**VSCODE 8051汇编指令集说明**

文件编号： 版本号：v0  
编写人：吴华桥 审核人：

批准人： 编写日期：2025-07-25

# 1.指令集概述

8051指令集包含111条指令，49条单字节指令，46条双字节指令，16条三字节指令。指令由操作码(指明指令类型)和一个或多个操作数(指定指令操作)组成。

8051指令集可以分为以下几种基本类型：

* 算术运算类指令
* 逻辑运算类指令
* 数据传送类指令
* 布尔/位运算类指令
* 程序分支类指令

每个指令周期由1个振荡周期组成(即1指令周期=1机器周期)，单字节指令对应1个指令周期，双字节指令对应2个指令周期，三字节指令对应3个指令周期。因此，对于频率为16MHz的系统时钟，1个指令周期的时间为62.5ns。

# 2.字段说明

|  |  |
| --- | --- |
| 字段 | 说明 |
| A | 累加器A |
| B | 寄存器B |
| direct | 直接寻址寄存器IRAM或SFR的地址 |
| data | 数据值 |
| Ri | R0~R1通用寄存器 |
| Rn | R0~R7通用寄存器 |
| DPTR | 16位数据指针，用于存储外部数据存储器或程序存储器的地址 |
| PC | 程序计数器，用于存储下一条待执行指令的内存地址 |
| C | 第7位产生进位或借位的标志位CY |
| bit | 位地址(内部RAM(IRAM或IDATA)区0x20~0x2F，共16字节，16\*8=128位对应0x00~0x7F位地址，例如：bit=0x00时，对应0x20的bit0，bit=0x08时，对应0x21的bit0，bit=0x7F时，对应0x2F的bit7) |
| IRAM | 内部RAM(IRAM或IDATA) |
| SFR | 特殊功能寄存器 |
| XDATA | 外部数据存储器 |
| CODE | 程序存储器 |
| DATA | 内部RAM低128字节 |
| IDATA | 内部RAM全256字节 |
| AC | 从第3位到第4位产生进位时的辅助进位标志位 |

# 3.指令集简介

|  |  |  |  |
| --- | --- | --- | --- |
| 指令 | 说明 | 操作码 | 大小(字节) |
| 算术运算指令 | | | |
| [ADD A, Rn](#_4.1.1_ADD_A,) | A的值加Rn内的值，结果给A | 0x28..  ..0x2F | 1 |
| [ADD A, direct](#_4.1.2_ADD_A,) | A的值加direct内的值，结果给A | 0x25 | 2 |
| [ADD A, @Ri](#_4.1.3_ADD_A,) | Ri内的值作为间接寻址地址，A加该地址内的值后，结果给A | 0x26  0x27 | 1 |
| [ADD A, #data](#_4.1.4_ADD_A,) | A的值加立即数，结果给A | 0x24 | 2 |
| [ADDC A, Rn](#_4.1.5_ADDC_A,) | A的值加Rn内的值，结果给A | 0x38..  ..0x3F | 1 |
| [ADDC A, direct](#_4.1.6_ADDC_A,) | A的值加C及direct内的值，结果给A | 0x35 | 2 |
| [ADDC A, @Ri](#_4.1.7_ADDC_A,) | A的值加C及Ri内的值，结果给A | 0x36  0x37 | 1 |
| [ADDC A, #data](#_4.1.8_ADDC_A,) | A的值加data及C的值，结果给A | 0x34 | 2 |
| [SUBB A, Rn](#_4.1.9_SUBB_A,) | A的值减C及Rn内的值，结果给A | 0x98..  ..0x9F | 1 |
| [SUBB A, direct](#_4.1.10_SUBB_A,) | A的值减C及direct内的值，结果给A | 0x95 | 2 |
| [SUBB A, @Ri](#_4.1.11_SUBB_A,) | A的值减C及Ri内的值，结果给A | 0x96  0x97 | 1 |
| [SUBB A, #data](#_4.1.12_SUBB_A,) | A的值减data及C的值，结果给A | 0x94 | 2 |
| [INC A](#_4.1.13_INC_A) | A的值加1，结果给A | 0x04 | 1 |
| [INC Rn](#_4.1.14_INC_Rn) | Rn内的值加1，结果给Rn | 0x08..  ..0x0F | 1 |
| [INC direct](#_4.1.15_INC_direct) | direct内的值加1，结果给direct | 0x05 | 2 |
| [INC @Ri](#_4.1.16_INC_@Ri) | Ri内的值加1，结果给Ri | 0x06  0x07 | 1 |
| [INC DPTR](#_4.1.17_INC_DPTR) | DPTR内的值加1，结果给DPTR | 0xA3 | 1 |
| [DEC A](#_4.1.18_DEC_A) | A的值减1，结果给A | 0x14 | 1 |
| [DEC Rn](#_4.1.19_DEC_Rn) | Rn内的值减1，结果给Rn | 0x18..  ..0x1F | 1 |
| [DEC direct](#_4.1.20_DEC_direct) | direct内的值减1，结果给direct | 0x15 | 2 |
| [DEC @Ri](#_4.1.21_DEC_@Ri) | Ri内的值减1，结果给Ri | 0x16  0x17 | 1 |
| [MUL AB](#_4.1.22_MUL_AB) | A与B的值相乘，结果高8位给B，低8位给A | 0xA4 | 1 |
| [DIV AB](#_4.1.23_DIV_AB) | A除以B的值，商给A，余数给B | 0x84 | 1 |
| [DA A](#_4.1.24_DA_A) | 校正相加后的BCD码 | 0xD4 | 1 |
| 逻辑运算指令 | | | |
| [ANL A, Rn](#_4.2.1_ANL_A,) | A的值与Rn内的值进行位与运算，结果给A | 0x58..  ..0x5F | 1 |
| [ANL A, direct](#_4.2.2_ANL_A,) | A的值与direct内的值进行位与运算，结果给A | 0x55 | 2 |
| [ANL A, @Ri](#_4.2.3_ANL_A,) | Ri内的值作为间接寻址地址，A与该地址内的值进行位与运算，结果给A | 0x56  0x57 | 1 |
| [ANL A, #data](#_4.2.4_ANL_A,) | A的值与data进行位与运算，结果给A | 0x54 | 2 |
| [ANL direct, A](#_4.2.5_ANL_direct,) | direct内的值与A的值进行位与运算，结果给direct | 0x52 | 2 |
| [ANL direct, #data](#_4.2.6_ANL_direct,) | direct内的值与data进行位与运算，结果给direct | 0x53 | 3 |
| [CLR A](#_4.2.7_CLR_A) | 清零A的值 | 0xE4 | 1 |
| [CPL A](#_4.2.8_CPL_A) | 取反A的值 | 0xF4 | 1 |
| [ORL A, Rn](#_4.2.9_ORL_A,) | A的值与Rn内的值进行位或运算，结果给A | 0x48  0x4F | 1 |
| [ORL A, direct](#_4.2.10_ORL_A,) | A的值与direct内的值进行位或运算，结果给A | 0x45 | 2 |
| [ORL A, @Ri](#_4.2.11_ORL_A,) | Ri内的值作为间接寻址地址，A与该地址内的值进行位或运算，结果给A | 0x46  0x47 | 1 |
| [ORL A, #data](#_4.2.12_ORL_A,) | A的值与data进行位或运算，结果给A | 0x44 | 2 |
| [ORL direct, A](#_4.2.13_ORL_direct,) | direct内的值与A的值进行位或运算，结果给direct | 0x42 | 2 |
| [ORL direct, #data](#_4.2.14_ORL_direct,) | direct内的值与data进行位或运算，结果给direct | 0x43 | 3 |
| [RL A](#_4.2.15_RL_A) | A的值循环左移1位 | 0x23 | 1 |
| [RLC A](#_4.2.16_RLC_A) | A的值连同C循环左移1位 | 0x33 | 1 |
| [RR A](#_4.2.17_RR_A) | A的值循环右移1位 | 0x03 | 1 |
| [RRC A](#_4.2.18_RRC_A) | A的值连同C循环右移1位 | 0x13 | 1 |
| [SWAP A](#_4.2.19_SWAP_A) | A的值进行高低4位的位置互换 | 0xC4 | 1 |
| [XRL A, Rn](#_4.2.20_XRL_A,) | A的值与Rn内的值进行按位异或运算，结果给A | 0x68..  ..0x6F | 1 |
| [XRL A, direct](#_4.2.21_XRL_A,) | A的值与direct内的值进行按位异或运算，结果给A | 0x65 | 2 |
| [XRL A, @Ri](#_4.2.22_XRL_A,) | Ri内的值作为间接寻址地址，A与该地址内的值进行按位异或运算，结果给A | 0x66  0x67 | 1 |
| [XRL A, #data](#_4.2.23_XRL_A,) | A的值与data进行按位异或运算，结果给A | 0x64 | 2 |
| [XRL direct, A](#_4.2.24_XRL_direct,) | direct内的值与A的值进行按位异或运算，结果给direct | 0x62 | 2 |
| [XRL direct, #data](#_4.2.25_XRL_direct,) | direct内的值与data进行按位异或运算，结果给direct | 0x63 | 3 |
| 数据传送指令 | | | |
| [MOV A, Rn](#_4.3.1_MOV_A,) | Rn内的值移动至A | 0xE8..  ..0xEF | 1 |
| [MOV A, direct](#_4.3.2_MOV_A,) | direct内的值移动至A | 0xE5 | 2 |
| [MOV A, @Ri](#_4.3.3_MOV_A,) | 将以Ri的值为间接寻址地址，将地址内的值移动至A | 0xE6  0xE7 | 1 |
| [MOV A, #data](#_4.3.4_MOV_A,) | data移动至A | 0x74 | 2 |
| [MOV Rn, A](#_4.3.5_MOV_Rn,) | A的值移动至Rn内 | 0xF8..  ..0xFF | 1 |
| [MOV Rn, direct](#_4.3.6_MOV_Rn,) | direct内的值移动至Rn内 | 0xA8..  ..0xAF | 2 |
| [MOV Rn, #data](#_4.3.7_MOV_Rn,) | data移动至Rn内 | 0x78..  ..0x7F | 2 |
| [MOV direct, A](#_4.3.8_MOV_direct,) | A的值移动至direct内 | 0xF5 | 2 |
| [MOV direct, Rn](#_4.3.9_MOV_direct,) | Rn内的值移动至direct内 | 0x88..  ..0x8F | 2 |
| [MOV direct1, direct2](#_4.3.10_MOV_direct1,) | direct2内的值移动至direct1内 | 0x85 | 3 |
| [MOV direct, @Ri](#_4.3.11_MOV_direct,) | 以Ri的值为地址，将地址内的值移动至direct内 | 0x86  0x87 | 2 |
| [MOV direct, #data](#_4.3.12_MOV_direct,) | data移动至direct内 | 0x75 | 3 |
| [MOV @Ri, A](#_4.3.13_MOV_@Ri,) | A的值移动至以Ri的值为间接寻址地址的地址内 | 0xF6 0xF7 | 1 |
| [MOV @Ri, direct](#_4.3.14_MOV_@Ri,) | direct内的值移动至以Ri的值为间接寻址地址的地址内 | 0xA6  0xA7 | 2 |
| [MOV @Ri, #data](#_4.3.15_MOV_@Ri,) | data移动至以Ri内的值为间接寻址地址的地址内 | 0x76  0x77 | 2 |
| [MOV DPTR, #data16](#_4.3.16_MOV_DPTR,) | 16位地址data移动至DPTR内 | 0x90 | 3 |
| [MOVC A, @A+DPTR](#_4.3.17_MOVC_A,) | DPTR内的值加上偏移量A的值构成的间接寻址地址内的代码移动到A | 0x93 | 1 |
| [MOVC A, @A+PC](#_4.3.18_MOVC_A,) | PC指定的地址加上偏移量A的值构成的间接寻址地址内的代码移动到A | 0x83 | 1 |
| [MOVX A, @Ri](#_4.3.19_MOVX_A,) | 将以Ri内的值与P2内的值所构成的16位地址的外部存储器XDATA内的数据给A | 0xE2  0xE3 | 1 |
| [MOVX A, @DPTR](#_4.3.20_MOVX_A,) | 将以DPTR内的值所指定的16位地址的外部存储器XDATA内的数据给A | 0xE0 | 1 |
| [MOVX @Ri, A](#_4.3.21_MOVX_@Ri,) | 将A的值移动至以Ri与P2内的值所构成的16位地址的外部存储器XDATA内 | 0xF2 0xF3 | 1 |
| [MOVX @DPTR, A](#_4.3.22_MOVX_@DPTR,) | 将A的值移动至DPTR内的值所指定的16位地址的外部存储器XDATA内 | 0xF0 | 1 |
| [PUSH direct](#_4.3.23_PUSH_direct) | 将direct内的值压入堆栈指针指定的地址 | 0xC0 | 2 |
| [POP direct](#_4.3.24_POP_direct) | 将堆栈指针指定的地址内的数据给direct | 0xD0 | 2 |
| [XCH A, Rn](#_4.3.25_XCH_A,) | A的值与Rn内的值进行交换 | 0xC8..  ..0xCF | 1 |
| [XCH A, direct](#_4.3.26_XCH_A,) | A的值与direct内的值进行交换 | 0xC5 | 2 |
| [XCH A, @Ri](#_4.3.27_XCH_A,) | A的值与以Ri内的值为地址，将地址内的值进行交换 | 0xC6  0xC7 | 1 |
| [XCHD A, @Ri](#_4.3.28_XCHD_A,) | A的值与以Ri内的值为地址，将地址内的值的低4位与A的低4位进行交换 | 0xD6  0xD7 | 1 |
| 布尔(位运算)指令 | | | |
| [ANL C, bit](#_4.4.1_ANL_C,) | 将C与读取的位进行逻辑与，结果给C | 0x82 | 2 |
| [ANL C, /bit](#_4.4.2_ANL_C,) | 将C与指定位(bit)反码进行逻辑与，结果给C | 0xB0 | 2 |
| [CLR C](#_4.4.3_CLR_C) | 将C清零 | 0xC3 | 1 |
| [CLR bit](#_4.4.4_CLR_bit) | 将指定位清零 | 0xC2 | 2 |
| [CPL C](#_4.4.5_CPL_C) | 取反C | 0xB3 | 1 |
| [CPL bit](#_4.4.6_CPL_bit) | 取反指定位 | 0xB2 | 2 |
| [MOV C, bit](#_4.4.7_MOV_C,) | 指定位移动至C | 0xA2 | 2 |
| [MOV bit, C](#_4.4.8_MOV_bit,) | C移动至指定位 | 0x92 | 2 |
| [ORL C, bit](#_4.4.9_ORL_C,) | 将C与读取的位进行逻辑或，结果给C | 0x72 | 2 |
| [ORL C, /bit](#_4.4.10_ORL_C,) | 将C与指定位(bit)反码进行逻辑或，结果给C | 0xA0 | 2 |
| [SETB C](#_4.4.11_SETB_C) | 将C置1 | 0xD3 | 1 |
| [SETB bit](#_4.4.12_SETB_bit) | 将指定位置1 | 0xD2 | 2 |
| 程序分支指令 | | | |
| [ACALL addr11](#_4.5.1_ACALL_addr11) | 页绝对调用指令，最大访问页内2kB地址 | 0baaa10001 | 2 |
| [AJMP addr11](#_4.5.2_AJMP_addr11) | 页绝对跳转指令，最大访问页内2kB地址 | 0baaa00001 | 2 |
| [CJNE A, direct, rel](#_4.5.3_CJNE_A,) | A的值与direct内的值比较，不相等跳转至rel地址，且A小于direct内的值C置1，否则C清零 | 0xB5 | 3 |
| [CJNE A, #data, rel](#_4.5.4_CJNE_A,) | A的值与data比较，不相等跳转至rel地址，且A小于data时C置1，否则C清零 | 0xB4 | 3 |
| [CJNE Rn, #data, rel](#_4.5.5_CJNE_Rn,) | Rn内的值与data比较，不相等跳转至rel地址，且Rn内的值小于data时C置1，否则C清零 | 0xB8..  ..0xBF | 3 |
| [CJNE @Ri, #data, rel](#_4.5.6_CJNE_@Ri,) | 以Ri内的值为地址，地址内的值与data比较，不相等跳转至rel地址，且以Ri内的值为地址，地址内的值小于data时C置1，否则C清零 | 0xB6  0xB7 | 3 |
| [DJNZ Rn, rel](#_4.5.7_DJNZ_Rn,) | Rn内的值减1，如果不为0则跳转 | 0xD8..  ..0xDF | 2 |
| [DJNZ direct, rel](#_4.5.8_DJNZ_direct,) | direct内的值减1，如果不为0则跳转 | 0xD5 | 3 |
| [JB bit, rel](#_4.5.9_JB_bit,) | 指定位为1则跳转 | 0x20 | 3 |
| [JBC bit, rel](#_4.5.10_JBC_bit,) | 指定位为1则跳转，设置位为0后，以整个字节的方式写回位地址 | 0x10 | 3 |
| [JC rel](#_4.5.11_JC_rel) | C为1时跳转 | 0x40 | 2 |
| [JZ rel](#_4.5.12_JZ_rel) | A的值为0时跳转 | 0x60 | 2 |
| [JMP @A+DPTR](#_4.5.13_JMP_@A+DPTR) | PC跳转至以A+DPTR内的值作为地址处 | 0x73 | 1 |
| [JNC rel](#_4.5.14_JNC_rel) | C为0时跳转 | 0x50 | 2 |
| [JNB bit, rel](#_4.5.15_JNB_bit,) | 指定位为0则跳转 | 0x30 | 3 |
| [JNZ rel](#_4.5.16_JNZ_rel) | A的值不为0时跳转 | 0x70 | 2 |
| [LCALL addr16](#_4.5.17_LCALL_addr16) | 长调用，可访问64kB地址空间 | 0x12 | 3 |
| [LJMP addr16](#_4.5.18_LJMP_addr16) | 长跳转，可访问64kB地址空间 | 0x02 | 3 |
| [RET](#_4.5.19_RET) | 从子程序返回 | 0x22 | 1 |
| [RETI](#_4.5.20_RETI) | 从中断返回 | 0x32 | 1 |
| [SJMP rel](#_4.5.21_SJMP_rel) | 短跳转，当前PC位置-128~127字节 | 0x80 | 2 |
| [NOP](#_4.5.22_NOP) | 无操作，PC+1 | 0x00 | 1 |

