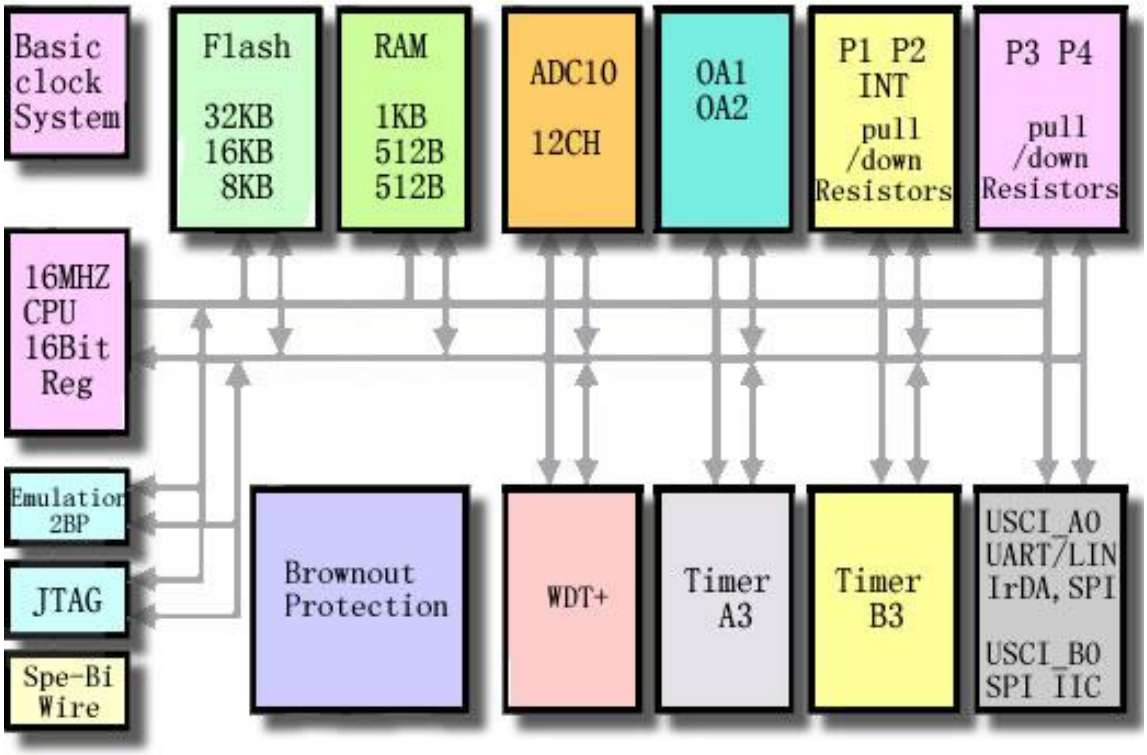
MSP430F2 系列 16 位超低功耗单片机模块原理

第 3 章 MSP430 CPU

版本: 1.3
日期: 2007.6.
原文: TI MSP430x2xxfamily.pdf
翻译: 袁德纯
编辑: DC 微控技术论坛版主

注：以下文章是翻译 TI MSP430x2xxfamily.pdf 文件中的部分内容。由于我们翻译水平有限，有整理过程中难免有所不足或错误；所以以下内容只供参考.一切以原文为准。
详情请密切留意微控技术论坛。



第三章 16位精简指令集计算机系统

本章节将介绍430的16CPU系统、寻址模式和指令系统
目录：
绪论

CPU的寄存器
指令系统

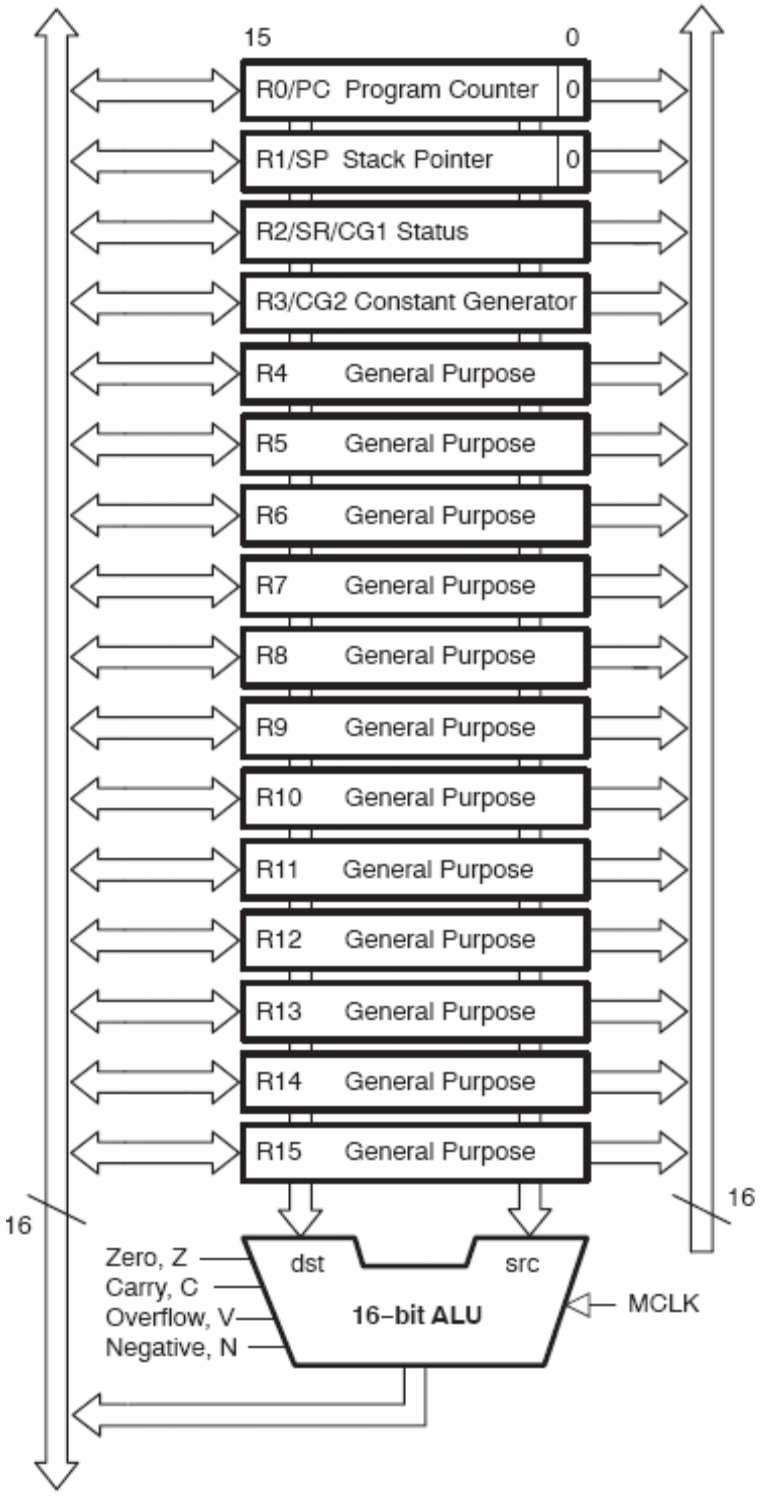
3.1 绪论

16位CPU的一体化系统明确的面向模块化的程序设计技术，如适当的程序分支、表格处理和可应用高级语言，如C语言。CPU能在不用进行页面调整的时候寻址整个地址空间

CPU的主要性质：

- 带有27条指令和7种寻址模式的精简指令集架构
- 每条指令能够采用任何一种寻址模式的交互式系统结构
- 完整的寄存器系统包括：程序计数器、状态寄存器、和堆栈的指针；
- 单周期的寄存器操作
- 16位寄存器操作简化了存储器的存取
- 16位的地址总线允许直接存取和在整个寄存器范围内的任意分支
- 16为的数据总线允许直接的更大范围的数据的处理
- 常数发生器能够产生多达6种常数
- 不需要寄存器的情况下便可进行存储器间的传输
- 字节和双字节的寻址和指令格式

MDB（存储器的数据总线）　　MAB（存储器的地址总线）



CPU的方框图如图**3-1**所示

注：**R0/PC Program Counter**：**R0/PC** 程序计数器
R1/SP Stack Pointer：**R1/SP** 堆栈指针

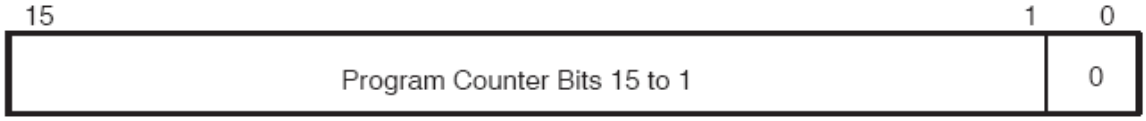
R2/SR/CG1 Status：**R2/SR/CG1** 状态
R3/CG2 Constant Generator：**R3/CG2** 常数发生器
General Purpose：普通用法

3.2 CPU的寄存器

16位的**CPU**整合了**16**位的寄存器。**R0**到**R3**用于特定用法，**R4**到**R15**用于普通用法。

3.2.1 程序计数器

程序计数器指向下一条将要被执行的指令。每条指令用偶（**2**，**4**，**6**，**8**）字节指令方式执行程序计数器增加。指令在**64KB**的地址空间中存取数据扮演着偶地址分界线的角色，程序计数器以偶地址方式排列。如图**3-2**：



程序计数器能够在任意的一种寻址模式种被任一条指令所寻址
例：

```
MOV #LABEL,PC ;
MOV LABEL,PC ;
MOV @R14,PC ;
```

3.2.2 堆栈指针

堆栈指针用于存放子程序的调用和中断子程序的返回地址。堆栈指针以先进后出的方式执行
堆栈指针能够在任意的一种寻址模式种被任一条指令所寻址，如图**3-2**所示。程序计数器以偶地址方式排列，用户能够使用堆栈指针初始化**RAM**。堆栈的用法如图**3-4**所示
例程：

```
MOV 2(SP),R6 ; I2 -> R6
MOV R7,0(SP) ; Overwrite TOS with R7
PUSH #0123h ; Put 0123h onto TOS
POP R8 ; R8 = 0123h
```

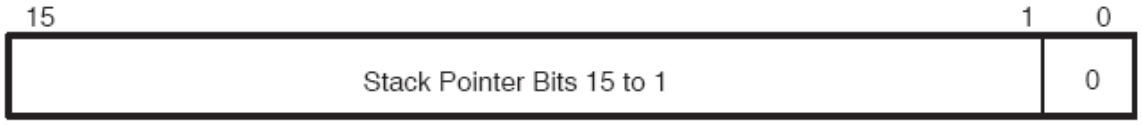


图**3-3**

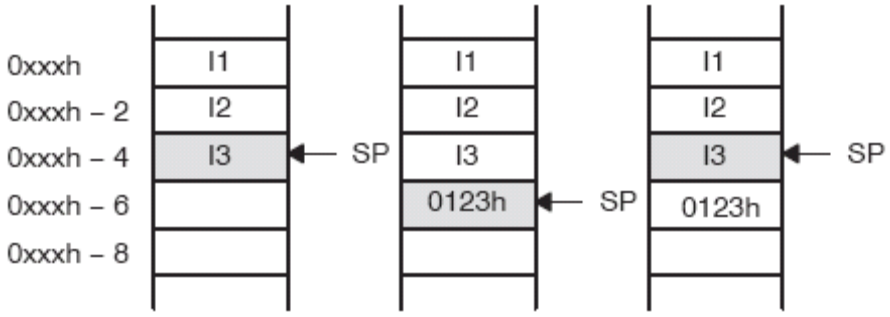


图**3-4**

图**3-5**描述了堆栈指针的程序进栈和出栈的顺序

图a 图b

图3-5

图a 在执行PUSH SP 指令后，堆栈指针将改变

图b 执行指令POP SP 后，堆栈指针不改变，指令POP SP将SP1压入指针位置（SP2=SP1）

3.2.3 状态寄存器

状态寄存器可作为原始寄存器和目的寄存器，只能在寄存器的模式下以双字节的方式进行存取。剩下的存取模式用于支持常数发生器，状态寄存器的各个位如图3-6所示：

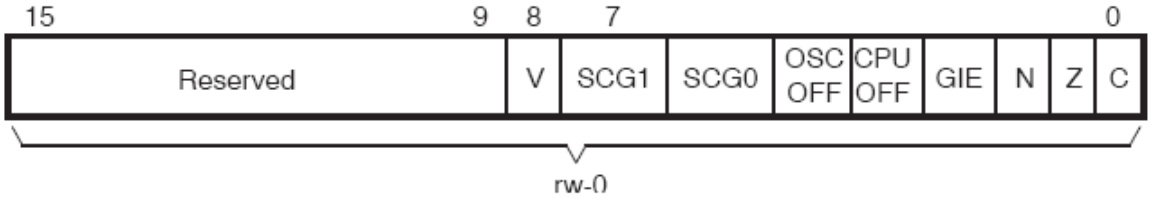


图3-6

表3-1描述了状态寄存器的各个位

位	描述
V	溢出标志位。程序运行过程中，结果超出规定范围的时候，此位将会被置位 ADD(.B),ADDC(.B) SUB(.B),SUBC(.B),CMP(.B)
SCG1	系统时钟发生器1。当此位置1时，SMCLK将会被关闭
SCG0	系统时钟发生器0。如果DCOCL没有用于MCLK或者SMCLK，当此位被置位时，将关闭DCO的直流发生器
OSCOFF	晶振关闭位。此位被置1时，当LFXT1CLK没有被用于MCLK或者SMCLK，LFXT1将会被关闭
CPUOFF	CPU关闭位。当此位被置1时，将会关闭CPU
GIE	中断使能位。此位被置1时，所有的中断将会被打开。复位时，所有的中断均不可用
N	负数位。当指令的操作结果是负数时，此位被设置时；当结果不为负数时，此位被清除
Z	零标志位。结果为0时，此位被设置；结果不为0时，此位被清除
C	进位标志位。

3.2.4 常数发生器CG1和CG2

六个普通用途的常数可由常数发生器的寄存器R2和R3产生，而不需要一个额外的双字节程序指令。常数由源寄存器的寻址模式所选择。如表3-2所示

Register	As	Constant	Remarks
R2	00	– – – – –	Register mode
R2	01	(0)	Absolute address mode
R2	10	00004h	+4, bit processing
R2	11	00008h	+8, bit processing
R3	00	00000h	0, word processing
R3	01	00001h	+1
R3	10	00002h	+2, bit processing
R3	11	0FFFFh	–1, word processing

常数发生器的优点在于：

没有特殊的指令需求

六个常数不需要额外的代码指令

No code memory access required to retrieve the constant

如果六个常数中的一个被用于立即源操作数，汇编程序将会自动使用常数发生器。**R2**和**R3**在用于常数模式的时候，**R2**和**R3**将不能够被明确的被寻址。它们仅作为源寄存器。

