



# 51 单片机指令系统

51 单片机 CPU 的指令系统是其所能执行的全部指令的集合,一条指令完成一个独立的操作,多条指令组成可执行程序,完成一项或多项功能(任务)。本章介绍 51 单片机的 CPU 指令系统,内容包括指令的格式、指令助记符、伪指令、由指令组成的可执行应用程序等。

## 3.1 51 单片机 CPU 指令系统概述

51 单片机 CPU 共有 111 条指令,它们是由表明操作性质的指令助记符结合对操作数的不同寻址方式形成的,该指令系统具有多种操作功能,按其功能可以归纳为 4 类:数据传送类指令(29 条);算术、逻辑运算类指令(24 条+24 条);控制转移类指令(17 条);位操作、位控制转移类指令(17 条)。

### 3.1.1 指令的格式

51 单片机的 CPU 指令格式的統一表现形式为

操作码 + [操作数]

操作码和操作数构成指令机器码,它是 CPU 能够执行的指令,在 51 单片机系统中,指令机器码存放于 ROM 区域,其存放格式如图 3-1 所示。

51 单片机的 CPU 指令系统中有无操作数(操作码 A)、单操作数(操作码 B+操作数 B)、双操作数(操作码 C+操作数 C1+操作数 C2)3 种格式的指令,按指令占存储器字节数统计单字节指令有 49 条、双字节指令有 45 条、三字节指令有 17 条,按指令占用 CPU (执行)时间统计单机器周期指令有 64 条、双机器周期指令有 45 条、四机器周期指令有 2 条。

### 3.1.2 指令操作码助记符以及操作数表示符号

51 单片机 CPU 指令按功能划分为 4 类,其操作码所使用的助记符如表 3-1 所示,其指令格式中常用的操作数表示符号以及含义如表 3-2 所示。

ROM		
地址0	x x x x x x x x	操作码A
地址1	x x x x x x x x	操作码B
地址2	y y y y y y y y	操作数B
地址3	x x x x x x x x	操作码C
地址4	y y y y y y y y	操作数C1
地址5	z z z z z z z z	操作数C2

图 3-1 CPU 指令机器码在 ROM 中存放顺序

表 3-1 51 单片机的 CPU 指令操作码助记符

指令功能	助记符
数据传送类指令	MOV、MOVX、MOVC、PUSH、POP、HCX、XCHD、SWAP
算术、逻辑运算类指令	算术：ADD、ADDC、INC、DA、SUBB、DEC、MUL、DIV 逻辑：ANL、ORL、XRL、CPL、CLR、RL、RR、RLC、RRC
控制转移类指令	LJMP、AJMP、SJMP、LCALL、ACALL、RET、RETI、JZ、JNZ、CJNE、DJNZ、NOP
位操作、控制转移类指令	CLR、SETB、ANL、ORL、CPL、MOV、JC、JNC、JB、JNB、JBC

表 3-2 51 单片机的 CPU 指令中操作数表示符号及含义

表示符号	含 义
Rn	表示工作寄存器,n=0~7,4 组,由 PSW 中 RS1、RS0 确定组号
Ri	表示间址寄存器,i=0,1,只有 R <sub>0</sub> 和 R <sub>1</sub> 可作间址寄存器使用
# data	表示 8 位立即数,取值范围 00H~0FFH,代表常量,存放在 ROM 程序存储区
# data16	表示 16 位立即数,取值范围 0000H~0FFFFH,代表常量,存放在 ROM 程序存储区
direct	表示 8 位芯片内 RAM 直接地址,地址范围 256B,取值范围 00H~0FFH
addr16	表示 16 位芯片内、外存储器直接地址,地址范围 64KB,A0~A15
addr11	表示 11 位芯片内、外存储器直接地址,地址范围 2KB,A0~A10
bit	表示位地址,RAM 位编址区 00H~7FH,SFR 可寻址位,取值范围 00H~0FFH
rel	表示地址偏移量,地址范围 256B,其数值为 8 位有符号数据,取值区域-128 ~+127
@	表示间接寻址,@R <sub>0</sub> ,@R <sub>1</sub> ,@A+DPTR、PC
/	表示取非位操作,/ bit = $\overline{\text{bit}}$

### 3.1.3 寻址方式

51 单片机 CPU 指令系统有 7 种寻址方式(寻找操作数或操作数所在地址的方式),7 种寻址方式的名称以及寻找操作数所对应的存储器地址空间如表 3-3 所示。

表 3-3 51 单片机 CPU 指令寻址方式

寻 址 方 式	存储器地址空间—操作数存放空间
立即寻址	立即数( #data),ROM 程序存储空间
直接寻址	RAM 区域(direct),芯片内 RAM 低 128B、特殊功能寄存器 SFR
寄存器寻址	工作寄存器 R <sub>0</sub> ~R <sub>7</sub> 、A、B、DPTR 等特殊功能寄存器 SFR
寄存器间接寻址	芯片内、外 RAM 空间,针对间接寄存器@R <sub>0</sub> 、@R <sub>1</sub> 、SP、DPTR
变址寻址	ROM 程序存储空间,针对变址基址 DPTR、PC 寄存器,@A+PC、@A+DPTR
相对寻址	ROM 程序存储空间,相对 PC 寄存器(PC+rel),rel 偏移量范围为-128~+127B
位寻址	芯片内 RAM(20H~2FH)位寻址空间和可位寻址的特殊功能寄存器 SFR 位地址,针对位操作

51 单片机 CPU 指令系统的 7 种寻址方式针对 CPU 中的寄存器、机器内部 RAM 和 ROM,以及机器外部的 RAM 和 ROM 数据存储区域,由于 51 单片机的输入/输出(I/O)接口地址采用内部存储器 RAM 映像方式,其地址区域为 80H~FFH,即特殊功能寄存器 SFR 地址区域,因此,所有适合操作内部存储器 RAM 的寻址方式都适合输入/输出(I/O)接口访问的操作。

51 单片机的多种寻址方式可方便地从某一地址处得到数据,或者将数据传送到另外一地址处,51 单片机 CPU 的指令系统通过各种寻址方式能实现的数据流动(CPU 读或存数据)方向如图 3-2 所示。

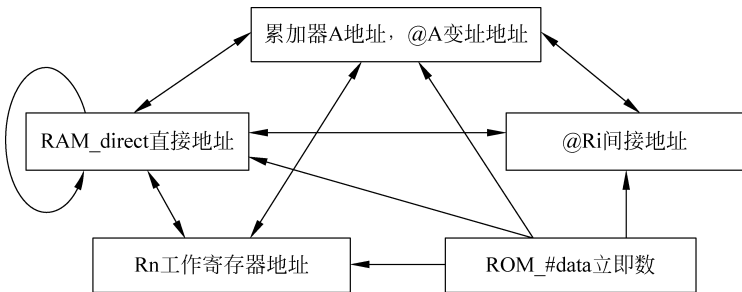


图 3-2 CPU 指令系统应用各种寻址方式使数据流动的方向图

由图 3-2 可知,在 ROM 中存储的立即数可以传送到 RAM、累加器 A、间接寄存器 R<sub>0</sub>和 R<sub>1</sub> 指示的地址区域;CPU 中的寄存器(RAM 高 128B 处的 SFR)以及 RAM 地址区域中的数据可以通过各种寻址方式相互传送。

3.2 数据传送类指令

51 单片机 CPU 的数据传送类指令是将数据从某一地址处取出传送到另外一个地址中,数据传送类指令的功能包括存储器之间、寄存器之间、存储器和寄存器之间、字节交换、堆栈操作、输入/输出操作等数据传送,并有多种寻址方式实现数据的传送。

### 3.2.1 数据传送指令

51 单片机 CPU 共有 29 条数据传送指令,这些指令的机器码、助记符、完成的功能、影响程序状态字 PSW 的标志位、占用存储器字节数、CPU 执行指令所用时间(机器周期)等如表 3-4 所示。

表 3-4 51 单片机 CPU 数据传送指令表

机器码	助 记 符		功 能	影响 PSW 标志位				字节数	机器周期
	操作码	操作数		CY	AC	OV	P		
E8~EF	MOV	A,Rn	$A \leftarrow R_n$ (n=0~7)	×	×	×	√	1	1
E5	MOV	A,direct	$A \leftarrow (\text{direct})$	×	×	×	√	2	1
E6,E7	MOV	A,@Ri	$A \leftarrow (R_i)$ (i=0,1)	×	×	×	√	1	1
74	MOV	A,#data	$A \leftarrow \text{data}$	×	×	×	√	2	1
F8~FF	MOV	Rn,A	$R_n \leftarrow A$ (n=0~7)	×	×	×	×	1	1
A8~AF	MOV	Rn,direct	$R_n \leftarrow (\text{direct})$ (n=0~7)	×	×	×	×	2	2
78~7F	MOV	Rn,#data	$R_n \leftarrow \text{data}$ (n=0~7)	×	×	×	×	2	1
F5	MOV	direct,A	$(\text{direct}) \leftarrow A$	×	×	×	×	2	1
88~8F	MOV	direct,Rn	$(\text{direct}) \leftarrow R_n$ (n=0~7)	×	×	×	×	2	2
85	MOV	direct1,direct2	$(\text{direct1}) \leftarrow (\text{direct2})$	×	×	×	×	3	2
86,87	MOV	direct,@Ri	$(\text{direct}) \leftarrow (R_i)$ (i=0,1)	×	×	×	×	2	2
75	MOV	direct,#data	$(\text{direct}) \leftarrow \text{data}$	×	×	×	×	3	2
F6,F7	MOV	@Ri,A	$(R_i) \leftarrow A$ (i=0,1)	×	×	×	×	1	1
A6,A7	MOV	@Ri,direct	$(R_i) \leftarrow (\text{direct})$ (i=0,1)	×	×	×	×	2	2
76,77	MOV	@Ri,#data	$(R_i) \leftarrow \text{data}$ (i=0,1)	×	×	×	×	2	1
90	MOV	DPTR,#data16	$\text{DPTR} \leftarrow \text{data16}$	×	×	×	×	3	2
E2,E3	MOVB	A,@Ri	$A \leftarrow (R_i)$ (i=0,1)	×	×	×	√	1	2
E0	MOVB	A,@DPTR	$A \leftarrow (\text{DPTR})$	×	×	×	√	1	2
F2,F3	MOVB	@Ri,A	$(R_i) \leftarrow A$ (i=0,1)	×	×	×	×	1	2
F0	MOVB	@DPTR,A	$(\text{DPTR}) \leftarrow A$	×	×	×	×	1	2
93	MOVB	A,@A+DPTR	$A \leftarrow (A + \text{DPTR})$	×	×	×	√	1	2
83	MOVB	A,@A+PC	$\text{PC} \leftarrow \text{PC} + 1, A \leftarrow (A + \text{PC})$	×	×	×	√	1	2
C0	PUSH	direct	$\text{SP} \leftarrow \text{SP} + 1, (\text{SP}) \leftarrow (\text{direct})$	×	×	×	×	2	2
D0	POP	direct	$(\text{direct}) \leftarrow (\text{SP}), \text{SP} \leftarrow \text{SP} - 1$	×	×	×	×	2	2
C8~CF	XCH	A,Rn	$A \leftrightarrow R_n$ (n=0~7)	×	×	×	√	1	1
C5	XCH	A,direct	$A \leftrightarrow (\text{direct})$	×	×	×	√	2	1
C6,C7	XCH	A,@Ri	$A \leftrightarrow (R_i)$ (i=0,1)	×	×	×	√	1	1
D6,D7	XCHD	A,@Ri	$A_0 \sim A_3 \leftrightarrow (R_i)_0 \sim (R_i)_3$ (i=0,1)	×	×	×	√	1	1
C4	SWAP	A	$A_0 \sim A_3 \leftrightarrow A_4 \sim A_7$	×	×	×	√	1	1

3.2.2 数据传送指令详解

数据传送指令包括 51 单片机内部数据传送指令 MOV、CPU 与外部数据交换数据指令 MOVB、读取程序存储器数据指令 MOVC、其他数据交换指令。

1. 机器内部数据传送指令

(1) MOV A,Rn 指令

指令机器码：

1 1 1 0	1 r r r
---------	---------

指令实现的操作：将工作寄存器 Rn(R<sub>0</sub>~R<sub>7</sub>)之一的内容送入累加器 A 中,指令机器码(CPU 可执行代码)中 r r r 3 位数据 000~111 表示 R<sub>0</sub>~R<sub>7</sub>,该指令影响程序状态字 PSW 的 P 标志位,P 表示送到累加器 A 中数据的奇偶性,累加器 A 中奇数个 1,P=1,否则 P=0。

(2) MOV A,direct 指令

指令机器码：

1 1 1 0	0 1 0 1
direct	

指令实现的操作：取出片内 RAM 直接地址单元 direct 中的内容送入累加器 A 中,该指令影响程序状态字 PSW 的 P 标志位。

(3) MOV A,@Ri 指令

指令机器码：

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

指令实现的操作：工作寄存器 Ri(R<sub>0</sub>~R<sub>1</sub>)之一中的数据作为片内 RAM 地址(通过 Ri 间接指示 RAM 地址),取出 Ri 指示的 RAM 地址中的内容送入累加器 A 中,指令机器码中 i 为一位数据 0~1 表示 R<sub>0</sub>、R<sub>1</sub>,该指令影响程序状态字 PSW 的 P 标志位。

(4) MOV A,#data 指令

指令机器码：

1 1 1 0	0 1 0 1
# data	

指令实现的操作：将 ROM 中的立即数送入累加器 A 中,该指令影响程序状态字 PSW 的 P 标志位。

(5) MOV Rn,A 指令

指令机器码：

1 1 1 1	1 r r r
---------	---------

指令实现的操作：将累加器 A 中的数据送入工作寄存器 Rn(R<sub>0</sub>~R<sub>7</sub>)之一中,指令机器码中 r r r 3 位数据 000~111 表示 R<sub>0</sub>~R<sub>7</sub>。

(6) MOV Rn,direct 指令

指令机器码：

1 0 1 0	1 r r r
direct	

指令实现的操作：取出片内 RAM 直接地址单元 direct 中的内容送入工作寄存器 Rn(R<sub>0</sub>~R<sub>7</sub>)之一中,指令机器码中 r r r 3 位数据 000~111 表示 R<sub>0</sub>~R<sub>7</sub>。

(7) MOV Rn, #data 指令

指令机器码:

0 1 1 1 1 r r r
# data

指令实现的操作：将 ROM 中的立即数送入工作寄存器 Rn(R<sub>0</sub>~R<sub>7</sub>)之一中。

(8) MOV direct,A 指令

指令机器码:

1 1 1 1 0 1 0 1
direct

指令实现的操作：将累加器 A 中的数据送入片内 RAM 直接地址单元 direct 中。

(9) MOV direct,Rn 指令

指令机器码:

1 0 0 0 1 r r r
direct

指令实现的操作：将工作寄存器 Rn(R<sub>0</sub>~R<sub>7</sub>)之一中的数据送入 RAM 直接地址单元 direct 中。

(10) MOV direct1,direct2 指令

指令机器码:

1 0 0 0 0 1 0 1
direct2(源)
direct1(目的)

指令实现的操作：取出片内 RAM 直接地址单元 direct2(源地址)中的内容送入 RAM 直接地址单元 direct1(目的地址)中。

(11) MOV direct,@Ri 指令

指令机器码:

1 0 0 0 0 1 1 i
direct

指令实现的操作：工作寄存器 Ri 之一中的数据作为片内 RAM 地址,取出 Ri 指示的 RAM 地址中的内容送入 RAM 直接地址单元 direct 中,指令机器码中 i(0 或 1)表示 R<sub>0</sub>、R<sub>1</sub>。

(12) MOV direct,#data 指令

指令机器码:

0 1 1 1 0 1 0 1
direct
# data

指令实现的操作：将 ROM 中的立即数送入片内 RAM 直接地址单元 direct 中。

(13) MOV @Ri,A 指令

指令机器码:

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

指令实现的操作：将累加器 A 中的数据送入工作寄存器 Ri(R<sub>0</sub>~R<sub>1</sub>)之一中的数据作为片内 RAM 地址的存储单元中,指令机器码中 i(0 或 1)表示 R<sub>0</sub>、R<sub>1</sub>。

(14) MOV @Ri,direct 指令

指令机器码:

0 1 0 1 0 1 1 i
direct

指令实现的操作：取出 RAM 直接地址单元 direct 中的数据送入工作寄存器 Ri(R<sub>0</sub>~R<sub>1</sub>)之一中的数据作为片内 RAM 地址(间址)的存储单元中,指令机器码中 i 表示 R<sub>0</sub>、R<sub>1</sub>。

(15) MOV @Ri,#data 指令

指令机器码:

0 1 1 1 0 1 1 i
#data

指令实现的操作：将 ROM 中的立即数送入工作寄存器 Ri(R<sub>0</sub>~R<sub>1</sub>)之一中的数据作为片内 RAM 地址(间址)的存储单元中,指令机器码中 i(0 或 1)表示 R<sub>0</sub>、R<sub>1</sub>。

(16) MOV DPTR,#data16 指令

指令机器码:

1 0 0 1 0 0 0 0
#data 高 8 位
#data 低 8 位

指令实现的操作：将 ROM 中的 16 位立即数送入地址寄存器 DPTR 中,立即数高 8 位送入 DPH,立即数低 8 位送入 DPL。

2. CPU 与外部存储器 RAM 交换数据指令

(1) MOVX A,@Ri 指令

指令机器码:

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

指令实现的操作：工作寄存器 Ri(R<sub>0</sub>~R<sub>1</sub>)之一中的数据作为外部存储器 RAM 地址(间接寻址范围 0~0FFH),取出 Ri 指示的外部 RAM 地址中的内容送入累加器 A 中,指令机器码中 i(0 或 1)表示 R<sub>0</sub>、R<sub>1</sub>,该指令影响程序状态字 PSW 的 P 标志位,该操作使总线读信号  $\overline{RD}=0$ 。

(2) MOVX A,@DPTR 指令

指令机器码:

1 1 1 0 0 0 0 0
-----------------

指令实现的操作：DPTR 寄存器的内容作为外部存储器 RAM 地址指针(间接寻址范围 0~0FFFFH),取出 DPTR 中指示的外部 RAM 地址中的内容送入累加器 A 中,该指令影响程序状态字 PSW 的 P 标志位,该操作使总线读信号  $\overline{RD}=0$ 。

(3) MOVX @Ri,A 指令

指令机器码:

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

指令实现的操作：工作寄存器 Ri(R<sub>0</sub>~R<sub>1</sub>)之一中的数据作为外部存储器 RAM 地址,将累加器 A 中的数据送入 Ri 指示的外部 RAM 地址中(间接寻址单元),指令机器码中 i(0 或 1)表示 R<sub>0</sub>、R<sub>1</sub>,该操作使总线写信号  $\overline{WR}=0$ 。

(4) MOVX @DPTR,A 指令

指令机器码: 

1	1	1	1	0	0	0	0
---	---	---	---	---	---	---	---

指令实现的操作: DPTR 寄存器的内容作为外部存储器 RAM 地址指针,将累加器 A 中的数据送入 DPTR 中指示的外部 RAM 地址中(间接寻址单元),该操作使总线写信号  $\overline{WR}=0$ 。

3. 读取程序存储器数据指令

(1) MOVC A,@ A+DPTR 指令

指令机器码: 

1	0	0	1	0	0	1	1
---	---	---	---	---	---	---	---

指令实现的操作: DPTR 寄存器的内容作为程序存储器 ROM 的基地址,将累加器 A 中的内容与 DPTR 寄存器的内容做无符号相加,形成一个相加后的 ROM 指示地址(基址变址寻址),取出该 ROM 地址单元中的内容送入累加器 A 中,该指令影响程序状态字 PSW 的 P 标志位。

MOVC A,@ A+DPTR 指令被称为查表指令,使用该指令时首先需要确定 DPTR 基址寄存器的内容,然后通过累加器 A 指出读取的数据相对 DPTR 指针所在的位置,其基址变址寻址范围为 ROM 的 0~0FFFFH,相对基址 DPTR 寻址范围为 0~256B。

(2) MOVC A,@ A+PC 指令

指令机器码: 

1	0	0	0	0	0	1	1
---	---	---	---	---	---	---	---

指令实现的操作: 首先将程序计数器 PC 的内容加 1(字节),然后将 PC 的内容作为程序存储器 ROM 的基地址,将累加器 A 中的内容与 PC 的内容做无符号相加,形成一个相加后的 ROM 指示地址(基址变址寻址),取出该 ROM 地址单元中的内容送入累加器 A 中,该指令影响程序状态字 PSW 的 P 标志位。

MOVC A,@ A+PC 指令同样也可以作为查表指令使用,但其是以 PC(CPU 将要执行的下一条指令的存储地址)的内容为基址的,由于 PC 的内容是不能通过指令修改的,而累加器 A 为 8 位寄存器,因此,该指令基址变址寻址范围为 0~256B。

4. 其他数据交换指令

(1) PUSH direct 指令

指令机器码: 

1	1	0	0	0	0	0	0
direct							

指令实现的操作: 首先将堆栈指针寄存器 SP 的内容加 1,然后将直接地址单元 direct 中的数据送入 SP 指示的地址单元中,SP 指向片内 RAM 处,其初值为 07H。

(2) POP direct 指令

指令机器码: 

1	1	0	1	0	0	0	0
direct							

指令实现的操作: 首先将堆栈指针寄存器 SP 指示的地址单元的内容送入直接地址单



元 direct 中,然后将 SP 内容减 1。

(3) XCH A,Rn 指令

指令机器码: 

1	1	0	0	1	r	r	r
---	---	---	---	---	---	---	---

指令实现的操作: 将工作寄存器 Rn(R<sub>0</sub>~R<sub>7</sub>)之一的内容与累加器 A 的内容互相交换,指令机器码中 r r r 表示 R<sub>0</sub>~R<sub>7</sub>,该指令影响程序状态字 PSW 的 P 标志位。

(4) XCH A,direct 指令

指令机器码: 

1	1	0	0	0	1	0	1
direct							

指令实现的操作: 将直接地址单元 direct 中的内容与累加器 A 的内容互相交换,该指令影响程序状态字 PSW 的 P 标志位。

(5) XCH A,@Ri 指令

指令机器码: 

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

指令实现的操作: 工作寄存器 Ri(R<sub>0</sub>~R<sub>7</sub>)之一中的数据作为内部存储器 RAM 地址,将 Ri 指示的内部 RAM 地址(间接寻址单元)中的内容与累加器 A 的内容互相交换,该指令影响程序状态字 PSW 的 P 标志位。

(6) XCHD A,@Ri 指令

指令机器码: 

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

指令实现的操作: 工作寄存器 Ri(R<sub>0</sub>~R<sub>7</sub>)之一中的数据作为内部存储器 RAM 地址,将 Ri 指示的内部 RAM 地址(间接寻址单元)中的低 4 位数据内容与累加器 A 的低 4 位数据内容互相交换,它们各自的高 4 位数据内容不变,该指令影响程序状态字 PSW 的 P 标志位。

(7) SWAP A 指令

指令机器码: 

1	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---

指令实现的操作: 将累加器 A 中的高 4 位数据与其低 4 位数据互相交换,该指令影响程序状态字 PSW 的 P 标志位。

### 3.3 算术运算类指令

51 单片机的算术运算指令是实现 8 位二进制数据运算的,其运算包括算术加、减、乘、除、十进制调整等。

#### 3.3.1 算术运算指令

51 单片机 CPU 有 24 条算术运算指令,算术运算指令的助记符、完成的功能、影响程序

状态字 PSW 的标志位、占用存储器字节数、CPU 执行指令所用时间(机器周期)等如表 3-5 所示。

表 3-5 51 单片机 CPU 算术运算指令表

机器码	助 记 符		功 能	影响 PSW 标志位				字节数	机器周期
	操作码	操作数		CY	AC	OV	P		
28~2F	ADD	A,Rn	$A \leftarrow A + R_n \quad (n=0 \sim 7)$	√	√	√	√	1	1
25	ADD	A,direct	$A \leftarrow A + (\text{direct})$	√	√	√	√	2	1
26,27	ADD	A,@Ri	$A \leftarrow A + (R_i) \quad (i=0,1)$	√	√	√	√	1	1
24	ADD	A,#data	$A \leftarrow A + \text{data}$	√	√	√	√	2	1
38~3F	ADDC	A,Rn	$A \leftarrow A + R_n + CY \quad (n=0 \sim 7)$	√	√	√	√	1	1
35	ADDC	A,direct	$A \leftarrow A + (\text{direct}) + CY$	√	√	√	√	2	1
36,37	ADDC	A,@Ri	$A \leftarrow A + (R_i) + CY \quad (i=0,1)$	√	√	√	√	1	1
34	ADDC	A,#data	$A \leftarrow A + \text{data} + CY$	√	√	√	√	2	1
98~9F	SUBB	A,Rn	$A \leftarrow A - R_n - CY \quad (n=0 \sim 7)$	√	√	√	√	1	1
95	SUBB	A,direct	$A \leftarrow A - (\text{direct}) - CY$	√	√	√	√	2	1
96,97	SUBB	A,@Ri	$A \leftarrow A - (R_i) - CY \quad (i=0,1)$	√	√	√	√	1	1
94	SUBB	A,#data	$A \leftarrow A - \text{data} - CY$	√	√	√	√	2	1
04	INC	A	$A \leftarrow A + 1$	×	×	×	√	1	1
08~0F	INC	Rn	$R_n \leftarrow R_n + 1 \quad (n=0 \sim 7)$	×	×	×	×	1	1
05	INC	direct	$(\text{direct}) \leftarrow (\text{direct}) + 1$	×	×	×	×	2	1
06,07	INC	@Ri	$(R_i) \leftarrow (R_i) + 1 \quad (i=0,1)$	×	×	×	×	1	1
A3	INC	DPTR	$DPTR \leftarrow DPTR + 1$	×	×	×	×	1	2
14	DEC	A	$A \leftarrow A - 1$	×	×	×	√	1	1
18~1F	DEC	Rn	$R_n \leftarrow R_n - 1 \quad (n=0 \sim 7)$	×	×	×	×	1	1
15	DEC	direct	$(\text{direct}) \leftarrow (\text{direct}) - 1$	×	×	×	×	2	1
16,17	DEC	@Ri	$(R_i) \leftarrow (R_i) - 1 \quad (i=0,1)$	×	×	×	×	1	1
A4	MUL	AB	$BA \leftarrow A \times B$	0	×	√	√	1	4
84	DIV	AB	$A \leftarrow A \div B$	0	×	√	√	1	4
D4	DA	A	$A \leftarrow BCD(A)$	√	√	×	√	1	1

3.3.2 算术运算指令详解

51 单片机 CPU 的算术运算指令实现了二进制数据的算术加、减、乘、除、加 1、减 1 等操作,其大部分指令都会影响程序状态字 PSW 的 CY、AC、OV、P 标志位。

1. 算术加运算指令

(1) ADD A,Rn 指令

指令机器码：

0	0	1	0	1	r	r	r
---	---	---	---	---	---	---	---

指令实现的操作：将工作寄存器  $R_n(R_0 \sim R_7)$  之一的内容与累加器 A 中的内容进行 8 位无符号数据相加,相加结果送入累加器 A 中,指令机器码中  $r\ r\ r\ 3$  位数据 000~111 表示  $R_0 \sim R_7$ 。

该指令影响程序状态字 PSW 的 CY、AC、OV、P 标志位,当第 7 位相加结果有进位时  $CY=1$ ;当第 3 位相加结果有进位时,  $AC=1$ ;当使用该指令实现有符号数据相加运算时,  $OV$  等于第 7 位与第 6 位数据的异或,表示运算有溢出;当送入累加器 A 中的运算结果有奇数个 1 时,  $P=1$ 。

(2) ADD A,direct 指令

指令机器码:

0 0 1 0   0 1 0 1
direct

指令实现的操作：将直接地址单元 direct 中的内容与累加器 A 的内容进行 8 位无符号数据相加,相加结果送入累加器 A 中,该指令影响 PSW 的 CY、AC、OV、P 标志位。

(3) ADD A,@Ri 指令

指令机器码:

0 0 1 0   0 1 1 i
-------------------

指令实现的操作：工作寄存器  $R_i(R_0 \sim R_7)$  之一中的数据作为内部存储器 RAM 地址,将  $R_i$  指示的内部 RAM 地址(间接寻址单元)中的数据内容与累加器 A 的数据内容进行 8 位无符号数据相加,相加结果送入累加器 A 中,该指令影响 PSW 的 CY、AC、OV、P 标志位。

(4) ADD A,#data 指令

指令机器码:

0 0 1 0   0 1 0 0
# data

指令实现的操作：将 ROM 中的立即数与累加器 A 的内容进行 8 位无符号数据相加,相加结果送入累加器 A 中,该指令影响 PSW 的 CY、AC、OV、P 标志位。

(5) ADDC A,Rn 指令

指令机器码:

0 0 1 1   1 r r r
-------------------

指令实现的操作：将工作寄存器  $R_n(R_0 \sim R_7)$  之一的内容与累加器 A 中的内容以及在最低位(第 0 位)上与 PSW 的 CY(进位位)进行三者数据相加,相加结果送入累加器 A 中,影响 PSW 的标志位情况与 ADD 指令相同。

(6) ADDC A,direct 指令

指令机器码:

0 0 1 1   0 1 0 1
direct

指令实现的操作：将直接地址单元 direct 中的内容与累加器 A 的内容以及在最低位上与 PSW 的 CY 位进行三者数据相加,相加结果送入累加器 A 中,影响 PSW 的标志位情况与 ADD 指令相同。

(7) ADDC A,@Ri 指令

指令机器码: 

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

指令实现的操作: 工作寄存器 Ri(R<sub>0</sub>~R<sub>7</sub>)之一中的数据作为内部存储器 RAM 地址, 将 Ri 指示的内部 RAM 地址(间接寻址单元)中的数据内容与累加器 A 的数据内容以及在最低位上与 PSW 的 CY 位进行三者数据相加, 相加结果送入累加器 A 中, 影响 PSW 的标志位情况与 ADD 指令相同。

(8) ADDC A,#data 指令

指令机器码: 

0	0	1	1	0	1	0	0
#data							

指令实现的操作: 将 ROM 中的立即数与累加器 A 的内容以及在最低位上与 PSW 的 CY 位进行三者数据相加, 相加结果送入累加器 A 中, 影响 PSW 的标志位情况与 ADD 指令相同。

2. 算术减运算指令

(1) SUBB A,Rn 指令

指令机器码: 

1	0	0	1	1	r	r	r
---	---	---	---	---	---	---	---

指令实现的操作: 实现 8 位无符号数的减法运算, 将累加器 A 中的内容减去工作寄存器 Rn(R<sub>0</sub>~R<sub>7</sub>)之一的内容, 同时在最低位(第 0 位)上减去 PSW 的 CY(进位位)的数值, 相减结果送入累加器 A 中。

该指令影响程序状态字 PSW 的 CY、AC、OV、P 标志位, 当第 7 位相减结果产生借位时 CY=1; 当第 3 位相减结果产生借位时 AC=1; 当使用该指令实现有符号数据相减运算时, OV=1 表示运算有溢出, 它将破坏正确的符号位运算结果; 当送入累加器 A 中的运算结果有奇数个 1 时, P=1。

51 单片机 CPU 的运算指令是对一个字节而言的, 在实现多字节减法运算时, 低字节相减会向高字节产生借位(CY=1), 当在高字节进行相减运算时使用该指令正好实现带借位的相减运算。由于该 CPU 没有不带借位相减运算的指令, 因此, 为保证运算结果的正确性, 在做单字节或多字节的减法运算之前, 需要将 CY 清 0(CLR CY)。

(2) SUBB A,direct 指令

指令机器码: 

1	0	0	1	0	1	0	1
direct							

指令实现的操作: 将累加器 A 中的内容减去直接地址单元 direct 中的内容, 同时, 在第 0 位上减去 PSW 的 CY 的数值, 相减结果送入累加器 A 中。SUBB 指令影响程序状态字 PSW 的标志位情况相同。

(3) SUBB A,@Ri 指令

指令机器码: 

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

指令实现的操作：工作寄存器 Ri(R<sub>0</sub>~R<sub>1</sub>)之一中的数据作为内部存储器 RAM 地址(间接寻址单元),将累加器 A 中的内容减去 Ri 指示的内部 RAM 地址中的内容,同时第 0 位上减去 PSW 的 CY 的数值,相减结果送入累加器 A 中。SUBB 指令影响程序状态字 PSW 的标志位情况相同。

(4) SUBB A, #data 指令

指令机器码：

1 0 0 1 0 1 0 0
#data

指令实现的操作：将累加器 A 中的内容减去 ROM 中的立即数,同时第 0 位上减去 PSW 的 CY 的数值,相减结果送入累加器 A 中。SUBB 指令影响程序状态字 PSW 的标志位情况相同。

3. 算术加 1 操作指令

(1) INC A 指令

指令机器码：

0 0 0 0 0 1 0 0
-----------------

指令实现的操作：累加器 A 中的内容加 1,该指令影响 PSW 中的 P 标志位,当加 1 后累加器 A 中有奇数个 1 时,P=1。

(2) INC Rn 指令

指令机器码：

0 0 0 0 1 r r r
-----------------

指令实现的操作：工作寄存器 Rn(R<sub>0</sub>~R<sub>7</sub>)之一的内容加 1,该指令不影响 PSW 标志位。

(3) INC direct 指令

指令机器码：

0 0 0 0 0 1 0 1
direct

指令实现的操作：RAM 直接地址单元 direct 中的内容加 1,该指令不影响 PSW 标志位。

(4) INC @Ri 指令

指令机器码：

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

指令实现的操作：工作寄存器 Ri(R<sub>0</sub>~R<sub>1</sub>)之一中的数据作为内部 RAM 地址(间接寻址单元),将 Ri 指示的内部 RAM 地址中的内容加 1,该指令不影响 PSW 标志位。

(5) INC DPTR 指令

指令机器码：

1 0 1 0 0 0 1 1
-----------------

指令实现的操作：DPTR 寄存器中的内容加 1,该指令不影响 PSW 标志位。

4. 算术减 1 操作指令

(1) DEC A 指令

指令机器码：

0 0 0 1 0 1 0 0
-----------------

指令实现的操作：累加器 A 中的内容减 1，该指令影响 PSW 中的 P 标志位，当减 1 后累加器 A 中有奇数个 1 时， $P=1$ 。

#### (2) DEC Rn 指令

指令机器码：

0	0	0	1	1	r	r	r
---	---	---	---	---	---	---	---

指令实现的操作：工作寄存器  $R_n(R_0 \sim R_7)$  之一的内容减 1，该指令不影响 PSW 标志位。

#### (3) DEC direct 指令

指令机器码：

0	0	0	1	0	1	0	1
direct							

指令实现的操作：RAM 直接地址单元 direct 中的内容减 1，该指令不影响 PSW 标志位。

#### (4) DEC @Ri 指令

指令机器码：

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

指令实现的操作：工作寄存器  $R_i(R_0 \sim R_7)$  之一中的数据作为内部 RAM 地址(间接寻址单元)，将  $R_i$  指示的内部 RAM 地址中的内容减 1，该指令不影响 PSW 标志位。

### 5. 算术乘、除运算指令

#### (1) MUL AB 指令

指令机器码：

1	0	1	0	0	1	0	0
---	---	---	---	---	---	---	---

指令实现的操作：两个 8 位无符号整数数据相乘，累加器 A 中的内容乘以寄存器 B 中的内容，所得 16 位乘积，乘积的高 8 位送入寄存器 B 中，乘积的低 8 位送入累加器 A 中，该乘法指令影响程序状态字 PSW 的 OV、P 标志位，当乘积大于 0FFH 时， $OV=1$ ；P 标志位反映累加器 A 中数据的奇偶性。

#### (2) DIV AB 指令

指令机器码：

1	0	0	0	0	1	0	0
---	---	---	---	---	---	---	---

指令实现的操作：两个 8 位无符号整数数据相除，累加器 A 中的内容除以寄存器 B 中的内容，所得商送入累加器 A 中，余数送入寄存器 B 中，该除法指令影响程序状态字 PSW 的 OV、P 标志位，当寄存器 B 中的内容(除数)为 0 时， $OV=1$ ，其操作非法，结果不正确；P 标志位反映累加器 A 中数据的奇偶性。

### 6. 十进制调整指令

#### DA A 指令

指令机器码：

1	1	0	1	0	1	0	0
---	---	---	---	---	---	---	---

指令实现的操作：十进制调整，将累加器 A 中的内容调整为 BCD 码，其调整的原则如下。

若  $A_0 \sim A_3 > 9$  或 PSW 的标志位  $AC=1$ ，则累加器 A 低 4 位( $A_0 \sim A_3$ )加 6 再送回累加

器 A 低 4 位中。

若  $A_4 \sim A_7 > 9$  或 PSW 的标志位  $CY=1$ , 则累加器 A 高 4 位 ( $A_4 \sim A_7$ ) 加 6 再送回累加器 A 高 4 位中。

即 CPU 根据累加器 A 中的原始数据和 PSW 的 CY、AC 标志位, 自动实现对累加器 A 中的内容加 06H 或 60H 或 66H 的操作。

该指令是对累加器 A 中的 BCD 码相加结果进行调整, 当两个 BCD 码按二进制数据相加后, 需要该指令参与调整才能得到正确的 BCD 码结果, 在该指令被执行后, 当  $CY=1$  时, 说明 BCD 码的结果(值) $\geq 100$ 。

3.4 逻辑运算类指令

51 单片机的逻辑运算指令是实现 8 位二进制数据按位进行逻辑运算的, 其运算包括逻辑与、或、异或、非, 以及左移、右移、循环移位等操作。


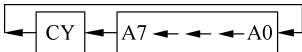
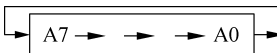
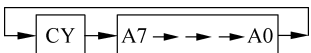
3.4.1 逻辑运算指令

51 单片机 CPU 有 24 条逻辑运算指令, 逻辑运算指令的助记符、完成的功能、影响程序状态字 PSW 的标志位、占用存储器字节数、CPU 执行指令所用时间(机器周期)等如表 3-6 所示。

表 3-6 51 单片机 CPU 逻辑运算指令表

机器码	助 记 符		功 能	影响 PSW 标志位				字 节 数	机 器 周 期
	操作码	操作数		CY	AC	OV	P		
58~5F	ANL	A,Rn	$A \leftarrow A \wedge R_n \quad (n=0 \sim 7)$	×	×	×	√	1	1
55	ANL	A,direct	$A \leftarrow A \wedge (\text{direct})$	×	×	×	√	2	1
56,57	ANL	A,@Ri	$A \leftarrow A \wedge (R_i) \quad (i=0,1)$	×	×	×	√	1	1
54	ANL	A,#data	$A \leftarrow A \wedge \text{data}$	×	×	×	√	2	1
52	ANL	direct,A	$(\text{direct}) \leftarrow (\text{direct}) \wedge A$	×	×	×	×	2	1
53	ANL	direct,#data	$(\text{direct}) \leftarrow (\text{direct}) \wedge \text{data}$	×	×	×	×	3	2
48~4F	ORL	A,Rn	$A \leftarrow A \vee R_n \quad (n=0 \sim 7)$	×	×	×	√	1	1
45	ORL	A,direct	$A \leftarrow A \vee (\text{direct})$	×	×	×	√	2	1
46,47	ORL	A,@Ri	$A \leftarrow A \vee (R_i) \quad (i=0,1)$	×	×	×	√	1	1
44	ORL	A,#data	$A \leftarrow A \vee \text{data}$	×	×	×	√	2	1
42	ORL	direct,A	$(\text{direct}) \leftarrow (\text{direct}) \vee A$	×	×	×	×	2	1
43	ORL	direct,#data	$(\text{direct}) \leftarrow (\text{direct}) \vee \text{data}$	×	×	×	×	3	2
68~6F	XRL	A,Rn	$A \leftarrow A \oplus R_n \quad (n=0 \sim 7)$	×	×	×	√	1	1
65	XRL	A,direct	$A \leftarrow A \oplus (\text{direct})$	×	×	×	√	2	1
66,67	XRL	A,@Ri	$A \leftarrow A \oplus (R_i) \quad (i=0,1)$	×	×	×	√	1	1

续表

机器码	助 记 符		功 能	影响 PSW 标志位				字节数	机器周期
	操作码	操作数		CY	AC	OV	P		
64	XRL	A, # data	$A \leftarrow A \oplus \text{data}$	×	×	×	√	2	1
62	XRL	direct, A	$(\text{direct}) \leftarrow (\text{direct}) \oplus A$	×	×	×	×	1	1
63	XRL	direct, # data	$(\text{direct}) \leftarrow (\text{direct}) \oplus \text{data}$	×	×	×	×	3	2
F4	CPL	A	$A \leftarrow \bar{A}$	×	×	×	×	1	1
E4	CLR	A	$A \leftarrow 0$	×	×	×	√	1	1
23	RL	A		×	×	×	×	1	1
33	RLC	A		√	×	×	√	1	1
03	RR	A		×	×	×	×	1	1
13	RRC	A		√	×	×	√	1	1

3.4.2 逻辑运算指令详解

51 单片机 CPU 的逻辑运算指令实现了二进制数据按位进行的逻辑与、逻辑或、逻辑异或、逻辑非、循环移位操作等,其部分指令只影响程序状态字 PSW 的 P 标志位。

1. 逻辑与运算指令

(1) ANL A,Rn 指令

指令机器码: 

0 1 0 1 1 r r r
-----------------

指令实现的操作: 将累加器 A 中的内容同工作寄存器 Rn(R<sub>0</sub>~R<sub>7</sub>)之一的内容按位进行“逻辑与”操作,结果送入累加器 A 中,该指令影响 PSW 的 P 标志位。

(2) ANL A,direct 指令

指令机器码: 

0 1 0 1 0 1 0 1
direct

指令实现的操作: 将累加器 A 中的内容同 RAM 直接地址单元 direct 中的内容按位进行“逻辑与”操作,结果送入累加器 A 中,该指令影响 PSW 的 P 标志位。

(3) ANL A,@Ri 指令

指令机器码: 

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

指令实现的操作: 工作寄存器 Ri(R<sub>0</sub>~R<sub>7</sub>)之一中的数据作为内部 RAM 地址(间接寻址单元),将累加器 A 中的内容同 Ri 指示的内部 RAM 地址中的内容按位进行“逻辑与”操



作,结果送入累加器 A 中,该指令影响 PSW 的 P 标志位。

(4) ANL A, #data 指令

指令机器码:

0 1 0 1 0 1 0 0
# data

指令实现的操作: 将累加器 A 中的内容同 ROM 中的立即数按位进行“逻辑与”操作,结果送入累加器 A 中,该指令影响 PSW 的 P 标志位。

(5) ANL direct, A 指令

指令机器码:

0 1 0 1 0 0 1 0
direct

指令实现的操作: 将 RAM 直接地址单元 direct 中的内容同累加器 A 中的内容按位进行“逻辑与”操作,结果送入直接地址单元 direct 中。

(6) ANL direct, #data 指令

指令机器码:

0 1 0 1 0 0 1 1
direct
# data

指令实现的操作: 将 RAM 直接地址单元 direct 中的内容同 ROM 中的立即数按位进行“逻辑与”操作,结果送入直接地址单元 direct 中。

2. 逻辑或运算指令

(1) ORL A, Rn 指令

指令机器码:

0 1 0 0 1 r r r
-----------------

指令实现的操作: 将累加器 A 中的内容同工作寄存器 Rn(R<sub>0</sub>~R<sub>7</sub>)之一的内容按位进行“逻辑或”操作,结果送入累加器 A 中,该指令影响 PSW 的 P 标志位。

(2) ORL A, direct 指令

指令机器码:

0 1 0 0 0 1 0 1
direct

指令实现的操作: 将累加器 A 中的内容同 RAM 直接地址单元 direct 中的内容按位进行“逻辑或”操作,结果送入累加器 A 中,该指令影响 PSW 的 P 标志位。

(3) ORL A, @Ri 指令

指令机器码:

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

指令实现的操作: 工作寄存器 Ri(R<sub>0</sub>~R<sub>1</sub>)之一中的数据作为内部 RAM 地址(间接寻址单元),将累加器 A 中的内容同 Ri 指示的内部 RAM 地址中的内容按位进行“逻辑或”操作,结果送入累加器 A 中,该指令影响 PSW 的 P 标志位。

(4) ORL A, #data 指令

指令机器码:

0 1 0 0 0 1 0 0
# data

指令实现的操作：将累加器 A 中的内容同 ROM 中的立即数按位进行“逻辑或”操作，结果送入累加器 A 中，该指令影响 PSW 的 P 标志位。

(5) ORL direct,A 指令

指令机器码：

0 1 0 0 0 0 1 0
direct

指令实现的操作：将 RAM 直接地址单元 direct 中的内容同累加器 A 中的内容按位进行“逻辑或”操作，结果送入直接地址单元 direct 中。

(6) ORL direct,#data 指令

指令机器码：

0 1 0 0 0 0 1 1
direct
#data

指令实现的操作：将 RAM 直接地址单元 direct 中的内容同 ROM 中的立即数按位进行“逻辑或”操作，结果送入直接地址单元 direct 中。

3. 逻辑异或运算指令

(1) XRL A,Rn 指令

指令机器码：

0 1 1 0 1 r r r
-----------------

指令实现的操作：将累加器 A 中的内容同工作寄存器 Rn(R<sub>0</sub>~R<sub>7</sub>)之一的内容按位进行“逻辑异或”操作，结果送入累加器 A 中，该指令影响 PSW 的 P 标志位。

(2) XRL A,direct 指令

指令机器码：

0 1 1 0 0 1 0 1
direct

指令实现的操作：将累加器 A 中的内容同 RAM 直接地址单元 direct 中的内容按位进行“逻辑异或”操作，结果送入累加器 A 中，该指令影响 PSW 的 P 标志位。

(3) XRL A,@Ri 指令

指令机器码：

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

指令实现的操作：工作寄存器 Ri(R<sub>0</sub>~R<sub>7</sub>)之一中的数据作为内部 RAM 地址(间接寻址单元)，将累加器 A 中的内容同 Ri 指示的内部 RAM 地址中的内容按位进行“逻辑异或”操作，结果送入累加器 A 中，该指令影响 PSW 的 P 标志位。

(4) XRL A,#data 指令

指令机器码：

0 1 1 0 0 1 0 0
#data

指令实现的操作：将累加器 A 中的内容同 ROM 中的立即数按位进行“逻辑异或”操作，结果送入累加器 A 中，该指令影响 PSW 的 P 标志位。

(5) XRL direct,A 指令

指令机器码:

0 1 1 0 0 0 1 0
direct

指令实现的操作：将 RAM 直接地址单元 direct 中的内容同累加器 A 中的内容按位进行“逻辑异或”操作，结果送入直接地址单元 direct 中。

(6) XRL direct,#data 指令

指令机器码:

0 1 1 0 0 0 1 1
direct
# data

指令实现的操作：将 RAM 直接地址单元 direct 中的内容同 ROM 中的立即数按位进行“逻辑异或”操作，结果送入直接地址单元 direct 中。

4. 逻辑非运算指令

(1) CPL A 指令

指令机器码:

1 1 1 1 0 1 0 0
-----------------

指令实现的操作：将累加器 A 中的内容按位进行“求反”操作，结果送入累加器 A 中。

(2) CLR A 指令

指令机器码:

1 1 1 0 0 1 0 0
-----------------

指令实现的操作：将累加器 A 中的内容按位进行“清 0”操作，结果送入累加器 A 中。

5. 循环移位操作指令

(1) RL A 指令

指令机器码:

0 0 1 0 0 0 1 1
-----------------

指令实现的操作：将累加器 A 中的内容按位循环左移一位，其左移过程为： $A_0 \leftarrow A_7 \leftarrow A_6 \leftarrow A_5 \leftarrow A_4 \leftarrow A_3 \leftarrow A_2 \leftarrow A_1 \leftarrow A_0$ 。

(2) RLC A 指令

指令机器码:

0 0 1 1 0 0 1 1
-----------------

指令实现的操作：将累加器 A 中的内容连同进位位按位循环左移一位，该指令影响 PSW 的 CY、P 标志位，其左移过程为： $CY \leftarrow A_7 \leftarrow A_6 \leftarrow A_5 \leftarrow A_4 \leftarrow A_3 \leftarrow A_2 \leftarrow A_1 \leftarrow A_0 \leftarrow CY$ 。

(3) RR A 指令

指令机器码:

0 0 0 0 0 0 1 1
-----------------

指令实现的操作：将累加器 A 中的内容按位循环右移一位，其右移过程为： $A_7 \rightarrow A_6 \rightarrow A_5 \rightarrow A_4 \rightarrow A_3 \rightarrow A_2 \rightarrow A_1 \rightarrow A_0 \rightarrow A_7$ 。

(4) RRC A 指令

指令机器码:

0 0 0 1 0 0 1 1
-----------------

指令实现的操作：将累加器 A 中的内容连同进位位按位循环右一位，该指令影响 PSW 的 CY、P 标志位，其右移过程为： $CY \rightarrow A_7 \rightarrow A_6 \rightarrow A_5 \rightarrow A_4 \rightarrow A_3 \rightarrow A_2 \rightarrow A_1 \rightarrow A_0 \rightarrow CY$ 。

3.5 控制转移类指令

控制转移类指令是用于控制改变程序执行流向的指令，该类指令包括无条件转移指令、条件转移指令、子程序调用指令、子程序返回指令、中断返回指令等。

3.5.1 控制转移指令

51 单片机 CPU 共有 17 条控制转移指令，这些指令的助记符、完成的功能、影响程序状态字 PSW 的标志位、占用存储器字节数、CPU 执行指令所用时间（机器周期）等如表 3-7 所示。

表 3-7 51 单片机 CPU 控制转移指令表

机器码	助 记 符		功 能	影响 PSW 标志位				字节数	机器周期
	操作码	操作数		CY	AC	OV	P		
*1	AJMP	addr11	$PC \leftarrow PC + 2, PC_{10 \sim 0} \leftarrow \text{addr11}$	×	×	×	×	2	2
02	LJMP	addr16	$PC \leftarrow \text{addr16}$	×	×	×	×	3	2
80	SJMP	rel	$PC \leftarrow PC + 2, PC \leftarrow PC + \text{rel}$	×	×	×	×	2	2
73	JMP	@A+DPTR	$PC \leftarrow (A + DPTR)$	×	×	×	×	1	2
60	JZ	rel	$PC \leftarrow PC + 2,$ if A=0; then $PC \leftarrow PC + \text{rel}$	×	×	×	×	2	2
70	JNZ	rel	$PC \leftarrow PC + 2,$ if A≠0; then $PC \leftarrow PC + \text{rel}$	×	×	×	×	2	2
B5	CJNE	A,direct,rel	$PC \leftarrow PC + 3,$ if A≠(direct); then $PC \leftarrow PC + \text{rel}$ if A<(direct); then $CY \leftarrow 1$	√	×	×	×	3	2
B4	CJNE	A,#data,rel	$PC \leftarrow PC + 3,$ if A≠data; then $PC \leftarrow PC + \text{rel}$ if A<data; then $CY \leftarrow 1$	√	×	×	×	3	2
B8~BF	CJNE	Rn,#data,rel	$PC \leftarrow PC + 3,$ if Rn≠data; then $PC \leftarrow PC + \text{rel}$ if Rn<data; then $CY \leftarrow 1 (n=0 \sim 7)$	√	×	×	×	3	2
B6、B7	CJNE	@Ri,#data,rel	$PC \leftarrow PC + 3,$ if (Ri)≠data; then $PC \leftarrow PC + \text{rel}$ if (Ri)<data; then $CY \leftarrow 1 (i=0,1)$	√	×	×	×	3	2
D8~DF	DJNZ	Rn,rel	$Rn \leftarrow Rn - 1, PC \leftarrow PC + 2,$ if Rn≠0; then $PC \leftarrow PC + \text{rel} (n=0 \sim 7)$	√	×	×	×	2	2

续表

机器码	助 记 符		功 能	影响 PSW 标志位				字节数	机器周期
	操作码	操作数		CY	AC	OV	P		
D5	DJNZ	direct,rel	$(direct) \leftarrow (direct) - 1, PC \leftarrow PC + 2,$ if $(direct) \neq 0$ ; then $PC \leftarrow PC + rel$	√	×	×	×	3	2
*1	ACALL	addr11	$PC \leftarrow PC + 2, SP \leftarrow SP + 1,$ $(SP) \leftarrow PCL, SP \leftarrow SP + 1,$ $(SP) \leftarrow PCH, PC_{10 \sim 0} \leftarrow addr11$	×	×	×	×	2	2
12	LCALL	addr16	$PC \leftarrow PC + 3, SP \leftarrow SP + 1,$ $(SP) \leftarrow PCL, SP \leftarrow SP + 1,$ $(SP) \leftarrow PCH, PC \leftarrow addr16$	×	×	×	×	3	2
22	RET		$PCH \leftarrow (SP), SP \leftarrow SP - 1,$ $PCL \leftarrow (SP), SP \leftarrow SP - 1$	×	×	×	×	1	2
32	RETI		$PCH \leftarrow (SP), SP \leftarrow SP - 1,$ $PCL \leftarrow (SP), SP \leftarrow SP - 1, CLR I\_flag$	×	×	×	×	1	2
00	NOP		no	×	×	×	×	1	1

3.5.2 控制转移指令详解

控制转移指令用于改变程序执行流向,包含无条件转移、条件转移、子程序调用、返回等指令。

1. 无条件转移指令

(1) AJMP addr11 指令

指令机器码:

$a_{10} \ a_9 \ a_8 \ 0 \ 0 \ 0 \ 0 \ 1$
$a_7 \sim a_0$

指令实现的操作: 无条件绝对转移,首先程序计数器 PC 的内容加 2,然后将指令机器码中  $a_{10} \sim a_0$  送入  $PC_{10} \sim PC_0$  中, $PC_{15} \sim PC_{11}$  内容不变。

AJMP 指令在指令机器码中指出低 11 位程序存储器 ROM 的地址,该指令可控制程序在 2KB 的范围内无条件转移执行,转移范围如图 3-3(a)所示。

(2) LJMP addr16 指令

指令机器码:

$0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0$
$a_{15} \sim a_8$
$a_7 \sim a_0$

指令实现的操作: 无条件绝对长转移,将指令机器码中  $a_{15} \sim a_0$  送入  $PC_{15} \sim PC_0$  中。

LJMP 指令在指令机器码中指出 16 位程序存储器 ROM 的地址,该指令可控制程序在 64KB 的范围内无条件转移执行,转移范围如图 3-3(b)所示。

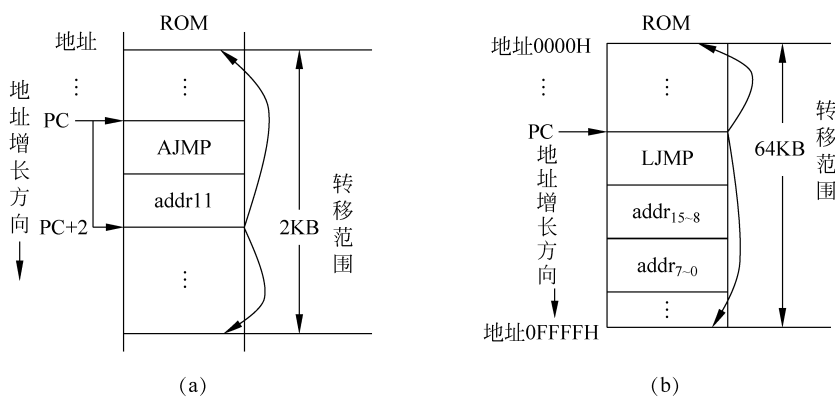


图 3-3 AJMP、LJMP 转移范围

(3) SJMP rel 指令

指令机器码:

1 0 0 0 0 0 0 0
rel(相对地址)

指令实现的操作：无条件相对转移，首先程序计数器 PC 的内容加 2，此时 PC 值是相对转移地址的基准值，然后将 PC 中的数值与转移偏移量 rel 数值(有符号数)按有符号数数据相加，相加结果送入 PC 中，rel 表示为 8 位有符号数据，因此，SJMP 指令的控制程序转移范围在  $-128 \sim +127B$  之间，如图 3-4(a)所示。

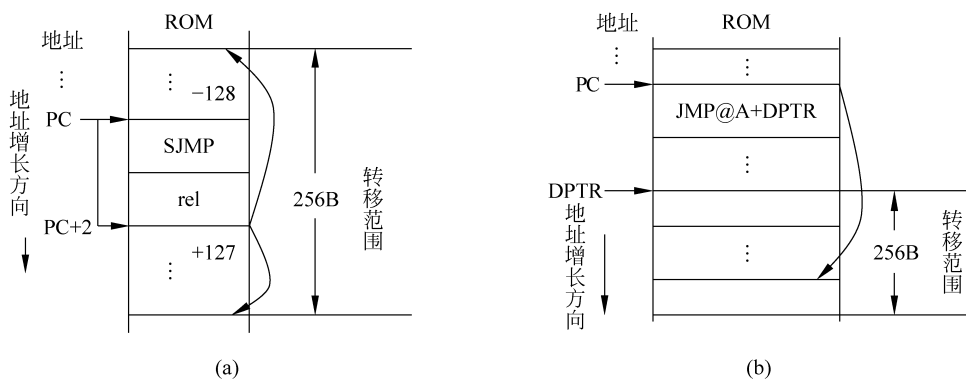


图 3-4 SJMP、JMP 转移范围

(4) JMP @A+DPTR 指令

指令机器码:

0 1 1 1 0 0 1 1
-----------------

指令实现的操作：无条件间接转移，也被称为散转指令，该指令的转移地址由 16 位数据指针寄存器 DPTR 的内容与 8 位累加器 A 的内容进行无符号数相加形成，将相加结果送入程序计数器 PC 中，其转移范围是以 DPTR 的内容为首地址的 256B 之内，并以 DPTR

的内容为基准向程序存储器 ROM 地址增长方向转移,如图 3-4(b)所示。

2. 比较(条件)转移指令

(1) JZ rel 指令

指令机器码:

0 1 1 0 0 0 0 0
rel(相对地址)

指令实现的操作: 首先程序计数器 PC 的内容加 2, 然后对累加器 A 的内容进行检测, 当 A 的各位全部为 0 时, 满足转移条件, 将 PC 中的数值与转移偏移量 rel 数值按有符号数据相加, 相加结果送入 PC 中, 实现程序转移执行, 该指令控制程序转移范围为 -128 ~ +127B; 当 A 不为 0 时, CPU 执行该指令的下一条指令, CPU 执行该指令的流程如图 3-5(a)所示。

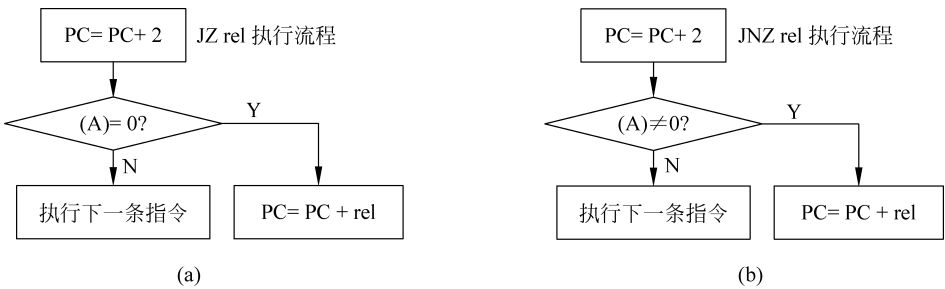


图 3-5 JZ、JNZ 指令执行流程

(2) JNZ rel 指令

指令机器码:

0 1 1 1 0 0 0 0
rel(相对地址)

指令实现的操作: 首先程序计数器 PC 的内容加 2, 然后对累加器 A 的内容进行检测, 当 A 不为 0 时, 满足转移条件, 将 PC 中的数值与转移偏移量 rel 数值按有符号数据相加, 相加结果送入 PC 中, 实现程序转移执行, 该指令控制程序转移范围为 -128 ~ +127B; 当 A 为全 0 时, CPU 执行该指令的下一条指令, CPU 执行该指令的流程如图 3-5(b)所示。

(3) CJNE A, direct, rel 指令

指令机器码:

1 0 1 1 0 1 0 1
direct
rel(相对地址)

指令实现的操作: 首先程序计数器 PC 的内容加 3, 然后将累加器 A 的内容与 RAM 直接地址单元 direct 中的内容进行比较, 当两者不相等时, 将 PC 中的数值与 rel 数值按有符号数据相加, 相加结果送入 PC 中, 实现程序转移执行, 该指令控制程序转移范围为 -128 ~ +127B, 另外, 该指令影响程序状态字 PSW 的 CY 标志位, 当 A 的内容大于 direct 中的内

容时,CY=0;当A的内容小于direct中的内容时,CY=1;当A与direct中的内容相等时,CY=0,CPU执行该指令的下一条指令,CPU执行该指令以及影响CY标志位的流程如图3-6所示。

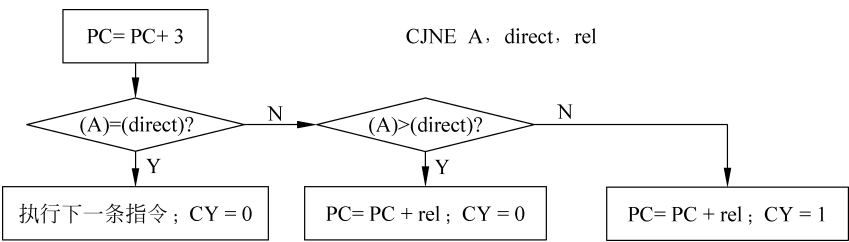


图 3-6 CJNE 指令执行流程

(4) CJNE A, #data,rel 指令

指令机器码:

1 0 1 1 0 1 0 0
# data
rel(相对地址)

指令实现的操作: 首先程序计数器 PC 的内容加 3,然后将累加器 A 的内容与 ROM 中的立即数进行比较,当两者不相等时,将 PC 中的数值与 rel 数值按有符号数据相加,相加结果送入 PC 中,实现程序转移执行,控制转移范围为-128~+127B;当参与比较的两个数据相等时,执行下一条指令,CJNE 指令执行流程以及影响程序状态字 PSW 的 CY 标志位都相同。

(5) CJNE Rn, #data,rel 指令

指令机器码:

1 0 1 1 1 r r r
# data
rel(相对地址)

指令实现的操作: 首先程序计数器 PC 的内容加 3,然后将工作寄存器 Rn(R<sub>0</sub>~R<sub>7</sub>)之一的内容与 ROM 中的立即数进行比较,当两者不相等时,将 PC 中的数值与 rel 数值按有符号数据相加,相加结果送入 PC 中,实现程序转移执行,控制转移范围为-128~+127B;当参与比较的两个数据相等时,执行下一条指令,CJNE 指令执行流程以及影响程序状态字 PSW 的 CY 标志位都相同。

(6) CJNE @Ri, #data,rel 指令

指令机器码:

1 0 1 1 0 1 1 i
# data
rel(相对地址)

指令实现的操作: 首先程序计数器 PC 的内容加 3,然后将工作寄存器 Ri(R<sub>0</sub>~R<sub>1</sub>)之一中的数据作为内部 RAM 地址(间接寻址单元),并将 Ri 指示的内部 RAM 地址中的内容



与 ROM 中的立即数进行比较,当两者不相等时,将 PC 中的数值与 rel 数值按有符号数据相加,相加结果送入 PC 中,实现程序转移执行,控制转移范围为-128~+127B;当参与比较的两个数据相等时,执行下一条指令,CJNE 指令执行流程以及影响程序状态字 PSW 的 CY 标志位都相同。

3. 循环转移指令

(1) DJNZ Rn,rel 指令

指令机器码:	1 1 0 1 1 r r r
	rel(相对地址)

指令实现的操作: 首先程序计数器 PC 的内容加 2,然后将工作寄存器 Rn(R<sub>0</sub>~R<sub>7</sub>)之一的内容减 1,并对寄存器 Rn 的内容进行检测,当 Rn 的内容不为 0 时,满足转移条件,将 PC 中的数值与 rel 数值按有符号数据相加,相加结果送入 PC 中,实现程序转移执行,转移范围为-128~+127B;当 Rn 的内容为 0 时,CPU 执行该指令的下一条指令,CPU 执行该指令的流程如图 3-7 所示。

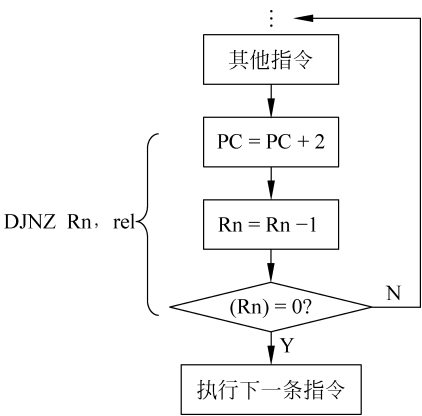


图 3-7 DJNZ 指令执行流程

(2) DJNZ direct,rel 指令

指令机器码:	1 1 0 1 0 1 0 1
	direct
	rel(相对地址)

指令实现的操作: 首先程序计数器 PC 的内容加 3,然后将 RAM 直接地址单元 direct 中的内容减 1,并对 direct 中的内容进行检测,当 direct 中的内容不为 0 时,满足转移条件,将 PC 中的数值与 rel 数值按有符号数据相加,相加结果送入 PC 中,实现程序转移执行,转移范围为-128~+127B;当 direct 中的内容为 0 时,CPU 执行该指令的下一条指令,CPU 执行 DJNZ 指令的流程是一样的,如图 3-7 所示。

4. 子程序调用指令

(1) ACALL addr11 指令

指令机器码:	a <sub>10</sub> a <sub>9</sub> a <sub>8</sub> 1 0 0 0 1
	a <sub>7</sub> ~ a <sub>0</sub>

指令实现的操作: 首先程序计数器 PC 的内容加 2,并将堆栈指针寄存器 SP 的内容加 1,PC<sub>7</sub>~PC<sub>0</sub> 低 8 位送入堆栈,SP 再加 1,PC<sub>15</sub>~PC<sub>8</sub> 高 8 位送入堆栈,然后将指令机器码中 a<sub>10</sub>~a<sub>0</sub> 送入 PC<sub>10</sub>~PC<sub>0</sub> 中,PC<sub>15</sub>~PC<sub>11</sub> 内容不变。

ACALL 指令在指令机器码中指出低 11 位程序存储器 ROM 的地址,该指令可控制程序在 2KB 范围内实现子程序的调用。

(2) LCALL addr16 指令

指令机器码:

0 0 0 1 0 0 1 0
a <sub>15</sub> ~ a <sub>8</sub>
a <sub>7</sub> ~ a <sub>0</sub>

指令实现的操作：首先程序计数器 PC 的内容加 3，并将堆栈指针寄存器 SP 的内容加 1，PC<sub>7</sub>~PC<sub>0</sub> 低 8 位送入堆栈，SP 再加 1，PC<sub>15</sub>~PC<sub>8</sub> 高 8 位送入堆栈，然后将指令机器码中 a<sub>15</sub>~a<sub>0</sub> 送入 PC<sub>15</sub>~PC<sub>0</sub> 中，该指令可控制程序在 64KB 范围内实现子程序的调用，另外，CALL 助记符与 LCALL 等同。

5. 返回指令

(1) RET 指令

指令机器码:

0 0 1 0 0 0 1 0
-----------------

指令实现的操作：子程序返回，将堆栈数据出栈送入 PC<sub>15</sub>~PC<sub>8</sub> 高 8 位，堆栈指针寄存器 SP 的内容减 1，再将堆栈数据出栈送入 PC<sub>7</sub>~PC<sub>0</sub> 低 8 位，SP 再减 1。

(2) RETI 指令

指令机器码:

0 0 1 1 0 0 1 0
-----------------

指令实现的操作：中断程序返回，将堆栈数据出栈送入 PC<sub>15</sub>~PC<sub>8</sub> 高 8 位，堆栈指针寄存器 SP 的内容减 1，再将堆栈数据出栈送入 PC<sub>7</sub>~PC<sub>0</sub> 低 8 位，SP 再减 1。另外，该指令还清除中断逻辑，例如，清除 CPU 内部中断标志。

6. 空操作指令

NOP 指令

指令机器码:

0 0 0 0 0 0 0 0
-----------------

指令实现的操作：空操作，程序计数器 PC 的内容加 1，该指令用于产生一个机器周期的延迟。

3.6 位操作、位控制转移类指令

位操作、位控制转移类指令是专门针对内部 RAM 字节地址为 20H~2FH(位地址 00H~7FH)的位存储单元以及特殊功能寄存器 SFR 中具有位访问能力的寄存器进行位操作而设计的，其操作的数据宽度是以位(1bit)为单位的，指令包括逻辑位运算、位数据传输、根据位状态实现程序的转移执行等操作指令。

3.6.1 位操作、位控制转移指令

51 单片机 CPU 共有 17 条位操作、位控制转移指令，这些指令的助记符、完成的功能、影响程序状态字 PSW 的标志位、占用存储器字节数、CPU 执行指令所用时间(机器周期)等如表 3-8 所示。

表 3-8 51 单片机 CPU 位操作、位控制转移指令表

机器码	助 记 符		功 能	影响 PSW 标志位				字节数	机器周期
	操作码	操作数		CY	AC	OV	P		
C3	CLR	C	$CY \leftarrow 0$	0	×	×	×	1	1
C2	CLR	bit	$bit \leftarrow 0$	×	×	×	×	2	1
D3	SETB	C	$CY \leftarrow 1$	1	×	×	×	1	1
D2	SETB	bit	$bit \leftarrow 1$	×	×	×	×	2	1
B3	CPL	C	$CY \leftarrow \overline{CY}$	√	×	×	×	1	1
B2	CPL	bit	$bit \leftarrow \overline{bit}$	×	×	×	×	2	1
82	ANL	C, bit	$CY \leftarrow CY \wedge bit$	√	×	×	×	2	2
B0	ANL	C, $\overline{bit}$	$CY \leftarrow CY \wedge \overline{bit}$	√	×	×	×	2	2
72	ORL	C, bit	$CY \leftarrow CY \vee bit$	√	×	×	×	2	2
A0	ORL	C, $\overline{bit}$	$CY \leftarrow CY \vee \overline{bit}$	√	×	×	×	2	2
A2	MOV	C, bit	$CY \leftarrow bit$	√	×	×	×	2	1
92	MOV	bit, C	$bit \leftarrow CY$	×	×	×	×	2	2
40	JC	rel	$PC \leftarrow PC + 2,$ if $CY = 1$ ; then $PC \leftarrow PC + rel$	×	×	×	×	2	2
50	JNC	rel	$PC \leftarrow PC + 2,$ if $CY \neq 1$ ; then $PC \leftarrow PC + rel$	×	×	×	×	2	2
20	JB	bit, rel	$PC \leftarrow PC + 3,$ if $bit = 1$ ; then $PC \leftarrow PC + rel$	×	×	×	×	3	2
30	JNB	bit, rel	$PC \leftarrow PC + 3,$ if $bit \neq 1$ ; then $PC \leftarrow PC + rel$	×	×	×	×	3	2
10	JBC	bit, rel	$PC \leftarrow PC + 3,$ if $bit = 1$ ; then $PC \leftarrow PC + rel, bit \leftarrow 0$	×	×	×	×	3	2

3.6.2 位操作、位控制转移指令详解

位操作、位控制转移指令有对某一位进行清 0 或置 1、位逻辑运算、位数据传输、根据位状态实现程序的转移执行等操作。

1. 位清 0、置 1 指令

(1) CLR C 指令

指令机器码：

1 1 0 0 0 0 1 1
-----------------

指令实现的操作：将程序状态字 PSW 中 CY 进位位清 0。

(2) CLR bit 指令

指令机器码：

1 1 0 0 0 0 1 0
bit(位地址)

指令实现的操作：将内部 RAM 位地址为 00H~7FH 之间的位地址为 bit 的位存储单元清 0。

(3) SETB C 指令

指令机器码：

1 1 0 1 0 0 1 1
-----------------

指令实现的操作：将程序状态字 PSW 中 CY 进位位置 1。

(4) SETB bit 指令

指令机器码：

1 1 0 1 0 0 1 0
bit(位地址)

指令实现的操作：将内部 RAM 位地址为 00H~7FH 之间的位地址为 bit 的位存储单元置 1。

2. 位逻辑运算指令

(1) CPL C 指令

指令机器码：

1 0 1 1 0 0 1 1
-----------------

指令实现的操作：将程序状态字 PSW 中 CY 进位位进行求反后送回 CY。

(2) CPL bit 指令

指令机器码：

1 0 1 1 0 0 1 0
bit(位地址)

指令实现的操作：将内部 RAM 位地址为 bit 位存储单元的内容进行求反后送回 bit 单元中。

(3) ANL C,bit 指令

指令机器码：

1 0 0 0 0 0 1 0
bit(位地址)

指令实现的操作：将内部 RAM 位地址为 bit 位存储单元的内容与程序状态字 PSW 中 CY 进位位进行“逻辑与”操作,将操作结果送入 CY 中。

(4) ANL C, $\overline{\text{bit}}$  指令

指令机器码：

1 0 1 1 0 0 0 0
bit(位地址)

指令实现的操作：将内部 RAM 位地址为 bit 位存储单元的内容求反后与程序状态字 PSW 中 CY 进位位进行“逻辑与”操作,将操作结果送入 CY 中。

(5) ORL C,bit 指令

指令机器码：

0 1 1 1 0 0 1 0
bit(位地址)

指令实现的操作：将内部 RAM 位地址为 bit 位存储单元的内容与程序状态字 PSW 中

CY 进位位进行“逻辑或”操作,将操作结果送入 CY 中。

(6) ORL C,bit 指令

指令机器码:	1 0 1 0 0 0 0 0
	bit(位地址)

指令实现的操作:将内部 RAM 位地址为 bit 位存储单元的内容求反后与程序状态字 PSW 中 CY 进位位进行“逻辑或”操作,将操作结果送入 CY 中。

3. 位传输指令

(1) MOV C,bit 指令

指令机器码:	1 0 1 0 0 0 1 0
	bit(位地址)

指令实现的操作:将内部 RAM 位地址为 bit 位存储单元的内容送入程序状态字 PSW 的 CY 进位位中。

(2) MOV bit,C 指令

指令机器码:	1 0 0 1 0 0 1 0
	bit(位地址)

指令实现的操作:将程序状态字 PSW 中 CY 进位位的内容送入内部 RAM 位地址为 bit 的位存储单元中。

4. 位判断转移指令

(1) JC rel 指令

指令机器码:	0 1 0 0 0 0 0 0
	rel(相对地址)

指令实现的操作:首先程序计数器 PC 的内容加 2,然后对程序状态字 PSW 中 CY 进位位的内容进行检测,当 CY 为 1 时,满足转移条件,将 PC 中的数值与 rel 数值按有符号数据相加,相加结果送入 PC 中,实现程序转移执行,该指令控制程序转移范围为-128~+127B;当 CY 为 0 时,CPU 执行该指令的下一条指令。

(2) JNC rel 指令

指令机器码:	0 1 0 1 0 0 0 0
	rel(相对地址)

指令实现的操作:首先程序计数器 PC 的内容加 2,然后对程序状态字 PSW 中 CY 进位位的内容进行检测,当 CY 不为 1 时,满足转移条件,将 PC 中的数值与 rel 数值按有符号数据相加,相加结果送入 PC 中,实现程序转移执行,该指令控制程序转移范围为-128~+127B;当 CY 为 1 时,CPU 执行该指令的下一条指令。

(3) JB bit,rel 指令

指令机器码:	0 0 1 0   0 0 0 0
	bit(位地址)
	rel(相对地址)

指令实现的操作：首先程序计数器 PC 的内容加 3，然后对内部 RAM 位地址为 bit 位存储单元的内容进行检测，当该单元内容为 1 时，将 PC 中的数值与 rel 数值按有符号数据相加，相加结果送入 PC 中，实现程序转移执行，该指令控制程序转移范围为 -128 ~ +127B；当 bit 单元内容为 0 时，CPU 执行该指令的下一条指令。

(4) JNB bit,rel 指令

指令机器码:	0 0 1 1   0 0 0 0
	bit(位地址)
	rel(相对地址)

指令实现的操作：首先程序计数器 PC 的内容加 3，然后对内部 RAM 位地址为 bit 位存储单元的内容进行检测，当该单元内容不为 1 时，将 PC 中的数值与 rel 数值按有符号数据相加，相加结果送入 PC 中，实现程序转移执行，该指令控制程序转移范围为 -128 ~ +127B；当 bit 单元内容为 1 时，CPU 执行该指令的下一条指令。

(5) JBC bit,rel 指令

指令机器码:	0 0 0 1   0 0 0 0
	bit(位地址)
	rel(相对地址)

指令实现的操作：首先程序计数器 PC 的内容加 3，然后对内部 RAM 位地址为 bit 位存储单元的内容进行检测，当该单元内容为 1 时，将 PC 中的数值与 rel 数值按有符号数据相加，相加结果送入 PC 中，实现程序转移执行，同时将 bit 位存储单元清 0；当 bit 单元内容为 0 时，CPU 执行该指令的下一条指令。

3.7 伪指令

伪指令是用于通知汇编器如何进行汇编源程序的指令，又被称为命令语句，在由指令助记符和伪指令组成的源程序代码中，伪指令语句除定义的具体数据要生成目标代码外，其他均没有对应的目标代码(或可执行的指令机器码)，伪指令的功能是由汇编器在汇编源程序时被实现的，并非在 CPU 运行指令机器码组成的机器码程序实现。另外，当硬件发生改变时，通过伪指令可使修改程序变得容易。

3.7.1 常用伪指令

51 汇编语言常用的伪指令及其功能如表 3-9 所示。

表 3-9 51 汇编语言常用伪指令表

助 记 符	功 能
ORG	汇编起始地址伪指令,定义指令机器码起始地址
END	汇编结束伪指令,通知汇编器不再汇编
EQU	等值定义伪指令,定义数据或存储器地址
DATA	地址数据赋值伪指令,定义(8 位或 16 位)存储器地址
DB,DW	程序存储器字节(字)数据类型定义伪指令,定义 ROM 中的字节(字)常量
DS	程序存储器地址保留量定义伪指令,在 ROM 中定义一段存储空间
BIT	位地址符号定义伪指令,定义位存储器单元地址

3.7.2 伪指令详解

伪指令用于指导汇编器对指令源程序进行汇编的,主要指明指令机器码放在程序存储器的什么位置、存储单元如何分配、定义存储单元地址、定义常量数据等。

1. ORG 伪指令

使用格式:

ORG 表达式

ORG 伪指令的功能是规定该伪指令后面的指令源程序汇编后生成机器代码存放在 ROM 中的起始地址,表达式可以是一个具体的数值,例如,16 位地址,也可以是包含变量名的表达式,而变量在汇编器汇编到该伪指令时已经是被赋予了实际整数数值的。

2. END 伪指令

使用格式:

END

END 伪指令的功能是通知汇编器停止汇编,在 END 伪指令后面的指令源程序不会被汇编成指令机器代码。

3. EQU 伪指令

使用格式:

符号名称 EQU 表达式

EQU 伪指令也被称为等值伪指令,它用来说明“符号名称”等于一个表达式得出的值,表达式的值可以是 16 位或 8 位的二进制数值,也可以是字符串,当表达式的值是字符串时,只取后两个字符,“符号名称”是以字母开头的字母和数字的组合,在源程序中“符号名称”不可出现重名,使用 EQU 赋值的“符号名称”可作为数据(立即数)、数据地址、代码地址、位地址等使用。

4. DATA 伪指令

使用格式:

符号名称 DATA 表达式

DATA 伪指令也被称为数据地址赋值伪指令,该伪指令用来定义 RAM 中一个字节类型的存储单元,即赋予一个字节类型的存储单元一个“符号名称”,以便在指令源程序中通过“符号名称”来访问(读、写)这个存储单元,“符号名称”的组成规则与 EQU 伪指令中的“符号名称”组成规则相同。

### 5. DB、DW 伪指令

使用格式:

[标号:] DB 表达式 1,表达式 2, ..., 表达式 n

使用格式:

[标号:] DW 表达式 1,表达式 2, ..., 表达式 n

DB 伪指令用于定义一个连续的字节存储区(在程序存储 ROM 区域),并给该存储区的字节存储单元赋予数值,表达式的值为一个字节类型的数据或字符,如果是字符需要括在单引号“'”中,每个数据或字符用逗号分开,该伪指令在使用时需要容纳在源程序代码的一行内,即不能换行,在该伪指令使用格式中“标号:”是可选项。

DW 伪指令与 DB 伪指令功能相同,DW 伪指令是为字存储单元(在 ROM 区域)赋值的,规则为高 8 位存放在前(在 ROM 区域存储单元地址为 N),低 8 位存放在后(在 ROM 区域存储单元地址为 N+1)。

### 6. DS 伪指令

使用格式:

[标号:] DS 表达式 1,[表达式 2]

DS 伪指令用于定义一个长度为表达式 1 计算值的连续存储空间,表达式 1 得出的值是存储空间包含的字节数,即在 ROM 区域保留长度为表达式 1 数值个字节数的存储单元,表达式 2 是可选项,表达式 2 用于指定填满存储空间的数值。

### 7. BIT 伪指令

使用格式:

符号名称 BIT 表达式

BIT 伪指令也被称为位数据地址赋值伪指令,该伪指令用来定义一个位存储单元,即赋予位地址一个“符号名称”,以便在指令源程序中通过“符号名称”来访问(读、写)这个位存储单元,“符号名称”的组成规则与 EQU 伪指令中的“符号名称”组成规则相同。

## 3.8 指令程序

指令程序分为指令源代码程序和指令机器代码程序,指令源代码程序由指令助记符组成,包括指令机器码助记符和伪指令,它是用文本格式表述的;指令机器代码程序是由指令



机器码组成的,它是 51 单片机 CPU 可执行的程序,用二进制数据格式表述。

3.8.1 指令源代码程序格式

计算机程序的设计是在指令源代码的基础上实现的。源代码由机器码助记符或伪指令组成,每一条机器码指令(助记符)或伪指令被称为一条程序语句,或称为汇编语句,源代码程序是由多条程序语句组成的,它也被称为汇编语言程序,51 单片机汇编语言程序中程序语句的编写格式为




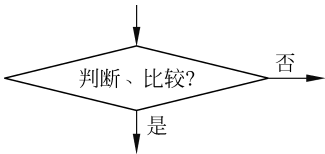
```
[标号:] 指令或伪指令  [; 注释]
```

指令即指机器码指令,包括操作码和操作数,按照每条指令的格式出现在程序语句中;伪指令同样也是按照伪指令的格式出现在程序语句中的,“标号:”是程序语句中的可选项,用于指明程序语句中指令或伪指令在程序存储 ROM 区域存放的存储单元地址,当指令机器码或伪指令中定义的数据占用多个存储单元时,标号指示的是首地址,“; 注释”也为可选项,为该条程序语句提供注释。

3.8.2 指令源代码程序设计

指令按照程序语句格式编写的多条语句的集合被称为源程序,程序的设计体现了安排 CPU 工作的过程,CPU 工作的步骤反映在程序设计上,遗憾的是,使用 51 单片机汇编语言设计程序只需遵守指令编写格式即可,并没有设计规则,其源程序只是汇编语句的堆积,每条汇编语句都指挥 CPU 完成一个动作,多条汇编语句的组合实现一个功能,而多条汇编语句组合的逻辑关系则需要人为确定,为了方便源程序的设计,建议使用逻辑框图来描述程序实现的功能,它也反映了 CPU 运行的流程,逻辑框图使用的图符如表 3-10 所示。

表 3-10 源程序设计经常使用的框图图符表

图 符	描 述
	程序开始执行和结束执行
	程序完成的单项任务
	程序执行流向指示
	程序中出现的判断分支

依据 51 单片机 CPU 指令系统提供的功能,其所能实现的程序设计有顺序程序设计、分支程序设计、循环程序设计、子程序设计等,其程序功能设计框图如图 3-8 所示。

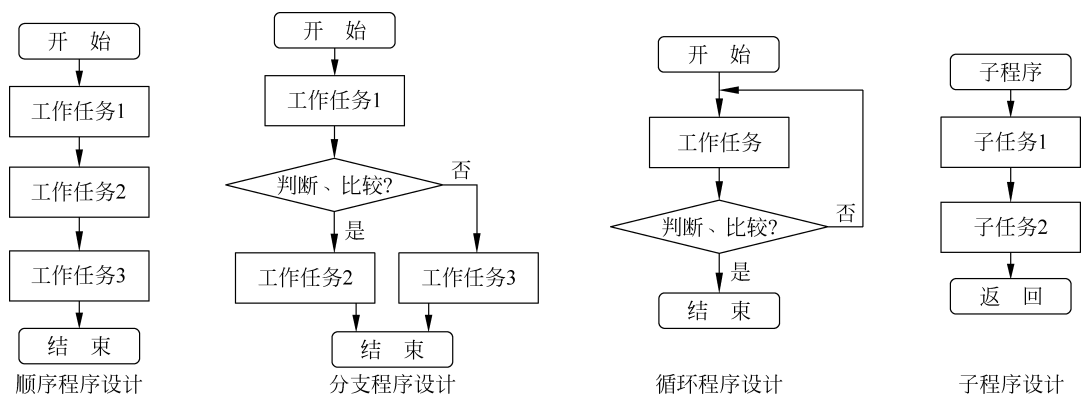


图 3-8 程序功能设计框图

3.8.3 源代码程序的编译

由汇编指令(助记符)和伪指令组成的源程序代码是不能在 51 单片机上运行的,需要将它编译成指令机器代码(可执行代码),编译的过程如图 3-9 所示,将指令机器代码写入 51 单片机的程序存储器 ROM 中才可以被执行。

源程序代码		编译	ROM_bin	地址	ROM_hex
ORG	0000H	→	0 0 0 0 0 0 1 0	0000H	02
LJMP	START		0 0 0 0 0 0 0 0	0001H	00
			0 0 1 1 0 0 0 0	0002H	30
			⋮	⋮	⋮
ORG	0030H	→	1 1 1 0 1 0 0 0	0030H	E8
START: MOV	A, R0	→	0 0 0 0 0 0 0 0	0031H	00
LOOP: NOP		→	1 0 0 0 0 0 0 0	0032H	80
SJMP	LOOP	→	1 1 1 1 1 1 0 1	0033H	FD
			0 0 0 0 0 0 0 1	0034H	01
DB	01H, 02H	→	0 0 0 0 0 0 1 0	0035H	02
END			⋮	⋮	⋮

图 3-9 源程序代码编译为指令机器代码

目前,51 单片机的生产厂商提供了多种编译工具,尤其是应用于 PC 的编译工具被广泛使用,例如 C51(Compiler51)汇编器等编译工具。在 PC 上编写 51 单片机的应用程序需要如下步骤。

- (1) 编码: 编写文本格式的源程序代码,并以 \*.asm 为文件名保存在磁盘中;

(2) 编译：使用编译工具将源程序代码编译成目标代码(由机器代码和重定位信息组成)或指令机器代码,有些汇编器输出的是目标代码文件,文件名为\*.obj,需要使用链接器将目标代码文件转换成指令机器代码,有些编译工具可直接产生指令机器代码,指令机器代码有两种文件格式,\*.bin(二进制可执行文件)和\*.hex(Intel 公司制定按地址排列的十六进制数据信息文件)。

(3) 执行：使用专用 51 单片机 ROM 数据文件写入设备将 \*.bin 或 \*.hex 文件写入 ROM 中,或应用 ISP(In System Programming)技术将可执行文件写入 51 单片机的 ROM 中。

3.8.4 源代码程序设计示例

通过设计汇编语言源代码程序可以熟悉和掌握 51 单片机指令的使用,下面是几个常用的汇编语言源代码程序的设计和程序执行流程框图。

1. 数据传送源程序

将内部 RAM 地址为 20H~2FH 单元中的数据按顺序传送到 70H~7FH 单元中,CPU 执行流程如图 3-10 所示,源程序代码如下。

```
ORG    0000H
LJMP   START      ;程序开始
ORG    0030H
START: MOV    R0, # 20H  ;初始化
      MOV    R1, # 70H
      MOV    R2, # 10H
LOOP:  MOV    A, @R0      ;移动数据
      MOV    @R1, A
      INC    R0            ;地址 + 1
      INC    R1
      DJNZ   R2, LOOP     ;计数器 - 1
      SJMP   $            ;结束,等待
END
```

说明：R<sub>0</sub> 存放源数据地址,R<sub>1</sub> 存放目的数据地址,R<sub>2</sub> 为一个计数器。

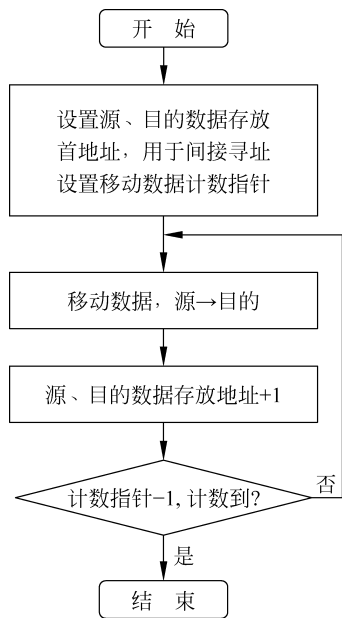


图 3-10 数据移动程序执行流程

2. 软件延时源程序

软件延时是经常使用的一种时间定时方法,当 51 单片机的时钟频率(振荡频率 f<sub>osc</sub>)确定后,每条指令的执行周期是确定的,根据指令的执行周期(包含的机器周期)可以计算出延时的时间,下面是一个软件延时子程序代码。

```
ORG    0100H
MOV    R7, # 00H      ;将 R7 清 0,执行该指令需要 1 个机器周期
MOV    R6, # 00H      ;将 R6 清 0,执行该指令需要 1 个机器周期
```

```
DELAY:  DJNZ    R7,DELAY    ;R7-1 非 0 转移,执行该指令需要 2 个机器周期
        DJNZ    R6,DELAY    ;R6-1 非 0 转移,执行该指令需要 12 个状态周期
        DJNZ    R5,DELAY    ;R5-1 非 0 转移,执行该指令需要 24 个振荡周期
        RET                ;子程序返回,执行该指令需要 2 个机器周期
```

说明:  $R_5$  为延时子程序的输入参数,改变  $R_5$  的内容可改变延时时间,当  $R_5$  内容不为 0,其 51 单片机的振荡频率为  $f_{osc}$  时,该子程序延时时间  $t$  的计算公式为

$$t = ((1+1+(2 \times 256 \times 256+2 \times 256) \times R_5+2 \times R_5+2) \times 12) \div f_{osc}$$

当  $R_5$  内容为 0 时,该延时子程序实现最大延时时间,在计算公式中  $R_5$  取值为 256。

3. 代码转换源程序

将存放在内部 RAM 地址为 30H 单元中的一个二进制整数转换成相应的 BCD 码,转换结果从高到低依次存放在 70H、71H、72H 单元中。

```
ORG      0000H
LJMP     START      ;程序开始
ORG      0030H
START:   MOV      R0, #70H    ;设置存放结果单元指针
        MOV      A, 30H      ;将二进制数据传送到 A
        MOV      B, #100     ;将除数 100 传送到 B
        DIV      AB          ;二进制数据除以 100
        MOV      @R0, A      ;将商存放到 70H(百位)单元中
        INC      R0          ;将存放结果单元指针加 1
        MOV      A, #10      ;将除数 10 传送到 A
        XCH      A, B        ;将被除数(余数)与除数互换位置
        DIV      AB          ;将余数再除以 10
        MOV      @R0, A      ;将商存放到 71H(十位)单元中
        INC      R0          ;将存放结果单元指针加 1
        MOV      @R0, B      ;将余数存放到 72H(个位)单元中
        SJMP     $           ;结束,等待中
END
```

说明: 转换方法为二进制整数除以 100 所得商为 BCD 码的百位(最高位),所得余数再除以 10 后,其商为 BCD 码的十位,余数为 BCD 码的个位(最低位)。

4. 多字节无符号数据相加源程序

存放于内部 RAM 中的两个等字节长度的数据做无符号数相加运算,CPU 相加流程如图 3-11 所示,通过加法子程序实现加运算操作,其源代码如下。

```
ORG      0100H
ADDITION: CLR      C
LOOP:    MOV      A, @R0
        ADDC     A, @R1
        MOV      @R0, A
```

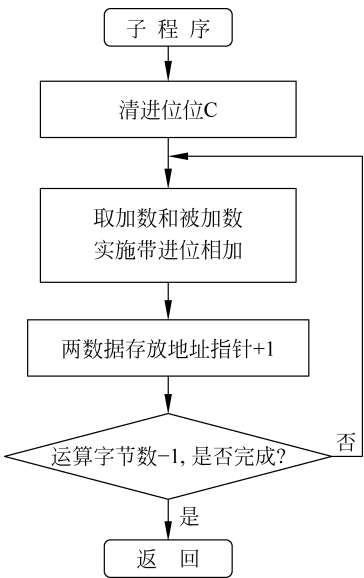


图 3-11 两数据相加流程

```
INC    R0
INC    R1
DJNZ   R2, LOOP
RET
```

说明：加法运算子程序的输入参数为  $R_0$ 、 $R_1$ 、 $R_2$ 。 $R_0$  为被加数存放在 RAM 中的地址指针，指向被加数据最低字节的存放位置，同时  $R_0$  也是相加结果存放在 RAM 中的地址指针，即相加运算完成后，被加数据存放的位置成为相加结果数据的存放位置； $R_1$  为加数存放在 RAM 中的地址指针，指向加数据最低字节的存放位置； $R_2$  为求和两数据所占的字节数，为子程序提供按字节数据相加的次数。

5. 多字节无符号数据相乘源程序

下面的子程序源代码实现两个无符号数据做相乘运算的操作。子程序的输入参数为  $R_4$ 、 $R_3$ 、 $R_2$ 。被乘数为 16 位数据，存放于  $R_4$ （高字节）、 $R_3$ （低字节）中；乘数为 8 位数据，存放于  $R_2$  中；子程序的输出参数为  $R_7$ 、 $R_6$ 、 $R_5$ ，存放相乘运算的结果； $R_7$  为最高字节， $R_6$  为次高字节， $R_5$  为最低字节。

```
ORG    0100H
MULTIPLICATION: MOV    A, R2      ;取乘数存放在 A
                MOV    B, R3      ;取被乘数低字节存放在 B
                MUL    AB         ;低字节相乘,第 1 次乘积
                MOV    R5, A       ;存放乘积的低字节数
                MOV    R6, B       ;存放乘积的高 8 位
                MOV    A, R2      ;取乘数存放在 A
                MOV    B, R4      ;取被乘数高字节存放在 B
                MUL    AB         ;高字节相乘,第 2 次乘积
                ADD    A, R6       ;第 1 次乘积的高位和第 2 次乘积的低位相加
                MOV    R6, A       ;存放乘积的次高字节
                MOV    A, B        ;乘积的高字节送 A
                ADDC   A, # 00H    ;与上次加法的进位相加
                MOV    R7, A       ;存放乘积的高字节
                RET                ;子程序返回
```