MSP430 系列汇编指令集

简介：

MSP430 的内核 CPU 结构是按照精简指令集和高透明指令的宗旨来设计的，使用的指令有硬件执行的内核指令和基于现有硬件结构的高效率的仿真指令。仿真指令使用内核指令及芯片额外配置的常数发生器 CG1，CG2。

MSP430 指令的寻址方式包括立即寻址、索引寻址、符号寻址和绝对寻址。这四种方式均可用于源操作数，而索引、符号和绝对寻址方式只可用于目的操作数。源操作数的指令集需占用代码存储器中的 1~3 个字。

寻址方式：

寻址方式	源操作数	目标操作数	语法	As	Ad	说明
寄存器寻址	Yes	Yes	MOV Rs, Rd	00	0	寄存器内容是操作数
索引寻址	Yes	Yes	MOV x(Rn), y(Rm)	01	1	(Rn+x)指向操作数，x 储存在下一个字中
符号寻址	Yes	Yes	MOV EDE, TON1	01	1	(PC+x)指向操作数，x 储存在下一个字中
绝对寻址	Yes	Yes	MOV &MEM, &TCDAT	01	1	跟随在指令后的字包含绝对地址
间接寻址	Yes		MOV @Rn, y(Rm)	10		Rn 用作指向操作数的指针
间接自动增量	Yes		MOV @Rn+, Rm	11		Rn 用作指向操作数的指针,其后 Rn 被增量
立即数寻址	Yes		MOV #x, TON1	11		跟随在指令后的字包含立即数 x，使用间接自动增量模式 @PC+

Rn：n=0~15， Rs：源寄存器 Rd：目标寄存器

寻址方式	例子	操作
寄存器寻址	MOV R10, R11	R10 → R11
索引寻址	MOV 2(R5), 6(R6)	M(2+R5) → M(6+R6)
符号寻址	MOV EDE, TON1	M(EDE) → M(TON1)
绝对寻址	MOV &MEM, &TCDAT	M(MEM) → M(TCDAT)
间接寻址	MOV @R10, Tab(R6)	M(R10) → M(Tab+R6)
间接自动增量	MOV @R10+, R11	M(R10) → R11, R10+2 →R10
立即数寻址	MOV #0AAH, TON1	#0AAH → M(TON1)

指令概述：（共 51 条指令）

操作码		操作	状态位 V	N	Z	C
ADC[.W]; ADC.B	dst	dst + C -> dst	*	*	*	*
ADD[.W]; ADD.B	src, dst	src + dst -> dst	*	*	*	*
ADDC[.W]; ADDC.B	src, dst	src + dst + C -> dst	*	*	*	*
AND[.W]; AND.B	src, dst	src .and. dst -> dst	0	*	*	*
BIC[.W]; BIC.B	src, dst	.not.src .and. dst -> dst	-	-	-	-
BIS[.W]; BIS.B	src, dst	src .or. dst -> dst	-	-	-	-
BIT[.W]; BIT.B	src, dst	src .and. dst	0	*	*	*
BR	dst	转移到	-	-	-	-
CALL	dst	PC + 2 ->堆栈, dst -> PC	-	-	-	-
CLR[.W];CLR.B	dst	清除目的操作数	-	-	-	-
CLRC		清除进位位	-	-	-	0
CLRN		清除负位	-	0	-	-
CLRZ		清除零位	-	-	0	-
CMP[.W];CMP.B	dst	dst - src	*	*	*	*

DADC[.W];DADC.B	dst	dst + C -> dst (十进制)	*	*	*	*
DADD[.W];DADD.B	src, dst	src + dst + C -> dst (十进制)	*	*	*	*
DEC[.W];DEC.B	dst	dst - 1 -> dst	*	*	*	*
DECD[.W];DECD.B	dst	dst - 2 -> dst	*	*	*	*
DINT		禁止中断	-	-	-	-
EINT		使能中断	-	-	-	-
INC[.W];INC.B	dst	dst + 1 -> dst 目的操作数增 1	*	*	*	*
INCD[.W];INCD.B	dst	dst + 2 -> dst 目的操作数增 2	*	*	*	*
INV[.W];INV.B	dst	目的操作数求反	*	*	*	*
JC/JHS	标号	进位位被置时转移到标号语句	-	-	-	-
JEQ/JZ	标号	零位被置时转移到标号语句	-	-	-	-
JGE	标号	（N .xor. V）= 0 时转移到标号语句	-	-	-	-
JL	标号	（N .xor. V）= 0 时转移到标号语句	-	-	-	-
JMP	标号	无条件转移到标号语句	-	-	-	-
JN	标号	负位被置时转移到标号语句	-	-	-	-
JNC/JLO	标号	进位位复位时转移到标号语句	-	-	-	-
JNE/JNZ	标号	零位复位时转移到标号语句	-	-	-	-
MOV[.W];MOV.B	src, dst	src -> dst	-	-	-	-
NOP		空操作	-	-	-	-
POP[.W];POP.B	dst	项目从堆栈弹出, SP + 2 -> SP	-	-	-	-
PUSH[.W];PUSH.B	src	SP – 2 -> SP , src -> @SP	-	-	-	-
RETI		从中断返回 TOS -> SR，SP + 2 -> SP TOS -> PC，SP + 2 -> SZP	-	-	-	-
RET		从子程序返回 TOS -> SR，SP + 2 -> SP	-	-	-	-
RLA[.W];RLA.B	dst	算术左移	*	*	*	*
RLC[.W];RLC.B	dst	带进位位左移	*	*	*	*
RRA[.W];RRA.B	dst	算术右移	0	*	*	*
RRC[.W];RRC.B	dst	带进位位右移	*	*	*	*
SBC[.W];SBC.B	dst	从目的操作数减去借位	*	*	*	*
SETC		置进位位	-	-	-	1
SETN		置负位	-	1	-	-
SETZ		置零位	-	-	1	-
SUB[.W];SUB.B	src, dst	dst + .not. src + 1 -> dst	*	*	*	*
SUBC[.W];SUBC.B	src, dst	dst + .not. src + C -> dst	*	*	*	*
SWAP	dst	交换字节	-	-	-	-
SXT	dst	dst 位 7 -> 位 8.....位 15	0	*	*	*
TST[.W];TST.B	dst	测试目的操作数	0	*	*	1
XOR[.W];XOR.B	src, dst	src .xor. dst -> dst	*	*	*	*

注：状态位中 * 表示影响；- 表示不影响；0 表示清零，1 表示置位。

指令格式：

双操作数指令 **由 4 个域组成，共有 16 位代码：**

- **操作码域** 4bit **操作码**

- 源域 6bit 源寄存器 + As (4 位寄存器数 0~15 + 2 个寻址位)
- 字节操作识别符 1bit B/W (B/W=1 以字节形式执行，B/W=0 以字形式执行)
- 目的域 5bit 目的寄存器 + Ad (4 位寄存器数 0~15 + 1 个寻址位)

15 12	11 8	7	6	5 4	3 0
操作码	源寄存器	Ad	B/W	As	目的寄存器

单操作数指令 由 2 个域组成，共有 16 位代码：

- 操作码域 9bit 操作码
- 字节操作识别符 1bit B/W (B/W=1 以字节形式执行，B/W=0 以字形式执行)
- 目的域 6bit 目的寄存器 + Ad (4 位寄存器数 0~15 + 1 个寻址位)

15 12	7	6	5 4	3 0
0 0 0 1 x x x x x x	B/W	Ad	目的寄存器	

条件转移和无条件转移指令 由 2 个域组成，共有 16 位代码：

- 操作码域 6bit
- 转移偏转域 10bit B/W (B/W=1 以字节形式执行，B/W=0 以字形式执行)

15 13	12 10	9 0
0 0 1 x x x	x x x x x x x x x x	
操作码	转移码	符号 偏 移

条件转移使指令可转移到相对于当前地址范围在 –512 ~ +512 字之间的地址。

指令集说明(51 条指令以字母为序)：

ADC[.W] 将进位加至目的操作数

语法 ADC dst 或 ADC.W dst

操作 dst + C -> dst

仿真 ADDC #0.dst

说明 进位 C 加至目的操作数。操作数以前的内容丢失。

状态位 N：结果为负时置位，为正时复位
Z：结果为零时置位，其它情况时复位
C：dst 从 0FFFFH 增至 0000 时置位，其它情况复位
V：发生算术溢出时置位，其它情况时复位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 R13 指向的 16 位数值加到 R12 指向的 32 位数值

```
ADD     @R13,   0(R12)      ;加 LSDs
ADC     2(R12)              ;将进位加至 MSD
```

ADC.B 将进位加至目的操作数

语法 ADC.B dst

操作 dst + C -> dst

仿真 ADDC.B #0.dst

说明 进位 C 加至目的操作数。操作数以前的内容丢失。

状态位 N：结果为负时置位，为正时复位
Z：结果为零时置位，其它情况时复位
C：dst 从 0FFFFH 增至 0000 时置位，其它情况复位
V：发生算术溢出时置位，其它情况时复位

方式位　　OscOff、CPUOff 和 GIE 不受影响

例子　　R13 指向的 8 位数值加到 R12 指向的 16 位数值

```
ADD.B    @R13,    0(R12)      ;加 LSDs
ADC.B    1(R12)               ;将进位加至 MSD
```

