第 2 章　CPU 结构与指令集

TMS320C6000 系列 DSP 芯片均基于 VelociTI 结构，采用高性能的甚长指令字(VLIW)，使得该系列 DSP 适合于多通道和多任务的应用。本章首先讲述 DSP 的中央处理器(CPU)结构和指令集，然后讲解流水线和中断。

2.1　CPU 的结构

TMS320C67xx CPU 的结构框图如图 2-1 所示，其中 CPU 部分包括：

- 程序取指、指令分配和译码机构：包括程序取指单元、指令分配单元和指令译码单元，程序取指单元由程序总线与片内程序存储器相连。
- 程序执行机构：包括两个对称数据通道(A 和B)、两个对称的通用寄存器组、两组对称的功能单元(每组 4 个)、控制寄存器、控制逻辑及中断逻辑等。每侧数据通道由数据总线与片内数据存储器相连。
- 芯片测试、仿真端口及其控制逻辑。

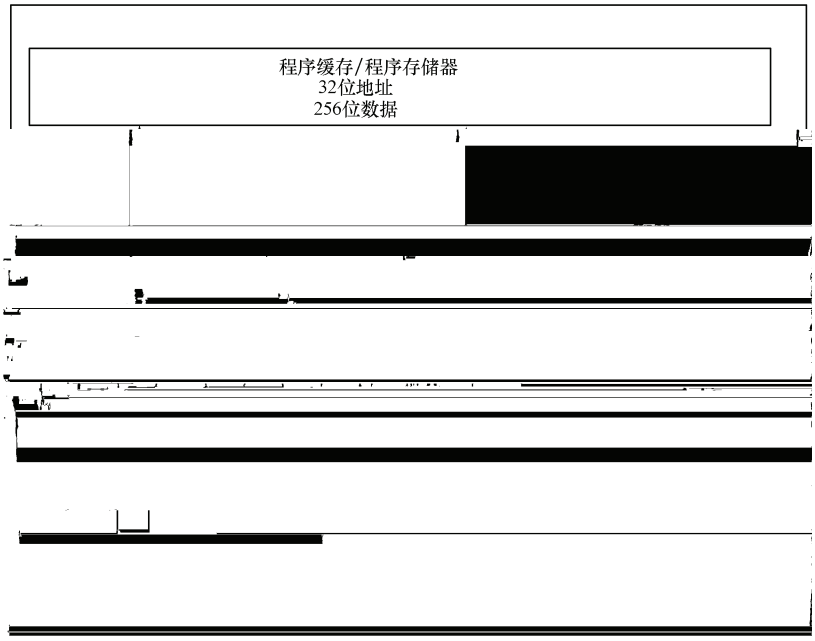


图 2-1　TMS320C67xx 的结构框图

C67xx 系列 CPU 采用哈佛结构，其程序总线与数据总线分开，可并行读取与执行指令。片内程序存储器保存指令代码，程序总线连接程序存储器与 CPU。C67xx 系列芯片的程序总线宽度为 256 位，每次取 8 条指令，称为一个取指包。

执行时，每条指令占用一个功能单元。取指、分配和译码单元都具备单周期读取并传递 8 条 32 位指令的能力。在两个数据通道(A 和 B)的功能单元内执行这些指令。控制寄存器控制操

作方式。从程序存储器读取一个取指包时起，VLIW 处理流程开始。一个取指包可能分成几个执行包，详细处理过程见 2.4 节。

C67xx 系列 DSP 片内的程序总线与数据总线分开，程序存储器与数据存储器分开，但片外的存储器及总线都不分，二者是统一的。全部存储空间(包括程序存储器和数据存储器，片内和片外)以字节为单位统一编址。无论是从片外读取指令或与片外交换数据，都要通过 EDMA 与 EMIF，相关操作将在后续章节介绍。在片内，仅在取指令时用到程序总线。

TI 公司的数据手册常把在指令执行过程中使用的物理资源统称为数据通道，其中包括执行指令的 8 个功能单元、通用寄存器组及片内数据存储器交换信息所使用的数据总线等。

C67xx 系列 CPU 有两个类似的可进行数据处理的数据通道 A 和 B，每个通道有 4 个功能单元(.L、.S、.M 和.D)以及 1 组包括 16 个 32 位寄存器的通用寄存器组。功能单元执行指令指定的操作。除读取(Load)、存储(Store)类指令及程序转移类指令外，其他所有算术逻辑运算指令均以通用寄存器为源操作数和目的操作数，使程序能够高速运行。读取和存储类指令用于在通用寄存器与片内数据存储器之间交换数据，此时两个数据寻址单元(.D1 和.D2)负责产生数据存储器地址。每个数据通道的 4 个功能单元有单独的数据总线连接到 CPU 另一侧的寄存器上(见图 2-2)，使得两组寄存器组可以交换数据。

