dsPIC33A 程序员参考手册

简介

本文档详细介绍了 dsPIC33A 系列数字信号控制器的指令集，旨在指导进行 dsPIC®器件的本机汇编语言开发，从而优化和直接控制指令执行。汇编代码可用于简化和加速开发时间敏感型应用，包括嵌入式控制环和数据处理。有关这些器件和产品的信息以及相应的技术文档，可从 Microchip 网站（www.microchip.com）获得。

本手册的宗旨

本手册是 dsPIC33A 系列器件软件开发人员的参考手册。本手册对指令集进行了详细介绍，并提供通用信息以帮助用户进行 dsPIC33A 系列器件的软件开发。

本手册的内容没有包括有关内核、外设、系统集成或针对具体器件的详细信息。有关器件内核、外设以及系统集成的信息，请参见具体器件的系列参考手册。有关针对具体器件的信息，请参见具体器件的数据手册。数据手册提供的信息包括：

- 器件存储器映射
- 器件引脚分配和封装细节
- 器件电气特性
- 器件上包含的外设

本手册中给出了大量的代码示例。这些代码示例适用于 dsPIC33A 系列中的任何器件。

开发支持

Microchip 提供了大量的开发工具，使用户可以高效地开发和调试应用代码。Microchip 的开发工具可以分为四类：

- 代码生成
- 硬件/软件调试
- 器件编程器
- 产品评估板

用户可从 Microchip 网站（www.microchip.com）或当地的 Microchip 销售办事处获取有关最新工具、产品简介和用户指南的信息。

Microchip 还提供了其他参考工具来帮助用户加速开发过程，包括：

- 应用笔记
- 参考设计
- Microchip 网站
- 当地销售办事处提供的现场应用支持
- 公司的技术支持热线

Microchip 的网站列出了其他一些有用的链接。

风格和符号的约定

本文档采用了某些风格和字体格式约定。 表 1 说明了本文档中所使用的约定。

表 1. 文档约定

符号或术语	说明
置 1	强制某一位/寄存器的值为逻辑 1。
清零	强制某一位/寄存器的值为逻辑 0。
复位	1. 强制某一寄存器/位回到默认状态。 2. 复位器件后的状态。某些位将被强制为 0（如中断允许位），而其他位被强制为 1（如 I/O 数据方向位）。
0xnnnn	指定数据 nnnn 为十六进制数。这种约定用于代码示例中。例如，0x013F 或 0xA800。
:（冒号）	用来指定范围，或寄存器/位/引脚的组合。 如 ACCAU:ACCAH:ACCAL 表示由三个寄存器组成一个 72 位累加器。 组合顺序（左-右）通常指定一种位置关系（MSb 到 LSb，高位到低位）。
[ ]	指定特定寄存器中位的位置。如 SR[7:5]（或 IPL[2:0]）指定了寄存器和相关位或位的位置。
LSb 和 MSb	表示位域中的最低有效位或最高有效位。
LSB 和 MSB	表示位域中的最低有效字节/最高有效字节。
lsw 和 msw	表示位域中的最低有效字/最高有效字。
`Courier New` 字体	用于代码示例、二进制数以及文本中的指令助记符。
Times New Roman 字体，斜体	用于公式和变量。
Times New Roman 字体，粗体，斜体	用于图表、公式或示例中的说明文本。
注：	“注”表示希望再次强调的信息，帮助您避免常见的错误，或提醒您注意同一系列器件间的操作区别。“注”可以位于方框中；或用于表格或图中，这时它位于表格或图的下方。

指令集符号

在指令集概述和指令集汇总表中的汇总表以及指令说明的指令说明中使用了表 2 中所示的符号。

表 2. 指令汇总表和说明中使用的符号

符号[1]	说明
{ }	可选字段或操作
[text]	由文本寻址的地址
(text)	文本内容
#text	由文本定义的立即数
{label:}	可选标号名
[n:m]	寄存器位域
.l	32 位长字模式选择
.b	8 位字节模式选择
.sl	24 位（立即数）字模式选择
.V	目标数据值选择（MAXABW、MINABW 和 FLIMW）
.w	16 位字模式选择（默认）
AWB	累加器回写目标地址寄存器
bit3	3 位位选择字段（用于字节寻址指令）（0:7）
bit4	4 位位选择字段（用于字寻址指令）（0:15）
C、N、OV 和 Z	ALU 状态位：进位、半进位、负、溢出和全零标志位
d	文件寄存器目标（W0 和无）
Expr	绝对地址、标号或表达式（由链接器解析）
f	文件寄存器地址（0x0000:0xFFFF）或（0x00000:0xFFFFF）（可寻址空间因指令类别而异）
Fd[2]	32 个 FPU 目标数据寄存器（F0:F31）之一（寄存器直接寻址）
Fs[2]	32 个 FPU 源数据寄存器（F0:F31）之一（寄存器直接寻址）
FSR、FSRH、FCR 和 FEAR 1[2]	FPU 特殊（控制和状态）协处理器寄存器（寄存器直接寻址）
label	转换为表示标号名称位置的立即数
lit1	1 位无符号立即数（0:1）
lit3	3 位无符号立即数（0:7）
lit5	5 位无符号立即数（0:31）
lit6	6 位无符号立即数（0:63）
lit8	8 位无符号立即数（0:255）
lit16	16 位无符号立即数（0:65535）
lit24	24 位无符号立即数（0:1677215；如果是地址，LSb 必须是 0）
lit32	32 位无符号立即数（0:4294967295）
none	字段无需内容，可为空
OA、OB、SA 和 SB	DSC 状态位：ACCA 溢出、ACCB 溢出、ACCA 饱和和 ACCB 饱和
PC	程序计数器
Rdo	目标工作寄存器
Rnd	指令舍入模式[E,Z,P,N]
Rso	源工作寄存器
Slit6	有符号 6 位立即数（-32:31）
Slit7	有符号 7 位立即数（-64:63）
Slit8	有符号 8 位立即数（-128:127）
Slit20	有符号 20 位立即数（-524288:524287）
SR	状态寄存器
text1 ∈ {text2, {text3,…}	text1 一定在集合 text2,text3,...中
v	选择 MULxxx 指令操作数数据类型

.....（续）	
符号(1)	**说明**
Wb	基本工作寄存器
Wd	目标工作寄存器
Wm	16 个工作寄存器（W0:W15）之一
Wn	既是源工作寄存器又是目标工作寄存器（W0:W15）
Wnd	16 个目标工作寄存器之一
Wns	16 个源工作寄存器之一
Wm * Wm	用于平方指令的被乘数和乘数工作寄存器
Wm * Wn	用于 DSP 指令的被乘数和乘数 W 寄存器
Ws	源工作寄存器
Wx	用于 DSP 指令的 X 数据空间取地址寄存器
Wy	用于 DSP 指令的 Y 数据空间取地址寄存器

注：

1. 每个符号的范围取决于指令。对于特定指令的范围，请参见指令说明。
2. 仅当存在 FPU 协处理器时适用。

目录

1. dsPIC33A 内核架构概述

本节简要介绍 dsPIC33A 器件系列的特性和功能。

1.1 特定于 dsPIC33A 内核的特性

dsPIC33A 器件采用改进的哈佛架构内核，数据线宽度为 32 位，采用增强指令集。内核采用具有 8 位操作码字段的 32 位指令字。程序计数器（Program Counter，PC）为 24 位宽，可对最大为 4M x 24 位的用户程序存储空间进行寻址。指令预取机制用来帮助维持吞吐量并提供可预测的执行。大多数指令都在单个周期内执行。

1.1.1 寄存器

dsPIC33A 器件拥有 16 个 32 位工作寄存器。每个工作寄存器可作为数据、地址或偏移量寄存器。第 16 个工作寄存器（W15）用作中断和调用时的软件堆栈指针（Software Stack Pointer，SSP）。

1.1.2 指令集

16 位 MCU 和 DSC 架构具有几乎完全相同的指令集。该指令集包括多种寻址模式，且设计为确保最佳的 C 编译器效率。

1.1.3 寻址模式

内核支持固有（无操作数）寻址、相对寻址、立即数寻址、存储器直接寻址、寄存器直接寻址、寄存器间接寻址以及寄存器偏移量寻址模式。根据功能性要求的不同，每一条指令都与预先定义的寻址模式组相关联。任何一条指令可支持多达 7 种寻址模式。

对于大多数指令，CPU 可在每一指令周期内执行数据（或程序数据）存储器读、工作寄存器（数据）读、数据存储器写以及程序（指令）存储器读操作。因此可支持 3 操作数指令，即允许在单个周期内执行 A + B = C 操作。

1.1.4 算术逻辑单元

dsPIC33A 器件包括一个高速 33 位 x 33 位乘法器，显著提高了内核的运算能力和吞吐量。乘法器支持有符号和无符号模式的 32 位 x 32 位或 16 位 x 16 位整数乘法。所有乘法指令都在单个周期内执行。

16 位算术逻辑单元（Arithmetic Logic Unit，ALU）用支持迭代不恢复余数除法算法的整数除法支持硬件进行了增强。它和 REPEAT 指令循环机制（以及一些迭代除法指令）一起使用，支持用 16 位整数除 32 位（或 16 位）的有符号和无符号除法。

1.1.5 异常处理

dsPIC33A 器件具有向量异常机制，可支持最多 8 个不可屏蔽陷阱源和最多 502 个中断源。可以为每个中断源分配 7 个优先级之一。

1.1.6 64 位结果的 MCU 乘法

32x32 位 MUL 指令包含一个将乘积存储到一个 32 位工作寄存器（而非一对寄存器）的选项。在相乘的数量值较小而预期得到 16 位结果的情况下，该特性可释放一个寄存器用于其他用途。更多详细信息，请参见指令说明中各 MUL 指令的说明。

1.1.7 DSP 现场切换支持

DSP 溢出和饱和状态位是可写的。这允许在 DSP 任务间切换时高效保存和恢复 DSP 引擎的状态。关于 DSP 状态位的更多详细信息，请参见 DSP ALU 状态位。此外，还有 7 组额外的 DSP 累加器 A 和 B 用于快速进行现场切换；每组均固有地分配给各自的 IPL。

1.1.8 DSP 类指令

DSP 类指令无缝地集成到架构中，并从单个执行单元执行。

1.1.9 数据空间寻址

数据空间被分成两块，称为 X 和 Y 数据存储空间。每个存储块有各自独立的地址发生单元（Address Generation Unit，AGU）。MCU 类指令只通过 X 数据空间 AGU 进行操作，可将整个存储器映射作为一个线性数据空间访问。那些具有两个源操作数的 DSP 类指令通过 X 和 Y 的 AGU 进行操作，这将数据地址空间分成两个部分，X 和 Y 数据空间的边界视具体器件而定。

1.1.10 模寻址和位反转寻址

X 和 Y 地址空间都支持无开销循环缓冲区（模寻址）。模寻址省去了 DSP 算法的软件边界检查开销。此外，X AGU 的循环寻址可以与任何 MCU 类指令一起使用。X AGU 还支持位反转寻址，大幅简化了基为 2 的 FFT 算法对输入或输出数据的重新排序。

1.1.11 DSP 引擎

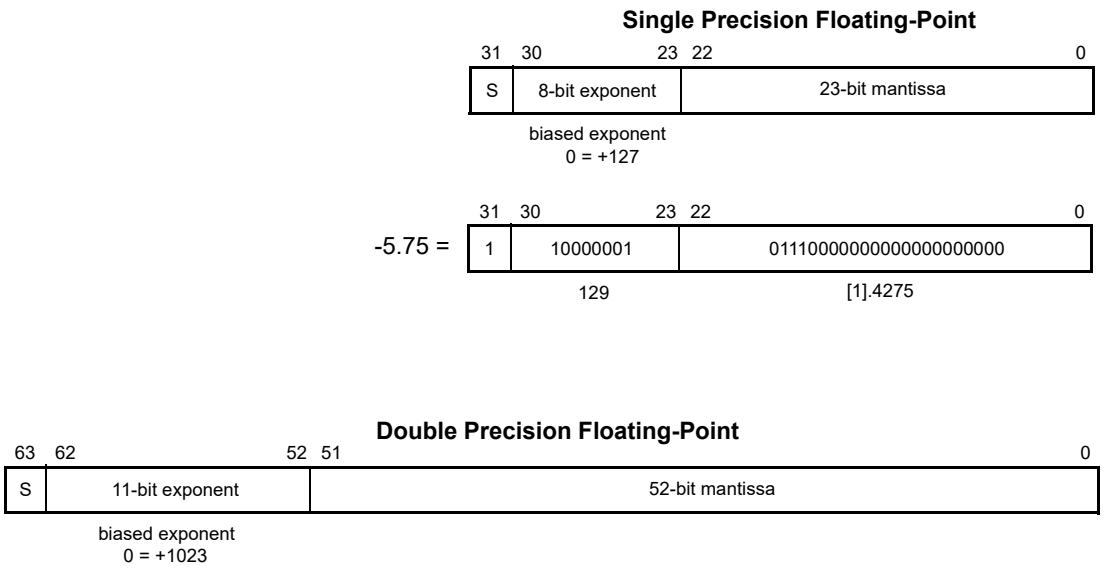
DSP 引擎具备一个高速 33 位 x 33 位乘法器、一个 72 位 ALU、两个 72 位饱和累加器和一个 72 位双向桶形移位寄存器。该桶形移位寄存器可在单个周期内将一个 72 位的值右移最多 32 位或左移最多 32 位。DSP 类指令可以无缝地与所有其他指令一起操作，设计为可实现最佳的实时性能。MAC 类指令和其他相关指令可以同时从存储器中取出两个数据操作数并将两个工作寄存器相乘。这要求数据空间对于这些指令拆分为两块，但对所有其他指令保持线性。这是通过为每个地址空间指定某些工作寄存器，以透明和灵活的方式实现的。

1.2 浮点单元（FPU）概述

IEEE 浮点运算标准（IEEE 754-2008）规定了浮点数据格式（如图 1-1 所示），具体由符号位、指数值和（小数）尾数值组成。dsPIC 浮点单元（Floating-Point Unit，FPU）支持大多数指令的单精度（32 位，SP）和双精度（64 位，DP）运算。

为避免指数中需要另一个符号位，IEEE 浮点格式的指数偏移 127（SP）或 1023（DP）。因此，对于任何数据，所需的 IEEE 指数值 = 数据指数 + 偏移量。此外，尾数最高有效位（Most Significant bit，MSb）左侧的 1 对于除非规格化数外的所有数字都是隐含的，所以被称为前导位约定“隐藏位”。因此，尾数是一个隐含整数值[1]的小数值。

图 1-1. IEEE 浮点数据格式和单精度示例



$(-1)^S$ x [1].m $_{base2}$ x $2^{(e-bias)}$

其中：

- “S”代表数字的符号（其值与有符号整型值相同）
- “e”代表指数值
- “m”代表小数尾数值
- “bias”为 127（SP）或 1023（DP）

例如，-5.75 = -(1.4275 x 2^2）。采用 IEEE SP 格式时，如下所示：

$(-1)^1$ x [1].4275 x $2^{(129-127)}$

或（如图 1-1 所示）：

S = 1，指数 = 129_{10}，尾数 = $[1].4275_{10}$

或： 0xC0B8 0000

1.2.1 浮点单元寄存器

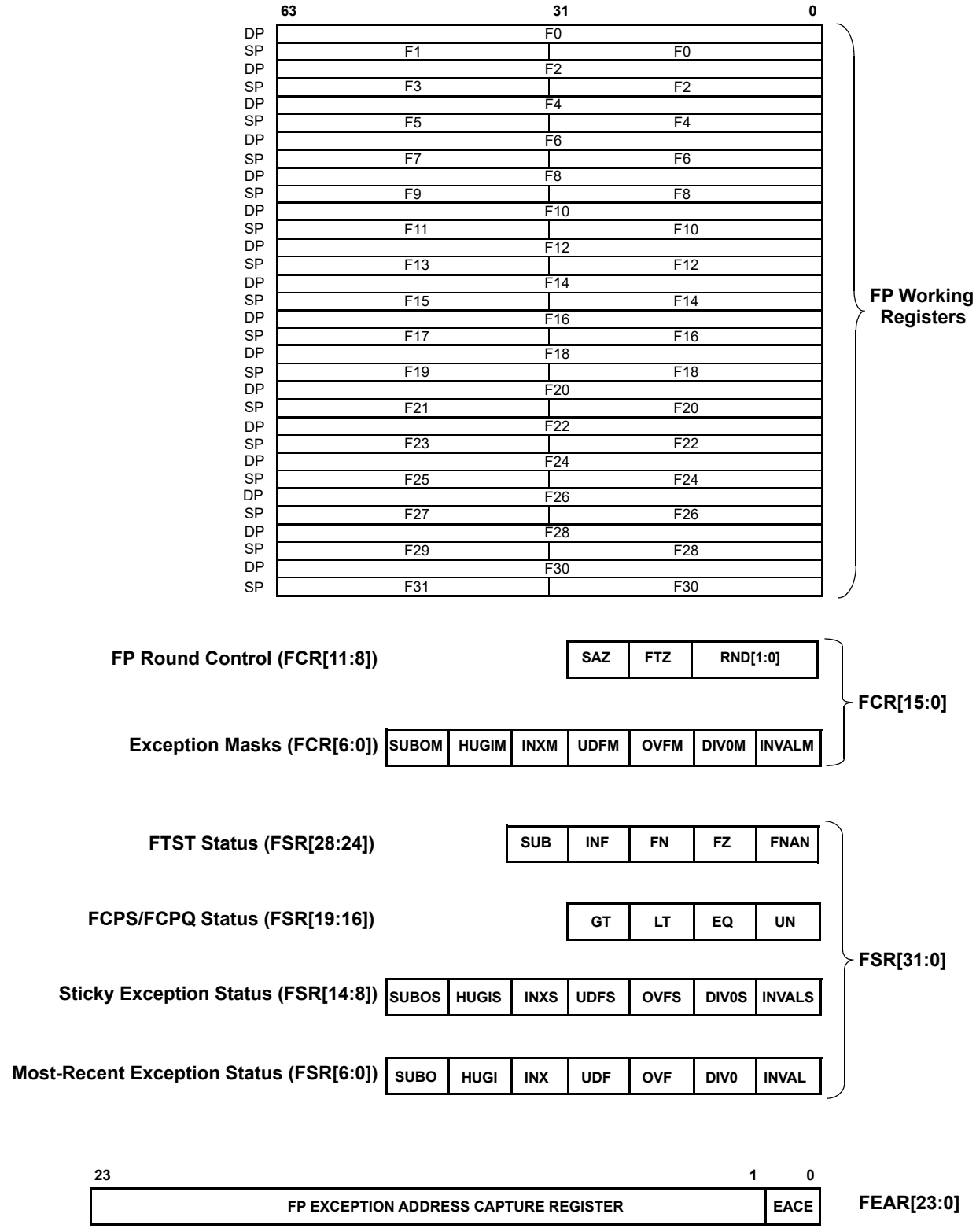
dsPIC 浮点单元（FPU）提供一大组工作寄存器（F-reg）：

- 32 位 x 32 位（单精度，F0 ...F31）
  或
- 16 位 x 64 位（双精度，F0，F2 ...F28, F30）
  或
- 两种大小的混合（按照图 1-2 所示进行排列）。

除了 F-reg 之外，还支持状态（FSR）和控制（FCR）寄存器，如图 1-2 所示：

- FSR（FPU 状态寄存器，32 位）：保存退役浮点指令的状态：
  - FSR[6:0]：指令“最近”异常状态
  - FSR[14:8]：指令“粘住”异常状态
  - FSR[19:16]：CPS/CPQ 指令状态
  - FSR[28:24]：FTST 指令状态
- FCR（FPU 控制寄存器，16 位）：
  - FCR[6:0]：异常屏蔽控制
  - FCR[9:8]：舍入模式控制
  - FCR[10]：非规格化结果“清零”（Flush To Zero，FTZ）控制
  - FCR[11]：非规格化操作数“非规格化为零”（Subnormal Are Zero，SAZ）控制
- FEAR：（FPU 异常地址捕捉寄存器，24 位）：
  - 保存导致异常的第一条指令的地址。对于 FPU 流水线中的所有后续指令，只要随后会退役，即使同样会产生异常，也不会影响 FEAR。

图 1-2. FPU 编程模型

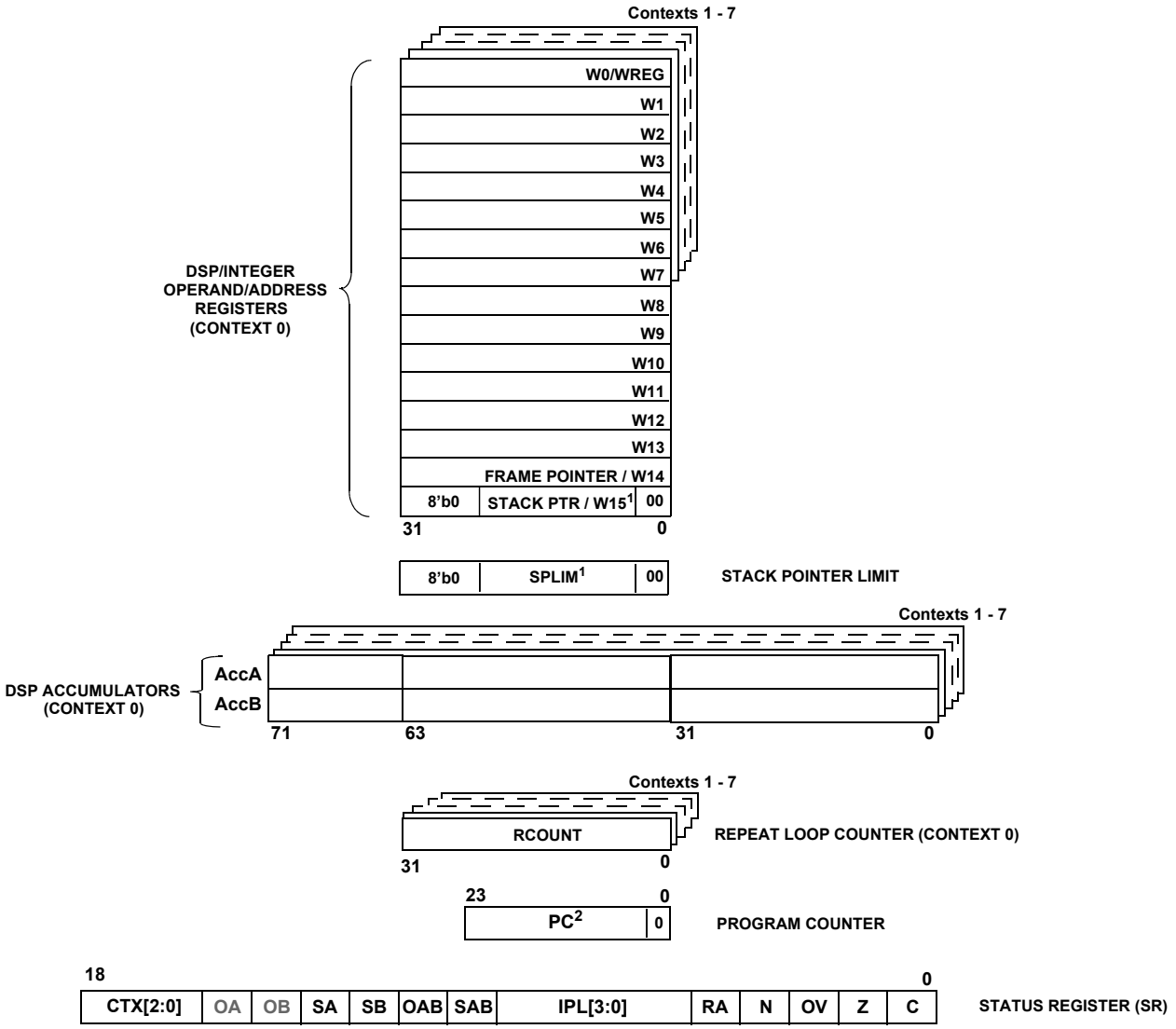


注：仅显示一个寄存器现场。

1.3 编程模型

图 1-3 给出了 dsPIC33A 系列器件的编程模型图。

图 1-3. dsPIC33A 编程模型



注：

1. W15[1:0]和 SPLIM[1:0]总是为 0b00。
2. PC[0]总是为 0b0。

表 1-1. 编程模型寄存器说明

寄存器	说明
CORCON	CPU 内核配置寄存器
PC	24 位程序计数器
RCOUNT	REPEAT 循环计数器寄存器
SPLIM	堆栈指针限制值寄存器
SR	ALU 和 DSP 引擎状态寄存器
W0-W15	工作寄存器阵列
ACCA 和 ACCB	72 位 DSP 累加器

1.4 工作寄存器阵列

16 个工作（W）寄存器可作为数据、地址或偏移量寄存器。W 寄存器的功能由访问它的指令所决定。

以工作寄存器阵列为目标寄存器的字节指令将只对目标寄存器的最低有效字节（Least Significant Byte，LSB）产生影响。

1.5 软件堆栈帧指针

帧是堆栈中用户定义的存储区域，函数使用帧来为局部变量分配存储区。W14 被指定作为分配堆栈帧（LNK）和释放堆栈帧（ULNK）指令中使用的堆栈帧指针。然而在未使用堆栈帧指针、LNK 和 ULNK 指令时，W14 可被任何指令以与所有其他 W 寄存器相同的方式使用。有关帧指针的详细信息，请参见软件堆栈帧指针。

1.6 软件堆栈指针

W15 专用作软件堆栈指针，将会被函数调用、异常处理以及返回指令自动修改。然而，W15 可像所有其他 W 寄存器一样被任何指令引用。这将简化对堆栈指针的读、写和控制操作。有关堆栈指针的详细信息，请参见软件堆栈指针。

1.7 堆栈指针限制寄存器（SPLIM）

SPLIM 是与堆栈指针有关的 32 位寄存器，用来防止堆栈指针上溢以及对超出用户分配堆栈存储区的存储空间进行访问。有关 SPLIM 的详细信息，请参见堆栈指针上溢。

1.8 累加器 A 和累加器 B

累加器 A（ACCA）和累加器 B（ACCB）皆为 72 位宽的寄存器。DSP 类指令通过累加器执行数学和移位操作。

此外，也可将累加器 A 和累加器 B 用作 MCU MUL.xx 指令中的目标寄存器。这有助于缩短扩展精度算术运算的执行时间。

有关 ACCA 和 ACCB 使用的详细信息，请参见图 3-13。

1.9 程序计数器

程序计数器（PC）为 24 位宽。通过 PC[22:1]对 4M x 24 位用户程序存储空间中的指令进行寻址，其中 PC[0]总是设置为 0，以保持指令字对齐并提供与数据存储空间寻址的兼容性。这表明在正常的指令执行过程中，PC 值将以 2 为步长进行递增。

1.10 RCOUNT 寄存器

32 位 RCOUNT 寄存器包含用于 REPEAT 指令的循环计数器。当执行 REPEAT 指令时，RCOUNT 将装载指令重复执行的次数，如“REPEAT #lit20”指令中的“lit20”、“REPEAT #lit5”指令中的“lit5”或“REPEAT Wn”指令中的 Wn 寄存器。REPEAT 循环将被执行 RCOUNT + 1 次。

注：如果 REPEAT 循环执行过程被中断，则当再次进入前台代码时中断服务程序（Interrupt Service Routine，ISR）可将 RCOUNT 清零以退出 REPEAT 循环。

1.11 状态寄存器

32 位状态寄存器用于保存最近执行指令的状态信息。其中操作状态位用于 MCU 操作、循环操作以及 DSP 操作。此外，状态寄存器还包含用作现场标识符和用于中断处理的 CPU 中断优先级位 IPL[2:0]。更多详细信息，请参见 SR。

1.11.1 MCU ALU 状态位

指令集中的大多数指令都会影响或使用 MCU 操作状态位。大多数逻辑、数学、循环/移位和位操作指令都将在执行后修改 MCU 状态位。条件转移指令使用各状态位的状态来确定程序执行流。条件转移指令列出了所有条件转移指令。

进位（C）、全零（Z）、溢出（OV）和负（N）位用来指示 MCU ALU 的当前状态。这些状态位的状态分别表明指令执行结果是否导致进位、全零、溢出或负的结果等事件。当执行减法操作时，C 标志位可用作借位标志。

Z 状态位对于扩展精度的算术运算很有用。除使用进位或借位输入的指令（ADDC、CPB、SUBB 和 SUBBR）外，所有指令都使用 Z 状态位作为普通 Z 标志。更多详细信息，请参见 Z 状态位。

注：

1. 在执行 PUSH.S 指令时，所有 MCU 位都被保存到影子寄存器中。在执行 POP.S 指令时，它们将恢复原先的内容。
2. 在异常处理过程中，所有 MCU 位都将被压入堆栈（见软件堆栈指针）。

1.11.2 REPEAT 循环激活（RA）状态位

REPEAT 循环激活位（RA）用来指明循环是否处于激活状态。RA 标志表明 REPEAT 指令是否正在执行，而且只有 REPEAT 指令才会对该标志位的状态产生影响。当被重复的指令开始执行时，RA 标志位将被置 1。当被重复的指令完成最后一次执行时，该标志位将被清零。

由于 RA 标志也是只读位，因此也不能对该位直接清零。 然而，如果 REPEAT 或其目标指令被中断，中断服务程序可将位于堆栈中的 RA 标志位清零。该操作将在程序执行从中断服务程序返回时禁止循环操作，因此此时恢复的 RA 将为 0。

1.11.3 DSP ALU 状态位

DSP 类指令使用状态寄存器的高位字节，当数据经过其中一个加法器时，高位字节中的内容将被修改。高位字节为两个累加器均提供了溢出和饱和状态信息。饱和 A、饱和 B、溢出 A 和溢出 B（SA、SB、OA 和 OB）状态位提供了单独的累加器状态。而饱和 AB 和溢出 AB（SAB 和 OAB）状态位提供了联合的累加器状态。SAB 和 OAB 状态位使得软件开发人员可高效地检查寄存器是否出现饱和或溢出状况。

OA 和 OB 状态位用来指示一个操作是否产生溢出至相应累加器的警戒位（bit 63-71）。只有当处理器处于超饱和模式或饱和模式被禁止时，这种情况才会发生。这表明用累加器的低 62 位无法表示操作产生的数值。

SA 和 SB 状态位用来指示一个操作是否从相应累加器的最高位产生溢出。无论何种饱和模式（禁止、普通或超饱和），SA 和 SB 位都处于“粘住”状态。即当 SA 或 SB 位置 1 时，该位只能由软件手动清零，而与后续的 DSP 操作无关。在需要时，可以使用 BCLR 指令对 SA 或 SB 位进行清零。

此外，SA 和 SB 位可以用软件置 1，从而支持高效的现场状态切换。

为简便起见，OA 和 OB 状态位通过逻辑或形成 OAB 标志，而将 SA 和 SB 状态位通过逻辑或形成 SAB 标志。当实现算法时，通过这些累加器状态位可实现高效的溢出和饱和检查。在进行算术运算溢出检查时，只需查询 OAB 即可完成，而不再需要单独查询 OA 和 OB 状态位。同样，当进行饱和检查时，可只检查 SAB 而不是所有 SA 和 SB 状态位。注意，清零 SAB 标志将同时对 SA 和 SB 位进行清零。

1.11.4 中断优先级状态位

SRL 中的三个中断优先级（Interrupt Priority Level，IPL）位（SR[7:5]）以及 IPL3 位（SR[8]）对用于异常处理的 CPU IPL 进行设置。异常包括中断和硬件陷阱。中断具有的优先级为 0 至 7，可由用户进行定义，而硬件陷阱的优先级是固定的，为 8 至 15。第 4 个中断优先级位 IPL3 为一个特殊的 IPL 位，只可由用户对其进行读或清零。该位只在硬件陷阱激活时才被置 1，在陷阱处理后被清零。

CPU 的 IPL 用来标识可中断处理器的最低优先级的异常。当等待处理的异常具有比 CPU IPL 更高的中断优先级时，CPU 才会对其进行处理。这表明，如果 IPL 为 0，所有优先级为 1 或更高的异常都可中断处理器。如果 IPL 为 7，只有硬件陷阱才可中断处理器。

当正在处理一个异常时，IPL 将自动设置为该异常的优先级，这将禁止所有同等或更低优先级的异常。然而，由于 IPL 位域为可读/写，用户可在中断服务程序中修改 IPL 的低 3 位值以控制何种异常具有优先处理

权。由于在异常处理过程中 SRL 将被压入堆栈，因此总是在异常处理之后恢复原先的 IPL。如果需要，用户也可通过置 1 NSTDIS 位（INTCON1[15]）来阻止异常嵌套。

1.12 内核控制寄存器

内核控制寄存器（CORCON）用于设置 CPU 的配置。

除了设置 CPU 模式之外，还可以通过 CORCON 寄存器使用以下功能：

- 使能 ACCA 和 ACCB 的饱和功能
- 设置数据空间写饱和模式
- 设置累加器饱和和舍入模式
- 设置 DSP 操作的乘法器模式

1.13 影子寄存器

影子寄存器用作暂存寄存器。在某些事件发生时，影子寄存器与其相关主寄存器之间能够实现存储内容的传送。编程模型中的一些寄存器具有影子寄存器。在 POP.S 或 PUSH.S 指令的执行过程中，将会使用到这些影子寄存器。表 1-2 给出了影子寄存器的使用。

表 1-2. 自动影子寄存器使用

存储单元	DO	POP.S/PUSH.S
状态寄存器——DC、N、OV、Z 和 C 位	—	使用
W0-W3	—	使用

注：所有影子寄存器的深度皆为一个寄存器，且不能对其进行直接访问。更多层影子寄存器深度可通过软件堆栈用软件实现。

1.14 CPU 状态寄存器

名称： SR

注：

1. 可对该位进行读或清零操作（不能置 1）。对该位进行清零将导致 SA 和 SB 清零，与当时写入 SA 和/或 SB 的值无关。
2. 如果 NSTDIS（INTCON1[15]）= 1（禁止嵌套），则 IPL[2:0]变为只读位。

图注： C = 可清零位

位	31	30	29	28	27	26	25	24
访问								
复位								

位	23	22	21	20	19	18	17	16
	VF					CTX[2:0]		
访问	R					R	R	R
复位	0					0	0	0

位	15	14	13	12	11	10	9	8
	OA	OB	SA	SB	OAB	SAB		IPL3
访问	R/W	R/W	R/W	R/W	R	R/C		R/C
复位	0	0	0	0	0	0		0

位	7	6	5	4	3	2	1	0
	IPL[2:0]			RA	N	OV	Z	C
访问	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W
复位	0	0	0	0	0	0	0	0

Bit 23 – VF 向量（取）失败状态

值	说明
1	向总线错误处理程序指示总线错误源自向量取。读取的向量数据将替换为向量失败地址（Vector Fail Address，VFA）SFR 的内容。
0	向总线错误处理程序指示总线错误并非源自向量取。

Bit 18:16 – CTX[2:0] 当前（W 寄存器）现场标识符
标识 CPU 当前正在使用哪个 W 寄存器现场

值	说明
111	当前正在使用现场 7
110	当前正在使用现场 6
101	当前正在使用现场 5
100	当前正在使用现场 4
011	当前正在使用现场 3
010	当前正在使用现场 2
001	当前正在使用现场 1
000	当前正在使用现场 0

Bit 15 – OA 累加器 A 小数溢出状态

值	说明
1	发生累加器 A 小数溢出（其内容无法再表示为 1.31 格式的小数值）
0	累加器 A 未溢出

Bit 14 – OB 累加器 B 小数溢出状态

值	说明
1	发生累加器 A 小数溢出（其内容无法再表示为 1.31 格式的小数值）
0	累加器 A 未溢出

Bit 13 – SA 累加器 A 饱和/符号溢出“粘住”状态

值	说明
1	累加器 A 饱和或在某时已经饱和，或者已溢出至 bit 71（如果禁止饱和）
0	累加器 A 未饱和，或者尚未溢出至 bit 71（如果禁止饱和）

Bit 12 – SB 累加器 B 饱和/符号溢出“粘住”状态

值	说明
1	累加器 B 饱和或在某时已经饱和，或者已溢出至 bit 71（如果禁止饱和）
0	累加器 B 未饱和，或者尚未溢出至 bit 71（如果禁止饱和）

Bit 11 – OAB OA || OB 联合累加器小数溢出状态

值	说明
1	发生累加器 A 或 B 小数溢出（至少其中之一的内容无法再表示为 1.31 格式的小数值）
0	累加器 A 和 B 均未溢出

Bit 10 – SAB SA || SB 联合累加器“粘住”状态(1)

值	说明
1	累加器 A 或 B 饱和或在某时已经饱和，或者已溢出至 bit 71（如果禁止饱和）
0	累加器 A 和 B 均未饱和，或者均尚未溢出至 bit 71（如果禁止饱和）

Bit 8 – IPL3 CPU 优先级半字节的 MSb（最高有效位）

值	说明
1	CPU 优先级 ≥ 8（正在发生陷阱异常）
0	CPU 优先级 < 8（未发生陷阱异常）

Bit 7:5 – IPL[2:0] CPU 中断优先级状态位(2)

值	说明
111	禁止所有中断
110	允许优先级 7 中断
101	允许优先级 6 和 7 中断
100	允许优先级 5-7 中断
011	允许优先级 4-7 中断
010	允许优先级 3-7 中断
001	允许优先级 2-7 中断
000	允许优先级 1-7 中断

Bit 4 – RA REPEAT 循环激活

值	说明
1	REPEAT 循环在进行
0	REPEAT 循环不在进行

Bit 3 – N MCU ALU 负位

值	说明
1	结果为负
0	结果非负（零或正值）

Bit 2 – OV MCU ALU 溢出位

该位用于有符号的算术运算（以二进制补码方式进行）。它表示量值上的溢出，这种溢出将导致符号位改变状态。

值	说明
1	有符号算术运算中发生溢出（本次算术运算）
0	未发生溢出

Bit 1 – Z MCU ALU“粘住”零位

值	说明
1	影响 Z 位的任何运算在过去某时已将该位置 1
0	影响 Z 位的最近一次运算已将该位清零（即运算结果非零）

Bit 0 – C MCU ALU 进位/$\overline{\text{借位}}$位

值	说明
1	结果的 MSb 发生进位
0	结果的 MSb 未发生进位

1.15 内核控制寄存器

名称： CORCON

注：

1. 如果 US = 1（无符号模式），则该位无作用。

位	31	30	29	28	27	26	25	24
访问								
复位								

位	23	22	21	20	19	18	17	16
访问								
复位								

位	15	14	13	12	11	10	9	8
				US				
访问				R/W				
复位				0				

位	7	6	5	4	3	2	1	0
	SATA	SATB	SATDW	ACCSAT			RND	IF
访问	R/W	R/W	R/W	R/W			R/W	R/W
复位	0	0	1	0			0	0

Bit 12 – US DSP 乘法无符号/有符号控制位

值	说明
1	使能 DSP 引擎无符号模式
0	使能 DSP 引擎有符号模式

Bit 7 – SATA ACCA 饱和使能位(1)

值	说明
1	使能累加器 A 饱和
0	禁止累加器 A 饱和

Bit 6 – SATB ACCB 饱和使能位(1)

值	说明
1	使能累加器 B 饱和
0	禁止累加器 B 饱和

Bit 5 – SATDW DSP 引擎的数据空间写饱和使能位

值	说明
1	使能数据空间写饱和
0	禁止数据空间写饱和

Bit 4 – ACCSAT 累加器饱和模式选择位

值	说明
1	9.31 饱和（超饱和）

值	说明
0	1.31 饱和（正常饱和）

Bit 1 – RND 舍入模式选择位

值	说明
1	使能有偏（常规）舍入
0	使能无偏（收敛）舍入

Bit 0 – IF 整数或小数乘法器模式选择位

值	说明
1	使能整数模式的 DSP 乘法运算
0	使能小数模式的 DSP 乘法运算

2. 指令集概述

dsPIC33A 指令集提供了一整套支持传统单片机应用的指令，以及一类支持数学密集型应用的指令。由于保留了几乎所有 16 位 PIC MCU 和 DSC 指令集的功能，该混合指令集使得已熟悉 PIC 单片机的用户可以很容易地移植到 32 位器件。

指令可分为不同的功能组，如表 2-1 所示。 表 2 对指令汇总表中使用的符号进行了定义。 表 2-1 至表 2-12 对每一条指令的语法、说明、存储以及执行要求进行了定义。存储要求以 32 位指令字的方式来表征，而执行要求以指令周期的方式来表征。

表 2-1. 指令组

功能组	汇总表
传送指令	表 2-2
数学指令	表 2-3
逻辑指令	表 2-4
循环/移位指令	表 2-5
位操作指令	表 2-6
比较/跳过和比较/转移指令	表 2-7
程序流指令	表 2-8
影子/堆栈指令	表 2-9
控制指令	表 2-10
DSP 类指令	表 2-11
FPU 指令	表 2-12

多数指令具有多种不同的寻址模式以及执行流，这需要不同的指令形式。例如，最多有 7 种不同的 ADD 指令，而每一种指令形式都具有其自身的指令编码。指令说明提供了指令格式的说明以及特定指令的操作。除此之外，参考信息给出了一个以阿拉伯字母顺序排列的指令集总表。

2.1 多周期指令

如指令汇总表所示，大多数指令的执行时间为一个指令周期，但存在以下例外：

- 指令 MOV.D、POP.D 和 PUSH.D 的执行时间为两个指令周期。
- 指令 DIV.S、DIV.U 和 DIVF 为单周期指令，应该作为 REPEAT 指令的目标被连续执行多次。
- 改变程序计数器的指令执行时间需要两个指令周期。CALL 等指令的指令时间同样需要两个指令周期。
- RETFIERETLW 和 RETURN 是改变程序计数器的特殊指令。这些指令的执行时间为三个指令周期。如果有异常等待处理，将执行两个周期。

2.2 多字指令

如表 2-2 至表 2-12 所定义，几乎所有的指令都占用一个指令字（32 位），但表 2-8 中所列的程序流指令 CALL 和 GOTO 例外。因为这些指令的操作码中包含较大的立即数操作数，因此这些指令需要两个字的存储空间。

2.3 指令集汇总表

表 2-2. 传送指令

汇编语法	说明	字数	周期数
EXCH Wns,Wnd	交换 Wns 和 Wnd 的内容	1	2
MOV Rso,Rdo	将 Ws 中内容传送至 Wd	0.5/1	1
MOV.l lit32,Wnd	将 32 位无符号立即数传送至 Wnd	2	2

.....（续）

汇编语法	说明	字数	周期数
MOV.sl lit24,Wnd	将 24 位无符号立即数传送至 Wnd；经过零扩展至 32 位	1	1
MOV.w lit16,Wnd	将 16 位无符号立即数传送至 Wnd；经过零扩展至 32 位	1	1
MOV.bz lit8,Wnd	将 8 位无符号立即数传送至 Wnd；经过零扩展至 32 位	1	1
MOV.l [W15-lit7], Wnd [W14+slit7], Wnd	使用 SP 或 FP 将[系统堆栈 + 立即数偏移量]中内容传送至 Wnd	0.5	1
MOV.l Wns, [W15-lit7] Wns, [W14+slit7]	使用 SP 或 FP 将 Wns 中内容传送至[系统堆栈 + 立即数偏移量]	0.5	1
MOV.l f,Wnd	将 f 中内容传送至 Wnd（字或长字）（f < ~1 MB）	1	1
MOV.w f,Wnd	将 f 中内容传送至 Wnd（字或长字）（f > ~1 MB）	2	2
MOV.b f,Wnd	将 f 中内容传送至 Wnd（字节）	1	1
MOV.l Wns,f	将 Wns 中内容传送至 f（字或长字）（f < ~1 MB）	1	1
MOV.w Wns,f	将 Wns 中内容传送至 f（字或长字）（f > ~ 1 MB）	2	2
MOV.b Wns,f	将 Wns 中内容传送至 f（字节）	1	1
MOV [Wns+Slit12],Wnd	将[Wns+Slit12]中内容传送至 Wnd	1	1
MOV Wns,[Wnd+Slit12]	将 Wns 中内容传送至[Wnd+Slit12]	1	1
MOVIF.l CC, Wb, Wns, Wd	如果 SR.Z = 1 将 W1 中内容传送至[W15++] 否则将 W2 中内容传送至[W15++]	1	1
MOVIF.w CC, Wb, Wns, Wd	如果 SR.Z = 1 将 W1 中内容传送至[W15++] 否则将 W2 中内容传送至[W15++]	1	1
MOVIF.bz CC, Wb, Wns, Wd	如果 SR.Z = 1 将 W1 中内容传送至[W15++] 否则将 W2 中内容传送至[W15++]	1	1
MOVIF.b CC, Wb, Wns, Wd	如果 SR.Z = 1 将 W1 中内容传送至[W15++] 否则将 W2 中内容传送至[W15++]	1	1
MOVR.l	将 Ws 中内容传送至 Wd 且目标位取反		
MOVR.w	将 Ws 中内容传送至 Wd 且目标位取反	1	1
MOVS.l slit16, Wd	将符号扩展 16 位立即数传送至 Wd	1	1
MOVS.w slit16, Wd	将 16 位立即数传送至 Wd；如果为寄存器直接寻址模式，则将经过符号扩展为 32 位。	1	1
MOVS.b slit8, Wnd	将 8 位立即数传送至 Wd；无扩展。	1	1
SWAP Wn	Wn = 将 Wn 的两个字或字节交换	1	1
TST f	测试 f	1	1
TST f,Wnd	测试 f 并将 f 中内容传送至 Wnd	1	1

表 2-3. 数学指令

汇编语法	说明	字数	周期数
ADD f,Wn	f = f + Wn	1	1

.....（续）

汇编语法	说明	字数	周期数
ADD f,Wn,Wn	Wn = f + Wn	1	1
ADD.l lit5,Wn	Wn = Wn + lit5	0.5	1
ADD lit16,Wn	Wn = Wn + lit16	1	1
ADD Wb,Ws,Wd	Wd = Wb + Ws	0.5/1	1
ADD Wb,lit7,Wd	Wd = Wb + lit7（立即数零扩展）	1	1
ADDC f,Wn	f = f + Wn + (C)	1	1
ADDC f,Wn,Wn	Wn = f + Wn + (C)	1	1
ADDC lit16,Wn	Wn = Wn + lit16 + (C)	1	1
ADDC Wb,Ws,Wd	Wd = Wb + Ws + (C)	0.5/1	1
ADDC Wb,lit7,Wd	Wd = Wb + lit7 + (C)（立即数零扩展）	1	1
DEC f	f = f -1	1	1
DEC f,Wd	W5 = f -1	1	1
DEC Ws,Wd	Wd = Ws - 1	1	1
DEC2 f	f = f -2	1	1
DEC2 f,Wd	W5 = f -2	1	1
DEC2 Ws,Wd	Wd = Ws - 2	1	1
DIVF Wm/Wn	可中断的有符号 16 位/16 位或 32 位/16 位小数除法	1	1
DIVFL Wm/Wn	可中断的有符号 32 位/32 位小数除法	1	1
DIVS.w Wm/Wn	可中断的有符号 16 位/16 位整数除法	1	1
DIVS.l Wm/Wn	可中断的有符号 32 位/16 位整数除法	1	1
DIVSL Wm/Wn	可中断的有符号 32 位/32 位整数除法	1	1
DIVU.w Wm/Wn	可中断的无符号 16 位/16 位整数除法	1	1
DIVU.l Wm/Wn	可中断的无符号 32 位/16 位整数除法	1	1
DIVUL Wm/Wn	可中断的无符号 32 位/32 位整数除法	1	1
FLIM Wb, Ws	强制执行数据范围限制（上限和下限），不保存超限结果	1	1
FLIM Wb, Ws, Wd	强制执行数据范围限制（上限和下限），保存超限标志（Wd = -1）	1	2
FLIM.V Wb, Ws, Wd	强制执行数据范围限制（上限和下限），保存超限结果	1	2
INC f	f = f + 1	1	1
INC f,Wd	W5 = f + 1	1	1
INC Ws,Wd	Wd = Ws + 1	1	1
INC2 f	f = f + 2	1	1
INC2 f,Wd	W5 = f + 2	1	1
INC2 Ws,Wd	Wd = Ws +2	1	1
MULSS Wb,Ws,Wnd	{Wd} = 有符号(Wb) * 有符号(Ws)	0.5/1	1
MULSU Wb,Ws,Wnd	{Wd} = 有符号(Wb) * 无符号(Ws)	0.5/1	1
MULUS Wb,Ws,Wnd	{Wd} = 无符号(Wb) * 有符号(Ws)	0.5/1	1
MULUU Wb,Ws,Wnd	{Wd} = 无符号(Wb) * 无符号(Ws)	0.5/1	1
MULSU Wb,lit8,Wnd	{Wd} = 有符号(Wb) * 无符号(lit8)	1	1
MULUU Wb,lit8,Wnd	{Wd} = 无符号(Wb) * 无符号(lit8)	1	1
MULSS Wb,slit8,Wnd	{Wd} = 有符号(Wb) * 有符号(slit8)	1	1
MULUS Wb,slit8,Wnd	{Wd} = 无符号(Wb) * 有符号(slit8)	1	1
MUL f, Wn	W2 = f * Wn	1	1

.....（续）

汇编语法	说明	字数	周期数
SE Rso,Wnd	Wd = 符号扩展 Ws 内容	0.5/1	1
SUB f,Wn	f = f - Wn	1	1
SUB f,Wn,Wn	Wn = f - Wn	1	1
SUB.l lit5,Wn	Wn = Wn - lit5	0.5	1
SUB lit16,Wn	Wn = Wn - lit16	1	1
SUB Wb,Ws,Wd	Wd = Wb - Ws	0.5/1	1
SUB Ws,lit7,Wd	Wd = Ws - lit7（立即数零扩展）	1	1
SUBB f,Wn	f = f - Wn - (C)	1	1
SUBB f,Wn,Wn	Wn = f - Wn - (C)	1	1
SUBB lit16,Wn	Wn = Wn - lit16 - (C)	1	1
SUBB Wb,Ws,Wd	Wd = Wb - Ws - (C)	0.5/1	1
SUBB Ws,lit7,Wd	Wd = Ws - lit7（立即数零扩展）	1	1
SUBR f,Wn	f = Wn - f	1	1
SUBR f,Wn,Wn	Wn = Wn - f	1	1
SUBR Wb,Ws,Wd	Wd = Ws - Wb	0.5/1	1
SUBR Ws,lit7,Wd	Wd = lit7 - Ws（立即数零扩展）	0.5/1	1
SUBBR f,Wn	f = Wn - f - (C)	1	1
SUBBR f,Wn,Wn	Wn = Wn -f - (C)	1	1
SUBBR Wb,Ws,Wd	Wd = Ws - Wb - (C)	0.5/1	1
SUBBR Ws,lit7,Wd	Wd = lit7 - Ws - (C)（立即数零扩展）	1	1
ZE Rso,Wnd	Wd = 零扩展 Ws	0.5/1	1

表 2-4. 逻辑指令

汇编语法	说明	字数	周期数
AND f,Wn	f = f .AND. Wn	1	1
AND f,Wn,Wn	W0 = f .AND. Wn	1	1
AND lit16,Wn	Wn = Wn .AND. lit16	1	1
AND Wb,Ws,Wd	Wd = Wb .AND.Ws	0.5/1	1
AND Wb,lit7,Wd	Wd = Wb .AND.Lit7（立即数零扩展）	1	1
AND1 Wb,lit7,Wd	Wd = Wb .AND.Lit7（立即数零扩展）	1	1
CLR f	f = 0x0000	1	1
CLR Wd	Wd = 0x0000	1	1
COM f	f = $\overline{f}$	1	1
COM f,Wd	Wd = $\overline{f}$	1	1
COM Ws,Wd	Wd = $\overline{Ws}$	0.5/1	1
IOR f,Wn	f = f .IOR. Wn	1	1
IOR f,Wn,Wn	Wn = f .IOR. Wn	1	1
IOR lit16,Wn	Wn = Wn .IOR. lit16	1	1
IOR Wb,Ws,Wd	Wd = Wb .IOR.Ws	0.5/1	1
IOR Wb,lit7,Wd	Wd = Wb .IOR. lit7	1	1
NEG f	f = $\overline{f}$ + 1	1	1
NEG f,Wd	Wd = $\overline{f}$ + 1	1	1
NEG Ws,Wd	Wd = $\overline{Ws}$ + 1	0.5/1	1
SETM f	f = 0xFFFF	1	1
SETM Wd	Wd = 0xFFFF	1	1
XOR f,Wn	f = f .XOR. Wn	1	1

.....（续）			
汇编语法	**说明**	**字数**	**周期数**
XOR f,Wn,Wn	Wn = f .XOR. Wn	1	1
XOR lit16,Wn	Wn = Wn .XOR. lit16	1	1
XOR Wb,Ws,Wd	Wd = Wb .XOR.Ws	0.5/1	1
XOR Wb,lit7,Wd	Wd = Wb .XOR.Lit7（立即数零扩展）	1	1

表 2-5. 循环/移位指令

汇编语法	说明	字数	周期数
ASR f	f = f 算术右移 1 位	1	1
ASR f,Wn	Wn = f 算术右移 1 位	1	1
ASR Ws,Wd	Wd = Ws 算术右移 1 位	0.5/1	1
ASR Ws,Wb,Wd	Wnd = Ws 算术右移 Wb 位	0.5/1	1
ASR Ws,lit5,Wd	Wnd = Ws 算术右移 lit5 位	0.5/1	1
ASRM Ws, lit5, Wnd	Wnd = Ws 算术右移 lit5 位，然后与下一个 lsw 进行逻辑或运算	1	2
ASRM Ws, Wb, Wnd	Wnd = Ws 算术右移 Wb 位，然后与下一个 lsw 进行逻辑或运算	1	2
LSR f	f = f 逻辑右移 1 位	1	1
LSR f,Wd	Wd = f 逻辑右移 1 位	1	1
LSR Ws,Wd	Wd = Ws 逻辑右移 1 位	0.5/1	1
LSR Ws,Wb,Wd	Wnd = Ws 逻辑右移 Wns 位	0.5/1	1
LSR Ws,lit5,Wd	Wnd = Ws 逻辑右移 lit5 位	0.5/1	1
LSRM Ws, lit5, Wnd	Wnd = Ws 逻辑右移 lit5 位，然后与下一个 lsw 进行逻辑或运算	1	2
LSRM Ws, Wb, Wnd	Wnd = Ws 逻辑右移 Wb 位，然后与下一个 lsw 进行逻辑或运算	1	2
RLC f	f = 带进位位循环左移 f 内容	1	1
RLC f,Wd	Wd = 带进位位循环左移 f 内容	1	1
RLC Ws,Wd	Wd = 带进位位循环左移 Ws 内容	0.5/1	1
RLNC f	f = 不带进位位循环左移 f 内容	1	1
RLNC f,Wd	Wd = 不带进位位循环左移 f 内容	1	1
RLNC Ws,Wd	Wd = 不带进位位循环左移 Ws 内容	0.5/1	1
RRC f	f = 带进位位循环右移 f 内容	1	1
RRC f,Wd	Wd = 带进位位循环右移 f 内容	1	1
RRC Ws,Wd	Wd = 带进位位循环右移 Ws 内容	0.5/1	1
RRNC f	f = 不带进位位循环右移 f 内容	1	1
RRNC f,Wd	Wd = 不带进位位循环右移 f 内容	1	1
RRNC Ws,Wd	Wd = 不带进位位循环右移 Ws 内容	0.5/1	1
SL f	f = f 左移 1 位	1	1
SL f,Wd	Wd = f 左移 1 位	1	1
SL Ws,Wd	Wd = Ws 左移 1 位	0.5/1	1
SL Ws,Wb,Wnd	Wnd = Wb 左移 Wns 位	0.5/1	1
SL Ws,lit5,Wnd	Wnd = Ws 左移 lit5 位	0.5/1	1
SLM Ws, lit5, Wnd	Wnd = Wb 左移 lit5 位，然后与下一个 msw 进行逻辑或运算	1	2
SLM Ws, Wb, Wnd	Wnd = Wb 左移 Wb 位，然后与下一个 msw 进行逻辑或运算	1	2

表 2-6. 位操作指令

汇编语法	说明	字数	周期数
BCLR.b f,bit3	将 f 中的指定位清零	1	1
BCLR Ws,bit4	将 Ws 中的指定位清零	0.5/1	1
BFEXT bit4,wid5,Ws,Wb	从 Ws 中提取位域，写入 Wb	1	1
BFEXT bit4,wid5,f,Wb	从 f 中提取位域，写入 Wb	2	2
BFINS bit4,wid5,Wb,Ws	将 Wb 中的位域插入 Ws	1	1
BFINS bit4,wid5,Wb,f	将 Wb 中的位域插入 f	2	2
BFINS bit4,wid5,lit8,Ws	将 lit8 中的位域插入 Ws	2	2
BSET.b f,bit3	将 f 中的指定位置 1	1	1
BSET Ws,bit4	将 Ws 中的指定位置 1	0.5/1	1
BSW.C Ws,Wb	将 C 或 Z 位内容写入 Ws<Wb>	1	1
BSW.Z Ws,Wb	将 C 或 Z 位内容写入 Ws<Wb>	0.5/1	1
BTG.b f,bit3	将 f 中的指定位翻转	1	1
BTG Ws,bit4	将 Ws 中的指定位翻转	0.5/1	1
BTST.b f,bit3	测试 f 中的指定位	1	1
BTST.C Ws,bit4	测试 Ws 中的指定位，并将被测试位的值存储到进位标志位 C 中	0.5/1	1
BTST.Z Ws,bit4	测试 Ws 中的指定位，并将被测试位的反码存储到全零标志位 Z 中	1	1
BTST.C Ws,Wb	测试 Ws<Wb>中的指定位，并将被测试位的值存储到进位标志位 C 中	0.5/1	1
BTST.Z Ws,Wb	测试 Ws<Wb>中的指定位，并将被测试位的反码存储到全零标志位 Z 中	1	1
BTSTS.b f,bit3	测试 f 中的指定位，然后将 f 中的该位置 1	1	1
BTSTS.C Ws,bit4	测试 Ws 中的指定位，并将被测试位的值存储到进位标志位 C 中，然后将 Ws 中的该位置 1	0.5/1	1
BTSTS.Z Ws,bit4	测试 Ws 中的指定位，并将被测试位的补码存储到全零标志位 Z 中，然后将 Ws 中的该位置 1	1	1
FBCL Ws,Wnd	搜索自左（MSb）起第一个位变化	1	1
FF1L Ws,Wnd	搜索自左（MSb）起第一个 1	1	1
FF1R Ws,Wnd	搜索自右（LSb）起第一个 1	1	1

表 2-7. 比较/跳过和比较/转移指令

汇编语法	说明	字数	周期数
CP f,Ws	比较 f 和 Ws	1	1
CP Ws,lit13	比较 Ws 和 lit13（立即数零扩展）	1	1
CP Wb,lit16	比较 Wb 和 lit16（立即数零扩展）	1	1
CP Wb, Ws	比较 Wb 和 Ws	0.5/1	1
CP0 f	比较 f 和 0x0000	1	1
CP0 Ws	比较 Ws 和 0x0000（替换 CPLS Ws，#0）	1	1
CPB f,Ws	带借位比较 f 和 Ws	1	1
CP Wb,lit13	带借位比较 Wb 和 lit13（立即数零扩展）	1	1
CP Wb,lit16	带借位比较 Wb 和 lit16（立即数零扩展）	1	1
CPB Wb,Ws	带借位比较 Wb 和 Ws	0.5/1	1
DTB Wn,Label	递减 Wn，如果非零则转移	1	1（2/3）

表 2-8. 程序流指令

汇编语法	说明	字数	周期数
BRA Label	无条件转移	1	1
BRA Wn	计算转移	1	2
BRA C,Label	如果进位位为 1 则转移	1	1（2/3）
BRA GE,Label	如果大于或等于则转移	1	1（2/3）
BRA GEU,Label	如果无符号大于或等于则转移	1	1（2/3）
BRA GT,Label	如果大于则转移	1	1（2/3）
BRA GTU,Label	如果无符号大于则转移	1	1（2/3）
BRA LE,Label	如果小于或等于则转移	1	1（2/3）
BRA LEU,Label	如果无符号小于或等于则转移	1	1（2/3）
BRA LT,Label	如果小于则转移	1	1（2/3）
BRA LTU,Label	如果无符号小于则转移	1	1（2/3）
BRA N,Label	如果为负则转移	1	1（2/3）
BRA NC,Label	如果进位位为零则转移	1	1（2/3）
BRA NN,Label	如果非负则转移	1	1（2/3）
BRA NOV,Label	如果未溢出则转移	1	1（2/3）
BRA NZ,Label	如果非零则转移	1	1（2/3）
BRA Z,Label	如果为零则转移	1	1（2/3）
BREAK	停止用户代码执行	0.5/1	1
CALL Label	调用子程序（标号 > ~16 MB）	1	1
	（标号 < ~16 MB）	2	2
CALL Wns	间接调用子程序（地址[W11]处）	1	2
GOTO Label	跳转到地址（地址 < ~16 MB）	1	1
	（地址 > ~16 MB）	2	2
GOTO Wn	间接跳转到地址[W11]	1	2
RCALL Label	相对调用	1	1
RCALL Wns	计算调用	1	2
REPEAT lit15	重复执行下一条指令 lit15+1 次	1	1
REPEAT lit5	重复执行下一条指令 lit5+1 次	0.5	1
REPEAT Wn	重复执行下一条指令(Wn)+1 次	1	1
RETFIE	从中断返回	0.5	4
RETLW lit16,Wn	从子程序返回并将立即数存入 Wn	1	3
RETURN	从子程序返回	0.5	3

表 2-9. 影子/堆栈/现场指令

汇编语法	说明	字数	周期数
BOOTSWP	交换活动和非活动地址空间	0.5	2
CTXTSWP lit3	CPU 寄存器现场切换至 lit3 定义的现场	0.5	2
CTXTSWP Wn	CPU 寄存器现场切换至 Wn[2:0]定义的现场	1	2
LNK lit16	分配堆栈帧	1	1
LNK lit7	分配堆栈帧（立即数 < 128）	0.5	1
POP f	将栈顶（Top Of Stack，TOS）的内容弹出到 f	1	1
POP {[--Ws],} Wnd	将 Wnd 寄存器内容从系统堆栈弹出。	0.5	1

.....（续）			
汇编语法	**说明**	**字数**	**周期数**
POP Fd	将 Fd 寄存器内容从系统堆栈弹出。	0.5	1
PUSH f	将 f 内容压入栈顶（TOS）	1	1
PUSH Wns, {[Wd++]}	将 Wns 寄存器内容压入系统堆栈	0.5	1
PUSH Fs	将 Fs 寄存器内容压入系统堆栈	0.5	1
ULNK	释放堆栈帧	0.5	1

表 2-10. 控制指令

汇编语法	说明	字数	周期数
CLRWDT	清零看门狗定时器	0.5	1
DISICTL lit3 {,Wd}	在 IPL <= lit3 时禁止中断，可选择将先前的 IPL 阈值保存到 Wd	1	1
DISICTL Wns {,Wd}	在 IPL <= Wns[2:0]时禁止中断，可选择将先前的 IPL 阈值保存到 Wd	1	1
NEOP	无执行时间的 NOP 指令（16 位指令填充）	0.5	0
NOP	空操作	1	1
NOPR	空操作	1	1
PWRSAV mode	进入待机模式	0.5	2
RESET	软件器件复位	1	1

表 2-11. DSP 类指令

汇编语法	说明	字数	周期数
ADD A	累加器相加	0.5	1
ADD Rso,Slit6, A	16 位有符号数加到累加器	1	1
BRA OA,Label	如果累加器 A 溢出则转移	1	1（2/3）
BRA OB,Label	如果累加器 B 溢出则转移	1	1（2/3）
BRA OV,Label	如果溢出则转移	1	1（2/3）
BRA SA,Label	如果累加器 A 饱和则转移	1	1（2/3）
BRA SB,Label	如果累加器 B 饱和则转移	1	1（2/3）
CLR A	清零累加器	0.5	1
ED Wxp * Wyp, A, AWB	求取欧几里德距离	1	2
EDAC Wxp * Wyp, A, AWB	求取欧几里德距离（累加）	1	2
LAC Rso,Slit6, A	装载累加器内容（16/32 位），按照立即数移位	1	1
LLAC.l Rso Slit6, A	装载累加器内容（32 位）的低位（LS 字），按照立即数移位	1	1
LUAC.l Rso, Slit6, A	装载累加器内容（32 位）的高位（LS 字节），按照立即数移位	1	1
MAC Wxp * Wyp, A, AWB	相乘并累加	1	1
MAX Wb, Ws	强制执行数据范围上限	1	1
MAX A	强制执行数据范围上限	0.5	1
MAX.V A, Rdo	强制执行数据范围上限，保存超限结果	1	2
MIN Wb, Ws	强制执行数据范围下限	1	1
MIN A	强制执行数据范围下限	0.5	1

.....（续）

汇编语法	说明	字数	周期数
MIN.V A, Rdo	强制执行数据范围下限，保存超限结果	1	2
MULISS Wb,Ws,A	整数：Acc(A 或 B) = 有符号(Wb) * 有符号(Ws)	1	1
MULFSS Wb,Ws,A	小数：Acc(A 或 B) = 有符号(Wb) * 有符号(Ws)	1	1
MULISU Wb,Ws,A	整数：Acc(A 或 B) = 有符号(Wb) * 无符号(Ws)	1	1
MULFSU Wb,Ws,A	小数：Acc(A 或 B) = 有符号(Wb) * 无符号(Ws)	1	1
MULIUS Wb,Ws,A	整数：Acc(A 或 B) = 无符号(Wb) * 有符号(Ws)	1	1
MULFUS Wb,Ws,A	小数：Acc(A 或 B) = 无符号(Wb) * 有符号(Ws)	1	1
MULIUU Wb,Ws,A	整数：Acc(A 或 B) = 无符号(Wb) * 无符号(Ws)	1	1
MULFUU Wb,Ws,A	小数：Acc(A 或 B) = 无符号(Wb) * 无符号(Ws)	1	1
MULISS Wb,slit8,A	整数：Acc(A 或 B) = 有符号(Wb) * 有符号(slit8)	1	1
MULFSS Wb,slit8,A	整数：Acc(A 或 B) = 有符号(Wb) * 有符号(slit8)	1	1
MULISU Wb,lit8,A	整数：Acc(A 或 B) = 有符号(Wb) * 无符号(lit8)	1	1
MULFSU Wb,lit8,A	整数：Acc(A 或 B) = 有符号(Wb) * 无符号(lit8)	1	1
MULIUS Wb,slit8,A	整数：Acc(A 或 B) = 有符号(Wb) * 有符号(slit8)	1	1
MULFUS Wb,slit8,A	整数：Acc(A 或 B) = 有符号(Wb) * 有符号(slit8)	1	1
MULIUU Wb,lit8,A	整数：Acc(A 或 B) = 有符号(Wb) * 无符号(lit8)	1	1
MULFUU Wb,lit8,A	整数：Acc(A 或 B) = 有符号(Wb) * 无符号(lit8)	1	1
MPY Wxp * Wyp, A, AWB	Wm 与 Wn 相乘，结果存入累加器	1	1
MPYN Wxp * Wyp, A, AWB	Wm 与 Wn 相乘并取反，结果存入累加器	1	1
MSC Wxp * Wyp, A, AWB	相乘并从累加器中减去乘积	1	1
NEG A	对累加器内容求补	0.5	1
NORM A, Rdo	将累加器内容归一化	1	1
SAC A,Slit6,Rdo	保存累加器内容（16/32 位）	1	1
SACR A,Slit6,Rdo	保存舍入后的累加器内容（16/32 位），按照立即数移位	1	1
SACRW A,Ws,Rdo	保存舍入后的累加器内容（16/32 位），按照 Wb 移位	1	1
SLAC.l A,Slit6,Rdo	保存累加器内容（32 位）的低位（LS 字），按照立即数移位	1	1
SUAC.l A,Slit6,Rdo	保存累加器内容（32 位）的符号扩展高位（MS 字节），按照立即数移位	1	1

.....（续）			
汇编语法	**说明**	**字数**	**周期数**
SFTAC A,Wn	算术移位累加器内容(Wn)位	1	1
SFTAC A,Slit7	算术移位累加器内容 Slit7 位	1	1
SQR Wxp, A, AWB	求平方，结果存入累加器	1	1
SQRAC Wxp, A, AWB	平方并累加	1	1
SQRN Wxp, A, AWB	平方并取反，结果存入累加器	1	1
SQRSC Wxp, A, AWB	求平方并从累加器中减去	1	1
SUB A	累加器相减	0.5	1
SUB Rso,Slit6, A	从累加器中减去 16 位有符号数	1	1

表 2-12. FPU 指令

汇编语法	**说明**	**字数**	**周期数**
ABS Fs, Fd	求 Fs 的绝对值	1	1
ADD Fb, Fs, Fd	Fd = Fb + Fs	1	2
AND lit16, FSR	FSR = FSR AND lit16	1	1
AND lit16, FCR	FCR = FCR AND lit16	1	1
AND lit16, FEAR	FEAR = FEAR AND lit16	1	1
COS Fs , Fd	Fd = COS(Fs)	1	4
CPQ Fb, Fs	比较 Fb 和 Fs，不发信号	1	1
CPS Fb, Fs	比较 Fb 和 Fs，发信号	1	1
DI2F Fs, Fd	将双字（64 位）整数转换为浮点数，Fs（整数）--> Fd（浮点数）	1	2
DIV Fb, Fs, Fd	有符号浮点除法，Fd = Fb/Fs	1	11/32
FBRA EQ,Label	如果等于则浮点转移	1	1（2/3）
FBRA NE,Label	如果不等于则浮点转移	1	1（2/3）
FBRA GT,Label	如果大于则浮点转移	1	1（2/3）
FBRA GE,Label	如果大于或等于则浮点转移	1	1（2/3）
FBRA LT,Label	如果小于则浮点转移	1	1（2/3）
FBRA LE,Label	如果小于或等于则浮点转移	1	1（2/3）
FBRA OR,Label	如果有序则浮点转移	1	1（2/3）
FBRA UNE,Label	如果无序或不等于转移	1	1（2/3）
FBRA UEQ,Label	如果无序或等于则浮点转移	1	1（2/3）
FBRA ULE,Label	如果无序、小于或等于则浮点转移	1	1（2/3）
FBRA ULT,Label	如果无序或小于则浮点转移	1	1（2/3）
FBRA UGE,Label	如果无序、大于或等于则浮点转移	1	1（2/3）
FBRA UGT,Label	如果无序或大于则浮点转移	1	1（2/3）
FBRA UN,Label	如果无序则浮点转移	1	1（2/3）
FLIM Fb, Fs, Fd	强制执行有符号数据限制，如果 Fd > Fs 则 Fd = Fs，如果 Fd < Fb 则 Fd = Fb	1	1
F2DI Fs, Fd	将浮点数 Fs 转换为双字（64 位）整数，Fs（浮点数）--> Fd（整数）	1	1/2
F2LI Fs, Fd	将浮点数 Fs 转换为长字（32 位）整数，Fs（浮点数）--> Fd（整数）	1	1/2

.....（续）

汇编语法	说明	字数	周期数
IOR lit16, FSR	FSR 逻辑或，FSR = FSR .IOR. lit16	1	1
IOR lit16, FCR	FCR 逻辑或，FCR = FCR .IOR. Lit16	1	1
IOR lit16, FEAR	FEAR 逻辑或，FEAR = FEAR .IOR.Lit16	1	1
LI2F Fs, Fd	将长字（32 位）整数转换为浮点数，Fs（整数）--> Fd（浮点数）	1	1
MAC Fb, Fs, Fd	浮点数有符号相乘并累加，Fd = Fd +(Fb * Fs)	1	3/4
MAX Fb, Fs, Fd	选择 Fb 和 Fs 之中的有符号最大值{IEEE 754-2019 maxmum(x,y)}，如果 Fs >= Fb 则 Fd = Fs，否则 Fd = Fb	1	1
MAXNM Fb, Fs, Fd	选择 Fb 和 Fs 之中的有符号最大值{IEEE 754-2019 maxmumNumber(x,y)}，如果 Fs >= Fb 则 Fd = Fs，否则 Fd = Fb	1	1
MIN Fb, Fs, Fd	选择 Fb 和 Fs 之中的有符号最小值{IEEE 754-2019 minmum(x,y)}，如果 Fs =< Fb 则 Fd = Fs，否则 Fd = Fb	1	1
MINNM Fb, Fs, Fd	选择 Fb 和 Fs 之中的有符号最小值[minmumNumber()]，如果 Fs =< Fb 则 Fd = Fs，否则 Fd = Fb	1	1
MOV.l Fs, Rdo	将协处理器寄存器中内容传送至 Wd	0.5/1	1
MOV.l Rso, Fd	将 Ws 中内容传送至协处理器寄存器	0.5/1	1
MOV.l lit32,Fd	将 32 位无符号立即数传送至协处理器寄存器	2	2
MOV Fs,[Wnd+Slit12]	将 Fs 中内容传送至 [Wnd+Slit12]	1	1
MOV [Wns+Slit12],Fd	将[Wns+Slit12]中内容传送至 Fd	1	1
MOV Fs, Fd	将 Fs 中内容传送至 Fd	1	1
MOV index, Fd	Fd = 常量表（索引）Fd	1	1
MUL Fb, Fs, Fd	Fd = Fb * Fs	1	3
NEG Fs, Fd	Fd = -Fs	1	1
SIN Fs, Fd	Fd = SIN(Fs)	1	4
SQRT Fs, Fd	Fd = √Fs	1	10/13
SUB Fb, Fs, Fd	Fd= Fb- Fs	1	2
TST Fs	测试 Fs	1	1

3. 指令集详解

3.1 数据空间寻址模式

dsPIC33A 器件对数据存储器的访问，除几种立即数寻址形式外，还支持三种固有寻址模式。通过文件寄存器寻址、寄存器直接寻址或寄存器间接寻址模式实现对数据存储器的访问，而立即数寻址使得指令可以使用固定值作为操作数。

通过文件寄存器寻址可实现对存放在最大 64 KB 数据空间中的数据进行操作（如果需要 WREG 操作数），并且 MOV 指令提供对所有 1 MB 数据空间的访问。寄存器直接寻址模式用来对 16 个存储器映射的工作寄存器 W0:W15 进行访问。寄存器间接寻址通过使用工作寄存器的内容作为有效地址（Effective Address，EA），可对存储在整个 1 MB 数据空间中的数据进行高效操作。立即数寻址模式并不直接访问数据存储区，但可使用常量值作为指令操作数。表 3-1 对所有模式的寻址范围进行了汇总。

表 3-1. dsPIC33A 的寻址模式

寻址模式	寻址范围
文件寄存器寻址	0x0000-0xFFFF(1)
寄存器直接寻址	0x0000-0x001F（工作寄存器阵列 W0:W15）
寄存器间接寻址	0x0000-0xFFFFF
立即数寻址	N/A（常量值）

文件寄存器 MOV 的寻址范围是 0x0000-0xFFFE。

3.1.1 文件寄存器寻址

文件寄存器寻址模式由使用预先确定的数据地址作为操作数的指令使用。大多数支持文件寄存器寻址模式的指令都可对最大 64 KB 的数据空间进行访问（如果需要 WREG 操作数）。然而，MOV 指令可使用文件寄存器寻址模式对所有 1 MB 存储空间进行访问。这将允许把数据存储区中任何地址单元中的数据装载到任一工作寄存器，以及将任一工作寄存器的内容存放到数据存储区的任何地址单元中。应注意，文件寄存器寻址模式支持字节、扩展字节、字和长字四种数据大小。例 3-1 中给出了文件寄存器寻址的指令示例。

