ColdFire 编程参考手册

（中文）

苏州大学飞思卡尔嵌入式系统实验室翻译

http://sumcu.suda.edu.cn

2009 年 1 月

目 录

表索引

图索引

第 1 章 绪 论

本手册包含了所有版本 ColdFire 微处理器指令集体系的详细信息。在 ColdFire 系列中，每一代硬件微架构都被视为一个版本，这开始于最早先的 V2 内核(首个 ColdFire 处理器)的实现。对于其指令集体系来说，每个定义都被称为一个 ISA_Revision，并被写成 ISA_Revision_A 和 ISA_Revision_B 等。然而，这些 ISA 版本通常使用短形式的名目（如 ISA_A 和 ISA_B 等）来描述。

早先的 ColdFire ISA 是由 M68000 指令集的精简版发展而来的，该精简版指令集被设计用于实现最起码的代码扩展（静态的和动态的）和硬件门逻辑操作。原来的指令集架构（具体如 ISA_A）支持相当规模的 M68000 操作码的子集以及从 M68020 中选出的增加的部分，并在最小化的代码扩展和核心门逻辑运算之间建立了适当的平衡，同时又充分保留了从 M68k 系列继承而来的用户模式编程模型。

随着 ColdFire 系列的发展，用户和工具开发者的输入，以及内部性能分析，表明了某些 ISA 的改进能够增强性能和代码密集度。因此，在基本的指令集架构上，不同的改进版已被详细定义，并在各种 ColdFire 处理器内核中得以实现。此外，某些可选的硬件执行部件，被用于需要对新的 ISA 扩展进行设计的特定应用领域。例如，硬件浮点型单元(FPU)和各种乘法累加(MAC)单元。

ColdFire 沿袭了处理器微架构和指令集体系改进版之间的关联性。随着较新的指令被应用到旧版本的内核中，这种关联性变得模糊了。本手册依照特定的 ISA 改进版本来定义指令集，而不是与任何给定的处理器版本相结合。

现存的 ISA 版本有：

- ISA_A：早先的 ColdFire 指令集架构；
- ISA_B：增加了改进过的数据转移指令、字节长度及字长度的比较等等；
- ISA_C：增加了位操作指令；
- FPU：早先用于浮点型单元(FPU)的 ColdFire 指令集架构；
- MAC：早先用于乘法累加单元(MAC)的 ColdFire 指令集架构；
- EMAC：改进的 ISA，具有增强型乘法累加单元(EMAC)；
- EMAC_B：新增了双乘法累加单元的操作指令。 以上这些都是指令集体系架构的主要改进版。此外，还有些被扩展了的 ISA 版本，

例如 ISA_A+，它在主要改进版的基础上整合了其他版本的可选指令。 复位时，处理器的配置信息通常被加载到两个程序可见的寄存器中。这些信息定义了 ColdFire 核心版本及其 ISA 应用版本。请参见 1.1 节，“整型单元用户编程模型”。

ColdFire 系列的编程模型包括两组寄存器：用户组和管理员组。用户模式下程序的执行，只能够使用用户组寄存器。而系统软件则运行在管理员模式下，不但可以存取所有的寄存器，而且可以使用管理员组的控制寄存器来实现管理者功能。后续章节会对用户模式和管理员模式下的寄存器以及寄存器中的数据组织方式作简要的描述。

1.1 整型单元用户编程模式

图 1-1 显示了用户编程模式的整型部件。它包括以下寄存器：

- 16 个通用的 32 位寄存器（D0–D7，A0–A7）
- 32 位的程序计数器（PC）
- 8 位的条件码寄存器(CCR）

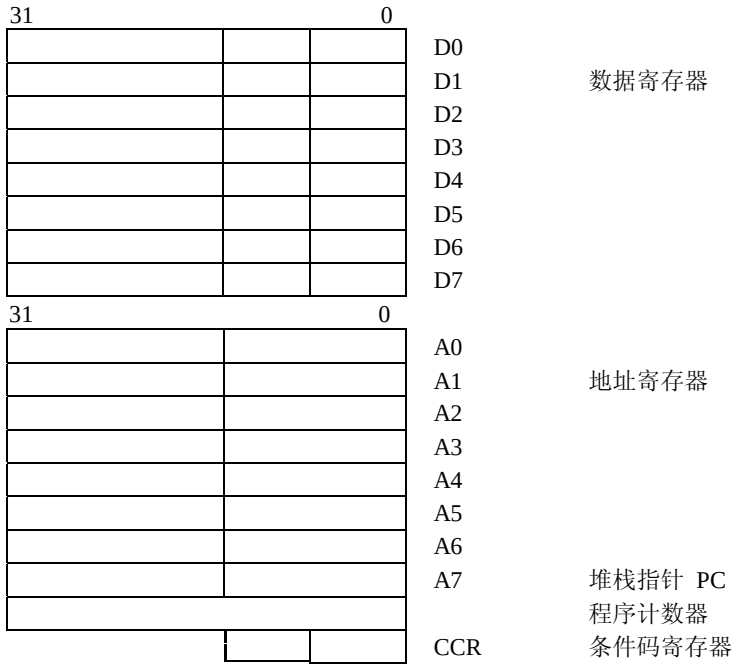


图 **1-1 ColdFire** 系列的用户编程模式

1.1.1 数据寄存器(D0–D7)

这些寄存器用于位、字节（8 位）、字（16 位）以及长字（32 位）的操作。它们还可以被用作变址寄存器。

1.1.2 地址寄存器(A0–A7)

这些寄存器被用作软件的堆栈指针、变址寄存器或基址寄存器。该基址寄存器可以被用来进行字或长字操作。在子程序调用或异常处理时，会用到堆栈空间，这时寄存器 A7 将作为硬件的堆栈指针。

1.1.3 程序计数器**(PC**） 程序计数器包含当前执行指令的地址。当指令正常运行或异常发生时，处理器自动

把它的内容增加或将新值放入 PC。对于某些寻址方式来说，PC 可以作为基于它自身的相对寻址的指针。

1.1.4 条件码寄存器（CCR）

条件码寄存器 CCR（状态寄存器的低字节）包含 5 位，它是 SR（状态寄存器）在用户模式下可用的部分。许多整型指令都影响 CCR，从而显示指令执行的结果。程序和系统控制指令，也会用到这些位的某些组合来控制程序和系统流程。

条件码有两个标准：

1、交叉一致性：

——指令，所有指令都是更加通用指令的特定应用，通过同样的方式影响条件码；

——用法，条件指令通过相似的方法测试条件码，并且不管比较、测试和传送指令是否设置了条件码，它都会给出相同的结果；

——实例，指令的所有实例应用都以相同的方式来影响条件码。

2、有意义的结果总是保持不变，除非提供了更有用的信息。 位[3:0]表示操作结果的情况。位 5，扩展位，是多精度计算的操作数。V3 版处理器在 CCR 中增加了位 7，作为分支预测位。

CCR 各位在图 1-2 中显示。

7	6	5	4	3	2	1	0
P[1]	—		X	N	Z	V	C

[1] P 位只在 V3 版内核中使用。

图 **1-2** 条件码寄存器**(CCR)**

表 1-1 描述了 CCR 的位。

表 **1-1 CCR** 位描述

位	域	描述
7	P	分支预测（仅 V3）。改变被在分支加速逻辑静态的预测法则
	—	预留；应该被清零（所有其他版本）
6-5	—	预留；应该被清零
4	X	扩展。为算法操作设置进位的值，否则不被影响或设置成特定的值。
3	N	负数标志。如果结果的最高位置位就设置该位，否则就清零。
2	Z	零标志位。如果结果等于 0，则设置该位，否则清零。
1	V	溢出标志位。如果发生算术溢出，就说明结果不能用操作数的字长来表示，就设置该位，否则清零。
0	C	进位标志。如果在加法时发生了进位，或减法时发生了借位，则设置该位，否则清零。

1.2 浮点单元用户编程模式

下面章节将描述 ColdFire 可选浮点型型单元的寄存器。图 1-3 描述浮点型型单元的用户编程模型，它包括下列寄存器：

8 个 64 位浮点型数据寄存器（FP0-FP7）

32 位浮点型控制寄存器（FPCR）

32 位浮点型状态寄存器（FPSR）

32 位浮点型指令地址寄存器（FPIAR）

（原文好象没有图!!!）

图 **1-3 ColdFire** 系列浮点单元的用户编程模式

1.2.1 浮点型数据寄存器（**FP0-FP7**）

对于 68K/ColdFire 系列来讲，浮点型型数据寄存器和整型数据寄存器类似。64 位的浮点型型数据寄存器总是以双精度格式来保存数据。所有外部操作数，不管源操作数的格式怎样，在计算或保存到浮点型型数据寄存器中时，都被转换成双精度格式。复位或置零操作会使得 FP0-FP7 变成正数的特性，而不是任何非数值类型(NANs)。

1.2.1.1 浮点型控制寄存器（**FPCR**）

如图 1-4，浮点型控制寄存器包括异常使能字节（EE）和模式控制字节（MC）。用户可以通过 FMOVE 或 FRESTORE 读写 FPCR。处理器复位或恢复空状态会清零 FPCR。当这个寄存器被清零时，FPU 不会产生异常。

	——————异常使能字节（EE）——————							——模式控制字节（MC）——				
31 16	15	14	13	12	11	10	9	8	7	6	5 4	3 0
—	BSUN	INAN	OPERR	OVFL	UNFL	DZ	INEX	IDE	—	PREC	RND	—

图 **1-4** 浮点型控制寄存器（**FPCR**）

表 1-2 描述了 FPCR 的域。

表 **1-2 FPCR** 域的描述

位	域	描述	
31-16	—	预留，应该被清零	
15-8	EE	异常使能位。每个 EE 位对应一个浮点型异常类。用户可以分别为浮点型数异常的每个类设置中断。	
15		BSUN	无序分支设置
14		INAN	输入非数值
13		OPERR	操作数错误
12		OVFL	上溢
11		UNFL	下溢
10		DZ	被 0 除

9	MC	INEX	不精确的操作
8		IDE	非正常输入
7-0		模式控制字节。控制 FPU 操作模式	
7		—	预留，应该被清零
6		PREC	凑整精度
5-4		RND	凑整模式
3-0		—	预留，应该被清零

1.2.2 浮点型状态寄存器(FPSR)

浮点型状态寄存器，如图 1-5，包括浮点型条件码字节(FPCC)、浮点型异常状态字节(EXC)和浮点型异常产生字节(AEXC)。用户可以读/写所有 FPSR 位。大多数浮点型指令都可修改 FPSR。FPSR 通过 FMOVE 或 FRESTORE 指令来加载数据。处理器复位或置零可清除 FPSR。

———FPCC———					———异常状态字节（EXC）———										———AEXC 字节———				
31 28	27	26	25	24	23 16	15	14	13	12	11	10	9	8	7	6	5	4	3	2 0
—	N	Z	I	NAN	—	BSUN	INAN	OPERR	OVFL	UNFL	DZ	INEX	IDE	IOP	OVFL	UNFL	DZ	INEX	—

图 **1-5** 浮点型状态寄存器**(FPSR)**

表 1-3 描述了 FPSR 域。

表 **1-3 FPSR** 域描述

位	域	描述	
31-24	FPCC	浮点型条件码。包含四个条件码位，这些位在完成了所有浮点型数据寄存器中的指令后被设定。	
31-28	FPPC (cont.)	—	预留，应该被清零
27		N	负数
26		Z	零
25		I	无穷大
24		NAN	非数值
23-16	—	预留，应该被清零	
15-8	EXC	异常状态字节。对于最近的算术指令或转移指令可能发生的异常都包含一个位。	
15		BSUN	分支/无序设置

14		INAN	非数值输入
13		OPERR	操作数错误
12		OVFL	上溢
11		UNFL	下溢
10		DZ	除数为 0
9		INEX	不精确的操作
8		IDE	非正常输入
7-0	AEXC	异常产生字节。包含了 IEEE754 对于异常消除操作所要求的 5 个位。这些异常都是 EXC 字节各位的逻辑组合。AEXC 记录了从用户上次清除 AEXC 以来的所有浮点型异常。	
7		IOP	非法操作
6		OVFL	下溢
5		UNFL	除数为 0
4		DZ	非精确的操作
3		INEX	非正常的输入
2-0	—	预留，应该被清零	

1.2.3 浮点型指令地址寄存器（**FPIAR**）

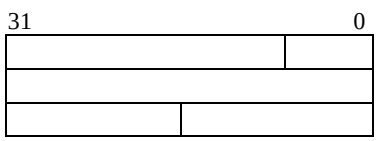
ColdFire 指令通过管道可以同时执行整型和浮点型指令。这样会导致，为了响应浮点数异常中断，程序计数器的值被处理器压入堆栈，而不能指向产生异常的指令。

对于那些可能产生异常中断的 FPU 指令，在 FPU 开始运行之前，32 位的 FPIAR 被加载为指令的程序计数器中的地址。如果出现 FPU 异常，中断处理机制可以通过 FPIAR 的内容来判断产生异常的指令。存取 FPCR、FPSR 或 FPIAR 的 FMOVE 指令和 FMOVEM 指令，不产生的浮点型异常，所以不会改变 FPIAR。复位或恢复空状态可以清除 FPIAR。

1.3 MAC 的用户编程模型

下面的章节描述 ColdFire 可选 MAC 单元所用到的寄存器。图 1-6 描述了 MAC 单元的用户编程模型，它包括了下面几个寄存器：

- 32 位 MAC 状态寄存器(MACSR)
- 32 位累加寄存器(ACC)
- 32 位 MAC 掩码寄存器(MASK)



MACSR MAC 状态寄存器
ACC MAC 累加器
MASK MAC 掩码寄存器

图 **1-6 MAC** 单元编程模型

1.3.1MAC 状态寄存器(MACSR)

状态寄存器，如图 1-7，包括一个操作模式域和一些标志位。

31	7			4	3			
8					0			
	操作模式				标志位			
—	OMC	S/U	F/I	R/T	N	Z	V	C

图 **1-7. MAC** 状态寄存器（**MACSR**）

表 1-4 描述了 MACSR 域。

表 **1-4 MACSR** 域描述

位	域	描述	
31-8	—	预留，应清零	
7-4	OMF	操作模式域。定义 MAC 单元的操作设置。	
7		OMC	上溢/饱和模式
6		S/U	有符号/无符号操作
5		F/I	分数/整数模式
4		R/T	完整/截断模式
3-0	Flags	标志位。包含了最后一次 MAC 指令执行的标志位。	
3		N	负数
2		Z	零
1		V	上溢
0		C	进位。该域始终为 0。

1.3.2 MAC 累加器(ACC)

该 32 位寄存器包含了 MAC 操作的结果。

1.3.3 MAC 掩码寄存器(MASK)

掩码寄存器是个 32 位的寄存器，但只有低 16 位有效。当 MASK 被加载，源操作数的低 16 位就被加载到该寄存器中。当它存储时，高 16 位被强制设成全 1。

当被指令使用时，该寄存器和指定的操作数地址作与操作。这样，MASK 保证了操作数地址，被有效地限制在所定义的 16 位固定范围内。该特性简化了对以下功能的支持，如筛选、循环，或其他使用(Ay)+寻址方式来实现的数组循环队列寻址。

对于有加载操作的 MAC 来说，MASK 的值被有选择性地包含到所有存储器有效地

址的运算中。

1.4 EMAC 用户编程模型

下面的章节描述了 ColdFire 可选 EMAC 单元所用的寄存器。图 1-8 显示了 EMAC 单元的用户编程模型。包括下面的寄存器：

• 32 位 MAC 状态寄存器（MACSR）包括四个标志位显示乘积或加法上溢（每一位对应一个加法器：PAV0-PAV3）

• 4 个 32 位加法器（ACCx=ACC0,ACC1,ACC2,ACC3）

• 8 个 8 位加法器扩展（每个加法器对应两个），为加载或存储操作而封装成两个 32 位值。

• 32 位计时寄存器（MASK）

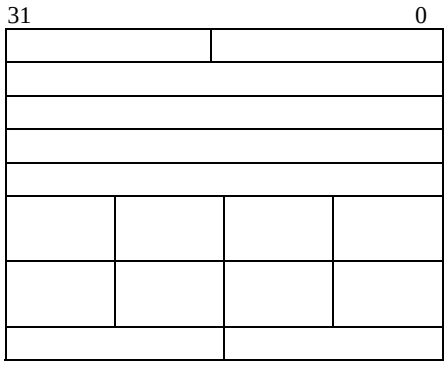


图 **1-8 EMAC** 编程模式

1.4.1MAC 状态寄存器**(MACSR)**

图 1-9 显示了 EMAC EMACSR，包括了一个操作模式域和两个标志位集。

31	11	10	9	8	7	6	5	4	3	2	1	0
12												

	乘法/加法上溢标记				操作模式				标志位			
—	PAV3	PAV2	PAV1	PAV0	OMC	S/U	F/I	R/T	N	Z	V	EV

图 **1-9 MAC** 状态寄存器（**MACSR**）

表 1-5 描述了 EMAC MACSR 域。

表 **1-5 MACSR** 域描述

位	域	描述
31-12	—	预留，应清零
11-8	PAVx	乘积/加法上溢标志，分别对应每个加法器

7-4	OMF	操作模式域。定义 EMAC 单元的操作设置。	
7		OMC	上溢/饱和模式
6		S/U	有符号/无符号操作
5		F/I	分数/整数模式
4		R/T	完整/截断模式
3-0	Flags	标志位。包含最后一次 MAC 指令操作的标志。	
3		N	负数
2		Z	零
1		V	上溢
0		C	进位。该位始终为 0。

1.4.2MAC 加法器(ACC[0:3])

EMAC 使用 48 位加法器。32 位 ACCx 寄存器和加法器扩展字包含了加法器的数据。图 1-10 显示了 EMAC 在分数模式操作时加法器及其扩展字所包含的数据。扩展乘积的高 8 位是乘积结果的 40 位的符号扩展字节。

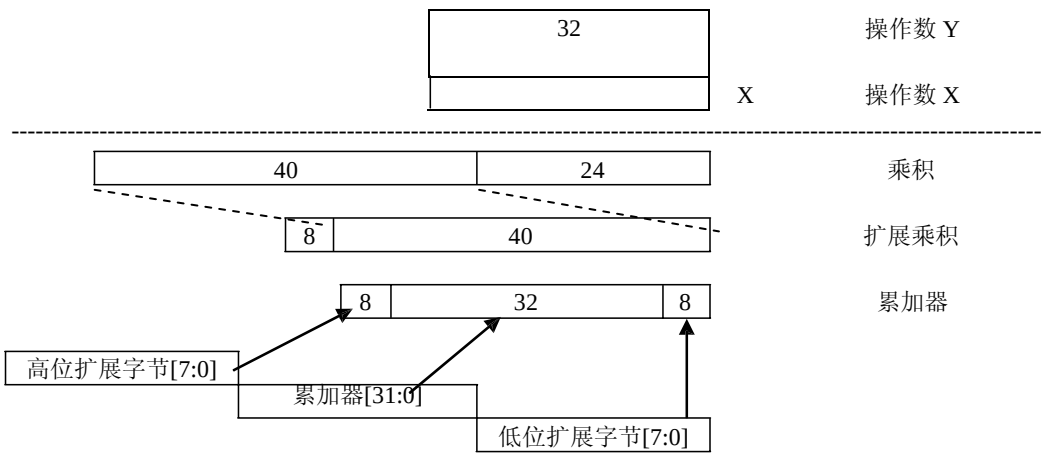
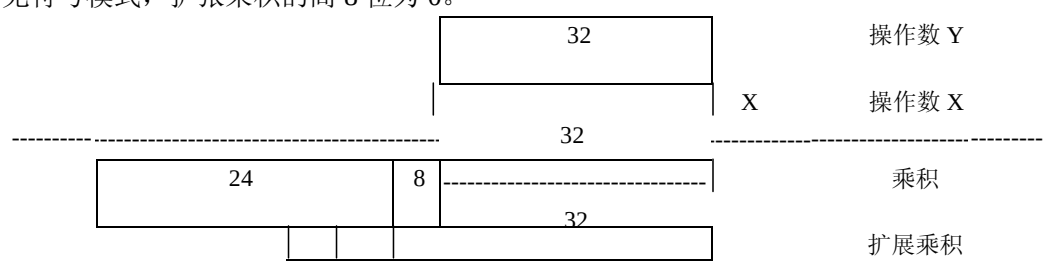


图 **1-10 EMAC** 分数队列

图 1-11 显示了 EMAC 在有符号或无符号的整数模式操作时加法器及其扩展字所包含的数据。在有符号模式，扩展乘积的高 8 位是从乘积结果的 40 位符号扩展而得。在无符号模式，扩张乘积的高 8 位为 0。



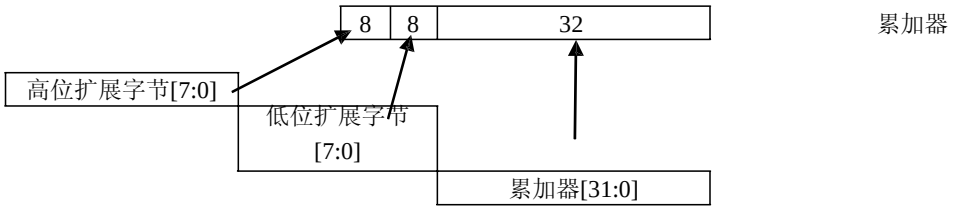


图 **1-11 EMAC** 有符号和无符号整数队列

1.4.3 加法器扩展**(ACCext01,ACCext23)**

32 位的加法扩展寄存器（ACCext01,ACCEXT23）允许 48 位的加法器的值在保存和恢复时高低位调换。图 1-12 显示了 ACC0 和 ACC1 被加载到一个寄存器中时如何保存。参考图 1-10 和图 1-11 有关扩展字节中数据的信息。

31 24	23 16	15 8	7 0
ACC1 高位扩展字节	ACC1 低位扩展字节	ACC0 高位扩展字节	ACC0 低位扩展字节

图 **1-12** 加法器 **0** 和 **1** 的扩展（**ACCext01**）

图 1-13 显示了 ACC0 和 ACC1 被加载到一个寄存器中时如何保存。参考图 1-10 和图 1-11 有关扩展字节中数据的信息。

31 24	23 16	15 8	7 0
ACC3 高位扩展字节	ACC3 低位扩展字节	ACC2 高位扩展字节	ACC2 低位扩展字节

图 **1-13** 加法器 **2** 和 **3** 的扩展（**ACCext23**）

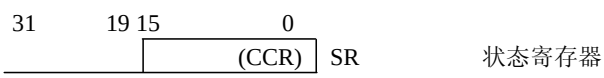
1.4.4 MAC 掩码寄存器**(MASK)**

32 位掩码寄存器仅低 16 位被使用。当 MASK 被加载，源操作数的低 16 位被加载到寄存器中。当存储时，高 16 被强制处理成 1。

当被使用时，该寄存器和指定的操作数地址作与操作。这样，MASK 保证操作数地址被有效地限制在所定义的 16 位的固定范围内。该特征最小化了对筛选，循环或其他任意使用数组通过(Ay)+寻址方式来实现循环队列寻址的支持。对于有加载操作的 MAC，MASK 的值被有选择性地包含到所有存储器有效地址的运算中。

1.5 管理员编程模式

系统程序员通过管理员编程模式来应用操作系统的功能。所有影响到 ColdFire 控制特性的存取必须在管理员模式下运行。下面的章节描述只能通过特权指令存取的管理员寄存器。管理员编程模式包括了对用户可用的寄存器以及图 1-14 列出的寄存器：



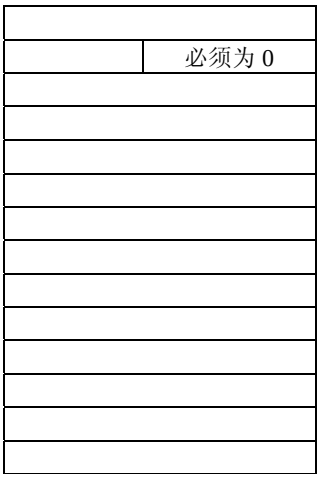


图 **1-14** 管理员编程模块

注意,不是所有的寄存器都被应用到每个 ColdFire 设备；参考表 1-6。将来的设备可能包括早期设备没有使用的寄存器。

表 **1-6** 设备使用的管理员寄存器

名称	V2	V3	V4	V5
SR	×	×	×	×
OTHER_V7	if ISA_A+	If ISA_A+	×	×
VBR	×	×	×	×
CACR	×	×	×	×
ASID			if MMU	if MMU
ACR0	×	×	×	×
ACR1	×	×	×	×
ACR2			×	×
ACR3			×	×
MMUBAR			if MMU	if MMU
ROMBAR0	DS	DS	DS	DS
ROMBAR1	DS	DS	DS	DS
RAMBAR0	DS	DS	DS	DS
RAMBAR1	DS	DS	DS	DS
MBAR	DS	DS	DS	DS

注：“×”表示管理员寄存器被应用了。DS 表示管理员寄存器是个“特殊设备”。请参考适当的设备参考手册来判断寄存器是否被使用。一些管理员寄存器只有在虚拟存储管理单元（MMU）被应用时才会使用(–if MMU‖)。一些管理员寄存器只有在指令集是 ISA_A+时才会使用(–if IAS_A+‖)。

1.5.1 状态寄存器**(SR)**

状态寄存器，如图 1-15，保存处理器状态，中断优先码和其他控制状态。管理员软件可以读写整个 SR；用户软件只能读写 SR[7-0]，见 1.1.4 节描述，“条件码寄存器”(CCR)。控制位描述处理器的状态：跟踪模式(T)，管理员或用户模式(S)和控制或中断

状态(M)。复位后，SR 被设置成 0x27*xx*。在复位之后或执行比较，Bcc,或 Scc 指令之前，SR 寄存器必须被重新加载。

15	14	13	12	11	10 — 8	7	6	5	4	3	2	1	0
系统字节						条件码(CCR)							
T	—	S	M	—	I	P[1]	—		X	N	Z	V	C

[1]P 位只使用在 V3 核心。

图 **1-15.** 状态寄存器（**SR**）

表 1-7 描述了 SR 域。

表 **1-7** 状态域描述

位	名称	描述
15	T	跟踪使能。当 T 设置后，处理器在每条指令后设置跟踪异常。
14	—	预留，应清零。
13	S	管理员/用户状态。显示处理器是管理员还是用户模式
12	M	管理/中断状态。通过中断异常清零。
11	—	预留，应清零。
10-8	I	中断优先码。定义当前中断优先级。拒绝优先级低于或等于当前优先级的中断请求，除了等级 7 的请求，该请求不可被屏蔽。
7-0	CCR	条件码寄存器(见图 1-2 和表 1-1)

1.5.2 管理员/用户堆栈指针**(A7** 和 **OTHER_A7)**

ISA_A 架构支持单个堆栈指针(A7)。A7 的初始值是从复位异常向量中加载，地址偏移是 0。所有其余的 ISA 版本支持两个独立堆栈指针(A7)寄存器:管理员堆栈指针(SSP)和用户堆栈指针(USP)。该支持在操作模式之间(管理员和用户)提供必需的独立。

这两个编程可见的 32 位寄存器的硬件应用被唯一确定为 SSP 和 USP，其中一个 32 位寄存器作为当前可用的 A7，而另一个作为 OTHER_A7。这样，寄存器内容就是处理器操作模式的一个功能，如下显示：

```
if SR[S] = 1
    then
        A7 =  管理员堆栈指针
        OTHER_A7 =  用户堆栈指针

else
        A7 =  用户堆栈指针
```

OTHER_A7 =管理员堆栈指针

1.5.3 向量基址寄存器(VBR)

向量基址寄存器包含了 1M 字节的处理器中的异常向量表的基地址序列。异常向量的偏移加到存取向量表的寄存器中。VBR[19-0]用 0 填充。

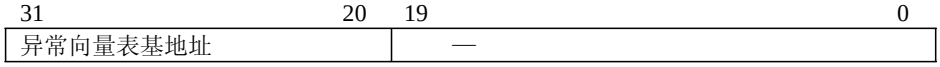


图 **1-16** 向量基址寄存器（**VBR**）

1.5.4 缓存控制寄存器(CACR)

缓存控制寄存器控制指令和数据缓存的操作，包括使能，锁存，以及使缓存内容无效的位，还包括定义默认缓存模式和写保护区域的位。在 ColdFire 处理器执行时，位的功能和位置可变化。更多信息，可参考特定的设备或核心的用户手册。

1.5.5 地址空间标示符(ASID)

32 位的地址空间标示符寄存器只有低 8 位使用。ASID 的值是由操作系统给系统中活动进程分配的 8 位标示符，有效地充当了 32 位虚拟地址的扩展。这样虚拟引用 40 位的值：8 位 ASID 连接 32 位虚拟地址。ASID 只在设备具有 MMU 时可用。更多信息，可参考特定的设备或核心的用户手册。

1.5.6 存取控制寄存器(ACR0-ACR3)

存取控制寄存器(ACR[0:3])给四个用户定义的存储区间定义属性。ACR0 和 ACR1 控制数据存储空间，ACR2 和 ACR3 控制指令存储空间。属性包括缓存模式定义，写保护和写缓冲使能。不是所有的 ColdFire 处理器都使用四个 ACR。在 ColdFire 处理器应用中，位的功能和位置可变化。更多信息，可参考特定的设备或核心的用户手册。

1.5.7 MMUC 基址寄存器(MMUBAR)

MMUBAR，如图 1-17，定义了一个存储器映射的，仅容特权数据的空间，这个空间在内部数据总线的有效地址运算中具有最高的优先权（也即 MMUBAR 优先级比 RAMBAR0 高）。如果虚拟模式激活，任何不涉及 MMUBAR、RAMBAR、ROMBAR 和 ACR 的正常模式存取都被认为正常模式，从 MMU 可以请求虚拟地址和产生它的存取属性。MMUBAR 只在设备有 MMU 时可用。更多信息，可参考特定的设备或核心的用户手册。

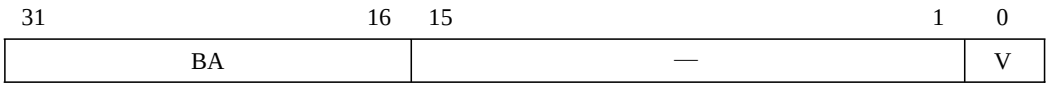


图 **1-17 MMU** 基址寄存器

表 1-8 描述了 MMU 基址寄存器域。

表 **1-8 MMU** 基址寄存器域描述

位	名称	描述
31-16	BA	基地址。为映射到 MMU 的 64K 地址空间定义基地址。
15-1	—	预留，应清零。
0	V	合法

1.5.8RAM 基地址寄存器(RAMBAR0/RAMBAR1)

RAM 基地址寄存器决定内部 SRAM 模块的基地址以及引用类型的映射。每个 RAMBAR 包括一个基地址，写保护位，地址空间掩码位以及一个使能位。RAM 基地址序列具有特殊的作用。特定的 ColdFire 处理器可能使用 2，1 或 0 个 RAMBAR。在 ColdFire 处理器执行时，位的功能和位置可变化。更多信息，可参考特定的设备或核心的用户手册。

1.5.9 ROM 基地址寄存器(ROMBAR0/ROMBAR1)

ROMBAR 寄存器决定内部 ROM 模块的基地址以及引用类型的映射。每个 ROMBAR 包括一个基地址，写保护位，地址空间掩码位以及一个使能位。ROM 基地址序列具有特殊的使用。特定的 ColdFire 处理器可能使用 2，1 或 0 个 ROMBAR。在 ColdFire 处理器应用中，位的功能和位置可能变化。更多信息，可参考特定的设备或核心的用户手册。

1.5.10 模块基地址寄存器(MBAR)

管理员级的模块基地址寄存器，如图 1-18，详细介绍了基地址和可用的内部外设的存取类型。MBAR 可以通过一个作为读写寄存器的调试模块来读写；只有调试模块可以读 MBAR。所有内部外设占用一个大小为 4K 的可重定位地址存储块。MBAR 用地址空间域来描述特定的地址空间。更多信息，可参考特定的设备或核心的用户手册。

——————属性掩码位——————

31 12	11 9	8	7	6	5	4	3	2	1	0
BA	—	WP	—	AM	C/I	SC	SD	UC	UD	V

图 **1-18** 模块基地址寄存器（**MBAR**）

表 1-9 描述了 MBAR 域。

表 **1-9 MBAR** 域描述

位	名称	描述	
31-12	BA	基地址。为映射到 MMU 的 4K 地址空间定义基地址。	
11-9	—	预留，应清零。	
8-1	AMB	属性位	
8		WP	写保护，MBAR 映射寄存器地址范围中的写周期标示位。

7		—	预留，应清零。
6		AM	改变控制码。
5		C/I	标示 CPU 空间和中断确认周期。
4		SC	设置 MBAR 地址范围中的管理员代码空间
3		SD	设置 MBAR 地址范围中的管理员数据空间
2		UC	设置 MBAR 地址范围中的用户代码空间
1		UD	设置 MBAR 地址范围中的用户数据空间
0	V	合法性，决定 MBAR 设置是否合法。	

1.6 整数数据格式

整数单元支持操作数格式，如表 1-10。整数单元操作数可以驻留在寄存器，存储器或指令中。每条指令的操作数长度可明确地在指令中编码或隐式地由指令操作定义。

表 **1-10** 整数数据格式

操作数格式	字长
位	1 位
整数字节	8 位
整数字	16 位
整数长字	32 位

1.7 浮点型数据格式

本节描述了可选的 FPU 的操作数数据格式。FPU 支持三种带符号的整数格式（字节，字和长字），这些和整数单元相同。FPU 还支持单精度和双精度二进制浮点型数格式，满足 IEEE-754 的要求。

1.7.1 浮点型数据类型

每种浮点型数据格式支持 5 种不同的数据类型：规格化数据，零，无穷大，非数值和非规格化的数据。规格化的数据类型，如图 1-19，给定的格式从不使用最大或最小的指数值。

1.7.1.1 规格化数据 规格化数据包括所有指数在最大值和最小值之间的所有正数和负数。对于单精度和

双精度的规格化数据，整数位数为 1，指数可为 0。

图 **1-19** 规格化数据格式

1.7.1.2 零

零可正可负，也可代表实际的值，+0.0 和-0.0。见图 1-20。

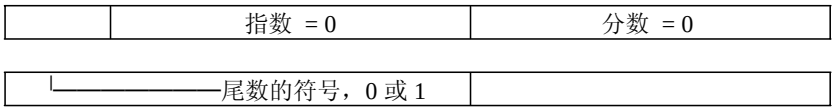


图 **1-20** 零格式

1.7.1.3 无穷大

无穷大可正可负，代表超出了溢出范围的实际值。结果的指数大于或等于最大指数值表明数据格式和操作的溢出。这个溢出忽略了凑整和用户可选凑整模式的影响。对于单精度和双精度的无穷大，分数是 0。见图 1-21。

图 **1-21** 无穷大格式

1.7.1.4 非数值

NAN 如果是 FPU 产生，代表操作的结果没有数学意义，如无穷大除以无穷大。以 NAN 作为输入的操作返回一个 NAN 结果。用户创建的 NAN 可以防止没有初始化的变量和数组或者代表用户自定义的数据类型。见图 1-22。

图 **1-22** 非数值格式

如果操作的输入是 NAN，则结果 FPU 产生的默认的 NAN。如果 FPU 产生 NAN，则 NAN 总是尾数位包括同样的格式：所有的尾数位为 1，符号位为 0。用户产生一个 NAN，所有非 0 的位可以存储到尾数和标志位中。

1.7.1.5 非规格化的数值 非规格化的数值代表接近下溢范围的实数。非规格化的数值可正可负。对于单精度

和双精度的非规格化的数值，隐式的整数位为 0。见图 1-23。

图 **1-23** 非规格化的数据格式

通常，下溢的检测导致浮点型数系统直接变成 0。IEEE-754 标准利用渐进的下溢：结果的尾数右移（非规格化），而指数增长直到最小值。如果所有尾数全部右移结束，结果就变成 0。

在硬件应用中非规格化数据并不被直接支持，如果需要可在软件中操作(可编写输入非规格化数据的异常处理程序来处理非规格化数据，处理下溢的异常处理程序可产生非规格化数据)。如果输入异常处理程序无效，则所有非规格化的数据被处理为零。

1.7.2FPU 数据格式和类型概述

表 1-11 简述了对于字节，字，长字，单精度和双精度数据格式的数据类型。

表 **1-11** 实数格式概述

参　数	单精度	双精度
数据格式	31 30 23 22 0 s e f	63 62 52 51 0 s e f
值域长度		
符号(s)	1	1
有偏指数（e）	8	11
分数(f)	23	52
总和	32	64
	符号的解释	
正分数	S=0	S=0
负分数	S=1	S=1
	规格化数值	
有偏指数的偏差	+127（0x7F）	+1023(0x3FF)
有偏指数的范围	0<e<255(0xFF)	0<e<2047(0x7FF)
分数的范围	零或非零	零或非零
尾数	1.f	1.f
实数表示法的关系	$(-1)^s \times 2^{e-127} \times 1.f$	$(-1)^s \times 2^{e-1023} \times 1.f$
有偏指数格式的最小值	非规格化数值 0(0x00)	0(0x000) 有
偏指数的偏差	+126(0x7E)	+1022(0x3FE)
分数的偏差	非零	非零

尾数	0.f	0.f
实数表示法的关系	$(-1)^s \times 2^{-126} \times 0.f$	$(-1)^s \times 2^{-1022} \times 0.f$
有符号 **0**		
有偏指数格式的最小值	0(0x00)	0(0x00)
尾数	0.f=0.0	0.f=0.0
有符号无穷大		
有偏指数格式的最大值	255(0xFF)	2047(0x7FF)
尾数	0.f=0.0	0.f=0.0
非数值		
符号	任意	0 或 1
有偏指数格式的最大值	255(0xFF)	255(0x7FF)
分数	非零	非零
分数的表示法(非零位格式由用户产生，分数由 FPU 产生)	xxxxx,, xxxx 11111,, 1111	xxxxx,, xxxx 11111,, 1111
近似范围		
规格化最大正数	3.4×10^{38}	1.8×10^{308}
规格化最小正数	1.2×10^{-38}	2.2×10^{-308}
非规格化最小正数	1.4×10^{-45}	4.9×10^{-324}

1.8 乘法累加器数据格式

MAC 和 EMAC 支持下面格式的 16-或 32-位的操作数输入：

• 带符号整数补码：这种格式，N 位的操作数值落在 $-2^{(N-1)} \le operand \le 2^{(N-1)}$ 。二进制小数点落在最低位的右边。

• 无符号整数：这种格式，N 位的操作数值落在 $0 \le operand \le 2^{N-1}$ 。二进制小数点落在最低位的右边。

• 补码，符号部分：N 位数值，第一位是符号位。剩余位即为二进制小数点后的 N-1 位。如 $a_{N-1}a_{N-2}a_{N-3}\ldots a_2a_1a_0$ ，其值在由图 1-24 给出：

$$value = -(1 \bullet a_{N-1}) + \sum_{i=0}^{N-2} 2^{(i+1-N)} \bullet ai$$

图 1-24 补码，有符号部分值的等式 这个格式可以表示范围在 $-1 \le operand \le 1-2^{(N-1)}$ 的数据。 对于字和长字，最大的负数可以表示为-1，内部分别表示成 0x8000 和 0x80000000。

最大的正数字是 0x7FFF 或（$1-2^{-15}$）；最大的正数长字是 0x7FFFFFFF 或（$1-2^{-31}$）。

1.9 寄存器的数据组织

本节描述了数据寄存器，地址寄存器和控制寄存器中的数据组织。

1.9.1 寄存器中的整数数据格式的组织

每个整数数据寄存器是 32 位。字节和字操作数分别占据了整数数据寄存器的低 8 位和低 16 位。长字占据了整个数据寄存器。数据寄存器保存的可能是源操作数或目的操作数，仅使用或改变其中的低 8 位或 16 位（分别在字节或字的操作），剩余的高位部分不变也不被使用。一个整型长字的最低位的地址是 0，最高位是 31。图 1-25 显示了数据寄存器中数据的组织形式。

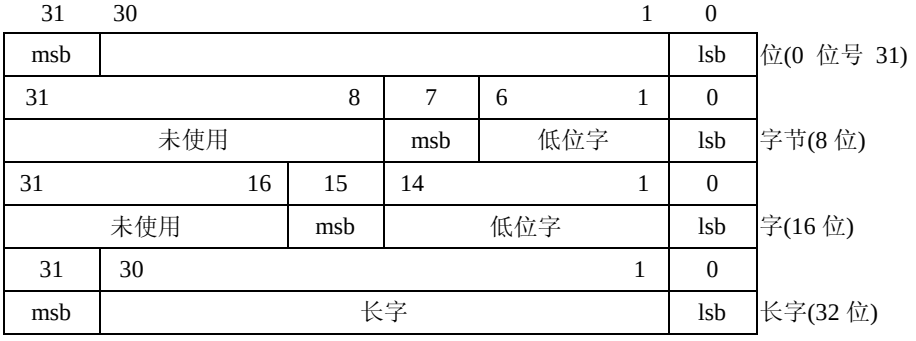


图 **1-25** 数据寄存器中整型数据的组织形式

由于地址寄存器和堆栈指针是 32 位，所以地址寄存器不可以被用作字节操作数。当地址寄存器是源操作数，则根据操作的长度来决定使用低位的字还是整个长字操作数。当地址寄存器是目的寄存器时，不管操作数的长度是多少，整个寄存器都受影响。如果源操作数是字，则先把符号扩展成 32 位，然后被应用到目的地址寄存器中的操作。地址寄存器主要用于地址和地址的运算。指令集解释如何去加，比较，以及如何赋值给地址寄存器。图 1-26 显示了地址寄存器中地址的组织形式。

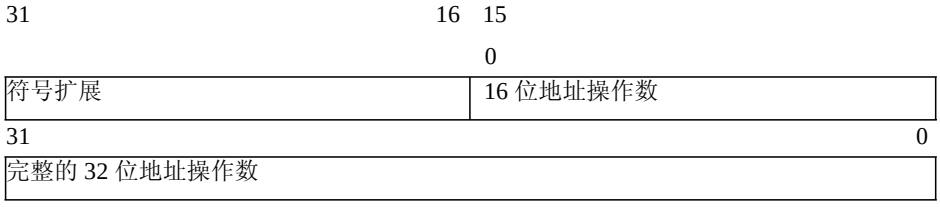


图 **1-26** 地址寄存器中地址的组织形式

控制寄存器根据功能来变化长度。Freescale 在一些控制寄存器为以后的定义预留了未定义的位。这些特殊的位为了以后的兼容读写的值都保持 0。

所有到状态寄存器和条件码寄存器的操作都是一个字长的操作。对于条件码寄存器的所有操作，高字节读到的值为 0 写入时被忽略，包括特权模式在内。只写指令 MOVEC 向系统控制寄存器中写入（VBR，CACR，等）。

1.9.3 存储器中整数数据格式的组织 对于存储器中字节编址的组织形式允许低地址对应高字节。长字数据单元的地址 N 对应最高字中的最高字节的地址。低位的字地址为 N+2，最低位为 N+3（见图 1-27）。最低地址（最近的 0x00000000）是最高字节的地址，连续的最低字节的安排下面紧接的位置（N+1，N+2，等）。最高地址（0xFFFFFFFF）是最低字节的地址。

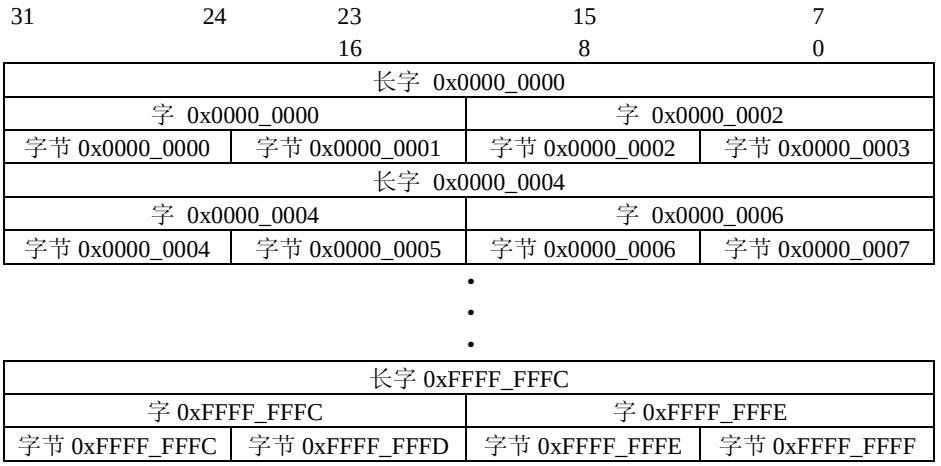
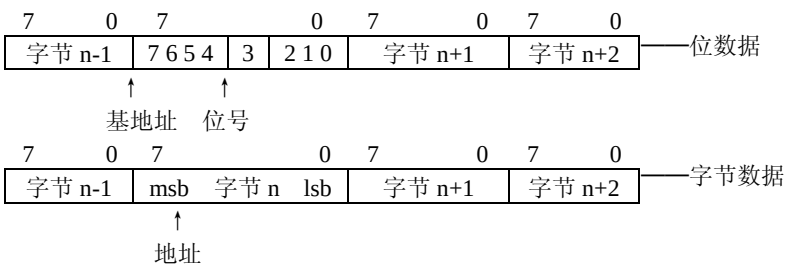


图 **1-27** 存储器的寻址方式

图 1-28 显示了存储器中的数据组织方式。基址寄存器从存储器中选择 1 个字节、1 基地址字节、在基地址字节中选择的位操作数的位号。最高位是 7。



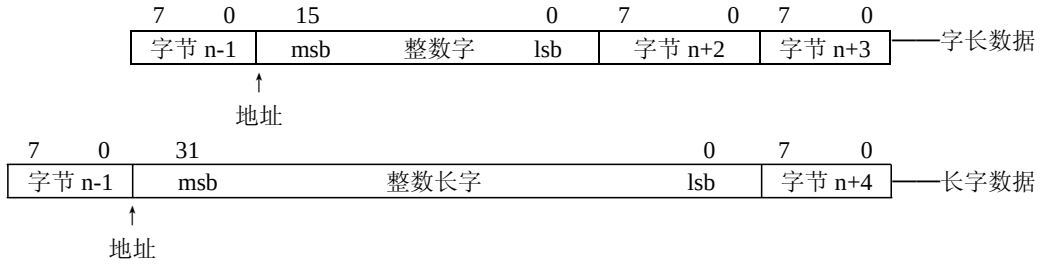


图 **1-28** 存储器整型操作数的组织形式

1.10 硬件配置信息

ColdFire 硬件配置信息在系统复位后被加载到 D0 和 D1 通用寄存器。硬件配置信息在复位信号取消后立即被加载，因为这样允许竞争者通过 BDM 读出这些寄存器中的内容，从而确定硬件配置信息。这个功能在早期的 V2 和 V3 中并不支持，从那以后的所有 ColdFire 核心都包含这个功能。D0 提供了处理器配置，D1 提供了本地存储器配置。这两个寄存器中的值在复位处理最开始就被存储到存储器中，以便存储器的配置信息随后就可以被检查。

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	1	1	0	0	1	1	1	1	VERSION[3:0]				PREV[3:0]			
W																

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MAC	DIV	EMAC	FPU	MMU	0	0	L2CC	ISA_REV[3:0]				DBG_REV[3:0]			
W																

= 未使用或预留

图 **1-29 D0** 处理器配置信息

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	CLSZ[1:0]		ICAS[1:0]		ICSZ[3:0]				SRAM0SZ[3:0]				ROM0SZ[3:0]			
W																

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	MBSZ[1:0]		DCAS[1:0]		DCSZ[3:0]				SRAM1SZ[3:0]				ROM1SZ[3:0]			
W																

=未使用或预留

图 **1-30 D1** 处理器配置信息

表 **1-12 D0** 处理器配置域描述

名称	描述	值
VERSION [3：0]	核心版本	这四位定义了 ColdFire 核心的硬件微架构（版本） 版本=0b0010，ColdFire 版本 2 版本=0b0011，ColdFire 版本 3 版本=0b0100，ColdFire 版本 4 版本=0b0101，ColdFire 版本 5 所有其他的值都预留以后使用。 D0 的高 12 为复位值直接描述了 ColdFire 核心版本，如“CF2”指 V2，“CF3”指 V3 等。
PREV [3：0]	处理器版本	这四位定义了处理器硬件版本号。默认为 0b0000。
MAC	MAC 介绍	如果处理器中有 MAC，则该位设置。 0 = 没有 MAC 1 = 有 MAC 如果有 EMAC，则该位清零。
DIV	除法器介绍	如果处理器中有 DIV，则该位设置。一些早期的 V2 核心，如 MCF5202，MCF5204，MCF5206 不包括整数除法操作的硬件支持。 0 = 没有 DIV 1 = 有 DIV
EMAC	EMAC 介绍	如果处理器中有 EMAC，则该位设置。 0 = 没有 EMAC 1 = 有 EMAC 如果有 MAC，则该位清零。
FPU	FPU 介绍	如果处理器中有 FPU，则该位设置。 0 = 没有 FPU 1 = 有 FPU
MMU	MMU 介绍	如果处理器中有 MMU，则该位设置。 0 = 没有 MMU 1 = 有 MMU

L2CC	二级缓存控制器介绍	如果处理器中有 L2CC，则该位设置。 0 = 没有 L2CC 1 = 有 L2CC
ISA_REV [3：0]	ISA 版本	这四位定义了 ColdFire 处理器核心的指令集架构版本。 ISA_REV=0b0000，ISA_A ISA_REV =0b0001，ISA_A+ ISA_REV =0b0010，ISA_B ISA_REV =0b0011，ISA_C 所有 其他的值都预留以后使用。
DBG_REV [3：0]	调试模块版本	这四位定义了 ColdFire 处理器核心的调试模块版本。 DBG_REV =0b0000，Debug_A DBG_REV =0b1000，Debug_A+ DBG_REV =0b0001，Debug_B DBG_REV =0b1001，Debug_B+ DBG_REV =0b0010，Debug_C DBG_REV =0b0011，Debug_D DBG_REV =0b0100，Debug_E 所有其他的值都预留以后使用。