# 4.指令集详解

本节对S8051XC3所执行的所有指令进行详细描述说明。

当提到“立即数”时，指的是相应的值作为指令本身的一部分。

当提到“直接”数据时，指的是使用直接寻址模式对IRAM(内部RAM)或SFR进行数据读写。在这种模式下，地址低于0x80的部分指向IRAM的低128字节，地址高于0x7F则指向特殊功能寄存器(SFR)区域。

当提到“间接”数据时，指的是使用间接寻址模式对IRAM(内部RAM)进行数据读写。地址取自当前选定的(使用PSW.RS位选择)寄存器组中的两个工作寄存器(R0或R1)之一，然后才会进行实际的IRAM(内部RAM)读/写操作。此模式涵盖整个256字节的IRAM空间。

当提到“位”寻址时，使用的一个8位长的位地址，该地址在内部被转换为“直接”字节地址，位地址的最低三个有效位(bit[2:0])决定了该位在其所在字节中的具体位置(0~7)。对于IRAM中的地址(低于0x80)，可位寻址的字节地址范围是0x20~0x2F，对应的位地址范围是0x00~0x7F (因为0x20~0x2F对应16\*8=128位)。对于地址高于0x7F时，位地址会被转换为SFR地址，且最低三位(bit[2:0])被0填充。

## 4.1 算术运算指令详解

### 4.1.1 ADD A, Rn

语法：

ADD A, Rn

操作：

PC ← PC+1

ACC ← ACC + Rn

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 1 | n2 | n1 | n0 |

说明：

选定的工作寄存器(Rn-R0至R7)内的值与累加器A(ACC)的值相加，结果赋给累加器(ACC)。辅助进位标志位(AC)：当第3位向第4位产生进位时，该标志位被置位。

进位标志位(CY)：当第7位产生进位时，该标志被置位。

溢出标志位(OV)：当第6位产生的进位状态与第7位产生的进位状态不同时，该标志位被置位。

示例：

ADD A,R1

指令执行前，A = 0xA0，R1 = 0x05

指令执行后，A = 0xA5，R1 = 0x05

### 4.1.2 ADD A, direct

语法：

ADD A, direct

操作：

PC ← PC+2

ACC ← ACC + (direct)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

读取可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内的值，并将其与累加器(ACC)的值相加，结果赋给累加器(ACC)。

辅助进位标志位(AC)：当第3位向第4位产生进位时，该标志位被置位。

进位标志位(CY)：当第7位产生进位时，该标志被置位。

溢出标志位(OV)：当第6位产生的进位状态与第7位产生的进位状态不同时，该标志位被置位。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

ADD A, Addr\_IRAM或ADD A, 0x30

指令执行前，A = 0xA0，(Addr\_IRAM) = 0x05

指令执行后，A = 0xA5，(Addr\_IRAM) = 0x05

特殊功能寄存器DPL直接操作

ADD A, DPL或ADD A,0x82

指令执行前，A = 0x30，(DPL) = 0x05

指令执行后，A = 0x35，(DPL) = 0x05

### 4.1.3 ADD A, @Ri

语法：

ADD A, @Ri

操作：

PC ← PC+1

ACC ← ACC + (Ri)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值作为间接寻址的地址，读取该地址内的值与累加器A(ACC)的值相加，结果赋给累加器(ACC)。

辅助进位标志位(AC)：当第3位向第4位产生进位时，该标志位被置位。

进位标志位(CY)：当第7位产生进位时，该标志被置位。

溢出标志位(OV)：当第6位产生的进位状态与第7位产生的进位状态不同时，该标志位被置位。

示例：

MOV R0,#0x30 ;R0赋值0x30

ADD A,@R0 ;读取以0x30作为间接寻址的地址内的值与累加器(ACC)相加

指令执行前，A = 0xA0，(R0) = 0x05

指令执行后，A = 0xA5，(R0) = 0x05

### 4.1.4 ADD A, #data

语法：

ADD A, #data

操作：

PC ← PC+2

ACC ← ACC + data

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

将8位立即数与累加器(ACC)的值相加，结果赋给累加器(ACC)。

辅助进位标志位(AC)：当第3位向第4位产生进位时，该标志位被置位。

进位标志位(CY)：当第7位产生进位时，该标志被置位。

溢出标志位(OV)：当第6位产生的进位状态与第7位产生的进位状态不同时，该标志位被置位。

示例：

ADD A,#0x05

指令执行前，A = 0xA0

指令执行后，A = 0xA5

### 4.1.5 ADDC A, Rn

语法：

ADDC A, Rn

操作：

PC ← PC+1

ACC ← ACC + Rn + CY

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 1 | n2 | n1 | n0 |

说明：

将累加器(ACC)的值与选定的工作寄存器(Rn-R0至R7)内的值及进位标志位(CY)相加，结果赋给累加器(ACC)。

辅助进位标志位(AC)：当第3位向第4位产生进位时，该标志位被置位。

进位标志位(CY)：当第7位产生进位时，该标志被置位。

溢出标志位(OV)：当第6位产生的进位状态与第7位产生的进位状态不同时，该标志位被置位。

示例：

MOV R7,#0x05 ;R7赋值立即数0x05

ADDC A,R7

指令执行前，A = 0xA0，R7 = 0x05，CY = 1

指令执行后，A = 0xA6，R7 = 0x05，CY = 0

### 4.1.6 ADDC A, direct

语法：

ADDC A, direct

操作：

PC ← PC+2

ACC ← ACC + (direct) + CY

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

读取可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内的值，并将其与进位标志位(CY)及累加器(ACC)的值相加，结果赋给累加器(ACC)。