支持文件寄存器寻址的大多数指令以指定文件寄存器和默认工作寄存器 WREG 为操作对象。如果指令中只提供一个操作数，WREG 将是一个隐含的操作数，且操作结果将存回文件寄存器。此时，该指令实际上是一条读-修改-写指令。然而，当指令中同时指定文件寄存器和 WREG 寄存器时，操作结果将存放在 WREG 寄存器中，而文件寄存器中的内容将不发生改变。例 3-2 中给出了文件寄存器和 WREG 寄存器之间相互作用的指令示例。

注： 如指令集概述中的指令汇总表所示，支持文件寄存器寻址的指令使用 f 作为操作数。

例 3-1. 文件寄存器寻址

```
DEC        0x0000100       ; decrement data stored at 0x00001000
```

指令执行前：

```
Data Memory 0x00001000 = 0x55555555
```

指令执行后：

```
Data Memory 0x00001000 = 0x55555554
```

```
MOV    0x000027FE, W0     ; move data stored at 0x000027FE to W0
```

指令执行前：

```
W0 = 0x55555555
Data Memory 0x000027FE = 0x12345678
```

指令执行后：

```
W0 = 0x12345678
Data Memory 0x000027FE = 0x12345678
```

例 3-2. 文件寄存器寻址和 WREG

```
AND    0x00001000       ; AND 0x00001000 with WREG, store to 0x00001000
```

指令执行前：

```
W0 (WREG) = 0x0000332C
Data Memory 0x00001000 = 0x55555555
```

指令执行后：

```
W0 (WREG) = 0x0000332C
Data Memory 0x00001000 = 0x00001104
```

```
AND    0x00001000, WREG    ; AND 0x00001000 with WREG, store to WREG
```

指令执行前：

```
W0 (WREG) = 0x0000332C
Data Memory 0x00001000 = 0x55555555
```

指令执行后：

```
W0 (WREG) = 0x00001104
Data Memory 0x00001000 = 0x55555555
```

3.1.2 寄存器直接寻址

采用寄存器直接寻址模式对 16 个工作寄存器（W0:W15）的内容进行访问。寄存器直接寻址模式为全正交模式，允许为使用寄存器直接寻址的任何指令指定任何工作寄存器，且支持字节、字和长字访问方式。采用寄存器直接寻址的指令使用指定工作寄存器中的内容作为数据来执行指令，因此只有当数据已存入工作寄存器内时才可使用该寻址模式。例 3-3 中给出了使用寄存器直接寻址模式的指令示例。

寄存器直接寻址的另一特点是它可提供动态流控制能力。由于 REPEAT 指令的不同形式支持寄存器直接寻址，因此可使用这些指令实现灵活的循环结构。

注：对于那些必须使用寄存器直接寻址模式的指令，需要用到符号 Wb、Wn、Wns 和 Wnd，详见指令集概述的汇总表。通常，可使用寄存器间接寻址模式的场合也可使用寄存器直接寻址。使用寄存器间接寻址的指令使用指令集概述汇总表中的符号 Wd 和 Ws。

例 3-3. 寄存器直接寻址

```
EXCH        W2, W3            ; Exchange W2 and W3
```

指令执行前：

```
W2 = 0x00003499
W3 = 0x0000003D
```

指令执行后：

```
W2 = 0x0000003D
W3 = 0x00003499
```

```
IOR        #0x44, W0        ; Inclusive-OR 0x44 and W0
```

指令执行前：

```
W0 = 0x12349C2E
```

指令执行后：

```
W0 = 0x12349C6E
```

```
SL        W6, W7, W8       ; Shift left W6 by W7, and store to W8
```

指令执行前：

```
W6 = 0x0000000C
W7 = 0x00000008
W8 = 0x12345678
```

指令执行后：

```
W6 = 0x0000000C
W7 = 0x00000008
W8 = 0x00000C00
```

3.1.3 寄存器间接寻址

寄存器间接寻址模式通过将工作寄存器中内容作为数据存储区有效地址（EA），可实现对数据空间中任何存储单元的访问。实质上，工作寄存器中的内容变为一个指向指令要访问的数据存储单元的指针。

因为该寻址模式允许在进行数据访问之前或之后，通过对 EA 进行递增或递减来修改工作寄存器中的内容，因此可实现强大的功能。通过在访问操作执行的同一周期内对 EA 中的内容进行修改，寄存器间接寻址可对顺序存储在存储区中的数据进行高效处理。表 3-2 给出了 dsPIC33A 器件所支持的间接寻址模式。

表 3-2. 间接寻址模式

间接模式	语法	功能（字节指令）	功能（字指令）	说明
无修改	[Wn]	EA = [Wn]	EA = [Wn]	Wn 中的内容形成 EA。
执行前递增	[++Wn]	EA = [Wn + = 1]	EA = [Wn + = 2]	对 Wn 中内容进行递增以形成 EA。
执行前递减	[--Wn]	EA = [Wn - = 1]	EA = [Wn - = 2]	对 Wn 中内容进行递减以形成 EA。
执行后递增	[Wn++]	EA = [Wn]+ = 1	EA = [Wn]+ = 2	Wn 中内容形成 EA，然后 Wn 中内容进行递增。
执行后递减	[Wn--]	EA = [Wn] - = 1	EA = [Wn] - = 2	Wn 中内容形成 EA，然后 Wn 中内容进行递减。
寄存器偏移量	[Wn+Wb]	EA = [Wn + Wb]	EA = [Wn + Wb]	Wn 和 Wb 的和形成 EA。Wn 和 Wb 中的内容未被修改。

表 3-2 显示了四种寻址模式对指令中使用的 EA 进行修改，且允许对工作寄存器进行以下更新：执行后递增、执行后递减、执行前递增以及执行前递减。

表 3-2 也阐明了使用寄存器偏移量模式寻址的情况，该寻址模式对距离一个工作寄存器中的基准 EA 某个偏移量的数据进行寻址。该模式使用另外一个工作寄存器的内容，通过将两个指定的工作寄存器相加来形成 EA。注意，用来形成 EA 的工作寄存器中内容将不会被修改。 例 3-4 说明了如何使用寄存器偏移量间接寻址对数据存储区进行访问。

注：带偏移量的 MOV 指令提供了在间接寻址时可使用的立即数偏移量寻址功能。在这些指令中，通过将工作寄存器的内容加到一个有符号立即数以形成 EA。 例 3-5 显示了如何利用这些指令实现从/向工作寄存器阵列传送数据。

例 3-4. 带有效地址更新的间接寻址

```
MOV.B        [W0++], [W13--]                ; byte move [W0] to [W13]
; post-inc W0, post-dec W13
```

指令执行前：

```
W0 = 0x2300
W13 = 0x2708
Data Memory 0x2300 = 0x7783
Data Memory 0x2708 = 0x904E
```

指令执行后：

```
W0 = 0x2301
W13 = 0x2707
Data Memory 0x2300 = 0x7783
Data Memory 0x2708 = 0x9083
```

```
ADD        W1, [--W5], [++W8]    ; pre-dec W5, pre-inc W8
; add W1 to [W5], store in [W8]
```

指令执行前：

```
W1 = 0x0800
W5 = 0x2200
W8 = 0x2400
Data Memory 0x21FE = 0x7783
Data Memory 0x2402 = 0xAACC
```

指令执行后：

```
W1 = 0x0800
W5 = 0x21FE
W8 = 0x2402
Data Memory 0x21FE = 0x7783
Data Memory 0x2402 = 0x7F83
```

例 3-5. 寄存器偏移量间接寻址

```
MOV.B        [W0+W1], [W7++]    ; byte move
;[W0+W1] to W7, post-inc W7
```

指令执行前：

```
W0 = 0x2300
W1 = 0x01FE
W7 = 0x1000
Data Memory 0x24FE = 0x7783
Data Memory 0x1000 = 0x11DC
```

指令执行后：

```
W0 = 0x2300
W1 = 0x01FE
W7 = 0x1001
Data Memory 0x24FE = 0x7783
Data Memory 0x1000 = 0x1183
```

```
LAC        [W0+W8], A       ; load ACCA with [W0+W8]
                                        ; (sign-extend and zero-backfill)
```

指令执行前：

```
W0 = 0x2344
W8 = 0x0008
ACCA = 0x00 7877 9321
Data Memory 0x234C = 0xE290
```

指令执行后：

```
W0 = 0x2344
W8 = 0x0008
ACCA = 0xFF E290 0000
Data Memory 0x234C = 0xE290
```

例 3-6. 用立即数偏移量进行传送的指令

```
MOV        [W0+0x20], W1         ; move [W0+0x20] to W1
```

指令执行前：

```
W0 = 0x1200
W1 = 0x01FE
Data Memory 0x1220 = 0xFD27
```

指令执行后：

```
W0 = 0x1200
W1 = 0xFD27
Data Memory 0x1220 = 0xFD27
```

```
MOV        W4, [W8-0x300]        ; move W4 to [W8-0x300]
```

指令执行前：

```
W4 = 0x3411
W8 = 0x2944
Data Memory 0x2644 = 0xCB98
```

指令执行后：

```
W4 = 0x3411
W8 = 0x2944
Data Memory 0x2644 = 0x3411
```

3.1.3.1 寄存器间接寻址和指令集

表 4-2 中给出的寻址模式展示了 dsPIC33A 器件的间接寻址模式能力。出于操作编码和功能上的考虑，并非所有支持间接寻址的指令都支持表 4-2 中所示的所有模式。使用间接寻址模式的大多数指令支持无修改、执行前递增、执行前递减、执行后递增和执行后递减寻址模式。MOV 指令和几种基于累加器的 DSP 类指令也具备使用寄存器偏移量寻址模式的能力。

注：使用寄存器间接寻址的指令使用指令集概述汇总表中的操作数符号 Wd 和 Ws。

3.1.3.2 DSP MAC 间接寻址模式

DSP MAC 类指令采用一类特殊的间接寻址模式。如后面 DSP MAC 类指令中所介绍，DSP MAC 类指令能够使用有效寻址两次从存储区取操作数。由于 DSP 算法经常需要更宽范围的地址更新，DSP MAC 类指令提供的寻址模式可实现更宽的有效地址更新大小范围。 表 3-3 显示 X 和 Y 存储区的预取都支持执行后递增和执行后递减寻址模式，且更新方式可为 2、4 和 6 字节。由于 DSP 类指令只以字模式的方式执行，EA 将不会以奇数长度方式进行更新。

表 3-3. DSP MAC 间接寻址模式

寻址模式	X 存储区	Y 存储区
无修改的间接寻址	EA = [Wx]	EA = [Wy]
执行后递增 2 的间接寻址	EA = [Wx] + = 2	EA = [Wy] + = 2
执行后递增 4 的间接寻址	EA = [Wx] + = 4	EA = [Wy] + = 4
执行后递增 6 的间接寻址	EA = [Wx] + = 6	EA = [Wy] + = 6
执行后递减 2 的间接寻址	EA = [Wx] – = 2	EA = [Wy] – = 2
执行后递减 4 的间接寻址	EA = [Wx] – = 4	EA = [Wy] – = 4
执行后递减 6 的间接寻址	EA = [Wx] – = 6	EA = [Wy] – = 6
寄存器偏移量间接寻址	EA = [W9 + W12]	EA = [W11 + W12]

3.1.3.3 模寻址和位反转寻址模式

dsPIC33A 架构支持两种通常用来实现 DSP 算法的特殊寄存器间接寻址模式。模寻址（又称循环寻址）提供了一种自动支持 X 和/或 Y 存储区中循环数据缓冲区的方法。模缓冲区使得软件不再需要进行地址边界检查，这将改善某些算法的性能。类似地，位反转寻址可以实现以一种非线性方式对缓冲区中的单元进行访问。这种寻址模式简化了用于基为 2 的 FFT 算法的数据重新排序，并大大减少了 FFT 的处理时间。

这两种寻址模式体现了 dsPIC33A 架构的强大功能。使用间接寻址的任何指令都可利用这两种寻址模式。

3.1.4 立即数寻址

在立即数寻址模式中，指令编码包含一个指令使用的预先定义常量操作数。该寻址模式可以独立使用，但通常是与文件寄存器、直接和间接寻址模式联合使用。根据指令类型的不同，可以使用长度不同的立即数操作数。常量的长度可以是 1 位（#lit1）、4 位（#bit4、#lit4 和#Slit4）、5 位（#lit5）、6 位（#Slit6）、8 位（#lit8）、10 位（#lit10 和#Slit10）、14 位（#lit14）以及 16 位（#lit16）。常量可以是有符号或无符号的，符号#Slit4、#Slit6 和#Slit10 指定为有符号常量，而所有其他立即数常量都是无符号的。 表 3-4 给出了每种立即数操作数在指令集中的使用。

表 3-4. 指令集中的立即数操作数

操作数	使用的指令
#lit1	PWRSAV
#lit3	CTXTSWP
#bit4	BCLR, BSET, BTG, BTST, BTST.C, BTST.Z, BTSTS, BTSTS.C, BTSTS.Z
#lit4	ASR, LSR, SL
#Slit4	ADD, LAC, SAC, SAC.R
#wid4	BFEXT, BFINS
#lit5	ADD, ADDC, AND, CP, CPB, IOR, MUL.SU, MUL.UU, SUB, SUBB, SUBBR, SUBR, XOR
#Slit7	SFTAC
#lit8	MOV.B, CP, CPB
#lit10	ADD, ADDC, AND, CP, CPB, IOR, RETLW, SUB, SUBB, XOR
#Slit10	MOV
#lit14	DISI, LNK, REPEAT

.....（续）

操作数	使用的指令
#lit15	REPEAT
#lit16	MOV

立即数寻址的语法规则要求必须将数值符号（#）放在常量操作数之前且紧靠常量操作数。“#”符号向汇编器表明该量为常量。如果指令中使用的常量超出范围，汇编器将产生一个错误。例 3-7 中给出了几个立即数寻址的示例。

例 3-7. 立即数寻址

```
PWRSAV        #1                ; Enter IDLE mode
ADD.B        #0x10, W0                  ; Add 0x10 to W0 (byte mode)
```

指令执行前：

```
W0 = 0x12A9
```

指令执行后：

```
W0 = 0x12B9
XOR        W0, #1, [W1++]                ; Exclusive-OR W0 and 0x1
; Store the result to [W1]
; Post-increment W1
```

指令执行前：

```
W0 = 0xFFFF
W1 = 0x0890
Data Memory 0x0890 = 0x0032
```

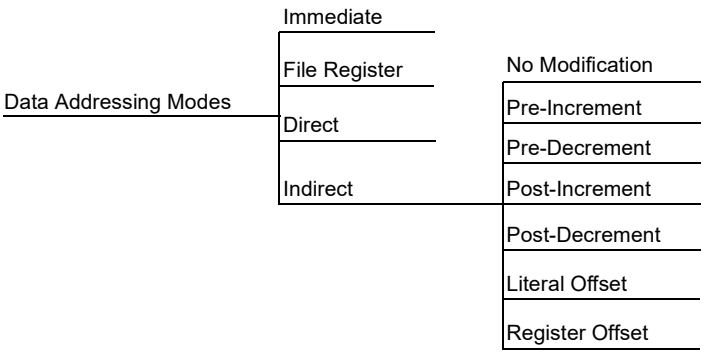
指令执行后：

```
W0 = 0xFFFF
W1 = 0x0892
Data Memory 0x0890 = 0xFFFE
```

3.2 数据空间寻址模式树

图 3-1 对 dsPIC33A 系列的数据空间寻址模式进行了汇总。

图 3-1. 数据空间寻址模式树



3.3 程序空间寻址模式

dsPIC33A 器件拥有一个 24 位程序计数器（PC）。PC 可对 24 位宽的程序存储空间进行寻址以取出将执行的指令。可通过几种不同方式实现对 PC 的装载。指令可以是 16 位、32 位或 64 位实体，因此在顺序执行 16 位、32 位或 64 位指令期间，PC 分别递增 2、4 或 8。

有几种不同的方法可实现以非顺序方式改变 PC 值，PC 值的改变可以是绝对或是相对的。PC 值的改变可能来自指令编码中的立即数或包含在工作寄存器中的动态值。对于异常处理，PC 将装载异常处理程序的入口地址，该地址存放在中断向量表（Interrupt Vector Table，IVT）中。在需要时，可用软件堆栈来保存现场，返回到程序流发生变化处的前台进程。

表 3-5 汇总了修改 PC 值的指令。当进行函数调用时，由于 RCALL 仅占用一个字的程序存储空间，因此建议使用 RCALL 而不使用 CALL 指令。

表 3-5. 改变程序流的方法

条件/指令	PC 改变	软件堆栈使用情况
顺序执行	PC = PC + 2	无
BRA Expr(1) （无条件转移）	PC = PC + 2 * Slit16	无
BRA Condition, Expr(1) （条件转移）	PC = PC + 2（条件为假） PC = PC + 2 * Slit16（条件为真）	无
CALL Expr(1) （调用子程序）	PC = lit23	PC + 4 压入堆栈(2)
CALL Wn （间接调用子程序）	PC = Wn	PC + 2 压入堆栈(2)
CALL.L Wn （间接长调用子程序）	PC = {Wn+1:Wn}	PC + 2 压入堆栈(2)
GOTO Expr(1) （无条件跳转）	PC = lit23	无
GOTO Wn （无条件间接跳转）	PC = Wn	无
GOTO.L Wn （无条件间接长跳转）	PC = {Wn+1:Wn}	无
RCALL Expr(1) （相对调用）	PC = PC + 2 * Slit16	PC + 2 压入堆栈(2)
RCALL Wn （计算相对调用）	PC = PC + 2 * Wn	PC + 2 压入堆栈(2)
异常处理	PC = 异常处理程序入口地址 （从向量表中读出）	PC + 2 压入堆栈(3)
PC = 目标 REPEAT 指令 （REPEAT 循环）	PC 未改变（如果 REPEAT 激活）	无

注：

1. 对于 BRA、CALL 和 GOTO 指令，Expr 可以是标号、绝对地址或表达式。Expr 将由链接器解析为一个 16 位或 23 位值（Slit16 或 lit23）。在表示地址偏移量值时，Expr 也可以搭配使用“.”与符号“+”或“-”来表示。例如，表达式“.+2”表示地址偏移量为+2（即，相对于程序计数器当前位置的下一条指令地址）。有关详细信息，请参见指令说明。
2. 在执行 CALL 或 RCALL 指令后，RETURN 或 RETLW 将把栈顶（TOS）内容弹回 PC。
3. 在异常处理结束后，RETFIE 将把栈顶（TOS）内容弹回 PC。

3.4 指令停顿

为尽可能缩短数据空间 EA 计算和取操作数的时间，部分 X 数据空间读和写访问采用流水线方式进行。这种流水线方式的一个可能后果是，在使用共用的寄存器进行连续的读写操作之间可能产生地址寄存器数据的相关性。

“写后读”（Read-After-Write，RAW）相关性问题发生在跨指令边界时，由硬件检测。RAW 相关性的一个示例是在修改 W5 的写操作结束后，使用 W5 作为地址指针的读操作。在前次写操作结束之前，W5 中的内容将不能作为读操作的有效数据。这个问题可通过对指令执行停顿一个指令周期来解决，这将允许在下一次读操作启动之前结束写操作。

3.4.1 RAW 相关性检测

在指令预译码期间，内核将确定地址寄存器相关性是否即将跨指令边界。停顿检测逻辑将用于当前执行指令的目标 EA 的 W 寄存器（如果有）和用于预取指令源 EA（如果有）的 W 寄存器进行比较。当发现目标寄存器和源寄存器相匹配时，将应用一组规则来判定是否将对指令执行停顿一个周期。 表 3-6 列出了导致指令执行停顿的各种 RAW 条件。

表 3-6. RAW 相关性规则（由硬件检测）

使用 Wn 的 目标寻址模式	使用 Wn 的 源寻址模式	需要停顿吗？	示例(2)（Wn = W2）
直接	直接	无停顿	`ADD.W W0, W1, W2` `MOV.W W2, W3`
间接	直接	无停顿	`ADD.W W0, W1, [W2]` `MOV.W W2, W3`
间接	间接	无停顿	`ADD.W W0, W1, [W2]` `MOV.W [W2], W3`
间接	带执行前/后修改的间接寻址	无停顿	`ADD.W W0, W1, [W2]` `MOV.W [W2++], W3`
带执行前/后修改的间接寻址	直接	无停顿	`ADD.W W0, W1, [W2++]` `MOV.W W2, W3`
直接	间接	停顿(1)	`ADD.W W0, W1, W2` `MOV.W [W2], W3`
直接	带执行前/后修改的间接寻址	停顿(1)	`ADD.W W0, W1, W2` `MOV.W [W2++], W3`
间接	间接	停顿(1)	`ADD.W W0, W1, [W2](2)` `MOV.W [W2], W3(2)`
间接	带执行前/后修改的间接寻址	停顿(1)	`ADD.W W0, W1, [W2](2)` `MOV.W [W2++], W3(2)`
带执行前/后修改的间接寻址	间接	停顿(1)	`ADD.W W0, W1, [W2++]` `MOV.W [W2], W3`
带执行前/后修改的间接寻址	带执行前/后修改的间接寻址	停顿(1)	`ADD.W W0, W1, [W2++]` `MOV.W [W2++], W3`
当检测到停顿时，指令执行时间将增加 1 个周期。 对于这些示例，W2 的内容 = W2 的映射地址（0x0004）。			

注：使用带偏移量的寄存器间接寻址模式指定指令的目标时，如果 `Ws` 与 `Wd` 为同一寄存器，则 `Ws` 的旧值将用于 `Wd`（即，忽略地址偏移量）。

3.4.2 指令停顿和异常

为保持确定的操作，将允许指令停顿发生，即使在停顿后紧接着异常处理。

3.4.3 指令停顿和改变程序流的指令

`CALL` 和 `RCALL` 使用 W15 写堆栈，因此如果下一条指令的读源操作数操作使用 W15 寄存器，则可能发生指令停顿。

`GOTO`、`RETFIE` 和 `RETURN` 指令的执行过程中将不会发生指令停顿的现象，因为它们不执行写入工作寄存器的操作。

3.4.4 指令停顿和 REPEAT 循环

如同其他任何指令，运行于 REPEAT 循环中的指令也易发生指令停顿的现象。停顿可能发生在循环入口、出口以及在循环处理过程中。

3.5 字节操作

由于数据存储区可字节寻址，大多数基本指令可运行在字节或字两种模式。当这些指令运行于字节模式时，以下规则将适用：

- 所有直接工作寄存器访问使用 32 位工作寄存器的最低有效字节（LSB），而最高有效字节（MSB）将不会被改动
- 所有间接工作寄存器访问使用由存放在工作寄存器中的 32 位地址指定的数据字节
- 所有文件寄存器访问使用由字节地址指定的数据字节
- 状态寄存器（SR）内容将被更新以反映字节操作的结果

应注意，数据地址总是以字节地址来表示。此外，固有的数据存放格式为小尾数法（little-endian），即将 LSB 存放在较低地址中，而将 MSB 存放在相邻的较高地址中（如图 3-2 所示）。 例 3-8 给出了字节传送操作的示例，而例 3-9 给出了字节数学操作的示例。

注：工作于字节模式的指令必须使用“.b”或“.B”指令扩展以指定字节指令。例如，以下两条指令为字节清零操作的有效形式：

```
CLR.b W0

CLR.B W0
```

例 3-8. 字节传送操作示例

```
MOV.B        #0x30, W0         ; move the literal byte 0x30 to W0
```

指令执行前：

```
W0 = 0x5555
```

指令执行后：

```
W0 = 0x5530
```

```
MOV.B        0x1000, W0        ; move the byte at 0x1000 to W0
```

指令执行前：

```
W0 = 0x5555
Data Memory 0x1000 = 0x1234
```

指令执行后：

```
W0 = 0x5534
Data Memory 0x1000 = 0x1234
```

```
MOV.B        W0, 0x1001    ; byte move W0 to address 0x1001
```

指令执行前：

```
W0 = 0x1234
Data Memory 0x1000 = 0x5555
```

指令执行后：

```
W0 = 0x1234
Data Memory 0x1000 = 0x3455
```

```
MOV.B        W0, [W1++]    ; byte move W0 to [W1], then post-inc W1
```

指令执行前：

```
W0 = 0x1234
W1 = 0x1001
Data Memory 0x1000 = 0x5555
```

指令执行后：

```
W0 = 0x1234
W1 = 0x1002
Data Memory 0x1000 = 0x3455
```

例 3-9. 字节数学操作示例

```
CLR.B        [W6--]        ; byte clear [W6], then post-dec W6
```

指令执行前：

```
W6 = 0x1001
Data Memory 0x1000 = 0x5555
```

指令执行后：

```
W6 = 0x1000
Data Memory 0x1000 = 0x0055
```

```
SUB.B        W0, #0x10, W1    ; byte subtract literal 0x10 from W0
; and store to W1
```

指令执行前：

```
W0 = 0x1234
W1 = 0xFFFF
```

指令执行后：

```
W0 = 0x1234
W1 = 0xFF24
```

```
ADD.B        W0, W1, [W2++]        ; byte add W0 and W1, store to [W2]
; and post-inc W2
```

指令执行前：

```
W0 = 0x1234
W1 = 0x5678
W2 = 0x1000
Data Memory 0x1000 = 0x5555
```

指令执行后：

```
W0 = 0x1234
W1 = 0x5678
```

```
W2 = 0x1001
Data Memory 0x1000 = 0x55AC
```

3.6 字传送操作

即使数据空间采用字节寻址模式，所有以字模式进行的传送操作也必须满足字对齐的原则。这表明对于所有源操作数和目标操作数，最低地址位必须是 0。如果发生字传送操作的一方是奇数地址，则将产生一个地址错误异常。类似地，所有双字传送操作也必须满足字对齐原则。 图 3-2 说明了如何实现字节和字在数据存储区的对齐。 例 3-10 给出了几种合法的字传送操作示例。

当由于未对齐访问导致异常产生时，将在指令执行后进行异常处理。如果在数据读操作期间发生非法访问，该操作将被允许完成，但是源地址的最低有效位（Least Significant bit，LSb）将被清零以强制字对齐。如果在数据写操作期间发生非法访问，该写操作将被禁止。 例 3-11 给出了几种非法的字传送操作示例。

图 3-2. 存储区中数据对齐方式

0x1001		**b0**	0x1000
0x1003	**b1**		0x1002
0x1005	**b3**	**b2**	0x1004
0x1007	**b5**	**b4**	0x1006
0x1009	**b7**	**b6**	0x1008
0x100B		**b8**	0x100A

Legend:
b0 – byte stored at 0x1000
b1 – byte stored at 0x1003
b3:b2 – word stored at 0x1005:1004 (b2 is LSB)
b7:b4 – double word stored at 0x1009:0x1006 (b4 is LSB)
b8 – byte stored at 0x100A

注：工作于字模式的指令不需要使用指令扩展。但如果需要可对其指定可选的“.w”或“.W”扩展。例如，以下指令为字清零操作的有效形式：

```
CLR W0

CLR.w W0

CLR.W W0
```

例 3-10. 合法的字传送操作

```
MOV        #0x30, W0         ; move the literal word 0x30 to W0
```

指令执行前：

```
W0 = 0x5555
```

指令执行后：

```
W0 = 0x0030
```

```
MOV        0x1000, W0        ; move the word at 0x1000 to W0
```

指令执行前：

```
W0 = 0x5555
Data Memory 0x1000 = 0x1234
```

指令执行后：

```
W0 = 0x1234
Data Memory 0x1000 = 0x1234
```

```
MOV        [W0], [W1++]        ; word move [W0] to [W1],
; then post-inc W1
```

指令执行前：

```
W0 = 0x1234
W1 = 0x1000
Data Memory 0x1000 = 0x5555
Data Memory 0x1234 = 0xAAAA
```

指令执行后：

```
W0 = 0x1234
W1 = 0x1002
Data Memory 0x1000 = 0xAAAA
Data Memory 0x1234 = 0xAAAA
```

例 3-11. 非法的字传送操作

```
MOV        0x1001, W0               ; move the word at 0x1001 to W0
```

指令执行前：

```
W0 = 0x5555
Data Memory 0x1000 = 0x1234
Data Memory 0x1002 = 0x5678
```

指令执行后：

```
W0 = 0x1234
Data Memory 0x1000 = 0x1234
Data Memory 0x1002 = 0x5678
```

产生地址出错陷阱

（源地址未对齐，因此执行 MOV）

```
MOV        W0, 0x1001               ; move W0 to the word at 0x1001
```

指令执行前：

```
W0 = 0x1234
Data Memory 0x1000 = 0x5555
Data Memory 0x1002 = 0x6666
```

指令执行后：

```
W0 = 0x1234
Data Memory 0x1000 = 0x5555
Data Memory 0x1002 = 0x6666
```

产生地址出错陷阱

（目标地址未对齐，因此不执行 MOV）

```
MOV        [W0], [W1++]               ; word move [W0] to [W1],
; then post-inc W1
```

指令执行前：

```
W0 = 0x1235
W1 = 0x1000
Data Memory 0x1000 = 0x1234
Data Memory 0x1234 = 0xAAAA
Data Memory 0x1236 = 0xBBBB
```

指令执行后：

```
W0 = 0x1235
W1 = 0x1002
Data Memory 0x1000 = 0xAAAA
Data Memory 0x1234 = 0xAAAA
Data Memory 0x1236 = 0xBBBB
```

产生地址出错陷阱

（源地址未对齐，因此执行 MOV）

3.7 使用 16 位立即数操作数

支持字节和字模式的几条指令具有 16 位操作数。对于字节指令，16 位立即数太大而不能使用。因此，在字节模式下使用 16 位立即数时，操作数的范围必须减至 8 位，否则汇编器将产生一个错误。 表 3-7 说明，在字模式下 16 位立即数的范围是 0:1023，而在字节模式下则为 0:255。

在字节和字模式下使用 16 位立即数的指令为 ADD、ADDC、AND、IOR、RETLW、SUB、SUBB 和 XOR。 例 3-12 以 ADD 指令为例说明了如何在字节模式下使用正或负的立即数。

表 3-7. 16 位立即数编码

立即数值	字模式 kk kkkk kkkk	字节模式 kkkk kkkk
0	0000 0000 0000 0000	0000 0000
1	0000 0000 0000 0001	0000 0001
2	0000 0000 0000 0010	0000 0010
127	0000 0000 0111 1111	0111 1111
128	0000 0000 1000 0000	1000 0000
255	0000 0000 1111 1111	1111 1111
256	0000 0001 0000 0000	N/A
512	0000 0010 0000 0000	N/A
1023	0000 0011 1111 1111	N/A
65535	1111 1111 1111 1111	N/A

注： 由于字节的 MSb 被置 1，在字节模式下使用大于 127 的立即数在功能上与使用负的二进制补码值等效。当工作于字节模式时，汇编器将接受正或负的立即数值（例如，#-10）。

例 3-12. 使用 16 位立即数作为字节操作数

```
ADD.B        #0x80, W0        ; add 128 (or -128) to W0
ADD.B        #0x380, W0             ; ERROR... Illegal syntax for byte mode
ADD.B        #0xFF, W0              ; add 255 (or -1) to W0
ADD.B        #0x3FF, W0             ; ERROR... Illegal syntax for byte mode
ADD.B        #0xF, W0               ; add 15 to W0
ADD.B        #0x7F, W0              ; add 127 to W0
ADD.B        #0x100, W0             ; ERROR... Illegal syntax for byte mode
```

3.8 位域插入/提取指令

dsPIC33A 提供了一组在目标字内操作位域的指令。

3.8.1 BFEXT

该指令可从 W 寄存器或数据存储单元提取多个位到目标 W 寄存器中。

3.8.2 BFINS

该指令可将源 W 寄存器或 8 位立即数值中的多个位插入 W 寄存器或数据存储单元。

在这两条指令中，目标字内位域的位置和宽度被定义为指令内的立即数值。

3.9 软件堆栈指针和帧指针

3.9.1 软件堆栈指针

dsPIC33A 器件具有一个软件堆栈，有助于函数调用以及异常处理。W15 为默认的堆栈指针（Stack Pointer，SP），在复位之后，它被初始化为 0x4000。这将确保 SP 将指向有效的 RAM 单元并允许异常处理可以使用堆栈，这可能发生在用户软件对 SP 进行设置之前。在初始化时，用户可重新编程 SP 为数据空间内的任何地址单元。

SP 总是指向栈顶第一个可供使用的字，并按照地址从低到高的原则对软件堆栈进行填充。SP 在弹出堆栈（读）前递减而在压入堆栈（写）后递增。

使用 PUSH 和 POP 指令对软件堆栈进行操作。PUSH 和 POP 指令等同于将 W15 用作目标指针的 MOV 指令。例如，可通过以下指令将 W0 中内容压入栈顶（TOS）：

```
    PUSH W0
```

该语法等同于：

```
MOV W0,[W15++]
```

通过以下指令可将 TOS 中内容返回 W0：

```
    POP W0
```

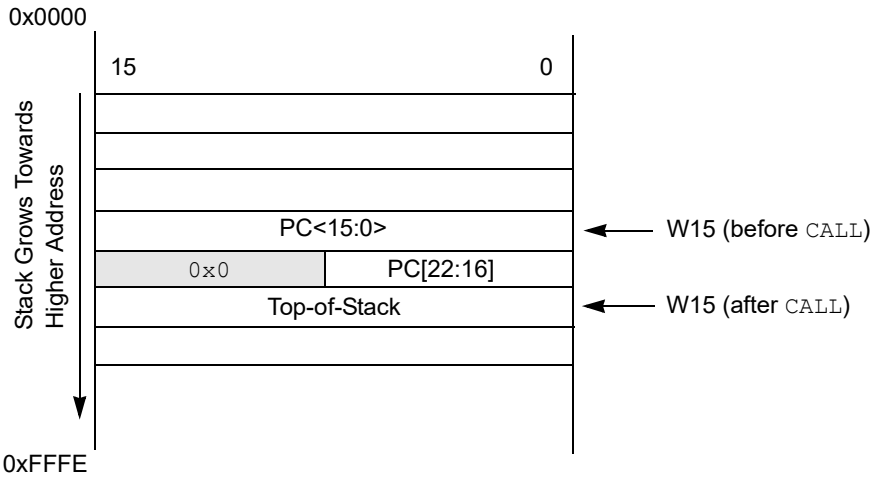
该语法等同于：

```
MOV [--W15],W0
```

在执行任何 CALL 指令过程中，PC 的内容将被压入堆栈，因此当执行完子程序时程序流将从正确地址继续执行。当将 PC 压入堆栈时，PC[15:0]被压入第一个可用的堆栈字，然后将 PC[22:16]压入堆栈。如图 3-3 所示，当 PC[22:16]被压入堆栈时，在压栈前 PC 的高 7 位将进行零扩展。 在异常处理过程中，PC 的高 7 位将与状态寄存器的低位字节（SRL）和 IPL[3]（CORCON[3]）组合起来，这将使得在中断过程中主要的状态寄存器内容和 CPU 中断优先级被自动保存。

注：为防止未对齐的堆栈访问操作发生，W15[0]总是清零。

图 3-3. CALL 指令执行时的堆栈操作



3.9.1.1 堆栈指针示例

图 3-4 至图 3-7 说明了例 3-13 的代码片段是如何修改软件堆栈的。 图 3-4 显示了第一条 PUSH 指令执行前软件堆栈的情况。注意，SP 已被初始化为 0x4000。此外，示例代码将 0x5A5A 和 0x3636 分别装入 W0 和 W1。在图 3-5 中第一次执行压入堆栈的操作，W0 中包含的值被复制到 TOS。W15 被自动更新以指向下一个可用的堆栈地址单元，且新的 TOS 为 0x4002。在图 3-6 中，W1 中内容被压入堆栈，此时新的 TOS 变为 0x4004。在图 3-7 中，堆栈被弹出，将最后一次压入堆栈的值（W1 内容）复制到 W3。在弹出操作过程中，SP 将递减，且在示例代码执行结束时 TOS 最终值为 0x4002。

图 3-4. 第一次执行 PUSH 指令前的堆栈指针

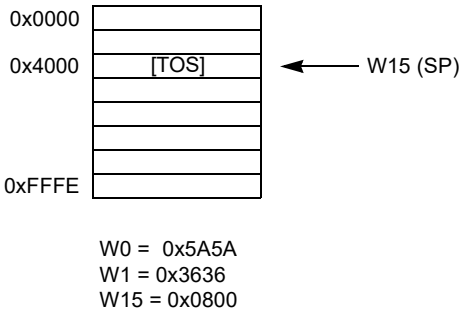


图 3-5. 执行“PUSH W0”指令后的堆栈指针

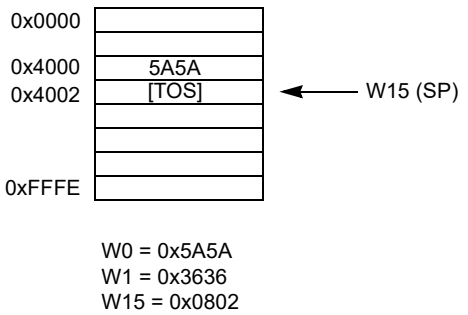


图 3-6. 执行“PUSH W1”指令后的堆栈指针

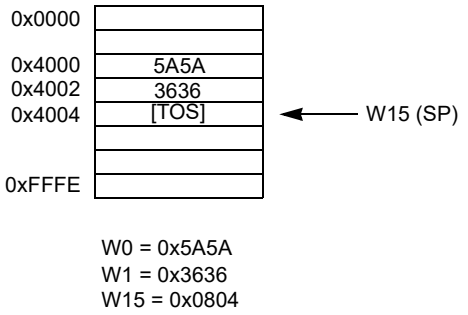
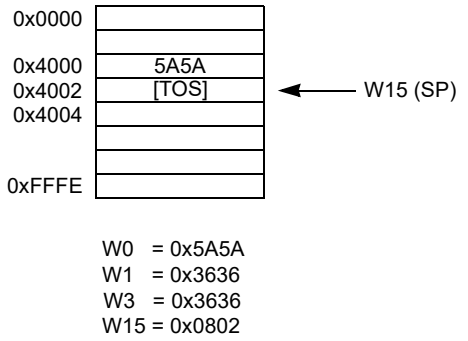


图 3-7. 执行“POP W3”指令后的堆栈指针



例 3-13. 堆栈指针的使用

```
MOV         #0x5A5A, W0          ; Load W0 with 0x5A5A
MOV         #0x3636, W1             ; Load W1 with 0x3636
PUSH         W0                  ; Push W0 to TOS
PUSH         W1                  ; Push W1 to TOS
POP         W3                 ; Pop TOS to W3
```

3.9.2 软件堆栈帧指针

堆栈帧是用户定义的驻留在软件堆栈中的存储区。它用来为函数使用的临时变量分配存储区，可为每个函数创建一个堆栈帧。W14 是默认的堆栈帧指针（Frame Pointer，FP），它在任何复位时被初始化为 0x0000。如果未使用堆栈帧指针，W14 可被当作普通的工作寄存器使用。

分配堆栈帧（LNK）和释放堆栈帧（ULNK）指令提供了堆栈帧功能。LNK 指令用来创建一个堆栈帧，在调用过程中用来调整 SP，这样堆栈可用来存放被调用函数使用的临时变量。在函数执行结束后，ULNK 指令用来移除 LNK 指令创建的堆栈帧。LNK 和 ULNK 指令必须总是一起使用以防止堆栈溢出。

3.9.2.1 堆栈帧指针示例

图 3-8 至图 3-10 说明了在例 3-14 的代码片段中是如何创建及移除堆栈帧的。该示例展示了堆栈帧是如何操作的而并不表示编译器生成的代码。 图 3-8 显示了程序开始时的堆栈状态，此时所有寄存器尚未被压入堆栈。这里，W15 指向第一个可用的堆栈地址单元（TOS），W14 指向分配给当前正在执行的子程序的堆栈存储区。

在调用子程序“COMPUTE”之前，子程序参数（W0、W1 和 W2）被压入堆栈。在“CALL COMPUTE”指令执行之后，PC 变为“COMPUTE”的地址，且子程序“TASKA”的返回地址被压入堆栈（图 3-9）。子程序“COMPUTE”随后使用“LNK #4”指令将调用子程序的帧指针值压入堆栈，且新的帧指针将被设置指向当前堆栈指针。随后，立即数 4 被加到 W15 中的堆栈指针地址，这将为临时数据保留两个字的存储区（图 3-10）。

在子程序“COMPUTE”中，FP 用来访问子程序参数以及临时（局部）变量。[W14 + n]将用来对子程序使用的临时变量进行访问，而[W14 – n]用来对参数进行访问。在子程序出口，ULNK 指令被用来将帧指针地址复制到堆栈指针，然后将调用子程序的帧指针弹回 W14 寄存器。ULNK 指令将使堆栈返回至图 3-9 中所示的状态。

RETURN 指令将使程序执行返回至调用子程序的代码。调用代码负责将参数从堆栈中移除。RETURN 和 POP 指令将堆栈恢复至图 3-8 中所示的状态。

图 3-8. 在帧指针用法示例开始时的堆栈情况

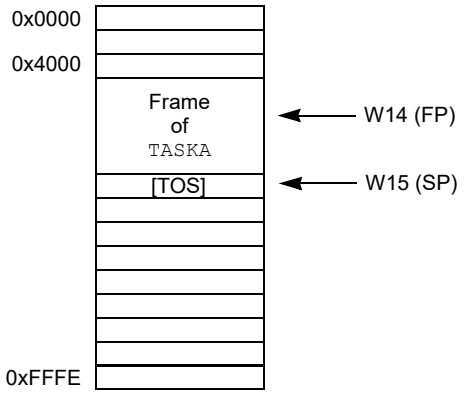


图 3-9. “CALL COMPUTE”指令执行后的堆栈

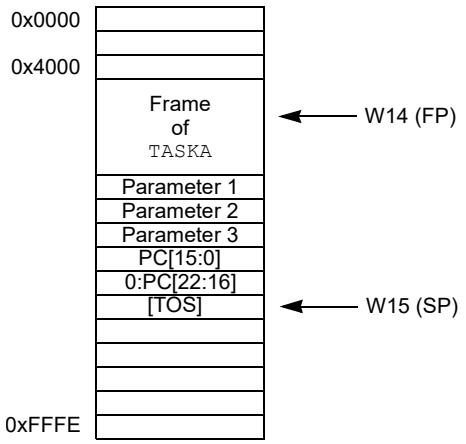
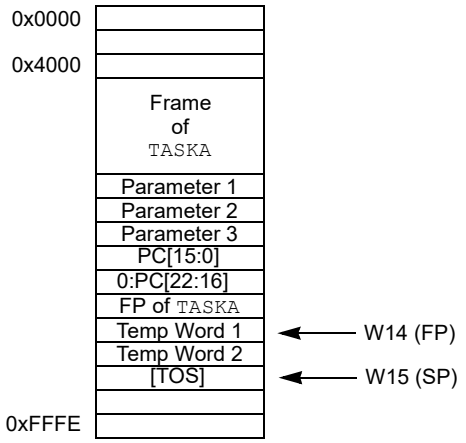


图 3-10. “LNK #4”指令执行后的堆栈



例 3-14. 帧指针用法

```
TASKA:
    ...
    PUSH        W0      ; Push parameter 1
    PUSH        W1      ; Push parameter 2
PUSH        W2            ; Push parameter 3
    CALL        COMPUTE   ; Call COMPUTE function
POP         W2          ; Pop parameter 3
POP         W1          ; Pop parameter 2
POP         W0          ; Pop parameter 1
...
COMPUTE:
LNK   #4                  ; Stack FP, allocate 4 bytes for local variables
...
ULNK                      ; Free allocated memory, restore original FP
RETURN                    ; Return to TASKA
```

3.9.3 堆栈指针上溢

SPLIM 是与堆栈指针相关联的堆栈限制寄存器，在复位时设置为 0x00000000。SPLIM 为 32 位寄存器，但 SPLIM[1:0]固定为 00，这是由于所有堆栈操作必须符合长字对齐的原则。为了匹配堆栈指针，器件地址空间被限制为 24 位，因此 SPLIM 的高 8 位总是为全 0。

在向 SPLIM 写入字之前，不使能堆栈上溢检查。在该功能使能之后，只有器件复位才能将其禁止。所有使用 W15 作为源寄存器或目标寄存器生成的有效地址将与 SPLIM 中的值进行比较。 如果有效地址大于 SPLIM 中内容，将产生堆栈出错陷阱。

如果已使能堆栈上溢检查，当 W15 有效地址计算越过了数据存储空间末端（0x00FFFFFF）时，也将产生堆栈出错陷阱。

3.9.4 堆栈指针下溢

在复位时，堆栈指针初始化为 0x4000。如果堆栈指针地址小于 0x4000，将会启动一个堆栈出错陷阱。

3.10 条件转移指令

条件转移指令用于根据状态寄存器的内容对程序流进行控制。这些指令通常与比较类指令联合使用，但可在任何改变状态寄存器内容的操作之后有效使用这些指令。

比较指令 CP、CP0 和 CPB 执行减法操作（被减数 - 减数），但实际上并不保存减法操作的结果。而是仅对状态寄存器中的标志位进行更新，因此随后的条件转移指令可通过对更新后的状态寄存器内容进行测试以对程序流进行改变。如果状态寄存器的结果测试为真，将进行转移。如果状态寄存器的结果测试为假，将不转移。

表 3-8 列出了支持的条件转移指令。该表指明了状态寄存器中何种条件为真将触发转移操作。在某些情况下，指令只对单个状态位进行测试（如在 BRA C 指令中），而在其他情况下，则会执行复杂的逻辑运算（如在 BRA GT 指令中）。通过 OA、OB、SA 和 SB 条件助记符，DSP 算法支持有符号和无符号的条件测试。

表 3-8. 条件转移指令

条件助记符(1)	说明	状态测试
C	进位（无借位）	C
GE	有符号大于或等于	($\overline{N}$&&$\overline{OV}$) || (N&&OV)
GEU(2)	无符号大于或等于	C
GT	有符号大于	($\overline{Z}$&&$\overline{N}$&&$\overline{OV}$) || (Z&&N&&OV)
GTU	无符号大于	C&&$\overline{Z}$
LE	有符号小于或等于	Z || ($\overline{N}$&&OV) || (N&&$\overline{OV}$)
LEU	无符号小于或等于	$\overline{C}$ || Z
LT	有符号小于	($\overline{N}$&&OV) || (N&&$\overline{OV}$)
LTU(3)	无符号小于	C
N	负	N
NC	无进位（有借位）	C
NN	非负	N
NOV	无溢出	OV
NZ	非零	Z
OA	累加器 A 溢出	OA
OB	累加器 B 溢出	OB
OV	溢出	OV
SA	累加器 A 饱和	SA
SB	累加器 B 饱和	SB
Z	零	Z

注：

1. 指令形式为：BRA 助记符, Expr。
2. GEU 等同于 C 且反汇编为 BRA C, Expr。
3. LTU 等同于 NC 且反汇编为 BRA NC, Expr。

注：比较和跳过指令（CPBEQ、CPBGT、CPBLT、CPBNE、CPSEQ、CPSGT、CPSLT 和 CPSNE）不改变状态寄存器内容。

3.10.1 浮点转移指令

在浮点 CPS/CPQ 指令将 FSR 排序关系状态位之一置 1 后，后续的浮点条件转移（FBRA）指令将（间接）检查这些状态位，以将其应用于表示所需条件的逻辑谓词。表 3-9 列出了支持的浮点转移和相应的谓词。

表 3-9. FPU 条件转移指令

条件助记符(1)	说明	状态测试
EQ	等于	FSR.EQ
UNE	无序或不等于	(FSR.GT || FSR.LT || FSR.UN)
NE	不等于	(FSR.GT || FSR.LT)
UEQ	无序或等于	(FSR.EQ || FSR.UN)
GT	大于	FSR.GT
ULE	无序、小于或等于	(FSR.LT || FSR.EQ || FSR.UN)
GE	大于或等于	(FSR.GT || FSR.EQ)

.....（续）

条件助记符(1)	说明	状态测试
ULT	无序或小于	(FSR.LT || FSR.UN)
LT	小于	FSR.LT
UGE	无序、大于或等于	(FSR.GT || FSR.EQ || FSR.UN)
LE	小于或等于	(FSR.LT || FSR.EQ)
UGT	无序或大于	(FSR.GT || FSR.UN)
OR	有序	(FSR.GT || FSR.LT || FSR.EQ)
UN	无序	FSR.UN

注：

1. 指令形式为：FBRA 助记符, Expr。

3.11 Z 状态位

Z 状态位是一个特殊的零状态位，有助于实现扩展精度的算术运算。除那些使用进位/借位输入的指令（ADDC、CPB、SUBB 和 SUBBR）外，在所有指令中 Z 位都用作普通的零标志位。对于 ADDC、CPB、SUBB 和 SUBBR 指令，Z 位只能被清零而不能被置 1。如果其中一条指令的结果是非零，无论后续 ADDC、CPB、SUBB 或 SUBBR 指令操作结果为何，Z 位将被清零并将保持该状态。这将允许使用 Z 位对一系列扩展精度运算的结果进行简便的零状态检查。

对多精度数据进行操作的一个指令序列（该序列的起始指令无进位/借位输入）将自动对连续的零状态测试结果进行逻辑与操作。只有所有结果必须为零，Z 标志才会在操作序列结束时保持置 1。如果 ADDC、CPB、SUBB 或 SUBBR 指令的执行结果为非零，Z 位将被清零且在所有后续的 ADDC、CPB、SUBB 或 SUBBR 指令执行过程中都将保持清零状态。

3.12 DSP 数据格式

3.12.1 整数和小数数据

DSP 引擎支持整数和小数数据类型。整型数据固有表示形式为有符号的二进制补码值，其中 MSb 定义为符号位。通常来说，N 位二进制补码整数的范围为-2^{N-1} 至 2^{N-1} - 1。对于 16 位整数，数据范围为-32768（0x8000）至 32767（0x7FFF），包括 0。对于 32 位整数，数据范围为-2,147,483,648（0x8000 0000）至 2,147,483,647（0x7FFF FFFF）。

小数数据表示为二进制补码数，其中 MSb 定义为符号位，小数点隐含于符号位之后。这种格式通常被称为 1.15（或 Q15）格式，其中 1 是用来表示数据的整数部分的位数，而 15 是用来表示小数部分的位数。采用这种隐含小数点的 N 位二进制补码表示的小数范围为-1.0 至（1 - 2^{1-N}）。对于 16 位小数，1.15 数据格式的范围为-1.0（0x8000）至 0.999969482（0x7FFF），包括 0.0 且精度为 3.05176×10^{-5}。在普通饱和模式下，32 位累加器使用 1.31 数据格式，可将精度提高至 4.6566×10^{-10}。在小数模式下，32x32 位 dsPIC 乘法器生成精度为 1.08420×10^{-19} 的 Q1.63 乘积。

通过使用累加器最高字节寄存器（ACCxU）的 8 个位作为警戒位，可以扩展累加器的动态范围。如果存放在累加器中的值发生溢出超出了 62 位的限制，此时将借助警戒位实现 DSP 算法。当 ACCSAT 位（CORCON[4]）置 1 时，将使能该模式并将累加器扩展为 72 位。当禁止累加器饱和时，也可使用警戒位。此时，累加器可支持的整数范围为 - 271（0x80 0000 0000 0000 0000）至 271-1（0x7F FFFF FFFF FFFF FFFF）。在小数模式下，累加器的警戒位不会改变小数点的位置且 72 位累加器使用 9.71 小数格式。注意，所有小数运算的结果都存入 72 位累加器，并采用 1.71 小数点格式进行对齐。在整数模式下，警戒位仅增加了累加器的动态范围。9.71 小数格式的范围为 - 256.0（0x80 0000 0000 0000 0000）至 256.0 - 1.08420x10-19（0x7F FFFF FFFF FFFF FFFF）。

表 3-10 指明了对于 16 位、32 位和 72 位寄存器，整数和小数的范围和精度。

除了 DSP 乘法操作之外，ALU 对于整数和小数数据的操作是相同的。也就是说，两个整数相加将产生与两个小数相加相同的结果（二进制数）。惟一的差别在于用户如何对该结果进行解释。然而，DSP 操作执行的乘法是不同的。在这些指令中，数据格式的选择是通过设置 IF 位（CORCON[0]）来进行的。因此必须对

其进行设置，将其设置为 0 选择小数模式，设置为 1 选择整数模式。因为小数使用隐含的小数点，因此需要进行上述设置。在整数模式下，两个 16 位整数相乘将产生一个 32 位整数结果。然而，两个 1.15 格式的值相乘将产生一个 2.30 格式的结果。由于 dsPIC33A 器件的累加器采用 1.31 数据格式，小数模式的 DSP 乘法执行过程也包括一个左移一位的操作以保持小数点能够正确对齐。该特点将使 DSP 乘法器的精度降低至 2^{-30}，但对于计算将不会产生其他影响（例如，0.5 x 0.5 = 0.25）。

表 3-10. dsPIC33A 的数据范围

寄存器宽度	整数范围	小数范围	小数精度
16 位	-32768 至 32767	-1.0 至（1.0 – 2^{-15}）	3.052 x 10^{-5}
32 位	-2,147,483,648 至 2,147,483,647	-1.0 至（1.0 – 2^{-31}）	4.657 x 10^{-10}
70 位	– 2^{71} 至 2^{71}	– 256.0 至 256.0 – 1.08420x10^{-19}	4.657 x 10^{-10}

3.12.2 整数和小数数据的表示

无论整数还是小数数据都将 MSb 作为符号位，且随着数据位的位置向 LSb 递进一位，其二进制指数将递减 1。N 位整数的二进制指数从（N-1）开始（MSb），最后为 0（LSb）。对于 N 位小数，二进制指数从 0 开始（MSb），最后为（1-N）（LSb）。图 3-11 和图 3-12 分别显示了正数和负数的表示方法。

整数和小数表示法之间的转换可通过使用简单的除法和乘法来实现。将整数值除以 2^{N-1} 可实现从 N 位整数至小数的转换。类似地，将小数值乘以 2^{N-1} 可实现从 N 位小数至整数的转换。

图 3-11. 0x4001 的不同表示

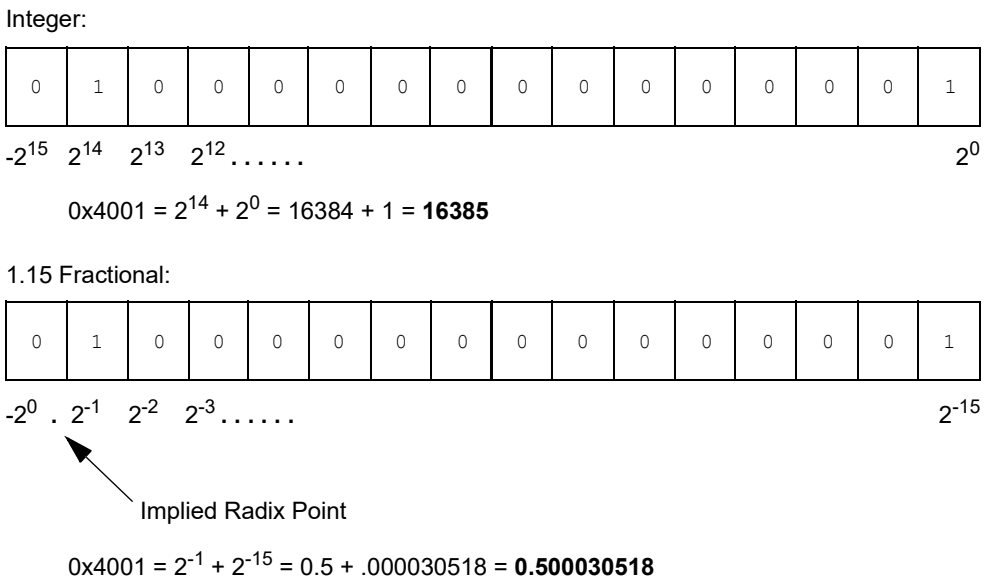
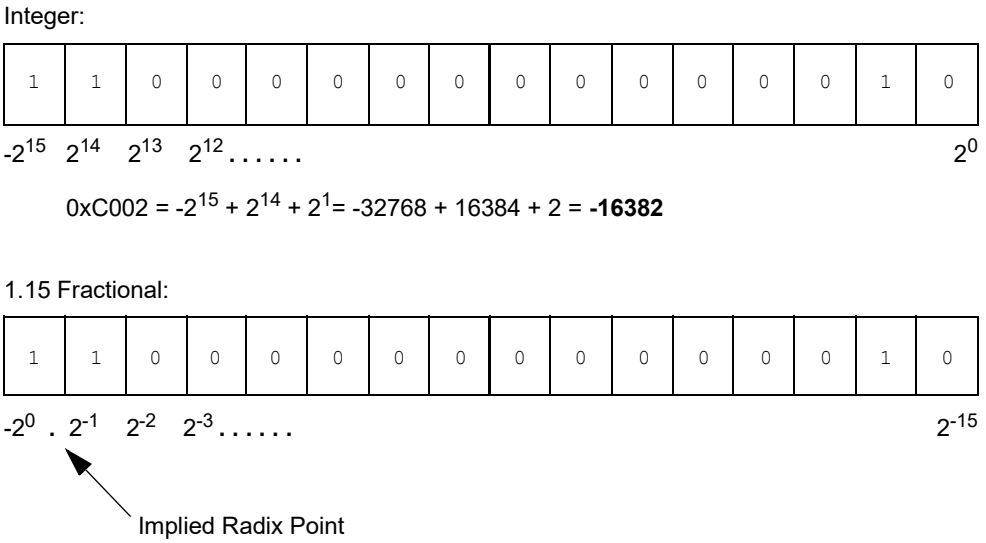


图 3-12. 0xC002 的不同表示



3.13 累加器的使用

DSP 类指令使用累加器 A 和 B 来执行数学和移位操作。由于累加器为 72 位宽而 X 和 Y 数据宽度只有 32 位，因此用户必须了解累加器执行装载和存储操作的方法。

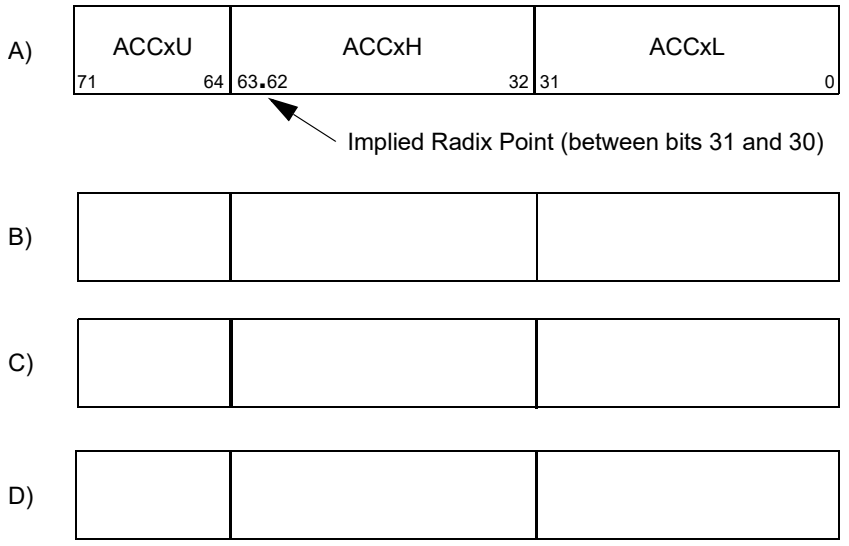
图 3-13 中 A 项显示每一个 72 位累加器（ACCA 和 ACCB）包含一个 8 位最高位寄存器（ACCxU），一个 32 位高位寄存器（ACCxH）和一个 32 位低位寄存器（ACCxL）。为满足总线对齐要求并提供实现 1.31 数学格式的能力，将 ACCxH 用作累加器装载操作的目标寄存器（使用 LAC 指令），以及累加器存储操作的源寄存器（使用 SAC.R 指令）。图 3-13 中的 B 项显示了该情形，其中累加器的最高字节和低位字部分采用阴影表示。实际上在累加器装载过程中，将对 ACCxL 进行零回填且对 ACCxU 进行符号扩展以表示装入 ACCxH 中值的符号。

注：dsPIC33A 器件提供了双字 `LAC.D` 和 `SAC.D` 指令，允许在一条指令中同时装载或存储 ACCxH 和 ACCxL。

当使能普通饱和模式（63 位）时，DSP 操作（如 `ADD`、`MAC` 和 `MSC` 等）仅利用 ACCxH:ACCxL（如图 3-13 中 C 项所示），而 ACCxU 仅用来保持存入 ACCxH:ACCxL 中值的符号。例如，当执行 `MPY` 指令时，结果将存入 ACCxH:ACCxL，而结果的符号通过 ACCxU 进行扩展。

当使能超饱和模式时，或禁止饱和时，可利用累加器中的所有寄存器（如图 3-13 中 D 项所示）且 DSP 操作结果将存入 ACCxU:ACCxH:ACCxL。如整数和小数数据中所介绍，ACCxU 的作用是使得累加器的动态范围得到扩展。有关普通和超饱和模式下累加器中存储值范围的介绍，请参见表 3-10。

图 3-13. 累加器的对齐和使用



A) 72-bit accumulator consists of ACCxU:ACCxH:ACCxL
B) Load and store operations
C) Operations in Normal Saturation mode
D) Operations in Super Saturation mode or with saturation disabled

3.14 累加器访问

累加器支持文件寄存器寻址和间接寻址模式，任何支持此两种寻址模式之一的指令均可对其进行访问。然而，建议使用 DSP 类指令 LAC、SAC 和 SAC.R 对累加器进行装载和存储操作，因为这些指令具备符号扩展、移位以及舍入能力。LAC、SAC 和 SAC.R 指令的细节将在指令说明给出。

注：

1. 为方便起见，将 ACCAU 和 ACCBU 符号扩展至 32 位。这为通过字节模式或字模式访问这些寄存器提供了灵活性（当使用文件寄存器寻址或间接寻址模式时）。
2. OA、OB、SA 或 SB 位不能通过使用 MOV 指令将溢出值写入存储器映射的累加器而置 1，因为这些状态位仅受 DSP 操作影响。
3. dsPIC33A 器件提供了双字 LAC.D 和 SAC.D 指令，允许在一条指令中同时装载或存储 ACCxH 和 ACCxL。

3.15 DSP MAC 类指令

DSP 乘-累加（MAC）操作是一组最高效利用 dsPIC33A 架构的特殊指令。表 3-11 中所示的 DSP MAC 指令使用 CPU 内核的 X 和 Y 数据线。

MAC 指令也能对一个累加器执行操作，并存储另一个累加器舍入后的内容。该功能称为累加器回写（Write Back，WB），可为软件开发人员提供较高的灵活性。例如，累加器回写可用来同时运行两个算法，或高效处理复数以及用于其他的用途。

表 3-11. DSP MAC 类指令

指令	说明
ED	求取欧几里德距离（无累加）
EDAC	求取欧几里德距离
MAC	相乘并累加
SQRAC	平方并累加

.....（续）	
指令	**说明**
MPY	相乘，结果存入累加器
MPY.N	相乘并取反，结果存入累加器
MSC	相乘并从累加器中减去乘积
SQRSC	求平方并从累加器中减去
SQR	求平方，结果存入累加器
SQRN	平方并取反，结果存入累加器

3.15.1 MAC 操作

DSP MAC 类指令执行的数学运算主要包括将两个工作寄存器中内容相乘或求平方，然后向/从累加器 A 或累加器 B 中加上或存入/减去结果。执行这些操作的指令包括 MAC、MPY、MPY.N、MSC、SQR、SQRAC、SQRSC 和 SQRN。

基于欧几里德距离运算的定义，对于 ED 和 EDAC 指令，指令必须指定相同的被乘数操作数。

3.15.2 MAC 回写

DSP MAC 类指令具有的回写能力有利于实现算法的高效处理。这个特点使得在同一个周期内，可使用一个累加器执行一个数学运算并将另一个累加器舍入后的内容进行存储。

支持以下寻址模式：

- W0、W1、W2、W3、W13 或 W14 寄存器直接寻址模式：将非目标累加器舍入后的内容以 1.15（字模式）或 1.31（长字模式）小数格式写入目标寄存器。与任何寄存器直接字写的情况一样，写入的值在写入目标 WREG 之前会经过零扩展至 32 位。
- [W13++]或[W15++]后递增的寄存器间接寻址模式：将非目标累加器舍入后的内容以 1.15（字模式）或 1.31（长字模式）小数格式写入由 W13 或 W15 指向的地址。目标 EA W 寄存器随后递增 2（对于字写）或 4（对于长字写）。

3.15.3 MAC 语法

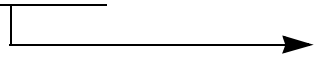
关于预取和累加器回写，根据指令类型和执行的操作，MAC 类指令的语法具有几种格式。所有 MAC 类指令必须指定目标累加器以及两个乘数，如例 3-15 所示。

如果指令中使用累加器回写，将在最后对其进行指定。根据定义，将对数学运算中未使用的累加器的内容进行存储，因此指令中不指定回写累加器。累加器回写（WB）的合法形式如例 3-16 所示。

例 3-15. 基本 MAC 语法

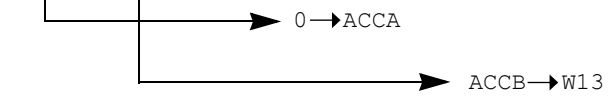
```
; MAC with no prefetch
MAC W4,W5, A


; MAC with no prefetch
MAC W7,W7, B
```

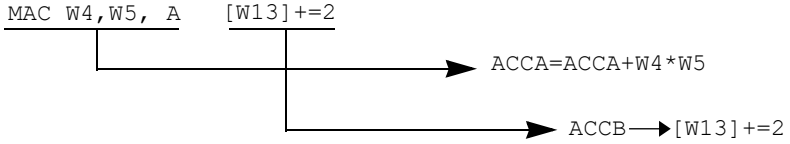


例 3-16. MAC 累加器回写语法

```
; CLR with direct WB of ACCB
CLR A,  W13
```




```
; MAC with indirect WB of ACCB
MAC W4,W5, A   [W13]+=2
```



3.16 DSP 累加器类指令

DSP 累加器指令提供了加法、求补、移位、装载以及将任一 72 位累加器内容进行存储的能力。此外，ADD 和 SUB 指令允许对两个累加器进行相加或相减。DSP 累加器类指令如表 3-12 所示，有关指令的详细介绍在指令说明给出。

表 3-12. DSP 累加器类指令

指令	说明	具有累加器回写功能？
ADD	累加器相加	无
ADD	16 位有符号数与累加器相加	无
LAC	装载累加器	无
LAC.D	装载累加器（双字）	无
NEG	对累加器内容求补	无
SAC	保存累加器内容	无
SAC.D	保存累加器内容（双字）	无
SAC.R	保存舍入后的累加器内容	无
SFTAC	对累加器进行算术移位，移位位数由立即数指定	无
SFTAC	对累加器进行算术移位，移位位数由(Wn)指定	无
SUB	累加器相减	无

3.17 使用 FBCL 指令换算数据

为尽可能缩小与使用 DSP 类指令进行的数据处理相关的量化误差，充分利用运算得到的全部数值结果很重要。这可能需要将数据按比例放大以避免出现下溢（例如，处理来自 12 位 ADC 的数据时），或将数据按比例缩小以避免出现上溢（例如，将数据发送至 10 位 DAC 时）。必须对数据进行换算处理以将量化误差最小化。换算比例的确定取决于被操作输入数据的动态范围以及所需的输出数据的动态范围。有时，这些条件是事先知晓的，因而可采用固定的换算系数。在其他一些场合，换算条件可能不是固定或已知的，这样就必须使用动态换算方法对数据进行处理。

FBCL 指令（检测从左开始首先发生变化的数据位）可用来高效实现动态换算，因为它可确定一个值的指数。一个定点或整数值的指数表征该值在出现溢出前可进行移位的位数。这个信息是很有用的，因为它可用来实现数据的“满量程”换算，即其换算后的数值表示可利用存放该数值的寄存器中的所有位。

FBCL 指令通过按照从值的符号位至 LSb 的顺序检测第一个发生变化的数据位来确定数据字的指数。由于 dsPIC DSC 器件的桶形移位寄存器使用负值指定左移操作，FBCL 指令返回值的指数的相反数。如果对值进

行按比例放大，这将允许立即执行随后的移位操作，而移位位数由 FBCL 指令的返回值确定。此外，由于 FBCL 指令只对有符号值进行操作，因此 FBCL 产生的结果处于-15:0（对于字操作）和-31:0（对于长字操作）范围内。当 FBCL 指令返回 0 时，表明该值已处于满量程。当指令返回-31 时，表明不能对该值进行换算（对于 0x0 以及 0xFFFFFFFF）。 表 3-13 显示不同动态范围的字数据，它们的指数以及数据经过换算实现动态范围最大化后的值。 例 3-17 说明了如何利用 FBCL 指令进行数据块处理。

表 3-13. 换算示例

字值	指数	满量程值 （字值 << 指数）
0x0001	14	0x4000
0x0002	13	0x4000
0x0004	12	0x4000
0x0100	6	0x4000
0x01FF	6	0x7FC0
0x0806	3	0x4030
0x2007	1	0x400E
0x4800	0	0x4800
0x7000	0	0x7000
0x8000	0	0x8000
0x900A	0	0x900A
0xE001	2	0x8004
0xFF07	7	0x8380

注： 对于字值 0x0000 和 0xFFFF，FBCL 指令返回-15。

作为一个实际的例子，假定以数据块处理的方式对一个数据序列进行处理。该数据系列以 1.15 小数格式存储且动态范围很小。为使量化误差最小，可对数据按比例放大以避免在对其进行处理时出现量化丢失。可通过对序列中具有最大幅值的样本数据执行 FBCL 指令以确定用于处理数据的最佳换算系数值。注意，通过将数据左移可使数据按比例增大。下面的代码片段说明了这一点。

例 3-17. 使用 FBCL 进行换算

```
; assume W0 contains the largest absolute value of the data block
; assume W4 points to the beginning of the data block
; assume the block of data contains BLOCK_SIZE words
; determine the exponent to use for scaling
FBCL        W0, W2                    ; store exponent in W2
; scale the entire data block before processing
DO        #(BLOCK_SIZE-1), SCALE
LAC         [W4], A                   ; move the next data sample to ACCA
SFTAC         A, W2                   ; shift ACCA by W2 bits
SCALE:
SAC         A, [W4++]                   ; store scaled input (overwrite
original)
; now process the data
; (processing block goes here)
```

3.18 数据范围限制指令

dsPIC33A 系列提供了特殊的指令，可自动将 W 寄存器或累加器中的数据限制在用户指定的数值范围内。这些指令包括 FLIM、MAX、MIN 和 MINZ。

3.18.1 FLIM/FLIM.V

FLIM 指令可将指定的 W 寄存器中内容（或 W 寄存器指向的数据）同时与数据上下限值进行比较，如果超出限值，则将目标数据限制为用户指定的限值。如果后续发生有符号转移，SR 状态位将相应地置 1。FLIM.V 指令中指定了一个额外的 W 寄存器，其中装载限制测试结果（称为“超限”）值。

3.18.2 MAX/MAX.V

MAX 指令可将目标累加器中内容与数据上限值（存储在 W 寄存器或其他累加器中）进行比较，如果超过该上限，则将目标累加器中内容限制为用户指定的限值。如果后续发生有符号转移，SR 状态位将相应地置 1。MAX.V 指令中指定了一个额外的 W 寄存器（或间接寻址模式下的 W 寄存器），其中装载超限值。

3.18.3 MIN/MIN.V/MINZ

MIN 指令可将目标累加器中内容与数据下限值（存储在 W 寄存器或其他累加器中）进行比较，如果数据小于该下限，则将目标累加器中内容限制为用户指定的限值。如果后续发生有符号转移，SR 状态位将相应地置 1。MIN.V 中指定了一个额外的 W 寄存器（或间接寻址模式下的 W 寄存器），其中装载超限值。MINZ 指令是 MIN 指令的有条件版本，仅当 Z = 1 时才执行。

3.19 使用 NORM 指令将累加器中内容归一化

NORM 指令可自动将目标累加器中内容归一化，以确保在未溢出的情况下获得尽可能大的小数值。目标累加器必须包含有符号小数数据，指令结果才有效。该指令将目标累加器中内容右移或左移所需的位数以将数据归一化，从而保持符号一致。移位值存储在目标地址中。N 状态位反映累加器移位的方向。

如果累加器无法归一化，则累加器内容不会受到影响。

4. 指令说明

4.1 指令符号

表 2 列出了指令说明中使用的所有符号。

4.2 指令编码字段描述符介绍

表 4-2 至表 4-10 显示了指令说明中使用的所有指令编码字段描述符。

表 4-1. 指令编码字段描述符

字段	说明
A	累加器选择位：0 = ACCA；1 = ACCB
aaa	累加器回写模式（见表 4-9）
B	字节模式选择位：0 = 字操作；1 = 字节操作
bbb	3 位位位置选择：000 = LSB；111 = MSB
bbbb	5 位位位置选择：00000 = LSB；11111 = MSB
cccc	位域指令 LSb 值
D	目标地址位：0 = 结果存入 W0；1 = 结果存入文件寄存器
dddd	Wd 目标寄存器选择：0000 = W0；1111 = W15
ddddd	协处理器目标寄存器选择（FPU 的 Fd，其中 00000 = F0；11111 = F31）
E	MULxx 指令结果大小：0 = 32 位（存入 Wnd）；1 = 64 位（存入(Wnd+1, Wnd)）
F	在 W15（F = 0）与 W14（F = 1）寄存器之间进行选择
(ffff) ffff ffff ffff ffff	16 位或 20 位寄存器文件地址（可寻址空间因指令类别而异）
G	位测试目标：0 = Z 标志位；1 = C 标志位
I	MULAxx 乘法模式：0 = 小数；1 = 整数
IIIii	X 数据取操作
JJJjj	Y 数据取操作
k	1 位立即数字段，常量数据
kkk	3 位立即数字段，常量数据
k kkkk	5 位立即数字段，常量数据
kk kkkk	6 位立即数字段，常量数据
kkkk kkkk	8 位立即数字段，常量数据
kkkk kkkk kkkk kkkk	16 位立即数字段，常量数据
kkkk kkkk kkkk kkkk kkkk kkkk kkkk kkkk	32 位立即数字段，常量数据
L	长字（32 位）模式选择位：0 = 字或字节操作；1 = 长字操作
mmmmm	位域指令 MSb 值
nnnn nnnn nnnn nnnn nnnn n	用于相对转移/调用指令的 21 位有符号指令字偏移量字段
nnnn nnnn nnnn nn00 nnnn nnnn	用于跳转/调用指令的 24 位程序地址
ppp	用于 Ws 源寄存器的寻址模式（见表 4-2）
qqq	用于 Wd 目标寄存器的寻址模式（见表 4-3）
R	在 FPU 协处理器特殊寄存器与 F-reg 之间进行选择
rrr	条件传送（MOVIF）指令的条件选择

.....（续）	
字段	**说明**
S	操作码大小选择（16 位：S = 0；32 位：S = 1）
ssss	Ws 源或 Wn 源/目标寄存器选择：0000 = W0；1111 = W15
sssss	协处理器源寄存器选择（FPU 的 Fs，其中 00000 = F0；11111 = F31）
T	在 Ws（T = 0）与 SR（T = 1）目标寄存器之间进行选择
U	未用（无关）指令位。汇编器赋值 0
V	FLIMW：选择 MULxx 指令中 Wnd 的结果格式（见指令说明）：在无符号 Ws（V = 0）与有符号 Ws（V = 1）之间进行选择。
W	目标写控制：0 = 不需要写入 Wd；1 = 需要写入 Wd
wwww	源 Wb 基址寄存器选择：0000 = W0；1111 = W15
zz	协处理器选择

表 4-2. 用于 Ws 源寄存器的寻址模式

ppp	寻址模式	源操作数
000	寄存器直接	Wns
001	间接	[Ws]
010	执行后递减的间接	[Ws--]
011	执行后递增的间接	[Ws++]
100	执行前递减的间接	[--Ws]
101	执行前递增的间接	[++Ws]
110	状态寄存器直接	SR（源）
111	寄存器偏移量间接	[Ws+Wb]

表 4-3. 用于 Wd 目标寄存器的寻址模式

qqq	寻址模式	目标操作数
000	寄存器直接	Wnd
001	间接	[Wd]
010	执行后递减的间接	[Wd--]
011	执行后递增的间接	[Wd++]
100	执行前递减的间接	[--Wd]
101	执行前递增的间接	[++Wd]
110	状态寄存器（SR）直接	SR（目标）
111	寄存器偏移量间接	[Wd+Wb]

表 4-4. 用于 MCU 乘法的目标寻址模式

dddd	目标
0000	W1:W0
0001	W0
0010	W3:W2
0011	W2
0100	W5:W4
0101	W4
0110	W7:W6
0111	W6
1000	W9:W8
1001	W8
1010	W11:W10
1011	W10

.....（续）	
dddd	**目标**
1100	W13:W12
1101	W12
1110	ACCA[39:0]
1111	ACCB[39:0]

表 4-5. 用于 Ws 源寄存器的偏移量寻址模式（带寄存器偏移量）

ggg	寻址模式	源操作数
000	寄存器直接	Ws
001	间接	[Ws]
010	执行后递减的间接	[Ws--]
011	执行后递增的间接	[Ws++]
100	执行前递减的间接	[--Ws]
101	执行前递增的间接	[++Ws]
11x	寄存器偏移量间接	[Ws+Wb]

表 4-6. 用于 Wd 目标寄存器的偏移量寻址模式（带寄存器偏移量）

hhh	寻址模式	源操作数
000	寄存器直接	Wd
001	间接	[Wd]
010	执行后递减的间接	[Wd--]
011	执行后递增的间接	[Wd++]
100	执行前递减的间接	[--Wd]
101	执行前递增的间接	[++Wd]
11x	寄存器偏移量间接	[Wd+Wb]

表 4-7. MAC 或 MPY 源操作数（相同工作寄存器）

mm	被乘数
00	W4 * W4
01	W5 * W5
10	W6 * W6
11	W7 * W7

表 4-8. MAC 或 MPY 源操作数（不同工作寄存器）

mmm	被乘数
000	W4 * W5
001	W4 * W6
010	W4 * W7
011	无效
100	W5 * W6
101	W5 * W7
110	W6 * W7
111	无效

表 4-9. MAC 累加器回写选择

aaa	回写选择
000	W0 = 其他累加器
001	W1 = 其他累加器
010	W2 = 其他累加器

.....（续）

aaa	回写选择
011	W3 = 其他累加器
100	W13 = 其他累加器（直接寻址）
101	[W13++] = 其他累加器（执行后递增的间接寻址）
110	[W15++] = 其他累加器
111	无累加器回写

表 4-10. 累加器选择

A	目标累加器
0	累加器 A
1	累加器 B

4.3 指令说明示例

以下示例说明是针对虚构指令 FOO 的，用来说明如何通过表字段（语法、操作数和操作等）对指令说明列出的指令进行说明。

FOO

FOO	指令标题字段对指令所执行的操作进行概括
语法：	语法字段包括可选的标号、指令助记符、任何可选的指令扩展以及指令的操作数。大多数指令都支持多个操作数的形式以支持不同的寻址模式。在这些情况下，所有可能的指令操作数都在下面列出并放在方括号内。
操作数：	操作数字段对每个操作数的取值集合进行说明。操作数可以是累加器、文件寄存器、立即数常量（有符号或无符号）或工作寄存器。
操作：	该字段概括指令执行的操作。
受影响的状态位：	该字段对指令执行将影响状态寄存器中哪一（些）位进行说明。状态位将以降序方式按照位位置顺序列出。
指令编码：	该字段说明了指令是如何进行位编码的。说明字段对各个位域进行了说明，完整的编码细节在表 5.2 中给出。
说明：	该字段详细说明了指令执行的操作。同时还给出了编码中关键位的信息。
指令字数：	该字段包含用于在存储器中存储指令的程序字数。
指令周期数：	该字段包含执行指令所需的指令周期数。
示例：	该字段包含说明指令如何操作的应用示例。其中给出了指令执行前和执行后寄存器的状态，这使得用户可清楚了解指令执行的操作。

4.4 指令说明（A 至 BZ）

ADD		Wb 与 Ws 相加			
语法：	{标号:}	ADD.b	Wb,	Ws,	Wd
		ADD.bz		[Ws],	[Wd]
		ADD{.w}		[Ws++],	[Wd++]
		ADD.l		[Ws--],	[Wd--]
				[++Ws],	[++Wd]
				[--Ws],	[--Wd]
操作数：	Wb ∈ [W0 ... W15]；Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]				
操作：	(Wb) + (Ws) → Wd				
受影响的状态位：	C、N、OV 和 Z				
指令编码：	S110 000L dddd ssss pppq qqww wwUU BU00				
说明：	将源寄存器 Ws 中的内容与基准寄存器 Wb 中的内容相加，并将结果存放在目标寄存器 Wd 中。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。				
指令字数：	1 或 0.5				
指令周期数：	1				

ADD		将 ACCA 加到 ACCB	
语法：	{标号:}	ADD	A B
操作数：	无		
操作：	ACCA + ACCB → ACC(A 或 B)		
受影响的状态位：	OA 和 SA，或者 OB 和 SB		
指令编码：	0111 001A UUUU 1000		
说明：	将 ACCA 加到 ACCB 并将结果写入所选的累加器。 A 位用于指定目标累加器。		
指令字数：	0.5		
指令周期数：	1		

ADD		有符号数加到累加器		
语法：	{标号:}	ADD{.w} Ws, ADD.l	{ Slit6, } [Ws], [Ws++] [Ws--] [--Ws], [++Ws], [Ws+Wb],	A B
操作数：	Ws ∈ [W0 ... W15]；Wb ∈ [W0 ...W15]；Slit6 ∈ [-32 ... +31]			
操作：	(ACC) + $Shift_{Slit6}$(符号扩展(Ws)) → ACC			
受影响的状态位：	OA 和 SA，或者 OB 和 SB			
指令编码：	1100 00AL wwww ssss pppU UUkk kkkk 1011			
说明：	假定有效地址处包含的操作数为 Q1.15 小数数据（对于字数据操作）或 Q1.31 小数数据（对于长字数据操作）。读取操作数后自动进行符号扩展和零回填，以创建与累加器大小相同的值。 将该值加到目标累加器之前，可选择进行（算术）移位。 L 位用于选择字或长字操作。 A 位用于指定目标累加器。 s 位用于指定源寄存器 Ws。 p 位用于选择源寻址模式。 w 位用于指定偏移量寄存器 Wb。 k 位用于对可选操作数 Slit6 进行编码，该操作数确定累加器内容预移位的位数；如果操作数 Slit6 不存在，则立即数位域设置为全 0。 **注：** 1. 操作数 Slit6 为正值表示算术右移。操作数 Slit6 为负值表示左移。 2. 该指令只能工作于长字或字模式。			
指令字数：	1			
指令周期数：	1			

ADDC		Wb 与 Ws 带进位位相加		
语法：	{标号:}	ADDC.b Wb, ADDC.bz ADDC{.w} ADDC.l	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	Wd [Wd] [Wd++] [Wd--] [++Wd] [--Wd]

.....（续）

ADDC	Wb 与 Ws 带进位位相加
操作数：	Wb ∈ [W0 ...W15]；Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]
操作：	(Wb) + (Ws) + (C) → Wd
受影响的状态位：	C、N、OV 和 Z
指令编码：	S110 001L dddd ssss pppq qqww wwUU BU00
说明：	将源寄存器 Ws 中的内容、基准寄存器 Wb 中的内容与进位位相加，并将结果存放在目标寄存器 Wd 中。 Z 位为“粘住”位（只能清零）。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 s 位用于选择源寄存器。 w 位用于选择基准寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。
指令字数：	1 或 0.5
指令周期数：	1

ADDC		Ws 与短立即数带进位位相加		
语法：	{标号:}	ADDC.b Ws,	lit7,	Wd
		ADDC.bz [Ws],		[Wd]
		ADDC.{w} [Ws++],		[Wd++]
		ADDC.l [Ws--],		[Wd--]
		[++Ws],		[++Wd]
		[--Ws],		[--Wd]
操作数：	Ws ∈ [W0 ...W15]；lit7 ∈ [0 ...127]；Wd ∈ [W0 ...W15]			
操作：	(Ws) + lit7 + (C) → Wd 注：立即数经过零扩展至操作所选的数据大小			
受影响的状态位：	C、N、OV 和 Z			
指令编码：	1110 001L dddd ssss pppq qqkk kkkk Bk10			
说明：	将源寄存器 Ws 中的内容、零扩展无符号立即数操作数与进位位相加，并将结果存放在目标寄存器 Wd 中。 Z 位为“粘住”位（只能清零）。 L 和 B 位用于选择操作数据宽度。 k 位用于提供有符号立即数操作数。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。 **注：**具有寄存器直接目标的字（.w）和零扩展字节（.bz）模式会将结果零扩展至 32 位。			
指令字数：	1			
指令周期数：	1			

ADDC		立即数带进位位加到 Wn	
语法：	{标号:}	ADDC.b lit16,	Wn
		ADDC.bz	
		ADDC{.w}	
		ADDC.l	

.....（续）

ADDC	立即数带进位位加到 Wn

操作数：lit16 ∈ [0 ... 65535]；Wn ∈ [W0 ...W15]

操作：(Wn) + lit16 + (C) → Wn
注：立即数经过零扩展至操作所选的数据大小

受影响的状态位：C、N、OV 和 Z

指令编码：

1100	001L	ssss	kkkk	kkkk	kkkk	kkkk	BU10

说明：将零扩展立即数操作数、工作寄存器 Wn 中的内容与进位位相加，并将结果存放在工作寄存器 Wn 中。
Z 位为“粘住”位（只能清零）。

L 和 B 位用于选择操作数据宽度。

s 位用于选择工作寄存器。

k 位用于指定有符号立即数操作数。

指令字数：1

指令周期数：1

ADDC	f 与 Wn 带进位位相加

语法：{标号:} ADDC.b f, Wn {,WREG}
ADDC.bz
ADDC{.w}
ADDC.l

操作数：f ∈ [0 ... 64 KB]；Wn ∈ [W0 ...W15]

操作：(f) + (Wn) + (C)→ 由 D 指定的目标寄存器

受影响的状态位：C、N、OV 和 Z

指令编码：

1110	001L	ssss	ffff	ffff	ffff	ffff	BD01

说明：将工作寄存器中的内容、文件寄存器中的内容与进位标志位相加，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wn 目标寄存器，则 D = 0 并将结果存放在 Wn 中；否则，D = 1 并将结果存放在文件寄存器中。
Z 位为“粘住”位（只能清零）。

L 和 B 位用于选择操作数据宽度。

D 位用于选择目标寄存器。

f 位用于选择文件寄存器的地址。

s 位用于选择工作寄存器。

指令字数：1

指令周期数：1

ADD	短立即数加到 Wn

语法：{标号:} ADD.l lit5, Wn

操作数：lit5 ∈ [0 ... 31]；Wn ∈ [W0 ...W15]

操作：(Wn) + lit5 → Wn
注：立即数经过零扩展至操作所选的数据大小

受影响的状态位：C、N、OV 和 Z

指令编码：

0111	010k	kkkk	ssss

说明：将零扩展立即数操作数与工作寄存器 Wn 中的内容相加，并将结果存放在工作寄存器 Wn 中。如果立即数 > 31 且/或需要字或字节操作，请汇编为 ADDLW 指令。
s 位用于选择工作寄存器。

k 位用于指定有符号立即数操作数。

.....（续）

ADD	短立即数加到 Wn
指令字数：	0.5
指令周期数：	1

ADD	Ws 与短立即数相加			
语法：	{标号:}	ADD.b	Ws,	lit7,	Wd
		ADD.bz	[Ws],		[Wd]
		ADD.{w}	[Ws++],		[Wd++]
		ADD.l	[Ws--],		[Wd--]
			[++Ws],		[++Wd]
			[--Ws],		[--Wd]
操作数：	Ws ∈ [W0 ...W15]；lit7 ∈ [0 ...127]；Wd ∈ [W0 ...W15]				
操作：	(Ws) + lit7 → Wd 注：立即数经过零扩展至操作所选的数据大小				
受影响的状态位：	C、N、OV 和 Z				
指令编码：	1110 000L dddd ssss pppq qqkk kkkk Bk10				
说明：	将源寄存器 Ws 中的内容与零扩展无符号立即数操作数相加，并将结果存放在目标寄存器 Wd 中。 L 和 B 位用于选择操作数据宽度。 k 位用于提供立即数操作数。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。				
指令字数：	1				
指令周期数：	1				

ADD	立即数加到 Wn		
语法：	{标号:}	ADD.b	lit16,	Wn
		ADD.bz		
		ADD{.w}		
		ADD.l		
操作数：	lit16 ∈ [0 ...65535]；Wn ∈ [W0 ...W15]			
操作：	(Wn) + lit16 → Wn 注：立即数经过零扩展至操作所选的数据大小			
受影响的状态位：	C、N、OV 和 Z			
指令编码：	1100 000L ssss kkkk kkkk kkkk kkkk BU10			
说明：	将零扩展立即数操作数与工作寄存器 Wn 中的内容相加，并将结果存放在工作寄存器 Wn 中。 L 和 B 位用于选择操作数据宽度。 s 位用于选择工作寄存器。 k 位用于指定有符号立即数操作数。			
指令字数：	1			
指令周期数：	1			

ADD	f 与 Wn 相加				
语法：	{标号:}	ADD.b	f	Wn	{,WREG}
		ADD.bz			
		ADD{.w}			

.....（续）

ADD	f 与 Wn 相加
	ADD.l

操作数：f ∈ [0 ...64 KB]；Wn ∈ [W0 ...W15]

操作：(f) + (Wn) → 由 D 指定的目标寄存器

受影响的状态位：C、N、OV 和 Z

指令编码：

1110	000L	ssss	ffff	ffff	ffff	ffff	BD01

说明：将工作寄存器中的内容与文件寄存器中的内容相加，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wn，则 D = 0 并将结果存放在 Wn 中；否则，D = 1 并将结果存放在文件寄存器中。
L 和 B 位用于选择操作数据宽度。

D 位用于选择目标寄存器。

f 位用于选择文件寄存器的地址。

s 位用于选择工作寄存器。

指令字数：1

指令周期数：1

AND	Wb 和 Ws 逻辑与

语法：

{标号:}	AND.b	Wb,	Ws,	Wd
	AND.bz		[Ws],	[Wd]
	AND{.w}		[Ws++],	[Wd++]
	AND.l		[Ws--],	[Wd--]
			[++Ws],	[++Wd]
			[--Ws],	[--Wd]
			SR	SR

操作数：Wb ∈ [W0 ...W14]；Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]

操作：(Wb).AND.(Ws) → Wd

受影响的状态位：N 和 Z

指令编码：

S110	100L	dddd	ssss	pppq	qqww	wwUU	BU00

说明：将源寄存器 Ws 中的内容和基准寄存器 Wb 中的内容进行逻辑与运算，并将结果存放在目标寄存器 Wd 中。
S 位用于选择指令大小。

L 和 B 位用于选择操作数据宽度。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

关于修改量寻址信息，请参见 和 。

注：

1. 选择 SR 时，SR 数据写操作将优先于由 AND 运算引起的任何 SR 更新操作。
2. 写入 SR 时，不允许使用.bz 数据大小/模式。

指令字数：1 或 0.5

指令周期数：1

AND	Ws 和零扩展短立即数逻辑与

语法：

{标号:}	AND.b	Ws,	lit7,	Wd
	AND.bz [Ws],			[Wd]
	AND{.w} [Ws++],			[Wd++]

.....（续）

AND	Ws 和零扩展短立即数逻辑与		
	AND.l	[Ws--],	[Wd--]
		[++Ws],	[++Wd]
		[--Ws],	[--Wd]
		SR	SR

操作数：Ws ∈ [W0 ...W15]；lit7 ∈ [0 ...127]；Wd ∈ [W0 ...W15]

操作：(Ws).AND.lit7[3] → Wd

受影响的状态位：N 和 Z（见注 1）

指令编码：

1110	100L	dddd	ssss	pppq	qqkk	kkkk	Bk10

说明：将源寄存器 Ws 中的内容和零扩展立即数操作数进行逻辑与运算，并将结果存放在目标寄存器 Wd 中。

L 和 B 位用于选择操作数据宽度。

k 位用于提供无符号立即数操作数。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

注：

1. 选择 SR 时，SR 数据写操作将优先于由 AND 运算引起的任何 SR 更新操作。
2. 写入 SR 时，不允许使用.bz 数据大小/模式。
3. 立即数经过零扩展至操作所选的数据大小。

指令字数：1

指令周期数：1

AND1	Ws 和一扩展短立即数逻辑与				
语法：	{标号:}	AND1.b	Ws,	lit7,	Wd
		AND1.bz [Ws],			[Wd]
		AND1{.w} [Ws++],			[Wd++]
		AND1.l	[Ws--],		[Wd--]
			[++Ws],		[++Wd]
			[--Ws],		[--Wd]
			SR		SR

操作数：Ws ∈ [W0 ...W15]；lit7 ∈ [0 ...127]；Wd ∈ [W0 ...W15]

操作：(Ws).AND.lit7[3] → Wd

受影响的状态位：N 和 Z（见注 1）

指令编码：

1110	110L	dddd	ssss	pppq	qqkk	kkkk	Bk10

.....（续）

AND1	Ws 和一扩展短立即数逻辑与

说明：

将源寄存器 Ws 中的内容和一扩展立即数操作数进行逻辑与运算，并将结果存放在目标寄存器 Wd 中。

L 和 B 位用于选择操作数据宽度。

k 位用于提供无符号立即数操作数。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

注：

1. 选择 SR 时，SR 数据写操作将优先于由 AND 运算引起的任何 SR 更新操作。
2. 写入 SR 时，不允许使用.bz 数据大小/模式。
3. 立即数经过一扩展至操作所选的数据大小。

指令字数：1

指令周期数：1

AND	立即数和 Wn 逻辑与

语法：

{标号:}	AND.b	lit16,	Wn
	AND.bz		SR
	AND{.w}		
	AND.l		

操作数：lit16 ∈ [0 ...65536]；Wn ∈ [W0 ...W15]；状态寄存器（SR）

操作：(Wn) .AND. lit16[3] → Wn 或(SR) .AND. lit16 → SR

受影响的状态位：N 和 Z（见注 1）

指令编码：

1100	100L	ssss	kkkk	kkkk	kkkk	kkkk	BT10

说明：

将立即数操作数和工作寄存器 Wn 或 SR 中的内容进行逻辑与运算，并将结果存放在工作寄存器 Wn 或 SR 中。

L 和 B 位用于选择操作数据宽度。

s 位用于选择工作寄存器。

k 位用于指定无符号立即数操作数。

T 位用于在 Ws（T = 0）与 SR（T = 1）目标寄存器之间进行选择。

注：

1. 选择 SR 时，SR 数据写操作将优先于由 AND 运算引起的任何 SR 更新操作。
2. 写入 SR 时，不允许使用.bz 数据大小/模式。
3. 对于长字操作，立即数经过零扩展至 32 位。

指令字数：1

指令周期数：1

AND	f 和 Wn 逻辑与

语法：

{标号:}	AND.b	f	,Wn	{,WREG}
	AND.bz			
	AND{.w}			
	AND.l			

操作数：f ∈ [0 ...64 KB]；Wn ∈ [W0 ...W14]

操作：(f).AND.(Wn) → 由 D 指定的目标寄存器

受影响的状态位：N 和 Z

.....（续）

AND		f 和 Wn 逻辑与						
指令编码：	1110	100L	ssss	ffff	ffff	ffff	ffff	BD01

说明：

将工作寄存器中的内容和文件寄存器中的内容进行逻辑与运算，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wn，则 D = 0 并将结果存放在 Wn 中；否则，D = 1 并将结果存放在文件寄存器中。

L 和 B 位用于选择操作数据宽度。

D 位用于选择目标寄存器。

f 位用于选择文件寄存器的地址。

s 位用于选择工作寄存器。

指令字数：1

指令周期数：1

ASR		算术右移 1 位		
语法：	{标号:}	ASR.b	Ws,	Wd
		ASR.bz	[Ws],	[Wd]
		ASR{.w}	[Ws++],	[Wd++]
		ASR.l	[Ws--],	[Wd--]
			[++Ws],	[++Wd]
			[--Ws],	[--Wd]

操作数：Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]

操作：

对于长字操作：

(Ws[31]) → Wd[31]，(Ws[31:1]) → Wd[30:0]，(Ws[0]) → C

对于字操作：

(Ws[15]) → Wd[15]，(Ws[15:1]) → Wd[14:0]，(Ws[0]) → C

对于字节操作：

(Ws[7]) → Wd[7]，(Ws[7:1]) → Wd[6:0]，(Ws[0]) → C

受影响的状态位：C、N 和 Z

指令编码：	S010	100L	dddd	ssss	pppq	qqUU	UUUU	B000

说明：

将源寄存器中的内容算术右移一位，并将结果存放在目标寄存器 Wd 中。

目标寄存器直接寻址扩展字节或字模式会将结果零扩展至 32 位，然后写入 Wd。

S 位用于选择指令大小。

L 和 B 位用于选择操作数据宽度。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

指令字数：1 或 0.5

指令周期数：1

ASR		算术右移 f			
语法：	{标号:}	ASR.b	f	{,Wnd}	{,WREG}
		ASR.bz			
		ASR{.w}			
		ASR.l			

操作数：f ∈ [0 ...64 KB]；Wnd ∈ [W0 ...W14]

.....（续）

ASR	算术右移 f
操作：	对于长字操作： (f[31]) → Dest[31]，(f[31]) → Dest[30], (f[30:1]) → Dest[29:0]，(f[0]) → C 对于字操作： (f[15]) → Dest[15]，(f[15]) → Dest[14] (f[14:1]) → Dest[13:0]，(f[0]) → C 对于字节操作： (f[7]) → Dest[7:1]，(f[7]) → Dest[6], (f[6:1]) → Dest[5:0]，(f[0]) → C
受影响的状态位：	C、N 和 Z
指令编码：	1010 000L dddd ffff ffff ffff ffff BD01
说明：	带进位标志位将文件寄存器 f 中的内容右移一位，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wnd，则 D = 0 并将结果存放在 Wnd 中；否则，D = 1 并将结果存放在文件寄存器中。 L 和 B 位用于选择操作数据宽度。 D 位用于选择目标寄存器。 f 位用于选择文件寄存器的地址。 d 位用于选择工作寄存器。
指令字数：	1
指令周期数：	1

ASR	算术右移（移位位数由短立即数确定）				
语法：	{标号:}	ASR{.w}	Ws,	lit5,	Wd
		ASR.l	[Ws],		[Wd]
			[Ws++],		[Wd++]
			[Ws--],		[Wd--]
			[++Ws],		[++Wd]
			[--Ws],		[--Wd]
操作数：	Ws ∈ [W0 ...W15]；lit5 ∈ [0...31]；Wd ∈ [W0 ...W15]				
操作：	lit5[4:0]→ Shift_Val 对于长字操作： Ws[31] → 右移输入（算术移位） Ws[31:0]和 32'b0 → Shift_In[63:0] 将 Shift_In[63:0]算术右移 Shift_Val 位 → Shift_Out[63:0] Shift_Out[63:32] → Wd 对于字操作： Ws[15] → 右移输入（算术移位） {16{Ws[15]}}、Ws[15:0]和 32'b0 → Shift_In[63:0] 将 Shift_In[63:0]算术右移 Shift_Val 位 → Shift_Out[63:0] Shift_Out[47:32] → Wd[15:0]				
受影响的状态位：	N 和 Z				
指令编码：	S010 000k kkkk dddd pppq qqss ssUU LU00				

.....（续）

ASR	算术右移（移位位数由短立即数确定）
说明：	将源寄存器 Ws 中的内容算术右移 lit5 位（最大移位位数为 31 位），并将结果存放在目标寄存器 Wd 中。 无论 lit5 中的移位值为何，该指令都会为字操作生成正确的结果。 如果 lit5 > 15 且 Ws[15] = 0，则 Wd[15:0] = 0x0000。 如果 lit5 > 15 且 Ws[15] = 1，则 Wd[15:0] = 0xFFFF。 寄存器直接寻址字模式会将结果零扩展至 32 位，然后写入 Wd。 S 位用于选择指令大小。 L 位用于选择字或长字操作。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。 k 位用于提供立即数操作数。 **注：** 1. 该指令只能工作于字或长字模式。
指令字数：	1 或 0.5
指令周期数：	1

ASRM	多精度算术右移（移位位数由短立即数确定）
语法：	{标号:} ASRM{.l} Ws, lit5, Wnd [Ws], [Ws++], [Ws--], [++Ws], [--Ws],
操作数：	Ws ∈ [W0 ...W15]；lit5 ∈ [0...31]；Wnd ∈ [W1 ...W14]
操作：	lit5[4:0]→ Shift_Val Ws[31] → 右移输入（算术移位） Ws[31:0]和 32'b0 → Shift_In[63:0] 将 Shift_In[63:0]算术右移 Shift_Val 位 → Shift_Out[63:0] Shift_Out[63:32] → Wnd Shift_Out[31:0] | Wnd-1 → Wnd-1
受影响的状态位：	N 和 Z
指令编码：	`1110 000k kkkk dddd pppU UUss ssUU 0011`
说明：	将源寄存器 Ws 中的内容算术右移 lit5 位（最大移位位数为 31 位），并将结果存放在目标寄存器 Wnd 中。包含下一个最低有效数据字的寄存器将已包含中间移位结果。将该值与从 Ws 移出的数据进行按位或运算以生成最终的移位结果，然后更新相应的目标寄存器。 Z 位为“粘住”位（只能清零）。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 k 位用于提供立即数操作数。 **注：** 1. 该指令只能工作于长字模式。
指令字数：	1
指令周期数：	2

ASRM		多精度算术右移 Wb 位			
语法：	{标号:}	ASRM{.l} Ws,		Wb,	Wnd
			[Ws],		
			[Ws++],		
			[Ws--],		
			[++Ws],		
			[--Ws],		

操作数：Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]；Wnd ∈ [W1 ...W14]

操作：
Wb[15:0]→ Shift_Val
Ws[31] → 右移输入（算术移位）
Ws[31:0]和 32'b0 → Shift_In[63:0]
将 Shift_In[63:0]算术右移 Shift_Val 位 → Shift_Out[63:0]
Shift_Out[63:32] → Wnd
Shift_Out[31:0] | Wnd-1 → Wnd-1

受影响的状态位：N 和 Z

指令编码：

1110	001U	ssss	dddd	pppU	UUww	wwUU	0011

说明：
将源寄存器 Ws 中的内容算术右移 Wb 位，并将结果存放在目标寄存器 Wnd 中。包含下一个最低有效数据字的寄存器将已包含中间移位结果。将该值与从 Ws 移出的数据进行按位或运算以生成最终的移位结果，然后更新相应的目标寄存器或存储器地址。
算术右移位数可以是 0 到 32 之间的任意值。如果 Wb[15:0]中保存的移位值超过 2'd32，则移位值将饱和至 2'd32 以保持一致性。Wb[31:16]中保存的任何数据都无效。

Z 位为“粘住”位（只能清零）。

w 位用于选择基准（移位计数）寄存器。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

注：该指令只能工作于长字模式。

指令字数：1

指令周期数：2

ASR		算术右移 Wb 位			
语法：	{标号:}	ASR.b	Ws,	Wb,	Wd
		ASR.bz	[Ws],		[Wd]
		ASR{.w}	[Ws++],		[Wd++]
		ASR.l	[Ws--],		[Wd--]
			[++Ws],		[++Wd]
			[--Ws],		[--Wd]

操作数：Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]

.....（续）

ASR	算术右移 Wb 位
操作：	Wb[15:0]→ Shift_Val 对于长字操作： Ws[31] → 右移输入（算术移位） Ws[31:0]和 32'b0 → Shift_In[63:0] 将 Shift_In[63:0]算术右移 Shift_Val 位 → Shift_Out[63:0] Shift_Out[63:32] → Wd 对于字操作： Ws[15] → 右移输入（算术移位） {16{Ws[15]}}、Ws[15:0]和 32'b0 → Shift_In[63:0] 将 Shift_In[63:0]算术右移 Shift_Val 位 → Shift_Out[63:0] Shift_Out[47:32] → Wd[15:0] 对于字节操作： Ws<7> → 右移输入（算术移位） {24{Ws<7>}}、Ws[7:0]和 32'b0 → Shift_In[63:0] 将 Shift_In[63:0]算术右移 Shift_Val 位 → Shift_Out[63:0] Shift_Out[39:32] → Wd[7:0]
受影响的状态位：	N 和 Z
指令编码：	`1010 100L wwww dddd pppq qqss ssUU B100`
说明：	将源寄存器中的内容算术右移 Wb 位，并将结果存放在目标寄存器 Wd 中。 无论 Wb[15:0]中的移位值为何，该指令都会生成正确的结果。 • 对于移位值 > 7 的字节操作： 如果 Ws[7] = 0，则 Wd[7:0] = 0x00，否则 Wd[7:0] = 0xFF • 对于移位值 > 15 的字操作： 如果 Ws[15] = 0，则 Wd[15:0] = 0x0000，否则 Wd[15:0] = 0xFFFF • 对于移位值 > 31 的长字操作： 如果 Ws[31] = 0，则 Wd[31:0] = 0x00000000，否则 Wd[31:0] = 0xFFFFFFFF Wb[31:16]中保存的任何数据都无效。 目标寄存器直接寻址扩展字节或字模式会将结果零扩展至 32 位，然后写入 Wd。 L 和 B 位用于选择操作数据宽度。 s 位用于选择源寄存器。 w 位用于选择基准（移位计数）寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。
指令字数：	1
指令周期数：	1

BRA C	如果进位/无符号大于或等于则转移			
语法：	{标号:}	BRA	C,	标号
	{标号:}	BRA	GEU,	
操作数：	标号由链接器解析为有符号字偏移量			

.....（续）

BRA C	如果进位/无符号大于或等于则转移

操作：	条件 = C
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移

受影响的状态位：	无
指令编码：	1010 101U nnnn nnnn nnnn nnnn nnnn 0010
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
指令字数：	1
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会执行取出的转移目标指令，从而使有效指令执行时间为一个周期。

BCLR	将 Ws 中的指定位清零			
语法：	{标号:}	BCLR.b BCLR{.w} BCLR.l	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws], SR	bit5
操作数：	Ws ∈ [W0 ...W15]； 字节：bit5 ∈ [0 ...7]； 字：bit5 ∈ [0 ...15]； 长字：bit5 ∈ [0 ...31]			
操作：	0 → Ws<bit5>			
受影响的状态位：	无（见注 1）			
指令编码：	S100 000b bbbb ssss pppU UUUU UUUU LB00			

.....（续）

BCLR	将 Ws 中的指定位清零
说明：	将寄存器 Ws 中由 bit5 指定的位清零。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 b 位用于定义指定要被清零的位位置的 bit5 值（对于 16 位操作码，为 bit5[4:0]）。 s 位用于选择源/目标寄存器。 p 位用于选择源寻址模式。 **注：** 1. 当以 SR 为目标时，SR 中的位将因指令操作而清零。
指令字数：	1 或 0.5
指令周期数：	1

BCLR	将 f 中的指定位清零
语法：	{标号:} BCLR.b f, bit3
操作数：	bit3 ∈ [0 ... 7]；f ∈ [0 ...1 MB]
操作：	0 → f[bit3]
受影响的状态位：	无
指令编码：	1100 000b ffff ffff ffff ffff ffff bb01
说明：	将文件寄存器 f 中由 bit3 指定的位清零。 w 位用于选择指定要被清零的位位置的 bit3 值。 f 位用于选择文件寄存器的地址。 **注：** 1. 该指令只能工作于字模式。 2. 操作码必须包括.b 扩展符。
指令字数：	1
指令周期数：	1

BFEXT	从 Ws 中提取位域，写入 Wb
语法：	{标号:} BFEXT{.w} bit5, wid6, Ws, Wb BFEXT.l [Ws], [Ws++], [Ws--], [++Ws], [--Ws], SR
操作数：	字：bit5 ∈ [0 ..15]；wid6 ∈ [0 .. 16] 长字：bit5 ∈ [0 ..31]；wid6 ∈ [0 .. 32] Ws ∈ [W0 ... W15]；Wb ∈ [W0 ...W14]
操作：	见下文
受影响的状态位：	无
指令编码：	1111 100m mmmm ssss pppc ccww wwcc L111

.....（续）

BFEXT	从 Ws 中提取位域，写入 Wb

说明：

从(Ws)中提取（复制）位域并将其写入 Wb 中。装入 Wb 的位域数据从 Wb[0]开始，Wb 内超出指定位域宽度的所有 MSb 都将清零。

要提取的位域的 LSb 在 Ws 内的位位置由操作数 bit5 定义。位域的宽度由操作数 wid6 定义，取值范围为 0 到 16 位（字操作）或 32 位（长字操作）。

在写入 Wb 之前，字模式会将结果零扩展至 32 位。

L 位用于选择字或长字操作。

w 位用于选择位域目标寄存器。

s 位用于选择数据源寄存器。

p 位用于选择源寻址模式。

ccccc 位用于定义位域 LSb 在目标字内的位置。

mmmmm 位用于定义位域 MSb 在目标字内的位置。

注：

1. 立即数 wid6 = 0 对于长字和字大小的操作都是合法值。

指令字数：1

指令周期数：1

BFEXT	从 f 中提取位域，写入 Wb

语法：{标号:} BFEXT{.w} bit5, wid6, f, Wb

BFEXT.l

操作数：bit5 ∈ [0 .. 31]；wid6 ∈ [0 .. 32]
Wb ∈ [W0 ... W14]；f ∈ [0 ...1 MB]

操作：见下文

受影响的状态位：无

指令编码：

第一个字	1100	110U	ffff	ffff	ffff	ffff	ffff	1001
第二个字	1100	110L	UUUm	mmmm	UUUc	ccww	wwcc	1101

说明：

从文件寄存器地址中提取（复制）位域并将其写入 Wb 中。装入 Wb 的位域数据从 Wb[0]开始，Wb 内超出指定位域宽度的所有 MSb 都将清零。

要提取的位域的 LSb 在文件寄存器内的位位置由操作数 bit5 定义。位域的宽度由操作数 wid6 定义，取值范围为 0 到 16 位（字操作）或 32 位（长字操作）。

字模式会将结果零扩展至 32 位，然后写入 Wb。

L 位用于选择字或长字操作。

w 位用于选择位域目标寄存器。

f 位用于选择源文件寄存器的地址。

ccccc 位用于定义位域 LSb 在目标字内的位置。

mmmmm 位用于定义位域 MSb 在目标字内的位置。

注：

1. 立即数 wid6 = 0 对于长字和字大小的操作都是合法值。
2. 可访问的文件地址空间为 1 MB。对于字操作，f 位的 LSb 为 1'b0。对于长字操作，f 位的低 2 位为 2'b00。

指令字数：2

指令周期数：2

BFINS	将 Wb 中的位域插入 Ws

语法：{标号:} BFINS{.w} bit5, wid6, Wb, Ws

BFINS.l [Ws]

[Ws++]

.....（续）

BFINS	将 Wb 中的位域插入 Ws						
					[Ws--]		
					[++Ws]		
					[--Ws]		
					SR		

操作数：字：bit5 ∈ [0 ..15]；wid6 ∈ [0 ..16]
长字：bit5 ∈ [0 ..31]；wid6 ∈ [0 ..32]
Wb ∈ [W0 ...W14]；Ws ∈ [W0 ...W15]

操作：见下文

受影响的状态位：无

指令编码：

1111	100m	mmmm	ssss	pppc	ccww	wwcc	L011

说明：从(Wb)读取位域并将其插入（复制）到 Ws 中。源自 Wb 的位域数据从 Wb[0]开始。Wb 内超出指定位域宽度的所有 MSb 都将被忽略，并且可设置为任何值。
插入的位域将覆盖 Ws 或 SR 中已有的位（即，该指令不会为了适应新的位域而对任何现有位进行移位）。
要插入的位域的 LSb 在 Ws 内的位位置由操作数 bit5 定义。位域的宽度由操作数 wid6 定义，取值范围为 0 到 16 位（字操作）或 32 位（长字操作）。
字模式会将 Ws 寄存器直接寻址结果零扩展至 32 位。
L 位用于选择字或长字操作。
w 位用于选择位域源寄存器。
s 位用于选择数据源/目标寄存器。
p 位用于选择源寻址模式。
ccccc 位用于定义位域 LSb 在目标字内的位置。
mmmmm 位用于定义位域 MSb 在目标字内的位置。

注：

1. 立即数 wid6 = 0 对于长字和字大小的操作都是合法值。

指令字数：1

指令周期数：1

BFINS	将 Wb 中的位域插入 f						
语法：	{标号:}	BFINS{.w} bit5,	wid6,	Wb,	f		
		BFINS.l					

操作数：字：bit5 ∈ [0 ..15]；wid6 ∈ [0 ..16]
长字：bit5 ∈ [0 ..31]；wid6 ∈ [0 ..32]
Wb ∈ [W0 ...W14]；f ∈ [0 ...1 MB]

操作：见下文

受影响的状态位：无

指令编码：

第一个字	1100	110U	ffff	ffff	ffff	ffff	ffff	0001
第二个字	1100	110L	UUUm	mmmm	UUUc	ccww	wwcc	0101

.....（续）

BFINS	将 Wb 中的位域插入 f

说明：

从(Wb)读取位域并将其插入（复制）到文件寄存器地址中。源自 Wb 的位域数据从 Wb[0]开始。Wb 内超出指定位域宽度的所有 MSb 都将被忽略，并且可设置为任何值。
插入的位域将覆盖文件寄存器中已有的位（即，该指令不会为了适应新的位域而对任何现有位进行移位）。

要插入的位域的 LSb 在文件寄存器内的位位置由操作数 bit5 定义。位域的宽度由操作数 wid6 定义，取值范围为 0 到 16 位（字操作）或 32 位（长字操作）。

w 位用于选择位域源寄存器。

f 位用于选择源/目标文件寄存器。

ccccc 位用于定义位域 LSb 在目标字内的位置。

mmmmm 位用于定义位域 MSb 在目标字内的位置。

注：

1. 立即数 wid6 = 0 对于长字和字大小的操作都是合法值。
2. 可访问的文件地址空间为 1 MB。对于字操作，f 位的 LSb 为 1'b0。对于长字操作，f 位的低 2 位为 2'b00。

指令字数：2

指令周期数：2

BFINS	将立即数中的位域插入 Ws

语法：

{标号:}	BFINS{.w} bit5,	wid6,	lit16,	Ws
	BFINS.l			[Ws]
				[Ws++]
				[Ws--]
				[++Ws]
				[--Ws]

操作数：bit5 ∈ [0 ..31]；wid6 ∈ [0 .. 32]
lit16 ∈ [0 .. 65536]；Ws ∈ [W0 ...W15]

操作：见下文

受影响的状态位：无

指令编码：

第一个字	1100	101U	UUUU	kkkk	kkkk	kkkk	kkkk	U001
第二个字	1100	101m	mmmm	ssss	pppc	ccUU	UUcc	L101

说明：

将立即数值中的位域插入（复制）到 Ws 中。源自立即数的位域数据从立即数的 LSb 开始。立即数值内超出指定位域宽度的所有 MSb 都将被忽略，并且可设置为任何值。
插入的位域将覆盖 Ws 中已有的位（即，该指令不会为了适应新的位域而对任何现有位进行移位）。

要插入的位域的 LSb 在 Ws 内的位位置由操作数 bit5 定义。位域的宽度由操作数 wid6 定义，取值范围为 0 到 16 位（字操作）或 32 位（长字操作）。

字模式会将 Ws 寄存器直接寻址结果零扩展至 32 位。

L 位用于选择字或长字操作。

k 位包含位域源值。

s 位用于选择源/目标寄存器。

p 位用于选择源寻址模式。

ccccc 位用于定义位域 LSb 在目标字内的位置。

mmmmm 位用于定义位域 MSb 在目标字内的位置。

注：

1. 立即数 wid6 = 0 对于长字和字大小的操作都是合法值。

指令字数：2

指令周期数：2

BRA GE	如果有符号大于或等于则转移

语法：{标号:} BRA GE, 标号

操作数：标号由链接器解析为有符号字偏移量（slit20）

操作：条件 = (N&&OV)||(!N&&!OV)
如果条件为真，则{
如果 slit20 = 1，则跳过下一条（16 位）指令
如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
否则(PC+4) + 2*slit20 → PC
}
否则无转移

受影响的状态位：无

指令编码：

1010	100U	nnnn	nnnn	nnnn	nnnn	nnnn	0110

说明：如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

注：如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。

指令字数：1

指令周期数：1（2 或 3）
如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

BRA GT	如果有符号大于则转移

语法：{标号:} BRA GT, 标号

操作数：标号由链接器解析为有符号字偏移量（slit20）

操作：条件 = (!Z&&N&&OV)||(!Z&&!N&&!OV);
如果条件为真，则{
如果 slit20 = 1，则跳过下一条（16 位）指令
如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
否则(PC+4) + 2*slit20 → PC
}
否则无转移

受影响的状态位：无

指令编码：

1010	100U	nnnn	nnnn	nnnn	nnnn	nnnn	0010

.....（续）

BRA GT	如果有符号大于则转移
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
指令字数：	1
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

BRA GTU	如果无符号大于则转移
语法：	{标号:} BRA GTU, 标号
操作数：	标号由链接器解析为有符号字偏移量（slit20）

操作：	条件 = (C&&!Z);
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移

受影响的状态位：	无
指令编码：	1010 111U nnnn nnnn nnnn nnnn nnnn 0110
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
指令字数：	1
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

BRA LE	如果有符号小于或等于则转移							
语法：	{标号:}	BRA	LE,	标号				
操作数：	标号由链接器解析为有符号字偏移量（slit20）							

操作：	条件 = Z||(N&&!OV)||(!N&&OV);
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移							

受影响的状态位：	无							
指令编码：	1010	100U	nnnn	nnnn	nnnn	nnnn	nnnn	1110
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。							
指令字数：	1							
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。							

BRA LEU	如果无符号小于或等于则转移							
语法：	{标号:}	BRA	LEU,	标号				
操作数：	标号由链接器解析为有符号字偏移量（slit20）							

操作：	条件 = !C||Z;
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移							

受影响的状态位：	无							
指令编码：	1010	111U	nnnn	nnnn	nnnn	nnnn	nnnn	0010

.....（续）

BRA LEU	如果无符号小于或等于则转移
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
指令字数：	1
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

BRA LT	如果有符号小于则转移
语法：	{标号:} BRA LT, 标号
操作数：	标号由链接器解析为有符号字偏移量（slit20）

操作：	条件 = (N&&!OV)||(!N&&OV);
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移

受影响的状态位：	无
指令编码：	1010 100U nnnn nnnn nnnn nnnn nnnn 1010
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
指令字数：	1
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

BRA NC/LTU	如果进位位为零/无符号小于则转移

语法：{标号:} BRA NC, 标号

{标号:} BRA LTU,

操作数：标号由链接器解析为有符号字偏移量（slit20）

操作：条件 = !C
如果条件为真，则{
如果 slit20 = 1，则跳过下一条（16 位）指令
如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
否则(PC+4) + 2*slit20 → PC
}
否则无转移

受影响的状态位：无

指令编码：

1010	101U	nnnn	nnnn	nnnn	nnnn	nnnn	0110

说明：如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

注：如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。

指令字数：1

指令周期数：1（2 或 3）
注：如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

BRA NN	如果非负则转移

语法：{标号:} BRA NN, 标号

操作数：标号由链接器解析为有符号字偏移量（slit20）

操作：条件 = !N
如果条件为真，则{
如果 slit20 = 1，则跳过下一条（16 位）指令
如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
否则(PC+4) + 2*slit20 → PC
}
否则无转移

受影响的状态位：无

指令编码：

1010	011U	nnnn	nnnn	nnnn	nnnn	nnnn	0110

.....（续）

BRA NN	如果非负则转移
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
指令字数：	1
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

BRA OV	如果未溢出则转移
语法：	{标号:} BRA NOV, 标号
操作数：	标号由链接器解析为有符号字偏移量（slit20）

操作：	条件 = !OV
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移

受影响的状态位：	无
指令编码：	1010 101U nnnn nnnn nnnn nnnn nnnn 1110
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
指令字数：	1
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

BRA NZ		如果非零则转移						
语法：	{标号:}	BRA	NZ,	标号				
操作数：	标号由链接器解析为有符号字偏移量（slit20）							

操作：	条件 = !Z
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移							

受影响的状态位：	无							
指令编码：	1010	011U	nnnn	nnnn	nnnn	nnnn	nnnn	1110
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。							
指令字数：	1							
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。							

BRA OA		如果累加器 A 溢出则转移						
语法：	{标号:}	BRA	OA,	标号				
操作数：	标号由链接器解析为有符号字偏移量（slit20）							

操作：	条件 = OA
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移							

受影响的状态位：	无							
指令编码：	1010	110U	nnnn	nnnn	nnnn	nnnn	nnnn	0010

.....（续）

BRA OA	如果累加器 A 溢出则转移
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
指令字数：	1
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

BRA OB	如果累加器 B 溢出则转移
语法：	{标号:} BRA OB, 标号
操作数：	标号由链接器解析为有符号字偏移量（slit20）

操作：	条件 = OB
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移

受影响的状态位：	无
指令编码：	1010 110U nnnn nnnn nnnn nnnn nnnn 0110
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
指令字数：	1
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

BOOTSWP		交换活动和非活动闪存地址空间					
语法：	{标号:}	BOOTSWP	Ws				
操作数：	Ws ∈ [W0 ...W14]						
操作：	如果(cfg_bootswap_disable = 0) 如果 Ws 有效（见下文） 1 * Z 如果(sec_dual_boot_active = 1) NVMCON.P2ACTIV * NVMCON.P2ACTIV 1 * NVMCON.SOFTSWAP 否则不交换分区 否则 0 * Z（且不交换分区） 否则作为双周期 NOP 指令执行（且不交换分区）						
受影响的状态位：	Z（使能 BOOTSWP 时）						
指令编码：	1111	101U	UUUU	ssss	UUUU	UUUU	UUU1	0011

说明：

如果使能 BOOTSWP 指令（cfg_bootswap_disable = 0），则在通知 NVM 控制器执行分区交换之前会先确认 Ws 中是否包含有效的引导序列（BTSEQ）值（见注 1）。

如果 Ws 有效且器件工作在双引导模式下（sec_dual_boot_active = 1），则会发生以下情况：

1. 翻转 NVMCON.P2ACTIV 状态位的状态，这将交换 PS 地址映射内的活动闪存地址空间与非活动闪存地址空间。
2. 将 NVMCON.SOFTSWAP 置 1，并且 Z = 1，表示分区交换成功。如果 Ws 有效，但器件未工作在双引导模式下，Z 位仍将设置为 1'b1，但不会发生分区交换。如果 Ws 无效，BOOTSWP 将设置 Z = 0（无论器件是否工作在双引导模式下），并且不会发生分区交换。如果未使能 BOOTSWP 指令，则将作为双周期 NOP 指令执行（Z 位将不受影响，并且不会发生分区交换）。s 位用于指定源寄存器 Ws。

注：需在执行 BOOTSWP 之前将非活动分区 BTSEQ 值装入 Ws。

指令字数：	1
指令周期数：	2（如果闪存地址空间交换成功，则需要额外的周期从目标空间取 PS 存储器指令。）

BRA OV		如果溢出则转移					
语法：	{标号:}	BRA	OV,	标号			
操作数：	标号由链接器解析为有符号字偏移量（slit20）						

操作：	条件 = OV
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移						

受影响的状态位：	无						
指令编码：	1010	101U	nnnn	nnnn	nnnn	nnnn	nnnn	1010

.....（续）

BRA OV	如果溢出则转移
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
指令字数：	1
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

BRA	无条件转移
语法：	{标号:} BRA 标号
操作数：	标号由链接器解析为有符号字偏移量（slit20）
操作：	(PC+4) + 2*slit20 → PC
受影响的状态位：	无
指令编码：	1010 111U nnnn nnnn nnnn nnnn nnnn 1010
说明：	程序将在向前或向后 1 MB 范围内无条件转移。 将二进制补码字节偏移量值“2*slit20”（PC 偏移量）加到 PC。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 n 位构成一个有符号立即数，用于指定自(PC + 4)的转移量（PS 字数）。
指令字数：	1
指令周期数：	1

BRA	计算转移
语法：	{标号:} BRA Wns
操作数：	Wns ∈ [W0 ... W14]
操作：	(PC + 4) + 2*Wns[19:0] → PC，NOP → 指令寄存器。
受影响的状态位：	无
指令编码：	1101 011U UUUU ssss UUUU UUUU UUUU 0010
说明：	在向前或向后 1 MB 的转移地址范围内进行计算转移。 Wns[19:0]中的有符号值表示自当前 PC 的 PS（16 位）字偏移量。将该值乘以 2 创建的字节地址随后会添加到 PC 的内容中以形成目标地址。因此，Wn 必须包含一个有符号值，用于指定转移时自(PC+4)的偏移量（PS 字数）。 s 位用于选择源寄存器。 **注：**如果 Wns[31:19]不为全 0 或全 1，则将启动地址错误陷阱。
指令字数：	1

.....（续）

BRA	计算转移
指令周期数：	2 **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

BREAK	中断
语法：	{标号:} BREAK
操作数：	无
操作：	应用（用户）模式： 作为 NOP 指令执行 调试器模式： 任务模式：停止用户代码执行。 调试模式：作为 NOP 指令执行
受影响的状态位：	无
指令编码：	S111 001U UUUU 0010 UUUU UUUU UUUU UU00
说明：	BREAK 仅在器件工作于任务模式时才会这样执行，在任何其他模式下则将作为 NOP 指令执行。 BREAK 将停止用户代码执行并从任务模式切换到调试模式，在调试模式下将执行驻留的调试执行程序（Debug Executive，DE）。该指令不会修改用户 PC。在调试模式下执行 BREAK 时，将阻止所有异常（包括陷阱）。 为了避免调试模式与任务模式代码之间可能出现的危险，BREAK 将在开始调试模式执行之前执行两个 FNOP。总指令周期计数包括这两个 FNOP。 S 位用于选择指令大小。 **注：**16 位编码（粗体）。
指令字数：	1 或 0.5
指令周期数：	3

BRA SA	如果 ACCA 饱和则转移
语法：	{标号:} BRA SA, 标号
操作数：	标号由链接器解析为有符号字偏移量（slit20）

操作：	条件 = SA
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移

受影响的状态位：	无
指令编码：	1010 110U nnnn nnnn nnnn nnnn nnnn 1010

.....（续）

BRA SA	如果 ACCA 饱和则转移
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
指令字数：	1
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

BRA SB	如果 ACCB 饱和则转移			
语法：	{标号:}	BRA	SB,	标号
操作数：	标号由链接器解析为有符号字偏移量（slit20）			

操作：	条件 = SB
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移			

受影响的状态位：	无			
指令编码：	1010 110U nnnn nnnn nnnn nnnn nnnn 1110			
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。			
指令字数：	1			
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。			

BSET		将 Ws 中的指定位置 1							
语法：	{标号:}	BSET.b BSET{.w} BSET.l	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws], SR	bit5					
操作数：	Ws ∈ [W0 ...W15]； 字节：bit5 ∈ [0 ...7]； 字：bit5 ∈ [0 ...15]； 长字：bit5 ∈ [0 ...31]								
操作：	1 → Ws<bit5>								
受影响的状态位：	无（见注 2）								
指令编码：	S100	001b	bbbb	ssss	pppU	UUUU	UUUU	LB00	
说明：	将寄存器 Ws 中由 bit5 指定的位置 1。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 b 位用于定义指定要被清零的位位置的 bit5 值（对于 16 位操作码，为 bit5[4:0]）。 s 位用于选择源/目标寄存器。 p 位用于选择源寻址模式。 关于修改量寻址信息，请参见 。 **注：** 1. 当以 SR 为目标时，SR 中的位将因指令操作而置 1。								
指令字数：	1 或 0.5								
指令周期数：	1								

BSET		将 f 中的指定位置 1							
语法：	{标号:}	BSET.b	f,	bit3					
操作数：	bit3 ∈ [0 ...7]；f ∈ [0 ...1 MB]								
操作：	1 → f<bit3>								
受影响的状态位：	无								
指令编码：	1100	001b	ffff	ffff	ffff	ffff	ffff	bb01	
说明：	将文件寄存器 f 中由 bit3 指定的位置 1。 w 位用于选择指定要被置 1 的位位置的 bit3 值。 f 位用于选择文件寄存器的地址。 **注：** 1. 该指令只能工作于字模式。 2. 操作码必须包括.b 扩展符。 3. 可访问的文件地址空间为 1 MB。								
指令字数：	1								
指令周期数：	1								

BSW		写 Ws 中的某位		
语法：	{标号:}	BSW.bC BSW.bZ BSW.{w}C	Ws, [Ws], [++Ws],	Wb

.....（续）

BSW	写 Ws 中的某位

BSW.{w}Z [--Ws],
BSW.lC [Ws++],
BSW.lZ [Ws--],

操作数：Wb ∈ [W0 ...W14]；Ws ∈ [W0 ...W15]

操作：如果是“.Z”选项，则 Z → Ws<(Wb)>
如果是“.C”选项，则 C → Ws<(Wb)>

受影响的状态位：无

指令编码：

S100	101L	wwww	ssss	pppU	UUUU	UUUG	BU00

说明：将 C 位或 Z 位的值写入 Ws 中由 Wb 定义的位编号。对于字节、字和长字操作，目标位编号分别由 Wb[2:0]、Wb[3:0]和 Wb[4:0]定义。Wb 内除了选择目标位所需的位之外的任何位都将被忽略，并且可以设置为任何值。

S 位用于选择指令大小。

L 和 B 位用于选择操作数据宽度。

w 位用于选择位选择寄存器。

G 位用于选择 Z 或 C 标志位作为源（G = 0 时选择 Z 标志）。

s 位用于选择源寄存器。

p 位用于选择源寻址模式。

指令字数：1 或 0.5

指令周期数：1

BTG	将 Ws 中的指定位翻转

语法：

{标号:}	BTG.b	Ws,	bit5
	BTG{.w}	[Ws],	
	BTG.l	[Ws++],	
		[Ws--],	
		[++Ws],	
		[--Ws],	
		SR	

操作数：Ws ∈ [W0 ...W15]；
字节：bit5 ∈ [0 ...7]；
字：bit5 ∈ [0 ...15]；
长字：bit5 ∈ [0 ...31]

操作：(Ws)[bit5] → Ws[bit5]

受影响的状态位：无（见注 2）

指令编码：

S100	010b	bbbb	ssss	pppU	UUUU	UUUU	LB00

说明：将寄存器 Ws 中由 bit5 指定的位翻转。

S 位用于选择指令大小。

L 和 B 位用于选择操作数据宽度。

b 位用于定义指定要被清零的位位置的 bit5 值（对于 16 位操作码，为 bit5[4:0]）

s 位用于选择源/目标寄存器。

p 位用于选择源寻址模式。

注：

1. 当以 SR 为目标时，SR 中的位将因指令操作而翻转。

指令字数：1 或 0.5

指令周期数：1

BTG		将 f 中的指定位翻转							
语法：	{标号:}	BTG.b	f,	bit3					
操作数：	bit3 ∈ [0 ...7]；f ∈ [0 ...1 MB]								
操作：	(f)[bit3] → (f)[bit3]								
受影响的状态位：	无								
指令编码：	1100	010b	ffff	ffff	ffff	ffff	ffff	bb01	
说明：	将文件寄存器 f 中由 bit3 指定的位翻转。 w 位用于选择指定要被清零的位位置的 bit3 值。 f 位用于选择文件寄存器的地址。 **注：** 1. 该指令只能工作于字模式。 2. 操作码必须包括.b 扩展符。 3. 可访问的文件地址空间为 1 MB。								
指令字数：	1								
指令周期数：	1								

BTST		测试 Ws 中的指定位		
语法：	{标号:}	BTST.bC	Ws,	bit5
		BTST.bZ	[Ws],	
		BTST.{w}C	[Ws++],	
		BTST.{w}Z	[Ws--],	
		BTST.lC	[++Ws],	
		BTST.lZ	[--Ws],	
操作数：	Ws ∈ [W0 ...W15]； 字节：bit5 ∈ [0 ...7]； 字：bit5 ∈ [0 ...15]； 长字：bit5 ∈ [0 ...31]			
操作：	如果是“.Z”选项，(Ws)[bit5] → Z 如果是“.C”选项，(Ws)[bit5] → C			
受影响的状态位：	C 或 Z			
指令编码：	S100	011b	bbbb	ssss	pppU	UUUU	UUUG	LB00
说明：	对寄存器 Ws 中由 bit5 指定的位进行测试。零标志包含该位的翻转值，或进位标志包含该位。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 G 位用于选择 Z 或 C 标志位作为源（G = 0 时选择 Z 标志）。 b 位用于定义指定要被清零的位位置的 bit5 值（对于 16 位操作码，为 bit5[4:0]）。 s 位用于选择源/目标寄存器。 p 位用于选择源寻址模式。			
指令字数：	1 或 0.5			
指令周期数：	1			

BTST		测试 f 中的指定位		
语法：	{标号:}	BTST.b	f,	bit3
操作数：	bit3 ∈ [0 ...7]；f ∈ [0 ...1 MB]			
操作：	(f)[bit3] → Z			

.....（续）

BTST		测试 f 中的指定位							
受影响的状态位：	Z								
指令编码：	1100	011b	ffff	ffff	ffff	ffff	ffff	bb01	
说明：	对文件寄存器 f 中由 bit3 指定的位进行测试，如果该位为零，则零标志位置 1，否则清零。文件寄存器内容保持不变。 b 位用于选择指定要被清零的位位置的 bit3 值。 f 位用于选择文件寄存器的地址。 **注：** 1. 该指令只能工作于字模式。 2. 操作码必须包括.b 扩展符。 3. 可访问的文件地址空间为 1 MB。								
指令字数：	1								
指令周期数：	1								

BTST		测试/置 1 Ws 中的指定位		
语法：	{标号:}	BTST.bC	Ws,	bit5
		BTST.bZ	[Ws],	
		BTST.{w}C	[Ws++],	
		BTST..{w}Z	[Ws--],	
		BTST.lC	[++Ws],	
		BTST..lZ	[--Ws],	
操作数：	Ws ∈ [W0 ...W15]； 字节：bit5 ∈ [0 ...7]； 字：bit5 ∈ [0 ...15]； 长字：bit5 ∈ [0 ...31]			
操作：	如果是“.Z”选项，首先(Ws)[bit5] → Z，然后 1 → Ws[bit5] 如果是“.C”选项，首先(Ws)[bit5] → C，然后 1 → Ws[bit5]			
受影响的状态位：	C 或 Z			
指令编码：	S100	100b	bbbb	ssss	pppU	UUUU	UUUG	LB00
说明：	对寄存器 Ws 中由 bit5 指定的位进行测试，然后置 1。 对于“.Z”选项，先将 Z 标志设置为由 bit5 指定的位的反码值再置 1 Ws 中由 bit5 指定的位，不修改 C 标志。对于“.C”选项，先将 C 标志设置为由 bit5 指定的位的值再置 1 Ws 中由 bit5 指定的位，不修改 Z 标志。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 G 位用于选择 Z 或 C 标志位作为源（G = 0 时选择 Z 标志）。 b 位用于定义指定要被清零的位位置的 bit5 值。 s 位用于选择源/目标寄存器。 p 位用于选择源寻址模式。							
指令字数：	1 或 0.5							
指令周期数：	1							

BTSTS		测试/置 1 f 中的指定位		
语法：	{标号:}	BTSTS.b	f,	bit3
操作数：	bit3 ∈ [0 ...7]；f ∈ [0 ...1 MB]			
操作：	首先(f)[bit3] → Z，然后 1 → (f)[bit3]			

.....（续）

BTSTS		测试/置 1 f 中的指定位						
受影响的状态位：	Z							
指令编码：	1100	100b	ffff	ffff	ffff	ffff	ffff	bb01

说明：

对文件寄存器 f 中由 bit3 指定的位进行测试，然后置 1。先将 Z 标志设置为由 bit3 指定的位的反码值再置 1 f 中由 bit3 指定的位。
w 位用于选择指定要被测试/置 1 的位位置的 bit3 值。

f 位用于选择文件寄存器的地址。

注：

1. 该指令只能工作于字模式。
2. 操作码必须包括.b 扩展符。
3. 可访问的文件地址空间为 1 MB。

指令字数： 1

指令周期数： 1

BTST		测试 Ws 中的指定位		
语法：	{标号:}	BTST.bC	Ws,	Wb
		BTST.bZ	[Ws],	
		BTST.{w}C	[Ws++]	
		BTST.{w}Z	[Ws--],	
		BTST.lC	[++Ws],	
		BTST.lZ	[--Ws],	

操作数： Wb ∈ [W0 ...W14]；Ws ∈ [W0 ...W15]

操作：
如果是“.Z”选项，(Ws)<(Wb)> → Z
如果是“.C”选项，(Ws)<(Wb)> → C

受影响的状态位：	C 或 Z						
指令编码：	S100	110L	wwww	ssss	pppU	UUUU	UUUG	BU00

说明：

在源寄存器 Ws 中对 Wb 中定义的位编号进行测试。对于字节、字和长字操作，目标位编号分别由 Wb[2:0]、Wb[3:0]和 Wb[4:0]定义。Wb 内除了选择目标位所需的位之外的任何位都将被忽略，并且可以设置为任何值。
对于“.Z”选项，将 Z 标志设置为被测试位的反码值，不修改 C 标志。对于“.C”选项，将 C 标志设置为被测试位的值，不修改 Z 标志。Wb[31:5]被忽略，并且可以设置为任何值。

S 位用于选择指令大小。

L 和 B 位用于选择操作数据宽度。

w 位用于选择位选择寄存器。

G 位用于选择 Z 或 C 标志位作为源（G = 0 时选择 Z 标志）。

s 位用于选择源寄存器。

p 位用于选择源寻址模式。

指令字数： 1 或 0.5

指令周期数： 1

BRA Z		如果为零则转移		
语法：	{标号:}	BRA	Z,	标号

操作数： 标号由链接器解析为有符号字偏移量（slit20）

.....（续）

BRA Z	如果为零则转移

操作：	条件 = Z
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移

受影响的状态位：	无
指令编码：	1010 011U nnnn nnnn nnnn nnnn nnnn 1010
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：**如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
指令字数：	1
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

4.5 指令说明（C 至 DTB）

CALL		调用子程序	
语法：	{标号:}	CALL	lit24

操作数： lit24 ∈ [0 ... 16 MB]

操作：
(PC) + 4 → PC,
(8'b0 和 PC[23:2]) → TOS[31:2]；2'b00 → TOS[1:0]，
(W15) + 4 → W15，
lit24→PC[23:0]；
NOP → 指令寄存器

受影响的状态位： 无

指令编码：

1101	111n	nnnn	nnnn	nnnn	nnnn	nnnn	nn10

说明：
对所有可执行存储器地址空间内的任何地址进行子程序调用。允许调用 32 位或 16 位指令。
长字对齐的返回地址(PC+4)被压入系统堆栈。24 位值“lit24”随后装入 PC（操作码不存储 LSb，该位总是为 1'b0）。
n 位形成目标地址。
注： （字节）PC 地址总是为字对齐或长字对齐。操作码不存储 LSb，因为该位总是为 1'b0。

指令字数： 1

指令周期数： 1

CALL		扩展调用子程序	
语法：	{标号:}	CALL	lit32

操作数： lit32 ∈ [16 MB[1] ...4 GB]

操作：
(PC) + 4 → PC,
PC[31:2]) → TOS[31:2]；2'b00 → TOS[1:0]，
(W15) + 4 → W15，
lit32 → PC[31:0]；
NOP → 指令寄存器

受影响的状态位： 无

指令编码：

第一个字	1111	010U	nnnn	nnnn	nnnn	nnnn	nnn0	0011
第二个字	1111	010U	UUUU	Unnn	nnnU	UUnn	nnnn	0111

说明：
对所有可执行存储器地址空间内的任何地址进行子程序调用。允许调用 32 位或 16 位指令。
长字对齐的返回地址(PC+4)被压入系统堆栈。32 位值“lit32”随后装入 PC。
n 位形成目标地址。
PC[31:0] = Word1[18:13]、Word2[9:4]和 Word1[23:4]
注： （字节）PC 地址总是为字对齐或长字对齐，使 LSb 总是为 1'b0。

指令字数： 2

指令周期数： 2

CALL		间接调用子程序	
语法：	{标号:}	CALL	Wns

操作数： Wns ∈ [W0...W14]

.....（续）

CALL		间接调用子程序
操作：	(PC) + 2 → PC， (8'b0 和 PC[23:2]) → TOS[31:2]；2'b00 → TOS[1:0]， (W15)+4 → W15， Wns[23:0] → PC[23:0]； NOP → 指令寄存器。	
受影响的状态位：	无	
指令编码：	1101 011U UUUU ssss UUUU UUUU UUUU 1110	
说明：	使用计算调用 PS（字）地址对程序存储器内的任何指令地址（16 位或 32 位）进行间接子程序调用。 长字对齐的返回地址(PC+4)被压入系统堆栈。24 位值（Wns[23:0]）随后装入 PC[23:0]。因此，Wns 必须包含 PS 字节地址。 Wns[0]的值会被忽略，PC[0]总是设置为 1'b0。 s 位用于指定源寄存器。 **注：** 1. 如果 Wns[31:24] != 8'h00，将启动地址错误陷阱。	
指令字数：	1	
指令周期数：	2 **注：**如果有异常等待处理，则不会执行取出的调用目标指令，从而使有效指令执行时间为一个周期。	

CLR		清零累加器
语法：{标号}	CLR	A B
操作数：	无	
操作：	0 → ACC(A 或 B)	
受影响的状态位：	OA 和 SA，*或者* OB 和 SB	
指令编码：	0111 001A UUUU 1100	
说明：	清零指定的累加器（A 或 B）。 A 位用于选择要清零的累加器。	
指令字数：	0.5	
指令周期数：	1	

CLR		将 f 清零	
语法：	{标号:}	CLR.b	f
		CLR{.w}	
		CLR.l	
操作数：	f ∈ [0 ...1 MB]		
操作：	0x00000000 → 文件寄存器（长字操作） 0x0000 → 文件寄存器（字操作） 0x00 → 文件寄存器（字节操作）		
受影响的状态位：	无		
指令编码：	1010 011L ffff ffff ffff ffff ffff B101		

.....（续）

CLR		将 f 清零						
说明：	清零文件寄存器。 L 和 B 位用于选择操作数据宽度。 f 位用于选择文件寄存器的地址。 **注：** 1. 流程与 D 位（操作码[2]）设置为 1 的任何其他文件操作相同。 2. 可访问的文件地址空间为 1 MB。							
指令字数：	1							
指令周期数：	1							

CLRWDT		清零看门狗定时器						
语法：	{标号:}	CLRWDT						
操作数：	无							
操作：	0 → WDT 寄存器							
受影响的状态位：	无							
指令编码：	0111	001U	UUUU	0101				
说明：	清零看门狗定时器寄存器。							
指令字数：	0.5							
指令周期数：	1							

COM		对 Ws 的内容取反						
语法：	{标号:}	COM.b COM.bz COM{.w} COM.l	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	Wd [Wd] [Wd++] [Wd--] [++Wd] [--Wd]				
操作数：	Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]							
操作：	(Ws) → Wd							
受影响的状态位：	Z 和 N							
指令编码：	S111	100L	dddd	ssss	pppq	qqUU	UUUU	BU00
说明：	计算源寄存器 Ws 内容的二进制反码，并将结果存放在目标寄存器 Wd 中。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。							
指令字数：	1 或 0.5							
指令周期数：	1							

COM		对 f 的内容取反				
语法：	{标号:}	COM.b COM.bz COM{.w} COM.l	f	{,Wnd}	{,WREG}	

.....（续）

COM		对 f 的内容取反							
操作数：	f ∈ [0 ...64 KB]；Wnd ∈ [W0 ...W14]								
操作：	(f) → 由 D 指定的目标寄存器								
受影响的状态位：	Z 和 N								
指令编码：	1101	100L	dddd	ffff	ffff	ffff	ffff	BD01	
说明：	计算文件寄存器内容的二进制反码，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wnd，则 D = 0 并将结果存放在 Wnd 中；否则，D = 1 并将结果存放在文件寄存器中。 L 和 B 位用于选择操作数据宽度。 D 位用于选择目标寄存器。 f 位用于选择文件寄存器的地址。 d 位用于选择工作寄存器。								
指令字数：	1								
指令周期数：	1								

CP		比较 Wb 和 Ws 并设置状态标志位		
语法：	{标号:}	CP.b	Wb,	Ws
		CP{.w}		[Ws]
		CP.l		[Ws++]
				[Ws--]
				[++Ws]
				[--Ws]

操作数：	Wb ∈ [W0 ...W15]；Ws ∈ [W0 ...W15]							
操作：	(Wb) - (Ws)							
受影响的状态位：	C、N、OV 和 Z							
指令编码：	S011	110L	wwww	ssss	pppU	UUUU	UUUU	BU00
说明：	计算(Wb) - (Ws)，然后设置标志位但不存储结果。相当于 SUB 指令（不带目标结果写操作）。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 w 位用于选择 Wb 源寄存器。 s 位用于选择 Ws 源寄存器。 p 位用于选择源寻址模式。							
指令字数：	1 或 0.5							
指令周期数：	1							

CPB		带借位位比较 Wb 和 Ws 并设置状态标志位		
语法：	{标号:}	CPB.b	Wb,	Ws
		CPB{.w}		[Ws]
		CPB.l		[Ws++]
				[Ws--]
				[++Ws]
				[--Ws]

操作数：	Wb ∈ [W0 ...W15]；Ws ∈ [W0 ...W15]							
操作：	(Wb) - (Ws) - (C)							
受影响的状态位：	C、N、OV 和 Z							
指令编码：	S011	111L	wwww	ssss	pppU	UUUU	UUUU	BU00

.....（续）

CPB	带借位位比较 Wb 和 Ws 并设置状态标志位
说明：	计算(Wb) - (Ws) - (C)，然后设置标志位但不存储结果。相当于不带目标结果写操作的 SUBB 指令。 Z 位为“粘住”位（只能清零）。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 w 位用于选择 Wb 源寄存器。 s 位用于选择 Ws 源寄存器。 p 位用于选择源寻址模式。
指令字数：	1 或 0.5
指令周期数：	1

CPB		带借位位比较 Ws 和 lit13 并设置状态标志位		
语法：	{标号:}	CPB.b CPB{.w} CPB.l	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	lit13
操作数：	Ws ∈ [W0 ...W15]； lit13 ∈ [0 ...8191]（32 位操作码）；lit4 ∈ [0 ...15]（16 位操作码）[1]			
操作：	(Ws) - lit13 - (C) 或 (Ws) - lit4 - (C)[1] **注：**立即数经过零扩展至操作所选的数据大小			
受影响的状态位：	C、N、OV 和 Z			
指令编码：	`S011 101L kkkk ssss pppk kkkk kkkk BU00`			
说明：	对立即数进行零扩展，然后计算(Ws) - 立即数 - (C)。设置标志位但不存储结果。 Z 位是粘住位（只能清零）。 L 和 B 位用于选择操作数据宽度。 p 位用于选择源寻址模式。 s 位用于选择源寄存器。 k 位用于提供立即数操作数。 lit13[12:0] = 操作码[12:4]和操作码[23:20] lit4[3:0] = 操作码[23:20]			
指令字数：	1 或 0.5			
指令周期数：	1			

CP		比较 Ws 和 lit13 并设置状态标志位		
语法：	{标号:}	CP.b CP{.w} CP.l	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	lit13
操作数：	Ws ∈ [W0 ...W15]； lit13 ∈ [0 ...8191]（32 位操作码）；lit4 ∈ [0 ...15]（16 位操作码）[1]			

.....（续）

CP	比较 Ws 和 lit13 并设置状态标志位
操作：	(Ws) - lit13（32 位操作码） 或(Ws) - lit4（16 位操作码）[1] 注：立即数经过零扩展至操作所选的数据大小
受影响的状态位：	C、N、OV 和 Z
指令编码：	S011 100L kkkk ssss pppk kkkk kkkk BU00
说明：	根据需要对立即数进行零扩展，然后计算(Ws) - 立即数。设置标志位但不存储结果。 Z 位是粘住位（只能清零）。 L 和 B 位用于选择操作数据宽度。 p 位用于选择源寻址模式。 s 位用于选择源寄存器。 k 位用于提供立即数操作数。 lit13[12:0] = 操作码[12:4]和操作码[23:20] lit4[3:0] = 操作码[23:20]
指令字数：	1 或 0.5
指令周期数：	1

CPB	带借位位比较 Wb 和 lit16 并设置状态标志位			
语法：	{标号:}	CPB.b CPB{.w} CPB.l	Wb,	lit16
操作数：	Wb ∈ [W0 ...W15]；lit16 ∈ [0 ...65535]			
操作：	(Wb) - lit16 - (C)			
受影响的状态位：	C、N、OV 和 Z			
指令编码：	1100 011L wwww kkkk kkkk kkkk kkkk B110			
说明：	计算(Wb) - lit16 - (C)，然后设置标志位但不存储结果。 Z 位为“粘住”位（只能清零）。 L 和 B 位用于选择操作数据宽度。 w 位用于选择源寄存器。 k 位用于提供立即数操作数。			
指令字数：	1			
指令周期数：	1			

CP	比较 f 和 Ws 并设置状态标志位			
语法：	{标号:}	CP.b CP{.w} CP.l	f,	Ws
操作数：	f ∈ [0 ...64 KB]；Ws ∈ [W0 ...W15]			
操作：	(f) - (Ws)			
受影响的状态位：	C、N、OV 和 Z			
指令编码：	1010 101L ssss ffff ffff ffff ffff BU01			
说明：	计算(f) - (Ws)，然后设置标志位但不存储结果。相当于带堆栈接收目标（[W15]）的 SUBWF 指令。 L 和 B 位用于选择操作数据宽度。 f 位用于选择文件寄存器的地址。 s 位用于选择工作寄存器。			
指令字数：	1			

.....（续）

CP		比较 f 和 Ws 并设置状态标志位						
指令周期数：	1							

CP0		比较 f 和零并设置状态标志位						
语法：	{标号:}	CP0.b	f					
		CP0{.w}						
		CP0.l						
操作数：	f ∈ [0 ...1 MB]							
操作：	长字： (f) - 0x0000000 字： (f) - 0x0000 字节： (f) - 0x00							
受影响的状态位：	C、N、OV 和 Z							
指令编码：	1010	100L	ffff	ffff	ffff	ffff	ffff	BU01
说明：	计算(f)减去所选数据大小的零值，设置标志位但不存储结果。该操作总是将 C 位置 1，将 OV 清零。 L 和 B 位用于选择操作数据宽度。 f 位用于选择文件寄存器的地址。 可访问的文件地址空间为 1 MB。							
指令字数：	1							
指令周期数：	1							