表 **1-13 D1** 处理器配置域描述

名称	描述	值
CLSZ[1：0]	缓存线长度	这两位定义了缓存线长度 CLSZ = 0b00，16 字节缓存线长度 CLSZ = 0b01，32 字节缓存线长度 所有其他的值都预留以后使用。
ICAS[1：0]	指令缓存组合	这两位定义了指令缓存组合 ICAS = 0b00，指令缓存由四路组合而成 ICAS = 0b01，指令缓存直接映射 所有 其他的值都预留以后使用。
ICSZ[3：0]	指令缓存大小	这四位定义了指令缓存大小 ICSZ = 0b0000，没有指令缓存 ICSZ = 0b0001，指令缓存 512 字节 ICSZ = 0b0010，指令缓存 1K 字节 ICSZ = 0b0011，指令缓存 2K 字节 ICSZ = 0b0100，指令缓存 4K 字节 ICSZ = 0b0101，指令缓存 8K 字节 ICSZ = 0b0110，指令缓存 16K 字节 ICSZ = 0b0111，指令缓存 32 K 字节 ICSZ = 0b1000，指令缓存 64K 字节 所有其他的值都预留以后使用。
SRAM0SZ [3：0]	SRAM0 大小	这四位定义了 SRAM0 大小 SRAM0SZ = 0b0000，没有 SRAM0 SRAM0SZ = 0b0001，SRAM0 512 字节 SRAM0SZ = 0b0010，SRAM0 1K 字节 SRAM0SZ = 0b0011，SRAM0 2K 字节 SRAM0SZ = 0b0100，SRAM0 4K 字节

		SRAM0SZ = 0b0101，SRAM0 8K 字节 SRAM0SZ = 0b0110，SRAM0 16K 字节 SRAM0SZ = 0b0111，SRAM0 32 K 字节 SRAM0SZ = 0b1000，SRAM0 64K 字节 SRAM0SZ = 0b1001，SRAM0 128K 字节 所有其他的值都预留以后使用。
ROM0SZ [3：0]	ROM0 大小	这四位定义了 ROM0 大小 ROM0= 0b0000，没有 ROM0 ROM0= 0b0001，ROM0 512 字节 ROM0= 0b0010，ROM0 1K 字节 ROM0= 0b0011，ROM0 2K 字节 ROM0= 0b0100，ROM0 4K 字节 ROM0= 0b0101，ROM0 8K 字节 ROM0= 0b0110，ROM0 16K 字节 ROM0= 0b0111，ROM0 32 K 字节 ROM0= 0b1000，ROM0 64K 字节 ROM0= 0b1001，ROM0 128K 字节 所有其他的值都预留以后使用。
MBSZ[1：0]	Mbus 大小	这两位定义了 ColdFire 控制总线宽度 MBSZ = 0b00,32 位系统总线宽度 MBSZ = 0b01,64 位系统总线宽度 所有其他的值都预留以后使用。
DCAS[1：0]	数据缓存组合	这两位定义了数据缓存组合 MBSZ = 0b00, 数据缓存由四路组合而成 MBSZ = 0b01, 数据缓存直接映射 所有 其他的值都预留以后使用。
DCSZ[3：0]	数据缓存大小	这四位定义了数据缓存大小 DCSZ = 0b0000，没有数据缓存 DCSZ = 0b0001，数据缓存 512 字节 DCSZ = 0b0010，数据缓存 1K 字节 DICSZ = 0b0011，数据缓存 2K 字节 DCSZ = 0b0100，数据缓存 4K 字节 DCSZ = 0b0101，数据缓存 8K 字节 DCSZ = 0b0110，数据缓存 16K 字节 DCSZ = 0b0111，数据缓存 32 K 字节 DCSZ = 0b1000，数据缓存 64K 字节 所有其他的值都预留以后使用。
SRAM1SZ [3：0]	SRAM1 大小	这四位定义了 SRAM1 大小 SRAM1SZ = 0b0000，没有 SRAM1 SRAM1SZ = 0b0001，SRAM1 512 字节 SRAM1SZ = 0b0010，SRAM1 1K 字节 SRAM1SZ = 0b0011，SRAM1 2K 字节 SRAM1SZ = 0b0100，SRAM1 4K 字节 SRAM1SZ = 0b0101，SRAM1 8K 字节 SRAM1SZ = 0b0110，SRAM1 16K 字节 SRAM1SZ = 0b0111，SRAM1 32 K 字节 SRAM1SZ = 0b1000，SRAM1 64K 字节 SRAM1SZ = 0b1001，SRAM1 128K 字节 所有其他的值都预留以后使用。

ROM1SZ [3：0]	ROM1 大小	这四位定义了 ROM1 大小 ROM1= 0b0000，没有 ROM1 ROM1= 0b0001，ROM1 512 字节 ROM1= 0b0010，ROM1 1K 字节 ROM1= 0b0011，ROM1 2K 字节 ROM1= 0b0100，ROM1 4K 字节 ROM1= 0b0101，ROM1 8K 字节 ROM1= 0b0110，ROM1 16K 字节 ROM1= 0b0111，ROM1 32 K 字节 ROM1= 0b1000，ROM1 64K 字节 ROM1= 0b1001，ROM1 128K 字节 所有其他的值都预留以后使用。

注：如果处理器使用了二级缓存，其存储空间可通过读 L2 _CACR（二级缓存控制寄存器）使用。

历史版本

内容随文档版本变化

版本号. 发行日期	描述	页码
Rev3 7-Mar-05	通用发行：各个章节以 ISA_x 架构命名，而不是 V2，V3，V4 （以前用来说明“ColdFire 版本 2”等）。在 ISA_x 上增强的指令在所有 ColdFire 版本后有注解。	全书

第 2 章 寻址方式

绝大部指令操作是针对源和目的操作数的计算，并将其结果存放在目的位置。单操作数指令只计算一个目的操作数，并将它存放在目的位置。与存储器紧密相关的微处理器外扩模块，主要有与程序空间有关的程序部件和与数据空间有关的数据部件。它们访问指令字段或者指令的操作数（数据项）。程序空间，是内存中包含程序指令代码及其驻留在指令流中的立即操作数的存储区域。而数据空间则是包含程序数据的内存区域。与程序计数器相对寻址方式，可依据数据引用方式来分类。

2.1 指令格式

ColdFire 系列指令由 1 至 3 个字构成。图 2-1 表示了指令的一般组成。指令的第一个字，叫做操作字段，用来指定该指令的长度、有效寻址方式及要执行的操作。余下的字段则用于具体定义指令及其相关操作。这些字段，可以有条件断言、立即操作数、扩展的有效寻址方式、条件转移、位号或特殊规格的寄存器、陷阱操作、条件计数（argument counts）或浮点型指令。ColdFire 系列指令的字长只限于这 3 种尺寸：16、32 或 48 位。

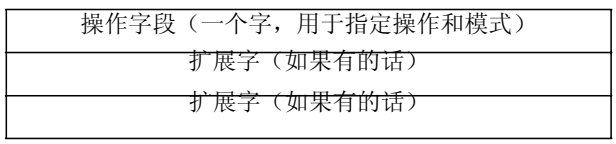


图 **2-1** 指令字的一般格式

每条指令通过操作码来确定其功能及其各操作数的位置。操作字段的形式是基本的指令字（见图 2-2）。模式域编码用于决定寻址方式。寄存器域包含通用的寄存器数字或一个当模式域=111 时用以选择寻址方式的数值。某些变址（Index）或间接寻址，会用到被扩展字段跟随的操作字段的组合。图 2-2 说明了指令字段中使用的两种格式，表 2-1 列出它们的域定义。

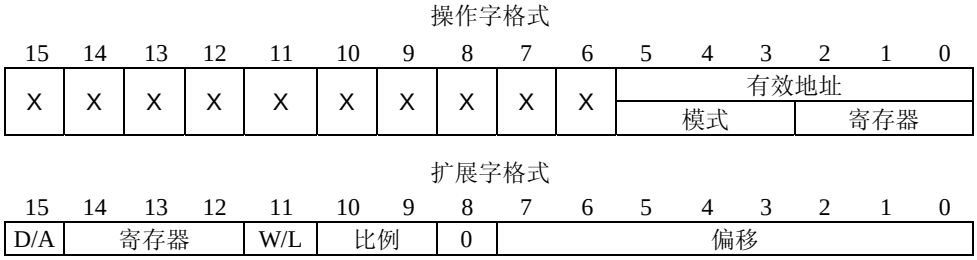


图 **2-2** 指令字的具体格式

表 2-1 定义了指令字段的格式。

表 **2-1** 指令字格式领域定义

域	定义
指 令	
模式	寻址方式(见表 2-3)
寄存器	通用寄存器数字(见表 2-3)
扩展部分	
D/A	变址寄存器类型 0 = Dn 1 = An
W/L	字/长字的变址长度 0 = 地址错误异常 1 = 长字
比例	比例因子 00 = 1 01 = 2 10 = 4 11 = 8（仅使用 FPU 时用到）

2.2 有效寻址方式

除了操作码规定指令执行时的功能外，指令还指明实现此功能的各个操作数的位置。指示通过以下 3 个方式来指定操作数的位置：

⑴ 用一个寄存器域，使得指令可以指定哪个寄存器被使用。

⑵ 用一个指令有效地址域去包含寻址方式信息。

⑶ 指令的定义也暗示着要使用特殊的寄存器。其他的指令域用于指定被选寄存器是否为地址或数据寄存器，以及这些寄存器是如何被使用的。

指令的寻址方式用来指定操作数的值、含有操作数的寄存器或者如何获得内存中操作数有效地址。每种寻址方式都有一个汇编语言的语法。某些指令可以暗含操作数的寻址方式，这些指令含有只使用一种寻址方式的操作数的域。

2.2.1 数据寄存器直接寻址方式 在数据寄存器直接寻址方式下，有效地址指明包含操作数的数据寄存器。

地址形成	EA = An
汇编格式	Dn
EA 模式域	000
EA 寄存器域	寄存器号
扩展字数	0

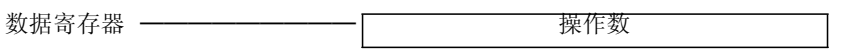


图 **2-3** 数据寄存器直接寻址

2.2.2 地址寄存器直接寻址方式 在地址寄存器直接寻址方式下，有效地址指明包含操作数的地址寄存器。

地址形成	EA = An
汇编格式	An
EA 模式域	001
EA 寄存器域	寄存器号
扩展字数	0

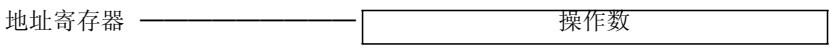


图 **2-4** 地址寄存器直接寻址

2.2.3 地址寄存器间接寻址方式 在地址寄存器间接寻接方式下，操作数存在于内存中。有效地址指明由地址寄存器包含内存中操作数的地址。

地址形成	EA = (An)
汇编格式	(An)
EA 模式域	010
EA 寄存器域	寄存器号
扩展字数	0

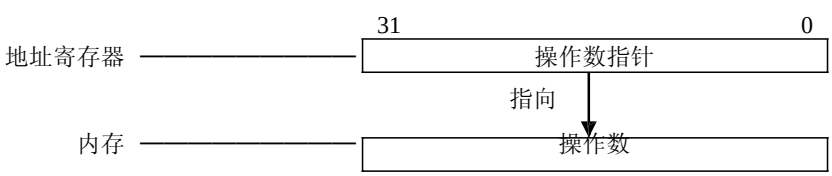


图 **2-5** 地址寄存器间接寻址

2.2.4 带后自增的地址寄存器间接寻址 在带后自增的地址寄存器间接寻址方式下，操作数存在于内存中。有效地址指定地址寄存器包含内存中操作数的地址。操作数地址被使用后，它会根据操作数长度（即分别为字节、字、长字）自动加 1、2 或 4。注意，堆栈指针（A7）被完全像任何其他地址寄存器那样对待。

地址形成	EA = (An); An = An + Size(尺寸)
汇编格式	(An)+
EA 模式域	011
EA 寄存器域	寄存器号

扩展字数	0	31	0

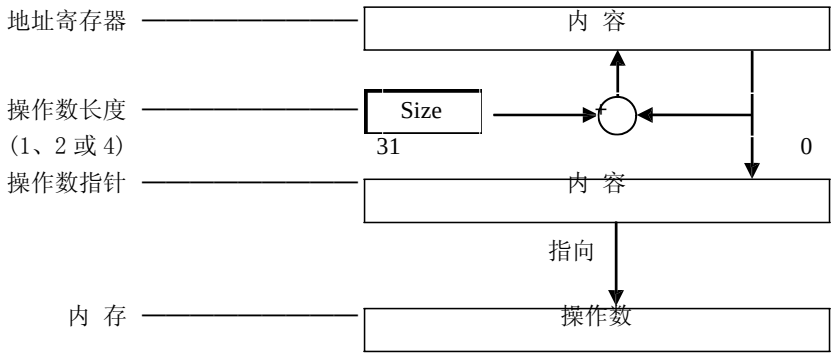


图 **2-6** 带后自增的地址寄存器间接寻址

2.2.5 带前自减的地址寄存器间接寻址

在带前自减的地址寄存器间接寻址方式下，操作数存在于内存中。有效地址指定地址寄存器包含操作数在内存中的地址。在操作数地址被使用前，它会根据操作数长度（即分别为字节、字、长字）自动加 1、2 或 4。注意，堆栈指针（A7）被完全像任何其他地址寄存器那样对待。

地址形成	EA = (An) – Size; An = An – Size
汇编格式	– (An)
EA 模式域	100
EA 寄存器域	寄存器号
扩展字数	0

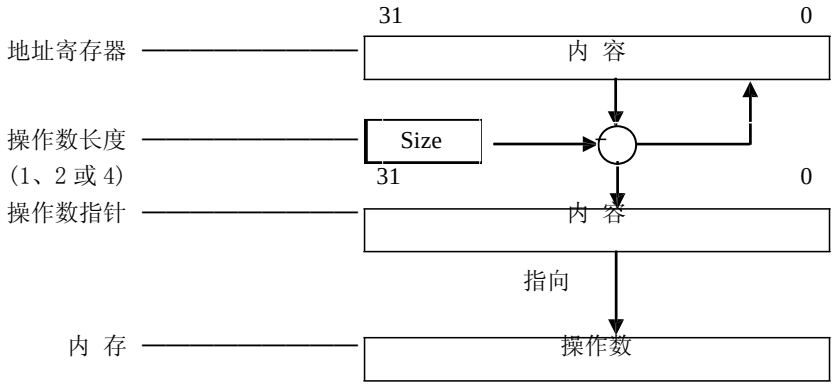


图 **2-7** 带前自减的地址寄存器间接寻址

2.2.6 带偏移的地址寄存器间接寻址 在带偏移的地址寄存器间接寻址方式下，操作数存在于内存中。内存中的操作地址

由地址寄存器中的地址（由有效地址指定）与在扩展字段中的有符号 16 位整数偏移量之和组成，偏移量在被用于有效地址计算之前，总是先被扩展成有符号 32 位。

地址形成	EA = (An) – + d_{16}
汇编格式	– (An)
EA 模式域	101
EA 寄存器域	寄存器号
扩展字数	1

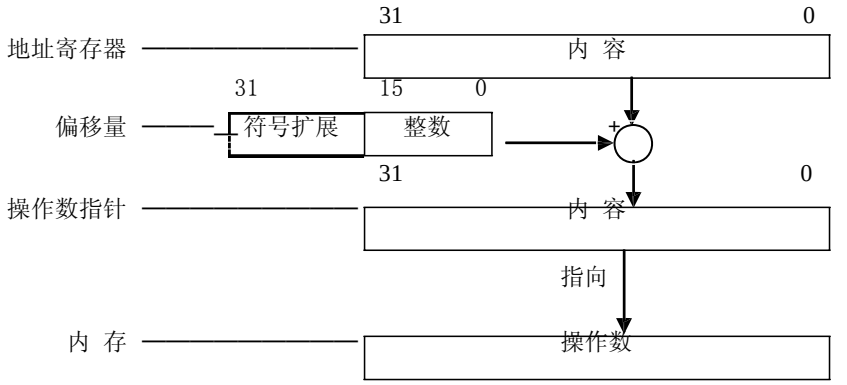


图 **2-8** 带前自减的地址寄存器间接寻址

2.2.7 带 **8** 位偏移的比例变址（**Scaled Index**）地址寄存器间接寻址

这种寻址方式，需要一个包含变址寄存器指示器的扩展字、可能的比例和一个 8 位偏移量。该变址寄存器指示器包括大小和比例的信息。在这种方式下，操作数存在于内存。该操作数的地址是以下三者之和：地址寄存器的内容、存在于扩展字低 8 位的有符号扩展偏移量和比例变址寄存器的有符号扩展内容。在该方式下，用户必须指定地址寄存器、偏移量、比例因子和变址寄存器。

地址形成	EA = (An) + ((Xi)*比例因子) + 有符号扩展 d_8
汇编格式	(d_8,An,Xi, 尺 寸 * 比 例)
EA 模式域	110
EA 寄存器域	寄存器号
扩展字数	1

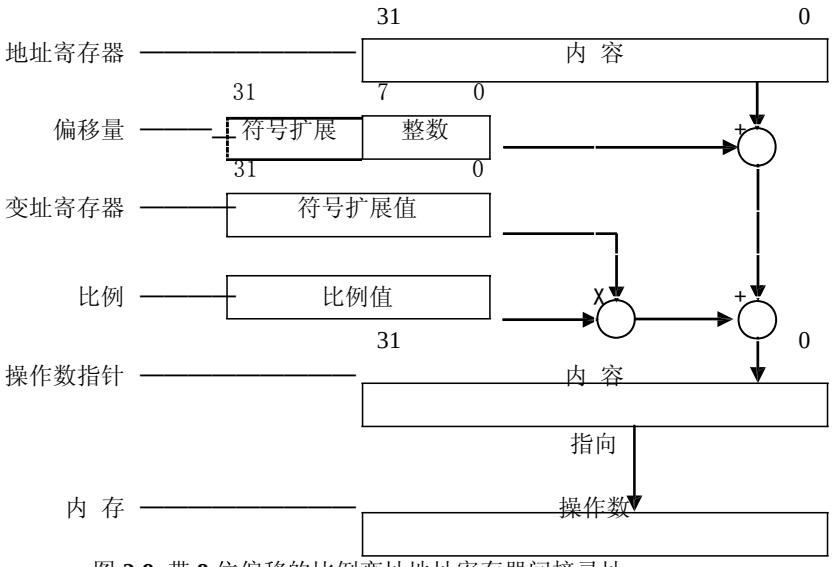


图 **2-9** 带 **8** 位偏移的比例变址地址寄存器间接寻址

2.2.8 带偏移的程序计数器间接寻址

在这种方式下，操作数存在于内存。其操作数的地址是以下两者之和：程序计数器中的地址（PC）和存在于扩展字中的 16 位有符号整型偏移量。PC 的值在地址形成后将变成 PC+2，这时的 PC 就是指令操作字段的地址。这是一个只读的程序部件。

地址形成	EA = (PC) + d_{16}
汇编格式	(d_{16}, An)
EA 模式域	111
EA 寄存器域	010
扩展字数	1

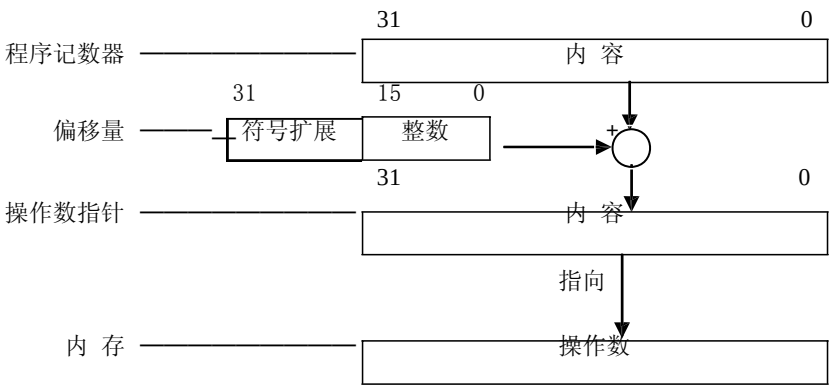


图 **2-10** 带偏移的程序计数器间接寻址

2.2.9 带 8 位偏移的比例变址（Scaled Index）程序计数器间接寻址

除了把 PC 作为基址寄存器外，该方式等同于 2.2.7 节的寻址方式：“带 8 位偏移的比例变址地址寄存器间接寻址方式”。在这种方式下，操作数存在于内存。该操作数的地址是以下各项之和：程序计数器中的地址、存在于扩展字低 8 位的有符号扩展偏移量和有特定尺寸与比例的有符号扩展变址操作数。PC 的值在地址形成后将变成 PC+2，这时的 PC 就是指令操作字段的地址。这是一个只读的程序部件。当使用该方式时，用户必须指定偏移量、比例和变址寄存器。

地址形成	EA = (PC) + ((Xi)*比例因子) + 有符号扩展 d_8
汇编格式	(d_8,PC,Xi, 尺 寸 * 比例)
EA 模式域	111
EA 寄存器域	011
扩展字数	1

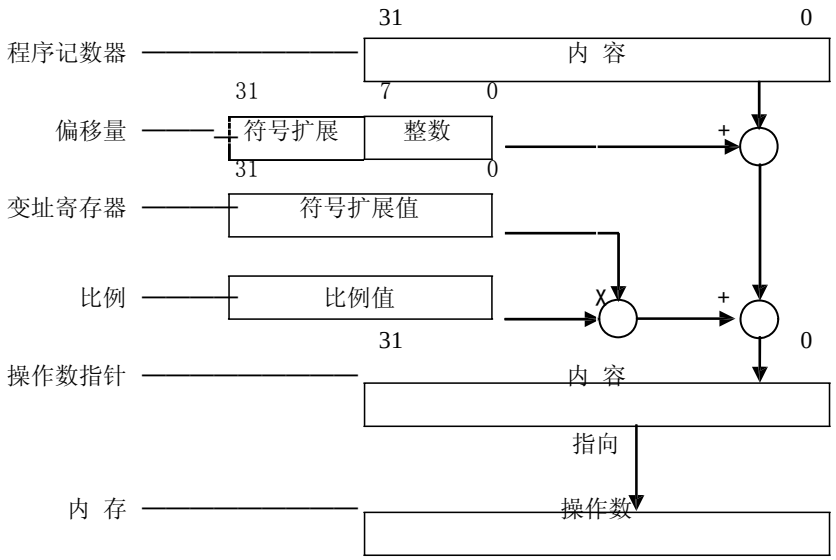


图 **2-11** 带 **8** 位偏移的比例变址程序计数器间接寻址

2.2.10 绝对的短地址寻址

在该寻址方式下，操作数存在于内存，而该操作数地址则存在于扩展字段中。16 位地址在使用之前会先被扩展成有符号 32 位地址。

地址形成	给定的 EA
汇编格式	(xxx).W
EA 模式域	111
EA 寄存器域	000

扩展字数 1

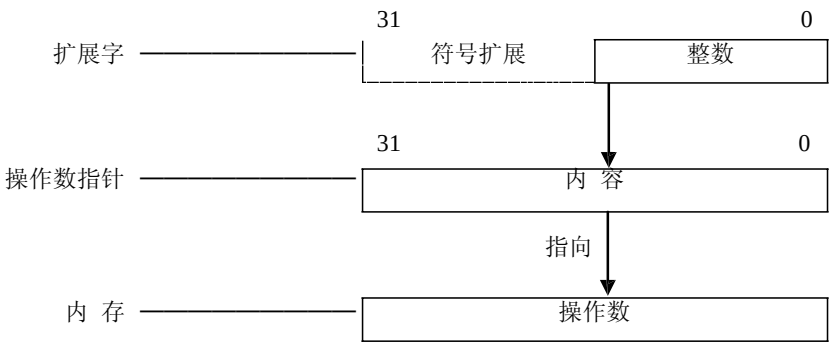


图 **2-12** 绝对的短地址寻址

2.2.11 绝对的长地址寻址 在该寻址方式下，操作数存在于内存，而该操作数地址占用内存中指令操作字段后
的两个扩展字。第一个字包含了地址的高序部分，而第二个字包含了地址的低序部分。

地址形成	给定的 EA
汇编格式	(xxx).W
EA 模式域	111
EA 寄存器域	000
扩展字数	1

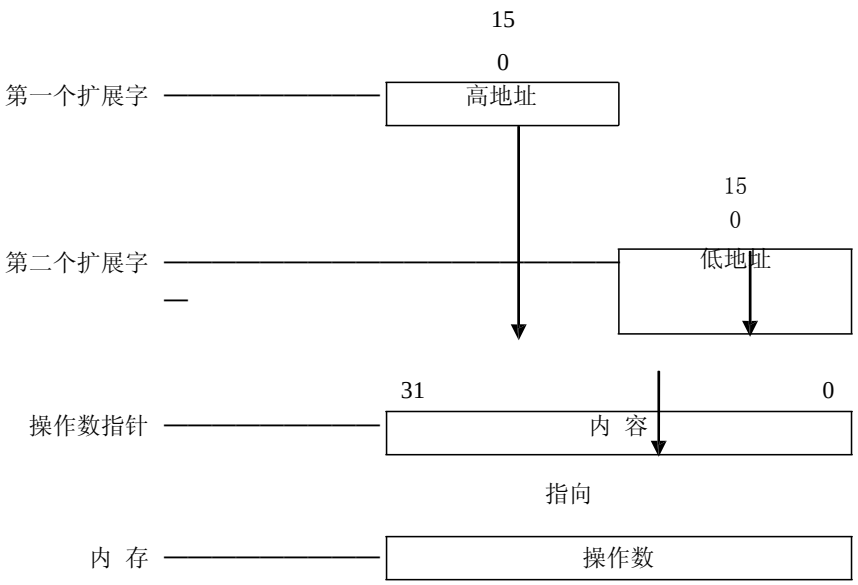


图 **2-12** 绝对的短地址寻址

2.2.12 立即数据

在该寻址方式下，操作数存在于 1 或 2 个扩展字中。表 2-2 列出了指令字格式中操作字的定位。立即数据的格式如下：

表 **2-2.** 立即操作数定位

操作数长度	位 置
字节	扩展字的低字节
字	整个扩展字
长字	操作数的高字在第一个扩展字，低字在第二个扩展字中。

产生	给定操作数
汇编语法	#<xxx>
EA 模式域	111
EA 寄存器域	100
扩展字数目	1 或 2

图 **2-14** 立即数寻址方式

2.2.13 有效寻址方式小结 有效寻址方式根据其用途来分组。数据型寻址方式用于数据操作数；存储器型寻址方式用于存储器操作数；可变型寻址方式用于可变（可写）的操作数；而控制型寻址方式用于没有确定尺寸的存储器操作数。

这些类型有时会被结合起来使用，以形成更多已被限定的新类型。两个典型的合并类型是可变的存储器（可变型和存储器型结合的寻址方式）和可变的数据（可变型和数据型结合的寻址方式）。表 2-3 列举出了各个有效寻址方式以及它们的类型的要素。

表 **2-3** 有效寻址方式和种类

寻址方式	语法	模式域	寄存器域	数据	存储器	控制	是否可变
寄存器直接寻址 数据 地址	 Dn An	 000 001	寄存器号 寄存器号	 × —	 — —	 — —	 × ×
寄存器间接寻址 地址 后自增 地址 前自减 地址 带偏移的地址	 (An) (An)+ -(An) (d_{16}, An)	 010 011 100 101	寄存器号 寄存器号 寄存器号	 × × × ×	 × × × ×	 × — × ×	 × × × ×

			寄存器号				
变址加 8 位偏移的地 址寄存器间接寻址	(d_8,An,Xi*SF)	110	寄存器号	×	×	×	×
带偏移的程序计数器间接寻址	(d_{16},PC)	111	寄存器号	×	×	×	—
变址加 8 位偏移的程 序计数器间接寻址	(d8,PC,Xi*SF)	111	寄存器号	×	×	×	—
绝对数据寻址 短 长	 (xxx).W (xxx).L	 111 111	 寄存器号 寄存器号	 × ×	 × ×	 × ×	 — —
立即数寻址	#<xxx>	111	寄存器号	×	×	—	—

2.3 堆栈

地址寄存器 A7 用堆栈来处理异常帧、子程序调用及返回、临时变量和参数传递等等，且它本身受如 LINK、UNLK、RTE 和 PEA 等指令影响。为使其性能最大化，任何时候 A7 都必为长字。因此，当修改 A7 时，请确定为 4 的倍数，以使得不会破坏这种考虑。进一步地，为保证在异常处理时 A7 这种对齐，ColdFire 体系在处理异常时使用了自动对齐的堆栈。

用户可以使用其他地址寄存器来实现另外的堆栈，这些寄存器采用后自增或前自减的间接寻址方式。利用地址寄存器用户可以实现，能存放来自高地址或低地址存储器数据的堆栈。请记住下面这些重要的内容：

- 在使用指针指向的堆栈内容之前，用前自减方式来减少寄存器中的值。
- 在使用指针指向的堆栈内容之后，用后自增方式来增加寄存器中的值。
- 当有字节、字和长字单元混用时，要保证堆栈中指针的正确性。 为实现堆栈从高存储区向低存储区的生长，用–(An)将数据压入堆栈，用(An)+把数据从堆栈中弹出。这种堆栈在一次压入和弹出操作后，指针将指向栈顶单元。

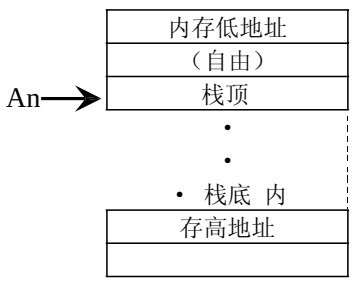


图 **2-15** 堆栈从高地址游历到低地址

为实现堆栈从低存储区向高存储区的生长，用(An)+压数据到堆栈，用−(An)从堆栈中弹出数据。这种堆栈在一次压入和弹出操作后，指针将指向栈中下一个可用单元。

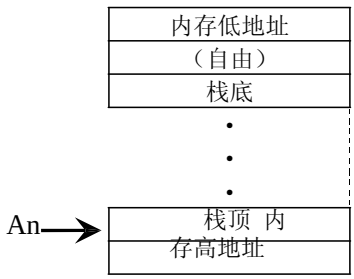


图 **2-16** 堆栈从低地址游历到高地址

第 3 章 指令集概述

本章通过介绍飞思卡尔汇编语言的语法和符号，来简要描述 ColdFire 系列指令集的信息。其中包括了指令符号和格式等的详细资料。

3.1 指令概述

指令集提供了以下类型的操作：

- 数据传输
- 程序控制
- 整数算术
- 浮点算术
- 逻辑操作
- 转移操作
- 位操作
- 系统控制
- 高速缓存维护

注意

DIVS/DIVU 和 RES/REMU 指令并没有被应用在早期基于 V2 的设备，如 MCR5202、MCF5204 和 MCF5206 等。尽管 MCF5407 应用了 ISA_B，但它不支持用户堆栈指针(USP)。

表 3-14 按字母顺序列出了所有的用户指令集。 表 3-15 按字母顺序列出了所有的特权指令集。 下面的段落详细讲述各种指令操作的类型。表 3-1 列出了本手册使用的所有符号。

指令定义的操作数语法描述中，右边操作数是目的操作数。

表 **3-1** 符号规范

单操作数和双操作数的操作	
+	算术加法或自增
-	算术减法或自减
*	算术乘法
/	算术除法
~	按位取反，操作数是逻辑补码
&	逻辑与
|	逻辑或
⊥	逻辑或非
→	源操作数发到目的操作数
←→	两操作数互换
<op>	任意双操作数操作
被测<操作数>	操作数与 0 比较，同时设置条件码
符号扩展	所有高字段的位置成低字段的最高位的值。

其他操作	
If <条件> Then <操作> Else <操作>	测试条件。如果为真，则执行“Then”后的操作。如果条件为假并且存在“Else”字句，则执行“Else”后的操作。 如果条件为假并且省略“Else”字句， 则不执行任何操作。参考 Bcc 指令描述。
寄存器描述	
An	地址寄存器 n (如 A3 是地址寄存器 3)
Ax, Ay	分别为目的地址寄存器和源地址寄存器
Dn	数据寄存器 n (如 D5 是数据寄存器 5)
Dx, Dy	分别为目的数据寄存器和源数据寄存器
Dw	存放余数的数据寄存器
Rc	控制寄存器
Rn	地址或数据寄存器
Rx, Ry	分别为目的寄存器和源寄存器
Xi	变址寄存器，可被用作地址或数据寄存器；32 位全用。
子域和限定词	
#<data>	指令字后的立即数
()	获得寄存器中的间接地址
dn	偏移值，n 位宽 (如 d16 是 16 位的偏移)
sz	字长，字节(B)、字(W)或长字(L)
lsb, msb	最低位，最高位
LSW, MSW	最低字，最高字
SF	变址寄存器的比例因子
寄存器名称	
CCR	条件码寄存器(状态寄存器的低字节)
PC	程序计数器
SR	状态寄存器
USP	用户堆栈指针
ic, dc, bc	指令，数据，或两者的高速缓存(标准高速缓存用 bc)
条件码	
*	常规格
C	条件码寄存器的进位标志位
cc	条件码寄存器的条件码
N	条件码寄存器的负值标志位
V	条件码寄存器的溢出标志位
X	条件码寄存器的扩展位
Z	条件码寄存器的零标志位
—	无影响或不可用
杂项	
<ea>x, <ea>y	分别是目的有效地址和源有效地址
<label>	汇编程序标签
#list	寄存器列表，如 D3:D0
MAC 操作	
ACC， ACCx	MAC 累加寄存器，特定的 EMAC 累加寄存器
ACCx,， ACCy	分别为目的累加器和源累加器
ACCext01	与 EMAC 累加器 0 和 1 相关的 4 个扩展字节
ACCext23	与 EMAC 累加器 2 和 3 相关的 4 个扩展字节

EV	乘积累加状态寄存器的扩展溢出标志位
MACSR	乘积累加状态寄存器
MASK	乘积累加掩码寄存器
PAVx	MACSR 的乘积/和溢出标志位
RxSF	保存被衡量的 MAC 操作数的寄存器
Rw	MAC 装载操作的目的寄存器
浮点操作	
fmt	操作格式：字节(B)、字(W)、长字(L)、单精度(S)或双精度(D)
+inf	正无穷
-inf	负无穷
FPx, FPy	分别是目的和源浮点数寄存器
FPCR	浮点控制寄存器
FPIAR	浮点指令地址寄存器
FPSR	浮点状态寄存器
NAN	非数值

3.1.1 数据传送指令

MOVE 和 FMOVE 指令及它们的寻址方式，是传输和存取地址或数据的最基本方法。MOVE 指令可以从存储器到存储器、存储器到寄存器、寄存器到存储器之间传输字节、字和长字操作数。MOVEA 指令用来传输字和长字操作数，并可确保只有合法的地址操作才被执行。相比于常规的传送指令，有些外加的数据传输指令，如 MOV3Q、MOVEM、MOVEQ、MVS、MVZ、LEA、PEA、LINK 和 UNLK，有特殊的功能。其中 MOV3Q、MVS 和 MVZ 是 ISA_B 对原先指令集的扩展。

FMOVE 指令在浮点数寄存器之间移入/移出操作数，也可对 FPCR、FPIAR 和 FPSR 做移入/移出操作。当操作数移入浮点数寄存器时，FSMOVE 和 FDMOVE 会对数值做单精度或双精度的舍入。FMOVEM 指令可以传输任意的浮点数寄存器。表 3-2 列出了整数和浮点数传输指令的通用格式。

表 **3-2** 数据传输操作格式

指令	操作数语法	操作长度	操作说明	最早出现在
FDMOVE	FPy, FPx	D	源→目的 目的操作数转换成双精度	FPU
FMOVE	<ea>y, FPx FPy, <ea>x FPy, FPx FPcr, <ea>x <ea>y, FPcr	B,W,L,S,D B,W,L,S,D D L L	源 → 目的 FPcr 可为任意浮点数寄存器：FPCR、FPIAR 或 FPSR	FPU
FMOVEM	#列表, <ea>x <ea>y, #列表	D	列出的寄存器→目的寄存器 源寄存器→列出的寄存器	FPU

FSMOVE	<ea>y, FPx	B,W,L,S,D	源→目的；目的操作数转换成单精度	FPU
LEA	<ea>y, Ax	L	<ea>y → Ax	ISA_A
LINK	Ay, #<偏移>	W	SP−4→SP；Ay→(SP)；SP→Ay, SP+dn→SP	ISA_A
MOV3Q	#<数据>, <ea>x	L	立即数→目的寄存器	ISA_B
MOVCLR	ACCy,Rx	L	加法器→目的寄存器 加法器清零	EMAC
MOVE MOVE from CCR MOVE to CCR	<ea>y, <ea>x MACcr, Dx <ea>y, MACcr CCR, Dx <ea>y, CCR	B,W,L L L W W	源→目的 MACcr 作为 MAC 控制寄存器的情况：ACCx, CCext01, ACCext23, MACSR, MASK	ISA_A[1] MAC[2] MAC[2] ISA_A ISA_A
MOVEA	<ea>y, Ax	W,L→L	源→目的	ISA_A
MOVEM	#列表, <ea>x <ea>y,#列表	L	列出的寄存器→目的寄存器 源寄存器→列出的寄存器	ISA_A
MOVEQ	#<数据>, Dx	B→L	立即数→目的寄存器	ISA_A
MVS	<ea>y, Dx	B,W	符号扩展源→目的	ISA_B
MVZ	<ea>y, Dx	B,W	零扩展源→目的	ISA_B
PEA	<ea>y	L	SP-4→SP； <ea>y→(SP)	ISA_A
UNLK	Ax	None	Ax→SP, (SP)→Ax, SP+4→SP	ISA_A

[1] 一些寻址方式的支持和 ISA_B 一起介绍。请结合表 3-16。

[2] 一些控制寄存器和 eEMAC 指令集一起介绍。请结合表 3-16。

3.1.2 程序控制指令

子程序调用与返回指令和条件与非条件转移指令完成程序控制操作，包括测试操作数指令(TST 和 FTST)，设置整数或浮点数条件码以便其他程序和系统控制指令执行。NOP 强制内部管道同步。TPF 是无操作的指令，不强制管道同步。表 3-3 概括了这些指令。

表 **3-3** 程序控制指令格式

指令	操作语法	操作长度	操作说明	最早出现在
有条件				
Bcc	<标签>	B,W,L	如果条件为真，则 PC+dn→PC	ISA_A[1]
FBCC	<标签>	W,L	如果条件为真，则 PC+dn→PC	FPU
Scc	Dx	B	如果条件为真，目的寄存器置全 1；否则为全 0	ISA_A

非条件				
BRA	<标签>	B,W,L	PC+dn→PC	ISA_A[1]
BSR	<标签>	B,W,L	SP-4→SP；nextPC→(SP)；PC+dn→PC	ISA_A[1]
FNOP	none	none	PC+2→PC(FPU 管道同步)	FPU
JMP	<ea>y	none	源地址→PC	ISA_A
JSR	<ea>y	none	SP-4→SP；nextPC→(SP)；源→PC	ISA_A
NOP	none	none	PC+2→PC(整数管道同步)	ISA_A
TPF	none #<数据> #<数据>	none W L	IPC+2→PC PC+4→PC PC+6→PC	ISA_A
返回				
RTS	none	none	(SP) →PC；SP+4→SP	ISA_A
测试操作数				
TAS	<ea>x	B	测试目的寄存器→CCR 目的寄存器位 7 置 1	ISA_B
FTST	<ea>y	B,W,L,S,D	测试源操作→FPCC	FPU
TST	<ea>y	B,W	测试源操作→CCR	ISA_A

[1] 有关操作数长度的支持和 ISA_B 一起介绍。请结合表 3-16。

整数指令 Bcc 和 Scc 中的字母 cc 用于指定测试下面条件中的一种：

CC—清除进位　　GE—大于或等于
LS—低于或等于　　PL—正的 CS—
设置进位　　GT—大于
LT—小于　　T—永真 [1]
EQ—等于　　HI—高于
MI—负的　　VC—清除溢出标志
F—永假 [1]　　LE—小于或等于
NE—不等于　　VS—设置溢出标志

[1] 对 **Bcc** 不适用。

FBcc 的定义参考 7.2 节，“条件测试”。

3.1.3 整数算术指令

整型算术操作包括 5 个基本操作：ADD、SUB、MUL、DIV 和 REM。另外还包括 CMP、CLR 和 NEG。指令集包含的 ADD、CMP 和 SUB 指令，既可用于地址运算，又可用于数据运算。CLR 指令适用于所有长度的操作数。有符号和无符号的 MUL、DIV 和 REM 指令包括：

- 用于产生长字结果的字乘数
- 用于产生长字结果的长字乘数
- 长字除以字得到一个字长的商和一个字长的余数
- 长字除以长字得到一个长字长的商

• 长字除以长字得到一个长字长的余数

有些扩展指令，如 ADDX、SUBX、EXT 和 NEGX，给多精度的可变字长的算术运算提供了支持。对于具有可选 MAC 或 EMAC 部件的芯片，MAC 和 MSAC 指令是很有用的。请参考表 3-4 的整型算术操作的概述。在表 3-4 中，X 表示 CCR 中的 X 位。

表 **3-4** 整数算术指令格式

指令	操作语法	操作长度	操作说明	最早出现在
ADD ADDA	Dy, <ea>x <ea>y, Dx <ea>y, Ax	L L L	源+目的→目的	ISA_A
ADDI ADDQ	#<数据>, Dx #<数据>, <ea>x	L L	立即数+目的→目的	ISA_A
ADDX	Dy, Dx	L	源+目的+CCR[X] →目的	ISA_A
CLR	<ea>x	B,W,L	目的寄存器清零	ISA_A
CMP CMPA	<ea>y, Dx <ea>y, Ax	B,W,L W,L	目的-源→CCR	ISA_A[1]
CMPI	#<数据>, Dx	B,W,L	目的-立即数→CCR	ISA_A[1]
DIVS/DIVU	<ea>y, Dx	W,L	目的/源→目的 (有符号或无符号)	ISA_A
EXT EXTB	Dx Dx Dx	B→W W→L B→L	符号扩展目的→目的	ISA_A
MAAAC	Ry, RxSF, ACCx, ACCw	W,L	ACCx+(Ry*Rx){<<I>>}SF→ACCx ACCw+(Ry*Rx){<<I>>}SF→ACCw	ISA_C EMAC_B
MAC	Ry ,RxSF, ACCx Ry, RxSF, <ea>y, Rw, ACCx	W,L W,L	ACCx+(Ry*Rx){<<I>>}SF→ACCx ACCx+(Ry*Rx){<<I>>}SF→ACCx; (<ea>y(&MASK)) →Rw	ISA_A
MASAC	Ry, RxSF, ACCx, ACCw	W,L	ACCx+(Ry*Rx){<<I>>}SF→ACCx ACCw-(Ry*Rx){<<I>>}SF→ACCw	ISA_C EMAC_B
MSAAC	Ry, RxSF, ACCx, ACCw	W,L	ACCx-(Ry*Rx){<<I>>}SF → ACCx ACCw+(Ry*Rx){<<I>>}SF→ACCw	ISA_C EMAC_B
MSAC	Ry, RxSF,	W,L	ACCx-(Ry*Rx){<<I>>}SF→ACCx	ISA_A

	ACCx Ry, RxSF, <ea>y, Rw, ACCx	W,L	ACCx-(Ry*Rx){<<I>>}SF→ACCx； (<ea>y(&MASK)) →Rw	
MSSAC	Ry, RxSF, ACCx, ACCw	W,L	ACCx-(Ry*Rx){<<I>>}SF→ACCx ACCw-(Ry*Rx){<<I>>}SF→ACCw；	ISA_C EMAC_B
MULS/MULU	<ea>y, Dx	W*W →L L*L→ L	源*目的→目的 (有符号或无符号)	ISA_A
NEG	Dx	L	0-目的→目的	ISA_A
NEGX	Dx	L	0-目的-CCR[X] →目的	ISA_A
REMS/REMU	<ea>y, Dw:Dx	L	目的/源→余数 (有符号或无符号)	ISA_ A
SATS	Dx	L	If CCR[V] == 1； Then if Dx[31] == 0； Then Dx[31:0]=0x80000000； Else Dx[31:0]=0x7FFFFFFF； Else Dx[31:0] 不变	ISA_ B
SUB SUBA	<ea>y, Dx Dy, <ea>x <ea>y, Ax	L L L	目的-源→目的	ISA_A
SUBI SUBQ	#<数据>, Dx #<数据>, <ea>x	L L	目的-立即数→目的	ISA_A
SUBX	Dy, Dx	L	目的-源-CCR[X] → 目的	ISA_A

[1] 有关操作数长度的支持和 ISA_B 一起介绍。请结合表 3-16。

3.1.4 浮点型算术指令

浮点型算术指令分为两种：双值型(两个操作数)和单值型(一个操作数)。双值型浮点数指令提供一些像 FADD 和 FSUB 等指令。对于这些操作，第一个操作数可以位于存储器、整型数据寄存器或浮点型数据寄存器中。第二个操作数总是位于浮点型数据寄存器中，结果将保存在第二个操作数指定的寄存器中。FPU 算术操作都支持所有的数据格式，其结果将被转换成单精度或双精度的格式。表 3-5 列出了双值型指令的通用格式。而表 3-6 则列出了它的相关的操作。

表 **3-5** 双值浮点型操作格式

指令	操作数语法	操作长度	操作说明	最早出现在

F<dop>	<ea>y, FPx FPy, FPx	B,W,L,S,D	FPx <Functioin> Source → FPx	FPU

表 **3-6** 双值浮点型操作

指令**(F<dop>)**	操作
FADD,FSADD,FDADD	加法
FCMP	比较
FDIV,FSDIV,FDDIV	除法
FUMUL,FSMUL,FDMUL	乘法
FSUB,FSSUB,FDSUB	减法

单值型浮点数指令提供了需要一个输入操作数的指令，例如 FABS。跟整型指令（如 NEG 等）对应的功能不同的是，它的源操作数和目的操作数是可被指定的。操作处理基于源操作数，而结果总是保存在浮点型数据目的寄存器中。它支持各种数据格式。表 3-7 列出了单值型指令的通用格式。表 3-8 列出了它的可用操作。

表 **3-7** 单值浮点型操作格式

指令	操作数语法	操作长度	操作说明	最早出现在
F<dop>	<ea>y, FPx FPy, FPx FPx	B,W,L,S,D	Sourcr→<Functioin> → FPx FPx→<Function>→FPX	FPU

表 **3-8** 单值浮点型操作

指令**(F<dop>)**	操作
FABS,FSABS,FDABS	绝对值
FINT	提取整数部分
FINTRZ	提取整数部分，向 0 靠拢
FNEG,FSNEG,FDNEG	取反
FSQRT,FSSQRT,FDSQRT	平方根

3.1.5 逻辑指令

AND、OR、EOR 和 NOT 是处理长字长整型数据的逻辑操作指令。相似的立即数操作指令，如 ANDI、ORI 和 EORI 等，则提供关于长整型立即数的逻辑操作。表 3-9 描述了这些逻辑指令。

表 **3-9** 逻辑操作格式

指令	操作数语法	操作长度	操作说明	最早出现在
AND	<ea>y, Dx Dy, <ea>x	L L	源 & 目的 → 目的	ISA_A
ANDI	#<数据>, Dx	L	立即数 & 目的 → 目的	ISA_A
EOR	Dy, <ea>x	L	源 ^ 目的→目的	ISA_A
EORI	#<数据>, Dx	L	立即数 ^ 目的→目的	ISA_A
NOT	Dx	L	目的操作数取反→目的	ISA_A
OR	<ea>y, Dx Dy, <ea>x	L L	源 | 目的→目的	ISA_A
ORI	#<数据>, Dx	L	立即数 | 目的 → 目的	ISA_A

3.1.6 移位指令

ASR、ASL、LSR 和 LSL 指令提供了左/右两种移位操作。所有移位指令都只适用于对寄存器的操作。

寄存器移位指令可移动长字。移动位数可以在指令操作字中指定（从 1 到 8）或在寄存器中指定（移位数为对 64 取模的结果）。

SWAP 指令交换寄存器中的高低 16 位。表 3-10 描述了移位指令。其中 C 和 X 表示 CCR 中的 C 位和 X 位。

表 **3-10** 移位操作格式

指令	操作数语法	操作长度	操作说明	最早出现在
ASL	Dy, Dx #<数据>, Dx	L L	CCR[X,C]←(Dx<<Dy) ← 0 CCR[X,C] ←(Dx<<#<数据>)← 0	ISA_A
ASR	Dy, Dx #<数据>, Dx	L L	msb→(Dx>>Dy) →CCR[X,C] msb→(Dx>>#<数据>)→CCR[X,C]	ISA_A
LSL	Dy, Dx #<数据>, Dx	L L	CCR[X,C]←(Dx<<Dy) ← 0 CCR[X,C] ←(Dx<<#<数据>)← 0	ISA_A
LSR	Dy, Dx #<数据>, Dx	L L	0→(Dx>>Dy) →CCR[X,C] 0→(Dx>>#<数据>)→CCR[X,C]	ISA_A
SWAP	Dx	W	MSW of Dx←→LSW of Dx	ISA_A

3.1.7 位操作指令

BTST、BSET、BCLR 和 BCHG 是位操作指令。位操作指令可以被应用在寄存器和存储器。位号可以用立即数，或由数据寄存器指定。寄存器操作数长度为 32 位，存储器每单元长度为 8 位。另外，BITREV、BYTEREV 和 FF1 指令对 32 位寄存器中的数据提供了更多的功能。表 3-11 给出了位操作指令的小结。

表 **3-11** 位操作指令格式

指令	操作数语法	操作长度	操作说明	最早出现在
BCHG	Dy, <ea>x #<数据>, <ea>x	B,L B,L	~(目的指定位的值)→CCR[Z] →目的指定位	ISA_A
BCLR	Dy, <ea>x #<数据>,<ea>x	B,L B,L	~(目的指定位的值)→CCR[Z]； 0→目的指定位	ISA_A
BITREV	Dx	L	按位取反 Dx→Dx	ISA_A+ ISA_C
BSET	Dy, <ea>x #<数据>,<ea>x	B,L B,L	~(目的指定位的值)→CCR[Z]； 1→目的指定位	ISA_A
BYTEREV	Dx	l	按位取反 Dx→Dx	ISA_A+ ISA_C
BTST	Dy, <ea>x #<数据>,<ea>x	B,L B,L	~(目的指定位的值)→CCR[Z]	ISA_A
FF1	Dx	l	Dx 中第一个为逻辑 1 的位的偏移→Dx	ISA_A+ ISA_C

3.1.8 系统控制指令

这种指令包含了特权指令和陷阱指令，以及利用到或能修改 CCR 的指令。FSAVE 和 FRESTORE 保存或恢复，浮点型运算单元在上下文切换选择时，程序可见的那部分信息。表 3-12 给出了这些指令的小结。

表 **3-12** 系统控制操作指令

指令	操作数语法	操作长度	操作说明	最早出现在
特权指令				
FRESTORE	<ea>y	none	FPU 状态框架→内部 FPU 状态	FPU
FSAVE	<ea>x	none	内部 FPU 状态→FPU 状态框架	FPU
HALT	none	none	停止处理器核心(同步管道)	ISA_A
MOVE for	SR, Dx	W	SR→目的寄存器	ISA_A

