AN-8203
FCM8531 用户手册

所有指令兼容二进制代码，与业界标准的 8051 微控制器执行的功能相同。

表 1. 指令集和寻址模式

符号	说明
Rn	当前选择的寄存器区中的寄存器 R0 ~ R7
直接	内部 DATA RAM 位置 (0~127) 或特殊功能寄存器 (SFR)
@Ri	间接内部 (0~255) 或外部 RAM 位置由寄存器 R0 或 R1 寻址
#data	指令中包含的 8 位常数（立即操作数）
#data16	作为指令第 2 或 3 字节的 16 位常数（立即操作数）
addr16	LCALL 或 LJMP 的 16 位目的地址，可以是程序存储器地址空间 64-Kb 内的任何位置
addr11	ACALL 或 AJMP 的 11 位目的地址，程序存储器的相同 2-Kb 作为下一指令的第一字节
rel	信号 8 位偏移字节，用于 SJMP 和所有条件跳转。范围为 -128 ~ +127 字节，与下一指令的第一字节相对
位	内部数据 RAM 的直接寻址位，或位可寻址特殊功能寄存器，包括 I/O 引脚和状态字
A	累加器

每个指令的持续时间可采用下面的公式计算：

```
IF (BYTES > 1 or CYCLES = 1)
       DURATION = CYCLES + (BYTES + R) × P + X × D
else
       DURATION = CYCLES + (2 + R) × P + X × D
其中：
       BYTES 是指令的字节数
       CYCLES 是无等候状态的周期数
       对于 MOVC 指令 R = 1，否则 R = 0
       对于 MOVX 指令 X = 1，否则 X = 0
       P = 程序存储器等待状态数 (CKCON[4:6])
       D = 数据存储器等待状态数 (CKCON[2:0])
```

1. 指令集

1.1. 函数顺序指令

表 2. 函数顺序指令

符号	说明	代码	字节	周期
算术运算				
ADD A，Rn	向累加器加寄存器	0x28~0x2F	1	2
ADD A，direct	向累加器直接加寻址数据	0x25	2	3
ADD A，@Ri	向累加器间接加寻址数据	0x26~0x27	1	4
ADD A，#data	向累加器加即时数据	0x24	2	2
ADDC A，Rn	向累加器带进位加寄存器	0x38~0x3F	1	2
ADDC A，direct	向累加器直接带进位加寻址数据	0x35	2	3
ADDC A，@Ri	向累加器间接加寻址数据	0x36~0x37	1	4
ADDC A，#data	向累加器带进位加即时数据	0x34	2	2
SUBB A，Rn	从累加器带借位减寄存器	0x98~0x9F	1	2
SUBB A，direct	从累加器带借位减直接寻址数据	0X95	2	2
SUBB A，@Ri	从累加器间接带借位减寻址数据	0x96~0x97	1	4
SUBB A，#data	从累加器带借位减即时数据	0x94	2	2
INC A	递增累加器	0x04	1	1
INC Rn	递增寄存器	0x08~0x0F	1	3
INC direct	直接递增寻址数据	0x05	2	4
INC @Ri	间接递增寻址数据	0x06~0x07	1	5
INC DPTR	递增数据指针	0xA3	1	1
DEC A	递减累加器	0x14	1	1
DEC Rn	递减寄存器	0x18~0x1F	1	3
DEC direct	直接递减寻址数据	0x15	2	4
DEC @Ri	间接递减寻址数据	0x16~0x17	1	5
MUL AB	A 除以 B	0xA4	1	4
DIV AB	A 除以 B	0x84	1	4
DA A	十进制调整累加器	0xD4	1	1
逻辑运算				
ANL A，Rn	AND 寄存器至累加器	0x58~0x5F	1	2
ANL A，direct	AND 直接寻址数据至累加器	0x55	2	3
ANL A，@Ri	AND 间接寻址数据至累加器	0x56~0x57	1	4
ANL A，#data	AND 即时数据至累加器	0x54	2	2
ANL direct，A	AND 累加器至直接数据	0x52	2	4
ANL direct，#data	AND 即时数据至直接数据	0x53	3	4
ORL A，Rn	OR 寄存器至累加器	0x48~0x4F	1	2
ORL A，direct	OR 直接寻址数据至累加器	0x45	2	3
ORL A，@Ri	OR 间接寻址数据至累加器	0x46~0x47	1	4
ORL A，#data	OR 即时数据至累加器	0x44	2	2

符号	说明	代码	字节	周期
ORL direct，A	OR 累加器至直接寻址数据	0x42	2	4
ORL direct，#data	OR 即时数据至直接寻址数据	0x43	3	4
XRL A，Rn	Exclusive-OR 寄存器至累加器	0x68~0x6F	1	2
XRL A，direct	Exclusive-OR 直接寻址数据至累加器	0x65	2	3
XRL A，@Ri	Exclusive-OR 间接寻址数据至累加器	0x66~0x67	1	4
XRL A，#data	Exclusive-OR 即时数据至累加器	0x64	2	2
XRL direct，A	Exclusive OR 累加器至直接寻址数据	0x62	2	4
XRL direct，#data	Exclusive OR 即时数据至直接寻址数据	0x63	3	4
CLR A	清除累加器	0xE4	1	1
CPL A	补偿累加器	0xF4	1	1
RL A	左循环指令	0x23	1	1
RLC A	带进位左循环指令	0x33	1	1
RR A	右循环指令	0x03	1	1
RRC A	带进位右循环指令	0x13	1	1
SWAP A	累加器内交换半字节	0xC4	1	1
数据传送操作				
MOV A，Rn	寄存器移至累加器	0xE8~0xEF	1	1
MOV A，direct	直接寻址数据移至累加器	0xE5	2	3
MOV A，@Ri	间接寻址数据移至累加器	0xE6~0xE7	1	4
MOV A，#data	即时数据移至累加器	0x74	2	2
MOV Rn，A	累加器移至寄存器	0xF8~0xFF	1	1
MOV Rn，direct	直接寻址数据移至寄存器	0xA8~0xAF	2	4
MOV Rn，#data	即时数据移至寄存器	0x78~0x7F	2	2
MOV direct，A	累加器移至直接寻址位置	0xF5	2	2
MOV direct，Rn	寄存器移至直接寻址位置	0x88~0x8F	2	3
MOV direct1，direct2	直接寻址数据移至直接寻址位置	0x85	3	4
MOV direct，@Ri	间接寻址数据移至直接寻址位置	0x86~0x87	2	5
MOV direct，#data	即时数据移至直接寻址位置	0x75	3	3
MOV @Ri，A	累加器移至间接寻址位置	0xF6~0xF7	1	3
MOV @Ri，direct	直接寻址数据移至间接寻址位置	0xA6~0xA7	2	4
MOV @Ri，#data	即时数据移至间接寻址位置	0x76~0x77	2	3
MOV DPTR，#data16	用 16 位常数加载数据指针	0x90	3	3
MOVC A，@A+DPTR	用关于 DPTR 的代码字节加载累加器	0x93	1	4
MOVC A，@A+PC	用关于 PC 的代码字节加载累加器	0x83	1	4
MOVX A，@Ri(1)	外部 RAM（8 位地址）移至累加器	0xE2~0xE3	1	5~12
MOVX A，@DPTR(1)	外部 RAM（16 位地址）移至累加器	0xE0	1	4~11
MOVX @Ri，A(1)	累加器移至外部 RAM（8 位地址）	0xF2~0xF3	1	6~13
MOVX @DPTR，A(1)	累加器移至外部 RAM（16 位地址）	0xF0	1	5~12
PUSH direct	将直接寻址数据压入堆栈	0xC0	2	4
POP direct	从堆栈弹出直接寻址数据	0xD0	2	3
XCH A，Rn	交换寄存器与累加器	0xC8~0xCF	1	2

