SONiX 8 位 MCU

指令集

SONiX 8 位单片机

修正记录

版本号	日期	说明
VER 1.90	2002 年 9 月	V1.90 第一版
VER 1.93	2003 年 2 月	修改了指令 ADD　M，A 的说明，将“M← M+A”改成了“M ← A+M”。

目 录

1 概述

本手册包括 SONIX 8-Bit MCU 系列每条指令的详细说明及示例。

指令说明格式如下：
— 指令格式
— 指令名称操作数
— 指令操作数源文件/目的文件格式
— 指令原文说明
— 指令应用示例说明

2 指令表

Field	Mnemonic		Description	C	DC	Z	Cycle
	MOV	A,M	A ← M	-	-	√	1
M	MOV	M,A	M ← A	-	-	-	1
O	B0MOV	A,M	A ← M (bnak 0)	-	-	√	1
V	B0MOV	M,A	M (bank 0) ← A	-	-	-	1
E	MOV	A,I	A ← I	-	-	-	1
	B0MOV	M,I	M ← I, (M = only for Working registers R, Y, Z , RBANK & PFLAG)	-	-	-	1
	XCH	A,M	A ← M	-	-	-	1
	B0XCH	A,M	A ← M (bank 0)	-	-	-	1
	MOVC		R, A ← ROM [Y,Z]	-	-	-	2
	ADC	A,M	A ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1
A	ADC	M,A	M ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1
R	ADD	A,M	A ← A + M, if occur carry, then C=1, else C=0	√	√	√	1
I	ADD	M,A	M ← A + M, if occur carry, then C=1, else C=0	√	√	√	1
T	B0ADD	M,A	M (bank 0) ← A + M (bank 0) , if occur carry, then C=1, else C=0	√	√	√	1
H	ADD	A,I	A ← A + I, if occur carry, then C=1, else C=0	√	√	√	1
M	SBC	A,M	A ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1
E	SBC	M,A	M ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1
T	SUB	A,M	A ← A - M, if occur borrow, then C=0, else C=1	√	√	√	1
I	SUB	M,A	M ← A - M, if occur borrow, then C=0, else C=1	√	√	√	1
C	SUB	A,I	A ← A - I, if occur borrow, then C=0, else C=1	√	√	√	1
	DAA		To adjust ACC's data format from HEX to DEC.	√	-	-	1
	MUL	A,M	R, A ← A * M, The LB of product stored in Acc and HB stored in R register. ZF affected by Acc.	-	-	√	2
	AND	A,M	A ← A and M	-	-	√	1
L	AND	M,A	M ← A and M	-	-	√	1
O	AND	A,I	A ← A and I	-	-	√	1
G	OR	A,M	A ← A or M	-	-	√	1
I	OR	M,A	M ← A or M	-	-	√	1
C	OR	A,I	A ← A or I	-	-	√	1
	XOR	A,M	A ← A xor M	-	-	√	1
	XOR	M,A	M ← A xor M	-	-	√	1
	XOR	A,I	A ← A xor I	-	-	√	1
	SWAP	M	A (b3~b0, b7~b4) ←→ M(b7~b4, b3~b0)	-	-	-	1
P	SWAPM	M	M(b3~b0, b7~b4) ←→ M(b7~b4, b3~b0)	-	-	-	1
R	RRC	M	A ← RRC M	√	-	-	1
O	RRCM	M	M ← RRC M	√	-	-	1
C	RLC	M	A ← RLC M	√	-	-	1
E	RLCM	M	M ← RLC M	√	-	-	1
S	CLR	M	M ← 0	-	-	-	1
S	BCLR	M.b	M.b ← 0	-	-	-	1
	BSET	M.b	M.b ← 1	-	-	-	1
		M.b	M(bank 0).b ← 0	-	-	-	1
		M.b	M(bank 0).b ← 1	-	-	-	1
	CMPRS	A,I	ZF,C ← A - I, If A = I, then skip next instruction	√	-	√	1 + S
B	CMPRS	A,M	ZF,C ← A – M, If A = M, then skip next instruction	√	-	√	1 + S
R	INCS	M	A ← M + 1, If A = 0, then skip next instruction	-	-	-	1 + S
A	INCMS	M	M ← M + 1, If M = 0, then skip next instruction	-	-	-	1 + S
N	DECS	M	A ← M - 1, If A = 0, then skip next instruction	-	-	-	1 + S
C	DECMS	M	M ← M - 1, If M = 0, then skip next instruction	-	-	-	1 + S
H	BTS0	M.b	If M.b = 0, then skip next instruction	-	-	-	1 + S
	BTS1	M.b	If M.b = 1, then skip next instruction	-	-	-	1 + S
	B0BTS0	M.b	If M(bank 0).b = 0, then skip next instruction	-	-	-	1 + S
	B0BTS1	M.b	If M(bank 0).b = 1, then skip next instruction	-	-	-	1 + S
	JMP	d	PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-	2
	CALL	d	Stack ← PC15~PC0, PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-	2
M	RET		PC ← Stack	-	-	-	2
I	RETI		PC ← Stack, and to enable global interrupt	-	-	-	2
S	RETLW		PC ← Stack, and to load a value by PC+A	-	-	-	2
C	NOP		No operation	-	-	-	1

3 指令说明

数据传送指令

MOV—存储器读/写指令

操作：

Mnemonic	Description	C	DC	Z	Cycle
MOV A,M	A ← M	-	-	√	1
MOV M,A	M ← A	-	-	-	1
MOV A,I	A ← I	-	-	-	1

注：

1. "I"为立即数；
2. "M"系统寄存器或用户自行定义的寄存器；
3. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