SR				
MOVE from USP	USP, Dx	L	USP→目的寄存器	ISA_B
MOVE to SR	<ea>y, SR	W	源寄存器→SR；仅 Dy 或 #<数据>(同步管道)	ISA_A
MOVE to USP	Ay, USP	L	源寄存器→USP	ISA_A
MOVEC	Ry, Rc	L	Ry→Rc(同步管道)	ISA_A
RTE	none	none	2(SP) → SR； 4(SP) → PC；SP+8→SP 根据格式调整栈空间(同步管道)	ISA_A
STOP	#<数据>	none	立即数→SR； STOP(同步管道)	ISA_A+ ISA_C
STLDSR	#<数据>	W	SP-4→SP； 零填充 SR→(SP)； 立即数→SR	ISA_A
WDEBUG	<ea>y	L	可寻址调试 WDMREG 命令执行(同步管道)	ISA_A
调试功能				
PULSE	none	none	设置 PST = 0X4	ISA_A
WDDATA	<ea>y	B,W,L	源操作数→DDATA 端口	ISA_A
陷阱产生				
ILLEGAL	none	none	SP-4→SP； PC→(SP)→PC；SP-2→SP； SP→(SP)； SP-2→SP； 向量偏移→(SP)； (VBR+0X10)→PC	ISA_A
TRAP	#<向量>	none	SR 的 S 位置 1； SP-4→SP；nextPC→(SP)；SP-2→SP；SR→(SP)； SP-2→SP； 格式/偏移→(SP) (VBR+0X80+4*n) →PC, 当 n 是中断号	ISA_A

某些指令提供比他们实际操作优先级更高的管道同步。对于这些操作码，指令进入 OEP，并等待直到下面的条件满足：

- 指令缓存处于静态，所有占用的缓存不被命中。
- 数据缓存处于静态，所有占用的缓存不被命中。
- 压入/存储缓冲区为空。

• 前面所有的指令全部执行完毕。 只要上面所有条件都满足，这些指令便会实际的去执行它的操作。对于在时序数据
项中列出的指令时序来说，为有关管道同步的指令作了以下约定：

• 指令缓存不处理缓存丢失。
• 数据缓存不处理缓存丢失。
• 压入/存储缓冲区为空。
• 在上一个周期OEP已经分配了一条指令或指令对。

下列指令处理管道同步：

• cpushl
• halt
• intouch
• move_to_sr
• movec
• nop
• rte
• stop
• wdebug

3.1.9 高速缓存保护指令

高速缓存指令提供了管理缓存的保护功能。CPUSHL 向堆栈压入指定的缓存队列，并可使之失效。INTOUCH 用来加载特定的数据到缓存。这些指令都是特权指令。表 3-13 描述了这些指令。

表 **3-13** 缓存保护操作格式

指令	语法	长度	操作说明	最早出现在
CPUSHL	ic, (Ax) dc, (Ax) bc, (Ax)	none	数据是合法或被修改，压入缓存线， 如果是对 CACR 编程，则压入无效线(管道同步)	ISA_A
INTOUCH	Ay	none	指令预取暂存在(Ay)(管道同步)	ISA_B

3.2 指令集概述

本节包含了描述 ColdFire 指令架构的表。

表 3-14 按字母顺序列出了所有用户指令集。表 3-15 按字母顺序列出了所有管理员指令集。回顾主要的 ISA 如下定义：

• ISA_A ： 原来的 ColdFire 指令集架构。
• ISA_B ： 增加了数据移动指令，字节和字长的比较等各种改进。
• ISA_C ： 增加位操作指令。

- FPU ： 采用原始的 ColdFire 指令集架构的浮点型单元(FPU)。
- MAC ： 采用原始的 ColdFire 指令集架构的乘法累加单元(MAC)。
- EMAC ： 修改 ISA，增强了乘法累加单元(EMAC)。
- EMAC_B ： 新增对偶运算操作指令。

表 **3-14 ColdFire** 用户指令集概述

指令	操作数语法	操作长度	操作说明	最早出现在
ADD ADDA	Dy, <ea>x <ea>y, Dx <ea>y,Ax	L L L	源 + 目的 → 目的	ISA_A
ADDI ADDQ	#<数据>, Dx #<数据>, <ea>x	L L	立即数 + 目的 → 目的	ISA_A
ADDX	Dy, Dx	L	立即数+目的+CCR[X]→目的	ISA_A
AND	<ea>y, Dx Dy, <ea>x	L L	源 & 目的 → 目的	ISA_A
ANDI	#<数据>, Dx	L	立即数 & 目的 → 目的	ISA_A
ASL	Dy, Dx #<数据>, Dx	L L	CCR[X,C]←(Dx<<Dy)←0 CCR[X,C]←(Dx<<#<数据>)←0	ISA_A
ASR	Dy, Dx #<数据>, Dx	L L	msb→(Dx>>Dy) →CCR[X,C] msb → (Dx>>#< 数 据 >) → CCR[X,C]	ISA_A
Bcc	<标签>	B,W	如果条件为真，则 PC+dn→PC	ISA_A
Bcc	<标签>	L	如果条件为真，则 PC+dn→PC	ISA_B
BCHG	Dy, <ea>x #<数据>, <ea>x	B,L B,L	~(目的指定位的值) → CCR[Z]→目的指定位	ISA_A
BCLR	Dy, <ea>x #<数据>, <ea>x	B,L B,L	~(目的指定位的值) → CCR[Z]；0→目的指定位	ISA_A
BITREV	Dx	L	目的数据寄存器内容按位取反	ISA_A+ ISA_C
BRA	<标签>	B,W	PC+dn→PC	ISA_A
BRA	<标签>	L	PC+dn→PC	ISA_B
BSET	Dy, <ea>x #<数据>, <ea>x	B,L B,L	~(目的指定位的值) → CCR[Z]；1→目的指定位	ISA_A
BSR	<lable>	B,W	SP-4→SP；nextPC→(SP)；PC+dn	ISA_A

			→PC	
BSR	<lable>	L	SP-4→SP；nextPC→(SP)；PC+dn →PC	ISA_B
BTST	Dy, <ea>x #<数据>, <ea>x	B,L B,L	~(目的指定位的值)→CCR[Z]	ISA_A
BYTEREV	Dx	L	目的数据寄存器按位取反	ISA_A+ ISA_C
CLR	<ea>x	B,W,L	0→目的寄存器	ISA_A
CMP CMPA	<ea>y, Dx <ea>y, Ax	L L	目的 - 源→CCR	ISA_B
CMP CMPA	<ea>y, Dx <ea>y, Ax	B,W W	目的-源→CCR	ISA_A
CMPI	#<数据>, Dx	L	目的 - 立即数→CCR	ISA_A
CMPI	#<数据>, Dx	B,W	目的 - 立即数→CCR	ISA_B
DIVS/DIVU	<ea>y, Dx	W,L	目的 / 源 → 目的（有符号或无符号）	ISA_A
EOR	Dy, <ea>x	L	源 ^ 目的 → 目的	ISA_A
EORI	#<数据>, Dx	L	立即数 ^ 目的 → 目的	ISA_A
EXT EXTB	Dx Dx Dx	B → W W → L B → L	符号扩展目的→目的	ISA_A
FABS	<ea>y, FPx FPy, FPx FPx	B,W,L,S,D D D	源的绝对值→FPx 源的 FPx→FPx	FPU
FADD	<ea>y, FPx FPy, FPx	B,W,L,S,D D	源 + FPx→FPx	FPU
FBcc	<标签>	W,L	如果条件为真，则 PC+dn→PC	FPU
FCMP	<ea>y, FPx FPy, FPx	B,W,L,S,D D	FPx - 源	FPU
FDABS	<ea>y, FPx FPy, FPx FPx	B,W,L,S,D D D	源的绝对值→FPx；目的转换为双精度 FPx 的绝对值→FPx；目的转换为双精度	FPU
FDADD	<ea>y, FPx FPy, FPx	B,W,L,S,D D	源 + FPx → FPx；目的转换为双精度	FPU
FDDIV	<ea>y, FPx FPy, FPx	B,W,L,S,D D	FPx / 源 → FPx；目的转换为双精度	FPU

FDIV	<ea>y, FPx FPy, FPx	B,W,L,S,D D	FPx / 源→FPx；	FPU
FDMOVE	FPy, FPx	D	源→目的；目的转换为双精度	FPU
FDMUL	<ea>y, FPx FPy, FPx	B,W,L,S,D D	FPx * 源→FPx；目的转换为双精度	FPU
FDNEG	<ea>y, FPx FPy, FPx FPx	B,W,L,S,D D D	- 源→FPx；目的转换为双精度 - FPx→FPx；目的转换为双精度	FPU
FDSQRT	<ea>y, FPx FPy, FPx FPx	B,W,L,S,D D D	源的平方根→FPx；目的转换为双精度 FPx 的平方根→FPx；目的转换为双精度	FPU
FDSUB	<ea>y, FPx FPy, FPx	B,W,L,S,D D	FPx - 源→FPx；目的转换为双精度	FPU
FF1	Dx	L	寄存器第一个逻辑位的偏移→目的寄存器	ISA_A+ ISA_C
FINT	<ea>y, FPx FPy, FPx FPx	B,W,L,S,D D D	源的整数部分→FPx； FPx 的整数部分→FPx；	FPU
FINTRZ	<ea>y, FPx FPy, FPx FPx	B,W,L,S,D D D	源的整数部分→FPx；向零取整 FPx 的整数部分→FPx；向零取整	FPU
FMOVE	<ea>y, FPx FPy, <ea>x FPy, FPx FPcr, <ea>x <ea>y, FPcr	B,W,L,S,D B,W,L,S,D D L L	源→目的 FPcr 可以作为任何浮点控制寄存器：FPCR,FPIAR,FPSR	FPU
FMOVEM	#列表, <ea>x <ea>y, #列表	D	列出的寄存器→目的寄存器 源寄存器→列出的寄存器	FPU
FMUL	<ea>y, PFx FPy, FPx	B,W,L,S,D D	源 * FPx→FPx	FPU
FNEG	<ea>y, FPx FPy, FPx FPx	B,W,L,S,D D D	- 源→FPx； - FPx→FPx；	FPU
FNOP	none	none	PC+2→PC（FPU 管道同步）	FPU
FSABS	<ea>y, FPx FPy, FPx	B,W,L,S,D D	源的绝对值→FPx；目的转换为单精度	FPU

	FPx	D	FPx 的绝对值→FPx；目的转换为单精度	
FSADD	<ea>y, FPx FPy, FPx	B,W,L,S,D	源 + FPx→FPx； 目的转换为单精度	FPU
FSDIV	<ea>y, FPx FPy, FPx	B,W,L,S,D D	FPx / 源→FPx； 目的转换为单精度	FPU
FSMOVE	<ea>y, FPx	B,W,L,S,D	源→目的； 目的转换为单精度	FPU
FSMUL	<ea>y, FPx FPy, FPx	B,W,L,S,D D	源 * FPx→FPx；目的转换为单精度	FPU
FSNEG	<ea>y, FPx FPy, FPx FPx	B,W,L,S,D D D	- 源→FPx； 目的转换为单精度 - FPx→FPx； 目的转换为单精度	FPU
FSQRT	<ea>y, FPx FPy, FPx FPx	B,W,L,S,D D D	源的平方根→FPx FPx 的平方根→FPx	FPU
FSSQRT	<ea>y, FPx FPy, FPx FPx	B,W,L,S,D D D	源的平方根→FPx； 目的转换为单精度 FPx 的平方根→FPx；目的转换为单精度	FPU
FSSUB	<ea>y, FPx FPy, FPx	B,W,L,S,D D	FPx - 源→FPx；目的转换为单精度	FPU
FSUB	<ea>y, FPx FPy, FPx	B,W,L,S,D D	FPx - 源→FPx	FPU
FTST	<ea>y	B,W,L,S,D	源测试结果→FPCC	FPU
ILLEGAL	None	none	SP-4→SP； PC→(SP)→PC； SP-2→SP； SR→(SP)； SP-2→SP； 向量 偏移 →(SP)； (VBR+0X10)→PC	ISA_A
JMP	<ea>y	none	源地址→PC	ISA_A
JSR	<ea>y	none	SP-4→SP； nextPC→(SP)； 源→PC	ISA_A
LEA	<ea>y, Ax	L	<ea>y→Ax	ISA_A
LINK	Ay, #<位移>	W	SP-4 → SP； Ay → (SP)； SP → Ay, SP+dn→SP	ISA_A
LSL	Dy, Dx #<数据>, Dx	L L	CCR[X,C]←(Dx<<Dy)←0 CCR[X,C] ← (Dx<<#<数据>)←	ISA_A

			0	
LSR	Dy, Dx #<数据>, Dx	L L	0→(Dx>>Dy) →CCR[X,C] 0→(Dx>>Dy) →CCR[X,C]	ISA_A
MAAAC	Ry, RxSF, ACCx, ACCw	L	ACCx+(Ry*Rx){<<I>>}SF → ACCx ACCw+(Ry*Rx){<<I>>}SF → ACCw	ISA_C EMAC_B
MAC	Ry, RxSF, ACCx Ry, RxSF, <ea>y, Rw, ACCx	W,L W,L	ACCx+(Ry*Rx){<<I>>}SF → ACCx; ACCx+(Ry*Rx){<<I>>}SF → ACCx (<ea>y(&MASK)) → Rw	MAC
MASAC	Ry, RxSF, ACCx, ACCw	L	ACCx+(Ry*Rx){<<I>>}SF → ACCx ACCw-(Ry*Rx){<<I>>}SF → ACCw	ISA_C EMAC_B
MOV3Q	#<数据>, <ea>x	L	立即数→目的寄存器	ISA_B
MOVCLR	ACCy, Rx	L	累加器→目的寄存器；0→累加器	EMAC
MOVE MOVE from CCR MOVE to CCR	<ea>y, <ea>x MACcr, Dx <ea>y, MACcr CCR, Dx <ea>y, CCR	B,W,L L L W W	源→目的 当 MACcr 可以作为任意的 MAC 控制寄存器：ACCx，ACCext01，ACCext23，MACSR，MASK	ISA_A MAC MAC ISA_A ISA_A
MOVE	#<数据>, d16(Ax)	B,W	立即数→目的寄存器	ISA_B
MOVEA	<ea>y, Ax	W,L→L	源→目的	ISA_A
MOVEM	#列表, <ea>x <ea>y, #列表	L	列出的寄存器→目的寄存器 源寄存器→目的寄存器	ISA_A
MOVEQ	#<数据>, Dx	B→L	立即数→目的寄存器	ISA_A
MSAAC	Ry, RxSF, ACCx Ry, RxSF, <ea>y, Rw, ACCx	L	ACCx-(Ry*Rx){<<I>>}SF → ACCx ACCw+(Ry*Rx){<<I>>}SF → ACCw	ISA_C EMAC_B
MSAC	Ry, RxSF, ACCx Ry, RxSF, <ea>y, Rw,	W,L W,L	ACCx-(Ry*Rx){<<I>>}SF → ACCx	MAC

	ACCx		ACCx-(Ry*Rx){<<I>>}SF →ACCx (<ea>y(&MASK)) →Rw	
MSSAC	Ry, RxSP, ACCx, ACCw	L	ACCx-(Ry*Rx){<<I>>}SF →ACCx ACCw-(Ry*Rx){<<I>>}SF →ACCw	ISA_C EMAC_B
MULS/MULU	<ea>y, Dx	W*W→L	源*目的→目的（有符号和无符号）	ISA_A
MVS	<ea>y, Dx	<ea>y, Dx	有符号扩展的源→目的	ISA_B
MVZ	<ea>y, Dx	<ea>y, Dx	零填充的源操作数→目的寄存器	ISA_B
NEG	Dx		0-目的寄存器→目的寄存器	ISA_A
NEGX	Dx	L	0-目的寄存器-CCR[X]→目的寄存器	ISA_A
NOP	none	none	PC+2→PC(整数管道同步)	ISA_A
NOT	Dx	L	~目的→目的	ISA_A
OR	<ea>y, Dx Dy, <ea>x	L L	源| 目的→目的	ISA_A
ORI	#<数据>, Dx	L	立即数|目的寄存器→目的寄存器	ISA_A
PEA	<ea>y	L	SP-4→SP；<ea>y→(SP)	ISA_A
PULSE	None	none	设置 PST = 0x4	ISA_A
REMS/REMU	<ea>y, Dw:Dx	L	目的寄存器/源操作数→余数(有符号或无符号)	ISA_A
RTS	none	none	(SP) →PC；SP+4→SP	ISA_A
SATS	Dx	L	If CCR[V] ==1； Then if Dx[31] ==0； Then Dx[31:0]=0x80000000； Else Dx[31:0] =0x7FFFFFFF； Else Dx[31:0] 不变	ISA_B
Scc	Dx	B	如果条件为真，目的寄存器置全 1；否则目的寄存器置全 0	ISA_A
SUB	<ea>y, Dx Dy, <ea>x	L L	目的 - 源→目的	ISA_A

SUBA	<ea>y, Ax	L		
SUBI SUBQ	#<数据>, Dx #<数据>, <ea>x	L L	目的 - 立即数→目的	ISA_A
SUBX	Dy, Dx	L	目的 - 源 - CCR[X] →目的	ISA_A
SWAP	Dx	W	Dx 的高位字←→Dx 的低位字	ISA_A
TAS	<ea>x	B	目的测试结果→CCR；1→目的寄存器的第七位	ISA_A
TPF	none #<数据> #<数据>	none W L	PC+2→PC PC+4→PC PC+6→PC	ISA_B
TRAP	#<向量>	none	SR 的 S 位置 1； SP-4→SP；nextPC→(SP)；SP-2→SP；SR→(SP) SP-2→SP；格式/偏移→(SP) (VBR+0X80+4*n) →PC,n 是中断号	ISA_A
TST	<ea>y	B,W,L	源操作数被测结果→CCR	ISA_A
UNLK	Ax	none	Ax→SP；(SP) →Ax；SP+4→SP	ISA_A
WDDATA	<ea>y	B,W,L	源→DDATA 端口	ISA_A

表 **3-15 ColdFire** 管理员指令集概述

指令	操作数语法	操作长度	操作说明	最早出现在
CPUSHL	ic, (Ax) dc, (Ax) bc, (Ax)	none	数据是合法或被修改了，压入缓存线；如果是对 CACR 编程，则压入无效线(管道同步)	ISA_A
FRESTORE	<ea>y	none	FPU 状态框架→内部 FPU 状态	FPU
FSAVE	<ea>x	none	内部 FPU 状态→FPU 状态框架	FPU
HALT	none	none	处理器中断	ISA_A
INTOUCH	Ay	none	指令预区取到 Ay	ISA_B
MOVE from SR	SR, Dx	W	SR→目的寄存器	ISA_A
MOVE from USP	USP, Dx	L	USP→目的寄存器	ISA_B
MOVE to SR	<ea>y, SR	W	源→SR；仅 Dy 或 #<数据>	ISA_A

			源	
MOVE to USP	Ay, USP	L	源→USP	ISA_B
MOVEC	Ry, Rc	L	Ry→Rc	ISA_A
RTE	none	none	2(SP) →SR；4(SP) →PC；SP+8→SP； 根据格式调整栈	ISA_A
STLDSR	#<数据>	W	SP-4→SP；零填充 SR→SR；立即数→SR	ISA_A+, ISA_C
STOP	#<数据>	none	立即数→SR；停止	ISA_A
WDEBUG	<ea>y	L	寻址调试指令 WDMREG 被执行	ISA_A

3.3 ColdFire 内核小结

本章对整个 ColdFire 指令集架构及其相应版本给出了简明的参考。为了方便记忆，表 3-16 按字母顺序给出了所有指令及其相应版本的简明列表。更为详细的描述请参见表 3-14（3-13 页）和表 3-15（3-18 页）。

在本文档出版时，可用标准产品及其可选的内核与外围模块，都被列于表 3-16 中。

表 **3-16 ColdFire** 指令集和处理器参考信息

指令	描述	**ISA_A**	**ISA_ A+**	**ISA_B**	**ISA_ C**	**FPU**	**MAC**	**EMAC**	**EMAC_B**
ADD	加法	×	×	×	×				
ADDA	地址加法	×	×	×	×				
ADDI	立即数加法	×	×	×	×				
ADDQ	快速加法	×	×	×	×				
ADDX	带扩展的加法	×	×	×	×				
AND	逻辑与	×	×	×	×				
ANDI	立即数的逻辑与	×	×	×	×				
ASL, ASR	算术左移和右移	×	×	×	×				
Bcc.{B,W}	条件分支，字节和字	×	×	×	×				
Bcc.L	条件分支，长字			×	×				
BCHG	测试位和变化	×	×	×	×				
BCLR	测试位和清除	×	×	×	×				
BITREV	位取反		×		×				

BRA.{B,W}	分支,字节和字	×	×	×	×				
BRA.L	分支,长字		×	×					
BSET	测试位和设置	×	×	×	×				
BSR.{B,W}	子程序的分支，字节和字	×	×	×	×				
BSR.L	子程序的分支，长字			×	×				
BTST	测试位	×	×	×	×				
BYTEREV	字节取反		×		×				
CLR	清除	×	×	×	×				
CMP.{B,W}	比较，字节和字			×	×				
CMP.L	比较，长字	×	×	×	×				
CMPA.W	地址比较，字			×	×				
CMPA.L	地址比较，长字	×	×	×	×				
CMPI.{B,W}	立即数比较，字节和字			×	×				
CMPI.L	立即数比较，长字	×	×	×	×				
CPUSHL	压入缓存或可使缓存无效	×	×	×	×				
DIVS	带符号除法	×	×	×	×				
DIVU	无符号除法	×	×	×	×				
EOR	逻辑异或	×	×	×	×				
EORI	立即数逻辑异或	×	×	×	×				
EXT,EXTB	符号扩展	×	×	×	×				
FABS FSABS FDABS	浮点数绝对值					×			
FADD FSADD FDADD	浮点数加法					×			
FBcc	浮点数条件分支					×			
FCMP	浮点数比较					×			
FDIV,FSDIV, FDDIV	浮点数除法					×			
FF1	查找第一个 1		×		×				
FFINT,FSINT, FSINT	浮点整数					×			

FINTRZ	归零浮点整数					×			
FMOVE FSMOVE FDMOVE	移出浮点数据积存器					×			
FMOVE from FPCR	从浮点控制寄存器移出					×			
FMOVE from FPIAR	从浮点指令地址寄存器移出					×			
FMOVE from FPSR	从浮点状态寄存器移出					×			
FMOVE to FPCR	移进浮点控制寄存器					×			
FMOVE to FPIAR	移进浮点指令地址寄存器					×			
FMOVE to FPSR	移进浮点状态寄存器					×			
FMOVEM	移出多个浮点数据寄存器					×			
FMUL,FSMUL,FDMUL	浮点数据寄存器					×			
FNEG,FSMUL,FDMUL	浮点数的补码					×			
FNOP	无浮点数操作					×			
FRESTORE	恢复内部浮点状态					×			
FSAVE	保存内部浮点状态					×			
FSQRT,FSSQRT,FDSQRT	浮点数平方根					×			
FSUB	浮点数减法					×			
FTST	测试浮点操作					×			
HALT	CPU 停止	×	×	×	×				
ILLEGAL	获得非法指令陷阱	×	×	×	×				
INTOUCH	指令获取			×	×				
JMP	跳转	×	×	×	×				
JSR	跳转到子程序	×	×	×	×				

LEA	加载有效地址	×	×	×	×				
LINK	连接和分配	×	×	×	×				
LSL, LSR	逻辑左移和右移	×	×	×	×				
MAAAC	乘和加放到第一个累加器，加法放到第二个累加器								×
MAC	乘积						×	×	×
MASAC	乘法和加法结果存到第一个累加器，减法从第二个累加器中读取								×
MOV3Q	快速移动位 3 的数据			×	×				
MOVCLR	从累加器中移出并清除							×	×
MOVE	移动	×	×	×	×				
MOVEI	移动立即数，字节或字到 Ax，代替原来的值			×	×				
MOVE ACC to ACC	拷贝累加器							×	×
MOVE from ACC	从累加器中移出						×	×	×
MOVE from ACCext01	从累加器 0 和 1 的扩展区间移出							×	×
MOVE ACCext23	从累加器 2 和 3 的扩展区间移出							×	×
MOVE from CCR	从条件码寄存器中移出	×	×	×	×				
MOVE from MACSR	从MAC状态寄存器中移出						×	×	×
MOVE MACSR to CCR	把MAC状态寄存器的值移到 CCR 中						×	×	×
MOVE from MASK	从MAC计时寄存器中移出						×	×	×
MOVE from	从状态寄存器中	×	×	×	×				

SR	移出								
MOVE from USP	从用户栈指针寄存器中移出		×	×	×				
MOVE to ACC	移入累加器						×	×	×
MOVE to ACCext01	移入累加器 0 和 1 的扩展单元							×	×
MOVE ACCext23	移入累加器 2 和 3 的扩展单元							×	×
MOVE to CCR	移入条件码寄存器	×	×	×	×				
MOVE to MACSR	移入 MAC 状态寄存器						×	×	×
MOVE to MASK	移入 MAC 计时寄存器						×	×	×
MOVE to SR	移入状态寄存器	×	×	×	×				
MOVE to USP	移入用户栈指针		×	×	×				
MOVEA	转移地址	×	×	×	×				
MOVEC	转移控制寄存器	×	×	×	×				
MOVEM	转移乘法寄存器	×	×	×	×				
MOVEQ	快速移动	×	×	×	×				
MSAAC	乘法和减法存入第一个累加器，加法存入第二个累加器								×
MSAC	乘法和减法						×	×	×
MSSAC	乘法和减法移入到第一个累加器，减法移入到第二个累加器								×
MULS	有符号乘法	×	×	×	×				
MULU	无符号乘法	×	×	×	×				
MVS	带符号扩展的转移			×	×				
MVZ	零填充转移			×	×				
NEG	取反	×	×	×	×				

NEGX	带扩展取反	×	×	×	×				
NOP	无操作	×	×	×	×				
NOT	逻辑补码	×	×	×	×				
OR	逻辑或	×	×	×	×				
ORI	立即数逻辑或	×	×	×	×				
PEA	把有效地址压入堆栈	×	×	×	×				
PULSE	产生处理器状态	×	×	×	×				
REMS	带符号的除法余数	×	×	×	×				
REMU	无符号的除法余数	×	×	×	×				
RTE	从异常返回	×	×	×	×				
RTS	从子程序返回	×	×	×	×				
SATS	符号填充			×	×				
Scc	根据条件设置	×	×	×	×				
STLDSR	存储或加载状态寄存器		×		×				
STOP	加载状态寄存器并停止	×	×	×	×				
SUB	减法	×	×	×	×				
SUBA	地址减法	×	×	×	×				
SUBI	立即数减法	×	×	×	×				
SUBQ	快速减法	×	×	×	×				
SUBX	带扩展的减法	×	×	×	×				
SWAP	交换寄存器的字	×	×	×	×				
TAS	测试，设置和操作			×	×				
TPF	陷阱错误	×	×	×	×				
TRAP	陷阱	×	×	×	×				
TST	测试操作	×	×	×	×				
UNLK	无连接	×	×	×	×				
WDDATA	写数据控制寄存器	×	×	×	×				
WDEBUG	写调试控制寄存器	×	×	×	×				

第 **4** 章 整型用户指令

本节叙述 ColdFire 系列的整型用户指令。为了便于记忆，每个指令的详细讨论按字母顺序排列。

并非所有的指令都被 ColdFire 处理器所支持。请参见第 3 章“就指令集定义的具体细节的指令集概要”。

ADD

Add

ADD

首先出现于 ISA_A

指令操作

源+目的→目的

汇编格式

ADD.L <ea>y,Dx

ADD.L Dy,<ea>x

相关属性

尺寸 = 长字长度。

指令描述

使用二进制将源操作数和目的操作数相加并将结果保存到目的地址。操作数的长度可以只是单个长字。指令的模式指出源和目的操作数分别是哪个以及他们的数据长度。

Dx 模式用于目的操作数是数据寄存器的情况，目的<ea>x 模式对于数据寄存器是不可用的。此外，ADDA 用于目的操作数是地址寄存器的情况。ADDI 和 ADDQ 用于源操作数是立即数的情况。

条件码

X	N	Z	V	C
*	*	*	*	*

X 与进位位的值相同
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 结果若溢出则置位，否则清零
C 结果有进位则置位，否则清零

指令格式

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	1	0	1	寄存器	操作模式	有效地址	
						模式	寄存器

指令域

寄存器域——指定数据寄存器。

操作模式域

字节	字	长字	操作
—	—	010	<ea>y + Dx → Dx
—	—	110	Dy + <ea>x → <ea>x

有效地址域——用于决定寻址方式。

对于源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	001	寄存器数:Ay
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

对于目的操作数<ea>x，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dx	—	—
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	110	寄存器数:Ax

寻址方式	模式	寄存器
(xxx).W	111	寄存器数:Dy
(xxx).L	111	寄存器数:Ay
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

ADDA　　ADDA

Add Address

首先出现于 ISA_A

指令操作

源+目的→目的

汇编格式

ADDA.L <ea>y,Ax

相关属性

尺寸 = 长字长度。

指令描述

类似于 ADD 操作，但不同的是，它用于目的寄存器是地址寄存器而不是数据寄存器的时候。将源操作数加到目的地址寄存器，并存储结果到地址寄存器。操作数尺寸被定为长字的长度。

条件码 不受影响。

指令格式

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	1	0	1	目的寄存器 Ax	1	1	1	源有效地址 模式	寄存器

指令域

目的寄存器域——指定目的寄存器 Ax。 源有效地址域——用于决定源操作数，所用寻址方式列表如下：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	001	寄存器数:Ay
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

ADDI　　Add Immediate　　ADDI

首先出现于 ISA_A

指令操作

立即数+目的→目的

汇编格式

ADDI.L #<data>,Dx

相关属性

尺寸 = 长字长度。

指令描述

类似于 ADD 操作，但不同的是，它用于源操作数是立即数的时候。将立即数加到目的操作数，并存储结果到目的数据寄存器 Dx。操作数尺寸被定为长字的长度。立即数尺寸也被定为长字的长度。注意，该立即数是包含在两个被扩展的字中，换言之，第一个扩展字的[15:0]位载有高字，第二个扩展字的[15:0]位载有低字。

条件码

X	N	Z	V	C
*	*	*	*	*

X　与进位位的值相同
N　结果为负数则置位，否则清零
Z　结果为零数则置位，否则清零
V　结果若溢出则置位，否则清零
C　结果有进位则置位，否则清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	1	0	0	0	0	寄存器 Dx		
立即数的高字															
立即数的低字															

指令域

目的寄存器域——指定目的寄存器 Dx。

ADDQ Add Quick ADDQ

首先出现于 ISA_A

指令操作

立即数+目的→目的

汇编格式

ADDQ.L #<data>,<ea>x

相关属性

尺寸 = 长字长度。

指令描述

类似于 ADD 操作，但不同的是，它用于源操作数是范围是从 1 到 8 的立即数的时候。它将立即数加到目的操作数。操作数尺寸被定为长字长度。立即数在被加到目的操作数之前先用 0 来填补为长字。注意，当需加到地址寄存器时，条件码不受影响。

条件码

X	N	Z	V	C
*	*	*	*	*

X 与进位位的值相同
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 结果若溢出则置位，否则清零
C 结果有进位则置位，否则清零

指令格式

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	1	0	1	数据	0	1	0	目的有效地址 模式	寄存器

指令域

数据域——用 3 位立即数表示 0 到 7 这 8 个数字，其中 1-7 分别表示数字 1-7，0 表示数字 8。

目的有效地址域——指定目的位置，<ea>X 仅使用以下表格所列出的那些可变的寻址方式：

寻址方式	模式	寄存器
Dx	000	寄存器数:Dx
Ax	001	寄存器数:Ax
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	110	寄存器数:Ax

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

ADDX　　ADDX

Add Extended

首先出现于 ISA_A

指令操作

源 + 目的 + CCR[X] → 目的

汇编格式

ADDX.L Dy,Dx

相关属性

尺寸 = 长字长度。

指令描述

把源操作数、目的操作数与 CCR[X]相加，结果保存到目的位置。操作符大小指定为长字。

条件码

X	N	Z	V	C
*	*	*	*	*

X　与进位位的值相同
N　结果为负数则置位，否则清零
Z　结果不为零则清零，否则不变
V　结果若溢出则置位，否则清零
C　结果有进位则置位，否则清零

通常CCR[Z]在 ADDX 操作开始前经过明确规划而置位，允许在多精度操作的完成的基础上对零结果的成功测试。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	寄存器 Dx			1	1	0	0	0	0	寄存器 Dy		

指令域

寄存器 Dx 域——指定目的数据寄存器 Dx。

寄存器 Dy 域——指定源数据寄存器 Dy。

AND

AND Logical

AND

首先出现于 ISA_A

指令操作

源&目的→目的

汇编格式

AND.L <ea>y,Dx

AND.L Dy,<ea>x

相关属性

尺寸 = 长字长度。

指令描述

对源操作数与目的操作数实行“与”操作，结果保存在目的地中。操作符大小指定为长字。地址寄存器中的地址可能不作为操作数。

Dx 模式用于目的操作数是数据寄存器的情况，目的<ea>x 模式对于数据寄存器是不可用的。

ANDI 用于源操作数是立即数的情况 。

条件码

X	N	Z	V	C
—	*	*	0	0

X 不受影响
N 结果最高有效位为 1 则置位，否则清零
Z 结果为零数则置位，否则清零
V 清零
C 清零

指令格式

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	1	0	0	数据寄存器	操作模式	有效地址	
						模式	寄存器

指令域

寄存器域——指定 8 个数据寄存器中的任一个。

操作模式域

字节	字	长字	操作
—	—	010	<ea>y + Dx → Dx
—	—	110	Dy + <ea>x → <ea>x

有效地址域——用于决定寻址方式。

对于源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

对于目的操作数<ea>x，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dx	—	—
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	110	寄存器数:Ax

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

ADDI　　ADDI

AND Immediate

首先出现于 ISA_A

指令操作

立即数&目的→目的

汇编格式

ANDI.L #<DATE>,Dx

相关属性

尺寸 = 长字长度。

指令描述

对立即数与目的操作数实行“与”操作，结果保存在目的数据寄存器 Dx 中。操作符大小指定为长字。注意，该立即数是包含在两个被扩展的字中，换言之，第一个扩展字的[15:0]位载有高字，第二个扩展字的[15:0]位载有低字。

条件码

X	N	Z	V	C
—	*	*	0	0

X　不受影响
N　结果最高有效位为 1 则置位，否则清零
Z　结果为零数则置位，否则清零
V　清零
C　清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	1	0	0	0	0	目的寄存器 Dx		
立即数的高字															
立即数的低字															

指令域

目的寄存器域——指定目的数据寄存器 Dx。

ASL,ASR　　Arithmetic Shift　　ASL,ASR

首先出现于 ISA_A

指令操作

计数转换后的目的→目的

汇编格式

ASd.L Dy,Dx

ASd.L #<DATE>,Dx

（d 表示方向，L 或 R）

相关属性

尺寸 = 长字长度。

指令描述

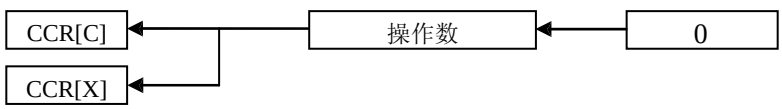
符合算法地对目的操作数 Dx 的位进行移位，按照指定好的方向进行（左或右）。操作符大小指定为长字。CCR[C]接受最后被移位的那个位。移位数是位移到目的寄存器的数目，可以由以下两种方法确定：

1 立即数——移位数在指令中被指定（移位范围 1-8）。

2 寄存器——移位数是数据寄存器 Dy 的值，在指令中被指定（以 64 为模）。

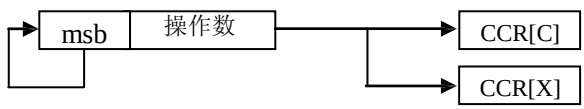
对于 ASL，操作数被左移，移位数等于被移动位置的号码。从高位按顺序移出的位转到进位和扩展位中；零被按顺序移入低位，溢出位总为零。

ASL



对于 ASR，操作数右移，被移动位置的号码等于移位数。从低位按顺序移出的位转到进位和扩展位中；符号位（最高位）被按顺序移入高位。

ASR



条件码

X	N	Z	V	C
*	*	*	0	*

X　按操作数被移出的最后位设置，不受零计数器变化的影响

N　结果最高位被置则置位，否则清零

Z　结果为零数则置位，否则清零

V　清零

C　按照操作数被移出的最后一位置位，若零计数器变化则清零

注意：CCR[V]被清零取决于 SAL 和 ASR，这与 68K 处理器家族不同。

指令格式

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	1	1	0	计数寄存器 Dy	dr	1	0	i/r	0	0	寄存器 Dx

指令域

计数或寄存器域——指定或寄存器 Dy，包含。

— 如果 i/r = 0，此域包含移位数，数字 1—7 代表 1—7 的转换；数字 0 指定 8 的转换计数。

— 如果 i/r = 1，此域数据寄存器 Dy 包含转换计数（以 64 为模）

dr 域——指定转换的方向。

— 0：右进位

— 1：左进位

i/r 域

— 如果 i/r=0，指定立即转换数。

— 如果 i/r=1，指定寄存器转换数。 寄存器

域——指定要转换的数据寄存器 Dx。

Bcc

Branch Conditionally

首先出现于 ISA_A

.L 首先出现于 ISA_B

Bcc

指令操作

如果条件为真，则 PC + d_n → PC

汇编格式

Bcc.sz <label>

相关属性

尺寸 = 字节、字、长字长度（以 ISA_B 开始支持长字长度）。

指令描述

如果条件正确，在(PC)+置换执行继续。用一正的或相反的（负的）置换传递枝。PC 保存 Bcc 指令加上 2 的指令字的地址。置换是一补码整型，它代表了目前 PC 和目的 PC 在一个字节中的相对距离。如果八位的置换域是 0x00，十六位的置换（指令之后的字）将执行。如果八位的置换域是 0xFF，三十二位的置换（指令之后的长字）将执行。因为八位的置换域是 0x00，下一立即指令的枝将执行十六位的置换。

条件码指定 C、N、V 和 Z 分别代表 CCR[C]、CCR[N]、CCR[V]和 CCR[Z]的条件码位，分别是：

代码	条件	编码	测试
CC(HS)	进位清零	0100	$\overline{C}$
CS(LO)	进位置位	0101	C
EG	相等	0111	Z
GE	大于等于	1100	N&V| $\overline{N}$ & $\overline{V}$
GT	大于	1110	N&V& Z | N & V & Z
HI	大于	0010	C & Z
LE	小于等于	1111	Z|N& $\overline{V}$ | $\overline{N}$ &V

代码	条件	编码	测试
LS	小于等于	0011	C|Z
LT	小于	1101	N& V | N &V
MI	负数	1011	N
NE	不相等	0110	Z
PL	正数	1010	N
VC	溢出位清零	1000	V
VS	溢出位置位	1001	V

条件码

不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

0	1	1	0	条件	8 位偏移
16 位偏移（当 8 位偏移等于 0x00）					
32 位偏移（当 8 位偏移等于 0xFF）					

指令域

条件域——表格中列出的一个条件的二进制编码。

8 位偏移域——符合条件时整数补码指定跳转与下条要执行的指令之间的位数。

16 位偏移域——8 位偏移包含 0x00 时使用。

32 位偏移域——8 位偏移包含 0xFF 时使用。

BCHG BCHG

Test a Bit and Change

首先出现于 ISA_A

指令操作

~（目的位数）→ CCR[Z]

~（目的位数）→ 目的的<位数>

汇编格式

BCHG.sz Dy,<ea>x

BCHG.sz #<date>,<ea>x

相关属性

尺寸 = 字节或长字长度。

指令描述

测试目的操作数中的某个位并根据情况给CCR[Z]置位，接着颠倒那个目的中指定的位。当目的操作数是一个数据寄存器时，32个位中每一个都可以被选中。当目的操作数是一个内存地址，这种操作是以一个字节（8位）为单位的。所有的情况中，零位意味最小的有意义位。操作数的位数由以下两种方法确定：

1 立即数——位于指令的第二个字中。

2 寄存器——位于指定的数据寄存器中。

条件码

X	N	Z	V	C
—	—	*	—	—

X 不受影响
N 不受影响
Z 测试位为零则置位，否则清零
V 不受影响
C 不受影响

静态位数按立即数规则指定：

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	1	目的有效地址					
										模式			寄存器		
0	0	0	0	0	0	0	0	位数							

指令域

目的有效地址域——指定目的位置<ea>x，仅用到以下表格所列出的寻址方式，

注意长字仅用在 Dx 模式，其余一律使用字节。

寻址方式	模式	寄存器
Dx	000	—
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

位数域——指定位数。

动态位数，按寄存器规则指定：

指令格式

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	0	0	0	数据寄存器 Dy	1	0	1	目的有效地址	
								模式	寄存器

指令域

数据寄存器域——指定包含位数的数据寄存器 Dy。 目的有效地址域——指定目的<ea>x 位置，仅使用下表中列出的数据可变寻址方式，注意，长字只在 Dx 模式里使用，其余模式均使用字节。

寻址方式	模式	寄存器
Dx	000	寄存器数:Dx
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	110	寄存器数:Ax

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

BCLR　　BCLR

Test a Bit and Clear

首先出现于 ISA_A

指令操作

~（目的的<位数>）→CCR[Z];
0→目的的<位数>

汇编格式

BCLR.sz Dy,<ea>x
BCLR.sz #<date>,<ea>x

相关属性

尺寸 = 字节、长字长度。

指令描述

测试目的操作数中的某个位并根据情况给CCR[Z]置位,接着清除那个指定的位。当目的操作数是一个数据寄存器时，32个位中每一个都可以被选中。当目的操作数是一个内存地址，这种操作是以一个字节（8位）为单位的。。所有的情况中，零位意味最小的有意义位。操作数的位数由以下两种方法确定：

1：立即数——位于指令的第二个字中。
2：寄存器——位于指定的数据寄存器中。

条件码

X	N	Z	V	C
*	*	*	*	*

X　不受影响
N　不受影响
Z　测试位为零则置位，否则清零
V　不受影响
C　不受影响

静态位数按立即数规则指定：

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	0	目的有效地址					
										模式			寄存器		
0	0	0	0	0	0	0	0	位数							

指令域

目的有效地址域——指定目的位置<ea>x，仅用到以下表格所列出的寻址方式，注意长字仅用在 Dx 模式，其余一律使用字节。

寻址方式	模式	寄存器
Dx	000	寄存器数:Dx
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

位数域——指定位数。

动态位数，按寄存器规则指定：

指令格式

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	0	0	0	数据寄存器 Dy	1	1	0	目的有效地址	
								模式	寄存器

指令域

数据寄存器域——指定包含位数的数据寄存器 Dy。 目的有效地址域——指定目的位置<ea>x，仅用到以下表格所列出的寻址方式，

注意长字仅用在 Dx 模式，其余模式均使用字节。

寻址方式	模式	寄存器
Dx	000	寄存器数:Dx
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	110	寄存器数:Ax

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

BITREV Bit Reverse Register BITREV

首先出现于 ISA_A

指令操作

Dx 位置翻转→Dx

汇编格式

BITREV.L Dx

相关属性

尺寸 = 长字长度。

指令描述

目的数据寄存器的内容为：翻转位，i.e.，新 Dx[31]=原 Dx[0]， 新 Dx[30]=原 Dx[1]，……，新 Dx[0]=原 Dx[31]。

条件码 不受影

响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	1	0	0	0	寄存器 Dx		

指令域

寄存器域——指定目的数据寄存器 Dx。

BRA BRA

Branch Always

首先出现于 ISA_A

.L 首先出现于 ISA_B

指令操作

PC+ $d_n \rightarrow PC$

汇编格式

BRA.sz <label>

相关属性

尺寸 = 字节长度，字长度，长字长度（从 ISA B 开始支持长字长度）

指令描述

在(PC)+置换域，条件继续执行。用一正的或相反的（负的）置换传递枝。PC 保存 BRA 指令加上 2 的指令字的地址。置换是一补码整型，它代表了目前 PC 和目的 PC 在一个字节中的相对距离。如果八位的置换域是 0x00，十六位的置换（指令之后的字）将执行。如果八位的置换域是 0xFF，三十二位的置换（指令之后的长字）将执行。因为八位的置换域是 0x00（置零），下一立即指令的枝将执行十六位的置换。

条件码 不受影

响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	8 位偏移							
16 位偏移（当 8 位偏移等于 0x00）															
32 位偏移（当 8 位偏移等于 0xFF）															

指令域

8 位偏移域——二进制补码指定分支指令和下条要执行的指令之间的位数。

16 位偏移域——8 位偏移包含 0x00 时使用。

32 位偏移域——8 位偏移包含 0xFF 时使用。

BSET BSET

Test a Bit and Set

首先出现于 ISA_A

指令操作

~（目的的<位数>）→CCR[Z];
1→目的的<位数>

汇编格式

BSET.sz Dy,<ea>x
BSET.sz #<date>,<ea>x

相关属性

尺寸 = 字节、长字长度。

指令描述

测试目的操作数中的某个位并根据情况给CCR[Z]置位,接着置位那个指定的位。当目的操作数是一个数据寄存器时，32个位中每一个都可以被选中。当目的操作数是一个内存地址，这种操作是以一个字节（8位）为单位的。所有的情况中，零位意味最小的有意义位。操作数的位数由以下两种方法确定：

1：立即数——位于指令的第二个字中。

2：寄存器——位于指定的数据寄存器中。

条件码

X	N	Z	V	C
—	*	—	—	—

X 不受影响
N 不受影响
Z 测试位为零则置位，否则清零
V 不受影响
C 不受影响

静态位数按立即数规则指定：

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	1	目的有效地址					
										模式			寄存器		
0	0	0	0	0	0	0	0	位数							

指令域

目的有效地址域——指定目的位置<ea>x，仅用到以下表格所列出的寻址方式，注意长字仅用在 Dx 模式，其余一律使用字节。

寻址方式	模式	寄存器
Dx	000	寄存器数:Dx
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

位数域——指定位数。

动态位数，按寄存器规则指定：

指令格式

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	0	0	0	数据寄存器 Dy	1	1	1	目的有效地址	
								模式	寄存器

指令域

数据寄存器域——指定包含位数的数据寄存器 Dy。 目的有效地址域——指定目的位置<ea>x，仅用到以下表格所列出的寻址方式，

注意长字仅用在 Dx 模式，其余一律使用字节。

寻址方式	模式	寄存器
Dx	000	寄存器数:Dx
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	110	寄存器数:Ax

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

BSR

Branch to Subroutine

BSR

首先出现于 ISA_A

.L 首先出现于 ISA_B

指令操作

SP － 4 → SP; nextPC→(SP); PC+ d_n→PC

汇编格式

BSR.sz <label>

相关属性

尺寸 = 字节、字、长字长度（从 ISA_B 开始支持长字长度）。

指令描述

在BSR指令执行后，立即把指令的长字地址压进系统堆栈。用一正的或相反的（负的）置换传递枝。PC保存BRA指令加上2的指令字的地址。置换是一补码整型，它代表了目前PC和目的PC在一个字节中的相对距离。如果八位的置换域是0x00，十六位的置换（指令之后的字）将执行。如果八位的置换域是0xFF，三十二位的置换（指令之后的长字）将执行。因为八位的置换域是0x00（置零），下一立即指令的枝将执行十六位的置换。

条件码 不受影

响。

指令格式

15	14	13	12	11	10	9	8	7 6 5 4 3 2 1 0
0	1	1	0	0	0	0	1	8 位偏移
16 位偏移（当 8 位偏移等于 0x00）								
32 位偏移（当 8 位偏移等于 0xFF）								

指令域

8 位偏移域——二进制补码指定分支指令和下一条要执行的指令之间的位数。

16 位偏移域——8 位偏移包含 0x00 时使用。

32 位偏移域——8 位偏移包含 0xFF 时使用。

BTST BTST

Test a Bit

首先出现于 ISA_A

指令操作

~（目的的<位数>）→ CCR[Z];

汇编格式

BTST.sz Dy,<ea>x

BTST.sz #<date>,<ea>x

相关属性

尺寸 = 字节、长字长度。

指令描述

测试目的操作数中的某个位并根据情况给CCR[Z]置位。当目的操作数是一个数据寄存器时，32个位中每一个都可以被选中。当目的操作数是一个内存地址，这种操作是以一个字节（8位）为单位的。所有的情况中，零位意味最小的有意义位。操作数的位数由以下两种方法确定：

1：立即数——位于指令的第二个字中。

2：寄存器——位于指定的数据寄存器中。

条件码

X	N	Z	V	C
—	—	*	—	—

X 不受影响
N 不受影响
Z 测试位为零则置位，否则清零
V 不受影响
C 不受影响

静态位数按立即数规则指定：

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	目的有效地址					
										模式			寄存器		
0	0	0	0	0	0	0	0	位数							

指令域

目的有效地址域——指定目的位置<ea>x，仅用到以下表格所列出的寻址方式，注意长字仅用在 Dx 模式，其余一律使用字节。

寻址方式	模式	寄存器
Dx	000	寄存器数:Dx
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

位数域——指定位数。

动态位数，按寄存器规则指定：

指令格式

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	0	0	0	数据寄存器 Dy	1	0	0	目的有效地址 模式	寄存器

指令域

数据寄存器域——指定包含位数的数据寄存器 Dy。 目的有效地址域——指定目的位置<ea>x，仅用到以下表格所列出的寻址方式，注意长字仅用在 Dx 模式，其余一律使用字节。

寻址方式	模式	寄存器
Dx	000	寄存器数:Dx
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	110	寄存器数:Ax

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

BYTEREV Byte Reverse Register BYTEREV

首先出现于 ISA_C

指令操作

Dx 位置翻转→Dx

汇编格式

BYTEREV.L Dx

相关属性

尺寸 = 长字长度。

指令描述

目的数据寄存器内容按如下定义颠倒：

表格 **1**

new Dx[31:24]	=	old Dx[7:0]
new Dx[23:16]	=	old Dx[15:8]
new Dx[15:8]	=	old Dx[23:16]
new Dx[7:0]	=	old Dx[31:24]