常数发生器的扩展指令设置：

精简指令集的**M430**单片机仅有**27**条系统指令。但是常数发生器允许**MSP430**的汇编程序使用附加的**24**条仿真指令。例如单操作数指令 **CLR dst**被双操作数指令**MOV R3,dst**所仿效，它们具有相同的深度。当**#0**被汇编程序所替代时，**R3**被用作**AS=00**。指令**ADD 0(R3),dst**取代指令**INC dst**

3.2.5 普通寄存器R4-R15

R4-R15的**12**个寄存器都是普通用法的寄存器。这些所有的寄存器均可被作为数据寄存器、地址指针、或者索引值都可以被字节指令或者字指令所寻址。如图**3-7**所示

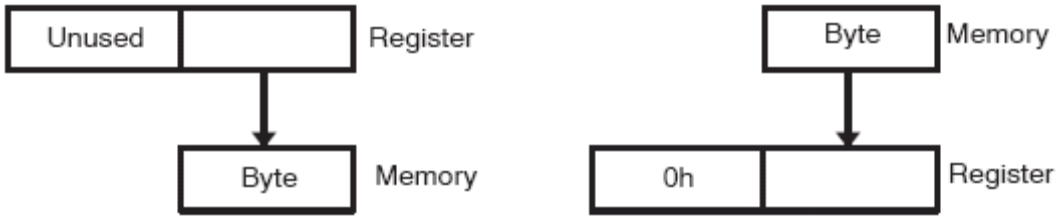


图**3-7**

Example Register-Byte Operation

R5 = 0A28Fh
R6 = 0203h
Mem(0203h) = 012h

```
ADD.B     R5,0(R6)

    08Fh
  + 012h
  -------
    0A1h
```

Mem (0203h) = 0A1h
C = 0, Z = 0, N = 1

(寄存器低字节)
+ (地址字节位)
->(地址字节位)

Example Byte-Register Operation

R5 = 01202h
R6 = 0223h
Mem(0223h) = 05Fh

```
ADD.B     @R6,R5

    05Fh
  + 002h
  -------
  00061h
```

R5 = 00061h
C = 0, Z = 0, N = 0

(地址字节位)
+ (寄存器低字节)
->(寄存器低字节，0存放在寄存器高字节)

3.3 寻址方式

七种源操作数寻址模式和四种目的操作数寻址模式能够访问所有的地址空间。表**3-3**描述了**AS**和**AD**模式中的各个位

As/Ad	寻址模式	语法结构	描述
00/0	寄存器模式	Rn	寄存器内容是操作数
01/1	索引模式	X(Rn)	(Rn + X) points to the operand. X is stored in the next word.
01/1	符号	ADDR	(PC + X) points to the operand. X is stored in the next word. Indexed mode X(PC) is used.
01/1	绝对模式	&ADDR	The word following the instruction contains the absolute address. X is stored in the next word. Indexed mode X(SR) is used.
10/–	间接寄存器模式	@Rn	Rn is used as a pointer to the operand.
11/–	间接自动增量	@Rn+	Rn is used as a pointer to the

			operand. Rn is incremented afterwards by 1 for .B instructions and by 2 for .W instructions.
11/–	立即模式	#N	The word following the instruction contains the immediate constant N. Indirect autoincrement mode @PC+ is used.

在下面的内容中将为大家具体介绍七种寻址模式。
注：标签 EDE，TONI，TOM，和 LEO的用法
在整篇文档中EDE，TONI，TOM,和 LEO仅仅作为普通的标签而没有特殊的含义

3.3.1 寄存器寻址模式

寄存器寻址模式如表3-4种所示

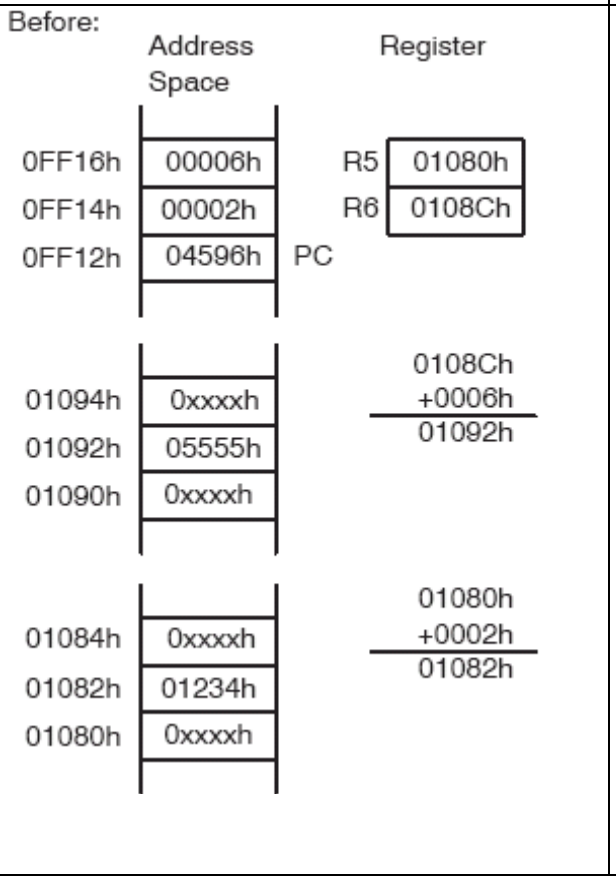
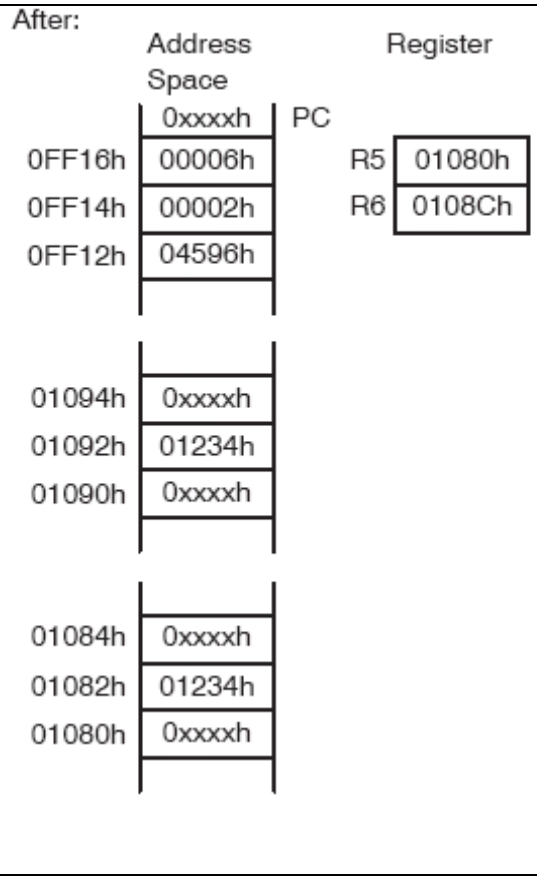
汇编代码	ROM中的内容
MOV R10 R11	MOV R10 R11
长度	一个或者两个字节
操作	将R10中的内容移动到R11中，R10和R11不会被影响
内容	源数据和目的数据均为有效数据
例子	MOV R10 R11
Before: R10 0A023h R11 0FA15h PC PC_{old}	After: R10 0A023h R11 0A023h PC PC_{old} + 2

表3-4

注：寄存器中的数据
寄存器中的数据能够被字节或者字指令访问，如果一个字节指令被使用，那么高字节位总是为零。状态位将依据字节指令的结果去操作

3.3.2 被变址寻址模式

寄存器的变址寻址模式如表3-5中所示

汇编代码	ROM中的内容
MOV 2（R5） 6（R6）	MOV X（R5） Y（R6） X=2，Y=6
长度	两个或者三个字节
操作	将源地址中的内容移动到目的地址中去，源地址和目的地址均不会被影响。在变址寻址模式，程序计数器将会自动地增加，所以程序能够继续运行下一条指令
内容	源数据和目的数据均为有效数据
例子	MOV 2（R5） 6（R6）
	

3.3.3 偏移量可变址寻址模式

可变址寻址模式如表格3-6中所示

汇编代码	ROM中的内容
MOV EDE,TONI	MOV X(PC),Y(PC) X = EDE – PC Y = TONI – PC
长度	两个或者三个双字节
操作	将源地址EDE(PC + X)中的内容移动到目的地址TONI(PC + Y)中去。在指令执行后PC

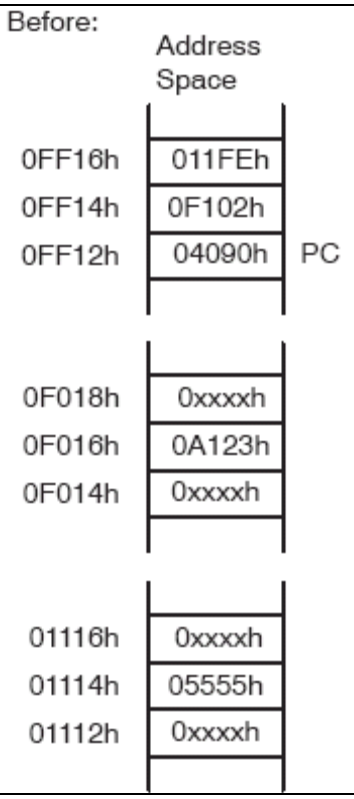
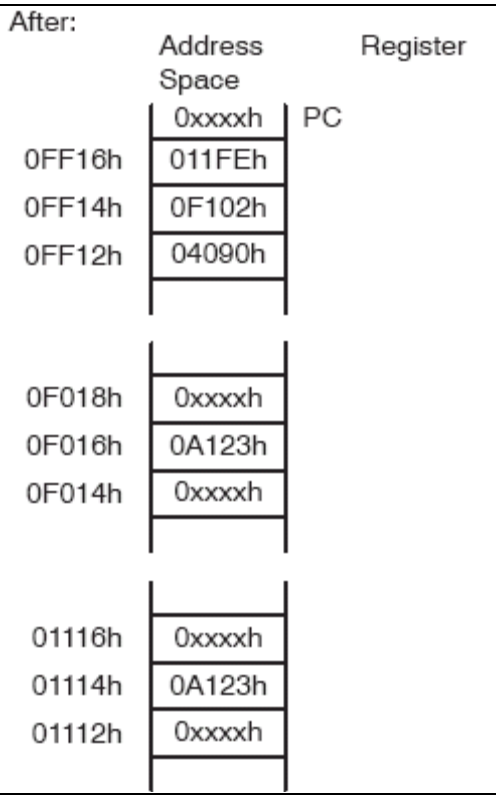
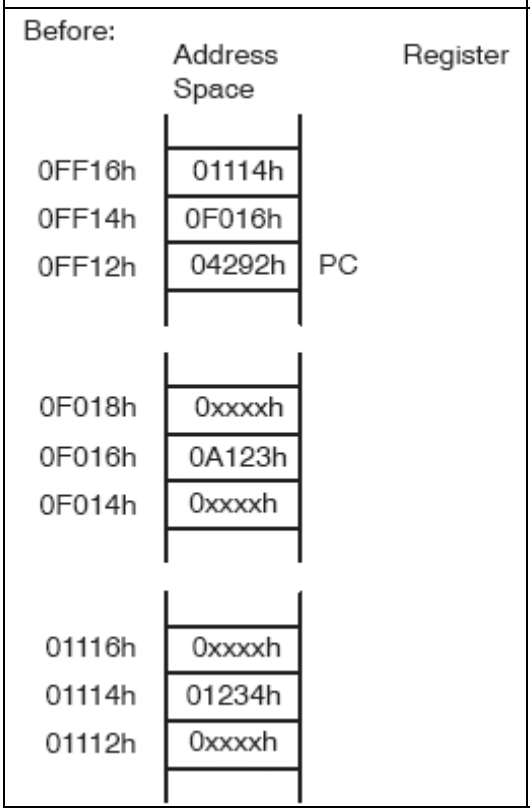
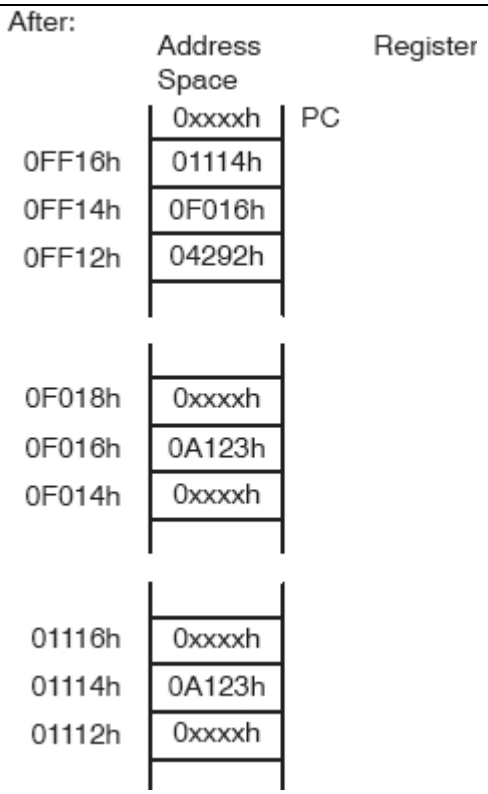
	中的源地址和目的地址中的内容均发生了改变。在汇编的过程中自动插入了偏移量**X、Y.**在偏移量可变址寻址模式，程序计数器将会自动地增加，所以程序能够继续运行下一条指令
内容	源数据和目的数据均为有效数据
例子	**MOV EDE,TONI** 源地址：**EDE = 0F016h** 目的地址：**TONI=01114h**
Before:  	After: 

表**3-6**

3.3.4 绝对寻址模式

汇编代码	**ROM**中的内容
MOV &EDE,&TONI	**MOV X(0),Y(0)** **X = EDE** **Y = TONI**
长度	两个或者三个双字节
操作	将源地址**EDE**中的内容移动到目的地址**TONI**中去。在指令执行后命令中包含了源地址和目的地址中的绝对地址。绝对寻址模式中，程序计数器将会自动地增加，所以程序能够继续运行下一条指令
内容	源数据和目的数据均为有效数据