辅助进位标志位(AC)：当第3位向第4位产生进位时，该标志位被置位。

进位标志位(CY)：当第7位产生进位时，该标志被置位。

溢出标志位(OV)：当第6位产生的进位状态与第7位产生的进位状态不同时，该标志位被置位。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

ADD A, Addr\_IRAM或ADD A, 0x30

指令执行前，A = 0xA0，(Addr\_IRAM) = 0x05，CY = 1

指令执行后，A = 0xA6，(Addr\_IRAM) = 0x05，CY = 0

特殊功能寄存器DPL直接操作

ADD A, DPL或ADD A,0x82

指令执行前，A = 0x30，(DPL) = 0x05，CY = 1

指令执行后，A = 0x36，(DPL) = 0x05，CY = 0

### 4.1.7 ADDC A, @Ri

语法：

ADDC A, @Ri

操作：

PC ← PC+1

ACC ← ACC + (Ri) + CY

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值作为间接寻址的地址，读取该地址内的值与进位标志位(CY)及累加器A(ACC)的值相加，结果赋给累加器(ACC)。

辅助进位标志位(AC)：当第3位向第4位产生进位时，该标志位被置位。

进位标志位(CY)：当第7位产生进位时，该标志被置位。

溢出标志位(OV)：当第6位产生的进位状态与第7位产生的进位状态不同时，该标志位被置位。

示例：

MOV R0,#0x81 ;R0赋值0x81

ADDC A,@R0

指令执行前，A = 0xA0，(R0) = 0x05，CY = 1

指令执行后，A = 0xA6，(R0) = 0x05，CY = 0

### 4.1.8 ADDC A, #data

语法：

ADDC A, #data

操作：

PC ← PC+2

ACC ← ACC + data + CY

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

将8位立即数与进位标志位(CY)及累加器(ACC)的值相加，结果赋给累加器(ACC)。

辅助进位标志位(AC)：当第3位向第4位产生进位时，该标志位被置位。

进位标志位(CY)：当第7位产生进位时，该标志被置位。

溢出标志位(OV)：当第6位产生的进位状态与第7位产生的进位状态不同时，该标志位被置位。

示例：

ADDC A,#0x05

指令执行前，A = 0xA0，CY = 1

指令执行后，A = 0xA6，CY = 0

ADDC A,#0xFF

指令执行前，A = 0x00，CY = 1

指令执行后，A = 0x00，CY = 1

### 4.1.9 SUBB A, Rn

语法：

SUBB A, Rn

操作：

PC ← PC+1

ACC ← ACC - Rn - CY

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 1 | 1 | n2 | n1 | n0 |

说明：

累加器(ACC)的值减选定的工作寄存器(Rn-R0至R7)内的值与借位标志位(CY)，结果赋给累加器(ACC)。

辅助进位标志位(AC)：当第3位向第4位产生借位时，该标志位被置位。

进位标志位(CY)：当第7位产生借位时，该标志被置位。

溢出标志位(OV)：当第6位产生的借位状态与第7位产生的借位状态不同时，该标志位被置位。

示例：

MOV R7,#0x05 ;R7赋值立即数0x05

SUBB A,R7

指令执行前，A = 0xA6，R7 = 0x05，CY = 1

指令执行后，A = 0xA0，R7 = 0x05，CY = 0

### 4.1.10 SUBB A, direct

语法：

SUBB A, direct

操作：

PC ← PC+2

ACC ← ACC - (direct) - CY

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

累加器(ACC)的值减可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内的值，与借位标志位(CY)，结果赋给累加器(ACC)。

辅助进位标志位(AC)：当第3位向第4位产生借位时，该标志位被置位。

进位标志位(CY)：当第7位产生借位时，该标志被置位。

溢出标志位(OV)：当第6位产生的借位状态与第7位产生的借位状态不同时，该标志位被置位。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

SUBB A, Addr\_IRAM或ADD A, 0x30

指令执行前，A = 0xA6，(Addr\_IRAM) = 0x05，CY = 1

指令执行后，A = 0xA0，(Addr\_IRAM) = 0x05，CY = 0

特殊功能寄存器DPL直接操作

SUBB A, DPL或ADD A,0x82

指令执行前，A = 0x36，(DPL) = 0x05，CY = 1

指令执行后，A = 0x30，(DPL) = 0x05，CY = 0

### 4.1.11 SUBB A, @Ri

语法：

SUBB A, @Ri

操作：

PC ← PC+1

ACC ← ACC - (Ri) - CY

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值作为间接寻址地址，将累加器(ACC)的值减该地址内的值与借位标志位(CY)，结果赋给累加器(ACC)。

辅助进位标志位(AC)：当第3位向第4位产生借位时，该标志位被置位。

进位标志位(CY)：当第7位产生借位时，该标志被置位。

溢出标志位(OV)：当第6位产生的借位状态与第7位产生的借位状态不同时，该标志位被置位。

示例：

MOV R0,#0x81 ;R0赋值0x30

SUBB A,@R0

指令执行前，A = 0xA6，(R0) = 0x05，CY = 1

指令执行后，A = 0xA0，(R0) = 0x05，CY = 0

### 4.1.12 SUBB A, #data

语法：

SUBB A, #data

操作：

PC ← PC+2

ACC ← ACC - data - CY

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

累加器(ACC)减8位立即数与借位标志位(CY)，结果赋给累加器(ACC)。

辅助进位标志位(AC)：当第3位向第4位产生借位时，该标志位被置位。

进位标志位(CY)：当第7位产生借位时，该标志被置位。

溢出标志位(OV)：当第6位产生的借位状态与第7位产生的借位状态不同时，该标志位被置位。

示例：

SUBB A,#0x05

指令执行前，A = 0xA6，CY = 1

指令执行后，A = 0xA0，CY = 0

SUBB A,#0x00

指令执行前，A = 0x00，CY = 1

指令执行后，A = 0xFF，CY = 1

### 4.1.13 INC A

语法：

INC A

操作：

PC ← PC+1

ACC ← ACC + 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

说明：

累加器(ACC)的值增加1，且此操作不影响任何状态标志位。

示例：

INC A

指令执行前，A = 0xA0

指令执行后，A = 0xA1

### 4.1.14 INC Rn

语法：

INC Rn

操作：

PC ← PC+1

Rn ← Rn + 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 | n2 | n1 | n0 |

说明：

选定内部RAM(IRAM)的工作寄存器(Rn-R0至R7)的值增加1，且此操作不影响任何状态标志位。

示例：

INC R7

指令执行前，R7 = 0xA0

指令执行后，R7 = 0xA1

### 4.1.15 INC direct

语法：

INC direct

操作：

PC ← PC+2

(direct) ← (direct) + 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)内的值增加1，且此操作不影响任何状态标志位。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

INC Addr\_IRAM或INC 0x30

指令执行前，(Addr\_IRAM) = 0xA0

指令执行后，(Addr\_IRAM) = 0xA1

特殊功能寄存器DPL直接操作

INC DPL或INC 0x82

指令执行前，(DPL) = 0x05

指令执行后，(DPL) = 0x06

### 4.1.16 INC @Ri

语法：

INC @Ri

操作：

PC ← PC+1

(Ri) ← (Ri) + 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值作为间接寻址的地址内的值增加1，且此操作不影响任何状态标志位。

示例：

MOV R0,#0x81 ;R0赋值0x81

INC @R0

指令执行前，(R0) = 0x05

指令执行后，(R0) = 0x06

### 4.1.17 INC DPTR

语法：

INC DPTR

操作：

PC ← PC+1

DPTR ← DPTR + 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

说明：

数据指针(DPTR)寄存器内的值增加1，且此操作不影响任何状态标志位。

示例：

INC DPTR

指令执行前，DPTR = 0x8005

指令执行后，DPTR = 0x8006

### 4.1.18 DEC A

语法：

DEC A

操作：

PC ← PC+1

ACC ← ACC - 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

说明：

累加器(ACC)的值减少1，且此操作不影响任何状态标志位。

示例：

DEC A

指令执行前，A = 0xA1

指令执行后，A = 0xA0

### 4.1.19 DEC Rn

语法：

DEC Rn

操作：

PC ← PC+1

Rn ← Rn - 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 1 | n2 | n1 | n0 |

说明：

选定内部RAM(IRAM)的工作寄存器(Rn-R0至R7)的值减少1，且此操作不影响任何状态标志位。

示例：

DEC R7

指令执行前，R7 = 0xA1

指令执行后，R7 = 0xA0

### 4.1.20 DEC direct

语法：

DEC direct

操作：

PC ← PC+2

(direct) ← (direct) - 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

由“直接字节”指定的内部RAM(IRAM)或特殊功能寄存器(SFR)内的值减少1，且此操作不影响任何状态标志位。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

DEC Addr\_IRAM或DEC 0x30

指令执行前，(Addr\_IRAM) = 0xA1

指令执行后，(Addr\_IRAM) = 0xA0

特殊功能寄存器DPL直接操作

DEC DPL或DEC 0x82

指令执行前，(DPL) = 0x06

指令执行后，(DPL) = 0x05

### 4.1.21 DEC @Ri

语法：

DEC @Ri

操作：

PC ← PC+1

(Ri) ← (Ri) - 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值作为间接寻址地址,该地址内的值减少1，且此操作不影响任何状态标志位。

示例：

MOV R0,#0x81 ;R0赋值0x81

DEC @R0

指令执行前，(R0) = 0x06

指令执行后，(R0) = 0x05

### 4.1.22 MUL AB

语法：

MUL AB

操作：

PC ← PC+1

{B,ACC} ← ACC \* B

CY ← 0

if (B != 0)

OV ← 1

else

OV ← 0

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

说明：

将累加器(ACC)和B寄存器中的两个无符号8位整数相乘，结果以16位整数形式存储(低8位存储在累加器(ACC)，高8位存储在B寄存器)。

进位标志位CY：强制清零。

溢出标志位OV：结果大于255时置1，否则清零。

示例：

MOV B,#30

MOV A,#10

MUL AB

指令执行前，A = 30，B = 10

指令执行后，A = 0x2C，B = 0x01

### 4.1.23 DIV AB

语法：

DIV AB

操作：

PC ← PC+1

if (B == 0)

OV ← 1

else

OV ← 0

ACC ← ACC / B

B ← ACC % B

CY ← 0

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

说明：

以累加器(ACC)中的8位无符号整数作为被除数，B寄存器中的8位无符号整数作为除数相除，商存储在累加器(ACC)，余数存储在B寄存器。

进位标志位CY：强制清零。

溢出标志位OV：B等于0时置1，否则清零。

示例：

MOV B,#5

MOV A,#101

MUL AB

指令执行前，A = 99，B = 10

指令执行后，A = 0x14，B = 0x01

### 4.1.24 DA A

语法：

DA A

操作：

PC ← PC+1

if (AC || ACC[3:0] > 9)

