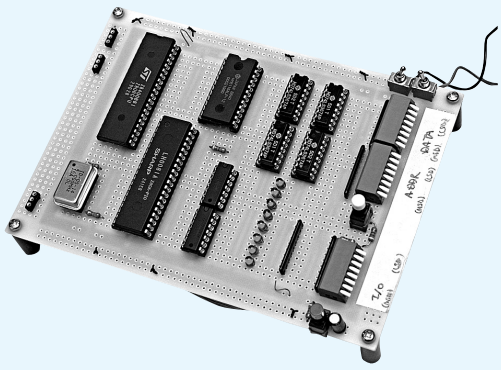
```
00001101        00000000
00001110        11010011
00001111        00000011
00010000        11011011
00010001        00000000
00010010        11010011
00010011        00000001
00010100        11000011
00010101        00010000
00010110        00000000
```

接通了微型计算机的电源后，请按下 Z80 CPU 上的 DMA 请求开关。在这个状态下，拨动用于输入内存程序和指定内存输入地址的两个指拨开关，把代码清单 2.1 所示的程序一行接一行地输入内存。先来输入第一行代码，拨动用于指定地址的指拨开关，设定出第一行代码所在的内存地址 00000000，然后拨动用于输入程序的指拨开关，设定出程序代码 00111110。再然后按下用于向内存写入程序的按键开关。接下来输入第二行代码，设定出内存地址 00000001，设定出程序代码 11001111，再次按下按键开关。反复进行这三步操作，直至输入完程序代码的最后一行。所有的指令都输入完成后，按下用于重置 CPU 的按键开关，控制 DMA 请求的快动开关就会还原成关闭状态，与此同时程序也就运行起来了。“太棒了，终于成功了！”这真是令人激动的一瞬间啊（如照片 2.2 所示）。

程序一旦运行起来，就可以用第 3 个指拨开关控制 LED 的亮与灭。只要拨动指拨开关，LED 的亮灭就会随之改变。LED 并不会只亮一下，而是一直亮着，时刻保持着指拨开关上的状态。



照片 2.2　运行中的微型计算机

☆　　☆　　☆

如今活跃在计算机行业第一线的工程师们，他们多数都在年轻的时候玩过微型计算机。诸位可以把这本书拿给他们看，他们也许会这样说：现在还有人玩这个？不过不管怎么说，对计算机理解程度的深浅还是和有没有制作过微型计算机有很大关系的。

笔者真的按照图 2.1 所示的电路图制作过微型计算机，收集零件就费了不少劲。而在单片机广泛应用的今天，CPU、I/O、内存都被集成到了一块 IC 上。可话又说回来，即便只是在纸上体验制作微型计算机的过程，也还是非常有益的。诸位在本章制作了微型计算机，想必这一体验定会加深诸位对计算机的理解，使诸位越来越喜欢计算机。

在接下来的第 3 章中，笔者会先用汇编语言为微型计算机编写程序，然后尝试“手工汇编”，即以手工作业的方式将这段程序转换成机器语言（原生代码）。敬请期待！

第3章 体验一次手工汇编

热身问答

在阅读本章内容前，让我们先回答下面的几个问题来热热身吧。

初级问题

什么是机器语言？

中级问题

通常把标识内存或 I/O 中存储单元的数字称作什么？

高级问题

CPU 中的标志寄存器（Flags Register）有什么作用？

怎么样？被这么一问，是不是发现有一些问题无法简单地解释清楚呢？下面，笔者就公布答案并解释。

答案

初级问题：由二进制数字构成的程序，CPU 可以直接对其解释、执行。

中级问题：标识内存或 I/O 中存储单元的数字叫作“地址”。

高级问题：用于在运算指令执行后，存储运算结果的某些状态。

解释

初级问题：不仅是汇编语言，用 C 语言、Java、BASIC 等编程语言编写的程序，也都需要先转换成机器语言才能被执行。机器语言有时也叫作“原生代码”（Native Code）。

中级问题：内存中有多个数据存储单元。计算机用从 0 开始的编号标识每个存储单元，这些编号就是地址（Address）。I/O 中的寄存器也可以用地址来标识。哪个寄存器对应哪个地址，取决于 CPU 和 I/O 之间的布线方式。

高级问题：Flag 的本意是“旗子”，这里引申为“标志”。一旦执行了算术运算、逻辑运算、比较运算等指令后，标志寄存器并不会存放运算结果的值，而是会把运算后的某些状态存储起来，例如运算结果是否为 0、是否产生了负数、是否有溢出（Overflow）等。

本章重点

本章的目标是通过编写程序使诸位亲身体验计算机的运行机制。为了达到这个目的，就需要使用一种叫作“汇编语言”的编程语言来编写程序，然后再把编好的程序通过手工作业转换成 CPU 可以直接执行的机器语言。

这样的转换工作叫作“手工汇编”（Hand Assemble）。也许会有人觉得听起来就好像挺麻烦的，事实上也的确如此，但是还是希望所有和计算机相关的技术人员都能亲身体验一下用汇编语言编程和手工汇编。

这次体验应该能加深诸位对计算机的理解，使诸位犹如拨云见日，找到长期困惑着自己的问题的答案，不仅能因“我能看懂程序了”而获得成就感，更能因发现“计算机原来很简单啊”而信心倍增。虽然本章的主题稍有些复杂，但是笔者会放慢讲解的步伐，还请诸位努力跟上。

3.1 从程序员的角度看硬件

为了体验手工汇编，下面我们就为在第 2 章制作的微型计算机编写一个程序吧。因为程序的作用是驱动硬件工作，所以在编写程序之前必须要先了解微型计算机的硬件信息。然而真正需要了解的硬件信息只有以下 7 种（如图 3.1 所示），所以没有必要在编程时还总是盯着详细的电路图看。

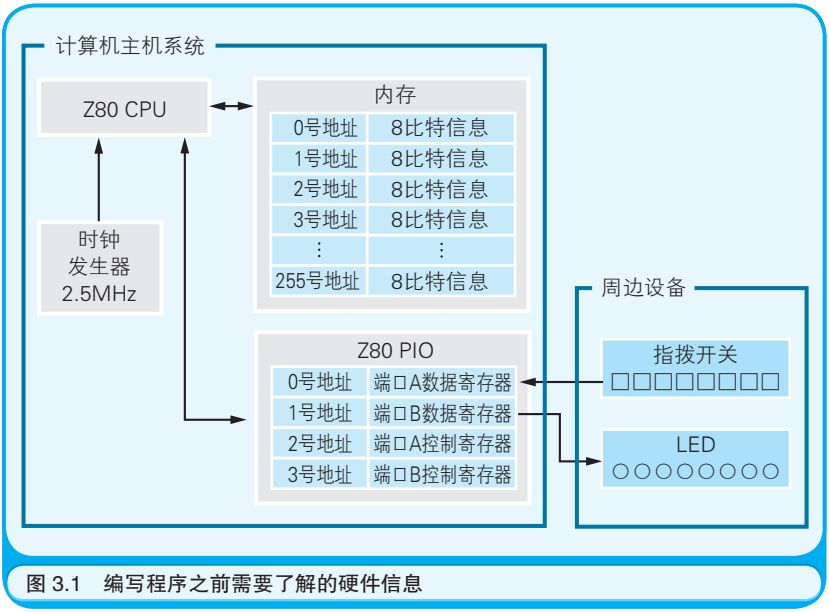


图 3.1　编写程序之前需要了解的硬件信息

【CPU（处理器）信息】

- CPU 的种类
- 时钟信号的频率

【内存信息】

- 地址空间
- 每个地址中可以存储多少比特的信息

【I/O 信息】

- I/O 的种类
- 地址空间
- 连接着何种周边设备

可以使用哪种机器语言取决于 CPU（也称作处理器）的种类。所谓

机器语言就是只用 0 和 1 两个二进制数书写的编程语言。即便是相同的机器语言，例如 01010011，只要 CPU 的种类不同，对它的解释也就不同。有的 CPU 会把它解释成是执行加法运算，有的 CPU 会把它解释成是向 I/O 输出。这就好比同样是 man 这个词，有的人会理解成“慢”，有的人会理解成“男人”。

由于微型计算机上的 CPU 是 Z80 CPU，所以就要使用适用于 Z80 CPU 的机器语言。顾名思义，机器语言就是处理器可以直接理解（与生俱来就能理解）的编程语言。机器语言有时也叫作原生代码（Native Code）。

所谓时钟信号的频率，就是由时钟发生器发送给 CPU 的电信号的频率。表示时钟信号频率的单位是 MHz（兆赫兹 = 100 万回 / 秒）。微型计算机使用的是 2.5MHz 的时钟信号。时钟信号是在 0 和 1 两个数之间反复变换的电信号，就像滴答滴答左右摆动的钟摆一样。通常把发出一次滴答的时间称作一个时钟周期。

在机器语言当中，指令执行时所需要的时钟周期数取决于指令的类型。程序员不但可以通过累加时钟周期数估算程序执行的时间，还可以仅在特定的时间执行点亮 LED（发光二极管）等操作。

每个地址都标示着一个内存中的数据存储单元，而这些地址所构成的范围就是内存的地址空间。在我们的微型计算机中，地址空间为 0～255，每一个地址中可以存储 8 比特（1 字节）的指令或数据。

连接着的 I/O 的种类，就是指连接着微型计算机和周边设备的 I/O 的种类。在微型计算机中，只安装了一个 I/O，即上面带有 4 个 8 比特寄存器的 Z80 PIO。只要用 CPU 控制 I/O 的寄存器，就可以设定 I/O 的功能，与周边设备进行数据的输入输出。

所谓 I/O 的地址空间，是指用于指定 I/O 寄存器的地址范围。在 Z80 PIO 上，地址空间为 0～3，每一个地址对应一个寄存器。

在内存中，每个地址的功能都一样，既可用于存储指令又可用于存储数据。而 I/O 则不同，地址编号不同（即寄存器的类型不同），功能也就不同。在微型计算机中，是这样分配 Z80 PIO 上的寄存器的：端口 A 数据寄存器对应 0 号地址，端口 B 数据寄存器对应 1 号地址，端口 A 控制寄存器对应 2 号地址，端口 B 控制寄存器对应 3 号地址。端口 A 数据寄存器和端口 B 数据寄存器存储的是与周边设备进行输入输出时所需的数据。其中，端口 A 连接用于输入数据的指拨开关，端口 B 连接用于输出数据的 LED。而端口 A 控制寄存器和端口 B 控制寄存器则存储的是用于设定 Z80 PIO 功能的参数。

3.2　机器语言和汇编语言

请看代码清单 3.1 中列出的机器语言程序，这段程序在第 2 章中已经介绍过了，功能是把由指拨开关输入的数据输入 CPU，然后 CPU 再把这些数据原封不动地输出到 LED。也就是说，可以通过拨动指拨开关控制 LED 的亮或灭。

代码清单 3.1　点亮 LED 的机器语言程序

```
地址            机器语言
00000000        00111110
00000001        11001111
00000010        11010011
00000011        00000010
00000100        00111110
00000101        11111111
00000110        11010011
00000111        00000010
00001000        00111110
00001001        11001111
00001010        11010011
```

```
00001011        00000011
00001100        00111110
00001101        00000000
00001110        11010011
00001111        00000011
00010000        11011011
00010001        00000000
00010010        11010011
00010011        00000001
00010100        11000011
00010101        00010000
00010110        00000000
```

这段由 8 比特二进制数构成的机器语言程序总共 23 个字节。若把这些字节一个接一个地依次写入内存中，所占据的内存空间就是 00000000～00010110。一旦重置了 CPU，CPU 就会从 0 号地址开始顺序执行这段程序。

在机器语言程序中，虽然到处都是 0 和 1 的组合，但是每个组合都是有特定含义的指令或数据。可是对人来说，如果只看 0 和 1 的话，恐怕很难判断各个组合都表示什么。

于是就有人发明出了一种编程方法，根据表示指令功能的英语单词起一个相似的昵称，并将这个昵称赋予给 0 和 1 的组合。这种类似英语单词的昵称叫作“助记符”，使用助记符的编程语言叫作“汇编语言”。无论是使用机器语言还是汇编语言，所实现的功能都是一样的，区别只在于程序是用数字表示，还是用助记符表示。也就是说，如果理解了汇编语言，也就理解了机器语言，更进一步也就理解了计算机的原始的工作方式。

代码清单 3.1 中的机器语言可以转换成如代码清单 3.2 所示的汇编语言。汇编语言的语法十分简单，以至于语法只有一个，即把“标签”“操作码（指令）”和“操作数（指令的对象）”并排写在一行上，仅

此而已。

代码清单 3.2　用汇编语言的代码表示代码清单 3.1 中的机器语言

```
标签            操作码          操作数
                LD              A, 207
                OUT             (2), A
                LD              A, 255
                OUT             (2), A
                LD              A, 207
                OUT             (3), A
                LD              A, 0
                OUT             (3), A
LOOP:           IN              A, (0)
                OUT             (1), A
                JP              LOOP
```

标签的作用是为该行代码对应的内存地址起一个名字。编程时如果总要考虑“这一行的内存地址是什么来着?”就会很不方便，所以在汇编语言中用标签来代替地址。用汇编语言编程时可以在任何需要标签的地方“贴上”名称任意的标签。在代码清单 3.2 所示的程序中，使用了名称为“LOOP:”的标签。

操作码就是表示“做什么”的指令。因为用助记符表示的指令是英语单词的缩写，比如 LD 是 Load（加载）的缩写，所以多多少少能猜出其中的含义。汇编语言中提供了多少种助记符，CPU 就有多少种功能。Z80 CPU 的指令全部加起来有 70 条左右。这里先把主要的指令列在表 3.1 中，请诸位粗略地浏览一下。在浏览的过程中请注意这些指令的分类，按功能这些指令可以分成运算、与内存的输入输出和与 I/O 的输入输出三类。这是因为计算机能做的事也只有输入、运算、输出这三种了。

操作数表示的是指令执行的对象。CPU 的寄存器、内存地址、I/O 地址或者直接给出的数字都可以作为操作数。如果某条指令需要多个