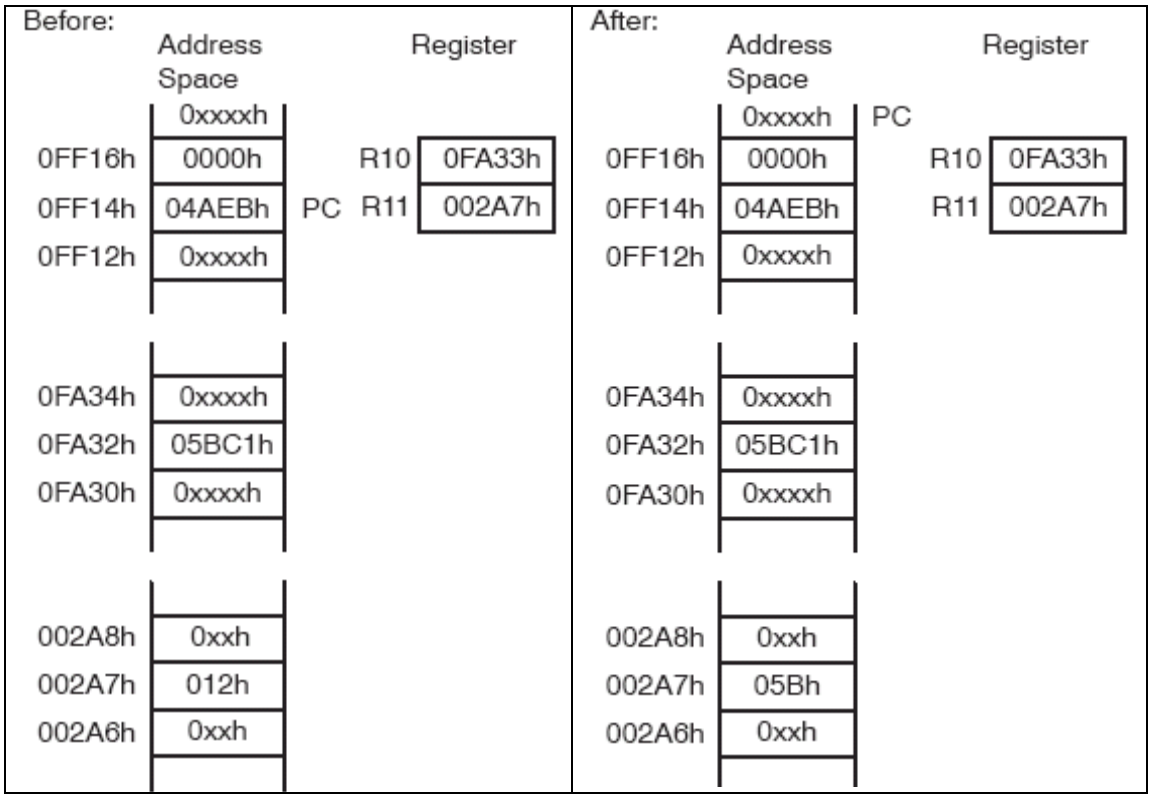
例子	MOV &EDE,&TONI 源地址：EDE = 0F016h 目的地址：TONI=01114h
	

这种地址模式主要是针对那些位于固定、绝对的地址的外围设备。这些外围模块用绝对寻址模式去寻址从而保证了程序运行的流畅性。

3.3.5寄存器间接寻址模式

寄存器间接寻址模式的具体描述如表**3-8**

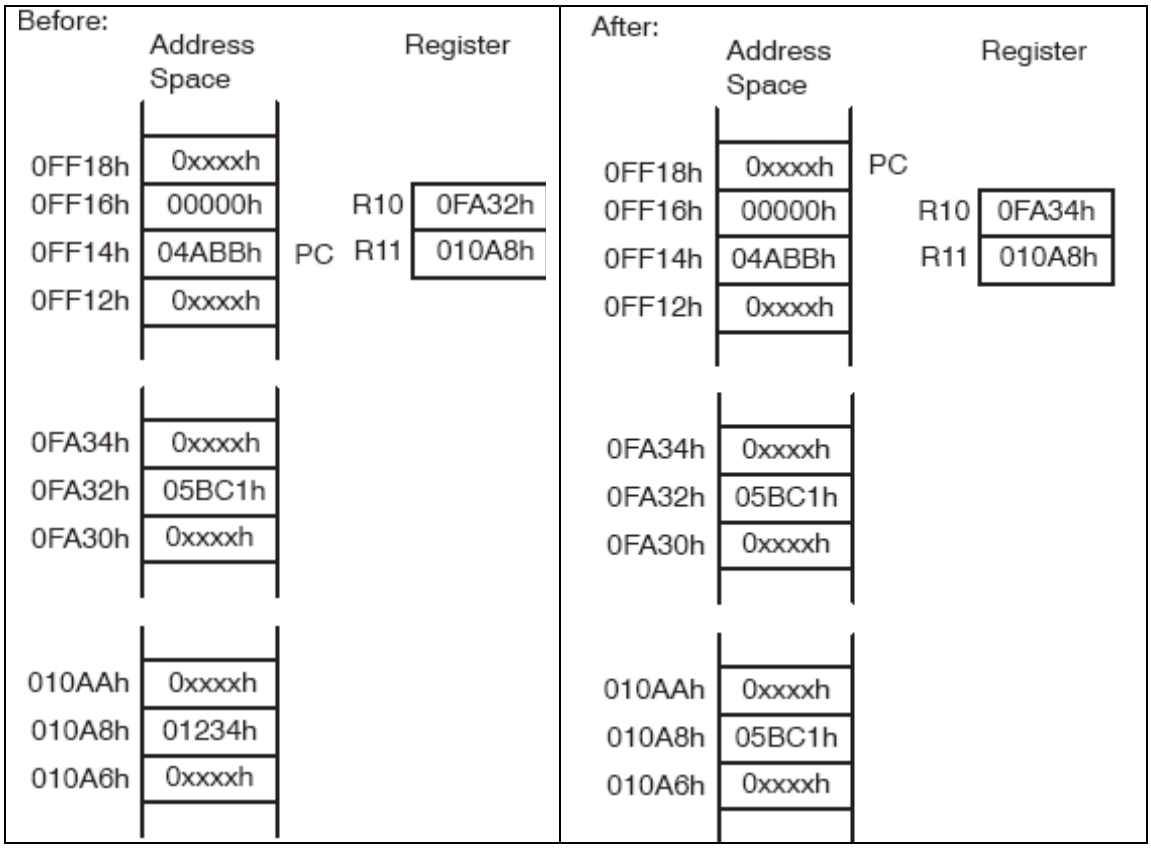
汇编代码	ROM中的内容
MOV @R10,0(R11)	MOV @R10,0(R11)
长度	两个或者三个双字节
操作	将源地址中的内容移动到目的地址中去。寄存器没有被改变
内容	源数据为有效数据。目的操作数被0(Rd)所取代
例子	MOV.B @R10,0(R11)



3.3.5 间接自动增量寻址模式

间接自动增量寻址模式如表**3-9**所示

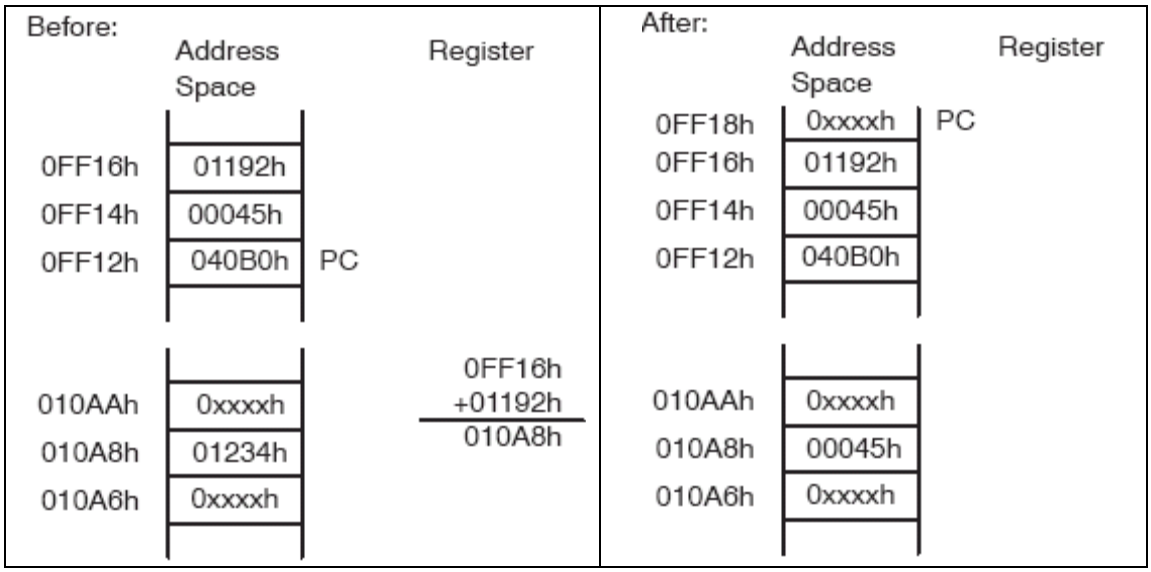
汇编代码	**ROM**中的内容
MOV @R10+,0(R11)	**MOV @R10+,0(R11)**
长度	两个或三个双字节
操作	将源地址中的内容移动到目的地址中去。寄存器没有被改变。在字节指令执行取数后，寄存器**R10**将自动加**1**；在双字节指令执行取数后，将自动加**2**；在地址没有溢出的情况下，将会自动指向下一个地址单元。这对于表格的处理非常方便
内容	源数据为有效数据。目的操作数被**0(Rd)**和下一个操作指令**INCD Rd**所取代
例子	**MOV @R10+,0(R11)**



3.3.7 立即数寻址模式

立即数寻址模式如表**3-9**所示

汇编代码	**ROM**中的内容
MOV #45h,TONI	**MOV @PC+,X(PC)** **45** **X = TONI – PC**
长度	两个或者三个双字节
操作	将常量立即数**45**移动到目的地址**TONI**中去。当取完数据时，程序计数器将自动指向下一条指令，并且将其中的内容移动到目的寄存器中
内容	源数据为有效数据。
例子	**MOV #45h,TONI**



3.4 指令系统

430的指令系统包括27条核心指令和24条仿真指令。核心指令由CPU完成相应的编码和解码工作。而仿真指令能够使得代码容易的进行读写，他们自己不能进行编码，但是他们能够自动的被相应的核心指令所代替从而进行汇编。

核心指令有三种方式：

双操作数指令
单操作数指令
跳转指令

所有的单操作数指令和双操作数指令能够被字节指令或者双字节指令以.B或者.W的方式进行扩展.字节指令可以访问数据的字节单位或者外围设备的字节地址。双字节指令可以访问双字节的数据单位的或者双字节地址的外围设备.如果指令没有被扩展，那么指令就是一个双字节指令。

源指令和目的指令在以下范围所定义：

SRC 源操作数被**As**和**S-reg**所定义
DST 目的操作数被**Ad**和**D-reg**所定义
AS 寻址位（依赖于所使用的寻址模式）
S-REG 用于源操作指令的工作寄存器
AD 寻址位（依赖于所使用的寻址模式）
D-REG 用于目的操作指令的寄存器
B/W 字节或者双字节操作位
0: 双字节模式 **1:** 字节模式

注意：目的地址

在任何的内存映射中，目的地址都是有效的.但是在用一条指令去修改目的寄存器的内容时，用户必须确保目的地址是可写的。例如：一个被屏蔽的**ROM**单位是一个有效的目的地址，但是里面存储的内容是不可修改的，所以指令的操作结果将会丢失。

3.4.1 双操作数指令

图**3-9**阐述了双操作数指令的方式

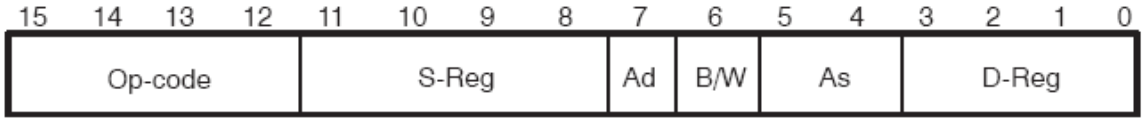


图3-9

表3-11列举和描述了一些双操作数指令

Mnemonic	SReg DReg	Operation	Status Bits V N Z C
MOV(.B)	src,dst	src →dst	– – – –
ADD(.B)	src,dst	src + dst →dst	* * * *
ADDC(.B)	src,dst	src + dst + C→dst	* * * *
SUB(.B)	src,dst	dst + .not.src + 1 →dst	* * * *
SUBC(.B)	src,dst	dst + .not.src + C→dst	* * * *
CMP(.B)	src,dst	dst – src	* * * *
DADD(.B)	src,dst	src + dst + C →dst (decimally)	* * * *
BIT(.B)	src,dst	src .and. dst	0 * * *
BIC(.B)	src,dst	.not.src .and. dst→dst	– – – –
BIS(.B)	src,dst	src .or. dst →dst	– – – –
XOR(.B)	src,dst	src .xor. dst →dst	* * * *
AND(.B)	src,dst	src .and. dst →dst	0 * * *

* 被影响的状态位
– 没有被影响的状态位
0 被清除的状态位
1 被置1的状态位

表3-11

注意：指令CMP和SUB
指令CMP和SUB不保存结果，指令BIT和AND也是同样如此

3.4.2 单操作数指令

图3-10阐述了单操作数指令的格式：

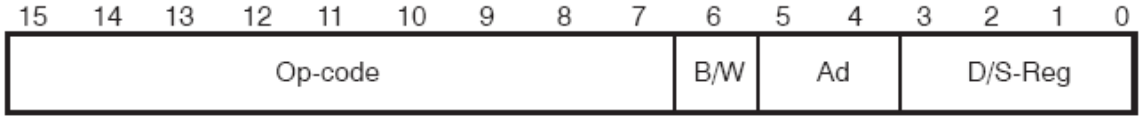


图3-10

表3-12描述和列举了一些单操作数指令

Mnemonic	SReg DReg	Operation	Status Bits V N Z C
RRC(.B)	dst	C→MSB→.....LSB →C	* * * *
RRA(.B)	dst	C→MSB→.....LSB →C	0 * * *
PUSH(.B)	src	SP – 2→SP, src→@SP	– – – –
SWPB	dst	Swap bytes	– – – –
CALL	dst	SP – 2→SP, PC+2 →@SP dst →PC	– – – –
RETI		TOS →SR, SP + 2 →SP	* * * *

		TOS →PC,SP + 2 →SP	
SXT	dst	Bit 7 →Bit 8.....Bit 15	0 * * *

* 被影响的状态位
– 没有被影响的状态位
0 被清除的状态位
1 被置1的状态位

表3-12

所有的寻址模式都有可能适合于CALL指令。如果Symbolic 模式(ADDRESS)，立即数寻址模式，完全寻址模式或者变址寻址模式被使用，那么接下来的字中含有地址的相关信息。

3.4.3 跳转指令

图3-11阐述了条件跳转指令的格式：

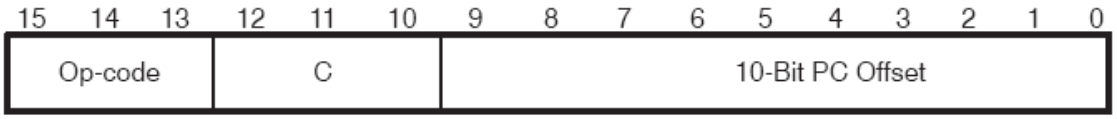


图3-11

表3-13描述和列举了一些跳转指令

Mnemonic	SReg, DReg	Operation
JEQ/JZ	Label	零标志位被设置则跳转
JNE/JNZ	Label	零标志位被复位则跳转
JC	Label	进位标志位为1则跳转
JNC	Label	进位标志位为0则跳转
JN	Label	负数标志位为1则跳转
JGE	Label	(N .XOR. V) = 0则跳转
JL	Label	(N .XOR. V) = 0则跳转
JMP	Label	无条件跳转指令

条件转移指令支持分支程序的设计而不影响状态位。在转移指令中跳转的范围在相对于PC的–511-- +512之间。一个10位的程序计数器偏移量作为一个单的10位的数乘2后被加到程序计数器中

PCnew = PCold + 2 + PCoffset x 2

* ADC[.W]