ACC ← ACC + 0x06

if (CY || ACC[7:4] > 9)

ACC ← ACC + 0x60

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |

说明：

BCD码完成加法运算后，通过累加器(ACC)调整为新的有效的BCD码。原理是首先检测累加器(ACC)的低4位大于9或辅助进位标志位被设置，则将0x06加入累加器(ACC)。然后检测累加器(ACC)的高4位大于9或进位标志位被设置，则将0x60加入累加器(ACC)，如果高4位进行操作后，有进位时，需要通过进位标志位(CY)实现进位操作，从而实现更多位数的BCD计算。

示例：

MOV A,#0x19 ;BCD码19

ADD A,#0x0A ;再加16进制0x0A

DA A

指令执行前，A = 0x23

指令执行后，A = 0x29 ;BCD码29

MOV A,#0x99 ;BCD码99

ADD A,#0x0A ;再加16进制0x0A

DA A

指令执行前，A = 0xA3

指令执行后，A = 0x09，CY = 1;BCD码109

MOV A,#0x99 ;BCD码99

ADD A,#0x99 ;再加16进制0x99

DA A

指令执行前，A = 0x32，CY = 1 ;BCD码变为132错误

指令执行后，A = 0x98，CY = 1 ;BCD码变为198正确

## 4.2 逻辑运算指令详解

### 4.2.1 ANL A, Rn

语法：

ANL A,Rn

操作：

PC ← PC+1

ACC ← ACC AND Rn

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 1 | n2 | n1 | n0 |

说明：

累加器(ACC)与选定的工作寄存器(Rn-R0至R7)进行按位与运算，结果赋给累加器(ACC)。

示例：

MOV R7,#0x11

MOV A,#0x15

ANL A,R7

指令执行前，A = 0x15，R7 = 0x11

指令执行后，A = 0x11，R7 = 0x11

### 4.2.2 ANL A, direct

语法：

ANL A,direct

操作：

PC ← PC+2

ACC ← ACC AND (direct)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

读取可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内的值，并将其与累加器(ACC)进行按位与操作，结果赋给累加器(ACC)。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

ANL A, Addr\_IRAM或ANL A, 0x30

指令执行前，A = 0x15，(Addr\_IRAM) = 0x11

指令执行后，A = 0x11，(Addr\_IRAM) = 0x11

特殊功能寄存器DPL直接操作

ANL A, DPL或ANL A,0x82

指令执行前，A = 0x01，(DPL) = 0x30

指令执行后，A = 0x00，(DPL) = 0x30

### 4.2.3 ANL A, @Ri

语法：

ANL A, @Ri

操作：

PC ← PC+1

ACC ← ACC AND (Ri)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值作为间接寻址的地址，读取该地址内的值与累加器A(ACC)的值进行按位与操作，结果赋给累加器(ACC)。

示例：

MOV R0,#0x81 ;R0赋值0x81

MOV @R0,#0x11

ANL A,@R0

指令执行前，A = 0x15，(R0) = 0x11

指令执行后，A = 0x11，(R0) = 0x11

### 4.2.4 ANL A, #data

语法：

ANL A, #data

操作：

PC ← PC+2

ACC ← ACC AND data

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

累加器A(ACC)地址与指令中提供的8位立即数进行按位与操作，结果赋给累加器(ACC)。

示例：

ANL A,#0x11

指令执行前，A = 0x15

指令执行后，A = 0x11

### 4.2.5 ANL direct, A

语法：

ANL direct,A

操作：

PC ← PC+2

(direct) ← (direct) AND ACC

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)内的值，与累加器(ACC)进行按位与操作，结果赋给直接寻址指定的地址。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

ANL Addr\_IRAM,A或ANL 0x30,A,

指令执行前，A = 0x11，(Addr\_IRAM) = 0x15

指令执行后，A = 0x11，(Addr\_IRAM) = 0x11

特殊功能寄存器DPL直接操作

ANL A, DPL或ANL A,0x82

指令执行前，A = 0x31，(DPL) = 0x32

指令执行后，A = 0x31，(DPL) = 0x30

### 4.2.6 ANL direct, #data

语法：

ANL direct,#data

操作：

PC ← PC+3

(direct) ← (direct) AND data

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

由“直接字节”指定的内部RAM(IRAM)或特殊功能寄存器(SFR)内的值，与指令提供的8位立即数进行按位与操作，结果赋给“直接字节”指定的地址。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

ANL Addr\_IRAM,#0x11 或ANL 0x30, #0x11

指令执行前，(Addr\_IRAM) = 0x15

指令执行后，(Addr\_IRAM) = 0x11

### 4.2.7 CLR A

语法：

CLR A

操作：

PC ← PC+1

ACC ← 0

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |

说明：

累加器(ACC)的值清零，且此操作不影响任何状态标志位。

示例：

CLR A

指令执行前，A = 0xA0

指令执行后，A = 0x00

### 4.2.8 CPL A

语法：

CPL A

操作：

PC ← PC+1

ACC ← !ACC

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |

说明：

累加器(ACC)的值取反，且此操作不影响任何状态标志位。

示例：

CPL A

指令执行前，A = 0xAA

指令执行后，A = 0x55

### 4.2.9 ORL A, Rn

语法：

ORL A,Rn

操作：

PC ← PC+1

ACC ← ACC OR Rn

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 1 | n2 | n1 | n0 |

说明：

累加器(ACC)的值与选定的工作寄存器(Rn-R0至R7)进行按位或运算，结果赋给累加器(ACC)。

示例：

MOV R7,#0x11

MOV A,#0x14

ORL A,R7

指令执行前，A = 0x14，R7 = 0x11

指令执行后，A = 0x15，R7 = 0x11

### 4.2.10 ORL A, direct

语法：

ORL A,direct

操作：

PC ← PC+2

ACC ← ACC OR (direct)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

读取可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)内的值，并将其与累加器(ACC)的值进行按位或操作，结果赋给累加器(ACC)。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

ORL A, Addr\_IRAM或ORL A, 0x30

指令执行前，A = 0x14，(Addr\_IRAM) = 0x11

指令执行后，A = 0x15，(Addr\_IRAM) = 0x11

### 4.2.11 ORL A, @Ri

语法：

ORL A, @Ri

操作：

PC ← PC+1

ACC ← ACC OR (Ri)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值作为间接寻址的地址，读取该地址内的值与累加器A(ACC)的值进行按位或操作，结果赋给累加器(ACC)。

示例：

MOV R0,#0x81 ;R0赋值0x81

ORL A,@R0

指令执行前，A = 0x14，(R0) = 0x11

指令执行后，A = 0x15，(R0) = 0x11

### 4.2.12 ORL A, #data

语法：

ORL A, #data

操作：

PC ← PC+2

ACC ← ACC OR data

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

将8位立即数与累加器(ACC)的值进行按位或操作，结果赋给累加器(ACC)。

示例：

ORL A,#0x05

指令执行前，A = 0xA0

指令执行后，A = 0xA5

### 4.2.13 ORL direct, A

语法：

ORL direct,A

操作：

PC ← PC+2

(direct) ← (direct) OR ACC

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内的值，与累加器(ACC)的值进行按位或操作，结果赋给“直接字节”指定的地址。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

ORL Addr\_IRAM,A或ORL 0x30,A

指令执行前，A = 0x14，(Addr\_IRAM) = 0x11

指令执行后，A = 0x14，(Addr\_IRAM) = 0x15

### 4.2.14 ORL direct, #data

语法：

ORL direct,#data

操作：

PC ← PC+3

(direct) ← (direct) OR data

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内的值，与指令提供的8位立即数进行按位或操作，结果赋给直接寻址地址。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

ORL Addr\_IRAM,#0x11 或ORL 0x30, #0x11

指令执行前，(Addr\_IRAM) = 0x14

指令执行后，(Addr\_IRAM) = 0x15

### 4.2.15 RL A

语法：

RL A

操作：

PC ← PC+1

ACC ← {ACC[6:0],ACC[7]}

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

说明：

累加器(ACC)的值循环左移，且此操作不影响任何状态标志位。

示例：

RL A

指令执行前，A = 0xA0

指令执行后，A = 0x41

### 4.2.16 RLC A

语法：

RLC A

操作：

PC ← PC+1

{CY,ACC} ← {ACC,CY}

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

说明：

累加器(ACC)的值带进位标志位(CY)循环左移，即累加器(ACC)的最高有效位(MSB)移入进位标志位(CY),进位标志位(CY)的旧值移入累加器(ACC)的最低有效位(LSB),其余位向左移动一位。

示例：

RLC A

指令执行前，A = 0xA0，CY = 1

指令执行后，A = 0x41，CY = 1

### 4.2.17 RR A

语法：

RR A

操作：

PC ← PC+1

ACC ← {ACC[0],ACC[7:1]}

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

说明：

累加器(ACC)的值循环右移，且此操作不影响任何状态标志位。

示例：

RR A

指令执行前，A = 0x81

指令执行后，A = 0xC0

### 4.2.18 RRC A

语法：

RRC A

操作：

PC ← PC+1

{ACC,CY} ← {CY,ACC}

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

说明：

累加器(ACC)的值带进位标志位(CY)循环右移，即累加器(ACC)的最低有效位(LSB)移入进位标志位(CY),进位标志位(CY)的旧值移入累加器(ACC)的最高有效位(MSB),其余位向右移动一位。

示例：

RRC A

指令执行前，A = 0x81，CY = 1

指令执行后，A = 0xC0，CY = 1

### 4.2.19 SWAP A

语法：

SWAP A

操作：

PC ← PC+1

ACC ← {ACC[3:0],ACC[7:4]}

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

说明：

累加器(ACC)的值高4位和低4位互换，结果还是给到累加器(ACC)。

示例：

SWAP A

指令执行前，A = 0x81

指令执行后，A = 0x18

### 4.2.20 XRL A, Rn

语法：

XRL A,Rn

操作：

PC ← PC+1

ACC ← ACC XOR Rn

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 1 | n2 | n1 | n0 |

说明：

累加器(ACC)的值与选定的工作寄存器(Rn-R0至R7)进行按位异或运算，结果赋给累加器(ACC)。

示例：

MOV R7,#0x11

MOV A,#0x14

XRL A,R7

指令执行前，A = 0x14，R7 = 0x11

指令执行后，A = 0x05，R7 = 0x11

### 4.2.21 XRL A, direct

语法：

XRL A,direct

操作：

PC ← PC+2

ACC ← ACC XOR (direct)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

读取可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内的值，并将其与累加器(ACC)的值进行按位异或操作，结果赋给累加器(ACC)。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

XRL A, Addr\_IRAM或XRL A, 0x30

指令执行前，A = 0x15，(Addr\_IRAM) = 0x11

指令执行后，A = 0x04，(Addr\_IRAM) = 0x11

### 4.2.22 XRL A, @Ri

语法：

XRL A, @Ri

操作：

PC ← PC+1

ACC ← ACC XOR (Ri)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值作为间接寻址的地址，读取该地址内的值与累加器A(ACC)的值进行按位异或操作，结果赋给累加器(ACC)。