ADD[.W]　源操作数加至目的操作数

语法　　**ADD　　src,　　dst　　或　　ADD.W　　src,　　dst**

操作　　src + dst -> dst

说明　　源操作数加至目的操作数。源操作数不受影响，目的操作数以前的内容丢失。

状态位　　N：结果为负时置位，为正时复位

Z：结果为零时置位，其它情况时复位

C：结果产生进位时置位，否则复位

V：发生算术溢出时置位，其它情况时复位

方式位　　OscOff、CPUOff 和 GIE 不受影响

例子　　R5 加上 10。由一个进位使程序“转移”到 TONI

```
ADD      #10,    R5          ;
JC       TONI                 产生进位
……                            无进位
```

ADD.B　源操作数加至目的操作数

语法　　**ADD.B　　src,　　dst**

操作　　src + dst -> dst

说明　　源操作数加至目的操作数。源操作数不受影响，目的操作数以前的内容丢失。

状态位　　N：结果为负时置位，为正时复位

Z：结果为零时置位，其它情况时复位

C：结果产生进位时置位，否则复位

V：发生算术溢出时置位，其它情况时复位

方式位　　OscOff、CPUOff 和 GIE 不受影响

例子　　R5 加上 10。由一个进位使程序“转移”到 TONI

```
ADD      #10,    R5          ;将近 10 加至 R5 的低位字节
JC       TONI                 ;若(R5)>=246[0AH + 0F6H]，则产生进位
……                            ;无进位
```

ADDC[.W]　源操作数加至目的操作数

语法　　**ADDC　　src,　　dst　　或　　ADDC.W　　src,　　dst**

操作　　src + dst + C -> dst

说明　　源操作数和进位 C 加至目的操作数。源操作数不受影响，目的操作数以前的内容丢失。

状态位　　N：结果为负时置位，为正时复位

Z：结果为零时置位，其它情况时复位

C：结果的 MSB 产生进位时置位，否则复位

V：发生算术溢出时置位，其它情况时复位

方式位　　OscOff、CPUOff 和 GIE 不受影响

例子　　R13 指向的 32 位数值加至 R13 中的指针之上 11 个字(20/2 +2/2)的 32 位计数器。

```
ADD      @R13+,    20(R13)    ;无进位加 LSDs
ADDC     @R13+,    20(R13)    ;带进位加 MSDs
……                            ;
```

ADDC.B　源操作数加至目的操作数

语法　　**ADDC.B　　src,　　dst**

操作　　src + dst + C -> dst

说明　　源操作数和进位 C 加至目的操作数。源操作数不受影响，目的操作数以前的内容丢失。

状态位　N：结果为负时置位，为正时复位
Z：结果为零时置位，其它情况时复位
C：结果的 MSB 产生进位时置位，否则复位
V：发生算术溢出时置位，其它情况时复位

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　R13 指向的 24 位数值加至 R13 中的指针之上 11 个字的 24 位计数器。

```
ADD.B    @R13+,    10(R13)    无进位加 LSDs
ADDC.B   @R13+,    10(R13)    ;带进位加中间位
ADDC.B   @R13+,    10(R13)    ;带进位加 MSDs
……                            ;
```

AND.[W]　源操作数和目的操作数与

语法　**AND　src,　dst　或　AND.W　src,　dst**

操作　src .AND. dst -> dst

说明　源操作数和目的操作数逻辑与。结果放入目的操作。

状态位　N：结果为 MSB 为 1 时置位，为 0 时复位
Z：结果为零时置位，其它情况时复位
C：结果不为 0 时置位，其它情况时复位(= .NOT. Zero)
V：复位

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　R5 中被置的位用作 TOM 寻址字的一个标志(#0AA55H)。如果结果为 0，程序转到 TON1 处

```
MOV     #0AA55H,    R5      ;将标志装入 R5 寄存器
AND     R5,         TOM     ;TOM 用 R5 寻址标志字
JZ      TON1                ;
……                          结果为非 0
;或
AND     #0AA55H,    TOM
JZ      TON1
```

AND.B　源操作数和目的操作数与

语法　**AND.B　src,　dst**

操作　src .AND. dst -> dst

说明　源操作数和目的操作数逻辑与。结果放入目的操作。

状态位　N：结果为 MSB 为 1 时置位，为 0 时复位
Z：结果为零时置位，其它情况时复位
C：结果不为 0 时置位，其它情况时复位(= .NOT. Zero)
V：复位

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　标志位#0A5H 和低位字节的 TOM 逻辑与。如果结果为 0，程序转移到标号 TON1 处

```
AND.B    #0A5H,    TOM     ; 用 R5 标记低位字节 TOM
JZ       TON1              ;
……                          结果为非 0
```

BIC[.W]　清 0 目的操作数的各位

语法　**BIC　src,　dst　或　BIC.W　src,　dst**

操作　.NOT. src .AND. dst -> dst

说明　求反后的源操作数和目的操作数逻辑与。结果放入目的操作，源操作数不变。

状态位　N：不影响
Z：不影响
C：不影响

V：不影响

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　RAM 字 LEO 的 6 个 MSBs 被清 0。

```
BIC     #0FC00H,     LEO        ; 清 0MEM（LEO）中的 6 个 MSBs
```

BIC.B　清0 目的操作数的各位

语法　**BIC.B　src,　dst**

操作　.NOT.　src　.AND.　dst　->　dst

说明　求反后的源操作数和目的操作数逻辑与。结果放入目的操作，源操作数不变。

状态位　N：不影响

Z：不影响

C：不影响

V：不影响

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　(1)、RAM 字 LEO 的 5 个 MSBs 被清 0。

```
BIC.B    #0F8H,     LEO        ; 清 0 RAM（LEO）中的 5 个 MSBs
```

(2)、清 0 端口引脚 P0 和 P1

```
P0OUT       .equ        011H        定义端口地址
P0          .equ        01H
P1          .equ        02H
BIC.B       #P0+P1,     $P0OUT      ;置 P0 和 P1 为低电平
```

BIS[.W]　设置目的操作数的各位

语法　**BIS　src,　dst　或　BIS.W　src,　dst**

操作　src　.OR.　dst　->　dst

说明　源操作数和目的操作数逻辑或。结果放入目的操作，源操作数不变。

状态位　N：不影响

Z：不影响

C：不影响

V：不影响

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　（1）、RAM 字 TOM 的 6 个 MSBs 被置位。

```
BIS     #003FH,     TOM        ; 设置 RAM 区域 TOM 中的 6 个 LSBs
```

（2）、启动一个 A/D 转换器

```
ASOC        .equ        1           开始转换位
ACTL        .equ        114H        ;ADC 控制寄存器
BIS         #ASOC,      &ACTL       启动 A/D 转换
```

BIS.B　设置目的操作数的各位

语法　**BIS.B　src,　dst**

操作　src　.OR.　dst　->　dst

说明　源操作数和目的操作数逻辑或。结果放入目的操作，源操作数不变。

状态位　N：不影响

Z：不影响

C：不影响

V：不影响

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　（1）、RAM 字 TOM 的 3 个 MSBs 被置位。

```
BIS.B    #0E0H,     TOM        ; 设置 RAM 区域 TOM 中的 3 个 MSBs
```

(2)、设置端口引脚 P0 和 P1 为高电平

```
P0OUT       .equ        011H        定义端口地址
P0          .equ        01H
P1          .equ        02H
BIS.B       #P0+P1,     $P0OUT      置 P0 和 P1 为高电平
```

BIT[.W]　测试目的操作数的各位

语法　**BIT　src,　dst　或　BIT.W　src,　dst**

操作　src　.AND.　dst　->　dst

说明　源操作数和目的操作数逻辑与。其结果只影响状态位。目的操作数和源操作数不变。

状态位　N：结果为 MSB 为 1 时置位，为 0 时复位

Z：结果为零时置位，其它情况时复位

C：结果不为 0 时置位，其它情况时复位(=　.NOT.　Zero)

V：复位

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　（1）、如果 R8 中的位 9 被置位，程序转移到标号语句 TOM 处。

```
BIT     #0200H,     R8          ; R8 的位 9 是否被置?
JNZ     TOM                     ;是，转换到 TOM
```

（2）、确定哪一个 A/D 通道由 MUX 配置

```
ACTL        .equ        114H        ;ADC 控制寄存器
BIT         #4,         &ACTL       ;通道 0 是否被选择?
JNZ         END                     ;是，则程序转移到 END
```

BIT.B　测试目的操作数的各位

语法　**BIT.B　src,　dst**

操作　src　.AND.　dst　->　dst

说明　源操作数和目的操作数逻辑与。其结果只影响状态位。目的操作数和源操作数不变。

状态位　N：结果为 MSB 为 1 时置位，为 0 时复位

Z：结果为零时置位，其它情况时复位

C：结果不为 0 时置位，其它情况时复位(=　.NOT.　Zero)

V：复位

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　如果 R8 中的位 3 被置位，程序转移到标号语句 TOM 处。

```
BIT.B   #8H,        R8          ; R8 的位 3 是否被置?
JC      TOM                     ;是，转换到 TOM
```

（2）、确定哪一个 A/D 通道由 MUX 配置

```
ACTL        .equ        114H        ;ADC 控制寄存器
BIT         #4,         &ACTL       ;通道 0 是否被选择?
JNZ         END                     ;是，则程序转移到 END
```