符号	说明	代码	字节	周期
XCH A，direct	交换直接寻址数据与累加器	0xC5	2	3
XCH A，@Ri	交换间接寻址数据与累加器	0xC6~0xC7	1	4
XCHD A，@Ri	交换间接寻址数据的低阶半字节与累加器	0xD6~0xD7	1	5
程序分支				
ACALL addr11	绝对子例程调用	xxx10001b	2	4
LCALL addr16	长子例程调用	0x12	3	4
RET	从子例程返回	0x22	1	5
RETI	从中断返回	0x32	1	5
AJMP addr11	绝对跳转	Xxx0001b	2	3
LJMP addr16	长跳转	0x02	3	4
SJMP rel	短跳转（相对地址）	0x80	2	3
JMP @A+DPTR	关于 DPTR 的间接跳转	0x73	1	3
JZ rel	如果累加器为零则跳转	0x60	2	3
JNZ rel	如果累加器非零则跳转	0x70	2	3
JC rel	如果进位标志置位则跳转	0x40	2	3
JNC rel	如果进位标志未置位则跳转	0x50	2	3
JB bit，rel	如果直接寻址位置位则跳转	0x20	3	5
JNB bit，rel	如果直接寻址位未置位则跳转	0x30	3	5
JBC bit，rel	如果直接寻址位置位则跳转，并清除位	0x10	3	5
CJNE A，direct,rel	比较直接寻址数据和累加器，不相等则跳转	0xB5	3	5
CJNE A，#data,rel	比较即时数据和累加器，不相等则跳转	0xB4	3	4
CJNE Rn，#data,rel	比较即时数据和寄存器，不相等则跳转	0xB8~0xBF	3	4
CJNE @Ri，#data,rel	比较即时数据和间接寻址数据，不相等则跳转	0xB6~0xB7	3	6
DJNZ Rn，rel	递减寄存器，非零则跳转	0xD8~0xDF	2	4
DJNZ direct，rel	递减直接寻址数据，非零则跳转	0xD5	3	5
NOP	无操作	00	1	1
布尔操作				
CLR C	清除进位标志	0xC3	1	1
CLR bit	清除直接寻址位	0xC2	2	4
SETB C	设置进位标志	0xD3	1	1
SETB bit	设置直接寻址位	0xD2	2	4
CPL C	补偿进位标志	0xB3	1	1
CPL bit	补偿直接寻址位	0xB2	2	4
ANL C，bit	AND 直接寻址位至进位标志	0x82	2	3
ANL C，bit	AND 直接寻址位补码至进位标志	0xB0	2	3
ORL C，bit	OR 直接寻址位至进位标志	0x72	2	3
ORL C，/bit	OR 直接寻址位补码至进位标志	0xA0	2	3
MOV C，bit	直接寻址位移至进位标志	0xA2	2	3
MOV bit，C	进位标志移至直接寻址位	0x92	2	4

注意：

1. MOVX 指令根据 “PMW” 位 (PCON.4) 的状态执行两种操作之一。请参考 AN-8202 — FCM8531 用户手册 - 硬件说明。

1.2. 十六进制顺序指令

表 3. 十六进制顺序指令

十六进制代码	字节	助记符	操作数	十六进制代码	字节	助记符	操作数
00	1	NOP		20	3	JB	位，代码地址
01	2	AJMP	code addr11	21	2	AJMP	code addr11
02	3	LJMP	code addr16	22	1	RET	
03	1	RR	A	23	1	RL	A
04	1	INC	A	24	2	ADD	A，#data
05	2	INC	直接	25	2	ADD	A，直接地址
06	1	INC	@R0	26	1	ADD	A，@R0
07	1	INC	@R1	27	1	ADD	A，@R1
08	1	INC	R0	28	1	ADD	A，R0
09	1	INC	R1	29	1	ADD	A，R1
0A	1	INC	R2	2A	1	ADD	A，R2
0B	1	INC	R3	2B	1	ADD	A，R3
0C	1	INC	R4	2C	1	ADD	A，R4
0D	1	INC	R5	2D	1	ADD	A，R5
0E	1	INC	R6	2E	1	ADD	A，R6
0F	1	INC	R7	2F	1	ADD	A，R7
10	3	JBC	位，代码地址	30	3	JNB	位，代码地址
11	2	ACALL	code addr11	31	2	ACALL	code addr11
12	3	LCALL	code addr16	32	1	RETI	
13	1	RRC	A	33	1	RLC	A
14	1	DEC	A	34	2	ADDC	A，#data
15	2	DEC	直接地址	35	2	ADDC	A，直接地址
16	1	DEC	@R0	36	1	ADDC	A，@R0
17	1	DEC	@R1	37	1	ADDC	A，@R1
18	1	DEC	R0	38	1	ADDC	A，R0
19	1	DEC	R1	39	1	ADDC	A，R1
1A	1	DEC	R2	3A	1	ADDC	A，R2
1B	1	DEC	R3	3B	1	ADDC	A，R3
1C	1	DEC	R4	3C	1	ADDC	A，R4
1D	1	DEC	R5	3D	1	ADDC	A，R5
1E	1	DEC	R6	3E	1	ADDC	A，R6
1F	1	DEC	R7	3F	1	ADDC	A，R7

十六进制顺序指令（续）

十六进制代码	字节	助记符	操作数	十六进制代码	字节	助记符	操作数
40	2	JC	代码地址	60	2	JZ	代码地址
41	2	AJMP	code addr11	61	2	AJMP	code addr11
42	2	ORL	直接地址，A	62	2	XRL	直接地址，A
43	3	ORL	直接地址，#data	63	3	XRL	直接地址，#data
44	2	ORL	A，#data	64	2	XRL	A，#data
45	2	ORL	A，直接地址	65	2	XRL	A，直接地址
46	1	ORL	A，@R0	66	1	XRL	A，@R0
47	1	ORL	A，@R1	67	1	XRL	A，@R1
48	1	ORL	A，R0	68	1	XRL	A，R0
49	1	ORL	A，R1	69	1	XRL	A，R1
4A	1	ORL	A，R2	6A	1	XRL	A，R2
4B	1	ORL	A，R3	6B	1	XRL	A，R3
4C	1	ORL	A，R4	6C	1	XRL	A，R4
4D	1	ORL	A，R5	6D	1	XRL	A，R5
4E	1	ORL	A，R6	6E	1	XRL	A，R6
4F	1	ORL	A，R7	6F	1	XRL	A，R7
50	2	JNC	代码地址	70	2	JNZ	代码地址
51	2	ACALL	code addr11	71	2	ACALL	code addr11
52	2	ANL	直接地址，A	72	2	ORL	C，位
53	3	ANL	直接地址，#data	73	1	JMP	@A+DPTR
54	2	ANL	A，#data	74	2	MOV	A，#data
55	2	ANL	A，直接地址	75	3	MOV	直接地址，#data
56	1	ANL	A，@R0	76	2	MOV	@R0，#data
57	1	ANL	A，@R1	77	2	MOV	@R1，#data
58	1	ANL	A，R0	78	2	MOV	R0，#data
59	1	ANL	A，R1	79	2	MOV	R1，#data
5A	1	ANL	A，R2	7A	2	MOV	R2，#data
5B	1	ANL	A，R3	7B	2	MOV	R3，#data
5C	1	ANL	A，R4	7C	2	MOV	R4，#data
5D	1	ANL	A，R5	7D	2	MOV	R5，#data
5E	1	ANL	A，R6	7E	2	MOV	R6，#data
5F	1	ANL	A，R7	7F	2	MOV	R7，#data

十六进制顺序指令（续）

80	2	SJMP	代码地址	A0	2	ORL	C，/位
81	2	AJMP	code addr11	A1	2	AJMP	code addr11
82	2	ANL	C，位	A2	2	MOV	C，位
83	1	MOVC	A，@A+PC	A3	1	INC	DPTR
84	1	DIV	AB	A4	1	MUL	AB
85	3	MOV	直接地址，直接地址	A5		-	
86	2	MOV	直接地址，@R0	A6	2	MOV	@R0，直接地址
87	2	MOV	直接地址，@R1	A7	2	MOV	@R1，直接地址
88	2	MOV	直接地址，R0	A8	2	MOV	R0，直接地址
89	2	MOV	直接地址，R1	A9	2	MOV	R1，直接地址
8A	2	MOV	直接地址，R2	AA	2	MOV	R2，直接地址
8B	2	MOV	直接地址，R3	AB	2	MOV	R3，直接地址
8C	2	MOV	直接地址，R4	AC	2	MOV	R4，直接地址
8D	2	MOV	直接地址，R5	AD	2	MOV	R5，直接地址
8E	2	MOV	直接地址，R6	AE	2	MOV	R6，直接地址
8F	2	MOV	直接地址，R7	AF	2	MOV	R7，直接地址
90	3	MOV	DPTR，#data16	B0	2	ANL	C，/位
91	2	ACALL	code addr11	B1	2	ACALL	code addr11
92	2	MOV	位，C	B2	2	CPL	位
93	1	MOVC	A，@A+DPTR	B3	1	CPL	C
94	2	SUBB	A，#data	B4	3	CJNE	A，#data，代码地址
95	2	SUBB	A，直接地址	B5	3	CJNE	A，直接，代码地址
96	1	SUBB	A，@R0	B6	3	CJNE	@R0，#data，代码地址
97	1	SUBB	A，@R1	B7	3	CJNE	@R1，#data，代码地址
98	1	SUBB	A，R0	B8	3	CJNE	R0，#data，代码地址
99	1	SUBB	A，R1	B9	3	CJNE	R1，#data，代码地址
9A	1	SUBB	A，R2	BA	3	CJNE	R2，#data，代码地址
9B	1	SUBB	A，R3	BB	3	CJNE	R3，#data，代码地址
9C	1	SUBB	A，R4	BC	3	CJNE	R4，#data，代码地址
9D	1	SUBB	A，R5	BD	3	CJNE	R5，#data，代码地址
9E	1	SUBB	A，R6	BE	3	CJNE	R6，#data，代码地址
9F	1	SUBB	A，R7	BF	3	CJNE	R7，#data，代码地址