示例：

MOV R0,#0x81 ;R0赋值0x81

XRL A,@R0

指令执行前，A = 0x14，(R0) = 0x11

指令执行后，A = 0x05，(R0) = 0x11

### 4.2.23 XRL A, #data

语法：

XRL A, #data

操作：

PC ← PC+2

ACC ← ACC XOR data

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

将8位立即数与累加器(ACC)进行按位异或操作，结果赋给累加器(ACC)。

示例：

XRL A,#0x25

指令执行前，A = 0xA0

指令执行后，A = 0x85

### 4.2.24 XRL direct, A

语法：

XRL direct,A

操作：

PC ← PC+2

(direct) ← (direct) XOR ACC

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)内的值，与累加器(ACC)的值进行按位异或操作，结果赋给“直接字节”指定的地址。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

XRL Addr\_IRAM,A或XRL 0x30,A

指令执行前，A = 0x14，(Addr\_IRAM) = 0x15

指令执行后，A = 0x14，(Addr\_IRAM) = 0x01

### 4.2.25 XRL direct, #data

语法：

XRL direct,#data

操作：

PC ← PC+3

(direct) ← (direct) XOR data

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内的值，与指令提供的8位立即数进行按位异或操作，结果赋给直接寻址地址。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

XRL Addr\_IRAM,#0x11 或XRL 0x30, #0x11

指令执行前，(Addr\_IRAM) = 0x14

指令执行后，(Addr\_IRAM) = 0x05

## 4.3 数据传送指令详解

### 4.3.1 MOV A, Rn

语法：

MOV A,Rn

操作：

PC ← PC+1

ACC ← Rn

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 1 | n2 | n1 | n0 |

说明：

将选定的工作寄存器(Rn-R0至R7)内的值加载到累加器(ACC)。

示例：

MOV R7,#0x11

MOV A,R7

指令执行前，A = 0x15，R7 = 0x11

指令执行后，A = 0x11，R7 = 0x11

### 4.3.2 MOV A, direct

语法：

MOV A,direct

操作：

PC ← PC+2

ACC ← (direct)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

读取可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内的值加载到累加器(ACC)。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

MOV A, Addr\_IRAM或MOV A, 0x30

指令执行前，A = 0x15，(Addr\_IRAM) = 0x11

指令执行后，A = 0x11，(Addr\_IRAM) = 0x11

### 4.3.3 MOV A, @Ri

语法：

MOV A, @Ri

操作：

PC ← PC+2

ACC ← (Ri)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值作为间接寻址的地址，读取该地址内的值加载到累加器(ACC)。

示例：

MOV R0,#0x81 ;R0赋值0x81

MOV A,@R0

指令执行前，A = 0x14，(R0) = 0x11

指令执行后，A = 0x11，(R0) = 0x11

### 4.3.4 MOV A, #data

语法：

MOV A, #data

操作：

PC ← PC+2

ACC ← data

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

将8位立即数加载到累加器(ACC)。

示例：

MOV A,#0x25

指令执行前，A = 0xA0

指令执行后，A = 0x25

### 4.3.5 MOV Rn, A

语法：

MOV Rn,A

操作：

PC ← PC+1

Rn ← ACC

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 1 | n2 | n1 | n0 |

说明：

将累加器(ACC)内的值加载到选定的工作寄存器(Rn-R0至R7)内。

示例：

MOV R7,A

指令执行前，A = 0x15

指令执行后，A = 0x15，R7 = 0x15

### 4.3.6 MOV Rn, direct

语法：

MOV Rn,direct

操作：

PC ← PC+2

Rn ← (direct)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 1 | n2 | n1 | n0 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

读取可直接寻址访问定的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内的值，加载到选定的工作寄存器(Rn-R0至R7)内。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

MOV R7, Addr\_IRAM或MOV R7, 0x30

指令执行前，(Addr\_IRAM) = 0x11

指令执行后，(Addr\_IRAM) = 0x11，R7 = 0x11

### 4.3.7 MOV Rn, #data

语法：

MOV Rn,#data

操作：

PC ← PC+2

Rn ← data

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 1 | n2 | n1 | n0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

将8位立即数加载到选定的工作寄存器(Rn-R0至R7)内。

示例：

MOV R7, #0x11

指令执行后，R7 = 0x11

### 4.3.8 MOV direct, A

语法：

MOV direct,A

操作：

PC ← PC+2

(direct) ← ACC

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

将累加器(ACC)内的值加载到可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

MOV Addr\_IRAM, A

指令执行前，A = 0xA0

指令执行后，A = 0xA0，(Addr\_IRAM) = 0xA0

### 4.3.9 MOV direct, Rn

语法：

MOV direct,Rn

操作：

PC ← PC+2

(direct) ← Rn

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 1 | n2 | n1 | n0 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

读取选定的工作寄存器(Rn-R0至R7)内的值加载到可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

MOV Addr\_IRAM, R7或MOV 0x30, R7

指令执行前，R7 = 0x11

指令执行后，R7 = 0x11，(Addr\_IRAM) = 0x11

### 4.3.10 MOV direct1, direct2

语法：

MOV direct1, direct2

操作：

PC ← PC+3

(direct1) ← (direct2)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| a27 | a26 | a25 | a24 | a23 | a22 | a21 | a20 |
| a17 | a16 | a15 | a14 | a13 | a12 | a11 | a10 |

说明：

将一个可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内的值加载到另外一个可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内。

示例：

Addr\_IRAM1 .EQU 0x30 ;内部RAM(IRAM)直接字节地址1

Addr\_IRAM2 .EQU 0x31 ;内部RAM(IRAM)直接字节地址2

MOV Addr\_IRAM1, Addr\_IRAM2或MOV 0x30, 0x31

指令执行前，(Addr\_IRAM2) = 0x11

指令执行后，(Addr\_IRAM2) = 0x11，(Addr\_IRAM1) = 0x11

### 4.3.11 MOV direct, @Ri

语法：

MOV direct, @Ri

操作：

PC ← PC+2

(direct) ← (Ri)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值作为间接寻址的地址，读取该地址内的值加载到可直接选址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

MOV R0,#0x31 ;R0赋值0x31

MOV Addr\_IRAM,@R0

指令执行前，(Addr\_IRAM) = 0x14，(R0) = 0x11

指令执行后，(Addr\_IRAM) = 0x11，(R0) = 0x11

### 4.3.12 MOV direct, #data

语法：

MOV direct, #data

操作：

PC ← PC+3

(direct) ← data

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

将8位立即数加载到可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

MOV Addr\_IRAM,#0x11

指令执行前，(Addr\_IRAM) = 0x14

指令执行后，(Addr\_IRAM) = 0x11

### 4.3.13 MOV @Ri, A

语法：

MOV @Ri, A

操作：

PC ← PC+1

(Ri) ← ACC

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值作为间接寻址的地址，将累加器(ACC)内的值加载到该地址内。

示例：

MOV @R0,A

指令执行前，A = 0x11，(R0) = 0x14

指令执行后，A = 0x11，(R0) = 0x11

### 4.3.14 MOV @Ri, direct

语法：

MOV @Ri, direct

操作：

PC ← PC+2

(Ri) ← (direct)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | i |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

选定寄存器(Ri-R0或R1)内的值作为间接寻址的地址，将由“直接字节”指定的内部RAM(IRAM)或特殊功能寄存器(SFR)内的值加载到该地址内。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

MOV @R0, Addr\_IRAM或MOV @R0, 0x30

指令执行前，(Addr\_IRAM) = 0x11

指令执行后，(Addr\_IRAM) = 0x11，(R0) = 0x11

### 4.3.15 MOV @Ri, #data

语法：

MOV @Ri, #data

操作：

PC ← PC+2

(Ri) ← data

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | i |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

选定寄存器(Ri-R0或R1)内的值作为间接寻址的地址，将8位立即数加载到该地址内。

示例：

MOV @R0, #0x11

指令执行前，data = 0x11

指令执行后，(R0) = 0x11

### 4.3.16 MOV DPTR, #data16

语法：

MOV DPTR, #data16

操作：

PC ← PC+3

DPTR ← data16

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| d15 | d14 | d13 | d12 | d11 | d10 | d9 | d8 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |

说明：

将16位立即数加载到数据指针寄存器(DPTR)。

示例：

MOV DPTR, #0x8030

指令执行后，DPL = 0x30，DPH = 0x80

### 4.3.17 MOVC A, @A+DPTR

语法：

MOVC A, @A+DPTR

操作：

PC ← PC+1

ACC ← Code(DPTR+unsigned(ACC))

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

说明：

将数据指针寄存器(DPTR)内的值加上累加器(ACC)内的值，将该值作为间接寻址的地址，读取该地址内的代码，结果赋给累加器(ACC)。

示例：

假设Code地址0x1031地址内的代码为0xC2

MOV DPTR, #0x1030

MOV A,#0x01

MOVC A,@A+DPTR

指令执行前，A = 0x01，DPTR = 0x1030

指令执行后，A = 0xC2，DPTR = 0x1030

### 4.3.18 MOVC A, @A+PC

语法：

MOVC A, @A+PC

操作：

PC ← PC+1

ACC ← Code(PC+unsigned(ACC))

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |

说明：

PC先增加1，然后将程序计数器(PC)内的值加上累加器(ACC)内的值，将该值作为间接寻址的地址，读取该地址内的代码，结果赋给累加器(ACC)。

示例：

假设Code地址0x1031地址内的代码为0xC2

MOV A,#0x00

MOVC A,@A+PC ;假设此时PC=0x1030

指令执行前，A = 0x00，PC = 0x1030

指令执行后，A = 0xC2，PC = 0x1031

### 4.3.19 MOVX A, @Ri

语法：

MOVX A, @Ri

操作：

PC ← PC+1

ACC ← Xdata({P2,Ri})

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值与P2寄存器内的值，构成的16位地址作为间接寻址的地址，读取外部数据存储器该地址内的值加载到累加器(ACC)。

示例：

假设Xdata地址0x1030地址内的值为0x82

MOVX A,@R0

指令执行前，P2 = 0x10，R0 = 0x30

指令执行后，A = 0x82，P2 = 0x10，R0 = 0x30

### 4.3.20 MOVX A, @DPTR

语法：

MOVX A, @DPTR

操作：

PC ← PC+1

ACC ← Xdata(DPTR)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

说明：

将数据指针寄存器(DPTR)内的值作为间接寻址的地址，读取外部数据存储区该地址内的值加载到累加器(ACC)。

示例：

假设Xdata地址0x1030地址内的值为0x82

MOVX A,@DPTR

指令执行前，DPTR = 0x1030

指令执行后，DPTR = 0x1030，A = 0x82

### 4.3.21 MOVX @Ri, A

语法：

MOVX @Ri, A

操作：

PC ← PC+1

Xdata({P2,Ri}) ← ACC或Code({P2,Ri}) ← ACC

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值与P2寄存器内的值，构成的16位地址作为间接寻址的地址，将累加器(ACC)内的值加载到该地址内。如果PCON.4位清零，操作的地址为外部数据存储区(Xdata)。如果PCON.4位置1，操作的地址为代码区(Code)。

示例：

MOVX @R0,A

指令执行前，A = 0x11

指令执行后，A = 0x11，({P2,R0}) = 0x11

### 4.3.22 MOVX @DPTR, A

语法：

MOVX @DPTR, A

操作：

PC ← PC+1

Xdata(DPTR) ← ACC或Code(DPTR) ← ACC

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

说明：

将数据指针寄存器(DPTR)内的16位值作为间接寻址的地址，将累加器(ACC)内的值加载到该地址内。如果PCON.4位清零，操作的地址为外部数据存储区(Xdata)。如果PCON.4位置1，操作的地址为代码区(Code)。