把进位标志位的加到目的操作数中

* ADC.B

把进位标志位加到目的操作数中

语法格式

ADC dst or ADC.W dst

ADC.B dst

操作

dst + C –> dst

仿真指令

ADDC #0,dst

ADDC.B #0,dst

描述
把进位标志位加到目的操作数中。目的操作数中先前的内容将会丢失.
状态位
N: 如果结果是负数将被置**1**，真数则置**0**
Z: 结果为**0**则置**1**，否则将置**0**
C: 目的操作数进位则置**1**，否则为**0**
V: 算术溢出则置位，否则置**0**
模式选择位
OSCOFF, CPUOFF, 和 **GIE** 不会被影响.
例子
指向**R13**的**16**位的计数器指针加到指向**R12**的**32**位计数器指针中去.
ADD @R13,0(R12) ; 低位相加
ADC 2(R12) ; 进位标志位加到**R12**的高位
例子
指向**R13**的**8**位的计数器指针加到指向**R12**的**16**位计数器指针中去.
ADD.B @R13,0(R12) 低位相加
ADC.B 1(R12) ; 进位标志位加到**R12**的高位

ADD[.W] 将源操作数加到目的操作数中
ADD.B 将源操作数加到目的操作数中
语法格式
ADD src,dst 或者 **ADD.W src,dst**
ADD.B src,dst
操作
src + dst -> dst
描述
源操数被加到目的操作数中，源操作数不受影响，先前目的操作数中的内容将会被覆盖
状态标志位
N: 如果结果是负数将被置**1**，真数则置**0**
Z: 结果为**0**则置**1**，否则将置**0**
C: 目的操作数进位则置**1**，否则为**0**
V: 算术溢出则置位，否则置**0**
模式选择位
OSCOFF, CPUOFF, 和 **GIE** 不会被影响.
例子
R5增加**10.** 如果进位标志位被置位则跳转到**TONI.**
ADD #10,R5
JC TONI ; 发生了进位
..... ; 没有发生进位
例子
R5 增加**10.**进位则跳转到**TONI.**
ADD.B #10,R5 ; R5的底字节加**10**
JC TONI ; 发生进位，如果**(R5)** ≥**246 [0Ah+0F6h]**
..... ; 未发生进位

ADDC[.W] 将源操作数和进位标志位加到目的操作数中
ADDC.B 将源操作数和进位标志位加到目的操作数中
语法格式
ADDC src,dst 或者**ADDC.W src,dst**
ADDC.B src,dst
操作
src + dst + C -> dst
描述
将源操作数和进位标志位加到目的操作数中。
将源操作数和进位标志位加到目的操作数中。
状态标志位
N: 如果结果是负数将被置**1**，真数则置**0**
Z: 结果为**0**则置**1**，否则将置**0**
C: 目的操作数进位则置**1**，否则为**0**
V: 算术溢出则置位，否则置**0**
模式选择位
OSCOFF, CPUOFF, 和 **GIE** 不会被影响**.**
例子
32位指向**R13**的计数器加到一个**32**位的计数器上**,**十一个字位于寄存器**R13**的高位
ADD @R13+,20(R13) ;加**LSD**不带进位
ADDC @R13+,20(R13) ;加**MSD**带进位
... ;
例如
一个**24**位的寄存器**R13**加到一**24**位的计数器**,** 十一个字加到**R13**的高位**.**
ADD.B @R13+,10(R13) ; 将**LSD**相加不带进位
ADDC.B @R13+,10(R13) ; 带进位的中间数相加
ADDC.B @R13+,10(R13) ; 带进位的高字节相加

AND[.W] 源操作**AND**目的操作数
AND.B 源操作**AND**目的操作数
语法
AND src,dst 或者 **AND.W src,dst**
AND.B src,dst
操作
src .AND. dst -> dst
描述
源操作数和目的操作数逻辑与。结果存放在目的操作数中
状态标志位
N: 如果结果**MSB**被设置则置位**,** 否则不设置
Z: 结果为**0**置位，否则不设置
C: 结果不为**0**置位，否则不设置

V: 复位
模式选择位
OSCOFF, CPUOFF, 和 **GIE** 不会被影响.
例如
位设置在**R5**中被作为一个屏蔽位供**TOM**作字节寻址使用。如果结果是**0**，则跳转到**TONI.**

```
MOV #0AA55h,R5 ; 载入屏蔽位到寄存器R5
AND R5,TOM ; 通过TOM和R5屏蔽字节地址
JZ TONI ;
..... ; 结果不为0
;
;
; 或者
;
;
AND #0AA55h,TOM
JZ TONI
```

例如
屏蔽位**#0A5h**和**TOM**的低字节逻辑与。如果结果是**0**，程序将跳转到**TONI.**

```
AND.B #0A5h,TOM ; 用0A5h评比TOM的低字节
JZ TONI ;
..... ; 结果不为0
```

BIC[.W] 清除目的操作数中的相应位
BIC.B 清除目的操作数中的相应位
语法格式
BIC src,dst 或者 **BIC.W src,dst**
BIC.B src,dst
操作
NOT.src .AND. dst -> dst
描述
反转的源操作数和目的操作数在逻辑上是相与的.结果保存在目的操作数中，源操作数不受影响.
状态标志位
状态标志位不受影响.
模式选择位
OSCOFF, CPUOFF, 和**GIE**不受影响.
例如
RAM的**MSB**的第六位被清除.

```
BIC #0FC00h,LEO ; 清除MEM的MSB的第六位(LEO)
```

例如
RAM的**MSB**的第五位被清除。

```
BIC.B #0F8h,LEO ; Clear 5 MSBs in Ram location 清除位于RAM中LEO的高五位
```

BIS[.W]
目的操作数中相应的位置位
BIS.B
目的操作数中相应的位置位
语法格式
BIS src,dst或者**BIS.W src,dst**
BIS.B src,dst
操作
src .OR. dst -> dst
描述
源操作数和目的操作数逻辑或。结果保存在目的操作数中.源操作数不受影响.
状态标志位
状态标志位不受影响.
模式选择位
OSCOFF, CPUOFF, 和**GIE**不受影响.
例如
用字**TOM**给**RAM**中的低的六位置位.
BIS #003Fh,TOM; 设置位于**RAM**中**TOM**的低**6**位**set the six LSBs in RAM location TOM**
例如
设置**RAM**中**TOM**的高三位.
BIS.B #0E0h,TOM ; 设置位于**RAM**中**TOM**的高三位

BIT[.W] 测试目的操作数中的某一位
BIT.B 测试目的操作数中的某一位
语法格式
BIT src,dst 或者 **BIT.W src,dst**
操作
src .AND. dst
描述
源操作数和目的操作数逻辑与.结果仅仅影响状态位. 源操作数和目的操作数均不受影响.
状态标志位
N: 如果结果的高字被设置则置位,否则复位
Z: 结果为**0**则置位, 否则复位
C: 结果不为**0**则置位，否则复位
V: 复位
模式选择位
OSCOFF, CPUOFF,和**GIE**不受影响.
例如
如果R8中的第九位为**1**，则跳转到**TOM.**
BIT #0200h,R8 ; R8是否被置位**?**
JNZ TOM ; 是则跳转到**TOM**
... ; 不是，则继续程序
例如

如果R8的第三位被设置，程序跳转到TOM.

```
BIT.B #8,R8
JC TOM
```

例如

一个连续的通讯接受位被测试。因为在用BIT指令测试一个单一的位时进位标志位等同于状态的测试位，进位标志位用于连续的指令，读信息转移到RECBUF寄存器中。一连串的通讯位低字节位移动:

```
; XXXX XXXX XXXX XXXX
BIT.B #RCV,RCCTL ; 位信息进入进位标志位
RRC RECBUF ; Carry -> MSB of RECBUF
; CXXX XXXX
..... ;重复前两条指令
..... ; 8 times
; CCCC CCCC
; ^ ^
; MSB LSB
;连续通讯高字节位先移动:
BIT.B #RCV,RCCTL ; 位信息载入进位标志位
RLC.B RECBUF ; 将LSB载入RECBUF
; XXXX XXXC
..... ; 重复前两条指令
..... ; 8 times
; CCCC CCCC
; | LSB
; MSB
```

*** BR, BRANCH** 跳转到目的操作数地址

语法

```
BR dst
```

操作

dst -> PC

仿真设置

```
MOV dst,PC
```

描述

一个无条件的跳转指令能够跳转到64K地址空间内的任意地址.所以的源地址寻址模式将会被使用.跳转指令是个双字节指令.

状态标志位

状态标志位不受影响

例如

所有寻址模式的举例.

```
BR #EXEC ;跳转到EXEC 或者直接跳转(e.g. #0A4h)
; 核心指令MOV @PC+,PC
BR EXEC ;跳转EXEC中的地址中
; 核心指令MOV X(PC),PC
```

; 间接寻址模式
BR &EXEC ; 跳转到绝对地址EXEC中包含的地址
;核心指令MOV X(0),PC
;间接寻址
BR R5 ;跳转到R5中包含的地址中去
; 核心指令MOV R5,PC
; R5间接寻址
BR @R5 ; 跳转到字中所包含的地址中去
; 指向R5.
; 核心指令MOV @R5,PC
; R5的间接，间接
BR @R5+ ; 跳转到字所包含的地址中去
; 指向后地址增加.
; 下一个时间里—S/W覆盖掉用户的R5指针所指向的地址—它能够改变程序的执行由于访问了R5
; 所指向表格中的下一个地址
; 核心指令MOV @R5,PC
; 间接寻址方式，用R5的自动增量寻址方式
BR X(R5) ; 使程序分支跳转到所含的地址中去
; 用R5 + X (例如：表格中的地址起始于X).
; X可能是一个地址或者是程序的标签
; 核心指令 MOV X(R5),PC
; 间接寻址方式，R5 + X方式的间接寻址方式

CALL Subroutine
语法 CALL dst
操作： dst -> tmp dst是一个估计和存储的值
SP – 2 -> SP
PC -> @SP PC 更新为 TOS
tmp -> PC dst 保存在 PC
描述：一个子程序命令是作为一个在64K寻址空间里的任意的一个地址.
可使用所有的寻址模式.返回地址（下一条指令的地址）是存储在堆栈的地址。调用指令是个双字节指令.
状态标志位
状态标志位不受影响
例子：以下给出了所有寻址模式的例子.
CALL #EXEC ; 访问标签EXEC 或者立即数地址(e.g. #0A4h)
; SP–2 →SP, PC+2 →@SP, @PC+ →PC
CALL EXEC ; 访问EXEC中所包含的地址
; SP–2→SP, PC+2 →@SP, X(PC) →PC
; 间接寻址
CALL &EXEC ; 访问绝对地址EXEC所包含的地址
; SP–2→SP, PC+2→@SP, X(0) →PC
; 间接寻址

CALL R5 ; 访问R5中所含有的地址
; SP–2→SP, PC+2→@SP, R5→PC
; R5的间接寻址
CALL @R5 ; 调用R5所指向的地址中所包含的字
; SP–2→SP, PC+2 →@SP, @R5 →PC
; R5的间接寻址
CALL @R5+ ; 调用R5所指向的地址中所包含的字R5所指向的指针增加。下一个时间里，—S/W覆盖掉用户的R5指针所指向的地址—它能够改变程序的执行--由于访问了R5所指向表格中的下一个地址
; SP–2 →SP, PC+2 →@SP, @R5 →PC
; 间接寻址方式，用R5的自动增量寻址方式
CALL X(R5) ; 访问R5 + X (例如：表格中的地址起始于X)所指向的指针中所包含的地址。X可以是
;个标签或地址
; SP–2 →SP, PC+2 →@SP, X(R5) →PC
; R5 + X的间接寻址

---

* CLR[.W] 清除目的操作数中的内容
* CLR.B 清除目的操作数中的内容
语法：
CLR dst or CLR.W dst
CLR.B dst
操作：0 –> dst
仿真设置：
MOV #0,dst
MOV.B #0,dst
描述：目的操作数中的内容被清除.
状态标志位
状态标志位不受影响
例如：RAM 中的字TONI 被清除.
CLR TONI ; 0 –> TONI
例如： 寄存器R5被清除.
CLR R5
例如：RAM 中的字节TONI 被清除.
CLR.B TONI ; 0 –> TONI

---

* CLRC 清除进位标志位
语法： CLRC
操作： 0 –> C
仿真设置：BIC #1,SR
描述：进位标志位(C)被清除. 清除进位标志位的指令是个双字节指令
状态标志位：

N: 不受影响
Z: 不受影响
C: 被清除
V: 不受影响
模式选择位：OSCOFF，CPUOFF，和 GIE 均不受影响.
例如：R13所指向的16位小数计数器被加到R12所指向的一个32位的计数器上.
CLRC ; C=0: 定义开始
DADD @R13,0(R12) ; 将16位计数器加到32位计数器的低字节
DADC 2(R12) ; 将进位位加到32位计数器的高字节

---

* CLRN 清除负数标志位
语法格式： CLRN
操作： 0 →N
或者(.NOT.src .AND. dst -> dst)
仿真设置： BIC #4,SR
描述：常量04h被反转(0FFFBh)和目的操作数逻辑与.结果放在目的操作数中. 清除负数标志位的指令是双字节指令.
Status Bits
N: 复位为0
Z: 不受影响
C: 不受影响
V: 不受影响
模式选择位： OSCOFF, CPUOFF, and GIE不受影响.
例如： 在状态寄存器中负数标志位被清除. 这将避免子程序中的负数被访问.

```
CLRN
CALL SUBR
......
......
SUBR JN SUBRET ; 如果输入是负数: 不做任何处理，程序返回
......
......
......
SUBRET RET
```

---

* CLRZ 清除零标志位
语法： CLRZ
操作： 0 →Z 或(.NOT.src .AND. dst -> dst)
仿真设置： BIC #2,SR
描述： 常数02h被反转(0FFFDh)和目的操作数相与. 结果放在目的操作数中. 清除零标志位的指令是双字节指令.
状态标志位：
N:不受影响

Z: 被清除
C: 不受影响
V: 不受影响
模式选择位：OSCOFF, CPUOFF, 和 **GIE** 均不受影响.
例如：零标志位在状态寄存器中被清除.
CLRZ

CMP[.W] 比较源操作数和目的操作数
CMP.B 比较源操作数和目的操作数
语法：
CMP src,dst or CMP.W src,dst
CMP.B src,dst
操作
dst + .NOT.src + 1或**(dst – src)**
描述：源操作数和目的操作数相减. 源操作数和目的操作数均不受影响，结果不保存，但是只有状态位受到影响.
状态标志位：
N: 如果结果为负数，则置位。否则置**0(src >= dst)**
Z: 如果结果为**0**，则置位。否则置**0 (src = dst)**
C: 如果结果产生进位，则置位。否则置**0**
V: 如果结果产生溢出，则置位。否则置**0**
模式选择位：OSCOFF, CPUOFF, 和 **GIE** 均不受影响.
例子：R5和**R6**相比较. 如果相等则程序跳转到**EQUAL**处执行
CMP R5,R6 ; R5 = R6?
JEQ EQUAL ; YES, JUMP
例子：两个**RAM**块比较. 如果他们不相等, 程序跳转到分支**ERROR**处.
MOV #NUM,R5 ; 字的数目比较
MOV #BLOCK1,R6 ; BLOCK1 开始地址在**R6**中
MOV #BLOCK2,R7 ; BLOCK2 开始地址在**R7**中
L$1 CMP @R6+,0(R7) ; 所有的字相等吗？**R6**增加
JNZ ERROR ; 不相等, 跳转到**ERROR**
INCD R7 ; 增加**R7**指针
DEC R5 ; 所有的都比较完了吗？
JNZ L$1 ; 没有, 继续比较
例子：RAM中的字节地址有**EDE** 和 **TONI** 相比较. 如果他们相等,程序跳转到**EQUAL**处继续运行.
CMP.B EDE,TONI ; MEM(EDE) = MEM(TONI)?
JEQ EQUAL ; YES, JUMP