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	1	1	0	0	0	寄存器 Dx		

指令域

寄存器域——指定目的数据寄存器 Dx。

CLR　　CLR

Clear an Operand

首先出现于 ISA_A

指令操作

0→目的

汇编格式

CLR.sz <ea>x

相关属性

尺寸 = 字节、字、长字长度。

指令描述

目的操作数清零。指令操作尺寸被指定为字节、字、长字长度。

条件码

X	N	Z	V	C
—	0	1	0	0

X 不受影响
N 清 零
Z 清 零
V 清 零
C 清零

指令格式

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
0	1	0	0	0	0	1	0	尺寸	目的有效地址	
									模式	寄存器

指令域

尺寸域——指定操作的尺寸。

— 00：字节　— 01：字　— 10：长字　— 11：翻转 有效地址域

——指定目的<ea>x 位置，仅用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dx	000	寄存器数:Dx
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

		数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	110	寄存器数:Ax

CMP CMP

Compare

首先出现于 ISA_A

B 和 W 首先出现于 ISA_B

指令操作

目的－源→cc

汇编格式

CMP.sz <ea>y,Dx

相关属性

尺寸 = 字节、字、长字长度（从 ISA_B 开始支持字节长度、字长度）。

指令描述

在数据寄存器中用目的操作数减去源操作数，根据结果给条件码置位。数据寄存器没有变化。操作数的尺寸可以是字节、字或长字。当目的操作数是地址寄存器时需要使用 CMPA，当源操作数是立即数时则使用 CMPI。

条件码

X	N	Z	V	C
—	*	*	*	*

X 不受影响
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 结果若溢出则置位，否则清零
C 结果有借位则置位，否则清零

指令格式

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	1	1	目的寄存器 Dx	操作模式	源有效地址	
						模式	寄存器

指令域

地址寄存器域——指定目的寄存器 Ax。

操作模式域

字节	字	长字	操作
—	011	011	Ax-<ea>y

源有效地址域——指定源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	001	寄存器数:Ay
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

CMPA CMPA

Compare Address

首先出现于 ISA_A W

首先出现于 ISA_B

指令操作

目的－源→cc

汇编格式

CMPA.sz <ea>y,Ax

相关属性

尺寸 = 字、长字长度（从 ISA_B 开始支持字长度）。

指令描述

操作近似于 CMP，适用于目的寄存器是地址寄存器时。操作数尺寸可以是字或长字。如果源操作数是字长度的则扩展为长字来进行操作。

条件码

X	N	Z	V	C
—	*	*	*	*

X 不受影响
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 结果若溢出则置位，否则清零
C 结果有借位则置位，否则清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	地址寄存器 Ax			操作模式			源有效地址					
										模式			寄存器		

指令域

地址寄存器域——指定目的寄存器 Ax。

操作模式域

字节	字	长字	操作
—	011	111	Ax-<ea>y

源有效地址域——指定源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	001	寄存器数:Ay
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

CMPI CMPI

Compare Immediate

首先出现于 ISA_A

B 和 W 首先出现于 ISA_B

指令操作

目的－立即数→cc

汇编格式

CMPI.sz #<date>,Dx

相关属性

尺寸 = 字节、字、长字长度（从 ISA_B 开始支持字节长度、字长度）。

指令描述

操作近似于 CMP，适用于源操作数是立即数时。操作数尺寸可以是字节、字、长字长度。立即数的尺寸与操作数的尺寸相匹配。注意：如果尺寸=字节，立即数包含在独立扩展字的位[7：0]。如果尺寸=字，立即数包含在独立扩展字的位[15：0]。如果尺寸=长字，立即数包含在两个扩展字，第一个扩展字位[15：0]，包含高字，第二个扩展字位[15：0]，包含低字。

条件码

X	N	Z	V	C
—	*	*	*	*

X 不受影响
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 结果若溢出则置位，否则清零
C 结果有借位则置位，否则清零

指令格式

15	14	13	12	11	10	9	8	7 6	5	4	3	2 1 0
0	0	0	0	1	1	0	0	Size	0	0	0	寄存器 Dx
立即数的高字												
立即数的低字												

指令域

寄存器域——指定目的寄存器 Dx。

尺寸域——指定操作的尺寸。

— 00：字节 — 01：字：

— 10：长字 — 11：翻转

DIVS DIVS

Signed Divide

首先出现于 ISA_A

在 MCF5202,MCF5204 和 MCF5206 中没有实现

指令操作

源/目的→目的

汇编格式

DIVS.W <ea>y,Dx　　32 位 Dx 或 Dx 的 16 位<ea>y (16r:16q)

DIVS.L <ea>y,Dx　　32 位 Dx 或 Dx 的 32 位<ea>y 32 q

（q 表示商，r 表示余数）

相关属性

尺寸 = 字、长字长度。

指令描述

用有符号的目的操作数除以有符号的源操作数，结果存放在目的操作符中。对以字为单位的操作，目的操作数是长字的而源操作数是字长度的。16 位的商在低字节中而 16 位的余数在高字节中。注意余数的正负与被除数相同。对以长字为单位的操作，目的操作数与源操作数是长字长度的。32 位的商存放于目的操作符中。如果想知道长字操作中的余数，可以使用 REMS 指令。 如果除数为零，自动使用特例状况，没有寄存器会被影响；如果结果溢出（商大于
操作的单位），目的寄存器不会被影响。

条件码

X	N	Z	V	C
—	*	*	*	0

X　不受影响

N　出现溢出则清零，商为负置位，为正清零

Z　出现溢出则清零，商为零置位，非零清零

V　结果若溢出则置位，否则清零

C　清零

指令格式（字）

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	0	0	0	寄存器 Dx	1	1	1	源有效地址	
								模式	寄存器

指令域（字）

寄存器域——指定目的寄存器 Dx。

源有效地址域——指定源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

指令格式（长字）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	源有效地址					
										模式			寄存器		
0	寄存器 Dx			1	0	0	0	0	0	0	0	0	寄存器 Dx		

指令域（长字）

寄存器域——指定目的寄存器 Dx，注意这个域在指令格式中出现两次。 源有效地址域——指定源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	—	—

DIVU DIVU

Unsigned Divide

首先出现于 ISA_A

在 MCF5202、MCF5204 和 MCF5206 中没有实现

指令操作

源/目的→目的

汇编格式

DIVU.W <ea>y,Dx　　32 位 Dx 或 Dx 的 16 位<ea>y (16r:16q)

DIVU.L <ea>y,Dx　　32 位 Dx 或 Dx 的 32 位<ea>y 32 q

（q 表示商，r 表示余数）

相关属性

尺寸 = 字、长字长度。

指令描述

用无符号的目的操作数除以无符号的源操作数，结果存放在目的操作数中。对以字为单位的操作，目的操作数是长字的而源操作数是字长度的。16位的商在低字节中，而16位的余数在高字节中。对以长字为单位的操作，目的操作数与源操作数是长字长度的。32位的商存放于目的操作符中。如果想知道长字操作中的余数，可以使用REMU指令。

如果除数为零，自动使用特例状况，没有寄存器会被影响；如果结果溢出（商大于操作的单位），目的寄存器不受影响。

条件码

X	N	Z	V	C
—	*	*	*	0

X　不受影响

N　出现溢出则清零，否则商为负置位，为正清零

Z　出现溢出则清零，否则商为零置位，非零清零

V　结果若溢出则置位，否则清零

C　清零

指令格式（字）

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	0	0	0	寄存器 Dx	0	1	1	源有效地址	
								模式	寄存器

指令域（字） 寄存器域——指定目的寄

存器 Dx。

源有效地址域——指定源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

指令格式（长字）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	源有效地址					
										模式			寄存器		
0	寄存器 Dx			0	0	0	0	0	0	0	0	0	寄存器 Dx		

指令域（长字）

寄存器域——指定目的寄存器 Dx，注意这个域在指令格式中出现两次。 源有效地址域——指定源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	—	—

EOR EOR

Exclusive-OR Logical

首先出现于 ISA_A

指令操作

源∧目的→目的

汇编格式

EOR.L Dy,<ea>x

指令描述 对目的操作数与源操作数进行“或”操作，结果储存在目的操作符中。操作数尺寸被指定为长字。源操作数必须放在数据寄存器中。目的操作数默认在有效的地址域中。当源是立即数时使用 EORI。

条件码

X	N	Z	V	C
—	-|	-|	0	0

X 不受影响
N 结果的最高位置位则置位，否则清零
Z 结果为零数则置位，否则清零
V 总是清零
C 总是清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	寄存器			1	1	0	有效地址					
										模式			寄存器		

指令域

寄存器域——指定数据寄存器。 有效地址域——用于决定寻址方式。 对于目的操作数<ea>x，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dx
Ay	001	寄存器数:Ax
(Ay)	010	寄存器数:Ax
(Ay)+	011	寄存器

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

		数:Ax
-(Ay)	100	寄存器 数:Ax
(d_{16},Ay)	101	寄存器 数:Ax
(d_8,Ay,Xi)	110	寄存器 数:Ax

EORI Exclusive-OR Immediate EORI

首先出现于 ISA_A

指令操作

立即数∧目的→目的

汇编格式

EORI.L #<data>,Dx

指令描述 对目的操作数与立即数进行“或”操作，结果储存在目的操作符中。操作数尺寸被指定为长字。目的操作数默认在有效的地址域中。注意，该立即数是包含在两个被扩展的字中，换言之，第一个扩展字的[15:0]位载有高字，第二个扩展字的[15:0]位载有低字。

条件码

X	N	Z	V	C
—	-|	-|	0	0

X 不受影响
N 结果的最高位置位则置位，否则清零
Z 结果为零数则置位，否则清零
V 总是清零
C 总是清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	0	0	0	0	寄存器 Dx		
立即数的高位															
立即数的低位															

指令域

寄存器域——目的数据寄存器 Dx。

EXT,EXTB　　Sign-Extend　　EXT,EXTB

首先出现于 ISA_A

指令操作

扩展标志+目的→目的

汇编格式

EXT.W Dx	扩充字节到字
EXT.L Dx	扩充字到长字
EXTB.L Dx	扩充字节到长字

相关属性

尺寸 = 字、长字长度。

指令描述

扩展数据寄存器 Dx 中的一个字节，变成字或长字或者把在数据寄存器中的一个字变成长字，把符号位复制到左边。当 EXT 操作把一个字节扩展为字时，指定的数据寄存器的 7 位被复制到数据寄存器的 15-8 位。当 EXT 操作把一个字扩展为长字时，指定的数据寄存器的 15 位被复制到数据寄存器的 31-16 位。EXTB 形式把指定的寄存器的 7 位复制到数据寄存器的 31-8 位。

条件码

X	N	Z	V	C
—	-|	-|	0	0

X　不受影响
N　结果为负数则置位，否则清零
Z　源操作数为零数则置位，否则清零
V　总是清零
C　总是清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	操作模式		0	0	0	0	寄存器 Dx		

指令域

操作模式域——指定有符号扩展操作数的尺寸。

— 010：把数据寄存器中有符号扩展的低位变成字

— 011：把数据寄存器中有符号扩展的低字变成长字

— 111：把数据寄存器中有符号扩展的低位变成长字 寄存器域——指定数据寄存器 Dx，变成有符号的扩展数。

FF1 FF1

Find First One in Register

首先出现于 ISA_C

指令操作

寄存器中第一个逻辑数的位移→目的

汇编格式

FF1.L Dx

相关属性

尺寸 = 长字长度。

指令描述

浏览数据寄存器 Dx，从最重要的 Dx[31]到最不重要的 Dx[0],寻找第一个非零的位。然后这个数据寄存器记下那个位的偏移。如果源数据是 0，那将得到 32 的偏移量。

表格 **0—1**

Old Dx[31:0]	New Dx[31:0]
0x8000 0000	0x0000 0000
0x4000 0000	0x0000 0001
0x2000 0000	0x0000 0002
...	...
0x0000 0002	0x0000 001E
0x0000 0001	0x0000 001F
0x0000 0000	0x0000 0020

条件码

X	N	Z	V	C
—	-|	-|	0	0

X 不受影响
N 源操作数的最高位置位则置位，否则清零
Z 源操作数为零数则置位，否则清零
V 总是清零
C 总是清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	1	0	0	0	目的寄存器 Dx		

ILLEGAL ILLEGAL

Take Illegal Instruction Trap

首先出现于 ISA_C

指令操作

SP - 4 → SP; PC → (SP) (迫使堆栈以长字排列)
SP - 2 → SP; SR → (SP)
SP - 2 → SP; 向量偏移 → (SP)
(VBR + 0x10) → PC

汇编格式

ILLEGAL

指令描述

执行这条指令造成“非法”异常处理，它的操作模式是 0x4AFC。

以 ISA_B 开始（对于有 MMU 的设备），主要的栈指针被用做指令。

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	1	1	1	1	0	0

JMP JMP

Jump

首先出现于 ISA_A

指令操作

目的地址 → PC

汇编格式

JMP <ea>y

相关属性 无固定尺寸。

指令描述 程序将执行此指令包含的有效地址所指向的指令。

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	1	有效源地址					
										模式			寄存器		

指令域

有效源地址域——决定下一指令的地址。 对于<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	-----	-----
Ay	-----	-----
(Ay)	010	寄存器数:Ay
(Ay)+	-----	-----
-(Ay)	-----	-----
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	-----	-----
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

JSR

Jump to Subroutine

JSR

首先出现于 ISA_A

指令操作

SP － 4 → SP; nextPC → (SP); 目的地址 → PC

汇编格式

JSR <ea>y

相关属性 无固定

尺寸。

指令描述

这个 JSR 指令当被传送给系统时，就把指令的尺寸为长字的地址进栈，然后程序将执行此指令包含的有效地址所指向的指令。

条件码 不受影

响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	源有效地址					
										模式			寄存器		

指令域

源有效地址域——指定下一条指令的地址<ea>y，会用到以下表格的控制寻址方式：

寻址方式	模式	寄存器
Dy	—	—
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	—	—
-(Ay)	—	—
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

LEA

Load Effective Address

首先出现于 ISA_A

LEA

指令操作

<ea>y→Ax

汇编格式

LEA.L <ea>y,Ax

相关属性

尺寸 = 长字长度。

指令描述

把需要的有效地址存入指定的有效地址寄存器 Ax 中。

条件码 不受影响。

指令格式

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	1	0	0	寄存器 Ax	1	1	1	源有效地址	
								模式	寄存器

指令域

寄存器域——指定地址寄存器 Ax 随有效地址更新。 源有效地址域——指定装入目的地址寄存器的地址，会用到下面表格的寻址方式：

寻址方式	模式	寄存器
Dy	—	—
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	—	—
-(Ay)	—	—
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

LINK LINK

Load and Allocate

首先出现于 ISA_A

指令操作

SP － 4 →SP; Ay → (SP);SP → Ay;SP + dn → SP

汇编格式

LINK.W Ay,#<displacement>

相关属性

尺寸 = 字长度。

指令描述

将指定地址寄存器中的内容压入堆栈，然后将新堆栈的指针装入地址寄存器，最后将偏移值加入堆栈指针。偏移是操作字后的标记扩展字。注意，尽管 LINK 指令是以个字长指令，但大多数汇编器也支持不限定尺寸的 LINK 指令。

条件码 不受影

响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	寄存器 Ay		
字偏移															

指令域

寄存器域——为 LINK 指令指定地址寄存器 Ay。

偏移域——指定加入堆栈指针的二进制补码。

LSL,LSR　　Logical Shift　　LSL,LSR

首先出现于 ISA_A

指令操作

根据计数转换的目的 → 目的

汇编格式

LSd.L Dy,Dx

LSd.L #<date>,Dx

（d 是方向，L 或 R）

相关属性

尺寸 = 长字长度。

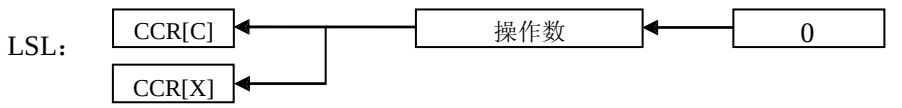
指令描述

转换由方向器（L 或 R）给出的目的操作数的位数。操作数尺寸是长字。CCR[C]存放操作数移出的最后一位。转换的位数是转换的目的寄存器数位位置数，可以以两种不同的方式指定：

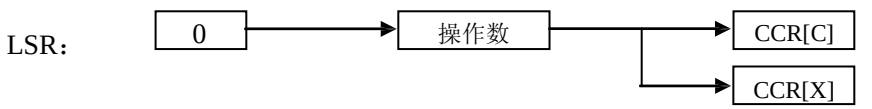
1 立即数——移位数在指令中被指定（移位范围 1-8）。

2 寄存器——移位数是数据寄存器 Dy 的值，在指令中被指定（以 64 为模）。

LSL 指令把操作数移入指定移位数位置数的左边。从高位按顺序移出的位转到进位和扩展位中；零被按顺序移入低位。



LSR 指令把操作数移入指定移位数位置数的右边。从低位按顺序移出的位转到进位和扩展位中；零被按顺序移入高位。



条件码

X	N	Z	V	C
*	*	*	0	*

X　按照操作数被移出的最后一位置位，不受零计数器变化的影响

N　结果为负数则置位，否则清零

Z　结果为零数则置位，否则清零

V 清零

C 按照操作数被移出的最后一位置位，零计数器变化则清零

指令格 ：

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	1	1	0	计数寄存器 Dy	dr	1	0	i/r	0	1	寄存器 Dx

指令域

计数或寄存器域——指定或寄存器 Dy，包含。

— 如果 i/r = 0，此域包含移位数，数字 1—7 代表 1—7 的转换；数字 0 指定 8 的转换计数。

— 如果 i/r = 1，此域数据寄存器 Dy 包含转换计数（以 64 为模）

dr 域——指定转换的方向。

— 0：右进位

— 1：左进位

i/r 域

— 如果 i/r=0，指定立即转换数。

— 如果 i/r=1，指定寄存器转换数。 寄存器

域——指定要转换的数据寄存器 Dx。

MOV3Q　　Move 3-Bit Date Quick　　MOV3Q

首先出现于 ISA_B

指令操作

3 位立即数→目的

汇编格式

MOV3Q.L #<date>,<ea>x

相关属性

尺寸 = 长字长度。

指令描述

把立即数移到目的位置的操作数里。数据范围从-1 到 7，不包括 0.。三位立即操作数被标记扩展为长字操作数，所有的 32 位被转到目的位置。

条件码

X	N	Z	V	C
—	*	*	0	0

X　不受影响
N　结果为负数则置位，否则清零
Z　结果为零数则置位，否则清零
V　清零
C　清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	立即数			1	0	1	目的有效地址					
										模式			寄存器		

指令域

立即数域—3 位数有（-1，1-7）列，这里数字 0 表示-1。 目的有效地址域——指定目的操作数<ea>x，仅用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dx	000	寄存器数:Dx
Ax	001	寄存器数:Ax
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

		数:Ax
-(Ax)	100	寄存器 数:Ax
(d_{16},Ax)	101	寄存器 数:Ax
(d_8,Ax,Xi)	110	寄存器 数:Ax

MOVE MOVE

Move Date from Source to Destination

首先出现于 ISA_A

指令操作

源→目的

汇编格式

MOVE.L <ea>y,<ea>x

相关属性

尺寸 =字节、字、长字长度。

指令描述

将源数据放到目的位置，并按此数据设置条件码。操作尺寸可以是字节、字或长字。当目的是一个地自寄存器时使用MOVEA指令。MOVEQ指令是将8位的立即数放到一个数据寄存器。MOVE3Q指令（从ISA_B支持）用于将一个3位的立即数放到任何有效目的地址。

条件码

X	N	Z	V	C
—	*	*	0	0

X 不受影响
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 清零
C 清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	尺寸		目的有效地址						源有效地址					
				寄存器			模式			模式			寄存器		

指令域

尺寸域—指定被移动的操作数的尺寸。

— 01：字节

— 11：字

— 10：长字

目的有效地址域——指定目的<ea>x 位置，下表列出了可能的数据可变寻址方式，在下一页底部的表中列出了源和目的合并的规则。

寻址方式	模式	寄存器
Dx	000	寄存器数:Dx
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	110	寄存器数:Ax

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

源有效地址域——指定源操作数<ea>y，下表列出了可能的寻址方式。在下一页底部的表中列出了源和目的合并的规则。

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	001	寄存器数:Ay
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

注意：

并非所有的源或目的寻址方式的合并都是可能的。 下表给出了可能的合并方式。

从 ISA_B 开始，#<xxx> 和 d_{16}(Ax)的合并可以使用 MOVE.B 和 MOVE.W 操作代码。

源寻址方式	目的寻址方式
Dy, Ay,(Ay), (Ay)+,-(Ay)	全部可能
(d_{16},Ay) ,(d_{16},PC)	除(d_8,Ax,Xi),(xxx).W,(xxx).L 外全部可能
(d_8,Ay,Xi),(d_8,PC,Xi),(xxx).W,(xxx).L,#<xxx>	除(d_{16},Ax) ,(d_8,Ax,Xi),(xxx).W,(xxx).L 外全部可能

MOVEA Move Address from Source to Destination MOVEA

首先出现于 ISA_A

指令操作

源→目的

汇编格式

MOVEA.sz <ea>y,Ax

相关属性

尺寸 = 字、长字长度。

指令描述

将源地址放到目的地址寄存器里。操作的尺寸可以是字或长字。字长操作数在操作执行前可以扩展到32位。

条件码 不受影

响。

指令格式

15	14	13 12	11 10 9	8	7	6	5 4 3	2 1 0
0	0	尺寸	目的寄存器 Ax	0	0	1	源有效地址	
							模式	寄存器

指令域

尺寸域—指定被移动的操作数的尺寸。

— 01：字节 — 11：字 — 10：长字 目的寄存器域—指定目的地址寄存器 Ax。 源有效地址域——指定源操作数<ea>y，下表列出了可能的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	001	寄存器数:Ay
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

		数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

MOVEM Move Multiple Registers MOVEM

首先出现于 ISA_A

指令操作

寄存器→目的
源→寄存器

汇编格式

MOVEM.L #list,<ea>x
MOVEM.L <ea>y,#list

相关属性 尺寸=长字

长度。

指令描述 将选定的寄存器的内容放入从被有效地址指定单元开始的连续储存单元，或从中取出。如果与掩码相对应的寄存器的位被置位，则寄存器被选定。

寄存器被转移到被指定地址开始的空间，地址根据操作数的长度（4）在每次转移后作相应增加。寄存器的次序从 D0 至 D7，然后从 A0 至 A7。

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	dr	0	0	1	1	有效地址					
										模式			寄存器		
寄存器列Mask															

指令域

dr 域——指定转换方向。

— 0:寄存器到内存

— 1:内存到寄存器 有效地址域——为数据转化指定内存地址。

对于目的操作数<ea>x，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dx	—	—
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	—	—
-(Ax)	—	—
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

对于源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	—	—
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	—	—
-(Ay)	—	—
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	111	100
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

寄存器列掩码域——指定转换的寄存器,低位与第一个被转换的寄存器一致，高位与最后一个被转换的寄存器一致。Mask 见下表：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
A7	A6	A5	A4	A3	A2	A1	A9	D7	D6	D5	D4	D3	D2	D1	D0

MOVEQ MOVEQ

Move Quick

首先出现于 ISA_A

指令操作

立即数→目的

汇编格式

MOVEQ.L #<data>,Dx

相关属性 尺寸=长字

长度。

指令描述

将立即数的一个字节移入 32 位数据寄存器 Dx.。在转移的过程中，数据寄存器里的 8 位域的数据，在操作字之内，被扩展成有符号长字操作数。

条件码

X	N	Z	V	C
—	*	*	0	0

X 不受影响
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 总是为零
C 总是为零

指令格式

15	14	13	12	11 10 9	8	7 6 5 4 3 2 1 0
0	1	1	1	寄存器 Rx	0	立即数

指令域

寄存器域——指定数据寄存器 Dx，并对其进行操作。

数据域——8 位的数被扩展为有符号的长字。

MOVE from CCR

Move from the Condition Code Register

首先出现于 ISA_A

MOVE from CCR

指令操作

CCR→目的

汇编格式

MOVE.W CCR,Dx

相关属性 尺寸=字长度。

指令描述

将条件码单位（零扩展为字大小）移入目的单元 Dx。操作数的大小为字，当读入零时位不执行。

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	1	1	0	0	0	寄存器 Rx		

指令域

寄存器域——指定目的数据寄存器 Dx。

MOVE to CCR — Move to the Condition Code Register — MOVE to CCR

首先出现于 ISA_A

指令操作 源

→CCR

汇编格式

MOVE.B Dy,CCR

MOVE.B #<data>,CCR

相关属性 尺寸=字节

长度。

指令描述 将源操作数的低字节按顺序移入条件码寄存器。源操作数的高字节被忽略，状态寄存器的高字节没有改变。

条件码

X	N	Z	V	C
*	*	*	*	*

X 置于源操作数的第 4 位
N 置于源操作数的第 3 位
Z 置于源操作数的第 2 位
V 置于源操作数的第 1 位
C 置于源操作数的第 0 位

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	源有效地址					
										模式			寄存器		

指令域

有效地址域——指定源操作数的位置，只使用下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	—	—
(Ay)	—	—
(Ay)+	—	—
-(Ay)	—	—
(d_{16},Ay)	—	—
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	111	100
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

MULS MULS

Signed Multiply

首先出现于 ISA_A

指令操作

源 * 目的 → 目的

汇编格式

MULS.W <ea>y,Dx	16 x 16 → 32
MULS.L <ea>y,Dx	32 x 32 → 32

相关属性

尺寸 = 字节或字的长度。

指令描述

两个有符号的操作数相乘得到一个有符号的结果。这个指令有字操作数和长字操作数两种形式。

当为字操作数形式时，乘数和被乘数都是字操作数，结果是长字操作数。寄存器操作数是按顺序存放的低字；寄存器的高字被忽略。结果的所有 32 位都被存放在目的数据寄存器中。

当为长字操作数形式时，乘数和被乘数都是长字操作。目的数据寄存器按顺序储存结果的低 32 位，结果的高 32 位被丢弃。

注意 CCR[V]总被 MULS 清零，与 68K 系列的处理器不同。

条件码

X	N	Z	V	C
—	*	*	0	0

X 不受影响
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 总是为零
C 总是为零

指令格式（字）

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	1	0	0	寄存器 Dx	1	1	1	有效地址	
								模式	寄存器

指令域（字）

寄存器域——指定目的数据寄存器 Dx。

有效地址域——用于决定寻址方式。对于源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

指令格式（长字）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	有效地址					
										模式			寄存器		
0	寄存器 Dx			1	0	0	0	0	0	0	0	0	0	0	0

指令域（长字）

源有效地址域——用于决定寻址方式。对于源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

寄存器域——指定数据寄存器 Dx 作为目的操作数；32 位的被乘数来自寄存器，积的低 32 位存入这个寄存器中。

MULU

Unsigned Multiply

首先出现于 ISA_A

MULU

指令操作

源*目的→目的

汇编格式

```
MULU.W <ea>y,Dx      16 x 16 → 32
MULU.L <ea>y,Dx      32 x 32 → 32
```

相关属性 尺寸=字节或字

长度。

指令描述 两个无符号的操作数相乘得到一个无符号的结果。这个指令有字操作数和长

字操作

数两种形式。

当为字操作数形式时，乘数和被乘数都是字操作数，结果是长字操作数。寄存器操作数是按顺序存放的低字；寄存器的高字被忽略。结果的所有 32 位都被存放在目的数据寄存器中。

当为长字操作数形式时，乘数和被乘数都是长字操作。目的数据寄存器按顺序储存结果的低 32 位，结果的高 32 位被丢弃。

注意 CCR[V]总被 MULU 清零，这与 68K 系列的处理器不同。

条件码

X	N	Z	V	C
—	*	*	0	0

X 不受影响
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 总是为零
C 总是为零

指令格式（字）

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	1	0	0	寄存器 Dx	0	1	1	有效地址	
								模式	寄存器

指令域（字）

寄存器域——指定目的数据寄存器 Dx。

有效地址域——用于决定寻址方式。对于源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

指令格式（长字）

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	有效地址					
										模式			寄存器		
0	寄存器 Dx			0	0	0	0	0	0	0	0	0	0	0	0

指令域（长字）

源有效地址域——用于决定寻址方式。对于源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

寄存器域——指定数据寄存器 Dx 作为目的操作数；32 位的被乘数来自寄存器，积的低 32 位存入这个寄存器中。

MVS　　MVS

Move with Sign Extend

首先出现于 ISA_A

指令操作

有符号扩展的源→目的

汇编格式

MVS.sz <ea>y,Dx

相关属性 尺寸=字节或字

长度。

指令描述

将有符号的源操作数扩展并移入目的寄存器。对于字节操作数，源的 7 位被复制到目的的 31-8 位；对于字操作数，源的 15 位被复制到目的的 31-16 位。

条件码

X	N	Z	V	C
—	*	*	0	0

X　不受影响
N　结果为负数则置位，否则清零
Z　结果为零数则置位，否则清零
V　总是清零
C　总是清零

指令格式

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	1	1	1	寄存器 Dx	1	0	Data	有效地址	
								模式	寄存器

指令域

寄存器域——指定数据寄存器 Dx。

size 域——指定操作数的大小。— 0：字节操作数　　— 1：字操作数 源有效地址域——用于决定源操作数<ea>y 的寻址方式，会用到以下的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	001	寄存器数:Ay
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

MVZ　　MVZ

Move with Zero_Fill

首先出现于 ISA_A

指令操作

零源→目的

汇编格式

MVZ.sz <ea>y,Dx

相关属性 尺寸=字节

或字。

指令描述

填零源操作数并能将其移入目的寄存器。对于字节操作数，源操作数被移入目的 7-0 位，31-8 全部填零；对于字操作数，源操作数被移入目的 15-0 位，31-16 全部填零。

条件码

X	N	Z	V	C
—	0	*	0	0

X　不受影响
N　总是清零
Z　结果为零数则置位，否则清零
V　总是清零
C　总是清零

指令格式

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	1	1	1	寄存器 Dx	1	0	Data	有效地址	
								模式	寄存器

指令域

寄存器域——指定数据寄存器 Dx。

size 域——指定操作数的大小。 — 0：字节操作数 — 1：字操作数 源有效地址域——用于决定源操作数<ea>y 的寻址方式，会用到以下寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	001	寄存器数:Ay
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

NEG NEG

Negate

首先出现于 ISA_A

指令操作

0-目的→目的

汇编格式

NEG.L Dx

相关属性

尺寸 = 长字的长度。

指令描述

从零中减去目的操作数，将结果存入目的单元。操作数的大小被指定为长字。

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	0	0	0	0	寄存器 Rx		

指令域

寄存器域——指定目的数据寄存器 Dx。

NEGX　　NEGX

Negate with Extend

首先出现于 ISA_A

指令操作

0 – 立即数 – CCR[X] → 目的

汇编格式

NEGX.L Dx

相关属性 尺寸=长字

长度。

指令描述 从零中减去目的操作数和CCR[X]，将结果存入目的单元。操作数的大小被指定为

长字。

条件码

X	N	Z	V	C
*	*	*	*	*

X 与进位标志一致
N 结果为负数则置位，否则清零
Z 结果无零数则清零，否则不变
V 若结果溢出则置位，否则清零
C 若产生借位则置位，否则清零

通常在 NEGX 操作开始之前 CCR[Z]会通过编程来明确，从而允许在多倍精度操作完成基础上的对零结果的成功调试。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	0	0	0	0	寄存器 Dx		

指令域

寄存器域——指定数据寄存器 Dx。

NOP NOP

No Operation

首先出现于 ISA_A

指令操作

无

汇编格式

NOP

相关属性 无固定尺寸。

指令描述 不执行任何操作，除了程序计数器，处理器的状态不受任何影响，NOP之后的指令继续执行；直到所有未知的循环完成，传递途径同步，预防指令重迭实现后开始执行NOP指令。 因为NOP指令是被指定执行传递途径同步的，虽然没有任何操作，执行时间是多重循环的。假设只有一个代码队列，使用TPF指令更好，它执行一次操作，没有操作指令。NOP的操作模式是0x4E71。

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

NOT NOT

Logical Complement

首先出现于 ISA_A

指令操作

~目的→目的

汇编格式

NOT.L Dx

相关属性 尺寸=长字

长度。

指令描述 计算运算数存储器的补码，将结果存入目的单元。操作数的大小被指定位长字。

条件码

X	N	Z	V	C
—	*	*	0	0

X 不受影响
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 总是清零
C 总是清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	0	0	0	0	寄存器 Rx		

指令域

寄存器域——指定数据寄存器 Dx。

OR

Inclusive-OR Logical

首先出现于 ISA_A

OR

指令操作

源|目的→目的

汇编格式

OR.L <ea>y,Dx

OR.L Dy,<ea>x

相关属性 尺寸=长字

长度。

指令描述

在源操作数和目的操作数上执行包括 OR 在内的操作，将结果存入目的单元。操作数的大小被指定为长字。地址寄存器的内容将不做操作数使用。

当目的是数据寄存器时，使用 Dx 模式，目的<ea>模式对数据寄存器不起作用。

另外当源是立即数时需要使用 OGI。

条件码

X	N	Z	V	C
—	*	*	0	0

X 与进位位的值相同
N 若最高位被置位则置位，否则清零
Z 结果为零数则置位，否则清零
V 总是清零
C 总是清零

指令格式

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	0	0	寄存器	操作模式	有效地址	
						模式	寄存器

指令域

寄存器域——指定数据寄存器。

操作模式域

字节	字	长字	操作
—	—	010	<ea>y | Dx → Dx
—	—	110	Dy | <ea>x → <ea>x

有效地址域——用于决定寻址方式。

对于源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

对于目的操作数<ea>x，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dx	—	—
Ax	—	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	110	寄存器数:Ax

寻址方式	模式	寄存器
(xxx).W	111	寄存器数:Dy
(xxx).L	111	寄存器数:Ay
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

ORI

Inclusive-OR

首先出现于 ISA_A

ORI

指令操作

立即数|目的→目的

汇编格式

ORI.L #<data>,Dx

相关属性 尺寸=长字

长度。

指令描述

在立即数和目的操作数上执行包括 OR 在内的操作，将结果存入目的数据寄存器 Dx。操作数的大小被指定为长字，立即数的大小被指定为长字。注意：立即数包括在两个扩展字之内，第一个扩展字为位[15：0]，包括高字；第二个扩展字为位[15：0]，包括低字。

条件码

X	N	Z	V	C
—	*	*	0	0

X 不受影响

N 若最高位被置位则置位，否则清零

Z 结果为零数则置位，否则清零

V 总是清零

C 总是清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	寄存器 Rx		
立即数的高位															
立即数的低位															

指令域

目的寄存器域——指定数据寄存器 Dx。

PEA

Push Effective Address

首先出现于 ISA_A

PEA

指令操作

SP – 4 → SP; <ea>y → (SP)

汇编格式

PEA.L <ea>y

相关属性 尺寸=长字

长度。

指令描述 计算有效地址并将其推入堆栈，有效地址是长字

地址。

条件码 不受影

响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	有效地址					
										模式			寄存器		

指令域

源有效地址域——用于决定源操作数<ea>y 的寻址方式然后压入堆栈，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	—	—
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	—	—
-(Ay)	—	—
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

PULSE PULSE

Generate Unique Processor Status

首先出现于 ISA_A

指令操作

置 PST 为 0x4

汇编格式

PULSE

相关属性 无固定

尺寸。

指令描述 不执行任何操作，除了程序计数器，处理器的状态不受任何影响。然而，

PULSE

产生了一个特殊处理机状态（PST）输出信号译码，它对外触发器的用途很有用。PULSE 的操作码是0x4ACC。

条件码 不受影

响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	1	1	1	0	0	1	1	0	0

REMU REMU

Unsigned Divide Remainder

首先出现于 ISA_A

在 MCF5202, MCF5204 和 MCF5206 中没有实现

指令操作

目的/源→余数

汇编格式

REMU.L <ea>y,Dw:Dx　　32-bit Dx/32-bit <ea>y Æ 32r in Dw（r 指余数）

指令描述

用有符号的目的操作数除以有符号的源，有符号的余数存入另一个寄存器。如果 Dw 被指定为和 Dx 相同的寄存器，则将执行 DIVS 指令而不执行 REMS 指令；要确定商的值则使用 DIVS 指令。

如果除数是 0 则结果是另外的，没有寄存器受到影响。因而发生的中断，堆栈结构指向 REMS 操作模式。如果发现溢出，目的寄存器则不受影响，当商大于 32 位有符号整数时发生溢出。

条件码

X	N	Z	V	C
—	*	*	*	0

X　不受影响

N　若溢出则清零，否则商为负置位，商为正清零

Z　若溢出则清零，否则商为零置位，商不为零清零

V　若发生溢出则置位，否则清零

C　总是清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	有效源地址					
										模式			寄存器		
0	寄存器 Dx			0	0	0	0	0	0	0	0	0	寄存器 Dw		

指令域

寄存器 Dx 域——指定目的寄存器 Dx。 有效地址域——指定源操作数 <ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

寄存器 Dw 域——指定余数寄存器 Dw。

RTS RTS

Return from Subroutine

首先出现于 ISA_A

指令操作

(SP) → PC; SP + 4 → SP

汇编格式

RTS 相关

属性

没有大小。

指令描述

将程序计数器的值从堆栈中取出，在程序寄存器前的值则丢失。RTS 的操作模式是 0x4E75。

条件码 不受影

响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

SATS Signed Saturate SATS

首先出现于 ISA_B

指令操作

```
If CCR[V] = = 1,
    then if Dx[31] = = 0,
        then Dx[31:0] = 0x80000000
        else Dx[31:0] = 0x7FFFFFFF
    else Dx[31:0] is unchanged
```

汇编格式

```
SATS.L Dx
```

指令描述

当且仅当 CCR 的溢出位被置时更新目的寄存器。如果操作数是负数，将结果设为最大正数；否则将结果设为最大的负数值。条件码根据结果进行指定。

条件码

X	N	Z	V	C
—	*	*	0	0

X 不受影响
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 总是清零
C 总是清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	1	0	0	0	0	寄存器 Dx		

指令域

寄存器域——指定目的数据寄存器 Dx。

Scc　　Set According to Condition　　Scc

首先出现于 ISA_A

指令操作

如果条件为真则 1s → 目的，反之 0s → 目的

汇编格式

Scc.B Dx

相关属性 尺寸=长字

长度。

指令描述

测试指定的条件码，如果条件为真，将目的数据寄存器的最地位设为 TRUE（全部置一），否则设为 FALSE（全部置零）。条件码 cc 指定下列有条件测试中的一种，这里 C，N，V 和 Z 分别代表 CCR[C]，CCR[N]，CCR[V]和 CCR[Z]：

代码	条件	编码	测试
CC(HS)	进位清零	0100	
CS(LO)	进位置位	0101	C
EG	相等	0111	Z
F	错误	0001	0
GE	大于等于	1100	N&V|&
GT	大于	1110	N&V&|&&
HI	大于	0010	&
LE	小于等于	1111	Z|N&|V

代码	条件	编码	测试
LS	小于等于	0011	C|Z
LT	小于	1101	N&|&
MI	负数	1011	N
NE	不相等	0110	
PL	正数	1010	
T	真	0000	1
VC	溢出位清零	1000	
VS	溢出位置位	1001	V

条件码

不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	条件				1	1	0	0	0	寄存器 Dx		

指令域

条件域——为表格中的的一种条件指定二进制码。

寄存器域——指定目的数据寄存器 Dx。

SUB SUB

Subtract

首先出现于 ISA_A

指令操作

目的 - 源→目的

汇编格式

SUB.L <ea>y,Dx

SUB.L Dy,<ea>x

相关属性 尺寸=长字

长度。

指令描述 从目的操作数中减去源操作数，结果存入目的寄存器。操作数的大小被指定为长字。

指令的模式指出哪个是源操作数，哪个是源操作数。

当目的是数据寄存器时使用 Dx 模式，目的<ea>模式对数据寄存器是无效的。 另外，当目的是地址寄存器时使用 SUBA；当源是立即数时使用 SUBI 和 SUBQ。

条件码

X	N	Z	V	C
*	*	*	*	*

X 与进位位的值相同则置位
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 结果若溢出则置位，否则清零
C 结果有进位则置位，否则清零

指令格式

15	14	13	12	11 10 9	8 7 6	5 4 3	2 1 0
1	0	0	1	寄存器	操作模式	有效地址	
						模式	寄存器

指令域

寄存器域——指定数据寄存器。

操作模式域

字节	字	长字	操作
—	—	010	<ea>y + Dx → Dx
—	—	110	Dy + <ea>x → <ea>x

有效地址域——决定寻址方式。

—对于源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	001	寄存器数:Ay
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

—对于目的操作数<ea>x，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dx	000	—
Ax	001	—
(Ax)	010	寄存器数:Ax
(Ax)+	011	寄存器数:Ax
-(Ax)	100	寄存器数:Ax
(d_{16},Ax)	101	寄存器数:Ax
(d_8,Ax,Xi)	110	寄存器数:Ax

寻址方式	模式	寄存器
(xxx).W	111	寄存器数:Dy
(xxx).L	111	寄存器数:Ay
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

SUBA SUBA

Subtract Address

首先出现于 ISA_A

指令操作

目的-源→目的

汇编格式

SUBA.L <ea>y,Ax

相关属性 尺寸=长字

长度。

指令描述

操作和 SUB 相似，但在目的是地址寄存器而不是数据寄存器时使用。从目的地址寄存器中减去源操作数，结果存入地址寄存器。操作数的大小被指定为长字。

条件码 不受影

响。

指令格式

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	0	0	1	目的寄存器 Ax	1	1	1	有效源地址	
								模式	寄存器

指令域

目的寄存器域——指定目的地址寄存器 Ax。 源有效地址域——指定源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	001	寄存器数:Ay
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

		数:Ay
(d_8,Ay,Xi)	110	寄存器 数:Ay

SUBI SUBI

Subtract Immediate

首先出现于 ISA_A

指令操作

目的-立即数→目的

汇编格式

SUBI.L #<data>,Dx

相关属性 尺寸=长字

长度。

指令描述

操作和 SUB 相似，但在源操作数是立即数时使用。从目的操作数中减去源立即数，结果存入目的数据寄存器 Dx。操作数的大小被指定为长字。注意：立即数包括在两个扩展字之内，第一个扩展字为位[15：0]，包括高字；第二个扩展字为位[15：0]，包括低字。

条件码

X	N	Z	V	C
*	*	*	*	*

X 与进位位的值相同
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 结果若溢出则置位，否则清零
C 结果有进位则置位，否则清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	寄存器 Rx		
立即数的高位															
立即数的低位															

指令域

目的寄存器域——指定目的数据寄存器域 Dx。

SUBQ SUBQ

Subtract Quick

首先出现于 ISA_A

指令操作

目的-立即数→目的

汇编格式

SUBQ.L #<data>,<ea>x

相关属性

尺寸 = 长字长度。

指令描述

操作和 SUB 相似，但在源操作数是 1 至 8 的立即数时使用。在目的单元里从操作数中减去立即数。操作数的大小被指定为长字。在被从目的中减去之前，立即数填零成一个长字。当加到地址寄存器时，条件码没有改变。

条件码

X	N	Z	V	C
*	*	*	*	*

X 与进位位的值相同
N 结果为负数则置位，否则清零
Z 结果为零数则置位，否则清零
V 结果若溢出则置位，否则清零
C 结果有进位则置位，否则清零

指令格式

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
0	1	0	1	数据	1	1	0	有效目的地址 模式	有效目的地址 寄存器

指令域

数据域——3 位立即数代表 8 个数值（0-7）；立即数值 1-7 代表 1-7，0 代表 8。
目的有效地址域——指定目的位置，只用到以下表格所列出的可变寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	001	寄存器数:Ay
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

		数:Ay
-(Ay)	100	寄存器 数:Ay
(d_{16},Ay)	101	寄存器 数:Ay
(d_8,Ay,Xi)	110	寄存器 数:Ay

SUBX SUBX

Subtract Extended

首先出现于 ISA_A

指令操作

目的-源- CCR[X]→目的

汇编格式

SUBX.L Dy,Dx

相关属性 尺寸=长字

长度。

指令描述

从目的操作数中减去源操作数 CCR[X]，结果存入目的单元。操作数的大小被指定为长字。

条件码

X	N	Z	V	C
*	*	*	*	*

X 与进位位的值相同
N 结果为负数则置位，否则清零
Z 结果不为零数则置位，否则不变
V 结果若溢出则置位，否则清零
C 结果有进位则置位，否则清零

通常在 SUBX 操作开始之前 CCR[Z]会通过编程来明确，从而允许在多倍精度操作完成基础上的对零结果的成功调试。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	0	1	寄存器 Dx			1	1	0	0	0	0	寄存器 Dy		

指令域

寄存器 Dx 域——指定目的数据寄存器域 Dx。

寄存器 Dy 域——指定源数据寄存器域 Dy。

SWAP SWAP

Swap Register Halves

首先出现于 ISA_A

指令操作

寄存器[31:16] ↔ 寄存器[15:0]

汇编格式

SWAP.W Dx

相关属性 尺寸=字

长度。

指令描述

交换数据寄存器中 16 位字(同分数)。

条件码

X	N	Z	V	C
—	*	*	0	0

X 不受影响
N 结果最高位被置则置位，否则清零
Z 结果为零数则置位，否则清零
V 总是清零
C 总是清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	寄存器 Dx		

指令域

寄存器域——指定目的数据寄存器 Dx。

TAS TAS

Test and Set an Operand

首先出现于 ISA_A

指令操作

测试的目的 → CCR; 1 → 目的的第七位字

汇编格式

TAS.B <ea>x

相关属性 尺寸=字

长度。

指令描述 测试并设置有效地址域的字节操作数。指令测试操作数的当前值并适当地指定

CCR[N]和 CCR[Z]。TAS 也按顺序指定操作数的高位。操作数使用读-改-写的存储周期来不中断地完成操作。指令支持对几个同等处理器使用标志。注意，与 68K 系列处理器不同，Dx 不支持寻址模式。

条件码

X	N	Z	V	C
—	*	*	0	0

X 不受影响
N 结果最高位被置则置位，否则清零
Z 结果为零数则置位，否则清零
V 总是清零
C 总是清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	有效目的地址					
										模式			寄存器		

指令域

有效目的地址域——指定目的位置<ea>x，可能会用到以下寻址方式：

寻址方式	模式	寄存器
Dy	—	—
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

		数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

TPF TPF

Trap False

首先出现于 ISA_A

指令操作

没有操作

汇编格式

```
TPF PC + 2 → PC
TPF.W #<data> PC + 4 → PC
TPF.L #<data> PC + 6 → PC
```

相关属性 尺寸=不确定，字或长字

长度。

指令描述

没有执行任何操作，TPT 能占用 16，32 或 48 位指令空间，有效地提供长度可变，单循环，没有操作的指令。当代码队列想得到时，与 NOP 指令相比，TPF 是首选的，因为 NOP 指令也同步处理器传递途径，在多循环后得到结果。

TPF{W，L}可以用来消除无条件转移，例如：

```
if (a == b)
    z = 1;
    else
        z = 2;
```

上述代码汇编成：

```
    cmp.l d0,d1             ; compare a == b
    beq.b label0            ; branch if equal
    movq.l #2,d2            ; z = 2
    bra.b label1            ; continue
label0:
        movq.l #1,d2        ; z = 1
label1
```

对于这种类型的次序，BRAW 指令可以用 TPF.W 或 TPF.L 操作模式代替（依靠在标志 10 指令的长度，既然这样，TPF.W 操作模式是适用的）。在首标志位的指令由于 TPF 指令扩展成字而被有效地隐藏，分歧完全被消除了。

条件码 不受影

响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	1	1	1	1	1	1	操作模式		
任意立即字															
任意立即字															

指令域

操作模式域——指定任意扩展字的数目。

— 010：一个扩展字

— 011：两个扩展字

— 100：没有扩展字

TRAP TRAP

Trap

首先出现于 ISA_A

指令操作

1 → S-Bit of SR
SP – 4 → SP; nextPC → (SP); SP – 2 → SP;
SR → (SP); SP – 2 → SP; Format/Offset → (SP);
(VBR + 0x80 + 4*n) → PC
（n 是 TRAP vector 数目）

汇编格式

TRAP #<vector>

相关属性 尺寸

不定。

指令描述

引起 TRAP #<vector>中断。TRAP 向量域与 4 相乘，然后把结果加到 0x80 上得到中断地址。中断地址加到 VBR 上来指出中断向量表。如果有 16 个向量，向量域的值可以从 0-15。

注意当 SR 被复制到中断堆栈结构中时，它表示 TRAP 指令开始执行时的值。在中断处理结束时，SR 将更新，T 位被清零，S 位被置位。

条件码 不受影

响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	Vector			

指令域

Vector 域——指定选用的陷阱向量 Vector。

TST TST

Test an Operand

首先出现于 ISA_A

指令操作

测试的源操作数→ CCR

汇编格式

TST.sz <ea>y

相关属性 尺寸=字节，字或长字

长度。

指令描述 将操作数与零比较，根据测试结果置条件码。操作数的大小被指定为字节，字或长

字。

条件码

X	N	Z	V	C
—	*	*	0	0

X 不受影响
N 操作数为负数则置位，否则清零
Z 操作数为零数则置位，否则清零
V 总是清零
C 总是清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	尺寸		目的有效地址					
										模式			寄存器		

指令域

尺寸域——指定操作数的大小。

— 00：字节操作数

— 01：字操作数

— 10：长字节操作数

— 11：字操作数 目的有效地址域——为目的操作数指定寻址方式<ea>x，

如以下表格所示：

寻址方式	模式	寄存器
Dy	000	寄存器数:Dy
Ay	001	寄存器数:Ay
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	111	100
(d_{16},PC)	111	010
(d_8,PC,Xi)	111	011

Ax 寻址方式只对字和长字操作数有效。

UNLK UNLK

Unlink

首先出现于 ISA_A

指令操作

Ax → SP; (SP) → Ax; SP + 4 → SP

汇编格式

UNLK Ax

相关属性 尺寸

不定。

指令描述 从指定的地址寄存器中装载堆栈指针，然后把从堆栈顶部取出的长字装载到地址寄存器。

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	寄存器 Ax		

指令域

寄存器域——为指令指定地址寄存器 Ax。

WDDATA Write to Debug Data WDDATA

首先出现于 ISA_A

指令操作

源→ DDATA 信号引脚

汇编格式

WDDATA.sz <ea>y

相关属性 尺寸=字节、字或长字

长度。

指令描述

这个指令取出被有效地址定义的操作数，并获取 ColdFire 调试模块中的数据来在 DDATA 输出引脚中显示。操作数的大小决定了在 DDATA 输出引脚中显示的半位元组的数目。调试模块结构/寄存器（CSR）状态的值不影响这条指令的操作。

