

# 第8章 给单片机下命令——指令畅谈

欢迎访问 电路飞翔网

<http://www.circuitfly.com> 获取更多信息

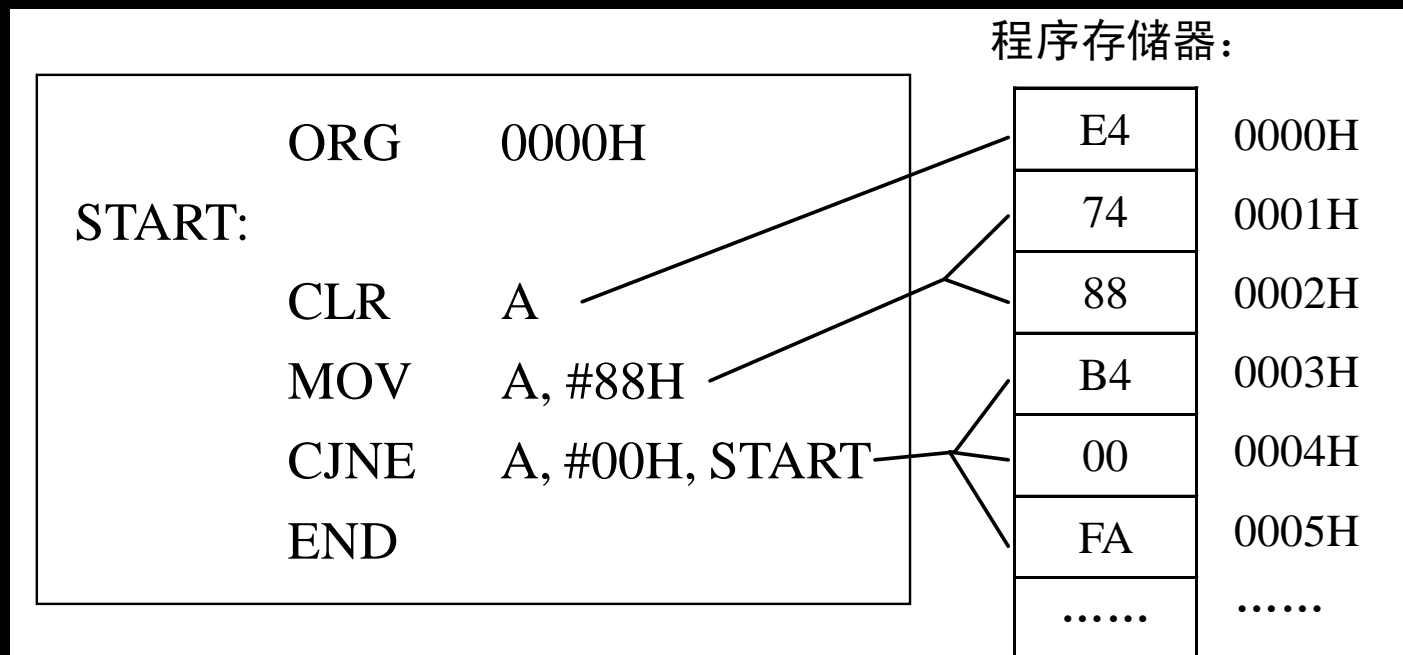
- 8.1 指令概述
- 8.2 算术指令
- 8.3 逻辑指令
- 8.4 片内数据装载指令
- 8.5 片外数据装载指令
- 8.6 查表指令
- 8.7 布尔指令
- 8.8 调用子程序指令
- 8.9 跳转与循环指令
- 8.10 单片机对于带符号数的处理和溢出问题
- 8.11 实例点拨——指令应用（程序）实例

# 单片机的命令

- 51单片机的指令总有111条，配合与不同的操作数共有255个指令的写法，即255个指令的执行代码。这个数量比起其他的处理器来说并不为多。但在实际应用中，常用的指令一般不过50个。
- 一条指令只要经过几次反复使用和理解就能掌握。

# 8.1 指令概述

## • 8.1.1 指令的长度



指令的长度

多数指令的长度为单字节或双字节的。指令的长度与指令周期（指令执行所花费的时间）没有必然的联系。有的单字节指令周期比双字节或三字节指令要长，如指令“**MUL AB**”长度为1个字节，而指令周期却为4个机器周期。

# 8.1 指令概述

## • 8.1.2 影响程序状态字PSW的指令

程序状态字PSW用来存放CPU运算和执行状态的寄存器，长度为1个字节，其8个位分别如下：

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	OV	保留	P

其中有4个位在执行某些指令时可能会被改变，以指示CPU的运算和执行状态，如下：

CY——进位标志。

AC——辅助进位标志。

OV——溢出标志。

P——奇偶标志位。

# 8.1 指令概述

## • 8.1.2 影响程序状态字PSW的指令

指 令	标 志 位			指 令	标 志 位		
	CY	OV	AC		CY	OV	AC
ADD	1/0	1/0	1/0	CLR C	0		
ADDC	1/0	1/0	1/0	CPL C	1/0		
SUBB	1/0	1/0	1/0	ANL C, bit	1/0		
MUL	0	1/0		ANL C, /bit	1/0		
DIV	0	1/0		ORL C, bit	1/0		
DA	1/0			ORL C, /bit	1/0		
RRC	1/0			MOV C, bit	1/0		
RLC	1/0			CJNE	1/0		
SETB C	1						

表中“1”表明执行该指令时会将标志位置1；“0”表明会清0标志位；“1/0”表明可能置1，也可能清0；没有标注的说明不会影响标志位。

## 8.2 算术指令

### • 8.2.1 加法指令——ADD A, <src-byte>

A代表累加器ACC，<src-byte>代表“源操作数-以字节形式”

指 令	说 明	字 节	机器周期
ADD A,Rn	将Rn与ACC的值相加，结果存回ACC	1	1
ADD A,direct	将直接地址direct的内容与ACC相加，结果存回ACC	2	1
ADD A,@Ri	将间接地址@Ri的内容与ACC相加，结果存回ACC	1	1
ADD A,#data	将立即数#data与ACC的值相加，结果存回ACC	2	1

Ri 表示R0或R1；Rn 表示R0~R7的其中之一

关于ADD指令有以下几点需要说明。

① 相加的操作总是在累加器ACC中发生，源操作数可以是一个工作寄存器的值、直接地址的内容、间接地址的内容或立即数。

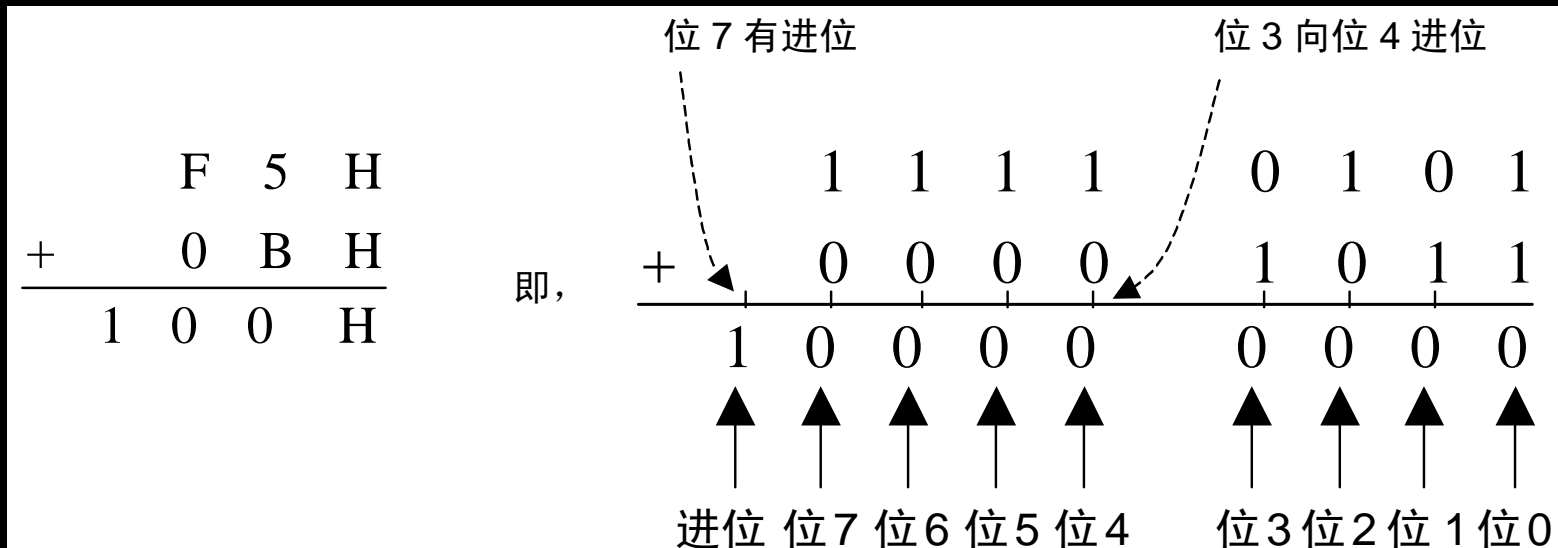
② ADD指令可能影响PSW中的标志位CY、OV、AC和P。如果相加过程中，位3有进位则辅助进位标志位AC=1；位7有进位则进位标志位CY=1。

