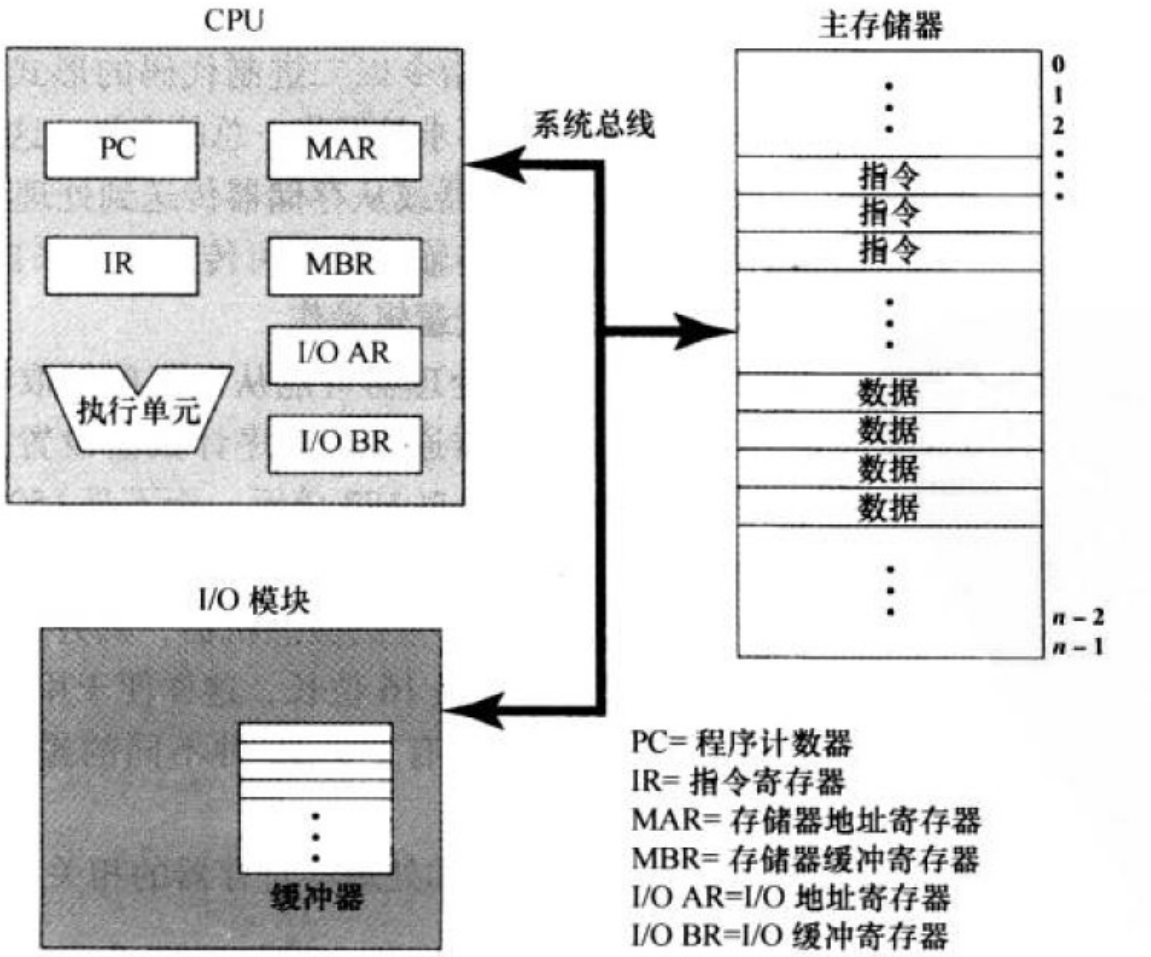
计算机组成 Lecture Notes 3

本文大致介绍计算机中的部件及其相互连接。后续会对每一个部分有更详细的介绍。

顶层视图



系统总线包括三总线：地址、数据、控制。

可能有的问题：

- 为什么 MR 和 I/O R 分开？因为两者来源不同，且有控制信号控制选择 I/O 还是 M。
- 运行时要用到的指令和数据能否存在 I/O 中？不能，因为首先冯诺依曼架构要求两者保存在存储器中，其次我们还要保证取指、取数据时地址上的值不变。如果使用 I/O 则有可能会出现问题（例如中途更换设备）。