十六进制顺序指令 （续）

十六进制代码	字节	助记符	操作数	十六进制代码	字节	助记符	操作数
C0	2	PUSH	直接地址	E0	1	MOVX	A，@DPTR
C1	2	AJMP	code addr11	E1	2	AJMP	code addr11
C2	2	CLR	位	E2	1	MOVX	A，@R0
C3	1	CLR	C	E3	1	MOVX	A，@R1
C4	1	SWAP	A	E4	1	CLR	A
C5	2	XCH	A，直接地址	E5	2	MOV	A，直接地址
C6	1	XCH	A，@R0	E6	1	MOV	A，@R0
C7	1	XCH	A，@R1	E7	1	MOV	A，@R1
C8	1	XCH	A，R0	E8	1	MOV	A，R0
C9	1	XCH	A，R1	E9	1	MOV	A，R1
CA	1	XCH	A，R2	EA	1	MOV	A，R2
CB	1	XCH	A，R3	EB	1	MOV	A，R3
CC	1	XCH	A，R4	EC	1	MOV	A，R4
CD	1	XCH	A，R5	ED	1	MOV	A，R5
CE	1	XCH	A，R6	EE	1	MOV	A，R6
CF	1	XCH	A，R7	EF	1	MOV	A，R7
D0	2	POP	直接地址	F0	1	MOVX	@DPTR，A
D1	2	ACALL	code addr11	F1	2	ACALL	code addr11
D2	2	SETB	位	F2	1	MOVX	@R0，A
D3	1	SETB	C	F3	1	MOVX	@R1，A
D4	1	DA	A	F4	1	CPL	A
D5	3	DJNZ	直接地址，代码地址	F5	2	MOV	直接地址，A
D6	1	XCHD	A，@R0	F6	1	MOV	@R0，A
D7	1	XCHD	A，@R1	F7	1	MOV	@R1，A
D8	2	DJNZ	R0，代码地址	F8	1	MOV	R0，A
D9	2	DJNZ	R1，代码地址	F9	1	MOV	R1，A
DA	2	DJNZ	R2，代码地址	FA	1	MOV	R2，A
DB	2	DJNZ	R3，代码地址	FB	1	MOV	R3，A
DC	2	DJNZ	R4，代码地址	FC	1	MOV	R4，A
DD	2	DJNZ	R5，代码地址	FD	1	MOV	R5，A
DE	2	DJNZ	R6，代码地址	FE	1	MOV	R6，A
DF	2	DJNZ	R7，代码地址	FF	1	MOV	R7，A

1.3. 指令定义

ACALL addr11

功能： 绝对调用（2 K 字节页以内）

说明： ACALL 无条件调用指定地址的子例程。该指令递增 PC 两次以获得下一指令的地址，然后将 16 位结果压入栈（低字节优先），然后递增 SP 两次。目的地址由递增后 PC 值的高阶位 (A15~A11)、操作码的 3 个高阶位 (A10~A8) 和指令第二字节 (A7~A0) 组合而成。

调用的子例程地址必须与 ACALL 之后的指令处于程序存储器的同一 2 K 块内。即，ACALL 指令后的 A16~A15 处于同一块内。

该指令不影响任何标志。

示例：
```
之前：SP = 20 h，PC = 0123 h，SUBROUTINE1 address = 0234 h
      ACALL  SUBROUTINE1
之后：SP = 22 h，PC = 0234 h，RAM(21 h) = 25 h，RAM(22 h) = 01 h
```

字节： 2

周期： 4

编码：

A10	A9	A8	1	0	0	0	1	A7	A6	A5	A4	A3	A2	A1	A0

运算：
```
ACALL
(PC) ← (PC) + 2
(SP) ← (SP) + 1
{SP} ← (PC7~0)
(SP) ← (SP) + 1
{SP} ← (PC15~8)
(PC10~0) ← A10~0（页面地址）
```

ADD A, < source-byte >

功能： 加法

说明： ADD 向累加器加指定字节。结果存储在累加器中。若位 7 或位 3 有进位，则进位标志或辅助进位标志设置为 1。否则清零。当该指令用于加总无符号整数时，进位标志指示溢出。如果位 6 有进位而位 7 没有，或者位 7 有进位而位 6 没有；OV 标志置位。其他情况下，OV 标志清零。当加总有符号整数时，OV 标志指示两个正操作数之和产生一个负数，或者两个负操作数之和产生一个正数。

该指令的源操作数允许四种寻址模式：寄存器、直接/间接寻址和即时值。

示例：
```
之前：累加器 = 80 h (10000000b)，R0 = AAh (10101010b)
      ADD    A，R0
之后：累加器 = 2 Ah(00101010b)，C = 1，OV = 1
```

ADD A,Rn

字节： 1

周期： 2

编码：

0	0	1	0	1	n	n	n

运算：
ADD
(A) ← (A) + (Rn)

ADD A,direct

字节： 2

周期： 3

编码：

0	0	1	0	0	1	0	1	直接地址

运算：
```
ADD
(A) ← (A) + (direct)
```

ADD A,@Ri

字节： 1

周期： 4

编码：

0	0	1	0	0	1	1	i

运算：

ADD
(A) ← (A) + {Ri}

ADD A,#data

字节：2

周期：2

编码：

0	0	1	0	0	1	0	0	即时数据

运算：

ADD
(A) ← (A) + data

ADDC A, < source-byte >

功能：带进位的加法

说明：ADDC 将指示字节、累加器和进位标志相加。结果存储在累加器中。若位 7 或位 3 有进位，则进位标志或辅助进位标志设置为 1。否则清零。如果该指令用于无符号整数的相加，进位标志指示发生溢出。如果位 6 有进位而位 7 没有，或者位 7 有进位而位 6 没有；OV 标志置位。其他情况下，OV 标志清零。当加总有符号整数时，OV 标志指示两个正操作数之和产生一个负数，或者两个负操作数之和产生一个正数。

该指令的源操作数允许四种寻址模式：寄存器、直接/间接寻址和即时值。

示例：

```
之前：累加器 = 80 h (10000000b)，R0 = AAh (10101010b)，C = 1
      ADDC   A，R0
之后：累加器 = 2 Bh(00101011b)，C = 1，OV = 1
```

ADDC A,Rn

字节：1

周期：2

编码：

0	0	1	1	1	n	n	n

运算：

ADDC
(A) ← (A) + (Rn) + (C)

ADDC A,direct

字节：2

周期：3

编码：

0	0	1	1	0	1	0	1	直接地址

运算：

ADDC
(A) ← (A) + (direct) + (C)

ADDC A,@Ri

字节：1

周期：4

编码：

0	0	1	1	0	1	1	i

运算：

ADDC
(A) ← (A) + {Ri} + (C)

ADDC A,#data

字节：2

周期：2

编码：

0	0	1	1	0	1	0	0	即时数据

运算：

ADD
(A) ← (A) + data + (C)

AJMP addr11

功能：绝对跳转（2 Kb 页以内）

说明： AJMP 是无条件跳转到指定地址。目的地址在执行过程中产生，由两次递增后的 PC 高阶五位 (A15~A11)、操作码的高阶三位 (A10~A8) 和第二字节值 (A7~A0) 组成而成。目的地址必须与 AJMP 之后的指令处于同一 2 K 程序存储块中。即，AJMP 后指令的 A16~A15 相同。
该指令不影响标志。

示例：
```
之前：PC = 0123 h，LABLE1 位地址= 0234 h
      AJMP   LABLE1
之后：PC = 0234 h
```

字节： 2

周期： 3

编码：

A10	A9	A8	0	0	0	0	1	A7	A6	A5	A4	A3	A2	A1	A0

运算：
```
AJMP
(PC) ← (PC) + 2
(PC10~0) ← A10~0 （页面地址）
```

ANL <destination-byte>, <source-byte>

功能： 用于字节变量的逻辑 AND

说明： ANL 指令让两个指定操作数执行逻辑 AND 运算，并将结果存储在目的地字节内。该指令不影响任何标志。该指令的操作数允许六 (6) 种寻址模式。当目标操作数为累加器时，源操作数可以是即时值、寄存器、直接或间接寻址字节。当目标操作数是直接地址字节时，源操作数可以是即时值或累加器。当该指令用于修改 I/O 端口时，用作初始端口数据的值从输出数据锁存器读取，而非输入引脚。

示例：
```
之前：累加器 = CAh (11001010b)，R0 = 55 h (01010101b)
      ANL    A，R0
之后：累加器 = 40 h (01000000b)
```