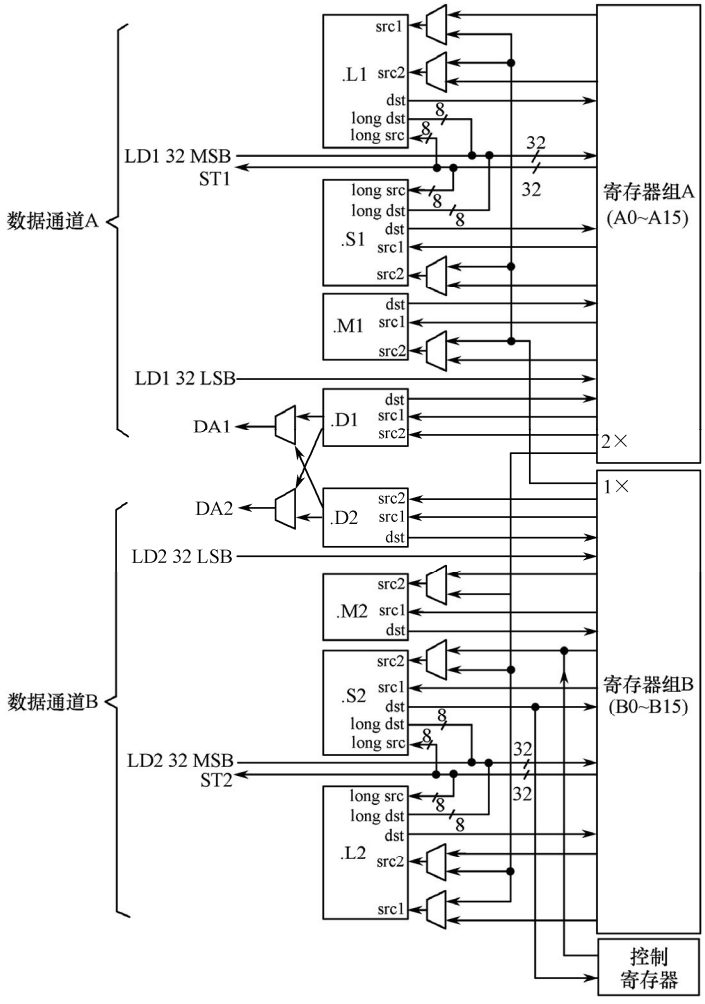


图 2-2　TMS320C67xx CPU 的数据通道

C67xx CPU 的数据通道如图 2-2 所示，由图可见，数据通道包括下述物理资源：

- 两个通用寄存器组(A 和 B)，分别包括 16 个寄存器；
- 8 个功能单元(.L1、.L2、.S1、.S2、.M1、.M2、.D1、.D2)；
- 两个数据读取通路(LD1 和 LD2)，每侧有两个 32 位读取总线；
- 两个数据存储通路(ST1 和 ST2)，每侧有一个 32 位存储总线；
- 两个寄存器组交叉通路(1×和 2×)；
- 两个数据寻址通路(DA1 和 DA2)。

1．通用寄存器组

在 C67xx CPU 数据通道中有两个通用寄存器组(A 和 B)，每个寄存器组包括 16 个 32 位寄存器，通用寄存器的作用是：

① 存放数据，作为指令的源操作数和目的操作数。图 2-2 中 src1、src2、long src、dst 和 long dst 表示通用寄存器与功能单元之间的数据联系、传送方向和数据字长。

② 作为间接寻址的地址指针，寄存器 A4～A7 和 B4～B7 还能够以循环寻址方式工作。

③ A1、A2、B0、B1 和 B2 可用作条件寄存器。

C67xx 所有芯片都支持 32 位和 40 位定点运算。32 位数据可放在任一通用寄存器内，40 位数据需存放在一个寄存器对内。一个寄存器对由一个偶寄存器及序号比它大 1 的奇寄存器组成，书写时奇寄存器在前面，两个寄存器之间加比号，C67xx 有效的寄存器对见表 2-1。数据的低 32 位放在偶寄存器，数据的高 8 位放在奇寄存器的低 8 位。C67xx DSP 芯片也以上述方式用寄存器对存放 64 位双精度数。

表 2-1　40 位/64 位寄存器对

寄存器组 A		寄存器组 B	
A1:A0	A9:A8	B1:B0	B9:B8
A3:A2	A11:A10	B3:B2	B11:B10
A5:A4	A13:A12	B5:B4	B13:B12
A7:A6	A15:A14	B7:B6	B15:B14

图 2-3 给出了 40 位长型数据在寄存器对中的存储方式。对长型数据进行读操作时，忽略掉行寄存器中的高 24 位；进行写操作时，用 0 填充奇寄存器的高 24 位。在指令操作码中指定所用的偶寄存器编码，就指定了所用的寄存器对。

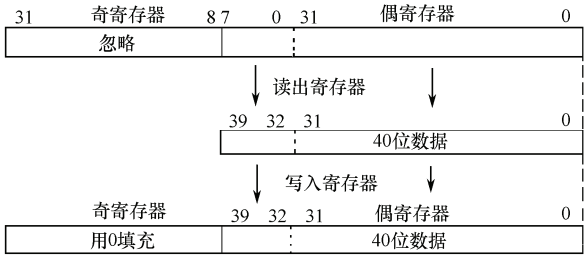


图 2-3　40 位长型数据在寄存器对中的存储方式

2. 功能单元

C67xx 每个数据通道有 4 个功能单元。两个数据通道具有功能基本相同的功能单元。.M 单元主要完成乘法运算，.D 单元是唯一能产生地址的功能单元，.L 与.S 单元是主要的算术逻辑运算单元(ALU)。表 2-2 描述了各功能单元的功能。

表 2-2　功能单元及其能执行的操作

功能单元	定点操作	浮点操作
.L 单元	32 位/40 位算术和比较操作 32 位最左边 1 或 0 的位数计数 32 位和 40 位归一化操作 32 位逻辑操作	算术操作 数据类型转换操作 DP(双精度)→SP(单精度) INT(整型)→DP，INT→SP
.S 单元	32 位算术操作 32 位/40 位移位和 32 位位域操作 32 位逻辑操作 转移 常数产生 寄存器与控制寄存器数据传递(仅.S2)	比较 倒数和倒数平方根操作 绝对值操作 SP→DP 数据类型转换
.M 单元	16×16 位乘法操作	32×32 位乘法操作 浮点乘法操作
.D 单元	32 位加、减、线性及循环寻址计算 带 5 位常数偏移量的字读取与存储 带 15 位常数偏移量的字读取与存储(仅.D2)	带 5 位常数偏移量的双字读取

CPU 内的数据总线支持 32 位操作数，有些支持长型(40 位)操作数。双精度操作数则分成最高位(MSB)、最低位(LSB)两组 32 位总线。图 2-2 中每个功能单元都有各自到通用寄存器的读/写端口。其中 A 组的功能单元(以 1 结尾)写到寄存器组 A 中，B 组的功能单元(以 2 结尾)写到寄存器组 B 中。每个功能单元都有两个 32 位源操作数 src1 和 src2 的读入口。为了实现长型(40 位)操作数的读/写，4 个功能单元(.L1、.L2、.S1 和.S2)分别配有额外的 8 位写端口和读端口。由于每个功能单元都有自己的 32 位写端口，所以在每个周期 8 个功能单元可以并行使用。