*** DADC[.W]** 以十进制方式进位标志位**(C)**加到目的操作数上.
*** DADC.B** 十进制方式进位标志位**(C)**以加到目的操作数上.
语法：
DADC dst 或 **DADC.W src,dst**

DADC.B dst
操作：
dst + C -> dst (decimally)
仿真设置
DADD #0,dst
DADD.B #0,dst
描述：
进位标志位(C)以十进制方式加到目的操作数上.
状态标志位
N: 如果MSB 为1则设置
Z: 如果dst是0则设置,否则复位
C: 如果从9999 变为 0000,则置位；否则复位
如果从9999 变为 0000,则置位；否则复位
V: 未被定义
模式选择位：OSCOFF, CPUOFF, 和 GIE 均不受影响.
例子：R5中的四位十进制数加到R8所指向R8所指向的8位十进制数.
CLRC ; 复位 carry位
; 下一条指令的起始条件是被定义的
DADD R5,0(R8) ; Add LSDs + C
DADC 2(R8) ; Add carry to MSD
例子：R5中的二位十进制数加到R8所指向R8所指向的8位十进制数.
CLRC ; 复位 carry位
;下一条指令的起始条件是被定义的
DADD.B R5,0(R8) ; Add LSDs + C
DADC 1(R8) ; 将进位加到MSDs

DADD[.W] 源操作数和目的操作数的十进制相加
DADD.B 源操作数和目的操作数的十进制相加
语法：
DADD src,dst or DADD.W src,dst
DADD.B src,dst
操作：
src + dst + C -> dst (decimally)
描述：
源操作数和目的操作数被作为带符号位的四位二进制BCD码来处理的. 源操作数和进位标志位以十进制方式加到了目的操作数中去。源操作数不受影响. 目的操作数的先前值丢失。 结果将不以非BCD码格式定义.
Status Bits
N: 如果MSB为1则置位，否则复位
Z: 结果为0置位，否则复位
C: 结果大于9999或者结果大于99置位，否则复位
V: 未定义
模式选择位：OSCOFF, CPUOFF, 和 GIE 均不受影响.
例子：一个8位的BCD码包含在R5和R6中以十进制的方式加到包含在R3和R4中的8位BCD数中d R4

(R6 和 R4存储的是高位).
CLRC ; 清除进位标志
DADD R5,R3 ; 加低位
DADD R6,R4 ; 带进位加高位
JC OVERFLOW ; 如果进位发生则跳转到错误处理子程序
例子:
两个十进制数的计数器在RAM字节CNT依次增1.
CLRC ; 清除进位
DADD.B #1,CNT ; 十进制计数器增加
或
SETC
DADD.B #0,CNT ; ≡DADC.B CNT

---

* DEC[.W] 目的操作数递减
* DEC.B 目的操作数递减
语法格式:
DEC dst or DEC.W dst
DEC.B dst
操作: dst – 1 –> dst
仿真设置: SUB #1,dst
仿真设置: SUB.B #1,dst
描述: 目的操作数递减1. 原先的计数值丢失.
状态寄存器:
N: 如果为负数则置位, 否则复位
Z: 如果dst包含1则置位, 否则复位
C: 如果dst包含0则复位, 否则置位
V: 如果有溢出发生则置位, 否则复位。
如果初始值是08000h则置位, 否则复位.
如果初始值是080h则置位, 否则复位.
模式选择位: OSCOFF, CPUOFF, 和 GIE 不受影响.
例如:
R10 递减1
DEC R10 ; 递减R10
; 从存储器中的起始地址EDE到TONI范围里移动255字节. 表格不能够交错: TONI的起始地址不能在EDE到EDE+0FEh的范围中
MOV #EDE,R6
MOV #255,R10
L$1 MOV.B @R6+,TONI-EDE-1(R6)
DEC R10
JNZ L$1
; 不要用交错的子程序去传送表格, 如图3-12.

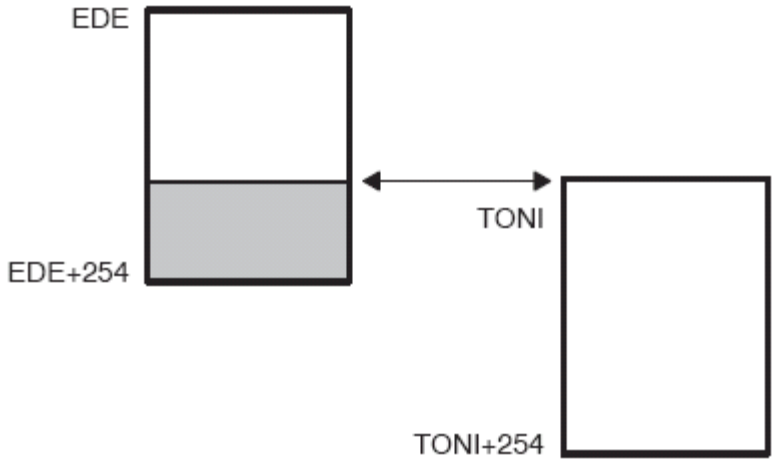


图3–12

EDE
EDE+254
TONI
TONI+254

*** DECD[.W]** 目的操作数的双精度递减
*** DECD.B** 目的操作数的双精度递减
语法：
DECD dst 或 **DECD.W dst**
DECD.B dst
操作：
dst – 2 –> dst
仿真设置：SUB #2,dst
仿真设置：SUB.B #2,dst
描述：
目的操作数每次减**2** ，原始的数据丢失**.**
状态寄存器：
N: 如果结果是负数则置位**,**否则复位
Z: 如果**dst**包含**2**则置位**,**否则复位
C: 如果**dst**包含**0** 或者 **1**则复位**,** 否则置位
V: 如果发生溢出则置位，否复位。
目的操作数的初始值**08001** 或 **08000h**则置位**,** 否则复位。
目的操作数的初始值为**081** 或者 **080h**时则置位**,** 否则复位。
模式选择位：OSCOFF, CPUOFF, 和 **GIE** 不受影响。
例：
R10 递减**2.**
DECD R10 ; R10 递减**2**
; 从存储器的起始地址 **EDE** 到**TONI**移动一个**255**字节的块，表格不能交错，目的地址的起始地址不能在**EDE** 到 **EDE+0FEh**范围里**;**
MOV #EDE,R6

```
MOV #510,R10
L$1 MOV @R6+,TONI-EDE-2(R6)
DECD R10
JNZ L$1
```

例：

存储器中位于LEO 处递减2。

```
DECD.B LEO ; 递减存储器MEM(LEO)
```

状态字节递减2。

```
DECD.B STATUS
```

*** DINT** 屏蔽一般的中断

语法： DINT

操作： 0→GIE

或者(0FFF7h .AND. SR →SR / .NOT.src .AND. dst -> dst)

仿真设置： BIC #8,SR

描述： 所有中断功能被屏蔽。

常量08h反转后与状态寄存器相与，结果保存在状态寄存器中.

状态寄存器： 状态位不受影响.

模式选择位： GIE 被复位. OSCOFF 和 CPUOFF 不受影响.

例：

普通的中断使能位(GIE) 在状态寄存器中被清除允许无中断移入到32位的计数器中。这个确保了在任何中断运行的时候计数器没有被修改。

```
DINT ; 所以的中断事件用GIE位来屏蔽
NOP
MOV COUNTHI,R5 ; 拷贝计数器
MOV COUNTLO,R6
EINT ; 所有的中断用GIE位使能
```

注：

屏蔽所有中断

如果任何的代码需要在中断期间保存，DINT在中断被屏蔽的最后一条指令前不能执行，或者跟一条 NOP指令。

EINT 中断使能

语法： EINT

操作： 1 →GIE

或者(0008h .OR. SR -> SR / .src .OR. dst -> dst)

仿真设置： BIS #8,SR

描述： 所有的中断被打开.

常量#08h 和SR 逻辑或. 结果保存在SR.

状态寄存器： 状态寄存器不受影响.

模式选择位： GIE 被设置. OSCOFF 和 CPUOFF 不受影响.

例：

普通的中断使能位(GIE)在状态寄存器中被设置。
; P1.2 到 P1.7的中断子程序
; P1IN 是所有端口被读的寄存器地址。P1IFG 是所有中断事件被屏蔽的寄存器。
PUSH.B &P1IN
BIC.B @SP,&P1IFG ; 仅仅复位受影响的标志
EINT ; 调整端口1的中断标志存储到堆栈中其他的中断是被允许的
BIT #Mask,@SP
JEQ MaskOK ; 标志被调整为屏蔽位去跳转
.....
MaskOK BIC #Mask,@SP
.....
INCD SP ; 事务管理: 反转为PUSH指令
;在所有中断服务子程序的开始，纠正堆栈的指针。
RETI
注:
中断使能
在中断使能之后的指令总是能够被执行的，即使在中断被使能的时候一个中断服务请求是未决的

---

* INC[.W] 目的操作数增加
* INC.B 目的操作数增加
语法格式:
INC dst or INC.W dst
INC.B dst
操作:
dst + 1 -> dst
仿真设置: ADD #1,dst
描述: 目的操作数递增1。原始内容丢失
状态位:
N: 如果结果为负数则置位，正数则复位
Z: 如果dst包含0FFFFh置位，否则复位
如果dst包含0FFh置位，否则复位
C: 如果dst包含0FFFFh置位，否则复位
如果dst包含0FFh置位，否则复位
V: 如果dst包含07FFFh置位，否则复位
如果dst包含07Fh置位，否则复位

模式位: OSCOFF, CPUOFF, 和GIE 是不受影响的.
例: 一个过程中的状态字节位STATUS是增加的. 当等于11,则跳转到处OVFL执行.
INC.B STATUS
CMP.B #11,STATUS
JEQ OVFL

*** INCD[.W]** 目的操作数递增**2**
*** INCD.B** 目的操作数递增**2**
语法格式：
INCD dst or INCD.W dst
INCD.B dst
操作：dst + 2 -> dst
仿真设置 ADD #2,dst
仿真设置 ADD.B #2,dst
例： 目的操作数递增**2**，原始的内容丢失**.**
状态位：
N: 如果结果为负数则置位，否则复位
Z: 如果**dst**包含**0FFFEh**置位，否则复位
如果**dst**包含**0FEh**置位，否则复位
C: 如果**dst**包含**0FFFEh**置位，否则复位
如果**dst**包含**0FEh**置位，否则复位
V: 如果**dst**包含**07FFEh**置位，否则复位
如果**dst**包含**07Eh**置位，否则复位
模式选择位：OSCOFF，CPUOFF，和 **GIE** 不受影响。
例：在堆栈的最顶部的一条未用任何一个寄存器即被移除**.**

```
.......
PUSH R5 ; R5 是计算的结果，被保存在系统的堆栈中
INCD SP ; 通过双增量从堆栈中移除TOS，不要用INCD.B指令，因为SP是一个双字节寄存器
RET
```

例： 在堆栈顶部的字节增**2**。

```
INCD.B 0(SP) ; 在TOS 处的字节增2。
```

*** INV[.W]** 反转目的操作数
*** INV.B** 反转目的操作数
语法格式：
INV dst
INV.B dst
操作： NOT.dst -> dst
仿真设置：XOR #0FFFFh,dst
仿真设置：XOR.B #0FFh,dst
描述： 目的操作数被反转。原始的内容丢失。
状态位：
N: 如果结果为负数则置位，否则复位
Z: 如果**dst**包含**0FFFFh**置位，否则复位
如果**dst**包含**0FFh**置位，否则复位
C: 如果结果不为**0**则置位，否则复位
V: 如果目的操作数的初始值为负数则置位，否则复位
模式选择位 OSCOFF，CPUOFF，和 **GIE** 不受影响**.**
例：

R5中的内容被反转(twos complement).
MOV #00AEh,R5 ; R5 = 00AEh
INV R5 ; Invert R5, R5 = 0FF51h
INC R5 ; R5 is now negated, R5 = 0FF52h
例:
存储器字节LEO中的内容被反转.
MOV.B #0AEh,LEO ; MEM(LEO) = 0AEh
INV.B LEO ; Invert LEO, MEM(LEO) = 051h
INC.B LEO ; MEM(LEO) 被反转,MEM(LEO) = 052h

JC 如果进位位被设置则跳转
JHS 如果大于或者高于则跳转
语法格式:
JC label
JHS label
操作:
If C = 1: PC + 2 + offset -> PC
如果C = 0: 执行下一条指令
描述:
状态寄存器的进位标志位被测试. 如果被置位, 则包含在指令LSB的中的10符号偏移量将会被加到程序计数器中. 如果C 被复位,跳转指令的下一条指令被执行. JC (如果进位标志位大于或等于则跳转) 无符号数的比较(0 to 65536).
状态位 状态位不受影响
例: The P1IN.1 信号用于定义或者控制程序的流程。
BIT #01h,&P1IN ; State of signal -> Carry
JC PROGA ; 如果C=1, 则执行子程序A
..... ; 如果C=0, 则执行这里的程序
例: R5和15做比较. 如果R5的结果是大于或者等于15, 则跳转到LABEL处执行.
CMP #15,R5
JHS LABEL ; 如果R5 15, 则跳转到LABEL
..... ; 如果R5 15则在这里继续执行

JEQ, JZ 如果相等则跳转, 如果为0则跳转
语法结构
JEQ label, JZ label
操作: 如果Z = 1: PC + 2 + offset -> PC
如果Z = 0: 执行下一条指令
描述: 状态寄存器的零标志位被测试. 如果零标志位被设置, 则指令中LSB中的10位偏移量被加到程序计数器中. 如果Z 没有被设置, 则执行跳转指令的下一条指令。
状态位: 状态位不受影响。
例:
如果R7包含零则跳转到地址TONI。

TST R7 ; 测试 R7
JZ TONI ; 如果为0: 则跳转
例：
如果R6等于表格中的内容，则跳转到地址LEO 。
CMP R6,Table(R5) ; 比较R6和存储器中的内容(表格地址 + R5的内容)
JEQ LEO ; 如果两个数据相等，则跳转到LEO
..... ; 不相等则继续执行
Example 如果R5是0则跳转到LABEL。
TST R5
JZ LABEL
.....