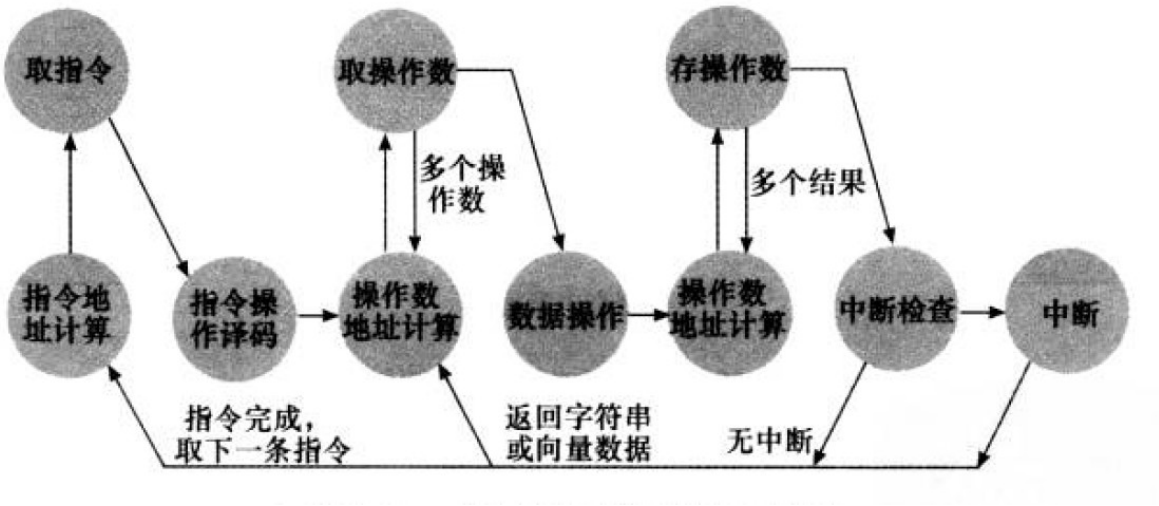
CPU

CPU 是冯诺依曼架构的核心。它有两个重要模块：控制单元（CU）和算术逻辑单元（ALU）。前者负责译码和产生控制信号。

指令执行过程

一条指令的处理用一个指令周期。其分为取指周期和执行周期。一般计算机从上电开始就一直执行指令，直到关机或者遇到某个致命的错误。

状态图如下。



一次执行中，有的状态可能不会停留，有的状态可能经过多次。中断机制会在后面阐述。

按照时间顺序描述：

- 预先：将要取出的指令的地址起始位置保存在 PC 中。**PC 是会被执行指令的唯一来源。**
- 取指令：将指令从存储器中读到 IR。
- 译码：控制单元译码，产生控制信号。
- 运行：包括取操作数、处理操作数、存结果这些动作。一般动作可以分成 4 类，前两类要用到三总线，后两类在 CPU 内部进行。一条指令是多个动作的组合。
  - CPU 和存储器交互。
  - CPU 和 I/O 交互。
  - 数据处理。如 ALU 所做的一些。
  - 控制。如改变指令顺序。
- 指令地址计算：计算出下一条指令的位置。
  - 一般情况下指令连续执行，因此只需要自增（指针自增）即可。

    > RISC 下 PC 自增的长度是一定的，因为指令定长。而 CISC 下指令中包含指令长度的数据，会反馈给 PC，让其自增正确的大小。

  - 由于有跳转指令，指令未必连续执行。因此有可能不是自增。

需要注意，有的机器有多个操作数或结果；有的机器允许单条指令处理向量或者字符串（一维数组），因此在存操作数后还可能继续运行，而不是算下一条指令。

> 这里是把下一条指令地址画在最后面，但流水线中未必。

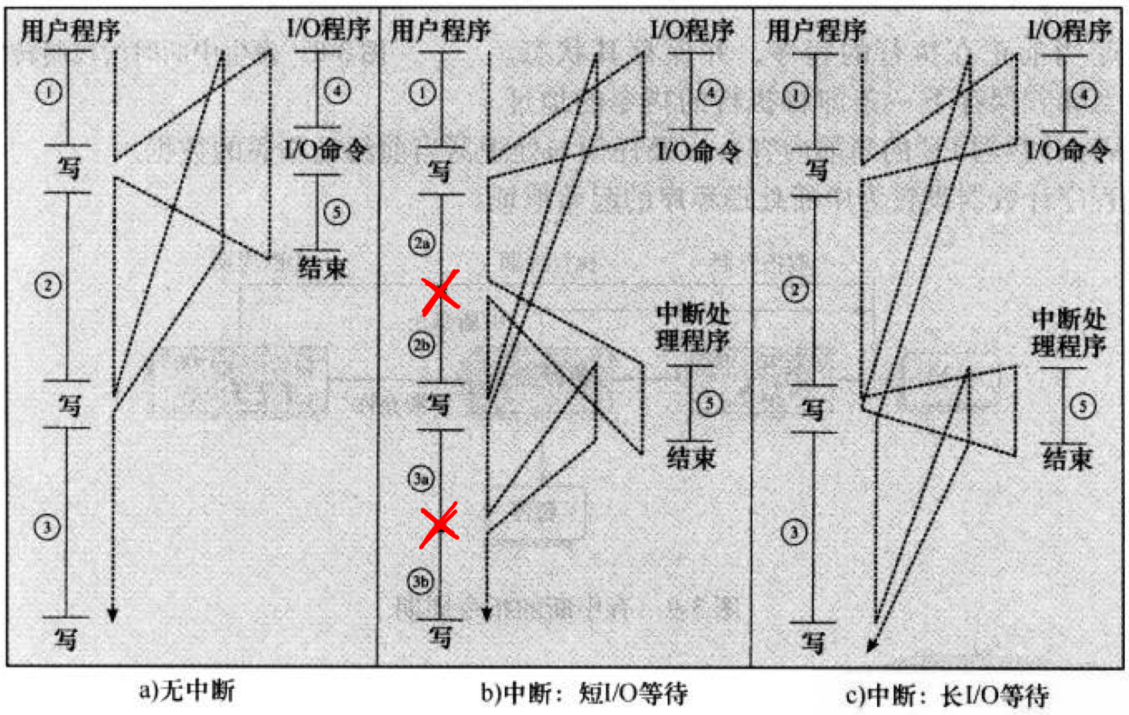
中断机制

中断是一个常用的通信机制，它由某些较为紧急的、需要得到注意的部件产生，让 CPU 暂停当前程序的执行，转而去处理相关问题。其产生原因可以分类如下：

中断类型	产生的原因
程序	由指令执行结果伴随而出现的某些条件所产生。例如，算术溢出、除以零、企图执行一些非法的机器指令或访问的地址超出了用户存储器空间
定时器	由处理器中的定时器产生，它允许操作系统以规整的时间间隔执行特定的功能
I/O	由 I/O 控制器产生，以通知操作的正常完成或各种出错情况
硬件故障	由电源故障或存储器奇偶校验出错这类的故障产生

在 CPU 与 I/O 的交互中，中断是三种常用方式的一种。后面会提到。

这里以一个经典例子说明中断：



左侧为用户程序，右侧为 I/O 程序。I/O 程序包含三个部分：

- 准备工作，标为 4。
- 实际运行，位于 4，5 之间。

在发生中断时，用户程序本身是不知情的。由 CPU 和 OS 负责用户程序的挂起和在中断处理完毕时的恢复等操作。流程：

- 挂起当前程序执行
- 保存程序运行环境，如寄存器的值，PSW（程序状态字），PC 等。
- PC 设置为中断处理程序的第一条指令
- 处理中断
- 恢复原程序

其中只有处理中断由软件完成，其余由硬件完成。

中断嵌套

中断本身可能有嵌套。处理方法：

- 禁止嵌套。可以通过设置标记位实现。
- 设置优先级。低优先级的中断可以被高优先级的中断打断。

存储器

冯诺依曼结构下，有**存储器（Memory）**这一结构用于保存指令和数据。两者都用二进制的方式保存，这也就是所谓的**存储程序概念（Stored-program Concept）**。

> 冯诺依曼结构下，计算机无法知道一个存储单元内是数据还是指令。只有 PC 指向的才会被当作是指令。

> 按照存储程序概念，指令也是被保存在存储器中的。那么可不可以对保存了指令的存储单元写呢？一般这是由 OS 控制的，OS 允许某些程序对某些存储单元有读或者写的权限。

存储器用**地址（Address）**来标识存储单元。按字节编址基本上是目前所有体系结构的共识，即一个地址对应一个字节。一个字节 8 位。

有时不严格地会将存储器和内存两个词混用。

物理表示

存储器一般可以看作是一个巨大的*一维数组*:

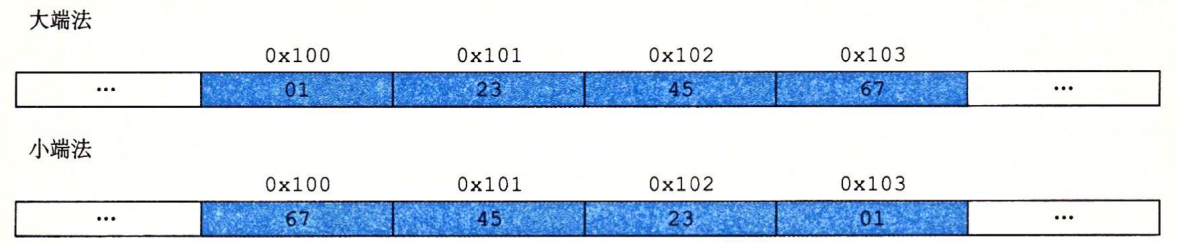
- 其有一个地址端用于接收地址，然后激活对应地址上的存储单元。
- 被激活的存储单元可以使用读写的数据端口，用于将自身数据送出或者更新自身数据。
- 还有一个控制口用于接收读写相关的控制信号。

大端小端

正如上面所说，存储器可以看作是一个巨大的一维数组。这个数组有地址的低端和高端。如果一个数据（如一个 32 位整数），若其的逻辑低端对应的是存储器的低端，逻辑高端对应的是存储器的高端，那么就是**小端模式（Little-endian）**，否则为**大端模式（Big-endian）**。

> 低（Low）对低（Low）是小端（Little-endian），可以记成 LLL。

以 `0x01234567` 为例:



可以使用一个简单的 C 程序测试大端小端:

```c
int test(){
    unsigned short t = 0x1234;
    if (*((unsigned char *) &t) == 0x12)
        return 1;
    else
        return 0;
}
```

上面这个程序对于大端返回 1，反之返回 0。

致谢

本文的主要内容参考自:

- 上海交通大学《计算机组成》（课程代号：EI332）一课的课程材料。
- 《Computer Organization and Architecture (8th edition)》