3. 寄存器组交叉通路

每个功能单元可以直接与所处数据通道的寄存器组进行读/写操作，即.L1、.S1、.D1 和.M1 可以直接读/写寄存器组 A，而.L2、.S2、.D2 和.M2 可以直接读/写寄存器组 B。两个寄存器组通过 1×和 2×交叉通路也可以与另一侧的功能单元相连。1×交叉通路允许数据通道 A 的功能单元从寄存器组 B 读它的源操作数，2×交叉通路则允许数据通道 B 的功能单元从寄存器组 A 读它的源操作数。

从图 2-2 中可以看出，.D 单元与交叉通路不连，只有其余 6 个单元可以访问另一侧的寄存器组。其中，.M1、.M2、.S1 和.S2 单元的源操作数 src2 在交叉通路和自身通路的寄存器组之间可选，.L1 和.L2 的两个源操作数 src1 和 src2 都可在交叉通路和自身通路的寄存器组之间选择。

在 C67xx 的 CPU 中仅有两个交叉通路 1×和 2×，在 1 个周期内只能从另一侧寄存器组读取 1 次源操作数，即在 1 个周期内总共只能进行两个交叉通路的源操作数读入，每个执行包的每个数据通道仅有 1 个功能单元可从对侧获得源操作数。

4. 数据存储器及读取存储通路

在 C67xx 的 CPU 中，有两个 32 位通路(每侧 1 个)把数据从存储器读取到寄存器(Load 指令)中，读入到寄存器组 A 中的通路为 LD1，读入到寄存器组 B 中的通路为 LD2。C67xx 除此之外，还有第 2 个 32 位读取通路，图 2-2 中标注为 LD1 32 MSB 和 LD2 32 MSB。C67xx 的 LDDW 指令一次可读取 64 位数据到 A 侧寄存器或到 B 侧寄存器中。C67xx 有两个 32 位写数据通道 ST1 和 ST2，可分别将各组寄存器的数据存储到数据存储器(Store 指令)中。

5. 数据地址通路

数据地址通路 DA1 和 DA2 来自数据通道的.D 功能单元，地址通路与两侧数据通道都相连，这使得一个寄存器组产生的数据地址能够支持任意一侧寄存器组对数据存储器的读/写操作。

在汇编语句内，数据通道(读数据线 LD、写数据线 ST)以.T1、.T2 表示。在 Load 和 Store 指令的汇编语句里.T1、.T2 与.D1、.D2 一起出现在功能单元区，用以说明产生地址的功能单元和读/写操作所用的数据通道。例如，下面的 Load 指令使用.D1 产生地址，用 LD2 数据通道读入数据到 B1 寄存器中：

```
LDW.D1T2   *A0[3]，B1
```

6. 控制寄存器组

用户可以通过控制寄存器组编程来选用 CPU 的部分功能。编程时应注意，仅功能单元.S2 可通过搬移指令 MVC 访问控制寄存器，从而对控制寄存器进行读/写操作。流水线 E1 节拍程序计数器(PCE1)，保留当前处于 E1 节拍取指包的 32 位地址。中断管理的 7 个寄存器将在 2.5 节中介绍。C67xx CPU 除上述控制寄存器外，为支持浮点运算，还另外配置了 3 个寄存器控制浮点运算。表 2-3 列出了 C67xx 的控制寄存器组，并对每个控制寄存器做了简单描述。

表 2-3　C67xx DSP 的控制寄存器组

寄存器	缩写	功能描述
寻址模式寄存器	AMR	指定是否使用线性或循环寻址。如果是循环寻址，还包括循环寻址的尺寸
控制状态寄存器	CSR	包括全局中断使能位，高速缓冲存储器控制位和其他各种控制和状态位
浮点加法配置寄存器	FADCR	指定.L 单元的溢出方式，舍入方式，记录 NaN 及其他异常
浮点辅助配置寄存器	FAUCR	记录.S 单元 NaN 及其他异常
浮点乘法配置寄存器	FMCR	指定.M 单元的溢出方式，舍入方式，记录 NaN 及其他异常
中断清除寄存器	ICR	允许软件清除挂起的中断
中断使能寄存器	IER	允许使能/禁止个别中断
中断标志寄存器	IFR	显示中断状态
中断返回指针	IRP	保存从可屏蔽中断返回时的地址
中断设置寄存器	ISR	允许软件控制设置中断
中断服务表指针	ISTP	指向中断服务表的起始地址
不可能屏蔽中断返回指针	NRP	保存从不可屏蔽中断返回时的地址
程序计数器	PCE1	保存处于流水线 E1 节拍的取指包地址

控制状态寄存器(CSR)包括控制位和状态位，如图 2-4 所示。控制状态寄存器各字段功能列于表 2-4 中。对于 EN，PWRD，PCC 和 DCC 字段，要查看有关数据手册来确定所用的芯片是否支持这些字段控制选择。

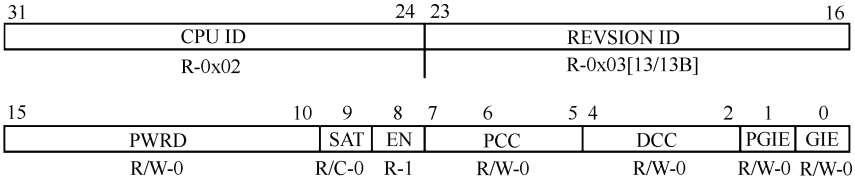


图 2-4　控制状态寄存器

图 2-4 中 TI 公司文献图表符号说明：R 代表可读，对控制寄存器须用 MVC 指令才能读；W 代表可写，对控制寄存器须用 MVC 指令才可写；x 代表复位后数值不定；0 代表复位后数值为 0(若为 1，则代表复位后数值为 1)；c 代表可清零，对控制寄存器须用 MVC 指令清零。

