51单片机汇编语言教程：第13课-单片机逻辑与或异或指令详解

（基于 HJ-1G、HJ-3G 实验板）

ANL A,Rn ;A 与 Rn 中的值按位’与’，结果送入 A 中

ANL A,direct ;A 与 direct 中的值按位’与’，结果送入 A 中

ANL A,@Ri ;A 与间址寻址单元@Ri 中的值按位’与’，结果送入 A 中

ANL A,#data ;A 与立即数 data 按位’与’，结果送入 A 中

ANL direct,A ;direct 中值与 A 中的值按位’与’，结果送入 direct 中

ANL direct,#data ;direct 中的值与立即数 data 按位’与’，结果送入 direct 中。

这几条指令的关键是知道什么是逻辑与。这里的逻辑与是指按位与

例：71H 和56H 相与则将两数写成二进制形式：

（71H） 01110001

（56H） 00100110

结果 00100000 即20H，从上面的式子能看出，两个参与运算的值只要其中有一个位上是0，则这位的结果就是0，两个同是1，结果才是1。

理解了逻辑与的运算规则，结果自然就出来了。看每条指令后面的注释

下面再举一些例程来看。

MOV A，#45H ;(A)=45H

MOV R1，#25H ;(R1)=25H

MOV 25H，#79H ;(25H)=79H

ANL A，@R1 ;45H 与79H 按位与，结果送入 A 中为 41H （A）=41H

ANL 25H,#15H ;25H 中的值（79H）与15H 相与结果为（25H）=11H）

ANL 25H,A ;25H 中的值（11H）与 A 中的值（41H)相与，结果为(25H)=11H

在知道了逻辑与指令的功能后，逻辑或和逻辑异或的功能就很简单了。逻辑或是按位“或”，即有“1”为1，全“0”为0。例：

10011000

或 01100001

结果 11111001

而异或则是按位“异或”，相同为“0”，相异为“1”。例：

10011000

异或 01100001

结果 11111001

而所有的或指令，就是将与指仿中的 ANL 换成 ORL，而异或指令则是将 ANL 换成 XRL。即

或指令：

ORL A,Rn ;A 和 Rn 中的值按位’或’，结果送入 A 中

ORL A,direct ;A 和与间址寻址单元@Ri 中的值按位’或’，结果送入 A 中

ORL A,#data ;A 和立 direct 中的值按位’或’，结果送入 A 中

ORL A,@Ri ;A 和即数 data 按位’或’，结果送入 A 中

ORL direct,A ;direct 中值和 A 中的值按位’或’，结果送入 direct 中

ORL direct,#data ;direct 中的值和立即数 data 按位’或’，结果送入 direct 中。

异或指令：

XRL A,Rn ;A 和 Rn 中的值按位’异或’，结果送入 A 中

XRL A,direct ;A 和 direct 中的值按位’异或’，结果送入 A 中

XRL A,@Ri ;A 和间址寻址单元@Ri 中的值按位’异或’，结果送入 A 中

XRL A,#data ;A 和立即数 data 按位’异或’，结果送入 A 中

XRL direct,A ;direct 中值和 A 中的值按位’异或’，结果送入 direct 中

XRL direct,#data ;direct 中的值和立即数 data 按位’异或’，结果送入 direct 中。

练习：

MOV A，#24H

MOV R0，#37H

ORL A，R0

XRL A，#29H

MOV 35H，#10H

ORL 35H，#29H

MOV R0，#35H

ANL A，@R0

四、控制转移类指令

无条件转移类指令