## 8.2 算术指令

### • 8.2.1 加法指令——ADD A, <src-byte>

例: MOV A, #0F5H ; ACC=0F5H

ADD A, #0BH ; ACC=0F5+0BH=100H



运算结果: ACC=00, 标志位情况如下:

CY=1——由于位7有进位, AC=1——由于位3向位4进位,

P=1——由于ACC=0000 0000, 1的个数是0 (偶数)。

## 8.2 算术指令

### • 8.2.1 加法指令——ADD A, <src-byte>

例: MOV A, #0F5H ; ACC=0F5H  
ADD A, #0BH ; ACC=0F5+0BH=100H

溢出标志位OV的变化是：如果位6有进位而位7没有进位，或者位7有进位而位6没有，则溢出标志OV=1，否则OV=0。注意，OV的状态只有在带符号数加法运算时才有意义。当两个带符号数相加时，OV=1表示加法运算超出了累加器ACC所能表示的带符号数的有效范围（-128~+127），即产生了溢出，因此运算结果是错误的；否则OV=0说明无溢出产生，运算结果是正确的。



## 8.2 算术指令

### • 8.2.2 带进位的加法指令——ADDC A, <src-byte>

带进位的加法指令ADDC相当于在ADD的运算基础上再加上进位CY。

指 令	说 明	字 节	机 器 周 期
ADDC A,Rn	将Rn与ACC的值及进位CY相加，结果存回ACC	1	1
ADDC A,direct	将直接地址direct的内容与ACC及进位CY相加，结果存回ACC	2	1
ADDC A,@Ri	将间接地址@Ri的内容与ACC及进位CY相加，结果存回ACC	1	1
ADDC A,#data	将立即数#data与ACC的值及进位CY相加，结果存回ACC	2	1

## 8.2 算术指令

### • 8.2.2 带进位的加法指令——ADDC A, <src-byte>

ADDC和ADD指令一起使用时一般用于处理2个字节（16位）的加法运算。由于累加器ACC的长度为1个字节，所以在处理2个字节的加法时先用ADD指令进行低位字节的运算，把结果保存在某一地址空间或寄存器中。如果低位字节相加有进位将会影响进位标志CY，于是再使用ADDC指令进行高位字节的加法运算，把结果保存在另一个地址空间或寄存器中，这两个加法运算的结果合在一起就是2个字节数据的和。

## 8.2 算术指令

### • 8.2.2 带进位的加法指令——ADDC A, <src-byte>

```
.....  
CLR      C                ; 清进位 CY  
MOV      A, #0E7H         ; ACC=0E7H, 载入低位字节  
ADD      A, #08DH         ; 低位字节加法, ACC=E7+8D=74H, 且 CY=1  
MOV      R0, A            ; 在 R0 中保存和的低位字节  
MOV      A, #3CH          ; ACC=3CH, 载入高位字节  
ADDC     A, #3BH          ; 带进位的加法运算, 3C + 3B + 1 = 78H  
MOV      R1, A            ; 和的高位字节保存在 R1 中  
.....
```

运算结果：R1=78H，R0=74。

如果在进行高位字节加法时也使用ADD指令将会漏掉低位字节相加的进位CY，而导致最终运算的错误。

## 8.2 算术指令

### • 8.2.3 带借位的减法指令——SUBB A,<src-byte>

减法指令SUBB是“subtract with borrow”的缩写，当进行减法运算时，PSW中的CY位就变成了借位标志位。

指 令	说 明	字 节	机器周期
SUBB A,Rn	ACC减去Rn的值及借位CY，结果存回ACC	1	1
SUBB A,direct	ACC减去直接地址direct的内容及借位CY，结果存回ACC	2	1
SUBB A,@Ri	ACC减去间接地址@Ri的内容及借位CY，结果存回ACC	1	1
SUBB A,#data	ACC减去立即数#data及借位CY，结果存回ACC	2	1

## 8.2 算术指令

### • 8.2.3 带借位的减法指令——SUBB A,<src-byte>

关于SUBB指令对标志位的影响如下：

- ① 当位7有借位时，标志CY=1；否则CY=0。也就是说，如果无符号数做减法时，减数比被减数大，CY=1。
- ② 当位3有借位时，标志AC=1；否则AC=0。
- ③ 溢出标志位OV=1表示溢出；OV=0表示未溢出。OV的值可由差的位7的借位与位6的借位做XOR（异或）的逻辑判断得到。
- ④ 由于SUBB指令连CY一起减，若不想减CY，可先将CY清0。

## 8.2 算术指令

- 8.2.4 自增/自减指令——INC <byte> / DEC <byte>  
进位标志CY不受影响，可对DPL和DPH进行INC/DEC操

指 令	说 明	字 节	机器周期
INC A	ACC的值自增1	1	1
INC Rn	工作寄存器Rn的值自增1	1	1
INC direct	直接地址direct的内容自增1	2	1
INC @Ri	间接地址@Ri的内容自增1	1	1
INC DPTR	DPTR值自增1	1	1
DEC A	ACC的值自减1	1	1
DEC Rn	工作寄存器Rn的值自减1	2	1
DEC direct	直接地址direct的内容自减1	1	1
DEC @Ri	间接地址@Ri的内容自减1	1	1

## 8.2 算术指令

### • 8.2.5 乘法指令——MUL AB

指 令	说 明	字节	机器周期
MUL AB	ACC与B寄存器的值相乘 ( $A \times B$ )，积的低位字节存回ACC，高位字节存回B寄存器	1	4

说明：

① 做无符号的1个字节运算时，累加器ACC与B寄存器的值相乘所得的积（2个字节），其低位字节存回ACC中，高位字节存入B寄存器。

② 如果积大于00FFH，则OV=1，否则OV=0。

③ 执行时，进位标志CY会被清除为0。

例： MOV            A, #66H        ; ACC=66H  
       MOV            B, #77H        ; B=77H  
       MUL            AB            ; 积=66×77=2F6AH