BR,　BRANCH　转换到目的操作数

语法　**BR　dst**

操作　dst　->　PC

仿真　MOV　dst,　PC

说明　无条件转移到 64K 地址空间的任一地址处，可使用所有的源寻址方式。转换指令是一个字指令。

状态位　不影响状态位。

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　所有寻址方式示例：

```
BR      #EXEC               转换到标号 EXEC 或直接转换（例：#0A4H）
                            ;内核指令：MOV　@PC+,　PC
BR      EXEC                转换到标号 EXEC 包含的地址
```

```
                                ;内核指令：MOV     x(PC),     PC    ，间接寻址
        BR      &EXEC           转换到绝对地址包含的地址
                                ;内核指令：MOV     x(0),      PC    ，间接寻址
        BR      R5              ;转换到 R5 包含的地址
                                ;内核指令：MOV     R5,        PC    ，用 R5 间接寻址
        BR      @R5             ;转换到 R5 指向的字的包含的地址
                                ;内核指令：MOV     @R5,       PC    ，用 R5 间接寻址
        BR      @R5+            ;转换到 R5 指向的字的包含的地址，R5 中的指针加 1。
                                ;内核指令：MOV     @R5,       PC    ，用 R5 间接寻址，R5 增 1
        BR      x(R5)           ;转换到 R5+1 指向的地址包含的地址，例：从 x 开始的地址表，
                                ;x 可以是一个地址或一个标号。
                                ;内核指令：MOV     x(R5),     PC    ，用 R5+x 间接寻址
```

CALL　　调用子程序

语法　　**CALL　　dst**

操作

```
dst     ->    tmp
SP – 2      ->    SP
PC      ->    @SP       将 PC 更新至 TOS
tmp     ->    PC        将 dst 保存至 PC
```

说明　　调用 64K 地址空间中任一地址处的子程序。可使用所有的寻址方式，返回地址(后续指令的地址)储存在堆栈中，调用指令是一个字指令。

状态位　　不影响状态位。

例子　　所有寻址方式示例：

```
        CALL    #EXEC           调用标号 EXEC 或直接调用（例：#0A4H）
                                ; SP – 2   ->   SP,     PC + 2   ->   @SP,    @PC+   ->   PC
        CALL    EXEC            调用 EXEC 包含的地址
                                ; SP – 2   ->   SP,     PC + 2   ->   @SP,    x(PC)+   ->   PC
                                ;间接寻址
        CALL    &EXEC           调用绝对地址 EXEC 中包含的地址
                                ; SP – 2   ->   SP,     PC + 2   ->   @SP,    x(PC)+   ->   PC
                                ;间接寻址
        CALL    R5              ;调用 R5 包含的地址
                                ; SP – 2   ->   SP,     PC + 2   ->   @SP,    R5   ->   PC
                                ;用 R5 间接寻址
        CALL    @R5             ;调用 R5 指向的字的包含的地址
                                ; SP – 2   ->   SP,     PC + 2   ->   @SP,    @R5   ->   PC
                                ;用 R5 间接寻址
        CALL    @R5+            ;调用 R5 指向的字的包含的地址，R5 中的指针加 1。
                                ; SP – 2   ->   SP,     PC + 2   ->   @SP,    @R5   ->   PC
                                ;用 R5 间接寻址，然后 R5 自动加 1
        CALL    x(R5)           ;调用 R5+1 指向的地址包含的地址，例：从 x 开始的地址表，
                                ; SP – 2   ->   SP,     PC + 2   ->   @SP,    @R5   ->   PC
                                ;用 R5+x 间接寻址
```

CLR[.W]　　清除目的操作数

语法　　**CLR　　dst　　或　CLR.W　　dst**

操作

```
0     ->    dst
```

仿真

```
MOV     #0,       dst
```

说明　　清除目的操作数

状态位　不影响状态位。

例子　(1)、RAM 字 TONI 被清除

```
CLR     TONI        ;0   ->   TONI
```

(2)、寄存器 R5 被清除

```
CLR     R5
```

CLR.B　清除目的操作数

语法　**CLR.B　　dst**

操作　0　->　dst

仿真　MOV.B　#0,　　dst

说明　清除目的操作数

状态位　不影响状态位。

例子　RAM 字 TONI 被清除

```
CLR.B     TONI        ;0   ->   TONI
```

CLRC　清除目的操作数

语法　**CLRC**

操作　0　->　C

仿真　BIC　#1,　　SR

说明　进位位被清零，清除进位位指令是一个字指令

状态位　N：不影响

Z：不影响

C：清零

V：不影响

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　R13 指向的 16 位十进制数值加至 R12 指向的 32 位数值

```
CLRC                            ;C = 0
DADD     @R13,     0(R12)       ;16 位数值加至 32 位数值的低位字
DADC     2(R12)                   进位加至 32 位数值的高位字
```

CLRN　清除负位

语法　**CLRN**

操作　0　->　N　　或　(.NOT. src .AND. dst -> dst)

仿真　BIC　#4,　　SR

说明　常数#04H 求反后(0FFFBH)和目的操作数逻辑与。结果放入目的操作数。清除负位指令是一个字指令

状态位　N：复位至 0

Z：不影响

C：不影响

V：不影响

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　状态寄存器中的负位被清零。这样可以避免用负数调用子程序时的特殊处理。

```
        CLRN                        ;
        CALL      SUBR
        ......
SUBR:   JN        SUBRET            ;如果输入为负，无操作并返回
        ......
SUBRET: RET
```

CLRZ　清除零位

语法　**CLRZ**

操作　　0　->　Z　　或　(.NOT. src .AND. dst -> dst)

仿真　　BIC　　#2,　　SR

说明　　常数#02H 求反后(0FFFDH)和目的操作数逻辑与。结果放入目的操作数。清除负位指令是一个字指令

状态位　　N：不影响

Z：复位至 0

C：不影响

V：不影响

方式位　　OscOff、CPUOff 和 GIE 不受影响

例子　　状态寄存器中的零位被清零。

```
CLRZ                    ;
```

CMP[.W]　比较源操作数和目的操作数

语法　　**CMP　　src,　　dst　　或　CMP.W　　src,　　dst**

操作　　dst + .NOT. src + 1　　或　(dst - src)

说明　　从目的操作数中减去源操作数。方法是将源操作数求反再加 1 。源操作数和目的操作数不受影响，不保存结果，只影响状态位。

状态位　　N：结果为负时置位，为正时复位 (src >= dst)

Z：结果为零时置位，其他情况时复位 (src = dst)

C：结果为 MSB 产生进位时置位，否则复位

V：发生算术溢出时置位，其他情况时复位

方式位　　OscOff、CPUOff 和 GIE 不受影响

例子　　(1)、比较 R5 和 R6，如果相等，程序继续从标号 EQUAL 处运行

```
CMP     R5,     R6      ;R5 = R6 ?
JEQ     EQUAL           ;是，程序跳转
```

(2)、比较两个 RAM 块。如果不等，程序转移到标号 ERROR

```
        MOV     #NUM,   R5              被比较的字数
L$1:    CMP     &BLOCK1,&BLOCK2         两个字相等吗?
        JNZ     ERROR                   ;不等，则转移到 ERROR
        DEC     R5                      被比较字数减 1
        JNZ     L$1                     ;下一次比较
```

CMP.B　比较源操作数和目的操作数

语法　　**CMP.B　　src,　　dst**

操作　　dst + .NOT. src + 1　　或　(dst - src)

说明　　从目的操作数中减去源操作数。方法是将源操作数求反再加 1 。源操作数和目的操作数不受影响，不保存结果，只影响状态位。

状态位　　N：结果为负时置位，为正时复位 (src >= dst)

Z：结果为零时置位，其他情况时复位 (src = dst)

C：结果为 MSB 产生进位时置位，否则复位

V：发生算术溢出时置位，其他情况时复位

方式位　　OscOff、CPUOff 和 GIE 不受影响

例子　　(1)、比较 EDE 和 TONI 寻址的 RAM 字节，如果相等，程序继续从标号 EQUAL 处运行

```
CMP.B   EDE,    TONI    ;MEM(EDE) = MEM(TONI) ?
JEQ     EQUAL           ;是，程序跳转
```

(2)、检查连至端口引脚 P0 和 P1 的两个键。如果按下键 1，程序转移到标号 MENU1，如果按下键 2，程序转移到标号 MENU2。

```
P0IN    .EQU    010H
KEY1    .EQU    01H
```

```
KEY2    .EQU    02H
CMP.B   #KEY1,  &P0IN
JEQ     MENU1
CMP.B   #KEY2,  &P0IN
JEQ     MENU2
```

DADC[.W]　加上十进制的进位位

语法　**DADC　dst　或　DADC.W　dst**

操作　dst + C -> dst (十进制)

仿真　DADD　#0,　dst

说明　进位位 C 作为十进制加至目的操作数

状态位　N：MSB 为 1 时置位

Z：dst 为零时置位，其他情况时复位

C：目的操作数从 9999 增至时置位，其他情况时复位

V：不确定

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　R5 中的 4 位十进制数加至 R8 指向的 8 位十进制数

```
CLRC                    ;复位进位位
DADD    R5,     0(R8)   ;加 LSDs + C
DADC    2(R8)           ;将进位位加至 MSDs
```

DADC.B　加上十进制的进位位

语法　**DADC.B　dst**

操作　dst + C -> dst (十进制)

仿真　DADD.B　#0,　dst

说明　进位位 C 作为十进制加至目的操作数

状态位　N：MSB 为 1 时置位

Z：dst 为零时置位，其他情况时复位

C：目的操作数从 9999 增至时置位，其他情况时复位

V：不确定

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　R5 中的 2 位十进制数加至 R8 指向的 4 位十进制数

```
CLRC                    ;复位进位位
DADD.B  R5,     0(R8)   ;加 LSDs + C
DADC.B  1(R8)           ;将进位位加至 MSDs
```

DADD[.W]　将十进制的进位位和源操作数加至目的操作数

语法　**DADD　src,　dst　或　DADD.W　src,　dst**

操作　src + dst + C -> dst (十进制)

说明　源操作数和目的操作数被当作 4 个带有正符号的二-十进制（BCD）数。十进制的源操作数和进位 C 被加至目的操作数。源操作数不受影响，目的操作数以前的内容丢失。些结果对于非二-十进制数是不确定的。

状态位　N：MSB 为 1 时置位，其他情况时复位

Z：结果为零时置位，其他情况时复位

C：结果大于 9999 增至时置位

V：不确定

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　R5 和 R6 中的 8 位二-十进制数加至 R3 和 R4 中的 8 位二-十进制数(R3 和 R4 含有 MSDs)。

```
CLRC                    ;清进位位
DADD    R5,     R3      ;加 LSDs
```

DADD R6, R4 ;带进位位加 MSDs
JC OVERFLOW ;若产生进位，转移至错误处理子程序

DADD.B 将十进制的进位位和源操作数加至目的操作数

语法 **DADD.B src, dst**

操作 src + dst + C -> dst (十进制)

说明 源操作数和目的操作数被当作 4 个带有正符号的二-十进制（BCD）数。十进制的源操作数和进位 C 被加至目的操作数。源操作数不受影响，目的操作数以前的内容丢失。些结果对于非二-十进制数是不确定的。

状态位 N：MSB 为 1 时置位，其他情况时复位
Z：结果为零时置位，其他情况时复位
C：结果大于 9999 增至时置位
V：不确定

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 RAMbyte CNT 中的 2 位十进制数值增 1

```
CLRC                    ;复位进位位
DADD.B  #1,     CNT     ;十进制数值增 1
```

或

```
SETC
DADD.B  #0,     1(R8)   ;相当于 DADC.B  CNT
```

DEC[.W] 目的操作数减 1

语法 **DEC dst 或 DEC.W dst**

操作 dst - 1 -> dst

仿真 SUB #1, dst

说明 目的操作数减 1，以前的内容丢失。

状态位 N：结果为负时置位，为正时复位
Z：dst 包含 1 时置位，其他情况时复位
C：dst 包含 0 时置位，其他情况时复位
V：产生算术溢出时置位，其他情况时复位
目的操作数的初始值为 08000H 时置位，其他情况时复位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 R10 减 1

```
        DEC     R10                     ;R10 减 1
        ;将从 EDE 开始的 255 字节存储区块移到从 TONI 开始的存储区
        ;地址表不能重叠：目的地址 TONI 的起始点不能位于 EDE 至 EDE + 0FEH 的范围内
        ;
        MOV     #EDE,   R6,
        MOV     #255,   R10
L$1:    MOV.B   @R6+,   TONI-EDE-1(R6)
        DEC     R10
        JNZ     L$1
```

DEC.B 目的操作数减 1

语法 **DEC.B dst**

操作 dst - 1 -> dst

仿真 SUB #1, dst

说明 目的操作数减 1，以前的内容丢失。

状态位 N：结果为负时置位，为正时复位
Z：dst 包含 1 时置位，其他情况时复位

C：dst 包含 0 时置位，其他情况时复位

V：产生算术溢出时置位，其他情况时复位

目的操作数的初始值为 08000H 时置位，其他情况时复位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 地址 LEO 处的存储器字节减 1

```
DEC.B    LEO                ;MEM(LEO)减 1
;将从 EDE 开始的 255 字节存储区块移到从 TONI 开始的存储区
;地址表不能重叠：目的地址 TONI 的起始点不能位于 EDE 至 EDE + 0FEH 的范围内
;
MOV      #EDE,     R6,
MOV      #255,     R10
L$1:     MOV.B     @R6+,     TONI-EDE-1(R6)
DEC.B    LEO
JNZ      L$1
```

DECD[.W] 目的操作数减 2

语法 **DECD dst 或 DECD.W dst**

操作 dst - 2 -> dst

仿真 SUB #2, dst

说明 目的操作数减 2，以前的内容丢失。

状态位 N：结果为负时置位，为正时复位

Z：dst 包含 2 时置位，其他情况时复位

C：dst 包含 0 或 1 时置位，其他情况时复位

V：产生算术溢出时置位，其他情况时复位

目的操作数的初始值为 08001H 或 08000H 时置位，其他情况时复位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 R10 减 2

```
DECD     R10                ;R10 减 2
;将从 EDE 开始的 255 字节存储区块移到从 TONI 开始的存储区
;地址表不能重叠：目的地址 TONI 的起始点不能位于 EDE 至 EDE + 0FEH 的范围内
;
MOV      #EDE,     R6,
MOV      #255,     R10
L$1:     MOV.B     @R6+,     TONI-EDE-2(R6)
DEC      R10
JNZ      L$1
```

DECD.B 目的操作数减 2

语法 **DECD.B dst**

操作 dst - 2 -> dst

仿真 SUB #2, dst

说明 目的操作数减 2，以前的内容丢失。

状态位 N：结果为负时置位，为正时复位

Z：dst 包含 2 时置位，其他情况时复位

C：dst 包含 0 或 1 时置位，其他情况时复位

V：产生算术溢出时置位，其他情况时复位

目的操作数的初始值为 081H 或 080H 时置位，其他情况时复位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 地址 LEO 处的存储器字节减 2

```
            DECD.B      LEO                 ;MEM(LEO)减 2
            状态字节 STATUS 减 2
            DECD.B      STATUS
```

DINT 禁止(一般)中断

语法 DINT

操作 0 -> GIE 或 (0FFF7H .AND. SR -> SR / .NOT. src .AND. dst -> dst)

仿真

```
BIC     #8,     SR
```

说明 禁止所有中断

状态位 N：不影响
Z：不影响
C：不影响
V：不影响

方式位 OscOff、CPUOff 不受影响。 GIE 复位

例子 状态寄存器中的一般中断使能位 GIE 清零可允许非中断移动一个 32 位数值。这样可保证在任何中断移动期间，该数值不会被修改

```
            DINT                            ;所有使用 GIE 位的中断事件被禁止
            MOV     COUNTHI,    R5          ;复制数值
            MOV     COUNTHO,    R6
            EINT                            ; 所有使用 GIE 位的中断事件被使能
```

注意：禁止中断---- 在 DINT 执行期间，当中断请求生效时才能执行禁止中断指令 DINT 后面的指令。如果任何代码顺序需要保护自身不被中断，则至少应该在些代码之前一个指令时执行 DINT 指令。

EINT 使能(一般)中断

语法 EINT

操作 1 -> GIE 或 (0008H .OR. SR -> SR / .NOT. src .OR. dst -> dst)

仿真

```
BIS     #8,     SR
```

说明 使能所有中断，常数 08H 与状态寄存器 SR 逻辑或，其结果放入 SR。

状态位 N：不影响
Z：不影响
C：不影响
V：不影响

方式位 OscOff、CPUOff 不受影响。 GIE 置位

例子 状态寄存器中的一般中断使能位 GIE 被置，P0IFG 是可锁存所有中断事件的寄存器地址。

```
            PUSH.B      &P0IN
            BIC.B       @SP,        &P0IFG  ;仅复位接收到的标记
            EINT                            ;预置保存在堆栈中的口 0 中断标记，允许其它中断。
            BIT         #Mask,      @SP
            JEQ         MaskOK              ;现有的标记与标志相同：程序跳转
            ......
MaskOK:     BIC         #Mask,      @SP
            ......
            INCD        SP                  ;在中断子程序开始时倒回 PUSH 指令，修改堆栈指针
            RETI
```

注意：使能中断---任何情况下都可执行使能中断指令 EINT 后面的指令，甚至在中断服务请求悬而未决时。

INC[.W] 目的操作数加 1

语法 INC dst 或 INC.W dst

操作 dst + 1 -> dst

仿真

```
ADD     #1,     dst
```

说明　　目的操作数加 1，以前的内容丢失。
状态位　　N：结果为负时置位，为正时复位
Z：dst 包含 0FFFFH 时置位，其他情况时复位
C：dst 包含 0FFFFH 时置位，其他情况时复位
V：dst 包含 07FFFH 时置位，其他情况时复位
方式位　　OscOff、CPUOff 和 GIE 不受影响
例子　　软件堆栈(不是系统堆栈)顶部的字节数据被子移动

```
SSP     .EQU    R4
        INC     SSP     通过给目的操作数据加 1 来移动 TOSS(SW 堆栈的顶部)
                        ;SSP 是一个字寄存器，所以不要使用 INC.B
```

INC.B　　目的操作数加 1

语法　　**INC.B　　dst**
操作　　dst + 1 -> dst
仿真　　ADD　　#1,　　dst
说明　　目的操作数加 1，以前的内容丢失。
状态位　　N：结果为负时置位，为正时复位
Z：dst 包含 0FFH 时置位，其他情况时复位
C：dst 包含 0FFH 时置位，其他情况时复位
V：dst 包含 07FH 时置位，其他情况时复位
方式位　　OscOff、CPUOff 和 GIE 不受影响
例子　　过程 STATUS 的状态字节加 1，当它等于 11 时，程序转移到 OVFL

```
INC.B     STATUS
CMP.B     #11,    STATUS
JEQ       OVFL
```

INCD[.W]　　目的操作数加 2

语法　　**INCD　　dst　　或　　INCD.W　　dst**
操作　　dst + 2 -> dst
仿真　　ADD　　#2,　　dst
说明　　目的操作数加 2，以前的内容丢失。
状态位　　N：结果为负时置位，为正时复位
Z：dst 包含 0FFFEH 时置位，其他情况时复位
C：dst 包含 0FFFEH 或 0FFFFH 时置位，其他情况时复位
V：dst 包含 07FFEH 或 0FFFFH 时置位，其他情况时复位
方式位　　OscOff、CPUOff 和 GIE 不受影响
例子　　无需使用寄存器即可移动堆栈顶部的项

```
SUB
        ......
        PUSH    R5          ;R5 是计算的结果，保存在堆栈中
        INCD    SP          ;通过给目的操作数加 2 从堆栈中移走 TOS
                            ;SP 是一个字寄存器，所以不要使用 INC.B
```

INCD.B　　目的操作数加 2

语法　　**INCD.B　　dst**
操作　　dst + 2 -> dst
仿真　　ADD.B　　#2,　　dst
说明　　目的操作数加 2，以前的内容丢失。
状态位　　N：结果为负时置位，为正时复位
Z：dst 包含 0FEH 时置位，其他情况时复位

C：dst 包含 0FEH 时置位，其他情况时复位

V：dst 包含 07EH 时置位，其他情况时复位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 堆栈顶部的字节加 2

```
INCD.B      0(SP)          ;TOS 处的字节加 2
```

INV[.W] 目的操作数求反

语法 **INV dst 或 INV.W dst**

操作 .NOT.dst -> dst

仿真

```
XOR     #0FFFFH,     dst
```

说明 目的操作数取反，以前的内容丢失。

状态位 N：结果为负时置位，为正时复位

Z：dst 包含 0FFFFH 时置位，其他情况时复位

C：结果不为零时置位，其他情况时复位 (= .NOT.Zero)

V：初始目的操作数为负时置位，其他情况时复位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 R5 的内容被取消(2 的补码)

```
MOV     #00AEH,     R5      ;              R5=00AEH
INV     R5                  ;R5 求反       R5=0FF51H
INC     R5                  取消 R5        R5=0FF52H
```

INV.B 目的操作数求反

语法 **INV dst 或 INV.W dst**

操作 .NOT.dst -> dst

仿真

```
XOR     #0FFH,     dst
```

说明 目的操作数取反，以前的内容丢失。

状态位 N：结果为负时置位，为正时复位

Z：dst 包含 0FFH 时置位，其他情况时复位

C：结果不为零时置位，其他情况时复位 (= .NOT.Zero)

V：初始目的操作数为负时置位，其他情况时复位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 存储字节 LEO 的内容被取消

```
MOV     #0AEH,     LEO      ;              MEM(LEO)=0AEH
INV.B   LEO                 ;LEO 求反      MEM(LEO)=051H
INC.B   LEO                 取消 LEO       MEM(LEO)=052H
```

JC 进位位为 1 时程序跳转

JHS 大于或等于时程序跳转

语法 **JC 标号**

JHS 标号

操作 若 C = 1： PC + 2 x 编移 -> PC

若 C = 0： 执行下一条指令

说明 测试状态寄存器的进位位 C。如果它被置，则指令的 LSB 中包含的 10 位符号偏移加至程序计数器。如果 C 被复位，则执行 JUMP 后面的一条指令。JC(有进位/大于等于时跳转)用于比较无符号数(0~65536)。

状态位 不影响状态位

例子 (1)、输入 P0IN.1 的信号用于定义或控制程序流

```
BIT     #10H,     &P0IN       ;信号状态进位
JC      PROGA                 ;若进位 =1,  则执行程序 A
.....                         ;若进位 =0,  程序继继续续执行
```

(2)、比较 R5 和 15。如果大于或等于，程序则转移到标号

```
CMP     #15,    R5
JHS     LABEL           ;若 R5 大于等于 15，程序跳转
 ... ...                ;若 R5 小于 15，程序继续执行
```

JEQ 等于时程序跳转

JZ 为零时程序跳转

语法 JEQ 标号

JZ 标号

操作 若 Z = 1： PC + 2 x 编移 -> PC

若 Z = 0： 执行下一条指令

说明 测试状态寄存器的零位 Z。如果它被置，则指令的 LSB 中包含的 10 位符号偏移加至程序计数器。如果 Z 被复位，则执行 JUMP 后面的一条指令。

状态位 不影响状态位

例子 (1)、若 R7 包含 0，则程序跳转至地址 TONI

```
TST     R7              ;测试 R7
JZ      TONI            ;若为 0,  则程序跳转
```

(2)、如果 R6 等于地址表的内容，程序则跳转到地址 LEO

```
CMP     R6,     Table(R5)   ;比较 R6 和 MEM 的内容( 地址表 + R5 的内容 )
JEQ     LEO                 ;若两个数相等则程序跳转
 ... ...                    ;否则，不等，程序继续执行
```

(3)、若 R5 为 0，则程序转移至标号

```
TST     R5              ;测试 R5
JZ      LABEL           ;若为 0,  则程序跳转 LABEL

JHS     LABEL           ;若 R5 大于等于 15，程序跳转
 ... ...                ;若 R5 小于 15，程序继续执行
```

JGE 大于或等于时程序跳转

语法 JGE 标号

操作 若(N.XOR.V) = 0： 则程序跳转至标号：PC + 2 x 编移 -> PC

若(N.XOR.V) = 1： 则执行下一条指令

说明 测试状态寄存器的负位 N 和溢出位 V。如果 N 和 V 均被置位或复位，则指令的 LSB 中包含的 10 位符号偏移加至程序计数器。如果其中之一被置位，则执行 JUMP 后面的一条指令。该指令允许比较符号整数。

状态位 不影响状态位

例子 如果 R6 的内容大于或等于 R7 指向的存储器内容，程序则在标号 EDE 继续。

```
CMP     @R7,    R6      ;R6>=(R7)  ? ,比较符号数
JGE     EDE             ;是，跳转
 ... ...                ;非，继续执行
```

JL 小于时程序跳转

语法 JL 标号

操作 若(N.XOR.V) = 1： 则程序跳转至标号：PC + 2 x 编移 -> PC

若(N.XOR.V) = 0： 则执行下一条指令

说明 测试状态寄存器的负位 N 和溢出位 V。如果 N 和 V 其中之一被置位，则指令的 LSB 中包含的 10 位符号偏移加至程序计数器。如果两都均被置位或复位，则执行 JUMP 后面的一条指令。该指令允许比较符号整数。

状态位 不影响状态位

例子 如果 R6 的内容小于 R7 指向的存储器内容，程序则在标号 EDE 继续。

```
        CMP     @R7,    R6      ;R6<(R7) ？,比较符号数
        JL      EDE             ;是，跳转
        ... ...                 ;非，继续执行
```

JMP　程序无条件跳转

语法　**JMP　标号**

操作　PC + 2 x **编移** -> PC

说明　**指令的** LSB **中包含的** 10 **位符号偏移加至程序计数器。**

状态位　**不影响状态位**

例子　**该字指令在相对于现有程序计数器的**-511 **至**+512 **的范围内可取代** BRANCH **指令。**

JN　为负时程序跳转

语法　**JN　标号**

操作　**若** N = 1：PC + 2 x **编移** -> PC

若 N = 0：**执行下一条指令**

说明　**测试状态寄存器的负位** N。**如果** N **被置位，则指令的** LSB **中包含的** 10 **位符号偏移加至程序计数器。如果** N **被复位，则执行** JUMP **后面的一条指令。该指令允许比较符号整数。**

状态位　**不影响状态位**

例子　**从** COUNT **中减去** R5 **中计算的结果，如果结果为负，则** COUNT **将被清除，且程序在另一个路径继续执行。**

```
        SUB     R5,     COUNT   ;COUNT – R5 ->COUNT
        JN      L$1             若为负，程序继续，COUNT = 0  PC  = L$1
        ... ...                 ;继续执行，  COUNT >= 0
L$1     CLR     COUNT
```

JNC　进位为零时程序跳转

JLO　小于时程序跳转

语法　**JNC　标号**

JLO　标号

操作　**若** C = 0：PC + 2 x **编移** -> PC

若 C = 1：**执行下一条指令**

说明　**测试状态寄存器的进位位** C。**如果它被复位，则指令的** LSB **中包含的** 10 **位符号偏移加至程序计数器。如果** C **被置位，则执行** JUMP **后面的一条指令。**JC(**有进位/大于等于时跳转**)**用于比较无符号数** (0~65536)。

状态位　**不影响状态位**

例子　R6 **中的结果加至** BUFFER。**如果产生溢出，将会使用地址** ERROR **处的错误处理程序**

```
        ADD     R6,     BUFFER      ;BUFFER + R6  ->  BUFFER
        JNC     CONT                ;无进位，  程序跳至 CONT
ERROR   .....                       ;错误处理程序开始
        ......
CONT    ... ...                     ;继续正常的程序流
```

JNE　不等时程序跳转

JNZ　不为零时程序跳转

语法　**JNE　标号**

JNZ　标号

操作　**若** Z = 0：PC + 2 x **编移** -> PC

若 Z = 1：**执行下一条指令**

说明　**测试状态寄存器的零位** Z。**如果它被复位，则指令的** LSB **中包含的** 10 **位符号偏移加至程序计数器。如果** Z **被复位，则执行** JUMP **后面的一条指令。**

状态位　**不影响状态位**

例子　　若 R7 和 R8 不等，则程序跳转至地址 TONI

```
CMP     R7,     R8      ;比较 R7 和 R8
JNE     TONI            ;若两个数不相等则程序跳转
... ...                 ;否则，程序继续执行
```

MOV[.W]　源操作数移至目的操作数

语法　　**MOV　src,　dst　或　MOV.W　src,　dst**

操作　　src　->　dst

说明　　源操作数被移至目的操作数。源操作数不受影响，目的操作数以前的内容丢失。

状态位　　不影响状态位

方式位　　OscOff、CPUOff 和 GIE 不受影响

例子　　地起码表 EDE(字数据)的内容被复制到表 TOM。地址表的长度为 020H

```
        MOV     #EDE,     R10                 ;准备指针
        MOV     #020H,    R9                  ;计数器
Loop    MOV     @R10+,    TOM-EDE-2(R10)      ;将 R10 中的指针用于两个表
        DEC     R9                            ;计数器减 1
        JNZ     Loop                          ;计数器 <>0，继续
        ........                              ;完成
```

MOV.B　源操作数移至目的操作数

语法　　**MOV.B　src,　dst**

操作　　src　->　dst

说明　　源操作数被移至目的操作数。源操作数不受影响，目的操作数以前的内容丢失。

状态位　　不影响状态位

方式位　　OscOff、CPUOff 和 GIE 不受影响

例子　　地起码表 EDE(字节数据)的内容被复制到表 TOM。地址表的长度为 020H

```
        MOV     #EDE,     R10                 ;准备指针
        MOV     #020H,    R9                  ;计数器
Loop    MOV.B   @R10+,    TOM-EDE-2(R10)      ;将 R10 中的指针用于两个表
        DEC     R9                            ;计数器减 1
        JNZ     Loop                          ;计数器 <>0，继续
        ........                              ;完成
```

NOP　空操作

语法　　**NOP**

操作　　无

仿真　　MOV　#0,　#0

说明　　不执行操作。此指令可用于在检查软件期间仿真指令或用于已确定的等待时间。

状态位　　不影响状态位

NOP 指令主要用作两个目的：

(1)、保持一个，两个或三个存储字

(2)、调整软件时序

注意　　其它指令可用于仿真空指令，可用不同数量的周期及不同数量的代码字来仿真空操作指令。

例：

```
MOV     0(R4),      0(R4)       ;6 个周期，3 个字
MOV     @R4,        0(R4)       ;5 个周期，2 个字
BIC     #0,         EDE(R4)     ; 4 个周期，2 个字
JMP     $ + 2                   ; 2 个周期，1 个字
BIC     #0,         R5          ; 1 个周期，1 个字
```

POP[.W]　字从堆栈弹出到目的操作数

语法　　**POP　dst**

操作　　@SP -> dst, SP + 2 ->SP

仿真　　MOV @SP+, dst 或 MOV.W @SP+, dst

说明　　堆栈指针(TOS)指向的栈区字移至目的操作数。随后堆栈指针加 2

状态位　　不影响状态位

例子　　从堆栈恢复 R7 和状态寄存器的内容

```
POP     R7      ;恢复 R7
POP     SR      ;恢复状态寄存器
```

注意　　系统堆栈指针 SP 通常加 2，并与字节后缀无关。这是强制性的，因为系统堆栈指针不仅用于 POP 指令，还可用于 RETI 指令。

POP.B　字节从堆栈弹出到目的操作数

语法　　**POP.B dst**

操作　　@SP -> dst, SP + 2 ->SP

仿真　　MOV.B @SP+, dst

说明　　堆栈指针(TOS)指向的栈区字移至目的操作数。随后堆栈指针加 2

状态位　　不影响状态位

例子　　(1)、从堆栈恢复 RAM 字节 LEO 的内容

```
POP.B   LEO     ;堆栈的低位字节移至 LEO
```

(2)、从堆栈恢复 R7 内容

```
POP.B   R7      ; 堆栈的低位字节移至 R7，R7 的高位字节为 00H
```

(3)、从堆栈恢复状态寄存器和 R7 指向的存储器的内容

```
POP.B   0(R7)   ; 堆栈的低位字节移至 R7 指向的字节
                ;Ex1: R7=203H，  Mem(R7) = 系统堆栈的低字节
                ;Ex2: R7=20AH，  Mem(R7) = 系统堆栈的低字节
POP     SR
```

注意　　系统堆栈指针 SP 通常加 2，并与字节后缀无关。这是强制性的，因为系统堆栈指针不仅用于 POP 指令，还可用于 RETI 指令。

PUSH[.W]　将字压进堆栈

语法　　**PUSH src 或 PUSH.W src**

操作　　SP - 2 -> SP, src -> @SP

说明　　堆栈指针减 2，然后源操作数移至由此指针(TOS)寻址的 RAM 字

状态位　　N：不影响

Z：不影响

C：不影响

V：不影响

方式位　　OscOff、CPUOff 和 GIE 不受影响。

例子　　状态寄存器和 R8 的内容保存在堆栈

```
PUSH    SR      ;保存状态寄存器
PUSH    R8      ;保存 R8
```

注意　　系统堆栈指针 SP 通常减 2，并与字节后缀无关。这是强制性的，因为系统堆栈指针不仅用于 PUSH 指令，还可用于中断服务程序。

PUSH.B　将字节压进堆栈

语法　　**PUSH.B src**

操作　　SP - 2 -> SP, src -> @SP

说明　　堆栈指针减 2，然后源操作数移至由此指针(TOS)寻址的 RAM 字

状态位　　N：不影响

Z：不影响

C：不影响

V：不影响

方式位　OscOff、CPUOff 和 GIE 不受影响。

例子　外设 TCDAT 的内容保存在堆栈

```
PUSH.B    &TCDAT        ;保存 8 位外设模块的数据
                        ;寻址 TCDAT，进栈
```

注意　系统堆栈指针 SP 通常减 2，并与字节后缀无关。这是强制性的，因为系统堆栈指针不仅用于 PUSH 指令，还可用于中断服务程序。

RET　从子程序返回

语法　RET

操作　@SP -> PC,　SP + 2 -> SP

仿真　MOV　@SP+,　PC

说明　由 CALL 指令压进栈的返回地址移至程序计数器。程序在子程序调用后的代码地址处继续执行。

状态位　不影响状态位。

RETI　从中断返回

语法　RETI

操作　TOS -> SR, SP + 2 -> SP,　TOS -> PC,　SP + 2 -> SP

说明　(1)、状态寄存器恢复到中断服务程序开始时的值。用 TOS 存储器中的值替换 SR 中的当前值可做到这一点，堆栈指针 SP 加 2。

(2)、程序计数器恢复到中断服务程序开始时的值。这是中断程序流的后续步骤。用 TOS 存储器中的值替换 PC 的当前值可实现这种恢复，堆栈指针 SP 加 1。

状态位　N：从系统堆栈恢复

Z：从系统堆栈恢复

C：从系统堆栈恢复

V：从系统堆栈恢复

方式位　OscOff、CPUOff 和 GIE 从系统堆栈恢复。

例子

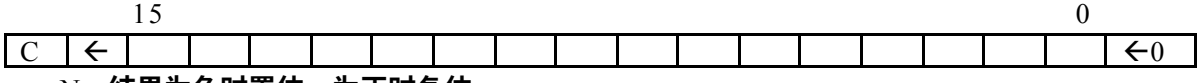
RLA[.W]　算术左移

语法　RLA　dst　或　RLA.W　dst

操作　C <- MSB <- MSB-1...LSB+1 <- LSB <- 0

仿真　ADD　dst,　dst

说明　目的操作数左移一位，MSB 成为进位位 C，LSB 填 0。RLA 指令可当作符号乘 2。在执行该操作前，如果 dst 大于等于 04000H 且小于 0C000H，则产生溢出；结果会改变符号。



状态位　N：结果为负时置位，为正时复位

Z：结果为零时置位，其它情况时复位

C：从 MSB 获取

V：如果产生算术溢出，即初始值为 04000H<= dst < 0C000H 时置位，其它情况时复位

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　R7 乘以 4

```
RLA     R7      ;R7 左移( x2 ) ----由 ADD     R7,     R7   仿真
RLA     R7      ;R7 左移( x4 ) ----由 ADD     R7,     R7   仿真
```

注意　替换 RLA：

汇编语言不识别指令 RLA　@R5+，必须将它替换成 ADD　@R5+,　-2(R5)。

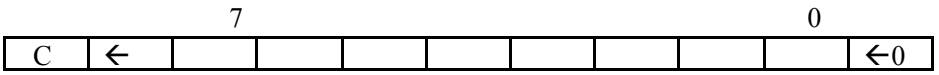
RLA.B　算术左移

语法　RLA.B　dst

操作　C <- MSB <- MSB-1...LSB+1 <- LSB <- 0

仿真　　ADD.B　　dst,　　dst

说明　　**目的操作数左移一位，**MSB**成为进位位** C，LSB **填** 0。RLA **指令可当作符号乘** 2。**在执行该操作前，如果** dst **大于等于** 040H **且小于** 0C0H，**则产生溢出；结果会改变符号。**



状态位　　N：**结果为负时置位，为正时复位**

Z：**结果为零时置位，其它情况时复位**

C：**从** MSB **获取**

V：**如果产生算术溢出，即初始值为** 040H<= dst < 0C0H **时置位，其它情况时复位**

方式位　　OscOff、CPUOff **和** GIE **不受影响**

例子　　R7 **的低位字节乘以** 4

RLA.B　　R7　　;R7 **左移**(x2) ----**由** ADD.B　　R7,　　R7　**仿真**

RLA.B　　R7　　;R7 **左移**(x4) ----**由** ADD.B　　R7,　　R7　**仿真**

注意　　**替换** RLA：

汇编语言不识别指令 RLA.B　　@R5+，**必须将它替换成** ADD.B　　@R5+,　　-1(R5)。

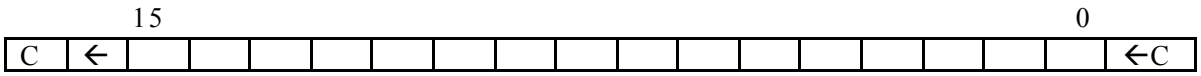
RLC[.W]　通过进位位左移

语法　　**RLC　　dst　　或　RLC.W　　dst**

操作　　C　<-　MSB　<-　MSB-1...LSB+1　<-　LSB　<-　0

仿真　　ADDC　　dst,　　dst

说明　　**目的操作数左移一位，进位位** C **移入** LSB。MSB**移入进位位** C。



状态位　　N：**结果为负时置位，为正时复位**

Z：**结果为零时置位，其它情况时复位**

C：**从** MSB **获取**

V：**如果产生算术溢出时置位，其它情况时复位**

03FFFFH< dst < 0C000H **时置位，其它情况时复位**

方式位　　OscOff、CPUOff **和** GIE **不受影响**

例子　　(1)、R5 **左移一位**

RLC　　R5　　;(R5x2) + C--→R5

(2)、**输入** P0IN.1 **的信息将被移入** R5 **的** LSB。

BIT.B　　#2,　　&P0IN　　**信息-→进位**

RLC　　R5　　;**进位** = P0IN.1 → R5 **的** LSB

注意　　**替换** RLC：

汇编语言不识别指令 RLC　　@R5+，**必须将它替换成** ADDC　　@R5+,　　-2(R5)。

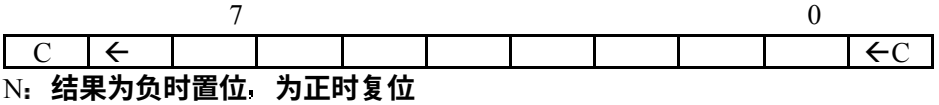
RLC.B　算术左移

语法　　**RLC.B　　dst**

操作　　C　<-　MSB　<-　MSB-1...LSB+1　<-　LSB　<-　0

仿真　　ADDC.B　　dst,　　dst

说明　　**目的操作数左移一位，进位位** C **移入** LSB。MSB**移入进位位** C。



状态位　　N：**结果为负时置位，为正时复位**

Z：**结果为零时置位，其它情况时复位**

C：**从** MSBB **获取**

V：**如果产生算术溢出时置位，其它情况时复位**

03FFH< dst < 0C0H **时置位，其它情况时复位**

方式位　　OscOff、CPUOff **和** GIE **不受影响**

例子 (1)、MEM(LEO)的内容左移一位

```
RLC.B    LEO        ;Mem(LEO) x2 + C → Mem(LEO)
```

(2)、输入 P0IN.1 的信息将被移入 R5 的 LSB。

```
BIT.B    #2,    &P0IN          信息-→进位
RLC.B    R5                    ;进位 = P0IN.1 → R5 的 LSB，R5 的高字节复位。
```

注意 替换 RLC：

汇编语言不识别指令 RLC.B @R5+，必须将它替换成 ADD.B @R5+, -1(R5)。

RRA[.W] 算术右移

语法 **RRA dst 或 RRA.W dst**

操作 MSB -> MSB, MSB -> MSB -1, MSB - 1 -> MSB –2...LSB +1 -> LSB, LSB -> C

说明 目的操作数右移一位，MSB 移入 MSB，MSB 移入 MSB – 1，LSB + 1 移入 LSB。

15 0

C		->	--	--	--	--	--	--	--	--	--	--	--	--	--	--	→	→C

15<--

状态位 N：结果为负时置位，为正时复位

Z：结果为零时置位，其它情况时复位

C：从 LSB 获取

V：复位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 R5 右移一位，MSB 保留原来的值，它相当于算术除 2

```
RRA      R5          ;R5/2 → R5
    ;R5 的值乘以 0.75   ( 0.5 x 0.25 )
PUSH     R5                用堆栈暂时保存 R5
RRA      R5                ;R5 x 0.5 →R5
ADD      @SP+,    R5       ;R5 x 0.5 + R5 = 1.5 x R5 → R5
RRA      R5                ;(1.5 x R5) x 0.5 =  0.75 x R5 →R5
.....
```

;或者

```
RRA      R5                ;R5 x 0.5 →R5
PUSH     R5                用堆栈暂时保存 R5
RRA      @SP               ;TOS x 0.5 = 0.5 x R5 x 0.5 = 0.25 x R5→ TOS
ADD      @SP+,    R5       ;R5 x 0.5 + R5 x 0.25 = 0.75 x R5 → R5
.....
```

RRA.B 算术右移

语法 **RRA.B dst**

操作 MSB -> MSB, MSB -> MSB -1, MSB - 1 -> MSB –2...LSB +1 -> LSB, LSB -> C

说明 目的操作数右移一位，MSB 移入 MSB，MSB 移入 MSB – 1，LSB + 1 移入 LSB。

7 0

C		--	--	--	--	--	--	--	→	→C

7←

状态位 N：结果为负时置位，为正时复位

Z：结果为零时置位，其它情况时复位

C：从 LSB 获取

V：复位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 R5 低字节右移一位，MSB 保留原来的值，它相当于算术除 2

```
RRA.B    R5          ;R5/2 → R5   操作仅针对低字节，R5 高字节复位
```

```
                ;R5 的值(仅低位字节)乘以 0.75   ( 0.5 x 0.25 )
                PUSH.B      R5                  用堆栈暂时保存 R5
                RRA.B       R5                  ;R5 x 0.5 →R5
                ADD.B       @SP+,     R5        ;R5 x 0.5 + R5 = 1.5 x R5 → R5
                RRA.B       R5                  ;(1.5 x R5) x 0.5 =  0.75 x R5 →R5
                .....
;或者
                RRA.B       R5                  ;R5 x 0.5 →R5
                PUSH.B      R5                  用堆栈暂时保存 R5
                RRA.B       @SP                 ;TOS x 0.5 = 0.5 x R5 x 0.5 = 0.25 x R5→ TOS
                ADD.B       @SP+,     R5        ;R5 x 0.5 + R5 x 0.25 = 0.75 x R5 → R5
                .....
```

RRC[.W]　通过进位右移

语法　**RRC　dst　或　RRC.W　dst**

操作　C → MSB -> MSB –1 ... LSB +1 -> LSB -> C

说明　**目的操作数右移一位，进位位 C 移入 MSB，LSB 移入进位位 C。**

15																		0
C	→	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	→C

状态位　N：**结果为负时置位，为正时复位**

Z：**结果为零时置位，其它情况时复位**

C：**从 LSB 获取**

V：**初始目的操作数为正且初始进位位被置时置位，其他情况复位**

方式位　OscOff、CPUOff **和** GIE **不受影响**

例子　R5 **右移一位，MSB 装入** 1

```
SETC                ;为 MSB 准备进位
RRA     R5          ;R5/2 + 8000H → R5
```

RRC.B　通过进位右移

语法　**RRC.B　dst**

操作　C → MSB -> MSB –1 ... LSB +1 -> LSB -> C

说明　**目的操作数右移一位，进位位 C 移入 MSB，LSB 移入进位位 C。**

7									0
C	→	--	--	--	--	--	--	--	→C

状态位　N：**结果为负时置位，为正时复位**

Z：**结果为零时置位，其它情况时复位**

C：**从 LSB 获取**

V：**初始目的操作数为正且初始进位位被置时置位，其他情况复位**

方式位　OscOff、CPUOff **和** GIE **不受影响**

例子　R5 **右移一位，MSB 装入** 1

```
SETC                ;为 MSB 准备进位
RRA.B   R5          ;R5/2 + 80H → R5    使用 R5 的低字节
```

SBC[.W]　从目的操作数减去借位

语法　**SBC　dst　或　SBC.W　dst**

操作　dst + 0FFFFH + C -> dst

仿真　SUBC　#0,　dst

说明　**进位 C 加到减 1 后的目的操作数，目的操作数原来的内容丢失。**

状态位　N：**结果为负时置位，为正时复位**

Z：**结果为零时置位，其它情况时复位**

C：dst 从 0000 减至 0FFFFH 时复位，其它情况时置位

V：初始 C = 0 且减至 dst = 08000H 时置位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 从 R12 指向的 32 位数值中减去 R13 指向的 16 位数值

```
SUB     @R13,    0(R12)      ;减 LSDs
SBC     2(R12)               ;从 MSD 中减去进位
```

注意 借位可视为非进位

借位可视为一种非进位：	借位	进位位
	是	0
	否	1

SBC.B 从目的操作数减去借位

语法 **SBC.B dst**

操作 dst + 0FFF + C -> dst

仿真

```
SUBC.B    #0,    dst
```

说明 进位 C 加到减 1 后的目的操作数，从目的操作数中减去借位，目的操作数原来的内容丢失。

状态位 N：结果为负时置位，为正时复位

Z：结果为零时置位，其它情况时复位

C：dst 从 0000 减至 0FFFFH 时复位，其它情况时置位

V：初始 C = 0 且减至 dst = 080H 时置位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 从 R12 指向的 16 位数值中减去 R13 指向的 8 位数值

```
SUB.B     @R13,    0(R12)      ;减 LSDs
SBC.B     1(R12)               ;从 MSD 中减去进位
```

注意 借位可视为非进位

借位可视为一种非进位：	借位	进位位
	是	0
	否	1

SETC 置进位位

语法 **SETC**

操作 1 -> C

仿真

```
BIS     #1,    SR
```

说明 进位 C 被置，这是一个常用的操作。

状态位 N：不影响

Z：不影响

C：置位

V：不影响

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 十进制减法的仿真：从 R6 中减去 R5 假设 R5=3987 和 R6 = 4137

```
DSUB:   ADD     #6666H,    R5          ;将 R5 的值从 0 ~ 9 移至 6 ~ 0FH
                                       ;R5 = 0397H + 6666H = 09FEDH
        INV     R5                     ;求反 (结果返回 0 ~ 9 )
                                       ;R5 = .NOT. R5  = 06012H
        SETC                           ;进位位 C = 1
        DADD    R5,        R6          ;用加上( 10000 – R5 – 1 )的方法来仿真减法
                                       ;R6 = R6 + R5 + 1
                                       ;R6 = 4137 + 06012 + 1 = 10150 = 0150
```

SETN **置负位**

语法 **SETN**

操作 1 -> N

仿真 BIS #4, SR

说明 **负位** N **被置。**

状态位 N：**置位**

Z：**不影响**

C：**不影响**

V：**不影响**

方式位 OscOff、CPUOff **和** GIE **不受影响**

SETZ **置零位**

语法 **SETZ**

操作 1 -> Z

仿真 BIS #2, SR

说明 **负位** Z **被置。**

状态位 N：**不影响**

Z：**置位**

C：**不影响**

V：**不影响**

方式位 OscOff、CPUOff **和** GIE **不受影响**

SUB[.W] **从目的操作数减去源操作数**

语法 **SUB src, dst 或 SUB.W src, dst**

操作 dst + .NOT.src + 1 -> dst **或** dst - src -> dst

说明 **从目的操作数中减去源操作数。方法是将源操作数求反再加上常数** 1**。源操作数不受影响，目的操作数以前的内容丢失。**

状态位 N：**结果为负时置位，为正时复位**

Z：**结果为零时置位，其它情况时复位**

C：**结果的** MSB **产生进位时置位，否则复位。无借位时置** 1**，有借位时复位**

V：**发生算术溢出时置位，其它情况时复位**

方式位 OscOff、CPUOff **和** GIE **不受影响**

例子

注意 **借位可视为非进位**

借位可视为一种非进位：

借位	**进位位**
是	0
否	1

SUB.B **从目的操作数减去源操作数**

语法 **SUB.B src, dst**

操作 dst + .NOT.src + 1 -> dst **或** dst - src -> dst

说明 **从目的操作数中减去源操作数。方法是将源操作数求反再加上常数** 1**。源操作数不受影响，目的操作数以前的内容丢失。**

状态位 N：**结果为负时置位，为正时复位**

Z：**结果为零时置位，其它情况时复位**

C：**结果的** MSB **产生进位时置位，否则复位。无借位时置** 1**，有借位时复位**

V：**发生算术溢出时置位，其它情况时复位**

方式位 OscOff、CPUOff **和** GIE **不受影响**

例子

注意 **借位可视为非进位**

借位可视为一种非进位：　借位　　进位位

是　　0

否　　1

SUBC[.W]，SBB[.W]　　从目的操作数减去源操作数和借位/非进位

语法　　**SUBC　　src,　　dst　　或　SUBC.W　　src,　　dst**

SUBB　　src,　　dst　　或　SUBB.W　　src,　　dst

操作　　dst + .NOT.src + C -> dst　**或**　dst - src -1 + C -> dst

说明　　从目的操作数中减去源操作数。方法是将源操作数求反再加上进位 C。源操作数不受影响，目的操作数以前的内容丢失。

状态位　　N：结果为负时置位，为正时复位

Z：结果为零时置位，其它情况时复位

C：结果的 MSB 产生进位时置位，否则复位。无借位时置 1，有借位时复位

V：发生算术溢出时置位，其它情况时复位

方式位　　OscOff、CPUOff 和 GIE 不受影响

例子　　减去两个浮点尾数

LSB 分别位于 R13 至 R10，而 MSB 分别位于 R12 至 R9。

```
SUB.W     R13,     R10          ;16 位，LSB
SUBC.B    R12,     R9           ;8 位，MSB
```

注意　　借位可视为非进位

借位可视为一种非进位：　借位　　进位位

是　　0

否　　1

SUBC.B，SBB.B　　从目的操作数减去源操作数和借位/非进位

语法　　**SUBC.B　　src,　　dst　　或　SUBC.B　　src,　　dst**

SUBB.B　　src,　　dst　　或　SUBB.B　　src,　　dst

操作　　dst + .NOT.src + C -> dst　**或**　dst - src -1 + C -> dst

说明　　从目的操作数中减去源操作数。方法是将源操作数求反再加上进位 C。源操作数不受影响，目的操作数以前的内容丢失。

状态位　　N：结果为负时置位，为正时复位

Z：结果为零时置位，其它情况时复位

C：结果的 MSB 产生进位时置位，否则复位。无借位时置 1，有借位时复位

V：发生算术溢出时置位，其它情况时复位

方式位　　OscOff、CPUOff 和 GIE 不受影响

例子　　从 R10 和 R11(MSD)16 位数值中减 R13 指向的 16 位数值

```
SUB.B     @R13,    R10          ;不带进位减去 LSDs
SUBC.B    @R13,    R11          ;带进位减去 MSDs，由 LSDs 引起
```

注意　　借位可视为非进位

借位可视为一种非进位：　借位　　进位位

是　　0

否　　1

SWPB　　交换字节

语法　　**SWPB　　dst**

操作　　位 15 至 8　->　位 7 至 0

说明　　目的操作数的高位字节和低位字节互换。

状态位　　N：不影响

Z：不影响

C：不影响

V：不影响

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 (1)、MOV #040BFH, R7 ;01000000 10111111 → R7

SWPB R7 ;R7 中的 10111111 01000000

(2)、R5 中的值乘以 256，结果保存在 R5、R4 中。

```
SWPB    R5
MOV     R5,     R4      ;将交换后的值复制到 R4
BIC     #0FF00H, R5     ;修正结果
BIC     #000FFH, R5     ;修正结果
```

SXT 扩展符号

语法 SXT dst

操作 位 7 -> 位 8.....位 15

说明 低位字节的符号扩展到高位字节。

状态位 N：结果为负时置位，为正时复位

Z：结果为零时置位，其它情况时复位

C：结果为非零时置位，其它情况时复位

V：复位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 R7 中装入定时器/计数器的值。符号扩展指令用位 7 的值将位 8 扩展到位 15，然后 R7 加至累加的 R6。

```
MOV.B   &TCDAT,  R7     ;TCDAT = 080H
SXT     R7              ;R7 = 0FF80H
ADD     R7,     R6      ;
```

TST[.W] 测试目的操作数

语法 TST dst 或 TST.W dst

操作 dst + 0FFFFH + 1

仿真 CMP #0, dst

说明 比较目的操作数和 0 。根据结果设置状态位。目的操作数不受影响。

状态位 N：结果为负时置位，为正时复位

Z：目的操作数包含 0 时置位，其它情况时复位

C：置位

V：复位

方式位 OscOff、CPUOff 和 GIE 不受影响

例子 测试 R7，如果为负，继续执行 R7NEG；如果为正且不为 0 ，继续执行 R7POS。

```
        TST     R7              ;测试 R7
        JN      R7NEG           ;R7 为负
        JZ      R7ZERO          ;R7 = 0
R7POS:  .....                   ;R7 为正且不为 0
        .....
R7NEG:  .....                   ;R7 为负
        .....
R7ZERO: .....                   ;R7 = 0
```

TST.B 测试目的操作数

语法 TST.B dst

操作 dst + 0FFH + 1

仿真 CMP.B #0, dst

说明 比较目的操作数和 0 。根据结果设置状态位。目的操作数不受影响。

状态位　N：结果为负时置位，为正时复位

Z：目的操作数包含 0 时置位，其它情况时复位

C：置位

V：复位

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　测试 R7 的低位字节，如果为负，继续执行 R7NEG；如果为正且不为 0 ，继续执行 R7POS。

```
        TST.B   R7              ;测试 R7 的低位字节
        JN      R7NEG           ;R7 的低位字节为负
        JZ      R7ZERO          ;R7 的低位字节为 0
R7POS:  .....                   ;R7 的低位字节为正且不为 0
        .....
R7NEG:  .....                   ;R7 的低位字节为负
        .....
R7ZERO: .....                   ;R7 的低位字节为 0
```

XOR[.W]　源操作数和目的操作数异或

语法　**XOR　src,　dst　或 XOR.W　src,　dst**

操作　src .XOR. dst -> dst

说明　源操作数和目的操作数异或，其结果放放目的操作数。源操作数不受影响。

状态位　N：结果的 MSB 为 1 时置位，为 0 时复位

Z：结果为 0 时置位，其它情况时复位

C：结果不为 0 时置位，其它情况时复位 (= .NOT. Zero)

V：两个操作数均为负时置位

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　R6 中被置的位触发 RAM 字 TONI 中的各位。

```
        XOR     R6,     TONI    ;根据 R6 中所置的位触发 TONI 字中的各位。
```

XOR.B　源操作数和目的操作数异或

语法　**XOR.B　src,　dst**

操作　src .XOR. dst -> dst

说明　源操作数和目的操作数异或，其结果放放目的操作数。源操作数不受影响。

状态位　N：结果的 MSB 为 1 时置位，为 0 时复位

Z：结果为 0 时置位，其它情况时复位

C：结果不为 0 时置位，其它情况时复位 (= .NOT. Zero)

V：两个操作数均为负时置位

方式位　OscOff、CPUOff 和 GIE 不受影响

例子　(1)、R6 中被置的位触发 RAM 字 TONI 中的各位。

```
        XOR.B   R6,     TONI    ;根据 R6 的低位字节中所置的位触发 TONI 字中的各位。
```

(2)、将 R7 的低位字节到 0 中不同于 RAM 字节 EDE 的各位复位

```
        XOR.B   EDE,    R7      ;设置不同的位为 1
        INV.B   R7              ;求反低位字节，高位字节为 0
```

==========