短转移类指令

AJMP addr11

长转移类指令

LJMP addr16

相对转移指令

SJMP rel

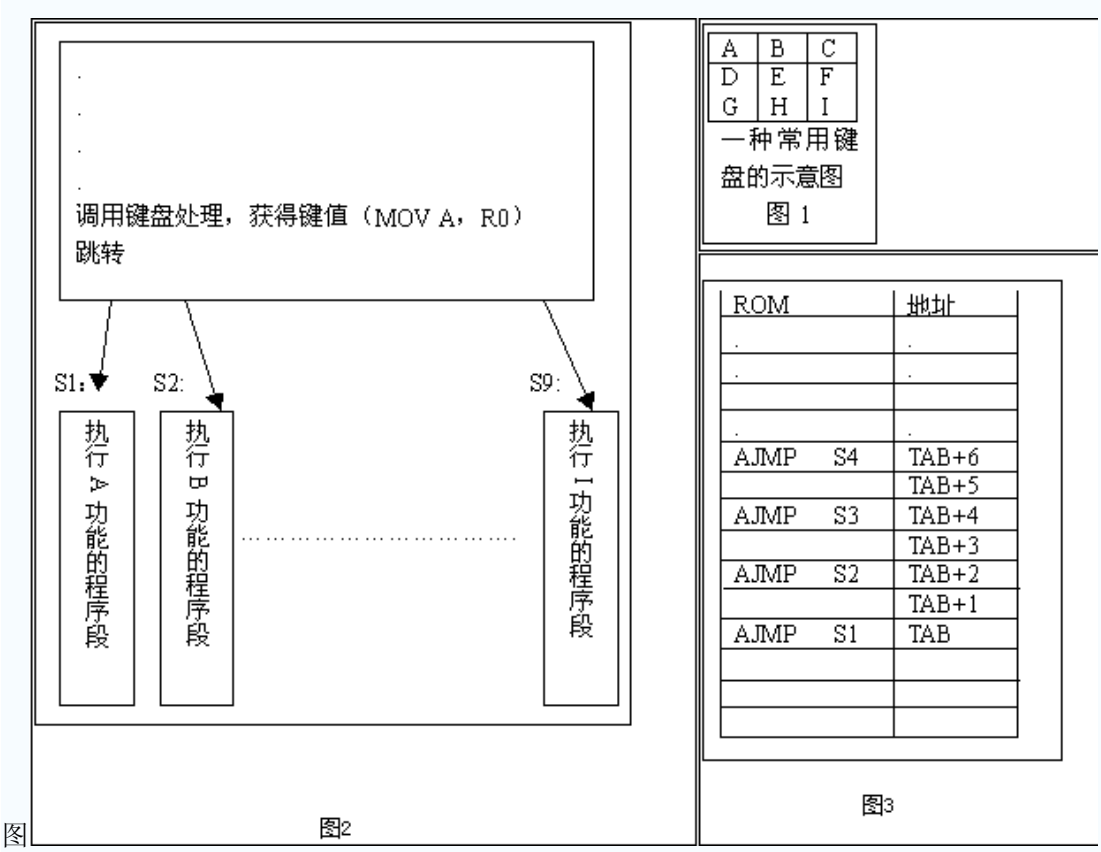
上面的三条指令，如果要仔细分析的话，区别较大，但开始学习时，可不理会这么多，统统理解成：JMP 标号，也就是跳转到一个标号处。事实上，LJMP 标号，在前面的例程中我们已接触过，并且也知道如何来使用了。而 AJMP 和 SJMP 也是一样。那么他们的区别何在呢？在于跳转的范围不一样。好比跳远，LJMP 一下就能跳64K 这么远（当然近了更没关系了）。而 AJMP 最多只能跳2K 距离，而 SJMP 则最多只能跳256这么远。原则上，所有用 SJMP 或 AJMP 的地方都能用 LJMP 来替代。因此在开始学习时，需要跳转时能全用 LJMP，除了一个场合。什么场合呢？先了解一下 AJMP，AJMP 是一条双字节指令，也就说这条指令本身占用存储器（ROM）的两个单元。而 LJMP 则是三字节指令，即这条指令占用存储器（ROM）的三个单元。下面是第四条跳转指令。

间接转移指令

JMP @A+DPTR

这条指令的用途也是跳转，转到什么地方去呢？这可不能由标号简单地决定了。让我们从一个实际的例程入手吧。

```
MOV DPTR，#TAB ;将 TAB 所代表的地址送入 DPTR
MOV A，R0 ;从 R0中取数（详见下面说明）
MOV B，#2
MUL A，B ;A 中的值乘2（详见下面的说明）
JMP A，@A+DPTR ;跳转
TAB: AJMP S1 ;跳转表格
AJMP S2
AJMP S3
```



应用背景介绍：在单片机开发中，经常要用到键盘，见上面的9个按钮的键盘。我们的要求是：当按下功能键 A……..G 时去完成不一样的功能。这用程序设计的语言来表达的话，就是：按下不一样的键去执行不一样的程序段，以完成不一样的功能。怎么样来实现呢？

看图2，前面的程序读入的是按钮的值，如按下’A’键后获得的键值是0，按下’B’键后获得的值是’1’等等，然后根据不一样的值进行跳转，如键值为0就转到 S1执行，为1就转到 S2执行。。。。如何来实现这一功能呢？

先从程序的下面看起，是若干个 AJMP 语句，这若干个 AJMP 语句最后在存储器中是这样存放的（见图3），也就是每个 AJMP 语句都占用了两个存储器的空间，并且是连续存放的。而 AJMP S1存放的地址是 TAB，到底 TAB 等于多少，我们不需要知道，把它留给汇编程序来算好了。

下面我们来看这段程序的执行过程：第一句 MOV DPTR，#TAB 执行完了之后，DPTR 中的值就是 TAB，第二句是 MOV A，R0，我们假设 R0是由按钮处理程序获得的键值，比如按下 A 键，R0中的值是0，按下 B 键，R0中的值是1，以此类推，现在我们假设按下的是 B 键，则执行完第二条指令后，A 中的值就是1。并且按我们的分析，按下 B 后应当执行 S2这段程序，让我们来看一看是否是这样呢？第三条、第四条指令是将 A 中的值乘2，即执行完第4条指令后 A 中的值是2。下面就执行 JMP @A+DPTR 了，现在 DPTR 中的值是 TAB，而 A+DPTR 后就是

TAB+2，因此，执行此句程序后，将会跳到 TAB+2这个地址继续执行。看一看在 TAB+2这个地址里面放的是什么？就是 AJMP S2这条指令。因此，马上又执行 AJMP S2指令，程序将跳到 S2处往下执行，这与我们的要求相符合。

请大家自行分析按下键“A”、“C”、“D”……之后的情况。

这样我们用 JMP @A+DPTR 就实现了按下一键跳到对应的程序段去执行的这样一个要求。再问大家一个问题，为什么取得键值后要乘2？如果例程下面的所有指令换成 LJMP，即：LJMP S1,LJMP S2……这段程序还能正确地执行吗？如果不能，应该怎么改？