JGE 大于或者等于则跳转
语法格式
JGE label
操作：
如果 (N .XOR. V) = 0 则跳转到label: PC + 2 offset -> PC
如果 (N .XOR. V) = 1 则继续执行下一条指令
描述：
状态寄存器中的N位和V位被测试。如果N和V被设置或复位，则指令中LSB中的10位偏移量加到程序计数器中。 如果仅仅是一个被设置，则跳转指令的下一条指令被执行。这将允许比较带符号的整数
状态标志位：不受影响。
例：
当R6中的内容高于或者等于R7所指向的存储器，程序将会在EDE处继续执行.
CMP @R7,R6 ; R6 (R7)吗?, 带符号数的比较
JGE EDE ; 是的, R6 (R7),跳转到EDE
..... ; 不, 继续执行
.....
.....

JL 如果小于则跳转
语法格式:
JL label
操作:
如果(N .XOR. V) = 1 则跳转到label: PC + 2 offset -> PC
如果(N .XOR. V) = 0 则继续执行下一条指令
描述:
状态寄存器中的N位和V位被测试。如果N和V被设置或复位，则指令中LSB中的10位偏移量加到程序计数器中。 如果两个都被设置，则跳转指令的下一条指令被执行。这将允许比较带符号的整数
状态标志位:不受影响.

例:
当R6中的内容少于R7所指向的存储器,则程序跳转到EDE处执行.
```
CMP @R7,R6 ; R6 < (R7)吗?, 比较带符号数字
JL EDE ; 是, R6 < (R7)
...... ; 不是, 则继续执行
......
......
```

JMP 无条件跳转指令
语法格式:
JMP label
操作:
PC + 2 ×offset -> PC
描述
指令中LSB中的10位偏移量加到程序计数器中.
状态标志位: 不受影响。
提示: 这是一个字指令取代在-511 to +512字范围之间和当前程序计数器相关的的分支指令

JN 负数则跳转
语法格式
JN label
操作:
如果 N = 1: PC + 2 ×offset -> PC
如果 N = 0: 执行下一条指令
描述:状态寄存器负数标志位N被测试. 如果被设置,则指令中的LSB的偏移量加到程序计数器中去.如果N被复位, 则执行跳转指令的下一条指令.
状态标志位: 不受影响。
例:
R5中的内容R5和COUNT相减的结果. 如果结果是负数, COUNT 将会被清除而且程序在另一路径执行.
```
SUB R5,COUNT ; COUNT - R5 -> COUNT
JN L$1 ; 如果为负数,COUNT=0 at PC=L$1
...... ; 如果COUNT 0则执行
......
......
......
L$1 CLR COUNT
......
......
......
```

JNC 进位标志位没有设置则跳转
JLO 如果较低则跳转
语法格式
JNC label
JLO label
操作:
如果 **C = 0: PC + 2 offset -> PC**
如果 **C = 1:** 执行下一条指令
描述:
状态寄存器的进位标志位被测试. 如果被复位,则指令中的**LSB10**位偏移量加到程序计数器中.
如果**C**被设置跳转指令的下一条指令被执行.**JNC**指令可用于无符号数的比较**(0 to 65536).**
状态标志位: 不受影响.
例:
结果在**R6**中加进**BUFFER**. 如果溢出发生,则一个错误处理程序在地址**ERROR** 处被使用.

```
ADD R6,BUFFER ; BUFFER + R6 -> BUFFER
JNC CONT ; 没有进位, 则跳转到CONT
ERROR ..... ; 错误处理开始
......
......
......
CONT ..... ; 依据正常的程序流程下继续执行
......
......
```

例:
跳转到**STL2** 如果状态位字节**STATUS** 是**1** 或 **0.**

```
CMP.B #2,STATUS
JLO STL2 ; STATUS < 2
..... ; STATUS >=2, 则继续执行
```

JNE 不相等则跳转
JNZ 不为**0**则跳转
语法格式:
JNE label
JNZ label
操作:
如果 **Z = 0: PC + 2 offset -> PC**
如果 **Z = 1:** 执行下一条指令
描述:
状态寄存器的**Z**被测试.如果被复位,则指令中**LSB**中的**10**偏移量将会被加到程序计数器中.如果**Z**被设置,则执行跳转指令的下一条指令.
状态标志位:状态标志位不受影响.
例: 如果R7和R8中的内容不同, 则跳转到地址TONI

```
CMP R7,R8 ; 比较R7和R8
JNE TONI ; 如果不同则跳转
```

..... ; 如果相同则继续执行

MOV[.W]　　将源操作数移动到目的操作数中
MOV.B　　将源操作数移动到目的操作数中
语法格式：
MOV src,dst 或 MOV.W src,dst
MOV.B src,dst
操作：src -> dst
描述：
源操作数被移动到目的操作数中去。源操作数不受影响.目的操作数先前的内容丢失.
状态标志位：状态标志位不受影响.
模式选择位 OSCOFF, CPUOFF, 和 GIE 不受影响.
例： 常数表格EDE (word data) 被拷贝到表格TOM中. 表格的长度起始于020h .
MOV #EDE,R10 ; 准备指针
MOV #020h,R9 ; 准备计数器
Loop MOV @R10+,TOM-EDE-2(R10) ; 用R10中的指针指向两个表格
DEC R9 ; 计数器递减
JNZ Loop ; 计数器不为0, 继续拷贝
..... ; 为0，则拷贝结束
.....
.....
例：表格EDE (byte data)中的内容拷贝到表格TOM中去. 设表格的长度为020h
MOV #EDE,R10 ; 准备指针
MOV #020h,R9 ; 准备计数器
Loop MOV.B @R10+,TOM-EDE-1(R10) ; 用R10中的指针指向两个表格
DEC R9 ; 计数器递减
JNZ Loop ; 计数器不为0, 继续拷贝
..... ; 为0，则拷贝结束
.....
.....

---

* NOP　空指令
语法格式：NOP
操作：不做任何操作
仿真设置：MOV #0, R3
描述：不做任何的操作. 此条指令用于指令软件检查或定义延迟.
状态标志位：状态标志位不受影响.
NOP指令主要用于两个目的:
_ 为了填充一个或者两个存储器的字
_ 为了矫正软件定时
注:
仿真不操作指令

其他的指令在不同的指令周期和代码字节数的时能够仿真NOP功能．如下的一些例子：
例：
MOV #0,R3 ; 1 个周期，1 个字长度
MOV 0(R4),0(R4) ; 6 个周期，3 个字长度
MOV @R4,0(R4) ; 5 个周期，2 个字长度
BIC #0,EDE(R4) ; 4 个周期，2 个字长度
JMP $+2 ; 2 周期，1 个字长度
BIC #0,R5 ; 1 周期，1 个字长度
但是，当用这些例子去阻止不可预见的结果的时要小心使用．例如：，如果 MOV 0(R4)，0(R4)被使用，而且R4里面的值是120h时，这时将和WDT定时器产生一个安全冲突，因为安全关键字没有被使用

POP[.W]　　双字节目的操作数出堆栈
* POP.B　　单字节目的操作数出堆栈
语法格式
POP dst
POP.B dst
操作
@SP –> temp
SP + 2 –> SP
temp –> dst
仿真设置　MOV @SP+,dst or MOV.W @SP+,dst
仿真设置　MOV.B @SP+,dst
描述：
将堆栈指针的单元内容移送到目的操作数中，然后堆栈指针增2
状态标志位：状态标志位不受影响．
例：　R7和SR中的内容被堆栈中弹出的内容所恢复．
POP R7 ; 恢复 R7
POP SR ; 恢复SR寄存器
例：　RAM中的字节LEO从堆栈中被恢复．
POP.B LEO ; 堆栈中的低字节被移送到LEO中．
例：　R7中的内容从字节中被恢复．
POP.B R7 ; R7中的低字节内容被移到R7中，R7的高字节内容是00h
例：　R7指向的存储器的内容和状态寄存器从堆栈中被恢复．
POP.B 0(R7) ; 堆栈中的低字节被移送到R7指向的字节
: 例：　R7 = 203h
; Mem(R7) = 系统堆栈中的低字节
: Example: R7 = 20Ah
; Mem(R7) = 系统堆栈中的低字节
POP SR ; 堆栈中最后的字移送到SR寄存器中
注：系统的堆栈指针
系统的堆栈指针每个两个单位递增，独立的字节后缀．

PUSH[.W] 字的入栈
PUSH.B 字节的入栈
语法： **PUSH src** 或 **PUSH.W src**
PUSH.B src
操作：
SP – 2 →SP
src →@SP
描述：堆栈的指针总是以两个为单位递增，这时源操作数被堆栈指针移动到**RAM**的字地址**.**
状态标志位： 不受影响**.**
模式选择位 OSCOFF, CPUOFF, 和 **GIE** 不受影响**.**
例： 状态寄存器中的内容和**R8**入栈。
PUSH SR ; 保存状态寄存器
PUSH R8 ; 保存 **R8**
例： 外部设备**TCDAT**中的内容入栈**.**
PUSH.B &TCDAT ; 从外部模块中保存**8**个位**,**
; 寻址**TCDAT,** 入栈
注：系统堆栈指针
系统的堆栈指针以两个单位递增，独立的字节后缀**.**

*** RET** 子程序返回
语法格式： **RET**
操作： **@SP→PC**
SP + 2 →SP
仿真设置： **MOV @SP+,PC**
描述： 返回的地址被一个**CALL**指令压入堆栈中，之后又被移送到程序计数器中**.** 程序在子程序调用之后的地址处继续执行**.**
状态标志位：不受影响

RETI 中断返回
语法格式： **RETI**
操作： **TOS →SR**
SP + 2 →SP
TOS →PC
SP + 2 →SP
描述：
同过堆栈中的内容覆盖**SR**中的内容，状态寄存器恢复到中断服务子程序执行之前的值。**SP**增加**2**。程序计数器的内容恢复到中断服务之前的值。在中断程序的流程中，这些是连续的步骤。恢复程序计数器的内容是通过用堆栈中存储器的内容取代当前**PC**的内容。**SP**增加**.**
状态标志位
N: 从系统堆栈中被恢复
Z: 从系统堆栈中被恢复

C: 从系统堆栈中被恢复
V: 从系统堆栈中被恢复
模式选择位： OSCOFF, CPUOFF, 和 **GIE** 从系统堆栈中被恢复.
例：
图**3–13** 阐述了主程序中断

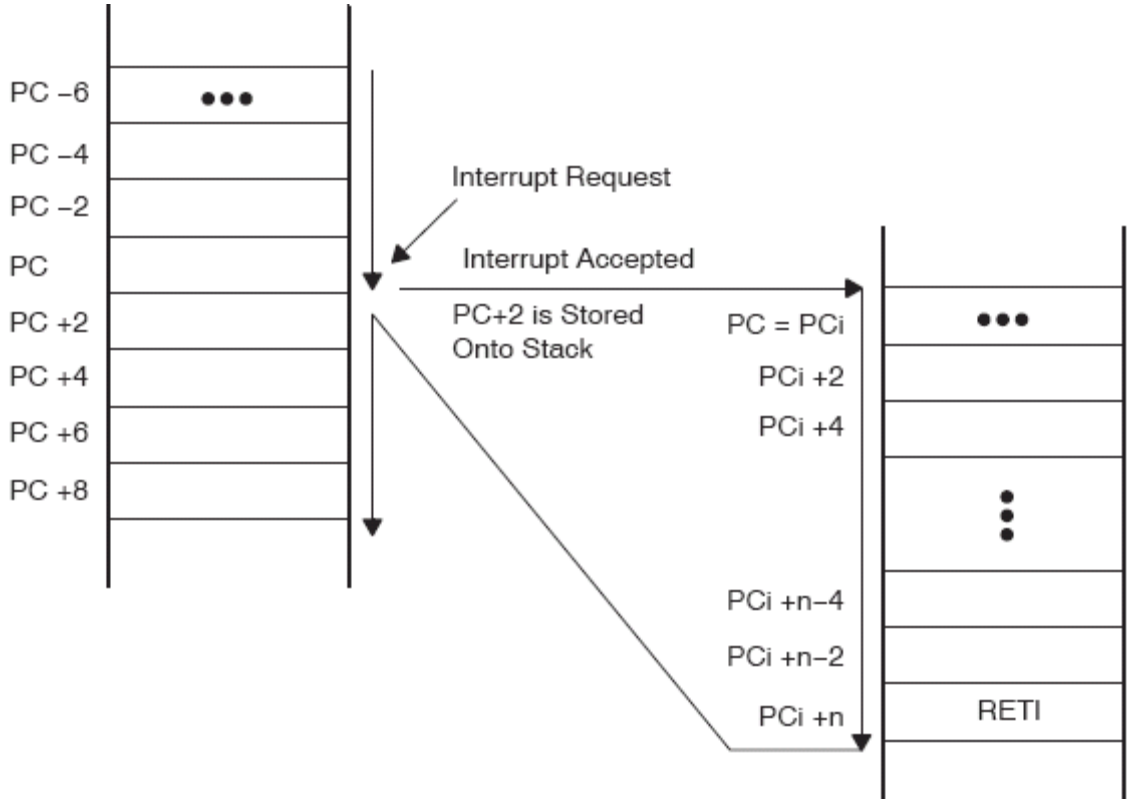
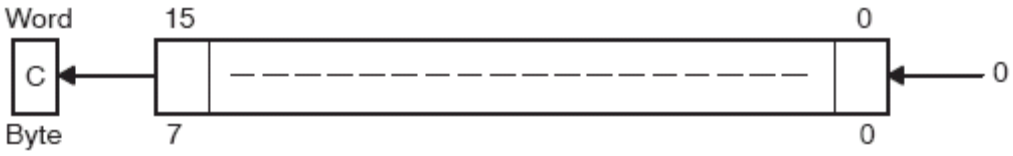


图**3–13**

*** RLA[.W]** 算术左移
*** RLA.B** 算术左移
语法格式： RLA dst 或 **RLA.W dst**
RLA.B dst
操作： C <– MSB <– MSB–1 .... LSB+1 <– LSB <– 0
仿真设置：
ADD dst,dst
ADD.B dst,dst
描述：
目的操作数被左移动一位，如图**3-14**所示**.**
高位被移动到进位标志位中，而低字节位被**0**填充**. RLA**指令的功能等同于一个带符号数乘**2**。
如果在操作之前 **dst >=04000h** 并且**dst < 0C000h**则将会有一个溢出产生**,**结果将改变符号**.**



15 0

状态标志位

N: 如果结果为负数则置位,否则复位

Z: 结果为**0**则置位,否则复位

C: 由高位载入

V: 如果有一个算术溢出产生则置位:初始值为 **04000h ≤dst < 0C000h;** 否则复位

如果有一个算术溢出产生则置位:初始值为**040h ≤dst < 0C0h;** 否则复位

模式选择位 OSCOFF, CPUOFF, 和 **GIE** 不受影响.