表 2-4　控制状态寄存器字段描述

字 段 名 称	功 能 描 述
CPU ID	CPU ID(识别号)，00b 代表 C62x，10b 代表 C67xx，1000b 代表 C64x
REVISION ID	修订版号
PWRD	控制低功耗模式，该值读时总为零
SAT	饱和位。任一功能单元执行一个饱和操作时被置 1，饱和位只能用 MVC 指令清除。当在同一周期内发生清除和置位时，功能单元对它的置位优先。饱和位在饱和发生一个周期后被置位
EN	字节存储次序：1=小端存储，0=大端存储
PCC	程序高速缓存控制模式
DCC	数据高速缓存控制模式
PGIE	当一个中断发生时，保存以前的全局中断使能位 GIE
GIE	全局中断使能位，它控制除复位和不可屏蔽中断之外的所有可屏蔽中断使能：GIE=1 时，可屏蔽中断使能；GIE=0 时，可屏蔽中断禁止

2.2　存储器映射

C67xx 系列 DSP 的存储空间(包括片内、片外存储器和控制寄存器)以字节为单位统一编址，地址宽度为 32 位，存储器映射见表 2-5，表中列出了每个存储区域的大小，以及起始地址和终止地址。

表 2-5　C67xx DSP 的存储器映射

描　　述	区域大小(B)	起 始 地 址	终 止 地 址
内部 RAM(L2)	192K	0x00000000	0x0002 FFFF
内部 RAM/Cache	64K	0x00030000	0x0003 FFFF
EMIF 寄存器	256K	0x01800000	0x0183 FFFF
L2 寄存器	128K	0x01840000	0x0185 FFFF
HPI 寄存器	256K	0x01880000	0x018B FFFF
McBSP 0 寄存器	256K	0x018C0000	0x018FFFFF
McBSP 1 寄存器	256K	0x01900000	0x0193 FFFF
Timer 0 寄存器	256K	0x01940000	0x0197 FFFF
Timer 1 寄存器	256K	0x01980000	0x019B FFFF
中断选择寄存器	512	0x019C0000	0x019C01FF
设备配置寄存器	4	0x019C0200	0x019C 0203
EDMA RAM and EDMA 寄存器	256K	0x01A00000	0x01A3FFFF
GPIO 寄存器	16K	0x01B00000	0x01B03FFF
I^2C 0 寄存器	16K	0x01B40000	0x01B43FFF
I^2C 1 寄存器	16K	0x01B4 4000	0x01B4 7FFF
McASP 0 寄存器	16K	0x01B4 C000	0x01B4 FFFF
McASP 1 寄存器	16K	0x01B5 0000	0x01B5 3FFF
PLL 寄存器	8K	0x01B7 C000	0x01B7 DFFF
仿真寄存器	256K	0x01BC 0000	0x01BF FFFF

（续表）

描　　述	区域大小(B)	起 始 地 址	终 止 地 址
QDMA 寄存器	52	0x0200 0000	0x0200 0033
McBSP 0 数据端口	64M	0x3000 0000	0x33FF FFFF
McBSP 1 数据端口	64M	0x3400 0000	0x37FF FFFF
McASP 0 数据端口	1M	0x3C00 0000	0x3C0F FFFF
McASP 1 数据端口	1M	0x3C10 0000	0x3C1F FFFF
EMIF CE 0(SDRAM 空间)	256M	0x8000 0000	0x8FFF FFFF
EMIF CE 1(FLASH 空间)	256M	0x9000 0000	0x9FFF FFFF
EMIF CE 2	256M	0xA000 0000	0xAFFF FFFF
EMIF CE 3	256M	0xB000 0000	0xBFFF FFFF

从应用的角度来看，处理器的速度较存储器的更快，因此大容量高速片上存储器是理想之选，但目前的高速存储器比低速存储器体积大且价格高。若采用扁平存储器结构，CPU 和内部存储器均工作在 300MHz 时钟频率，这样可以避免存储器阻塞现象的发生，如图 2-5(a)所示。但是，在访问外部存储器时，需要 CPU 暂停。在这种情况下，实际的 CPU 处理速度接近于较慢的存储器速度。

解决上述问题的方法是采用分级存储器体系结构，如图 2-5(b)所示。在 CPU 附近放置一个高速、小容量的存储器，使 CPU 访问时无须暂停；下一级放置一个速度较慢、体积增加的存储器，其与 CPU 距离较远。在分级结构中，地址由大容量存储器映射到小容量高速存储器中。较高级别的存储器即为高速缓冲存储器(Cache)，采用这种分级结构，可以使存储器的平均访问时间接近最快的存储器访问时间。

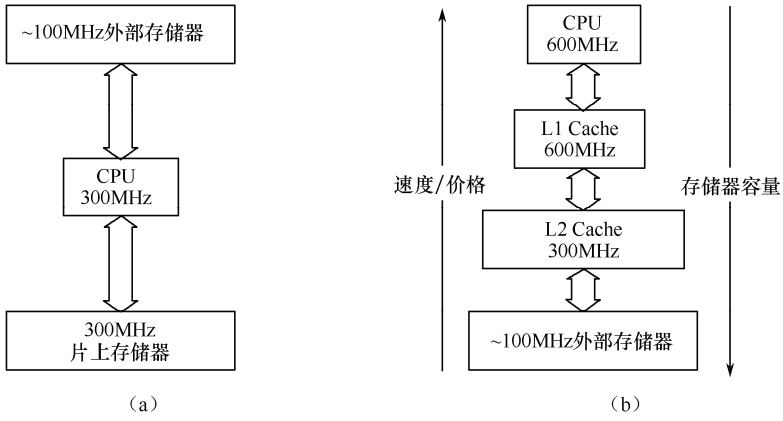


图 2-5　扁平和分级的存储器体系结构

C67xx 的片内 RAM 就是采用两级高速缓冲存储器结构，程序和数据拥有各自独立的高速缓存。第一级程序高速缓存称为 L1P，第 1 级数据高速缓存称为 L1D。第二级存储器/高速缓存称为 L2，由程序和数据高速缓存共同使用。片内两级高速缓存的结构如图 1-5 所示。

C67xx 的 L1P 是容量为 4KB 的直接映射缓存，L1P 操作由 CPU 控制状态寄存器(CSR)、L1P 冲洗基地址寄存器(L1PFBAR)、L1P 冲洗字计数寄存器(L1PFWC)及缓存配置寄存器(CCFG)控制。