这条指令的执行产生处理器状态编码，在引用的操作数在 DDATA 输出中显示前与 PULSE 指令（0x4）相匹配。

条件码 不受影

响。

指令格式

15	14	13	12	11	10	9	8	7 6	5 4 3	2 1 0
1	1	1	1	1	0	1	1	尺寸	源有效地址	
									模式	寄存器

指令域

尺寸域——指定操作数数据的大小。

— 00：字节操作数

— 01：字操作数

— 10：长字节操作数

— 11：保留 源有效地址域——确定源操作数<ea>y的寻址方式，被写入DDATA 信号引脚，只

会用到以下表格所列出的可变存储地址寻址方式：

寻址方式	模式	寄存器
Dy	—	—
Ay	—	—
(Ay)	010	寄存器数:Ay
(Ay)+	011	寄存器数:Ay
-(Ay)	100	寄存器数:Ay
(d_{16},Ay)	101	寄存器数:Ay
(d_8,Ay,Xi)	110	寄存器数:Ay

寻址方式	模式	寄存器
(xxx).W	111	000
(xxx).L	111	001
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

第 5 章 MAC 用户指令

本章描述了在 ColdFire 系列处理器中关于可选的乘法累加单元（MAC）的用户指令。为了方便指令的记忆，每条指令描述的讨论将按照字母顺序来讲述。

若要了解由增强乘法累加单元（EMAC）所实现的指令，请见第六章“EMAC 用户指令”。

MAC　　Multiply Accumulate　　MAC

指令操作

ACC + (Ry ＊Rx){<< | >>} SF → ACC

汇编格式

MAC.sz　Ry , {U , L},Rx.{U , L}SF

相关属性

尺寸 = 字或长字。

指令描述

两个 16 或 32 位的数相乘得到 32 位的数，将它与累加器相加，其和由比例因子按照定义进行转换后送到一个累加器中。如果操作数是 16 位，那么每个寄存器的高字或低字必须被指定。

条件码**(MACSR)**

N	Z	V
*	*	*

N　结果的最高位被置则置位，否则清零
Z　结果为零则置位，否则清零
V　若溢出位被置则置位，否则不变

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx			0	0	Rx	0	0	寄存器 Ry			
—	—	—	—	sz	比例因子		0	U/Lx	U/Ly	—	—	—	—	—	—

指令域

寄存器 Rx[6,11-9]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。注意：字操作数的第 6 位是寄存器数字域的最高位。

寄存器 Ry[3-0]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

sz 域——指定输入操作数的大小。　— 0：字　— 1：长字

比例因子域——指定比例因子，当操作数为实型时不考虑该域。

— 00：无　— 01：结果<<1　— 10：保留　— 11：结果>>1

U/Lx——指定源寄存器操作数 Rx 的哪 16 位被作为字操作数来使用。

— 0：低字　— 1：高字

U/Ly——指定源寄存器操作数 Ry 的哪 16 位被作为字操作数来使用。

— 0：低字　— 1：高字

MAC　　Multiply Accumulate with Load　　MAC

指令操作

ACC + (Ry * Rx){<< | >>} SF → ACC

(<ea>y)→Rw

汇编格式

MAC.sz Ry,{U,L},Rx,{U,L}SF,<ea>y&,Rw（&允许使用掩码）

相关属性

尺寸=字或长字。

指令描述

两个 16 或 32 位的数相乘得到 32 位的数，将它与累加器相加，其和由比例因子按照定义进行转换后送到一个累加器中。如果操作数是 16 位，那么每个寄存器的高字或低字必须被指定。

与该操作同时执行的是，从由<ea>y 指定的内存空间中取出一个 32 位的操作数，将其值赋给目的寄存器 Rw。若指定使用掩码寄存器，则<ea>y 操作数在被该指令使用之前与掩码寄存器的内容进行与运算。

条件码**(MACSR)**

N	Z	V
*	*	*

N　结果的最高位被置则置位，否则清零

Z　结果为零则置位，否则清零

V　若溢出位被置则置位，否则不变

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rw			0	1	Rw	有效地址					
										模式			寄存器		
寄存器 Rx				sz	比例因子		0	U/Lx	U/Ly	掩码	0	寄存器 Ry			

指令域

寄存器 Rx[6,11-9]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。注意：字操作数的第 6 位是寄存器数字域的最高位。

有效地址域——指定了原操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	—	—
Ay	—	—
(Ay)	010	寄存器号 Ay
(Ay)+	011	寄存器号 Ay
-(Ay)	100	寄存器号 Ay
(d_{16},Ay)	101	寄存器号 Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

寄存器 Rx 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

sz 域——指定输入操作数的大小。

— 0：字

— 1：长字 比例因子域——指定比例因子，操作数为实型时

不考虑该域。

— 00：无

— 01：结果<<1

— 10：保留

— 11：结果>>1

U/Lx 和 U/Ly——指定源寄存器操作数 Rx/Ry 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

掩码域——指定在取有效地址<ea>y 时是否使用掩码寄存器。

— 0：不使用掩码寄存器

— 1：使用掩码寄存器

寄存器 Ry 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

MOVE from ACC — Move from Accumulator — MOVE from ACC

指令操作 累加器
→目的

汇编格式

MOVE.L ACC,Rx

相关属性 尺寸=
长字。

指令描述

将累加器中一个32位的值移到一个通用寄存器Rx中。当操作的是实型时(MACSR[F/I] = 1)，若 MACSR[S/U]被置，则累加器的内容舍入到16位的值并且存储到目的寄存器Rx的低16位中。目的寄存器的高16位均为零。累加器的值将不会被该舍入操作影响。

MACSR

不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	1	1	0	0	0	寄存器 Rx			

指令域

寄存器 Rx 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

MOVE from MACSR — Move from the MACSR — MOVE from MACSR

指令操作

MAC 标志寄存器→目的

汇编格式

MOVE.L MACSR,Rx

相关属性 尺寸=

长字。

指令描述 将MAC标志寄存器中的内容移到一个通用寄存器Rx中，Rx[31:8]中的内容被清零。

MACSR

不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	1	1	0	0	0	寄存器 Rx			

指令域

寄存器 Rx 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

MOVE from MASK — Move to the MAC MASK Register — MOVE from MASK

指令操作

MASK→目的

汇编格式

MOVE.L MASK,Rx

相关属性 尺寸=

长字。

指令描述 将掩码寄存器的内容移到一个通用寄存器Rx中，Rx[31:16]的位置是0xFFFF。

MACSR

不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	1	0	0	0	0	寄存器 Rx			

指令域

寄存器 Rx 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

MOVE MACSR to CCR

Move from the MACSR to the CCR

MOVE MACSRtoCCR

指令操作

MACSR →CCR

汇编格式

MOVE.L MACSR,CCR

相关属性 尺寸=

长字。

指令描述

将 MACSR 的条件码移到条件码寄存器中，操作码为 0xA9C0。

MACSR

不受影响。

条件码

X	N	Z	V	C
0	*	*	*	0

X 总是清零

N 若MACSR[N]=1则置位，否则清零

Z 若 MACSR[Z]=1 则置位，否则清零

V 若 MACSR[V]=1 则置位，否则清零

C 总是清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0

MOVE to ACC — Move to Accumulator — MOVE to ACC

指令操作 源→

累加器

汇编格式

MOVE.L Ry,ACC

MOVE.L #<data>,ACC

相关属性 尺寸=

长字。

指令描述 将寄存器中的一个32位的值或立即数移到一个累加器中。

条件码（**MACSR**）

N	Z	V
*	*	0

N 若最高位被置则置位，否则清零

Z 若结果为零则置位，否则清零

V 总是清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	1	0	0	有效地址					
										模式			寄存器		

指令域

有效地址域——指定源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器号 Dy
Ay	001	寄存器号 Ay
(Ay)	—	—
(Ay)+	—	—
−(Ay)	—	—
(d_{16},Ay)	—	—
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	111	100
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

MOVE to MACSR — Move to the MAC Status Register — MOVE to MACSR

指令操作

源→MAC 标志寄存器

汇编格式

MOVE.L Ry,MACSR
MOVE.L #<data>,MACSR

相关属性 尺寸=

长字。

指令描述 将寄存器中的一个32位的值或立即数移到MACSR寄存器中。

MACSR

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	—	—	—	—	OMC	S/U	F/I	R/T	N	Z	V	—
源<ea>位	—	—	—	—	—	—	—	—	[7]	[6]	[5]	[4]	[3]	[2]	[1]	—

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	1	0	0	有效地址					
										模式			寄存器		

指令域

有效地址域——指定源操作数，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器号 Dy
Ay	001	寄存器号 Ay
(Ay)	—	—
(Ay)+	—	—
−(Ay)	—	—
(d_{16},Ay)	—	—
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	111	100
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

MOVE to MASK

Move to the MAC MASK Register

MOVE to MASK

指令操作 源→

MASK

汇编格式

MOVE.L Ry,MASK

MOVE.L #<data>,MASK

相关属性 尺寸=

长字。

指令描述 将寄存器中的低16位或立即数移到掩码寄存器中。

MACSR

不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	1	0	0	有效地址					
										模式			寄存器		

指令域

有效地址域——指定源操作数，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器号 Dy
Ay	001	寄存器号 Ay
(Ay)	—	—
(Ay)+	—	—
−(Ay)	—	—
(d_{16},Ay)	—	—
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	111	100
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

MSAC Multiply Subtract MSAC

指令操作

ACC - (Ry * Rx){<< | >>} SF → ACC

汇编格式

MSAC.sz Ry,{U，L}Rx.{U，L}SF

相关属性 尺寸=字或

长字。

指令描述

两个 16 或 32 位的数相乘得到 32 位的数，将累加器与它相减，其差由比例因子按照定义进行转换后送到一个累加器中。如果操作数是 16 位，那么每个寄存器的高字或低字必须被指定。

条件码**(MACSR)**

N	Z	V
*	*	*

N 结果的最高位被置则置位，否则清零
Z 结果为零则置位，否则清零
V 若溢出位被置则置位，否则不变

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx			0	0	Rx	0	0	寄存器 Ry			
—	—	—	—	sz	比例因子		1	U/Lx	U/Ly	—	—	—	—	—	—

指令域

寄存器 Rx[6,11-9]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。注意：字操作数的第 6 位是寄存器数字域的最高位。

寄存器 Ry[3-0]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

sz 域——指定输入操作数的大小。

— 0： 字 — 1：长字 比例因子域——指定比例因子，操作数为实型时不考虑该域。

— 00：无 — 01：结果<<1 — 10：保留 — 11：结果>>1

U/Lx——指定源寄存器操作数 Rx 的哪 16 位被作为字操作数来使用。

— 0：低字 — 1：高字

U/Ly——指定源寄存器操作数 Ry 的哪 16 位被作为字操作数来使用。

— 0：低字 — 1：高字

MSAC Multiply Subtract with Load MSAC

指令操作

ACCx - (Ry * Rx){<< | >>} SF → ACC

(<ea>y)→Rw

汇编格式

MSAC.sz Ry,{U,L},Rx,{U,L}SF,<ea>y&,Rw（&允许使用掩码）

相关属性

尺寸=字或长字。

指令描述

两个 16 或 32 位的数相乘得到 32 位的数，将累加器与它相减，其差由比例因子按照定义进行转换后送到一个累加器中。如果操作数是 16 位，那么每个寄存器的高字或低字必须被指定。

与该操作同时执行的是，从由<ea>y 指定的内存空间中取出一个 32 位的操作数，将其值赋给目的寄存器 Rw。若指定使用掩码寄存器，则<ea>y 操作数在被该指令使用之前与掩码寄存器的内容进行与运算。

条件码**(MACSR)**

N	Z	V
*	*	*

N 若结果的最高位被置则置位，否则清零

Z 若结果为零则置位，否则清零

V 若有一溢出位被置则置位，否则不变

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rw			0	1	Rw	有效地址					
										模式			寄存器		
寄存器 Rx				sz	比例因子		1	U/Lx	U/Ly	掩码	0	寄存器 Ry			

指令域

寄存器 Rx[6,11-9]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。注意：字操作数的第 6 位是寄存器数字域的最高位。

有效地址域——指定了原操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	—	—
Ay	—	—
(Ay)	010	寄存器号 Ay
(Ay)+	011	寄存器号 Ay
-(Ay)	100	寄存器号 Ay
(d_{16},Ay)	101	寄存器号 Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

寄存器 Rx 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

sz 域——指定输入操作数的大小。

— 0：字

— 1：长字 比例因子域——指定比例因子，操作数为实型时

不考虑该域。

— 00：无

— 01：结果<<1

— 10：保留

— 11：结果>>1

U/Lx 和 U/Ly——指定源寄存器操作数 Rx/Ry 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

掩码域——指定在取有效地址<ea>y 时是否使用掩码寄存器。

— 0：不使用掩码寄存器

— 1：使用掩码寄存器

寄存器 Ry 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

第 6 章 EMAC 用户指令

本章描述了在 ColdFire 系列处理器中关于可选的增强乘法累加器单元（EMAC）的用户指令。为了方便指令的记忆，每条指令描述的讨论将按照字母顺序来讲述。

本章包括了原 EMAC 指令以及首次出现在 B 版本的 EMAC 定义中的四个乘法累加单元指令（MAAAC、MASAC、MSAAC 和 MSSAC）。

若要了解由乘法累加单元（MAC）所实现的指令，请见第五章“MAC 用户指令”。

MAAAC MAAAC

Multiply and Add to First Accumulator,Add to Second Accumulator

首先出现于 EMAC_B

指令操作

ACCx + (Ry * Rx) { << | >> } SF → ACCx
ACCw + (Ry *Rx) { << | >> } SF → ACCw

汇编格式

MAAAC.sz Ry,RxSF,ACCx,ACCw

相关属性 尺寸=字或长字。

指令描述

两个 16 或 32 位的数相乘得到 40 位的数，将它与累加器相加，其和由比例因子按照定义进行转换后送到累加器中 ACCx，同时也将其存储到另外的累加器 ACCw 中。

条件码**(MACSR)**

N	Z	V	PAVx	EV
*	*	*	*	*

N	第二结果最高位被置则置位，否则清零
Z	第二结果为零则置位，否则清零
V	若有一结果或第二次加法溢出位被置位或 PAVw=1 则置位，否则清零
PAVx,w	任一结果或加法若溢出则置位，否则不变 第二加法整型低32位溢出或实型
EV	低40 位溢出，则置位，否则清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx			0	ACCx 最低位	Rx 最高位	0	0	寄存器 Ry			
—	—	—	—	sz	比例因子		0	U/Lx	U/Ly	—	ACCx 最高位	ACCw		0	1

指令域

寄存器 Rx[6,11-9]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，

0x8 至 0xF 分别代表 A0 至 A7。注意：字操作数的第 6 位是寄存器数字域的最高位。

ACCx 域——指定第目标累加器 ACCx，字扩展位的第 4 位是最高位，字操作数的第 7 位是最低位。这两位的值按照下表所示来指定累加器的值：

字扩展[4]	字操作符[7]	累加器
0	0	ACC0
0	1	ACC1
1	0	ACC2
1	1	ACC3

寄存器 Ry[3-0]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

sz 域——指定输入操作数的大小。

— 0：字

— 1：长字 比例因子域——指定比例因子，操作数为实型时

不考虑该域。

— 00：无

— 01：结果<<1

— 10：保留

— 11：结果>>1

U/Lx——指定源寄存器操作数 Rx 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

U/Ly——指定源寄存器操作数 Ry 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

ACCw 域——指定第二个目标累加器 ACCw；00=累加器 0，11=累加器 3。

MAC Multiply Accumulate MAC

指令操作

ACCx + (Ry * Rx){<< | >>} SF → ACCx

汇编格式

MAC.sz Ry,{U，L}SF,ACCx

相关属性 尺寸=字或

长字。

指令描述

两个 16 或 32 位的数相乘得到 40 位的数，将它与累加器相加，其和由比例因子按照定义进行转换后送到一个累加器中 ACCx，如果操作数是 16 位，那么每个寄存器的高字或低字必须被指定。

条件码**(MACSR)**

N	Z	V	PAVx	EV
*	*	*	*	*

N	结果的最高位被置则置位，否则清零
Z	结果为零则置位，否则清零
V	若有一结果或加法溢出位被置位或 PAVw=1 则置位，否则清零
PAVx	结果或加法若溢出则置位，否则不变 整
EV	型低 32 位或实型低 40 位加法溢出，则置位，否则清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx			0	ACC 最低位	Rx 最高位	0	0	寄存器 Ry			
—	—	—	—	sz	比例因子		0	U/Lx	U/Ly	—	ACC 最高位	—	—	—	—

指令域

寄存器 Rx[6,11-9]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。注意：字操作数的第 6 位是寄存器数字域的最高位。

ACC 域——指定第目标累加器 ACCx。字扩展位的第 4 位是最高位，字操作数的第 7 位是最低位。这两位的值按照下表所示指定了累加器的值：

字扩展[4]	字操作符[7]	累加器
0	0	ACC0
0	1	ACC1
1	0	ACC2
1	1	ACC3

寄存器 Ry[3-0]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

sz 域——指定输入操作数的大小。

— 0：字

— 1：长字 比例因子域——指定比例因子，操作数为实型时

不考虑该域。

— 00：无

— 01：结果<<1

— 10：保留

— 11：结果>>1

U/Lx——指定源寄存器操作数 Rx 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

U/Ly——指定源寄存器操作数 Ry 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

MAC　　Multiply Accumulate with Load　　MAC

指令操作

ACCx + (Ry * Rx){<< | >>} SF → ACCx
(<ea>y)→Rw

汇编格式

MAC.sz Ry,{U,L},Rx,{U,L}SF,<ea>y&,Rw,ACCx（&允许使用掩码）

相关属性

尺寸=字或长字。

指令描述

两个 16 或 32 位的数相乘得到 40 位的数，将累加器与它相加，其和由比例因子按照定义进行转换后送到一个累加器中。如果操作数是 16 位，那么每个寄存器的高位或低位必须被指定。

与该操作同时执行的是，从由<ea>y 指定的内存空间中取出一个 32 位的操作数，将其值赋给目的寄存器 Rw。若指定使用掩码寄存器，则<ea>y 操作数在被该指令使用之前与掩码寄存器的内容进行与运算。

条件码**(MACSR)**

N	Z	V	PAVx	EV
*	*	*	*	*

N	结果的最高位被置则置位，否则清零
Z	结果为零则置位，否则清零
V	若有一结果或加法溢出位被置位或 PAVw=1 则置位，否则清零 任一结果或
PAVx	加法若溢出则置位，否则不 变 整型低 32位或实型低40位的加法溢出， 则置
EV	位，否则清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rw			0	ACC 最低位	Rw 最高位	有效地址					
										模式			寄存器		
寄存器				sz	比例因子		0	U/Lx	U/Ly	掩码	ACC 最高位		寄存器 Ry		

指令域

寄存器 Rx[6,11-9]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。注意：字操作数的第 6 位是寄存器数字域的最高位。

ACC 域——指定目的累加器 ACCx。字扩展的第 4 位是最高位，字操作数的第 7 位是最低位的反码（与 MAC 不产生进位不同）。这两位值按下表所示指定累加器的值：

字扩展[4]	字操作[7]	累加器
0	1	ACC0
0	0	ACC1
1	1	ACC2
1	0	ACC3

有效地址域——指定了原操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	—	—
Ay	—	—
(Ay)	010	寄存器号 Ay
(Ay)+	011	寄存器号 Ay
−(Ay)	100	寄存器号 Ay
(d_{16},Ay)	101	寄存器号 Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

寄存器 Rx 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

sz 域——指定输入操作数的大小。

— 0：字

— 1：长字 比例因子域——指定比例因子，操作数为实型时

不考虑该域。

— 00：无

— 01：结果<<1

— 10：保留

— 11：结果>>1

U/Lx 和 U/Ly——指定源寄存器操作数 Rx/Ry 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

掩码域——指定在取有效地址<ea>y 时是否使用掩码寄存器。

— 0：不使用掩码寄存器

— 1：使用掩码寄存器

寄存器 Ry 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8

至 0xF 分别代表 A0 至 A7。

MASAC MASAC

Multiply and Add to First

Accumulator,Subtract from Second Accumulator

首先出现于 EMAC_B

指令操作

ACCx + (Ry * Rx){<< | >>} SF → ACCx
ACCw - (Ry * Rx){<< | >>} SF → ACCw

汇编格式

MASAC.sz Ry,RxSF,ACCx,ACCw

相关属性 尺寸=字或

长字。

指令描述

两个 16 或 32 位的数相乘得到 40 位的数，将它与累加器相加，其和由比例因子按照定义进行转换后送到一个累加器中 ACCx，同时从另一累加器 ACCw 中减去该结果，并由比例因子按照定义进行转换。

条件码**(MACSR)**

N	Z	V	PAVx	EV
*	*	*	*	*

N	第二结果最高位被置则置位，否则清零
Z	第二结果为零则置位，否则清零 若有
V	一结果或第二次加法溢出位被置位
PAVx	或 PAVw=1 则置位，否则清零 任一结果或加法若溢出则置位，否则不 变
EV	第二加法整型低32位溢出或实型低40位溢出，则置位，否则清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx			0	ACCx 最低位	Rx 最高位	0	0	寄存器 Ry			
—	—	—	—	sz	比例因子		0	U/Lx	U/Ly	—	ACCx 最高位	ACCw		1	1

指令域

寄存器 Rx[6,11-9]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。注意：字操作数的第 6 位是寄存器数字域的最高位。

ACCx 域——指定第目标累加器 ACCx。字扩展位的第 4 位是最高位，字操作数的第 7 位是最低位。这两位的值按照下表所示指定了累加器的值：

字扩展**[4]**	字操作**[7]**	累加器
0	0	ACC0
0	1	ACC1
1	0	ACC2
1	1	ACC3

寄存器 Ry[3-0]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

sz 域——指定输入操作数的大小。

— 0：字

— 1：长字 比例因子域——指定比例因子，操作数为实型时

不考虑该域。

— 00：无

— 01：结果<<1

— 10：保留

— 11：结果>>1

U/Lx——指定源寄存器操作数 Rx 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

U/Ly——指定源寄存器操作数 Ry 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

ACCw 域——指定第二个目标累加器 ACCw；00=累加器 0，11=累加器 3。

MOVCLR Move from Accumulator and Clear MOVCLR

指令操作

累加器 → 目的;
0 → 累加器

汇编格式

MOVCLR.L ACCy , Rx

相关属性 尺寸=

长字。

指令描述

将一个 32 位的累加器里的数值放到一个通用寄存器 Rx 里，这个被选定的累加器里的内容将在被存放到 Rx 寄存器内后清除。清除操作也会影响累加器的扩展字节，累加可能溢出指示器。存储累加器的功能十分复杂，EMAC 构造功能由 MACSR 定义。下面的伪代码定义了它的操作;在描述中 ACC[47：0]代表了 32 位累加器和 16 位扩展字。

```
if   MACSR[S/U,F/I] == 00                                  /* 有符号整型模式
        if MACSR[OMC] == 0
            then ACC[31:0] → Rx                            /* 不饱和状态
            else if ACC[47:31] == 0x0000_0 or 0xFFFF_1
                then ACC[31:0] → Rx
                else if ACC[47] == 0
                    then 0x7FFF_FFFF → Rx
                    else 0x8000_0000 → Rx

if MACSR[S/U,F/I] == 10                                    /* 无符号整型模式
        if MACSR[OMC] == 0
            then ACC[31:0] → Rx                            /* 不饱和状态
            else if ACC[47:32] == 0x0000
                then ACC[31:0] → Rx
                else 0xFFFF_FFFF → Rx
```

```
if MACSR[F/I] == 1                                  /* 带符号实型模式
        if MACSR[OMC,S/U,R/T] == 000                /* 不饱和，无 16 位 rnd，无 32 位 rnd
                then ACC[39:8] → Rx
        if MACSR[OMC,S/U,R/T] == 001                /*不饱和，无 16 位 rnd，32 位 rnd
                then ACC[39:8] rounded by contents of [7:0] → Rx
        if MACSR[OMC,S/U] == 01                     /*不饱和，16 位舍入
                then 0 → Rx[31:16]
                        ACC[39:24] rounded by contents of [23:0] → Rx[15:0]
        if MACSR[OMC,S/U,R/T] == 10                 /*饱和，无 16 位 rnd，无 32 位 rnd
                if ACC[47:39] == 0x00_0 or 0xFF_1
                        then ACC[39:8] → Rx
                        else if ACC[47] == 0
                                then 0x7FFF_FFFF → Rx
                                else 0x8000_0000 → Rx
        if MACSR[OMC,S/U,R/T] == 101                /*饱和，无 16 位 rnd，32 位 rnd 舍入
                Temp[47:8] = ACC[47:8] rounded by contents of [7:0]
                if Temp[47:39] == 0x00_0 or 0xFF_1
                        then Temp[39:8] → Rx
                        else if Temp[47] == 0
                                then 0x7FFF_FFFF → Rx
                                else 0x8000_0000 → Rx
```

MOVCLR Move from Accumulator and Clear MOVCLR

```
if MACSR[OMC,S/U] == 11                                    /* 饱和，16 位舍入
        Temp[47:24] = ACC[47:24] rounded by the contents of [23:0]
        if Temp[47:39] == 0x00_0 or 0xFF_1
                then 0 → Rx[31:16]
                        Temp[39:24] → Rx[15:0]
                else if Temp[47] == 0
                        then 0x0000_7FFF → Rx
                        else 0x0000_8000 → Rx
0 → ACCx, ACCextx, MACSR[PAVx]
```

条件码（**MACSR**）

N	Z	V	PAVx	EV
—	—	—	0	—

N	无影响
Z	无影响
V	无影响
PAVx	清零
EV	无影响

指令格式

15	14	13	12	11	10 9	8	7	6	5	4	3 2 1 0
1	0	1	0	0	ACC	1	1	1	0	0	寄存器 Rx

指令域

ACC——指定目标累加器，位[10:9]的值指定累加器的值。

寄存器 Rx 域——指定目标寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

MOVE fromACC

MOVE fromACC

Move from Accumulator

指令操作

累加器→目的

汇编格式

MOVE.L ACCy,Rx

相关属性 尺寸=

长字。

指令描述

将一个 32 位数值从累加器移至通用寄存器 Rx。这个存储累加器的功能十分复杂，EMAC 构造功能由 MACSR 定义。下面伪代码定义了它的操作;在描述中 ACC[47：0]代表了 32 位累加器和 16 位扩展字。

```
if MACSR[S/U,F/I] == 00                                  /*有符号整型模式
    if MACSR[OMC] == 0
        then ACC[31:0] → Rx                                  /*不饱和状态
        else if ACC[47:31] == 0x0000_0 or 0xFFFF_1
            then ACC[31:0] → Rx
            else if ACC[47] == 0
                then 0x7FFF_FFFF → Rx
                else 0x8000_0000 → Rx
if MACSR[S/U,F/I] == 10                                  /*无符号整型模式
        if MACSR[OMC] == 0
            then ACC[31:0] → Rx                              /*不饱和状态
            else if ACC[47:32] == 0x0000
                    then ACC[31:0] → Rx
                    else 0xFFFF_FFFF → Rx
if MACSR[F/I] == 1                                       /*带符号实型模式
    if MACSR[OMC,S/U,R/T]==000                           /*不饱和，无 16 位 rnd，无 32 位 rnd
```

```
        then ACC[39:8] → Rx
    if MACSR[OMC,S/U,R/T] == 001                    /*不饱和，无 16 位 rnd，32 位 rnd
        then ACC[39:8] rounded by contents of [7:0] → Rx
    if MACSR[OMC,S/U] == 01                          /*不饱和，16 位舍入
        then 0 → Rx[31:16]
            ACC[39:24] rounded by contents of [23:0] → Rx[15:0]
    if MACSR[OMC,S/U,R/T] == 100                    /*饱和，无 16 位 rnd，无 32 位 rnd
        if ACC[47:39] == 0x00_0 or 0xFF_1
            then ACC[39:8] → Rx
            else if ACC[47] == 0
                then 0x7FFF_FFFF → Rx
                else 0x8000_0000 → Rx
if MACSR[OMC,S/U,R/T] == 101                        /*饱和，无 16 位 rnd，32 位 rnd 舍入
    Temp[47:8] = ACC[47:8] rounded by contents of [7:0]
    if Temp[47:39] == 0x00_0 or 0xFF_1
        then Temp[39:8] → Rx
        else if Temp[47] == 0
            then 0x7FFF_FFFF → Rx
            else 0x8000_0000 → Rx
```

MOVE from ACC — Move from an Accumulator — MOVE from ACC

```
if MACSR[OMC,S/U] == 11                          /* 饱和，16 位舍入
    Temp[47:24] = ACC[47:24] rounded by the contents of [23:0]
    if Temp[47:39] == 0x00_0 or 0xFF_1           /* 饱和，16 位舍入
        then 0 → Rx[31:16]
            Temp[39:24] → Rx[15:0]
        else if Temp[47] == 0
        then 0x0000_7FFF → Rx
    else 0x0000_8000 → Rx
    0 → ACCx, ACCextx, MACSR[PAVx]
```

条件码（**MACSR**）

N	Z	V	PAVx	EV
*	*	*	*	*

N	无影响
Z	无影响
V	无影响
PAVx	无影响
EV	无影响

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	ACC		1	1	0	0	0	寄存器 Rx			

指令域

ACC——指定目标累加器，位[10:9]的值指定累加器的值。

寄存器 Rx 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

MOVE from ACCext01 — Move from Accumulator 0 and 1 Extensions — MOVE from ACCext01

指令操作

累加器 0 和 1 的扩展字 → 目的

汇编格式

MOVE.L ACCext01,Rx

相关属性 尺寸=

长字。

指令描述

将四个扩展字节相关联的累加器 0 和 1 的内容移至一个通用寄存器。累加器扩展字节的存储如下表所示。注意：在 48 位逻辑累加的范围之内，LSB 的扩展字节的位置取决于 EMAC 的操作模式（整型相对与实型）。

累加器扩展字节	目的数据位
ACCext1[15:8]	[31:24]
ACCext1[7:0]	[23:16]
ACCext0[15:8]	[15:8]
ACCext0[7:0]	[7:0]

MACSR

不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	1	1	0	0	0	寄存器 Rx			

指令域

寄存器 Rx 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

MOVE from ACCext23 — Move from Accumulator 2 and 3 Extensions — MOVE from ACCext23

指令操作

累加器 2 和 3 的扩展字→目的

汇编格式

MOVE.L ACCext23,Rx

相关属性 尺寸=

长字。

指令描述

将四个扩展字节相关联的累加器 2 和 3 的内容移至一个通用寄存器。累加器扩展字节的存储如下表所示。注意：在 48 位逻辑累加的范围之内，最低位的扩展字节的位置取决于 EMAC 的操作模式（整型与实型相对）。

累加器扩展字节	目的数据位
ACCext3[15:8]	[31:24]
ACCext3[7:0]	[23:16]
ACCext215:8]	[15:8]
ACCext2[7:0]	[7 : 0]

MACSR

不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	1	1	0	0	0	寄存器 Rx			

指令域

寄存器 Rx 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

MOVE from MACSR

Move from the MACSR

MOVE from MACSR

指令操作

MAC 标志寄存器→目的

汇编格式

MOVE.L MACSR,Rx

相关属性 尺寸=

长字。

指令描述

将 MAC 标志寄存器的内容移至通用寄存器 Rx，Rx[31:12]的内容将被清除。

MACSR

不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	1	1	0	0	0	寄存器 Rx			

指令域

寄存器 Rx 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

MOVE from MASK

Move from the MAC MASK Register

MOVE from MASK

指令操作

MASK→目的

汇编格式

MOVE.L MAS,Rx

相关属性 尺寸=

长字。

指令描述

将掩码寄存器的内容移至通用寄存器 Rx，Rx[31:16]的内容被设置为 0xFFFF。

MACSR

不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	1	1	0	0	0	寄存器 Rx			

指令域

寄存器 Rx 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

MOVE

Copy an Accumulator

MOVE

ACC to ACC

ACC to ACC

指令操作 源累加器→目标

累加器

汇编格式

MOVE.L ACCy,ACCx

指令特征 尺寸=

长字。

指令描述

将 48 位源累加器的内容和其相关的 PAV 标志移入目标寄存器。这个操作完全在 EMAC 之内进行传递，所以没有传递空间与其相关联。与先将累加器的内容移至一个通用寄存器 Rn，再将 Rn 的内容移至目标累加器中的两次操作相比，这个指令提供了一个更好的方式来实现上述过程。

条件码（**MACSR**）

N	Z	V	PAVx	EV
*	*	*	*	*

N	结果最高位被置则置位，否则清零
Z	结果为零则置位，否则清零
V	如果 PAVw=1 则置位，否则清零
PAVx	设定源 PAVy 标志位的值 如果源累加器
EV	溢出整型低于 32 位或实型低于 40 位则置位，否则清零

指令格式

15	14	13	12	11	10 9	8	7	6	5	4	3	2	1 0
1	0	1	0	0	ACCx	1	0	0	0	1	0	0	ACCy

指令域

ACCx——指定目标累加器，位[10:9]的值指定了累加器的值。

ACCy——指定源累加器，位[1:0]的值指定了累加器的值。

MOVE MACSR to CCR

Move from the MACSR to the CCR

MOVE
MACSR
to CCR

指令操作

MACSR → CCR

汇编格式

MOVE.L MACSR,CCR

指令描述

将 MACSR 的条件码移到条件码寄存器中。MOVE MACSR to CCR 操作码为 0xA9C0。

MACSR

不受影响。

条件码

X	N	Z	V	C
0	*	*	*	*

X 总是清零
N 若MACSR[N]=1则置位，否则清零
Z 若 MACSR[Z]=1 则置位，否则清零
V 若 MACSR[V]=1 则置位，否则清零
C 若 MACSR[EV]=1 则置位，否则清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0

MOVE to ACC — Move to Accumulator — MOVE to ACC

指令操作 源→
累加器

汇编格式

MOVE.L Ry,ACCx
MOVE.L #<data>,ACCx

相关属性 尺寸=
长字。

指令描述 将一寄存器中的一个32位的值或一个立即数移到一个累加器中。如果EMAC
操作的
是有符号的整型数(MACSR[6:5] = 00)，16位的累加器扩展装入原操作数的第31位的符号位如果是无符号的整型数操作数(MACSR[6:5] = 10)则将整个16位的域都清零时。如果操作的是实型数(MACSR[5] = 1)则当扩展位的低8位操作码被清零时，累加器扩展的高8位将被置以源操作数的第31位的符号扩展位，对应的结果/累加器溢出位被清零。

条件码（**MACSR**）

N	Z	V	PAVx	EV
*	*	0	0	0

N	若最高位被置则置位，否则清零
Z	若结果为负数则置位，否则清零
V	总是清零
PAVx	总是清零
EV	总是清零

指令格式

15	14	13	12	11	10 9	8	7	6	5 4 3	2 1 0
1	0	1	0	0	ACC	1	0	0	有效地址	
									模式	寄存器

指令域

ACC——指定目标累加器，[10：9]位的值指定累加器数。 有效地址域——
指定源操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器号 Dy
Ay	001	寄存器号 Ay
(Ay)	—	—
(Ay)+	—	—
−(Ay)	—	—
(d_{16},Ay)	—	—
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	111	100
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

MOVE to ACCext01

Move to Accumulator 0 and 1 Extensions

MOVE to ACCext01

指令操作

源→累加器 0 和 1 的扩展字

汇编格式

MOVE.L Ry,ACCext01
MOVE.L #<data>,ACCext01

相关属性 尺寸=

长字。

指令描述

将一寄存器中的一个32位的值或一个立即数移到与累加器0和1相关的四个扩展位中。扩展位将会如下表所示被置。在结合的48位累加器逻辑中的扩展位的最低位的位置与EMAC操作码有关（整型相对于实型）。

源数据位	累加器扩展影响
[31:24]	ACCext1[15:8]
[23:16]	ACCext1[7:0]
[15:8]	ACCext0[15:8]
[7:0]	ACCext0[7:0]

MACSR

不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	1	0	0	有效地址					
										模式			寄存器		

指令域

有效地址域——指定源操作数，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器号 Dy
Ay	001	寄存器号 Ay
(Ay)	—	—
(Ay)+	—	—
−(Ay)	—	—
(d_{16},Ay)	—	—
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	111	100
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

MOVE to ACCext23

Move to Accumulator 2 and 3 Extensions

MOVE to ACCext23

指令操作

源→累加器 2 和 3 的扩展字

汇编格式

MOVE.L Ry,ACCext23

MOVE.L #<data>,ACCext23

相关属性 尺寸=

长字。

指令描述

将一寄存器中的一个32位的值或一个立即数移到与累加器2和3相关的四个扩展位中，扩展位将会如下表所示被置。在结合的48位累加器逻辑中的扩展位的最低位的位置与EMAC操作码有关（整型相对于实型）。

源数据位	累加器扩展影响
[31:24]	ACCext3[15:8]
[23:16]	ACCext3[7:0]
[15:8]	ACCext2[15:8]
[7:0]	ACCext2[7:0]

MACSR

不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	1	1	0	0	有效地址					
										模式			寄存器		

指令域

有效地址域——指定源操作数，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器号 Dy
Ay	001	寄存器号 Ay
(Ay)	—	—
(Ay)+	—	—
−(Ay)	—	—
(d_{16},Ay)	—	—
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	111	100
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

MOVE to MACSR — Move to the MAC Status Register — MOVE to MACSR

指令操作

源→MAC 标志寄存器

汇编格式

MOVE.L Ry,MACSR
MOVE.L #<data>,MACSR

相关属性 尺寸=

长字。

指令描述 将一寄存器中的一个32位的值或一个立即数移到MAC标志寄存器中。

MACSR

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	—	—	—	—	PAV3	PAV2	PAV1	PAV0	OMC	S/U	F/I	R/T	N	Z	V	EV
源<ea>位	—	—	—	—	[11]	[10]	[9]	[8]	[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	1	0	0	有效地址					
										模式			寄存器		

指令域

有效地址域——指定源操作数，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器号 Dy
Ay	001	寄存器号 Ay
(Ay)	—	—
(Ay)+	—	—
−(Ay)	—	—
(d_{16},Ay)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	111	100
(d_{16},PC)	—	—

(d_8,Ay,Xi)	—	—	(d_8,PC,Xi)	—	—

MOVE to MASK

Move to the MAC MASK Register

MOVE to MASK

指令操作 源→

MASK

汇编格式

MOVE.L Ry,MASK

MOVE.L #<data>,MASK

相关属性 尺寸=

长字。

指令描述 将一寄存器的低16位或立即数移到掩码寄存器中。

MACSR

不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	1	0	0	有效地址					
										模式			寄存器		

指令域

有效地址域——指定源操作数，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	000	寄存器号 Dy
Ay	001	寄存器号 Ay
(Ay)	—	—
(Ay)+	—	—
−(Ay)	—	—
(d_{16},Ay)	—	—
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	111	100
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

MSAAC MSAAC

Multiply and Subtract from First

Accumulator,Add to Second Accumulator

首先出现于 EMAC_B

指令操作

ACCx - (Ry * Rx){<< | >>} SF → ACCx
ACCw + (Ry * Rx){<< | >>} SF → ACCw

汇编格式

MSAAC.sz Ry,RxSF,ACCx,ACCw

相关属性 尺寸=字或

长字。

指令描述

两个 16 或 32 位的数相乘得到 40 位的数，从累加器中 ACCx 减去它，其差由比例因子按照定义进行转换，同时它与另一累加器 ACCw 相加，其和由比例因子按照定义进行转换后存储到 ACCw 中。

条件码**(MACSR)**

N	Z	V	PAVx	EV
*	*	*	*	*

N	第二结果最高位被置则置位，否则清零
Z	第二结果为零则置位，否则清零
V	若有一结果或第二次加法溢出位被置位或 PAVw=1 则置位，否则清零
PAVx	任结果或加法若溢出则置位，否则不变 第
EV	二加法整型低32位溢出或实型低40位溢出，则置位，否则清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx			0	ACCx 最低位	Rx 最高位	0	0	寄存器 Ry			
—	—	—	—	sz	比例因子		1	U/Lx	U/Ly	—	ACCx 最高位	ACCw		0	1

指令域

寄存器 Rx[6,11-9]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。注意：字操作数的第 6 位是寄存器数字域的最高位。

ACCx 域——指定第一个目标累加器 ACCx。字扩展位的第 4 位是最高位，字操作

数的第 7 位是最低位。这两位的值按照下表所示指定了累加器的值：

字扩展[4]	字操作[7]	累加器
0	0	ACC0
0	1	ACC1
1	0	ACC2
1	1	ACC3

寄存器 Ry[3-0]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

sz 域——指定输入操作数的大小。

— 0：字

— 1：长字 比例因子域——指定比例因子，操作数为实型时

不考虑该域。

— 00：无

— 01：结果<<1

— 10：保留

— 11：结果>>1

U/Lx——指定源寄存器操作数 Rx 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

U/Ly——指定源寄存器操作数 Ry 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

ACCw 域——指定第二个目标累加器 ACCw；00=累加器 0，11=累加器 3。

MSAC Multiply Subtract MSAC

指令操作

ACCx - (Ry * Rx){<< | >>} SF → ACCx

汇编格式

MAC.sz Ry,{U，L}SF,ACCx

相关属性 尺寸=字或

长字。

指令描述

两个 16 位或 32 位的数相乘得到一个 40 位的积，从累加器 ACCx 中减去该积其差由比例因子按照定义进行转换。如果操作数是 16 位的，那么每一个寄存器的高位或低位必须是指定的。

两个 16 或 32 位的数相乘得到 40 位的数，从累加器 ACCx 中减去它，其差由比例因子按照定义进行转换，如果操作数是 16 位，那么每个寄存器高位或低位必须被指定。

条件码**(MACSR)**

N	Z	V	PAVx	EV
*	*	*	*	*

N	结果的最高位被置则置位，否则清零
Z	结果为零则置位，否则清零
V	若有一结果或加法溢出位被置位或 PAVx=1 则置位，否则清零
PAVx	结果或加法若溢出则置位，否则不变
EV	整型低32位或实型低40位的加法溢出，则置位，否则清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx			0	ACC 最低位	Rx 最高位	0	0	寄存器 Ry			
—	—	—	—	sz	比例因子		1	U/Lx	U/Ly	—	ACC 最高位	—	—	—	—

指令域

寄存器 Rx[6,11-9]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。注意：字操作数的第 6 位是寄存器数字域的最高位。

ACC 域——指定第一个目标累加器 ACCx。字扩展位的第 4 位是最高位，字操作数的第 7 位是最低位。这两位的值按照下表所示指定了累加器的值：

字扩展[4]	字操作[7]	累加器
0	0	ACC0
0	1	ACC1
1	0	ACC2
1	1	ACC3

寄存器 Ry[3-0]域——指定一源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

sz 域——指定输入操作数的大小。

— 0：字

— 1：长字 比例因子域——指定比例因子。操作数为实型时

不考虑该域。

— 00：无

— 01：结果<<1

— 10：保留

— 11：结果>>1

U/Lx——指定源寄存器操作数 Rx 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

U/Ly——指定源寄存器操作数 Ry 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

MSAC Multiply Subtract with Load MSAC

指令操作

ACCx - (Ry * Rx){<< | >>} SF → ACCx；(<ea>y)→Rw

汇编格式

MAC.sz Ry,{U,L},Rx,{U,L}SF,<ea>y&,Rw,ACCx（&允许使用掩码）

相关属性

尺寸=字或长字。

指令描述

两个 16 或 32 位的数相乘得到 40 位的数，从累加器 ACCx 中减去它，其差由比例因子按照定义进行转换，如果操作数是 16 位，那么每个寄存器的高位或低位必须被指定。

与该操作同时执行的是，从由<ea>y 指定的内存空间中取出一个 32 位的操作数，将其值赋给目的寄存器 Rw。若指定使用掩码寄存器，则<ea>y 操作数在被该指令使用之前与掩码寄存器的内容进行与运算。

条件码**(MACSR)**

N	Z	V	PAVx	EV
*	*	*	*	*

N	若结果的最高位被置则置位，否则清零
Z	若结果为零则置位，否则清零
V	若有一结果或加法溢出位被置位或 PAVw=1 则置位，否则清零
PAVx	若结果或加法若溢出则置位，否则不变
EV	若整型低32位或实型低40位的加法溢出，则置位，否则清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rw			0	ACC 最低位	Rw 最高位	有效地址					
										模式			寄存器		
寄存器 Rx				sz	比例因子		1	U/Lx	U/Ly	掩码	ACC 最高位		寄存器 Ry		

指令域

寄存器 Rx[6,11-9]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。注意：字操作数的第 6 位是寄存器数字域的最高位。

ACC 域——指定第一个目标累加器 ACCx。字扩展位的第 4 位是最高位，字操作数的第 7 位是最低位的反码（与 MAC 不产生进位不同）。这两位的值按照下表所示指定了累加器的值：

字扩展[4]	字操作[7]	累加器
0	1	ACC0
0	0	ACC1
1	1	ACC2
1	0	ACC3

有效地址域——指定了原操作数<ea>y，会用到以下表格所列出的寻址方式：

寻址方式	模式	寄存器
Dy	—	—
Ay	—	—
(Ay)	010	寄存器号 Ay
(Ay)+	011	寄存器号 Ay
−(Ay)	100	寄存器号 Ay
(d_{16},Ay)	101	寄存器号 Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

寄存器 Rx 域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

sz 域——指定输入操作数的大小。

— 0：字

— 1：长字 比例因子域——指定比例因子，操作数为实型时

不考虑该域。

— 00：无

— 01：结果<<1

— 10：保留

— 11：结果>>1

U/Lx 和 U/Ly——指定源寄存器操作数 Rx/Ry 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

掩码域——指定在取有效地址<ea>y 时是否使用掩码寄存器。

— 0：不使用掩码寄存器

— 1：使用掩码寄存器

寄存器 Ry 域——指定源寄存器操作数，该处该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

MSSAC MSSAC

Multiply and Subtract from First

Accumulator,Subtract from Second Accumulator

首先出现于 EMAC_B

指令操作

ACCx - (Ry * Rx){<< | >>} SF → ACCx
ACCw - (Ry * Rx){<< | >>} SF → ACCw

汇编格式

MSAAC.sz Ry,RxSF,ACCx,ACCw

相关属性 操作大小=字或
长字。

指令描述

两个 16 或 32 位的数相乘得到 40 位的数，从累加器 ACCx 中减去它，其差由比例因子按照定义进行转换，也从另外累加器 ACCw 中减去它。

条件码**(MACSR)**

N	Z	V	PAVx	EV
*	*	*	*	*

N	第二结果最高位被置则置位，否则清零
Z	第二结果为零则置位，否则清零 若有
V	一结果或第二次加法溢出位被置位
PAVx	或 PAVw=1 则置位，否则清零 任一结果或加法若溢出则置位，否则不 变
EV	第二加法整型低32位溢出或实型低40位溢出，则置位，否则清零

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx			0	ACCx 最低位	Rx 最高位	0	0	寄存器 Ry			
—	—	—	—	sz	比例因子		1	U/Lx	U/Ly	—	ACCx 最高位	ACCw		1	1

指令域

寄存器 Rx[6,11-9]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。注意：字操作数的第 6 位是寄存器数字域的最高位。

ACCx 域——指定第一个目标累加器 ACCx。字扩展位的第 4 位是最高位，字操作

数的第 7 位是最低位。这两位的值按照下表所示指定了累加器的值：

字扩展[4]	字操作[7]	累加器
0	0	ACC0
0	1	ACC1
1	0	ACC2
1	1	ACC3

寄存器 Ry[3-0]域——指定源寄存器操作数，该处 0x0 至 0x7 分别代表 D0 至 D7，0x8 至 0xF 分别代表 A0 至 A7。

sz 域——指定输入操作数的大小。

— 0：字

— 1：长字 比例因子域——指定比例因子，操作数为实型时

不考虑该域。

— 00： 无

— 01：结果 << 1

— 10：保留

— 11：结果 >> 1

U/Lx——指定源寄存器操作数 Rx 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

U/Ly——指定源寄存器操作数 Ry 的哪 16 位被作为字操作数来使用。

— 0：低字

— 1：高字

ACCw 域——指定第二个目标累加器 ACCw；00=累加器 0，11=累加器 3。

第 7 章 浮点运算单元(FPU)用户指令

本章包含了，由可选浮点型单元(FPU)实现的指令的功能描述。有关浮点型状态寄存器(FPSR)的作用机制和条件测试的通用信息，将在本章的开始部分加以整理。

7.1 浮点型状态寄存器(FPSR)

图 7-1，浮点型条件码寄存器(FPSR)包括浮点型条件码(FPCC)字节、浮点型异常状态(EXC)字节和浮点型异常生成(AEXC）字节。用户可以读/写 FPSR 的所有的位。绝大多数浮点型指令的执行都会修改到 FPSR。

———FPCC———				————异常状态字(EXC)————										————AEXC字————					
31 28	27	26	25	24	23 16	15	14	13	12	11	10	9	8	7	6	5	4	3	20
—	N	Z	I	NAN	—	BSUN	INAN	OPERR	OVPL	UNFL	DZ	INEX	IDE	IOP	OVFL	UNFL	DZ	INEX	—

图 **7-1** 浮点型状态寄存器**(FPSR)**

表7-1描述FPSR的各个域。 对于AEXC[OVFL]、AEXC[DZ]和AEXC[INEX]来说，新的值由当前的AEXC值和
相应EXC位的OR操作结果来决定，如下所示：

新的 AEXC[OVFL] = 当前的 AEXC[OVFL] | EXC[OVFL]

新的 AEXC[DZ] = 当前的 AEXC[DZ] | EXC[DZ]

新的 AEXC[INEX] = 当前的 AEXC[INEX] | EXC[INEX]

对于AEXC[IOP]和AEXC[UNFL]来说，新的值由当前AEXC值和EXC各位的组合型OR操作结果来决定，如下：

新的 AEXC[IOP] = 当前的 AEXC[IOP] | EXC[BSUN | INAN | OPERR]

新的 AEXC[UNFL] = 当前的 AEXC[UNFL] | EXC[UNFL & INEX]

表 7-2 显示了指令执行过程是如何影响 FPS EXC 各位的。

表 **7-1** 有关 **FPSR** 域的描述

位	域	描 述	
31-24	FPCC	浮点条件码字节。包含的四个条件码位是所有算术指令涉及浮点数据寄存器完成后被置位的。浮点存储指令FMOVEM和系统控制寄存器传送指令不影响FPCC的内容。	
31-28		预留位，应予以清零。	
27		N	负
26		Z	0
25		I	无穷大
24		NAN	不是数值
23-16	—	预留位，应予以清零。	
15-8	EXC	异常状态字节，包含当前算术指令或移动操作可能引发的浮点型异常位	
15		BSUN	分支/无序设定
14		INAN	输入不是数值
13		OPERR	操作错误
12		OVEL	溢出
11		UNFL	下溢
10		DZ	除数是0
9		INEX	不正确的结果
8		IDE	输入不正确
7-0	AECX	异常产生位。在算术操作结束后，EXC位逻辑上与AEXC值结合，更确切的说，逻辑OR的结果会被保存到现有的AEXC字节（FBcc只更新IOP）该操作改变AEXC中的粘滞性浮点型异常位，使用者只有在一系列浮点型操作结束后才能获得其内容。粘滞性的位在用户清零前保持为1。	
7		IOP	错误操作
6		OVFL	负
5		UNFL	0
4		DZ	无穷大
3		INEX	非数值类型
2-0		—	保留的，应该被清零

