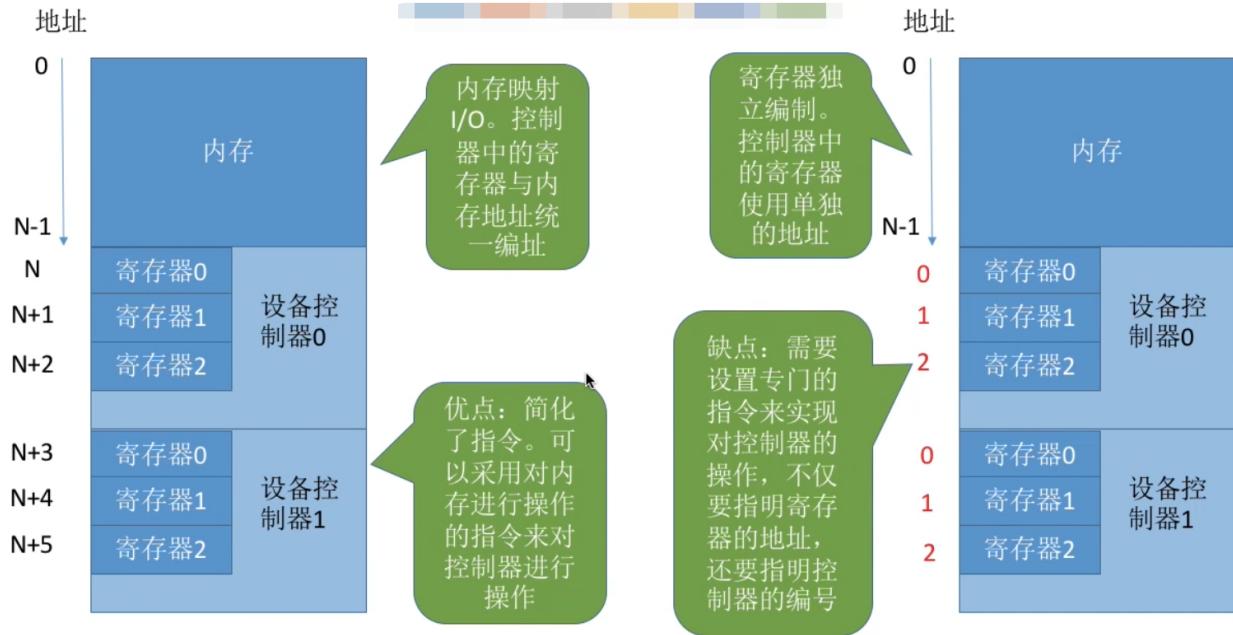


I/O控制器的组成



1. 一个I/O控制器可能会对应多个设备；
2. 数据寄存器、控制寄存器、状态寄存器可能有多个（如：每个控制/状态寄存器对应一个具体的设备），且这些寄存器都要有相应的地址，才能方便CPU操作。有的计算机会让这些寄存器占用内存地址的一部分，称为内存映像I/O；另一些计算机则采用I/O专用地址，即寄存器独立编址。

内存映像I/O 对比 寄存器独立编址



I/O控制方式

	完成一次读/写的过程	CPU干预频率	每次I/O的数据传输单位	数据流向	优缺点
程序直接控制方式	CPU发出I/O命令后需要不断轮询	极高	字	设备→CPU→内存 内存→CPU→设备	每一个阶段的优点都是解决了上一阶段的最大缺点。 总体来说, 整个发展过程就是要尽量减少CPU对I/O过程的干预, 把CPU从繁杂的I/O控制事务中解脱出来, 以便更多地去完成数据处理任务。
中断驱动方式	CPU发出I/O命令后可以做其他事, 本次I/O完成后设备控制器发出中断信号	高	字	设备→CPU→内存 内存→CPU→设备	
DMA方式	CPU发出I/O命令后可以做其他事, 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	
通道控制方式	CPU发出I/O命令后可以做其他事。通道会执行通道程序以完成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	