示例：

MOVX @DPTR,A

指令执行前，A = 0x11

指令执行后，A = 0x11，(DPTR) = 0x11

### 4.3.23 PUSH direct

语法：

PUSH direct

操作：

PC ← PC+2

SP ← SP + 1

(SP) ← (direct)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

从可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)中读取一个8位数据，且将这个数据写入由当前堆栈指针(SP)指向的IRAM地址中。

示例：

PUSH A

指令执行前，A = 0x11

指令执行后，A = 0x11，(SP) = 0x11

### 4.3.24 POP direct

语法：

POP direct

操作：

PC ← PC+2

(direct) ← (SP)

SP ← SP - 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

从当前堆栈指针(SP)指向的IRAM地址中读取一个8位数据，并将数据加载到可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)中。

示例：

POP A

指令执行前，(SP) = 0x11

指令执行后，A = 0x11

### 4.3.25 XCH A, Rn

语法：

XCH A,Rn

操作：

PC ← PC+1

ACC ← Rn

Rn ← (previous)ACC

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 1 | n2 | n1 | n0 |

说明：

读取选定的工作寄存器(Rn-R0至R7)内的值，并将值写入累加器(ACC)，然后将累加器(ACC)之前的值写入选定的工作寄存器(Rn-R0至R7)内，实现数据互换。

示例：

XCH A,R7

指令执行前，A = 0x11，R7 = 0x15

指令执行后，A = 0x15，R7 = 0x11

### 4.3.26 XCH A, direct

语法：

XCH A,direct

操作：

PC ← PC+2

ACC ← (direct)

(direct) ← (previous)ACC

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

读取可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内的值，并将值写入累加器(ACC)，然后将累加器(ACC)之前的值写入可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)地址内，实现数据互换。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

XCH A, Addr\_IRAM或XCH A, 0x30

指令执行前，A = 0x11，(Addr\_IRAM) = 0x15

指令执行后，A = 0x15，(Addr\_IRAM) = 0x11

### 4.3.27 XCH A, @Ri

语法：

XCH A, @Ri

操作：

PC ← PC+1

ACC ← (Ri)

(Ri) ← (previous)ACC

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值为间接寻址的地址，读取内部RAM(IRAM)该地址内的值加载到累加器(ACC)。然后选定寄存器(Ri-R0或R1)内的值为间接寻址的地址，将累加器(ACC)之前的值，写入内部RAM(IRAM)该地址内，实现数据互换。

示例：

XCH A,@R0

指令执行前，A = 0x11，(R0) = 0x15

指令执行后，A = 0x15，(R0) = 0x11

### 4.3.28 XCHD A, @Ri

语法：

XCHD A, @Ri

操作：

PC ← PC+1

ACC[3:0] ← (Ri)[3:0]

(Ri) ← {(Ri)[7:4],(previous)ACC[3:0]}

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | i |

说明：

选定寄存器(Ri-R0或R1)内的值为间接寻址的地址，读取内部RAM(IRAM)该地址内的值，且将低4位替换累加器(ACC)的低4位。然后选定寄存器(Ri-R0或R1)内的值为间接寻址的地址，将累加器(ACC)之前的值的低4位，替换内部RAM(IRAM)该地址内值的低4位，实现低4位数据互换。

示例：

XCHD A,@R0

指令执行前，A = 0x2A，(R0) = 0x15

指令执行后，A = 0x25，(R0) = 0x1A

## 4.4 布尔(位运算)指令详解

### 4.4.1 ANL C, bit

语法：

ANL C, bit

操作：

PC ← PC+2

CY ← CY AND (bit)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

说明：

根据位地址从内部RAM(IRAM)或特殊功能寄存器(SFR)中读取一个字节，然后使用bit地址中的[b2:b0](即最低三位)来定位该字节中的某一位，将当前进位标志位(CY)与该位进行逻辑与，结果赋给进位标志位(CY)。

示例：

ANL C,0x08 ;位地址0x08对应位寻址地址0x21的第0位

指令执行前，CY = 1，(0x21) = 0x00

指令执行后，CY = 0，(0x21) = 0x00

### 4.4.2 ANL C, /bit

语法：

ANL C, /bit

操作：

PC ← PC+2

CY ← CY AND NOT(bit)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

说明：

根据位地址从内部RAM(IRAM)或特殊功能寄存器(SFR)中读取一个字节，然后使用bit地址中的[b2:b0](即最低三位)来定位该字节中的某一位，将当前进位标志位(CY)与该位的反码进行逻辑与操作，结果取反后赋给进位标志位(CY)。

示例：

ANL C,/0x08 ;位地址0x08对应位寻址地址0x21的第0位

指令执行前，CY = 1，(0x21) = 0x01

指令执行后，CY = 0，(0x21) = 0x01

### 4.4.3 CLR C

语法：

CLR C

操作：

PC ← PC+1

CY ← 0

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |

说明：

将进位标志位(CY)清零。

示例：

CLR C

指令执行前，CY = 1

指令执行后，CY = 0

### 4.4.4 CLR bit

语法：

CLR bit

操作：

PC ← PC+2

if (bit < 0x80)

byte = (0x20+(bit>>3))//IRAM 地址范围 0x20至0x2F

byte[bit[2:0]] = 0

(0x20+(bit>>3)) = byte

else

byte = (bit & 0xF8)//特殊功能寄存器(SFR)地址低三位为0

byte[bit[2:0]] = 0

(bit & 0xF8) = byte

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

说明：

根据位地址从内部RAM(IRAM)或特殊功能寄存器(SFR)中读取一个字节，其中[b2:b0]用于从所读取的字节中选择一位清零，然后整个字节写回内部RAM(IRAM)或特殊功能寄存器(SFR)中。

示例：

CLR 0x08 ;位地址0x08对应位寻址地址0x21的第0位

指令执行前，(0x21) = 0x01

指令执行后，(0x21) = 0x00

### 4.4.5 CPL C

语法：

CPL C

操作：

PC ← PC+1

CY ← !CY

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |

说明：

将进位标志位(CY)取反。

示例：

CPL C

指令执行前，CY = 0

指令执行后，CY = 1

### 4.4.6 CPL bit

语法：

CLR bit

操作：

PC ← PC+2

if (bit < 0x80)

byte = (0x20+(bit>>3))//IRAM 地址范围 0x20至0x2F

byte[bit[2:0]] = !byte[bit[2:0]]

(0x20+(bit>>3)) = byte

else

byte = (bit & 0xF8)//特殊功能寄存器(SFR)地址低三位为0

byte[bit[2:0]] = !byte[bit[2:0]]

(bit & 0xF8) = byte

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

说明：

根据位地址从内部RAM(IRAM)或特殊功能寄存器(SFR)中读取一个字节，其中[b2:b0]用于从所读取的字节中选择一位取反，然后整个字节写回内部RAM(IRAM)或特殊功能寄存器(SFR)中。

示例：

CPL 0x08 ;位地址0x08对应位寻址地址0x21的第0位

指令执行前，(0x21) = 0x00

指令执行后，(0x21) = 0x01

### 4.4.7 MOV C, bit

语法：

MOV C,bit

操作：

PC ← PC+2

if (bit < 0x80)

byte = (0x20+(bit>>3))//IRAM 地址范围 0x20至0x2F

else

byte = (bit & 0xF8)//特殊功能寄存器(SFR)地址低三位为0

CY = byte[bit[2:0]]

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

说明：

根据位地址从内部RAM(IRAM)或特殊功能寄存器(SFR)中读取一个字节，其中[b2:b0]用于从所读取的字节中选择一位，将其赋值给进位标志位(CY)。

示例：

MOV C ,0x08 ;位地址0x08对应位寻址地址0x21的第0位

指令执行前，CY = 0，(0x21) = 0x01

指令执行后，CY = 1，(0x21) = 0x01

### 4.4.8 MOV bit, C

语法：

MOV bit,C

操作：

if (bit < 0x80)

byte = (0x20+(bit>>3))//IRAM 地址范围 0x20至0x2F

byte[bit[2:0]] = CY

(0x20+(bit>>3)) = byte

else

byte = (bit & 0xF8)//特殊功能寄存器(SFR)地址低三位为0

byte[bit[2:0]] = CY

(bit & 0xF8) = byte

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

说明：

根据位地址从内部RAM(IRAM)或特殊功能寄存器(SFR)中读取一个字节，其中[b2:b0]用于从所读取的字节中选择一位，并且将进位标志位(CY)赋给该位，然后整个字节写回内部RAM(IRAM)或特殊功能寄存器(SFR)中。

示例：

MOV 0x08,C ;位地址0x08对应位寻址地址0x21的第0位

指令执行前，CY = 1，(0x21) = 0x00

指令执行后，CY = 1，(0x21) = 0x01

### 4.4.9 ORL C, bit

语法：

ORL C, bit

操作：

PC ← PC+2

CY ← CY OR (bit)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

说明：

根据位地址从内部RAM(IRAM)或特殊功能寄存器(SFR)中读取一个字节，然后使用bit地址中的[b2:b0](即最低三位)来定位该字节中的某一位，将当前进位标志位(CY)与该位进行逻辑或，结果赋给进位标志位(CY)。

示例：

ORL C,0x08 ;位地址0x08对应位寻址地址0x21的第0位

指令执行前，CY = 0，(0x21) = 0x01

指令执行后，CY = 1，(0x21) = 0x01

### 4.4.10 ORL C, /bit

语法：

ORL C, /bit

操作：

PC ← PC+2

CY ← CY OR NOT(bit)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

说明：

根据位地址从内部RAM(IRAM)或特殊功能寄存器(SFR)中读取一个字节，然后使用bit地址中的[b2:b0](即最低三位)来定位该字节中的某一位，将当前进位标志位(CY)与该位进行逻辑或，结果取反后赋给进位标志位(CY)。

示例：

ORL C,/0x08 ;位地址0x08对应位寻址地址0x21的第0位

指令执行前，CY = 0，(0x21) = 0x00

指令执行后，CY = 1，(0x21) = 0x00

### 4.4.11 SETB C

语法：

SETB C

操作：

PC ← PC+1

CY ← 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |

说明：

进位标志位(CY)置1。

示例：

SETB C

指令执行前，CY = 0

指令执行后，CY = 1

### 4.4.12 SETB bit

语法：

SETB bit

操作：

PC ← PC+2

if (bit < 0x80)

byte = (0x20+(bit>>3))//IRAM 地址范围 0x20至0x2F

byte[bit[2:0]] = 1

(0x20+(bit>>3)) = byte

else

byte = (bit & 0xF8)//特殊功能寄存器(SFR)地址低三位为0

byte[bit[2:0]] = 1

(bit & 0xF8) = byte

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |

说明：

根据位地址从内部RAM(IRAM)或特殊功能寄存器(SFR)中读取一个字节，然后使用bit地址中的[b2:b0](即最低三位)来定位该字节中的某一位，将该位置1后，写回内部RAM(IRAM)或特殊功能寄存器(SFR)中。

示例：

SETB 0x08 ;位地址0x08对应位寻址地址0x21的第0位

指令执行前，(0x21) = 0x00

指令执行后，(0x21) = 0x01

## 4.5 程序分支指令详解

### 4.5.1 ACALL addr11

语法：

ACALL addr11

操作：

PC ← PC+2

SP ← SP + 1

(SP) ← PC[7:0]

SP ← SP + 1

(SP) ← PC[15:8]

PC[10:0] ← addr11

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a10 | a9 | a8 | 1 | 0 | 0 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

页内绝对调用指令，程序计数器(PC)先递增2，然后将递增后的置存储到堆栈(由SP指向的内部RAM)中，随后将11位立即寻址地址加载到程序计数器，并清除预取缓冲器。最大访问页内2kB地址。