例: R7 被乘**2**

RLA R7 ; R7左移

例: R7的低字节乘**4.**

RLA.B R7 ; 左移**R7**的低字节

RLA.B R7 ; 左移**R7**的低字节

注: RLA替代用法

在汇编过程中不能识别以下指令:

RLA @R5+, RLA.B @R5+, or RLA(.B) @R5

这些可由下列指令所替代:

ADD @R5+,–2(R5) ADD.B @R5+,–1(R5) or ADD(.B) @R5

*** RLC[.W]** 带进位的左移

*** RLC.B** 带进位的左移

语法格式: **RLC dst** 或 **RLC.W dst** **RLC.B dst**

操作: **C <– MSB <– MSB–1 .... LSB+1 <– LSB <– C**

仿真设置: **ADDC dst,dst**

描述: 目的操作数左移一位,如图**3-15**所示.

进位标志位移到**LSB**中而**MSB**移动到进位标志位

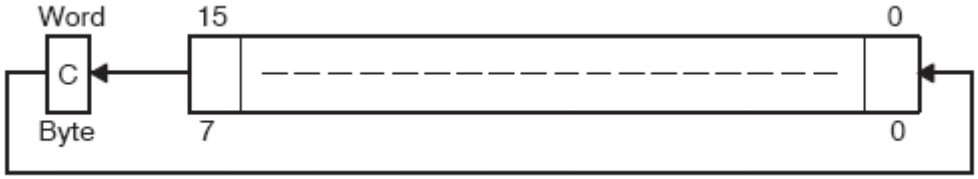


图**3-15**

状态标志位

N: 负数则置位,否则复位

Z: 结果为**0**则置位,否则复位

C: 由**MSB**载入

如果有一个算术溢出产生则置位:初始值为 **04000h ≤dst < 0C000h;** 否则复位

如果有一个算术溢出产生则置位:初始值为**040h ≤dst < 0C0h;** 否则复位

模式选择位 OSCOFF, CPUOFF, 和 **GIE** 不受影响.

例: R5左移一位.

RLC R5 ; (R5 x 2) + C -> R5
例: 输入P1IN.1 信息移到R5的LSB中.
BIT.B #2,&P1IN ; 信息 -> Carry
RLC R5 ; Carry=P0in.1 -> LSB of R5
例: MEM(LEO)中的内容左移一位.
RLC.B LEO ; Mem(LEO) x 2 + C -> Mem(LEO)
注意: RLC 和 RLC.B 置换用法
在汇编程序中不能识别以下的指令:
RLC @R5+, RLC.B @R5+, or RLC(.B) @R5
它必须有以下的指令所代替:
ADDC @R5+,-2(R5) ADDC.B @R5+,-1(R5) or ADDC(.B) @R5

RRA[.W] 算术右移
RRA.B 算术右移
语法格式: RRA dst RRA.W dst RRA.B dst
操作: MSB -> MSB, MSB -> MSB-1, ... LSB+1 -> LSB, LSB -> C
描述: 目的操作数右移一位,如图3-16所示

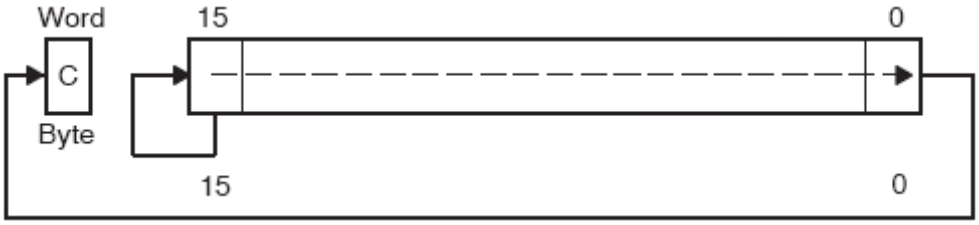


图3-16

MSB被移动到C, MSB 被移进MSB-1, 并且LSB+1 被移到LSB.LSB被移到C中
状态标志位:
N: 结果为负数则置位,否则复位
Z: 结果为0则置位,否则复位
C: LSB载入
V: 复位
模式选择位 OSCOFF, CPUOFF, 和 GIE 不受影响.
例: R5 被右移一位. MSB 仍然为旧的值. 此指令的操作等同于算术除以2.
RRA R5 ; R5/2 -> R5 R5的值乘以 0.75 (0.5 + 0.25).
PUSH R5 ; 用堆栈临时保存R5
RRA R5 ; R5 .0.5 -> R5
ADD @SP+,R5 ; R5 .0.5 + R5 = 1.5 .R5 -> R5
RRA R5 ; (1.5 .R5) .0.5 = 0.75 .R5 -> R5
.....
例: R5的低字节右移一位. MSB 仍然为旧的数值.此条指令操作等同于除以2.
RRA.B R5 ; R5/2 -> R5: 仅仅操作低字节; R5的高字节复位
PUSH.B R5 ; R5 .0.5 -> TOS
RRA.B @SP ; TOS .0.5 = 0.5 .R5 .0.5 = 0.25 .R5 -> TOS
ADD.B @SP+,R5 ; R5 .0.5 + R5 .0.25 = 0.75 .R5 -> R5
.....

RRC[.W] 带进位的右移
RRC.B 带进位的右移
语法格式: **RRC dst** **RRC.W dst** **RRC dst**
操作: **C -> MSB -> MSB-1 .... LSB+1 -> LSB -> C**
描述: 目的操作数右移一位如图**3-17**所示.进位标志位被移进**MSB,LSB**被移到进位标志位中.

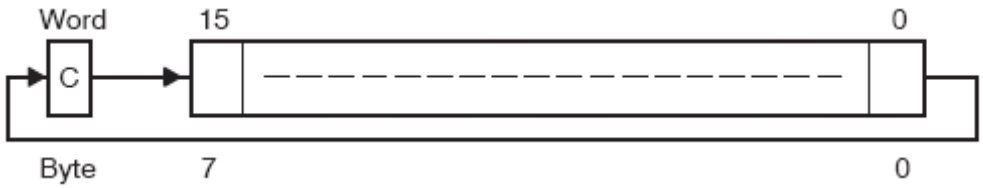


图**3-17**

状态标志位
N: 如果结果为负数则置位，否则复位
Z: 结果为**0**则置位，否则复位
C: 由**LSB**位载入
V: 复位
模式选择位
OSCOFF, CPUOFF, 和 **GIE** 不受影响.
例:
R5 右移一位. **MSB**补**1.**
SETC ; MSB补**1**
RRC R5 ; R5/2 + 8000h -> R5
例:
R5 右移一位. **MSB**置**1.**
SETC ; 准备把**C**（进位标志位）移到**MSB**中
RRC.B R5 ; R5/2 + 80h -> R5; R5的低字节位被使用

*** SBC[.W]** 带借位的源操作数和目的操作数的减法
*** SBC.B** 带借位的源操作数和目的操作数的减法
语法: **SBC dst** 或 **SBC.W dst** **SBC.B dst**
操作: **dst + 0FFFFh + C -> dst** **dst + 0FFh + C -> dst**
仿真设置 **SUBC #0,dst** **SUBC.B #0,dst**
描述: 进位标志位加到目的操作数中减一个数，目的操作数先前内容丢失.
状态标志位:
N:如果结果为负数则置位，否则复位
Z: 结果为**0**则置位，否则复位
C: 如果从结果的**MSB**位中产生进位则置位，否则复位.
有进位则置位，没有进位则复位.
V: 如果有算术溢出发生则置位，否则复位.
模式选择位 OSCOFF, CPUOFF, 和 **GIE** 不受影响.
例:

R13指向的一个16指针计数器和R12指向的一个32位计数器相减.
SUB @R13,0(R12) ; 低位相减
SBC 2(R12) ; 减数从高位借1
例:
R13指向的一个16指针计数器和R12指向的一个32位计数器相减.
SUB.B @R13,0(R12) ; 低位相减
SBC.B 1(R12) ; 从高位借位
注: 借位的执行.
借位是作为进位标志位的非操作来执行的。有借位，进位标志位为0，没有借位，进位标志位为1

---

* SETC 进位标志位置位
语法: SETC
操作: 1 -> C
仿真设置: BIS #1,SR
描述: 进位标志位被置位.
状态标志位:
N: 不受影响
Z: 不受影响
C: 置位
V: 不受影响
模式选择位 OSCOFF, CPUOFF, 和 GIE 不受影响.
例:
仿效十进制的减法:
从R6中减去R5的十进制减法
设R5 = 03987h , R6 = 04137h

```
DSUB    ADD #06666h,R5 ; 将R5中的内容变为十进制数R5 = 03987h + 06666h = 09FEDh
          INV R5 ;   R5中的内容取非
          SETC ;    设置进位标志位
          DADD R5,R6 ; R5的反码加到R6上实现减法
                     ; (010000h - R5 - 1)
                     ; R6 = R6 + R5 + 1
                     ; R6 = 0150h
```

---

* SETN 设置负数标志位
语法: SETN
操作: 1 -> N
仿真设置: BIS #4,SR
描述: 负数标志位置位.
状态标志位:
N: 置位

Z: 不受影响
C: 不受影响
V: 不受影响
模式选择位： OSCOFF, CPUOFF, 和 **GIE** 不受影响.

*** SETZ** 设置零标志位
语法格式： SETZ
操作： 1 -> Z
仿真设置： BIS #2,SR
描述： 零标志位被设置.
状态标志位：
N: 不受影响
Z: 置位
C: 不受影响
V: 不受影响
模式选择位 OSCOFF, CPUOFF, 和 **GIE** 不受影响.

SUB[.W] 从目的操作数中减去源操作数
SUB.B 从目的操作数中减去源操作数
语法格式： SUB src,dst 或 **SUB.W src,dst SUB.B src,dst**
操作： dst + .NOT.src + 1 -> dst 或 **[(dst – src -> dst)]**
描述：
从目的操作数中通过加上源操作数的反码加**1**来实现和源操作数的减法的。源操作数不受影响，目的操作数先前的内容丢失
状态标志位：
N: 结果为负数则置位，否则复位
Z: 为**0**则置位，否则复位
C: 如果结果从高位借位则置位，否则复位；没有借位发生则置位，有借位发生则置**0.**
V: 如果有算术溢出发生则置位，否则复位
模式选择位 OSCOFF, CPUOFF, 和 **GIE** 不受影响.
例：参照**SBC**指令.
例：参照**SBC.B**指令.
注：借位的执行.
借位是作为进位标志位的非操作来执行的。有借位，进位标志位为**0**，没有借位，进位标志位为**1**

SUBC[.W]SBB[.W] 带借位的源操作数和目的操作数的减法
SUBC.B,SBB.B 带借位的源操作数和目的操作数的减法
语法格式
SUBC src,dst 或 **SUBC.W src,dst**

SBB src,dst 或 **SBB.W src,dst**
SUBC.B src,dst 或 **SBB.B src,dst**
操作： dst + .NOT.src + C -> dst 或 **(dst – src – 1 + C -> dst)**
描述：
从目的操作数中通过加上源操作数的反码加**1**来实现和源操作数的减法的。源操作数不受影响，目的操作数先前的内容丢失**.**
状态标志位：
N: 结果为负数则置位，否则复位
Z: 为**0**则置位，否则复位
C: 如果结果从高位借位则置位，否则复位；没有借位发生则置位，有借位发生则置**0.**
V: 如果有算术溢出发生则置位，否则复位
模式选择位 OSCOFF, CPUOFF, 和 **GIE** 不受影响**.**
例：
两个**24**位的浮点数减法**.**
低字节在**R13** 和 **R10,** 高字节在**R12** 和 **R9.**
SUB.W R13,R10 ; 16位的低字节部分
SUBC.B R12,R9 ; 8位的高字节部分
例：
R13指向的**16**的计数器和**R10**和**R11**（高字节）指向的**16**位计数器相减
SUB.B @R13+,R10 ; 不带进位的低字节位相减
SUBC.B @R13,R11 ; 带进位的高字节位相减
... ; 结果由低字节位产生
注：借位的执行.
借位是作为进位标志位的非操作来执行的。有借位，进位标志位为**0**，没有借位，进位标志位为**1**

SWPB 交换字节
语法格式： SWPB dst
操作： 高**8**位**15**和低**8**位交换
描述： 目的操作的的高**8**位**15**和低**8**位发生交换
如图 **3–18**所示

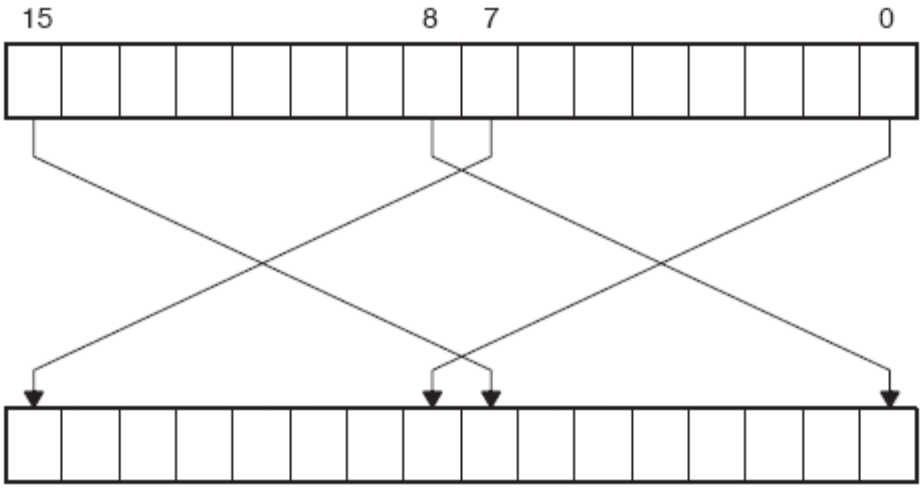


图 3–18

状态标志位： 不受影响.
模式选择位： OSCOFF, CPUOFF, 和 GIE不受影响.

例：

```
MOV #040BFh,R7 ; 0100000010111111 -> R7
SWPB R7 ; 1011111101000000 in R7
```

例：
R5中的内容乘256，结果保存在R5,R4中.

```
SWPB R5 ;
MOV R5,R4 ;拷贝交换后的值到R4
BIC #0FF00h,R5 ;矫正结果
BIC #00FFh,R4 ; 矫正结果
```

SXT 符号扩展
语法格式： SXT dst
操作： Bit 7 -> Bit 8 ..... Bit 15
描述：
低字节的符号扩展到高字节中去，如图3–19所示.
状态标志位
N: 如果结果为负数则置位，否则复位
Z: 结果为0则置位，否则复位
C: 结果不为0则置位，否则复位
V: 复位
模式选择位 OSCOFF, CPUOFF, 和 GIE 不受影响.