CPB		带借位位比较 f 和 Ws 并设置状态标志位						
语法：	{标号:}	CPB.b	f,	Ws				
		CPB{.w}						
		CPB.l						
操作数：	f ∈ [0 ...64 KB]；Ws ∈ [W0 ...W15]							
操作：	(f) - (Ws) - (C)							
受影响的状态位：	C、N、OV 和 Z							
指令编码：	1010	110L	ssss	ffff	ffff	ffff	ffff	BU01
说明：	计算(f) - (Ws) - (C)，设置标志位但不存储结果。相当于带堆栈接收目标写操作（[W15]）的 SUBBWF 指令。 Z 位为“粘住”位（只能清零）。 L 和 B 位用于选择操作数据宽度。 f 位用于选择文件寄存器的地址。 s 位用于选择工作寄存器。							
指令字数：	1							
指令周期数：	1							

CP		比较 Wb 和 lit16 并设置状态标志位						
语法：	{标号:}	CP.b	Wb,	lit16				
		CP{.w}						
		CP.l						
操作数：	Wb ∈ [W0 ...W15]；lit16 ∈ [0 ...65535]							
操作：	(Wb) - lit16							
受影响的状态位：	C、N、OV 和 Z							
指令编码：	1100	010L	wwww	kkkk	kkkk	kkkk	kkkk	B110

.....（续）

CP		比较 Wb 和 lit16 并设置状态标志位			
说明：	计算(Wb) - lit16，然后设置标志位但不存储结果。 L 和 B 位用于选择操作数据宽度。 w 位用于选择源寄存器。 k 位用于提供立即数操作数。				
指令字数：	1				
指令周期数：	1				

CTXTSWP		CPU 寄存器现场切换至立即数定义的现场			
语法：	{标号:}	CTXTSWP lit3			
操作数：	lit3 ∈ [0 ... 7]				
操作：	将 CPU 寄存器现场切换至 lit3 定义的现场 lit3 → SR.CTX[2:0]				
受影响的状态位：	无				
指令编码：	0111	001U	Ukkk	0110	
说明：	该指令将强制将 CPU 寄存器现场从当前现场切换到由 lit3 值定义的目标现场。如果支持，协处理器现场也将相应地切换。 现场切换将更新当前现场标识符（SR.CTX[2:0]），以反映新激活的 CPU 寄存器现场。 k 位用于选择目标寄存器现场。				
指令字数：	0.5				
指令周期数：	2				

CTXTSWP		CPU 寄存器现场切换至 Ws 定义的现场							
语法：	{标号:}	CTXTSWP Wns							
操作数：	Wns ∈ [W0 ...W14]								
操作：	将 CPU 寄存器现场切换至 Wns[2:0]中定义的现场 [1] Wns[2:0] → SR.CTX[2:0]								
受影响的状态位：	无								
指令编码：	1101	100U	UUUU	ssss	UUUU	UUUU	UUUU	U110	
说明：	该指令将强制将 CPU 寄存器现场从当前现场（W0 至 W7）切换到由 Wns[2:0]的内容定义的目标现场。任何实例化协处理器中支持的寄存器现场也将切换。 现场切换将更新当前现场标识符（SR.CTX[2:0]），以反映新激活的 CPU 寄存器现场。 s 位用于选择源寄存器。 **注：** 1. Wns[31:3]的内容会被忽略。								
指令字数：	1								
指令周期数：	2								

DEC		f 内容递减 1			
语法：	{标号:}	DEC.b DEC.bz DEC{.w} DEC.l	f	{,Wnd}	{,WREG}
操作数：	f ∈ [0 ...64 KB]；Wnd ∈ [W0 ...W14]				
操作：	(f) - 1 → 由 D 指定的目标寄存器				

.....（续）

DEC		f 内容递减 1						
受影响的状态位：	C、N、OV 和 Z							
指令编码：	1101	011L	dddd	ffff	ffff	ffff	ffff	BD01

说明：

将文件寄存器中的内容减 1，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wnd，则 D = 0 并将结果存放在 Wnd 中；否则，D = 1 并将结果存放在文件寄存器中。
L 和 B 位用于选择操作数据宽度。

D 位用于选择目标寄存器。

f 位用于选择文件寄存器的地址。

d 位用于选择工作寄存器。

指令字数：1

指令周期数：1

DEC2		f 内容递减 2			
语法：	{标号:}	DEC2.b DEC2.bz DEC2{.w} DEC2.l	f	{,Wnd}	{,WREG}

操作数：f ∈ [0 ...64 KB]；Wnd ∈ [W0 ...W14]

操作：(f) - 2 → 由 D 指定的目标寄存器

受影响的状态位：C、N、OV 和 Z

指令编码：	1101	111L	dddd	ffff	ffff	ffff	ffff	BD01

说明：

将文件寄存器中的内容减 2，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wnd，则 D = 0 并将结果存放在 Wnd 中；否则，D = 1 并将结果存放在文件寄存器中。
L 和 B 位用于选择操作数据宽度。

D 位用于选择目标寄存器。

f 位用于选择文件寄存器的地址。

d 位用于选择工作寄存器。

指令字数：1

指令周期数：1

DISICTL		禁止中断控制立即数		
语法：	{标号:}	DISICTL	lit3	{ ,Wd} { ,[Wd]} { ,[Wd++]} { ,[Wd--]} { ,[++Wd]} { ,[--Wd]}

操作数：lit3 ∈ [0 ...7]；Wd ∈ [W0 ...W15]

操作：禁止等于或低于由 lit3 定义的 IPL 阈值（IPLT[2:0]）的中断
设置 lit3 = 0 将允许不高于 SR.IPL[2:0]的所有中断优先级。

如果声明了 Wd，则在 IPLT[2:0] → Wd[31:0]之前先对其进行零扩展

受影响的状态位：无

指令编码：	1101	110U	dddd	UUUU	UUUq	qqUU	Ukkk	0W10

.....（续）

DISICTL	禁止中断控制立即数
说明：	该指令用于禁止等于或低于由 lit3 定义的 IPL 阈值（IPLT[2:0]）的中断，从后续指令开始有效。该指令无法禁止陷阱。 当前的 DISICTL 3 位 IPLT 映射到存储器（DISIIPL.IPLT[2:0]），用户可以随时读取。写入 DISIIPL 不会产生任何影响。 此外，如果声明了目标（W = 1），则可选择将在执行 DISICTL 之前确定的 IPLT 写入目标寄存器 Wd。写入 IPLT[2:0]之前，先将其零扩展至 32 位。如果未声明目标操作数，则不会写入任何内容（W = 0）。这有利于嵌套 DISICTL 和/或 DISICTLW 指令。 W 位用于确定是否需要目标写操作。 k 位为无符号立即数，用于指定 DISICTL IPL 阈值。 d 位用于选择目标寄存器。 q 位用于选择目标寻址模式。 **注：**该指令不会修改 SR.IPL[2:0]。
指令字数：	1
指令周期数：	1

DISICTL	禁止中断控制 Wns			
语法：	{标号:}	DISICTL	Wns	{ ,Wd} { ,[Wd]} { ,[Wd++]} { ,[Wd--]} { ,[++Wd]} { ,[--Wd]}
操作数：	Wns ∈ [W0 ...W14]；Wd ∈ [W0 ...W15]			
操作：	禁止等于或低于由 Wns[2:0]定义的 IPL 阈值（IPLT[2:0]）的中断 Wns[2:0] = 0 将允许所有中断优先级 如果声明了 Wd，则在 IPLT[2:0] → Wd[31:0]之前先进行零扩展			
受影响的状态位：	无			
指令编码：	`1101 110U dddd ssss UUUq qqUU UUUU 1W10`			
说明：	该指令用于禁止等于或低于由源 Wns 的低 3 位定义的 IPL 阈值(IPLT[2:0])的中断（Wns 中剩余的高位被忽略），从后续指令开始有效。该指令无法禁止陷阱。 当前的 DISICTLW 3 位 IPLT 映射到存储器（DISIIPL.IPLT[2:0]），用户可以随时读取。写入 DISIIPL 不会产生任何影响。 此外，如果声明了目标（W = 1），则可选择将在执行 DISICTL 之前确定的 IPLT 写入目标寄存器 Wd。写入 IPLT[2:0]之前，先将其零扩展至 32 位。如果未声明目标操作数，则不会写入任何内容（W = 0）。这有利于嵌套 DISICTL(W)指令。 该指令可用于（可选择与 DISICTLR 一起使用）控制系统中断的影响。 W 位用于确定是否需要目标写操作。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 q 位用于选择目标寻址模式。 **注：**该指令不会修改 SR.IPL[2:0]。			
指令字数：	1			
指令周期数：	1			

DIVF	有符号小数 16 位/16 位和 32 位/16 位除法			
语法：	{标号:}	DIVF.w DIVF.l	Wm	Wn

.....（续）

DIVF	有符号小数 16 位/16 位和 32 位/16 位除法

操作数：Wn ∈ [W0 ... W14]；Wm ∈ [W0 ... W13]

操作：DIVF.w (16/16)：Wm[15:0] = 被除数，Wn[15:0] = 除数：
Wm << 16；16'b0 → Wm[15:0]；

Wm / Wn → 16'b0 和 Wm[15:0]；
余数 → 16'b0 和 W(m+1)[15:0]

DIVF.l (32/16)；Wm[31:0] = 被除数；Wn[15:0] = 除数：

Wm / Wn → 16'b0 和 Wm[15:0]；
余数 → 16'b0 和 W(m+1)[15:0]

受影响的状态位：C、N、OV 和 Z

指令编码：

1110	100L	wwww	ssss	UUUU	UUUU	UUUU	1011

说明：迭代的有符号小数 32 位（或 16 位）/16 位除法，商和余数均为 16 位，这两者在分别写入 Wm 和 W(m+1)之前均进行零扩展。余数的符号将与被除数的符号相同。执行 16 位/16 位除法之前会将被除数换算为 32 位小数值。
该指令必须执行 6 次才能生成正确的商和余数。这只能通过执行迭代次数为 5（即，总共 5+1 次迭代）的 REPEAT 指令并将 DIVF.x 指令作为其目标来实现。可在任何迭代边界处中断 REPEAT 循环。

根据除法算法修改 C。

如果余数清零，则将 Z 置 1。否则，将 Z 清零。

如果余数为负，则将 N 置 1。否则，将 N 清零。
如果除法将导致溢出，则将 OV 置 1。商和余数是确定的，但无意义。
w 位用于选择源（被除数）寄存器。

s 位用于选择源（除数）寄存器。

注：

1. 第一次迭代期间会测试除数是否为零。尝试以零作为除数时，将在除法执行过程中的第一个 DIVF.x 指令周期内触发一个算术错误陷阱。
2. Wn 不能与 Wm 或 W(m+1)共用同一个 W 寄存器。

指令字数：1

指令周期数：6

DIVFL	有符号小数 32 位/32 位除法

语法：{标号:} DIVFL Wm Wn

操作数：Wn ∈ [W0 ...W14]；Wm ∈ [W0 ...W13]

操作：32 位/32 位；Wm = 被除数，Wn = 除数：
Wm / Wn → Wm[31:0]；余数 → W(m+1)[31:0]

受影响的状态位：C、N、OV 和 Z

指令编码：

1110	101U	wwww	ssss	UUUU	UUUU	UUUU	1011

.....（续）

DIVFL	有符号小数 32 位/32 位除法

说明：

迭代的有符号小数 32 位/32 位除法，商和余数均为 32 位。余数的符号将与被除数的符号相同。

该指令必须执行 10 次才能生成正确的商和余数。这只能通过执行迭代次数为 9（即，总共 9+1 次迭代）的 REPEAT 指令并将 DIVFL 指令作为其目标来实现。可在任何迭代边界处中断 REPEAT 循环。

根据除法算法修改 C。

如果余数清零，则将 Z 置 1。否则，将 Z 清零。
如果余数为负，则将 N 置 1。否则，将 N 清零。
如果除法将导致溢出，则将 OV 置 1。商和余数是确定的，但无意义。
w 位用于选择源（被除数）寄存器。

s 位用于选择源（除数）寄存器。

注：

1. 第一次迭代期间会测试除数是否为零。尝试以零作为除数时，将在除法执行过程中的第一个 DIVFL 指令周期内触发一个算术错误陷阱。
2. Wn 不能与 Wm 或 W(m+1)共用同一个 W 寄存器。

指令字数：1

指令周期数：10

DIVS	有符号整数 16 位/16 位和 32 位/16 位除法

语法：{标号:} DIVS.w Wm, Wn
DIVS.l

操作数：Wn ∈ [W0 ...W14]，Wm ∈ [W0 ...W13]

操作：DIVS.w (16/16)：Wm[15:0] = 被除数，Wn[15:0] = 除数：
{16{Wm[15]}}和 Wm[15:0] → Wm[31:0]；

Wm / Wn → 16'b0 和 Wm[15:0]；
余数 → 16'b0 和 W(m+1)[15:0]

DIVS.l (32/16)：Wm[31:0] = 被除数，Wn[15:0] = 除数：

Wm / Wn → 16'b0 和 Wm[15:0]；
余数 → 16'b0 和 W(m+1)[15:0]

受影响的状态位：C、N、OV 和 Z

指令编码：

1110	100L	wwww	ssss	UUUU	UUUU	UUUU	0011

.....（续）

DIVS	有符号整数 16 位/16 位和 32 位/16 位除法

说明：

迭代的有符号整数 32 位（或 16 位）/16 位除法，商和余数均为 16 位，这两者在分别写入 Wm 和 W(m+1)之前均经过零扩展至 32 位。余数的符号将与被除数的符号相同。执行 16 位/16 位除法之前会对被除数进行符号扩展。
该指令必须执行 6 次才能生成正确的商和余数。这只能通过执行迭代次数为 5（即，总共 5+1 次迭代）的 REPEAT 指令并将 DIVS 指令作为其目标来实现。可在任何迭代边界处中断 REPEAT 循环。

根据除法算法修改 C。

如果余数清零，则将 Z 置 1。

否则，将 Z 清零。
如果余数为负，则将 N 置 1。

否则，将 N 清零。
如果除法将导致溢出，则将 OV 置 1。

否则，将 OV 清零。商和余数是确定的，但无意义。

L 位用于选择 32 位或 16 位被除数大小。

w 位用于选择源（被除数）寄存器。

s 位用于选择源（除数）寄存器。

注：

1. 第一次迭代期间会测试除数是否为零。尝试以零作为除数时，将在除法执行过程中的第一个 DIVS.x 指令周期内触发一个算术错误陷阱
2. Wn 不能与 Wm 或 W(m+1)共用同一个 W 寄存器。

指令字数：1

指令周期数：6

DIVSL	有符号整数 32 位/32 位除法

语法：{标号:} DIVSL Wm, Wn

操作数：Wn ∈ [W0 ...W14]，Wm ∈ [W0 ...W13]

操作：32 位/32 位：Wm = 被除数，Wn = 除数：
Wm / Wn → Wm；余数 → W(m+1)

受影响的状态位：C、N、OV 和 Z

指令编码：

1110	101U	wwww	ssss	UUUU	UUUU	UUUU	0011

说明：

迭代的有符号整数 32 位/32 位除法，商和余数均为 32 位。余数的符号将与被除数的符号相同。
该指令必须执行 10 次才能生成正确的商和余数。这只能通过执行迭代次数为 9（即，总共 9+1 次迭代）的 REPEAT 指令并将 DIVSL 指令作为其目标来实现。可在任何迭代边界处中断 REPEAT 循环。

根据除法算法修改 C。

如果余数清零，则将 Z 置 1。否则，将 Z 清零。

如果余数为负，则将 N 置 1。否则，将 N 清零。

如果除法将导致溢出，则将 OV 置 1。否则，将 OV 清零。商和余数是确定的，但无意义。

w 位用于选择源（被除数）寄存器。

s 位用于选择源（除数）寄存器。

注：

1. 第一次迭代期间会测试除数是否为零。尝试以零作为除数时，将在除法执行过程中的第一个 DIVSL 指令周期内触发一个算术错误陷阱。
2. Wn 不能与 Wm 或 W(m+1)共用同一个 W 寄存器。

指令字数：1

指令周期数：10

DIVU		无符号整数 16 位/16 位和 32 位/16 位除法		
语法：	{标号:}	DIVU.w	Wm,	Wn
		DIVU.l		

操作数： Wn ∈ [W0 ...W14]，Wm ∈ [W0 ...W13]

操作： DIVU.w (16/16)：Wm[15:0] = 被除数，Wn[15:0] = 除数：
16'b0 → Wm[31:16]；
Wm / Wn → 16'b0 和 Wm[15:0]；余数 → 16'b0 和 W(m+1)[15:0]
DIVU.l (32/16)：Wm[31:0] = 被除数，Wn[15:0] = 除数：
Wm / Wn → 16'b0 和 Wm[15:0]；余数 → 16'b0 和 W(m+1)[15:0]

受影响的状态位： C、N、OV 和 Z

指令编码：

1110	100L	wwww	ssss	UUUU	UUUU	UUUU	0111

说明： 迭代的无符号整数 32 位（或 16 位）/16 位除法，商和余数均为 16 位，这两者在分别写入 Wm 和 W(m+1)之前均经过零扩展至 32 位。执行 16 位/16 位除法之前也会对被除数进行零扩展。
该指令必须执行 6 次才能生成正确的商和余数。这只能通过执行迭代次数为 5（即，总共 5+1 次迭代）的 REPEAT 指令并将 DIVU 指令作为其目标来实现。可在任何迭代边界处中断 REPEAT 循环。

根据除法算法修改 C。

如果余数清零，则将 Z 置 1。否则，将 Z 清零。

N 总是清零。

OV（DIVU.w）总是清零，因为不可能发生溢出。

如果除法将导致溢出，则将 OV（DIVU.l）置 1。

否则，将 OV 清零。

L 位用于选择 32 位或 16 位被除数大小。

w 位用于选择源（被除数）寄存器。

s 位用于选择源（除数）寄存器。

注：

1. 第一次迭代期间会测试除数是否为零。尝试以零作为除数时，将在除法执行过程中的第一个 DIVU.x 指令周期内触发一个算术错误陷阱
2. Wn 不能与 Wm 或 W(m+1)共用同一个 W 寄存器。

指令字数： 1

指令周期数： 6

DIVUL		无符号整数 32 位/32 位除法		
语法：	{标号:}	DIVUL	Wm,	Wn

操作数： Wn ∈ [W0 ...W14]，Wm ∈ [W0 ...W13]

操作： 32 位/32 位：Wm = 被除数，Wn = 除数：
Wm / Wn → Wm；余数 → W(m+1)

受影响的状态位： C、N、OV 和 Z

指令编码：

1110	101U	wwww	ssss	UUUU	UUUU	UUUU	0111

.....（续）

DIVUL	无符号整数 32 位/32 位除法

说明：

迭代的无符号整数 32 位/32 位除法，商和余数均为 32 位。
该指令必须执行 10 次才能生成正确的商和余数。这只能通过执行迭代次数为 9（即，总共 9+1 次迭代）的 REPEAT 指令并将 DIVUL 指令作为其目标来实现。可在任何迭代边界处中断 REPEAT 循环。

根据除法算法修改 C。

如果余数清零，则将 Z 置 1。

否则，将 Z 清零。

N 总是清零。

OV 总是清零，因为不可能发生溢出。

w 位用于选择源（被除数）寄存器。

s 位用于选择源（除数）寄存器。

注：

1. 第一次迭代期间会测试除数是否为零。尝试以零作为除数时，将在除法执行过程中的第一个 DIVUL 指令周期内触发一个算术错误陷阱。
2. Wn 不能与 Wm 或 W(m+1)共用同一个 W 寄存器。

指令字数：1

指令周期数：10

DTB	递减、测试和转移

语法：{标号:} DTB Wn, 标号

操作数：Wn ∈ [W0 ...W14]；
标号由链接器解析为有符号字偏移量（slit16）

操作：Wn = Wn - 1；
如果(Wn != 0 [见下文])，则{
如果 slit16 = 1，则跳过下一条（16 位）指令
如果(slit16 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
否则(PC+4) + 2*slit16 → PC
}
否则无转移

受影响的状态位：无

指令编码：

1011	100U	ssss	nnnn	nnnn	nnnn	nnnn	UU01

.....（续）

DTB	递减、测试和转移
说明：	递减 Wn[31:0]并将结果回写到 Wn（见注 2）。递减后测试 Wn[31:0]，如果 Wn[31:0] != 0，则转移至目标地址。当 Wn[31:0] = 0 时，不转移。 当作为代码块循环计数器使用（DTB 位于循环的末尾）时，DTB 将迭代循环（即，转移）Wn 次，并将在 Wn = 0x0000_0000 时退出循环（见注 2）。 如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 64 KB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit16”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit16”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit16”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit16。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 **注：** 1. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。 2. 在 Wn = 0 时执行 DTB 将导致循环计数为 2^{31}。
指令字数：	1
指令周期数：	1（2 或 3） **注：**如果有异常等待处理，则不会获取转移目标指令，从而使有效指令执行时间为一个周期。

4.6 指令说明（E 至 MULUU）

ED		相减后求平方，结果存入累加器（部分欧几里德距离）			
语法：{标号}	ED{.w}	Wx,	Wy,	A	{,AWB}
	ED.l	[Wx],	[Wy],	B	
		[Wx]+=kx	, [Wy]+=ky,		
		[Wx]-=kx	, [Wy]-=ky,		
		[Wx+=kx]	, [Wy+=ky],		
		[Wx-=kx]	, [Wy-=ky],		
		[Wx+W12]	, [Wy+W12],		

操作数：

Wx ∈ {W0 ... W14}；Wy ∈ {W0 ... W14}
字模式：kx ∈ {-8, -6, -4, -2, 2, 4, 6, 8}；ky ∈ {-8, -6, -4, -2, 2, 4, 6, 8}；

长字模式：kx ∈ {-16, -12, -8, -4, 4, 8, 12, 16}；ky ∈ {-16, -12, -8, -4, 4, 8, 12, 16}；

AWB ∈ {W0, W1, W2, W3, W13, [W13++], [W15++][4]}

操作：

$((Wx) - (Wy))^2$ → ACC(A 或 B)；
舍入后的(ACC(B 或 A)) → AWB

当使用执行前修改/执行后修改的间接寻址时：

(Wx)+kx→Wx 或(Wx)-kx→Wx；

(Wy)+ky→Wy 或(Wy)-ky→Wy；

受影响的状态位：

OA 和 SA，*或者* OB 和 SB

指令编码：

```
1101        10AL        wwww        ssss        IIIi        iJJJ      jjaa      a011
```

说明：

用于计算$(A-B)^2$ 函数的指令。对同时从 Wx 和 Wy 读取的数据或同时从 X 和 Y 地址空间获取的数据进行有符号或无符号（由 CORCON.US 定义）减法，然后求平方（见注 3）。结果经过符号扩展或零扩展至 72 位，然后写入指定的累加器。此外，小数结果也会在累加器更新之前进行换算，以对齐操作数和累加器（msw）小数点。
当 X 和 Y 地址空间中至少有一个选择间接寻址时，Wx 和 Wy 寄存器都会提供相应的间接地址。地址修改量值分别为 kx 和 ky，表示有效地址中要修改的数据字节数。

可选的 AWB 用于指定是直接还是间接（见注 4）存储非 ED 操作目标的累加器舍入后的小数内容（32 位）。舍入模式由 CORCON.RND 定义。写入数据宽度由所选指令数据大小决定（见注 5）。当 DSP 引擎工作于整数模式时，不适合使用 AWB。

读取的数据可以是 16 位或 32 位值。所有间接地址修改都会相应地进行换算。

L 位用于选择字或长字操作。

A 位用于选择存放结果的累加器。

I 位用于选择操作 X 空间寻址模式。

i 位用于选择 kx 修改值。

J 位用于选择操作 Y 空间寻址模式。

j 位用于选择 ky 修改值。

s 位用于选择 Wx 寄存器

w 位用于选择 Wy 寄存器。

a 位用于选择累加器回写目标和寻址模式。

注：

1. 在小数还是整数数据模式下工作由 CORCON.IF 定义。
2. 将操作数视为有符号还是无符号总是根据 CORCON.US 的状态来判断。
3. 如果源是执行前修改或执行后修改的有效地址，则不允许对间接源（X 或 Y）和 AWB 间接目标使用相同的 W 寄存器。
4. 堆栈必须保持长字对齐。因此，只允许将[W15++] AWB 与长字 MAC 类指令一起使用。

指令字数：1

指令周期数：2

EDAC	相减、平方并累加（求取欧几里德距离）				
语法：{标号}	EDAC{.w} Wx,	Wy,	A	{,AWB}	
	EDAC.l	[Wx],	[Wy],	B	
		[Wx]+=kx	, [Wy]+=ky,		
		[Wx]-=kx	, [Wy]-=ky,		
		[Wx+=kx]	, [Wy+=ky],		
		[Wx-=kx]	, [Wy-=ky],		
		[Wx+W12]	, [Wy+W12],		

操作数：

Wx ∈ {W0 ...W14}；Wy ∈ {W0 ...W14}
字模式：kx ∈ {-8, -6, -4, -2, 2, 4, 6, 8}；ky ∈ {-8, -6, -4, -2, 2, 4, 6, 8}；

长字模式：kx ∈ {-16, -12, -8, -4, 4, 8, 12, 16}；ky ∈ {-16, -12, -8, -4, 4, 8, 12, 16}；

AWB ∈ {W0, W1, W2, W3, W13, [W13++], [W15++][4]}

操作：

ACC(A 或 B) + ((Wx) - (Wy))^2 → ACC(A 或 B)；
舍入后的(ACC(B 或 A)) → AWB

当使用执行前修改/执行后修改的间接寻址时：

(Wx)+kx→Wx 或(Wx)-kx→Wx；

(Wy)+ky→Wy 或(Wy)-ky→Wy；

受影响的状态位：

OA 和 SA，*或者* OB 和 SB

指令编码：

```
1101        10AL        wwww        ssss        IIIi        iJJJ      jjaa      a111
```

说明：

用于计算(A-B)^2 函数的指令。对同时从 Wx 和 Wy 读取的数据或同时从 X 和 Y 地址空间获取的数据进行有符号或无符号（由 CORCON.US 定义）减法，然后求平方。结果经过符号扩展或零扩展至 72 位，然后加到指定的累加器。结果在累加器更新之前是否进行换算取决于是小数运算还是整数运算（由 CORCON.IF 定义）。小数运算将对结果进行换算以对齐操作数和累加器（msw）小数点（见注 3）。整数运算会将结果的 LSb 与累加器的 LSb 对齐。
当 X 和 Y 地址空间中至少有一个选择间接寻址时，Wx 和 Wy 寄存器都会提供相应的间接地址。地址修改量值分别为 kx 和 ky，表示有效地址中要修改的数据字数。

可选的 AWB 用于指定是直接还是间接（见注 4）存储非 EDAC 操作目标的累加器舍入后的小数内容（32 位）。舍入模式由 CORCON.RND 定义。写入数据宽度由所选指令数据大小决定（见注 5）。当 DSP 引擎工作于整数模式时，不适合使用 AWB。

读取的数据可以是 16 位或 32 位值。所有间接地址修改都会相应地进行换算。

L 位用于选择字或长字操作。

A 位用于选择存放结果的累加器。

I 位用于选择操作 X 空间寻址模式。

i 位用于选择 kx 修改值。

J 位用于选择操作 Y 空间寻址模式。

j 位用于选择 ky 修改值。

s 位用于选择 Wx 寄存器。

w 位用于选择 Wy 寄存器。

a 位用于选择累加器回写目标和寻址模式。

注：

1. 在小数还是整数数据模式下工作由 CORCON.IF 定义。
2. 将操作数视为有符号还是无符号总是根据 CORCON.US 的状态来判断。
3. 当在小数模式下使用字大小的数据时，ACCx 的低位部分不受影响。因此，之前的（32 位数据）操作中可能存在的低位数据会得以保留。如果不需要该数据，用户应在初始化阶段清零 ACCx。
4. 如果源是执行前修改或执行后修改的有效地址，则不允许对间接源（X 或 Y）和 AWB 间接目标使用相同的 W 寄存器。
5. 堆栈必须保持长字对齐。因此，只允许将[W15++] AWB 与长字 MAC 类指令一起使用。

指令字数：1

指令周期数：2

EXCH		交换 Ws 和 Wd		
语法：	{标号:}	EXCH	Wns,	Wnd

操作数： Wns ∈ [W0 ...W15]；Wnd ∈ [W0 ...W15]

操作： (Wns) ↔ (Wnd)

受影响的状态位： 无

指令编码：

1000	001U	dddd	ssss	UUUU	UUUU	UUUU	0100

说明： 该指令用于交换两个工作寄存器的内容，具体操作分为两个周期。在第一个周期中，读取 Wnd 并将其传送至 Wns。在第二个周期中，读取 Wns 并将其传送至 Wnd。
s 位用于选择 Wns 寄存器。

d 位用于选择 Wnd 寄存器。

注：

1. 使用子操作码与 MOV（32 位形式）共用操作码。
2. 假定是长字操作。
3. 尽管操作数顺序对最终结果没有影响，但可能会影响对危险的检测，因为各个操作数在不同的周期内进行读取。

指令字数： 1

指令周期数： 2

FBCL		搜索自左起第一个位变化		
语法：	{标号:}	FBCL{.w}	Ws,	Wnd
		FBCL.l	[Ws],	
			[Ws++],	
			[Ws--],	
			[++Ws],	
			[--Ws],	

操作数： Ws ∈ [W0 ...W15]；Wnd ∈ [W0 ...W14]

操作： 请参见指令说明

受影响的状态位： C

指令编码：

1110	011L	dddd	ssss	pppU	UUUU	UUUU	1011

说明： 按照从字操作数符号位后的 MSb 至 LSb 的顺序，搜索第一个出现的 1（对于有符号正值）或 0（对于有符号负值）。位编号经过符号扩展为 32 位并写入目标寄存器。
紧邻符号位之后的 MSb 被指定为位编号 0。对于字操作，LSb 被指定为位编号-14，结果为-15（C = 1）表示未找到该位。对于长字操作，LSb 被指定为位编号-30，结果为-31（C = 1）表示未找到该位。

所有非零结果都会清零 C。

L 位用于选择字或长字操作。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

关于修改量寻址信息，请参见 和 。

注： 该指令只能工作于字和长字模式。

指令字数： 1

指令周期数： 1

FBRA EQ		协处理器转移 0（如果等于则 CP0 FPU 转移）		
语法：	{标号:}	FBRA	EQ,	Expr

操作数： 标号由链接器解析为有符号字偏移量（slit20）

.....（续）

FBRA EQ	协处理器转移 0（如果等于则 CP0 FPU 转移）

操作：	条件 = FSR.EQ；
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移

受影响的状态位：	无
指令编码：	1011 0000 nnnn nnnn nnnn nnnn nnnn zz10
说明：	如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。 如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。 n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。 z 位用于选择目标协处理器。 **注：** 1. 对于浮点协处理器，协处理器选择位域 zz = 2'b00。 2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。 3. 转移条件在协处理器内部进行评估。
指令字数：	1
指令周期数：	1（2 或 3）

FBRA GE	协处理器转移 6（如果大于或等于则 CP0 FPU 转移）
语法：	{标号:} FBRA GE, Expr
操作数：	Expr 由链接器解析为有符号字偏移量（slit20）

操作：	条件 = (FSR.GT || FSR.EQ)；
	如果条件为真，则{
	如果 slit20 = 1，则跳过下一条（16 位）指令
	如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
	否则(PC+4) + 2*slit20 → PC
	}
	否则无转移

受影响的状态位：	无
指令编码：	1011 0110 nnnn nnnn nnnn nnnn nnnn zz10

.....（续）

FBRA GE	协处理器转移 6（如果大于或等于则 CP0 FPU 转移）

说明：

如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

z 位用于选择目标协处理器。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
3. 转移条件在协处理器内部进行评估。

指令字数：1

指令周期数：1（2 或 3）

FBRA GT	协处理器转移 4（如果大于则 CP0 FPU 转移）

语法：{标号:} FBRA GT, Expr

操作数：Expr 由链接器解析为有符号字偏移量（slit20）

操作：

条件 = FSR.GT；
如果条件为真，则{
如果 slit20 = 1，则跳过下一条（16 位）指令
如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
否则(PC+4) + 2*slit20 → PC
}
否则无转移

受影响的状态位：无

指令编码：

1011	0100	nnnn	nnnn	nnnn	nnnn	nnnn	zz10

.....（续）

FBRA GT	协处理器转移 4（如果大于则 CP0 FPU 转移）

说明：

如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

z 位用于选择目标协处理器。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
3. 转移条件在协处理器内部进行评估。

指令字数：1

指令周期数：1（2 或 3）

FBRA LE	协处理器转移 10（如果小于或等于则 CP0 FPU 转移）

语法：{标号:} FBRA LE, Expr

操作数：Expr 由链接器解析为有符号字偏移量（slit20）

操作：
条件 = (FSR.LT || FSR.EQ)；
如果条件为真，则{

如果 slit20 = 1，则跳过下一条（16 位）指令

如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令

否则(PC+4) + 2*slit20 → PC

}

否则无转移

受影响的状态位：无

指令编码：

1011	1010	nnnn	nnnn	nnnn	nnnn	nnnn	zz10

.....（续）

FBRA LE	协处理器转移 10（如果小于或等于则 CP0 FPU 转移）

说明：

如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

z 位用于选择目标协处理器。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
3. 转移条件在协处理器内部进行评估。

指令字数：1

指令周期数：1（2 或 3）

FBRA LT	协处理器转移 8（如果小于则 CP0 FPU 转移）

语法：{标号:} FBRA LT, Expr

操作数：Expr 由链接器解析为有符号字偏移量（slit20）

操作：

条件 = FSR.LT；
如果条件为真，则{

如果 slit20 = 1，则跳过下一条（16 位）指令

如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令

否则(PC+4) + 2*slit20 → PC

}

否则无转移

受影响的状态位：无

指令编码：

1011	1000	nnnn	nnnn	nnnn	nnnn	nnnn	zz10

.....（续）

FBRA LT	协处理器转移 8（如果小于则 CP0 FPU 转移）

说明：

如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

z 位用于选择目标协处理器。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
3. 转移条件在协处理器内部进行评估。

指令字数：1

指令周期数：1（2 或 3）

FBRA NE	协处理器转移 2（如果不等于则 CP0 FPU 转移）

语法：{标号:} FBRA NE, Expr

操作数：Expr 由链接器解析为有符号字偏移量（slit20）

操作：

条件 = (FSR.GT || FSR.LT)；
如果条件为真，则{

如果 slit20 = 1，则跳过下一条（16 位）指令

如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令

否则(PC+4) + 2*slit20 → PC

}

否则无转移

受影响的状态位：无

指令编码：

1011	0010	nnnn	nnnn	nnnn	nnnn	nnnn	zz10

.....（续）

FBRA NE	协处理器转移 2（如果不等于则 CP0 FPU 转移）

说明：

如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

z 位用于选择目标协处理器。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
3. 转移条件在协处理器内部进行评估。

指令字数：1

指令周期数：1（2 或 3）

FBRA OR	协处理器转移 12（如果有序则 CP0 FPU 转移）

语法：{标号:} FBRA OR, Expr

操作数：Expr 由链接器解析为有符号字偏移量（slit20）

操作：

条件 = (FSR.GT || FSR.LT || FSR.EQ)
如果条件为真，则{

如果 slit20 = 1，则跳过下一条（16 位）指令

如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令

否则(PC+4) + 2*slit20 → PC

}

否则无转移

受影响的状态位：无

指令编码：

1011	1100	nnnn	nnnn	nnnn	nnnn	nnnn	zz10

.....（续）

FBRA OR	协处理器转移 12（如果有序则 CP0 FPU 转移）

说明：

如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

z 位用于选择目标协处理器。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
3. 转移条件在协处理器内部进行评估。

指令字数：1

指令周期数：1（2 或 3）

FBRA UGE	协处理器转移 9（如果无序、大于或等于则 CP0 FPU 转移）

语法：{标号:} FBRA UGE, Expr

操作数：Expr 由链接器解析为有符号字偏移量（slit20）

操作：

条件 = (FSR.GT || FSR.EQ || FSR.UN)；
如果条件为真，则{

如果 slit20 = 1，则跳过下一条（16 位）指令

如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令

否则(PC+4) + 2*slit20 → PC

}

否则无转移

受影响的状态位：无

指令编码：

1011	1001	nnnn	nnnn	nnnn	nnnn	nnnn	zz10

.....（续）

FBRA UGE	协处理器转移 9（如果无序、大于或等于则 CP0 FPU 转移）

说明：

如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

z 位用于选择目标协处理器。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
3. 转移条件在协处理器内部进行评估。

指令字数：1

指令周期数：1（2 或 3）

FBRA UGT	协处理器转移 11（如果无序或大于则 CP0 FPU 转移）

语法：{标号:} FBRA UGT, Expr

操作数：Expr 由链接器解析为有符号字偏移量（slit20）

操作：

条件 = (FSR.GT || FSR.UN);
如果条件为真，则{

如果 slit20 = 1，则跳过下一条（16 位）指令

如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令

否则(PC+4) + 2*slit20 → PC

}

否则无转移

受影响的状态位：无

指令编码：

1011	1011	nnnn	nnnn	nnnn	nnnn	nnnn	zz10

.....（续）

FBRA UGT	协处理器转移 11（如果无序或大于则 CP0 FPU 转移）

说明：

如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

z 位用于选择目标协处理器。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
3. 转移条件在协处理器内部进行评估。

指令字数：1

指令周期数：1（2 或 3）

FBRA UEQ	协处理器转移 3（如果无序或等于则 CP0 FPU 转移）

语法：{标号:} FBRA UEQ, Expr

操作数：Expr 由链接器解析为有符号字偏移量（slit20）

操作：

条件 = (FSR.EQ || FSR.UN)；
如果条件为真，则{

如果 slit20 = 1，则跳过下一条（16 位）指令

如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令

否则(PC+4) + 2*slit20 → PC

}

否则无转移

受影响的状态位：无

指令编码：

1011	0011	nnnn	nnnn	nnnn	nnnn	nnnn	zz10

.....（续）

FBRA UEQ	协处理器转移 3（如果无序或等于则 CP0 FPU 转移）

说明：

如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

z 位用于选择目标协处理器。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
3. 转移条件在协处理器内部进行评估。

指令字数：1

指令周期数：1（2 或 3）

FBRA ULE	协处理器转移 5（如果无序、小于或等于则 CP0 FPU 转移）

语法：{标号:} FBRA ULE, Expr

操作数：Expr 由链接器解析为有符号字偏移量（slit20）

操作：
条件 = (FSR.LT || FSR.EQ || FSR.UN)；
如果条件为真，则{

如果 slit20 = 1，则跳过下一条（16 位）指令

如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令

否则(PC+4) + 2*slit20 → PC

}

否则无转移

受影响的状态位：无

指令编码：

1011	0101	nnnn	nnnn	nnnn	nnnn	nnnn	zz10

.....（续）

FBRA ULE	协处理器转移 5（如果无序、小于或等于则 CP0 FPU 转移）

说明：

如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

z 位用于选择目标协处理器。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
3. 转移条件在协处理器内部进行评估。

指令字数：1

指令周期数：1（2 或 3）

FBRA ULT	协处理器转移 7（如果无序或小于则 CP0 FPU 转移）

语法：{标号:} FBRA ULT, Expr

操作数：Expr 由链接器解析为有符号字偏移量（slit20）

操作：

条件 = (FSR.LT || FSR.UN)；
如果条件为真，则{

如果 slit20 = 1，则跳过下一条（16 位）指令

如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令

否则(PC+4) + 2*slit20 → PC

}

否则无转移

受影响的状态位：无

指令编码：

1011	0111	nnnn	nnnn	nnnn	nnnn	nnnn	zz10

.....（续）

FBRA ULT	协处理器转移 7（如果无序或小于则 CP0 FPU 转移）

说明：

如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

z 位用于选择目标协处理器。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
3. 转移条件在协处理器内部进行评估。

指令字数：1

指令周期数：1（2 或 3）

FBRA UN	协处理器转移 13（如果无序则 CP0 FPU 转移）

语法：{标号:} FBRA UN, Expr

操作数：Expr 由链接器解析为有符号字偏移量（slit20）

操作：

条件 = FSR.UN；
如果条件为真，则{
如果 slit20 = 1，则跳过下一条（16 位）指令
如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令
否则(PC+4) + 2*slit20 → PC
}
否则无转移

受影响的状态位：无

指令编码：

1011	1101	nnnn	nnnn	nnnn	nnnn	nnnn	zz10

.....（续）

FBRA UN	协处理器转移 13（如果无序则 CP0 FPU 转移）

说明：

如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

z 位用于选择目标协处理器。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
3. 转移条件在协处理器内部进行评估。

指令字数：1

指令周期数：1（2 或 3）

FBRA UNE	协处理器转移 1（如果无序或不等于则 CP0 FPU 转移）

语法：{标号:} FBRA UNE, Expr

操作数：标号由链接器解析为有符号字偏移量（slit20）

操作：
条件 = (FSR.GT || FSR.LT || FSR.UN)；
如果条件为真，则{

如果 slit20 = 1，则跳过下一条（16 位）指令

如果(slit20 = 2 && next_op[31] = 1)，则跳过下一条（32 位）指令

否则(PC+4) + 2*slit20 → PC

}

否则无转移

受影响的状态位：无

指令编码：

1011	0001	nnnn	nnnn	nnnn	nnnn	nnnn	zz10

.....（续）

FBRA UNE	协处理器转移 1（如果无序或不等于则 CP0 FPU 转移）

说明：

如果满足转移条件，则指令将跳过下一条 16 位或 32 位指令，或者转移至向前或向后 1 MB 范围内任意大小的指令。
如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 2，则执行条件转移时将按照条件跳过一条 16 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果二进制补码字节偏移量值“2*slit20”（PC 偏移量）等于 4，并且转移后的指令为 32 位操作码（见注释），则执行条件转移时将按照条件跳过相应的 32 位指令。该指令将被推测执行，直到可以确定转移决策时才跳过。

如果不满足条件跳过的要求，则二进制补码字节偏移量值“2*slit20”将加到 PC 以创建新的 PS 字地址。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。

转移预测基于转移的方向。如果向后转移（负），预测将转移。如果向前转移（正），预测不会转移。之后，将在预测路径中推测执行转移后的两条指令，直到可以评估实际转移决策时为止。预测正确的转移将在同一路径中继续正常执行指令。预测错误的转移将中止推测执行的指令，并从正确的路径开始执行。

n 位构成一个有符号立即数，用于指定自(PC + 4)的偏移量（PS 字数）。

z 位用于选择目标协处理器。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 如果字节偏移量等于 4 且转移后的指令为 16 位操作码，则满足条件时将发生转移（跳过接下来的两个 16 位操作码）。
3. 转移条件在协处理器内部进行评估。

指令字数：1

指令周期数：1（2 或 3）

FF1L	搜索自左起第一个 1

语法：

{标号:}	FF1L{.w}	Ws,	Wnd
	FF1L.l	[Ws],	
		[Ws++],	
		[Ws--],	
		[++Ws],	
		[--Ws],	

操作数：Ws ∈ [W0 ...W15]；Wnd ∈ [W0 ...W14]

操作：请参见指令说明

受影响的状态位：C

指令编码：

1110	011L	dddd	ssss	pppU	UUUU	UUUU	0011

说明：

按照字操作数 MSb 至 LSb 的顺序，搜索第一个出现的 1。位编号结果经过零扩展为 32 位并写入目标寄存器。
MSb 被指定为位编号 1。对于字操作，LSb 被指定为位编号 16，结果为零（C = 1）表示未找到该位。对于长字操作，LSb 被指定为位编号 32，结果为零（C = 1）表示未找到该位。

所有非零结果都会清零 C。

L 位用于选择字或长字操作。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

注：该指令只能工作于字和长字模式。

指令字数：1

指令周期数：1

FF1R	搜索自右起第一个 1

语法：

{标号:}	FF1R{.w}	Ws,	Wnd

.....（续）

FF1R		搜索自右起第一个 1						
		FF1R.l	[Ws],					
			[Ws++],					
			[Ws--],					
			[++Ws],					
			[--Ws],					
操作数：	Ws ∈ [W0 ...W15]；Wnd ∈ [W0 ...W14]							
操作：	请参见指令说明							
受影响的状态位：	C							
指令编码：	1110	011L	dddd	ssss	pppU	UUUU	UUUU	0111
说明：	按照字操作数 LSb 至 MSb 的顺序，搜索第一个出现的 1。位编号结果经过零扩展为 32 位并写入目标寄存器。 LSb 被指定为位编号 1。对于字操作，MSb 被指定为位编号 16，结果为零（C = 1）表示未找到该位。对于长字操作，MSb 被指定为位编号 32，结果为零（C = 1）表示未找到该位。 所有非零结果都会清零 C。 L 位用于选择字或长字操作。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 **注：**该指令只能工作于字和长字模式。							
指令字数：	1							
指令周期数：	1							

FLIM		强制执行（有符号）数据范围限制						
语法：	{标号:}	FLIM{.w}	Wb,	Ws,				
		FLIM.l		[Ws],				
				[Ws++],				
				[Ws--],				
				[++Ws],				
				[--Ws],				
操作数：	Ws ∈ [W0 ...W14]；Wb ∈ [W0 ...W13]；							
操作：	如果(Ws) > (Wb)，则{ (Wb) → (Ws)； 0→ Z；0→ N；0→ OV } 如果(Ws) < (Wb+1)，则{ (Wb+1) → Ws； 0→ Z；1→ N；0→ OV } 否则{ (Ws) → Ws^4； 1→ Z；0→ N；0→ OV }							
受影响的状态位：	N、Z 和 OV							
指令编码：	1110	010L	UUUU	ssss	pppU	UUww	wwUU	0011

.....（续）

FLIM	强制执行（有符号）数据范围限制

说明：

将 Ws 中任意大小的有符号数据值同时与 Wb 中保存的有符号上限值和 Wb+1 中保存的有符号下限值进行比较。

对于长字数据大小，如果 Ws[31:0]大于 Wb[31:0]，则将 Ws[31:0]设置为 Wb[31:0]中保存的限值。对于字数据大小，如果 Ws[15:0]大于 Wb[15:0]，则将 Ws[15:0]设置为 Wb[15:0]中保存的限值（忽略 Ws[31:16]）。Z、N 和 OV 状态位会更新，这样后续 BGT 指令将执行转移。

对于长字数据大小，如果 Ws[31:0]小于 Wb+1[31:0]，则将 Ws[31:0]设置为 Wb+1[31:0]中保存的限值。对于字数据大小，如果 Ws[15:0]小于 Wb+1[15:0]，则将 Ws[15:0]设置为 Wb+1[15:0]中保存的限值（忽略 Ws[31:16]）。Z、N 和 OV 状态位会更新，这样后续 BLT 指令将执行转移。

如果 Ws[31:0]小于或等于 Wb[31:0]中的上限值并且大于或等于 Wb+1[31:0]中的下限值（对于长字大小的数据），或者 Ws[15:0]小于或等于 Wb[15:0]中的上限值并且大于或等于 Wb+1[15:0]中的下限值（对于字大小的数据），则不应用任何数据限制，这样 Ws 的内容将不会改变（对于给定的数据大小）。Z 状态位会置 1，这样后续 BZ 指令将执行转移（OV 和 N 状态位均清零）。

对于字大小的寄存器直接寻址操作，Ws 总是用零扩展至 32 位（即使没有应用任何数据限制也如此）[4]。

请注意，该指令总是先与上限值进行比较。如果 Ws 小于上限值，再与下限值进行比较，如果 Ws 不小于下限值，则 Ws 将保持不变。

此外，即使上限值 Wb 被意外设置为小于或等于下限值 Wb+1，也仍将按照上述顺序进行比较。不会检测到该条件。

该指令总是将 OV 状态位清零。

s 位用于选择源（数据值）寄存器。

w 位用于选择基准（数据限制）寄存器。

p 位用于选择源寻址模式。

注：

1. 尽管该指令假定所有操作数都是有符号值，但上限值与下限值的符号可能相同。
2. 根据数据比较结果设置状态位。
3. 通过 FLIM{.w}写入 Ws 的字数据经过零扩展至 32 位。
4. 经过零扩展至 32 位与任何寄存器直接寻址字写操作一致。但是，即使没有超出限制，写入 Ws 也总是可能造成 RAW 危险。

指令字数：1

指令周期数：1

FLIM	强制执行（有符号）数据范围限制，保存超限结果

语法：

{标号:}	FLIM{.w}{v} Wb,	Ws,	Wd
	FLIM.l{v}	[Ws],	[Wd]
		[Ws++],	[Wd++]
		[Ws--],	[Wd--]
		[++Ws],	[++Wd]
		[--Ws],	[--Wd]

操作数：Ws ∈ [W0 ...W14]；Wb ∈ [W0 ...W13]；Wnd ∈ [W0 ...W14]

操作：

如果(Ws) > (Wb)，则(如果为 FLIM.v，则(Ws) - (Wb) → Wnd，否则+1 → Wnd；(Wb) → (Ws)；
0→ Z；0→ N；0→ OV；
)

如果(Ws) < (Wb+1)，则(如果为 FLIM.v，则(Ws) - (Wb+1) → Wnd，否则-1 → Wnd；(Wb+1) → Ws；
0→ Z；1→ N；0→ OV；
)
否则(
0 → Wnd；
(Ws) → Ws[4]；
1→ Z；0→ N；0→ OV；
)

.....（续）

FLIM	强制执行（有符号）数据范围限制，保存超限结果

受影响的状态位：N、Z 和 OV

指令编码：

1110	010L	dddd	ssss	pppq	qqww	wwUV	0111

说明：

将 Ws 中任意大小的有符号数据值同时与 Wb 中保存的有符号上限值和 Wb+1 中保存的有符号下限值进行比较。将超限值写入 Wnd。
对于长字数据大小，如果 Ws[31:0]大于 Wb[31:0]，则将 Ws[31:0]设置为 Wb[31:0]中保存的限值。对于字数据大小，如果 Ws[15:0]大于 Wb[15:0]，则将 Ws[15:0]设置为 Wb[15:0]中保存的限值（忽略 Ws[31:16]）。

在所有情况下，将（有符号）超限值写入 Wnd[31:0]（FLIM.v，其中操作码位字段 V = 1）或将 Wnd 设置为+1（FLIM，其中操作码位字段 V = 0）。每当 Ws 大于 Wb 时，Wnd 将总是为正值。Z、N 和 OV 状态位会更新，这样后续 BGT 指令将执行转移。

对于长字数据大小，如果 Ws[31:0]小于 Wb+1[31:0]，则将 Ws[31:0]设置为 Wb+1[31:0]中保存的限值。对于字数据大小，如果 Ws[15:0]小于 Wb+1[15:0]，则将 Ws[15:0]设置为 Wb+1[15:0]中保存的限值（忽略 Ws[31:16]）。在所有情况下，将（有符号）超限值写入 Wnd[31:0]（FLIM.v，其中操作码位字段 V = 1）或将 Wnd 设置为+1（FLIM，其中操作码位字段 V = 0）。每当 Ws 小于 Wb+1 时，Wnd 将总是为负值。Z、N 和 OV 状态位会更新，这样后续 BLT 指令将执行转移。

如果 Ws[31:0]小于或等于 Wb[31:0]中的上限值并且大于或等于 Wb+1[31:0]中的下限值（对于长字大小的数据），或者 Ws[15:0]小于或等于 Wb[15:0]中的上限值并且大于或等于 Wb+1[15:0]中的下限值（对于字大小的数据），则不应用任何数据限制，这样 Ws 的内容将不会改变（对于给定的数据大小）。Z 状态位会置 1，这样后续 BZ 指令将执行转移（OV 和 N 状态位均清零）。
对于字大小的寄存器直接寻址操作，Ws 总是经过零扩展至 32 位（即使没有应用任何数据限制也如此）[4]。

请注意，该指令总是先与上限值进行比较。如果 Ws 小于上限值，再与下限值进行比较，如果 Ws 不小于下限值，则 Ws 将保持不变。
此外，即使上限值 Wb 被意外设置为小于或等于下限值 Wb+1，也仍将按照上述顺序进行比较。不会检测到该条件。

该指令总是将 OV 状态位清零。

s 位用于选择源（数据值）寄存器。

w 位用于选择基准（数据限制）寄存器。

d 位用于选择目标（限制测试结果）寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

V 位用于选择 Wnd 的结果格式。

关于修改量寻址信息，请参见 。

注：

1. 尽管该指令假定所有操作数都是有符号值，但上限值与下限值的符号可能相同。
2. 根据装入 Wnd 的值设置状态位。
3. 通过 FLIM{.w}写入 Ws（和通过 FLIM{.w}v 写入 Wnd）的字数据经过零扩展至 32 位。
4. 经过零扩展至 32 位与任何寄存器直接寻址字写操作一致。但是，即使没有超出限制，写入 Ws 也总是可能造成 RAW 危险。
5. Ws 与 Wd 不能是同一个寄存器。

指令字数：1

指令周期数：2

GOTO	无条件跳转

语法：{标号:} GOTO lit24

操作数：lit24 ∈ [0 ...16 MB]

操作：lit24 → PC[23:0]，NOP → 指令寄存器。

受影响的状态位：无

指令编码：

1101	101n	nnnn	nnnn	nnnn	nnnn	nnnn	nn10

.....（续）

GOTO	无条件跳转

说明：无条件跳转到一个周期（对应于延时）的存储空间内的任何地址。24 位值“lit24”装入 PC。允许跳转到 32 位或 16 位指令。
n 位形成目标地址。

注：（字节）PC 地址总是为字对齐或长字对齐。操作码不存储 LSb，因为该位总是为 1'b0。

指令字数：1

指令周期数：1

GOTO	扩展无条件跳转

语法：{标号:} CALL lit32

操作数：lit32 ∈ [16 MB[1] ...4 GB]

操作：lit32 → PC[31:0];
NOP → 指令寄存器

受影响的状态位：无

指令编码：

第一个字	1111	010U	nnnn	nnnn	nnnn	nnnn	nnn0	1011
第二个字	1111	010U	UUUU	Unnn	nnnU	UUnn	nnnn	1111

说明：无条件跳转到一个周期（对应于延时）的存储空间内的任何地址。32 位值“lit32”装入 PC。允许跳转到 32 位或 16 位指令。
n 位形成目标地址。

PC[31:0] = Word1[18:13]、Word2[9:4]和 Word1[23:4]

注：（字节）PC 地址总是为字对齐或长字对齐，使 LSb 总是为 1'b0。

指令字数：2

指令周期数：2

GOTO	无条件间接跳转

语法：{标号:} GOTO Wns

操作数：Wns ∈ [W0 ...W14]

操作：(Wns) → PC[23:0];
NOP → 指令寄存器。

受影响的状态位：无

指令编码：

1101	011U	UUUU	ssss	UUUU	UUUU	UUUU	1010

说明：无条件间接跳转到可执行存储器地址空间内的任何地址。
Wns[23:0]的内容装入 PC[23:0]。因此，Wns 必须包含 PS 字节地址。

Wns[0]的值会被忽略，PC[0]总是设置为 1'b0。

GOTO 是一条双周期指令。

s 位用于选择源寄存器。

如果 Wns[31:24] != 8'h00，将启动地址错误陷阱。

指令字数：1

指令周期数：2

注：如果有异常等待处理，则不会执行取出的跳转目标指令，从而使有效指令执行时间为一个周期。

INC	f 内容递增 1

语法：{标号:} INC.b f {,Wnd} {,WREG}
INC.bz
INC{.w}

.....（续）

INC		f 内容递增 1						
		INC.l						
操作数：	f ∈ [0 ...64 KB]；Wnd ∈ [W0 ...W14]							
操作：	(f) + 1 → 由 D 指定的目标寄存器							
受影响的状态位：	C、N、OV 和 Z							
指令编码：	1101	010L	dddd	ffff	ffff	ffff	ffff	BD01
说明：	将文件寄存器中的内容加 1，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wnd，则 D = 0 并将结果存放在 Wnd 中；否则，D = 1 并将结果存放在文件寄存器中。 L 和 B 位用于选择操作数据宽度。 D 位用于选择目标寄存器。 f 位用于选择文件寄存器的地址。 d 位用于选择工作寄存器。							
指令字数：	1							
指令周期数：	1							

INC2		f 内容递增 2			
语法：	{标号:}	INC2.b INC2.bz INC2{.w} INC2.l	f	{,Wnd}	{,WREG}
操作数：	f ∈ [0 ...64 KB]；Wnd ∈ [W0 ...W14]				
操作：	(f) + 2 → 由 D 指定的目标寄存器				
受影响的状态位：	C、N、OV 和 Z				
指令编码：	1101	110L	dddd	ffff	ffff	ffff	ffff	BD01
说明：	将文件寄存器中的内容加 2，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wnd，则 D = 0 并将结果存放在 Wnd 中；否则，D = 1 并将结果存放在文件寄存器中。 L 和 B 位用于选择操作数据宽度。 D 位用于选择目标寄存器。 f 位用于选择文件寄存器的地址。 d 位用于选择工作寄存器。							
指令字数：	1							
指令周期数：	1							

IOR		Wb 和 Ws 逻辑或						
语法：	{标号:}	IOR.b IOR.bz IOR{.w} IOR.l	Wb,	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws], SR	Wd [Wd] [Wd++] [Wd--] [++Wd] [--Wd] SR			
操作数：	Wb ∈ [W0 ...W14]；Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]							
操作：	(Wb).IOR.(Ws) → Wd							
受影响的状态位：	N 和 Z							
指令编码：	S110	101L	dddd	ssss	pppq	qqww	wwUU	BU00

.....（续）

IOR	Wb 和 Ws 逻辑或

说明：

将源寄存器 Ws 中的内容和基准寄存器 Wb 中的内容进行逻辑或运算，并将结果存放在目标寄存器 Wd 中。

S 位用于选择指令大小。

L 和 B 位用于选择操作数据宽度。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

注：

1. 选择 SR 时，SR 数据写操作将优先于由 IOR 运算引起的任何 SR 更新操作。
2. 写入 SR 时，不允许使用.bz 数据大小/模式。

指令字数：1 或 0.5

指令周期数：1

IOR	Wb 和短立即数逻辑或

语法：

{标号:}	IOR.b IOR.bz IOR{.w} IOR.l	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws], SR	lit7,	Wd [Wd] [Wd++] [Wd--] [++Wd] [--Wd] SR

操作数：Ws ∈ [W0 ...W15]；lit7 ∈ [0 ...127]；Wd ∈ [W0 ...W15]

操作：(Ws).IOR.lit7 → Wd

注：立即数经过零扩展至操作所选的数据大小

受影响的状态位：N 和 Z

指令编码：

1110	101L	dddd	ssss	pppq	qqkk	kkkk	Bk10

说明：

将源寄存器 Ws 中的内容和零扩展立即数操作数进行逻辑或运算，并将结果存放在目标寄存器 Wd 中。

L 和 B 位用于选择操作数据宽度。

k 位用于提供立即数操作数（op[2]中的 MSb）。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

关于修改量寻址信息，请参见 和 。

注：

1. 选择 SR 时，SR 数据写操作将优先于由 AND 运算引起的任何 SR 更新操作。
2. 写入 SR 时，不允许使用.bz 数据大小/模式。

指令字数：1

指令周期数：1

IOR	立即数和 Wn 逻辑或

语法：

{标号:}	IOR.b IOR.bz IOR{.w} IOR.l	lit16,	Wn SR

.....（续）

IOR	立即数和 Wn 逻辑或						
操作数：	lit16 ∈ [0 ...65535]；Wn ∈ [W0 ...W14]；状态寄存器（SR）						
操作：	(Wn) .IOR. lit16[3] → Wn 或(SR) .IOR. lit16 → SR						
受影响的状态位：	N 和 Z						
指令编码：	1100	101L	ssss	kkkk	kkkk	kkkk	kkkk	BT10
说明：	将立即数操作数和工作寄存器 Wn 或 SR 中的内容进行逻辑或运算，并将结果存放在工作寄存器 Wn 或 SR 中。 L 和 B 位用于选择操作数据宽度。 s 位用于选择工作寄存器。 k 位用于指定立即数操作数。 T 位用于在 Ws（T = 0）与 SR（T = 1）目标寄存器之间进行选择。 **注：** 1. 选择 SR 时，SR 数据写操作将优先于由 AND 运算引起的任何 SR 更新操作。 2. 写入 SR 时，不允许使用.bz 数据大小/模式。 3. 对于长字操作，立即数经过零扩展至 32 位。						
指令字数：	1						
指令周期数：	1						

IOR	f 和 Wn 逻辑或				
语法：	{标号:}	IOR.b IOR.bz IOR{.w} IOR.l	f	,Wn	{,WREG}
操作数：	f ∈ [0 ...64 KB]；Wn ∈ [W0 ...W14]				
操作：	(f).IOR.(Wn) → 由 D 指定的目标寄存器				
受影响的状态位：	N 和 Z				
指令编码：	1110	101L	ssss	ffff	ffff	ffff	ffff	BD01
说明：	将工作寄存器中的内容和文件寄存器中的内容进行逻辑或运算，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wn，则 D = 0 并将结果存放在 Wn 中；否则，D = 1 并将结果存放在文件寄存器中。 L 和 B 位用于选择操作数据宽度。 D 位用于选择目标寄存器。 f 位用于选择文件寄存器的地址。 s 位用于选择工作寄存器。				
指令字数：	1				
指令周期数：	1				

LAC	装载累加器				
语法：	{标号:}	LAC{.w} LAC.l	Ws, [Ws], [Ws++] [Ws--] [--Ws], [++Ws], [Ws+Wb],	{ Slit6, }	A B
操作数：	Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]；Slit6 ∈ [-32 ...+31]				

.....（续）

LAC	装载累加器							
操作：	对于字操作： 8{Ws[15]}、Ws[15:0]和 48'h000000000000 → ACC[71:0] $Shift_{Slit6}$(ACC) → ACC 对于长字操作： 8{Ws[31]}、Ws[31:0]和 32'h00000000 → ACC[71:0] $Shift_{Slit6}$(ACC) → ACC							
受影响的状态位：	OA 和 SA，*或者* OB 和 SB							
指令编码：	1100	00AL	wwww	ssss	pppU	UUkk	kkkk	0011
说明：	读取有效地址的内容。将读取的数据放入目标累加器，然后可选择将整个累加器内容移位。 对于字操作(LAC{.w})，假定有效地址处包含的值为 Q1.15 有符号小数数据，并将其写入 ACCx[63:48]。ACCx[47:0]设置为全 0，ACCx[63:0]自动进行符号扩展（通过 bit 71）。累加器内容随后进行算术移位，移位位数由 slit6 中保存的值指定。 对于长字操作（LAC.l），假定有效地址处包含的值为 Q1.31 有符号小数数据，并将其写入 ACCx[63:32]。ACCx[31:0]设置为全 0，ACCx[63:0]自动进行符号扩展（通过 bit 71）。累加器内容随后进行算术移位，移位位数由 slit6 中保存的值指定。 L 位用于选择字或长字操作。 A 位用于指定目标累加器。 s 位用于指定源寄存器 Wns。 p 位用于选择源寻址模式。 w 位用于指定偏移量寄存器 Wb。 k 位用于对可选操作数 Slit6 进行编码，该操作数确定累加器内容移位的位数；如果操作数 Slit6 不存在，则立即数位域设置为全 0。 **注：**操作数 Slit6 为正值表示算术右移。操作数 Slit6 为负值表示左移。							
指令字数：	1							
指令周期数：	1							

LLAC	装载累加器低位				
语法：	{标号:}	LLAC.l	Ws, [Ws], [Ws++] [Ws--] [--Ws], [++Ws], [Ws+Wb],	{ Slit6, }	A B
操作数：	Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]；Slit6 ∈ [-32 ...+31]				
操作：	Ws[31:0] → ACC[31:0] $Shift_{Slit6}$(ACC) → ACC				
受影响的状态位：	OA 和 SA，*或者* OB 和 SB				
指令编码：	1100	00A1	wwww	ssss	pppU	UUkk	kkkk	0111
说明：	读取有效地址的内容。将读取的数据放入目标累加器的低 32 位（ACCx[31:0]），然后可选择将整个累加器内容进行算术移位。 A 位用于指定目标累加器。 s 位用于指定源寄存器 Wns。 p 位用于选择源寻址模式。 w 位用于指定偏移量寄存器 Wb。 k 位用于对可选操作数 Slit6 进行编码，该操作数确定累加器内容移位的位数；如果操作数 Slit6 不存在，则立即数位域设置为全 0。 **注：**操作数 Slit6 为正值表示算术右移。操作数 Slit6 为负值表示左移。				

.....（续）

LLAC	装载累加器低位
指令字数：	1
指令周期数：	1

LNK	分配堆栈帧							
语法：	{标号:}	LNK	lit16					
操作数：	lit16 ∈ [0 ...65535]							
操作：	(W14) → (TOS) (W15) + 4 → W15 (W15) → W14； (W15) + lit16 → W15							
受影响的状态位：	无							
指令编码：	1101	010U	UUUU	kkkk	kkkk	kkkk	kkkk	1010
说明：	该指令用于分配大小为 lit16 字节的堆栈帧（尽管总是按长字对齐）并调整堆栈指针和帧指针。 k 位用于指定堆栈帧的大小（以字节为单位）。 **注：** 1. 该指令只能对按长字对齐的操作数进行操作。因此，立即数编码的低 2 位总是设置为 2'b00。							
指令字数：	1							
指令周期数：	1							

LNK	分配堆栈帧（短）			
语法：	{标号:}	LNK	lit7	
操作数：	lit7∈ [0 ... 127]			
操作：	(W14) → (TOS) (W15) + 4 → W15 (W15) → W14； (W15) + lit5 和 2'b00 → W15			
受影响的状态位：	无			
指令编码：	0111	001k	kkkk	1110
说明：	该指令用于分配大小为 lit7 字节的堆栈帧并调整堆栈指针和帧指针。 k 位用于指定堆栈帧的大小（以长字为单位）。 **注：** 1. 该指令只能对按长字对齐的操作数进行操作。因此，假定帧大小的低 2 位总是设置为 2'b00（即，只需要对操作码内的 5 位立即数进行编码）。			
指令字数：	0.5			
指令周期数：	1			

LSR	逻辑右移 1 位			
语法：	{标号:}	LSR.b LSR.bz LSR{.w} LSR.l	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	Wd [Wd] [Wd++] [Wd--] [++Wd] [--Wd]
操作数：	Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]			

.....（续）

LSR	逻辑右移 1 位

操作：

对于长字操作：
0 → Wd[31]，(Ws[31:1]) → Wd[30:0]，(Ws[0]) → C

对于字操作：

0 → Wd[15]，(Ws[15:1]) → Wd[14:0]，(Ws[0]) → C

对于字节操作：

0 → Wd[7]，(Ws[7:1]) → Wd[6:0]，(Ws[0]) → C

0 C

受影响的状态位：C、N 和 Z

指令编码：

S010	101L	dddd	ssss	pppq	qqUU	UUUU	B000

说明：

将源寄存器 Ws 中的内容逻辑右移一位，并将结果存放在目标寄存器 Wd 中。
目标寄存器直接寻址扩展字节或字模式会将结果零扩展至 32 位，然后写入 Wd。

S 位用于选择指令大小。

L 和 B 位用于选择操作数据宽度。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

注：

1. 仅当 Ws 的 MSb = 1 且移位值为零时，才能置 1 N 标志。

指令字数：1 或 0.5

指令周期数：1

LSR	逻辑右移 f

语法：

{标号:}	LSR.b LSR.bz LSR{.w} LSR.l	f	{,Wnd}	{,WREG}

操作数：f ∈ [0 ....64 KB]；Wnd ∈ [W0 ...W14]

操作：

对于字节操作：
0 → Dest[7:1]，(f[7:1]) → Dest[6:0]，(f[0]) → C

对于字操作：

0 → Dest[15]，(f[15:1]) → Dest[14:0]，(f[0]) → C

对于长字操作：0 → Dest[31]，(f[31:1]) → Dest[30:0]，(f[0]) → C

受影响的状态位：C、N 和 Z

指令编码：

1010	001L	dddd	ffff	ffff	ffff	ffff	BD01

说明：

将文件寄存器 f 中的内容右移一位，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wnd，则 D = 0 并将结果存放在 Wnd 中；否则，D = 1 并将结果存放在文件寄存器中。如果文件寄存器的 LSb 为 1，则进位标志位置 1。N 标志总是清零。
L 和 B 位用于选择操作数据宽度。

D 位用于选择目标寄存器。

f 位用于选择文件寄存器的地址。

d 位用于选择工作寄存器。

指令字数：1

指令周期数：1

LSR		逻辑右移（移位位数由短立即数确定）			
语法：	{标号:}	LSR{.w}	Ws,	lit5,	Wd
		LSR.l	[Ws],		[Wd]
			[Ws++],		[Wd++]
			[Ws--],		[Wd--]
			[++Ws],		[--Wd]
			[--Ws],		[++Wd]

操作数： Ws ∈ [W0 ...W15]；lit5 ∈ [0...32]；Wd ∈ [W0 ...W14]

操作：
lit5[4:0]→ Shift_Val
1'b0 → 右移输入（逻辑移位）
对于长字操作：
Ws[31:0]和 32'b0 → Shift_In[63:0]
将 Shift_In[63:0]逻辑右移 Shift_Val 位 → Shift_Out[63:0]
Shift_Out[63:32] → Wd
对于字操作：
16'b0、Ws[15:0]和 32'b0 → Shift_In[63:0]
将 Shift_In[63:0]逻辑右移 Shift_Val 位 → Shift_Out[63:0]
Shift_Out[47:32] → Wd[15:0]

受影响的状态位： N 和 Z

指令编码：

S010	001k	kkkk	dddd	pppq	qqss	ssUU	L000

说明：
将源寄存器 Ws 中的内容逻辑右移 lit5 位（最大移位位数为 31 位），并将结果存放在目标寄存器 Wd 中。除非移位值为零，否则 N 标志总是清零（见注 1）。
无论 lit5 中的移位值为何，该指令都会生成正确的结果（字操作移位值 > 15，Wd[15:0] = 0x0000）。
寄存器直接寻址字模式会将结果零扩展至 32 位，然后写入 Wd。
S 位用于选择指令大小。
L 位用于选择字或长字操作。
s 位用于选择源寄存器。
d 位用于选择目标寄存器。
p 位用于选择源寻址模式。
q 位用于选择目标寻址模式。
k 位用于提供立即数操作数。
注：
1. 仅当 Ws 的 MSb = 1 且移位值为零时，才能置 1 N 标志。
2. 该指令只能工作于字或长字模式。

指令字数： 1 或 0.5

指令周期数： 1

LSR		逻辑右移 Wb 位			
语法：	{标号:}	LSR.b	Ws,	Wb,	Wd
		LSR.bz	[Ws],		[Wd]
		LSR{.w}	[Ws++],		[Wd++]
		LSR.l	[Ws--],		[Wd--]
			[++Ws],		[++Wd]
			[--Ws],		[--Wd]

操作数： Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]

.....（续）

LSR	逻辑右移 Wb 位

操作：

Wb[15:0]→ Shift_Val

对于长字操作：

Ws[31:0]和 32'b0 → Shift_In[63:0]

将 Shift_In[63:0]逻辑右移 Shift_Val 位 → Shift_Out[63:0]

Shift_Out[63:32] → Wd

对于字操作：

16'b0、Ws[15:0]和 32'b0 → Shift_In[63:0]

将 Shift_In[63:0]逻辑右移 Shift_Val 位 → Shift_Out[63:0]

Shift_Out[47:32] → Wd[15:0]

对于字节操作：

24'b0、Ws[7:0]和 32'b0 → Shift_In[63:0]

将 Shift_In[63:0]逻辑右移 Shift_Val 位 → Shift_Out[63:0]

Shift_Out[39:32] → Wd[7:0]

受影响的状态位：N 和 Z

指令编码：

```
1010     101L     wwww     dddd     pppq     qqss     ssUU     B100
```

说明：

将源寄存器 Ws 中的内容逻辑右移 Wb 位，并将结果存放在目标寄存器 Wd 中。
单独使用该指令时，无论 Wb[15:0]中的移位值为何，都会生成正确的结果。

- 对于字节操作，移位值 > 7，Wd[7:0] = 0x00
- 对于字操作，移位值 > 15，Wd[15:0] = 0x0000
- 对于长字操作，移位值 > 31，Wd = 0x00000000
  当该指令与 ASRMW 和/或 LSRMW 指令一起使用进行多精度多位移位操作时，如果移位值大于 32，将不会生成正确的结果。

  Wb[31:16]中保存的任何数据都无效。

  除非移位值为 0，否则 N 标志总是清零（见注 1）。

  目标寄存器直接寻址扩展字节或字模式会将结果零扩展至 32 位，然后写入 Wd。

  L 和 B 位用于选择操作数据宽度。

  s 位用于选择源寄存器。

  w 位用于选择基准（移位计数）寄存器。

  d 位用于选择目标寄存器。

  p 位用于选择源寻址模式。

  q 位用于选择目标寻址模式。

  注：

  1. 仅当 Ws 的 MSb = 1 且移位值为零时，才能置 1 N 标志。

指令字数：1

指令周期数：1

LSRM	多精度逻辑右移（移位位数由短立即数确定）

语法：	{标号:}	LSRM{.l}	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	lit5,	Wnd

操作数：Ws ∈ [W0 ...W15]；lit5 ∈ [0...31]；Wnd ∈ [W1 ...W14]

.....（续）

LSRM	多精度逻辑右移（移位位数由短立即数确定）							
操作：	lit5[4:0]→ Shift_Val 0 → 右移输入（逻辑移位） Ws[31:0]和 32'b0 → Shift_In[63:0] 将 Shift_In[63:0]逻辑右移 Shift_Val 位 → Shift_Out[63:0] Shift_Out[63:32] → Wnd Shift_Out[31:0] | Wnd-1 → Wnd-1							
受影响的状态位：	N 和 Z							
指令编码：	1110	000k	kkkk	dddd	pppU	UUss	ssUU	0111
说明：	将源寄存器 Ws 中的内容逻辑右移 lit5 位（最大移位位数为 31 位），并将结果存放在目标寄存器 Wnd 中。包含下一个最低有效数据字的寄存器将已包含中间移位结果。将该值与从 Ws 移出的数据进行按位或运算以生成最终的移位结果，然后更新相应的目标寄存器。 Z 位为“粘住”位（只能清零）。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 k 位用于提供立即数操作数。 **注：** 1. 仅当 Ws 的 MSb = 1 且移位值为零时，才能置 1 N 标志。 2. 该指令只能工作于长字模式。							
指令字数：	1							
指令周期数：	2							

LSRM	多精度逻辑右移 Wb 位							
语法：	{标号:}	LSRM{.l}	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	Wb,	Wnd			
操作数：	Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]；Wnd ∈ [W1 ...W14]							
操作：	Wb[15:0]→ Shift_Val 0 → 右移输入（逻辑移位） Ws[31:0]和 32'b0 → Shift_In[63:0] 将 Shift_In[63:0]逻辑右移 Shift_Val 位 → Shift_Out[63:0] Shift_Out[63:32] → Wnd Shift_Out[31:0] | Wnd-1 → Wnd-1							
受影响的状态位：	N 和 Z							
指令编码：	1110	001U	ssss	dddd	pppU	UUww	wwUU	0111

.....（续）

LSRM		多精度逻辑右移 Wb 位			
说明：	将源寄存器 Ws 中的内容逻辑右移 Wb 位，并将结果存放在目标寄存器 Wnd 中。包含下一个最低有效数据字的寄存器将已包含中间移位结果。将该值与从 Ws 移出的数据进行按位或运算以生成最终的移位结果，然后更新相应的目标寄存器。 右移位数可以是 0 到 32 之间的任意值。如果 Wb[15:0]中保存的移位值超过 2'd32，则移位值将饱和至 2'd32 以保持一致性。Wb[31:16]中保存的任何数据都无效。 Z 位为“粘住”位（只能清零）。 w 位用于选择基准（移位计数）寄存器。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 **注：**该指令只能工作于长字模式。				
指令字数：	1				
指令周期数：	2				

LUAC		装载累加器高位			
语法：	{标号:}	LUAC.l	Ws, [Ws], [Ws++] [Ws--] [--Ws], [++Ws], [Ws+Wb],	{ Slit6, }	A B
操作数：	Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]；Slit6 ∈ [-32 ...+31]				
操作：	Ws[7:0] → ACC[71:64] $Shift_{Slit6}$(ACC) → ACC				
受影响的状态位：	OA 和 SA，*或者* OB 和 SB				
指令编码：	`1100 11A1 wwww ssss pppU UUkk kkkk 1011`				
说明：	读取有效地址的内容。将读取数据的 LSb 放入目标累加器的高 8 位（ACCx[71:64]），然后可选择将整个累加器内容进行算术移位。 A 位用于指定目标累加器。 s 位用于指定源寄存器 Wns。 p 位用于选择源寻址模式。 w 位用于指定偏移量寄存器 Wb。 k 位用于对可选操作数 Slit6 进行编码，该操作数确定累加器内容移位的位数；如果操作数 Slit6 不存在，则立即数位域设置为全 0。 **注：**操作数 Slit6 为正值表示算术右移。操作数 Slit6 为负值表示左移。				
指令字数：	1				
指令周期数：	1				

MAC		相乘并累加			
语法：{标号:}	MAC{.w} MAC.l	Wx [Wx] [Wx]+=kx [Wx]-=kx [Wx+=kx] [Wx-=kx] [Wx+W12]	, Wy, , [Wy], , [Wy]+=ky, , [Wy]-=ky, , [Wy+=ky], , [Wy-=ky], , [Wy+W12],	A B	{,AWB}

.....（续）

MAC	相乘并累加
操作数：	Wx ∈ {W0 ...W14}；Wy ∈ {W0 ...W14} 字模式：kx ∈ {-8, -6, -4, -2, 2, 4, 6, 8}；ky ∈ {-8, -6, -4, -2, 2, 4, 6, 8}； 长字模式：kx ∈ {-16, -12, -8, -4, 4, 8, 12, 16}；ky ∈ {-16, -12, -8, -4, 4, 8, 12, 16}； AWB ∈ {W0, W1, W2, W3, W13, [W13++]，[W15++][3]}
操作：	(ACC(A 或 B)) + ((Wx) * (Wy)) → ACC(A 或 B)； 舍入后的(ACC(B 或 A)) → AWB 当使用执行前修改/执行后修改的间接寻址时： (Wx)+kx→Wx 或(Wx)-kx→Wx； (Wy)+ky→Wy 或(Wy)-ky→Wy；
受影响的状态位：	OA 和 SA，*或者* OB 和 SB
指令编码：	1101 00AL wwww ssss IIIi iJJJ jjaa a011
说明：	对同时从 Wx 和 Wy 读取的数据或同时从 X 和 Y 地址空间获取的数据进行有符号/无符号（由 CORCON.US 定义）乘法。结果经过符号扩展（当至少一个操作数被视为有符号时）或零扩展（当两个操作数均为无符号时）至 72 位，然后加到指定的累加器。结果在累加器更新之前是否进行换算取决于是小数运算还是整数运算（由 CORCON.IF 定义）。小数运算将对结果进行换算以对齐操作数和累加器（msw）小数点（见注 3）。整数运算会将结果的 LSb 与累加器的 LSb 对齐。 当 X 和 Y 地址空间中至少有一个选择间接寻址时，Wx 和 Wy 寄存器都会提供相应的间接地址。地址修改量值分别为 kx 和 ky，表示有效地址中要修改的数据字节数。 可选的 AWB 用于指定是直接还是间接（见注 4）存储非 MAC 操作目标的累加器舍入后的小数内容（32 位）。舍入模式由 CORCON.RND 定义。写入数据宽度由所选指令数据大小决定。当 DSP 引擎工作于整数模式时，不适合使用 AWB。 读取的数据可以是 16 位或 32 位值。所有间接地址修改都会相应地进行换算。 L 位用于选择字或长字操作。 A 位用于选择存放结果的累加器。 I 位用于选择操作 X 空间寻址模式。 i 位用于选择 kx 修改值。 J 位用于选择操作 Y 空间寻址模式。 j 位用于选择 ky 修改值。 s 位用于选择 Wx 寄存器 w 位用于选择 Wy 寄存器。 a 位用于选择累加器回写目标和寻址模式。 **注：** 1. 在小数还是整数数据模式下工作由 CORCON.IF 定义。 2. MAC 寄存器直接寻址，其中 Wx = Wy 相当于 SQRAC Wx, ACCx 3. 当在小数模式下使用字大小的数据时，ACCx 的低位部分不受影响。因此，之前的（32 位数据）操作中可能存在的低位数据会得以保留。如果不需要该数据，用户应在初始化阶段清零 ACCx。 4. 如果源是执行前修改或执行后修改的有效地址，则不允许对间接源（X 或 Y）和 AWB 间接目标使用相同的 W 寄存器。 5. 堆栈必须保持长字对齐。因此，只允许将[W15++] AWB 与长字 MAC 类指令一起使用。
指令字数：	1
指令周期数：	1

MAX	强制执行（有符号）数据范围上限			
语法：	{标号:}	MAX{.w}	Wb,	Ws
		MAX.l		[Ws]
				[Ws++]
				[Ws--]
				[++Ws]

.....（续）

MAX	强制执行（有符号）数据范围上限

[--Ws]

操作数：Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W14]

操作：
如果(Ws) > (Wb)，则{ (Wb) → (Ws)；
0→ Z；0→ N；0→ OV
}
否则{
1→ Z；0→ N；0→ OV
}

受影响的状态位：N、Z 和 OV

指令编码：

```
1110      010L      UUUU      ssss      pppU      UUww      wwUU      1111
```

说明：
将 Ws 中任何大小的有符号数据值与 Wb 中保存的有符号上限值进行比较。
对于长字数据大小，如果 Ws[31:0]大于 Wb[31:0]，则将 Ws[31:0]设置为 Wb[31:0]中保存的限值。对于字数据大小，如果 Ws[15:0]大于 Wb[15:0]，则将 Ws[15:0]设置为 Wb[15:0]中保存的限值（忽略 Ws[31:16]）。Z、N 和 OV 状态位会置 1，这样后续 BGT 指令将执行转移。

如果 Ws[31:0]小于或等于 Wb[31:0]中的上限值（对于长字数据大小），或者 Ws[15:0]小于或等于 Wb[15:0]中的上限值（对于字数据大小），则不应用任何数据限制，这样 Ws 的内容将不会改变（对于给定的数据大小）。Z、N 和 OV 状态位会置 1，这样后续 BZ 或 BLE 指令将执行转移。

该指令总是将 OV 状态位清零。

s 位用于选择源（数据值）寄存器。

w 位用于选择基准（数据限制）寄存器。

p 位用于选择源寻址模式。

注：

1. 根据数据比较结果设置状态位。

指令字数：1

指令周期数：1

MAX	强制执行累加器数据范围上限

语法：{标号:} MAX A
B

操作数：无

操作：
如果为 MAX A，则(
如果 ACCA - ACCB > 0，则(
ACCB → ACCA；0→ Z；0→ N；0→ OV；
)
否则(
1→ Z；0→ N；0→ OV；)
)
如果为 MAX B，则(
如果 ACCB - ACCA > 0，则(ACCA → ACCB；
0→ Z；0→ N；0→ OV；
)
否则(
1→ Z；0→ N；0→ OV；)
)

受影响的状态位：N、OV 和 Z

指令编码：

```
0111      001A      UUUU      1101
```

.....（续）

MAX	强制执行累加器数据范围上限

说明：

将目标累加器中内容限制（在指令中定义）到之前装入其他累加器中的上限值。
比较操作检查目标累加器中的全部 72 位值，因此将对发生溢出的累加器进行限制（变量饱和）。

如果目标累加器的内容大于限制累加器的内容，则将限制累加器的内容装入目标累加器。Z 和 N 状态位会置 1，这样后续 BGT 指令将执行转移。

如果目标累加器的内容不大于限制累加器的内容，则目标累加器不受影响。Z 状态位会置 1，这样后续 BZ 指令将执行转移。

该指令总是将 OV 状态位清零。

A 位用于指定目标累加器。

注：

1. 该指令不会修改 OA 和 SA 或者 OB 和 SB 状态位。在 MAXAB 操作后执行 SFTAC <ACCx>, 0 以更新 DSP 状态，从而反映 ACCx 的内容。

指令字数：0.5

指令周期数：1

MAX	强制执行累加器数据范围上限，保存超限结果

语法：

{标号:}	MAX{.w}{.v} A,	Wd
	MAX.l{v} B,	[Wd]
		[Wd++]
		[Wd--]
		[++Wd]
		[--Wd]
		[Wd+Wb]

操作数：Wd ∈ [W0 ... W15]（见注 4）；Wb ∈ [W0 ...W15]

操作：

如果为 MAX A，则(
如果 ACCA - ACCB > 0，则(
+1 → Wd 或 ACCA - ACCB → Wd（见下文）；ACCB → ACCA；0→ Z；0→ N；0→ OV；
)
否则(
0 → Wd；
1→ Z；0→ N；0→ OV；)
)
如果为 MAX B，则(
如果 ACCB - ACCA > 0，则(+1 → Wd 或 ACCB - ACCA → Wd（见下文）；
ACCA → ACCB；
0→ Z；0→ N；0→ OV；
)
否则(
0→ Wd；
1→ Z；0→ N；0→ OV；)
)

受影响的状态位：N、OV 和 Z

指令编码：

1100	10AL	wwww	dddd	UUUq	qqUU	UUUV	1011

.....（续）

MAX	强制执行累加器数据范围上限，保存超限结果

说明：

将目标累加器中内容限制（在指令中定义）到之前装入其他累加器中的上限值。
比较操作检查目标累加器中的全部 72 位值，因此将对发生溢出的累加器进行限制（变量饱和）。

如果目标累加器的内容大于限制累加器的内容，则将限制累加器的内容装入目标累加器。对于 MAX（指令位域 V = 0，未声明时的默认值），将 Wd 设置为+1。对于 MAX.v（指令位域 V = 1），将超出限制的部分对应的（有符号）值写入 Wd：

- 字操作：仅考虑比较结果的 bit [47:32]并将其写入 Wd（目标也为字大小，请参见注 3）。如果超出限制的部分对应的值过大，无法用有符号的 16 位数字表示，则将其饱和处理为最大 16 位正值后再写入 Wd（即，将 Wd[15:0]设置为 0x7FFF）。
- 长字操作：仅考虑比较结果的 bit [31:0]并将其写入 Wd。如果超出限制的部分对应的值过大，无法用有符号的 32 位数字表示，则将其饱和处理为最大正值后再写入 Wd（即，将 Wd 设置为 0x7FFFFFFF）。

如果超出限制，则 Z 和 N 状态位会置 1，这样后续 BGT 指令将执行转移。

如果目标累加器的内容不大于限制累加器的内容，则目标累加器不受影响，Wd 清零。Z 状态位会置 1，这样后续 BZ 指令将执行转移。

该指令总是将 OV 状态位清零。

L 位用于选择字或长字操作。

A 位用于指定目标累加器。

d 位用于选择目标寄存器。

q 位用于选择目标寻址模式。

w 位用于定义偏移量 Wb。

V 位用于定义 Wd 的存在和结果格式。

注：

1. 该指令不会修改 OA 和 SA 或者 OB 和 SB 状态位。在 MAXAB 操作后执行 SFTAC <ACCx>, 0 以更新 DSP 状态，从而反映 ACCx 的内容。
2. 为了与所有字大小的寄存器直接写操作保持一致，Wd[15:0]将总是经过零扩展至 32 位。
3. 不允许使用寄存器直接寻址目标 W15。

指令字数：1

指令周期数：2

MIN	强制执行（有符号）数据范围下限

语法：

{标号:}	MIN{.w}	Wb,	Ws
	MIN.l		[Ws]
			[Ws++]
			[Ws--]
			[++Ws]
			[--Ws]

操作数：Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W14]

操作：
如果(Ws) < (Wb)，则{ (Wb) → (Ws)；
0→ Z；1→ N；0→ OV
}
否则{
1→ Z；0→ N；0→ OV
}

受影响的状态位：N、Z 和 OV

指令编码：

1110	010L	UUUU	ssss	pppU	UUww	wwUU	1011

.....（续）

MIN	强制执行（有符号）数据范围下限

说明：

将 Ws 中字大小或长字大小的有符号数据值与 Wb 中保存的有符号下限值进行比较。
对于长字数据大小，如果 Ws[31:0]小于 Wb[31:0]，则将 Ws[31:0]设置为 Wb[31:0]中保存的限值。对于字数据大小，如果 Ws[15:0]小于 Wb[15:0]，则将 Ws[15:0]设置为 Wb[15:0]中保存的限制值（忽略 Ws[31:16]）。Z、N 和 OV 状态位会置 1，这样后续 BLT 指令将执行转移。

如果 Ws[31:0]大于或等于 Wb[31:0]中的下限值（对于长字数据大小），或者 Ws[15:0]大于或等于 Wb[15:0]中的下限值（对于字数据大小），则不应用任何数据限制，这样 Ws 的内容将不会改变（对于给定的数据大小）。Z、N 和 OV 状态位会置 1，这样后续 BZ 或 BGE 指令将执行转移。

对于字大小的寄存器直接寻址操作，Ws 总是经过零扩展至 32 位（即使大于或等于 Wb 中的限值也如此）[2]。

该指令总是将 OV 状态位清零。

s 位用于选择源（数据值）寄存器。

w 位用于选择基准（数据限制）寄存器。

p 位用于选择源寻址模式。

注：

1. 根据数据比较结果设置状态位。

指令字数：1

指令周期数：1

MIN	强制执行累加器数据范围下限（无条件执行）

语法：

```
{标号:}  MIN  A
              B
```

操作数：无

操作：

如果为 MIN A，则(
如果 ACCA - ACCB < 0，则(
ACCB → ACCA；0→ Z；1→ N；0→ OV；
)
否则(
1→ Z；0→ N；0→ OV；)
)
如果为 MIN B，则(
如果 ACCB - ACCA < 0，则(
ACCA→ ACCB；
0→ Z；1→ N；0→ OV；
)
否则(
1→ Z；0→ N；0→ OV；
)
)

受影响的状态位：N、OV 和 Z

指令编码：

```
0111    001A    UUUU    1011
```

.....（续）

MIN	强制执行累加器数据范围下限（无条件执行）

说明：

将目标累加器中内容限制（在指令中定义）到之前装入其他累加器中的下限值。
比较操作检查目标累加器中的全部 72 位值，因此将对发生溢出的累加器进行限制（变量饱和）。

如果目标累加器的内容小于限制累加器的内容，则将限制累加器的内容装入目标累加器。Z 和 N 状态位会置 1，这样后续 BLT 指令将执行转移。

如果目标累加器的内容不小于限制累加器的内容，则目标累加器不受影响。Z 状态位会置 1（Z = 1），这样后续 BLT 指令将执行转移。

该指令总是将 OV 状态位清零。

A 位用于指定目标累加器。

注：

1. 该指令不会修改 OA 和 SA 或者 OB 和 SB 状态位。在执行 MAXAB 后执行 SFTAC <ACCx>, 0 以更新 DSP 状态，从而反映 ACCx 的内容。

指令字数：0.5

指令周期数：1

MIN	强制执行累加器数据范围下限，保存超限结果（无条件执行）

语法：

{标号:}	MIN{.w}{.v} A,	Wd
	MIN.l{v} B,	[Wd]
		[Wd++]
		[Wd--]
		[++Wd]
		[--Wd]
		[Wd+Wb]

操作数：Wd ∈ [W0 ...W15]（见注 4）；Wb ∈ [W0 ...W15]

操作：

如果为 MIN A，则(
如果 ACCA - ACCB < 0，则(
-1 → Wd 或 ACCA - ACCB → Wd（见下文）；ACCB→ ACCA；0→ Z；1→ N；0→ OV；
)
否则(
0→ Wd；
1→ Z；0→ N；0→ OV；)
如果为 MIN B，则(
如果 ACCB - ACCA < 0，则(
-1 → Wd 或 ACCB - ACCA → Wd（见下文）；ACCA→ ACCB；
0→ Z；1→ N；0→ OV；
)
否则(
0→ Wd；
1→ Z；0→ N；0→ OV；
)
)

受影响的状态位：N、OV 和 Z

指令编码：

1100	10AL	wwww	dddd	UUUq	qqUU	UUUV	0011

.....（续）

MIN	强制执行累加器数据范围下限，保存超限结果（无条件执行）

说明：

将目标累加器中内容限制（在指令中定义）到之前装入其他累加器中的下限值。
比较操作检查目标累加器中的全部 72 位值，因此将对发生溢出的累加器进行限制（变量饱和）。

如果目标累加器的内容小于限制累加器的内容，则将限制累加器的内容装入目标累加器。对于 MIN（指令位域 V = 0，未声明时的默认值），将 Wd 设置为-1。对于 MIN.V（指令位域 V = 1），将超出限制的部分对应的（有符号）值写入 Wd：

- 字操作：仅考虑比较结果的 bit [47:32]并将其写入 Wd（目标也为字大小，请参见注 3）。如果超出限制的部分对应的值过大，无法用有符号的 16 位数字表示，则将其饱和处理为最大 16 位负值后再写入 Wd（即，将 Wd[15:0]设置为 0x8000）。
- 长字操作：仅考虑比较结果的 bit [31:0]并将其写入 Wd。如果超出限制的部分对应的值过大，无法用有符号的 32 位数字表示，则将其饱和处理为最大负值后再写入 Wd（即，将 Wd 设置为 0x80000000）。

如果超出限制，则 Z 和 N 状态位会置 1，这样后续 BLT 指令将执行转移。

如果目标累加器的内容不小于限制累加器的内容，则目标累加器不受影响，Wd 清零。Z 状态位会置 1，这样后续 BZ 指令将执行转移。

该指令总是将 OV 状态位清零。

L 位用于选择字或长字操作。

A 位用于指定目标累加器。

d 位用于选择目标寄存器。

q 位用于选择目标寻址模式。

w 位用于定义偏移量 Wb。

V 位用于定义 Wd 的结果格式。

注：

1. 该指令不会修改 OA 和 SA 或者 OB 和 SB 状态位。在执行 MINABW 后执行 SFTAC <ACCx>, 0 以更新 DSP 状态，从而反映 ACCx 的内容。
2. 为了与所有字大小的寄存器直接写操作保持一致，Wd[15:0]将总是经过零扩展至 32 位（与结果的符号无关）。
3. 不允许使用寄存器直接寻址目标 W15。

指令字数：1

指令周期数：2

MOV	将协处理器寄存器中内容传送至[Wns + 有符号立即数偏移量]

语法：{标号:} MOV.l Fs, [Wnd+Slit14]

操作数：
Wnd ∈ [W0 ... W15]
Fs ∈ [F0 ... F31]

操作：Wns → [Wnd+Slit14]

受影响的状态位：无

指令编码：

1000	011s	ssss	dddd	kkkk	kkkk	kkkk	zz01

说明：

将目标寄存器的内容传送至源有效地址。上面所示的语法对应于浮点协处理器 F-reg。该操作不会修改 Wnd 的内容。
指令编码包括用于 12 位立即数的空间，立即数会针对长字数据传送进行相应的换算，以便生成相应的字节偏移量值。

z 位用于选择目标协处理器。

s 位用于选择源寄存器。

d 位用于选择浮点目标寄存器。

注：

1. 该指令只能工作于长字模式。

指令字数：1

指令周期数：1

MOV		将 f 中内容传送至 Wnd（字或长字）							
语法：	{标号:}	MOV{.w} MOV.l	f,	Wnd					
操作数：	f ∈ [0 ...1 MB]；Wnd ∈ [W0 ...W15]								
操作：	(f) → Wnd								
受影响的状态位：	无								
指令编码：	1001	00dd	ffff	ffff	ffff	ffff	fffL	dd01	
说明：	将任意文件寄存器的内容传送至指定的 W 寄存器。文件地址为字地址。在写入目标之前，字数据将经过零扩展至 32 位。 L 位用于选择字或长字操作。 f 位用于选择文件寄存器的地址。 d 位用于选择目标寄存器。 **注：** 1. 可访问的文件地址空间为 1 MB。 2. 文件地址总是为字对齐或长字对齐。操作码不存储 LSb，因为该位总是为 1'b0。								
指令字数：	1								
指令周期数：	1								

MOV		将 f 中内容传送至 Wnd（字节）							
语法：	{标号:}	MOV.bz	f,	Wnd					
操作数：	f ∈ [0 ...1 MB]；Wnd ∈ [W0 ...W14]								
操作：	(f) → Wnd								
受影响的状态位：	无								
指令编码：	1001	10dd	ffff	ffff	ffff	ffff	ffff	dd01	
说明：	将任意文件寄存器的内容传送至指定的 W 寄存器。文件地址为字节地址。在写入目标之前，字节数据将经过零扩展至 32 位。 f 位用于选择文件寄存器的地址。 d 位用于选择目标寄存器。								
指令字数：	1								
指令周期数：	1								

MOV		将 f（扩展）中内容传送至 Wd							
语法：	{标号:}	MOV.l MOV{.w} MOV{.bz} MOV{.b}	f,	Wd [Wd] [Wd++] [Wd--] [++Wd] [--Wd]					
操作数：	f ∈ [1 MB[1] ...4 GB]；Wd ∈ [W0 ...W15]								
操作：	(f) → Wd								
受影响的状态位：	无								
指令编码：									
第一个字	1111	000U	ffff	ffff	ffff	ffff	ffff	U011	
第二个字	1111	000L	dddd	Ufff	fffq	qqff	ffff	B111	

.....（续）

MOV		将 f（扩展）中内容传送至 Wd							
说明：	将任意文件寄存器的内容传送至 Wd。文件地址为字节地址。 d 位用于选择工作寄存器。 q 位用于选择目标寻址模式。 f 位用于选择文件寄存器的地址： file[31:0] = Word1[18:13]、Word2[9:4]和 Word1[23:4]								
指令字数：	2								
指令周期数：	2 **注：** 1. 如果文件地址小于 2^{20}，则汇编器使用 LDW。 2. 即使在 Bcc op 后跳过，也仍保留两个周期。								

MOV		将 Wns 中内容传送至 f（字节）							
语法：	{标号:}	MOV.b	Wns	f					
操作数：	f ∈ [0 ...1 MB]；Wns ∈ [W0 ...W15]								
操作：	Wns → (f)								
受影响的状态位：	无								
指令编码：	1001	11ss	ffff	ffff	ffff	ffff	ffff	ss01	
说明：	将指定 W 寄存器的内容传送至任意文件寄存器。文件地址为字节地址。 f 位用于选择文件寄存器的地址。 s 位用于选择源寄存器。								
指令字数：	1								
指令周期数：	1								

MOV		将 Wns 中内容传送至 f（字/长字）							
语法：	{标号:}	MOV{.w} MOV.l	Wns	f					
操作数：	f ∈ [0 ...1 MB]；Wns ∈ [W0 ...W15]								
操作：	Wns → (f)								
受影响的状态位：	无								
指令编码：	1001	01ss	ffff	ffff	ffff	ffff	fffL	ss01	
说明：	将指定 W 寄存器的内容传送至任意文件寄存器。文件地址为字地址。 L 位用于选择字或长字操作。 f 位用于选择文件寄存器的地址。 s 位用于选择源寄存器。 **注：**文件地址总是为字对齐或长字对齐。操作码不存储 LSb，因为该位总是为 1'b0。								
指令字数：	1								
指令周期数：	1								

MOV		将 Ws 中内容传送至 f（扩展）		
语法：	{标号:}	MOV.l MOV{.w} MOV{.b}	Ws, [Ws], [Ws++], [Ws--], [++Ws],	f

.....（续）

MOV	将 Ws 中内容传送至 f（扩展）							
			[--Ws],					
操作数：	f ∈ [1 MB[1] ...4 GB]；Ws ∈ [W0 ...W15]							
操作：	Ws → (f)							
受影响的状态位：	无							
指令编码：								
第一个字	1111	001U	ffff	ffff	ffff	ffff	ffff	U011
第二个字	1111	001L	ssss	Ufff	pppf	ffff	ffff	B111
说明：	将 Ws 的内容传送至任意文件寄存器。文件地址为字节地址。 s 位用于选择工作寄存器。 p 位用于选择源寻址模式。 f 位用于选择文件寄存器的地址： file[31:0] = Word1[18:13]、Word2[9:4]和 Word1[23:4]							
指令字数：	2							
指令周期数：	2							

MOV	将[Wns + 有符号立即数偏移量]中内容传送至 Wnd		
语法：	{标号:}	MOV.b	[Wns+Slit12], Wnd
		MOV.bz	[Wns+Slit12], Wnd
		MOV{.w}	[Wns+Slit13], Wnd
		MOV.l	[Wns+Slit14], Wnd
操作数：	Wns ∈ [W0 ...W15] Wnd ∈ [W0 ...W15]		
操作：	[Wns+Slit12/13/14] → Wnd		
受影响的状态位：	无		
指令编码：	1111	110L	dddd ssss kkkk kkkk kkkk B011
说明：	将源有效地址的内容传送至指定的 W 寄存器。该操作不会修改 Wns 的内容。 指令编码包括用于 12 位立即数的空间，立即数会针对字节、字和长字数据传送进行相应的换算，以便生成相应的字节偏移量值。 L 和 B 位用于选择操作数据宽度。 d 位用于选择目标寄存器。 s 位用于选择源寄存器。 k 位用于指定立即数操作数。		
指令字数：	1		
指令周期数：	1		

MOV	将[Wns + 有符号立即数偏移量]中内容传送至协处理器寄存器		
语法：	{标号:}	MOV.l	[Wns+Slit14], Fd
操作数：	Wns ∈ [W0 ...W15] Fd ∈ [F0 ... F31]		
操作：	[Wns+Slit14] → Fd		
受影响的状态位：	无		
指令编码：	1000	010d	dddd ssss kkkk kkkk kkkk zz01

.....（续）

MOV	将[Wns + 有符号立即数偏移量]中内容传送至协处理器寄存器

说明：

将源有效地址的内容传送至目标寄存器。上面所示的语法对应于浮点协处理器 F-reg。该操作不会修改 Wns 的内容。
指令编码包括用于 12 位立即数的空间，立即数会针对长字数据传送进行相应的换算，以便生成相应的字节偏移量值。

z 位用于选择目标协处理器。

s 位用于选择源寄存器。

d 位用于选择浮点目标寄存器。

注：

1. 仅当实例化协处理器时才支持该指令。
2. 该指令只能工作于长字模式。

指令字数：1

指令周期数：1

MOV	将 Wns 中内容传送至[Wnd + 有符号立即数偏移量]

语法：

{标号:}	MOV.b	Wns, [Wnd+Slit12]
	MOV{.w}	Wns, [Wnd+Slit13]
	MOV.l	Wns, [Wnd+Slit14]

操作数：Wnd ∈ [W0 ...W15]
Wns ∈ [W0 ...W15]

操作：Wns → [Wnd+Slit12/13/14]

受影响的状态位：无

指令编码：

1111	110L	dddd	ssss	kkkk	kkkk	kkkk	B111

说明：

将指定 W 寄存器的内容移至目标有效地址。该操作不会修改 Wnd 的内容。
指令编码包括用于 12 位立即数的空间，立即数会针对字节、字和长字数据传送进行相应的换算，以便生成相应的字节偏移量值。

L 和 B 位用于选择操作数据宽度。

d 位用于选择目标寄存器。

s 位用于选择源寄存器。

k 位用于指定立即数操作数。

指令字数：1

指令周期数：1

MOV	将 Ws 中内容传送至 Wd

语法：

{标号:}	MOV.b	Ws,	Wd
	MOV.bz	[Ws],	[Wd]
	MOV{.w}	[Ws++]	[Wd++]
	MOV.l	[Ws--]	[Wd--]
		[--Ws],	[--Wd]
		[++Ws],	[++Wd]
		[Ws+Wb],	[Wd+Wb]
		SR	SR

操作数：Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]；Wb ∈ [W0 ...W14]（见注 6）

操作：(EAs) → (EAd)

受影响的状态位：无（见注 3）

指令编码：

S000	001L	dddd	ssss	pppq	qqww	wwUU	B000

.....（续）

MOV	将 Ws 中内容传送至 Wd

说明：

将源寄存器中的内容传送至目标寄存器。
S 位用于选择指令大小。

L 和 B 位用于选择操作数据宽度。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

w 位用于定义偏移量 Wb。

注：

1. 当以 SR 为目标时，SR 将因指令操作而被修改。
2. 写入 SR 时，不允许使用.bz 数据大小/模式。
3. 如果源寻址模式与目标寻址模式均使用寄存器偏移量寻址，则 Wb 无法设置为 W15。

指令字数：1 或 0.5

指令周期数：1

MOV	将协处理器寄存器中内容传送至 Wd

语法：

{标号:}	MOV.l	Fs,	Wd
		FSR,	[Wd]
		FSRH,	[Wd++]
		FCR,	[Wd--]
		FEAR,	[--Wd]
			[++Wd]
			[Wd+Wb]

操作数：Fs ∈ [F0 ...F31]；Wd ∈ [W0 ...W15]；Wb ∈ [W0 ...W14]（见注 2）

操作：(Fs、FSR、FSRH、FCR 或 FEAR) → (EAd)

受影响的状态位：无

指令编码：

S000	011s	ssss	dddd	UUUq	qqww	wwRU	zz00

说明：

将源协处理器数据、状态或控制寄存器中的内容传送至目标寄存器。上面所示的语法对应于浮点协处理器寄存器。
S 位用于选择指令大小。

z 位用于选择目标协处理器。

s 位用于选择协处理器源寄存器。

d 位用于选择目标寄存器。

q 位用于选择目标寻址模式。

w 位用于定义偏移量 Wb。

R 位用于在 F-reg 与 FPU 特殊寄存器之间进行选择。

注：

1. 该指令只能工作于长字模式。
2. 仅当 Wd = W15 时执行堆栈指针（W15）限制检查。因此，不允许将 Wb 设置为 W15。

指令字数：1 或 0.5

指令周期数：1

MOV	将立即数传送至 Wnd

语法：

{标号:}	MOV.bz	lit8	Wnd
	MOV{.w}	lit16	
	MOV.sl	lit24	

.....（续）

MOV	将立即数传送至 Wnd

操作数：lit8∈ [0 ... 255]；lit16 ∈ [0 ...65535]；lit24 ∈ [0 ...16 MB]；Wnd ∈ [W0 ...W15]

操作：8'h00 和 lit24 → Wnd

受影响的状态位：无

指令编码：

10dd	ddkk	kkkk	kkkk	kkkk	kkkk	kkkk	kk11

说明：将立即数 k 零扩展至 32 位后装入 Wnd 寄存器。
d 位用于选择工作寄存器。

k 位用于指定 8 位、16 位或 24 位立即数的值。汇编器通过对 8 位和 16 位立即数进行零扩展，以生成操作码的 24 位值。

指令字数：1

指令周期数：1

MOV	将长立即数传送至协处理器寄存器

语法：

{标号:}	MOV.l	lit32,	Fd
			FSR
			FCR

操作数：lit32 ∈ [0 ... 4 GB]；Fd ∈ [F0 ...F31]

操作：lit32 → (Fd、FSR 或 FCR)

受影响的状态位：无

指令编码：

第一个字	1000	001z	kkkk	kkkk	kkkk	kkkk	kkkk	z001
第二个字	1000	001d	dddd	Ukkk	kkkU	UUkk	kkkk	R101

说明：将立即数 k 装入协处理器目标数据寄存器。上面所示的语法对应于浮点协处理器寄存器。
d 位用于选择协处理器目标寄存器。

z 位用于选择目标协处理器。

zz[1:0] = Word1[24]和 Word1[3]

R 位用于在 F-reg 与 FPU 特殊寄存器之间进行选择。

k 位用于指定 32 位立即数的值：

lit32[31:0] = Word1[18:13]、Word2[9:4]和 Word1[23:4]

指令字数：2

指令周期数：2

MOV	将长立即数传送至 Wd

语法：

{标号:}	MOV.l	lit32,	Wd
			[Wd]
			[Wd++]
			[Wd--]
			[++Wd]
			[--Wd]

操作数：lit32 ∈ [0 ...4 GB]；Wd ∈ [W0 ...W15]

操作：lit32 → Wd

受影响的状态位：无

指令编码：

第一个字	1000	000U	kkkk	kkkk	kkkk	kkkk	kkkk	U001
第二个字	1000	000U	dddd	Ukkk	kkkq	qqkk	kkkk	U101

.....（续）

MOV	将长立即数传送至 Wd
说明：	将立即数 k 装入 Wd 寄存器。 d 位用于选择工作寄存器。 q 位用于选择目标寻址模式。 k 位用于指定 32 位立即数的值： lit32[31:0] = Word2[18:13]、Word2[9:4]和 Word1[23:4]
指令字数：	2
指令周期数：	2

MOV	将短立即数传送至 Wnd
语法：	{标号:} MOV.l lit5, Wnd
操作数：	lit5 ∈ [0 ...31]；Wnd ∈ [W0 ...W14]
操作：	27'h0000000 和 lit5 → Wnd
受影响的状态位：	无
指令编码：	0001 000k kkkk dddd
说明：	将立即数 k 零扩展至 32 位后装入 Wnd 寄存器。 d 位用于选择工作寄存器。 k 位用于指定 5 位立即数的值。
指令字数：	0.5
指令周期数：	1

MOV	使用位反转寻址将 Ws 中内容传送至 Wd
语法：	{标号:} MOVR{.w} Ws, [Wd++] MOVR.l [Ws], [++Wd] [Ws++], [++Ws],
操作数：	Ws ∈ [W0 ...W14]；Wd ∈ [W0 ...W14]
操作：	使用位反转寻址的(EAs) → (EAd)
受影响的状态位：	无
指令编码：	1000 001L dddd ssss pppq qqUU UUUU 1100
说明：	使用位反转寻址将源地址的内容传送至目标地址以生成目标 EA。目标位反转寻址修改量源自 XBREV.XB[14:0]。 L 位用于选择操作数据宽度。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。 **注：** 1. 仅当源寻址模式为无修改的寄存器间接寻址（[Ws]）时才允许 Ws = Wd。
指令字数：	1
指令周期数：	1

MOV	使用立即数偏移量将堆栈中内容传送至 Wnd
语法：	{标号:} MOV.l [W15-lit7], Wnd [W14+slit7], Wnd

.....（续）

MOV	使用立即数偏移量将堆栈中内容传送至 Wnd

操作数：Wnd ∈ [W0 ...W14]；
lit7 ∈ [4, 8 ....124, 128]（见注 2）
slit7 ∈ [-64, -60 .... +56, +60]

操作：([W15 - lit7]) → Wnd 或([W14 + slit7]) → Wnd

受影响的状态位：无

指令编码：

```
0000    11Fk    kkkk    dddd
```

说明：搭配使用 SP（W15）或 FP（W14）与（字节）偏移量作为源地址，将系统堆栈的内容传送至目标寄存器。堆栈源有效地址定义为[W15 - lit7]或[W14 + slit7]，但按 32 位边界对齐（即，仅传送长字数据）。CPU 使用的偏移量是通过操作码内存储的 5 位立即数生成的有符号值。有效地址为长字对齐，因此地址的低 2 位总是为 2'b00。

- 当 W15 为源地址寄存器时，偏移量总是为负，因此符号位隐含（即，不存储在操作码内）为 1'b1。汇编器将生成 lit7 的二进制补码，将其截断，并丢弃符号位以创建 5 位操作码立即数值。偏移量不能为零。
- 当 W14 为源地址寄存器时，偏移量可正亦可负，因此符号位存储在 5 位立即数内。汇编器将截断 Slit7 以创建 5 位操作码立即数值。

  d 位用于选择目标寄存器。

  F 位用于在 W15（F = 0）与 W14（F = 1）之间进行选择。

  k 位用于定义 5 位有符号立即数。

  注：

  1. 该指令只能工作于长字模式。

指令字数：0.5

指令周期数：1

MOV	将 Ws 中内容传送至协处理器寄存器

语法：

{标号:}	MOV.l	Ws,	Fd
		[Ws],	FSR
		[Ws++]	FCR
		[Ws--]	FEAR
		[--Ws],	
		[++Ws],	
		[Ws+Wb],	

操作数：Fd ∈ [F0 ...F31]；Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W14]（见注 2）

操作：(EAs) → Fd、FSR、FCR 或 FEAR

受影响的状态位：无

指令编码：

```
S000    010d    dddd    ssss    pppU    UUww    wwRU    zz00
```

说明：将源寄存器中的内容传送至协处理器目标数据、状态或控制寄存器。上面所示的语法对应于浮点协处理器寄存器。

S 位用于选择指令大小。

z 位用于选择目标协处理器。

s 位用于选择源寄存器。

d 位用于选择协处理器目标寄存器。

p 位用于选择源寻址模式。

w 位用于定义偏移量 Wb。

R 位用于在 F-reg 与 FPU 特殊寄存器之间进行选择。

注：

1. 该指令只能工作于长字模式。
2. 仅当 Wd = W15 时执行堆栈指针（W15）限制检查。因此，不允许将 Wb 设置为 W15。

指令字数：1 或 0.5

.....（续）

MOV	将 Ws 中内容传送至协处理器寄存器
指令周期数：	1

MOV	使用立即数偏移量将 Wns 中内容传送至堆栈			
语法：	{标号:}	MOV.l	Wns,	[W15 - lit7] [W14 + slit7]
操作数：	Wns ∈ [W0 ...W14]； lit8 ∈ [4, 8 ....124, 128]（见注 2） slit7 ∈ [-64, -60 ....+56, +60]			
操作：	Wns → ([W15 - lit7])或 Wns → ([W14 + slit7])			
受影响的状态位：	无			
指令编码：	0000 10Fk kkkk ssss			
说明：	搭配使用 SP（W15）或 FP（W14）与（字节）偏移量作为目标地址，将系统堆栈的内容传送至目标寄存器。堆栈目标有效地址定义为[W15 - lit7]或[W14 + slit7]，但按 32 位边界对齐（即，仅传送长字数据）。CPU 使用的偏移量是通过操作码内存储的 5 位立即数生成的有符号值。有效地址为长字对齐，因此地址的低 2 位总是为 2'b00。 • 当 W15 为目标地址寄存器时，偏移量总是为负，因此符号位隐含（即，不存储在操作码内）为 1'b1。汇编器将生成 lit7 的二进制补码，将其截断，并丢弃符号位以创建 5 位操作码立即数值。偏移量不能为零。 • 当 W14 为目标地址寄存器时，偏移量可正亦可负，因此符号位存储在 5 位立即数内。汇编器将截断 Slit7 以创建 5 位操作码立即数值。 s 位用于选择源寄存器。 F 位用于在 W15（F = 0）与 W14（F = 1）之间进行选择。 k 位用于指定立即数操作数。 **注：** 1. 该指令只能工作于长字模式。			
指令字数：	0.5			
指令周期数：	1			

MOVIF	按照条件将 Wb 或 Wns 传送至 Wd					
语法：	{标号:}	MOVIF.b MOVIF.bz MOVIF{.w} MOVIF.l	CC,	Wb,	Wns,	Wd [Wd] [Wd++] [Wd--] [--Wd] [++Wd]
操作数：	Wb ∈ [W0 ...W15]；Wns ∈ [W0 ...W15]；Wd ∈ [W0 ... W15] CC ∈ Z, N, C, OV, GT, LT, GTU					
操作：	如果 CC 为真，则 Wb → (EAd) 否则 Wns → (EAd)					
受影响的状态位：	无					
指令编码：	1111 011L dddd ssss rrrq qqww wwUU B011					

.....（续）

MOVIF	按照条件将 Wb 或 Wns 传送至 Wd

说明：测试指定的条件。如果为真，则将 Wb 源寄存器的内容传送至目标，否则将 Wns 源寄存器的内容传送至目标。
L 和 B 位用于选择操作数据宽度。
w 位用于选择（条件为真）Wb 源寄存器。
s 位用于选择（条件为假）Ws 源寄存器。
d 位用于选择目标寄存器。
q 位用于选择目标寻址模式。
r 位用于选择传送条件。

指令字数：1
指令周期数：1

MOVS	将有符号立即数传送至 Wd

语法：

{标号:}	MOVS.b	slit16,	Wd
	MOVS{.w}		[Wd]
	MOVS.l		[Wd++]
			[Wd--]
			[++Wd]
			[--Wd]
			SR

操作数：slit16 ∈ [-32768 ... 32767]；Wd ∈ [W0 ...W15]
操作：字节：slit16[7:0] → (EAd[7:0])
字：slit16[15:0] → (EAd[15:0])
长字或字寄存器直接寻址：{16{slit16[15]}}和 slit16[15:0] → (EAd[31:0])
受影响的状态位：无（见注 4）

指令编码：

1000	1kkk	dddd	kkkk	kkkq	qqkk	kkkk	LB01

说明：将有符号的立即数值写入目标。
MOVS.b 会将立即数的低 8 位传送至目标。对于寄存器直接寻址，寄存器的其余位将不受影响。
MOVS.w 会将 16 位立即数传送至目标（见注 1）。对于寄存器直接寻址，该值将符号扩展至 32 位。
MOVS.l 会将立即数符号扩展至 32 位，然后再将其传送至目标。
L 和 B 位用于选择操作数据宽度。
d 位用于选择工作寄存器。
q 位用于选择目标寻址模式。
k 位用于指定立即数的值。对于.b 模式，汇编器将对 8 位立即数值进行零扩展，以生成操作码的 16 位值。

注：

1. 对于字节大小的数据，立即数既可视为有符号值也可视为无符号值，因为只写入 8 位值。声明的 slit16 立即数值必须介于 0x0000 与 0x00FF 之间。
2. 对于使用任何间接寻址模式的字大小数据，立即数既可视为有符号值也可视为无符号值，因为只写入 16 位值。
3. 长字寄存器直接寻址模式与字寄存器直接寻址模式是等效的操作，因为两者均将立即数符号扩展至 32 位。
4. 当以 SR 为目标时，SR 将因指令操作而被修改。

指令字数：1
指令周期数：1

MPY	相乘，结果存入累加器

语法：{标号:}

MPY{.w}	Wx	, Wy,	A	{,AWB}
MPY.l	[Wx]	, [Wy],	B	

.....（续）

MPY	相乘，结果存入累加器

[Wx]+=kx , [Wy]+=ky,
[Wx]-=kx , [Wy]-=ky,
[Wx+=kx] , [Wy+=ky],
[Wx-=kx] , [Wy-=ky],
[Wx+W12] , [Wy+W12],

操作数：
Wx ∈ {W0 ...W14}；Wy ∈ {W0 ...W14}
字模式：kx ∈ {-8, -6, -4, -2, 2, 4, 6, 8}；ky ∈ {-8, -6, -4, -2, 2, 4, 6, 8}；
长字模式：kx ∈ {-16, -12, -8, -4, 4, 8, 12, 16}；ky ∈ {-16, -12, -8, -4, 4, 8, 12, 16}；
AWB ∈ {W0, W1, W2, W3, W13, [W13++]，[W15++][3]}

操作：
((Wx) * (Wy)) → ACC(A 或 B)；
舍入后的(ACC(B 或 A)) → AWB
当使用执行前修改/执行后修改的间接寻址时：
(Wx)+kx→Wx 或(Wx)-kx→Wx；
(Wy)+ky→Wy 或(Wy)-ky→Wy；

受影响的状态位：
OA 和 SA，*或者* OB 和 SB

指令编码：

```
1101      01AL      wwww      ssss      IIIi      iJJJ      jjaa      a011
```

说明：
对同时从 Wx 和 Wy 读取的数据或同时从 X 和 Y 地址空间获取的数据进行有符号/无符号（由 CORCON.US 定义）乘法。结果经过符号扩展（当至少一个操作数被视为有符号时）或零扩展（当两个操作数均为无符号时）至 72 位，然后写入指定的累加器。此外，小数结果也会在累加器更新之前进行换算，以对齐操作数和累加器（msw）小数点。
当 X 和 Y 地址空间中至少有一个选择间接寻址时，Wx 和 Wy 寄存器都会提供相应的间接地址。地址修改量值分别为 kx 和 ky，表示有效地址中要修改的数据字节数。
可选的 AWB 用于指定是直接还是间接（见注 2）存储非 MPY 操作目标的累加器舍入后的小数内容（32 位）。舍入模式由 CORCON.RND 定义。写入数据宽度由所选指令数据大小决定（见注 3）。当 DSP 引擎工作于整数模式时，不适合使用 AWB。
读取的数据可以是 16 位或 32 位值。所有间接地址修改都会相应地进行换算。
L 位用于选择字或长字操作。
A 位用于选择存放结果的累加器。
I 位用于选择操作 X 空间寻址模式。
i 位用于选择 kx 修改值。
J 位用于选择操作 Y 空间寻址模式。
j 位用于选择 ky 修改值。
s 位用于选择 Wx 寄存器
w 位用于选择 Wy 寄存器。
a 位用于选择累加器回写目标和寻址模式。

注：

1. 在小数还是整数数据模式下工作由 CORCON.IF 定义。
2. 如果源是执行前修改或执行后修改的有效地址，则不允许对间接源（X 或 Y）和 AWB 间接目标使用相同的 W 寄存器。
3. 堆栈必须保持长字对齐。因此，只允许将[W15++] AWB 与长字 MAC 类指令一起使用。

指令字数：1
指令周期数：1

MPYN	相乘并取反，结果存入累加器

语法：{标号:}

MPYN{.w} Wx , Wy, A {,AWB}
MPYN.l [Wx] , [Wy], B
[Wx]+=kx , [Wy]+=ky,
[Wx]-=kx , [Wy]-=ky,

.....（续）

MPYN		相乘并取反，结果存入累加器			
		[Wx+=kx]	, [Wy+=ky],		
		[Wx-=kx]	, [Wy-=ky],		
		[Wx+W12]	, [Wy+W12],		

操作数：Wx ∈ {W0 ...W14}；Wy ∈ {W0 ...W14}
字模式：kx ∈ {-8, -6, -4, -2, 2, 4, 6, 8}；ky ∈ {-8, -6, -4, -2, 2, 4, 6, 8}；
长字模式：kx ∈ {-16, -12, -8, -4, 4, 8, 12, 16}；ky ∈ {-16, -12, -8, -4, 4, 8, 12, 16}；
AWB ∈ {W0, W1, W2, W3, W13, [W13++], [W15++][2]}

操作：- ((Wx) * (Wy)) → ACC(A 或 B)；
舍入后的(ACC(B 或 A)) → AWB
当使用执行前修改/执行后修改的间接寻址时：
(Wx)+kx→Wx 或(Wx)-kx→Wx；
(Wy)+ky→Wy 或(Wy)-ky→Wy；

受影响的状态位：OA 和 SA，*或者* OB 和 SB

指令编码：

```
1101      01AL      wwww      ssss      IIIi      iJJJ      jjaa      a111
```

说明：对同时从 Wx 和 Wy 读取的数据或同时从 X 和 Y 地址空间获取的数据进行有符号/无符号（由 CORCON.US 定义）乘法。结果经过符号扩展（当至少一个操作数被视为有符号时）或零扩展（当两个操作数均为无符号时）至 72 位并取反，然后写入指定的累加器。此外，小数结果也会在累加器更新之前进行换算，以对齐操作数和累加器（msw）小数点。
当 X 和 Y 地址空间中至少有一个选择间接寻址时，Wx 和 Wy 寄存器都会提供相应的间接地址。地址修改量值分别为 kx 和 ky，表示有效地址中要修改的数据字节数。

可选的 AWB 用于指定是直接还是间接存储非 MPYN 操作目标的累加器舍入后的小数内容（32 位）。舍入模式由 CORCON.RND 定义。写入数据宽度由所选指令数据大小决定（见注 3）。当 DSP 引擎工作于整数模式时，不适合使用 AWB。

读取的数据可以是 16 位或 32 位值。所有间接地址修改都会相应地进行换算。

L 位用于选择字或长字操作。

A 位用于选择存放结果的累加器。

I 位用于选择操作 X 空间寻址模式。

i 位用于选择 kx 修改值。

J 位用于选择操作 Y 空间寻址模式。

j 位用于选择 ky 修改值。

s 位用于选择 Wx 寄存器

w 位用于选择 Wy 寄存器。

a 位用于选择累加器回写目标和寻址模式。

注：

1. 在小数还是整数数据模式下工作由 CORCON.IF 定义。
2. 如果源是执行前修改或执行后修改的有效地址，则不允许对间接源（X 或 Y）和 AWB 间接目标使用相同的 W 寄存器。
3. 堆栈必须保持长字对齐。因此，只允许将[W15++] AWB 与长字 MAC 类指令一起使用。

指令字数：1

指令周期数：1

MSC		相乘并从累加器中减去乘积			
语法：{标号:}	MSC{.w}	Wx	, Wy,	A	{,AWB}
	MSC.l	[Wx]	, [Wy],	B	
		[Wx]+=kx	, [Wy]+=ky,		
		[Wx]-=kx	, [Wy]-=ky,		
		[Wx+=kx]	, [Wy+=ky],		
		[Wx-=kx]	, [Wy-=ky],		

.....（续）

MSC	相乘并从累加器中减去乘积
	[Wx+W12] , [Wy+W12],
操作数：	Wx ∈ {W0 ...W14}；Wy ∈ {W0 ...W14} 字模式：kx ∈ {-8, -6, -4, -2, 2, 4, 6, 8}；ky ∈ {-8, -6, -4, -2, 2, 4, 6, 8}； 长字模式：kx ∈ {-16, -12, -8, -4, 4, 8, 12, 16}；ky ∈ {-16, -12, -8, -4, 4, 8, 12, 16}； AWB ∈ {W0, W1, W2, W3, W13, [W13++]，[W15++][3]}
操作：	(ACC(A 或 B)) – ((Wx) * (Wy)) → ACC(A 或 B)； 舍入后的(ACC(B 或 A)) → AWB 当使用执行前修改/执行后修改的间接寻址时： (Wx)+kx→Wx 或(Wx)-kx→Wx； (Wy)+ky→Wy 或(Wy)-ky→Wy；
受影响的状态位：	OA 和 SA，*或者* OB 和 SB
指令编码：	1101 00AL wwww ssss IIIi iJJJ jjaa a111
说明：	对同时从 Wx 和 Wy 读取的数据或同时从 X 和 Y 地址空间获取的数据进行有符号/无符号（由 CORCON.US 定义）乘法。结果经过符号扩展（当至少一个操作数被视为有符号时）或零扩展（当两个操作数均为无符号时）至 72 位，然后从指定的累加器中减去。结果在累加器更新之前是否进行换算取决于是小数运算还是整数运算（由 CORCON.IF 定义）。小数运算将对结果进行换算以对齐操作数和累加器（msw）小数点。整数运算会将结果的 LSb 与累加器的 LSb 对齐。 对于字大小的操作数操作，保留 ACCx[31:0]（整数模式）或 ACCx[32:0]（小数模式）（见注 2）。 当 X 和 Y 地址空间中至少有一个选择间接寻址时，Wx 和 Wy 寄存器都会提供相应的间接地址。地址修改量值分别为 kx 和 ky，表示有效地址中要修改的数据字节数。 可选的 AWB 用于指定是直接还是间接存储非 MSC 操作目标的累加器舍入后的小数内容（32 位）。舍入模式由 CORCON.RND 定义。写入数据宽度由所选指令数据大小决定（见注 4）。当 DSP 引擎工作于整数模式时，不适合使用 AWB。 读取的数据可以是 16 位或 32 位值。所有间接地址修改都会相应地进行换算。 L 位用于选择字或长字操作。 A 位用于选择存放结果的累加器。 I 位用于选择操作 X 空间寻址模式。 i 位用于选择 kx 修改值。 J 位用于选择操作 Y 空间寻址模式。 j 位用于选择 ky 修改值。 s 位用于选择 Wx 寄存器 w 位用于选择 Wy 寄存器。 a 位用于选择累加器回写目标和寻址模式。 **注：** 1. 在小数还是整数数据模式下工作由 CORCON.IF 定义。 2. 当在小数模式下使用字大小的数据时，ACCx 的低位部分不受影响。因此，之前的（32 位数据）操作中可能存在的低位数据会得以保留。如果不需要该数据，用户应在初始化阶段清零 ACCx。 3. 如果源是执行前修改或执行后修改的有效地址，则不允许对间接源（X 或 Y）和 AWB 间接目标使用相同的 W 寄存器。 4. 堆栈必须保持长字对齐。因此，只允许将[W15++] AWB 与长字 MAC 类指令一起使用。
指令字数：	1
指令周期数：	1

MUL	f 与 Wn 相乘			
语法：	{标号:}	MUL.b	f,	Wns
		MUL{.w}		
		MUL.l		

.....（续）

MUL	f 与 Wn 相乘
操作数：	f ∈ [0 ...64 KB]；Wn ∈ [W0 ...W14]
操作：	如果是字节模式： (Wn)[7:0] * (f)[7:0] → 16'b0 和 W2[15:0]。 如果是字模式： (Wn)[15:0] * (f)[15:0] → W2 如果是长字模式： (Wn) * (f) → W2
受影响的状态位：	无
指令编码：	1011 101L ssss ffff ffff ffff ffff BU01
说明：	将 Wn 与文件寄存器进行无符号整数乘法，然后将结果写入默认的目标寄存器 W2。 对于长字操作，64 位结果的低 32 位写入 W2。 L 和 B 位用于选择操作数据宽度。 f 位用于选择文件寄存器的地址。
指令字数：	1
指令周期数：	1

MULISS/MULFSS	有符号-有符号乘法，结果存入累加器			
语法：	{标号:}	MULISS.w Wb,	Ws,	A
		MULFSS.w	[Ws],	B
		MULISS.l	[Ws++],	
		MULFSS.l	[Ws--],	
			[++Ws],	
			[--Ws],	
操作数：	Wb ∈ [W0 ...W14]；Ws ∈ [W0 ...W15]			
操作：	有符号(Wb) * 有符号(Ws) → ACC(A 或 B)[63:0] 8{ACC(A 或 B)[63]} → ACC(A 或 B)[71:64]			
受影响的状态位：	无			
指令编码：	1111 100L wwww ssss pppA UUUU UUUU I010			
说明：	对 Wb 与 Ws 的有符号内容执行字或长字整数（MULISS）或小数（MULFSS）乘法。源操作数被解释为二进制补码有符号值。 对于 MULISS.l，64 位结果将进行符号扩展并写入 ACCx[71:0]。 对于 MULFSS.l，64 位结果将进行符号扩展，左移一位（以对齐结果和累加器小数点），并写入 ACCx[71:0]。ACCx[0]将总是清零。 对于 MULISS.w，32 位结果将进行符号扩展并写入 ACCx[71:32]。ACCx[31:0]将保存乘法结果。 对于 MULFSS.w，32 位结果将进行符号扩展，左移 33 位（以对齐结果和累加器小数点），并写入 ACCx[71:33]。ACCx[32:0]将总是清零。 L 位用于选择字或长字操作。 I 位用于在整数运算与小数运算之间进行选择。 A 位用于选择目标累加器。 s 位用于选择源寄存器。 w 位用于选择基准寄存器。 p 位用于选择源寻址模式。 **注：**乘法器模式位（CORCON.US）的状态对该指令的操作不会产生任何影响。			
指令字数：	1			
指令周期数：	1			

MULISU/MULFSU		有符号-无符号乘法，结果存入累加器		
语法：	{标号:}	MULISU.w Wb,	Ws,	A
		MULFSU.w	[Ws],	B
		MULISU.l	[Ws++],	
		MULFSU.l	[Ws--],	
			[++Ws],	
			[--Ws],	

操作数： Wb ∈ [W0 ...W14]；Ws ∈ [W0 ...W15]

操作： 有符号(Wb) * 无符号(Ws) → ACC(A 或 B)[63:0]
8{ACC(A 或 B)[63]} → ACC(A 或 B)[71:64]

受影响的状态位： 无

指令编码：

1111	101L	wwww	ssss	pppA	UUUU	UUUU	I110

说明： 对 Wb 的有符号内容与 Ws 的无符号内容执行字或长字整数（MULISU）或小数（MULFSU）乘法。Wb 被解释为二进制补码有符号值。
对于 MULISU.l，64 位结果将进行符号扩展并写入 ACCx[71:0]。

对于 MULFSU.l，64 位结果将进行符号扩展，左移一位（以对齐结果和累加器小数点），并写入 ACCx[71:0]。ACCx[0]将总是清零。

对于 MULISU.w，32 位结果将进行符号扩展并写入 ACCx[71:32]。ACCx[31:0]将保存乘法结果。

对于 MULFSU.w，32 位结果将进行符号扩展，左移 33 位（以对齐结果和累加器小数点），并写入 ACCx[71:33]。ACCx[32:0]将总是清零。

L 位用于选择字或长字操作。

I 位用于在整数运算与小数运算之间进行选择。

A 位用于选择目标累加器。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

p 位用于选择源寻址模式。

注：

1. 乘法器模式位（CORCON.US）的状态对该指令的操作不会产生任何影响。
2. 无符号小数操作数包含一个整数位（即，最大值可达 1.999...）。因此，混合符号小数结果包含一个整数位（即，最大值可达 1.999...）。如果结果等于或超出范围±1.0，可能需要经过归一化处理后才能继续使用。

指令字数： 1

指令周期数： 1

MULISU/MULIUU		有符号/无符号-无符号立即数乘法，结果存入累加器		
语法：	{标号:}	MULISU.w Ws,	lit8,	A
		MULIUU.w [Ws],		B
		MULISU.l [Ws++],		
		MULIUU.l [Ws--],		
		[++Ws],		
		[--Ws],		

操作数： Ws ∈ [W0 ...W15]；lit8 ∈ [0 ... 255]

操作： MULISU：有符号(Ws) * 无符号 lit8 → ACC(A 或 B)[63:0]
8{ACC(A 或 B)[63]} → ACC(A 或 B)[71:64]

MULIUU：无符号(Ws) * 无符号 lit8 → ACC(A 或 B)[63:0]

8'b00 → ACC(A 或 B)[71:64]

注：立即数经过零扩展至操作所选的数据大小

受影响的状态位： 无

指令编码：

1111	111L	UUUU	ssss	pppA	kkkk	kkkk	1V10

.....（续）

MULISU/MULIUU	有符号/无符号-无符号立即数乘法，结果存入累加器

说明：

对 Ws 的有符号（MULISU）或无符号（MULIUU）内容与无符号 lit8 执行字或长字整数乘法。V 位设置为 1'b1 时选择有符号的 Ws 值，设置为 1'b0 时选择无符号的 Ws 值。
对于 MULISU.l，64 位结果将进行符号扩展并写入 ACCx[71:0]。

对于 MULIUU.l，64 位结果将进行零扩展并写入 ACCx[71:0]。

对于 MULISU.w，32 位结果将进行符号扩展并写入 ACCx[71:32]。ACCx[31:0]将保存乘法结果。

对于 MULIUU.w，32 位结果将进行零扩展并写入 ACCx[71:32]。ACCx[31:0]将保存乘法结果。

L 位用于选择字或长字操作。

V 位用于在有符号与无符号的 Ws 值之间进行选择。

A 位用于选择目标累加器。

s 位用于选择源寄存器。

p 位用于选择源寻址模式。

k 位用于确定 8 位立即数值。

注：

1. 乘法器模式位（CORCON.US）的状态对该指令的操作不会产生任何影响。
2. 这些指令不支持小数模式。

指令字数：1

指令周期数：1

MULIUU/MULFUU	无符号-无符号乘法，结果存入累加器

语法：	{标号:}	MULIUU.w Wb,	Ws,	A
		MULFUU.w	[Ws],	B
		MULIUU.l	[Ws++],	
		MULFUU.l	[Ws--],	
			[++Ws],	
			[--Ws],	

操作数：Wb ∈ [W0 ...W14]；Ws ∈ [W0 ...W15]

操作：无符号(Wb) * 无符号(Ws) → ACC(A 或 B)[63:0]
8'h00 → ACC(A 或 B)[71:64]

受影响的状态位：无

指令编码：

1111	101L	wwww	ssss	pppA	UUUU	UUUU	I010

.....（续）

MULIUU/MULFUU	无符号-无符号乘法，结果存入累加器

说明：

对 Wb 的无符号内容与 Ws 的有符号内容执行字或长字整数（MULIUU）或小数（MULFUU）乘法（见注 2）。

对于 MULIUU.l，64 位结果将进行零扩展并写入 ACCx[71:0]。

对于 MULFUU.l，64 位结果将进行零扩展，左移一位（以对齐结果和累加器小数点），并写入 ACCx[71:0]。ACCx[0]将总是清零。

对于 MULIUU.w，32 位结果将进行零扩展并写入 ACCx[71:32]。ACCx[31:0]将保存乘法结果。

对于 MULFUU.w，32 位结果将进行零扩展，左移 33 位（以对齐结果和累加器小数点），并写入 ACCx[71:33]。ACCx[32:0]将总是清零。

L 位用于选择字或长字操作。

I 位用于在整数运算与小数运算之间进行选择。

A 位用于选择目标累加器。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

p 位用于选择源寻址模式。

注：

1. 乘法器模式位（CORCON.US）的状态对该指令的操作不会产生任何影响。
2. 无符号小数操作数包含一个整数位（即，最大值可达 1.999...）。因此，无符号小数结果包含两个整数位（即，最大值可达 3.999...）。如果结果大于或等于 1.0，可能需要经过归一化处理后才能继续使用。

指令字数：1

指令周期数：1

MULIUS/MULFUS	无符号-有符号乘法，结果存入累加器

语法：	{标号:}	MULIUS.w Wb,	Ws,	A
		MULFUS.w	[Ws],	B
		MULIUS.l	[Ws++],	
		MULFUS.l	[Ws--],	
			[++Ws],	
			[--Ws],	

操作数：Wb ∈ [W0 ...W14]；Ws ∈ [W0 ...W15]

操作：无符号(Wb) * 有符号(Ws) → ACC(A 或 B)[63:0]
8{ACC(A 或 B)[63]} → ACC(A 或 B)[71:64]

受影响的状态位：无

指令编码：

1111	100L	wwww	ssss	pppA	UUUU	UUUU	I110

.....（续）

MULIUS/MULFUS	无符号-有符号乘法，结果存入累加器

说明：

对 Wb 的无符号内容与 Ws 的无符号内容执行字或长字整数（MULIUS）或小数（MULFUS）乘法（见注 2）。

对于 MULIUS.l，64 位结果将进行符号扩展并写入 ACCx[71:0]。

对于 MULFUS.l，64 位结果将进行符号扩展，左移一位（以对齐结果和累加器小数点），并写入 ACCx[71:0]。ACCx[0]将总是清零。

对于 MULIUS.w，32 位结果将进行符号扩展并写入 ACCx[71:32]。ACCx[31:0]将保存乘法结果。

对于 MULFUS.w，32 位结果将进行符号扩展，左移 33 位（以对齐结果和累加器小数点），并写入 ACCx[71:33]。ACCx[32:0]将总是清零。

L 位用于选择字或长字操作。

I 位用于在整数运算与小数运算之间进行选择。

A 位用于选择目标累加器。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

p 位用于选择源寻址模式。

注：

1. 乘法器模式位（CORCON.US）的状态对该指令的操作不会产生任何影响。
2. 无符号小数操作数包含一个整数位（即，最大值可达 1.999...）。因此，混合符号小数结果包含一个整数位（即，最大值可达 1.999...）。如果结果等于或超出范围±1.0，可能需要经过归一化处理后才能继续使用。

指令字数：1

指令周期数：1

MULISS/MULIUS	有符号/无符号-有符号立即数乘法，结果存入累加器

语法：

{标号:}	MULISS.w Ws,	slit8,	A
	MULIUS.w [Ws],		B
	MULISS.l [Ws++],		
	MULIUS.l [Ws--],		
	[++Ws],		
	[--Ws],		

操作数：Ws ∈ [W0 ...W15]；slit8 ∈ [-128 ...127]

操作：

MULxUS：无符号(Ws) * 有符号 slit8 → ACC(A 或 B)[63:0]
MULxSS：有符号(Ws) * 有符号 slit8 → ACC(A 或 B)[63:0]

8{ACC(A 或 B)[63]} → ACC(A 或 B)[71:64]

注：立即数经过符号扩展至操作所选的数据大小

受影响的状态位：无

指令编码：

1111	110L	UUUU	ssss	pppA	kkkk	kkkk	1V10

.....（续）

MULISS/MULIUS	有符号/无符号-有符号立即数乘法，结果存入累加器

说明：

对 Ws 的有符号（MULISS）或无符号（MULIUS）内容与有符号 slit8 执行字或长字整数乘法。V 位设置为 1'b1 时选择有符号的 Ws 值，设置为 1'b0 时选择无符号的 Ws 值。
对于 MULISS.l，64 位结果将进行符号扩展并写入 ACCx[71:0]。

对于 MULIUS.l，64 位结果将进行符号扩展并写入 ACCx[71:0]。

对于 MULISS.w，32 位结果将进行符号扩展并写入 ACCx[71:32]。ACCx[31:0]将保存乘法结果。

对于 MULIUS.w，32 位结果将进行符号扩展并写入 ACCx[71:32]。ACCx[31:0]将保存乘法结果。

L 位用于选择字或长字操作。

V 位用于在有符号与无符号的 Ws 值之间进行选择。

A 位用于选择目标累加器。

s 位用于选择源寄存器。

p 位用于选择源寻址模式。

k 位用于确定 8 位立即数值。

注：

1. 乘法器模式位（CORCON.US）的状态对该指令的操作不会产生任何影响。
2. 这些指令不支持小数模式。

指令字数：1

指令周期数：1

MULSS/MULUS	有符号/无符号-有符号立即数整数乘法

语法：

{标号:}	MULSS.d	Ws,	slit8,	Wnd
	MULSS.l [Ws],			
	MULSS.w [Ws++],			
	MULUS.d	[Ws--],		
	MULUS.l [++Ws],			
	MULUS.w [--Ws],			

操作数：Ws ∈ [W0 ...W15]；slit8 ∈ [-128 ...127]；Wnd ∈ [W0 ...W14]

操作：

MULSS.d：有符号(Ws) * 有符号 slit8 → {Wnd+1[31:0]和 Wnd[31:0]}
MULSS.l：有符号(Ws) * 有符号 slit8 → {Wnd[31:0]}

MULSS.w：有符号(Ws[15:0]) * 有符号 slit8 → {Wnd[31:0]}

MULUS.d：无符号(Ws) * 有符号 slit8 → {Wnd+1[31:0]和 Wnd[31:0]}

MULUS.l：无符号(Ws) * 有符号 slit8 → {Wnd[31:0]}

MULUS.w：无符号(Ws[15:0]) * 有符号 slit8 → {Wnd[31:0]}

注：立即数经过符号扩展至操作所选的数据大小

受影响的状态位：无

指令编码：

1111	010L	dddd	ssss	pppE	kkkk	kkkk	0V10

.....（续）

MULSS/MULUS	有符号/无符号-有符号立即数整数乘法

说明：

对 Ws 的有符号或无符号内容与符号扩展 slit8 执行 32 位 x 32 位或 16 位 x 16 位整数乘法。V 位设置为 1'b1 时选择有符号的 Ws 值，设置为 1'b0 时选择无符号的 Ws 值。
对于 MULSS.d 和 MULUS.d（L = 1，E = 1），将对 Ws 与符号扩展 slit8 执行 32 位 x 32 位乘法。64 位结果的高 32 位将写入 Wnd+1，低 32 位将写入 Wnd。

对于 MULSS.l 和 MULUS.l（L = 1，E = 0），将对 Ws 与符号扩展 slit8 执行 32 位 x 32 位乘法。64 位结果的低 32 位将写入 Wnd。

对于 MULSS.w 和 MULUS.w（L = 0，E = 0），将对 Ws 的 lsw 与符号扩展 slit8 执行 16 位 x 16 位乘法。结果将为 32 位并写入 Wnd。

L 位用于选择操作数数据大小。

V 位用于在有符号与无符号的 Ws 值之间进行选择。

E 位用于在写入 32 位与 64 位结果之间进行选择。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

k 位用于确定 8 位立即数值。

注：

1. 乘法器模式位（CORCON.US）的状态对该指令的操作不会产生任何影响。
2. 如果写入 32 位结果，用户需负责确保结果不会溢出到符号位 Wnd[31]中。

指令字数：1

指令周期数：1

MULSS	有符号-有符号整数乘法

语法：

{标号:}	MULSS.d MULSS.l MULSS.w	Wb,	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	Wnd

操作数：Wb ∈ [W0 ...W15]；Ws ∈ [W0 ...W15]；Wnd ∈ [W0 ...W14]

操作：
MULSS.d：有符号(Wb) * 有符号(Ws) → {Wnd+1[31:0]和 Wnd[31:0]}
MULSS.l：有符号(Wb) * 有符号(Ws) → {Wnd[31:0]}

MULSS.w：有符号(Wb[15:0]) * 有符号(Ws[15:0]) → {Wnd[31:0]}

受影响的状态位：无

指令编码：

S001	010L	dddd	ssss	pppE	UUww	wwUU	0000

.....（续）

MULSS	有符号-有符号整数乘法

说明：

对 Wb 与 Ws 的有符号内容执行 16 位 x 16 位或 32 位 x 32 位整数乘法。
对于 MULSS.d（L = 1，E = 1），将对 Wb 与 Ws 执行 32 位 x 32 位乘法。64 位结果的高 32 位将写入 Wnd+1，低 32 位将写入 Wnd。

对于 MULSS.l（L = 1，E = 0），将对 Wb 与 Ws 执行 32 位 x 32 位乘法。64 位结果的低 32 位将写入 Wnd。

对于 MULSS.w（L = 0，E = 0），将对 Wb 的 lsw 与 Ws 的 lsw 执行 16 位 x 16 位乘法。结果将为 32 位并写入 Wnd。

两个源操作数均被解释为二进制补码整数有符号值。

S 位用于选择指令大小。

L 位用于选择操作数数据大小。

E 位用于在写入 32 位与 64 位结果之间进行选择。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

注：

1. 乘法器模式位（CORCON.US）的状态对该指令的操作不会产生任何影响。
2. 如果写入 32 位结果，用户需负责确保结果不会溢出到符号位 Wnd[31]中。

指令字数：1 或 0.5

指令周期数：1

MULSU	有符号-无符号整数乘法

语法：

{标号:}	MULSU.d MULSU.l MULSU.w	Wb,	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	Wnd

操作数：Wb ∈ [W0 ...W15]；Ws ∈ [W0 ...W15]；Wnd ∈ [W0 ...W14]

操作：
MULSU.d：有符号(Wb) * 无符号(Ws) → {Wnd+1[31:0]和 Wnd[31:0]}
MULSU.l：有符号(Wb) * 无符号(Ws) → {Wnd[31:0]}

MULSU.w：有符号(Wb[15:0]) * 无符号(Ws[15:0]) → {Wnd[31:0]}

受影响的状态位：无

指令编码：

1001	011L	dddd	ssss	pppE	UUww	wwUU	0100

.....（续）

MULSU	有符号-无符号整数乘法
说明：	对 Wb 的有符号内容与 Ws 的无符号内容执行 16 位 x 16 位或 32 位 x 32 位整数乘法。Wb 被解释为二进制补码有符号值。 对于 MULSU.d（L = 1，E = 1），将对 Wb 与 Ws 执行 32 位 x 32 位乘法。64 位结果的高 32 位将写入 Wnd+1，低 32 位将写入 Wnd。 对于 MULSU.l（L = 1，E = 0），将对 Wb 与 Ws 执行 32 位 x 32 位乘法。结果的低 32 位将写入 Wnd。 对于 MULSU.w（L = 0，E = 0），将对 Wb 的 lsw 与 Ws 的 lsw 执行 16 位 x 16 位乘法。结果将为 32 位并写入 Wnd。 L 位用于选择操作数数据大小。 E 位用于在写入 32 位与 64 位结果之间进行选择。 s 位用于选择源寄存器。 w 位用于选择基准寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 **注：** 1. 乘法器模式位（CORCON 中的 CORCON.US）的状态对该指令的操作不会产生任何影响。 2. 如果写入 32 位结果，用户需负责确保结果不会溢出到符号位 Wnd[31]中。
指令字数：	1
指令周期数：	1

MULSU/MULUU		有符号/无符号-无符号短立即数整数乘法			
语法：	{标号:}	MULSU.d MULSU.l MULSU.w MULUU.d MULUU.l MULUU.w	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	lit8,	Wnd
操作数：	Ws ∈ [W0 ...W15]；lit8 ∈ [0...255]；Wnd ∈ [W0 ...W14]				
操作：	MULSU.d：有符号(Ws) * 无符号 lit8 → {Wnd+1[31:0]和 Wnd[31:0]} MULSU.l：有符号(Ws) * 无符号 lit8 → {Wnd[31:0]} MULSU.w：有符号(Ws[15:0]) * 无符号 lit8 → {Wnd[31:0]} MULUU.d：无符号(Ws) * 无符号 lit8 → {Wnd+1[31:0]和 Wnd[31:0]} MULUU.l：无符号(Ws) * 无符号 lit8 → {Wnd[31:0]} MULUU.w：无符号(Ws[15:0]) * 无符号 lit8 → {Wnd[31:0]} 注：立即数经过零扩展至操作所选的数据大小				
受影响的状态位：	无				
指令编码：	1111 011L dddd ssss pppE kkkk kkkk 0V10				

.....（续）

MULSU/MULUU	有符号/无符号-无符号短立即数整数乘法

说明：

对 Ws 的有符号或无符号内容与无符号 lit8 执行 16 位 x 16 位或 32 位 x 32 位整数乘法。V 位设置为 1'b1 时选择有符号的 Ws 值，设置为 1'b0 时选择无符号的 Ws 值。
对于 MULSU.d 和 MULUU.d（L = 1，E = 1），将对 Ws 与零扩展 lit8 执行 32 位 x 32 位乘法。64 位结果的高 32 位将写入 Wnd+1，低 32 位将写入 Wnd。

对于 MULSU.l 和 MULUU.l（L = 1，E = 0），将对 Ws 与零扩展 lit8 执行 32 位 x 32 位乘法。64 位结果的低 32 位将写入 Wnd。

对于 MULSU.w 和 MULUU.w（L = 0，E = 0），将对 Ws 的 lsw 与零扩展 lit8 执行 16 位 x 16 位乘法。结果将为 32 位并写入 Wnd。

L 位用于选择操作数数据大小。

V 位用于在有符号与无符号的 Ws 值之间进行选择。

E 位用于在写入 32 位与 64 位结果之间进行选择。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

k 位用于确定 8 位立即数值。

注：

1. 乘法器模式位（CORCON.US）的状态对该指令的操作不会产生任何影响。
2. 如果写入有符号 32 位结果，用户需负责确保结果不会溢出到符号位 Wnd[31]中。

指令字数：1

指令周期数：1

MULUS	无符号-有符号整数乘法

语法：	{标号:}	MULUS.d MULUS.l MULUS.w	Wb,	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	Wnd

操作数：Wb ∈ [W0 ...W15]；Ws ∈ [W0 ...W15]；Wnd ∈ [W0 ...W14]

操作：
MULUS.d：无符号(Wb) * 有符号(Ws) → {Wnd+1[31:0]和 Wnd[31:0]}
MULUS.l：无符号(Wb) * 有符号(Ws) → {Wnd[31:0]}

MULUS.w：无符号(Wb[15:0]) * 有符号(Ws[15:0]) → {Wnd[31:0]}

受影响的状态位：无

指令编码：

1001	010L	dddd	ssss	pppE	UUww	wwUU	0100

.....（续）

MULUS	无符号-有符号整数乘法

说明：

对 Wb 的无符号内容与 Ws 的有符号内容执行 16 位 x 16 位或 32 位 x 32 位整数乘法。Ws 被解释为二进制补码有符号值。
对于 MULUS.d（L = 1，E = 1），将对 Wb 与 Ws 执行 32 位 x 32 位乘法。64 位结果的高 32 位将写入 Wnd+1，低 32 位将写入 Wnd。

对于 MULUS.l（L = 1，E = 0），将对 Wb 与 Ws 执行 32 位 x 32 位乘法。结果的低 32 位将写入 Wnd。

对于 MULUS.w（L = 0，E = 0），将对 Wb 的 lsw 与 Ws 的 lsw 执行 16 位 x 16 位乘法。结果将为 32 位并写入 Wnd。

L 位用于选择操作数数据大小。

E 位用于在写入 32 位与 64 位结果之间进行选择。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

注：

1. 乘法器模式位（CORCON.US）的状态对该指令的操作不会产生任何影响。
2. 如果写入 32 位结果，用户需负责确保结果不会溢出到符号位 Wnd[31]中。

指令字数：1

指令周期数：1

MULUU	无符号-无符号整数乘法

语法：

{标号:}	MULUU.d MULUU.l MULUU.w	Wb,	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	Wnd

操作数：Wb ∈ [W0 ...W15]；Ws ∈ [W0 ...W15]；Wnd ∈ [W0 ...W14]

操作：
MULUU.d：无符号(Wb) * 无符号(Ws) → {Wnd+1[31:0]和 Wnd[31:0]}
MULUU.l：无符号(Wb) * 无符号(Ws) → {Wnd[31:0]}

MULUU.w：无符号(Wb[15:0]) * 无符号(Ws[15:0]) → {Wnd[31:0]}

受影响的状态位：无

指令编码：

S001	011L	dddd	ssss	pppE	UUww	wwUU	0000

说明：

对 Wb 与 Ws 的无符号内容执行 16 位 x 16 位或 32 位 x 32 位整数乘法。
对于 MULUU.d（L = 1，E = 1），将对 Wb 与 Ws 执行 32 位 x 32 位乘法。64 位结果的高 32 位将写入 Wnd+1，低 32 位将写入 Wnd。

对于 MULUU.l（L = 1，E = 0），将对 Wb 与 Ws 执行 32 位 x 32 位乘法。结果的低 32 位将写入 Wnd。

对于 MULUU.w（L = 0，E = 0），将对 Wb 的 lsw 与 Ws 的 lsw 执行 16 位 x 16 位乘法。结果将为 32 位并写入 Wnd。

S 位用于选择指令大小。

L 位用于选择操作数数据大小。

E 位用于在写入 32 位与 64 位结果之间进行选择。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

注：

1. 乘法器模式位（CORCON.US）的状态对该指令的操作不会产生任何影响。

.....（续）

MULUU	无符号-无符号整数乘法
指令字数：	1 或 0.5
指令周期数：	1

4.7 指令说明（N 至 XORWF）

NEG		对累加器内容求补						
语法：	{标号:}	NEG	A					
			B					
操作数：	无							
操作：	如果为 NEGAB A，则-ACCA → ACCA 如果为 NEGAB B，则-ACCB → ACCB							
受影响的状态位：	OA 和 SA，或者 OB 和 SB							
指令编码：	0111	001A	UUUU	1010				
说明：	计算指定累加器内容的二进制补码，并将结果存回指定累加器。 A 位用于指定选择的累加器。如果选择 AccA，OA 和 SA 会受到影响。如果选择 AccB，OB 和 SB 会受到影响。							
指令字数：	0.5							
指令周期数：	1							

NEG		对 f 内容求补							
语法：	{标号:}	NEG.b	f	{,Wnd}	{,WREG}				
		NEG.bz							
		NEG{.w}							
		NEG.l							
操作数：	f ∈ [0 ...64 KB]；Wnd ∈ [W0 ...W14]								
操作：	(f) + 1 → 由 D 指定的目标寄存器								
受影响的状态位：	C、N、OV 和 Z								
指令编码：	1101	101L	dddd	ffff	ffff	ffff	ffff	BD01	
说明：	计算文件寄存器内容的二进制补码，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wnd，则 D = 0 并将结果存放在 Wnd 中；否则，D = 1 并将结果存放在文件寄存器中。 L 和 B 位用于选择操作数据宽度。 D 位用于选择目标寄存器。 f 位用于选择文件寄存器的地址。 d 位用于选择工作寄存器。								
指令字数：	1								
指令周期数：	1								

NEOP		无执行时间空操作			
语法：	{标号:}	NEOP			
操作数：	无				
操作：	空操作				
受影响的状态位：	无				
指令编码：	0000	0000	UUUU	UUUU	
说明：	不执行任何操作。该指令也不会耗费任何执行时间。当某条指令无法与另一条 16 位指令配对（以保持对齐）时，可以使用该指令来填充 16 位指令（以使用 32 位字）。				
指令字数：	0.5				
指令周期数：	0				

NOP		空操作			
语法：	{标号:}	NOP			

.....（续）

NOP	空操作

操作数：无
操作：空操作
受影响的状态位：无

指令编码：	0000	0001	UUUU	UUUU

说明：不执行任何操作。
指令字数：1
指令周期数：1

NOPR	空操作（32 位）

语法：{标号:} NOPR

操作数：无
操作：空操作
受影响的状态位：无

指令编码：	1111	111U	UUUU	UUUU	UUUU	UUUU	UUUU	UU11

说明：不执行任何操作。
指令字数：1
指令周期数：1

NORM	将累加器内容归一化

语法：{标号:} NORM{.w} A, Wd
NORM.l B, [Wd]
[Wd++]
[Wd--]
[++Wd]
[--Wd]
[Wd+Wb]

操作数：Wd ∈ [W0 ...W15]，Wb ∈ [W0 ...W15]
操作：见下文
受影响的状态位：OA 或 OB、N 和 Z

指令编码：	1100	10AL	wwww	dddd	UUUq	qqUU	UUUU	1111

.....（续）

NORM	将累加器内容归一化

说明：

将目标累加器的内容归一化。如果累加器包含溢出值，则累加器的内容将进行右移，右移位数为消除溢出所需的最小位数。如果累加器不包含溢出值，则累加器的内容将进行左移，左移位数为在不溢出的前提下生成最大小数数据值所需的最小位数。
如果无法将目标累加器的内容归一化（即，目标累加器的内容已经归一化，或者为全 0 或全 1），则 Wd 清零，Z 位置 1（并且 N 位清零）。目标累加器的内容不受影响。

如果可以将目标累加器的内容归一化，则将目标累加器的内容归一化所需的指数（移位值）写入 Wd。结果为正表明将累加器的内容归一化需要进行右移。结果为负表明将累加器的内容归一化需要进行左移。通过设置 N 位反映结果的符号，Z 位清零。

L 位用于选择字或长字 Wd 目标。

A 位用于指定目标累加器。

d 位用于选择目标寄存器。

q 位用于选择目标寻址模式。

w 位用于定义偏移量 Wb。

注：

1.

根据目标累加器的内容设置 OA 或 OB 状态位。由于 NORM 可消除任何溢出，因此 OA 或 OB 将总是清零。SA/SB 为“粘住”位，因此只要在执行 NORM 之前置 1 就会保持置 1 状态，永远不会受到该指令的影响。

指令字数：1

指令周期数：1

POP	将返回栈顶的内容弹出

语法：{标号:} POP f

操作数：f ∈ [0 ...1 MB]

操作：(W15) - 4 → W15;
(TOS) → (f,2'b00)

受影响的状态位：无

指令编码：

1011	1101	ffff	ffff	ffff	ffff	ffff	UU01

说明：

先将堆栈指针（W15）递减，然后将栈顶（TOS）值从堆栈中弹出并写入文件寄存器。
f 位用于选择文件寄存器的地址。

注：

1. 该指令只能工作于长字模式。因此，文件地址的低 2 位将总是为 2'b00。
2. 可访问的文件地址空间为 1 MB。

指令字数：1

指令周期数：1

POP	将栈顶内容弹出至 FPU 协处理器寄存器

语法：{标号:} POP.l Fd
FSR
FCR
FEAR

操作数：Fd ∈ [F0 ...F31]

操作：(W15)-4 → W15
(TOS) → (Fd、FSR、FCR 或 FEAR)

受影响的状态位：无

指令编码：

0010	111d	dddd	zzRU

.....（续）

POP	将栈顶内容弹出至 FPU 协处理器寄存器

说明：将系统堆栈顶部的内容传送至协处理器目标寄存器。隐含使用系统堆栈指针（[--W15]）。上面所示的语法对应于浮点协处理器寄存器。
d 位用于选择目标寄存器。

z 位用于选择目标协处理器。

R 位用于在 F-reg 与 FPU 特殊寄存器之间进行选择。

注：

1. 该指令只能工作于长字模式。
2. 不支持 FSRH，因为它会清除 FSR[15:0]中的异常状态。

指令字数：1

指令周期数：1

POP	将栈顶内容弹出至工作（W）寄存器

语法：

{标号:}	POP.l	Wnd
	MOV.l	[--Ws], Wnd
	MOV{.w}	

操作数：Ws ∈ [W0 ...W15] [1,2]；Wnd ∈ [W0 ...W15]

操作：
POP.l [1,2]：(W15) - 4 → W15；(TOS) → Wnd
MOV.l [2]：(Ws) - 4 → Ws；(EAs) → Wnd

MOV.w [1]：(Ws) - 2 → Ws；(EAs) → Wnd

受影响的状态位：无

指令编码：

```
0001    110L    dddd    ssss
```

说明：将 LIFO 堆栈数据结构（或数据存储器）顶部的内容传送至目标工作寄存器 Wnd。POP.l 隐含使用系统堆栈指针（[--W15]）。
L 位用于选择操作数据宽度。

s 位用于选择工作源寄存器。

d 位用于选择工作目标寄存器。

指令字数：0.5

指令周期数：1

PUSH	压入返回栈顶（TOS）

语法：

{标号:}	PUSH	f

操作数：f ∈ [0 ...1 MB]

操作：
(f) → (TOS);
(W15)+4 → W15

受影响的状态位：无

指令编码：

```
1011    1111    ffff    ffff    ffff    ffff    ffff    UU01
```

说明：将文件寄存器内容写入栈顶（TOS）位置。堆栈指针（W15）随后递增。
f 位用于选择文件寄存器的地址。

注：

1. 该指令只能工作于长字模式。因此，文件地址的低 2 位将总是为 2'b00。
2. 可访问的文件地址空间为 1 MB。

指令字数：1

指令周期数：1

PUSH		将 FPU 协处理器寄存器的内容压入栈顶	
语法：	{标号:}	PUSH.l	Fs FSR FSRH FCR FEAR
操作数：	Fs ∈ [F0 ...F31]		
操作：	(Fs、FSR、FSRH、FCR 或 FEAR) → (TOS) (W15)+4 → W15		
受影响的状态位：	无		
指令编码：	0010 011s ssss zzRU		
说明：	将源浮点寄存器的内容传送至系统堆栈顶部。隐含使用系统堆栈指针（[W15++]）。上面所示的语法对应于浮点协处理器寄存器。 选择 FSRH 时，将长字值{FSR[31:16], 16'b0}压入堆栈。FSR[31:16]仅包含 FCPS/FCPQ 和 FTST 指令的状态。 s 位用于选择浮点源寄存器。 z 位用于选择目标协处理器。 R 位用于在 F-reg 与 FPU 特殊寄存器之间进行选择。 **注：** 1. 该指令只能工作于长字模式。		
指令字数：	0.5		
指令周期数：	1		

PUSH		将工作（W）寄存器的内容压入栈顶		
语法：	{标号:}	PUSH.l	Wns	
		MOV.l MOV{.w}	Wns,	[Wd++]
操作数：	Wns ∈ [W0 ...W15]；Wd ∈ [W0 ... W15] [1, 2]			
操作：	PUSH.l [1,2]：Wns → (TOS)；(W15)+4 → W15 MOV.l [2]：Wns → (EAd)；(Wd)+4 → Wd MOV.w [1]：Wns → (EAd)；(Wd)+2 → Wd			
受影响的状态位：	无			
指令编码：	0001 111L dddd ssss			
说明：	将源工作寄存器 Wns 的内容传送至 LIFO 堆栈数据结构（或数据存储器）的顶部。PUSH.l 隐含使用系统堆栈指针（[W15++]）。 L 位用于选择操作数据宽度。 s 位用于选择工作源寄存器。 d 位用于选择工作目标寄存器。			
指令字数：	0.5			
指令周期数：	1			

PWRSAV		进入节能模式	
语法：	{标号:}	PWRSAV	lit1
操作数：	lit1 ∈ [0 ... 1]		

.....（续）

PWRSAV	进入节能模式
操作：	0 → WDT 1 → WDTO 0 → SLEEP 0 → IDLE 进入空闲或休眠模式
受影响的状态位：	SLEEP 和 IDLE
指令编码：	0111 001U UUUk 0111
说明：	如果 lit1 = 0，器件进入休眠模式。 如果 lit1 = 1，器件进入空闲模式。 WDT 复位。SLEEP 和 IDLE 状态位（位于 RCON 寄存器内）清零。如果选择休眠模式，则器件关闭。振荡器源停止工作。如果选择空闲模式，则 CPU 关闭，但外设继续工作。 处理器将通过中断、复位或看门狗超时退出休眠或空闲模式。 如果退出空闲模式，则重新将时钟源加到 CPU。如果退出休眠模式，则重启时钟源。 如果从休眠模式唤醒：1 → SLEEP（在 RCON 寄存器中） 如果从空闲模式唤醒：1 → IDLE（在 RCON 寄存器中） 如果通过 WDT 超时唤醒：1 → WDTO k 位用于选择节能模式。
指令字数：	0.5
指令周期数：	2

RCALL	相对调用
语法：	{标号:} RCALL 标号
操作数：	标号由链接器解析为有符号字偏移量（slit20）
操作：	(PC) + 4 → PC, (8'b0 和 PC[23:2]) → TOS[31:2]；2'b00 → TOS[1:0]， (W15) + 4 → W15， PC+ 2*slit20 → PC； NOP → 指令寄存器
受影响的状态位：	无
指令编码：	1101 010U nnnn nnnn nnnn nnnn nnnn 0010
说明：	对于 32 位或 16 位指令，在向前或向后 1 MB 的转移地址范围内进行子程序调用。 长字对齐的返回地址(PC+4)被压入系统堆栈。将二进制补码字节偏移量值“2*slit20”（PC 偏移量）加到 PC。由于 PC 将递增以便取出下一条指令，所以新地址将为(PC+4) + 2*slit20。 n 位构成一个有符号立即数，用于指定自(PC + 4)的转移量（PS 字数）。
指令字数：	1
指令周期数：	1

RCALL	计算调用
语法：	{标号:} RCALL Wns
操作数：	Wns ∈ [W0 ...W15]

.....（续）

RCALL	计算调用
操作：	(PC) + 4 → PC, (8'b0, PC[23:0]) → TOS[31:0]， (W15) + 4 → W15， (PC) + 2*Wns[19:0] → PC； NOP → 指令寄存器
受影响的状态位：	无
指令编码：	1101 011U UUUU ssss UUUU UUUU UUUU 0110
说明：	对于任意大小的指令，在向前或向后 1 MB 的转移地址范围内进行计算子程序调用。 长字对齐的返回地址(PC+4)被压入系统堆栈。Wns[19:0]中的有符号值表示自当前 PC 的 PS（16 位）字偏移量。将该值乘以 2 创建的字节地址随后会添加到 PC 的内容中以形成目标地址。因此，Wn 必须包含一个有符号值，用于指定调用时自(PC+4)的偏移量（PS 字数）。 RCALLW 是一条双周期指令。 s 位用于指定源寄存器。 **注：**如果 Wns[31:19]不为全 0 或全 1，则将启动地址错误陷阱。
指令字数：	1
指令周期数：	2 **注：**如果有异常等待处理，则不会执行取出的调用目标指令，从而使有效指令执行时间为一个周期。

REPEAT	重复执行下一条指令，重复次数由立即数确定
语法：	{标号:} REPEAT lit20
操作数：	lit20 ∈ [0 ... 1048575]
操作：	(12'h000 和 lit20) → RCOUNT[31:0]（循环计数寄存器） (PC) + 4 → PC 1 → RA（使能代码循环）
受影响的状态位：	RA（如果 lit20 > 0）
指令编码：	1101 010U kkkk kkkk kkkk kkkk kkkk 0110
说明：	执行紧跟 REPEAT 指令之后的指令(lit20+1)次。 在所有重复执行过程中，重复执行的指令将被保存在指令寄存器中且该指令只被取一次（在 REPEAT 指令期间，符合预期）。最后一次重复执行指令时会取下一条指令。 每次重复执行期间，重复计数会递减。当该计数等于 0 时（在倒数第二次循环期间），PC 递增并重新使能取指（RA = 0），以便能够在最后一次重复执行期间取下一条指令并继续正常执行。 可以在任何一次重复执行之前通过任何中断将重复执行的指令中断。请注意，用户必须保存并恢复 RCOUNT 以支持嵌套 REPEAT 指令（例如，从中断服务程序内）。 k 位是用于指定循环计数值的无符号立即数。 **注：** 1. 如果 lit20 = 0，则 RA 不置 1，后续指令正常执行（相当于循环计数为 1）。
指令字数：	1
指令周期数：	1

REPEAT	重复执行下一条指令，重复次数由短立即数确定
语法：	{标号:} REPEAT lit5
操作数：	lit5 ∈ [0 ...31]

.....（续）

REPEAT	重复执行下一条指令，重复次数由短立即数确定
操作：	(27'h000000 和 lit5) → RCOUNT[31:0]（循环计数寄存器） (PC) + 2 → PC 1 → RA（使能代码循环）
受影响的状态位：	RA（如果 lit5 > 0）
指令编码：	0111 001k kkkk 1111
说明：	执行紧跟 REPEAT 指令之后的指令(lit5+1)次。如果 lit5 > 31，请汇编为 REPEAT 指令。 在所有重复执行过程中，重复执行的指令将被保存在指令寄存器中且该指令只被取一次（在 REPEAT 指令期间，符合预期）。最后一次重复执行指令时会取下一条指令。 每次重复执行期间，重复计数会递减。当该计数等于 0 时（在倒数第二次循环期间），PC 递增并重新使能取指（RA = 0），以便能够在最后一次重复执行期间取下一条指令并继续正常执行。 可以在任何一次重复执行之前通过任何中断将重复执行的指令中断。请注意，用户必须保存并恢复 RCOUNT 以支持嵌套 REPEAT 指令（例如，从中断服务程序内）。 k 位是用于指定循环计数值的无符号立即数。 **注：**如果 lit5 = 0，则 RA 不置 1，后续指令正常执行（相当于循环计数为 1）
指令字数：	0.5
指令周期数：	1

REPEAT	重复执行下一条指令，重复次数为变量
语法：	{标号:} REPEAT Wn
操作数：	Wn ∈ [W0 ...W15]
操作：	(Wn[31:0]) → RCOUNT[31:0]（循环计数寄存器） (PC) + 2 → PC 1 → RA（使能代码循环）
受影响的状态位：	RA（如果(Wn) > 0）
指令编码：	1101 100U UUUU ssss UUUU UUUU UUUU U010
说明：	执行紧跟 REPEAT 指令之后的指令(Wn)+1 次。在所有重复执行过程中，重复执行的指令将被保存在指令寄存器中且该指令只被取一次（在 REPEAT 指令期间，符合预期）。最后一次重复执行指令时会取下一条指令。 每次重复执行期间，重复计数会递减。当该计数等于 0 时（在倒数第二次循环期间），PC 递增并重新使能取指（RA = 0），以便能够在最后一次重复执行期间取下一条指令并继续正常执行。 可以在任何一次重复执行之前通过任何中断将重复执行的指令中断。请注意，用户必须保存并恢复 RCOUNT 以支持嵌套 REPEAT 指令（例如，从中断服务程序内）。 s 位用于指定包含循环计数值的 Wn 寄存器。 **注：**如果 Wn = 0，则 RA 不置 1，后续指令正常执行（相当于循环计数为 1）。
指令字数：	1
指令周期数：	1

RESET	复位
语法：	{标号:} RESET
操作数：	无
操作：	将所有受 MCLR 复位影响的寄存器和标志位的内容强制为其复位状态。
受影响的状态位：	无
指令编码：	1111 101U UUUU UUUU UUUU UUUU UUUU 0011
说明：	该指令提供一种执行软件复位的方法。
指令字数：	1

.....（续）

RESET		复位
指令周期数：	1	

RETFIE		从中断返回
语法：	{标号:}	RETFIE
操作数：	无	
操作：	(W15) - 4 → W15 TOS[23:1] → (PC[23:1])， (W15) - 4 → W15 TOS[15:0] → (SR[31:0])， 1'b0 → PC[0] NOP → 指令寄存器	
受影响的状态位：	OA、OB、SA、SB、(OAB)、(SAB)、IPL[3:0]、RA、N、OV、Z 和 C	
指令编码：	0111 001U UUUU 0000	
说明：	从中断服务程序返回至 32 位或 16 位指令。 执行出栈操作，将栈顶（TOS）内容装入 PC[23:1]（PC[0]总是清零）。再次执行出栈操作，将栈顶（TOS）内容装入 SR。 此外，该指令还将管理硬件现场切换（基于 IPL）。 **注：** 1. OAB 和 SAB 将分别反映 OA/OB 和 SA/SB 在返回后的状态。	
指令字数：	0.5	
指令周期数：	4 **注：** 1. 如果有异常等待处理，则不会执行取出的返回 PC 指令，从而使有效指令执行时间（对应于延时）为一个周期。	

RETLW		返回并将立即数存入 Wd		
语法：	{标号:}	RETLW.b	lit16,	Wnd
		RETLW.bz		
		RETLW{.w}		
		RETLW.l		
操作数：	Wnd ∈ [W0 ...W14]；lit16 ∈ [0 ...65535]			
操作：	(W15) - 4 → W15 TOS[23:1] → (PC[23:1])， 1'b0 → PC[0] 字节模式：lit16[7:0] → Wnd[7:0] 扩展字节模式：24'b0 和 lit16[7:0] → Wnd 字或长字模式：16'b0 和 lit16 → Wnd NOP → 指令寄存器			
受影响的状态位：	无			
指令编码：	1100 110L dddd kkkk kkkk kkkk kkkk BU10			

.....（续）

RETLW	返回并将立即数存入 Wd

说明：返回并将立即数存入 Wd。
当工作于长字、字或扩展字节模式时，立即数会经过零扩展至 32 位之后再写入 Wnd。
当工作于字节模式时，立即数装入 Wnd 的 LSb，而 Wnd 的 MSb 保持不变。
L 和 B 位用于选择操作数据宽度。
d 位用于选择目标寄存器。
k 位用于定义立即数。

注：

1. W15 被排除在返回立即数值的有效目标之外。
2. 字和长字工作模式是等效的。
3. 如果有异常等待处理，则不会执行取出的返回 PC 指令，从而使有效指令执行时间（对应于延时）为一个周期。

指令字数：1
指令周期数：3（见注 3）

RETURN	返回

语法：{标号:} RETURN

操作数：无
操作：(W15) - 4 → W15
TOS[23:1] → (PC[23:1])
1'b0 → PC[0]
NOP → 指令寄存器

受影响的状态位：无
指令编码：`0111 001U UUUU 0001`
说明：从子程序返回至 32 位或 16 位指令。
执行出栈操作，将栈顶（TOS）内容装入 PC[23:0]。PC[0]总是清零。

指令字数：0.5
指令周期数：3

注：

1. 如果有异常等待处理，则不会执行取出的返回 PC 指令，从而使有效指令执行时间（对应于延时）为 1 个周期。

RLC	带进位位循环左移 Ws 内容

语法：

{标号:}	RLC.b	Ws,	Wd
	RLC.bz	[Ws],	[Wd]
	RLC{.w}	[Ws++],	[Wd++]
	RLC.l	[Ws--],	[Wd--]
		[++Ws],	[++Wd]
		[--Ws],	[--Wd]

操作数：Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]
操作：
对于字节操作：
(C) → Wd[0]，(Ws[6:0]) → Wd[7:1]，(Ws<7>) → C
对于字操作：
(C) → Wd[0]，(Ws[14:0]) → Wd[15:1]，(Ws[15]) → C
对于长字操作：
(C) → Wd[0]，(Ws[30:0]) → Wd[31:1]，(Ws[31]) → C

.....（续）

RLC		带进位位循环左移 Ws 内容						
受影响的状态位：	C、N 和 Z							
指令编码：	S011	000L	dddd	ssss	pppq	qqUU	UUUU	BU00

说明：

带进位标志位将源寄存器 Ws 中的内容循环左移一位，并将结果存放在目标寄存器 Wd 中。
在所有情况下，根据使用操作的数据大小对结果的评估来设置 N 和 Z。

S 位用于选择指令大小。

L 和 B 位用于选择操作数据宽度。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

指令字数： 1 或 0.5

指令周期数： 1

RLC		带进位位循环左移 f 内容			
语法：	{标号:}	RLC.b	f	{,Wnd}	{,WREG}
		RLC.bz			
		RLC{.w}			
		RLC.l			

操作数： f ∈ [0 ...64 KB]；Wnd ∈ [W0 ...W14]

操作：

对于字节操作：
(C) → Dest[0]，(f[6:0]) → Dest[7:1]，(f<7>) → C

对于字操作：
(C) → Dest[0]，(f[14:0]) → Dest[15:1]，(f[15]) → C

对于长字操作：
(C) → Dest[0]，(f[30:0]) → Dest[31:1]，(f[31]) → C

受影响的状态位：	C、N 和 Z							
指令编码：	1011	000L	dddd	ffff	ffff	ffff	ffff	BD01

说明：

带进位标志位将文件寄存器 f 中的内容循环左移一位，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wnd，则 D = 0 并将结果存放在 Wnd 中；否则，D = 1 并将结果存放在文件寄存器中。
L 和 B 位用于选择操作数据宽度。

D 位用于选择目标寄存器。

f 位用于选择文件寄存器的地址。

d 位用于选择工作寄存器。

指令字数： 1

指令周期数： 1

RLNC		循环左移 Ws 内容（不带进位）		
语法：	{标号:}	RLNC.b	Ws,	Wd
		RLNC.bz	[Ws],	[Wd]
		RLNC{.w}	[Ws++],	[Wd++]
		RLNC.l	[Ws--],	[Wd--]
			[++Ws],	[++Wd]
			[--Ws],	[--Wd]

操作数： Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]

.....（续）

RLNC	循环左移 Ws 内容（不带进位）

操作：

对于字节操作：
(Ws[6:0]) → Wd[7:1]，(Ws<7>) → Wd[0]

对于字操作：

(Ws[14:0]) → Wd[15:1]，(Ws[15]) → Wd[0]

对于长字操作：

(Ws[30:0]) → Wd[31:1]，(Ws[31]) → Wd[0]

受影响的状态位：N 和 Z

指令编码：

S011	001L	dddd	ssss	pppq	qqUU	UUUU	BU00

说明：将源寄存器 Ws 中的内容循环左移一位，并将结果存放在目标寄存器 Wd 中。进位标志位不受影响。
S 位用于选择指令大小。

L 和 B 位用于选择操作数据宽度。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

指令字数：1 或 0.5

指令周期数：1

RLNC	循环左移 f 内容（不带进位）

语法：

{标号:}	RLNC.b	f	{,Wnd}	{,WREG}
	RLNC.bz			
	RLNC{.w}			
	RLNC.l			

操作数：f ∈ [0 ...64 KB]；Wnd ∈ [W0 ...W14]

操作：

对于字节操作：
(f[6:0]) → Dest[7:1]，(f<7>) → Dest[0]

对于字操作：

(f[14:0]) → Dest[15:1]，(f[15]) → Dest[0]

对于长字操作：

(f[30:0]) → Dest[31:1]，(f[31]) → Dest[0]

受影响的状态位：N 和 Z

指令编码：

1011	001L	dddd	ffff	ffff	ffff	ffff	BD01

说明：将文件寄存器 f 中的内容循环左移一位，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wnd，则 D = 0 并将结果存放在 Wnd 中；否则，D = 1 并将结果存放在文件寄存器中。进位标志位不受影响。
L 和 B 位用于选择操作数据宽度。

D 位用于选择目标寄存器。

f 位用于选择文件寄存器的地址。

d 位用于选择工作寄存器。

指令字数：1

指令周期数：1

RRC	带进位位循环右移 Ws 内容

语法：

{标号:}	RRC.b	Ws,	Wd
	RRC.bz	[Ws],	[Wd]
	RRC{.w}	[Ws++],	[Wd++]
	RRC.l	[Ws--],	[Wd--]

.....（续）

RRC		带进位位循环右移 Ws 内容							
			[++Ws],	[++Wd]					
			[--Ws],	[--Wd]					
操作数：	Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]								
操作：	对于字节操作： (C) → Wd<7>，(Ws[7:1]) → Wd[6:0]，(Ws[0]) → C 对于字操作： (C) → Wd[15]，(Ws[15:1]) → Wd[14:0]，(Ws[0]) → C 对于长字操作： (C) → Wd[31]，(Ws[31:1]) → Wd[30:0]，(Ws[0]) → C								
受影响的状态位：	C、N 和 Z								
指令编码：	S011	010L	dddd	ssss	pppq	qqUU	UUUU	BU00	
说明：	带进位标志位将源寄存器 Ws 中的内容循环右移一位，并将结果存放在目标寄存器 Wd 中。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。								
指令字数：	1 或 0.5								
指令周期数：	1								

RRC		带进位位循环右移 f 内容			
语法：	{标号:}	RRC.b	f	{,Wnd}	{,WREG}
		RRC.bz			
		RRC{.w}			
		RRC.l			
操作数：	f ∈ [0 ...64 KB]；Wnd ∈ [W0 ...W14]				
操作：	对于字节操作： (C) → Dest[7]，(f[7:1]) → Dest[6:0]，(f[0]) → C 对于字操作： (C) → Dest[15]，(f[15:1]) → Dest[14:0]，(f[0]) → C 对于长字操作： (C) → Dest[31]，(f[31:1]) → Dest[30:0]，(f[0]) → C				
受影响的状态位：	C、N 和 Z				
指令编码：	1011	010L	dddd	ffff	ffff	ffff	ffff	BD01
说明：	带进位标志位将文件寄存器 f 中的内容循环左移一位，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wnd，则 D = 0 并将结果存放在 Wnd 中；否则，D = 1 并将结果存放在文件寄存器中。 L 和 B 位用于选择操作数据宽度。 D 位用于选择目标寄存器。 f 位用于选择文件寄存器的地址。 d 位用于选择工作寄存器。				
指令字数：	1				
指令周期数：	1				

RRNC		循环右移 Ws 内容（不带进位）		
语法：	{标号:}	RRNC.b	Ws,	Wd

.....（续）

RRNC	循环右移 Ws 内容（不带进位）		
	RRNC.bz	[Ws],	[Wd]
	RRNC{.w}	[Ws++],	[Wd++]
	RRNC.l	[Ws--],	[Wd--]
		[++Ws],	[++Wd]
		[--Ws],	[--Wd]

操作数：Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]

操作：

对于字节操作：

(Ws[7:1]) → Wd[6:0]，(Ws[0]) → Wd<7>

对于字操作：

(Ws[15:1]) → Wd[14:0]，(Ws[0]) → Wd[15]

对于长字操作：

(Ws[31:1]) → Wd[30:0]，(Ws[0]) → Wd[31]

受影响的状态位：N 和 Z

指令编码：

S011	011L	dddd	ssss	pppq	qqUU	UUUU	BU00

说明：将源寄存器 Ws 中的内容循环右移一位，并将结果存放在目标寄存器 Wd 中。进位标志位不受影响。

S 位用于选择指令大小。

L 和 B 位用于选择操作数据宽度。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

指令字数：1 或 0.5

指令周期数：1

RRNC	循环右移 f 内容（不带进位）				
语法：	{标号:}	RRNC.b	f	{,Wnd}	{,WREG}
		RRNC.bz			
		RRNC{.w}			
		RRNC.l			

操作数：f ∈ [0 ...64 KB]；Wnd ∈ [W0 ...W14]

操作：

对于字节操作：

(f[7:1]) → Dest[6:0]，(f[0]) → Dest[7]

对于字操作：

(f[15:1]) → Dest[14:0]，(f[0]) → Dest[15]

对于长字操作：

(f[31:1]) → Dest[30:0]，(f[0]) → Dest[31]

受影响的状态位：N 和 Z

指令编码：

1011	011L	dddd	ffff	ffff	ffff	ffff	BD01

说明：将文件寄存器 f 中的内容循环右移一位，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wnd，则 D = 0 并将结果存放在 Wnd 中；否则，D = 1 并将结果存放在文件寄存器中。进位标志位不受影响。

L 和 B 位用于选择操作数据宽度。

D 位用于选择目标寄存器。

f 位用于选择文件寄存器的地址。

s 位用于选择工作寄存器。

指令字数：1

.....（续）

RRNC		循环右移 f 内容（不带进位）			
指令周期数：	1				

SAC		保存累加器内容			
语法：	{标号:}	SAC{.w}	A,	{ Slit6, }	Wd
		SAC.l	B,		[Wd]
					[Wd++]
					[Wd--]
					[--Wd]
					[++Wd]
					[Wd+Wb]

操作数： Slit6 ∈ [-32 ... +31]；Wd ∈ [W0 ...W15]（见注 4）；Wb ∈ [W0 ...W15]

操作：

对于长字操作：

$Shift_{Slit6}$(ACC)

(ACC[63:32]) → Wd[31:0]

对于字操作：

$Shift_{Slit6}$(ACC)

(ACC[63:48]) → Wd[15:0]（见注 3）

受影响的状态位： 无

指令编码：

1100	01AL	wwww	dddd	UUUq	qqkk	kkkk	0011

说明：

读取累加器值并可选择将其移位，然后将截取的结果存放在目标有效地址。任何移位均仅应用于从累加器读取的数据，因此不会修改累加器的内容。移位值源自有符号立即数值。

移位后，字操作（SAC{.w}）假定该值为 Q1.15 有符号小数。因此，移位后的结果[63:48]随后写入有效地址。

移位后，长字操作（SAC.l）假定该值为 Q1.31 有符号小数。因此，移位后的结果[63:32]随后写入有效地址。

L 位用于选择字或长字操作。

A 位用于指定源累加器。

d 位用于指定目标寄存器 Wd。

q 位用于选择目标寻址模式。

w 位用于指定偏移量寄存器 Wb。

k 位用于编码可选的操作数 Slit6，从而决定累加器内容移位的位数。如果操作数 Slit6 不存在，则将立即数设置为全 0。

注：

1. 操作数 Slit6 为正值表示算术右移。操作数 Slit6 为负值表示左移。
2. 当目标为寄存器直接寻址（W 寄存器）时，结果会经过零扩展（CORCON.US = 1）或符号扩展（CORCON.US = 0）至 32 位。
3. 不允许使用寄存器直接寻址目标 W15。

指令字数： 1

指令周期数： 1

SE		符号扩展 Ws 内容		
语法：	{标号:}	SE.b	Ws,	Wnd
		SE.w	[Ws],	
			[Ws++],	
			[Ws--],	
			[++Ws],	
			[--Ws],	
			[Ws+Wb],	

.....（续）

SE	符号扩展 Ws 内容
操作数：	Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]；Wnd ∈ [W0 ...W14]
操作：	如果是字节模式： {24{Ws[7]}}和 Ws[7:0] → Wnd[31:0] 如果是字模式： {16{Ws[15]}}和 Ws[15:0] → Wnd[31:0]
受影响的状态位：	C、N 和 Z
指令编码：	S111 110B dddd ssss pppU UUww wwUU UU00
说明：	将 Ws 中的 8 位或 16 位有符号值符号扩展为 32 位有符号值，然后将结果回写到 Wnd。 C 设置为 N 的反码。 S 位用于选择指令大小。 B 位用于选择字节或字操作。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 w 位用于定义偏移量 Wb。 **注：** 1. 该操作总是写入一个长字。 2. 与所有其他字模式指令不同，字数据并不总是进行零扩展。
指令字数：	1 或 0.5
指令周期数：	1

SETM	将 f 置 1		
语法：	{标号:}	SETM.b	f
		SETM{.w}	
		SETM.l	
操作数：	f ∈ [0 ...1 MB]		
操作：	0xFFFFFFFF → 文件寄存器（长字操作） 0xFFFF → 文件寄存器（字操作） 0xFF → 文件寄存器（字节操作）		
受影响的状态位：	无		
指令编码：	1010 111L ffff ffff ffff ffff ffff B101		
说明：	将文件寄存器置 1。 L 和 B 位用于选择操作数据宽度。 f 位用于选择文件寄存器的地址。 **注：** 1. 可访问的文件地址空间为 1 MB。		
指令字数：	1		
指令周期数：	1		

SFTAC	将累加器内容算术移位		
语法：	{标号:}	SFTAC{.w} A,	Ws
		SFTAC.l B,	[Ws]
			[Ws++]
			[Ws--]
			[++Ws]
			[--Ws]

.....（续）

SFTAC	将累加器内容算术移位							
				[Ws+Wb]				
操作数：	Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]							
操作：	$Shift_{(Ws)}$(ACC)							
受影响的状态位：	OA 和 SA，或者 OB 和 SB							
指令编码：	1100	11AL	wwww	ssss	pppU	UUUU	UUUU	0011
说明：	将累加器内容算术移位。 移位值源自 Ws[5:0]的字或长字有符号内容。如果 Ws 为正数或零，则为右移，移位位数介于 1 与 31 之间（如果 Ws 为 0，则不移位）。如果 Ws 为负数，则为左移，移位位数介于 1 与 32 之间。 如果 Ws 超出范围（即，当 Ws[15:5]（对于字）或 Ws[31:5]（对于长字）不等于全 1 或全 0 时），则在执行过程中将请求数学错误陷阱。该指令将继续执行，但结果不会写入目标累加器，并且 SR 不会发生修改。 L 位用于选择字或长字 Ws（移位计数）。 A 位用于选择存放结果的累加器。 s 位用于选择 Ws 寄存器。 p 位用于选择源寻址模式。 w 位用于定义偏移量 Wb。							
指令字数：	1							
指令周期数：	1							

SFTAC	将累加器内容算术移位			
语法：	{标号:}	SFTAC	A, B,	Slit6
操作数：	Slit6 ∈ [-32 ...31]			
操作：	$Shift_k$(ACC)			
受影响的状态位：	OA 和 SA，或者 OB 和 SB			
指令编码：	1100	11AU	UUUU	UUUU	UUUU	UUkk	kkkk	0111
说明：	将累加器内容算术移位。 Slit6 用作移位位数。如果 Slit6 为正数或零，则为右移，移位位数介于 0 与 31 之间。如果 Slit6 为负数，则为左移，移位位数介于 1 与 32 之间。 A 位用于选择存放结果的累加器。 k 位用于确定移位的位数。			
指令字数：	1			
指令周期数：	1			

SL	左移 1 位			
语法：	{标号:}	SL.b SL.bz SL{.w} SL.l	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	Wd [Wd] [Wd++] [Wd--] [++Wd] [--Wd]
操作数：	Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]			

.....（续）

SL	左移 1 位							
操作：	对于长字操作： (Ws[31]) → C，(Ws[30:0]) → Wd[31:1]，0 → Wd[0] 对于字操作： (Ws[15]) → C，(Ws[14:0]) → Wd[15:1]，0 → Wd[0] 对于字节操作： (Ws[7]) → C，(Ws[6:0]) → Wd[7:1]，0 → Wd[0]							
受影响的状态位：	C、N 和 Z							
指令编码：	S010	110L	dddd	ssss	pppq	qqUU	UUUU	B000
说明：	将源寄存器 Ws 中的内容左移一位，并将结果存放在目标寄存器 Wd 中。 目标寄存器直接寻址扩展字节或字模式会将结果零扩展至 32 位，然后写入 Wd。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。							
指令字数：	1 或 0.5							
指令周期数：	1							

SL		左移 f 内容			
语法：	{标号:}	SL.b SL.bz SL{.w} SL.l	f	{,Wnd}	{,WREG}
操作数：	f ∈ [0 ...64 KB]；Wnd ∈ [W0 ...W14]				
操作：	对于字节操作： (f<7>) → (C)，(f[6:0]) → Dest[7:1]，0 → Dest[0] 对于字操作： (f[15]) → (C)，(f[14:0]) → Dest[15:1]，0 → Dest[0] 对于长字操作： (f[31]) → (C)，(f[30:0]) → Dest[31:1]，0 → Dest[0]				
受影响的状态位：	C、N 和 Z				
指令编码：	1010	010L	dddd	ffff	ffff	ffff	ffff	BD01
说明：	将文件寄存器 f 的内容左移一位并用 0 填充。如果 f 的 MSb 为 1，则进位标志置 1。将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wnd，则 D = 0 并将结果存放在 Wnd 中；否则，D = 1 并将结果存放在文件寄存器中。 L 和 B 位用于选择操作数据宽度。 D 位用于选择目标寄存器。 f 位用于选择文件寄存器的地址。 d 位用于选择工作寄存器。				
指令字数：	1				
指令周期数：	1				

SL		左移（移位位数由短立即数确定）			
语法：	{标号:}	SL{.w} SL.l	Ws, [Ws], [Ws++],	lit5,	Wd [Wd] [Wd++]

.....（续）

SL	左移（移位位数由短立即数确定）			
		[Ws--],	[Wd--]
		[++Ws],	[--Wd]
		[--Ws],	[++Wd]

操作数：Ws ∈ [W0 ...W14]；lit5 ∈ [0...31]；Wd ∈ [W0 ...W14]

操作：
lit5[4:0]→ Shift_Val
1'b0 → 左移输入

对于长字操作：

32'b0 和 Ws[31:0] → Shift_In[63:0]

将 Shift_In[63:0]左移 Shift_Val 位 → Shift_Out[63:0]

Shift_Out[31:0] → Wnd

对于字操作：

48'b0 和 Ws[15:0] → Shift_In[63:0]

将 Shift_In[63:0]左移 Shift_Val 位 → Shift_Out[63:0]

Shift_Out[15:0] → Wnd[15:0]

受影响的状态位：N 和 Z

指令编码：

```
S010    010k    kkkk    dddd    pppq    qqss    ssUU    LU00
```

说明：
将源寄存器 Ws 中的内容左移 lit5 位（最大移位位数为 31 位），并将结果存放在目标寄存器 Wd 中。无论 lit5 中的移位值为何，该指令都会生成正确的结果（字操作移位值 > 15，Wnd[15:0] = 0x0000）。

当该指令与 SLMK 指令一起使用进行多精度多位移位操作时，如果移位值大于 32，将不会生成正确的结果。

寄存器直接寻址字模式会将结果零扩展至 32 位，然后写入 Wd。

S 位用于选择指令大小。

L 位用于选择字或长字操作。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

k 位用于提供立即数操作数。

注：

1. 该指令只能工作于字或长字模式。

指令字数：1 或 0.5

指令周期数：1

SL	左移 Wb 位				
语法：	{标号:}	SL.b	Ws,	Wb,	Wd
		SL.bz	[Ws],		[Wd]
		SL{.w}	[Ws++],		[Wd++]
		SL.l	[Ws--],		[Wd--]
			[++Ws],		[++Wd]
			[--Ws],		[--Wd]

操作数：Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]

.....（续）

SL	左移 Wb 位
操作：	Wb[15:0]→ Shift_Val 对于长字操作： 32'b0 和 Ws[31:0] → Shift_In[63:0] 将 Shift_In[63:0]左移 Shift_Val 位 → Shift_Out[63:0] Shift_Out[31:0] → Wd 对于字操作： 48'b0 和 Ws[15:0] → Shift_In[63:0] 将 Shift_In[63:0]左移 Shift_Val 位 → Shift_Out[63:0] Shift_Out[15:0] → Wd 对于字节操作： 56'b0 和 Ws[7:0] → Shift_In[63:0] 将 Shift_In[63:0]左移 Shift_Val 位 → Shift_Out[63:0] Shift_Out[7:0] → Wd
受影响的状态位：	N 和 Z
指令编码：	1010 110L wwww dddd pppq qqss ssUU B100
说明：	将源寄存器 Ws 中的内容左移 Wb 位，并将结果存放在目标寄存器 Wd 中。 单独使用该指令时，无论 Wb[15:0]中的移位值为何，都会生成正确的结果。 • 对于字节操作，移位值 > 7，Wd[7:0] = 0x00 • 对于字操作，移位值 > 15，Wd[15:0] = 0x0000 • 对于长字操作，移位值 > 31，Wd = 0x00000000 当该指令与 SLMW 指令一起使用进行多精度多位移位操作时，如果移位值大于 32，将不会生成正确的结果。 Wb[31:16]中保存的任何数据都无效。 目标寄存器直接寻址扩展字节或字模式会将结果零扩展至 32 位，然后写入 Wd。 L 和 B 位用于选择操作数据宽度。 s 位用于选择源寄存器。 w 位用于选择基准（移位计数）寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。
指令字数：	1
指令周期数：	1

SLAC	保存累加器内容的低位						
语法：	{标号:}	SLAC{.l}	A,	{ Slit6 , }	Wd		
			B,		[Wd]		
					[Wd++]		
					[Wd--]		
					[--Wd]		
					[++Wd]		
					[Wd+Wb]		
操作数：	Slit6 ∈ [-32 ...+31]；Wd ∈ [W0 ...W15]（见注 2）；Wb ∈ [W0 ...W15]						
操作：	$Shift_{Slit6}$(ACC) (ACC[31:0]) → Wd[31:0]						
受影响的状态位：	无						
指令编码：	1100	01A1	wwww	dddd	UUUq	qqkk	kkkk	0111

.....（续）

SLAC					保存累加器内容的低位
说明：	读取累加器值并可选择将其移位，然后将移位后的 ACC[31:0]存放在目标有效地址。任何移位均仅应用于从累加器读取的数据，因此不会修改累加器的内容。移位值源自有符号立即数值。 A 位用于指定源累加器。 d 位用于指定目标寄存器 Wd。 q 位用于选择目标寻址模式。 w 位用于指定偏移量寄存器 Wb。 k 位用于编码可选的操作数 Slit6，从而决定累加器内容移位的位数。如果操作数 Slit6 不存在，则将立即数设置为全 0。 **注：** 1. 操作数 Slit6 为正值表示算术右移。操作数 Slit6 为负值表示左移。 2. 不允许使用寄存器直接寻址目标 W15。				
指令字数：	1				
指令周期数：	1				

SLM					多精度左移（移位位数由短立即数确定）
语法：	{标号:}	SLM{.l}	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	lit5,	Wnd
操作数：	Ws ∈ [W0 ...W15]；lit5 ∈ [0...31]；Wnd ∈ [W0 ...W13]				
操作：	lit5[4:0]→ Shift_Val 0 → 左移输入 32'b0 和 Ws[31:0] → Shift_In[63:0] 将 Shift_In[63:0]左移 Shift_Val 位 → Shift_Out[63:0] Shift_Out[31:0] → Wnd Shift_Out[63:32] | Wnd+1 → Wnd+1				
受影响的状态位：	Z				
指令编码：	`1110 000k kkkk dddd pppU UUss ssUU 1011`				
说明：	将源寄存器 Ws 中的内容左移 lit5 位（最大移位位数为 31 位），并将结果存放在目标寄存器 Wnd 中。包含下一个最高有效数据字的寄存器将已包含中间移位结果。将该值与从 Ws 移出的数据进行按位或运算以生成最终的移位结果，然后更新相应的目标寄存器。 该指令专门与 SLK 指令一起使用，以支持多精度多位移位操作。 Z 位为“粘住”位（只能清零）。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 k 位用于提供立即数操作数。 **注：**该指令只能工作于长字模式。				
指令字数：	1				
指令周期数：	2				

SLM					多精度左移 Wb 位
语法：	{标号:}	SLM{.l}	Ws, [Ws], [Ws++],	Wb,	Wnd

.....（续）

SLM	多精度左移 Wb 位

[Ws--],
[++Ws],
[--Ws],

操作数：Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]；Wnd ∈ [W0 ...W13]

操作：
Wb[15:0]→ Shift_Val
0 → 左移输入
32'b0 和 Ws[31:0] → Shift_In[63:0]
将 Shift_In[63:0]左移 Shift_Val 位 → Shift_Out[63:0]
Shift_Out[31:0] → Wnd
Shift_Out[63:32] | Wnd+1 → Wnd+1

受影响的状态位：Z

指令编码：

```
1110      001U      ssss      dddd      pppU      UUww      wwUU      1011
```

说明：
将源寄存器 Ws 中的内容左移 Wb 位，并将结果存放在目标寄存器 Wnd 中。包含下一个最高有效数据字的寄存器将已包含中间移位结果。将该值与从 Ws 移出的数据进行按位或运算以生成最终的移位结果，然后更新相应的目标寄存器。
左移位数可以是 0 到 32 之间的任意值。如果 Wb[15:0]中保存的移位值超过 2'd32，则移位值将饱和至 2'd32 以保持一致性。Wb[31:16]中保存的任何数据都无效。
该指令专门与 SLW 指令一起使用，以支持多精度多位移位操作。
Z 位为“粘住”位（只能清零）。
w 位用于选择基准（移位计数）寄存器。
s 位用于选择源寄存器。
d 位用于选择目标寄存器。
p 位用于选择源寻址模式。
注：该指令只能工作于长字模式。

指令字数：1

指令周期数：2

SQR	求平方，结果存入累加器

语法：{标号}				
	SQR{.w}	Wx,	A	{,AWB}
	SQR.l	[Wx],	B	
		[Wx]+=kx,		
		[Wx]-=kx,		
		[Wx+=kx],		
		[Wx-=kx],		
		[Wx+W12],		

操作数：
Wx ∈ {W0 ...W15}；
字模式：kx ∈ {-8, -6, -4, -2, 2, 4, 6, 8}；
长字模式：kx ∈ {-16, -12, -8, -4, 4, 8, 12, 16}；
AWB ∈ {W0, W1, W2, W3, W13, [W13++], [W15++][5]}

操作：
$(Wx)^2$ → ACC(A 或 B)；
舍入后的(ACC(B 或 A)) → AWB
当使用执行前修改/执行后修改的间接寻址时：
(Wx)+kx→Wx 或(Wx)-kx→Wx；

受影响的状态位：OA 和 SA，*或者* OB 和 SB

指令编码：

```
1101      01AL      wwww      ssss      IIIi      i110      UUaa      a011
```

.....（续）

SQR	求平方，结果存入累加器
说明：	用于计算(A)^2 函数的指令。对从 Wx 读取的数据或从 X 地址空间获取的数据进行有符号或无符号（由 CORCON.US 定义）求平方。请注意，由于只需要一个操作数，DS 不会拆分为 X 和 Y 地址空间以进行并发读访问，因此该指令中的 X 空间代表所有可用的 DS（其中将包括 Y 空间）。 结果经过符号扩展或零扩展至 72 位，然后写入指定的累加器。此外，小数结果也会在累加器更新之前进行换算，以对齐操作数和累加器（msw）小数点。 当选择执行前修改/执行后修改的间接寻址时，地址修改量的值为 kx，表示有效地址中要修改的数据字节数。 可选的 AWB 用于指定是直接还是间接存储非 SQR 操作目标的累加器舍入后的小数内容（32 位）。舍入模式由 CORCON.RND 定义。写入数据宽度由所选指令数据大小决定（见注 4）。当 DSP 引擎工作于整数模式时，不适合使用 AWB。 读取的数据可以是 16 位或 32 位值。所有间接地址修改都会相应地进行换算。 L 位用于选择字或长字操作。 A 位用于选择存放结果的累加器。 I 位用于选择操作寻址模式。 i 位用于选择 kx 修改值。 s 位用于选择 Wx、数据寄存器或 X 空间源地址寄存器。 w 位将是 s 位的副本。 a 位用于选择累加器回写目标和寻址模式。 **注：** 1. 在小数还是整数数据模式下工作由 CORCON.IF 定义。 2. 将操作数视为有符号还是无符号总是根据 CORCON.US 的状态来判断。 3. 如果源是执行前修改或执行后修改的有效地址，则不允许对间接源和 AWB 间接目标使用相同的 W 寄存器。 4. 堆栈必须保持长字对齐。因此，只允许将[W15++] AWB 与长字 MAC 类指令一起使用。
指令字数：	1
指令周期数：	1

SQRAC	平方并累加		
语法：{标号}	SQRAC{.w} Wx,	A	{,AWB}
	SQRAC.l [Wx],	B	
	[Wx]+=kx,		
	[Wx]-=kx,		
	[Wx+=kx],		
	[Wx-=kx],		
	[Wx+W12],		
操作数：	Wx ∈ {W0 ...W15}； 字模式：kx ∈ {-8, -6, -4, -2, 2, 4, 6, 8}； 长字模式：kx ∈ {-16, -12, -8, -4, 4, 8, 12, 16}； AWB ∈ {W0, W1, W2, W3, W13, [W13++], [W15++][5]}		
操作：	ACC(A 或 B) + (Wx)^2 → ACC(A 或 B)； 舍入后的(ACC(B 或 A)) → AWB 当使用执行前修改/执行后修改的间接寻址时： (Wx)+kx→Wx 或(Wx)-kx→Wx；		
受影响的状态位：	OA 和 SA，*或者* OB 和 SB		
指令编码：	1101 00AL wwww ssss IIIi i110 UUaa a011		

.....（续）

SQRAC	平方并累加

说明：

用于计算累加器 + $(A)^2$ 函数的指令。对从 Wx 读取的数据或从 X 地址空间获取的数据进行有符号或无符号（由 CORCON.US 定义）求平方。请注意，由于只需要一个操作数，DS 不会拆分为 X 和 Y 地址空间以进行并发读访问，因此该指令中的 X 空间代表所有可用的 DS（其中将包括 Y 空间）。
结果经过符号扩展或零扩展至 72 位，然后加到指定的累加器。结果在累加器更新之前是否进行换算取决于是小数运算还是整数运算（由 CORCON.IF 定义）。小数运算将对结果进行换算以对齐操作数和累加器（msw）小数点（见注 3）。整数运算会将结果的 LSb 与累加器的 LSb 对齐。

当选择执行前修改/执行后修改的间接寻址时，地址修改量的值为 kx，表示有效地址中要修改的数据字节数。

可选的 AWB 用于指定是直接还是间接（见注 4）存储非 SQRAC 操作目标的累加器舍入后的小数内容（32 位）。舍入模式由 CORCON.RND 定义。写入数据宽度由所选指令数据大小决定。当 DSP 引擎工作于整数模式时，不适合使用 AWB。

读取的数据可以是 16 位或 32 位值。所有间接地址修改都会相应地进行换算。

L 位用于选择字或长字操作。

A 位用于选择存放结果的累加器。

I 位用于选择操作寻址模式。

i 位用于选择 kx 修改值。

s 位用于选择 Wx、数据寄存器或 X 空间源地址寄存器。

w 位将是 s 位的副本。

a 位用于选择累加器回写目标和寻址模式。

注：

1. 在小数还是整数数据模式下工作由 CORCON.IF 定义。
2. 将操作数视为有符号还是无符号总是根据 CORCON.US 的状态来判断。
3. 当在小数模式下使用字大小的数据时，ACCx 的低位部分不受影响。因此，之前的（32 位数据）操作中可能存在的低位数据会得以保留。如果不需要该数据，用户应在初始化阶段清零 ACCx。
4. 如果源是执行前修改或执行后修改的有效地址，则不允许对间接源和 AWB 间接目标使用相同的 W 寄存器。
5. 堆栈必须保持长字对齐。因此，只允许将[W15++] AWB 与长字 MAC 类指令一起使用。

指令字数：1

指令周期数：1

SQRN	平方并取反，结果存入累加器

语法：{标号}

```
SQRN{.w} Wx,          A       {,AWB}
SQRN.l   [Wx],        B
         [Wx]+=kx,
         [Wx]-=kx,
         [Wx+=kx],
         [Wx-=kx],
         [Wx+W12],
```

操作数：
Wx ∈ {W0 ...W15}；
字模式：kx ∈ {-8, -6, -4, -2, 2, 4, 6, 8}；
长字模式：kx ∈ {-16, -12, -8, -4, 4, 8, 12, 16}；
AWB ∈ {W0, W1, W2, W3, W13, [W13++], [W15++][5]}

操作：
$-(Wx)^2$ → ACC(A 或 B)；
舍入后的(ACC(B 或 A)) → AWB

当使用执行前修改/执行后修改的间接寻址时：

(Wx)+kx→Wx 或(Wx)-kx→Wx；

受影响的状态位：OA 和 SA，*或者* OB 和 SB

指令编码：

```
1101      01AL      wwww      ssss      IIIi      i110      UUaa      a111
```

.....（续）

SQRN	平方并取反，结果存入累加器

说明：

用于计算-(A)^2 函数的指令。对从 Wx 读取的数据或从 X 地址空间获取的数据进行有符号或无符号（由 CORCON.US 定义）求平方。请注意，由于只需要一个操作数，DS 不会拆分为 X 和 Y 地址空间以进行并发读访问，因此该指令中的 X 空间代表所有可用的 DS（其中将包括 Y 空间）。
结果经过符号扩展或零扩展至 72 位并取反，然后加到指定的累加器。此外，小数结果也会在累加器更新之前进行换算，以对齐操作数和累加器（msw）小数点。对于字大小的操作数操作，当选择执行前修改/执行后修改的间接寻址时，地址修改量的值为 kx，表示有效地址中要修改的数据字节数。

可选的 AWB 用于指定是直接还是间接（见注 3）存储非 SQRN 操作目标的累加器舍入后的小数内容（32 位）。舍入模式由 CORCON.RND 定义。写入数据宽度由所选指令数据大小决定（见注 4）。当 DSP 引擎工作于整数模式时，不适合使用 AWB。

读取的数据可以是 16 位或 32 位值。所有间接地址修改都会相应地进行换算。

L 位用于选择字或长字操作。

A 位用于选择存放结果的累加器。

I 位用于选择操作寻址模式。

i 位用于选择 kx 修改值。

s 位用于选择 Wx、数据寄存器或 X 空间源地址寄存器。

w 位将是 s 位的副本。

a 位用于选择累加器回写目标和寻址模式。

注：

1. 在小数还是整数数据模式下工作由 CORCON.IF 定义。
2. 如果源是执行前修改或执行后修改的有效地址，则不允许对间接源和 AWB 间接目标使用相同的 W 寄存器。
3. 如果源是执行前修改或执行后修改的有效地址，则不允许对间接源和 AWB 间接目标使用相同的 W 寄存器。
4. 堆栈必须保持长字对齐。因此，只允许将[W15++] AWB 与长字 MAC 类指令一起使用。

指令字数：1

指令周期数：1

SQRSC	求平方并从累加器中减去

语法：{标号}

```
SQRSC{.w} Wx,          A      {,AWB}
SQRSC.l [Wx],          B
          [Wx]+=kx,
          [Wx]-=kx,
          [Wx+=kx],
          [Wx-=kx],
          [Wx+W12],
```

操作数：

Wx ∈ {W0 ...W15}；
字模式：kx ∈ {-8, -6, -4, -2, 2, 4, 6, 8}；
长字模式：kx ∈ {-16, -12, -8, -4, 4, 8, 12, 16}；
AWB ∈ {W0, W1, W2, W3, W13, [W13++], [W15++][5]}

操作：

ACC(A 或 B) - (Wx)^2 → ACC(A 或 B)；
舍入后的(ACC(B 或 A)) → AWB

当使用执行前修改/执行后修改的间接寻址时：

(Wx)+kx→Wx 或(Wx)-kx→Wx；

受影响的状态位：OA 和 SA，*或者* OB 和 SB

指令编码：

```
1101    00AL    wwww    ssss    IIIi    i110    UUaa    a111
```

.....（续）

SQRSC	求平方并从累加器中减去

说明：

用于计算累加器 - $(A)^2$ 函数的指令。对从 Wx 读取的数据或从 X 地址空间获取的数据进行有符号或无符号（由 CORCON.US 定义）求平方。请注意，由于只需要一个操作数，DS 不会拆分为 X 和 Y 地址空间以进行并发读访问，因此该指令中的 X 空间代表所有可用的 DS（其中将包括 Y 空间）。
结果经过符号扩展或零扩展至 72 位，然后从指定的累加器中减去。结果在累加器更新之前是否进行换算取决于是小数运算还是整数运算（由 CORCON.IF 定义）。小数运算将对结果进行换算以对齐操作数和累加器（msw）小数点（见注 4）。整数运算会将结果的 LSb 与累加器的 LSb 对齐。

当选择执行前修改/执行后修改的间接寻址时，地址修改量的值为 kx，表示有效地址中要修改的数据字节数。

可选的 AWB 用于指定是直接还是间接（见注 4）存储非 SQRSC 操作目标的累加器舍入后的小数内容（32 位）。舍入模式由 CORCON.RND 定义。写入数据宽度由所选指令数据大小决定。当 DSP 引擎工作于整数模式时，不适合使用 AWB。

读取的数据可以是 16 位或 32 位值。所有间接地址修改都会相应地进行换算。

L 位用于选择字或长字操作。

A 位用于选择存放结果的累加器。

I 位用于选择操作寻址模式。

i 位用于选择 kx 修改值。

s 位用于选择 Wx、数据寄存器或 X 空间源地址寄存器。

w 位将是 s 位的副本。

a 位用于选择累加器回写目标和寻址模式。

注：

1. 在小数还是整数数据模式下工作由 CORCON.IF 定义。
2. 将操作数视为有符号还是无符号总是根据 CORCON.US 的状态来判断。
3. 如果源是执行前修改或执行后修改的有效地址，则不允许对间接源和 AWB 间接目标使用相同的 W 寄存器。
4. 当在小数模式下使用字大小的数据时，ACCx 的低位部分不受影响。因此，之前的（32 位数据）操作中可能存在的低位数据会得以保留。如果不需要该数据，用户应在初始化阶段清零 ACCx。
5. 堆栈必须保持长字对齐。因此，只允许将[W15++] AWB 与长字 MAC 类指令一起使用。

指令字数：1

指令周期数：1

SACR	保存舍入后的累加器内容，按照立即数移位

语法：

{标号:}	SACR{.w} A,	{ Slit6, }	Wd
	SACR.l B,		[Wd]
			[Wd++]
			[Wd--]
			[--Wd]
			[++Wd]
			[Wd+Wb]

操作数：Slit6 ∈ [-32 ...+31]；Wd ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]；

操作：

对于长字操作：
长字模式： Post-shift(Round{ACC[63:32]}) → Wd[31:0]

对于字操作：

字模式： Post-shift(Round{ACC[63:48]}) → Wd[15:0] [1]

受影响的状态位：无

指令编码：

1100	01AL	wwww	dddd	UUUq	qqkk	kkkk	1011

.....（续）

SACR	保存舍入后的累加器内容，按照立即数移位

说明：

读取累加器的内容并可选择将其算术移位，然后将舍入后的结果存放在目标有效地址。使用的舍入模式由 CORCON.RND 位定义。该指令不修改目标累加器的内容。
移位值源自 6 位有符号立即数。如果该立即数为正数或零，则为右移，移位位数介于 0 与 31 之间。如果该立即数为负数，则为左移，移位位数介于 1 与 32 之间。如果未声明该立即数，则在操作码中为其赋值 0，不进行移位。

移位后，字操作（SACR.w）假定该值为 Q1.15 有符号小数。因此，对移位结果 bit 48 执行舍入，然后将移位结果[63:48]写入有效地址。当目标为 W 寄存器时，该值进行零扩展。

移位后，长字操作（SACR.l）假定该值为 Q1.31 有符号小数。因此，对移位结果 bit 32 执行舍入，然后将移位结果[63:32]写入有效地址。

L 位用于选择字或长字操作。

A 位用于指定源累加器。

d 位用于指定目标寄存器 Wd。

q 位用于选择目标寻址模式。

w 位用于指定偏移量寄存器 Wb。

k 位用于编码可选的操作数 Slit6，从而决定累加器内容移位的位数。如果操作数 Slit6 不存在，则将立即数设置为全 0。

注：

1. 当目标为寄存器直接寻址（W 寄存器）时，结果会经过零扩展（CORCON.US = 1）或符号扩展（CORCON.US = 0）至 32 位。
2. 不允许使用寄存器直接寻址目标 W15。

指令字数：1

指令周期数：1

SACR	移位后保存舍入后的累加器内容

语法：

{标号:}	SACR{.w} A,	Ws,	Wd
	SACR.l B,		[Wd]
			[Wd++]
			[Wd--]
			[--Wd]
			[++Wd]
			[Wd+Wb]

操作数：Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]（见注 2）；Wb ∈ [W0 ...W15]；

操作：

对于长字操作：
长字模式：Post-shift(Round{ACC[63:32]}) → Wd[31:0]

对于字操作：

字模式：Post-shift(Round{ACC[63:48]}) → Wd[15:0] [1]

受影响的状态位：无

指令编码：

1100	10AL	wwww	dddd	UUUq	qqss	ssUU	0111

.....（续）

SACR	移位后保存舍入后的累加器内容

说明：

读取累加器的内容并将其算术移位，然后将舍入后的结果存放在目标有效地址。使用的舍入模式由 CORCON.RND 位定义。该指令不修改目标累加器（或 SR）的内容。
移位值源自 Ws[5:0]的有符号内容。如果 Ws 为正数，则为右移，移位位数介于 1 与 31 之间（如果 Ws 为 0，则不移位）。如果 Ws 为负数，则为左移，移位位数介于 1 与 32 之间。

如果 Ws 超出范围（即，Ws[31:5]不等于全 1 或全 0 时），则在执行过程中将请求数学错误陷阱。该指令将基于 Ws[5:0]中保存的值继续执行。

移位后，字操作（SACR.w）假定该值为 Q1.15 有符号小数。因此，对移位结果 bit 48 执行舍入，然后将移位结果[63:48]写入有效地址。

移位后，长字操作（SACR.l）假定该值为 Q1.31 有符号小数。因此，对移位结果 bit 32 执行舍入，然后将移位结果[63:32]写入有效地址。

L 位用于选择字或长字操作。

A 位用于指定源累加器。

d 位用于指定目标寄存器 Wd。

s 位用于指定移位值源寄存器 Ws。

q 位用于选择目标寻址模式。

w 位用于指定偏移量寄存器 Wb。

关于修改量寻址信息，请参见 。

注：

1. 当目标为寄存器直接寻址（W 寄存器）时，结果会经过零扩展（CORCON.US = 1）或符号扩展（CORCON.US = 0）至 32 位。
2. 不允许使用寄存器直接寻址目标 W15。

指令字数：1

指令周期数：1

SUAC	保存累加器内容的高字节

语法：

{标号:}	SUAC{.l}	A,	{ Slit6 , }	Wd
		B,		[Wd]
				[Wd++]
				[Wd--]
				[--Wd]
				[++Wd]
				[Wd+Wb]

操作数：Slit6 ∈ [-32 ...+31]；Wd ∈ [W0 ...W15]（见注 2）；Wb ∈ [W0 ...W15]

操作：
$Shift_{Slit6}(ACC)$
(24{ACC[71]}和 ACC[71:64]) → Wd[31:0]

受影响的状态位：无

指令编码：

1100	01A1	wwww	dddd	UUUq	qqkk	kkkk	1111

.....（续）

SUAC	保存累加器内容的高字节

说明：读取累加器值并可选择将其移位，然后将移位后的高字节（ACC[71:64]）存放在目标有效地址。任何移位均仅应用于从累加器读取的数据，因此不会修改累加器的内容。移位值源自有符号立即数值。
当目标为寄存器直接寻址（W 寄存器）时，字节结果会经过零扩展（CORCON.US = 1）或符号扩展（CORCON.US = 0）至 32 位后再写入。

A 位用于指定源累加器。

d 位用于指定目标寄存器 Wd。

q 位用于选择目标寻址模式。

w 位用于指定偏移量寄存器 Wb。

k 位用于编码可选的操作数 Slit6，从而决定累加器内容移位的位数。如果操作数 Slit6 不存在，则将立即数设置为全 0。

注：

1. 操作数 Slit6 为正值表示算术右移。操作数 Slit6 为负值表示左移。
2. 不允许使用寄存器直接寻址目标 W15。

指令字数：1

指令周期数：1

SUB	Wb 减 Ws

语法：

{标号:}	SUB.b	Wb,	Ws,	Wd
	SUB.bz		[Ws],	[Wd]
	SUB{.w}		[Ws++],	[Wd++]
	SUB.l		[Ws--],	[Wd--]
			[++Ws],	[++Wd]
			[--Ws],	[--Wd]

操作数：Wb ∈ [W0 ...W15]；Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]

操作：(Wb) - (Ws) → Wd

受影响的状态位：C、N、OV 和 Z

指令编码：

S110	010L	dddd	ssss	pppq	qqww	wwUU	BU00

说明：从基准寄存器 Wb 内容中减去源寄存器 Ws 中的内容，并将结果存放在目标寄存器 Wd 中。
S 位用于选择指令大小。

L 和 B 位用于选择操作数据宽度。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

指令字数：1 或 0.5

指令周期数：1

SUB	累加器相减

语法：

{标号:}	SUB	A
		B

操作数：无

操作：如果为 SUBAB A，则 ACCA - ACCB → ACCA
如果为 SUBAB B，则 ACCB - ACCA → ACCB

受影响的状态位：OA 和 SA，或者 OB 和 SB

指令编码：

0111	001A	UUUU	1001

.....（续）

SUB	累加器相减
说明：	将两个累加器的内容相减，并将结果写入所选的累加器。 A 位用于指定目标累加器。
指令字数：	0.5
指令周期数：	1

SUB	从累加器中减去有符号数			
语法：	{标号:}	SUB{.w} Ws,	{ Slit6, }	A
		SUB.l [Ws],		B
		[Ws++]		
		[Ws--]		
		[--Ws],		
		[++Ws],		
		[Ws+Wb],		
操作数：	Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]；Slit6 ∈ [-32 ...+31]			
操作：	(ACC) - $Shift_{Slit6}$(符号扩展(Ws)) → ACC			
受影响的状态位：	OA 和 SA，或者 OB 和 SB			
指令编码：	1100 00AL wwww ssss pppU UUkk kkkk 1111			
说明：	假定有效地址处包含的操作数为 Q1.15 小数数据（对于字数据操作）或 Q1.31 小数数据（对于长字数据操作）。读取操作数后自动进行符号扩展和零回填，以创建与累加器大小相同的值。 将从目标累加器中减去该值之前，可选择进行（算术）移位。 L 位用于选择字或长字操作。 A 位用于指定目标累加器。 s 位用于指定源寄存器 Ws。 p 位用于选择源寻址模式。 w 位用于指定偏移量寄存器 Wb。 k 位用于对可选操作数 Slit6 进行编码，该操作数确定累加器内容预移位的位数：如果操作数 Slit6 不存在，则立即数位域设置为全 0。 **注：** 1. 操作数 Slit6 为正值表示算术右移。 操作数 Slit6 为负值表示左移。 2. 该指令只能工作于长字或字模式。			
指令字数：	1			
指令周期数：	1			

SUBB	Wb 减 Ws（带借位）				
语法：	{标号:}	SUBB.b	Wb,	Ws,	Wd
		SUBB.bz		[Ws],	[Wd]
		SUBB{.w}		[Ws++],	[Wd++]
		SUBB.l		[Ws--],	[Wd--]
				[++Ws],	[++Wd]
				[--Ws],	[--Wd]
操作数：	Wb ∈ [W0 ...W15]；Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]				
操作：	(Wb) - (Ws) - (C) → Wd				
受影响的状态位：	C、N、OV 和 Z				
指令编码：	S110 011L dddd ssss pppq qqww wwUU BU00				

.....（续）

SUBB	Wb 减 Ws（带借位）

说明：从基准寄存器 Wb 内容中减去源寄存器 Ws 的内容和进位标志，并将结果存放在目标寄存器 Wd 中。
Z 位为“粘住”位（只能清零）。
S 位用于选择指令大小。
L 和 B 位用于选择操作数据宽度。
s 位用于选择源寄存器。
w 位用于选择基准寄存器。
d 位用于选择目标寄存器。
p 位用于选择源寻址模式。
q 位用于选择目标寻址模式。

指令字数：1 或 0.5

指令周期数：1

SUBB	Ws 减短立即数（带借位）

语法：

{标号:}	SUBB.b SUBB.bz SUBB{.w} SUBB.l	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	lit7,	Wd [Wd] [Wd++] [Wd--] [++Wd] [--Wd]

操作数：Ws ∈ [W0 ...W15]；lit7 ∈ [0 ...127]；Wd ∈ [W0 ...W15]

操作：(Ws) - lit7 - (C) → Wd
注：立即数经过零扩展至操作所选的数据大小

受影响的状态位：C、N、OV 和 Z

指令编码：

1110	011L	dddd	ssss	pppq	qqkk	kkkk	Bk10

说明：从源寄存器 Ws 的内容中减去零扩展无符号立即数操作数和进位位，并将结果存放在目标寄存器 Wd 中。
Z 位为“粘住”位（只能清零）。
L 和 B 位用于选择操作数据宽度。
k 位用于提供立即数操作数（op[2]中的 MSb）。
s 位用于选择源寄存器。
d 位用于选择目标寄存器。
p 位用于选择源寻址模式。
q 位用于选择目标寻址模式。

指令字数：1

指令周期数：1

SUBB	Wn 减立即数（带借位）

语法：

{标号:}	SUBB.b SUBB.bz SUBB{.w} SUBB.l	lit16,	Wn

操作数：lit16 ∈ [0 ...65535]；Wn ∈ [W0 ...W15]

操作：(Wn) - lit16 - (C) → Wn
注：立即数经过零扩展至操作所选的数据大小

受影响的状态位：C、N、OV 和 Z

指令编码：

1100	011L	ssss	kkkk	kkkk	kkkk	kkkk	B010

.....（续）

SUBB	Wn 减立即数（带借位）

说明：从工作寄存器 Wn 的内容中减去零扩展立即数操作数和进位位，并将结果存放在工作寄存器 Wn 中。在字节模式下，只写入 Wn 的 LSb。在字模式下，字结果经过零扩展至 32 位并写入 Wn。

Z 位为“粘住”位（只能清零）。

L 和 B 位用于选择操作数据宽度。

s 位用于选择工作寄存器。

k 位用于指定立即数操作数。

指令字数：1

指令周期数：1

SUBBR	Wn 减 f（带进位）

语法：{标号:} SUBBR.bf ,Wn {,WREG}
SUBBR.bz
SUBBR{.w}
SUBBR.l

操作数：f ∈ [0 ...64 KB]；Wn ∈ [W0 ...W15]

操作：(Wn) - (f) - (C) → 由 D 指定的目标寄存器

受影响的状态位：C、N、OV 和 Z

指令编码：

1110	011L	ssss	ffff	ffff	ffff	ffff	BD01

说明：从工作寄存器内容中减去文件寄存器的内容与进位位，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wn，则 D = 0 并将结果存放在 Wn 中；否则，D = 1 并将结果存放在文件寄存器中。Z 位为“粘住”位（只能清零）。

L 和 B 位用于选择操作数据宽度。

D 位用于选择目标寄存器。

f 位用于选择文件寄存器的地址。

s 位用于选择工作寄存器。

指令字数：1

指令周期数：1

SUBBR	Ws 减 Wb（带借位）

语法：{标号:} SUBBR.b Wb, Ws, Wd
SUBBR.bz [Ws], [Wd]
SUBBR{.w} [Ws++], [Wd++]
SUBBR.l [Ws--], [Wd--]
[++Ws], [++Wd]
[--Ws], [--Wd]

操作数：Wb ∈ [W0 ...W15]；Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]

操作：(Ws) - (Wb) - (C) → Wd

受影响的状态位：C、N、OV 和 Z

指令编码：

S110	111L	dddd	ssss	pppq	qqww	wwUU	BU00

.....（续）

SUBBR	Ws 减 Wb（带借位）

说明：从源寄存器 Ws 内容中减去基准寄存器 Wb 的内容和进位位，并将结果存放在目标寄存器 Wd 中。
Z 位为“粘住”位（只能清零）。
S 位用于选择指令大小。
L 和 B 位用于选择操作数据宽度。
s 位用于选择源寄存器。
w 位用于选择基准寄存器。
d 位用于选择目标寄存器。
p 位用于选择源寻址模式。
q 位用于选择目标寻址模式。

指令字数：1 或 0.5

指令周期数：1

SUBBR	短立即数减 Ws（带借位）

语法：

{标号:}	SUBBR.b	Ws,	lit7	Wd
	SUBBR.bz	[Ws],		[Wd]
	SUBBR{.w}	[Ws++],		[Wd++]
	SUBBR.l	[Ws--],		[Wd--]
		[++Ws],		[++Wd]
		[--Ws],		[--Wd]

操作数：Ws ∈ [W0 ...W15]；lit7 ∈ [0 ...127；Wd ∈ [W0 ...W15]
注：立即数经过零扩展至操作所选的数据大小

操作：lit7 - (Ws) - (C) → Wd

受影响的状态位：C、N、OV 和 Z

指令编码：

1110	111L	dddd	ssss	pppq	qqkk	kkkk	Bk10

说明：从零扩展无符号立即数中减去源寄存器 Ws 的内容和进位标志，并将结果存放在目标寄存器 Wd 中。
Z 位为“粘住”位（只能清零）。
L 和 B 位用于选择操作数据宽度。
k 位用于提供立即数操作数（op[2]中的 MSb）。
s 位用于选择源寄存器。
d 位用于选择目标寄存器。
p 位用于选择源寻址模式。
q 位用于选择目标寻址模式。

指令字数：1

指令周期数：1

SUBB	f 减 Wn（带进位）

语法：

{标号:}	SUBB.b	f	,Wn	{,WREG}
	SUBB.bz			
	SUBB{.w}			
	SUBB.l			

操作数：f ∈ [0 ...64 KB]；Wn ∈ [W0 ...W15]

操作：(f) - (Wn) - (C) → 由 D 指定的目标寄存器

受影响的状态位：C、N、OV 和 Z

指令编码：

1110	111L	ssss	ffff	ffff	ffff	ffff	BD01

.....（续）

SUBB	f 减 Wn（带进位）

说明：从文件寄存器内容中减去工作寄存器的内容与进位位，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wn，则 D = 0 并将结果存放在 Wn 中；否则，D = 1 并将结果存放在文件寄存器中。
Z 位为“粘住”位（只能清零）。

L 和 B 位用于选择操作数据宽度。

D 位用于选择目标寄存器。

f 位用于选择文件寄存器的地址。

s 位用于选择工作寄存器。

指令字数：1

指令周期数：1

SUB	Wn 减 f

语法：

{标号:}	SUBR.b	f	,Wn	{,WREG}
	SUBR.bz			
	SUBR{.w}			
	SUBR.l			

操作数：f ∈ [0 ...64 KB]；Wn ∈ [W0 ...W15]

操作：(Wn) - (f) → 由 D 指定的目标寄存器

受影响的状态位：C、N、OV 和 Z

指令编码：

1110	010L	ssss	ffff	ffff	ffff	ffff	BD01

说明：从工作寄存器内容中减去文件寄存器的内容，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wn，则 D = 0 并将结果存放在 Wn 中；否则，D = 1 并将结果存放在文件寄存器中。
L 和 B 位用于选择操作数据宽度。

D 位用于选择目标寄存器。

f 位用于选择文件寄存器的地址。

s 位用于选择工作寄存器。

指令字数：1

指令周期数：1

SUB	Wn 减短立即数

语法：

{标号:}	SUB.l	lit5,	Wn

操作数：lit5 ∈ [0 ...31]；Wn ∈ [W0 ...W15]

操作：(Wn) - lit5 → Wn

受影响的状态位：C、N、OV 和 Z

指令编码：

0111	011k	kkkk	ssss

说明：从工作寄存器内容中减去零扩展立即数操作数，并将结果存放在工作寄存器 Wn 中。如果立即数 > 31 且/或需要字或字节操作，请汇编为 SUBLW 指令。
s 位用于选择工作寄存器。

k 位用于指定立即数操作数。

指令字数：0.5

指令周期数：1

SUB	Ws 减短立即数

语法：

{标号:}	SUB.b	Ws,	lit7,	Wd
	SUB.bz	[Ws],		[Wd]
	SUB{.w}	[Ws++],		[Wd++]
	SUB.l	[Ws--],		[Wd--]

.....（续）

SUB	Ws 减短立即数			
		[++Ws],	[++Wd]	
		[--Ws],	[--Wd]	

操作数：Ws ∈ [W0 ...W15]；lit7 ∈ [0 ...127]；Wd ∈ [W0 ...W15]

操作：(Ws) - lit7 → Wd

注：立即数经过零扩展至操作所选的数据大小

受影响的状态位：C、N、OV 和 Z

指令编码：	1110	010L	dddd	ssss	pppq	qqkk	kkkk	Bk10

说明：从源寄存器 Ws 内容中减去零扩展无符号立即数操作数，并将结果存放在目标寄存器 Wd 中。
L 和 B 位用于选择操作数据宽度。

k 位用于提供立即数操作数（op[2]中的 MSb）。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

指令字数：1

指令周期数：1

SUB			Wn 减立即数				
语法：	{标号:}	SUB.b	lit16,	Wn			
		SUB.bz					
		SUB{.w}					
		SUB.l					

操作数：lit16 ∈ [0 ...65535]；Wn ∈ [W0 ...W15]

操作：(Wn) - lit16 → Wn

受影响的状态位：C、N、OV 和 Z

指令编码：	1100	010L	ssss	kkkk	kkkk	kkkk	kkkk	B010

说明：从工作寄存器内容中减去零扩展立即数操作数，并将结果存放在工作寄存器 Wn 中。如果立即数 <= 31 且需要长字操作，请汇编为 SUBLN 指令。
L 和 B 位用于选择操作数据宽度。

s 位用于选择工作寄存器。

k 位用于指定立即数操作数。

指令字数：1

指令周期数：1

SUBR		Ws 减 Wb				
语法：	{标号:}	SUBR.b	Wb,	Ws,	Wd	
		SUBR.bz		[Ws],	[Wd]	
		SUBR{.w}		[Ws++],	[Wd++]	
		SUBR.l		[Ws--],	[Wd--]	
				[++Ws],	[++Wd]	
				[--Ws],	[--Wd]	

操作数：Wb ∈ [W0 ...W15]；Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]

操作：(Ws) - (Wb) → Wd

受影响的状态位：C、N、OV 和 Z

指令编码：	S110	110L	dddd	ssss	pppq	qqww	wwUU	BU00

.....（续）

SUBR	Ws 减 Wb
说明：	从源寄存器 Ws 内容中减去基准寄存器 Wb 的内容，并将结果存放在目标寄存器 Wd 中。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 s 位用于选择源寄存器。 w 位用于选择基准寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。
指令字数：	1 或 0.5
指令周期数：	1

SUBR	短立即数减 Ws				
语法：	{标号:}	SUBR.b SUBR.bz SUBR{.w} SUBR.l	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws],	lit7,	Wd [Wd] [Wd++] [Wd--] [++Wd] [--Wd]
操作数：	Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]；lit7 ∈ [1 ...63] （当 lit7 = 0 时，请参见注释）				
操作：	lit7 - (Ws) → Wd 注：立即数经过零扩展至操作所选的数据大小				
受影响的状态位：	C、N、OV 和 Z				
指令编码：	S111 101L dddd ssss pppq qqkk kkkk Bk00				
说明：	从零扩展无符号立即数中减去源寄存器 Ws 的内容，并将结果存放在目标寄存器 Wd 中。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 k 位用于提供立即数操作数（op[2]中的 MSb）。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。				
指令字数：	1 或 0.5				
指令周期数：	1				

SUB	f 减 Wn			
语法：	{标号:}	SUB.b SUB.bz SUB{.w} SUB.l	f	Wn	{,WREG}
操作数：	f ∈ [0 ...64 KB]；Wn ∈ [W0 ...W15]				
操作：	(f) - (Wn) → 由 D 指定的目标寄存器				
受影响的状态位：	C、N、OV 和 Z				
指令编码：	1110 110L ssss ffff ffff ffff ffff BD01				

.....（续）

SUB	f 减 Wn
说明：	从文件寄存器内容中减去工作寄存器的内容，并将结果存放在由 D 指定的目标寄存器中。如果指定了可选的 Wn 目标寄存器，则 D = 0 并将结果存放在 Wn 中；否则，D = 1 并将结果存放在文件寄存器中。 L 和 B 位用于选择操作数据宽度。 D 位用于选择目标寄存器。 f 位用于选择文件寄存器的地址。 s 位用于选择工作寄存器。
指令字数：	1
指令周期数：	1

SWAP		Wn 内容字、字节或半字节交换	
语法：	{标号:}	SWAP.b	Wn
		SWAP.bz	
		SWAP{.w}	
		SWAP.l	
操作数：	Wn ∈ [W0 ...W15]		
操作：	字节模式： Wn[7:4] ↔ Wn[3:0] 扩展字节模式：Wn[7:4] ↔ Wn[3:0]；24'h000000 → Wn[31:8] 字模式： Wn[15:8] ↔ Wn[7:0] 长字模式：Wn[31:24] ↔ Wn[7:0]；Wn[23:16] ↔ Wn[15:8]		
受影响的状态位：	无		
指令编码：	1111 011L UUUU ssss UUUU UUUU UUUU B111		
说明：	如果处于长字模式，则在 Wn 寄存器的 msw 和 lsw 中均进行字节交换。 如果处于字模式，则在 Wn 寄存器的 lsw 中进行字节交换。Wn[31:16]清零。 如果处于扩展字节模式，则在 Wn 寄存器的 LSb 中进行半字节交换。Wn[31:8]清零。 如果处于字节模式，则在 Wn 寄存器的 LSb 中进行半字节交换。Wn[31:8]不受影响。 L 和 B 位用于选择操作数据宽度。 s 位用于选择工作寄存器。 长字模式下的字节交换用于数据字节顺序转换。		
指令字数：	1		
指令周期数：	1		

TSTF		测试 f 并（可选）传送至 Wd			
语法：	{标号:}	TST.b	f	{,Wnd}	{,WREG}
		TST.bz			
		TST{.w}			
		TST.l			
操作数：	f ∈ [0 ...64 KB]；Wnd ∈ [W0 ...W14]；				
操作：	(f) → 由 D 指定的目标寄存器				
受影响的状态位：	Z 和 N				
指令编码：	1101 001L dddd ffff ffff ffff ffff BD01				
说明：	读取并测试文件寄存器的内容。如果指定了可选的 Wnd，则 D = 0 并将读取的数据存放在 Wnd 中。否则，D = 1 并仅修改状态标志（即，测试文件寄存器），不写入数据。 L 和 B 位用于选择操作数据宽度。 D 位用于选择是否写入数据。 f 位用于选择文件寄存器的地址。				
指令字数：	1				

.....（续）

TSTF		测试 f 并（可选）传送至 Wd			
指令周期数：	1				

ULNK		释放堆栈帧			
语法：	{标号:}	ULNK			
操作数：	无				
操作：	W14 → W15 (W15) - 4 → W15 [实现： (W14) - 4 → W15] (TOS) → W14				
受影响的状态位：	无				
指令编码：	0111	001U	UUUU	0011	
说明：	该指令用于释放堆栈帧并调整堆栈指针和帧指针。 **注：** 1. 该指令只能对按长字对齐的操作数进行操作。				
指令字数：	0.5				
指令周期数：	1				

XOR		Wb 和 Ws 逻辑异或			
语法：	{标号:}	XOR.b XOR.bz XOR{.w} XOR.l	Wb,	Ws, [Ws], [Ws++], [Ws--], [++Ws], [--Ws], SR	Wd [Wd] [Wd++] [Wd--] [++Wd] [--Wd] SR
操作数：	Wb ∈ [W0 ...W15]；Ws ∈ [W0 ...W15]；Wd ∈ [W0 ...W15]				
操作：	(Wb).XOR.(Ws) → Wd				
受影响的状态位：	N 和 Z				
指令编码：	S111	000L	dddd	ssss	pppq qqww wwUU BU00
说明：	将源寄存器 Ws 中的内容和基准寄存器 Wb 中的内容进行逻辑异或运算，并将结果存放在目标寄存器 Wd 中。 S 位用于选择指令大小。 L 和 B 位用于选择操作数据宽度。 s 位用于选择源寄存器。 w 位用于选择基准寄存器。 d 位用于选择目标寄存器。 p 位用于选择源寻址模式。 q 位用于选择目标寻址模式。 **注：** 1. 选择 SR 时，SR 数据写操作将优先于由 XOR 运算引起的任何 SR 更新操作。 2. 写入 SR 时，不允许使用.bz 数据大小/模式。				
指令字数：	1 或 0.5				
指令周期数：	1				

XOR		Ws 和短立即数逻辑异或			
语法：	{标号:}	XOR.b XOR.bz	Ws, [Ws],	lit7,	Wd [Wd]

.....（续）

XOR	Ws 和短立即数逻辑异或		
	XOR{.w}	[Ws++],	[Wd++]
	XOR.l	[Ws--],	[Wd--]
		[++Ws],	[++Wd]
		[--Ws],	[--Wd]
		SR	SR

操作数：Ws ∈ [W0 ...W15]；lit7 ∈ [0 ...127]；Wd ∈ [W0 ...W15]

操作：(Ws).XOR.lit7 → Wd

注：立即数经过零扩展至操作所选的数据大小

受影响的状态位：N 和 Z

指令编码：

1111	000L	dddd	ssss	pppq	qqkk	kkkk	Bk10

说明：将源寄存器 Ws 中的内容和零扩展立即数操作数进行逻辑异或运算，并将结果存放在目标寄存器 Wd 中。
L 和 B 位用于选择操作数据宽度。

k 位用于提供立即数操作数（op[2]中的 MSb）。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

q 位用于选择目标寻址模式。

注：

1. 选择 SR 时，SR 数据写操作将优先于由 XOR 运算引起的任何 SR 更新操作。
2. 写入 SR 时，不允许使用.bz 数据大小/模式。

指令字数：1

指令周期数：1

XOR	立即数和 Wn 逻辑异或			
语法：{标号:}	XOR.b	lit16,	Wn	
	XOR.bz		SR	
	XOR{.w}			
	XOR.l			

操作数：lit16∈ [0 ... 65535]；Wn ∈ [W0 ...W15]；状态寄存器（SR）

操作：(Wn) .XOR. lit16 → Wn 或(SR) .XOR. lit16 → SR

受影响的状态位：N 和 Z

指令编码：

1101	000L	ssss	kkkk	kkkk	kkkk	kkkk	BT10

说明：将零扩展立即数操作数和工作寄存器 Wn 或 SR 中的内容进行逻辑异或运算，并将结果存放在工作寄存器 Wn 或 SR 中。
L 和 B 位用于选择操作数据宽度。

s 位用于选择工作寄存器。

k 位用于指定立即数操作数。

T 位用于在 Ws（T = 0）与 SR（T = 1）目标寄存器之间进行选择。

注：

1. 选择 SR 时，SR 数据写操作将优先于由 XOR 运算引起的任何 SR 更新操作。
2. 写入 SR 时，不允许使用.bz 数据大小/模式。

指令字数：1

指令周期数：1

XOR	f 和 Wn 逻辑异或			
语法：{标号:}	XOR.b	f	,Wn	{,WREG}

.....（续）

XOR	f 和 Wn 逻辑异或
	XOR.bz
	XOR{.w}
	XOR.l

操作数：f ∈ [0 ...64 KB]；Wn ∈ [W0 ...W15]

操作：(f).XOR.(Wn) → 由 D 指定的目标寄存器

受影响的状态位：N 和 Z

指令编码：

1111	000L	ssss	ffff	ffff	ffff	ffff	BD01

说明：将工作寄存器中的内容和文件寄存器中的内容进行逻辑异或运算，并将结果存放在目标寄存器中。如果指定了可选的 Wn，则 D = 0 并将结果存放在 Wn 中；否则，D = 1 并将结果存放在文件寄存器中。
L 和 B 位用于选择操作数据宽度。

D 位用于选择目标寄存器。

f 位用于选择文件寄存器的地址。

s 位用于选择工作寄存器。

指令字数：1

指令周期数：1

ZE		零扩展 Ws		
语法：	{标号:}	ZE.bz	Ws,	Wnd
		ZE.w	[Ws],	
			[Ws++],	
			[Ws--],	
			[++Ws],	
			[--Ws],	
			[Ws+Wb],	

操作数：Ws ∈ [W0 ...W15]；Wb ∈ [W0 ...W15]；Wnd ∈ [W0 ...W14]

操作：如果是字节模式：
Ws[7:0] → Wd[7:0]；

0 → Wnd[31:8]；

如果是字模式：

Ws[15:0] → Wd[15:0]；

0 → Wnd[31:16]；

受影响的状态位：C、N 和 Z

指令编码：

S111	111B	dddd	ssss	pppU	UUww	wwUU	UU00

说明：将 Ws 中的 8 位或 16 位有符号值零扩展为 32 位值，然后将结果回写到 Wnd。
N 总是清零。C 总是置 1。

S 位用于选择指令大小。

B 位用于选择字节或字操作。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

p 位用于选择源寻址模式。

w 位用于定义偏移量 Wb。

注：

1. 该操作总是写入一个长字。

指令字数：1 或 0.5

指令周期数：1

4.8 FPU 指令编码和操作码字段说明

表 4-11. FPU 指令编码字段说明中使用的符号

字段	说明
lit16	16 位无符号立即数 Î {0...65535}
index	进入 32 条目 SP 和 DP 常量表的立即数地址索引
.s	32 位单精度选项
.d	64 位双精度选项
Fd	32 个目标浮点寄存器{F0..F31}之一（寄存器直接寻址）
Fb	32 个源浮点寄存器{F0..F31}之一（寄存器直接寻址）
Fs	32 个源浮点寄存器{F0..F31}之一（寄存器直接寻址）
SUB、INF、FN、FZ、FNAN、 GT、LT、EQ、UN、 SUBO、HUGI、INX、UDF、OVF、 DIV0 和 INVAL	FPU 状态位（FSR） **注：**当某个指令无法发出某个异常信号时，对应的异常标志总是由该指令清零（粘住标志位不受影响）

表 4-12. FPU 指令操作码字段说明

字段	说明
P	选择单精度（P = 0）或双精度（P = 1）运算
R	在 FPU 协处理器特殊寄存器（R = 1）或 F-reg（R = 0）之间进行选择
U	未用（无关）指令位。汇编器赋值 0
ddddd	R = 0：FPU 协处理器 Fd 目标寄存器选择：00000 = F0；11111 = F31 R = 1：FPU 协处理器特殊寄存器选择
eee	FPU 舍入模式选择
kkkk kkkk kkkk kkkk	16 位立即数字段，常量数据
sssss	R = 0：FPU 协处理器 Fs 源寄存器选择：00000 = F0；11111 = F31 R = 1：FPU 协处理器特殊寄存器选择

4.9 浮点指令说明

ABS		求 Fs 的绝对值							
语法：	{标号:}	ABS.s	Fs,	Fd					
		ABS.d							
操作数（.s）：	Fs ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]								
操作数（.d）：	Fs ∈ [F0, F2 ... F30]；Fd ∈ [F0, F2 ... F30]								
操作：	ABS Fs → Fd								
受影响的状态位：	SUBO								
可能异常：	SUBO								
指令编码：	1000	100P	zzdd	dddd	ssss	UUUU	UUUU	0010	
说明：	将源寄存器 Fs 的内容的符号清零，并将结果存放在目标寄存器 Fd 中。 P 位用于选择单精度（.s）或双精度（.d）运算。 z 位用于选择目标协处理器。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。								
指令字数：	1								
SP 执行周期数：	1								
DP 执行周期数：	1								
重复率：	1								

注：该指令只影响符号位，不会放弃 sNaN 输入操作数。

ADD		Fb 与 Fs 相加							
语法：	{标号:}	ADD.s Fb,	Fs,	Fd					
		ADD.d							
操作数（.s）：	Fs ∈ [F0 ...F31]；Fb ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]								
操作数（.d）：	Fs ∈ [F0, F2 ...F30]；Fb ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]								
操作：	Fb + Fs → Fd								
受影响的状态位：	SUBO、INX、UDF、OVF 和 INVAL（见注 1）								
可能异常：	SUBO、INX、UDF、OVF 和 INVAL								
指令编码：	1000	000P	zzdd	dddd	ssss	wwww	wUUU	0010	
说明：	将源寄存器 Fb 中的内容与寄存器 Fs 中的内容相加，并将结果存放在目标寄存器 Fd 中。 (∞+(-∞))将导致可区分的 qNaN 并发出 INVAL 异常信号。 P 位用于选择单精度（.s）或双精度（.d）运算。 z 位用于选择目标协处理器。 s 位用于选择源寄存器。 w 位用于选择基准寄存器。 d 位用于选择目标寄存器。 **注：** 1. 如果尚未置 1，则相应的粘住异常状态也将置 1。 2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。								
指令字数：	1								
SP 执行周期数：	2								
DP 执行周期数：	2								
重复率：	1								

AND		浮点控制/状态寄存器和立即数逻辑与		
语法：	{标号:}	AND	lit16,	FSR

.....（续）

AND	浮点控制/状态寄存器和立即数逻辑与
	FCR
	FEAR

操作数：lit16 ∈ [0 ...65535]

操作：(FP 特殊寄存器[15:0]).AND.lit16 → FP 特殊寄存器[15:0]

受影响的状态位：无（作为指令逻辑与运算的结果时除外）

可能异常：无

指令编码：

1001	000U	zzss	kkkk	kkkk	kkkk	kkkk	0010

说明：将所选浮点协处理器特殊寄存器的 lsw 内容和立即数操作数进行逻辑与运算，并将结果回写到所选寄存器的 lsw。

z 位用于选择目标协处理器。

s 位用于选择浮点协处理器源寄存器。

k 位用于指定 16 位立即数操作数。

注：

1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
2. 该指令的作用只是将目标寄存器中的位清零。执行该指令不会产生 FPU 中断。
3. FEAR 目标包括 EACE 控制/状态位。

指令字数：1

执行周期数：1

重复率：1

COS	计算 Fs 的余弦

语法：{标号:} COS.s Fs, Fd

操作数（.s）：Fs ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]

操作：cos(Fs) → Fd

受影响的状态位：SUBO、INX、UDF 和 INVAL（见注 1）

可能异常：SUBO、INX、UDF 和 INVAL

指令编码：

1000	110P	zzdd	dddd	ssss	UUUU	UUUU	0110

说明：计算源寄存器 Fs 内容的余弦（其中 Fs = [2 π k + θ rads]），并将结果存放在目标寄存器 Fd 中。

z 位用于选择目标协处理器。

s 位用于选择源寄存器

d 位用于选择目标寄存器。

注：

1. 如果尚未置 1，则相应的粘住异常状态也将置 1。
2. 仅当使用单精度浮点数时才支持该操作。
3. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。

指令字数：1

SP 执行周期数：4

重复率：1

CPQ	比较 Fb 和 Fs，不发信号

语法：{标号:} CPQ.s Fb, Fs

CPQ.d

操作数（.s）：Fs ∈ [F0 ...F31]；Fb ∈ [F0 ...F31]

操作数（.d）：Fs ∈ [F0, F2 ...F30]；Fb ∈ [F0, F2 ...F30]

.....（续）

CPQ	比较 Fb 和 Fs，不发信号
操作：	Fb - Fs（带 IEEE 发信号谓词）
受影响的状态位：	LT、GT、EQ、UN、SUBO 和 INVAL（见注 1）
可能异常：	SUBO 和 INVAL
指令编码：	1000 101P zzUU UUUs ssss wwww wUUU 1010
说明：	比较源寄存器 Fb 与 Fs 的内容（Fb - Fs），然后设置标志，但不存储结果。仅当任一源操作数为 sNaN 时，才会产生无效异常信号。qNaN 源操作数不会导致产生异常信号。 如果任一源操作数为 sNaN 或 qNaN，则设置 FSR.UN = 1。 根据结果设置 LT、GT 和 EQ 状态位。 P 位用于选择单精度（32 位）或双精度（64 位）运算。 z 位用于选择目标协处理器。 s 位用于选择源寄存器。 w 位用于选择基准寄存器。 **注：** 1. 如果尚未置 1，则相应的粘住异常状态也将置 1。 2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。 3. 该指令将-0 和+0 视为等效。
指令字数：	1
SP 执行周期数：	1
DP 执行周期数：	1
重复率：	1

CPS	比较 Fb 和 Fs，发信号
语法：	{标号:} CPS.s Fb, Fs CPS.d
操作数（.s）：	Fs ∈ [F0 ...F31]；Fb ∈ [F0 ...F31]
操作数（.d）：	Fs ∈ [F0, F2 ...F30]；Fb ∈ [F0, F2 ...F30]
操作：	Fb - Fs（带 IEEE 发信号谓词）
受影响的状态位：	LT、GT、EQ、UN、SUBO 和 INVAL（见注 1）
可能异常：	SUBO 和 INVAL
指令编码：	1000 101P zzUU UUUs ssss wwww wUUU 1110
说明：	比较源寄存器 Fb 与 Fs 的内容（Fb - Fs），然后设置标志，但不存储结果。如果源操作数为 sNaN 或 qNaN，则将产生无效异常信号。 如果任一源操作数为 sNaN 或 qNaN，则设置 FSR.UN = 1。 根据结果设置 LT、GT 和 EQ 状态位。 P 位用于选择单精度（32 位）或双精度（64 位）运算。 z 位用于选择目标协处理器。 s 位用于选择源寄存器。 w 位用于选择基准寄存器。 **注：** 1. 如果尚未置 1，则相应的粘住异常状态也将置 1。 2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。 3. 该指令将-0 和+0 视为等效。
指令字数：	1
SP 执行周期数：	1
DP 执行周期数：	1
重复率：	1

DI2F		将双字（64 位）整数转换为浮点数			
语法：	{标号:}	DI2F.s{rnd} DI2F.d{rnd}	Fs,	Fd	

操作数（.s）：Fs ∈ [F0, F2 ...F30]；Fd ∈ [F0 ...F31]；rnd ∈ [e, z, p, n]

操作数（.d）：Fs ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]；rnd ∈ [e, z, p, n]

操作：Fs（整数） → Fd（浮点数）

受影响的状态位：INX[1]

可能异常：INX

指令编码：

1000	111P	zzdd	dddd	ssss	UUUU	Ueee	1110

说明：将源寄存器 Fs 的 64 位整数内容转换为浮点数并进行舍入，然后将结果存放在目标寄存器 Fd 中。如果未指定舍入模式{rnd}，则默认为由 FCR.RND[1:0]定义的舍入模式。如果无法精确地将整数表示为浮点值，则会产生不精确的异常信号。

P 位用于选择单精度（.s）或双精度（.d）运算。

e 位用于定义应用的舍入模式。

z 位用于选择目标协处理器。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

注：

1. 如果尚未置 1，则相应的粘住异常状态也将置 1。
2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。

指令字数：1

SP 执行周期数：2

DP 执行周期数：2

重复率：1

DIV		有符号浮点除法			
语法：	{标号:}	DIV.s DIV.d	Fb	Fs,	Fd

操作数（.s）：Fs ∈ [F0 ...F31]；Fb ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]

操作数（.d）：Fs ∈ [F0, F2 ...F30]；Fb ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]

操作：Fb ÷ Fs → Fd

受影响的状态位：SUBO、INX、UDF、OVF、DIV0 和 INVAL（见注 1）

可能异常：SUBO、INX、UDF、OVF、DIV0 和 INVAL

指令编码：

1000	010P	zzdd	dddd	ssss	wwww	wUUU	1010

说明：将源寄存器 Fb 的内容除以源寄存器 Fs 的内容，并将结果存放在目标寄存器 Fd 中。

(∞/0)的结果是带正确符号的无穷大。而（(0/0)或(∞/∞)）将导致可区分的 qNaN 并发出 FPU INVAL 异常信号。

P 位用于选择单精度（.s）或双精度（.d）运算。

z 位用于选择目标协处理器。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

d 位用于选择目标寄存器。

注：

1. 如果尚未置 1，则相应的粘住异常状态也将置 1。
2. 尝试以零作为除数时，可能会也可能不会发出 DIV0 信号，具体取决于操作数。
3. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。

指令字数：1

SP 执行周期数：11

.....（续）

DIV	有符号浮点除法
DP 执行周期数：	32
重复率：	待定

FLIM	强制执行有符号数据限制							
语法：	{标号:}	FLIM.s FLIM.d	Fb,	Fs,	Fd			
操作数（.s）：	Fs ∈ [F0 ...F31]；Fb ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]							
操作数（.d）：	Fs ∈ [F0, F2 ...F30]；Fb ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]							
操作：	如果(Fd) > (Fs)，则(Fs) → (Fd)； 如果(Fd) < (Fb)，则(Fb) → (Fd)；							
受影响的状态位：	SUBO 和 INVAL（见注 1）							
可能异常：	SUBO 和 INVAL							
指令编码：	1001	001P	zzdd	dddd	ssss	wwww	wUUU	0010
说明：	将 Fd 中的有符号浮点值同时与 Fs 中保存的有符号浮点值上限和 Fb 中保存的有符号浮点值下限进行比较。 如果 Fd 大于 Fs，则将 Fd 设置为 Fs 中保存的限值。 如果 Fd 小于 Fb，则将 Fd 设置为 Fb 中保存的限值。 如果 Fd 小于或等于 Fs 中的上限值，并且大于或等于 Fb 中的下限值，则不修改 Fd（即，数据处于限制范围内，不应用限制）。请参见注 2。 如果任一操作数（限值或要测试的值）为 sNaN 或 qNaN，则指令将返回 qNaN 作为结果。此外，如果任一操作数为 sNaN，则 INVAL 状态位将置 1。 如果上限值 Fs 被意外设置为小于下限值 Fb，则指令将返回可区分的 qNaN 作为结果并将 INVAL 状态位置 1。 这是一项有符号的比较操作。上下限值的符号可能相同。 P 位用于选择单精度（32 位）或双精度（64 位）运算。 z 位用于选择目标协处理器。 s 位用于选择源寄存器。 w 位用于选择基准寄存器。 d 位用于选择目标寄存器。 如果尚未置 1，则相应的粘住异常状态也将置 1。 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。							
指令字数：	1							
SP 执行周期数：	1							
DP 执行周期数：	1							
重复率：	1							

F2DI	将浮点数 Fs 转换为双字（64 位）整数							
语法：	{标号:}	F2DI.s{rnd} F2DI.d{rnd}	Fs,	Fd				
操作数（.s）：	Fs ∈ [F0 ...F31]；Fd ∈ [F0, F2 ...F30]；rnd ∈ [e, z, p, n]							
操作数（.d）：	Fs ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]；rnd ∈ [e, z, p, n]							
操作：	Fs（浮点数）→ Fd（整数）							
受影响的状态位：	SUBO、HUGI、INX 和 INVAL（见注 1）							
可能异常：	SUBO、HUGI、INX 和 INVAL							
指令编码：	1000	111P	zzdd	dddd	ssss	UUUU	Ueee	0110

.....（续）

F2DI	将浮点数 Fs 转换为双字（64 位）整数
说明：	将源寄存器 Fs 的浮点内容转换为舍入的 64 位整数，并将结果存放在目标寄存器 Fd 中。如果未指定舍入模式{rnd}，则默认为由 FCR.RND[1:0]定义的舍入模式。 如果源寄存器包含±NaN 或±∞，则将会发出 INVAL 异常信号，结果将为无限整数值 0x8000_0000_0000_0000（最大负整数）。 如果源寄存器包含有限浮点数，但结果舍入为大于 2^{63}-1 或小于-2^{63} 的数字，则将发出 HUGI 和 INVAL 异常信号，默认结果将分别为无限整数值 0x7FFF_FFFF_FFFF_FFFF 或 0x8000_0000_0000_0000。 如果源寄存器包含非规格化值，则结果将总是为零并将发出 INX 异常信号。 P 位用于选择单精度（.s）或双精度（.d）运算。 e 位用于定义应用的舍入模式。 z 位用于选择目标协处理器。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 **注：** 1. 如果尚未置 1，则相应的粘住异常状态也将置 1。 2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
指令字数：	1
SP 执行周期数：	1（可能变更，恕不另行通知）
DP 执行周期数：	2（可能变更，恕不另行通知）
重复率：	1

F2LI	将浮点数 Fs 转换为长字（32 位）整数
语法：	{标号:} F2LI.s{rnd} Fs, Fd F2LI.d{rnd}
操作数（.s）：	Fs ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]；rnd ∈ [e, z, p, n]
操作数（.d）：	Fs ∈ [F0, F2 ...F30]；Fd ∈ [F0 ...F31]；rnd ∈ [e, z, p, n]
操作：	Fs（浮点数）→ Fd（整数）
受影响的状态位：	SUBO、HUGI、INX 和 INVAL（见注 1）
可能异常：	SUBO、HUGI、INX 和 INVAL
指令编码：	1000 111P zzdd dds ssss UUUU Ueee 0010
说明：	将源寄存器 Fs 的浮点内容转换为舍入的 32 位整数，并将结果存放在目标寄存器 Fd 中。如果未指定舍入模式{rnd}，则默认为由 FCR.RND[1:0]定义的舍入模式。 如果源寄存器包含±NaN 或±∞，则将会发出 INVAL 异常信号，结果将为无限整数值 0x8000_0000（最大负整数）。 如果源寄存器包含有限浮点数，但结果舍入为大于 2^{31}-1 或小于-2^{31} 的数字，则将发出 HUGI 和 INVAL 异常信号，默认结果将分别为无限整数值 0x7FFF_FFFF 或 0x8000_0000。 如果源寄存器包含非规格化值，则结果将总是为零并将发出 INX 异常信号。 P 位用于选择单精度（.s）或双精度（.d）运算。 e 位用于定义应用的舍入模式。 z 位用于选择目标协处理器。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 **注：** 1. 如果尚未置 1，则相应的粘住异常状态也将置 1。 2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
指令字数：	1
SP 执行周期数：	1

.....（续）

F2LI	将浮点数 Fs 转换为长字（32 位）整数
DP 执行周期数：	2
重复率：	1

IOR	浮点控制/状态寄存器和立即数逻辑或
语法：	{标号:} IOR lit16, FSR FCR FEAR
操作数	lit16 ∈ [0 ...65535]
操作：	(FP 特殊寄存器[15:0]).IOR.lit16 → FP 特殊寄存器[15:0]
受影响的状态位：	无（作为指令逻辑或运算的结果时除外）
可能异常：	无
指令编码：	1001 000U zzss kkkk kkkk kkkk kkkk 0110
说明：	将所选浮点协处理器寄存器的 lsw 内容和立即数操作数进行逻辑或运算，并将结果回写到所选寄存器的 lsw。 z 位用于选择目标协处理器。 s 位用于选择浮点协处理器源寄存器。 k 位用于指定 16 位立即数操作数。 **注：** 1. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。 2. 该指令的作用只是将目标寄存器中的位置 1。执行该指令不会产生 FPU 中断。 3. FEAR 目标包括 EACE 控制/状态位。
指令字数：	1
执行周期数：	1
重复率：	1

LI2F	将长字（32 位）整数转换为浮点数
语法：	{标号:} LI2F.s{rnd} Fs, Fd LI2F.d{rnd}
操作数（.s）：	Fs ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]；rnd ∈ [e, z, p, n]
操作数（.d）：	Fs ∈ [F0 ...F31]；Fd ∈ [F0, F2 ...F30]；rnd ∈ [e, z, p, n][3]
操作：	Fs（整数） → Fd（浮点数）
受影响的状态位：	INX[1, 3]
可能异常：	INX
指令编码：	1000 111P zzdd dddd ssss UUUU Ueee 1010

.....（续）

LI2F	将长字（32 位）整数转换为浮点数

说明：

将源寄存器 Fs 的 32 位整数内容转换为浮点数并进行舍入，然后将结果存放在目标寄存器 Fd 中。如果未指定舍入模式{rnd}，则默认为由 FCR.RND[1:0]定义的舍入模式。如果无法精确地将整数表示为浮点值，则会产生不精确的异常信号（见注 3）。

P 位用于选择单精度（.s）或双精度（.d）运算。

e 位用于定义应用的舍入模式。

z 位用于选择目标协处理器。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

注：

1. 如果尚未置 1，则相应的粘住异常状态也将置 1。
2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
3. 长整数到双精度浮点数（LI2F.d）的转换总是精确的，因此不精确（INX）状态位永远不会置 1。类似地，舍入模式对 LI2F.d 结果没有影响。

指令字数：1

SP 执行周期数：1

DP 执行周期数：1

重复率：1

MAC	浮点有符号相乘并累加

语法：{标号:} MAC.s Fb, Fs, Fd

MAC.d

操作数（.s）：Fs ∈ [F0 ...F31]；Fb ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]

操作数（.d）：Fs ∈ [F0, F2 ...F30]；Fb ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]

操作：(Fd) + ((Fb) * (Fs)) → Fd；

受影响的状态位：SUBO、INX、UDF、OVF 和 INVAL（见注 1）

可能异常：SUBO、INX、UDF、OVF 和 INVAL

指令编码：

```
1000    010P    zzdd    dddd    ssss    wwww    wUUU    0010
```

说明：

将源寄存器 Fb 的有符号内容与源寄存器 Fs 的有符号内容相乘，并将乘积加到 Fd 中。

P 位用于选择单精度（.s）或双精度（.d）运算。

z 位用于选择目标协处理器。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

d 位用于选择目标寄存器。

注：

1. 如果尚未置 1，则相应的粘住异常状态也将置 1。
2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。

指令字数：1

SP 执行周期数：3

DP 执行周期数：4

重复率：1

MAX	选择 Fb 与 Fs 中的有符号最大值 {IEEE 754-2019 maxmum(x,y)}

语法：{标号:} MAX.s Fb, Fs, Fd

MAX.d

操作数（.s）：Fs ∈ [F0 ...F31]；Fb ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]

.....（续）

MAX	选择 Fb 与 Fs 中的有符号最大值 {IEEE 754-2019 maxmum(x,y)}
操作数（.d）：	Fs ∈ [F0, F2 ...F30]；Fb ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]
操作：	如果(Fs) >= (Fb)，则(Fs) → (Fd)； 否则(Fb) → (Fd)
受影响的状态位：	SUBO 和 INVAL（见注 1）
可能异常：	SUBO 和 INVAL
指令编码：	1000 001P zzdd dds s ssss wwww wUUU 0010
说明：	找到 Fb 与 Fs 中保存的两个浮点值中的有符号最大值。该指令符合 IEEE 754-2019 maximum()运算。比较源寄存器 Fb 与 Fs 的内容，然后将两个值中的最大值存放在目标寄存器 Fd 中。这是一项有符号的比较操作（见注 3）。如果 Fb = Fs（且符号相同），则 Fd 中将装入 Fs 的内容。 当两个输入操作数中至少有一个为 NaN 时，指令将返回 qNaN。 P 位用于选择单精度（32 位）或双精度（64 位）运算。 z 位用于选择目标协处理器。 s 位用于选择源寄存器。 w 位用于选择基准寄存器。 d 位用于选择目标寄存器。 **注：** 1. 如果尚未置 1，则相应的粘住异常状态也将置 1。 2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。 3. 操作数值-0 小于+0。
指令字数：	1
SP 执行周期数：	1
DP 执行周期数：	1
重复率：	1

MAXNM	选择 Fb 与 Fs 中的有符号最大值 {IEEE 754-2019 maxmumNumber(x,y)}
语法：	{标号:} MAXNM.s Fb, Fs, Fd MAXNM.d
操作数（.s）：	Fs ∈ [F0 ...F31]；Fb ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]
操作数（.d）：	Fs ∈ [F0, F2 ...F30]；Fb ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]
操作：	如果(Fs) >= (Fb)，则(Fs) → (Fd)； 否则(Fb) → (Fd)
受影响的状态位：	SUBO 和 INVAL（见注 1）
可能异常：	SUBO 和 INVAL
指令编码：	1000 001P zzdd dds s ssss wwww wUUU 0110

.....（续）

MAXNM	选择 Fb 与 Fs 中的有符号最大值 {IEEE 754-2019 maxmumNumber(x,y)}

说明：

找到 Fb 与 Fs 中保存的两个浮点值中的有符号最大值。该指令符合 IEEE 754-2019 maximumNumber()运算。

比较源寄存器 Fb 与 Fs 的内容，然后将两个值中的最大值存放在目标寄存器 Fd 中。如果 Fb = Fs，则 Fd 中将装入 Fs 的内容。这是一项有符号的比较操作（见注 3）。如果 Fb = Fs（且符号相同），则 Fd 中将装入 Fs 的内容。

当一个输入操作数为 NaN，另一个输入操作数为浮点数（即，不是 NaN）时，指令将返回浮点数。

如果两个输入操作数均为 NaN，指令将返回 qNaN。

P 位用于选择单精度（32 位）或双精度（64 位）运算。

z 位用于选择目标协处理器。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

d 位用于选择目标寄存器。

注：

1. 如果尚未置 1，则相应的粘住异常状态也将置 1。
2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
3. 操作数值-0 小于+0。

指令字数：1

SP 执行周期数：1

DP 执行周期数：1

重复率：1

MIN	选择 Fb 与 Fs 中的有符号最小值 {IEEE 754-2019 minmum(x,y)}

语法：{标号:} MIN.s Fb, Fs, Fd
MIN.d

操作数（.s）：Fs ∈ [F0 ...F31]；Fb ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]

操作数（.d）：Fs ∈ [F0, F2 ...F30]；Fb ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]

操作：如果(Fs) <= (Fb)，则(Fs) → (Fd)；
否则(Fb) → (Fd)

受影响的状态位：SUBO 和 INVAL（见注 1）

可能异常：SUBO 和 INVAL

指令编码：

1000	001P	zzdd	dddd	ssss	wwww	wUUU	1010

说明：

找到 Fb 与 Fs 中保存的两个浮点值中的有符号最小值。该指令符合 IEEE 754-2019 minimum()运算。

比较源寄存器 Fb 与 Fs 的内容，然后将两个值中的最小值存放在目标寄存器 Fd 中。这是一项有符号的比较操作（见注 3）。如果 Fb = Fs（且符号相同），则 Fd 中将装入 Fs 的内容。

当两个输入操作数中至少有一个为 NaN 时，指令将返回 qNaN。

P 位用于选择单精度（32 位）或双精度（64 位）运算。

z 位用于选择目标协处理器。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

d 位用于选择目标寄存器。

注：

1. 如果尚未置 1，则相应的粘住异常状态也将置 1。
2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
3. 操作数值-0 小于+0。

.....（续）

MIN	选择 Fb 与 Fs 中的有符号最小值 {IEEE 754-2019 minmum(x,y)}
指令字数：	1
SP 执行周期数：	1
DP 执行周期数：	1
重复率：	1

MINNM	选择 Fb 与 Fs 中的有符号最小值[minmumNumber()]
语法：	{标号:} MINNM.s Fb, Fs, Fd MINNM.d
操作数（.s）：	Fs ∈ [F0 ...F31]；Fb ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]
操作数（.d）：	Fs ∈ [F0, F2 ...F30]；Fb ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]
操作：	如果(Fs) <= (Fb)，则(Fs) → (Fd)； 否则(Fb) → (Fd)
受影响的状态位：	SUBO 和 INVAL（见注 1）
可能异常：	SUBO 和 INVAL
指令编码：	1000 001P zzdd dddd ssss wwww wUUU 1110
说明：	找到 Fb 与 Fs 中保存的两个浮点值中的有符号最小值。该指令符合 IEEE 754-2019 minimumNumber()运算。 比较源寄存器 Fb 与 Fs 的内容，然后将两个值中的最小值存放在目标寄存器 Fd 中。如果 Fb = Fs，则 Fd 中将装入 Fs 的内容。这是一项有符号的比较操作（见注 3）。如果 Fb = Fs（且符号相同），则 Fd 中将装入 Fs 的内容。 当一个输入操作数为 NaN，另一个输入操作数为浮点数（即，不是 NaN）时，指令将返回浮点数。 如果两个输入操作数均为 NaN，指令将返回 qNaN。 P 位用于选择单精度（32 位）或双精度（64 位）运算。 z 位用于选择目标协处理器。 s 位用于选择源寄存器。 w 位用于选择基准寄存器。 d 位用于选择目标寄存器。 **注：** 1. 如果尚未置 1，则相应的粘住异常状态也将置 1。 2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。 3. 操作数值-0 小于+0。
指令字数：	1
SP 执行周期数：	1
DP 执行周期数：	1
重复率：	1

MOV	将 Fs 中内容传送至 Fd
语法：	{标号:} MOV.s Fs, Fd MOV.d
操作数（.s）：	Fs ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]
操作数（.d）：	Fs ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]
操作：	Fs → Fd
受影响的状态位：	无
可能异常：	无
指令编码：	1000 011P zzdd dddd ssss UUUU UUUU 0010

.....（续）

MOV	将 Fs 中内容传送至 Fd

说明：

将源寄存器 Fs 的内容传送（复制）至目标寄存器 Fd。
P 位用于选择单精度（.s）或双精度（.d）运算。

z 位用于选择目标协处理器。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

注：对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。

指令字数：1

SP 执行周期数：1

DP 执行周期数：1

重复率：1

MOVC	将常数值传送至 Fd

语法：

```
{标号:}    MOVC.s index,    Fd
           MOVC.d
```

操作数（.s）：Fd ∈ [F0 ...F31]；index ∈ [0 ...31]

操作数（.d）：Fd ∈ [F0, F2 ... F30]；索引 ∈ [0 ...31]

操作：常量表（索引） → Fd

受影响的状态位：无

可能异常：无

指令编码：

```
1000    011P    zzdd    dddU    UUUU    UUUk    kkkk    0110
```

说明：

将由 5 位索引指定的常量值传送至 Fd 寄存器，如下表所定义。
P 位用于选择单精度（.s）或双精度（.d）运算。

z 位用于选择目标协处理器。

d 位用于选择目标寄存器。

k 位用于选择索引。

注：对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。

指令字数：1

SP 执行周期数：1

DP 执行周期数：1

重复率：1

单精度：

索引	数据	说明
31	32'H7FFF_FFFF	QNAN - RANGE ENDS
30	32'H7FC0_0000	QNAN - RANGE STARTS
29	32'H7FBF_FFFF	SNAN - RANGE ENDS
28	32'H7F80_0001	SNAN - RANGE STARTS
27	32'H7F80_0000	+INF
26	32'H7F7F_FFFF	+MAX MAG NORMAL
25	32'H0080_0000	+MIN MAG NORMAL
24	32'H007F_FFFF	+MAX MAG SUB-NORMAL
23	32'H0000_0001	+MIN MAG SUB-NORMAL
22	32'H0000_0000	+0.0
21	32'H8000_0000	-0.0
20	32'H8000_0001	-MIN MAG SUB-NORMAL
19	32'H807F_FFFF	-MAX MAG SUB-NORMAL
18	32'H8080_0000	-MIN MAG NORMAL

.....（续）

MOVC		将常数值传送至 Fd	
	17	32'HFF7F_FFFF	-MAX MAG NORMAL
	16	32'HFF80_0000	-INF
	15	32'HFF80_0001	SNAN - RANGE STARTS
	14	32'HFFBF_FFFF	SNAN - RANGE ENDS
	13	32'HFFC0_0000	QNAN - RANGE STARTS
	12	32'HFFFF_FFFF	QNAN - RANGE ENDS
	11	32'H3F35_04F3	1/√2 =0.70710...
	10	32'H3FB5_04F3	√2 = 1.41421...
	9	32'H4054_9A78	LOG2(10) = 3.321
	8	32'H3F31_7218	LN(2) = 0.6931471 ...
	7	32'H402D_F854	E = 2.71828...
	6	32'H3F49_0FDB	Π/4 = 0.785398...
	5	32'H3FC9_0FDB	Π/2 = 1.5707...
	4	32'H4049_0FDB	Π = 3.14159...
	3	32'H40C9_0FDB	2Π = 6.283185...
	2	32'H4000_0000	2.0
	1	32'H3F80_0000	1.0
	0	32'H4120_0000	10.0

双精度：

索引	数据	说明
31	64'H7FFF_FFFF_FFFF_FFFF	QNAN - RANGE ENDS
30	64'H7FF8_0000_0000_0000	QNAN RANGE STARTS
29	64'H7FF7_FFFF_FFFF_FFFF	SNAN RANGE ENDS
28	64'H7FF0_0000_0000_0001	SNAN RANGE STARTS
27	64'H7FF0_0000_0000_0000	+INF
26	64'H7FEF_FFFF_FFFF_FFFF	+MAX MAG NORMAL
25	64'H0010_0000_0000_0000	+MIN MAG NORMAL
24	64'H000F_FFFF_FFFF_FFFF	+MAX MAG SUB-NORMAL
23	64'H0000_0000_0000_0001	+MIN MAG SUB-NORMAL
22	64'H0000_0000_0000_0000	+0.0
21	64'H8000_0000_0000_0000	-0.0
20	64'H8000_0000_0000_0001	-MIN MAG SUB-NORMAL
19	64'H800F_FFFF_FFFF_FFFF	-MAX MAG SUB-NORMAL
18	64'H8010_0000_0000_0000	-MIN MAG NORMAL
17	64'HFFEF_FFFF_FFFF_FFFF	-MAX MAG NORMAL
16	64'HFFF0_0000_0000_0000	-INF
15	64'HFFF0_0000_0000_0001	SNAN RANGE STARTS
14	64'HFFF7_FFFF_FFFF_FFFF	SNAN RANGE ENDS
13	64'HFFF8_0000_0000_0000	QNAN RANGE STARTS
12	64'HFFFF_FFFF_FFFF_FFFF	QNAN RANGE ENDS
11	64'H3FE6_A09E_667F_3BCD	1/√2=0.7071...
10	64'H3FF6_A09E_667F_3BCD	√2=1.41421..
9	64'H400A_934F_0979_A371	LOG2(10)= 3.321
8	64'H3FE6_2E42_FEFA_39EF	LN(2)=0.69314
7	64'H4005_BF0A_8B14_5769	E = 2.71828...
6	64'H3FE9_21FB_5444_2D18	π/4= 0.7853...
5	64'H3FF9_21FB_5444_2D18	π/2 = 1.5707..
4	64'H4009_21FB_5444_2D18	π = 3.14159...

.....（续）

MOVC		将常数值传送至 Fd	
	3	64'H4019_21FB_5444_2D18	2 π = 6.28...
	2	64'H4000_0000_0000_0000	2.0
	1	64'H3FF0_0000_0000_0000	1.0
	0	64'H4024_0000_0000_0000	10.0

MUL		浮点有符号乘法			
语法：	{标号:}	MUL.s	Fb,	Fs,	Fd
		MUL.d			

操作数（.s）：Fs ∈ [F0 ...F31]；Fb ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]

操作数（.d）：Fs ∈ [F0, F2 ...F30]；Fb ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]

操作：(Fb) * (Fs) → Fd；

受影响的状态位：SUBO、INX、UDF、OVF 和 INVAL（见注 1）

可能异常：SUBO、INX、UDF、OVF 和 INVAL

指令编码：

1000	010P	zzdd	ddds	ssss	wwww	wUUU	0110

说明：将源寄存器 Fb 的有符号内容与源寄存器 Fs 的有符号内容相乘，并将乘积写入 Fd 中。
(0 * ∞)或(∞ * 0)将导致可区分的 qNaN 并发出 INVAL 异常信号。

P 位用于选择单精度（.s）或双精度（.d）运算。

z 位用于选择目标协处理器。

s 位用于选择源寄存器。

w 位用于选择基准寄存器。

d 位用于选择目标寄存器。

注：

1. 如果尚未置 1，则相应的粘住异常状态也将置 1。
2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。

指令字数：1

SP 执行周期数：3

DP 执行周期数：3

重复率：1

NEG		对 Fs 内容取反			
语法：	{标号:}	NEG.s	Fs,	Fd	
		NEG.d			

操作数（.s）：Fs ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]

操作数（.d）：Fs ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]

操作：-Fs → Fd

受影响的状态位：SUBO[1]

可能异常：SUBO

指令编码：

1000	100P	zzdd	ddds	ssss	UUUU	UUUU	0110

说明：对源寄存器 Fs 的内容的符号取反，并将结果存放在目标寄存器 Fd 中。
P 位用于选择单精度（.s）或双精度（.d）运算。

z 位用于选择目标协处理器。

s 位用于选择源寄存器。

d 位用于选择目标寄存器。

注：

1. 如果尚未置 1，则相应的粘住异常状态也将置 1。
2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。

.....（续）

NEG	对 Fs 内容取反
指令字数：	1
SP 执行周期数：	1
DP 执行周期数：	1
重复率：	1

注：该指令只影响符号位，不会放弃 sNaN 输入操作数。

SIN	计算 Fs 的正弦
语法：	{标号:} SIN.s Fs, Fd
操作数（.s）：	Fs ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]
操作：	Sin(Fs) → Fd
受影响的状态位：	SUBO、INX、UDF 和 INVAL（见注 1）
可能异常：	SUBO、INX、UDF 和 INVAL
指令编码：	1000 110P zzdd dddd ssss UUUU UUUU 0010
说明：	计算源寄存器 Fs 内容的正弦（其中 Fs = [2 π k + θ rads]），并将结果存放在目标寄存器 Fd 中。 z 位用于选择目标协处理器。 s 位用于选择源寄存器 d 位用于选择目标寄存器。 **注：** 1. 如果尚未置 1，则相应的粘住异常状态也将置 1。 2. 仅当使用单精度浮点数时才支持该操作。 3. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
指令字数：	1
SP 执行周期数：	4
重复率：	1

SQRT	计算 Fs 的平方根，结果存入 Fd
语法：	{标号:} SQRT.s Fs, Fd SQRT.d
操作数（.s）：	Fs ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]
操作数（.d）：	Fs ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]
操作：	√Fs → Fd
受影响的状态位：	SUBO、INX 和 INVAL（见注 1）
可能异常：	SUBO、INX 和 INVAL
指令编码：	1000 100P zzdd dddd ssss UUUU UUUU 1010
说明：	计算源寄存器 Fs 内容的平方根，并将结果存放在目标寄存器 Fd 中。 如果 Fs < 0，则将向目标写入可区分的 qNaN 并发出 FPU INVAL 异常信号。 P 位用于选择单精度（.s）或双精度（.d）运算。 z 位用于选择目标协处理器。 s 位用于选择源寄存器。 d 位用于选择目标寄存器。 **注：** 1. 如果尚未置 1，则相应的粘住异常状态也将置 1。 2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
指令字数：	1
SP 执行周期数：	10

.....（续）

SQRT	计算 Fs 的平方根，结果存入 Fd
DP 执行周期数：	13
重复率：	待定

SUB	Fb 减 Fs
语法：	{标号:} SUB.s Fb, Fs, Fd SUB.d
操作数（.s）：	Fs ∈ [F0 ...F31]；Fb ∈ [F0 ...F31]；Fd ∈ [F0 ...F31]
操作数（.d）：	Fs ∈ [F0, F2 ...F30]；Fb ∈ [F0, F2 ...F30]；Fd ∈ [F0, F2 ...F30]
操作：	Fb - Fs → Fd
受影响的状态位：	SUBO、INX、UDF、OVF 和 INVAL（见注 1）
可能异常：	SUBO、INX、UDF、OVF 和 INVAL
指令编码：	1000 000P zzdd dds s ssss wwww wUUU 0110
说明：	从基准寄存器 Fb 内容中减去源寄存器 Fs 中的内容，并将结果存放在目标寄存器 Fd 中。 (∞-∞)将导致可区分的 qNaN 并发出 FPU INVAL 异常信号。 P 位用于选择单精度（.s）或双精度（.d）运算。 z 位用于选择目标协处理器。 s 位用于选择源寄存器。 w 位用于选择基准寄存器。 d 位用于选择目标寄存器。 **注：** 1. 如果尚未置 1，则相应的粘住异常状态也将置 1。 2. 对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
指令字数：	1
SP 执行周期数：	2
DP 执行周期数：	2
重复率：	1

TST	测试 Fs
语法：	{标号:} TST.s Fs TST.d
操作数（.s）：	Fs ∈ [F0 ...F31]
操作数（.d）：	Fs ∈ [F0, F2 ...F30]
操作：	如果 Fs 为+/-非规格化，则 FSR.SUB = 1，否则 FSR.SUB = 0 如果 Fs 为+/-无限，则 FSR.INF = 1，否则 FSR.INF = 0 如果 Fs 为负，则 FSR.FN = 1，否则 FSR.FN = 0 如果 Fs 是+/-零，则 FSR.FZ = 1，否则 FSR.FZ = 0 如果 Fs 为 qNaN 或 sNaN，则 FSR.FNAN = 1，否则 FSR.FNAN = 0
受影响的状态位：	SUB、INF、FN、FZ、FNAN、
可能异常：	无
指令编码：	1001 010P zzUU UUUs ssss UUUU UUUU 0010
说明：	检查 Fs 的内容并相应地设置 FSR 状态。 P 位用于选择单精度（.s）或双精度（.d）运算。 z 位用于选择目标协处理器。 s 位用于选择源寄存器。 **注：**对于浮点协处理器宏，协处理器选择位域 zz = 2'b00。
指令字数：	1

.....（续）

TST	测试 Fs
SP 执行周期数：	1
DP 执行周期数：	1
重复率：	1

5. 内建函数

5.1 简介

内建函数使 C 语言程序员可以访问目前只能通过行内汇编访问但却十分有用、适用于广泛应用程序的汇编操作符或机器指令。内建函数在 C 源文件中编写，语法类似于函数调用，但它们被编译为直接实现函数的汇编代码，不涉及函数调用或库程序。

有一些原因使得提供内建函数优于要求程序员使用行内汇编。这些原因包括：

1. 提供用于特定目的的内建函数可以简化编码。
2. 使用行内汇编时，某些优化会被禁止。内建函数则不会如此。
3. 对于使用专用寄存器的机器指令，在编写行内汇编代码时，需要特别细心处理，以避免寄存器分配错误。内建函数让这个过程变得简单，因为您不需要关心每条机器指令的具体寄存器要求。

下面列出了这些内建函数，后面各小节将对各个内建函数进行详细说明。

- __builtin_addab
- __builtin_add
- __builtin_btg
- __builtin_clr
- __builtin_divf
- __builtin_divmodsd
- __builtin_divmodud
- __builtin_divsd
- __builtin_divud
- __builtin_dmaoffset
- __builtin_ed
- __builtin_edac
- __builtin_fbcl
- __builtin_lac
- __builtin_mac
- __builtin_modsd
- __builtin_modud
- __builtin_mpy
- __builtin_mpyn
- __builtin_msc
- __builtin_mulss
- __builtin_mulsu
- __builtin_mulus
- __builtin_muluu
- __builtin_nop
- __builtin_readsfr

- __builtin_return_address
- __builtin_sac
- __builtin_sacr
- __builtin_sftac
- __builtin_subab

本章仅介绍与 CPU 操作相关的内建函数。编译器提供用于写闪存程序存储器和更改振荡器设置等操作的其他内建函数。关于编译器内建函数的完整列表，请参见《用于 PIC24 MCU 和 dsPIC® DSC 的 MPLAB® C 编译器用户指南》（DS51284H_CN）。

5.2 内建函数列表

本节介绍编译器内建函数的程序员接口。由于这些函数是“内建”的，所以不存在与之关联的头文件。类似地，不存在与内建函数关联的命令行开关；它们总是可用。所选择的内建函数名称使它们属于编译器的命名空间（均具有前缀：_builtin_），所以它们不会与程序员命名空间中的函数或变量名称发生冲突。

_builtin_addab

说明：
将累加器 A 和 B 相加，并将结果回写到指定的累加器。例如：

```
register int result asm("A");
register int B asm("A");
result = _builtin_addab(result,B);
```

将生成：

```
add A
```

原型：

```
int _builtin_addab(int Accum_a, int Accum_b);
```

参数：
Accum_a 要相加的第一个累加器。

Accum_b 要相加的第二个累加器。

返回值：
将加法结果返回到累加器。

汇编操作符/机器指令：
add

错误消息：
如果结果不是累加器寄存器，则将显示一条错误消息。

_builtin_add

说明：
将 value 加到 result 指定的累加器，并按立即数指定的移位量进行移位。例如：

```
register int result asm("A");
int      value;
result = _builtin_add(result,value,0);
```

如果 value 存放在 w0 中，将生成以下代码：

```
add w0, #0, A
```

原型：

```
int   _builtin_add(int Accum,int value,
const int shift);
```

.....（续）

_builtin_add

参数：

Accum 要相加的累加器。

value 要与累加器值相加的整型数。

shift 对得到的累加器值的移位量。

返回值：

将移位后的加法结果返回到累加器。

汇编操作符/机器指令：

`add`

错误消息：

如果出现以下情况，将显示一条错误消息：

- 结果不是累加器寄存器
- 参数 0 不是累加器
- 移位值不是处于范围内的立即数

_builtin_btg

说明：

该函数将生成一条 `btg` 机器指令。一些示例包括：

```
int i;   /* near by default */
int l    _attribute_((far));
struct foo {
  int bit1:1;
} barbits;
int  bar;
void some_bittoggles() {
  register int j asm("w9");
  int k;
  k = i;
  _builtin_btg(&i,1);
  _builtin_btg(&j,3);
_builtin_btg(&k,4);
  _builtin_btg(&l,11);
  return j+k;
}
```

请注意，获取位于寄存器中的变量的地址时，编译器会产生警告，并导致寄存器被保存到堆栈中（从而可以获取它的地址）；建议不要使用这种形式。该警告仅适用于由编程人员显式存放在寄存器中的变量。

原型：

```
void _builtin_btg(unsigned int *, unsigned int 0xn);
```

参数：

*指向要翻转其中某个位的数据项的指针。

0x*n* 介于 0 到 15 范围内的立即数。

返回值：

返回一条 `btg` 机器指令。

汇编操作符/机器指令：

`btg`

错误消息：

如果参数值不处于范围内，将显示一条错误消息。

_builtin_clr

说明：
清零指定的累加器。例如：

```
register int result asm("A");
result = _builtin_clr();
```

将生成：

```
clr A
```

原型：

```
int _builtin_clr(void);
```

参数：
无。

返回值：
将清零值结果返回到累加器。

汇编操作符/机器指令：
`clr`

错误消息：
如果结果不是累加器寄存器，则将显示一条错误消息。

_builtin_divf

说明：
计算商：*num*/*den*。如果 *den* 为零，会发生数学错误异常。函数的参数和结果都是无符号的。

原型：

```
unsigned int _builtin_divf(unsigned int num,
unsigned int den);
```

参数：
num 分子。

den 分母。

返回值：
返回商的无符号整型值：*num*/*den*。

汇编操作符/机器指令：
`div.f`

_builtin_divmodsd

说明：
发出 16 位架构的本机有符号除法支持。值得注意的是，如果商无法装入 16 位结果中，结果（包括余数）将是非预期的。这种形式的内建函数会同时捕捉商和余数。

原型：

```
signed int  _builtin_divmodsd(
signed long dividend, signed int divisor,
signed int  *remainder);
```

参数：
dividend 被除数。

divisor 除数。

remainder 指向余数的指针。

返回值：
商和余数。

.....（续）

_builtin_divmodsd

汇编操作符/机器指令：
`divmodsd`

错误消息：
无。

_builtin_divmodud

说明：
发出 16 位架构的本机无符号除法支持。值得注意的是，如果商无法装入 16 位结果中，结果（包括余数）将是非预期的。这种形式的内建函数会同时捕捉商和余数。

原型：

```
unsigned int  _builtin_divmodud(
unsigned long dividend, unsigned int divisor,
unsigned int  *remainder);
```

参数：
dividend 被除数。

divisor 除数。

remainder 指向余数的指针。

返回值：
商和余数。

汇编操作符/机器指令：
`divmodud`

错误消息：
无。

_builtin_divsd

说明：
计算商：*num*/*den*。如果 *den* 为零，会发生数学错误异常。函数的参数和结果都是有符号的。可以使用命令行选项 `-Wconversions` 来检测意外的符号转换。

原型：

```
int _builtin_divsd(const long num, const int den);
```

参数：
num 分子。

den 分母。

返回值：
返回商的有符号整型值：*num*/*den*。

汇编操作符/机器指令：
`div.sd`

_builtin_divud

说明：
计算商：*num*/*den*。如果 *den* 为零，会发生数学错误异常。函数的参数和结果都是无符号的。可以使用命令行选项 `-Wconversions` 来检测意外的符号转换。

原型：

```
unsigned int  _builtin_divud(const unsigned
long     num, const unsigned int den);
```

.....（续）

_builtin_divud

参数：
num 分子。

den 分母。

返回值：
返回商的无符号整型值：*num*/*den*。

汇编操作符/机器指令：

```
div.ud
```

_builtin_dmaoffset

说明：
获取 DMA 存储区中的某个符号的偏移量。

例如：

```
unsigned int         result;
char     buffer[256] _attribute_((space(dma)));
result = _builtin_dmaoffset(&buffer);
```

可能生成：

```
mov #dmaoffset(buffer), w0
```

原型：

```
unsigned int _builtin_dmaoffset(const void *p);
```

参数：
**p* 指向 DMA 地址值的指针。

返回值：
返回位于 DMA 存储区中的某个变量的偏移量。

汇编操作符/机器指令：

```
dmaoffset
```

错误消息：
如果参数不是全局符号的地址，将显示一条错误消息。

_builtin_ed

说明：
求平方值 `sqr`，并将它作为结果返回。此外，为之后的平方运算预取数据，方法是计算`**xptr - **yptr`，并将结果存储在 `*distance` 中。

`xincr` 和 `yincr` 可以为立即数：-6、-4、-2、0、2、4、6 或整型值。

例如：

```
register int result asm("A");
int      *xmemory, *ymemory;
int      distance;
result = _builtin_ed(distance,
                     &xmemory, 2,
                     &ymemory, 2,
                     &distance);
```

可能生成：

```
ed w4*w4, A, [w8]+=2, [W10]+=2, w4
```

.....（续）

_builtin_ed

原型：

```
int _builtin_ed(int sqr, int **xptr, int xincr,
int **yptr, int yincr, int *distance);
```

参数：

sqr 整型平方值。

xptr 指向 X 预取的指针的整型指针。

xincr X 预取的整型递增值。

yptr 指向 Y 预取的指针的整型指针。

yincr Y 预取的整型递增值。

distance 指向 *distance* 的整型指针。

参数 *xptr* 和 *yptr* 必须分别指向位于 X 数据存储区和 Y 数据存储区中的数组。

返回值：

将平方结果返回到累加器。

汇编操作符/机器指令：

```
ed
```

错误消息：

如果出现以下情况，将显示一条错误消息：

- 结果不是累加器寄存器
- *xptr* 为空
- *yptr* 为空
- *distance* 为空

_builtin_edac

说明：

求平方值 `sqr`，并与指定的累加器寄存器相加，将它作为结果返回。此外，为之后的平方运算预取数据，方法是计算`**xptr - **yptr`，并将结果存储在`*distance` 中。

`xincr` 和 `yincr` 可以为立即数：-6、-4、-2、0、2、4、6 或整型值。

例如：

```
register int result asm("A");
int      *xmemory, *ymemory;
int      distance;
result = _builtin_ed(result, distance,
                     &xmemory, 2,
                     &ymemory, 2,
                     &distance);
```

可能生成：

```
edac w4*w4, A, [w8]+=2, [W10]+=2, w4
```

原型：

```
int _builtin_edac(int Accum, int sqr,
int **xptr, int xincr, int **yptr, int yincr,
int *distance);
```

.....（续）

_builtin_edac

参数：

Accum 要相加的累加器。

sqr 整型平方值。

xptr 指向 X 预取的指针的整型指针。

xincr X 预取的整型递增值。

yptr 指向 Y 预取的指针的整型指针。

yincr Y 预取的整型递增值。

distance 指向 *distance* 的整型指针。

参数 *xptr* 和 *yptr* 必须分别指向位于 X 数据存储区和 Y 数据存储区中的数组。

返回值：

将平方结果返回到指定累加器。

汇编操作符/机器指令：

```
edac
```

错误消息：

如果出现以下情况，将显示一条错误消息：

- 结果不是累加器寄存器
- *Accum* 不是累加器寄存器
- *xptr* 为空
- *yptr* 为空
- *distance* 为空

_builtin_fbcl

说明：

在 *value* 中从左边开始查找第一个位变化。它对于定点数据的动态比例换算很有用。例如：

```
int result, value;
result = _builtin_fbcl(value);
```

可能生成：

```
fbcl w4, w5
```

原型：

```
int _builtin_fbcl(int value);
```

参数：

value 第一个有位变化的整型数。

返回值：

将移位后的加法结果返回到累加器。

汇编操作符/机器指令：

```
fbcl
```

错误消息：

如果结果不是累加器寄存器，则将显示一条错误消息。

_builtin_lac

说明：
将值移位 *shift*（介于-8 和 7 之间的立即数）位，并返回要存储到累加器寄存器中的值。例如：

```
register int result asm("A");
int      value;
result = _builtin_lac(value,3);
```

可能生成：

```
lac w4, #3, A
```

原型：

```
int _builtin_lac(int value, int shift);
```

参数：
value 要进行移位的整型数。
shift 表示移位量的立即数。

返回值：
将移位后的加法结果返回到累加器。

汇编操作符/机器指令：

```
lac
```

错误消息：
如果出现以下情况，将显示一条错误消息：

- 结果不是累加器寄存器
- 移位值不是处于范围内的立即数

_builtin_mac

说明：
计算 *a* x *b*，并与累加器相加；此外，为之后的 MAC 操作预取数据。
xptr 可以为空，表示不需要执行 X 预取，这种情况下 *xincr* 和 *xval* 的值会被忽略，但它们是必需的。
yptr 可以为空，表示不需要执行 Y 预取，这种情况下 *yincr* 和 *yval* 的值会被忽略，但它们是必需的。
xval 和 *yval* 指定 C 变量的地址，预取值将存储在上述变量中。
xincr 和 *yincr* 可以为立即数：-6、-4、-2、0、2、4、6 或整型值。
如果 *AWB* 非空，则另一个累加器会被回写到所引用的变量中。
例如：

```
register int result asm("A");
register int B asm("B");
int      *xmemory;
int      *ymemory;
int      xVal, yVal;
result = _builtin_mac(result, xVal, yVal,
                      &xmemory, &xVal, 2,
                      &ymemory, &yVal, 2, 0, B);
```

可能生成：

```
mac w4*w5, A, [w8]+=2, w4, [w10]+=2, w5
```

原型：

```
int _builtin_mac(int Accum, int a, int b,
int **xptr, int *xval, int xincr,
int **yptr, int *yval, int yincr, int *AWB,
int AWB_accum);
```

.....（续）

_builtin_mac

参数：

Accum 要相加的累加器。

a 整型被乘数。

b 整型乘数。

xptr 指向 X 预取的指针的整型指针。

xval 指向 X 预取的值的整型指针。

xincr X 预取的整型递增值。

yptr 指向 Y 预取的指针的整型指针。

yval 指向 Y 预取的值的整型指针。

yincr Y 预取的整型递增值。

AWB 累加器回写目标。

AWB_accum 要被回写的累加器。

参数 *xptr* 和 *yptr* 必须分别指向位于 X 数据存储区和 Y 数据存储区中的数组。

返回值：

将清零值结果返回到累加器。

汇编操作符/机器指令：

```
mac
```

错误消息：

如果出现以下情况，将显示一条错误消息：

- 结果不是累加器寄存器
- *Accum* 不是累加器寄存器
- *xval* 是一个空值，但 *xptr* 不为空
- *yval* 是一个空值，但 *yptr* 不为空
- *AWB_accum* 不是累加器寄存器，并且 *AWB* 不为空

_builtin_modsd

说明：

发出 16 位架构的本机有符号除法支持。值得注意的是，如果商无法装入 16 位结果中，结果（包括余数）将是非预期的。这种形式的内建函数只会捕捉余数。

原型：

```
signed int _builtin_modsd(signed long dividend,
signed int divisor);
```

参数：

dividend 被除数。

divisor 除数。

返回值：

余数。

汇编操作符/机器指令：

```
modsd
```

错误消息：

无。

_builtin_modud

说明：
发出 16 位架构的本机无符号除法支持。值得注意的是，如果商无法装入 16 位结果中，结果（包括余数）将是非预期的。这种形式的内建函数只会捕捉余数。

原型：

```
unsigned int _builtin_modud(unsigned long dividend,
unsigned int divisor);
```

参数：
dividend 被除数。

divisor 除数。

返回值：
余数。

汇编操作符/机器指令：

```
modud
```

错误消息：
无。

_builtin_mpy

说明：
计算 *a* x *b*；此外，为之后的 MAC 操作预取数据。

xptr 可以为空，表示不需要执行 X 预取，这种情况下 *xincr* 和 *xval* 的值会被忽略，但它们是必需的。

yptr 可以为空，表示不需要执行 Y 预取，这种情况下 *yincr* 和 *yval* 的值会被忽略，但它们是必需的。

xval 和 *yval* 指定 C 变量的地址，预取值将存储在上述变量中。

xincr 和 *yincr* 可以为立即数：-6、-4、-2、0、2、4、6 或整型值。

例如：

```
register int result asm("A");
int      *xmemory;
int      *ymemory;
int      xVal, yVal;
result = _builtin_mpy(xVal, yVal,
                      &xmemory, &xVal, 2,
                      &ymemory, &yVal, 2);
```

可能生成：

```
mac w4*w5, A, [w8]+=2, w4, [w10]+=2, w5
```

原型：

```
int _builtin_mpy(int a, int b,
int **xptr, int *xval, int xincr,
int **yptr, int *yval, int yincr);
```

参数：
a 整型被乘数。

b 整型乘数。

xptr 指向 X 预取的指针的整型指针。

xval 指向 X 预取的值的整型指针。

xincr X 预取的整型递增值。

yptr 指向 Y 预取的指针的整型指针。

yval 指向 Y 预取的值的整型指针。

yincr Y 预取的整型递增值。

AWB 指向所选累加器的整型指针。

.....（续）

_builtin_mpy

参数 *xptr* 和 *yptr* 必须分别指向位于 X 数据存储区和 Y 数据存储区中的数组。

返回值：
将清零值结果返回到累加器。

汇编操作符/机器指令：

```
mpy
```

错误消息：
如果出现以下情况，将显示一条错误消息：

- 结果不是累加器寄存器
- *xval* 是一个空值，但 *xptr* 不为空
- *yval* 是一个空值，但 *yptr* 不为空

_builtin_mpyn

说明：
计算-*a* x *b*；此外，为之后的 MAC 操作预取数据。

xptr 可以为空，表示不需要执行 X 预取，这种情况下 *xincr* 和 *xval* 的值会被忽略，但它们是必需的。

yptr 可以为空，表示不需要执行 Y 预取，这种情况下 *yincr* 和 *yval* 的值会被忽略，但它们是必需的。

xval 和 *yval* 指定 C 变量的地址，预取值将存储在上述变量中。

xincr 和 *yincr* 可以为立即数：-6、-4、-2、0、2、4、6 或整型值。

例如：

```
register int result asm("A");
int      *xmemory;
int      *ymemory;
int      xVal, yVal;
result = _builtin_mpy(xVal, yVal,
                      &xmemory, &xVal, 2,
                      &ymemory, &yVal, 2);
```

可能生成：

```
mac w4*w5, A, [w8]+=2, w4, [w10]+=2, w5
```

原型：

```
int _builtin_mpyn(int a, int b,
int **xptr, int *xval, int xincr,
int **yptr, int *yval, int yincr);
```

参数：
a 整型被乘数。

b 整型乘数。

xptr 指向 X 预取的指针的整型指针。

xval 指向 X 预取的值的整型指针。

xincr X 预取的整型递增值。

yptr 指向 Y 预取的指针的整型指针。

yval 指向 Y 预取的值的整型指针。

yincr Y 预取的整型递增值。

AWB 指向所选累加器的整型指针。

参数 *xptr* 和 *yptr* 必须分别指向位于 X 数据存储区和 Y 数据存储区中的数组。

返回值：
将清零值结果返回到累加器。

.....（续）

_builtin_mpyn

汇编操作符/机器指令：

```
mpyn
```

错误消息：

如果出现以下情况，将显示一条错误消息：

- 结果不是累加器寄存器
- *xval* 是一个空值，但 *xptr* 不为空
- *yval* 是一个空值，但 *yptr* 不为空

_builtin_msc

说明：

计算 *a* x *b*，并从累加器中减去；此外，为之后的 MAC 操作预取数据。

xptr 可以为空，表示不需要执行 X 预取，这种情况下 *xincr* 和 *xval* 的值会被忽略，但它们是必需的。

yptr 可以为空，表示不需要执行 Y 预取，这种情况下 *yincr* 和 *yval* 的值会被忽略，但它们是必需的。

xval 和 *yval* 指定 C 变量的地址，预取值将存储在上述变量中。

xincr 和 *yincr* 可以为立即数：-6、-4、-2、0、2、4、6 或整型值。

如果 *AWB* 非空，则另一个累加器会被回写到所引用的变量中。

例如：

```
register int result asm("A");
int      *xmemory;
int      *ymemory;
int      xVal, yVal;
result = _builtin_msc(result, xVal, yVal,
                      &xmemory, &xVal, 2,
                      &ymemory, &yVal, 2, 0, 0);
```

可能生成：

```
msc w4*w5, A, [w8]+=2, w4, [w10]+=2, w5
```

原型：

```
int _builtin_msc(int Accum, int a, int b,
int **xptr, int *xval, int xincr,
int **yptr, int *yval, int yincr, int *AWB,
int AWB_accum);
```

参数：

Accum 要相加的累加器。

a 整型被乘数。

b 整型乘数。

xptr 指向 X 预取的指针的整型指针。

xval 指向 X 预取的值的整型指针。

xincr X 预取的整型递增值。

yptr 指向 Y 预取的指针的整型指针。

yval 指向 Y 预取的值的整型指针。

yincr Y 预取的整型递增值。

AWB 累加器回写目标。

AWB_accum 要被回写的累加器。

参数 *xptr* 和 *yptr* 必须分别指向位于 X 数据存储区和 Y 数据存储区中的数组。

返回值：

将清零值结果返回到累加器。

.....（续）

_builtin_msc

汇编操作符/机器指令：

```
msc
```

错误消息：

如果出现以下情况，将显示一条错误消息：

- 结果不是累加器寄存器
- *Accum* 不是累加器寄存器
- *xval* 是一个空值，但 *xptr* 不为空
- *yval* 是一个空值，但 *yptr* 不为空
- *AWB_accum* 不是累加器寄存器，并且 *AWB* 不为空

_builtin_mulss

说明：

计算乘积 *p0* x *p1*。函数参数为有符号整型，函数结果为有符号长整型。可以使用命令行选项-Wconversions 来检测意外的符号转换。

原型：

```
signed long _builtin_mulss(const signed int p0, const signed int p1);
```

参数：

p0 被乘数。

p1 乘数。

返回值：

返回乘积 *p0* x *p1* 的有符号长整型值。

汇编操作符/机器指令：

```
mul.ss
```

_builtin_mulsu

说明：

计算乘积 *p0* x *p1*。函数参数为混合符号整型，函数结果为有符号长整型。可以使用命令行选项-Wconversions 来检测意外的符号转换。该函数支持指令的所有寻址模式，包括操作数 *p1* 的立即数寻址模式。

原型：

```
signed long _builtin_mulsu(const signed int p0, const unsigned int p1);
```

参数：

p0 被乘数。

p1 乘数。

返回值：

返回乘积 *p0* x *p1* 的有符号长整型值。

汇编操作符/机器指令：

```
mul.su
```

_builtin_mulus

说明：

计算乘积 *p0* x *p1*。函数参数为混合符号整型，函数结果为有符号长整型。可以使用命令行选项-Wconversions 来检测意外的符号转换。该函数支持指令的所有寻址模式。

.....（续）

_builtin_mulus

原型：

```
signed long _builtin_mulus(const unsigned int p0, const signed int p1);
```

参数：
p0 被乘数。
p1 乘数。

返回值：
返回乘积 *p0* x *p1* 的有符号长整型值。

汇编操作符/机器指令：

```
mul.us
```

_builtin_muluu

说明：
计算乘积 *p0* x *p1*。函数参数为无符号整型，函数结果为无符号长整型。可以使用命令行选项-Wconversions 来检测意外的符号转换。该函数支持指令的所有寻址模式，包括操作数 *p1* 的立即数寻址模式。

原型：

```
unsigned long _builtin_muluu(const unsigned int p0, const unsigned int p1);
```

参数：
p0 被乘数。
p1 乘数。

返回值：
返回乘积 *p0* x *p1* 的有符号长整型值。

汇编操作符/机器指令：

```
mul.uu
```

_builtin_nop

说明：
生成一条 NOP 指令。

原型：

```
void _builtin_nop(void);
```

参数：
无。

返回值：
返回无操作（NOP）。

汇编操作符/机器指令：

```
NOP
```

_builtin_readsfr

说明：
读取 SFR。

原型：

```
unsigned int _builtin_readsfr(const void *p);
```

.....（续）

_builtin_readsfr

参数：
p 对象的地址。

返回值：
返回 SFR。

汇编操作符/机器指令：

```
readsfr
```

_builtin_return_address

说明：
返回当前函数或其调用函数之一的返回地址。对于 *level* 参数，值为 0 将产生当前函数的返回地址，值为 1 将产生当前函数的调用函数的返回地址，以此类推。当 *level* 超出当前堆栈深度时，将返回 0。该函数应仅使用非零参数，用于调试目的。

原型：

```
int _builtin_return_address (const int level);
```

参数：
level 要在调用堆栈中扫描的帧数。

返回值：
返回当前函数或其调用函数之一的返回地址。

汇编操作符/机器指令：

```
return_address
```

_builtin_sac

说明：
将值移位 *shift*（介于-8 和 7 之间的立即数）位，并返回值。

例如：

```
register int value asm("A");
int      result;
result = _builtin_sac(value,3);
```

可能生成：

```
sac A, #3, w0
```

原型：

```
int _builtin_sac(int value, int shift);
```

参数：
value 要进行移位的整型数。

shift 表示移位位数的立即数。

返回值：
将移位后的结果返回到累加器。

汇编操作符/机器指令：

```
sac
```

错误消息：
如果出现以下情况，将显示一条错误消息：

- 结果不是累加器寄存器
- 移位值不是处于范围内的立即数

_builtin_sacr

说明：
将值移位 *shift*（介于-8 和 7 之间的立即数）位，并返回使用由 RND（CORCON<1>）控制位决定的舍入模式进行舍入后的值。

例如：

```
register int value asm("A");
int      result;
result = _builtin_sac(value,3);
```

可能生成：

```
sac.r A, #3, w0
```

原型：

```
int _builtin_sacr(int value, int shift);
```

参数：
value 要进行移位的整型数。

shift 表示移位位数的立即数。

返回值：
将移位后的结果返回到 CORCON 寄存器。

汇编操作符/机器指令：

```
sacr
```

错误消息：
如果出现以下情况，将显示一条错误消息：

- 结果不是累加器寄存器
- 移位值不是处于范围内的立即数

_builtin_sftac

说明：
将累加器移位 *shift* 位。有效移位范围为-16 至 16。

例如：

```
register int result asm("A");
int      i;
result = _builtin_sftac(result,i);
```

可能生成：

```
sftac A, w0
```

原型：

```
int _builtin_sftac(int Accum, int shift);
```

参数：
Accum 要移位的累加器。

shift 要移位的位数。

返回值：
将移位后的结果返回到累加器。

汇编操作符/机器指令：

```
sftac
```

.....（续）

_builtin_sftac

错误消息：
如果出现以下情况，将显示一条错误消息：

- 结果不是累加器寄存器
- *Accum* 不是累加器寄存器
- 移位值不是处于范围内的立即数

_builtin_subab

说明：
将累加器 A 和 B 相减，并将结果回写到指定的累加器。例如：

```
register int result asm("A");
register int B asm("B");
result = _builtin_subab(result,B);
```

将生成：

```
sub A
```

原型：

```
int _builtin_subab(int Accum_a, int Accum_b);
```

参数：
Accum_a 作为被减数的累加器。

Accum_b 作为减数的累加器。

返回值：
将减法结果返回到累加器。

汇编操作符/机器指令：

```
sub
```

错误消息：
如果结果不是累加器寄存器，则将显示一条错误消息。

6. 参考信息

6.1 指令位映射

图 6-1 中对 16 位 MCU 和 DSC 系列器件的指令编码进行了汇总。表中包括每一条指令最高字节的编码。表中第一列表示操作码的 bit [23:20]，而表中第一行表示操作码的 bit [19:16]。操作码的第一个字节由第一行给出的位值附加第一列的位值而形成。例如，PUSH 指令的 MSB（最后一行，第九列）的编码为 11111000b（0xF8）。

注：可根据指令说明中的指令说明确定每一条指令的完整操作码。

图 6-1. 指令编码

Col Ref → / Row Ref ↓	Opcode Size	Opcode Extension	Opcode LSbits → MSbits ↓	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	10 1010	11 1011	12 1100	13 1101	14 1110	15 1111
a	16/32-bit	00	00	NEOP NOP (16-bit only)	MOV MOVR (W→W) EXCH (W⟷W)	GP Co-processor Move Ops (16-bit) MOV MOVKCR (W → F)	MOV MOVKRW (F → W)	MOV MOVWS W→(W15-lit7) W→(W14-slit7)		MOV MOVSW (W15-lit7) → W (W14-slit7) → W		MOVLN lit5		MULSS MULUS	MULUU MULSU			POPW (16-bit only)	PUSHW (16-bit only)
b			01	ASRK	LSRK	SLK	PUSH PUSHCR	ASR ASRW	LSR LSRW	SL SLW	POP POPCR	RLC	RLNC	RRC	RRNC	CPLS	CPBLS	CP	CPB
c			10	BCLR	BSET	BTG	BTST	BTSTS	BSW	BTSTW									
d			11	ADD	ADDC	SUB	SUBB	AND	IOR	SUBR	SUBBR	XOR	Inherent	ADDLN lit5	SUBLN lit5	CDM	SUBRLS	SE	ZE
e	32-bit	01	00	MOV.l MOVLL	GP Co-processor Move Ops (32-bit) MOV.l MOVLCR	MOV.l LDWLCOR	MOV.l STWLCOR	MOV.b/.w/.l literal MOVS				MOV.w/.l LDW.w/l		MOV.w/.l STW.w/l		MOV.bz LDW.bz		MOV.bz STW.bz	
f			01	ASRF	LSRF	SLF	CLRF	CPF0	CPF	CPFB	SETF	RLCF	RLNCF	RRCF	RRNCF	DBT	MULWF	POPF	PUSHF
g			10	BCLRF	BSETF	BTGF	BTSTF	BTSTSF	BFINSL	BFINSF BFEXTF			TSTF (formerly MOVF)	INCF	DECF	COMF	NEGF	INCF2	DECF2
h			11	ADDWF	ADDCWF	SUBFW	SUBBFW	ANDWF	IORWF	SUBWF	SUBBWF	XORWF							
i		10	00	Reserved for Co-processor Ops FADD FSUB	FMAX FMAX2 FMIN FMIN2	FMAC FMUL FDIV	FMOV FMOVC	FABS FNEG FSQRT	FCPS FCPQ	FSIN FCOS	FF2LI FF2DI FLI2F FDI2F	FAND FIOR	FFLIM	FTST	Reserved	Reserved	Reserved	Reserved	Reserved
j			01				BOA BOB BSA BSB	BGT BGE BLT BLE	BC BNC BOV BNOV	BN BNN BZ BNZ	BLEU BGTU BRA	Reserved for Co-processor Branches CBRA0 CBRA1 FBRA EQ FBRA UNE	CBRA2 CBRA3 FBRA NE FBRA UEQ	CBRA4 CBRA5 FBRA GT FBRA ULE	CBRA6 CBRA7 FBRA GE FBRA ULT	CBRA8 CBRA9 FBRA LT FBRA UGE	CBRA10 CBRA11 FBRA LE FBRA UGT	CBRA12 CBRA13 FBRA OR FBRA UN	CBRA14 CBRA15 Reserved
k			10	ADDLW	ADDCLW	SUBLW CPLW (formerly CPLS)	SUBBLW CPBLW (formerly CPBLS)	ANDLW	IORLW	RETLW		XORLW		RCALL REPEAT LNK	BRAW RCALLW GOTOW CALLW	REPEATW CTXTSWPW	GOTO lit24	DISICTL DISICTLW	CALL lit24
l			11	ADDLS	ADDCLS	SUBLS	SUBBLS	ANDLS	IORLS	ANDLSL	SUBBRLS	XORLS	LDSLV VFSLV	MULSLS slit8	MULULS lit8	MULASS/US MULAxx	MULAUU/SU MULAxx	MULASLS slit8	MULAULS lit8
m		11	00	MOVL.bz/.w/.sl literal															
n			01																
o			10	LAC, LLAC ADDAC, SUBAC		SAC, SLAC SRAC, SUAC		MINABW, SRACW MAXABW, NORMACW		SFTAC, SFTACK LLIAC		MAC/SQRAC MSC/SQRSC		MPY/SQR MPYN/SQRN		EDAC ED			
p			11	ASRMK LSRMK SLMK	ASRMW LSRMW SLMW	FLIM FLIMW MIN MAX	FF1L FF1R FBCL	DIVS DIVU DIVF	DIVSL DIVUL DIVFL			LDWE	STWE	GOTOE CALLE	MOVIF SWAP	BFINS BFEXT	RESET BOOTSWAP	MOV LDWLO STWLO	NOPR (32-bit)

6.2 指令集汇总表

表 6-1 对 16 位 MCU 和 DSC 器件的整个指令集进行了汇总。表中给出了按字母顺序排列的指令集列表。表中包括指令汇编语法、说明、指令长度（以 24 位字为单位）、执行时间（以指令周期为单位）、受影响的状态位以及指令详细说明所处的页码。 表 6-2 说明了指令集汇总表中使用的符号。

表 6-1. 指令集汇总表

汇编语法 助记符，操作数		说明	指令字数	指令周期数	OA	OB	SA	SB(1)	OAB	SAB(1)	DC	N	OV	Z	C
ADD	f {,WREG}	目标寄存器 = f + WREG	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
ADD	#lit10,Wn	Wn = lit10 + Wn	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
ADD	Wb,#lit5,Wd	Wd = Wb + lit5	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
ADD	Wb,Ws,Wd	Wd = Wb + Ws	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
ADD	Acc	累加器相加	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
ADD	Ws,#Slit4,Acc	16 位有符号数加到累加器	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
ADDC	f {,WREG}	目标寄存器 = f + WREG + (C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
ADDC	#lit10,Wn	Wn = lit10 + Wn + (C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
ADDC	Wb,#lit5,Wd	Wd = Wb + lit5 + (C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
ADDC	Wb,Ws,Wd	Wd = Wb + Ws + (C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
AND	f {,WREG}	目标寄存器 = f .AND.WREG	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
AND	#lit10,Wn	Wn = lit10 .AND.Wn	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
AND	Wb,#lit5,Wd	Wd = Wb .AND. lit5	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
AND	Wb,Ws,Wd	Wd = Wb .AND.Ws	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
ASR	f {,WREG}	目标寄存器 = 算术右移 f，LSb → C	1	1	—	—	—	—	—	—	—	⇕	—	⇕	⇕
ASR	Ws,Wd	Wd = 算术右移 Ws，LSb → C	1	1	—	—	—	—	—	—	—	⇕	—	⇕	⇕
ASR	Wb,#lit4,Wnd	Wnd = Wb 算术右移 lit4 位，LSb → C	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
ASR	Wb,Wns,Wnd	Wnd = Wb 算术右移 Wns 位，LSb → C	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
BCLR	f,#bit4	将 f 中的指定位清零	1	1	—	—	—	—	—	—	—	—	—	—	—
BCLR	Ws,#bit4	将 Ws 中的指定位清零	1	1	—	—	—	—	—	—	—	—	—	—	—
BFEXT	#bit4,#wid5,Ws,Wb	从 Ws 中提取位域，写入 Wb	2	2	—	—	—	—	—	—	—	—	—	—	—
BFEXT	#bit4,#wid5,f,Wb	从 f 中提取位域，写入 Wb	2	2	—	—	—	—	—	—	—	—	—	—	—
BFINS	#bit4,#wid5,Wb,Ws	将 Wb 中的位域插入 Ws	2	2	—	—	—	—	—	—	—	—	—	—	—
BFINS	#bit4,#wid5,Wb,f	将 Wb 中的位域插入 f	2	2	—	—	—	—	—	—	—	—	—	—	—
BFINS	#bit4,#wid5,#lit8,Ws	将#lit8 中的位域插入 Ws	2	2	—	—	—	—	—	—	—	—	—	—	—
BOOTSWP		交换活动和非活动闪存程序空间	1	2	—	—	—	—	—	—	—	—	—	⇕	—
BRA	Expr	无条件转移	1	2	—	—	—	—	—	—	—	—	—	—	—
BRA	Wn	计算转移	1	2	—	—	—	—	—	—	—	—	—	—	—

图注： ⇕ 置 1 或清零；⇓ *可以*被清零，但永远不会被置 1；⇑ *可以*被置 1，但永远不会被清零；“1”总是置 1；“0”总是清零；—不变

注：

1. 只有在相应的饱和功能使能时，SA、SB 和 SAB 才可被修改，否则将不会发生改变。

.....（续）

汇编语法 助记符，操作数 .		说明	指令字数	指令周期数	OA	OB	SA	SB(1)	OAB	SAB(1)	DC	N	OV	Z	C
BRA	Wn	计算转移	1	2	—	—	—	—	—	—	—	—	—	—	—
BRA C	Expr	如果进位位为 1 则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA GE	Expr	如果有符号大于或等于则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA GEU	Expr	如果无符号大于或等于则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA GT	Expr	如果有符号大于则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA GTU	Expr	如果无符号大于则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA LE	Expr	如果有符号小于或等于则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA LEU	Expr	如果无符号小于或等于则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA LT	Expr	如果有符号小于则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA LTU	Expr	如果无符号小于则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA N	Expr	如果为负则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA NC	Expr	如果进位位为零则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA NN	Expr	如果非负则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA NOV	Expr	如果未溢出则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA NZ	Expr	如果非零则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA OA	Expr	如果累加器 A 溢出则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA OB	Expr	如果累加器 B 溢出则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—

图注： ⇳ 置 1 或清零；⇓ *可以*被清零，但永远不会被置 1；⇑ *可以*被置 1，但永远不会被清零；“1”总是置 1；“0”总是清零；—不变

注：

1. 只有在相应的饱和功能使能时，SA、SB 和 SAB 才可被修改，否则将不会发生改变。

.....（续）

汇编语法 助记符，操作数 .		说明	指令字数	指令周期数	OA	OB	SA	SB(1)	OAB	SAB(1)	DC	N	OV	Z	C
BRA OV	Expr	如果溢出则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA SA	Expr	如果累加器 A 饱和则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA SB	Expr	如果累加器 B 饱和则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BRA Z	Expr	如果为零则转移	1	1（2）	—	—	—	—	—	—	—	—	—	—	—
BSET	f,#bit4	将 f 中的指定位置 1	1	1	—	—	—	—	—	—	—	—	—	—	—
BSET	Ws,#bit4	将 Ws 中的指定位置 1	1	1	—	—	—	—	—	—	—	—	—	—	—
BSW	Ws,Wb	写 Ws<Wb>中的某位	1	1	—	—	—	—	—	—	—	—	—	—	—
BTG	f,#bit4	将 f 中的指定位翻转	1	1	—	—	—	—	—	—	—	—	—	—	—
BTG	Ws,#bit4	将 Ws 中的指定位翻转	1	1	—	—	—	—	—	—	—	—	—	—	—
BTSC	f,#bit4	测试 f 中的指定位，为零则跳过	1	1（2 或 3）	—	—	—	—	—	—	—	—	—	—	—
BTSC	Ws,#bit4	测试 Ws 中的指定位，为零则跳过	1	1（2 或 3）	—	—	—	—	—	—	—	—	—	—	—
BTSS	f,#bit4	测试 f 中的指定位，为 1 则跳过	1	1（2 或 3）	—	—	—	—	—	—	—	—	—	—	—
BTSS	Ws,#bit4	测试 Ws 中的指定位，为 1 则跳过	1	1（2 或 3）	—	—	—	—	—	—	—	—	—	—	—
BTST	f,#bit4	测试 f 中的指定位	1	1	—	—	—	—	—	—	—	—	—	⇕	—
BTST	Ws,#bit4	测试 Ws 中的指定位	1	1	—	—	—	—	—	—	—	—	—	—	⇕
BTST	Ws,Wb	测试 Ws 中的指定位	1	1	—	—	—	—	—	—	—	—	—	⇕	—
BTSTS	f,#bit4	测试/置 1 f 中的指定位	1	1	—	—	—	—	—	—	—	—	—	⇕	—
BTSTS	Ws,#bit4	测试/置 1 Ws 中的指定位	1	1	—	—	—	—	—	—	—	—	—	—	⇕
CALL	Expr	调用子程序	2	2	—	—	—	—	—	—	—	—	—	—	—

图注：⇕ 置 1 或清零；⇓ *可以*被清零，但永远不会被置 1；⇑ *可以*被置 1，但永远不会被清零；“1”总是置 1；“0”总是清零；—不变

注：

1. 只有在相应的饱和功能使能时，SA、SB 和 SAB 才可被修改，否则将不会发生改变。

.....（续）

汇编语法 助记符，操作数 .		说明	指令字数	指令周期数	OA	OB	SA	SB(1)	OAB	SAB(1)	DC	N	OV	Z	C
CALL	Expr	调用子程序	2	2	—	—	—	—	—	—	—	—	—	—	—
CALL	Wn	间接调用子程序	1	2	—	—	—	—	—	—	—	—	—	—	—
CALL	Wn	间接调用子程序	1	2	—	—	—	—	—	—	—	—	—	—	—
CALL.L	Wn	间接长调用子程序（长地址）	1	4	—	—	—	—	—	—	—	—	—	—	—
CLR	f,WREG	清零 f 或 WREG	1	1	—	—	—	—	—	—	—	—	—	—	—
CLR	Wd	清零 Wd	1	1	—	—	—	—	—	—	—	—	—	—	—
CLR	Acc,[Wx],Wxd,[Wy],Wyd,AWB	清零累加器	1	1	0	0	0	0	0	0	—	—	—	—	—
CLRWDT		清零看门狗定时器	1	1	—	—	—	—	—	—	—	—	—	—	—
COM	f {,WREG}	目标寄存器 = $\overline{f}$	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
COM	Ws,Wd	Wd = $\overline{Ws}$	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
CP	f	比较(f – WREG)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
CP	Wb,#lit5	比较(Wb – lit5)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
CP	Wb,#lit8	比较(Wb – lit8)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
CP	Wb,Ws	比较(Wb – Ws)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
CP0	f	比较(f – 0x0)	1	1	—	—	—	—	—	—	1	⇕	⇕	⇕	1
CP0	Ws	比较(Ws – 0x0)	1	1	—	—	—	—	—	—	1	⇕	⇕	⇕	1
CPB	f	带借位比较(f – WREG – C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
CPB	Wb,#lit5	带借位比较(Wb – lit5 – C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
CPB	Wb,#lit8	带借位比较(Wb – lit8 – C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
CPB	Wb,Ws	带借位比较(Wb – Ws – C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
CPBEQ	Wb,Wn,Expr	比较 Wb 和 Wn，如果相等则转移	1	1（5）	—	—	—	—	—	—	—	—	—	—	—
CPBGT	Wb,Wn,Expr	有符号比较 Wb 和 Wn，如果大于则转移	1	1（5）	—	—	—	—	—	—	—	—	—	—	—
CPBLT	Wb,Wn,Expr	有符号比较 Wb 和 Wn，如果小于则转移	1	1（5）	—	—	—	—	—	—	—	—	—	—	—
CPBNE	Wb,Wn,Expr	比较 Wb 和 Wn，如果不相等则转移	1	1（5）	—	—	—	—	—	—	—	—	—	—	—

图注： ⇕ 置 1 或清零；⇓ *可以*被清零，但永远不会被置 1；⇑ *可以*被置 1，但永远不会被清零；“1”总是置 1；“0”总是清零；—不变

注：

1. 只有在相应的饱和功能使能时，SA、SB 和 SAB 才可被修改，否则将不会发生改变。

.....（续）

汇编语法 助记符，操作数 .		说明	指令字数	指令周期数	OA	OB	SA	SB(1)	OAB	SAB(1)	DC	N	OV	Z	C
CPSEQ	Wb,Wn	比较 Wb 和 Wn，如果相等则跳过	1	1（2 或 3）	—	—	—	—	—	—	—	—	—	—	—
CPSEQ	Wb,Wn	比较 Wb 和 Wn，如果相等则跳过	1	1（2 或 3）	—	—	—	—	—	—	—	—	—	—	—
CPSGT	Wb,Wn	有符号比较 Wb 和 Wn，如果大于则跳过	1	1（2 或 3）	—	—	—	—	—	—	—	—	—	—	—
CPSGT	Wb,Wn	有符号比较 Wb 和 Wn，如果大于则跳过	1	1（2 或 3）	—	—	—	—	—	—	—	—	—	—	—
CPSLT	Wb,Wn	有符号比较 Wb 和 Wn，如果小于则跳过	1	1（2 或 3）	—	—	—	—	—	—	—	—	—	—	—
CPSLT	Wb,Wn	有符号比较 Wb 和 Wn，如果小于则跳过	1	1（2 或 3）	—	—	—	—	—	—	—	—	—	—	—
CPSNE	Wb,Wn	有符号比较 Wb 和 Wn，如果不相等则跳过	1	1（2 或 3）	—	—	—	—	—	—	—	—	—	—	—
CPSNE	Wb,Wn	有符号比较 Wb 和 Wn，如果不相等则跳过	1	1（2 或 3）	—	—	—	—	—	—	—	—	—	—	—
CTXTSWP	#lit3	CPU 寄存器现场交换立即数	1	2	—	—	—	—	—	—	—	—	—	—	—
CTXTSWP	Wn	CPU 寄存器现场交换 Wn	1	2	—	—	—	—	—	—	—	—	—	—	—
DAW.B	Wn	Wn = 对 Wn 的内容进行十进制调整	1	1	—	—	—	—	—	—	—	—	—	—	⇕
DEC	f {,WREG}	目标寄存器 = f - 1	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
DEC	Ws,Wd	Wd = Ws – 1	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
DEC2	f {,WREG}	目标寄存器 = f - 2	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
DEC2	Ws,Wd	Wd = Ws – 2	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕

图注： ⇕ 置 1 或清零；⇓ *可以*被清零，但永远不会被置 1；⇑ *可以*被置 1，但永远不会被清零；“1”总是置 1；“0”总是清零；—不变

注：

1. 只有在相应的饱和功能使能时，SA、SB 和 SAB 才可被修改，否则将不会发生改变。

.....（续）

汇编语法 助记符，操作数 .		说明	指令字数	指令周期数	OA	OB	SA	SB(1)	OAB	SAB(1)	DC	N	OV	Z	C
DISI	#lit14	禁止中断，持续时间为 lit14 个指令周期	1	1	—	—	—	—	—	—	—	—	—	—	—
DIV.S	Wm,Wn	有符号 16 位/16 位整数除法	1	18	—	—	—	—	—	—	—	⇕	⇕	⇕	⇕
DIV.U	Wm,Wn	无符号 16 位/16 位整数除法	1	18	—	—	—	—	—	—	—	0	0	⇕	⇕
DIVF	Wm,Wn	有符号 16 位/16 位小数除法	1	18	—	—	—	—	—	—	—	⇕	⇕	⇕	⇕
DIVF2	Wm,Wn	有符号 16 位/16 位小数除法（W1:W0 保留）	1	6	—	—	—	—	—	—	—	⇕	⇕	⇕	⇕
DIV2.S	Wm, Wn	有符号 16 位/16 位整数除法（W1:W0 保留）	1	6	—	—	—	—	—	—	—	⇕	⇕	⇕	⇕
DIV2.U	Wm,Wn	无符号 16 位/16 位整数除法（W1:W0 保留）	1	6	—	—	—	—	—	—	—	0	0	⇕	⇕
DO	#lit14,Expr	执行 DO 循环代码到 PC + Expr，执行次数为 (lit14 + 1)次	2	2	—	—	—	—	—	—	—	—	—	—	—
DO	#lit15,Expr	执行 DO 循环代码到 PC + Expr，执行次数为 (lit15 + 1)次	2	2	—	—	—	—	—	—	—	—	—	—	—
DO	Wn,Expr	执行 DO 循环代码到 PC + Expr，执行次数为 (Wn + 1)次	2	2	—	—	—	—	—	—	—	—	—	—	—
DO	Wn,Expr	执行 DO 循环代码到 PC + Expr，执行次数为 (Wn + 1)次	2	2	—	—	—	—	—	—	—	—	—	—	—
ED	Wm*Wm,Acc,[Wx],[Wy],Wxd	求取欧几里德距离（无累加）	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
EDAC	Wm*Wm,Acc,[Wx],[Wy],Wxd	求取欧几里德距离	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
EXCH	Wns,Wnd	交换 Wns 和 Wnd 的内容	1	1	—	—	—	—	—	—	—	—	—	—	—
FBCL	Ws,Wnd	搜索自左（MSb）起第一个位变化	1	1	—	—	—	—	—	—	—	—	—	—	⇕
FF1L	Ws,Wnd	搜索自左（MSb）起第一个 1	1	1	—	—	—	—	—	—	—	—	—	—	⇕
FF1R	Ws,Wnd	搜索自右（LSb）起第一个 1	1	1	—	—	—	—	—	—	—	—	—	—	⇕
FLIM	Wb,Ws	强制执行（有符号）数据范围限制	1	1	—	—	—	—	—	—	—	⇕	⇕	⇕	—
FLIM.V	Wb,Ws,Wnd	强制执行（有符号）数据范围限制，保存超限结果	1	1	—	—	—	—	—	—	—	⇕	⇕	⇕	—
GOTO	Expr	无条件跳转	2	2	—	—	—	—	—	—	—	—	—	—	—
GOTO	Wn	无条件间接跳转	1	2	—	—	—	—	—	—	—	—	—	—	—
GOTO	Wn	无条件间接跳转	1	2	—	—	—	—	—	—	—	—	—	—	—
GOTO.L	Wn	无条件间接长跳转	1	4	—	—	—	—	—	—	—	—	—	—	—
INC	f {,WREG}	目标寄存器 = f + 1	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕

图注：⇕ 置 1 或清零；⇓ *可以*被清零，但永远不会被置 1；⇑ *可以*被置 1，但永远不会被清零；“1”总是置 1；“0”总是清零；—不变

注：

1. 只有在相应的饱和功能使能时，SA、SB 和 SAB 才可被修改，否则将不会发生改变。

.....（续）

汇编语法 助记符，操作数 .		说明	指令字数	指令周期数	OA	OB	SA	SB(1)	OAB	SAB(1)	DC	N	OV	Z	C
INC	Ws,Wd	Wd = Ws + 1	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
INC2	f {,WREG}	目标寄存器 = f + 2	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
INC2	Ws,Wd	Wd = Ws + 2	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
IOR	f {,WREG}	目标寄存器 = f .IOR.WREG	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
IOR	#lit10,Wn	Wn = lit10 .IOR.Wn	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
IOR	Wb,#lit5,Wd	Wd = Wb .IOR. lit5	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
IOR	Wb,Ws,Wd	Wd = Wb .IOR.Ws	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
LAC	Ws,{#Slit4,},Acc	装载累加器	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
LAC.D	Ws,[,#Slit4],Acc	装载累加器（双字）	1	2	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
LDSLV	[Wns],[Wnd++],#lit2	将单个指令字从主 PRAM 传送至从 PRAM	1	2	—	—	—	—	—	—	—	—	—	—	—
LNK	#lit14	分配堆栈帧	1	1	—	—	—	—	—	—	—	—	—	—	—
LNK	#lit14	分配堆栈帧	1	1	—	—	—	—	—	—	—	—	—	—	—
LSR	f {,WREG}	目标寄存器 = 逻辑右移 f，MSb → C	1	1	—	—	—	—	—	—	—	0	—	⇕	⇕
LSR	Ws,Wd	Wd = 逻辑右移 Ws，MSb → C	1	1	—	—	—	—	—	—	—	0	—	⇕	⇕
LSR	Wb,#lit4,Wnd	Wnd = Wb 逻辑右移 lit4 位，MSb → C	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
LSR	Wb,Wns,Wnd	Wnd = Wb 逻辑右移 Wns 位，MSb → C	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
MAC	Wm*Wn,Acc,[Wx],Wxd,[Wy],Wyd,AWB	相乘并累加	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
MAC	Wm*Wm,Acc,[Wx],Wxd,[Wy],Wyd	平方并累加	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
MAX	Acc	强制执行累加器数据范围上限	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
MAX.V	Acc,Wd	强制执行累加器数据范围上限，保存超限结果	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
MIN	Acc	强制执行累加器数据范围下限	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
MIN.V	Acc,Wd	强制执行累加器数据范围下限，保存超限结果	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
MINZ	Acc	如果 Z 标志置 1，则强制执行累加器数据范围下限	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
MINZ.V	Acc,Wd	如果 Z 标志置 1，则强制执行累加器数据范围下限并保存超限结果	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
MOV	f {,WREG}	将 f 中内容传送至目标寄存器	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—

图注： ⇕ 置 1 或清零；⇓ *可以*被清零，但永远不会被置 1；⇑ *可以*被置 1，但永远不会被清零；“1”总是置 1；“0”总是清零；—不变

注：

1. 只有在相应的饱和功能使能时，SA、SB 和 SAB 才可被修改，否则将不会发生改变。

.....（续）

汇编语法 助记符，操作数 .		说明	指令字数	指令周期数	OA	OB	SA	SB(1)	OAB	SAB(1)	DC	N	OV	Z	C
MOV	WREG,f	将 WREG 中内容传送至 f	1	1	—	—	—	—	—	—	—	—	—	—	—
MOV	f,Wnd	将 f 中内容传送至 Wnd	1	1	—	—	—	—	—	—	—	—	—	—	—
MOV	Wns,f	将 Wns 中内容传送至 f	1	1	—	—	—	—	—	—	—	—	—	—	—
MOV.B	#lit8,Wnd	将 8 位无符号立即数传送至 Wnd	1	1	—	—	—	—	—	—	—	—	—	—	—
MOV	#lit16,Wnd	将 16 位立即数传送至 Wnd	1	1	—	—	—	—	—	—	—	—	—	—	—
MOV	[Ws+Slit10],Wnd	将[Ws + Slit10]中内容传送至 Wnd	1	1	—	—	—	—	—	—	—	—	—	—	—
MOV	Wns,[Wd+Slit10]	将 Wns 中内容传送至[Wd + Slit10]	1	1	—	—	—	—	—	—	—	—	—	—	—
MOV	Ws,Wd	将 Ws 中内容传送至 Wd	1	1	—	—	—	—	—	—	—	—	—	—	—
MOV.D	Wns,Wnd	双字传送 Wns:Wns + 1 内容至 Wnd	1	2	—	—	—	—	—	—	—	—	—	—	—
MOVPAG	#lit10,DSRPAG	将 10 位立即数传送至 DSRPAG	1	1	—	—	—	—	—	—	—	—	—	—	—
MOVPAG	Wn,DSRPAG	将 Wn 中内容传送至 DSRPAG	1	1	—	—	—	—	—	—	—	—	—	—	—
MOVSAC	Acc,[Wx],Wxd,[Wy],Wyd,AWB	将[Wx]中内容传送至 Wxd，并将[Wy]中内容传送至 Wyd	1	1	—	—	—	—	—	—	—	—	—	—	—
MPY	Wm*Wn,Acc,[Wx],Wxd,[Wy],Wyd	Wm 与 Wn 相乘，结果存入累加器	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
MPY	Wm*Wm,Acc,[Wx],Wxd,[Wy],Wyd	求平方，结果存入累加器	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
MPY.N	Wm*Wn,Acc,[Wx],Wxd,[Wy],Wyd	Wn 和 Wm 相乘并取反，结果存入累加器	1	1	⇑	⇑	—	—	⇑	—	—	—	—	—	—
MSC	Wm*Wn,Acc,[Wx],Wxd,[Wy],Wyd,AWB	相乘并从累加器中减去乘积	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
MUL	f	W3:W2 = f * WREG	1	1	—	—	—	—	—	—	—	—	—	—	—
MUL.SS	Wb,Ws,Wnd	{Wnd + 1, Wnd} = 有符号(Wb) * 有符号(Ws)	1	1	—	—	—	—	—	—	—	—	—	—	—
MUL.SS	Wb,Ws,Acc	累加器 = 有符号(Wb) * 有符号(Ws)	1	1	—	—	—	—	—	—	—	—	—	—	—
MUL.SU	Wb,#lit5,Wnd	{Wnd + 1, Wnd} = 有符号(Wb) * 无符号(lit5)	1	1	—	—	—	—	—	—	—	—	—	—	—
MUL.SU	Wb,Ws,Wnd	{Wnd + 1, Wnd} = 有符号(Wb) * 无符号(Ws)	1	1	—	—	—	—	—	—	—	—	—	—	—
MUL.SU	Wb,Ws,Acc	累加器 = 有符号(Wb) * 无符号(Ws)	1	1	—	—	—	—	—	—	—	—	—	—	—
MUL.SU	Wb,#lit5,Acc	累加器 = 有符号(Wb) * 无符号(lit5)	1	1	—	—	—	—	—	—	—	—	—	—	—

图注：⇕ 置 1 或清零；⇓ *可以*被清零，但永远不会被置 1；⇑ *可以*被置 1，但永远不会被清零；“1”总是置 1；“0”总是清零；—不变

注：

1. 只有在相应的饱和功能使能时，SA、SB 和 SAB 才可被修改，否则将不会发生改变。

.....（续）

汇编语法 助记符，操作数 .		说明	指令字数	指令周期数	OA	OB	SA	SB(1)	OAB	SAB(1)	DC	N	OV	Z	C
MUL.US	Wb,Ws,Wnd	{Wnd + 1, Wnd} = 无符号(Wb) * 有符号(Ws)	1	1	—	—	—	—	—	—	—	—	—	—	—
MUL.US	Wb,Ws,Acc	累加器 = 无符号(Wb) * 有符号(Ws)	1	1	—	—	—	—	—	—	—	—	—	—	—
MUL.UU	Wb,#lit5,Wnd	{Wnd + 1, Wnd} = 无符号(Wb) * 无符号(lit5)	1	1	—	—	—	—	—	—	—	—	—	—	—
MUL.UU	Wb,Ws,Wnd	{Wnd + 1, Wnd} = 无符号(Wb) * 无符号(Ws)	1	1	—	—	—	—	—	—	—	—	—	—	—
MUL.UU	Wb,Ws,Acc	累加器 = 无符号(Wb) * 无符号(Ws)	1	1	—	—	—	—	—	—	—	—	—	—	—
MUL.UU	Wb,#lit5,Acc	累加器 = 无符号(Wb) * 无符号(lit5)	1	1	—	—	—	—	—	—	—	—	—	—	—
MULW.SS	Wb,Ws,Wnd	Wnd = 有符号(Wb) * 有符号(Ws)	1	1	—	—	—	—	—	—	—	—	—	—	—
MULW.SU	Wb,Ws,Wnd	Wnd = 有符号(Wb) * 无符号(Ws)	1	1	—	—	—	—	—	—	—	—	—	—	—
MULW.SU	Wb,#lit5,Wnd	Wnd = 有符号(Wb) * 无符号(lit5)	1	1	—	—	—	—	—	—	—	—	—	—	—
MULW.US	Wb,Ws,Wnd	Wnd = 无符号(Wb) * 有符号(Ws)	1	1	—	—	—	—	—	—	—	—	—	—	—
MULW.UU	Wb,Ws,Wnd	Wnd = 无符号(Wb) * 无符号(Ws)	1	1	—	—	—	—	—	—	—	—	—	—	—
MULW.UU	Wb,#lit5,Wnd	Wnd = 无符号(Wb) * 无符号(lit5)	1	1	—	—	—	—	—	—	—	—	—	—	—
NEG	f {,WREG}	目标寄存器 = $\overline{f}$ + 1	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
NEG	Ws,Wd	Wd = $\overline{Ws}$ + 1	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
NEG	Acc	对累加器内容求补	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
NOP		空操作	1	1	—	—	—	—	—	—	—	—	—	—	—
NOPR		空操作	1	1	—	—	—	—	—	—	—	—	—	—	—
NORM	Acc,Wd	将累加器内容归一化	1	1	0	0	Ù	Ù	Ù	—	—	—	—	—	—
POP	f	栈顶内容弹出至 f	1	1	—	—	—	—	—	—	—	—	—	—	—
POP	Wd	栈顶内容弹出至 Wd	1	1	—	—	—	—	—	—	—	—	—	—	—
POP.D	Wnd	将栈顶内容双字弹出至 Wnd:Wnd + 1	1	2	—	—	—	—	—	—	—	—	—	—	—
POP.S		弹出影子寄存器内容	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
PUSH	f	将 f 内容压入栈顶	1	1	—	—	—	—	—	—	—	—	—	—	—
PUSH	Ws	将 Ws 内容压入栈顶	1	1	—	—	—	—	—	—	—	—	—	—	—

图注： ⇕ 置 1 或清零；⇓ *可以*被清零，但永远不会被置 1；⇑ *可以*被置 1，但永远不会被清零；“1”总是置 1；“0”总是清零；—不变

注：

1. 只有在相应的饱和功能使能时，SA、SB 和 SAB 才可被修改，否则将不会发生改变。

.....（续）

汇编语法 助记符，操作数 .		说明	指令字数	指令周期数	OA	OB	SA	SB(1)	OAB	SAB(1)	DC	N	OV	Z	C
PUSH.D	Wns	将 Wns:Wns + 1 内容双字压入栈顶	1	2	—	—	—	—	—	—	—	—	—	—	—
PUSH.S		压入影子寄存器	1	1	—	—	—	—	—	—	—	—	—	—	—
PWRSAV	#lit1	进入节能模式	1	1	—	—	—	—	—	—	—	—	—	—	—
RCALL	Expr	相对调用	1	2	—	—	—	—	—	—	—	—	—	—	—
RCALL	Expr	相对调用	1	2	—	—	—	—	—	—	—	—	—	—	—
RCALL	Wn	计算相对调用	1	2	—	—	—	—	—	—	—	—	—	—	—
RCALL	Wn	计算相对调用	1	2	—	—	—	—	—	—	—	—	—	—	—
REPEAT	#lit14	重复执行下一条指令(lit14 + 1)次	1	1	—	—	—	—	—	—	—	—	—	—	—
REPEAT	#lit5	重复执行下一条指令(lit5 + 1)次	1	1	—	—	—	—	—	—	—	—	—	—	—
REPEAT	Wn	重复执行下一条指令(Wn + 1)次	1	1	—	—	—	—	—	—	—	—	—	—	—
REPEAT	Wn	重复执行下一条指令(Wn + 1)次	1	1	—	—	—	—	—	—	—	—	—	—	—
RESET		软件器件复位	1	1	—	—	—	—	—	—	—	—	—	—	—
RETFIE		从中断返回	1	3（2）	—	—	—	—	—	—	—	⇕	⇕	⇕	⇕
RETFIE		从中断返回	1	3（2）	—	—	—	—	—	—	—	⇕	⇕	⇕	⇕
RETLW	#lit10,Wn	从子程序返回并将 lit10 存入 Wn	1	3（2）	—	—	—	—	—	—	—	—	—	—	—
RETLW	#lit10,Wn	从子程序返回并将 lit10 存入 Wn	1	3（2）	—	—	—	—	—	—	—	—	—	—	—
RETURN		从子程序返回	1	3（2）	—	—	—	—	—	—	—	—	—	—	—
RETURN		从子程序返回	1	3（2）	—	—	—	—	—	—	—	—	—	—	—
RLC	f {,WREG}	目标寄存器 = 带进位位循环左移 f 内容	1	1	—	—	—	—	—	—	—	⇕	—	⇕	⇕
RLC	Ws,Wd	Wd = 带进位位循环左移 Ws 内容	1	1	—	—	—	—	—	—	—	⇕	—	⇕	⇕
RLNC	f {,WREG}	目标寄存器 = 不带进位位循环左移 f 内容	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
RLNC	Ws,Wd	Wd = 不带进位位循环左移 Ws 内容	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
RRC	f {,WREG}	目标寄存器 = 带进位位循环右移 f 内容	1	1	—	—	—	—	—	—	—	⇕	—	⇕	⇕
RRC	Ws,Wd	Wd = 带进位位循环右移 Ws 内容	1	1	—	—	—	—	—	—	—	⇕	—	⇕	⇕

图注：⇕ 置 1 或清零；⇓ *可以*被清零，但永远不会被置 1；⇑ *可以*被置 1，但永远不会被清零；“1”总是置 1；“0”总是清零；—不变

注：

1. 只有在相应的饱和功能使能时，SA、SB 和 SAB 才可被修改，否则将不会发生改变。

.....（续）

汇编语法 助记符，操作数 .		说明	指令字数	指令周期数	OA	OB	SA	SB(1)	OAB	SAB(1)	DC	N	OV	Z	C
RRNC	f {,WREG}	目标寄存器 = 不带进位位循环右移 f 内容	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
RRNC	Ws,Wd	Wd = 不带进位位循环右移 Ws 内容	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
SAC	Acc,#Slit4,Wd	保存累加器内容	1	1	—	—	—	—	—	—	—	—	—	—	—
SAC.D	Acc,#Slit4,Wnd	保存累加器内容（双字）	1	1	—	—	—	—	—	—	—	—	—	—	—
SAC.R	Acc,#Slit4,Wd	保存舍入后的累加器内容	1	1	—	—	—	—	—	—	—	—	—	—	—
SE	Ws,Wnd	Wd = 符号扩展 Ws 内容	1	1	—	—	—	—	—	—	—	⇕	—	⇕	⇕
SETM	f	f = 0xFFFF	1	1	—	—	—	—	—	—	—	—	—	—	—
SETM	Wd	Wd = 0xFFFF	1	1	—	—	—	—	—	—	—	—	—	—	—
SFTAC	Acc,#Slit6	算术移位累加器内容 Slit6 位	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
SFTAC	Acc,Wb	算术移位累加器内容(Wb)位	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
SL	f {,WREG}	目标寄存器 = 算术左移 f 内容	1	1	—	—	—	—	—	—	—	⇕	—	⇕	⇕
SL	Ws,Wd	Wd = 算术左移 Ws 内容	1	1	—	—	—	—	—	—	—	⇕	—	⇕	⇕
SL	Wb,#lit4,Wnd	Wnd = Wb 左移 lit4 位	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
SL	Wb,Wns,Wnd	Wnd = Wb 左移 Wns 位	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
SUB	f {,WREG}	目标寄存器 = f - WREG	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
SUB	#lit10,Wn	Wn = Wn – lit10	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
SUB	Wb,#lit5,Wd	Wd = Wb – lit5	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
SUB	Wb,Ws,Wd	Wd = Wb – Ws	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
SUB	Acc	累加器相减	1	1	⇕	⇕	⇑	⇑	⇕	⇑	—	—	—	—	—
SUBB	f {,WREG}	目标寄存器 = f – WREG – (C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
SUBB	#lit10,Wn	Wn = Wn – lit10 – (C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
SUBB	Wb,#lit5,Wd	Wd = Wb – lit5 – (C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
SUBB	Wb,Ws,Wd	Wd = Wb – Ws – (C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
SUBBR	f {,WREG}	目标寄存器 = WREG – f – (C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
SUBBR	Wb,#lit5,Wd	Wd = lit5 – Wb – (C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
SUBBR	Wb,Ws,Wd	Wd = Ws – Wb – (C)	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
SUBR	f {,WREG}	目标寄存器 = WREG – f	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
SUBR	Wb,#lit5,Wd	Wd = lit5 – Wb	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕

图注：⇕ 置 1 或清零；⇓ *可以*被清零，但永远不会被置 1；⇑ *可以*被置 1，但永远不会被清零；“1”总是置 1；“0”总是清零；—不变

注：

1. 只有在相应的饱和功能使能时，SA、SB 和 SAB 才可被修改，否则将不会发生改变。

.....（续）

汇编语法 助记符，操作数 .		说明	指令字数	指令周期数	OA	OB	SA	SB(1)	OAB	SAB(1)	DC	N	OV	Z	C
SUBR	Wb,Ws,Wd	Wd = Ws – Wb	1	1	—	—	—	—	—	—	⇕	⇕	⇕	⇕	⇕
SWAP	Wn	Wn = 对 Wn 进行字节或半字节交换	1	1	—	—	—	—	—	—	—	—	—	—	—
TBLRDH	[Ws],Wd	将高位程序字读入 Wd	1	2	—	—	—	—	—	—	—	—	—	—	—
TBLRDL	[Ws],Wd	将低位程序字读入 Wd	1	2	—	—	—	—	—	—	—	—	—	—	—
TBLWTH	Ws,[Wd]	将 Ws 中的内容写入高位程序字	1	2	—	—	—	—	—	—	—	—	—	—	—
TBLWTL	Ws,[Wd]	将 Ws 中的内容写入低位程序字	1	2	—	—	—	—	—	—	—	—	—	—	—
ULNK		释放堆栈帧	1	1	—	—	—	—	—	—	—	—	—	—	—
ULNK		释放堆栈帧	1	1	—	—	—	—	—	—	—	—	—	—	—
VFSLV	Wns,Wnd,#lit2	验证从处理器程序 RAM	1	1	—	—	—	—	—	—	—	—	—	—	—
XOR	f {,WREG}	目标寄存器 = f .XOR.WREG	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
XOR	#lit10,Wn	Wn = lit10 .XOR.Wn	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
XOR	Wb,#lit5,Wd	Wd = Wb .XOR. lit5	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
XOR	Wb,Ws,Wd	Wd = Wb .XOR.Ws	1	1	—	—	—	—	—	—	—	⇕	—	⇕	—
ZE	Ws,Wnd	Wnd = 零扩展 Ws	1	1	—	—	—	—	—	—	—	0	—	⇕	1

图注： ⇕ 置 1 或清零；⇓ *可以*被清零，但永远不会被置 1；⇑ *可以*被置 1，但永远不会被清零；“1”总是置 1；“0”总是清零；—不变

注：

1. 只有在相应的饱和功能使能时，SA、SB 和 SAB 才可被修改，否则将不会发生改变。

表 6-2. dsPIC33A CPU 指令状态标志操作

设计助记符	受影响的状态位	C	N	OV	Z	OA	OB	SA1	SB	SFA
传送操作										
CLRF	无	—	—	—	—	—	—	—	—	—
EXCH	无	—	—	—	—	—	—	—	—	—
LDW	无	—	—	—	—	—	—	—	—	—
LDWB	无	—	—	—	—	—	—	—	—	—
LDWE	无	—	—	—	—	—	—	—	—	—
LDWLO	无	—	—	—	—	—	—	—	—	—
LDWLOCR	无	—	—	—	—	—	—	—	—	—
MOV	无	—	—	—	—	—	—	—	—	—
MOVIF	无	—	—	—	—	—	—	—	—	—
MOVL	无	—	—	—	—	—	—	—	—	—
MOVR	无	—	—	—	—	—	—	—	—	—
MOVS	无	—	—	—	—	—	—	—	—	—
MOVWS	无	—	—	—	—	—	—	—	—	—
MOVSW	无	—	—	—	—	—	—	—	—	—
MOVWCR	无	—	—	—	—	—	—	—	—	—
MOVCRW	无	—	—	—	—	—	—	—	—	—
POPCR	无	—	—	—	—	—	—	—	—	—
PUSHCR	无	—	—	—	—	—	—	—	—	—
POPW	无	—	—	—	—	—	—	—	—	—
PUSHW	无	—	—	—	—	—	—	—	—	—
SETF	无	—	—	—	—	—	—	—	—	—
STW	无	—	—	—	—	—	—	—	—	—
STWB	无	—	—	—	—	—	—	—	—	—
STWE	无	—	—	—	—	—	—	—	—	—
STWLO	无	—	—	—	—	—	—	—	—	—
STWLOCR	无	—	—	—	—	—	—	—	—	—
TSTF	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
数学操作——W 寄存器										
ADD	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
ADDC	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
AND	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
FLIM(W)	N、Z 和 OV	—	⇕	0	⇕	—	—	—	—	—
IOR	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
SUB	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
SUBB	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
SUBR	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
SUBBR	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
XOR	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
数学操作——短立即数（立即数 0 至 31）										

.....（续）

设计助记符	受影响的状态位	C	N	OV	Z	OA	OB	SA1	SB	SFA
ADDLN	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
ADDLS	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
ADDCLS	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
ANDLS	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
ANDLS1	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
IORLS	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
SUBLN	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
SUBLS	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
SUBBLS	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
SUBRLS	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
SUBBRLS	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
XORLS	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
数学操作——W 寄存器单操作数										
COM	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
数学操作——文件寄存器										
ADDWF	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
ADDCWF	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
ANDWF	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
IORWF	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
SUBFW	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
SUBBFW	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
SUBWF	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
SUBBWF	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
XORWF	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
数学操作——文件寄存器单操作数										
COMF	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
DECF	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
DECF2	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
INCF	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
INCF2	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—

.....（续）

设计助记符	受影响的状态位	C	N	OV	Z	OA	OB	SA1	SB	SFA
NEGF	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
数学操作——立即数（立即数-512 至 511）										
ADDLW	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
ADDCLW	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
ANDLW	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
IORLW	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
SUBLW	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
SUBBLW	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
XORLW	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
数学操作——除法、乘法和调整										
DIVF	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
DIVFL	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
DIVS	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
DIVSL	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
DIVU.l	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
DIVU.w	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
DIVUL	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
MULASLS	无	—	—	—	—	—	—	—	—	—
MULAULS	无	—	—	—	—	—	—	—	—	—
MULASS	无	—	—	—	—	—	—	—	—	—
MULASU	无	—	—	—	—	—	—	—	—	—
MULAUS	无	—	—	—	—	—	—	—	—	—
MULAUU	无	—	—	—	—	—	—	—	—	—
MULSLS	无	—	—	—	—	—	—	—	—	—
MULULS	无	—	—	—	—	—	—	—	—	—
MULSS	无	—	—	—	—	—	—	—	—	—
MULSU	无	—	—	—	—	—	—	—	—	—
MULUS	无	—	—	—	—	—	—	—	—	—
MULUU	无	—	—	—	—	—	—	—	—	—
MULWF	无	—	—	—	—	—	—	—	—	—
SE	C、N 和 Z	⇕	⇕	—	⇕	—	—	—	—	—
ZE	C、N 和 Z	1	0	—	⇕	—	—	—	—	—
SWAP	无	—	—	—	—	—	—	—	—	—
循环/移位操作——W 寄存器										
ASR	C、N 和 Z	⇕	⇕	—	⇕	—	—	—	—	—

.....（续）

设计助记符	受影响的状态位	C	N	OV	Z	OA	OB	SA1	SB	SFA
LSR	C、N 和 Z	⇕	⇕	—	⇕	—	—	—	—	—
RLC	C、N 和 Z	⇕	⇕	—	⇕	—	—	—	—	—
RLNC	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
RRC	C、N 和 Z	⇕	⇕	—	⇕	—	—	—	—	—
RRNC	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
SL	C、N 和 Z	⇕	⇕	—	⇕	—	—	—	—	—
循环/移位操作——文件寄存器										
ASRF	C、N 和 Z	⇕	⇕	—	⇕	—	—	—	—	—
LSRF	C、N 和 Z	⇕	0	—	⇕	—	—	—	—	—
RLCF	C、N 和 Z	⇕	ô	—	⇕	—	—	—	—	—
RLNCF	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
RRCF	C、N 和 Z	⇕	⇕	—	⇕	—	—	—	—	—
RRNCF	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
SLF	C、N 和 Z	⇕	⇕	—	⇕	—	—	—	—	—
桶形移位操作——W 寄存器（移位范围-16 至 15）										
ASRW	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
ASRMW	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
LSRW	N 和 Z	—	0	—	⇕	—	—	—	—	—
LSRMW	N 和 Z	—	0	—	⇕	—	—	—	—	—
SLW	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
SLMW	N 和 Z	—	—	—	⇕	—	—	—	—	—
桶形移位操作——短立即数（移位范围-16 至 15）										
ASRK	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
ASRMK	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
LSRK	N 和 Z	—	0	—	⇕	—	—	—	—	—
LSRMK	N 和 Z	—	0	—	⇕	—	—	—	—	—
SLK	N 和 Z	—	⇕	—	⇕	—	—	—	—	—
SLMK	N 和 Z	—	—	—	⇕	—	—	—	—	—
DSP 操作——累加器操作										
ADDAB	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
ADDAC	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
LAC	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
LLAC	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
LUAC	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—

.....（续）

设计助记符	受影响的状态位	C	N	OV	Z	OA	OB	SA1	SB	SFA
NEGAB	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
NORMACW	N 和 Z									
OA 或 OB	—	⇕	—	⇕	0	0	—	—	—	
MAXAB(W)	N、OV 和 Z	—	⇕	0	⇕	—	—	—	—	—
MINAB(W)	N、OV 和 Z	—	⇕	0	⇕	—	—	—	—	—
SAC	无	—	—	—	—	—	—	—	—	—
SLAC	无	—	—	—	—	—	—	—	—	—
SUAC	无	—	—	—	—	—	—	—	—	—
SFTAC	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
SFTACK	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
SRAC	无	—	—	—	—	—	—	—	—	—
SRACW	无	—	—	—	—	—	—	—	—	—
SUBAB	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
SUBAC	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
DSP 操作——MAC 操作										
ED	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
EDAC	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
MAC	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
MPY	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
MPYN	OA 和 OB	—	—	—	—	⇕	⇕	⇑	⇑	—
MSC	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
SQR	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
SQRAC	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—

.....（续）

设计助记符	受影响的状态位	C	N	OV	Z	OA	OB	SA1	SB	SFA
SQRN	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
SQRSC	OA 和 SA，或者 OB 和 SB	—	—	—	—	⇕	⇕	⇑	⇑	—
位操作——W 寄存器										
BCLR	无	—	—	—	—	—	—	—	—	—
BSET	无	—	—	—	—	—	—	—	—	—
BSW	无	—	—	—	—	—	—	—	—	—
BTG	无	—	—	—	—	—	—	—	—	—
BTST	C 或 Z	⇕	—	—	⇕	—	—	—	—	—
BTSTS	C 或 Z	⇕	—	—	⇕	—	—	—	—	—
BTSTW	C 或 Z	⇕	—	—	⇕	—	—	—	—	—
位操作——文件寄存器										
BCLRF	无	—	—	—	—	—	—	—	—	—
BSETF	无	—	—	—	—	—	—	—	—	—
BTGF	无	—	—	—	—	—	—	—	—	—
BTSTF	Z	—	—	—	⇕	—	—	—	—	—
BTSTSF	Z	—	—	—	⇕	—	—	—	—	—
位查找操作										
FBCL	C	⇕	—	—	—	—	—	—	—	—
FF1L	C	⇕	—	—	—	—	—	—	—	—
FF1R	C	⇕	—	—	—	—	—	—	—	—
位域操作										
BFEXT	无	—	—	—	—	—	—	—	—	—
BFEXTF	无	—	—	—	—	—	—	—	—	—
BFINS	无	—	—	—	—	—	—	—	—	—
BFINSF	无	—	—	—	—	—	—	—	—	—
BFINSL	无	—	—	—	—	—	—	—	—	—
比较和限制操作——W 寄存器										
CP	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
CPB	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
MAX	N、OV 和 Z	—	0	0	⇕	—	—	—	—	—
MIN	N、OV 和 Z	—	⇕	0	⇕	—	—	—	—	—
比较操作——字节立即数（立即数 0 至 255）										
CPLW	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
CPBLW	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
比较操作——文件寄存器										
CPF0	C、N、OV 和 Z	1	⇕	⇕	⇕	—	—	—	—	—

.....（续）

设计助记符	受影响的状态位	C	N	OV	Z	OA	OB	SA1	SB	SFA
CPF	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
CPFB	C、N、OV 和 Z	⇕	⇕	⇕	⇕	—	—	—	—	—
转移操作										
BC	无	—	—	—	—	—	—	—	—	—
BGE	无	—	—	—	—	—	—	—	—	—
BGT	无	—	—	—	—	—	—	—	—	—
BGTU	无	—	—	—	—	—	—	—	—	—
BLE	无	—	—	—	—	—	—	—	—	—
BLEU	无	—	—	—	—	—	—	—	—	—
BLT	无	—	—	—	—	—	—	—	—	—
BN	无	—	—	—	—	—	—	—	—	—
BNC	无	—	—	—	—	—	—	—	—	—
BNN	无	—	—	—	—	—	—	—	—	—
BNOV	无	—	—	—	—	—	—	—	—	—
BNZ	无	—	—	—	—	—	—	—	—	—
BOA	无	—	—	—	—	—	—	—	—	—
BOB	无	—	—	—	—	—	—	—	—	—
BOV	无	—	—	—	—	—	—	—	—	—
BRA	无	—	—	—	—	—	—	—	—	—
BSA	无	—	—	—	—	—	—	—	—	—
BSB	无	—	—	—	—	—	—	—	—	—
BZ	无	—	—	—	—	—	—	—	—	—
跳转/调用/返回操作										
BRAW	无	—	—	—	—	—	—	—	—	—
CALL	无	—	—	—	—	—	—	—	—	0
CALLW	无	—	—	—	—	—	—	—	—	0
GOTO	无	—	—	—	—	—	—	—	—	—
GOTOW	无	—	—	—	—	—	—	—	—	—
RCALL	无	—	—	—	—	—	—	—	—	0
RCALLW	无	—	—	—	—	—	—	—	—	0
RETFIE	IPL[3:0]、RA、N、OV、Z 和 C	⇕	⇕	⇕	⇕	—	—	—	—	⇕
RETLW	无	—	—	—	—	—	—	—	—	⇕
RETURN	无	—	—	—	—	—	—	—	—	⇕
循环操作										
DTB	无	—	—	—	—	—	—	—	—	—
REPEAT	无	—	—	—	—	—	—	—	—	—
REPEATLS	无	—	—	—	—	—	—	—	—	—
REPEATW	无	—	—	—	—	—	—	—	—	—
堆栈操作										
LNK	无	—	—	—	—	—	—	—	—	1
POP	无	—	—	—	—	—	—	—	—	—
POPW	无	—	—	—	—	—	—	—	—	—

.....（续）

设计助记符	受影响的状态位	C	N	OV	Z	OA	OB	SA1	SB	SFA
PUSH	无	—	—	—	—	—	—	—	—	—
PUSHW	无	—	—	—	—	—	—	—	—	—
MOVWS	无	—	—	—	—	—	—	—	—	—
MOVSW	无	—	—	—	—	—	—	—	—	—
ULNK	无	—	—	—	—	—	—	—	—	0
控制操作										
BREAK	无	—	—	—	—	—	—	—	—	—
BOOTSWP	无	—	—	—	—	—	—	—	—	—
CLRWDT	TO 和 PD	—	—	—	—	—	—	—	—	—
CTXTSWP	无	—	—	—	—	—	—	—	—	—
CTXTSWPW	无	—	—	—	—	—	—	—	—	—
DISICTL	无	—	—	—	—	—	—	—	—	—
DISICTLW	无	—	—	—	—	—	—	—	—	—
NOP	无	—	—	—	—	—	—	—	—	—
NOPR	无	—	—	—	—	—	—	—	—	—
PWRSAV	WDTO、SLEEP 和 IDLE	—	—	—	—	—	—	—	—	—
RESET	无	—	—	—	—	—	—	—	—	—

图注： ⇳ 置 1 或清零，“1”总是置 1，“0”总是清零，—不变
⇓ 可以被清零，但永远不会被置 1（粘住），⇑ 可以被置 1，但永远不会被清零（粘住）

注：

1. 只有在相应的饱和位置 1 时，SA、SB 和 SAB 才可被修改，否则将不会发生改变

6.3 版本历史

版本 A（2023 年 10 月）

这是本文档的初始版本。

Microchip 信息

Microchip 网站

Microchip 网站（www.microchip.com）为客户提供在线支持。客户可通过该网站方便地获取文件和信息。我们的网站提供以下内容：

- **产品支持**——数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南以及硬件支持文档、最新的软件版本以及归档软件
- **一般技术支持**——常见问题解答（FAQ）、技术支持请求、在线讨论组以及 Microchip 设计伙伴计划成员名单
- **Microchip 业务**——产品选型和订购指南、最新 Microchip 新闻稿、研讨会和活动安排表、Microchip 销售办事处、代理商以及工厂代表列表

产品变更通知服务

Microchip 的产品变更通知服务有助于客户了解 Microchip 产品的最新信息。注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请访问 www.microchip.com/pcn，然后按照注册说明进行操作。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师（ESE）
- 技术支持

客户应联系其代理商、代表或 ESE 寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过 www.microchip.com/support 获得网上技术支持。

Microchip 器件代码保护功能

请注意以下有关 Microchip 产品代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术规范。
- Microchip 确信：在正常使用且符合工作规范的情况下，Microchip 系列产品非常安全。
- Microchip 注重并积极保护其知识产权。严禁任何试图破坏 Microchip 产品代码保护功能的行为，这种行为可能会违反《数字千年版权法案》（Digital Millennium Copyright Act）。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

本出版物及其提供的信息仅适用于 Microchip 产品，包括设计、测试以及将 Microchip 产品集成到您的应用中。以其他任何方式使用这些信息都将被视为违反条款。本出版物中的器件应用信息仅为您提供便利，将来可能会发生更新。如需额外的支持，请联系当地的 Microchip 销售办事处，或访问 www.microchip.com/en-us/support/design-help/client-support-services。

Microchip“按原样”提供这些信息。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对非侵权性、适销性和特定用途的适用性的暗示担保，或针对其使用情况、质量或性能的担保。

在任何情况下，对于因这些信息或使用这些信息而产生的任何间接的、特殊的、惩罚性的、偶然的或间接的损失、损害或任何类型的开销，Microchip 概不承担任何责任，即使 Microchip 已被告知可能发生损害或损害可以预见。在法律允许的最大范围内，对于因这些信息或使用这些信息而产生的所有索赔，Microchip 在任何情况下所承担的全部责任均不超出您为获得这些信息向 Microchip 直接支付的金额（如有）。如果将 Microchip 器件用于生命维持和/或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切损害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任。除非另外声明，在 Microchip 知识产权保护下，不得暗中或以其他方式转让任何许可证。

商标

Microchip 的名称和徽标组合、Microchip 徽标、Adaptec、AVR、AVR 徽标、AVR Freaks、BesTime、BitCloud、CryptoMemory、CryptoRF、dsPIC、flexPWR、HELDO、IGLOO、JukeBlox、KeeLoq、Kleer、LANCheck、LinkMD、maXStylus、maXTouch、MediaLB、megaAVR、Microsemi、Microsemi 徽标、MOST、MOST 徽标、MPLAB、OptoLyzer、PIC、picoPower、PICSTART、PIC32 徽标、PolarFire、Prochip Designer、QTouch、SAM-BA、SenGenuity、SpyNIC、SST、SST 徽标、SuperFlash、Symmetricom、SyncServer、Tachyon、TimeSource、tinyAVR、UNI/O、Vectron 及 XMEGA 均为 Microchip Technology Incorporated 在美国和其他国家或地区的注册商标。

AgileSwitch、ClockWorks、The Embedded Control Solutions Company、EtherSynch、Flashtec、Hyper Speed Control、HyperLight Load、Libero、motorBench、mTouch、Powermite 3、Precision Edge、ProASIC、ProASIC Plus、ProASIC Plus 徽标、Quiet-Wire、SmartFusion、SyncWorld、TimeCesium、TimeHub、TimePictra、TimeProvider 和 ZL 均为 Microchip Technology Incorporated 在美国的注册商标。

Adjacent Key Suppression、AKS、Analog-for-the-Digital Age、Any Capacitor、AnyIn、AnyOut、Augmented Switching、BlueSky、BodyCom、Clockstudio、CodeGuard、CryptoAuthentication、CryptoAutomotive、CryptoCompanion、CryptoController、dsPICDEM、dsPICDEM.net、Dynamic Average Matching、DAM、ECAN、Espresso T1S、EtherGREEN、EyeOpen、GridTime、IdealBridge、IGaT、In-Circuit Serial Programming、ICSP、INICnet、Intelligent Paralleling、IntelliMOS、Inter-Chip Connectivity、JitterBlocker、Knob-on-Display、MarginLink、maxCrypto、maxView、memBrain、Mindi、MiWi、MPASM、MPF、MPLAB Certified 徽标、MPLIB、MPLINK、mSiC、MultiTRAK、NetDetach、Omniscient Code Generation、PICDEM、PICDEM.net、PICkit、PICtail、Power MOS IV、Power MOS 7、PowerSmart、PureSilicon、QMatrix、REAL ICE、Ripple Blocker、RTAX、RTG4、SAM-ICE、Serial Quad I/O、simpleMAP、SimpliPHY、SmartBuffer、SmartHLS、SMART-I.S.、storClad、SQI、SuperSwitcher、SuperSwitcher II、Switchtec、SynchroPHY、Total Endurance、Trusted Time、TSHARC、Turing、USBCheck、VariSense、VectorBlox、VeriPHY、ViewSpan、WiperLock、XpressConnect 和 ZENA 均为 Microchip Technology Incorporated 在美国和其他国家或地区的商标。

SQTP 为 Microchip Technology Incorporated 在美国的服务标记。

Adaptec 徽标、Frequency on Demand、Silicon Storage Technology 和 Symmcom 均为 Microchip Technology Inc.在除美国外的国家或地区的注册商标。

GestIC 为 Microchip Technology Inc.的子公司 Microchip Technology Germany II GmbH & Co. KG 在除美国外的国家或地区的注册商标。

在此提及的所有其他商标均为各持有公司所有。

ISBN： 978-1-6683-3503-1

质量管理体系

有关 Microchip 质量管理体系的信息，请访问 www.microchip.com/quality。

全球销售及服务网点

美洲	亚太地区	亚太地区	欧洲
公司总部 2355 West Chandler Blvd. Chandler, AZ 85224-6199 电话：480-792-7200 传真：480-792-7277 技术支持： www.microchip.com/support 网址： www.microchip.com	**澳大利亚 - 悉尼** 电话：61-2-9868-6733	**印度 - 班加罗尔** 电话：91-80-3090-4444	**奥地利 - 韦尔斯** 电话：43-7242-2244-39 传真：43-7242-2244-393
亚特兰大 德卢斯，佐治亚州 电话：678-957-9614 传真：678-957-1455	**中国 - 北京** 电话：86-10-8569-7000	**印度 - 新德里** 电话：91-11-4160-8631	**丹麦 - 哥本哈根** 电话：45-4485-5910 传真：45-4485-2829
奥斯汀，德克萨斯州 电话：512-257-3370	**中国 - 成都** 电话：86-28-8665-5511	**印度 - 浦那** 电话：91-20-4121-0141	**芬兰 - 埃斯波** 电话：358-9-4520-820
波士顿 韦斯特伯鲁，马萨诸塞州 电话：774-760-0087 传真：774-760-0088	**中国 - 重庆** 电话：86-23-8980-9588	**日本 - 大阪** 电话：81-6-6152-7160	**法国 - 巴黎** 电话：33-1-69-53-63-20 传真：33-1-69-30-90-79
芝加哥 艾塔斯卡，伊利诺伊州 电话：630-285-0071 传真：630-285-0075	**中国 - 东莞** 电话：86-769-8702-9880	**日本 - 东京** 电话：81-3-6880-3770	**德国 - 加兴** 电话：49-8931-9700
达拉斯 阿迪森，德克萨斯州 电话：972-818-7423 传真：972-818-2924	**中国 - 广州** 电话：86-20-8755-8029	**韩国 - 大邱** 电话：82-53-744-4301	**德国 - 哈恩** 电话：49-2129-3766400
底特律 诺维，密歇根州 电话：248-848-4000	**中国 - 杭州** 电话：86-571-8792-8115	**韩国 - 首尔** 电话：82-2-554-7200	**德国 - 海尔布隆** 电话：49-7131-72400
休斯顿，德克萨斯州 电话：281-894-5983	**中国 - 香港特别行政区** 电话：852-2943-5100	**马来西亚 - 吉隆坡** 电话：60-3-7651-7906	**德国 - 卡尔斯鲁厄** 电话：49-721-625370
印第安纳波利斯 诺布尔斯维尔，印第安纳州 电话：317-773-8323 传真：317-773-5453 电话：317-536-2380	**中国 - 南京** 电话：86-25-8473-2460	**马来西亚 - 槟榔屿** 电话：60-4-227-8870	**德国 - 慕尼黑** 电话：49-89-627-144-0 传真：49-89-627-144-44
洛杉矶 米慎维荷，加利福尼亚州 电话：949-462-9523 传真：949-462-9608 电话：951-273-7800	**中国 - 青岛** 电话：86-532-8502-7355	**菲律宾 - 马尼拉** 电话：63-2-634-9065	**德国 - 罗森海姆** 电话：49-8031-354-560
罗利，北卡罗来纳州 电话：919-844-7510	**中国 - 上海** 电话：86-21-3326-8000	**新加坡** 电话：65-6334-8870	**以色列 - 霍德夏沙隆** 电话：972-9-775-5100
纽约，纽约州 电话：631-435-6000	**中国 - 沈阳** 电话：86-24-2334-2829	**台湾地区 - 新竹** 电话：886-3-577-8366	**意大利 - 米兰** 电话：39-0331-742611 传真：39-0331-466781
圣何塞，加利福尼亚州 电话：408-735-9110 电话：408-436-4270	**中国 - 深圳** 电话：86-755-8864-2200	**台湾地区 - 高雄** 电话：886-7-213-7830	**意大利 - 帕多瓦** 电话：39-049-7625286
加拿大 - 多伦多 电话：905-695-1980 传真：905-695-2078	**中国 - 苏州** 电话：86-186-6233-1526	**台湾地区 - 台北** 电话：886-2-2508-8600	**荷兰 - 德卢内市** 电话：31-416-690399 传真：31-416-690340
	中国 - 武汉 电话：86-27-5980-5300	**泰国 - 曼谷** 电话：66-2-694-1351	**挪威 - 特隆赫姆** 电话：47-72884388
	中国 - 西安 电话：86-29-8833-7252	**越南 - 胡志明市** 电话：84-28-5448-2100	**波兰 - 华沙** 电话：48-22-3325737
	中国 - 厦门 电话：86-592-2388138		**罗马尼亚 - 布加勒斯特** 电话：40-21-407-87-50
	中国 - 珠海 电话：86-756-3210040		**西班牙 - 马德里** 电话：34-91-708-08-90 传真：34-91-708-08-91
			瑞典 - 哥德堡 电话：46-31-704-60-40
			瑞典 - 斯德哥尔摩 电话：46-8-5090-4654
			英国 - 沃金厄姆 电话：44-118-921-5800 传真：44-118-921-5820