示例：

ACALL addr11 ; addr11范围页内2kB，例如：0x0000至0x07FF

### 4.5.2 AJMP addr11

语法：

AJMP addr11

操作：

PC ← PC+2

PC[10:0] ← addr11

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| a10 | a9 | a8 | 0 | 0 | 0 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

页内绝对跳转指令，程序计数器(PC)先递增2，然后将11位立即寻址地址加载到程序计数器，并清除预取缓冲器。最大访问页内2kB地址。

示例：

AJMP addr11 ; addr11范围页内2kB，例如：0x0000至0x07FF

### 4.5.3 CJNE A, direct, rel

语法：

CJNE A,direct,rel

操作：

signed(rel) = signed(rel) - PC - 3

PC ← PC+3

if (ACC!=(direct))

PC ← PC + signed(rel)

if (ACC < (direct))

CY ← 1

else

CY ← 0

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

读取“直接字节”指定的内部RAM(IRAM)或特殊功能寄存器(SFR)的值，并且与累加器(ACC)内的值相比较。如果不相等，则跳转到rel指定的相对地址执行，否则顺序执行。如果累加器(ACC)的值小于直接寻址数据，则进位标志位(CY)置1，否则进位标志位(CY)清零，rel相对地址范围-128至127。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

CJNE A, Addr\_IRAM,loop 或 CJNE A, 0x30,loop

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x12

指令执行前，A = 0xAA，(Addr\_IRAM) = 0xA5，CY = 1，PC = 0x0200

指令执行后，A = 0xAA，(Addr\_IRAM) = 0xA5，CY = 0，PC = 0x0215

### 4.5.4 CJNE A, #data, rel

语法：

CJNE A,#data,rel

操作：

signed(rel) = signed(rel) - PC - 3

PC ← PC+3

if (ACC!=data)

PC ← PC + signed(rel)

if (ACC < data)

CY ← 1

else

CY ← 0

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

累加器(ACC)内的值与8位立即数相比较。如果不相等，则跳转到rel指定的相对地址执行。如果累加器(ACC)的值小于立即数，则进位标志位(CY)置1，否则进位标志位(CY)清零，rel相对地址范围-128至127。

示例：

CJNE A, #0xA5,loop

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x12

指令执行前，A = 0xAA，CY = 1，PC = 0x0200

指令执行后，A = 0xAA，CY = 0，PC = 0x0215

### 4.5.5 CJNE Rn, #data, rel

语法：

CJNE Rn,#data,rel

操作：

signed(rel) = signed(rel) - PC - 3

PC ← PC+3

if (Rn != data)

PC ← PC + signed(rel)

if (Rn < data)

CY ← 1

else

CY ← 0

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 1 | n2 | n1 | n0 |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

选定工作寄存器(Rn-R0至R7)内的值与8位立即数相比较。如果不相等，则跳转到rel指定的相对地址执行。如果选定的工作寄存器(Rn-R0至R7)内的值小于立即数，则进位标志位(CY)置1，否则进位标志位(CY)清零，rel相对地址范围-128至127。

示例：

CJNE R7, #0xA5,loop

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x12

指令执行前，R7 = 0xAA，CY = 1，PC = 0x00200

指令执行后，R7 = 0xAA，CY = 0，PC = 0x00215

### 4.5.6 CJNE @Ri, #data, rel

语法：

CJNE @Ri,#data,rel

操作：

signed(rel) = signed(rel) - PC - 3

PC ← PC+3

if ((Ri) != data)

PC ← PC + signed(rel)

if ((Ri) < data)

CY ← 1

else

CY ← 0

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | i |
| d7 | d6 | d5 | d4 | d3 | d2 | d1 | d0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

将选定寄存器(Ri-R0至R1)内的值作为间接寻址的地址，读取内部RAM(IRAM)该地址内的值与8位立即数相比较。如果不相等，则跳转到rel指定的相对地址执行。如果地址内的值小于立即数，则进位标志位(CY)置1，否则进位标志位(CY)清零，rel相对地址范围-128至127。

示例：

CJNE @R0, #0xA5,loop

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x12

指令执行前，(R0) = 0xAA，CY = 1，PC = 0x0200

指令执行后，(R0) = 0xAA，CY = 0，PC = 0x0215

### 4.5.7 DJNZ Rn, rel

语法：

DJNZ Rn ,rel

操作：

signed(rel) = signed(rel) - PC - 2

PC ← PC+2

Rn ← Rn - 1

if (Rn != 0)

PC ← PC + signed(rel)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 1 | n2 | n1 | n0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

选定工作寄存器(Rn-R0至R7)内的值减1，结果不等于0，则跳转到rel指定的相对地址执行，且此操作不影响任何状态标志位，rel相对地址范围-128至127。

示例：

DJNZ R7, loop

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x13

指令执行前，R7 = 0x02，PC = 0x0200

指令执行后，R7 = 0x01，PC = 0x0215

### 4.5.8 DJNZ direct, rel

语法：

DJNZ direct ,rel

操作：

signed(rel) = signed(rel) - PC - 3

PC ← PC+3

(direct) ← (direct) - 1

if ((direct) != 0)

PC ← PC + signed(rel)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

将可直接寻址访问的内部RAM(IRAM)或特殊功能寄存器(SFR)内的值减1，结果不等于0，则跳转到rel指定的相对地址执行，且此操作不影响任何状态标志位，rel相对地址范围-128至127。

示例：

Addr\_IRAM .EQU 0x30 ;内部RAM(IRAM)直接字节地址

DJNZ Addr\_IRAM, loop 或 DJNZ 0x30, loop

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x12

指令执行前，(Addr\_IRAM) = 0x02，PC = 0x0200

指令执行后，(Addr\_IRAM) = 0x01，PC = 0x0215

### 4.5.9 JB bit, rel

语法：

JB bit,rel

操作：

signed(rel) = signed(rel) - PC - 3

PC ← PC+3

if (bit < 0x80)

byte = (0x20+(bit>>3))//IRAM 地址范围 0x20至0x2F

else

byte = (bit & 0xF8)//特殊功能寄存器(SFR)地址低三位为0

if (byte[bit[2:0]] == 1)

PC ← PC + signed(rel)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

根据位地址读取内部RAM(IRAM)或特殊功能寄存器(SFR)的字节数据，其中[b2:b0]用于从所读取的字节中选择一位，如果该位等于1，则跳转到rel指定的相对地址执行，且此操作不影响任何状态标志位，rel相对地址范围-128至127。

示例：

JB 0x08,loop ;位地址0x08对应位寻址地址0x21的第0位

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x12

指令执行前，(0x21) = 0x01，PC = 0x0200

指令执行后，(0x21) = 0x01，PC = 0x0215

### 4.5.10 JBC bit, rel

语法：

JBC bit,rel

操作：

signed(rel) = signed(rel) - PC - 3

PC ← PC+3

if (bit < 0x80)

byte = (0x20+(bit>>3))//IRAM 地址范围 0x20至0x2F

else

byte = (bit & 0xF8)//特殊功能寄存器(SFR)地址低三位为0

if (byte[bit[2:0]] == 1)

PC ← PC + signed(rel)

byte[bit[2:0]] = 0

if (bit < 0x80)

(0x20+(bit>>3)) = byte

else

(bit & 0xf8) = byte

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

根据位地址读取内部RAM(IRAM)或特殊功能寄存器(SFR)的字节数据，其中[b2:b0]用于从所读取的字节中选择一位，如果该位等于1，则跳转到rel指定的相对地址执行。然后清除该位，以字节的方式写回内部RAM(IRAM)或特殊功能寄存器(SFR)，rel相对地址范围-128至127。

示例：

JBC 0x08,0x10 ;位地址0x08对应位寻址地址0x21的第0位

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x12

指令执行前，(0x21) = 0x01，PC = 0x0200

指令执行后，(0x21) = 0x00，PC = 0x0215

### 4.5.11 JC rel

语法：

JC rel

操作：

signed(rel) = signed(rel) - PC - 2

PC ← PC+2

if (CY == 1)

PC ← PC + signed(rel)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

如果进位标志位(CY)置1，则跳转到rel指定的相对地址执行，rel相对地址范围-128至127。

示例：

JC loop

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x13

指令执行前，CY = 1，PC = 0x0200

指令执行后，CY = 1，PC = 0x0215

### 4.5.12 JZ rel

语法：

JZ rel

操作：

signed(rel) = signed(rel) - PC - 2

PC ← PC+2

if (ACC == 0)

PC ← PC + signed(rel)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

如果累加器(ACC)为0，则跳转到rel指定的相对地址执行，rel相对地址范围-128至127。

示例：

JZ loop

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x13

指令执行前，ACC = 0x00，PC = 0x0200

指令执行后，ACC = 0x00，PC = 0x0215

### 4.5.13 JMP @A+DPTR

语法：

JMP @A+DPTR

操作：

PC ← DPTR + unsigned(ACC)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |

说明：

使用数据指针(DPTR)内的值加累加器(ACC)内无符号的值作为地址修改程序计数器(PC)，并清空预取缓冲器。

示例：

JMP @A+DPTR

指令执行前，ACC = 0x01，DPTR = 0x0112

指令执行后，ACC = 0x01，DPTR = 0x0112，PC = 0x0113

### 4.5.14 JNC rel

语法：

JNC rel

操作：

signed(rel) = signed(rel) - PC - 2

PC ← PC+2

if (CY == 0)

PC ← PC + signed(rel)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

如果进位标志位(CY)清零，则跳转到rel指定的相对地址执行，rel相对地址范围-128至127。

示例：

JNC loop

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x13

指令执行前，CY = 0，PC = 0x0200

指令执行后，CY = 0，PC = 0x0215

### 4.5.15 JNB bit, rel

语法：

JNB bit,rel

操作：

signed(rel) = signed(rel) - PC - 3

PC ← PC+3

if (bit < 0x80)

byte = (0x20+(bit>>3))//IRAM 地址范围 0x20至0x2F

else

byte = (bit & 0xF8)//特殊功能寄存器(SFR)地址低三位为0

if (byte[bit[2:0]] ==0)

PC ← PC + signed(rel)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| b7 | b6 | b5 | b4 | b3 | b2 | b1 | b0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

根据位地址读取内部RAM(IRAM)或特殊功能寄存器(SFR)的字节数据，其中[b2:b0]用于从所读取的字节中选择一位，如果该位等于0，则跳转到rel指定的相对地址执行，且此操作不影响任何状态标志位，rel相对地址范围-128至127。

示例：

JNB 0x08,0x10 ;位地址0x08对应位寻址地址0x21的第0位

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x12

指令执行前，(0x21) = 0x00，PC = 0x0200

指令执行后，(0x21) = 0x00，PC = 0x0215

### 4.5.16 JNZ rel

语法：

JNZ rel

操作：

signed(rel) = signed(rel) - PC - 2

PC ← PC+2

if (ACC != 0)

PC ← PC + signed(rel)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

如果累加器(ACC)不等于0，则跳转到rel指定的相对地址执行，rel相对地址范围-128至127。

示例：

JNZ loop

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x13

指令执行前，ACC = 0x01，PC = 0x0200

指令执行后，ACC = 0x01，PC = 0x0215

### 4.5.17 LCALL addr16

语法：

LCALL addr16

操作：

PC ← PC+3

SP ← SP + 1

(SP) ← PC[7:0]

SP ← SP + 1

(SP) ← PC[15:8]

PC ← addr16

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| a15 | a14 | a13 | a12 | a11 | a10 | a9 | a8 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