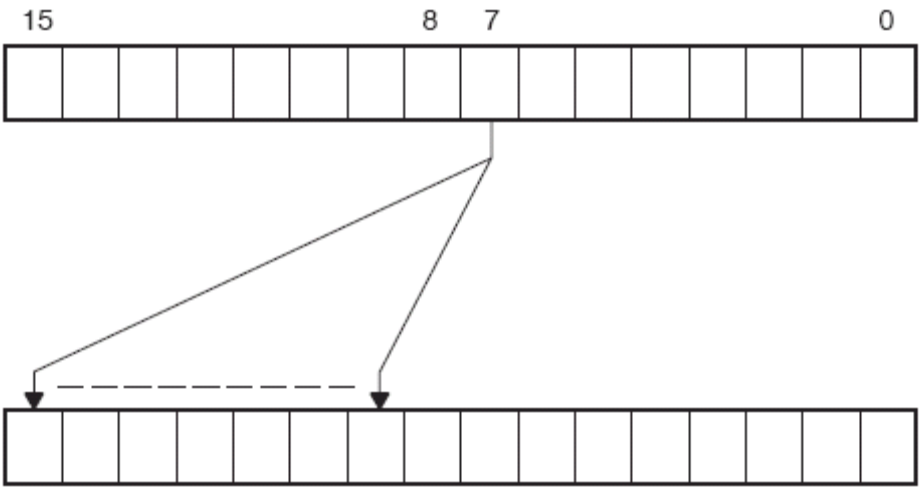


图3–19

例：

R7 载入P1IN中的值．符号扩展指令将用第7位中值使第8位扩展到第15位，然后将R7加到R6中去

```
MOV.B &P1IN,R7 ; P1IN = 080h: . . . . . 1000 0000
SXT R7 ; R7 = 0FF80h: 1111 1111 1000 0000
```

*** TST[.W]** 测试目的操作数

*** TST.B** 测试目的操作数

语法： TST dst 或 TST.W dst TST.B dst

操作： dst + 0FFFFh + 1 dst + 0FFh + 1

仿真设置： CMP #0,dst CMP.B #0,dst

描述：

目的操作数和0做比较．状态标志位依据结果而设置．目的操作数不受影响．

状态标志位：

N: 如果为负数则置位，否则复位

Z: 如果目的操作数为0则置位，否则复位

C: 置位

V: 复位

模式选择位 OSCOFF，CPUOFF，和 GIE 不受影响．

例：

R7被测试．如果结果为负数，则程序跳转到R7NEG处继续执行；如果结果是负数但是不为0，则程序跳转到R7POS处继续执行．

```
TST R7 ; 测试 R7
JN R7NEG ; R7 为负数
JZ R7ZERO ; R7 为0
R7POS ..... ; R7 为负数但是不为0
R7NEG ..... ; R7 为负数
R7ZERO ..... ; R7 为0
```

例：

测试R7的低字节。如果为负数，则程序跳转到R7NEG处继续执行；如果R7是负数但不为0，则R7跳转到R7POS处继续执行.

```
TST.B    R7 ; 测试R7的低字节
JN       R7NEG ; R7的低字节为负数
JZ       R7ZERO ; R7的低字节为0
R7POS ..... ;  R7的低字节为负数但不为0
R7NEG ..... ;   R7的低字节为负数
R7ZERO ..... ;  R7的低字节为0
```

XOR[.W] 源操作数和目的操作数的异或
XOR.B 源操作数和目的操作数的异或
语法格式： XOR src,dst 或 XOR.W src,dst XOR.B src,dst
操作： src .XOR. dst -> dst
描述：
源操作数和目的操作数异或，结果存放在目的操作数中，源操作数不受影响
状态标志位
N: 如果最高位被设置则置位，否则复位
Z: 结果为0则置位，否则复位
C: 结果不为0则置位，否则复位
V: 如果两个操作数均为负数，则置位否则复位
模式选择位 OSCOFF, CPUOFF, 和 GIE 不受影响.
例：
在R6中的设置的位连接RAM中的字TONI

```
XOR R6,TONI ; 字TONI中的连接位在R6中的相关位中被设置
```

例：
在R6中的设置的位连接RAM中的字节TONI.

```
XOR.B R6,TONI ; 字节TONI中的连接位在R6中的低字节中相关位中被设置
; low byte of R6
```

例：
将R7中的低字节设为0使得异或RAM中的字节EDE.

```
XOR.B EDE,R7 ; 不同的反转为“1s”
INV.B R7 ; 反转低字节，高字节为0h
```

3.4.4 指令周期和指令长度

CPU的时钟周期数The number of CPU clock cycles required for an instruction depends on the
instruction format and the addressing modes used - not the instruction itself.
时钟的周期数可参考MCLK.
中断和复位周期Interrupt and Reset Cycles
表3–14 列出了中断和复位时CPU的周期

Action	No. of Cycles	Length of Instruction
Return from interrupt (RETI)	5	1
Interrupt accepted	6	–
WDT reset	4	–
Reset ($\overline{RST}$/NMI)	4	–

表3–14

单操作数的指令周期和长度

表3–15列出了格式2所有寻址模式格式的CPU周期

Addressing Mode	No. of Cycles: RRA, RRC SWPB, SXT	No. of Cycles: PUSH	No. of Cycles: CALL	Length of Instruction	Example
Rn	1	3	4	1	SWPB R5
@Rn	3	4	4	1	RRC @R9
@Rn+	3	5	5	1	SWPB @R10+
#N	(See note)	4	5	2	CALL #0F000h
X(Rn)	4	5	5	2	CALL 2(R7)
EDE	4	5	5	2	PUSH EDE
&EDE	4	5	5	2	SXT &EDE

表3–15

注：指令格式2的立即数模式

勿在立即数模式下的目的操作数中使用指令RRA，RRC，SWPB，和SXT．这些指令在立即数模式中的将导致是不可预见程序操作．

格式3，跳转指令的指令周期和指令长度

无论跳转是否发生，所有的跳转指令都将占用一个字指令和两个CPU周期去执行．

格式1，双操作数的指令周期和指令长度

表3–16列出了在格式1的情况下所有寻址模式的CPU周期

寻址模式 Src	Dst	指令周期	指令长度	例	
Rn	Rm	1	1	MOV	R5,R8
	PC	2	1	BR	R9
	x(Rm)	4	2	ADD	R5,4(R6)
	EDE	4	2	XOR	R8,EDE
	&EDE	4	2	MOV	R5,&EDE
@Rn	Rm	2	1	AND	@R4,R5
	PC	2	1	BR	@R8
	x(Rm)	5	2	XOR	@R5,8(R6)
	EDE	5	2	MOV	@R5,EDE
	&EDE	5	2	XOR	@R5,&EDE
@Rn+	Rm	2	1	ADD	@R5+,R6
	PC	3	1	BR	@R9+
	x(Rm)	5	2	XOR	@R5,8(R6)
	EDE	5	2	MOV	@R9+,EDE
	&EDE	5	2	MOV	@R9+,&EDE
#N	Rm	2	2	MOV	#20,R9
	PC	3	2	BR	#2AEh
	x(Rm)	5	3	MOV	#0300h,0(SP)
	EDE	5	3	ADD	#33,EDE
	&EDE	5	3	ADD	#33,&EDE
x(Rn)	Rm	3	2	MOV	2(R5),R7
	PC	3	2	BR	2(R6)
	TONI	6	3	MOV	4(R7),TONI
	x(Rm)	6	3	ADD	4(R4),6(R9)
	&TONI	6	3	MOV	2(R4),&TONI
EDE	Rm	3	2	AND	EDE,R6
	PC	3	2	BR	EDE
	TONI	6	3	CMP	EDE,TONI
	x(Rm)	6	3	MOV	EDE,0(SP)
	&TONI	6	3	MOV	EDE,&TONI
&EDE	Rm	3	2	MOV	&EDE,R8
	PC	3	2	BRA	&EDE
	TONI	6	3	MOV	&EDE,TONI
	x(Rm)	6	3	MOV	&EDE,0(SP)
	&TONI	6	3	MOV	&EDE,&TONI

表3–16

3.4.5 指令设置的描述

核心指令图如图3–20中所示，所有的指令设置在表3–17作了一个总结.

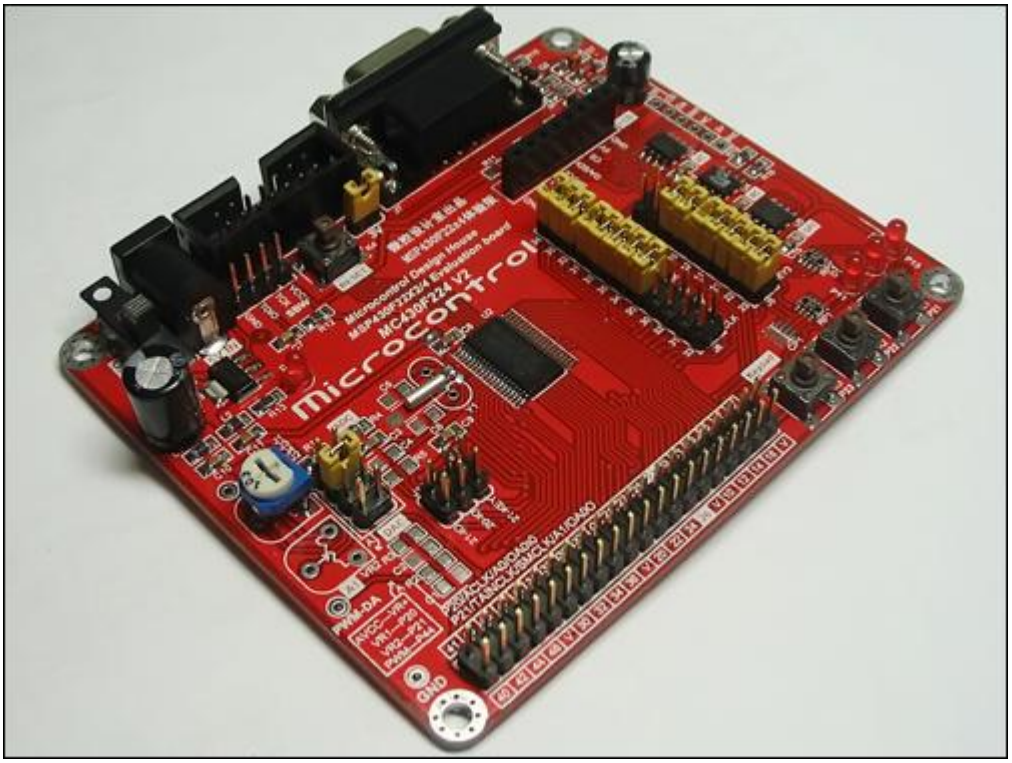
	000	040	080	0C0	100	140	180	1C0	200	240	280	2C0	300	340	380	3C0
0xxx																
4xxx																
8xxx																
Cxxx																
1xxx	RRC	RRC.B	SWPB		RRA	RRA.B	SXT		PUSH	PUSH.B	CALL		RETI			
14xx																
18xx																
1Cxx																
20xx	JNE/JNZ															
24xx	JEQ/JZ															
28xx	JNC															
2Cxx	JC															
30xx	JN															
34xx	JGE															
38xx	JL															
3Cxx	JMP															
4xxx	MOV, MOV.B															
5xxx	ADD, ADD.B															
6xxx	ADDC, ADDC.B															
7xxx	SUBC, SUBC.B															
8xxx	SUB, SUB.B															
9xxx	CMP, CMP.B															
Axxx	DADD, DADD.B															
Bxxx	BIT, BIT.B															
Cxxx	BIC, BIC.B															
Dxxx	BIS, BIS.B															
Exxx	XOR, XOR.B															
Fxxx	AND, AND.B															

图3–20

Mnemonic	Description	V N Z C
ADC(.B) ↑ dst	将C 加到目的操作数 dst + C →dst	* * * *
ADD(.B) src,dst	目的操作数和源操作数相加 src + dst → dst	* * * *
ADDC(.B) src dst	带进位的目的操作数和源操作数相加 src + dst + C →dst	* * * *
AND(.B) src,dst	操作数和源操作数相与 src .and. dst →dst	0* * *
BIC(.B) src dst	清除目的操作数的位 . not.src .and. dst →dst	– – – –
BIS(.B) src,dst	设置目的操作数的位 src .or. dst →dst	– – – –
BIT(.B) src,dst	测试目的操作数中的位 src .and. dst	0* * *
BR ↑ dst	转移程序到目的操作数中的地址中去 dst →PC	– – – –
CALL dst	调用目的地址处的程序 PC+2→stack, dst→PC	– – – –
CLR(.B) ↑ dst	清除目的操作数为 0 →dst	– – – –
CLRC ↑	清除进位标志位C 0 →C	– – – 0
CLRN ↑	清除负数标志位N 0→N	– 0 – –

CLRZ ↑	清除零标志位Z 0→Z	– – 0 –
CMP(.B) src,dst	比较操作数和源操作数 dst – src	* * * *
DADC(.B) ↑ dst	将进位标志位加到十进制的目的操作数中去 dst + C →dst	* * * *
DADD(.B) src,dst	操作数和源操作数的十进制加法 dst. src + dst + C→dst	* * * *
DEC(.B) ↑ dst	目的操作数减1 dst – 1→dst	* * * *
DECD(.B) ↑ dst	目的操作数减2 dst – 2 →dst	* * * *
DINT ↑	屏蔽中断 0 →GIE	– – – –
EINT ↑	中断使能 1 →GIE	– – – –
INC(.B) ↑ dst	目的操作数加1 dst +1 →dst	* * * *
INCD(.B) ↑ dst	目的操作数加2 dst+2 →dst	* * * *
INV(.B) ↑ dst	反转目的操作数 .not.dst →dst	* * * *
JC/JHS label	进位标志位被设置C跳转/大于或等于则跳转	– – – –
JEQ/JZ label	相等则跳转/零标志位被设置则跳转	– – – –
JGE label	大于或等于则跳转	– – – –
JL label	小于则跳转	– – – –
JMP label	无条件跳转 PC + 2 x offset →PC	– – – –
JN label	负数标志位为N被设置则跳转	– – – –
JNC/JLO label	进位标志位没有被设置则跳转/低于则跳转	– – – –
JNE/JNZ label	不相等则跳转/零标志位没有被设置则跳转	– – – –
MOV(.B) src,dst	将源操作数移到目的操作数中 src →dst	– – – –
NOP ↑	空指令	– – – –
POP(.B) ↑ dst	出栈 @SP →dst, SP+2 →SP	– – – –
PUSH(.B) src	入栈 SP – 2 →SP, src →@SP	– – – –
RET ↑	子程序返回 @SP →PC, SP + 2 →SP	– – – –
RETI	中断返回	* * * *
RLA(.B) ↑ dst	算术左移	* * * *
RLC(.B) ↑ dst	带进位的算术左移	* * * *
RRA(.B) dst	算术右移	0* * *
RRC(.B) dst	带进位的算术右移	* * * *
SBC(.B) ↑ dst	目的操作数减法借位 dst + 0FFFFh + C →dst	* * * *
SETC ↑	设置 C 1 →C	– – – 1
SETN ↑	设置 N 1 →N	–1 – –
SETZ ↑	设置 Z 1 →C	– – 1 –
SUB(.B) src,dst	从目的操作数和源操作数中相减 dst + .not.src + 1 →dst	* * * *
SUBC(.B) src,dst	带借位的源操作数和目的操作数相减. dst + .not.src + C →dst	* * * *
SWPB dst	交换字节	– – – –
SXT dst	符号扩展	0* * *
TST(.B) ↑ dst	测试目的操作数 dst + 0FFFFh + 1	0 * * 1
XOR(.B) src,dst	异或源操作数和目的操作数 src .xor. dst →dst	* * * *

表3–17



MSP430F22x4 评估板

专业提供 MSP430 单片机开发工具