ANL A,Rn

字节： 1

周期： 2

编码：

0	1	0	1	1	n	n	n

运算：
```
ANL
(A) ← (A) AND (Rn)
```

ANL A，direct

字节： 2

周期： 3

编码：

0	1	0	1	0	1	0	1	直接地址

运算：
```
ANL
(A) ← (A) AND (direct)
```

ANL A,@Ri

字节： 1

周期： 4

编码：

0	1	0	1	0	1	1	i

运算：
```
ANL
(A) ← (A) AND {Ri}
```

ANL A,#data

字节： 2

周期： 2

编码：

0	1	0	1	0	1	0	0	即时数据

运算：
```
ANL
(A) ← (A) AND data
```

ANL direct,A

字节： 2

周期： 4

编码：

0	1	0	1	0	0	1	0	直接地址

运算：

```
ANL
(direct) ← (direct) AND (A)
```

ANL direct,#data

字节：3

周期：4

编码：

0	1	0	1	0	0	1	1	直接地址	即时数据

运算：

```
ANL
(direct) ← (direct) AND data
```

ANL C, <source-bit>

功能：对位变量进行逻辑与运算

说明：在指示位和进位标志上执行逻辑 AND 运算，结果存储在进位标志内。如果源位的布尔值为逻辑 0，则清除进位标志。否则，进位标志保持初始状态。操作数前面的斜线 ('/') 指示将操作数的逻辑补码用作源值，且不影响源位。除 C 标志外，该指令不影响其他标志。源位必须使用直接位寻址模式。

示例：

```
之前：P1.4 = 1，C = 1
      ANL   C，P1.4
之后：P1.4 = 1，C = 1
      ANL   C，/P14
之后：P1.4 = 1，C = 0
```

ANL C,bit

字节：2

周期：3

编码：

1	0	0	0	0	0	1	0	位地址

运算：

```
ANL
(C) ← (C) AND (bit)
```

ANL C,/bit

字节：2

周期：3

编码：

1	0	1	1	0	0	0	0	位地址

运算：

```
ANL
(C) ← (C) AND NOT (bit)
```

CJNE < destination-byte>, <source-byte>, rel

功能：比较，如果不相等则跳转

说明：比较源字节与该指令中指定的目的地字节。如果不相等，则分行至指示的相对地址。PC 值填充指令中的末位字节（有符号的相对位移）与指令后的 PC 值之和。如果目的地字节的无符号整数值小于源字节的对应值，则 C 标志置位。否则清除 C 标志。该指令不会更改这两个操作数。

该指令中，前两个操作数有四种寻址模式。累加器可以与任何直接寻址字节或即时值进行比较。寄存器或间接寻址字节可以与即时值进行比较。

示例：

```
之前：累加器 = 56 h
      HOLDING:    CJNE   A，P1，HOLDING
之后：如果端口 1 的值不等于 56 h，程序停止于 HOLDING，直到端口 1 = 56 h。
```

CJNE A，direct，rel

字节：3

周期：5

编码：

1	0	1	1	0	1	0	1	直接地址	rel. 地址

运算：

```
(PC) ← (PC) + 3
IF (A) <> (direct) THEN
(PC) ← (PC) + rel. address
```

```
IF (A) < (direct) THEN
(C) ← 1=
ELSE
(C) ← 0
```

CJNE A，#data，rel

字节： 3

周期： 4

编码：

1	0	1	1	0	1	0	0	即时数据	rel. 地址

运算：

```
(PC) ← (PC) + 3
IF (A) <> data THEN
(PC) ← (PC) + rel. address
IF (A) < data THEN
(C) ← 1
ELSE
(C) ← 0
```

CJNE Rn，#data，rel

字节： 3

周期： 4

编码：

1	0	1	1	1	n	n	n	即时数据	rel. 地址

运算：

```
(PC) ← (PC) + 3
IF (Rn) <> data THEN
(PC) ← (PC) + rel. address
IF (Rn) < data THEN
(C) ← 1
ELSE
(C) ← 0
```

CJNE @Ri，#data，rel

字节： 3

周期： 6

编码：

1	0	1	1	0	1	1	i	即时数据	rel. 地址

运算：

```
(PC) ← (PC) + 3
IF {Ri} <> data THEN
(PC) ← (PC) + rel. Address
IF {Ri} < data THEN
(C) ← 1
ELSE
(C) ← 0
```

CLR A

功能： 清除累加器

说明： 累加器的所有位清零。该指令不影响标志。

示例：

```
之前：累加器 = 56 h
      CLR    A
之后：累加器 = 00 h
```

字节： 1

周期： 1

编码：

1	1	1	0	0	1	0	0

运算：

```
CLR
(A) ← 0
```

CLR bit

功能： 清除位

说明： 将直接寻址位清零。除间接位外，该指令不影响任何标志。

示例：
```
之前：P1 = 56 h (01010110b)
      CLR   P1.2
之后：P1 = 52 h (01010010b)
```

CLR C

字节：1

周期：1

编码：

1	1	0	0	0	0	1	1

运算：
```
CLR
(C) ← 0
```

CLR bit

字节：2

周期：4

编码：

1	1	0	0	0	0	1	0	位地址

运算：
```
CLR
(bit) ← 0
```

CPL A

功能：补偿累加器

说明：累加器的所有位设置为一的补码。
（将最初为 1 的位更改为 0，将最初为 0 的位更改为 1。） 该指令不影响标志。

示例：
```
之前：累加器 = 56 h (01010110b)
      CPL   A
之后：累加器 = A9 h (10101001b)
```

字节：1

周期：1

编码：

1	1	1	1	0	1	0	0

运算：
```
CPL
(A) ←NOT (A)
```

CPL bit

功能：补偿位

说明：将直接寻址位设置为一的补码。
（将最初为 1 的位更改为 0，将最初为 0 的位更改为 1。）除指定位外，该指令不影响任何标志。

示例：
```
之前：P1 = 56 h (01010110b)
      CPL P1.0
      CPL P1.1
之后：P1 = 55 h (01010101b)
```

CPL C

字节：1

周期：1

编码：

1	0	1	1	0	0	1	1

运算：
```
CPL
(C) ← NOT (C)
```

CPL bit

字节：2

周期：4

编码：

1	0	1	1	0	0	1	0	位地址

运算：
```
CPL
(bit) ← NOT (bit)
```

DA A

功能： 十进制调整累加器以执行加法

说明： 调整累加器中两个变量的先前加法产生的值（BCD 格式），产生两个 4 位。任何 ADD 或 ADC 指令均可用于此种加法运算。如果累加器中的低阶字节的位 3~0 大于 9 (xxxx1010~xxxx1111) 或者 AC 标志已经置位，则累加器递增 6，从而在低阶半字节中产生合适的 BCD。如果内部加法产生进位，则进位标志置位。即使没有进位，也不会清除初始状态。如果进位标志已经置位或者高阶位已经超过 9 (1010xxxx~1111xxxx)，这些高阶半字节加 6。如果发生进位，则进位标志置位。即使没有进位，也不会清除初始状态。进位标志可表达初始两位 BCD 变量值是否超过 100，以处理更精确的大数系统加法运算。OV 标志不受影响。以上所有任务发生在一个指令周期内。

该指令根据累加器和 PSW 设置，将累加器 00 h、06 h、60 h 和 66 h 加总，以执行 BCD 转换。如果累加器的初始值是十六进制数，指令 DA A 不能直接将其转换为 BCD 符号。此外，DA A 不能用于十进制减法运算。

示例：
```
之前：累加器 = 56 h (01010110b)，R0 = 78 h (01111000b)
      ADD    A,R0
之后：累加器 = CEh (11001111b)
      DA      A
之后：累加器 = 34 h (00110100b)，C = 1，AC = 1
```

字节： 1

周期： 1

编码：

1	1	0	1	0	1	0	0

运算：
```
DA
-累加器的内容为 BCD
IF {A3-0) > 9 OR (AC) = 1) THEN
(A) ← (A) + 6
IF {A7-4) > 9 OR (C) = 1) THEN
(A) ← (A) + 60h
```

DEC Byte

功能： 递减

说明： DEC 将指定字节递减 1。如果初始值为 00 h，则由于下溢变为 FFh。
该指令不影响其他标志。允许四种操作数寻址模式：累加器、寄存器、直接/间接寻址。当该指令用于修改输出端口时，用作初始端口数据的值从输出数据锁存器读取，而非输入引脚。

示例：
```
之前：R0 = AAh (10101010b)
      DEC    R0
之后：R0 = A9h(00101001b)
```

DEC A

字节： 1

周期： 1

编码：

0	0	0	1	0	1	0	0

运算：
```
DEC
(A) ← (A) – 1
```

DEC Rn

字节： 1

周期： 3

编码：

0	0	0	1	1	n	n	n

运算：
```
DEC
(Rn) ← (Rn) – 1
```

DEC direct

字节： 2

周期： 4

编码：

0	0	0	1	0	1	0	1	直接地址

运算：

```
DEC
(direct)←(direct) – 1
```

DEC @Ri

字节： 1

周期： 5

编码：

0	0	0	1	0	1	1	i

运算：

```
DEC
{Ri} ← {Ri} – 1
```

DIV AB

功能： 除法

说明： DIV AB 将累加器中的无符号 8 位整数除以寄存器 B 中的无符号 8 位整数。商存储回累加器，余数存储在寄存器 B 中。

执行指令之后，C 标志和 OV 标志清零。如果寄存器 B 中的初始值为 0，则存储在累加器和寄存器 B 中的值不确定，OV 标志置位。C 标志清零。

示例：

```
之前：A = FAh (11111010b)，B = 26 h (00100110b)
      DIV   AB
之后：A = 06 h (00000110b)，B = 16 h (00010110b)，C = 0，OV = 0
```

字节： 1

周期： 4

编码：

1	0	0	0	0	1	0	0

运算：

```
DIV
(A) (B) ← (A) / (B)
A：商，B：余数
```

DJNZ <byte>, <rel-addr>

功能： 递减，若非零则跳转

说明： 递减指定字节的值，结果存储在初始位置。如果结果不等于 0，分行至指定相对地址，然后继续执行。此时，PC 值填充第二操作数（有符号的相对位移）和当前 PC 值之和。如果初始值是 00 h，递减后变为 FFh。该指令不影响标志。要递减的地址可以是寄存器或直接寻址字节。如果递减源是 I/O 端口，初始数据从输出数据锁存器获取，而非输入引脚。

示例：

```
之前：R5 = 02 h (00000010b)，LABLE0 地址 = 1234 h
      DJNZ   R5，LABLE0
之后：R5 = 01 h (00000001b)，PC = 1234 h
```

DJNZ Rn，rel

字节： 2

周期： 4

编码：

1	1	0	1	1	n	n	n	rel. 地址

运算：

```
DJNZ
(PC) ← (PC) + 2
(Rn) ← (Rn) – 1
IF (Rn) ≠ 0 THEN
(PC) ← (PC) + rel. 地址
```

DJNZ Direct，rel

字节： 3

周期： 5

编码：

1	1	0	1	0	1	0	1	直接地址	rel. 地址

运算：

```
DJNZ
(PC) ← (PC) + 2
(direct) ← (direct) – 1
IF(direct) ≠ 0 THEN
(PC) ← (PC) + rel. 地址
```

INC <byte>

功能：递增

说明：递增指定字节的值，结果存储在初始位置。如果初始值为 FFh，递增后变为 00 h。该指令不影响标志。该指令允许 4 种寻址模式：累加器、寄存器或直接/间接寻址。如果递减源是 I/O 端口，初始数据从输出数据锁存器获取，而非输入引脚。

示例：

```
之前：R0 = 10 h (00010000b)
      INC    R0
之后：R0 = 11 h (00010001b)
```

INC A

字节：1

周期：1

编码：

0	0	0	0	0	1	0	0

运算：

```
INC
(A) ← (A) + 1
```

INC Rn

字节：1

周期：3

编码：

0	0	0	0	1	n	n	n

运算：

```
INC
(Rn) ← (Rn) + 1
```

INC direct

字节：2

周期：4

编码：

0	0	0	0	0	1	0	1

运算：

```
INC
(direct) ← (direct) + 1
```

INC @Ri

字节：1

周期：5

编码：

0	0	0	0	0	1	1	i

运算：

```
INC
{Ri} ← {Ri} + 1
```

INC DPTR

功能：递增数据指针

说明：递增指定数据指针的值，结果存储在初始位置。执行 16 位递增后，如果数据指针的低阶字节 (DPL) 从 FFh 递增至 00 h，则发生溢出。同时递增高阶字节 (DPH)。该指令不影响标志。

示例：

```
之前：DPH = 03h，DPL = FEh
       INC    DPTR
之后：DPH = 03h，DPL = FFh
       INC    DPTR
之后：DPH = 04h，DPL = 00h
       INC    DPTR
之后：DPH = 04h，DPL = 01h
```

字节： 1

周期： 1

编码：

1	0	1	0	0	0	1	1

运算：

```
INC
(DPTR) ← (DPTR) + 1
```

JB bit, rel

功能： 如果位置位则跳转

说明： 如果指定位为 1，分行至相对地址；否则，继续处理下一个指令。发生跳转时，PC 值填充第三操作数（有符号的相对位移）和当前 PC 值之和。执行后，位状态改变，不影响标志。

示例：

```
之前：A = 55 h(01010101b)，P1 = AAh (10101010b)
      JB    P1.0，LABLE0
      JB    ACC.2，LABLE1
之后：A = 55 h(01010101b)，P1.0 = AAh (10101010b)，PC = LABLE1 位地址。
```

字节： 3

周期： 5

编码：

0	0	1	0	0	0	0	0	位地址	rel. 地址

运算：

```
JBC
(PC) ← (PC) + 3
IF(bit) = 1 THEN
(PC) ← (PC) + rel. 地址
```

JBC bit, rel

功能： 如果位置位则跳转，并清除位

说明： 如果指定位为 1，分行至相对地址并清除该位；否则，继续处理下一个指令。发生跳转时，PC 值填充第三操作数（有符号的相对位移）和当前 PC 值之和。该指令不影响标志。如果指定源是 I/O 端口，初始数据从输出数据锁存器获取，而非输入引脚。

示例：

```
之前：A = 55 h (01010101b)，P1 = AAh (10101010b)
      JBC   P1.0，LABLE0
      JBC   ACC.2，LABLE1
之后：A = 51 h (01010001b)，P1.0 = AAh (10101010b)，PC = LABLE1 位地址。
```

字节： 3

周期： 5

编码：

0	0	0	1	0	0	0	0	位地址	rel. 地址

运算：

```
JBC
(PC) ← (PC) + 3
IF(bit) = 1 THEN
(bit) ← 0
(PC) ← (PC) + rel. 地址
```

JC rel

功能： 如进位置位则跳转

说明： 如果进位标志置位，分行至相对地址并执行；否则，继续处理下一个指令。发生跳转时，PC 值填充第三操作数（有符号的相对位移）和当前 PC 值之和。该指令不影响标志。

示例：

```
之前：C = 0
      JC    C，LABLE0
      CPL   C
      JC    C，LABLE1
之后：C = 1，PC = LABLE1 位地址。
```

字节： 2

周期： 3

编码：

0	1	0	0	0	0	0	0	rel. 地址

运算：

```
JC
(PC) ← (PC) + 2
IF (C) = 1 THEN
(PC) ← (PC) + rel. 地址
```

JMP @A+DPTR

功能： 间接跳转

说明： 间接跳转至数据指针和累加器之和代表的地址。在累加器中的 8 位无符号数值和 16 位数据指针相加期间，如果低字节产生进位，则传输至高字节。结果填充到 PC 中，该地址用于随后的指令取出。执行后，累加器和数据指针都无变化。该指令不影响标志。

示例：

```
当：A = 04 h
        MOV    DPTR，#LABEL0
        JMP    @A+DPTR
        LABLE0:       AJMP   SUB0
        LABLE1:       AJMP   SUB1
        LABLE2:       AJMP   SUB2
        LABLE3:       AJMP   SUB3
之后：A = 04h，PC = LABLE2 位地址。
下一个运行 AJMP SUB2 的命令。（由于 AJMP 是 2 字节命令）
```

字节： 1

周期： 3

编码：

0	1	1	1	0	0	1	1

运算：

```
JMC
(PC) ← (A) + (DPTR)
```

JNB bit, rel

功能： 如果位未置位则跳转

说明： 如果指示位为 0，分行至相对地址并执行；否则，继续处理下一个指令。发生跳转时，PC 值填充第三操作数（有符号的相对位移）和当前 PC 值之和。执行后，不会修改该位。该指令不影响标志。

示例：

```
之前：A = 55 h (01010101b)，P1 = AAh (10101010b)
        JNB    P1.0，LABLE0
        JNB    ACC.2，LABLE1
之后：A = 55h (01010101b)，P1.0 = AAh (10101010b)，PC = LABLE0 位地址。
```

字节： 3

周期： 5

编码：

0	0	1	1	0	0	0	0	位地址	rel. 地址

运算：

```
JNB
(PC) ← (PC) + 3
IF(bit) = 0 THEN
(PC) ← (PC) + rel. 地址
```

JNC rel

功能： 如果进位未置位则跳转

说明： 如果进位标志为 0，分行至相对地址并执行；否则，继续处理下一个指令。发生跳转时，PC 值填充第三操作数（有符号的相对位移）和当前 PC 值之和。该指令不影响标志。

示例：

```
之前：C = 1
        JNC    C，LABLE0
        CPL    C
        JNC    C，LABLE1
之后：C = 0，PC = LABLE1 位地址。
```

字节： 2

周期： 3

编码：

0	1	0	1	0	0	0	0	rel. 地址

运算：

```
JC
(PC) ← (PC) + 2
IF (C) = 0 THEN
(PC) ← (PC) + rel. 地址
```

JNZ rel

功能： 如果累加器非零则跳转。

说明： 如果累加器的内容非 0，分行至相对地址并执行；否则，继续处理下一个指令。发生跳转时，PC 值填充第三操作数（有符号的相对位移）和当前 PC 值之和。累加器无变化。该指令不影响标志。

示例：

```
之前：A = 00 h
      JNZ    LABLE0
      INC    A
      JNZ    LABLE1
之后：A = 01h，PC = LABLE1 位地址。
```

字节： 2

周期： 3

编码：

0	1	1	1	0	0	0	0	rel. 地址

运算：

```
JNZ
(PC) ← (PC) + 2
IF (A) ≠ 0 THEN
(PC) ← (PC) + rel. 地址
```

JZ rel

功能： 如果累加器为零则跳转

说明： 如果累加器的内容为 0，分行至相对地址并执行；否则，继续处理下一个指令。发生跳转时，PC 值填充第三操作数（有符号的相对位移）和当前 PC 值之和。累加器无变化。该指令不影响标志。

示例：

```
之前：A = 01 h
      JZ     LABLE0
      DEC    A
      JZ     LABLE1
之后：A = 00h，PC = LABLE1 位地址。
```

字节： 2

周期： 3

编码：

0	1	1	0	0	0	0	0	rel. 地址

运算：

JZ
(PC) ← (PC) + 2
IF (A) = 0 THEN
(PC) ← (PC) + rel. 地址

LCALL addr16

功能： 长调用

说明： LCALL 无条件调用指定地址的子例程。该指令将 PC 递增 3，以产生用于执行下一个指令的地址。然后将 16 位 PC 值压入堆栈（低字节优先），同时 SP 递增两次。将指令的第二字节和第三字节分别加载到 PC 的高阶字节 (A15~A8) 和低阶字节 (A7~A0)。程序在该地址继续执行指令。因此，程序可始于全 64 Kb 程序存储地址空间的任何地址。该指令不影响标志。

示例：

```
之前：SP = 20 h，PC = 0123 h，SUBROUTINE1 位地址 = 1234 h
      LCALL  SUBROUTINE1
之后：SP = 22 h，PC = 1234 h，RAM(21h) = 26 h，RAM(22h) = 01 h
```

字节： 3

周期： 4

编码：

0	0	0	1	0	0	1	0	A15~A8	A7~A0

运算：

```
LCALL
(PC) ← (PC) + 3
(SP) ← (SP) + 1
{SP} ← (PC7~0)
(SP) ← (SP) + 1
{SP} ← (PC15~8)
(PC) ← A15~0
```

LJMP addr16

功能： 长跳转

说明： LJUMP 无条件分行至指示地址以执行程序。把指令的第二字节和第三字节分别加载到 PC 的高阶 (A15~A8) 和低阶 (A7~A0) 字节。程序在该地址继续执行指令。目的地可以是全 64 Kb 程序存储空间的任何地址。该指令不影响标志。

示例：

```
之前：PC = 0123 h，LABLE0 位地址 = 1234 h
      LJMP   LABLE0
之后：PC = 1234 h
```

字节： 3

周期： 4

编码：

0	0	0	0	0	0	1	0	A15~A8	A7~A0

运算：

```
LJMP
(PC) ← A15~0
```

MOV <destination-byte>, <source-byte>

功能： 传送字节变量

说明： 将第二操作数指示的字节变量复制到第一操作数指定的位置。源操作数值不会更改。该指令不影响寄存器或标志。源和目的地寻址模式允许 15 种组合。

示例：

```
之前：累加器 = FFh，RAM(40 h) = 30 h，P1 = 40 h
      MOV    R0，#30H
      MOV    @R0，P1
      MOV    A，@R0
      MOV    R1，A
      MOV    P2，P1
之后：累加器 = 40 h，R0 = 30 h，R1 = 40 h，RAM(30h) = 40 h，P2 = 40 h
```

MOV A，Rn

字节： 1

周期： 1

编码：

1	1	1	0	1	n	n	n

运算：

```
MOV
(A) ← (Rn)
```

MOV A，direct

字节： 2

周期： 3

编码：

1	1	1	0	0	1	0	1	直接地址

运算：

```
MOV
(A) ← (direct)
```

MOV A，@Ri

字节： 1

周期： 4

编码：

1	1	1	0	0	1	1	i

运算：

```
MOV
(A) ← {Ri}
```

MOV A，#data

字节： 2

周期： 2

编码：

0	1	1	1	0	1	0	0	即时数据

运算：

```
MOV
(A) ← data
```

MOV Rn，A

字节： 1

周期： 1

编码：

1	1	1	1	1	n	n	n

运算：

```
MOV
(Rn) ← (A)
```

MOV Rn，direct

字节： 2

周期： 4

编码：

1	0	1	0	1	n	n	n	直接地址

运算：

```
MOV
(Rn) ← (direct)
```

MOV Rn，#data

字节： 2

周期： 2

编码：

0	1	1	1	1	n	n	n	即时数据

运算：

```
MOV
(Rn) ← data
```

MOV direct，A

字节： 2

周期： 2

编码：

1	1	1	1	0	1	0	1	直接地址

运算：

```
MOV
(direct) ← (A)
```

MOV direct，Rn

字节： 2

周期： 3

编码：

1	0	0	0	1	n	n	n	直接地址

运算：

```
MOV
(direct) ← (Rn)
```

MOV direct1，direct2

字节： 3

周期： 4

编码：

1	0	0	0	0	1	0	1	direct2	direct1

运算：

```
MOV
{direct1} ← {direct2}
```

MOV direct，@Ri

字节： 2

周期： 5

编码：

1	0	0	0	0	1	1	i	直接地址

运算：

```
MOV
(direct) ← {Ri}
```

MOV direct，#data

字节： 3

周期： 3

编码：

0	1	1	1	0	1	0	1	直接地址	即时数据

运算：

```
MOV
(direct) ← data
```

MOV @Ri，A

字节： 1

周期： 3

编码：

1	1	1	1	0	1	1	i

运算：

```
MOV
{Ri} ← (A)
```

MOV @Ri，direct

字节： 2

周期： 4

编码：

1	0	1	0	0	1	1	i	直接地址

运算：

```
MOV
{Ri} ← (direct)
```

MOV @Ri，#data

字节： 2

周期： 3

编码：

0	1	1	1	0	1	1	i	即时数据

运算：

```
MOV
{Rn} ← data
```

MOV <destination-bit>, <source-bit>

功能： 移动位数据。

说明： 将第二操作数指示的布尔变量复制到第一操作数指定的位置。两个操作数必须有一个是进位标志。另一个操作数可以是任一可直接寻址位。该指令不影响其他寄存器或标志。

示例：

```
之前：C = 1，P1 = 50 h (01010000b)
      MOV   P1.0，C
之后：C = 1，P1 = 51 h (01010001b)
```

MOV C,bit

字节： 2

周期： 3

编码：

1	0	1	0	0	0	1	0	位地址

运算：

```
MOV
(C) ← (bit)
```

MOV bit，C

字节： 2

周期： 4

编码：

1	0	0	1	0	0	1	0	位地址

运算：
```
MOV
(bit) ← (C)
```

MOV DPTR, #data16

功能：用 16 位常数加载数据指针

说明：将指示的 16 位常数加载至数据指针。该指令的第二操作数是高阶字节 (DPH)，第三操作数是低阶字节 (DPL)。该指令不影响寄存器或标志。

示例：
```
       MOV    DPTR，#1234H
之后：DPH = 12h，DPL = 34 h
```

字节：3

周期：3

编码：

1	0	0	1	0	0	0	0	Data 15~8	Data 7~0

运算：
```
MOV
(DPTR) ← Data15~0
```

MOVC A, @A+<base-reg>

功能：移动代码字节

说明：将代码或常数从程序存储器加载到累加器。加载地址是初始累加器（无符号 8 位）和 16 位基础寄存器之和。基础寄存器可以是数据指针或 PC。如果将 PC 用作基础寄存器，在与累加器相加前，将 PC 递增到下一个指令的地址。执行 16 位加法，以便将低阶字节产生的进位传递至高阶字节。该指令不影响寄存器或标志。

示例：
```
之前：A = 03 h
       MOV    A，@A+PC
       RET
       DB     01 h
       DB     23 h
       DB     45 h
       DB     67 h
之后：A = 45 h（由于 RET 占 1 个字节）
```

MOVCA，@A+DPTR

字节：1

周期：4

编码：

1	0	0	1	0	0	1	1

运算：
```
MOVC
(A) ← {A) + (DPTR}
```

MOVCA，@A+PC

字节：1

周期：4

编码：

1	0	0	0	0	0	1	1

运算：
```
MOVC
(PC) ← (PC) + 1
(A) ← {A) + (PC}
```

MOVX < destination -byte>, <source-byte>

功能：外部移动

说明：在累加器和外部数据存储器之间传送数据。该指令有两种类型：
8 位和 16 位间接寻址模式。在 8 位寻址模式中，当前寄存器区的 R0 或 R1 提供一个 8 位地址。16 位寻址模式将数据指针的 DPH 用作高阶字节 (A15~A8)，DPL 用作低阶字节 (A7~A0)，用于外部 RAM 的加载位置。

示例：
```
之前：R0 = 12 h，R1 = 34 h，外部 RAM (0012 h) = 56 h
```

```
        MOVX   A，@R0
        MOVX   @R1，A
之后：A = 56 h，外部 RAM (0034 h) = 56 h
```

MOVX A，@Ri

字节： 1

周期： 5~12

编码：

1	1	1	0	0	0	1	i

运算：

```
MOVX
(A) ← {Ri}
```

MOVX A，@DPTR

字节： 1

周期： 4~11

编码：

1	1	1	0	0	0	0	0

运算：

```
MOVX
(A) ← {DPTR}
```

MOVX @Ri，A

字节： 1

周期： 6~13

编码：

1	1	1	1	0	0	1	i

运算：

```
MOVX
{Ri} ← (A)
```

MOVX @DPTR，A

字节： 1

周期： 5~12

编码：

1	1	1	1	0	0	0	0

运算：

```
MOVX
{DPTR} ← (A)
```

MUL AB

功能： 乘法

说明： MUL AB 执行累加器和寄存器 B 上的无符号 8 位乘法。16 位乘积的低阶字节返回累加器。高阶字节存储在寄存器 B 中。如果乘积大于 255 (FFh)，则溢出标志置位；否则清除。执行指令之后，清除进位标志。

示例：

```
之前：A = 55 h，B = 04 h
        MUL    AB
之后：A = 54 h，B = 01 h，C = 0，OV = 1
```

字节： 1

周期： 4

编码：

1	0	1	0	0	1	0	0

运算：

```
MUL
(B) (A) ← (A) X (B)
```

NOP

功能： 无操作

说明： NOP 指令不执行任何操作，但 PC 递增 1。寄存器和标志不会更改。

示例：

```
        CLR    P1.4
        NOP
        NOP
        NOP
```

```
        SET    P1.4
之后：引脚 P1.4 中出现一个持续 3 个周期的低脉冲。
```

字节： 1

周期： 1

编码：

0	0	0	0	0	0	0	0

运算：

```
NOP
(PC) ← (PC) + 1
```

ORL <destination-byte>, <source-byte>

功能： 用于字节变量的逻辑 OR

说明： ORL 对指令中的两个指示操作数执行逻辑 OR 运算。结果存储在目标字节中。该指令不影响任何标志。

该指令的操作数允许六 (6) 种寻址模式。当目标操作数为累加器时，源操作数可以是即时值、寄存器、直接或间接寻址字节。当目的地是直接寻址字节时，源操作数可以是即时值或累加器。

当该指令用于修改 I/O 端口时，初始端口数据从输出数据锁存器读取，而非输入引脚。

示例：

```
之前：累加器 = CAh (11001010b)，R0 = 55h (01010101b)
      ORL    A，R0
之后：累加器 = DFh (11011111b)
```

ORL A，Rn

字节： 1

周期： 2

编码：

0	1	0	0	1	n	n	n

运算：

```
ORL
(A) ← (A) OR (Rn)
```

ORL A，direct

字节： 2

周期： 3

编码：

0	1	0	0	0	1	0	1	直接地址

运算：

```
ORL
(A) ← (A) OR (direct)
```

ORL A，@Ri

字节： 1

周期： 4

编码：

0	1	0	0	0	1	1	i

运算：

```
ORL
(A) ← (A) OR {Ri}
```

ORL A，#data

字节： 2

周期： 2

编码：

0	1	0	0	0	1	0	0	即时数据

运算：

```
ORL
(A) ← (A) OR data
```

ORL direct，A

字节： 2

周期： 4

编码：

0	1	0	0	0	0	1	0	直接地址

运算：

```
ORL
(direct) ← (direct) OR (A)
```

ORL direct，#data

字节： 3

周期： 4

编码：

0	1	0	0	0	0	1	1	直接地址	即时数据

运算：

```
ORL
(direct) ← (direct) OR data
```

ORL C, <source-bit>

功能： 位变量的逻辑或运算

说明： 在进位标志和指令中的指示位之间执行逻辑 OR 运算。结果存储在进位标志中。如果源位的布尔值为逻辑 1，则进位标志置位；否则，进位标志保持初始状态。操作数前面的斜线 ('/') 指示将操作数的逻辑补码用作源值，且不影响源位。除 C 标志外，该指令不影响其他标志。源位必须使用直接位寻址模式。

示例：

```
之前：P1.4 = 0，C = 0
      ORL    C，P1.4
之后：P1.4 = 0，C = 0
      ORL    C，/P1.4
之后：P1.4 = 0，C = 1
```

ORL C，bit

字节： 2

周期： 3

编码：

0	1	1	1	0	0	1	0	位地址

运算：

```
ORL
(C) ← (C) OR （位）
```

ORL C，/位

字节： 2

周期： 3

编码：

1	0	1	0	0	0	0	0	位地址

运算：

```
ORL
(C) ← (C) OR NOT （位）
```

POP direct

功能： 出栈

说明： 检索通过堆栈指针寻址的内部 RAM 位置的内容，并将其存储在由第二操作数指示的直接寻址地址中。实施该指令之后，递减堆栈指针。不影响标志。

示例：

```
之前：SP = 36 h，RAM(36 h) = 22 h，RAM(35 h) = 21 h，RAM(34 h) = 20 h
      POP    DPH
      POP    DPL
之后：SP = 34 h，DPH = 22 h，DPL = 21 h
      POP    SP
之后：SP = 20 h
```

字节： 2

周期： 3

编码：

1	1	0	1	0	0	0	0	直接地址

运算：

```
POP
(direct) ← {SP}
(SP) ← (SP) – 1
```

PUSH direct

功能： 压栈

说明： 将指示直接地址的内容存储至通过堆栈指针寻址的内部 RAM 位置。递增堆栈指针。不影响标志。

示例：
```
之前：SP = 07 h，DPTR = 0123 h
      PUSH   DPL
      PUSH   DPH
之后：SP = 09 h，RAM(08h) = 23 h，RAM(09h) = 01 h
```

字节： 2

周期： 4

编码：

1	1	0	0	0	0	0	0	直接地址

运算：
```
PUSH
(SP) ← (SP) + 1
{SP} ← (direct)
```

RET

功能： 从子例程返回

说明： RET 指令将 PC 值的高阶字节和低阶字节从堆栈中依次弹出。然后，堆栈指针递减 2。程序继续执 PC 指定的位置。一般情况下，该地址紧跟 ACALL 或 LCALL 指令。该指令不影响标志。

示例：
```
之前：PC = 1200 h，SP = 12 h，RAM(11 h) = 01 h，RAM(12 h) = 23 h
      RET
之后：PC = 0123 h，SP = 10 h
```

字节： 1

周期： 5

编码：

0	0	1	0	0	0	1	0

运算：
```
RET
(PC15-8) ← {SP}
(SP) ← (SP) –1
(PC7-0) ← {SP}
(SP) ← (SP) – 1
```

RETI

功能： 从中断返回

说明： RETI 指令从堆栈中依次弹出 PC 值的高阶和低阶字节，恢复中断逻辑，以接受相同优先级的其他中断请求。堆栈指针递减 2。PSW 不会自动恢复到中断前的状态。程序继续执行位置，一般是检测到中断请求的地址后的指令。执行 RETI 指令时，如果有较低或相同优先级的中断挂起，则在处理挂起中断前需要执行一个指令。该指令的执行不影响寄存器或标志。

示例：
```
之前：PC = 1200 h，SP = 12 h，RAM(11h) = 01 h，RAM(12h) = 23 h
      RETI
之后：PC = 0123，SP = 10 h
```

字节： 1

周期： 5

编码：

0	0	1	1	0	0	1	0

运算：
```
RETI
(PC15-8) ← {SP}
(SP) ← (SP) – 1
(PC7-0) ← {SP}
(SP) ← (SP) – 1
```

RL A

功能： 左循环指令

说明： 将累加器中的 8 位数据左移一位。位7移至位 0 位置。
该指令不影响标志。

示例：
```
之前：A = 94 h (10010100b)
      RL A
之后：A = 29 h (00101001b)
```

字节： 1

周期： 1

编码：

0	0	1	0	0	0	1	1

运算：
RL
$(A_{n+1}) \leftarrow (A_n)$ n = 0 to 6
$(A_0) \leftarrow (A_7)$

RLC A

功能： 通过进位标志左移累加器

说明： 累加器中的 8 位数据和进位标志一起左移一位。位 7 移至进位标志。进位标志的初始状态移至位 0 位置。该指令不影响其他标志。

示例：
```
之前：A = 94 h (10010100b), C = 0
      RL A
之后：A = 28 h (00101000b), C = 1
```

字节： 1

周期： 1

编码：

0	0	1	1	0	0	1	1

运算：
RLC
$(A_{n+1}) \leftarrow (A_n)$ n = 0 to 6
$(A_0) \leftarrow (C)$
$(C) \leftarrow (A_7)$

RR A

功能： 右循环指令

说明： 将累加器中的 8 位数据右移一位。位 0 移至位 7 位置。
该指令不影响标志。

示例：
```
之前：A = 94 h (10010100b)
      RR A
之后：A = 4 Ah (01001010b)
```

字节： 1

周期： 1

编码：

0	0	0	0	0	0	1	1

运算：
RR
$(A_n) \leftarrow (A_{n+1})$ n = 0 to 6
$(A_7) \leftarrow (A_0)$

RRC A

功能： 通过进位标志右移累加器

说明： 累加器中的 8 位数据和进位标志一起右移一位。位 0 移至进位标志。进位标志的初始状态移至位 7 位置。

该指令不影响其他标志。

示例：
```
之前：A = 94 h (10010100b)，C = 1
      RL A
之后：A = CAh (11001010b)，C = 0
```

字节：1

周期：1

编码：

0	0	0	1	0	0	1	1

运算：
```
RRC
(An) ← (An+1) n = 0 至 6
(A7) ← (C)
(C) ← (A0)
```

SETB <bit>

功能：置位

说明：将指示位设置为 1。SETB 可在进位标志或任意可直接寻址位上运行。该指令不影响标志。

示例：
```
之前：C = 0，P1 = 80 h (10000000b)
      SETBC
      SETB P1.0
之后：C = 1，P1 = 81 h (10000001b)
```

SETB C

字节：1

周期：1

编码：

1	1	0	1	0	0	1	1

运算：
```
SETB
(C) ← 1
```

SETB bit

字节：2

周期：4

编码：

1	1	0	1	0	0	1	0	位地址

运算：
```
SETB
(bit) ← 1
```

SJMP rel

功能：短跳转（在 -127~+128 字节以内）

说明：SJUMP 无条件分行至指示地址。执行过程中产生的分支目的地可通过将 PC+2 与第二操作数相加得到。因此，目的地址的允许范围是从该指令前的 128 字节至该指令后的 127 字节。

示例：
```
之前：PC = 1000 h，LABLE1 位地址 = 1022 h
      SJMP   LABLE1
之后：PC = 1022 h
```

字节：2

周期：3

编码：

1	0	0	0	0	0	0	0	rel. 地址

运算：
```
SJMP
(PC) ← (PC) + 2
(PC) ←(PC) + rel. 地址
```

SUBB A, <source-byte>

功能：带借位减法

说明：从累加器的内容中减去指示源字节和进位标志，结果留在累加器中。如果执行减法时位 7 需要借位，进位标志置位；否则清零。如果在执行 SUBB 指令前，进位标志已经设置为 1，这意味着先前减法运算中发生借位。因此，须将进位标志与源字节一起从累加器中减去。如果位 3 中发生借位，AV 标志置位；

否则清零。如果位 6 中需要借位而位7不需要，或是位 7 中需要借位而位 6 不需要，则 OV 标志置位。当减去有符号整数时，OV 标志指示从正数减负数的过程中产生负数结果，或者从负数减正数的过程中产生正数结果。该指令中的源操作数可采用四 (4) 种寻址模式：寄存器、直接/间接寻址和即时值。

示例：
```
之前：累加器 = A8 h，R3 = 8Ch，C = 1
      SUBB   A，R3
之后：累加器 = 1Bh，C = 0，AC = 1，OV = 0
```

SUBB A，Rn

字节：1

周期：2

编码：

1	0	0	1	1	n	n	n

运算：
```
SUBB
(A) ← (A) – (C) – (Rn)
```

SUBB A，direct

字节：2

周期：2

编码：

1	0	0	1	0	1	0	1	直接地址

运算：
```
SUBB
(A) ← (A) – (C) – (direct)
```

SUBB A，@Ri

字节：1

周期：4

编码：

1	0	0	1	0	1	1	i

运算：
```
SUBB
(A) ← (A) – (C) – {Ri}
```

SUBB A，#data

字节：2

周期：2

编码：

1	0	0	1	0	1	0	0	即时数据

运算：
```
SUBB
(A) ← (A) – (C) – data
```

SWAP A

功能：交换累加器中的半字节

说明：交换累加器的低阶半字节 (bit3~bit0) 和高阶半字节 (bit7~bit4)。如果执行四次循环，则结果相同（无进位标志）。该指令不影响标志。

示例：
```
之前：A = B3h (10110011b)
      SWAP A
之后：A = 3Bh (00111011b)
```

字节：1

周期：1

编码：

1	1	0	0	0	1	0	0

运算：
```
SWAP
(A3-0) ↔ (A7-4)
```

XCH A, <byte>

功能：交换累加器和字节变量

说明：将指示变量的内容加载到累加器。同时，将累加器的初始值写回指示变量。指令中的变量可采用：寄存器和直接/间接寻址。

示例：
```
之前：A = 11 h，R5 = 22 h
      XCH A，R5
之后：A = 22 h，R5 = 11 h
```

XCH A，Rn

字节：1

周期：2

编码：

1	1	0	0	1	n	n	n

运算：
```
XCH
(A) ↔ (Rn)
```

XCH A，direct

字节：2

周期：3

编码：

1	1	0	0	0	1	0	1	直接地址

运算：
```
XCH
(A) ↔ (direct)
```

XCH A，@Ri

字节：1

周期：4

编码：

1	1	0	0	0	1	1	i

运算：
```
XCH
(A) ↔ {Ri}
```

XCHD A, @Ri

功能：交换数

说明：交换累加器的低阶半字节 (bit3~bit0) 与间接寻址的内容。累加器和 RAM 的高阶半字节 (bit7~bit4) 不受影响。该指令不影响标志。

示例：
```
之前：A = 31 h，R0 = 40 h，RAM(40h) = 64 h
      XCHD A，@R0
之后：A = 34 h，RAM(40h) = 61 h
```

字节：1

周期：5

编码：

1	1	0	1	0	1	1	i

运算：
```
XCHD
(A3-0) ↔ {Ri)3-0)
```

XRL <destination-byte>, <source-byte>

功能：用于字节变量的逻辑 Exclusive-OR

说明：XRL 在指令的两个操作数之间执行逻辑 Exclusive-OR 运算。结果存储在目的地中。该指令不影响任何标志。该指令的两个操作数允许六 (6) 种寻址模式。当目标操作数为累加器时，源操作数可以是即时值、寄存器、直接或间接寻址字节。目的地是直接地址时，源操作数可以是即时数据或累加器。当该指令用于修改 I/O 端口时，数据从输出数据锁存器读取，而非输入引脚。

示例：
```
之前：累加器 = CAh (11001010b)，R0 = C8h (11001000b)
      XRL    A，R0
之后：累加器 = 02 h (00000010b)
```

XRL A，Rn

字节：1

周期：2

编码：

0	1	1	0	1	n	n	n

运算：

```
XRL
(A) ← (A) XOR (Rn)
```

XRL A，direct

字节：2

周期：3

编码：

0	1	1	0	0	1	0	1	直接地址

运算：

```
XRL
(A) ← (A) XOR (direct)
```

XRL A，@Ri

字节：1

周期：4

编码：

0	1	1	0	0	1	1	i

运算：

```
XRL
(A) ← (A) XOR {Ri}
```

XRL A，#data

字节：2

周期：2

编码：

0	1	1	0	0	1	0	0	即时数据

运算：

```
XRL
(A) ← (A) XOR data
```

XRL direct，A

字节：2

周期：4

编码：

0	1	1	0	0	0	1	0	直接地址

运算：

```
XRL
(direct) ← (direct) XOR (A)
```

XRL direct，#data

字节：3

周期：4

编码：

0	1	1	0	0	0	1	1	直接地址	即时数据

运算：

```
XRL
(direct) ← (direct) XOR data
```

相关数据手册

FCM8531 — 嵌入式 MCU 和可配置三相 PMSM / BLDC 电机控制器