难点理解:
通道=弱鸡版CPU
通道程序=任务清单

I/O控制方式

即：用什么样的方式来控制I/O设备的数据读/写

程序直接控制方式

中断驱动方式

DMA方式

通道控制方式

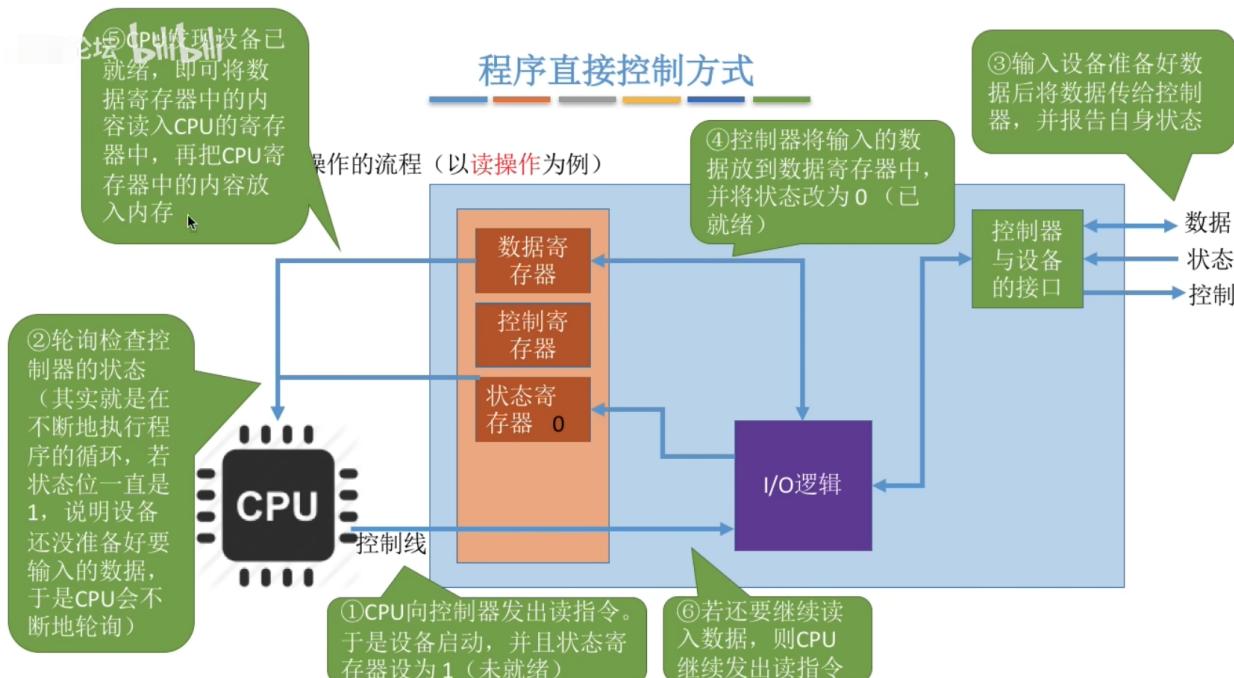
需要注意的问题：

1. 完成一次读/写操作的流程；
2. CPU干预的频率；
3. 数据传送的单位；
4. 数据的流向；
5. 主要缺点和主要优点。

程序直接控制方式

Key word: 轮询

完成一次读/写操作的流程（以读操作为例）



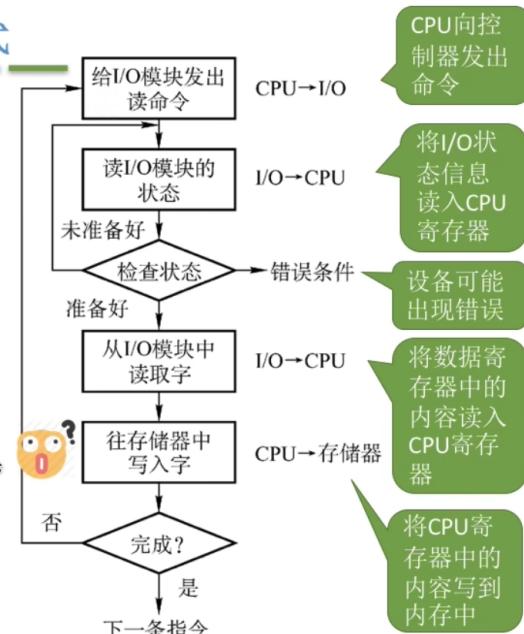
程序直接控制方式

1. 完成一次读/写操作的流程 (见右图, Key word: 轮询)

```
01. #include <stdio.h>
02. #include <stdlib.h>
03. int main()
04. {
05.     int a, b, c, d;
06.     scanf("%d", &a); //输入整数并赋值给变量a
07.     scanf("%d", &b); //输入整数并赋值给变量b
08.     printf("a+b=%d\n", a+b); //计算a+b的值
09.     scanf("%d %d", &c, &d); //输入两个整数并分别赋值给c、d
10.     printf("c*d=%d\n", c*d); //计算c*d的值
11.
12.     system("pause");
13.     return 0;
14. }
```

输入的数据最终要放到内存中 (a/b/c/d 变量存放在内存中)

同理, 输出的数据也存放在内存中, 需要从内存取出



(a) 程序直接控制方式

CPU干预的频率

很频繁, I/O操作开始之前、完成之后需要CPU介入, 并且在等待I/O完成的过程中CPU需要不断地轮询检查。

数据传送的单位

每次读/写一个字

数据流向

读操作 (数据输入) : I/O设备->CPU寄存器->内存

写操作 (数据输出) : 内存->CPU寄存器->I/O设备

每个字的读/写都需要CPU的帮助

主要优点和主要缺点

• 优点

实现简单。在读/写指令之后, 加上实现循环检查的一系列指令即可 (因此才称为“程序直接控制方式”)

• 缺点

CPU和I/O设备只能串行工作, CPU需要一直轮询检查, 长期处于“忙等状态”, CPU利用率低。

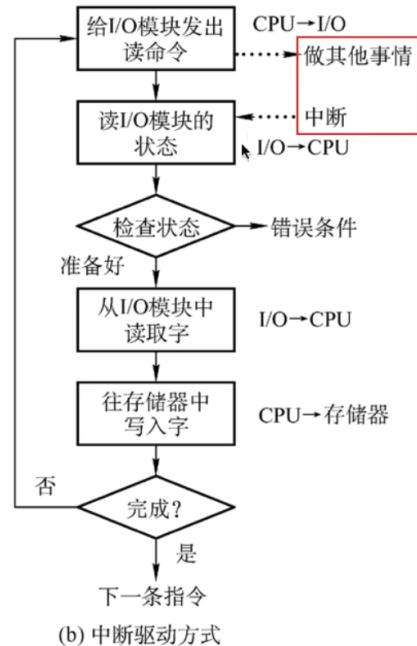
中断驱动方式

完成一次读/写操作的流程

中断驱动方式

引入**中断机制**。由于I/O设备速度很慢，因此在CPU发出读/写命令后，可将等待I/O的进程阻塞，先切换到别的进程执行。当I/O完成后，控制器会向CPU发出一个中断信号，CPU**检测到中断信号后**，会保存当前进程的运行环境信息，转去执行中断处理程序处理该中断。处理中断的过程中，CPU从I/O控制器读一个字的数据传送到CPU寄存器，再写入主存。接着，**CPU恢复等待I/O的进程（或其他进程）的运行环境，然后继续执行。**

注意：①CPU会在每个指令周期的末尾检查中断；
 ②中断处理过程中需要保存、恢复进程的运行环境，这个过程是需要一定时间开销的。可见，如果中断发生的频率太高，也会降低系统性能。



(b) 中断驱动方式

CPU干预的频率

每次I/O操作开始之前、完成之后需要CPU介入。

等待I/O完成的过程中CPU可以切换到别的进程执行。

数据传送的单位

每次读/写一个字

数据流向

读操作（数据输入）：I/O设备->CPU寄存器->内存

写操作（数据输出）：内存->CPU寄存器->I/O设备

主要优点和主要缺点

- 优点

与程序直接控制方式相比，在中断驱动方式中，I/O控制器会通过中断信号主动报告I/O已完成，CPU不再需要不停地轮询。CPU和I/O设备可并行工作，CPU利用率得到明显提升。

- 缺点

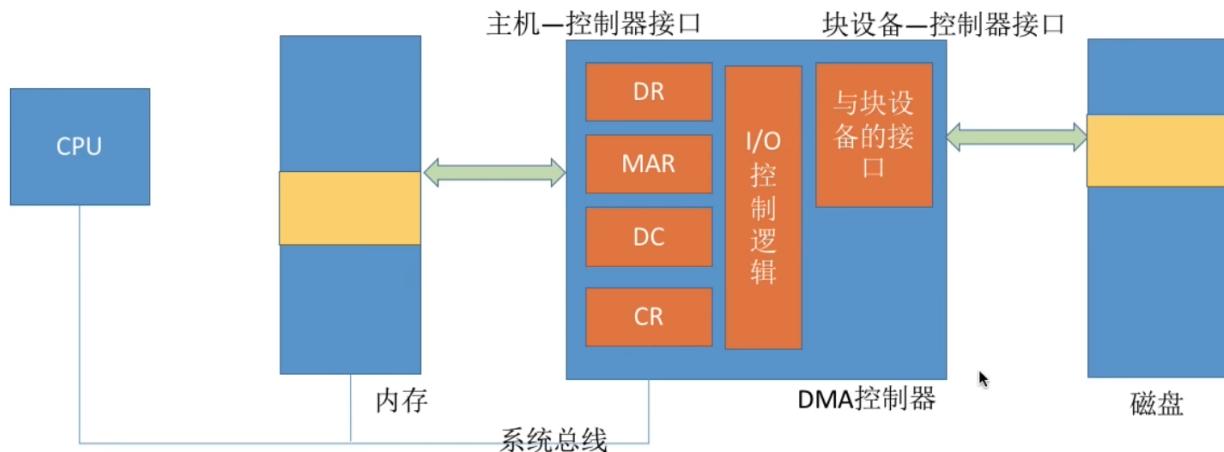
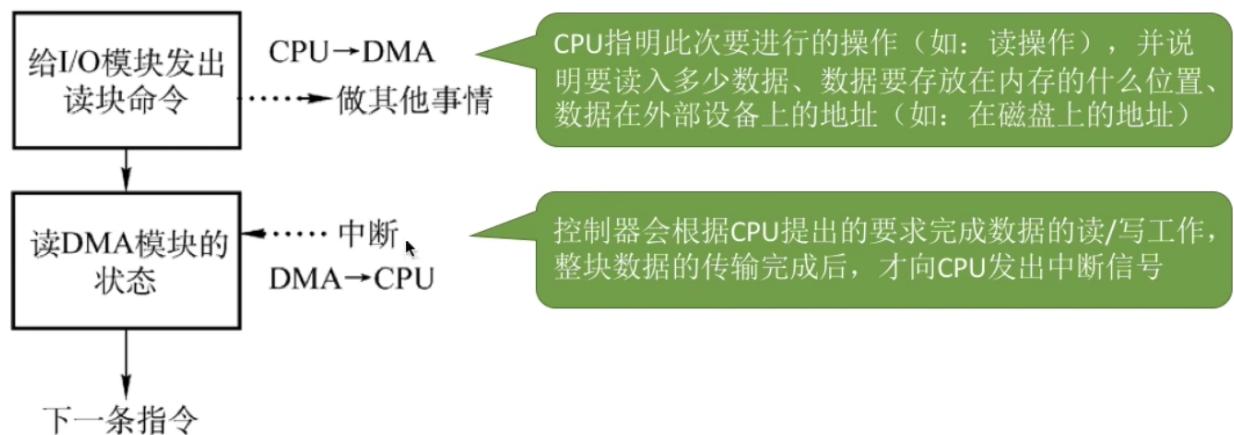
每个字在I/O设备与内存之间的传输，都需要经过CPU。而频繁地中断处理会消耗较多的CPU时间。

DMA方式

与中断驱动方式相比，DMA方式（Direct Memory Access，直接存储器存取。主要用于块设备的I/O控制）有这样几个改进：

1. 数据的传送单位是“块”。不再是一个字、一个字的传送；
2. 数据的流向是从设备直接放入内存，或者从内存直接到设备。不再需要CPU作为“快递小哥”。
3. 仅在传送一个或多个数据块的开始和结束时，才需要CPU干预。

完成一次读/写操作的流程



DR(Data Register, 数据寄存器): 暂存从设备到内存, 或从内存到设备的数据。

MAR(Memory Address Register, 内存地址寄存器): 在输入时, MAR表示数据应放到内存中的什么位置; 输出时MAR表示要输出的数据放在内存中的什么位置。

DC(Data Counter, 数据计数器): 表示剩余要读/写的字节数。

CR(Command Register, 命令/状态寄存器): 用于存放CPU发来的I/O命令, 或设备的状态信息。

CPU干预的频率

仅在传送一个或多个数据块的开始和结束时, 才需要CPU干预。

数据传送的单位

每次读/写一个字或多个块 (注意: 每次读写的只能是连续的多个块, 且这些块读入内存后在内存中也必须是连续的)

数据流向

不再需要经过CPU

读操作 (数据输入) : I/O设备->内存

写操作 (数据输出) : 内存->I/O设备

主要优点和主要缺点

- 优点

数据传输以“块”为单位，CPU接入频率进一步降低。数据的传输不再需要先经过CPU再写入内存，数据传输效率进一步增加。CPU和I/O设备的并行性得到提升。

- 缺点

CPU每发出一条I/O指令，只能读/写一个或多个连续的数据块。

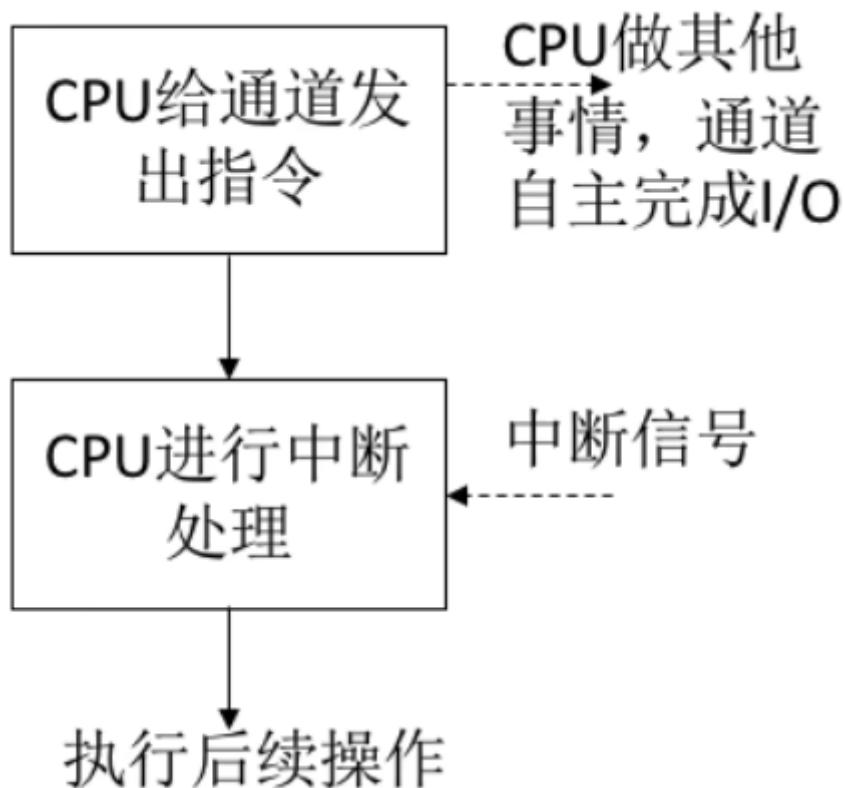
如果要读/写多个离散存储的数据块，或者要将数据分别写到不同的内存区域时，CPU要分别发出多条I/O指令，进行多次中断处理才能完成。

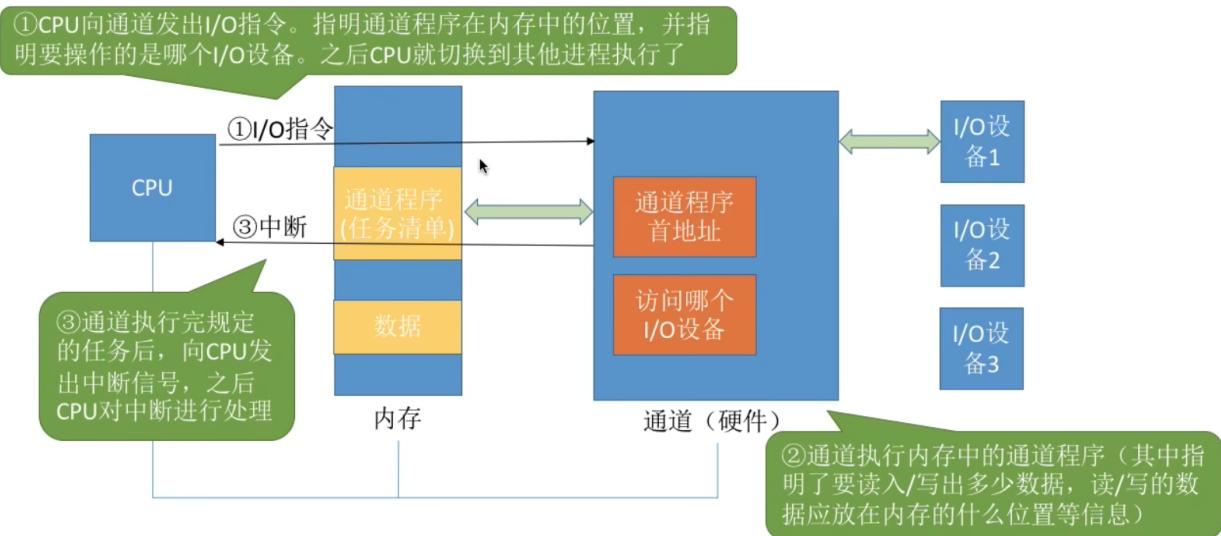
通道控制方式

通道：一种硬件，可以理解为是“弱鸡版的CPU”。通道可以识别并执行一系列通道指令

与CPU相比，通道可以执行的指令很单一，并且通道程序是放在主机内存中的，也就是说通道与CPU共享内存

完成一次读/写操作的流程





CPU干预的频率

极低，通道会根据CPU的指示执行相应的通道程序，只有完成一组数据块的读/写后才需要发出中断信号，请求CPU干预。

数据传送的单位

每次读/写一组数据块

数据流向

在通道的控制下进行

读操作（数据输入）：I/O设备->内存

写操作（数据输出）：内存->I/O设备

主要优点和主要缺点

- 优点
 - CPU、通道、I/O设备可并行工作，资源利用率很高
- 缺点
 - 实现复杂，需要专门的通道硬件支持