表 **7-2 FPSR EXC** 位

EXC位	描　述
BSUN	无序时跳转/设定，如果NAN位被置位且选择条件是IEEE未知测试，则置位FBcc；否则清零。
INAN	输入为非数字类型。如果输入操作数是NAN类型则置位；否则清零。
IDE	输入非规格化操作数。如果输入操作数是非规格化数据则置位；否则清零。
OPERR	操作数错误。以下情况出现时会被置位： FADD[(+∞) + (-∞)]或[(-∞) + (+∞)] FDIV(0 ÷0)或(∞ ÷∞) FMOVE OUT (to B,W,L)整型溢出，源是源是NAN或±∞ FMUL源是<0或-∞ FSQRT一个操作数是0且另一操作数是±∞ FSUB[(+∞) - (+∞)]或[(-∞) - (-∞)] 否则清零。
OVFL	溢出。如果目的是浮点数据寄存器或内存时，算术运算中间结果的指数大于或等于所选择的舍入精度的最大指数值时，则置位。否则清零。只有当目的数是单或双精度时，溢出发生；其它格式的溢出被视为操作错误。
UNFL	下溢。如果算术操作的中间结果由于太小，而无法用浮点数寄存器或内存的可选舍入精度的规格化数值来表示，也就是说，当中间的结果的指数小于或等于所选择的舍入精度的最小指数值时，则置位，否则清零。只有当目的数格式是单精度或双精度数时下溢才会发生。当目的数格式是字节、字或长字时，下溢将被转变成0，而不会引起下溢或操作错误。
DZ	如果FDIV操作的除数是零因子，则置位，否则清零。
INEX	满足下列条件会被置位： ①如果浮点数的中间结果的尾数比舍入精度或目的格式所代表的数拥有更多的有效位 ②如果输入数据是非规格化数据，但输入规范化的异常(IDE)被禁止。 ③结果产生了溢出。 ④在下溢异常被禁止时出现下溢结果。 否则清零。

7.2 条件测试

不像与运行相关的整型条件码，指令始终用同样的方式去设定 FPCC 的位内容或根本不对其改弦更张。因此本手册的指令说明中不包括 FPCC 设置，而由本节来叙述如何设置 FPCC 位。

FPCC 位与整型条件码略有不同。FPU 操作的最终结果有依据的使 FPCC 位置位或清零，这独立于操作本身。整型条件码位 CCR[N]和 CCR[Z]具有这种特点，但 CCR[V]和 CCR[C]位，在执行不同指令时以不同的方式被设置。表 7-3 列举了 FPCC 对于每个数据类型的相关设置。用另外的位组合加载 FPCC 并执行条件指令，可能产生意想不到的跳转条件。

表 **7-3 FPCC** 编码表

数据类型	N	Z	I	NAN
+ 规格化或非规格化	0	0	0	0
- 规格化或非规格化	1	0	0	0
+ 0	0	1	0	0
- 0	1	1	0	0
+无穷大	0	0	1	0
-无穷大	1	0	1	0
+ NAN	0	0	0	1
- NAN	1	0	0	1

在 IEEE 的浮点型数制中列入该 NAN 数据类型需要对布尔条件做包括 FPCC[NAN] 在内的条件测试。因为不能确定 NAN 是否比序列中数字更大还是更小（亦即它是无序的），当尝试进行无序比较时，比较指令将设置 FPCC[NAN]。所有影响 NAN 的算术指令都将设置 NAN 位。条件指令将 NAN 被置位视为无序状态。

IEEE-754 标准定义了以下 4 个条件：

(1)等于(EQ)

(2)大于(GT)

(3)小于(LT) (4)无序(UN) 该标准规定，只有条件码的形成才能作为浮点型比较运算结果。在执行任何影响条

件码的操作过程结束时，FPU 可以测试这些条件以及其他的 28 个条件。对于浮点型条件分支指令，处理器在逻辑上结合 4 位 FPCC 条件码以形成 32 种条件测试，若在条件测试时，无序条件被检测到存在，则其中 16 起将引发异常（IEEE 无意测试）。其他的 16 起则不会引起异常（IEEE 感知测试）。IEEE 感知测试的规定很适用于下列情况：

(1)当要将程序从不支持 IEEE 标准的系统移植到某个类似系统时。

(2)当生成不支持 IEEE 浮点型概念的高级语言代码时（即是无序状态）。 在浮点比较操作中，当 1 个或 2 个操作数为 NAN 时，无序条件产生。浮点型分部

的无序状态破坏了存在于整数运算中的三分关系（大于、等于和小于）。举例来说，浮点型分部的大于(FBGT）并不是完全相反于浮点型小于或等于(FBLE)。而相反的条件是浮点型不大于(FBNGT)。如果先前指令的结果是无序的，FBNGT 为真，然而这时无论是 FBGT 还 FBLE 都为假，因为无序条件会使得上述两项测试（同时设置 BSUN）都失败。因为编译器转换测试条件的情况非常普遍，所以编译代码发生器应对浮点型分部的三分关系特别小心。

当使用 IEEE 感知测试时，如果出现分支转移和 FPCC[NAN]被置位，而且分支转移并不是由 FBEQ 或 FBNE 引发，用户程序将收到 BSUN 异常。如果 FPCR 寄存器中的 BSUN 异常使能开启，则将引发 BSUN 陷阱(trap)。因此，如果有意外条件产生，IEEE 无意测试程序将被中断。明智的 IEEE-754 标准使用者应该在那些包含有序和无序状况的程序中使用 IEEE 感知测试，因为有序和无序情况的特征是明确包含在有测试条件中

的，无序状况出现时，EXC[BSUN]不被设置。表 7-4 总结了条件的记录法、定义、公式、谓词以及在测试 32 浮点型条件时 EXC[BSUN]是否被置位。公式栏列出了每个被测公式形式的 FPCC 位组合。条件码上的上划线表示对该位清零，所有其他位被置位。

表 **7-4** 浮点型条件测试

标识符	定义	等式	谓词[1]	EXC[BSUN]设置
IEEE无意测试				
EQ	等于	Z	000001	否
NE	不等	$\overline{Z}$	001110	否
GT	大于	$\overline{NAN \mid Z \mid N}$	010010	是
NGT	不大于	NAN | Z | N	011101	是
GE	大于或等于	Z | ($\overline{NAN \mid N}$)	010011	是
NGE	不大于或等于	NAN | (N & $\overline{Z}$)	011100	是
LT	小于	N & ($\overline{NAN \mid Z}$)	010100	是
NLT	不小于	NAN | (Z | $\overline{N}$)	011011	是
LE	小于或等于	Z | (N & $\overline{NAN}$)	010101	是
NLE	不小于或等于	NAN | ($\overline{N \mid Z}$)	011010	是
GL	大于或小于	$\overline{NAN \mid Z}$	010110	是
NGL	不大于或小于	NAN | Z	011001	是
GLE	大于、小于或等于	$\overline{NAN}$	010111	是
NGLE	不大于、小于或等于	NAN	011000	是
IEEE有意测试				
EQ	等于	Z	000001	否
NE	不等于	$\overline{Z}$	001110	否
OGT	有序大于	$\overline{NAN \mid Z \mid N}$	000010	否
ULE	无序或小于或等于	NAN | Z | N	001101	否
OGE	有序大于或等于	Z | ($\overline{NAN \mid N}$)	000011	否
ULT	无序小于或等于	NAN | (N & $\overline{Z}$)	001100	否
OLT	有序小于	N & ($\overline{NAN \mid Z}$)	000100	否
UGE	无序或大于或等于	NAN | (Z | $\overline{N}$)	001011	否
OLE	有序或小于或等于	Z | (N & $\overline{NAN}$)	000101	否
UGT	无序或大于	NAN | ($\overline{N \mid Z}$)	001010	否
OGL	有序大于或小于	$\overline{NAN \mid Z}$	000110	否
UEQ	无序或等于	NAN | Z	001001	否
OR	有序	$\overline{NAN}$	000111	否
UN	无序	NAN	001000	否
混杂测试				

F	错误	错误	000000	否
T	正确	正确	001111	否
SF	发错误信号	错误	010000	是
ST	发正确信号	正确	011111	是
SEQ	发等于信号	Z	010001	是
SNE	发不等信号	$\overline{Z}$	011110	是

[1] 这一列提及叙述这个测试的在指导的有条件描述的域的值。

7.3 异常发生的指令结果

如表 7-5 所示，指令执行的结果可能有所不同，这要看 FPCR 中的异常使能是否被开启。当 EXC 位的值为 1 时异常使能，为 0 时异常被禁止。注意，如果某个异常被启用，且是出现在 FMOVE OUT 的情况下，则目的地不受影响。

表 **7-5 FPCR EXC** 位异常可用或不可用的结果

EXC位	异常	描 述
BSUN	不可用	浮点条件被认为好像是相应的IEEE有意型条件谓词。无异常发生。
	可用	处理器产生浮点数预处理异常。
INAN	不可用	如果目的数据格式是单精度或双精度，NAN将和转到目的的零标志一起发生。如果目的地数据格式是B、W或L，将逐个的被写到目的寄存器。
	可用	除非异常发生在目的单元格不受影响的FMOVE OUT指令上，被写到目的寄存器的结果和寄存器不可用一样。
IDE	不可用	运算元被当做零，INE别设定，而且处理继续。
	可用	如果一个运算元是非规格化，IDE被处理，但是INEX没被设定以便处理的人能适当地设定INEX。除非异常发生在目的单元格不受影响的FMOVE OUT指令上，否则目的地正在以相同的结果重新写，如果IDE不可用。
OPERR	不可用	当目的地是一个浮点数据寄存器的时候，结果是双精密NAN,它的所有的尾数和标志全部设零(有意义)。对于带有S或D格式的FMOVE OUT指令OPERR是不可能的。带有B、W、或L的OPERR只有在转化为整数溢出或源是无穷大或NAN之一时是可能的。在整数溢出或源是无穷大情形，能适合指定的目的格式(B、W或L)的最大的有意义或无意义的整数被储存。
	可用	除非异常发生在目的单元格不受影响的FMOVE OUT指令上，被写到目的寄存器的结果和寄存器不可用一样。
OVFL	不可用	被以FPCR[MODE]中所定义的模式所储存的值。 RN无穷大。 RZ 带有中间结果符号的大的量级数。 RM 对于有意义的溢出，大的有意义的格式化的数。 对于无意义的溢出，-∞。 RP 对于有意义的溢出，-∞。 对于无意义的溢出， 大的有意义的格式化的数。

	可用	除非异常发生在目的单元格不受影响的FMOVE OUT指令上，被写到目的寄存器的结果和寄存器不可用一样。
UNFL	不可用	储存的结果按如下定义。如果UNFL异常不可用，UFFL也设定INEX。 RN 带有中间结果符号的0。 RZ 带有中间结果符号的0。 RM 对于有意义的溢出，-0。 对于无意义的溢出，小的有意义的格式化的数。 RP 对于有意义的溢出，小的有意义的格式化的数。 对于无意义的溢出，-0。
	可用	除非异常发生在目的单元格不受影响的FMOVE OUT指令上，被写到目的寄存器的结果和寄存器不可用一样。
DZ	不可用	目的浮点数据寄存器用除输入操作标志的OR之外的带有符号最大的写入。
	可用	目的地浮点数据寄存器在异常中不可用情形的写入。
INEX	不可用	结果被舍入且写入目的寄存器。
	可用	除非异常发生在目的单元格不受影响的FMOVE OUT指令上，被写到目的寄存器的结果和寄存器不可用一样。

7.4 ColdFire 和 MC680x0 FPU 编程模型的本质不同

本节是为编译器开发和开发从 68K 到 ColdFire 汇编语言端口例程而编写的，它突出描述了 ColdFire 系列 FPU 指令集架构(ISA)和与其并驾齐驱的 68K 系列 ISA 的主要差异（以 MC68060 为例）。它们的内部的 FPU 数据宽度明显不同，ColdFire 使用 64 位双精度而 68K 系列使用 80 位的扩展精度。其他差异则存在于对寻址模式的支持方面，无论是所有 FPU 指令还具体的操作码。表 7-6 列出了关键的不同之处。所有的 ColdFire 芯片因为只支持大小为 48 位或更少的指令尺寸，不能满足 68K 系列所需的大指令长度。

表 **7-6** 设计模板的不同点

特　征	68K	ColdFire
内部的数据宽度	80位	64位
支持fpGEN d8(An,Xi),FPx	是	否
支持fpGEN xxx.{w,l},FPx	是	否
支持fpGEN d8(PC,Xi),FPx	是	否
支持fpGEN #xxx,FPx	是	否
支持fmovem (Ay)+,#list	是	否
支持fmovem #list,-(Ax)	是	否
支持fmovem FP控制寄存器	是	否

某些差异会影响到函数的激活和返回。68K 子程序通常开始于 fmovem #list,(A7)，以将寄存器内容保存到系统堆栈，每个寄存器占有三个长字。而在 ColdFire 芯片中，每个寄存器占有二个长字，且堆栈指针必须在 fmovem 指令之前调整。类似的序列通常发生在函数运行结束时，用来准备例行调用的返回控制。

表 7-7、7-8 和 7-9 给出的实例表示 68K 及其同等次的 ColdFire 系列的操作序列。

表**7-7 68K/ColdFire**操作顺序**1**[1]

68K	同等次的ColdFire
fmovem.x #list,-(a7)	lea -8*n(a7),a7;分配堆栈空间 fmovem.d #list,(a7) ;
fmovem.x (a7)+,#list	fmovem.d (a7),#list ; 保持FPU寄存器 lea 8*n(a7),a7 ;逻辑指派堆栈空间

[1] n 是 PF 寄存器储存或恢复的数据。

如果子程序包括 LINK 和 UNLK 指令，可以为这些操作考虑使用 FPU 寄存器存储时所需的堆栈空间，以避免使用 LEA 指令。

只用一个指令，68K 的 FPU 就能支持乘法控制寄存器（FPCR、FPSR 和 FPIAR）的加载和存储操作。对于 ColdFire 系列芯片，每次只能有一种传送操作。

见表 7-8，对于不需要寻址方式支持的指令，操作数地址可由 LEA 指令在 FPU 操作之前形成。

表 **7-8 68K/ColdFire** 操作顺序 **2**

68K	同等次的ColdFire
fadd.s label,fp2	lea label,a0 ;从指针到数据 fadd.s (a0),fp2
fmul.d (d8,a1,d7),fp5	lea (d8,a1,d7),a0 ;从指针到数据 fmul.d (a0),fp5
fcmp.l (d8,pc,d2),fp3	lea (d8,pc,d2),a0 ;从指针到数据 fcmp.l (a0),fp3

68K 的 FPU 允许浮点指令直接指明立即数的值，而 ColdFire 系列的 FPU 不支持这种类型的即时常量。因此建议将浮点型立即数的值放入一个常数表，以方便它们能用于 PC（指令指针）的相对寻址处理，或作为基于其他地址指针的偏移量，见表 7-9。请注意对于 ColdFire 来说，如果 PC 相对有效地址是由 FPU 指令指定，则 PC 总要往 16 位操作字的地址上加 2。

表 **7-9 68K/ColdFire** 操作顺序 **3**

68K	同等次的ColdFire
fadd.l #imm1,fp3	fadd.l (imm1_label,pc),fp3
fsub.s #imm2,fp4	fsub.s (imm2_label,pc),fp3
fdiv.d #imm3,fp5	fdiv.d (imm3_label,pc),fp3 align 4 imm1_label: long imm1 ;长字 imm2_label: long imm2 ;单精度 align 8 imm3_label: long imm3_upper, imm3_lower ;双精度

最后，ColdFire 和 68K 的不同之处在于异常是如何被视若罔闻的。ColdFire 的异常处理模型，规定 FPSR 异常标志位和相应的 FPCR 使能位用于搁置异常。因此，异常被搁置的状态，可以通过装载 FPSR 和（或）FPCR 来创建。对于 68K 而言，这种异常被搁置类型的情况是不可能有的。

从对编译过的浮点型应用程序实例的分析中可得出，这些差异是导致绝大多数兼容 68K 系列的源文本不同于 ColdFire 系列程序文本的原因。

7.5 指令描述

为了方便记忆，本节按照字母顺序叙述浮点指令。操作列表给出了每个指令所能遇到的各种情况的运行结果。每个列表的顶部和左侧，列出了所有可能的输入（不管是正或是负），而结果在列表的其他条目中给出。在多数情况下，结果是浮点值（数字、无穷、零点或是 NAN 的内容），但对 FCMP 和 FTST 来说，唯一的结果是设置了条件码位。如果状态没有确定，任何条件码位都不会被设置。注意，如果 PC 的相对有效地址是由 FPU 指令指定，则 PC 总要往 16 位操作字的地址上加 2。

若需要了解异常条件下的浮点型指令（溢出、NAN 操作等）操作，请参考表 7-5。

表 7-10 显示了源数据和目的数据的数据格式编码以及怎样通过 FMOVE 指令实现从寄存器到内存的操作。

表 **7-10** 数据格式编码

源格式	描　述
000	长字(L)
001	实单精度 (S)
100	字整型(W)
101	实双精度(D)
110	字节整型(B)

FABS　　Floating-Point Absolute Value　　FABS

（浮点数的绝对值）

指令操作

源的绝对值 → FPx

汇编格式

FABS.fmt　　<ea>y,FPx
FABS.D　　FPy,FPx
FABS.D　　FPx
FrABS.fmt　　<ea>y,FPx
FrABS.D　　FPy,FPx
FrABS.D　　FPx

其中 r 表示四舍五入精度，S 或 D。

相关属性

格式 = 字节、字、长字、单精度或双精度

指令描述

将源操作数转化为双精度(如果需要)而且把它的绝对值储存到浮点型目的寄存器。

FABS 操作结果的精度是由 FPCR 所决定的。FSABS 和 FDABS 操作结果是单精度或双精度的，不由 FPCR 决定

条件码

目的	源 [1]		
	+ 在范围内 -	+ **0** -	+ 无穷大 -
结果	绝对值	绝对值	绝对值

[1] 如果原操作数是一个 NAN，则阅读第 1.7.1.4 节，‖不是一个数‖。

FPSR[FPCC]:

见第 7.2 节，“条件测试”。

FPSR	**BSUN**	**INAN**	**IDE**	**OPERR**	**OVFL**	**UNFL**	**DZ**	**INEX**
[EXC]:	0	见表 7-2		0	0	0	0	0

FPSR[AEXC]:

见第 7.1 节，“浮点状态寄存器(FPSR)”。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx				操作码					

指令域

源有效的地址域——外部操作数决定寻址方式。

如果 R/M = 1，这个域指定源操作数的地址,<ea>y。只有下列的表列出的寻址方式能被使用。

寻址方式	模式	寄存器
Dy[1]	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

[1] 只有在格式为字节，字，长字或单精度。如果 R/M = 0，这个域不可用，所有的必须清 0。

R/M 域——指定源操作数的寻址方式。

— 1：从<ea>y 存到寄存器。

— 0：从寄存器存到寄存器。

源指定域——指定源寄存器或数据模式。

如果 R/M = 1，指定源数据的模式，表 7-10 显示源数据模式编码。

如果 R/M = 0，指定源源浮点型数据寄存器，FPy。

目的寄存器域——指定源目的浮点型数据寄存器，FPx。

操作码域——指定指令和舍入精度。

操作码	指令	舍入精度
0011000	FABS	用 FPCR 说明舍入精度
1011000	FSABS	单精度
1011100	FDABS	双精度

FADD　　Floating-Point Add　　FADD

（浮点数的加法）

指令操作

源+FPx → FPx

汇编格式

FADD.fmt　　<ea>y,FPx
FADD.D　　FPy,FPx
FrADD.fmt　　<ea>y,FPx
FrADD.D　　FPy,FPx

其中 r 表示四舍五入精度，S 或 D。

相关属性

格式 = 字节、字、长字、单精度或双精度

指令描述

将源操作数转化为双精度（如果需要）后，加上浮点型目的寄存器里的数，结果存到浮点型目的寄存器。

FADD 操作结果的精度是由 FPCR 所决定的。FSADD 和 FDADD 操作结果是单精度或双精度的，不由 FPCR 决定

操作表

目的	源[1]		
	+ 在范围内 -	+ **0** -	+ 无穷大 -
在范围内 + / -	加	加	+inf　-inf
0 + / -	加	+0.0　0.0[2] 0.0[2]　-0.0	+inf　-inf
无穷大 + / -	+inf -inf	+inf -inf	+inf　NAN[3] NAN[3]　-inf

[1] 如果原操作数是一个 NAN，则阅读第 1.7.1.4 节，–不是一个数‖。

[2] 在圆化模态 RN、RZ 和 RP 中的回返+0.0;在 RM 的回返-0.0。

[3] 在 FPSR 异常位中设置 OPERR。

FPSR[FPCC]:

见第 7.2 节, “条件测试”。

FPSR [EXC]：	BSUN	INAN	IDE	OPERR	OVFL	UNFL	DZ	INEX
	0	见表7-2		设定如果源和目的异常无穷大，则清零	见表7-2		0	见表 7-2

FPSR[AEXC]:

见第 7.1 节, “浮点状态寄存器(FPSR)”。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx				操作码					

指令域

源有效地址域——指定寻址方式。

如果 R/M = 1，这个域描述了源操作数的地址，<ea>y，只有下列的表列出的寻址方式能被使用。

寻址方式	模式	寄存器
Dy[1]	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

[1] 只有在格式为字节，字，长字或单精度。

如果 R/M = 0，这个域不可用，所有的必须清 0。

R/M 域——指定了源操作数的寻址方式。

— 1：从<ea>y 存到寄存器。

— 0：从寄存器存到寄存器。

源指定域——指定源寄存器或数据模式。

如果 R/M = 1，指定源数据的模式，见表 7-10。

如果 R/M = 0，指定源源浮点型数据寄存器，FPy。

目的寄存器域——指定目的浮点型数据寄存器，FPx。

操作码域——指定指令和舍入精度。

操作码	指令	舍入精度
0100010	FADD	用FPCR说明舍入精度
1100010	FSADD	单精度
1100110	FDADD	双精度

FBcc　　Floating-Point Branch Conditionally　　FBcc

（有条件的浮点分支）

指令操作

如果条件正确，则 PC+d_n → PC

汇编格式

FBcc.fmt　　<label>

相关属性

格式 = 字或长字

指令描述

如果描述的情况被遇见，用(PC)+置换继续执行，一 2's—补足整数在 8 位中计算的相对距离。PC 值所决定的目的单元格是由分支地址加 2。

对于字的置换，一个 16 位的数在执行指令操作命令后被储存到一个字当中。

对于长字的置换，一个 32 位的数在执行指令操作命令后被储存到一个长字当中。

描述 cc 选择一个在第 7-2 节–测试条件‖中被描述的测试。

FPSR[FPCC]:

不受影响。

	BSUN	INAN	IDE	OPERR	OVFL	UNFL	DZ	INEX
FPSR [EXC]:	如果NAN位被设置并且选择条件是一IEEE未知测试，则设置	不受影响						

	IOP	OVFL	UNFL	DZ	INEX
FPSR [AEXC]:	如果ECX[BSUN]被设置，则设置	不受影响			

指令格式

15	14	13	12	11	10	9	8	7	6	5 4 3 2 1 0
1	1	1	1	0	0	1	0	1	大小	条件指定
16位的置换或大多数32位有意义的置换										
最少的32位有意义的置换（如果需要）										

指令域 大小域——指定有符号置换的大

小。

如果大小为 1，则置换是 32 位的。

如果大小为 0，则置换是 16 位的，并且是使用前的符合扩展。

条件指定域——指定表 7-4 所定义的有条件的测试。

注意

BSUN 异常会导致指令前异常。如果句柄没有在 FBcc 后更新堆栈框架指向指令指针的 PC 图像，它可能清除 NAN 位或使 BSUN 失去作用，或在返回是再次出现异常。

FCMP　　Floating-point Compare　　FCMP

（浮点数的比较）

指令操作

FPx—源

汇编格式

FCMP.fmt　　<ea>y,FPx

FCMP.D　　FPy,FPx

相关属性

格式 = 字节、字、长字、单精度或双精度

指令描述

将源操作数转变成双精度操作数（如果需要）并且减去浮点型目的寄存器里的操作数。减去的结果没有保存，但被用以第 7-2 节“条件测试”里描述来设定浮点型条件代码。

注意，如果任一操作数被规格化，则它被当作 0。因此，两个被规格化的操作数在比较时是相等的（设置 FPCC[Z]），即使它们不相等。这种情形将被 INEX 或 IDE 发现。

这张表的入口不同于大多数浮点型指令。对于每个输入操作类型的组合，可能设置的条件代码位将被显示。如果条件代码位名字被设定并且没有不在支架中被附上，则它总是被设定的。如果名字在支架中被附上，则位被适当的设定或清除。如果名字没有给出，则操作总是清除位。FCMP 总是清除无穷大位，因为它不是被任何有条件的谓词因素使用。

操作表

目的	源[1]

		+ 在范围内 -		+ **0** -		+ 无穷大 -	
在范围内	+	{NZ}	无	无	无	N	无
	-	N	{NZ}	N	N	N	无
0	+	N	无	Z	Z	N	无
	-	N	无	NZ	NZ	N	无
无穷大	+	无	无	无	无	Z	无
	-	N	N	**N**	**N**	N	NZ

[1] 如果源操作数是一 NAN，请查找章节 1.7.1.4，–不是一个数‖。

注意

NAN 位不能显示，因为 NANs 一样是以同样的方式操作(见章节 1.7.1.4，–不是一个数‖)。

FPSR[FPCC]： 见前面的操作表。

	BSUN	**INAN**	**IDE**	**OPERR**	**OVFL**	**UNFL**	**DZ**	**INEX**
FPSR [EXC]:	0	见表7-2		0	0	0	0	如果任一操作数被规格化，并且同时操作数不正确，IDE无效，则设定；清除其他的。

FPSR[AEXC]:

见章节 7.1，“浮点型状态寄存器(FPSR)”。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx			0	1	1	1	0	0	0

指令域

有效的地址域——指定外部操作数的寻址方式。

如果 R/M = 1，这个域指定源操作数的地址,<ea>y。只有下列的表列出的寻址方式能被使用。

寻址方式	模式	寄存器
Dy[1]	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

[1] 只有在格式为字节，字，长字或单精度。

如果 R/M = 0，这个域不可用，所有的必须清 0。

R/M 域——指定源操作数的寻址方式。

— 1：从<ea>y 存到寄存器。

— 0：从寄存器存到寄存器。

源指定域——指定源寄存器或数据模式。

如果 R/M = 1，指定源数据的模式，见表 7-10。

如果 R/M = 0，指定源源浮点型数据寄存器，FPy。 目的寄存器域——指定目的浮点型数据寄存器，FPx。FCMP 不重写被这个域所指 定的寄存器。

FDIV

Floating-Point Divide

（浮点数的除法）

FDIV

指令操作

FPx/源 → FPx

汇编格式

FDIV.fmt <ea>y,FPx

FDIV.D FPy,FPx

FrDIV.fmt <ea>y,FPx

FrDIV.D FPy,FPx

其中 r 表示四舍五入精度，S 或 D。

相关属性

格式 = 字节、字、长字、单精度或双精度

指令描述

将源操作数转成双精度浮点型操作数（如果需要）,并且用浮点型目的寄存器里的数除以它。把所得的结果放到浮点型目的寄存器里。

FDIV 操作结果的精度是由 FPCR 所决定的。FSDIV 和 FDDIV 操作结果是单精度或双精度的，不由 FPCR 决定。

操作表

目的		源[1]					
		+	在范围内 -	+	**0** -	+	无穷大 -
在范围内	+	除		+inf[2]	-inf[2]	+0.0	-0.0
	-			-inf[2]	+inf[2]	-0.0	+0.0
0	+	+0.0	-0.0	NAN[3]		+0.0	-0.0
	-	-0.0	+0.0			-0.0	+0.0
无穷大	+	+inf	-inf	+inf	-inf	NAN[3]	
	-	-inf	+inf	-inf	+inf		

[1] 如果原操作数是一个 NAN，则阅读第 1.7.1.4 节，–不是一个数‖。

[2] 在 FPSR 异常位中设置 DZ。

[3] 在 FPSR 异常位中设置 OPERR。

FPSR[FPCC]:

见第 7.2 节, “条件测试”。

	BSUN	**INAN**	**IDE**	**OPERR**	**OVFL**	**UNFL**	**DZ**	**INEX**
FPSR [EXC]:	0	见表 7-2		如果 0÷0 或 X÷X，则设定；清楚其他	见表 7-2		如果源是 0 并且目的在范围内，则设定； 清楚其他	见表 7-2

FPSR[AEXC]:

见第 7.1 节, “浮点状态寄存器(FPSR)”。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

1	1	1	1	0	0	1	0	0	0	源操作数有效地址	
										模式	寄存器
0	R/M	0	源指定			目的寄存器 FPx				操作码	

指令域

有效的地址域——指定外部操作数的寻址方式。

如果 R/M = 1，这个域指定源操作数的地址,<ea>y。只有下列的表列出的寻址方式能被使用。

寻址方式	模式	寄存器
Dy[1]	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

[1] 只有在格式为字节，字，长字或单精度。

如果 R/M = 0，这个域不可用，所有的必须清 0。

R/M 域——指定源操作数的寻址方式。

—1：从<ea>y 存到寄存器。

—0：从寄存器存到寄存器。

源指定域——指定源寄存器或数据模式。

如果 R/M = 1，指定源数据的模式，见表 7-10。

如果 R/M = 0，指定源源浮点型数据寄存器，FPy。

目的寄存器域——指定目的浮点型数据寄存器，FPx。

操作码域——指定指令和舍入精度。

操作码	指令	舍入精度
0100000	FDIV	用 FPCR 说明舍入精度
1100000	FSDIV	单精度
1100100	FDDIV	双精度

FINT　　FINT

Floating-point Integer

（浮点数取整）

指令操作

取源的整数部分 → FPx

汇编格式

FINT.fmt　　<ea>y,FPx

FINT.D　　FPy,FPx

FINT.D　　FPx

其中 r 表示四舍五入精度，S 或 D。

相关属性

格式 = 字节、字、长字、单精度或双精度

指令描述

将源操作数转成双精度浮点型操作数（如果需要）,取出整数部分，并且转化成双精度值。把结果储存到发短信目的寄存器。整数部分在使用 FPCR 状态控制位所决定的当前的模式由双精度转化成整数。因此，当指数是 0 时整数部分是小数点的左边部分。

如：

137.57 的整数部分是 137.0（在向负无穷舍入模式时，137.57 向零取整的整数部分是 137.0，而在向正无穷舍入模式时，最近舍入是 138.0）注意：这操作的结果是一个浮点数。

操作表

目的	源[1]		
	+ 在范围内 -	+ **0** -	+ 无穷大 -
结果	整数	+0.0 -0.0	+inf -inf

[1] 如果原操作数是一个 NAN，则阅读第 1.7.1.4 节，–不是一个数‖。

FPSR[FPCC]:

见第 7.2 节,“条件测试”。

FPSR	**BSUN**	**INAN**	**IDE**	**OPERR**	**OVFL**	**UNFL**	**DZ**	**INEX**
[EXC]:	0	见表 7-2		0	0	0	0	见表 7-2

FPSR[AEXC]:

见章节 7.1，“浮点型状态寄存器(FPSR)”。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx			0	0	0	0	0	0	1

指令域

源有效的地址域——指定外部操作数的寻址方式。

如果 R/M = 1，这个域指定源操作数的地址,<ea>y。只有下列的表列出的寻址方式能被使用。

寻址方式	模式	寄存器
Dy[1]	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

[1] 只有在格式为字节，字，长字或单精度。

如果 R/M = 0，这个域不可用，所有的必须清 0。

R/M 域——指定源操作数的寻址方式。

—1：从<ea>y 存到寄存器。

—0：从寄存器存到寄存器。

源指定域——指定源寄存器或数据模式。

如果 R/M = 1，指定源数据的模式，见表 7-10。

如果 R/M = 0，指定源源浮点型数据寄存器，FPy。

目的寄存器域——指定目的浮点型数据寄存器，FPx。

如果 R/M = 0，并且源和目的域相等，输入的操作数来自指定的浮点寄存器，结果被写到同样的寄存器。如果是单个寄存器被使用，则摩托罗拉组合器将源和目的域设定为相同的数。

FINTRZ Floating-Point Integer Round-to-Zero FINTRZ

（浮点型向零取整）

指令操作

取源的整数部分 → FPx

汇编格式

FINTRZ.fmt <ea>y,FPx
FINTRZ.D FPy,FPx
FINTRZ.D FPx

其中r表示四舍五入精度，S或D。

相关属性

格式 = 字节、字、长字、单精度或双精度

指令描述

将源操作数转成双精度浮点型操作数（如果需要）,取出整数部分，并且转化成双精度值。把结果储存到发短信目的寄存器。整数部分是使用向零取整模式由双精度转化为整数，不由 FPCR 状态控制位决定。因此，当指数是 0 时整数部分是小数点的左边部分。

如：

137.57 的整数部分是 137.0。 注意：这

操作的结果是一个浮点数字。

操作表

目的	源[1]		
	+ 在范围内 -	+ **0** -	+ 无穷大 -
结果	整数，强制向零取整	+0.0 -0.0	+inf -inf

[1] 如果原操作数是一个 NAN，则阅读第 1.7.1.4 节，–不是一个数‖。

FPSR[FPCC]：

见第 7.2 节, “条件测试”。

FPSR [EXC]：

BSUN	INAN	IDE	OPERR	OVFL	UNFL	DZ	INEX
0	见表 7-2		0	0	0	0	见表 7-2

FPSR[AEXC]:

见章节 7.1，“浮点型状态寄存器(FPSR)”。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx			0	0	0	0	0	1	1

指令域

有效的地址域——指定外部操作数的寻址方式。

如果 R/M = 1，这个域指定源操作数的地址,<ea>y。只有下列的表列出的寻址方式能被使用。

寻址方式	模式	寄存器
Dy[1]	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

[1] 只有在格式为字节，字，长字或单精度。

如果 R/M = 0，这个域不可用，所有的必须清 0。

R/M 域——指定源操作数的寻址方式。

— 1：从<ea>y 存到寄存器。

— 0：从寄存器存到寄存器。

源指定域——指定源寄存器或数据模式。

如果 R/M = 1，指定源数据的模式，见表 7-10。

如果 R/M = 0，指定源源浮点型数据寄存器，FPy。

目的寄存器域——指定目的浮点型数据寄存器，FPx。

如果 R/M = 0，并且源和目的域相等，输入的操作数来自指定的浮点寄存器，

结果被写到同样的寄存器。如果是单个寄存器被使用，则摩托罗拉组合器将源和目的域设定为相同的数。

FMOVE　　Move Floating-Point Data Register　　FMOVE

（浮点型寄存器的移动）

指令操作

源 → 目的

汇编格式

FMOVE.fmt	<ea>y,FPx
FMOVE.fmt	FPy,<ea>x
FMOVE.D	FPy,FPx
FrMOVE.fmt	<ea>y,FPx
FrMOVE.D	FPy,FPx

其中 r 表示四舍五入精度，S 或 D。

相关属性

格式 = 字节、字、长字、单精度或双精度

指令描述

把源操作数的内容移到目的操作数中。虽然 FMOVE 的原函数的功能是数据移动，但是它考虑到算术指令，因为从源操作格式到目的操作格式的转变在暗中发生。并且，源操作数的舍入是根据舍入精度和格式的选择。

不像 MOVE，FMOVE 不支持从内存到内存的格式。对于像浮点数之间的移动，MOVE 比 FMOVE 快许多。FMOVE 支持从内存到寄存器的，寄存器到寄存器的和寄存器到内存的移动操作(这里的内存包括字节、字、长字和单精度的整形数据寄存器)。从内存到寄存器和寄存器到寄存器的操作使用的指令字编码异于从寄存器到内存的;这两种操作指令集被分开描述。

对于从内存到寄存器和寄存器到寄存器的操作(<ea>y,FPx；FPy,FPx)： 将源操作数转成双精度浮点型操作数（如果需要），并且转化成双精度值。把结果储存到发短信目的寄存器，Fpx。FMOVE 结果的精度是由 FPCR 所决定的。FSMOVE 和 FDMOVE 的结果是单精度或双精度，不由 FPCR 决定。注意，如果源格式是长字或双精度，当为单精度时，结果可能是错误的。其他的所有源格式合并起来，并且精确是一正确的结果。

FPSR[FPCC]:

见第 7.2 节,“条件测试”。

FPSR	BSUN	INAN	IDE	OPERR	OVFL	UNFL	DZ	INEX
[EXC]:	0	见表 7-2		0	0	0	0	见表 7-2

FPSR[AEXC]:

见章节 7.1，“浮点型状态寄存器(FPSR)”。

指令格式

<ea>y,FPx；FPy,FPx

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx				操作码					

指令域

有效的地址域——指定外部操作数的寻址方式。

如果 R/M = 1，这个域指定源操作数的地址,<ea>y。只有下列的表列出的寻址方式能被使用。

寻址方式	模式	寄存器
Dy[1]	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

[1] 只有在格式为字节，字，长字或单精度。

如果 R/M = 0，这个域不可用，所有的必须清 0。

R/M 域——指定源操作数的寻址方式。

—1：从<ea>y 存到寄存器。

—0：从寄存器存到寄存器。 源指定域——指定源寄存器或数据模式。

如果 R/M = 1，指定源数据的模式，见表 7-10。 如果 R/M = 0，指定源源浮点型数据寄存器，FPy。

目的寄存器域——指定目的浮点型数据寄存器，FPx。

操作码域——指定指令和舍入精度。

操作码	指令	舍入精度
0000000	FMOVE	用 FPCR 说明舍入精度
1000000	FSMOVE	单精度
1000100	FDMOVE	双精度

从寄存器的内存操作**(FPy,<ea>x)**：

把源操作数转化为指定的目的格式，并且储存到目的有效地址，<ea>x。

FPSR[FPCC]：

不受影响。

<table>
<tr><th>FPSR</th><th>BSUN</th><th>INAN</th><th>IDE</th><th>OPERR</th><th>OVFL</th><th>UNFL</th><th>DZ</th><th>INEX</th></tr>
<tr><td rowspan="2">[EXC]：
格式:.B,.W 或.L
格式:.S 或.D</td><td rowspan="2">0</td><td rowspan="2" colspan="2">见表 7-2</td><td>在转化和舍入后，如果源操作数是 X 或目的大小太大，则设置。</td><td>0</td><td>0</td><td>0</td><td rowspan="2">见表 7-2</td></tr>
<tr><td>0</td><td colspan="2">见表 7-2</td><td>0</td></tr>
</table>

FPSR[AEXC]：

见第 7.1 节, “浮点状态寄存器(FPSR)”。

指令格式

FPy,<ea>x

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
0	1	1	目的指定			源寄存器 FPy			0	0	0	0	0	0	0

指令域

目的有效的地址域——指定目的地址，<ea>x。只有下列的表列出的寻址方式能被使用。

寻址方式	模式	寄存器
Dx[1]	000	寄存器码:Dx
Ax	—	—
(Ax)	010	寄存器码:Ax
(Ax)+	011	寄存器码:Ax
-(Ax)	100	寄存器码:Ax
(d_{16},Ax)	101	寄存器码:Ax
(d_8,Ax,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

[1] 只有在格式为字节，字，长字或单精度。 目的格式域——指定目的操作数的格式，见表 7-10。 源寄存器域——指定源浮点型数据寄存器，FPy。

FMOVE from FPCR

FMOVE from FPCR

Move from the floating-point Control Register

（从浮点型控制寄存器的移动）

指令操作

FPCR → 目的

汇编格式

FMOVE.L FPCR,<ea>x

相关属性

格式 = 长字。

指令描述

移动FPCR的内容移到一有效地地址。一32位的移动总是被执行，即使FPCR的32位没有全部被执行。控制寄存器中未执行的位将被读做0。异常将不执行这个指令。

FPSR： 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

指令域

有效的地址域——指定寻址方式，<ea>x。见下表：

寻址方式	模式	寄存器
Dx	000	寄存器码:Dx
Ax	—	—
(Ax)	010	寄存器码:Ax
(Ax)+	011	寄存器码:Ax
-(Ax)	100	寄存器码:Ax
(d_{16},Ax)	101	寄存器码:Ax
(d_8,Ax,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

FMOVE from FPIAR

Move from the floating-point Instruction Address Register

（从浮点型地址指令寄存器的移动）

指令操作

FPIAR → 目的

汇编格式

FMOVE.L FPIAR,<ea>x

相关属性

格式 = 长字。

指令描述

移动浮点型顺序控制寄存器的内容移到一有效地地址。异常将不执行这个指令。

FPSR： 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0

指令域

有效的地址域——指定寻址方式，<ea>x。见下表：

寻址方式	模式	寄存器
Dx	000	寄存器码:Dx
Ax	001	寄存器码:Ax
(Ax)	010	寄存器码:Ax
(Ax)+	011	寄存器码:Ax
-(Ax)	100	寄存器码:Ax
(d_{16},Ax)	101	寄存器码:Ax
(d_8,Ax,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

FMOVE from FPSR FMOVE from FPSR

Move from the floating-point Status Register

（从浮点型状态寄存器的移动）

指令操作

FPSR → 目的

汇编格式

FMOVE.L FPSR,<ea>x

相关属性

格式 = 长字。

指令描述

移动 FPSR 的内容移到一有效地地址。一 32 位的移动总是被执行，即使 FPSR 的 32 位没有全部被执行。控制寄存器中未执行的位将被读做 0。异常将不执行这个指令。

FPSR： 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0

指令域

有效的地址域——指定寻址方式，<ea>x。见下表：

寻址方式	模式	寄存器
Dx	000	寄存器码:Dx
Ax	—	—
(Ax)	010	寄存器码:Ax
(Ax)+	011	寄存器码:Ax
-(Ax)	100	寄存器码:Ax
(d_{16},Ax)	101	寄存器码:Ax
(d_8,Ax,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	—	—
(d_8,PC,Xi)	—	—

FMOVE to FPCR

FMOVE to FPCR

Move to the floating-point Control Register

（移动到浮点型控制寄存器）

指令操作

源 → FPCR

汇编格式

FMOVE.L <ea>y,FPCR

相关属性

格式 = 长字。

指令描述

从一有效地地址载入 FPCR。一 32 位的移动总是被执行，即使 FPCR 的 32 位没有全部被执行。在储存时未执行的位将被忽略（由于将来安装设备驱动程序的兼容性而必须置零）。异常将不执行这个指令。

FPSR： 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					

										模式			寄存器		
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

指令域

有效的地址域——指定寻址方式，<ea>y。见下表：

寻址方式	模式	寄存器
Dy	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

FMOVE to FPIAR

FMOVE to FPIAR

Move to the floating-point Instruction Address Register

（移动到浮点型地址指令寄存器）

指令操作

源 → FPIAR

汇编格式

FMOVE.L <ea>y,FPIAR

相关属性

格式 = 长字。

指令描述

从一有效地地址载入浮点型顺序控制寄存器。异常将不执行这个指令。

FPSR： 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

指令域

有效的地址域——指定寻址方式，<ea>y。见下表：

寻址方式	模式	寄存器
Dy	000	寄存器码:Dy
Ay	001	寄存器码:Ay
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

		码:Ay
(d_{16},Ay)	101	寄存器 码:Ay
(d_8,Ay,Xi)	—	—

FMOVE to FPSR

FMOVE to FPSR

Move to the Floating-Point Status Register

（移动到浮点型状态寄存器）

指令操作

源 → FPSR

汇编格式

FMOVE.L <ea>y,FPSR

相关属性

格式 = 长字。

指令描述

从一有效地地址载入 FPSR。一 32 位的移动总是被执行，即使 FPSR 的 32 位没有全部被执行。在储存时未执行的位将被忽略（由于将来安装设备驱动程序的兼容性而必须置零）。异常将不执行这个指令。

FPSR： 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

指令域 有效的地址域——指定寻址方式，<ea>y。见下表：

寻址方式	模式	寄存器
Dy	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

FMOVEM Move Multiple floating-point Data Registers FMOVEM

（浮点型寄存器乘法的移动）

指令操作

寄存器表列 → 目的
源 → 寄存器表列

汇编格式

FMOVEM.D #list,<ea>x
FMOVEM.D <ea>y,#list

相关属性

格式 = 双精度。

指令描述

移动一个或更多双精度数储存到浮点型数据寄存器列表中，或从其中移出。在这个指令中不可能在 FPSR 不受指令的影响下转化或舍入被完成。异常将不执行这个指令。用用户所指定的掩码来指定可选择的寄存器，八位浮点数据寄存器的任何组合能被转移这个标记是个 8 位的数，并且每一位对应一个寄存器；如果一字节在标记中被设定，则寄存器被移动。注意，空寄存器（全 0）产生一行 F 的异常。

FMOVEM 允许的两种寻址方式：间接的地址寄存器和基址寄存器加上 16 位的偏移，这里基址是一个寄存器，或只适合于加载程序计数器。总之，处理器只计算起始地址和后来寄存器移动所产生的 8 位增量。转移指令总是 FP0-FP7。

注意

FMOVEM 在没有转换数据或特殊的条件代码和异常状态位的情况下，提供唯一的方法在 FPU 和内存之间的浮点型数据移动。

FPSR： 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	有效地址					
										模式			寄存器		
1	1	dr	1	0	0	0	0	寄存器列表							

指令域

有效的地址域——指定寻址方式。从内存到寄存器允许<ea>y 模式见下表：

寻址方式	模式	寄存器
Dy	—	—
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	—	—
-(Ay)	—	—
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

有效的地址域——指定寻址方式。从寄存器到内存允许<ea>x 模式见下表：

寻址方式	模式	寄存器
Dx	—	—
Ax	—	—
(Ax)	010	寄存器码:Ax
(Ax)+	—	—
-(Ax)	—	—
(d_{16},Ax)	101	寄存器码:Ax
(d_8,Ax,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

dr 域——指定转移的传递方向。

— 0：把寄存器列表从内存移动到 FPU。

— 1：把寄存器列表从 FPU 移动到内存。

寄存器列表域—包括寄存器选择标准。如果一个寄存器被移动，相应的屏蔽位被如下设定；其他清零。

7	6	5	4	3	2	1	0
FP0	FP1	FP2	FP3	FP4	FP5	FP6	FP7

FMUL　　Floating-Point Multiply　　FMUL

（浮点数的乘法）

指令操作

源*FPx → FPx

汇编格式

FMUL.fmt　<ea>y,FPx
FMUL.D　FPy,FPx
FrMUL.fmt　<ea>y,FPx
FrMUL.D　FPy,FPx

其中 r 表示四舍五入精度，S 或 D。

相关属性

格式 = 字节、字、长字、单精度或双精度

指令描述

将源操作数转成双精度浮点型操作数（如果需要）,乘以储存在浮点型目的数据寄存器里的数。结果储存到浮点型目的寄存器。

FMUL 操作结果的精度是由 FPCR 所决定的。FSMUL 和 FDMUL 操作结果是单精度或双精度的，不由 FPCR 决定。

操作表

目的		源[1]					
		+ 在范围内 -		+ **0** -		+ 无穷大 -	
在范围内	+	乘		+0.0	-0.0	+inf	-inf
	-			-0.0	+0.0	-inf	+inf
0	+	+0.0	-0.0	+0.0	-0.0	NAN	
	-	-0.0	+0.0	-0.0	+0.0		
无穷大	+	+inf	-inf	NAN[2]		+inf	-inf
	-	-inf	+inf			-inf	+inf

[1] 如果原操作数是一个 NAN，则阅读第 1.7.1.4 节，–不是一个数‖。

[2] 在 FPSR 异常位中设置 OPERR。

FPSR[FPCC]:

见第 7.2 节, “条件测试”。

FPSR [EXC]:	**BSUN**	**INAN**	**IDE**	**OPERR**	**OVFL**	**UNFL**	**DZ**	**INEX**
	0	见表7-2		设定，如果 0XX；清楚其他	见表7-2		0	见表 7-2

FPSR[AEXC]:

见第 7.1 节,‖浮点状态寄存器(FPSR)‖。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx				操作码					

指令域

有效的地址域——指定外部操作数的寻址方式。

如果 R/M = 1，这个域指定源操作数的地址。只有下列的表列出的寻址方式能被使用。

寻址方式	模式	寄存器
Dy[1]	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

[1] 只有在格式为字节，字，长字或单精度。

如果 R/M = 0，这个域不可用，所有的必须清 0。

R/M 域——指定源操作数的寻址方式。

— 1：从<ea>y 存到寄存器。

— 0：从寄存器存到寄存器。

源指定域——指定源寄存器或数据模式。

如果 R/M = 1，指定源数据的模式，见表 7-10

如果 R/M = 0，指定源源浮点型数据寄存器，FPy。

目的寄存器域——指定目的浮点型数据寄存器，FPx。

操作码域——指定指令和舍入精度。

操作码	指令	舍入精度
0100011	FMUL	用FPCR说明舍入精度
1100011	FSMUL	单精度
1100111	FDMUL	双精度

FNEG　　Floating-Point Negate　　FNEG

（浮点数取反）

指令操作

-(源) → FPx

汇编格式

FNEG.fmt　　<ea>y,FPx
FNEG.D　　FPy,FPx
FNEG.D　　FPx
FrNEG.fmt　　<ea>y,FPx
FrNEG.D　　FPy,FPx
FrNEG.D　　FPx

其中 r 表示四舍五入精度，S 或 D。

相关属性

格式 = 字节、字、长字、单精度或双精度

指令描述

将源操作数转成双精度浮点型操作数（如果需要）颠倒尾数的符号。结果储存到浮点型目的寄存器，FPx。

FNEG 操作结果的精度是由 FPCR 所决定的。FSNEG 和 FDNEG 操作结果是单精度或双精度的，不由 FPCR 决定。

操作表

目的	源[1]		
	+ 在范围内 -	+ **0** -	+ 无穷大 -
结果	否定	+0.0 -0.0	+inf -inf

[1] 如果原操作数是一个 NAN，则阅读第 1.7.1.4 节，–不是一个数‖。

FPSR[FPCC]:

见第 7.2 节,“条件测试”。

FPSR	**BSUN**	**INAN**	**IDE**	**OPERR**	**OVFL**	**UNFL**	**DZ**	**INEX**
[EXC]:	0	见表7-2		0	0	0	0	0

FPSR[AEXC]:

见章节 7.1，“浮点型状态寄存器(FPSR)”。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx			操作码						

指令域

有效的地址域——指定外部操作数的寻址方式。

如果 R/M = 1，这个域指定源操作数的地址。只有下列的表列出的寻址方式能被使用。

寻址方式	模式	寄存器
Dy[1]	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

[1] 只有在格式为字节，字，长字或单精度。

如果 R/M = 0，这个域不可用，所有的必须清 0。

R/M 域——指定源操作数的寻址方式。

—1：从<ea>y 存到寄存器。

—0：从寄存器存到寄存器。

源指定域——指定源寄存器或数据模式。

如果 R/M = 1，指定源数据的模式，见表 7-10

如果 R/M = 0，指定源源浮点型数据寄存器，FPy。

目的寄存器域——指定目的浮点型数据寄存器，FPx。

如果 R/M = 0，并且源和目的域相等，输入的操作数来自指定的浮点寄存器，结果被写到同样的寄存器。如果是单个寄存器被使用，则摩托罗拉组合器将源和目的域设定为相同的数。

操作码域——指定指令和舍入精度。

操作码	指令	舍入精度
0011010	FNEG	用FPCR说明舍入精度
1011010	FSNEG	单精度
1011110	FDNEG	双精度

FNOP FNOP

No Operation

（空操作）

指令操作

无

汇编格式

FNOP

相关属性

无

指令描述

FNOP 运行不清楚的操作。它通常与带整数单元的 FPU 同步或强行执行未决的异常。由于大部分的浮点操作，一旦 FPU 没有操作需要执行，整数单元继续执行下一步指令，从而支撑整型和浮点型指令的同时执行。FNOP 导致整型单元等待先前的浮点型指令来完善。它也强制使先前的浮点型指令未执行的异常作预处理指令执行。FNOP 的操作码是 0xF280 0000。

FPSR： 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

注意

FNOP 和 FBcc.W 使用同样的代码指令。cc = F（捕错）且<label> = + 2（结果是 0 的置换）。

Floating-Piont Square Root

（浮点数的平方根）

指令操作

源的平方根 → FPx

汇编格式

FSQRT.fmt <ea>y,FPx
FSQRT.D FPy,FPx
FSQRT.D FPx
FrSQRT.fmt <ea>y,FPx
FrSQRT.D FPy,FPx
FrSQRT.D FPx

其中 r 表示四舍五入精度，S 或 D。

相关属性

格式 = 字节、字、长字、单精度或双精度

指令描述

将源操作数转成双精度浮点型操作数（如果需要）计算那个数的平方根。结果储存到浮点型目的寄存器，FPx。这个函数没有定义负数。

FSQRT 操作结果的精度是由 FPCR 所决定的。FSSQRT 和 FDSQRT 操作结果是单精度或双精度的，不由 FPCR 决定。

操作表

目的	源[1]		
	+ 在范围内 -	+ **0** -	+ 无穷大 -
结果	$\sqrt{x}$ NAN[2]	+0.0 -0.0	+inf -inf

[1] 如果原操作数是一个 NAN，则阅读第 1.7.1.4 节，–不是一个数‖。

[2] 在 FPSR 异常位中设置 OPERR。

FPSR[FPCC]:

见第 7.2 节,“条件测试”。

FPSR [EXC]:

BSUN	INAN	IDE	OPERR	OVFL	UNFL	DZ	INEX
0	见表7-2		如果源操作数不是0且是无意义的则设定，清除其他的。	0	0	0	见表7-2

FPSR[AEXC]:

见章节 7.1，“浮点型状态寄存器(FPSR)”。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx				操作码					

指令域

有效的地址域——指定外部操作数的寻址方式。

如果 R/M = 1，这个域指定源操作数的地址，<ea>y。只有下列的表列出的寻址方式能被使用。

寻址方式	模式	寄存器
Dy[1]	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

[1] 只有在格式为字节，字，长字或单精度。

如果 R/M = 0，这个域不可用，所有的必须清 0。

R/M 域——指定源操作数的寻址方式。

— 1：从<ea>y 存到寄存器。

— 0：从寄存器存到寄存器。

源指定域——指定源寄存器或数据模式。

如果 R/M = 1，指定源数据的模式，见表 7-10

如果 R/M = 0，指定源源浮点型数据寄存器，FPy。

目的寄存器域——指定目的浮点型数据寄存器，FPx。

如果 R/M = 0，并且源和目的域相等，输入的操作数来自指定的浮点寄存器，结果被写到同样的寄存器。如果是单个寄存器被使用，则摩托罗拉组合器将源和目的域设定为相同的数。

操作码域——指定指令和舍入精度。

操作码	指令	舍入精度

0000100	FSQRT	用FPCR说明舍入精度
1000001	FSSQRT	单精度
1000101	FDSQRT	双精度

FSUB　　FSUB

Floating-point Subtract

（浮点数的减法）

指令操作

FPx-源 → FPx

汇编格式

FSUB.fmt　　<ea>y,FPx

FSUB.D　　FPy,FPx

FrSUB.fmt　　<ea>y,FPx

FrSUB.D　　FPy,FPx

其中r表示四舍五入精度，S或D。

相关属性

格式 = 字节、字、长字、单精度或双精度

指令描述

将源操作数转成双精度浮点型操作数（如果需要）用目的地浮点数据寄存器的数字减去它。结果储存到浮点型目的寄存器。

操作表

目的	源[1]		
	+ 在范围内 -	+ **0** -	+ 无穷大 -
在范围内 +	减	减	-inf
在范围内 -			+inf
0 +	减	0.0[2] +0.0	-inf
0 -		-0.0 0.0[2]	+inf
无穷大 +	+inf	+inf	NAN[3] +inf
无穷大 -	-inf	-inf	-inf NAN[3]

[1] 如果原操作数是一个 NAN，则阅读第 1.7.1.4 节，–不是一个数‖。

[2]在圆化模态RN、RZ和RP中的回返+0.0;在RM的回返-0.0。

[3] 在 FPSR 异常位中设置 OPERR。

FPSR[FPCC]:

见第 7.2 节, “条件测试”。

FPSR [EXC]:

BSUN	INAN	IDE	OPERR	OVFL	UNFL	DZ	INEX
0	见表7-2		如果源操和目的是同号无穷大则设定，清除其他的。	见表7-2		0	见表7-2

FPSR[AEXC]:

见章节 7.1，“浮点型状态寄存器(FPSR)”。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx				操作码					

指令域

有效的地址域——指定外部操作数的寻址方式。

如果 R/M = 1，这个域指定源操作数的地址，<ea>y。只有下列的表列出的寻址方式能被使用。

寻址方式	模式	寄存器
Dy^1	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

[1] 只有在格式为字节，字，长字或单精度。

如果 R/M = 0，这个域不可用，所有的必须清 0。

R/M 域——指定源操作数的寻址方式。

— 1：从<ea>y 存到寄存器。

— 0：从寄存器存到寄存器。

源指定域——指定源寄存器或数据模式。

如果 R/M = 1，指定源数据的模式，见表 7-10

如果 R/M = 0，指定源源浮点型数据寄存器，FPy。

目的寄存器域——指定目的浮点型数据寄存器，FPx。

操作码域——指定指令和舍入精度。

操作码	指令	舍入精度
0101000	FSUB	用FPCR说明舍入精度
1101000	FSSUB	单精度
1101100	FDSUB	双精度

FTST　　Test floating-Point Opreand　　FTST

（浮点数的测试）

指令操作

源操作数的测试 → FPCC

汇编格式

FTST.fmt　　<ea>y

FTST.D　　FPy

相关属性

格式 = 字节、字、长字、单精度或双精度

指令描述

将源操作数转成双精度浮点型操作数（如果需要）且根据结果的数据类型来设定条件码位。注意，对于非规格化操作数，FPCC[Z]被设定（因为非规格化操作数通常被当作 0 处理）。当 Z 被设定时，如果操作数是非规格化数据（且 IDE 不可用）时 INEX 被设定。如果操作数是 0 则 INEX 被清除。

操作表

目的	源[1]								
	+	在范围内	-	+	**0**	-	+	无穷大	-
结果	无	整数	N	Z		NZ	I		NI

[1] 如果原操作数是一个 NAN，则阅读第 1.7.1.4 节，–不是一个数‖。 注意，此操作表不同与其他的。表的入口字母，FTST 经常指定条件代码位。

在操作时所有的未指定条件码都被清除。

FPSR[FPCC]:

见第 7.2 节,“条件测试”。

	BSUN	INAN	IDE	OPERR	OVFL	UNFL	DZ	INEX
FPSR [EXC]:	0	见表7-2		0	0	0	0	如果非规格化且IDE不可用则置位；否则清零。

FPSR[AEXC]:

见章节 7.1，“浮点型状态寄存器(FPSR)”。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源操作数有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx			0	1	1	1	0	1	0

指令域

有效的地址域——指定外部操作数的寻址方式。

如果 R/M = 1，这个域指定源操作数的地址，<ea>y。只有下列的表列出的寻址方式能被使用。

寻址方式	模式	寄存器
Dy[1]	000	寄存器码:Dy
Ay	—	—
(Ay)	010	寄存器码:Ay
(Ay)+	011	寄存器码:Ay
-(Ay)	100	寄存器码:Ay
(d_{16},Ay)	101	寄存器码:Ay
(d_8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d_{16},PC)	111	010
(d_8,PC,Xi)	—	—

[1] 只有在格式为字节，字，长字或单精度。

如果 R/M = 0，这个域不可用，所有的必须清 0。

R/M 域——指定源操作数的寻址方式。

— 1：从<ea>y 存到寄存器。

— 0：从寄存器存到寄存器。

源指定域——指定源寄存器或数据模式。

如果 R/M = 1，指定源数据的模式，见表 7-10

如果 R/M = 0，指定源源浮点型数据寄存器，FPx。

目的寄存器域—FTST 使用的指令字格式是所有的 FPU 算术指令所使用的但忽略并不写过多的这个域所指定的寄存器。由于将来安装设备驱动程序的兼容性而必须

置零；然而因为这个域为 FTST 指令所忽略，如果这个域不是 0，FPU 不发一异常信号。

第 8 章 超级用户(特权)指令

本节包含了 ColdFire 系列超级用户(特权)指令的相关信息。为了便于记忆，每条指令的详细说明按字母顺序排列。可选核心模块（如浮点型运算单元）的超级用户指令也详载于本节。

并非所有的指令都被 ColdFire 处理器所支持。请参见第 3 章“就指令集定义的具体细节的指令集概要”。

CPUSHL CPUSHL

Push and Possibly Invalidate Cache

首先出现于 ISA_A

指令操作

If 特权状态

Then If 数据可用且已经被更改

压入缓存行单元

Then 若相关的 CACR 位已被设置，则使行单元无效

Else 特权违反异常

汇编格式

CPUSHL dc,(Ax) 数据缓存

CPUSHL ic,(Ax) 指令缓存

CPUSHL bc,(Ax) 数据/指令缓存或通用缓存

相关属性 尺寸不

确定。

指令描述

把指定的已经被修改缓存行单元入栈，若设置 CACR[DPI]，此操作同时使得它无效。若缓存大小不同，则两种类型缓存行单元被清零时，应小心行事。举例来说，采用 16K 指令高速缓存和 8K 数据高速缓存设备时，地址 0x800（适用于两种类型的缓存）对指令类型的缓存来说，关联于地址 0x80，但对数据类型的缓存来说，却关联于地址 0x00。注意，该指令可以同步管道（Pipeline）。

条件码 不受影

响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	缓存		1	0	1	寄存器 Ax		

指令域

缓存域——指明缓存类型如下：

— 00：保留

— 01：数据缓存（dc）

— 10：指令缓存（ic）

— 11：两种类型缓存或标准缓存（bc）。也有可能的是，有些指令缓存部件也使用此编码，而非数据缓存。

寄存器 Ax 域——指明用于定义缓存行单元的地址寄存器是被压入堆栈还是暂不适用。Ax 被编码如下：

— Ax[4]是地址域的 LSB，可根据给定的缓存大小向上扩展。计算该地址域大小的方法具体如下：

范围 = 缓存的字节数 / (联合路数 * 16)。

以大小为 16K 的 4 路联合缓存为例：

范围 = 16384 / (4 * 16) = 256 = 2^8 。

因此，地址范围为 Ax[11:4]。

— Ax[1:0]指行单元被确定时的缓存方式和级别。

FRESTORE FRESTORE

Restore Internal

Floating-Point State

首先出现于 ISA_A

指令操作

If 特权状态

Then FPU 状态帧 → 内部状态

Else 特权违反异常

汇编格式

CPUSHL dc,(Ax) data cache

CPUSHL ic,(Ax) instruction cache

CPUSHL bc,(Ax) both caches or unified cache

相关属性

尺寸不确定。

指令描述

异常中止任何浮点型操作，并用有效地址的状态帧装载新的 FPU 内部状态。该帧格式是由字节<ea>y 指定，且其内部异常处理向量载于字节<ea>y+1 中。如果帧格式无效，FRESTORE 将产生异常中止，格式异常将形成（向量 14）。如果该格式有效，则帧内容将被加载到 FPU，先是指定的地址，继而是较高的地址。

所有向量都产生于 FPCR 和 FPSR 异常位，因此 FRESTORE 忽略了字节<ea>y+1 中指定的向量。该向量将被提供给管理器（Handler）。

除空状态帧外，FRESTORE 一般不会影响到 FPU 编程模型。通常它会与 FMOVEM 连用，以充分恢复包括浮点数据和系统控制寄存器在内的 FPU 上下语境（context）。为完成修复，首先 FMOVEM 加载数据寄存器，然后 FRESTORE 加载内部状态、FPCR 和 FPSR。表 8-1 列出了相关的状态帧。如果帧格式不是 0x00、0x05 或 0xe5，则处理器将报告格式错误异常（向量 14），此时内部 FPU 状态不受影响。

表 **8-1** 状态帧

状态	格式	描述
NULL	0x00	对该状态帧做 FRESTORE 操作，很类似于 FPU 硬件复位。程序员的编程模式将进入复位状态，同时浮点数据寄存器为 NANs 状态，FPCR、FPSR 和 FPIAR 全为零。
IDLE	0x05	对 IDLE 和 EXCP 状态帧做 FRESTORE 操作，会产生同样的效果。若没有正在等候的异常，FPU 将恢复到闲置状态，等待下条指令的开始。但是，如果 FPSR[EXC]位和与其相应的 FPCR 使能位都被置位，则 FPU 将进入异常状态。在该状态下，初始化不同于 FSAVE、FMOVEM 和 FMOVE 系统寄存器的浮点指令，或 FRESTORE 造成等候异常。程序员的编程模式，在加载这种类型的状态帧时不受影响（FPSR 和 FPCR 从状态帧加载除外）。
EXCP	0xE5	

FPSR

若格式为 NULL 则清零，否则从状态帧加载。

FPCR

若格式为 NULL 则清零，否则从状态帧加载。

FPIAR

若格式为 NULL 则清零，否则不变。

浮点数据寄存器

若为 NULL 则被设为 NANs，否则不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	0	1	源有效地址					
										模式			寄存器		

指令域

源有效地址域——指明寻址方式、<ea>y，用于状态帧。只有以下模式被用到：

寻址方式	模式	寄存器
Dy	—	—
Ay	—	—
(Ay)	010	寄存器号:Ay
(Ay)+	—	—
-(Ay)	—	—
(d16,Ay)	101	寄存器号:Ay
(d8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d16,PC)	111	010
(d8,PC,Xi)	—	—

FSAVE FSAVE

Save Intenal

Floating-Point State

首先出现于 ISA_A

指令操作

If 特权状态

Then FPU 内部状态 → <ea>x

Else 特权违反异常

汇编格式

FSAVE<ea>x

相关属性 尺寸不

确定。

指令描述

允许完成运行中的任何浮点型操作之后，FSAVE 将 FPU 内部状态保存到有效地址指定的帧中。保存操作后，FPCR 被清零，FPU 处于空闲状态，直到下个指令执行。被写入状态帧的首个长字包含了格式域的数据。运行中的浮点运算，当遇到 FSAVE 指令时，在 FSAVE 执行前，若无异常发生，则它将创建一个闲置（IDLE）状态帧，否则 EXCP 状态帧将被创建。状态帧由表 8-2 给出。

表 **8-2** 状态帧

状态	描 述
NULL	FSAVE 将生成这种状况帧，用来表示，因为上次处理器复位或执行了带 NULL 状态帧的 FRESTORE 操作，FPU 状态没有改变。这表明，编程器的模型处于复位状态，浮点数据寄存器为 NANs 状态，FPCR、FPSR 和 FPIAR 全为零。系统寄存器存储、FSAVE 和 FMOVEM 存储，并不引发 FPU 从零到其他状态的转变。
IDLE	FSAVE 生成这种状况帧，用来表示 FPU 在闲置状态下已执行完毕，并在没有等候异常的条件，等待下条指令的启动。
EXCP	若有任何 FPSR[EXC]位或与其相应的 FPCR 异常使能位被置位，FSAVE 将生成该状态帧。该状态通常表示 FPU 当试图去完成先前浮点指令时遇到了异常

FSAVE 没有保存 FPU 可编程模式寄存器。它使用 FMOVEM 指令来保存包含浮点数和系统控制寄存器的 FPU。为了能完全保存，首先执行 FSAVE 指令来保存中间状态，然后执行 FMOVEM 来保存数据寄存器。FPCR 和 FPSR 被用来保存 FSAVE 状态帧的其中一部分。并且，在 FSAVE 指令结束时，FPCR 被清除，从而防止了处理浮点指令时的异常。

FPSR

不受影响。

FPCR

清零。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	0	0	目的有效地址					
										模式			寄存器		

指令域

源有效地址域——指明寻址方式、<ea>x，用于状态帧。只有以下模式被用到：

寻址方式	模式	寄存器
Dx	—	—
Ax	—	—
(Ax)	010	寄存器号:Ax
(Ax)+	—	—
-(Ax)	—	—
(d16,Ax)	101	寄存器号:Ax
(d8,Ax,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d16,PC)	111	010
(d8,PC,Xi)	—	—

HALT HALT

Halt the CPU

首先出现于 ISA_A

指令操作

If 特权状态

Then 挂起处理器内核

Else 特权违反异常

汇编格式

HALT

相关属性 尺寸不确定。

指令描述 处理器核心被同步（指当前所有指令和总线周期结束）后停止运行。处理器停顿状态，会被处理器状态输出引脚（PST=0xF）打出。若收到 GO 调试指令，则处理器恢复执行下条指令。请注意，该指令可同步管道。HALT 的操作码为 0x4AC8。 注意，通过调试模块所进行的 CSR[UHE]设置，可允许 HALT 执行在用户模式下。

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	0	0	1	0	0	0

INTOUCH Instruction Fetch Touch INTOUCH

首先出现于 ISA_A

指令操作

If 特权状态

Then **Instruction Fetch Touch at (Ay)**

Else 特权违反异常

汇编格式

INTOUCH(Ay)

相关属性 尺寸不

确定。

指令描述

生成地址（Ay）z 中的取指信息。若信息地址空间是支持高速缓冲的区域，则该指令可用来预取 16 字节的信息包到处理器的指令缓存中；若信息指令地址是不支持高速缓冲的区域，则指令有效地执行无操作过程。注意，该指令同步管道。

该 INTOUCH 指令可用于预取，同时，可通过对 CACR 的编程，来锁定在处理器的指令高速缓存内的具体的内存行。对于那些实时性能至关重要的系统来说，这个功能可能会很合乎人意。

条件码 不受影

响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	0	0	1	0	1	寄存器 Ax		

指令域

寄存器域——指明源地址寄存器 Ay。

MOVE fromSR

MOVE fromSR

Move from the Status Register

首先出现于 ISA_A

指令操作

If 特权状态

Then SR → 目的

Else 特权违反异常

汇编格式

MOVE.W SR,Dx

相关属性

尺寸 = 字的长度

指令描述

移动状态寄存器中数据到目的位置。目的地是字的长度。无效位读时为零。

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2 1 0
0	1	0	0	0	0	0	0	1	1	0	0	0	寄存器 Dx

指令域

寄存器域——指明目的数据寄存器 Dx。

MOVE fromUSP MOVE fromUSP

Move from User Stack Pointer

首先出现于 ISA_B

指令操作

If 特权状态

Then USP → 目的

Else 特权违反异常

汇编格式

MOVE.L USP,Ax

相关属性

尺寸 = 长字的长度

指令描述

移动用户堆栈指针的内容到特定地址寄存器。如果该指令试图在 ISA_A 类型的处理器或 MCF5407 上执行，则会产生非法指令异常。对于所有处理器而言，若 CACR[EUSP]被置位，则该指令都会正确执行。

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2 1 0
0	1	0	0	1	1	1	0	0	1	1	0	1	寄存器 Ax

指令域

寄存器域——指明目的地址寄存器 Ax。

MOVE to SR

Move to the Status Register

首先出现于 ISA_A

MOVE to SR

指令操作

If 特权状态

Then 源 → SR

Else 特权违反异常

汇编格式

MOVE.W <ea>y,SR

相关属性

尺寸 = 字的长度

指令描述

移动源操作地址中的数据到状态寄存器。该源操作数是一个字，且全部的状态寄存器有效位都受到影响。注意，该指令可同步管道。

条件码

X	N	Z	V	C
*	*	*	*	*

X	根据源操作数第 4 位的值设置
N	根据源操作数第 3 位的值设置
Z	根据源操作数第 2 位的值设置
V	根据源操作数第 1 位的值设置
C	根据源操作数第 0 位的值设置

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	源有效地址					
										模式			寄存器		

指令域

源有效地址域——指定源操作数位置，只使用列于下表的这些数据寻址方式。

寻址方式	模式	寄存器
Dy	010	寄存器号:Ay
Ay	—	—
(Ay)	—	—
(Ay)+	—	—
-(Ay)	—	—
(d16,Ay)	101	—
(d8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	111	100
(d16,PC)	—	—
(d8,PC,Xi)	—	—

MOVE to USP

MOVE to USP

Move to User Stack Pointer

首先出现于 ISA_B

指令操作

If 特权状态

Then 源 → USP

Else 特权违反异常

汇编格式

MOVE.L Ay,USP

相关属性

尺寸 = 长字长度。

指令描述

移动地址寄存器中的内容到用户堆栈指针。如果该指令试图在 ISA_A 类型的处理器或 MCF5407 上执行，则会产生非法指令异常。对于所有处理器而言，若 CACR[EUSP]被置位，则该指令都会正确执行。

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	0	寄存器 Ay		

指令域

寄存器域——指明源地址寄存器。

MOVEC Move Control Register MOVEC

首先出现于 ISA_A

指令操作

If 特权状态

Then Ry → Rc

Else 特权违反异常

汇编格式

MOVEC.L Ry,Rc

相关属性

尺寸 = 长字长度。

指令描述

移动通用寄存器中的内容到特殊控制寄存器。该传输始终是 32 位的，即使控制寄存器在执行时没有达到 32 位。注意，控制寄存器为只写，但片上调试模块可读取该控制寄存器。注意，该指令可同步管道（Pipeline）。

在所有 ColdFire 处理器上，并非每个的控制寄存器都已实现。对于某种特定芯片，请参阅其用户手册，以找出哪些寄存器已经实现。试图存取未定义的或未实现的控制寄存器空间，将产生不确定的结果。

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	1
A/D	寄存器 Ry			控制寄存器 Rc											

指令域

A/D 域——指明源寄存器 Ry 的类型。

— 0：数据寄存器

— 1：地址寄存器

寄存器 Ry 域——指明寄存器 Ry。

控制寄存器 Rc 域——指明表 8-3 中的设定值怎样影响控制寄存器。

表 **8-3 ColdFire CPU** 的空间分配

名称	CPU 空间分配	寄存器名
内存管理控制寄存器		
CACR	0x002	高速缓存控制寄存器
ASID	0x003	地址空间标识寄存器
ACR0	0x004	存取控制寄存器 0
ACR1	0x005	存取控制寄存器 1
ACR2	0x006	存取控制寄存器 2
ACR3	0x007	存取控制寄存器 3
MMUBAR	0x008	MMU 基地址寄存器
处理器杂项寄存器		
VBR	0x801	向量基寄存器
PC	0x80F	程序记数器
本地内存和模块控制寄存器		
ROMBAR0	0xC00	ROM 基地址寄存器 0
ROMBAR1	0xC01	ROM 基地址寄存器 1
RAMBAR0	0xC04	RAM 基地址寄存器 0
RAMBAR1	0xC05	RAM 基地址寄存器 1
MPCR	0xC0C	多处理器控制寄存器 [1]
EDRAMBAR	0xC0D	嵌入式 DRA 基地址寄存器 [1]
SECMBAR	0xC0E	二级模块基地址寄存器 [1]
MBAR	0xC0F	一级模块基地址寄存器
本地内存地址置换控制寄存器 [1]		
PCR1U0	0xD02	RAM0 的置换控制寄存器 1 的高 32 位
PCR1L0	0xD03	RAM0 的置换控制寄存器 1 的低 32 位
PCR2U0	0xD04	RAM0 的置换控制寄存器 2 的高 32 位
PCR2L0	0xD05	RAM0 的置换控制寄存器 2 的低 32 位
PCR1U0	0xD06	RAM0 的置换控制寄存器 3 的高 32 位
PCR3L0	0xD07	RAM0 的置换控制寄存器 3 的低 32 位
PCR1U1	0xD0A	RAM1 的置换控制寄存器 1 的高 32 位
PCR1L1	0xD0B	RAM1 的置换控制寄存器 1 的低 32 位
PCR2U1	0xD0C	RAM1 的置换控制寄存器 2 的高 32 位
PCR2L1	0xD0D	RAM1 的置换控制寄存器 2 的低 32 位
PCR1U1	0xD0E	RAM1 的置换控制寄存器 3 的高 32 位
PCR3L1	0xD0F	RAM1 的置换控制寄存器 3 的低 32 位

[1] 可选寄存器的域定义属特殊实现的情况。

RTE

Return from Exception

RTE

首先出现于 ISA_A

指令操作

If 特权状态

Then 2 + (SP) → SR; 4 + (SP) → PC; (SP) + 8→ SP;

根据格式调整堆栈

Else 特权违反异常

汇编格式

RTE

相关属性 尺寸不

确定。

指令描述 将存储于栈顶的异常堆栈帧中的处理器状态信息加载到处理器中。该指令检查格式

/偏移（Format/Offset Word）字段中的栈格式域，以决定有多少信息将会得到恢复。如果从内存加载时 SR[S]=0，则当从异常中恢复之时，该处理器会处于在用户模式；否则处理器仍在监控模式。注意，该指令可同步管道（Pipeline）。

条件码 根据状态寄存器值（该值是从堆栈恢复过来的）中条件码位来设置。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

STRLDSR Store/Load Status Register STRLDSR

首先出现于 ISA_C

指令操作

If 特权状态

Then SP－4 → SP; 用 0 填充过的 SR → (SP); 立即数 → SP

Else TRAP

汇编格式

STRLDSR #<data>

相关属性

尺寸 = 字的长度

指令描述

把状态寄存器的内容压入栈中，并将立即数据值加载到状态寄存器。该指令被设计用作，在多级中断请求中的中断服务例程的第一条指令。它允许刚刚发生的中断请求被储存在存储器中（用 SR[IML]域），然后通过把 0x7 加载到 SR[IML]（如果有需要）中来屏蔽该中断。若程序试图把立即数据的第 13 位清除（即把处理器置于用户模式），则特权违反异常将会产生。STRLDSR 的操作码是 0x40E746FC。

条件码

X	N	Z	V	C
*	*	*	*	*

X	根据立即操作数第 4 位的值设置
N	根据立即操作数第 3 位的值设置
Z	根据立即操作数第 2 位的值设置
V	根据立即操作数第 1 位的值设置
C	根据立即操作数第 0 位的值设置

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	1	0	0	1	1	1
0	1	0	0	0	1	1	0	1	1	1	1	1	1	0	0
立即数据															

STOP STOP

Load Status Register and Stop

首先出现于 ISA_A

指令操作

If 特权状态

Then 立即数据 → SR; STOP

Else 特权违反异常

汇编格式

STOP #<data>

相关属性 尺寸不确定。

指令描述 移动立即字操作数到状态寄存器（不管是用户模式还是特权模式）中，更新程序计数器以指向下条指示，并停止指令的获取和执行。踪迹、中断或复位异常会导致处理器恢复指令执行。如果停止指令开始执行时，指令追踪已被启用（T0 = 1 ），或者立即操作数的第 15 位是 1，则踪迹异常将有可能发生。如果立即数据中与 S 位对应的位被清零，则执行该指令将引发特权违反异常。如果当前中断请求被确定，其优先级高于由新中断的状态寄存器的值所代表的优先级，中断异常就会发生，否则当前中断请求将被忽略。外部复位总是激发复位异常处理进程。STOP 指令使得处理器处于低功耗状态。注意，该指令同步管道（Pipeline）。STOP 的操作码是 0x4E72，立即数据紧跟其后。

条件码

X	N	Z	V	C
*	*	*	*	*

X	根据立即操作数第 4 位的值设置
N	根据立即操作数第 3 位的值设置
Z	根据立即操作数第 2 位的值设置
V	根据立即操作数第 1 位的值设置
C	根据立即操作数第 0 位的值设置

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0
立即数据															

指令域

立即数据域——指明加载要到状态寄存器的数据。

WDEBEG Write Debug Control Register WDEBUG

首先出现于 ISA_A

指令操作

If 特权状态

Then 写在 Debug 中执行的控制寄存器命令

Else 特权违反异常

汇编格式

WDEBEG.L <ea>y

相关属性

尺寸 = 长字的长度

指令描述

从有效地址定义的内存位置获取两个连续的长字。这两个操作数都被 ColdFire 调试模块用于写一个调试控制寄存器（DRc）。注意，该指令可同步管道（Pipeline）。有效地址定义的内存位置必须按长字对齐，否则会有不确定的运行结果。调试命令必须组织在如下页所示的内存中。

条件码 不受影响。

指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	1	1	1	源有效地址					
										模式			寄存器		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

指令域

源有效地址域——指明用于操作的地址<ea>y ，只有下表所列寻址方式可用：

寻址方式	模式	寄存器
Dy	—	—
Ay	—	—
(Ay)	010	寄存器号:Ay
(Ay)+	—	—
-(Ay)	—	—
(d16,Ay)	101	寄存器号:Ay
(d8,Ay,Xi)	—	—

寻址方式	模式	寄存器
(xxx).W	—	—
(xxx).L	—	—
#<data>	—	—
(d16,PC)	—	—
(d8,PC,Xi)	—	—

内存中命令的组织：

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	1	1	0	0	1	0	0	DRc				
DATA[31:16]															
DATA[15: 0]															
不使用															

其中：

第一个字的位[15:4]将 WDREG 命令定义到调试模式。 第一个字的位[3:0]将特殊控制寄存器和 DRc 定义到写入模式。下表包含了 DRc 的定义。注意一些内核实现了调试寄存器的一个子集。在用户参考手册上能得到关于一个特殊设备或内核的更详细的信息。

DRc[4-0]	寄存器名称
0x00	配置/状态寄存器
0x01-0x0	保留未用
0x04	PC 中断点 ASID 控制
0x05	BDM 地址属性寄存器
0x06	地址属性触发寄存器
0x07	触发定义寄存器
0x08	程序计数器中断点寄存器
0x09	程序计数器中断点主控寄存器
0x0A-0x0B	保留未用
0x0C	地址中断点高位寄存器
0x0D	地址中断点低位寄存器
0x0E	数据中断点寄存器
0x0F	数据中断点主控寄存器

DRc[4-0]	寄存器名称
0x10-0x1	保留未用
0x14	PC 中断点 ASID 控制
0x15	保留未用
0x16	地址属性触发寄存器 1
0x17	扩展触发定义寄存器
0x18	程序计数器中断点寄存器 1
0x19	保留未用
0x1A	程序计数器中断点寄存器 2
0x1B	程序计数器中断点寄存器 3
0x1C	地址中断点高位寄存器 1
0x1D	地址中断点低位寄存器 1
0x1E	数据中断点寄存器 1
0x1F	数据中断点主控寄存器 1

被写入的数据数据[31:0]是 32 位操作数。

第四个字没有被用。

第 **9** 章 指令格式摘要

本章描述按数字顺序排列的 ColdFire 系列指令的二进制格式。这种二进制编码的整个指令的位元组都是事先确定的，位元组的十六进制值写在页面的右边缘，另外，破折号（–）用来表示它是个可变的数。

9.1 操作码映射

表 9-1 列出了 15-12 位的编码及其相关操作。

表 **9-1** 操作码映射表

15-12 位	十六进制值	操作
0000	0	位操作/立即寻址
0001	1	字节传送
0010	2	长字传送
0011	3	字传送
0100	4	混合操作
0101	5	ADDQ/SUBQ/Scc/TPF
0110	6	Bcc/BSR/BRA
0111	7	MOVEQ/MVS/MVZ
1000	8	OR/DIV
1001	9	SUB/SUBX
1010	A	MAC/EMAC/MOV3Q 指令
1011	B	CMP/EOR
1100	C	AND/MUL
1101	D	ADD/ADDX
1110	E	移位运算
1111	F	浮点/调试/Cache 指令

ORI **0x008–**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	寄存器 Dx		
立即数的高字															
立即数的低字															

BITREV **0x00C–**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	1	0	0	0	寄存器 Dx		
立即数的高字															
立即数的低字															

BTST 0x0——

位号是动态的，由寄存器来确定

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	数据寄存器 Dy			1	0	0	目的有效地址					
										模式			寄存器		
										模式			寄存器		

BCHG 0x0——

位号是动态的，由寄存器来确定

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	数据寄存器 Dy			1	0	1	目的有效地址					
										模式			寄存器		

BCLR 0x0——

位号是动态的，由寄存器来确定

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	数据寄存器 Dy			1	1	0	目的有效地址					
										模式			寄存器		

BEST 0x0——

位号是动态的，由寄存器来确定

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	数据寄存器 Dy			1	1	1	目的有效地址					
										模式			寄存器		

ANDI 0x028–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	1	0	0	0	0	寄存器 Dx		
立即数的高字															
立即数的低字															

BYTEREV 0x02C–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	1	1	0	0	0	寄存器 Dx		
立即数的高字															
立即数的低字															

SUBI 0x048–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	0	0	0	0	寄存器 Dx		
立即数的高字															
立即数的低字															

FF1 0x04C–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	1	1	0	0	0	寄存器 Dx		
立即数的高字															
立即数的低字															

ADDI 0x068–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0	1	0	0	0	0	寄存器 Dx		
立即数的高字															
立即数的低字															

BTST 0x08–00–

位号是动态的，由寄存器来确定

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	0	目的有效地址					
										模式			寄存器		
0	0	0	0	0	0	0	0	位号							

BCHG 0x08–00–

位号是动态的，由寄存器来确定

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	0	1	目的有效地址					
										模式			寄存器		
0	0	0	0	0	0	0	0	位号							

BCLR 0x08–00–

位号是动态的，由寄存器来确定

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	0	目的有效地址					
										模式			寄存器		
0	0	0	0	0	0	0	0	位号							

BSET 0x08–00–

位号是动态的，由寄存器来确定

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	0	0	1	1	目的有效地址					
										模式			寄存器		
0	0	0	0	0	0	0	0	位号							

EORI 0x0A8–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	0	1	0	1	0	0	0	0	寄存器 Dx		
立即数的高字															
立即数的低字															

CMPI 0x0C–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	0	0	尺寸		0	0	0	寄存器 Dx		
立即数的高字															
立即数的低字															

MOVE 0x–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	尺寸		目的有效地址						源有效地址					
				寄存器			模式			模式			寄存器		

STLDSR 0x40E7–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	1	0	0	1	1	1
0	1	0	0	0	1	1	0	1	1	1	1	1	1	0	0
<立即数>															

MOVEA 0x–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	尺寸		目的寄存器			0	0	1	源有效地址					
										模式			寄存器		

NEGX 0x408–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	0	0	0	0	寄存器 Dx		

MOVE from SR 0x40C–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	0	0	1	1	0	0	0	寄存器 Dx		

LEA 0x4–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	寄存器 Ax			1	1	1	源有效地址					
										模式			寄存器		

CLR 0x42–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	尺寸		目的有效地址					
										模式			寄存器		

MOV from CCR 0x42C–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	0	1	1	0	0	0	寄存器 Dx		

NEG 0x448–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	0	0	0	0	寄存器 Dx		

MOVE to CCR 0x44–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	0	0	1	1	源有效地址					
										模式			寄存器		

NOT 0x468–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	0	0	0	0	寄存器 Dx		

MOVE to SR 0x46–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	1	1	0	1	1	源有效地址					
										模式			寄存器		

SWAP 0x484–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	0	0	0	寄存器 Dx		

PEA 0x48–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	0	0	1	源有效地址					
										模式			寄存器		

EXT，EXTB 0x4–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	0	操作模式			0	0	0	寄存器 Dx		

MOVEM 0x4–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	dr	0	0	1	1	有效地址					
										模式			寄存器		
寄存器列表掩码															

TST 0x4A–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	尺寸		目的有效地址					
										模式			寄存器		

TAS 0x4A–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	目的有效地址					
										模式			寄存器		

HALT 0x4AC8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	0	0	1	0	0	0

PULSE 0x4ACC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	0	0	1	1	0	0

ILLEGAL 0x4AFC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	0	1	0	1	1	1	1	1	1	0	0

MULU.L 0x4C— –000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	源有效地址					
										模式			寄存器		
0	寄存器 Dx			0	0	0	0	0	0	0	0	0	0	0	0

MULS.L 0x4C— –800

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	0	源有效地址					
										模式			寄存器		
0	寄存器 Dx			1	0	0	0	0	0	0	0	0	0	0	0

DIVU.L 0x4C— –00–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	源有效地址					
										模式			寄存器		
0	寄存器 Dx			0	0	0	0	0	0	0	0	0	寄存器 Dx		

REMU.L 0x4C— –00–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	源有效地址					
										模式			寄存器		
0	寄存器 Dx			0	0	0	0	0	0	0	0	0	寄存器 Dw		

DIVS.L 0x4C— –80–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	源有效地址					
										模式			寄存器		
0	寄存器 Dx			1	0	0	0	0	0	0	0	0	寄存器 Dx		

REMS.L 0x4C— –80–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	0	1	源有效地址					
										模式			寄存器		
0	寄存器 Dx			1	0	0	0	0	0	0	0	0	寄存器 Dw		

SATS 0x4C8–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	0	0	1	0	0	0	0	寄存器 Dx		

TRAP 0x4E4–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	0	向量			

LINK 0x4E5–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	0	寄存器 Ay		
字偏移															

UNLK 0x4E5–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	0	1	1	寄存器 Ax		

MOVE to USP 0x4E6–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	0	寄存器 Ay		

MOVE from USP 0x4E6–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	0	1	寄存器 Ax		

NOP 0x4E71

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	0	1

STOP 0x4E72

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	0
立即数															

RTE 0x4E73

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	0	1	1

RTS 0x4E75

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	0	1	0	1

MOVEC 0x4E7B

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	0	1	1	1	1	0	1	1
D/A	寄存器 Ry			控制寄存器 Rc											

JSR 0x4E—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	0	源有效地址					
										模式			寄存器		

JMP 0x4E—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	1	1	1	0	1	1	源有效地址					
										模式			寄存器		

ADDQ 0x5—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	数据			0	1	0	目的有效地址					
										模式			寄存器		

Scc 0x5–C–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	条件				1	1	0	0	0	寄存器 Dx		

SUBQ 0x5—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	数据			1	1	0	目的有效地址					
										模式			寄存器		

TPF 0x51F–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	1	0	0	0	1	1	1	1	1	1	操作模式		
可选立即字															
可选立即字															

BRA 0x60–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	0	8 位偏移							
16 位偏移（当 8 位偏移等于 0x00 时）															
32 位偏移（当 8 位偏移等于 0xFF 时）															

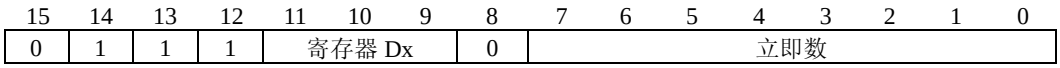
BSR 0x61–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	0	0	0	1	8 位偏移							
16 位偏移（当 8 位偏移等于 0x00 时）															
32 位偏移（当 8 位偏移等于 0xFF 时）															

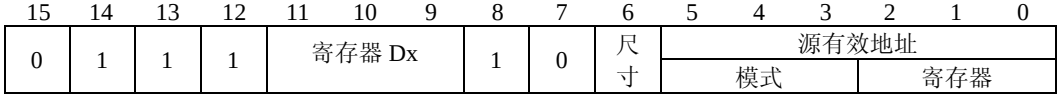
Bcc 0x6——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	条件				8 位偏移							
16 位偏移（当 8 位偏移等于 0x00 时）															
32 位偏移（当 8 位偏移等于 0xFF 时）															

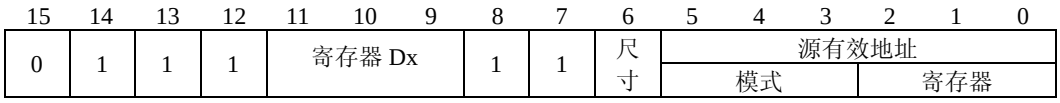
MOVEQ 0x7——



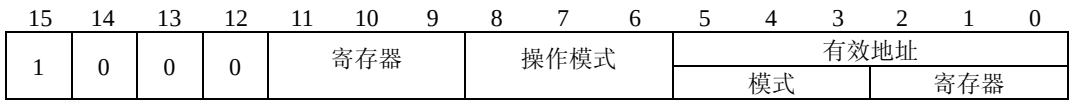
MVS 0x7——



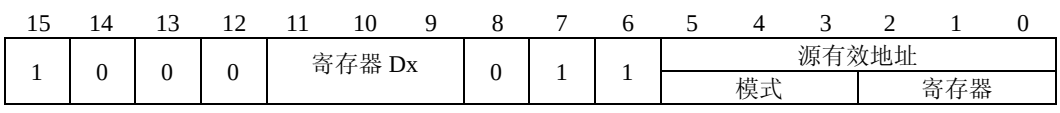
MVZ 0x7——



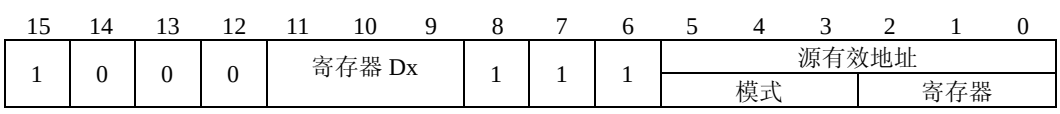
OR 0x8——



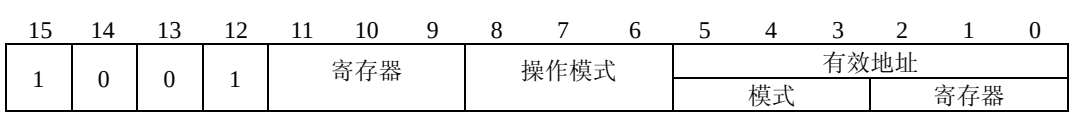
DIVU.W 0x8——



DIVS.W 0x8——



SUB 0x9——



SUBX 0x9——

15	14	13	12	11 10 9	8	7	6	5	4	3	2 1 0
1	0	0	1	寄存器 Dx	1	1	0	0	0	0	寄存器 Dy

SUBA 0x9——

15	14	13	12	11 10 9	8	7	6	5 4 3	2 1 0
1	0	0	1	目的寄存器 Ax	1	1	1	源有效地址	
								模式	寄存器

MAC（MAC） 0xA——

15	14	13	12	11 10 9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx	0	0	Rx	0	0	寄存器 Ry			
—	—	—	—	sz 比例因子	0	U/Lx	U/Ly	—	0	—	—	0	0

MAC（EMAC） 0xA——

15	14	13	12	11 10 9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx	0	ACC lsb	Rx msb	0	0	寄存器 Ry			
—	—	—	—	sz 比例因子	0	U/Lx	U/Ly	—	ACCmsb	—	—	0	0

MAC with load (MAC) 0xA——

15	14	13	12	11 10 9	8	7	6	5 4	3 2	1 0
1	0	1	0	寄存器 Rw	0	1	Rw	源有效地址		
								模式	寄存器	
寄存器 Rx				sz 比例因子	0	U/Lx	U/Ly	掩码	0	寄存器 Ry

MAC with load (EMAC) 0xA——

15	14	13	12	11 10 9	8	7	6	5 4	3 2	1 0
1	0	1	0	寄存器 Rw	0	ACC Lsb	Rw	源有效地址		
								模式	寄存器	
寄存器 Rx				sz 比例因子	0	U/Lx	U/Ly	掩码	ACCmsb	寄存器 Ry

MAAAC（EMAC_B） 0xA——

15	14	13	12	11 10 9	8	7	6	5	4	3 2	1	0
1	0	1	0	寄存器 Rx	0	ACClsb	Rxmsb	0	0	寄存器 Ry		
—	—	—	—	sz 比例	0	U/Lx	U/Ly	—	ACCmsb	ACCw	0	1

					因子								

MSAAC (EMAC_B) 0xA——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx			0	ACClsb	Rxmsb	0	0	寄存器 Ry			
—	—	—	—	sz	比例因子		0	U/Lx	U/Ly	—	ACC msb	ACCw		1	1

MSAC (MAC） 0xA——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx			0	0	Rx	0	0	寄存器 Ry			
—	—	—	—	sz	比例因子		1	U/Lx	U/Ly	—	0	—	—	0	0

MSAC (EMAC) 0xA——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx			0	ACC lsb	Rx msb	0	0	寄存器 Ry			
—	—	—	—	sz	比例因子		1	U/Lx	U/Ly	—	ACC msb	—	—	0	0

MSAC with load (MAC) 0xA——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rw			0	1	Rw	源有效地址					
										模式			寄存器		
寄存器 Rx				sz	比例因子		1	U/Lx	U/Ly	掩码	0	寄存器 Ry			

MSAC with load (EMAC) 0xA——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rw			0	ACC lsb	Rw	源有效地址					
										模式			寄存器		
寄存器 Rx				sz	比例因子		1	U/Lx	U/Ly	掩码	ACC msb	寄存器 Ry			

MSAAC (EMAC_B) 0xA——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx			0	ACC lsb	Rx Msb	0	0	寄存器 Ry			
—	—	—	—	sz	比例		1	U/Lx	U/Ly	—	ACC	ACCw		0	1

					因子					msb			

MSAAC(EMAC_B)(WRONG VMA) 0xA——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	寄存器 Rx			0	ACC lsb	Rx msb	0	0	寄存器 Ry			
—	—	—	—	sz	比例因子		1	U/Lx	U/Ly	—	ACCmsb	ACCw		1	1

MOVE to ACC (MAC) 0xA1——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	1	0	0	源有效地址					
										模式			寄存器		

MOVE to ACC (EMAC) 0xA——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	ACC		1	0	0	源有效地址					
										模式			寄存器		

MOVE ACC to ACC (EMAC) 0xA–1–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	ACCx		1	0	0	0	1	0	0	ACCy	

MOVE from ACC (MAC) 0xA18——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	0	1	1	0	0	0	寄存器 Rx			

MOVE from ACC (EMAC) 0xA–8–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	ACC		1	1	0	0	0	寄存器 Rx			

MOVCLR (EMAC) 0xA–C–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	ACC		1	1	1	0	0	寄存器 Rx			

MOVE from MACSR 0xA98–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	1	1	0	0	0	寄存器 Rx			

MOVE to MACSR 0xA9—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	1	0	0	源有效地址					
										模式			寄存器		

MOVE from MACSR to CCR 0xA9C0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	0	1	1	1	0	0	0	0	0	0

MOVE to ACCext01 (EMAC) 0xAB—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	1	0	0	源有效地址					
										模式			寄存器		

MOVE from ACCext01(EMAC) 0xAB8–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	0	1	1	1	0	0	0	寄存器 Rx			

MOVE to Mask 0xAD—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	1	0	0	源有效地址					
										模式			寄存器		

MOVE from Mask 0xAD8–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	0	1	1	0	0	0	寄存器 Rx			

MOVE to ACCext23 (EMAC) 0xAF—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	1	1	0	0	源有效地址					
										模式			寄存器		

MOVE from ACCext23(EMAC) 0xAF8–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	1	1	1	1	1	0	0	0	寄存器 Rx			

MOV3Q 0xA——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	立即数			1	0	1	目的有效地址					
										模式			寄存器		

CMP 0xB——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	寄存器 Dx			操作模式			源有效地址					
										模式			寄存器		

CMPA 0xB——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	控制寄存器 Ax			操作模式			源有效地址					
										模式			寄存器		

EOR 0xB——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	1	寄存器 Dy			1	1	0	目的有效地址					
										模式			寄存器		

AND 0xC——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	数据寄存器			操作模式			有效地址					
										模式			寄存器		

MULU.W 0xC——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	寄存器 Dx			0	1	1	源有效地址					
										模式			寄存器		

MULS.W 0xC——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	寄存器 Dx			1	1	1	源有效地址					
										模式			寄存器		

ADD 0xD——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	寄存器			操作模式			有效地址					
										模式			寄存器		

ADDX 0xD–8–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	寄存器 Dx			1	1	0	0	0	0	寄存器 Dy		

ADDA 0xD——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	目的寄存器 Ax			1	1	1	源有效地址					
										模式			寄存器		

ASL,ASR 0xE——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	计数或寄存器 Dx			dr	1	0	i/r	0	0	寄存器 Dx		

LSL,LSR 0xE——

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	0	计数或寄存器 Dy			dr	1	0	i/r	0	1	寄存器 Dx		

FMOVE 0xF2—

存储器操作和寄存器到寄存器操作(<ea>y,FPx;FPy,FPx)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx				操作模式 （0000000，1000000 或 1000100）					

FMOVE 0xF2— ——0

寄存器到存储器操作(FPy,<ea>)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

1	1	1	1	0	0	1	0	0	0	目的有效地址						
										模式			寄存器			
0	1	1	目的格式			源寄存器 FPy			0	0	0	0	0	0	0	

FINT 0xF2— —1

寄存器到存储器操作(FPy,<ea>)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	目的有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx			0	0	0	0	0	0	1

FINTRZ 0xF2— —3

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx			0	0	0	0	0	1	1

FSQRT 0xF2—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx			操作模式 （0000100，1000001 或 1000100）						

FABS 0xF2—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx			操作模式 （0011000，1011000 或 1011100）						

FNEG 0xF2—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx			操作模式 （0011010，1011010 或 1011110）						