MOV 指令为寄存器读/写指令，通过累加器 ACC 传送数据，共有 3 种操作格式。

- **MOV A，M：从存储器中读取数据并存入 ACC 中，结果为 0，零标志（Z）置 1，否则零标志置 0。**
- **MOV M，A：从 ACC 中的数据写入存储器中，此操作不影响 PFLAG。**
- **MOV A，I：将立即数赋予 ACC，此操作不影响 PFLAG。**

➲ **例：**

```
MOV     A, #55H     ; 写入立即数 55H 到 ACC。
MOV     WK00, A     ; 将 ACC 中的数据写入到用户自定义的 WK00 中。
MOV     A, WK00     ; 从 WK00 中读取数据并存储到 ACC 中。
```

B0MOV—存储器 BANK0 读/写指令

操作：

Mnemonic	Description	C	DC	Z	Cycle
B0MOV A,M	A ← M (bank 0)	-	-	√	1
B0MOV M,A	M (bank 0) ← A	-	-	-	1
B0MOV M,I	M ← I (M = Working registers, RBANK & PFLAG)	-	-	-	1

注：

1. "I"为立即数；
2. "M"为系统寄存器或用户自行定义存储器 BANK0 位置的寄存器；
3. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

B0MOV 指令为寄存器读/写指令，存储器必须位于 BANK0，通过 ACC 传送数据，共有 3 种操作格式。

- **B0MOV A，M：从 BANK0 存储器中读取数据存入到 ACC 中。结果为 0，零标志（Z）置 1，否则零标志置 0。**
- **B0MOV M，A：将 ACC 的数据写入到 BANK0 存储器中，操作不影响 PFLAG。**
- **B0MOV M，I：将立即数赋予 BANK0 存储器，存储器必须是工作寄存器（H，L，R，X，Y，Z）、RBANK 或 PFLAG，操作不影响 PFLAG。**

➲ **例：**

```
B0MOV   A, WK00     ; 读取 WK00 中的数据并存储到 ACC 中，WK00 是用户在 BANK0 中自定义的。
                    ;
B0MOV   WK00, A     ; 将 ACC 中的数据写入到用户自定义的 WK00 中。

B0MOV   R, #10      ; 直接写入一个立即数到 R 寄存器中。
```

XCH—累加器与存储器的数据交换

操作：

Mnemonic	Description	C	DC	Z	Cycle
XCH A,M	A ←→M	-	-	-	1

注：

1. “M”为系统寄存器或用户自行定义的寄存器；

2. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

XCH 指令用于累加器 ACC 和存储器之间交换数据。执行 XCH 指令后，ACC 的数据就存入存储器中，而存储器中的数据则存入 ACC 中

- **XCH M，A：在 ACC 和存储器之间交换数据，操作不影响 PFLAG。**

➲ **例：**

```
XCH     A, WK00     ; 互换 ACC 和用户自定义的 WK00 中的数据。
```

B0XCH—累加器与存储器（BANK0）的数据交换

操作：

Mnemonic	Description	C	DC	Z	Cycle
B0XCH A,M	A ←→M (bank 0)	-	-	-	1

注：

1. “M”为系统寄存器或用户自行定义存储器 BANK0 位置的寄存器。

2. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

B0XCH 指令用于 ACC 和存储器之间交换数据。存储器必须位于 BANK0，执行 B0XCH 后，ACC 的数据存入存储器中，而存储器中的数据则存入 ACC 中。

- **B0XCH M，A：在 ACC 和 BANK0 之间交换数据。操作不影响 PFLAG。**

➲ **例：**

```
B0XCH   A, WK00     ; 互换 ACC 和 WK00 中的数据，WK00 是用户在 BANK0 中自定义的。
```

MOVC—从 ROM 中读取数据

操作：

Mnemonic	Description	C	DC	Z	Cycle
MOVC	R, A ← ROM [X,Y,Z]	-	-	-	2

注：

1. 指令周期为 2 个周期，一个周期等于 1/Fcpu。

说明：

MOVC 指令用于从 ROM 中读取数据，常应用于查表。在定义了 X，Y，Z 数据指向 ROM 地址后，用 MOVC 指令，则读取的数据高字节存入 R 寄存器中，低字节存入 ACC 中。

- **MOVC：读取 ROM 中的数据并存入到 R 寄存器和 ACC 中，操作不影响 PFLAG。**

➲ **例：查表位于“table_1”的 ROM 数据。**

```
        B0MOV   Y, #TABLE1$M    ; 取得表格地址中间字节
        B0MOV   Z, #TABLE1$L    ; 取得表格地址低字节
        MOVC                    ; 查表，R = 00H，ACC = 35H
        .                       ;
        INCMS   Z               ; Z+1
        NOP                     ;
        MOVC                    ; 查表，R = 51H，ACC = 05H.
        .       .               ;
TABLE1: DW      0035H           ; 定义表格数据（16 位）
        DW      5105H           ; “
        DW      2012H           ; “
        .       .
```

算术指令

ADC—带进位加法运算

操作：

Mnemonic		Description	C	DC	Z	Cycle
ADC	A,M	A ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1
ADC	M,A	M ← A + M + C, if occur carry, then C=1, else C=0	√	√	√	1

注：

1. "M"为系统寄存器或用户自行定义的寄存器；
2. "C"为进位标志；
3. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

ADC 指令用于取累加器 ACC、存储器和进位标志的总和。结果为 0，零标志（Z）置 1，否则置 0。如果结果溢出，则进位标志 C 置 1，反之进位标志 C 置 0。如果结果的低半字节向高半字节进位，十进制标志（DC）置 1，反之置 0。

- **ADC A，M：将 ACC、存储器和进位标志（C）相加，结果存储在 ACC 中。**
  ADC M，A：将 ACC、存储器和进位标志（C）相加，结果存储在存储器中。