运算结果：

A=6AH,  
 B=2FH,  
 CY=0, 由于  
 2F6AH>FFH  
 , 所以  
 OV=1。

## 8.2 算术指令

### • 8.2.6 除法指令——DIV AB

指 令	说 明	字节	周期
DIV AB	ACC除以B寄存器的值 ( $A \div B$ )， 商存回ACC，余数存回B寄存器	1	4

说明：

① DIV指令做无符号的除法运算。如果B寄存器=0，执行时OV被置1，表示运算是错误的，因为除数不应该为0。

② DIV指令正确执行后，商存回ACC，余数存回B寄存器，进位标志CY及溢出标志OV都等于0。例：

MOV A, #66H ; ACC=66H

MOV B, #04H ; B=04H

DIV AB ;  $66 \div 04 = 19H \dots 02H$ ，即商=19H，  
; 余数=02H

运算结果：A=19H，B=02H，CY=0，OV=0。



## 8.2 算术指令

### • 8.2.7 十进制调整指令——DA A

指令DA只适用于加法指令ADD或ADDC后且只对累加器ACC产生作用，而不能使用在如INC等指令之后。

指 令	说 明	字 节	机器周期
DA A	累加器ACC作十进制调整	1	1

DA指令进行十进制调整的方法为：

① 在ADD或ADDC运算后，若ACC低位>9或AC=1，则ACC+06H。

② 在ADD或ADDC运算后，若ACC高位>9或CY=1，则ACC+60H。

例：MOV A, #47H ; ACC=47H  
 MOV B, #25H ; B=25H  
 ADD A, B ; ACC=6CH, ACC的低位C>9, ACC+06H  
 DA A ; 十进制调整后ACC=6C+06=72H

## 8.3 逻辑指令

### • 8.3.1 AND操作——ANL <dest-byte>,<src-byte>

指 令	说 明	字 节	机器周期
ANL A,Rn	将Rn与ACC的值做AND运算，结果存回ACC	1	1
ANL A,direct	将直接地址direct的内容与ACC做AND运算，结果存回ACC	2	1
ANL A,@Ri	将间接地址@Ri的内容与ACC做AND运算，结果存回ACC	1	1
ANL A,#data	立即数#data与ACC做AND运算，结果存回ACC	2	1
ANL direct,A	ACC与直接地址direct的内容做AND运算，结果存回该直接地址中	2	1
ANL direct,#data	立即数#data与直接地址direct的内容做AND运算，结果存回该直接地址中	3	2

## 8.3 逻辑指令

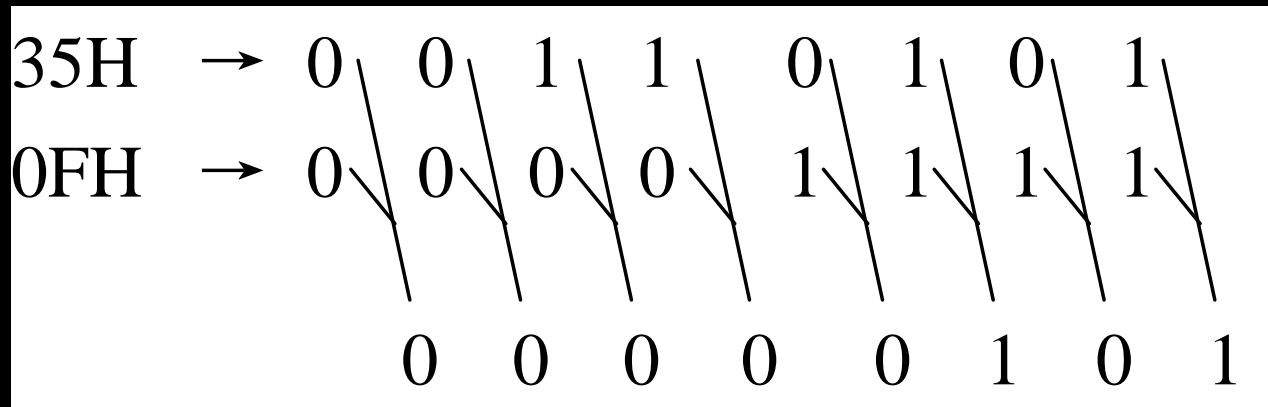
### • 8.3.1 AND操作——ANL <dest-byte>,<src-byte>

例:

MOV A, #35H ; ACC=35H

ANL A, #0FH ; ACC=35H AND 0FH

运算过程:



X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

运算结果: 0000 0101 → ACC=05H

## 8.3 逻辑指令

### • 8.3.2 OR操作——ORL <dest-byte>,<src-byte>

指 令	说 明	字节	机器周期
ORL A,Rn	将Rn与ACC的值做OR运算，结果存回ACC	1	1
ORL A,direct	将直接地址direct的内容与ACC做OR运算，结果存回ACC	2	1
ORL A,@Ri	将间接地址@Ri的内容与ACC做OR运算，结果存回ACC	1	1
ORL A,#data	立即数#data与ACC做OR运算，结果存回ACC	2	1
ORL direct,A	ACC与直接地址direct的内容做OR运算，结果存回该直接地址中	2	1
ORL direct,#data	立即数#data与直接地址direct的内容做OR运算，结果存回该直接地址中	3	2

## 8.3 逻辑指令

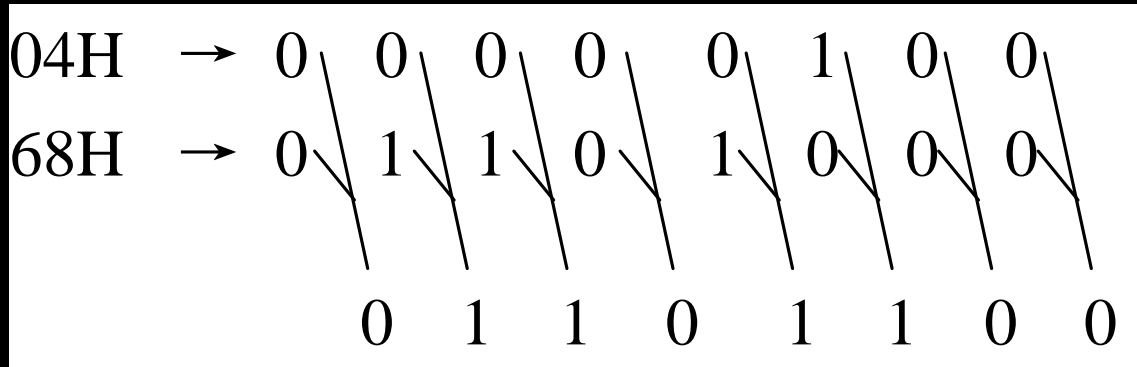
### 8.3.2 OR操作——ORL <dest-byte>, <src-byte>

例:

MOV A, #04H ; ACC=04H

ORL A, #68H ; ACC=04H OR 68H

运算过程:



X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	1

运算结果: **0110 1100 → ACC=6CH**

## 8.3 逻辑指令

### • 8.3.3 XOR操作——XRL <dest-byte>, <src-byte>

指 令	说 明	字节	机器周期
XRL A,Rn	将Rn与ACC的值做XOR运算，结果存回ACC	1	1
XRL A,direct	将直接地址direct的内容与ACC做XOR运算，结果存回ACC	2	1
XRL A,@Ri	将间接地址@Ri的内容与ACC做XOR运算，结果存回ACC	1	1
XRL A,#data	立即数#data与ACC做XOR运算，结果存回ACC	2	1
XRL direct,A	ACC与直接地址direct的内容做XOR运算，结果存回该直接地址中	2	1
XRL direct,#data	立即数#data与直接地址direct的内容做XOR运算，结果存回该直接地址中	3	2

## 8.3 逻辑指令

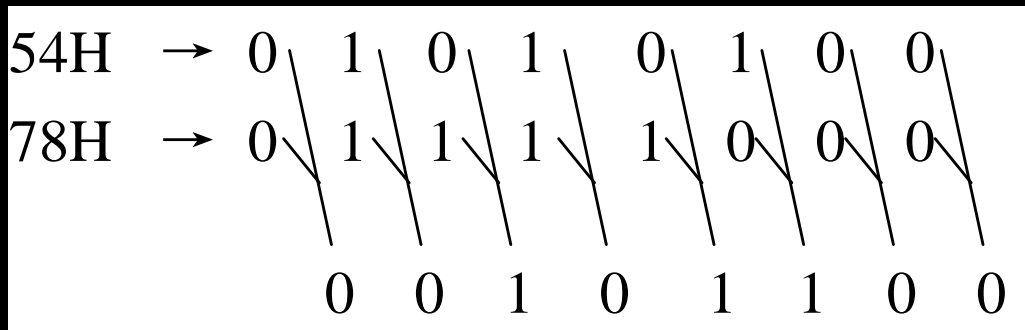
### • 8.3.3 XOR操作——XRL <dest-byte>, <src-byte>

例:

MOV A, #54H ; ACC=54H

XRL A, #78H ; ACC=54H XOR 78H

运算过程:



运算结果: **0010 1100 → ACC=2CH**

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

## 8.3 逻辑指令

### • 8.3.4 清0操作——CLR A

指 令	说 明	字 节	机器周期
CLR A	累加器ACC清0	1	1

说明：将累加器ACC清0。

执行指令“CLR A”后，ACC=00H。



## 8.3 逻辑指令

### • 8.3.5 取反操作——CPL A

指 令	说 明	字 节	机器周期
CPL A	累加器ACC每一位的值反相	1	1

例：

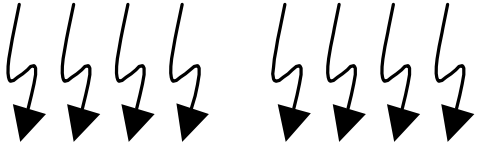
MOV A, #47H ; ACC=47H

CPL A ; ACC=B8H

运算过程：

X	Y
0	1
1	0

取反每一位，  
运算结果：

47H → 0 1 0 0 0 1 1 1  
  
 1 0 1 1 1 0 0 0 → ACC = B8H

## 8.3 逻辑指令

### • 8.3.6 位移动操作——RL、RLC、RR、RRC

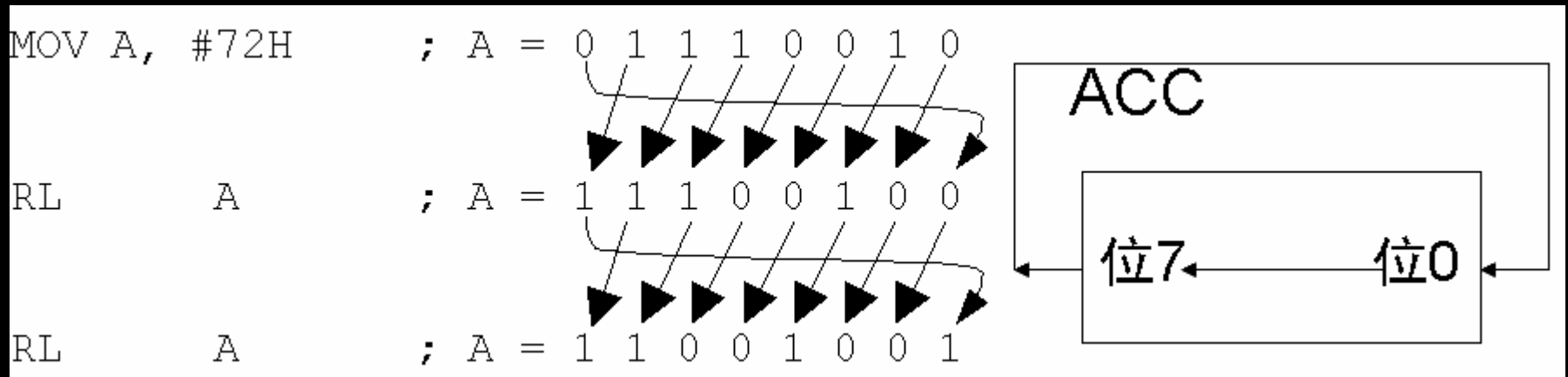
指 令	说 明	字 节	机器周期
RL A	累加器ACC左移一位	1	1
RLC A	累加器ACC含进位CY左移一位	1	1
RR A	累加器ACC右移一位	1	1
RRC A	累加器ACC含进位CY右移一位	1	1

- 这4条指令用于累加器ACC内部位的移动，这4条指令只适用于ACC。

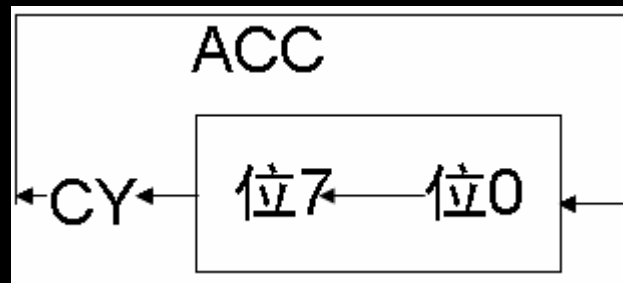
## 8.3 逻辑指令

### • 8.3.6 位移动操作——RL、RLC、RR、RRC

① **RL A**——ACC左移一位。每次移出ACC的位7进入位0。



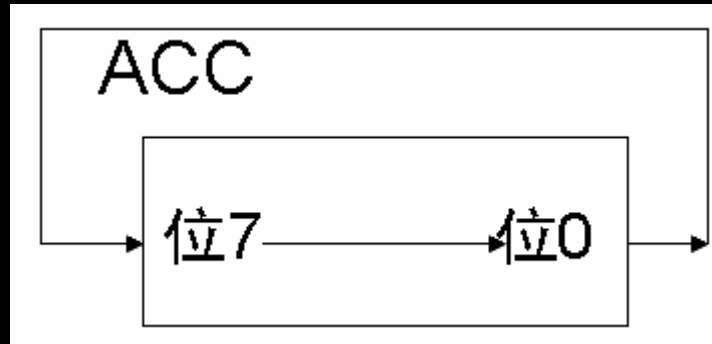
② **RLC A**——ACC含进位CY左移一位。每次移出ACC的位7进入进位CY中，而进位CY则进入位0中。



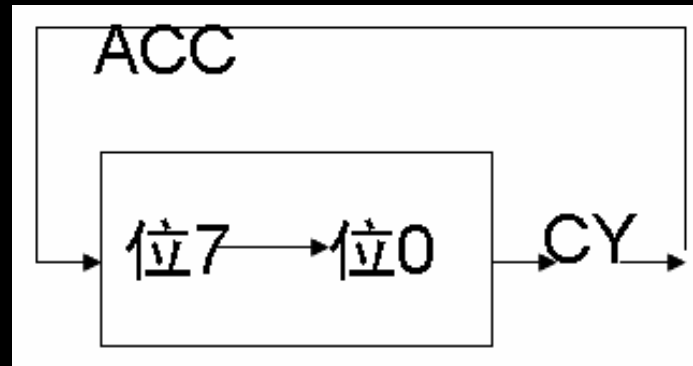
## 8.3 逻辑指令

### • 8.3.6 位移动操作——RL、RLC、RR、RRC

③ RR A——ACC右移一位。每次移出ACC的位0进入位7。



④ RRC A——ACC含进位CY右移一位。每次移出ACC的位0进入进位CY中，而进位CY则进入位7中。



## 8.3 逻辑指令

### • 8.3.7 累加器高低位交换操作——SWAP A

指 令	说 明	字 节	机器周期
SWAP A	累加器ACC的高4位与低4位互换	1	1

例：

MOV     A, #28H                                 ; ACC=28H

SWAP    A   ; ACC=82H

运行结果：

累加器ACC中的高4位与低4位互换，得  
ACC=82H。

## 8.4 片内数据装载指令

- 片内数据装载指令是一类在片内存储器内“搬运”数据的指令，在单片机程序中使用频率最高。
- 这类指令中一多半都以MOV为助记符，指令形式为“MOV <dest>, <src>”，其中<dest>为目的操作数，<src>为源操作数。
- <dest>和<src>代表片内数据存储器空间地址或特殊功能寄存器SFR，指令在进行不同地址空间或寄存器之间装载时不需要通过累加器ACC的参与。

# 8.4 片内数据装载指令

## • 8.4.1 数据装载指令——MOV <dest>, <src>

指 令	说 明	字 节	机 器 周 期
MOV A,Rn	将Rn的内容载入ACC	1	1
MOV A,direct	将直接地址direct的内容载入ACC	2	1
MOV A,@Ri	将间接地址@Ri的内容载入ACC	1	1
MOV A,#data	将立即数#data载入ACC	2	1
MOV Rn,A	将ACC的值载入Rn	1	1
MOV Rn,direct	将直接地址的内容载入Rn	2	2
MOV Rn,#data	将立即数#data载入Rn	2	1
MOV direct,A	将ACC的值载入直接地址direct中	2	1
MOV direct,Rn	将Rn的值载入直接地址direct中	2	2

## 8.4 片内数据装载指令

### • 8.4.1 数据装载指令——MOV <dest>, <src>

指 令	说 明	字节	机器周期
MOV direct1,direct2	将直接地址direct2中的内容载入直接地址direct1中	3	2
MOV direct,@Ri	将间接地址@Ri的内容载入直接地址direct中	2	2
MOV direct,#data	将立即数#data载入直接地址direct中	3	2
MOV @Ri,A	将ACC的值载入间接地址@Ri中	1	1
MOV @Ri,direct	将直接地址direct的内容载入间接地址@Ri中	2	2
MOV @Ri,#data	将立即数#data载入间接地址@Ri中	2	1



## 8.4 片内数据装载指令

### • 8.4.1 数据装载指令——MOV <dest>, <src>

根据目的操作数的不同，MOV指令分成以累加器ACC、工作寄存器Rn、直接地址direct、间接地址@Ri为目的操作数的4种不同形式。

MOV指令都不会改变源操作数，也不会影响标志位。

注意：

- ① 立即数前必须要有“#”号，否则装载的将是地址空间上的数据（直接地址的内容）。
- ② 如果立即数小于10H，即0~F，则高位会被系统自动补上0。
- ③ 累加器ACC或工作寄存器等一些寄存器加载大于FFH的立即数将会引发错误，因为这些寄存器的长度只有1个字节。

## 8.4 片内数据装载指令

- 8.4.2 数据指针寄存器装载指令——MOV DPTR, #data16

指 令	说 明	字 节	机器周期
MOV DPTR, #data16	将16位的立即数#data载入 DPTR	3	2

- ✓ 说明：该指令用于向数据指针寄存器DPTR装载16位的地址数据，地址空间的范围为0000H~FFFFH。
- ✓ AT89S51单片机为了方便同时访问片内、片外数据存储单元，提供了两个16位的DPTR寄存器——DPTR0和DPTR1。在特殊功能寄存器中，每一个DPTR分成两个8位的寄存器DPH和DPL。
- ✓ 例：当AUXR1中的DPS位=0时，指令影响DPTR0，“MOV DPTR, #33FFH”，向DP0H装载33H，向DP0L装载FFH。

## 8.4 片内数据装载指令

### • 8.4.3 堆栈指令——PUSH和POP

堆栈（**stack**）是单片机在片内数据存储器中开辟用于存储临时数据的区域。

程序运行过程中需要把某寄存器或地址空间中的数据暂时保存起来，以释放寄存器或地址空间暂作它用，等使用完毕后，再将刚才暂时保存的数据恢复到寄存器或地址空间中。

堆栈指针**SP**在特殊功能寄存器中的**81H**上，是一个8位的寄存器。

## 8.4 片内数据装载指令

### • 8.4.3 堆栈指令——PUSH和POP

- ✓ 单片机上电复位时SP=07H，SP指向的是片内数据存储器中的07H，正好是第0组工作寄存器R7的地址。此时如果执行压栈指令“PUSH direct”，执行的步骤是SP先自增1，于是SP=08H，然后direct的数据就压入08H中，此时（08H）=direct，数据就被保存到了08H上。

指 令	说 明	字 节	机器周期
PUSH direct	将直接地址direct的内容压入堆栈中，执行前SP+1	2	2
POP direct	从堆栈中弹出数据到直接地址direct中，执行后SP-1	2	2

## 8.4 片内数据装载指令

### • 8.4.3 堆栈指令——PUSH和POP（例）

```
ORG          00H
MOV          A, #89H          ; 向累加器ACC中载入立即数89H
PUSH        ACC              ; 压ACC的数值入堆栈, (08H) =89H
MOV          A, #3FH          ; ACC=3FH
MOV          R0, #55H         ; R0=55H
ADD          A, R0            ; ACC=94H
MOV          30H, A           ; (30H) =94H, ACC完成运算任务
POP  ACC ; 弹栈指令将刚才压入堆栈的立即数89H弹回到ACC中,
      ACC=89H
MOV          SP, #40H         ; 堆栈指针SP=40H
PUSH  30H    ; 把30H中的数据94H压入堆栈中, (41H) =94H
JMP          $               ; 停机
END
```

## 8.4 片内数据装载指令

### • 8.4.4 数据交换指令——XCH和XCHD

指 令	说 明	字节	周期
XCH A,Rn	将累加器ACC与Rn的值交换	1	1
XCH A,direct	将累加器ACC与直接地址direct的内容交换	1	2
XCH A,@Ri	将累加器ACC与间接地址@Ri的内容交换	1	1
XCHD A,@Ri	将累加器ACC的低4位与间接地址@Ri的低4位内容交换	1	1

XCH指令交换累加器ACC和工作寄存器、直接或间接地址的内容。例如，ACC=65H，R2=99H，执行“XCH A, R2”之后，ACC=99H，R2=65H。

指令XCHD交换的只是累加器ACC与间接地址的低4位内容，高4位内容不变。例如，地址空间（40H）=2FH，ACC=3DH，R1=40H，执行“XCHD A, @R1”之后，ACC=3FH，（40H）=2DH。

## 8.5 片外数据装载指令

- 访问单片机的片外存储器，寻址方式只有一种——间接寻址。用到的操作数除了累加器ACC外，还有1个字节长度的间接地址——@Ri（R0或R1）或2个字节长度的间接地址——@DPTR。至于程序中选择使用@Ri还是@DPTR要根据寻址的空间大小来确定。

指 令	说 明	字节	机器周期
MOVX A,@Ri	将间接地址@Ri所指的片外存储器的内容载入ACC中（8位）	1	2
MOVX A,@DPTR	将DPTR所指定的片外存储器的内容载入ACC中（16位）	1	2
MOVX @Ri,A	将ACC的值载入间接地址@Ri所指的片外存储器中（8位）	1	2
MOVX @DPTR,A	将ACC的值载入DPTR所指的片外存储器中（16位）	1	2

## 8.5 片外数据装载指令

- 例：片外程序存储器的1000H开始存储有8个数据，循环将这8个数据送到P1口用于显示。

```

MYXDATA EQU 1000H ; 开始地址1000H赋给变量MYXDATA
COUNT EQU 8 ; 计数器COUNT=8
ORG 00H ; 起始地址
START:
MOVDPTR, #MYXDATA ; DPTR指向地址1000H, DPTR = 1000H
MOVR1, #COUNT ; 计数器值载入R1, R1 = 8
LOOP:
MOVX A, @DPTR ; 间接寻址, @DPTR所指片外数据载入ACC
MOV P1, A ; 送到P1口输出
INC DPTR ; DPTR自增1, 准备取下一个数据
DJNZ R1, LOOP ; 判断是否取完8个数据
JMP START ; 跳回一开始, 循环
END

```



## 8.6 查表指令

指 令	说 明	字 节	机 器 周 期
MOVC A, @A+DP TR	ACC加DPTR的值作为间接地址，将该地址的内容载入ACC中	1	2
MOVC A, @A+PC	ACC加PC的值作为间接地址，将该地址的内容载入ACC中	1	2

✓使用“MOVC A, @A+DPTR”前，数据指针寄存器DPTR一般载入数据表的表头地址，这样指令就能实现将数据表中的数据逐一读入累加器ACC中。

## 8.6 查表指令

- 指令“MOVC A, @A+PC”讲解：

数据表TABLE要紧跟在USETABLE子程序段RET后。PC指向RET的地址，如果设置访问序号ENTRY\_NUMBER=1，那么指令“MOVC A, @A+PC”就指向RET的下一个地址，即数据表TABLE的表头。

MOVA, ENTRY\_NUMBER ; 将访问序号ENTRY\_NUMBER载入ACC  
CALL USETABLE ; 调子程序USETABLE

.....

USETABLE: ; USETABLE子程序段  
MOVC A, @A+PC ; 此时PC已能指向RET随后的表TABLE  
RET

TABLE: ; 数据表TABLE  
DB 0FEH, 0FDH, 0FBH, 0F7H  
DB 0EFH, 0DFH, 0BFH, 07FH

.....

END

## 8.7 布尔指令

- 51单片机提供了一套完整的布尔指令，布尔指令是一类针对位操作的指令。
- 除了片内数据存储器提供位寻址区外（20H～2FH），特殊功能寄存器SFR也提供了128个可寻址位，I/O口也可以进行位操作。
- 单片机提供位寻址的机制，是为了增强系统的灵活性。

## 8.7 布尔指令

### • 8.7.1 清0、置1与取反操作——CLR、SETB、CPL

指 令	说 明	字 节	机器周期
CLR C	进位CY清0	1	1
CLR bit	将可位寻址的位清0	2	1
SETB C	将进位CY置1	1	1
SETB bit	将可位寻址的位置1	2	1
CPL C	将进位CY反向	1	1
CPL bit	将可位寻址的位反向	2	1

“bit”代表可位寻址的地址或寄存器，包括：  
片内数据存储器的位寻址区20H~2FH（映射地址），  
累加器ACC，B寄存器，程序状态字PSW，  
中断优先级控制寄存器IP，中断使能寄存器IE，  
串行控制寄存器SCON，定时/计数器控制寄存器TCON，  
I/O口：P0、P1、P2、P3。

另外：对进位CY操作的指令会直接影响标志位。

# 8.7 布尔指令

## • 8.7.2 布尔逻辑指令——ANL、ORL

指 令	说 明	字 节	机器周 期
ANL C,bit	将进位CY与可位寻址的位做AND运算，结果存回进位CY	2	2
ANL C,/bit	将进位CY与可位寻址的位的反相做AND运算，结果存回进位CY	2	2
ORL C,bit	将进位CY与可位寻址的位做OR运算，结果存回进位CY	2	2
ORL C,/bit	将进位CY与可位寻址的位的反相做OR运算，结果存回进位CY	2	2

说明：这类指令是对位的逻辑运算，执行结果存回进位CY，影响标志位。

## 8.7 布尔指令

- 8.7.3 位数据装载指令——MOV <dest-bit>,<src-bit>

指 令	说 明	字节	机器周期
MOV C,bit	将可位寻址的位载入进位CY	2	1
MOV bit,C	将进位CY载入可位寻址的位	2	2

说明：指令完成的是位之间的装载，

例：

```
SETBC ; CY=1
```

ORL C, P2.0 ; P2.0与CY做OR运算，由于CY=1，  
;所以无论P2.0是1或0，OR运算结果都为1，于是CY=1

## 8.7 布尔指令

### • 8.7.4 布尔跳转指令——JC、JNC、JB、JNB、

指 令	说 明	字节	机器周期
JC rel	若进位CY=1则跳到rel	2	2
JNC rel	若进位CY=0则跳到rel	2	2
JB bit,rel	若可位寻址的位=1则跳到rel	3	2
JNB bit,rel	若可位寻址的位=0则跳到rel	3	2
JBC bit,rel	若可位寻址的位=1则跳到rel，并将该位清0	3	2

说明：以上指令检测进位CY或可位寻址的位，如果条件成立就跳转到rel所指的地址，否则执行下一条指令。

## 8.8 调用子程序指令

- 调用子程序指令用于调用子程序，子程序一般是一些被频繁使用的实现特定功能的程序段，如延时子程序等。使用子程序可以节省存储空间和使程序结构科学化。在单片机的指令系统中，有两条指令用于调用子程序——**LCALL**（长调用）和**ACALL**（绝对调用），它们的主要区别在于调用子程序的目标地址范围不同。对于**AT89S51**单片机来说，可以笼统地使用**CALL**指令来自动配置子程序目标地址的范围。



## 8.8 调用子程序指令

### • 8.8.1 长调用指令——LCALL

指 令	说 明	字节	机器周期
LCALL addr16	调用整个64K Bytes程序存储器范围内的子程序	3	2

LCALL拥有两个字节来指定子程序的目标地址，调用地址范围达64K Bytes，即0000H~FFFFH。为了确保子程序执行完毕之后单片机能正确回到原来主程序的地址继续执行，LCALL执行时把LCALL指令的下一条指令的地址保存在堆栈中。

## 8.8 调用子程序指令

### • 8.8.2 绝对调用指令——ACALL

指 令	说 明	字 节	机器周期
ACALL addr11	调用2K Bytes程序存储器范围内的子程序	2	2

ACALL指令长度为2个字节，2个字节中的11位用来指向子程序的地址，所以说子程序的目标地址不能超过2K Bytes的范围。

## 8.8 调用子程序指令

### • 8.8.3 返回指令——RET和RETI

指 令	说 明	字 节	机器周期
RET	子程序执行完毕后返回主程序	1	2
RETI	中断子程序执行完毕后返回主程序	1	2

- **RET**指令用于子程序的末尾，提示子程序结束，以返回主程序，**RETI**指令用于结束中断服务子程序。

## 8.9 跳转与循环指令

- 跳转与循环指令在程序中使用相当频繁，它们控制程序的走向和实现循环执行特定程序段。
- 用好跳转与循环指令，可以使系统程序得到优化和节省程序存储空间。

## 8.9 跳转与循环指令

### • 8.9.1 无条件跳转指令——LJMP、AJMP、SJMP、JMP

指 令	说 明	字节	机器周期
LJMP addr16	长跳转	3	2
AJMP addr11	绝对跳转	2	2
SJM P rel	短跳转（相对地址）	2	2
JMP @A+DPTR	间接跳转至@A+DPTR 所指的地址	1	2

LJMP是长跳转指令，跳转地址范围达64K Bytes。

AJMP是绝对跳转指令，目标地址不能超过2K Bytes。

SJMP是一个相对跳转指令，跳转范围为向前128个字节，向后127个字节。

JMP一个笼统的跳转指令，自动判断跳转的范围。

## 8.9 跳转与循环指令

### • 8.9.2 条件跳转指令——JZ和JNZ

指 令	说 明	字 节	机器周期
JZ rel	ACC=0则跳转到rel	2	2
JNZ rel	ACC≠0则跳转到rel	2	2

这是两条以累加器ACC的值为条件的跳转指令。

JZ指令，当ACC=0时程序跳转到所指定的地址开始执行；

JNZ指令，当ACC≠0时才跳转到指定地址开始执行。

# 8.9 跳转与循环指令

## • 8.9.2 条件跳转指令——JZ和JNZ

行号	PC	执行代	指令	
01	0000		ORG	00H
02	0000	7800	MOV	R0, #00H
03	0002	7455	MOV	A, #55H
04	0004	6003	JZ	NEXT
05	0006	08	INC	R0
06	0007	04	LOOP: INC	A
07	0008	04	INC	A
08	0009	2477	NEXT: ADD	A, #77H

分析：第4行的指令“JZ NEXT”，执行代码由60和03组成，分别存放在0004和0005地址空间中，执行代码03就是一个相对地址。“JZ NEXT”的下一条指令（第5行）的PC值为0006，所以跳转到 $0006+03=0009$ 的目标地址，即第8行的NEXT标号处。

## 8.9 跳转与循环指令

- 8.9.3 比较跳转指令——CJNE <dest-byte>, <src-byte>, rel

指 令	说 明	字 节	机 器 周 期
CJNE A,direct,rel	将直接地址direct的内容与ACC作比较，不相等则跳转到rel	3	2
CJNE A,#data,rel	将ACC的值与立即数#data作比较，不相等则跳转到rel	3	2
CJNE Rn,#data,rel	将立即数#data与Rn作比较，不相等则跳转到rel	3	2
CJNE @Ri,#data,rel	将立即数#data与间接地址@Ri的内容作比较，不相等则跳转到rel	3	2



## 8.9 跳转与循环指令

### • 8.9.3 比较跳转指令——CJNE <dest-byte>, <src-byte>, rel

- ✓ CJNE指令是“compare and jump if not equal”的缩写，意思是将源操作数<src-byte>与目的操作数<dest-byte>的值相比，如果不相等就跳转到rel所指的地址。
- ✓ 它还改变进位标志CY的值以显示目的操作数较大还是较小。
- ✓ CJNE指令不会改变源操作数或目的操作数的值。

比较条件	影响进位CY
目的操作数>源操作数	CY=0
目的操作数<源操作数	CY=1

## 8.9 跳转与循环指令

### • 8.9.4 循环指令——DJNZ <byte>,<rel-addr>

单片机中将一段程序重复执行一定次数的过程称为循环，通过指令DJNZ来实现的循环是一种最广泛应用的方法。

DJNZ执行时，工作寄存器或地址内容减1，如果不等于0，则程序跳转到rel指示地址。在执行DJNZ指令前，需要向相关工作寄存器或地址中载入计数值，该计数值就是循环的次数。

指 令	说 明	字 节	机器周期
DJNZ Rn,rel	将Rn的值减1，若不等于0则跳转到rel	3	2
DJNZ direct,rel	将直接地址direct的内容减1，若不等于0则跳转到rel	3	2

## 8.9 跳转与循环指令

### • 8.9.5 无操作指令——NOP

**NOP**指令不进行任何操作，只是空耗费时间，并且更新程序计数器**PC**的计数值。每执行一次花去1个机器周期，执行完**NOP**后，就执行下一条指令。如果程序中需要等待一个很短的时间（若干个机器周期时间），可以结合使用**NOP**和循环指令**DJNZ**来实现。

指 令	说 明	字 节	机器周期
NOP	不执行任何动作	1	1

# 8.10 单片机对于带符号数的处理和溢出问题

## • 8.10.1 无符号数与带符号数

8位时存储无符号数的数值范围是0~255,

7位存储带符号数的数值范围变成了-128~+127。



# 8.10 单片机对于带符号数的处理和溢出问题

## • 8.10.2 二进制的二补数

在数字系统中，使用二进制表示负数时，最高有效位放置的是符号，而数值的大小以二进制的二补数（**2's complement**）形式存放在次高有效位到最低有效位。

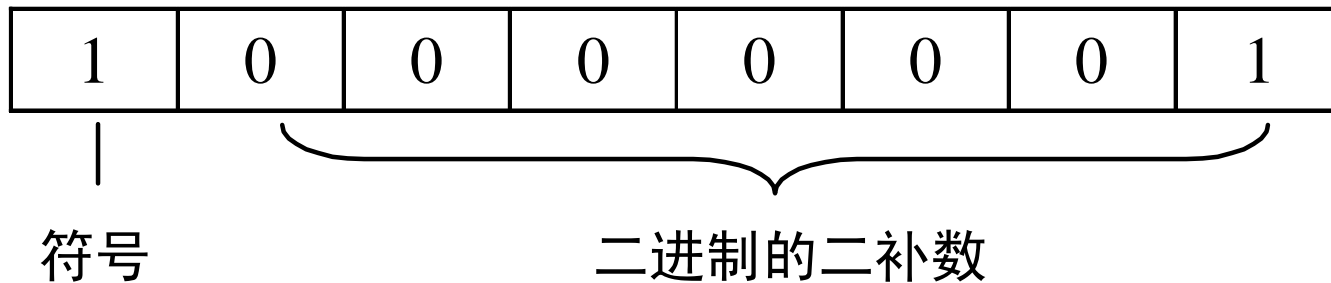
二进制的二补数是这样得到的：

- ① 把负数的绝对值用二进制表示。
- ② 将每一位取反。
- ③ 结果加1。

# 8.10 单片机对于带符号数的处理和溢出问题

## • 8.10.2 二进制的二补数

例：-127这个数，其绝对值127表示成二进制是1111111，将每一位取反得000 0000，最后加1就得到了如图8-6所示的结果：最高有效位1表示是负数，剩下的是二进制的二补数。



# 8.10 单片机对于带符号数的处理和溢出问题

## • 8.10.3 带符号数操作中的溢出问题（例）

MOV           A, #+96                               ; ACC=0110 0000  
  (60H)

MOV           R1, #+70                             ; R1=0100 0110 (46H)

ADD           A, R1                                ; ACC=1010 0110  
  (A6H)

运算结果:

ACC=A6H, OV=1。

分析:

+96和+70是正数（带符号数），它们的和应该是  
 $(+96) + (+70) = +166$ ，而运算的结果ACC=A6H，  
 即-90，溢出标志位OV=1。为什么会出现-90这个错误的  
 结果呢？这是因为+96与+70的和+166超出了8位空间  
 存储带符号数的范围-128~+127，于是溢出标志

# 8.10 单片机对于带符号数的处理和溢出问题

## • 8.10.4 什么时候会溢出

✓ 位6有进位而位7没有进位（CY=0），OV=1。

✓ 位7有进位（CY=1）而位6没有进位，OV=1。

例：MOV           A, # -128    ; ACC=1000 0000 (80H)  
       MOV           R1, # -2     ; R1=1111 1110 (FEH)  
       ADD            A, R1       ; ACC=0111 1110 (7EH)

运算过程：

	1	0	0	0	0	0	0	0
+	1	1	1	1	1	1	1	0
<hr/>								
	1	0	1	1	1	1	1	0

运算结果：

由于位7有进位，而位6没有——溢出。ACC=7EH，OV=1



# 8.10 单片机对于带符号数的处理和溢出问题

- 8.10.5 如何防止进位或溢出产生的错误
  - ✓ 在带符号数运算中，溢出标志OV暗示了运算结果是否正确——OV=1时结果错误，OV=0时结果正确。
  - ✓ 在无符号数操作中我们需要留意进位标志CY，而在带符号数操作中需要留意溢出标志OV。
  - ✓ OV是程序状态字PSW中的一位，且PSW可以被位寻址，所以我们可以用以下的办法来对溢出标志OV进行监视，从而控制程序的走向。  
JB PSW.2, rel ; OV=1（溢出）则跳到rel  
JNB PSW.2, rel ; OV=0（无溢出）则跳到rel

# 8.11 实例点拨

## ——指令应用（程序）实例

### • 8.11.1 数据求和

- ✓ 5个数据以BCD码形式连续存储在片内数据存储器中，起始地址为40H。求这5个数据的和。

(40H) = 78, (41H) = 12, (42H) = 44, (43H) = 81, (44H) = 99,

```
ORG      00H

START:

MOV      R0, #40H      ; R0 作为数据的地址指针
MOV      R2, #5         ; 计数器
CLR      A              ; ACC 清 0
MOV      R7, A          ; R7 清 0, R7 用于存储和的高位字节
```

# 8.11 实例点拨

## ——指令应用（程序）实例

### • 8.11.1 数据求和

```
LOOP:
    ADD    A, @R0        ; 将 R0 所指地址中的内容与 ACC 相加
    DA     A             ; 十进制调整
    JNC    NEXT          ; 如果 CY=0, 说明没有进位, 跳到 NEXT
    INC    R7            ; 如果 CY=1, 说明有进位, 和的高位字节加
NEXT:
    INC    R0            ; 地址指针增加 1
    DJNZ   R2, LOOP      ; 循环, 直到 5 个数据相加完成为止
    JMP    $             ; 停机
END
```

• 运行结果：R7=03，ACC=14。

# 8.11 实例点拨

## ——指令应用（程序）实例

### • 8.11.2 减法与二进制的二补数

- ✓ 带借位的减法指令**SUBB**，在单片机执行该指令时，如果借位**CY=0**，说明被减数大于减数，差是正数；如果借位**CY=1**，说明减数大于被减数，差是负数，且以二进制的二补数的形式存储在累加器**ACC**中。
- ✓ 在**SUBB**执行时，单片机并不是直接将两个数相减，而是按照下面的方法进行：

相减操作 = 被减数 + 减数的二进制的二补数

# 8.11 实例点拨

## ——指令应用（程序）实例

### • 8.11.2 减法与二进制的二补数

例：25H-1EH，减数1EH转换成其二进制的二补数得1110 0010，被减数25H与该二进制的二补数相加。

	0	0	1	0		0	1	0	1
+	1	1	1	0		0	0	1	0
<hr/>									
	1	0	0	0	0		0	1	1

差是0000 0111，即07H。位7有进位，进位标志CY应该是1，但我们使用二进制的二补数形式分析减法时，进位标志CY、辅助进位标志AC、奇偶标志P的值要取反，也就是说CY=0，没有借位，表明差是正数。

# 8.11 实例点拨

## ——指令应用（程序）实例

### • 8.11.2 减法与二进制的二补数

例：4CH-6EH，将减数6E转换成其二进制的二补数得1001 0010，被减数4CH与该二进制的二补数相加。

		0	1	0	0		1	1	0	0
+		1	0	0	1		0	0	1	0
<hr/>										
	0	1	1	0	1		1	1	1	0

- 差是1101 1110，即DEH。这不是一个有效的结果，4CH-6EH应该得到一个负数的结果：-(6E-4C) = -22H。所以需要差DE再进行一次二进制的二补数转换，得0010 0010，即22H。从运算过程来看，位7没有进位，进位标志CY应该是0，但和①情况相同，标志位需要取反，于是得CY=1，有借位，表明差是负数。

# 8.11 实例点拨

## ——指令应用（程序）实例

- 8.11.3 XRL指令应用于比较寄存器数值

XRL指令可用来比较两个寄存器的数值是否相等。例如“XRL A, R1”将比较累加器ACC和工作寄存器R1的数值。如果ACC=R1，XRL执行结果是0，并存回ACC中，否则执行的结果为1，并存回ACC中。然后就可以使用指令JZ或JNZ来判断ACC的值从而控制程序的走向。

# 8.11 实例点拨

## ——指令应用（程序）实例

### • 8.11.3 XRL指令应用于比较寄存器数值

P1口作为数据输入使用，程序判断P1口的输入数据是否等于45H，如果是则执行处理程序PROCESSING，否则继续作判断

```

断例：  ORG          00H
        START:
            MOV  A, #45H          ; 将45H载入ACC
        LOOP:
            MOV  R1, P1           ; 将P1口的输入数据载入R1
            XRL  A, R1             ; 比较ACC和R1的值
            JZ   PROCESSING       ; 如果相等，跳转到 PROCESSING
            JMP  LOOP             ; 如果不相等，跳转到LOOP继续判断
        PROCESSING:
            .....
  
```

XRL指令还可以将位取反。如指令“XRL A, #0000 0100B”能把累加器ACC的位2取反，而保持其他位不变。



# 8.11 实例点拨

## ——指令应用（程序）实例

### • 8.11.4 位数据的串行输出

有些场合，需要把并行的数据转换成串行输出，这时可以利用**RRC**指令，将累加器**ACC**中的位数据输出到进位**CY**中，然后将**CY**中的数据从**I/O**口输出，

RRC	A	； 将ACC的位0数据右移到CY中
MOV	P1. 3, C	； 从P1. 3口输出CY中的数据
RRC	A	； 将第2个位数据移到CY
MOV	P1. 3, C	； 从P1. 3口输出
RRC	A	； 将第3个位数据移到CY
MOV	P1. 3, C	； 从P1. 3口输出

# 8.11 实例点拨

## ——指令应用（程序）实例

### • 8.11.5 利用布尔指令产生矩形波信号

矩形波信号简介：

矩形波中的主要参数：

脉冲幅度：脉冲信号所能达到的最大幅度值。

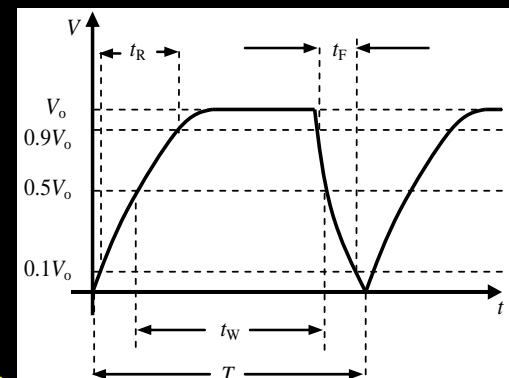
上升时间：脉冲从0.1上升到0.9所经历的时间。

下降时间：脉冲从0.9下降到0.1所经历的时间。

脉冲宽度：脉冲前0.5到后0.5的时间间隔。

脉冲周期 $T$ ：对一个周期性的脉冲波，周期信号指两相邻脉冲出现的时间间隔。

占空比 $D$ ：脉冲宽度与脉冲周期 $T$ 之比称为占空比 $D$ ,



$$D = \frac{t_w}{T} \times 100\%$$

# 8.11 实例点拨

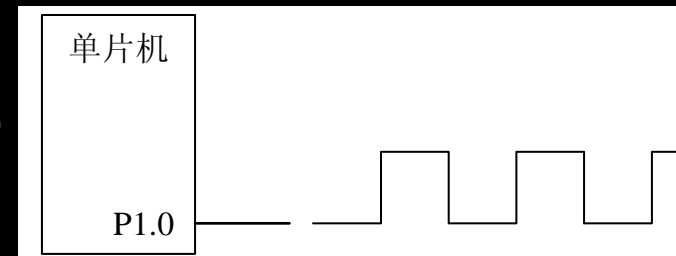
## ——指令应用（程序）实例

### • 8.11.5 利用布尔指令产生矩形波信号

单片机的执行速度很快，指令SETB和CLR在晶振频率为12MHz时只需要1 $\mu$ s，也就是说如果使用某一I/O口作为波形的输出，那上升时间（SETB置1）或下降时间（CLR清0）过程只需1 $\mu$ s，可以应付一般场合的需要。

WAVEOUTPUT:

```
SETB  P1.0      ; P1.0口置1
LCALL DELAY     ; 延时子程序保持1
CLR    P1.0     ; P1.0口清0
LCALL DELAY     ; 延时子程序保持0
SJMP  WAVEOUTPUT ; 循环
```



# 8.11 实例点拨

## ——指令应用（程序）实例

- 8.11.6 布尔指令应用于控制

- ✓ 在一台电话机中，有一个单片机用于系统的控制，假设数据存储器22H上的B2位，即映射地址12H上的数据指示是否有电话打进来。如果映射地址12H上的数据是1，表明有电话打进来。写一段程序，控制液晶屏在映射地址12H=1时，显示“New Incoming Call”，否则显示“Standby”。

# 8.11 实例点拨

## ——指令应用（程序）实例

.....

CHECKBIT:

MOV	C, 12H	; 将 12H 载入进位 CY 中
JNC	STANDBY	; 如果 CY=0, 跳转到 STANDBY
MOV	DPTR, #NEWCALL	; DPTR 指向显示数据表 NEWCALL
LCALL	DISPLAY	; 调显示子程序
SJMP	CHECKBIT	; 循环

STANDBY:

MOV	DPTR, #NOCALL	; 如果没有电话打进, DPTR 指向显示数据表
LCALL	DISPLAY	; 调显示子程序
SJMP	CHECKBIT	; 循环

.....

# 8.11 实例点拨

## ——指令应用（程序）实例

- 8.11.6 布尔指令应用于控制

； 以下是液晶屏的显示数据表

```
NEWCALL :    DB      'New Incoming Call'  
NOCALL:      DB      'Standby'  
  
END
```