FDIV 0xF2—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx				操作模式 （0100000，1100000 或 1100100）					

FADD 0xF2—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx				操作模式 （0100010，1100010 或 1100110）					

FMUL 0xF2—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx				操作模式 （0100011，1100011 或 1100111）					

FSUB 0xF2—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx				操作模式 （0101000，1101000 或 1101100）					

FCMP 0xF2— ——8

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		
0	R/M	0	源指定			目的寄存器 FPx			0	1	1	1	0	0	0

FTST 0xF2— ——A

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		

0	R/M	0	源指定	目的寄存器 FPx	0	1	1	1	0	1	0

FBcc 0xF2—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	1	尺寸	条件谓词					
16 位偏移或 32 位偏移的最高有效字															
32 位偏移的最低有效最字（必要时）															

FMOVE to FPIAR 0xF2—8400

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

FMOVE to FPSR 0xF2—8800

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0

FMOVE to FPCR 0xF2—9000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	源有效地址					
										模式			寄存器		
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0

FMOVE from FPIAR 0xF2—A400

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	目的有效地址					
										模式			寄存器		
1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0

FMOVE from FPSR 0xF2—A800

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

1	1	1	1	0	0	1	0	0	0	目的有效地址						
										模式			寄存器			
1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	

FMOVE from FPCR 0xF2—B000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	目的有效地址					
										模式			寄存器		
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

FMOVEM 0xF2— –0—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	0	0	有效地址					
										模式			寄存器		
1	1	dr	1	0	0	0	0	寄存器列表							

FNOP 0xF280 0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	0	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

FSAVE 0xF3—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	0	0	目的有效地址					
										模式			寄存器		

FRESTORE 0xF3—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	0	1	1	0	1	源有效地址					
										模式			寄存器		

INTOUCH 0xF42–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	0	1	0	0	0	0	1	0	1	寄存器 Ax		

CPUSHL 0xF4–

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

1	1	1	1	0	1	0	0	缓存	1	0	1	寄存器 Ax

WDDATA 0xFB—

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	1	尺寸		源有效地址					
										模式			寄存器		

WDEBUG 0xFB—0003

位号是动态的，由寄存器来确定

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	0	1	1	1	1	源有效地址					
										模式			寄存器		
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1

第 10 章 PST/DDATA 编码

本章从指令的角度详细描述 ColdFire 芯片的处理器状态（Processor Status PST）的形成和 Debug 模块的调试数据（Debug Data DDATA）的输出。通常 PST/DDATA 对于指令的输出被定义为以下格式：

PST=0x1,{PST={0x8,0x9,0xB},DDATA = 操作数}

{...}里所定义的内容是由 CSR 中的设置信息所定义的可选的操作数信息。CSR 对基于参考类型（读、写或两者都）的操作数的显示提供了可能性。PST 的值{0x8,0x9,0xB}用于定义长度和有无跟随在 DDATA 输出数据后的有效数据{1、2 或 4 字节}。此外，对于某些流程改变的跳转指令，CSR[BTB]给目标指令地址的显示提供了可能性，用 PST 的值{0x9,0xA,0xB}来区分 DDATA 输出数据{2,3 或 4 字节}。

对于 V2 和 V3 版本的芯片，PST 和 DDATA 是两个不同的端口，实时跟踪信息可以在这两个端口上并发显示。从 V4 版本开始，PST 和 DDATA 的输出被合并为一个单一的端口。实时跟踪信息表现为一个没有任何队列限制的 4 位数据值序列；也就是说，处理器的状态（PST）值和操作数（DDATA）可能会表现在 PSTDDATA[7:0]的任一个半位元组中。上半位（PSTDDATA[7:4]是最重要的，它优先产生输出值。值得注意的是联合的 PSTDDATA 的输出，以与独立的 PST 和 DDATA 输出一致的方式，仍然显示了处理器的状态和调试数据。至于更深入的信息，请查阅芯片的调试部分或内核用户手册。

V5 版本的处理器提供了增强的 PST 和 DDATA 功能，特别是在 PST 跨越多个机器周期和可选的操作数地址跟踪能力的压缩形式上。要获取更多信息，请查阅设备的调试部分或内核参考手册。值得注意的是：不是所有的指令都可以被所有的 ColdFire 处理器执行。详情见第 3 章，“描述指令集定义具体细节的指令集概要。”

10.1 用户指令集

表 10-1 显示了用户模式指令的 PST/DDATA 详细说明。Rn 是代表任何一个{Dn, An}寄存器。“y”下标表示源操作数，“x”下标表示目的操作数。对于给定的指令，可选操作数据只有那些访问内存的有效地址才显示。“DD”的命名参照了 DDATA 的输出。

表 **10-1** 用户模式下指令 **PST/DDATA** 细节

指 令	操作数格式	PST/DDATA
add.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = 源操作数}
add.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = 源},{PST = 0xB, DD = 目的}
adda.l	<ea>y,Ax	PST = 0x1, {PST = 0xB, DD = 源操作数}
addi.l	#<data>,Dx	PST = 0x1
addq.l	#<data>,<ea>x	PST = 0x1, {PST = 0xB, DD = 源},{PST = 0xB, DD = 目的}
addx.l	Dy,Dx	PST = 0x1
and.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = 源操作数}
and.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = 源},{PST = 0xB, DD = 目的}

andi.l	#<data>,Dx	PST = 0x1
asl.l	{Dy,#<data>},Dx	PST = 0x1
asr.l	{Dy,#<data>},Dx	PST = 0x1
bcc.{b,w,l}		if taken, then PST = 0x5, else PST = 0x1
bchg.{b,l}	#<data>,<ea>x	PST = 0x1, {PST = 0x8, DD = 源},{PST = 0x8, DD = 目的}
bchg.{b,l}	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = 源},{PST = 0x8, DD = 目的}
bclr.{b,l}	#<data>,<ea>x	PST = 0x1, {PST = 0x8, DD = 源},{PST = 0x8, DD = 目的}
bclr.{b,l}	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = 源},{PST = 0x8, DD = 目的}
bitrev	Dx	PST = 0x1
bra.{b,w,l}		PST = 0x5
bset.{b,l}	#<data>,<ea>x	PST = 0x1, {PST = 0x8, DD = 源},{PST = 0x8, DD = 目的}
bset.{b,l}	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = 源},{PST = 0x8, DD = 目的}
bsr.{b,w,l}		PST = 0x5, {PST = 0xB, DD = 目的操作数}
btst.{b,l}	#<data>,<ea>x	PST = 0x1, {PST = 0x8, DD = 源操作数}
btst.{b,l}	Dy,<ea>x	PST = 0x1, {PST = 0x8, DD = 源操作数}
byterev	Dx	PST = 0x1
clr.b	<ea>x	PST = 0x1, {PST = 0x8, DD = 目的操作数}
clr.l	<ea>x	PST = 0x1, {PST = 0xB, DD = 目的操作数}
clr.w	<ea>x	PST = 0x1, {PST = 0x9, DD = 目的操作数}
cmp.b	<ea>y,Dx	PST = 0x1, {0x8, 源操作数}
cmp.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = 源操作数}
cmp.w	<ea>y,Dx	PST = 0x1, {0x9, 源操作数}
cmpa.l	<ea>y,Ax	PST = 0x1, {PST = 0xB, DD = 源操作数}
cmpa.w	<ea>y,Ax	PST = 0x1, {0x9, 源操作数}
cmpi.b	#<data>,Dx	PST = 0x1
cmpi.l	#<data>,Dx	PST = 0x1
cmpi.w	#<data>,Dx	PST = 0x1
divs.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = 源操作数}
divs.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = 源操作数}
divu.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = 源操作数}
divu.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = 源操作数}
eor.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = 源},{PST = 0xB, DD = 目的}
eori.l	#<data>,Dx	PST = 0x1
ext.l	Dx	PST = 0x1
ext.w	Dx	PST = 0x1
extb.l	Dx	PST = 0x1
ff1	Dx	PST = 0x1
illegal		PST = 0x1[1]
jmp	<ea>y	PST = 0x5, {PST = {0x9,0xA,0xB}, DD = 目的地址}[2]
jsr	<ea>y	PST = 0x5, {PST = {0x9,0xA,0xB}, DD = 目的地址},{PST = 0xB , DD = 目的操作数}[2]
lea.l	<ea>y,Ax	PST = 0x1
link.w	Ay,#<displacement>	PST = 0x1, {PST = 0xB, DD = 目的操作数}
lsl.l	{Dy,#<data>},Dx	PST = 0x1
lsr.l	{Dy,#<data>},Dx	PST = 0x1
mov3q.l	#<data>,<ea>x	PST = 0x1, {0xB, 目的操作数}

move.b	<ea>y,<ea>x	PST = 0x1, {PST = 0x8, DD = 源},{PST = 0x8, DD = 目的}
move.l	<ea>y,<ea>x	PST = 0x1, {PST = 0xB, DD = 源},{PST = 0xB, DD = 目的}
move.w	<ea>y,<ea>x	PST = 0x1, {PST = 0x9, DD = 源},{PST = 0x9, DD = 目的}
move.w	CCR,Dx	PST = 0x1
move.w	{Dy,#<data>},CCR	PST = 0x1
movea.l	<ea>y,Ax	PST = 0x1, {PST = 0xB, DD = 源}
movea.w	<ea>y,Ax	PST = 0x1, {PST = 0x9, DD = 源}
movem.l	#list,<ea>x	PST = 0x1, {PST = 0xB, DD = 目的},... [3]
movem.l	<ea>y,#list	PST = 0x1, {PST = 0xB, DD = 源},... [3]
moveq.l	#<data>,Dx	PST = 0x1
muls.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = 源操作数}
muls.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = 源操作数}
mulu.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = 源操作数}
mulu.w	<ea>y,Dx	PST = 0x1, {PST = 0x9, DD = 源操作数}
mvs.b	<ea>y,Dx	PST = 0x1, {0x8, 源操作数}
mvs.w	<ea>y,Dx	PST = 0x1, {0x9, 源操作数}
mvz.b	<ea>y,Dx	PST = 0x1, {0x8, 源操作数}
mvz.w	<ea>y,Dx	PST = 0x1, {0x9, 源操作数}
neg.l	Dx	PST = 0x1
negx.l	Dx	PST = 0x1
nop		PST = 0x1
not.l	Dx	PST = 0x1
or.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = 源操作数}
or.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = 源},{PST = 0xB, DD = 目的}
ori.l	#<data>,Dx	PST = 0x1
pea.l	<ea>y	PST = 0x1, {PST = 0xB, DD = 目的操作数}
pulse		PST = 0x4
rems.l	<ea>y,Dw:Dx	PST = 0x1, {PST = 0xB, DD = 源操作数}
remu.l	<ea>y,Dw:Dx	PST = 0x1, {PST = 0xB, DD = 源操作数}
rts		PST = 0x1, PST = 0x5, {{0x9,0xA,0xB}, 目的地址}, PST = 0x1,{PST = 0xB, DD = 源操作数}, PST = 0x5, {PST = {0x9,0xA,0xB}, DD = 目的地址}
sats.l	Dx	PST = 0x1
scc.b	Dx	PST = 0x1
sub.l	<ea>y,Dx	PST = 0x1, {PST = 0xB, DD = 源操作数}
sub.l	Dy,<ea>x	PST = 0x1, {PST = 0xB, DD = 源},{PST = 0xB, DD = 目的}
suba.l	<ea>y,Ax	PST = 0x1, {PST = 0xB, DD = 源操作数}
subi.l	#<data>,Dx	PST = 0x1
subq.l	#<data>,<ea>x	PST = 0x1, {PST = 0xB, DD = 源}, {PST = 0xB, DD =目的}
subx.l	Dy,Dx	PST = 0x1
swap.w	Dx	PST = 0x1
tas.b	<ea>x	PST = 0x1, {0x8, 源}, {0x8, 目的}
tpf		PST = 0x1
tpf.l	#<data>	PST = 0x1
tpf.w	#<data>	PST = 0x1
trap	#<data>	PST = 0x1 [1]

tst.b	<ea>x	PST = 0x1, {PST = 0x8, DD = 源操作数}
tst.l	<ea>y	PST = 0x1, {PST = 0xB, DD = 源操作数}
tst.w	<ea>y	PST = 0x1, {PST = 0x9, DD = 源操作数}
unlk	Ax	PST = 0x1, {PST = 0xB, DD = 目的操作数}
wddata.b	<ea>y	PST = 0x4, {PST = 0x8, DD = 源操作数}
wddata.l	<ea>y	PST = 0x4, {PST = 0xB, DD = 源操作数}
wddata.w	<ea>y	PST = 0x4, {PST = 0x9, DD = 源操作数}

[1] 在正常的异常处理中，PST 的输出被置为 0xc，标志了异常处理状态。除了将异常写如入栈操作外，异常管理的矢量读和目标地址也被显示出来。

异常处理 PST = 0xC,{PST = 0xB,DD = destination},// stack frame

{PST = 0xB,DD = destination},// stack frame

{PST = 0xB,DD = source},// vector read

PST = 0x5,{PST = [0x9AB],DD = target}// handler PC

PST/DDATA 的异常重置规范显示如下：

异常处理 PST = 0xC,PST = 0x5,{PST = [0x9AB],DD = target}// handler PC

自从取指令访问后，地址 0 和 4 的初始信息从来没有被抓获，也没有显示。所有类型的异常处理中，如果 PST 输出不被要求为可选标记值中的任一个或不为跳转指标（0x5），PST 将一直被置为 0xc 值。

[2] 对于 JMP 和 JSR 指令，可选的目标指令地址只为那些有效定义变量寻址方式的地址字段显示。这包括以下<ea> x 值：(An)，(d16,An)，(d8,An,Xi)，(d8,PC,Xi)。

[3] 对于转移多指令（MOVEM），如果操作数地址达到 0-模-16 边界，以及有 4 个或更多的寄存器要被转移，处理器则自动生成线性传输。对于这些线性传输，不论 CSR 是何值，操作数是永远不会被捕获，也不会被显示。在顺序内存访问操作期间，这些自动生成的线性脉冲传输则被提供了最大限度的性能。

表 10-2 显示了用户模式下多累加器指令 PST 详细说明。

表 **10-2** 用户模式下多累加器指令 **PST/DDATA** 值

指 令	操作数格式	PST/DDATA
m*ac.l[1]	Ry,Rx,ACCx{,ACCw}	PST = 0x1
m*ac.l1	Ry,Rx,<ea>y,Rw,ACCx	PST = 0x1, {PST = 0xB, DD = 源操作数}
m*ac.w1	Ry,Rx,ACCx{,ACCw}	PST = 0x1
m*ac.w1	Ry,Rx,<ea>y,Rw,ACCx	PST = 0x1, {PST = 0xB, DD = 源操作数}
move.l	{Ry,#<data>},ACCext01	PST = 0x1
move.l	{Ry,#<data>},ACCext23	PST = 0x1
move.l	{Ry,#<data>},ACCx	PST = 0x1
move.l	{Ry,#<data>},MACSR	PST = 0x1
move.l	{Ry,#<data>},MASK	PST = 0x1
move.l	ACCext01,Rx	PST = 0x1
move.l	ACCext23,Rx	PST = 0x1
move.l	ACCy,ACCx	PST = 0x1
move.l	ACCy,Rx	PST = 0x1
move.l	MACSR,CCR	PST = 0x1
move.l	MACSR,Rx	PST = 0x1
move.l	MASK,Rx	PST = 0x1

表 10-3 显示了用户模式下浮点型指令 PST/DDATA 详细说明。注意<ea>y 包含 FPy、Dy、Ay 和<mem>y 寻址方式。可选操作捕获和显示，只应用于<mem>y 寻址方

式。说明还需注意的是，对于某一特定指令，不管是否明确四舍五入的精度，PST/DDATA 值是相同的。

表 **10-3** 用户模式下浮点型指令 **PST/DDATA** 值

指 令	操作数格式	PST/DDATA
fabs.sz	<ea>y,FPx	PST = 0x1, [89B], 源}
fadd.sz	<ea>y,FPx	PST = 0x1, [89B], 源}
fbcc.{w,l}	<label>	if taken, then PST = 5, else PST = 0x1
fcmp.sz	<ea>y,FPx	PST = 0x1, [89B], 源}
fdiv.sz	<ea>y,FPx	PST = 0x1, [89B], 源}
fint.sz	<ea>y,FPx	PST = 0x1, [89B], 源}
fintrz.sz	<ea>y,FPx	PST = 0x1, [89B], 源}
fmove.sz	<ea>y,FPx	PST = 0x1, [89B], 源}
fmove.sz	FPy,<ea>x	PST = 0x1, [89B], 目的}
fmove.l	<ea>y,FP*R1	PST = 0x1, B, 源}
fmove.l	FP*R,<ea>x1	PST = 0x1, B, 目的}
fmovem	<ea>y,#list	PST = 0x1
fmovem	#list,<ea>x	PST = 0x1
fmul.sz	<ea>y,FPx	PST = 0x1, [89B], 源}
fneg.sz	<ea>y,FPx	PST = 0x1, [89B], 源}
fnop		PST = 0x1
fsqrt.sz	<ea>y,FPx	PST = 0x1, [89B], 源}
fsub.sz	<ea>y,FPx	PST = 0x1, [89B], 源}
ftst.sz	<ea>y	PST = 0x1, [89B], 源}

[1]FP*R 符号指浮点数控制寄存器: FPCR、FPSR 和 FPIAR。

依赖于任何外存储器大小的操作用 f<op>表示。Fmt 字段和数字标记定义如表 10-4。

表 **10-4** 数字标记与 **FPU** 操作数格式指定

格式指定	标记数字
.b	8
.w	9
.l	B
.s	B
.d	不被获取

10.2 特权指令集

特权指令集已经完全有权使用用户模式的指令，再加上下面所示的操作码。这些操作码的 PST/DDATA 的详细说明见表 10-5。

表 **10-5** 特权模式下指令 **PST/DDATA** 细节

指 令	操作数格式	PST/DDATA
cpushl	dc,(Ax) ic,(Ax) bc,(Ax)	PST = 0x1
frestore	<ea>y	PST = 0x1
fsave	<ea>x	PST = 0x1
halt		PST = 0x1,PST = 0xF
intouch	(Ay)	PST = 0x1
move.l	Ay,USP	PST = 0x1
move.l	USP,Ax	PST = 0x1
move.w	SR,Dx	PST = 0x1
move.w	{Dy,#<data>},SR	PST = 0x1,{PST = 0x3}
movec.l	Ry,Rc	PST = 0x1,{8,ASID}
rte		PST = 0x7,{PST = 0xB,DD = 源操作数},{PST=3} {PST = 0xB,DD =源操作数},{DD}, PST = 0x5,{[PST = 0x9AB],DD = 目的地址}
stldsr	#<data>	PST = 0x1,{PST = 0xB,DD = 目的}
stop	#<data>	PST = 0x1, PST = 0xE
wdebug.l	<ea>y	PST = 0x1,{PST = 0XB, DD = 目的}

指令 move-to-SR 和 RTE 包括一个可选的 PST=0x3 值，标志着进入用户模式的入口。另外，如果一个 RTE 指令的执行结果将处理器返回到竞争模式，多重循环状态会被标志为 0xD。

类似于异常处理模式，停止状态（PST=0xE）和暂停状态（PST=0xF）将显示在整个时间中 ColdFire 处理器在给定模式下的状态。

第 11 章 异常处理

本章主要讲述 ColdFire 系列芯片的异常处理。

11.1 概述

ColdFire 系列处理器的异常处理在性能上改进了很多，在以下方面它不同于早先的 M68000 系列处理器：

⑴ 有一个简单的异常向量表。
⑵ 具有使用向量基址寄存器简化了的定位能力。
⑶ 具有简单的堆栈帧格式。
⑷ 使用了简单的有自动矫正功能的堆栈指针(这适用于对 ISA_A 的实现)。

从 V4 版本的内核开始，各个版本便对可选的虚拟内存管理单元提供了支持。对于那些具有 MMU 的芯片，异常处理被做了一些改进。不同于早先的 ColdFire 系列处理器，它还与译码故障(如 TLB 故障)和存取故障的指令重启模式息息相关。这扩充了先前的 ColdFire 系列芯片的错误故障向量和异常堆栈结构。

早先的 ColdFire 系列处理器(V2 和 V3 版)使用了一个指令重启的异常模式，但它需要额外的软件支持才能从特定的存取错误中恢复过来。

异常处理就是从检测到错误条件到第一个处理指令被调用的过程。它包括以下四个主要步骤：

⑴ 处理器执行一个状态寄存器（SR）的内部复制，然后通过设置 SR[S]进入管理方式模式，通过清除 SR[T]进入禁止跟踪模式。中断异常的发生也会清除 SR[M]和设置中断优先级字、SR[I]以符合当前的中断请求级别。

⑵ 处理器确定异常向量号。除了中断以外的所有错误，处理器都要考虑异常类型。而对于中断，处理器执行一个中断确认（IACK）总线周期，从外围设备获得向量号。中断确认周期被映射到一个特殊的确认地址空间并与地址中的中断级别编码对应。

⑶ 处理器在系统堆栈中创建一个异常堆栈结构来保存当前内容。实现 ISA_A 的处理器支持单一堆栈指针 A7，所以没有区分管理员和普通用户堆栈指针的概念，于是，异常堆栈结构被创建在当前系统栈顶部的以 0-4 为模的地址中。实现其他 ISA 的处理器支持两个堆栈指针，异常堆栈被创建在当前系统堆栈顶部的以 0-4 为模的管理堆栈指针（SSP）指向的地方。所有的 ColdFire 处理器适用简化的固定长度的堆栈设计，如图 11-1，适用于所有的异常。另外，处理器内核支持 MMU 使用相同的固定长度的堆栈设计，并使用异常状态（FS）编码来支持 MMU。某些异常类型中，异常堆栈结构中的程序计数器包含了错误指令的地址；而另一些异常类型中，它包含了下一条即将执行的指令的地址。如果异常是由 FPU 指令引起的，且异常是预编译指令时，程序计数器包括下一条浮点指令的地址，或者当异常是已处理的指令时，程序计数器包括下条要执行的指令的地址。

⑷ 处理器通过取得向量基址寄存器中的异常表中的一个值来取得异常处理的第一条指令地址。表中的索引计算方法是 4*向量号。索引值一旦产生，就可以通过向量表的内容确定需要的第一条指令的地址。初始化取得的第一条操作码后，异常处理结束，管理者将能继续进行正常的指令处理。

向量基址寄存器的描述见章节 1.5.3，“向量基址寄存器（VBR）”异常向量表的基址保存在存储器中。通过异常向量的偏移量与基址寄存器的值相加来访问向量表。VBR[19-0]没有被执行并且假定为零，强制使向量表以 1Mb 为边界排成列。

ColdFire 处理器支持 1024 字节向量表，如表 11-1 所示。表中包含 256 个异常向量，前 64 个向量是由 Motorola 定义的，其他是用户定义的中断向量。

表 **11-1** 异常向量分配

向量号	向量偏移(十六进制)	栈程序计数器 [1]	分配
0	000	—	初始化堆栈指针 (SSP 为主要的双向堆栈指针)
1	004	—	初始化程序计数器
2	008	Fault	访问出错
3	00C	Fault	地址错误
4	010	Fault	非法指令
5[2]	014	Fault	0 作除数
6–7	018–01C	—	保留
8	020	Fault	特例
9	024	Next	描述
10	028	Fault	无效的 a 行操作码
11	02C	Fault	无效的 f 行操作码
12[3]	030	Next	非 PC 断点编译中断
13[3]	034	Next	PC 断点编译中断
14	038	Fault	尺度误差
15	03C	Next	为初始化的中断
16–23	040–05C	—	保留
24	060	Next	假中断
25–31[4]	064–07C	Next	Level 1 – 7 外向量中断
32–47	080–0BC	Next	Trap #0 – 15 指令
48[5]	0C0	Fault tt	浮点无序接通条件
49[5]	0C4	NextFP or Fault	浮点不精确结果
50[5]	0C8	NextFP	浮点除以零
51[5]	0CC	NextFP or Fault	浮点溢出
52[5]	0D0	NextFP or Fault	浮点操作数出错
53[5]	0D4	NextFP or Fault	浮点溢出
54[5]	0D8	NextFP or Fault	浮点输入不是数字
55[5]	0DC	NextFP or Fault	浮点输入非规格化数字
56–60	0E0–0F0	—	保留
616	0F4	Fault	不支持指令
62–63	0F8–0FC	—	保留
64–255	100–3FC	Next	用户定义指令

[1]Fault 指的是错误指令的 PC。Next 指的是错误指令后的指令的 PC。NextFP 指的是下条浮点指令的 PC。

[2] 如果除法单元不使用（5202，5204，5206）,向量 5 被保留。

[3] 在 V2 和 V3 中，所有的调试中断使用向量 12，向量 13 被保留。

[4] 对外向量中断的支持依赖于中断控制器的执行。另外的详细描述参考特殊的芯片参考手册。

[5] 如果 FPU 不是能使用，向量 48-55 被保留。

[6] 有些芯片不支持异常处理；参考表 11-3。

ColdFire 处理器禁止在异常操作的第一条指令中发生中断。允许任何操作有效的禁止中断，如果必要可以提高 SR 中的中断级别。

11.1.1 管理员/用户堆栈指针（A7 和 OTHER_A7）

一些 ColdFire 系列支持两个独立的栈指针（A7）寄存器：管理员堆栈指针（SSP）和用户栈指针（USP）。这一支持在运行模式间提供了必要的独立。注意只有 SSP 是用于创建异常堆栈框架的。

这两个程序可见的 32 位寄存器的硬件执行不是唯一鉴别 SSP 和 USP 的标志。硬件用一个 32 位寄存器作为当前活动的 A7 而另一个作为 OTHER_A7。因此，寄存器内容是处理器执行模式的函数：

```
if SR[S] = 1
then
        A7=Supervisor Stack Pointer
        Other_A7=User Stack Pointer
        else
                A7=User Stack Pointer
                Other_A7=Supervisor Stack Pointer
```

BDM 编程模式支持直接对 A7 和 OTHER_A7 读写。外设的任务是基于设置 SR[S] 并决定将 A7 和 OTHER_A7 映射到两个程序可见的堆栈（SSP 和 USP）中。这个功能通过设定双栈指针使能位，CACR[DSPE]才能有效。如果该位被清零，只有 A7（用于早先的 ColdFire 版本）可用。DSPE 在复位时为零。

如果 DSPE 被设置了，相应的堆栈指针寄存器（SSP 或者 USP）将被作为处理器运行模式的函数来访问。为了支持双栈指针，以下两个特殊的用于装载/存储 USP 的 MC680x0 指令作为 ISA_B 的一部分被加载到 ColdFire 指令集体系中：

```
mov.l Ay,USP # move to USP: opcode = 0x4E6(0xxx)
mov.l USP,Ax # move from USP: opcode = 0x4E6(1xxx)
```

地址寄存器号编码到操作码的低三位。

11.1.2 异常栈框架定义

异常栈框架中的第一个长字，如图 11-1，用于保存 16 位格式/向量字（F/V）和 16 位状态寄存器。第二个用于保存 32 位程序计数器的地址。

	31	28	27 26	25 18	17 16	15 0
A7→	格式		FS[3－2]	VEC	FS[1－0]	状态寄存器
+ 0x04	程序计数器[31:0]					

图 **11-1** 异常堆栈结构

表 11-2 描述 F/V 域。FS 编码用来支持 MMU。

表 **11-2** 格式/向量字

<table>
<tr><th>位</th><th>域</th><th>描述</th></tr>
<tr><td>31-28</td><td>FORMAT</td><td>格式域。用标明 2 个长字结构格式的处理器来写入{4,5,6,7}中一个值。当异常发生时，FORMAT 记录了任何未对准的长字堆栈指针。
<table>
<tr><th>发生异常时的 A7,位[1:0]</th><th>处理第一条指令时的 A7</th><th>FORMAT</th></tr>
<tr><td>00</td><td>初始化A7—8</td><td>0100</td></tr>
<tr><td>01</td><td>初始化 A7—9</td><td>0101</td></tr>
<tr><td>10</td><td>初始化 A7—10</td><td>0110</td></tr>
<tr><td>11</td><td>初始化 A7—11</td><td>0111</td></tr>
</table></td></tr>
<tr><td>27-26</td><td>FS[3-2]</td><td>出错状态为中断调试服务定义的存取与地址访问错误。
0000 没有访问或存取错误，也没有中断调试服务定义。
0001 保留未用
0010 在对致命错误和其他存取错误的调试服务的中断(如果是 MMU,高于 V4)
0011 保留未用
0100 发生在指令存取的错误(例如保护性的异常)
0101 指令获取时的 TLB 错误(如果是 MMU,高于 V4)
0110 基于扩展字的指令获取时的 TLB 错误(如果是 MMU,高于 V4)
0111 当执行在模拟器模式下的 IFP 存取错误(如果是 MMU,高于 V4)
1000 数据写入时的错误
1001 试图写入到保护区的错误
1010 数据写入时的 TLB 异常(如果是 MMU,高于 V4)
1011 保留未用
1100 读数据时的错误
1101 试图对保护区进行读改写的错误(如果是 MMU,高于 V4)
1110 读数据时的 TLB 异常(如果是 MMU,高于 V4)
1111 当执行在模拟器模式下的 OEP 存取错误(如果是 MMU,高于 V4)</td></tr>
<tr><td>25-18</td><td>VEC</td><td>向量号。定义异常类型。它由处理器对内部错误计算而得，也由外围设备的中断来提供。参照表 11-1</td></tr>
<tr><td>17-16</td><td>FS[1—0]</td><td>参照 27-26 位</td></tr>
</table>

虽然上面提到的是关于调试服务中的 I/O 中断，但是这也适用于其他类型的错误。如果在调试服

务中发生了存取错误，当在指令存取时 FS 被设置成 0111，在数据存取时 FS 被设置成 1111。这只适用于 MMU 的存取错误。如果不是 MMU 的存取错误，FS 被设置成 0010。

11.1.3 处理器异常

表 11-3 描述了 ColdFire 内核异常。注意当处理其他错误时，要注意的是：如果当 ColdFire 处理器正在处理异常，同时又出现其他错误，它将作为一个致命错误来处理并立即停止执行。若想退出停止状态，那必须复位。

11.1.4 浮点算法异常

本节描述浮点算法异常。表 11-3 按优先级顺序列出了这些异常：

表 **11-3** 异常优先级

优先级	异常
1	分支/开始无序(BSUN)
2	输入非数字(INAN)
3	输入规格化数字(IDE)
4	操作数错误(OPERR)
5	溢出 (OVFL)
6	下溢(UNFL)
7	除数为零(DZ)
8	不精确 (INEX)

当遇到下个浮点算法指令时，往往会发生浮点异常（这就是预指令异常）。当存储一个浮点数到存储器或者到一个整数寄存器时会立即发生异常(这是已处理指令异常)。

注意因为结果是全面的，FMOVE 被认为是个算法指令。只有 FMOVE 的操作目的地不是浮点寄存器时（否则称为 FMOVE 溢出）能产生已处理指令异常。已处理指令异常从不写目的地址。已处理指令异常发生后，继续处理下一条指令。

当浮点指令结果设置了 FPSR[EXC]位和相应的 FPCR[ENABLE]位被设置后，浮点运算异常变成了未定的。用户写 FPSR 和 FPCR 会导致 FPSR[EXC]的异常位连同相应的 FPCR 的异常使能的重新设置，而 FPU 处于异常未定状态。在开始执行下一条算法指令时会发生相应的预指令异常。

执行多个指令会产生多重异常。当多重异常发生并激活多个异常类时，优先级最高的异常将被执行。这决定于异常管理者对多重异常的检查。下面是可能发生的多重异常：

- 操作数错误(OPERR) 和不精确结果(INEX)
- 溢出(OVFL)和不精确结果(INEX)
- 下溢(UNFL)和不精确结果(INEX)
- 除数为零(DZ)和不精确结果(INEX)
- 输入规格化数字(IDE)和不精确结果(INEX)
- 输入非数字(INAN) 和输入规格化数字(IDE)

一般来说，所有的异常都是相似的。如果异常条件存在，而异常不可捕捉，没有发生异常，错误的结果将被写到目的地址（除 BSUN 异常没有目的地址），然后正常执行。

如果有效的异常发生，与以上讲到的相同的默认结果将被写到预指令异常中，但没有结果写到已处理指令异常中。

我们期望异常处理将 FSAVE 作为第一条浮点指令。并且会清 FPCR，FPCR 用来阻止异常处理期间发生异常。因为目的地址是写在浮点寄存器目的地址中的，原始的浮点目的寄存器值对 FSAVE 状态的管理是可用的。指令地址引起的异常在 FPIAR 中是可用的。当异常处理完成时，需要清空 FSAVE 状态中适当的 FPSR 异常位，再执行 FRESTORE。如果状态中的异常状态位没有被清空，相同的异常会再次发生。 相比较而言，执行 FSAVE 异常处理可以简单的清除适当的 FPSR 异常位，任意改变 FPCR，并从异常中返回。注意异常从来不会随 FMOVE 发生在状态和控制寄存器中，也不会随 FMOVEM 发生在浮点数据寄存器中。

在异常管理完成时，RTE 指令必须被执行返回到正常的指令流中。

11.1.5 分支开始无序(BSUN)

当无序条件存在时，IEEE 无意识的条件性测试与 FBcc 指令的联合会产生 BSUN。在条件指令重启后任何一个浮点异常首先都是被预指令处理的。在执行条件指令之前条件谓词会被估计和检查。当条件谓词是 IEEE 无意识的条件分支并且 FPCC[NAN]被置位时就会发生 BSUN 异常发生。一旦条件被检测到，FPSR[BSUN]就被置位。表 11-4 显示了异常有效或是无效时的结果。

表 **11-4 BSUN** 异常有效/无效结果

条 件	BSUN	描 述
异常无效	0	浮点条件估计为 IEEE 条件谓词。不发生任何异常。
异常有效	1	处理器发生浮点预指令异常。 BSUN 异常是唯一的，因为在条件谓词被估计之前异常已经发生。如果用户 BSUN 异常管理在返回时没能够在异常指令之后更新 PC 到指令中，异常将会再次被执行。下面行为能够防止异常被再次执行： • 清空 FPSR[NAN] • 使 FPCR[BSUN]失效 • 在栈中的条件指令旁增加存储 PC。用在可能出现错误的 地方。注意要得到精确的 PC 增量 得到条件指令的大

11.1.6 输入非数字(INAN)

INAN 异常是处理用户定义的机制，它不是 IEEE 的数据类型。输入的操作数不是数字时，FPSR[INAN]被置位。有了 INAN 异常，用户可以忽略操作数为非数字的错误。由于 FMOVEM、FMOVE、FPCR 和 FSAVE 指令不会改变状态位，所以不会产生异常。所以这些指令在操作非数字类型时是有效的。见表 11-5。

表 **11-5 INAN** 异常有效/无效结果

条 件	INAN	描 述
异常无效	0	如果目的数据格式是单或双精度，会产生一个带尾数的非数字和零标志转存在目的地址。如果目的数据格式是 B，W，或者 L，则写一个常数到目的地址。

异常有效	1	除了异常发生在 FMOVE OUT 模式下写到目的地址中的结果和异常无效是一样的，否则目的地址不受影响。

11.1.7 输入规格化数字(IDE)

输入标准位 FPCR[IDE]，为规格化操作数提供软件支持。当 IDE 异常无效时，操作数被视为零，FPSR[INEX]被置位，操作成功。如果 IDE 异常有效且操作数是规格化的，则发生 IDE 异常，但 FPSR[INEX]不置位以允许适当的置位处理。见表 11-6。

注意 FPU 从来不产生规格化数字。如果必要，软件会在下溢异常处理中创建。

表 **11-6 IDE** 异常有效/无效结果

条 件	IDE	描 述
异常无效	0	任何规格化操作数被看作为 0，FPSR[INEX]置位，操作成功。
异常有效	1	除了异常发生在 FMOVE OUT 模式下写到目的地址中的结果和异常无效是一样的，否则目的地址不受影响。FPSR[INEX]不置位以允许管理可作适当的设置。

11.1.8 操作数错误(OPERR)

操作数错误异常涵盖各种操作引发的问题，包括过少的或是过多的特殊异常条件的错误。基本上，当对操作数的操作没有准确的解释时，操作数错误通常会发生。表 11-7 列出了操作数错误。异常发生时，FPSR[OPERR]置位。

表 **11-7** 可能的操作数错误

指令	引起操作数错误的条件
FADD	[(+∞) + (-∞)] 或 [(-∞) + (+∞)]
FDIV	(0 ÷ 0) 或 (∞ ÷ ∞)
FMOVE OUT(B、W 或者 L)	整数溢出，源数据是非数字或者±∞
FMUL	一个操作数是 0 另一个是±∞
FSQRT	源是小于 0 的或者是-∞
FSUB	[(+∞) - (+∞)] 或者 [(-∞) - (-∞)]

表 11-8 描述了异常有效和无效时发生异常的结果。

表 **11-8 OPERR** 异常有效/无效结果

条 件	OPERR	描 述
异常无效	0	当目的地址是个浮点数据寄存器时，结果是双精度非数字带尾数，标志设为 0（正）。 格式为 S 或者 D 的 FMOVE OUT 模式指令不会发生 OPERR 异常，对于格式为 B，W，或者 L 的情况只有在转变为整数溢出或者源数据为无穷大或非数字时才有可能发生。 整数溢出和源数据为无穷大时，符合特殊目的地址大小(B， W 或者 L)的最大正数或负整数倍存储。 非数字源情况下，常数被写到目的地址中。
异常有效	1	除了异常发生在 FMOVE OUT 模式下写到目的地址中的结果和

		异常无效是一样的，否则目的地址不受影响。如果需要，用户 OPERR 处理可以写入默认结果。

11.1.9 溢出(OVFL)

当中间结果大于或等于已选择舍入精度的最大指数值时，目的地址是浮点寄存器或存储器中的算法操作能检测到溢出异常。溢出只在目的地址为 S 或者 D 精度格式时发生，对于其他格式管理类似操作数错误。在任何操作结束时，都可能发生溢出，检查中间结果以防下溢，然后在存储到目的地址之前检查溢出。如果溢出发生 FPSR[OVFL,INEX]置位。

即使中间结果足够小可以被看成双精度数字，如果中间结果的数量级超出了所选舍入精度格式的范围也会发生溢出。见表 11-9。

表 **11-9 OVFL** 异常有效/无效结果

条 件	OVFL	描 述
异常无效	0	目的地址中的值是基于 FPCR[MODE]的舍入模式定义。 RN 无穷大，带中间结果标志 RZ L 大数量级数字，带中间结果标志。 RM 正溢出，最大正规格化数字。负溢出，- RPF 正溢出，＋。负溢出，最大负规格化数字。
异常有效	1	除了异常发生在 FMOVE OUT 模式下写到目的地址中的结果和异常无效是一样的，否则目的地址不受影响。如果需要，用户 OVFL 管理可以写默认结果。

11.1.10 下溢(UNFL)

下溢异常发生是由于算法指令的中间结果太小而不能在浮点寄存器或存储器中用已选择的舍入精度表示一个规格化数字。也就是说发生在中间结果指数小于或等于已选舍入精度的最小指数值的情况下。下溢只在目的地址格式为单精度或双精度时发生。若目的地址是一个字节，字或长字，则将下溢处理为 0，从而不会引起下溢或操作数错误。在任何操作结束时，都可能发生溢出，检查中间结果以防下溢，然后在存储到目的地址之前检查溢出。如果溢出发生则 FPSR[UNFL]被置位。如果下溢无效，FPSR[INEX]被置位。

即使中间结果足够大，可以被看成双精度数字，若它的数量级太小超出了所选舍入精度格式的范围还是会发生下溢。见表 11-10 描述了异常有效或无效的结果。

表 **11-10 UNFL** 异常有效或无效结果

条 件	UNFL	描 述
异常无效	0	存储结果定义如下。如果 UNFL 异常无效，UNFL 异常也会设置 FPSR[INEX] RN 0，带中间结果标志 RZ 0，带中间结果标志 RM 正下溢，+0。负溢出，最小负规格化数字

		RP 正下溢，最小正规格化数字。负溢出，-0
异常有效	1	除了异常发生在 FMOVE OUT 模式下，写入到目的地址的结果与异常无效的情况是相同的，否则目的地址不受影响。如果需要，用户 UNFL 处理可以写默认结果。UNFL 异常不置位 FPSR[INEX]，如果 UNFL 异常有效，异常处理可以基于产生的结果情况来置位 FPSR[INEX]。

11.1.11 除数为零(DZ)

若除法指令使用 0 作为除数会产生除数为零的异常。当检测到此异常时，FPSR[DZ]被置位。表 11-11 显示了异常有效或无效结果。

表 **11-11 DZ** 异常有效或无效结果

条 件	DZ	描 述
异常无效	0	目标浮点数据寄存器写入无穷大标志，这种标志是唯一的标志或者是输入操作数的标志。
异常有效	1	目标浮点数据寄存器的写入与异常无效的情况相同。

11.1.12 不精确结果(INEX)

当浮点类型中间结果的无穷精度尾数的关键位能在舍入精度或在目的格式中准确的表示，INEX 异常就有可能发生。如果异常发生，FPSR[INEX]置位，无穷精度结果参考表 11-12。

表 **11-12** 不精确舍入模式值

模式	结果
RN	最接近无穷精度的中间值作为结果。 如果两个最接近的值几乎相等，lsb 为 0（偶数）的是结果。通常被称为就近取偶(round-to-nearest-even).
RZ	结果是一些中间值，这些值在级数上最接近并且不大于无穷精度。 有时被称为截断模式(chop-mode), 因为结果将清空当前舍入点的位。
RM	结果为最接近并且不大于无穷精度的中间值（可能为-x）。
RP	结果为最接近并且不小于无穷精度的中间值（可能为+x）。

FPSR[INEX]在以下任何的条件下也会被置位：

- 如果输入操作数是规格化数字并且 IDE 异常失效。
- 溢出结果
- 下溢异常失效的下溢结果

表 11-13 显示了当异常有效或失效时的结果

表 **11-13 INEX** 异常有效或无效结果

条 件	INEX	描 述
异常无效	0	结果是舍入的并且被写入目的地址
异常有效	1	除了异常发生在 FMOVE OUT 模式下，写入到目的地址的结果与异常无效的情况是相同的，否则目的地址不受影响。如果需要，用户 INEX 管理可以写默认结果。

11.1.13 MMU 转变成异常处理模式

当 MMU 模块出现在 ColdFire 芯片中，所有的存储器相关部件都需要对可恢复错误(recoverable faults)提供支持。本节将详细描述 ColdFire 系列在加入 MMU 后的异常处理模式的变化。

ColdFire 指令重启机制保证了错误的指令从执行开始重启，也就是说异常发生时没有内部状态信息被保存，异常处理结束时没有被恢复。通过转变控制到给定的位置作为 RTE 指令的一部分，处理器使程序执行复原，给出异常堆栈结构中的 PC 地址定义。

如果指令发生后继错误，指令重启恢复模式需要程序可见寄存器改变执行到未完成模式。

对于 V4 及以上核心，大多数指令中操作数执行管道（OEP）结构自然支持这个概念；只有在 OEP 末级也就是当错误的集合完成时，程序可见寄存器才会更新。任何类型的异常发生，未定的寄存器更新都会被放弃。

大多数指令已经支持精度出错和指令重启。而一些复杂的指令不支持。思考以下存储器到存储器的移动：

mov.l (Ay)+,(Ax)+ #从源地址复制四个字节数到目的地址 这条指令用了一周期去读源操作数（Ay）加 1 并将数据写到（Ax）。源和目的地址指针都被更新作为执行的一部分。表 11-14 列出了执行过程中的操作。

表 **11-14 OEP EX** 周期操作

EX 周期	操　作
1	从存储器@(Ay)读源操作数，更新 Ay，新 Ay = 旧 Ay + 4
2	写操作数到目的存储器@(Ax)，更新 Ax，新 Ax =旧 Ax + 4，更新 CCR

第二个周期中报告错误被检测到并且写到目的存储器中。此时，第一个周期的操作执行完成，所以如果目的写操作发生任何类型的访问错误，Ay 被更新。访问错误管理执行完成后，错误指令重启，处理器操作是错误的因为源地址寄存器错误（已经存在的增量）的值。

为了恢复所有指令的设计模式的初始状态，V4 及以上核心添加了必要的硬件来支持全寄存器恢复。硬件允许程序可见寄存器存储多周期指令的原始状态，所以可支持指令重启机制。存储器到存储器的移动和重载代表复杂指令需要的特殊恢复支持。

附录 A　S 记录输出格式

用于输出模块的 S 记录格式，是用于在可打印格式下编码程序或者是数据文件，这些程序或数据文件是用于在计算机系统之间传输。传输过程可有效通过显示器来监测，而且 S 记录可很容易地被编辑。

A.1 S 记录内容

直观上，S 记录实质是由定义该记录的类型、长度、内存地址、代码/数据和校验和等几个域组成。每个字节的二进制数据被编码为具有双元特征的十六进制数，第一个元代表高 4 位，而第二个元则代表低 4 位。图 A-1 表示一个 S 记录包含的 5 个域，表 A-1 列出每个 S 记录的组成细节。

类 型	记录长度	地 址	代码/数据	校验和

图 **A-1** 组成 **S** 记录的五个部分

表 **A-1 S** 记录的组成部分

部 分	可打印字符	内　容
类型	2	S 记录类型—S0，S1 等等。
记录长度	2	记录中的字符数，不包括类型和记录长度的字符。
地址	4、6 或 8	第 2、3 或 4 字节表示被写到存储器的数据域的地址。
代码/数据	0–2n	从执行代码的第 0 到第 n 个字节，存储器可装载数据或叙述性信息。某些程序限制字节数在 28 之内（S 记录中的 56 个可打印字符）。
校验和	2	由纪录长度，地址和代码/数据和的补码组成的低位字节。这个字节的实际意义最少。

下载 S 记录时，每条必须以 CR 结束。另外，每条 S 记录都有一个初始域区别于其他数据，就像某些分时系统产生的队列数字。记录长度（字节数）和校验和保证了传输的正确性。

A.2 S 记录类型

共有八种类型的 S 记录提供编码、传输和译码功能。不同的 Motorola 记录传输控制程序（例如上载、下载等等），交叉汇编程序、连接器和其他文件创建或者调试程序只利用 S 记录来保存程序。更多的支持特殊 S 记录的信息，请参考用户程序手册。

每条 S 记录格式模型可能包括以下类型的 S 记录：

S0　每个 S 记录块的记录头。代码/数据域可能包括叙述性信息以识别下面的 S 记录块。记录头可以用来指明模型名，版本号，修订号和叙述性信息。地址域通常为 0。

S1 该记录包括代码/数据和它们所在的两个字节的地址。

S2 该记录包括代码/数据和它们所在的三个字节的地址。

S3 该记录包括代码/数据和它们所在的四个字节的地址。

S5 该记录包括 S1、S2 和 S3 在特殊快情况下的记录数。这个数字显示在地址域。没有代码/数据域。

S7 S3 记录块的终止记录。地址域可能选择性的包括 4 字节的已传递的指令地址，没有代码/数据域。

S8 S2 记录块的终止记录。地址域可能选择性的包括 3 字节的已传递的指令地址，没有代码/数据域。

S9 S1 记录块的终止记录。地址域可能选择性的包括 2 字节的已传递的指令地址。如果地址没有详细说明，则使用第一个目标模块的入口。没有代码/数据域。

每个 S 记录块使用唯一的终止记录。S7 和 S8 记录只在控制传给第 3、4 个字节时使用，另外 S9 用来结束。通常，没有头记录，可能会出现多重的头记录。

A.3 S 记录创建

优化器、调试器或交叉汇编程序和连接器产生 S 记录格式程序。当下载或上载从主系统到微处理器系统的 S 记录格式文件时，这些程序就起作用了。

典型的 S 记录格式模型打印如下：

```
S00600004844521B
S1130000285F245F2212226A000424290008237C2A
S11300100002000800082629001853812341001813
S113002041E900084E42234300182342000824A952
S107003000144ED492
S9030000FC
```

该模型包含 1 个 S0 记录，4 个 S1 记录和 1 个 S9 记录。以下字符组成 S 记录格式模型。

S0 记录：

S0	S0 型记录,表示这是记录头
06	十六进制 06(十进制 6), 表示接着六个字符(或 ASCII 字节)
0000	一个 4-字符, 2-字节地址域;本例中为 0
48	ASCII H
44	ASCII D
52	ASCII R
1B	校验和

第一个 S1 记录：

S1	S1 型记录, 表示加载/检验代码/数据记录到 2-字节地址
13	十六进制 13 (十进制 19), 表示接着的 19 个字符，代表 19 个字节的二

进制数。

0000　一个 4 字符，2 字节地址域（十六进制地址 0000）表示后面要加的数据。

接下来第一个 S1 记录的 16 个字符是实际程序代码/数据的 ASCII 码字节。在这个汇编语言例子中，程序十六进制编码继续写在 S1 记录的代码/数据域中。

操作码	指　令
285F	MOVE.L (A7) +, A4
245F	MOVE.L (A7) +, A2
2212	MOVE.L (A2), D1
226A0004	MOVE.L 4(A2), A1
24290008	MOVE.L FUNCTION(A1), D2
237C	MOVE.L #FORCEFUNC, FUNCTION(A1)

余下的代码延续了 S1 记录的代码/数据域并且存储在存储器中的位置是 0010。

2A　第一个 S1 记录的校验和

第二和第三个 S1 记录包括十六进制 13（十进制 19）个字符并且分别以校验和为 13 和 52 结束。第四个 S1 记录包括 07 个字符，校验和为 92。

S9 记录：

S9　S 记录类型 S9,表示一个终止记录

03　十六进制 03, 表示后续的 3 个字符(3 个字节)

0000　地址域，0

FC　S9 记录的校验和

每个 S 记录的可打印字符以十六进制（本例中为 ASCII）编码，并且代表了传输的二进制位。图 A-2 描述了 S1 记录的发送过程。表 A-2 列出了 S 记录的 ASCII 码。

类 型				记录长度				地　址								代 码 / 数 据										校验和			
S		1		1		3		0		0		0		0		2		8		5		F		**		2		A	
5	3	3 1		3 1		3 3		3 0		3 0		3 0		3 0		3 2		3 8		3 5		4 6		* *		3	2	4	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	*	0	0	0	0
01	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	*	0	0	1	0
01	1	1	0	1	0	1	1	1	0	1	0	1	0	1	0	1	1	1	0	1	0	0	1	*	1	1	0	0
	1	1	1	1	1	1	1	1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	0	*	1	0	0	1

图 **A-2. S1** 记录的发送

表 **A-2 ASCII** 码

低位字节	高位字节							
	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	‘	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	“	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL