51 单片机汇编语言教程：8 课单片机寻址方式与指令系统

（基于 HJ-1G、HJ-3G 实验板）

通过前面的学习，我们已经了解了单片机内部的结构，并且也已经知道，要控制单片机，让它为我们干活，要用指令，我们已学了几条指令，但很零散，从现在开始，我们将要系统地学习 8051 单片机的指令部份。

一、概述

1、指令的格式

我们已知，要让计算机做事，就得给计算机发送各种指令，并且我们已知，计算机很“笨”，只能懂得数字，如前面我们写进机器的 75H，90H，00H 等等，所以指令的第一种格式就是机器码格式，也说是数字的形式。但这种形式实在是为难我们人了，太难记了，于是有另一种格式，助记符格式，如 MOV P1，#0FFH，这样就好记了。 这两种格式之间的关系呢，我们不难理解，本质上它们完全等价，只是形式不一样而已。

2、汇编

我们写指令使用汇编格式，而计算机和单片机只懂机器码格式，所以要将我们写的汇编格式的指令转换为机器码格式，这种转换有两种办法：手工汇编和机器汇编。手工汇编实际上就是查表，因为这两种格式纯粹是格式不一样，所以是一一对应的，查一张表格就行了。不过手工查表总是嫌麻烦，所以就有了计算机软件，用计算机软件来替代手工查表，这就是机器汇编。

二、单片机的寻址

让我们先来复习一下我们学过的一些指令：MOV P1，#0FFH，MOV R7，#0FFH 这些指令都是将一些数据送到对应的位置中去，为什么要送数据呢？第一个因为送入的数能让灯全灭掉，第二个是为了要实现延时，从这里我们能看出来，在用单片机的编程语言编程时，经常要用到数据的传递，事实上数据传递是单片机编程时的一项重要工作，一共有 28 条指令（单片机共 111 条指令）。下面我们就从数据传递类指令开始吧。

分析一下 MOV P1，#0FFH 这条指令，我们不难得出结论，第一个词 MOV 是命令动词，也就是决定做什么事情的，MOV 是 MOVE 少写了一个 E，所以就是“传递”，这就是指令，规定做什么事情，后面还有一些参数，分析一下，数据传递必须要有一个“源”也就是你要送什么数，必须要有一个“目的”，也就是你这个数要送到什么地方去，显然在上面那条单片机指令中，要送的数（源）就是 0FFH，而要送达的地方（目的地）就是 P1 这个寄存器。在数据传递类指令中，均将目的地写在指令的后面，而将源写在最后。

这条指令中，送给 P1 是这个数本身，换言之，做完这条指令后，我们能明确地知道，P1 中的值是 0FFH，但是并不是任何时候都能直接给出数本身的。例如，在我们前面给出的单片机延时程序例是这样写的：

```
MAIN:  SETB P1.0          ;（1）
       LCALL DELAY ;（2）
         CLR P1.0          ;（3）
       LCALL DELAY      ;（4）
         AJMP MAIN        ;（5）
; 以下子程序
DELAY:  MOV R7, #250      ;（6）
D1:  MOV R6, #250      ;（7）
D2:  DJNZ R6, D2         ;（8）
     DJNZ R7, D1       ;（9）
```

```
      RET                ;（1 0）
      END                ;（1 1）
```

表 1

```
-------------------------------------------------
MAIN:  SETB P1.0          ;（1）
       MOV 30H，#255
        LCALL DELAY ;
        CLR P1.0          ;（3）
        MOV 30H,#200
        LCALL DELAY       ;（4）
        AJMP MAIN         ;（5）
; 以下子程序
DELAY:  MOV R7，30H       ;（6）
D1:  MOV R6，#250         ;（7）
D2:  DJNZ R6，D2          ;（8）
      DJNZ R7，D1         ;（9）
      RET                 ;（1 0）
      END                 ;（1 1）
```

表 2

这样一来，我每次调用延时程序延时的时间都是相同的（大致都是 0.13S），如果我提出这样的要求：灯亮后延时时间为 0.13S 灯灭，灯灭后延时 0.1 秒灯亮，如此循环，这样的程序还能满足要求吗？不能，怎么办？我们能把延时程序改成这样(见表 2)：调用则见表 2 中的主程，也就是先把一个数送入 30H，在子程序中 R7 中的值并不固定，而是根据 30H 单元中传过来的数确定。这样就能满足要求。

从这里我们能得出结论，在数据传递中要找到被传递的数，很多时候，这个数并不能直接给出，需要变化，这就引出了一个概念：如何寻找操作数，我们把寻找操作数所在单元的地址称之为寻址。在这里我们直接使用数所在单元的地址找到了操作数，所以称这种办法为直接寻址。除了这种办法之外，还有一种，如果我们把数放在工作寄存器中，从工作寄存器中寻找数据，则称之为寄存器寻址。例：MOV A，R0 就是将 R0 工作寄存器中的数据送到累加器 A 中去。提一个问题：我们知道，工作寄存器就是内存单元的一部份，如果我们选择工作寄存器组 0，则 R0 就是 RAM 的 00H 单元，那么这样一来，MOV A，00H，和 MOV A，R0 不就没什么区别了吗？为什么要加以区别呢？的确，这两条指令执行的结果是完全相同的，都是将 00H 单元中的内容送到 A 中去，但是执行的过程不一样，执行第一条指令需要 2 个周期，而第二条则只需要 1 个周期，第一条指令变成最终的目标码要两个字节（E5H 00H），而第二条则只要一个字节（E8h）就能了。

这么斤斤计较！不就差了一个周期吗，如果是 12M 的晶体震荡器的话，也就 1 个微秒时间了，一个字节又能有多少？

不对，如果这条指令只执行一次，也许无所谓，但一条指令如果执行上 1000 次，就是 1 毫秒，如果要执行 1000000 万次，就是 1S 的误差，这就很可观了，单片机做的是实时控制的事，所以必须如此“斤斤计较”。字节数同样如此。

再来提一个问题，现在我们已知，寻找操作数能通过直接给的方式（立即寻址）和直接给出数所在单元地址的方式（直接寻址），这就够了吗？

看这个问题，要求从 30H 单元开始，取 20 个数，分别送入 A 累加器。

就我们目前掌握的办法而言，要从 30H 单元取数，就用 MOV A，30H，那么下一个数呢？是 31H 单元的，怎么取呢？还是只能用 MOV A，31H，那么 20 个数，不是得 20 条指令才能写完吗？这里只有 20 个数，如果要送 200 个或 2000 个数，那岂不要写上 200 条或 2000 条命令这未免太笨了吧。为什么会出现这样的状况？是因为我们只会把地址写在指令中，所以就没办法了，如果我们不是把地址直接写在指令中，而是把地址放在另外一个寄存器单元中，根据这个寄存器单元中的数值决定该到哪个单元中取数据，比如，当前这个寄存器中的值是 30H，那么就到 30H 单元中去取，如果是 31H 就到 31H 单元中去取，就能解决这个问题了。怎么个解决法呢？既然是看的寄存器中的值，那么我们就能通过一定的办法让这里面的值发生变化，比如取完一个数后，将这个寄存器单元中的值加 1，还是执行同一条指令，可是取数的对象却不一样了，不是吗。通过例程来说明吧。

```
MOV R7，#20
MOV R0，#30H
LOOP：MOV A，@R0
INC R0
DJNZ R7,LOOP
```

这个例程中大部份指令我们是能看懂的，第一句，是将立即数 20 送到 R7 中，执行完后 R7 中的值应当是 20。第二句是将立即数 30H 送入 R0 工作寄存器中，所以执行完后，R0 单元中的值是 30H，第三句，这是看一下 R0 单元中是什么值，把这个值作为地址，取这个地址单元的内容送入 A 中，此时，执行这条指令的结果就相当于 MOV A，30H。第四句，没学过，就是把 R0 中的值加 1，因此执行完后，R0 中的值就是 31H，第五句，学过，将 R7 中的值减 1，看是否等于 0，不等于 0，则转到标号 LOOP 处继续执行，因此，执行完这句后，将转去执行 MOV A，@R0 这句话，此时相当于执行了 MOV A，31H（因为此时的 R0 中的值已是 31H 了），如此，直到 R7 中的值逐次相减等于 0，也就是循环 20 次为止，就实现了我们的要求：从 30H 单元开始将 20 个数据送入 A 中。

这也是一种寻找数据的办法，由于数据是间接地被找到的，所以就称之为间址寻址。注意，在间址寻址中，只能用 R0 或 R1 存放等寻找的数据。