➲ **例：ADC 指令用来进行数据的加法运算。一个字的数据分为 DAH 和 DAL，DAH 为高字节，而 DAL 为低字节。将 0x10ff 和 0x0103 相加，将其和存在 DAH 和 DAL 中。**

```
MOV     A,#03H      ; 输入 0x0103 的低字节到 DAL
B0MOV   DAL,A
MOV     A,#01H      ; 输入 0x0103 的高字节到 DAH
B0MOV   DAH,A

MOV     A,#0FFH     ; 和低字节数据相加，DAL = 02H
ADD     DAL,A

MOV     A,#10H      ; 和高字节数据相加，DAH = 12H
ADC     DAH,A
                    ; 总和是 0x1202.
```

ADD—不带进位的加法运算

操作：

Mnemonic		Description	C	DC	Z	Cycle
ADD	A,M	A ← A + M, if occur carry, then C=1, else C=0	√	√	√	1
ADD	M,A	M ← A + M, if occur carry, then C=1, else C=0	√	(	(	1
ADD	A,I	A (A + I, if occur carry, then C=1, else C=0	(	(	(	1

注：

1. "M"为系统寄存器或用户自行定义的寄存器；
2. "C"为进位标志；
3. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

ADD 指令为加法指令，结果为 0，零标志（Z）置 1，反之置 0；如果结果溢出，进位标志（C）置 1，否则置 0；如果结果中有从低半字节向高半字节进位，十进制进位标志置 1，反之置 0。

- **ADD A，M：将 ACC 和存储器中的数据相加，结果存在 ACC 中。**
- **ADD M，A：将 ACC 和存储器中的数据相加，结果存在存储器中。**
- **ADD A，I：将 ACC 中的内容和立即数相加，结果存在 ACC 中。**

➲ **例：**

```
ADD     A,WK00      ; A = A + WK00

ADD     WK00,A      ; WK00 = A + WK00

ADD     A,#10       ; A = A + 10
```

B0ADD—累加器和存储器 BANK0 内容相加

操作：

Mnemonic	Description	C	DC	Z	Cycle
B0ADD M,A	M (bank 0) ← A + M (bank 0), if occur carry, then C=1, else C=0	√	√	√	1

注：

1. "M"为系统寄存器或用户自行定义存储器 BANK0 位置的寄存器；
2. "C"为进位标志；
3. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

B0ADD 是加法指令，寄存器必须位于存储器 BANK0 中。结果为 0，零标志 Z 置 1，否则置 0；结果溢出则进位标志置 1，反之置 0；结果有从低半位向高半位进位，十进制标志 DC 置 1，反之置 0。

- **B0ADD M，A：将 ACC 中的内容和 BANK0 中的内容相加，结果存在存储器中。**

➲ **例：**

```
B0ADD     WK00,A      ; WK00 = A + WK00
```

SBC—带借位减法

操作：

Mnemonic	Description	C	DC	Z	Cycle
SBC A,M	A ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1
SBC M,A	M ← A - M - /C, if occur borrow, then C=0, else C=1	√	√	√	1

注：

1. "M"为系统寄存器或用户自行定义的寄存器；
2. "C"为进位标志；
3. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

SBC 指令将 ACC 的数据减去存储器的数据和借位标志，如果结果为 0，零标志置 1，反之置 0；结果有借位发生，进位标志 C 置 0，反之置 1；结果产生低半字节的借位信号，十进制标志 DC 置 0，反之置 1。

- **SBC A，M：从 ACC 中减去存储器中的内容和借位标志，结果存在 ACC 中。**
- **SBC M，A：从 ACC 中减去存储器中的内容和借位标志，结果存在存储器中。**

➲ **例：SBC 指令用来进行数据减法运算，一个字的数据分为 DAH 和 DAL，DAH 是高字节而 DAL 是低字节。将 0x10FF 减去 0x0103，结果存在 DAH 和 DAL 中。**

```
MOV       A,#03H      ; 输入 0x0103 的低字节到 DAL
B0MOV     DAL,A
MOV       A,#01H      ; 输入 0x0103 的高字节到 DAH
B0MOV     DAH,A

MOV       A,#0FFH     ; 和低字节相加，DAL = 0FCH
SUB       DAL,A

MOV       A,#10H      ; 和高字节相加，DAH = 0FH
SBC       DAH,A
                      ; 结果是 0x0FFC.
```

SUB—不带借位减法

操作：

Mnemonic		Description	C	DC	Z	Cycle
SUB	A,M	A ← A - M, if occur borrow, then C=0, else C=1	√	√	√	1
SUB	M,A	M ← A - M, if occur borrow, then C=0, else C=1	√	√	√	1
SUB	A,I	A (A - I, if occur borrow, then C=0, else C=1	√	√	√	1

注：

1．“M”系统寄存器或用户自行定义的寄存器；
2．“C”为进位标志；
3．指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

SUB 指令是减法指令，结果为 0，零标志 Z 置 1，反之置 0；结果中有借位，进位标志置 0，反之置 1；结果中产生低半字节的借位信号，十进制进位标志 DC 置 0，反之置 1。

- **SUB A，M：从 ACC 中减去存储器中的内容，将结果存在 ACC 中。**
- **SUB M，A：从 ACC 中减去存储器中的内容，将结果存在存储器中。**
- **SUB A，I：从 ACC 中减去一个立即数，将结果存在 ACC 中。**

➲ **例：**

```
SUB     A,WK00      ; A = A - WK00
SUB     WK00,A      ; WK00 = A - WK00
SUB     A,#10       ; A = A - 10
```

DAA

操作：

Mnemonic	Description	C	DC	Z	Cycle
DAA	Adjust ACC’s data format from HEX to DEC	√	-	-	1

DAA 指令用来将十六进制数据转换成十进制数据。此条指令的操作数将 ACC 分成高低半字节，每个半字节的初始值如果大于 9，就加上 6；反之初始值就保持不变，结果存在 ACC 中。

For ACC.3~ACC.0>9
ACC.3~ACC.0←(ACC.3~ACC.0)+6
IF (ACC.7~ACC.4+1)>9
  Then ACC.7~ACC.4←(ACC.7~ACC.4)+1+6(C 标志为 1)
Else ACC.7~ACC.4←(ACC.7~ACC.4)+1

For ACC.3~ACC.0<=9
ACC.3~ACC.0←ACC.3~ACC.0
IF ACC.7~ACC.4>9
  Then ACC.7~ACC.4←(ACC.7~ACC.4)+6(C 标志为 1)
Else ACC.7~ACC.4←ACC.7~ACC.4

➲ **例 1：**

```
MOV     A,#055H     ; A = 55H
DAA                 ; A = 55H
```

➲ **例 2：**

```
MOV     A,#01FH     ; A = 1FH
DAA                 ; A = 25H
```

MUL—不带符号的乘法

操作：

Mnemonc	Description	C	DC	Z	Cycle
MUL A,M	R, A ← A * M, The LB of product stored in Acc and HB stored in R register. ZF affected by Acc.	-	-	√	2

注：

1. "M"系统寄存器或用户自行定义的寄存器；

2. 指令周期为 2 个周期，一个周期等于 1/Fcpu。

说明：

MUL 指令是在 ACC 和存储器中进行 2 个 8-bit 不带符号二进制数据的乘法指令，其结果的高字节存到 R 寄存器中，低字节存在 ACC 中。如果结果为 0，零标志 Z 置 1，反之置 0。

➲ **例：**

```
        MOV     A, #03H         ; 输入数据 03H 到 BANK0 中的 DATA_1
        B0MOV   DATA_1,A        ;
        MOV     A, #0FFH        ; 输入数据 0FFH 到 ACC
        MUL     A, DATA_1       ; 两数相乘，结果是 R = 02H，A=0FDH，Z_flag=0
                                ;
    :
        MOV     A, #00H         ; 输入数据 00H 到 BANK0 中的 DATA_1
        B0MOV   DATA_1,A        ;
        MOV     A, #0FFH        ; 输入数据 0FFH 到 ACC
        MUL     A, DATA_1       ; 两数相乘，结果是 R = 00H，A=00H，Z_flag=1
```

逻辑指令

AND—逻辑与

操作：

Mnemonic		Description	C	DC	Z	Cycle
AND	A,M	A (A and M	-	-	√	1
AND	M,A	M (A and M	-	-	√	1
AND	A,I	A (A and I	-	-	√	1

注：

1. “M”系统寄存器或用户自行定义的寄存器；
2. “Z”为零标志；
3. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

AND 指令为逻辑与指令。两个操作数的相应位都是“1”，相应位的操作结果也为“1”。结果为 0，零标志 Z 置 1，反之置 0。

- **AND A，M：将 ACC 和存储器中的内容进行逻辑与运算，结果存在 ACC 中。**
- **AND M，A：将 ACC 和存储器中的内容进行逻辑与运算，结果存在存储器中。**
- **AND A，I：将 ACC 和立即数进行逻辑与运算，结果存在 ACC 中。**

➲ **例：读取 Port2 的数据，仅保持低字节数据。**

```
; 存储结果到 ACC
        B0MOV   A,P2

        AND     A,#00001111B    ; 将高字节清零并和低字节比较
                                ; 若 P2＝10101010B，结果就是 00001010B
; 或
        B0MOV   A, #00001111B

        AND     A,P2            ; 将高字节清零并和低字节比较
                                ; 若 P2＝10101010B，结果就是 00001010B

; 存储结果到 P2
        B0MOV   A, #00001111B

        AND     P2,a            ; 将高字节清零并和低字节比较
                                ; 若 P2＝10101010B，结果就是 00001010B
```

OR—逻辑或运算

操作：

Mnemonic		Description	C	DC	Z	Cycle
OR	A,M	A ← A or M	-	-	√	1
OR	M,A	M ← A or M	-	-	√	1
OR	A,I	A ← A or I	-	-	√	1

注：

1. “M”系统寄存器或用户自行定义的寄存器；

2. “Z”为零标志；

3. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

OR 指令为逻辑或指令。两个操作数相应的位任何一位为 1，其相应位为 1。如果结果为 0，零标志置 1，反之置 0。

- **OR A，M：将 ACC 和存储器中的内容进行逻辑或运算，结果存入 ACC 中。**
- **OR M，A：将 ACC 和存储器中的内容进行逻辑或运算，结果存入存储器中。**
- **OR A，I：将 ACC 中的内容和一个立即数据进行逻辑或运算，结果存入 ACC 中。**

➲ **例：**

```
; 存储结果到 ACC
        B0MOV       A,P2

        OR          A,#0FH      ; 保持高字节，并将所有的低字节设为“1”
                                ; 如 P2 = 10101010B，结果为 10101111B
                                ;
; 或
        B0MOV       A, #0FH

        OR          A,P2        ; 保持高字节，并将所有的低字节设为“1”
                                ; 如 P2 = 10101010B，结果为 10101111B
                                ;

; 存储结果到 P2
        B0MOV       A, #0FH

        OR          P2,a        ; 保持高字节，并将所有的低字节设为“1”
                                ; 如 P2 = 10101010B，结果为 10101111B
                                ;
```

XOR—逻辑异或运算

操作：

Mnemonic	Description	C	DC	Z	Cycle
XOR A,M	A ← A xor M	-	-	√	1
XOR M,A	M ← A xor M	-	-	√	1
XOR A,I	A ← A xor I	-	-	√	1

注：

1. "M"系统寄存器或用户自行定义的寄存器；
2. "Z"为零标志；
3. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

XOR 指令为逻辑异或运算指令。两个操作数的相应位不相同，其相应为为 1。结果为 0，零标志置 1，反之置 0。

- **XOR A，M：将 ACC 和存储器中的内容进行逻辑异或运算，结果存入 ACC 中。**
- **XOR M，A：将 ACC 和存储器中的内容进行逻辑异或运算，结果存入存储器中。**
- **XOR A，I：将 ACC 中的内容和一个立即数进行逻辑异或运算，结果存入 ACC 中。**

➲ **例：**

```
; 结果存在 ACC 中
        B0MOV   A,P2
        OR      A,#0FFH         ; 将 P2 取反，若 P2 = 10101010B，结果为 01010101B
; 或
        B0MOV   A, #0FFH
        OR      A,P2            ; 将 P2 取反，若 P2 = 10101010B，结果为 01010101B

; 结果存在 P2 中
        B0MOV   A, #0FFH
        OR      P2,a            ; 将 P2 取反，若 P2 = 10101010B，结果为 01010101B
```

PROCESS 指令

操作：

Mnemonic	Description	C	DC	Z	Cycle
SWAP M	A (b3~b0, b7~b4) ←→M(b7~b4, b3~b0)	-	-	-	1
SWAPM M	M(b3~b0, b7~b4) ←→ M(b7~b4, b3~b0)	-	-	-	1

注：

1. "M"为系统寄存器或用户自行定义的寄存器；
2. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

两条指令是将一个存储器的高低半字节互换，此操作不影响 PFLAG。

- **SWAP M：将寄存器高低半字节的数据互换，结果存入 ACC。**
- **SWAPM M：将寄存器高低半字节的数据互换，结果存入在存储器中。**

➲ **例：读取 P2 的值，互换高低字节，并将结果存在 WK00 中。**

```
        SWAP    P2              ; 将 P2 的值高低字节互换并将结果存在 ACC 中
        B0MOV   WK00,A          ;
```

➲ **例：读取 P2 的值，互换高低字节，并加载新的值到 P2 寄存器中。**

```
        SWAPM   P2              ; 将 P2 的值高低字节互换并将结果存在 P2 中
```

RLC&RLCM—左移指令

操作：

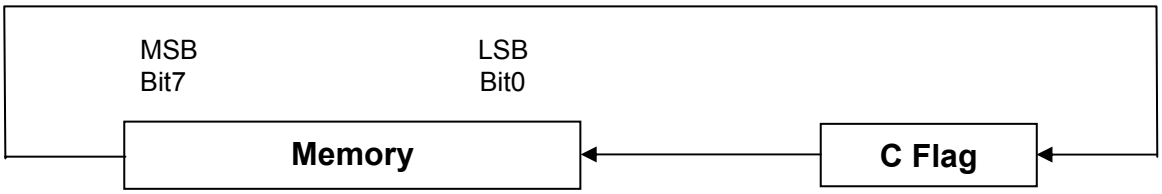
Mnemonic		Description	C	DC	Z	Cycle
RLC	M	A ← RLC M	√	-	-	1
RLCM	M	M ← RLC M	√	-	-	1

注：

1. "M"为系统寄存器或用户自行定义的寄存器；
2. "C"为进位标志；
3. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

两条指令让存储器中的内容左移一位。Bit0 根据 C 标志置位，C 标志根据 Bit7 置位。



- **RLC M：存储器的内容左移一位，结果存在 ACC 中。**
- **RLCM M：存储器的内容左移一位，结果存在存储器中。**

➲ **例：WK00=10101010B，将 WK00 换成 01010101B 并将结果存在 ACC 中。**

```
B0BSET      FC          ; 设置标志位 C

RLC         WK00        ; ACC = 01010101B，C = 1
```

➲ **例：WK00=11110011B，将 WK00 换成 11001111B 并将结果存在 WK00 中。**

```
B0BCLR      FC          ; 清标志位 C
RLCM        WK00        ; 左移一次
                        ; WK00 = 11100110B，C =1

B0BCLR      FC          ; 清标志位 C
RLCM        WK00        ; 左移两次
                        ; WK00 = 11001100B，C =1
```

RRC&RRCM—存储器右移

操作：

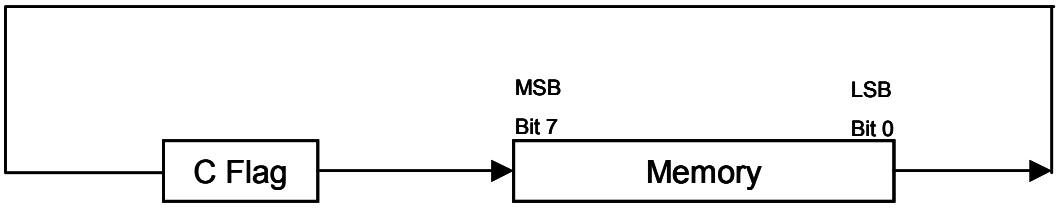
Mnemonic	Description	C	DC	Z	Cycle
RRC M	A (RRC M	(	-	-	1
RRCM M	M (RRC M	(	-	-	1

注：

1. “M”为系统寄存器或用户自行定义的寄存器；
2. “C”为进位标志；
3. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

两条指令使存储器内容向右移一位，bit7 根据 C 标志位置位，C 标志位根据 bit0 置位。



- **RRC M：存储器的内容向右移一位，结果存在 ACC 中。**
- **RRCM M：存储器的内容向右移一位，结果存在存储器中。**

➲ **例：WK00=10101010B，将 WK00 变成 01010101B 并将结果存在 ACC 中。**

```
B0BCLR      FC          ; 清标志位 C

RRC         WK00        ; ACC = 01010101B，C = 0
```

➲ **例：WK00=11110011B，将 WK00 变成 00111100B 并将结果存在 WK00 中。**

```
B0BCLR      FC          ; 清标志位 C
RRCM        WK00        ; 右移一次
                        ; WK00 = 01111001B，C = 1

B0BCLR      FC          ; 清标志位 C
RRCM        WK00        ; 右移一次
                        ; WK00 = 00111100B，C = 1
```

CLR—清零

操作：

Mnemonic	Description	C	DC	Z	Cycle
CLR M	M ← 0	-	-	-	1

注：

1. “M”为系统寄存器或用户自行定义的寄存器；
2. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

CLR 指令将存储器的数据清零，此操作不影响 PFLAG。

- **CLR M：清除存储器的内容。**

➲ **例：将系统寄存器清零。**

```
CLR         Y           ; 清寄存器 Y

CLR         P0          ; 错误：“CLR”不支持只读寄存器
```

BCLR&B0BCLR—位清零

操作：

Mnemonic	Description	C	DC	Z	Cycle
BCLR M.b	M.b ← 0	-	-	-	1
B0BCLR M.b	M(bank 0).b ← 0	-	-	-	1

注：
1. “M”为系统寄存器或用户自行定义的寄存器；
2. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：
两条指令是将存储器的某一位清零，不影响 PFLAG。

- **BCLR M.b：将存储器“b”位清零。**
- **B0BCLR M.b：将存储器“b”位清零，存储器必须位于 BANK 0。**

➲ **例：将进位标志 C 清零。**

```
B0BCLR    FC
```

➲ **例：将 WK00 的第六位清零。WK00 为用户自定义寄存器。**

```
BCLR      WK00.6      ; 清 WK00 的第六位
```

BSET&B0BSET—位设置

操作：

Mnemonic	Description	C	DC	Z	Cycle
BSET M.b	M.b (1	-	-	-	1
B0BSET M.b	M(bank 0).b (1	-	-	-	1

注：
1. “M”为系统寄存器或用户自行定义的寄存器；
2. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：
两条指令是将存储器的某位置 1，不影响 PFLAG。

- **BSET M.b：将存储器中“b”位置 1。**
- **B0BSET M.b：将存储器中“b“位置 1，存储器必须位于 BANK 0 中。**

➲ **例：将进位标志 C 置位。**

```
B0BSET    FC
```

➲ **例：将 WK00 的第六位置位，WK00 为用户自定义寄存器。**

```
B0BSET    WK00.6
```

跳转指令

CMPRS—比较指令

操作：

Mnemonic	Description	C	DC	Z	Cycle
CMPRS A,I	ZF,C ← A - I, If A = I, then skip next instruction	√	-	√	1 + S
CMPRS A,M	ZF,C ← A - M, If A = M, then skip next instruction	√	-	√	1 + S

注：

1. "M"为系统寄存器或用户自行定义的寄存器；
2. "C"为进位标志；
3. "Z"为零标志；
4. 指令周期为 1+S 个周期，如果略过下条指令则 S=1，如果执行下一条指令则 S=0，一个周期等于 1/Fcpu。

说明：

两条指令将 ACC 的内容和存储器中的内容或者一个立即数作比较。如果两者相等，程序计数器就会略过下一条指令。"CMPRS"就是将 ACC 中的内容减去存储器中的内容或者一个立即数的值进行判断（实际 ACC 的值并没有变化，只是由此做判断）。结果影响零标志和进位标志。结果为 0，零标志位置 1，反之置 0。结果中若借位发生，进位标志置 0，反之置 1。十进制标志不受影响，保持为 0。

- **CMPRS A，I：将 ACC 的内容和一个立即数作比较。**
- **CMPRS A，M：将 ACC 的内容和存储器的内容作比较。**

➲ **例：将 ACC 的内容和 55H 作比较，如果两者相等，程序就跳到 SUB1，否则就跳到 SUB2。**

```
        CMPRS   A,#055H
        JMP     SUB2            ; ACC 不等于 55H，跳转到 SUB2
        JMP     SUB1            ; ACC = 55H，调整到 SUB1
```

➲ **例：将存储器 WK00 和 55H 作比较，如果两者相等，程序就跳到 SUB1，否则就跳到 SUB2。**

```
        B0MOV   A,#055H
        CMPRS   A,WK00
        JMP     SUB2            ; WK00 不等于 55H，跳转到 SUB2
        JMP     SUB1            ; WK00 = 55H，调整到 SUB1
```

INCS&INCMS—自加 1 指令

操作：

Mnemonic	Description	C	DC	Z	Cycle
INCS M	A ← M + 1, If A = 0, then skip next instruction	-	-	-	1 + S
INCMS M	M ← M + 1, If M = 0, then skip next instruction	-	-	-	1 + S

注：

1. "M"为系统寄存器或用户自行定义的寄存器；
2. 指令周期为 1+S 个周期，如果略过下条指令则 S=1，如果执行下一条指令则 S=0，一个周期等于 1/Fcpu。

说明：

两条指令是 ACC 或存储器的内容自加 1，结果为 0，程序略过下一条指令。此操作不影响 PFLAG。

- **INCS M：存储器自加 1，结果存在 ACC 中。**
- **INCMS M：存储器自加 1，结果存在存储器中。**

➲ **例：WK00 自加 1 直到溢出。**

```
        MOV     A, #0xFD
        MOV     WK00, A
INCSTEST:
        INCS    WK00
        NOP
        MOV     WK00, A
        CMPRS   A, #0x00
        JMP     INCSTEST        ; WK00 不等于 0，跳转到 INCSTEST
        .
```

➲ **例：Y 和 Z 自加，Y 为高字节，Z 为低字节。Y 一直自加直到 Z 溢出。**

```
        INCMS   Z
        JMP EXIT                ; Z 不等于 0，退出
        INCMS   Y               ; Z = 0, Y = Y+1
        NOP
EXIT:
        .
```

DECS&DECMS—自减 1 指令

操作：

Mnemonic	Description	C	DC	Z	Cycle
DECS M	A ← M - 1, If A = 0, then skip next instruction	-	-	-	1 + S
DECMS M	M ← M - 1, If M = 0, then skip next instruction	-	-	-	1 + S

注：

1. "M"系统寄存器或用户自行定义的寄存器；

2. 指令周期为 1+S 个周期，如果略过下条指令则 S=1，如果执行下一条指令则 S=0，一个周期等于 1/Fcpu。

说明：

两条指令是 ACC 或存储器内容自减 1。结果为 0，程序计数器略过下一条指令。此操作不影响 PFLAG。

- **DECS M：存储器自减 1，结果存在 ACC 中。**
- **DECMS M：存储器自减 1，结果存在存储器中。**

➲ **例：WK00 自减 1 直到等于 0。**

```
DECSTEST:
        DECMS       WK00
        JMP         DECSTEST        ; 不等于 0。继续计算
        .                           ; WK00 = 0.
```

BTS0&B0BTS0—位检测指令

操作：

Mnemonic	Description	C	DC	Z	Cycle
BTS0 M.b	If M.b = 0, then skip next instruction	-	-	-	1 + S
B0BTS0 M.b	If M(bank 0).b = 0, then skip next instruction	-	-	-	1 + S

注：

1. "M"为系统寄存器或用户自行定义的寄存器；

2. 指令周期为 1+S 个周期，如果略过下条指令则 S=1，如果执行下一条指令则 S=0，一个周期等于 1/Fcpu。

说明：

两条指令将存储器的某位和 0 作比较，结果为 0，程序计数器略过下一条指令，操作不影响 PFLAG。

- **BTS0 M.b：将存储器的某位和 0 作比较。**
- **B0BTS0 M.b：将存储器的某位和 0 作比较，存储器必须位于 BANK0。**

➲ **例：检查进位标志。如果 C=0，就跳到 SUB1，否则就跳到 SUB2。**

```
        B0BTS0      FC
        JMP         SUB2            ; C = 1，跳转到 SUB2
        JMP         SUB1            ; C = 0，跳转到 SUB1
```

➲ **例：检查 WK00 的第三位。如果 WK00.3=0，WK00 就自加 1，否则就退出。**

```
        B0BTS0      WK00.3          ; 检查 WK00 的第三位
        JMP         EXIT            ; WK00.3=1 退出
        INCMS       WK00            ; WK00.3=0，WK00 自加
        NOP                         ;
EXIT:
```

BTS1&B0BTS1—位检测指令

操作：

Mnemonic	Description	C	DC	Z	Cycle
BTS1 M.b	If M.b = 1, then skip next instruction	-	-	-	1 + S
B0BTS1 M.b	If M(bank 0).b = 1, then skip next instruction	-	-	-	1 + S

注：

1. "M"为系统寄存器或用户自行定义的寄存器；

2. 指令周期为 1+S 个周期，如果略过下条指令则 S=1，如果执行下一条指令则 S=0，一个周期等于 1/Fcpu。

说明：

两条指令是将存储器的某位和 1 作比较，结果为 1，程序计数器略过下一条指令，操作不影响 PFLAG。

- **BTS1 M.b：将存储器的某位和 1 作比较。**
- **B0BTS1 M.b：将存储器的某位和 1 作比较，存储器必须位于 BANK0。**

➲ **例：检测进位标志。如果 C=1，就跳到 SUB1，否则就跳到 SUB2。**

```
        B0BTS1      FC
        JMP         SUB2        ; C = 0，跳转到 SUB2
        JMP         SUB1        ; C = 1，跳转到 SUB1
```

➲ **例：检测 WK00 的第三位，如果 WK00.3=1，WK00 就自加 1，否则就退出。**

```
        B0BTS1      WK00.3      ; 检查 WK00 的第三位
        JMP         EXIT        ; WK00.3 = 0 退出
        INCMS       WK00        ; WK00.3 = 1，WK00 自加
        NOP                     ;
EXIT:
```

JMP—跳转指令

操作：

Mnemonic	Description	C	DC	Z	Cycle
JMP d	PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-	2

注：

1. "d"是目的地址的标签名。

2. 指令周期为 2 个周期，一个周期等于 1/Fcpu。

说明：

"JMP"为无条件跳转到指定地址指令，指定的位置可以在 ROM 的任何地址。此操作不影响 PFLAG。

- **JMP d：程序跳到地址"d"。**

➲ **例：跳到 SUB1。**

```
        JMP         SUB1
        .
SUB1:
```

➲ **例：用程序计数器和 JMP 指令形成一个跳转表功能。通过程序计数器可以执行不同的子程序。**

```
        MOV         A,WK00      ; 限制 WK00 的 0～7 位
        AND         A,#00000111B
        MOV         WK00,A

        B0ADD       PCL,A
        JMP         SUB0        ; WK00=0，跳转到 SUB0
        JMP         SUB1        ; WK00=1，跳转到 SUB1
        JMP         SUB2        ; WK00=2，跳转到 SUB2
        JMP         SUB3        ; WK00=3，跳转到 SUB3
        JMP         SUB4        ; WK00=4，跳转到 SUB4
        JMP         SUB5        ; WK00=5，跳转到 SUB5
        JMP         SUB6        ; WK00=6，跳转到 SUB6
        JMP         SUB7        ; WK00=7，跳转到 SUB7
        .
```

CALL—程序调用指令

操作：

Mnemonic	Description	C	DC	Z	Cycle
CALL d	Stack ← PC15~PC0, PC15/14 ← RomPages1/0, PC13~PC0 ← d	-	-	-	2

注：
1. "d"是目的地址的标签名。
2. 指令周期为 2 个周期，一个周期等于 1/Fcpu。

说明：
"CALL"指令用于 ROM 中调用子程序。执行指令时，程序计数器存储在堆栈缓冲器中，堆栈层数减一。子程序标签的新地址加载到程序计数器中并执行。子程序可以从 ROM 中的任何地址开始。此操作不影响 PFLAG。

- **CALL d：程序跳到地址"d"。**

➲ **例：SUB1 的地址是 0x0213，CALL 指令的地址是 0x0030。堆栈指针是 0FH。执行 CALL 指令后，堆栈缓冲器是 0x0031，堆栈指针是 0EH，程序计数器的地址为 0x0213。**

```
        ORG     0X0030
        CALL    SUB1            ; 地址为 0x0030 时跳转到 SUB1
        .                       ; 地址为 0x0031 时，程序返回
        ORG     0X0213
SUB1:                           ;
        .
        RET
```

RET—程序调用返回

操作：

Mnemonic	Description	C	DC	Z	Cycle
RET	PC ← Stack	-	-	-	2

注：
1. 指令周期为 2 个周期，一个周期等于 1/Fcpu。

说明：
完成调用子程序后，"RET"指令就返回到上一个 ROM 地址。当执行 RET 指令后，程序计数器就从堆栈缓冲器重新加载，堆栈指针加一。程序返回上一层堆栈并继续执行。子程序可以在 ROM 的任何位置完成。此操作不影响 PFLAG。

- **RET ： 返回到上一层堆栈。**

➲ **例：SUB1 的地址是 0x0213，CALL 指令的地址是 0x0030，堆栈指针位于 0EH。执行 RET 指令后，堆栈缓冲器重新加载给程序计数器，堆栈指针位于 0FH，程序将从 0x0031 开始运行。**

```
        ORG     0X0030
        CALL    SUB1            ; 地址为 0x0030 时跳转到 SUB1
        .                       ; 地址为 0x0031 时，程序返回
        .
        ORG     0X0213
SUB1:                           ;
        .
        RET                     ; 返回到地址 0x0031
```

RETI—中断返回指令

操作：

Mnemonic	Description	C	DC	Z	Cycle
RETI	PC ← Stack, and to enable global interrupt	-	-	-	2

注：

1. 指令周期为 2 个周期，一个周期等于 1/Fcpu。

说明：

一旦有中断发生，程序计数器存储在堆栈缓冲器中，堆栈指针加一。屏蔽全局中断控制位，程序跳到中断向量开始执行中断服务程序。RETI 是中断服务程序结束指令。将堆栈缓冲器的内容加载给程序计数器并开放全局中断控制位，程序退出中断服务程序。此操作不影响 PFLAG。

➢ **RETI：从中断返回并开放全局中断控制位。**

➲ **例：**

```
                ORG         8               ; 中断向量地址
                B0XCH       A,ACCBUF
                B0MOV       A,PFLAG
                B0MOV       PFLAGBUF,A
                .
                B0MOV       A,PFLAGBUF
                B0MOV       PFLAG,A
                B0XCH       A,ACCBUF
                RETI                        ; 退出中断服务程序
MAIN:
                .                           ; 中断发生，程序跳转到中断向量地址
                .                           ; 从中断向量地址返回
```

NOP—空操作

操作：

Mnemonic	Description	C	DC	Z	Cycle
NOP	No operation	-	-	-	1

注：

1. 指令周期为 1 个周期，一个周期等于 1/Fcpu。

说明：

“NOP”是空操作指令，常用作时间延迟。一个“NOP”指令耗时一个指令周期。此操作不影响 PFLAG。

➢ **NOP：空操作。**

➲ **例：延迟时间 1024 个指令周期，如果振荡器为 4MHz，则延迟 1024 微秒。**

```
DLY1024U:
                MOV         A,#0            ; 清 ACC

DLY1024U_10:
                ADD         A,#01H          ; ACC 加 1
                B0BTS1      FZ              ; 检查 ACC 是否溢出
                JMP         DLY1024U_10     ; 没有溢出，ACC 继续加 1
                .                           ; 退出延迟服务程序
```

SONIX 公司保留对以下所有产品在可靠性、功能和设计方面的改进做进一步说明的权利。SONIX 不承担由本手册所涉及的产品或电路的运用和使用所引起的任何责任。SONIX 的产品不是专门设计来应用于外科植入、生命维持和任何 SONIX 产品的故障会对个体造成伤害甚至死亡的领域。如果将 SONIX 的产品应用于上述领域，即使这些是由 SONIX 在产品设计和制造上的疏忽引起的，用户应赔偿所有费用、损失、合理的人身伤害或死亡所直接或间接产生的律师费用，并且用户保证 SONIX 及其雇员、子公司、分支机构和销售商与上述事宜无关。

Main Office:
Address: 9F, NO. 8, Hsien Cheng 5th St, Chupei City, Hsinchu, Taiwan R.O.C.
Tel: 886-3-551 0520
Fax: 886-3-551 0523

Taipei Office:
Address: 15F-2, NO. 171, Song Ted Road, Taipei, Taiwan R.O.C
Tel: 886-2-2759 1980
Fax: 886-2-2759 8180

Hong Kong Office:
Address: Flat 3 9/F Energy Plaza 92 Granville Road, Tsimshatsui East Kowllon.
Tel: 852-2723 8086
Fax: 852-2723 9179

Technical Support by Email:
Sn8fae@sonix.com.tw

深圳技术支持中心：
地址：深圳市科技园南区 T2-B 栋 2 楼
电话：0755－26719666
传真：0755－26719786