长跳转指令，程序计数器(PC)先递增3，然后将递增后的值存储到堆栈(由SP指向的内部RAM)中，随后将16位立即数地址加载到程序计数器，并清除预取缓冲器。最大访问64kB地址。

示例：

LCALL Init\_Test

假设Init\_Test地址为0x0215，当前PC地址为0x0200

指令执行前，PC = 0x0200

指令执行后，PC = 0x0215

### 4.5.18 LJMP addr16

语法：

LJMP addr16

操作：

PC ← addr16

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| a15 | a14 | a13 | a12 | a11 | a10 | a9 | a8 |
| a7 | a6 | a5 | a4 | a3 | a2 | a1 | a0 |

说明：

长跳转指令，将16位立即数地址加载到程序计数器，并清除预取缓冲器。最大访问64kB地址。

示例：

LJMP loop

假设loop地址为0x0215，当前PC地址为0x0200

指令执行前，PC = 0x0200

指令执行后，PC = 0x0215

### 4.5.19 RET

语法：

RET

操作：

PC ← PC + 1

PC[15:8] ← (SP)

SP ← SP – 1

PC[7:0] ← (SP)

SP ← SP – 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |

说明：

返回指令，从子程序返回。

示例：

RET

### 4.5.20 RETI

语法：

RETI

操作：

PC ← PC + 1

PC[15:8] ← (SP)

SP ← SP – 1

PC[7:0] ← (SP)

SP ← SP – 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |

说明：

返回指令，从中断中返回，程序计数器(PC)从堆栈中弹出，高8位先弹出，低8位后弹出，预取缓冲区被清空。中断控制寄存器中的“服务中”寄存器会清除当前正在执行的中断服务例程的优先级，以便同优先级或更低优先级的中断可再次响应，执行RETI后，通常会禁止一个指令周期内的中断。

此命令主要功能是恢复中断前的程序计数器(PC)值，是程序从断点处继续执行，同时清除中断优先级触发器，允许新的中断请求被响应。它不会自动保存或恢复寄存器A和DPTR的内容，另外如果中断程序中使用改变程序状态字(PSW)内的标志位时，需要进入中断前先缓存累加器(ACC)、数据指针寄存器(DPTR)、程序状态字(PSW)，中断返回前恢复。

示例：

RETI

### 4.5.21 SJMP rel

语法：

SJMP rel

操作：

signed(rel) = signed(rel) - PC - 2

PC ← PC + 2

PC ← PC + signed(rel)

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| r7 | r6 | r5 | r4 | r3 | r2 | r1 | r0 |

说明：

短跳转指令，程序计数器(PC)先递增2，则跳转到rel指定的相对地址执行，rel相对地址范围-128至127。

示例：

SJMP loop

假设loop地址为0x0215，当前PC地址为0x0200，那么rel就是0x13

指令执行前，PC = 0x0200

指令执行后，PC = 0x0215

### 4.5.22 NOP

语法：

NOP

操作：

PC ← PC + 1

指令格式：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

说明：

无操作指令。

示例：

NOP

# 5.内部数据存储器

内部数据存储器是8051唯一可分配堆栈的位置，因此其容量必须足够容纳软件所需的堆栈大小。

下图是内部数据存储器地址分布图。



工作寄存器区(0x00~0x1F)，分为4个寄存器组(bank0~3)，每组8个寄存器(R0~R7)。通过程序状态字寄存器(PSW)选择当前工作寄存器组。

位寻址区(0x20~0x2F，16字节，128位)，每个字节对应一个可单独寻址的位地址(位地址0x00~0x7F,0x00对应0x20的位0，0x7F对应0x2F的位7)，支持位操作指令(如SETB、CLR)，适合布尔逻辑运算(如标志位控制)

通用RAM区(0x30~0x7F,共128字节)，可通过直接寻址(如MOV A,0x30)或间接寻址(如MOV A,@R0，以R0的内容作为地址，如R0赋值0x30，通过@R0就是间接寻址访问0x30地址)，用于存储临时变量、数据缓冲区等。

通用RAM区(0x80~0xFF，共128字节)，间接寻址访问时作为通用RAM功能，直接寻址访问时作为特殊功能寄存器(SFR)。

# 6.程序状态字寄存器(PSW)

程序状态字(PSW)寄存器包含多个与算术逻辑单元(ALU)相关的标志位，如进位标志位、溢出标志位、辅助进位标志位和奇偶标志位。

此外，还有一个8位堆栈指针(SP)寄存器，在执行调用、返回、压栈或弹栈指令时使用。堆栈指针的值始终指向内部数据数据存储器中栈顶的顶部。复位后，默认值为0x07，因此第一次压栈操作将发生在地址0x08(以免意外写入工作寄存器)，压栈前SP自增，弹栈后SP自减。

下表是PSW寄存器详情。

PSW寄存器(地址：0xD0，复位值：0x00)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| CY | AC | F0 | RS1 | RS0 | OV | F1 | P |

|  |  |  |  |
| --- | --- | --- | --- |
| 位号# | 名称 | 描述 | 类型 |
| 7 | CY | 进位标志  执行加法时第7位产生进位或减法时第7位产生借位时置位，否则清零。比较指令CJNE中，当第一操作数小于第二操作数时置位。乘法(MUL)和除法(DIV)会清除该标志。受循环移位指令(RLC、RRC)和位操作指令直接影响。 | 读/写 |
| 6 | AC | 辅助进位标志  加法时累加器第3位向第4位产生进位，或减法时第3位向第4位产生借位时置位，否则清零 | 读/写 |
| 5 | F0 | 标志0  供用户使用的通用标志位 | 读/写 |
| 4:3 | RS1:RS0 | 工作寄存器组选择  00：bank0(地址：0x00~0x07)  01：bank1(地址：0x08~0x0F)  10：bank2(地址：0x10~0x17)  11：bank3(地址：0x18~0x1F) | 读/写 |
| 2 | OV | 溢出标志  当加法运算使累加器第6位和第7位的进位不同时，或减法使累加器第6位和第7位产生借位时，该标志置位，否则清零。  OV标志表示有符号8位数的运算结果超出了范围(大于127或小于-128)。  当乘法运算结果大于255，或者尝试除以0时，溢出标志也会置位。 | 读/写 |
| 1 | F1 | 标志1  供用户使用的通用标志位，调试器下载时目前使用该位控制指令MOVX @DPTR,A和MOVX @Ri,A将累计器(ACC)数据写入XDATA区还是CODE区，置1时，写入CODE区，清零时写入XDATA区。 | 读/写 |
| 0 | P | 奇偶标志  始终存储累加器中所有位的模2求和结果(即奇偶校验位：偶校验为1，奇校验为0) | 只读 |

# 7. 汇编器简介

## 7.1 汇编器语法说明

* 汇编源文件后缀格式只能是“源文件名.asm”，否则编译器无法识别源文件；
* 汇编源文件不能直接放在工程目录下面，需要放在工程目录下创建的子目录内，否则编译器无法识别；
* 另外汇编源文件也不能放在子目录下的子目录内，否则编译器无法识别；
* 一个汇编源文件可以包含很多行；
* 一个指令代码必须在同一行中编写完毕；
* 汇编指令不能在每行的起始处编写，至少需要在行首留有一个空格符，用Tab键保留多个更佳；
* 每行可以包含一个程序跳转用的标号(必须顶格写)，标号只能包含‘a’~‘z’、‘A’~‘Z’的字母及‘0’~‘9’的数字和下划线‘\_’。标号后面需要跟冒号‘;’,否则会报错。
* 标号和指令代码可以单独成行，也可直接在标号‘:’后面直接写汇编指令；
* 一条汇编指令一般包含一个指令助记符和n个操作数，多个操作数之间用‘,’进行分隔；
* 汇编内的保留字(指令码或伪指令)其大小写一视同仁，但自定义的变量名或常量名区分大小写；
* 汇编文件中的注释用‘;’表示，该行‘;’号之后的内容汇编器不作处理；
* 程序中立即数的描述方法有以下几种：

1)、二进制：0B01010101;

2)、八进制：0o77或者0q77(对应十进制63)；

3)、十进制：85(无前缀)；

4)、十六进制：0x55；

5)、ASCII码：'A'(单引号表示，A=0x41(A的ASCII码))。

* 程序中允许使用的运算操作符有以下几种，按照优先级顺序从低往高排列，相同优先级的操作符，其计算顺序为从左到右。

|  |  |  |
| --- | --- | --- |
| 操作符 | 说明 | 优先级 |
| = | 赋值运算 | 0 |
| & | 位与 | 1 |
| | | 位或 |
| ^ | 位异或 |
| == | 等于 | 2 |
| != | 不等于 |
| << | 左移 | 3 |
| >> | 右移 |
| + | 加法 | 4 |
| - | 减法 |
| \* | 乘法 | 5 |
| / | 除法 |
| % | 取模 |
| - | 取负数 | 6 |
| ~ | 位非 |
| > | 取高字节 |
| < | 取低字节 |

## 7.2 伪指令

* .include：包含头文件和其它文件作用，该命令属于预处理命令，在整个汇编过程最先被处理，汇编器分析到“.include”命令时，会自动把文件的内容插到“.include”所在行的后面，例如:.include “SD82P153”；
* .equ：定义符号常量，将一个数值、表达式或字符串赋给一个符号名，后续可用符号名代替该值，例如COUNT .equ 10，之后可用COUNT代替数值10。另外还可简化代码，提高程序可读性，例如ADDR .equ 0x8000，用右意义的符号替代直接使用地址，同时自此表达式计算；
* .define：定义符号，配合条件汇编使用，例如：.define User\_Flag，不可使用表达式；
* .org：定义程序代码的绝对地址，一般用来程序开头地址，例：.org 0x0000，表示从地址0x0000开始存放代码，控制内存布局；
* .IF、.ELSE、.IFDEF、.IFNDEF、.ENDIF：条件汇编指令，条件汇编指令本身不会产生实际的代码，只有当条件成立时，该部分代码才会被处理。使用“.IFDEF、.ENDIF”判断符号是否定义，与“.define”配合使用是否编译此段程序。使用“.IF、.ELSE、.ENDIF”进行条件判断处理，不可使用表达式，只能判断“1”和“0”或常量内的值，非零即真；
* .area：定义和切换内存区域，指定代码、数据或未初始化数据应该放置在什么区域，格式：.area 区域名(区域属性)，例如：.area XSEG(ABS,XDATA)表示选择外部RAM区，且是绝对地址、.area PSEG (PAG)表示选择分页外部存储区XDATA、.area HOME (DATA)表示选择内部RAM区、.area CODE (ABS,CODE)表示选择CODE区，且是绝对地址。
* .DB、.DW、.DS：.DB字节定义指令，在程序存储区中定义一串字节单元，例如：.DB 0x80, 0x95, 0x74, 0x55，配合.org指令向指定的绝对地址写入字节数据。.DW字定义指令，在程序存储区中定义一串字单元，例如：.DW 0x8095, 0x7455，配合.org指令向指定的绝对地址写入字数据。.DS定义存储区并预留空间，保留DS之后表达式的值所规定的存储单元，例如：.DS 10，配合.org指令向指定的绝对地址开始预留10字节空间。

## 7.3 宏定义

汇编器中使用.macro/.endm伪指令来定义宏定义，基本的宏定义和使用的语法如下：

.macro \_clrf f1

mov dptr,#f1

mov a,#0

movx @dptr,a

.endm

对于宏的使用，可以简单的这样调用，完成相应的操作

\_clrf tmp ;清除tmp内容

# 9.修改记录

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 修改日期 | 作者 | 修改记录 |
| v0 | 2025-07-23 | 吴华桥 | 初始版本。 |