C67xx 的 L1D 是容量为 4KB 的组相联高速缓存，L1D 操作由 CPU 控制状态寄存器(CSR)、

L1D 冲洗基地址寄存器(L1DFBAR)、L1D 冲洗字计数寄存器(L1DFWC)和缓存配置寄存器(CCFG)控制。

数据访问如果命中 L1D，将在一个周期内返回数据，不阻塞 CPU。L1D 缺失而 L2 命中将使 CPU 阻塞 4 个周期。若 L1D 缺失，L2 也缺失，则 CPU 将一直阻塞，直到 L2 从外部存储器中得到数据并将其传输到 L1D，L1D 再把数据返回到 CPU。

C67xx 的 L2 容量为 64KB，由 CCFG 寄存器的 L2MODE 字段配置为 5 种模式，如图 2-6 所示。L2 操作由缓存配置寄存器(CCFG)、L2 冲洗基地址寄存器(L2FBAR)、L2 冲洗字计数寄存器(L2FWC)、L2 清除基地址寄存器(L2CBAR)、L2 清除字寄存器(L2CWC)、L2 冲洗寄存器(L2FLUSH)、L2 清除寄存器(L2CLEAN)控制。

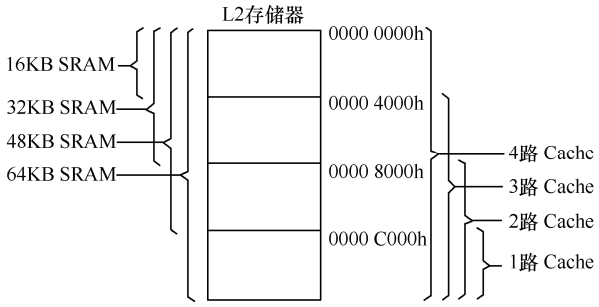


图 2-6　C67xx 的 L2 存储器配置模式

除了上述缓存控制寄存器外，存储器属性寄存器(MAR)控制外存某一段空间的高速缓存使能。如果某一段片外地址范围设置为不可高速缓存，对该地址的访问将略过所有的高速缓存，直接访问外部存储器，这种访问称为远距离访问。设置为可高速缓存时，访问外存会返回整个高速缓存行大小的数据。表 2-6 列出了与高速缓存有关的控制寄存器。

表 2-6　C67xx 内部高速缓存控制寄存器

寄存器缩写	寄存器名称
CCFG	缓存配置寄存器
L2FBAR	L2 冲洗基地址寄存器
L2FWC	L2 冲洗字计数寄存器
L2CBAR	L2 清除基地址寄存器
L2CWC	L2 清除字计数寄存器
L1PFBAR	L1P 冲洗基地址寄存器
L1PFWC	L1P 冲洗字计数寄存器
L1DFBAR	L1D 冲洗基地址寄存器
L1DFWC	L1D 冲洗字计数寄存器
L2FLUSH	L2 冲洗寄存器
L2CLEAN	L2 清除寄存器
MAR0	控制 CE0 范围为 80000000h～80FFFFFFh
MAR1	控制 CE0 范围为 81000000h～81FFFFFFh
MAR2	控制 CE0 范围为 82000000h～82FFFFFFh
MAR3	控制 CE0 范围为 83000000h～83FFFFFFh
MAR4	控制 CE1 范围为 90000000h～90FFFFFFh
MAR5	控制 CE1 范围为 91000000h～91FFFFFFh
MAR6	控制 CE1 范围为 92000000h～92FFFFFFh

（续表）

寄存器缩写	寄存器名称
MAR7	控制 CE1 范围为 93000000h～93FFFFFFh
MAR8	控制 CE2 范围为 A0000000h～A0FFFFFFh
MAR9	控制 CE2 范围为 A1000000h～A1FFFFFFh
MAR10	控制 CE2 范围为 A2000000h～A2FFFFFFh
MAR11	控制 CE2 范围为 A3000000h～A3FFFFFFh
MAR12	控制 CE3 范围为 B0000000h～B0FFFFFFh
MAR13	控制 CE3 范围为 B1000000h～B1FFFFFFh
MAR14	控制 CE3 范围为 B2000000h～B2FFFFFFh
MAR15	控制 CE3 范围为 B3000000h～B3FFFFFFh

2.3 汇编指令集

C67xx CPU 公共指令集是一个定点运算指令集。为方便讲述，将它们分为读取/存储类指令、算术运算类指令、逻辑及字段操作类指令、搬移类指令、程序转移类指令及浮点运算类指令等。

2.3.1 指令集概述

1．指令和功能单元之间的映射

C67xx CPU 的汇编语言每条指令只能在一定的功能单元执行，因此就形成了指令和功能单元之间的映射关系。表 2-2 列出了功能单元所能执行的操作。因此可以相应地给出指令到功能单元的映射，指出每条指令可在哪些功能单元运行；也可以给出功能单元到指令的映射，指出每个功能单元可以运行哪些指令。一般而言，与乘法相关的指令都在.M 单元执行；而需要产生数据存储器地址的指令则要用到.D 功能单元；算术逻辑运算大多在.L 与.S 单元执行。

2．延迟间隙

C67xx CPU 采用流水线结构，从指令进入 CPU 的取指单元到指令执行完毕，需要多个时钟周期。所谓的单指令周期是指它最高的流水处理速度。由于指令复杂程度的不同，各种指令的执行周期也不相同，因此程序员就需要了解指令执行的相对延迟，以掌握每条指令的执行结果何时可以被后续指令利用。指令的执行速度可以用延迟间隙(Delay Slots)来说明。延迟间隙在数量上等于从指令的源操作数被读取，直到执行的结果可以被访问所需要的指令周期数。对于单周期类型指令(如 ADD)，源操作数在第 i 周期被读取，计算结果在第(i+1)周期即可被访问，等效于无延迟。对乘法指令，若源操作数在第 i 周期被读取，计算结果在第(i+2)周期才能被访问，延迟周期为 1。表 2-7 列出了各类指令的延迟间隙和功能单元的等待时间。

表 2-7 C67xx 公共指令集的延迟间隙和功能单元的等待时间

指令类型	延迟间隙	功能单元等待时间	读周期	写周期
NOP	0	1		
存储指令 STx	0	1	i	i
读取指令 LDx	4	1	i	$i, i+4$
跳转指令 B	5	1	i	$i+5$
2 周期 DP 指令	1	1	i	$i, i+1$
4 周期指令	3	1	i	$i+3$
INT 到 DP 转换	4	1	i	$i+3, i+4$
DP 比较指令	1	2	$i, i+1$	$i+1$

（续表）

指令类型	延迟间隙	功能单元等待时间	读周期	写周期
ADDDP/SUBDP	6	2	$i, i+1$	$i+5, i+6$
MPYSP2DP	4	2	i	$i+3, i+4$
MPYSPDP	6	3	$i, i+1$	$i+5, i+6$
MPYI	8	4	$i, i+1, i+2, i+3$	$i+8$
MPYID	9	4	$i, i+1, i+2, i+3$	$i+8, i+9$
MPYDP	9	4	$i, i+1, i+2, i+3$	$i+8, i+9$

表 2-7 中第 4、5 列以进入流水线 E1 节拍为第 *i* 周期，列出了各类指令做读/写操作所需要的周期数。对于转移类指令，如果是转移到标号地址的指令或是由中断 IRP 和 NRP 引起的转移，则没有读操作；对于 Load 指令，在第 *i* 周期读地址指针并且在该周期内修改基地址，在第 *i*+4 周期向寄存器写，使用的是不同于.D 单元的另一个写端口。

C67xx 所有的公共指令都只有一个功能单元等待时间，这意味着每个周期功能单元都能够开始一个新指令。单周期功能单元等待时间的另一术语是单周期吞吐量。

3．指令操作码映射图

C67xx 的每条指令都是 32 位。每条指令都有自己的代码，详细指明指令的内容。图 2-7 给出了指令操作码映射图(opcode map)。其中，op 为指令操作代码，creg 指定条件寄存器的代码，z 指定条件，src、dst 分别指定源操作数及目的操作数代码，s 选择寄存器组 A 或 B 作为目的操作数，x 指定源操作数 2 是否使用交叉通道，p 指定是否并行执行等。把汇编语句变成机器代码，由汇编器(assembler)完成，把机器代码反汇编成汇编语句也是由专用工具程序实现的。

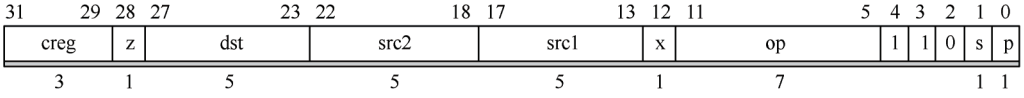


图 2-7　指令操作码映射图

4．并行操作

CPU 运行时，总是一次取 8 条指令，组成一个取指包，取指包的基本格式如图 2-8 所示。取指包一定在地址的 256 位(8 个字)边界定位。

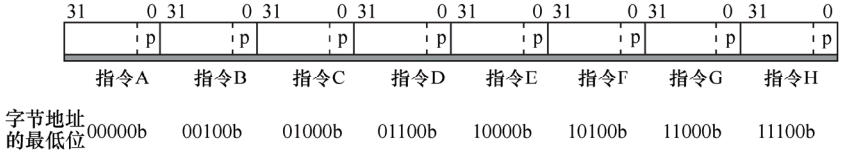


图 2-8　取指包的基本格式

每条指令的最后一位是并行执行位(p 位)，p 位决定本条指令是否与取指包中的下一条指令并行执行。CPU 对 p 位从左至右(从低地址到高地址)进行扫描：如果指令 *i* 的 p 位是 1，则指令 *i*+1 就将与指令 *i* 在同一周期内并行执行；如果指令 *i* 的 p 位是 0，则指令 *i*+1 将在指令 *i* 的下一周期内执行。所有并行执行的指令组成一个执行包，其中最多可以包括 8 条指令。执行包中的每条指令使用的功能单元必须各不相同。执行包不能超出 256 位边界，因此，取指包最后一条指令的 p 位必须设定为 0，而每一取指包的开始也将是一个执行包的开始。

一个取指包中 8 条指令的执行顺序可能有几种不同形式：完全串行，即每次执行一条指令；完全并行，即 8 条指令是一个执行包；部分串行，即分成几个执行包。图 2-9 给出指令部分串行执行的例子，根据 p 位的数值，此取指令包将按表 2-8 所列的顺序执行。

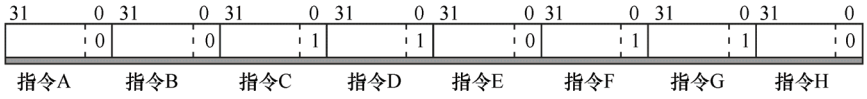


图 2-9　指令包的并行标志位

注意：指令 C，D，E 和 F，G，H 不能使用相同的功能单元、交叉通路或其他的路径资源。

表 2-8 指令执行包

周期/执行包	指　令	周期/执行包	指　令
1	A	3	C, D, E
2	B	4	F, G, H

如果有转移指令使程序在执行过程中向外跳转到某一执行包中间的某一条指令，则程序从该条指令继续执行，该执行包中跳转目标之前的指令将被忽略。在上例中，如果跳转目标是指令 D，则只有指令 D 和 E 将被执行。虽然指令 C 与 D 处于同一执行包中，但也得不到执行。至于指令 A 和 B，由于处于前一执行包中，更不会得到执行。如果程序的运行结果依赖于指令 A、B 或 C 的执行结果，这样直接向指令 D 的跳转将会产生错误。

5．条件操作

所有的 C67xx 指令都可以是有条件执行的，反映在指令代码的 4 个最高有效位(见图 2-7)。其中，3 位操作码字段 creg 指定条件寄存器，1 位字段 z 指定是零测试还是非零测试。在流水操作的 E1 节拍，对指定的条件寄存器进行测试：如果 z=1，进行零测试，即条件寄存器的内容为 0 是真；如果 z=0，进行非零测试，即条件寄存器的内容非零是真。如果设置为 creg=0，z=0，则意味着指令将无条件地执行。对 C67xx，可使用 A1、A2、B0、B1 和 B2 这 5 种寄存器作为条件寄存器。

在书写汇编程序时，以方括号对条件操作进行描述，方括号内是条件寄存器的名称。下面所示的执行包中含有两条并行的 ADD 指令：

```
     [B0]    ADD.L1   A1,A2,A3
|| [!B0]    ADD.L2   B1,B2,B3
```

第 1 条 ADD 指令在寄存器 B0 非零的条件下执行，第 2 条 ADD 指令在 B0 为零的条件下执行。以上两条指令是相互排斥的，即只有一条指令会被执行。

6．字节存储次序

Endian 是指多字节数据内部高低有效位的存放顺序。在小端存储格式(Little-Endian)下，数据的高有效位字节存放在地址高位字节，低有效位放在地址低位字节，这与 Intel 公司数据存放惯例相同。大端存储格式(Big-Endian)则相反，与 Motorola 等公司的数据存放惯例相同。字节存储次序由芯片的相应引脚 HD8 的电平决定，并反映在 CSR 寄存器的 EN 位。HD8=1 为小端存储，HD8=0 为大端存储。

2.3.2　寻址方式

寻址方式指 CPU 如何访问其数据存储空间，C67xx DSP 全部采用间接寻址，所有寄存器都可以作为线性寻址的地址指针。表 2-9 列出了读取/存储类指令访问数据存储器地址的汇编语法格式。其中，ucst5 代表无符号二进制 5 位常数偏移量。对寄存器 B14 和 B15 可用 ucst15(无符号二进制 15 位常数偏移量)。变址计算的符号与常用的 C 语言惯例相同。

表 2-9　读取/存储类指令访问数据存储器地址

寻　址　方　式	不修改地址寄存器	先修改地址寄存器	后修改地址寄存器
寄存器间接寻址	*R	*＋＋R *－－R	*R＋＋ *R－－
寄存器相对寻址	*＋R[ucst5] *－R[ucst5]	*＋＋R[ucst5] *－－R[ucst5]	*R＋＋[ucst5] *R－－[ucst5]
基地址+变址	*＋R[offseR] *－R[offseR]	*＋＋R[offseR] *－－R[offseR]	*R＋＋[offseR] *R－－[offseR]
带 15 位常数偏移量的寄存器相对寻址	*＋B14/B15[ucst15]		

除作为线性寻址指针外，A4～A7、B4～B7 这 8 个寄存器还可作为循环寻址的地址指针，由寻址模式寄存器 AMR 控制地址修改方式：线性方式(默认方式)或循环方式。图 2-10 给出了寻址模式寄存器各个字段的定义，表 2-10 给出了由模式(Mode)的两位数值所确定的选择。

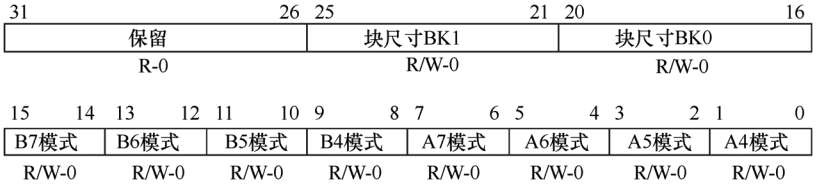


图 2-10　寻址模式寄存器各个字段的定义

表 2-10　寻址模式寄存器模式选择字段编码

模式	描　　述	模式	描　　述
00	线性寻址(复位后默认值)	10	循环寻址使用 BK1 字段
01	循环寻址使用 BK0 字段	11	保留

在线性寻址方式下，基地址按照指定的加减量线性修改；在循环寻址方式下，地址在块尺寸范围内循环修改。块尺寸字段 BK0 和 BK1 含有 5 位数值，用于计算循环寻址的块尺寸，块尺寸与 BK0/BK1 内 5 位数值 N 的关系为：

$$\text{块尺寸} = 2^{(N+1)} \text{ 字节}$$

例如，设 N 的二进制数为 00010，等于十进制数 2，则块尺寸为 $2^{(2+1)} = 8$ 字节。

2.3.3　读取/存储类指令

读取指令可从数据存储器读取数据送到通用寄存器，存储指令可把通用寄存器的内容送到数据存储器中保存。这些指令包括：

- 读取指令：LDB/LDBU/LDH/LDHU/LDW/LDDW
- 存储指令：STB/STH/STW

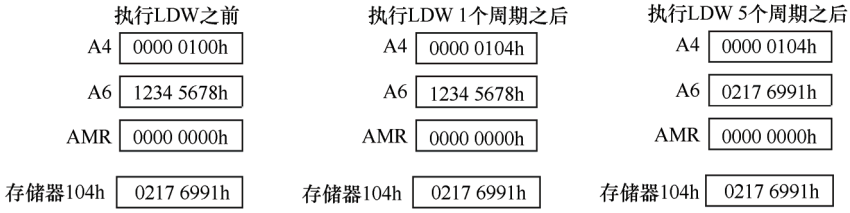
在 C67xx 的读取/存储指令里，数据长度有单字节(B)、双字节(半字 H)和四字节(字 W)等多种。双字节型数据的地址必须从偶数开始，即其地址最低位为 0，四字节数据地址最低 2 位必须为 0，分别称为半字、字边界。在计算或书写地址时，均以它们的最低位地址作为存储单元地址的代表。在汇编语言或 C 语言中开辟数据或变量区时，需要根据数据类型调节其地址起始点，称为边界调整(alignment)。LDB(U)/LDH(U)/LDW 指令分别读入字节/半字/字，因为地址均以字节为单位，所以在计算地址修正量时，要分别乘以相应的比例因子 1、2 和 4。STB/STH/STW 同样计算地址。

LDB/LDH 指令读入的是有符号数(补码数)，将字节/半字写入寄存器时，应对高位做符号扩展。LDBU/LDHU 指令读入的是无符号数，将字节/半字写入寄存器时，应对高位补零。

【例 2-1】 线性寻址方式下的地址计算：

```
LDW.D1   *++A4[1],  A6
```

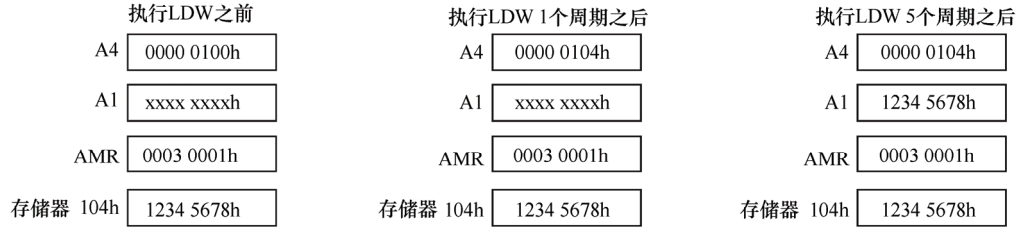
此例为先修改地址，地址偏移量按 1×4 计算，计算结果如下所示：



【例 2-2】 循环寻址方式下的地址计算：

```
LDW.D1   *++A4[9],  A1
```

假设寻址模式寄存器 AMR=0x00030001，则 A4 已被设定为循环寻址方式，块尺寸为 16=10h(*N*=3)。因为是以字为单位读取的，所以变址偏移量为 9×4=24h。线性寻址时，地址应为 0x00000124h；循环寻址时，24h 对 10h 取模，余数为 4，故实际寻址地址为 0x00000104h，计算结果如下所示：



2.3.4 算术运算类指令

1．加、减运算指令及溢出问题

加、减运算指令可分为以下几类。

① 有符号数加、减运算指令，包括操作数为整型(32 位)或长整型(40 位)的 ADD、SUB 指令，以及操作数为半字(16位)的 ADD2/SUB2 指令。ADD2/SUB2 指令的特点是同时进行两个 16 位补码数的加、减运算，高半字与低半字之间没有进/借位，各自独立进行。

② 无符号数加、减运算指令 ADDU 和 SUBU，操作数为 32 位或 40 位无符号数。

③ 带饱和的有符号数加、减运算指令 SADD 和 SSUB，操作数为 32 位或 40 位有符号数。

④ 与 16 位常数进行加法操作的指令 ADDK。

⑤ 按寻址方式的加、减运算类指令 ADDAB/ADDAH/ADDAW/ADDAD，SUBAB/SUBAH/SUBAW。

采用补码的形式表示所有参与运算的操作数，最高位为符号位，0 表示“正”，1 表示“负”，其余为数值位。正数的补码与原码相同，最高位为 0，其余位为数值；负数用补码表示，最高位为 1，可先写出该负数对应的正数，再将其按位取反，最后在末位加 1 得到。

【例 2-3】若字长为 32 位，求十进制数 365 和–365 的补码。

$[+365]_{补}$=0000 0000 0000 0000 0000 0001 0110 1101_B = 0000 016D $_H$
按位取反：1111 1111 1111 1111 1111 1110 1001 0010
末位加 1：1111 1111 1111 1111 1111 1110 1001 0011
$[-365]_{补}$=1111 1111 1111 1111 1111 1110 1001 0011_B = FFFF FE93 $_H$

操作数可能是整数或小数，小数有两种表示方法：定点数表示法和浮点数表示法。定点数就是小数点位置固定的数，若小数点位于数据第 n 位的右侧，则称为 Q_n 格式数，当小数点的位置固定在数据的最高位之前时为定点小数，当固定在最低位之后时为定点整数。对 32 位 DSP 来说，其 Q_n 格式数有 Q_0、Q_1、…、Q_n、…、Q_{31}，共 32 种。

定点小数(Q_0)：定点小数是纯小数，小数点位置在符号位之后、有效数值最高位之前，形式为 $x = x_0.x_1x_2\cdots x_n$ (其中 x_0 为符号位，x_1～x_n 是有效数值，x_1 为最高有效位)，如图 2-11 所示，表示的数据范围为 $2^{-n} \leqslant |x| \leqslant 1-2^{-n}$。

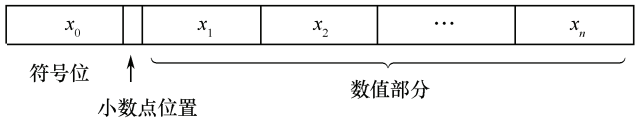


图 2-11　定点小数(Q_0)

定点整数：定点整数是纯整数，小数点位置在有效数值部分最低位之后，形式为 $x=x_0x_1x_2\cdots x_n$，如图 2-12 所示，表示的数据范围为 $1 \leqslant |x| \leqslant 2^n-1$。

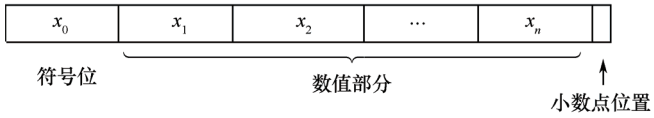


图 2-12　定点整数

浮点数是用科学计数法来表示的数据，小数点的位置可按数值大小自动变化，浮点数的定义见 2.3.8 节。

如果运算结果超出目的操作数字长所能表示的范围，造成运算结果的高位丢失，使保存的运算结果不正确，称为溢出。由于定点数以全字长表示一个整数，其能覆盖的数据动态范围较小，定点加减运算指令(无论是无符号或有符号加减运算指令)易产生溢出。例如，若目的操作数选定为半字(16 位字长)，它所能表示的补码数范围是−32768～+32767，能表示的无符号数范围是 0～65535。如果运算结果超出此范围，它将只保留运算结果的低 16 位，高位丢失，运算结果不正确。有符号数(补码数)在产生溢出时，会改变运算结果的符号。这两种情况都是十分有害的。在 C67xx 的指令里，普通加减运算指令产生溢出时在 CPU 内不会留下任何标志，所以对此要充分重视。通常有 3 种办法解决溢出问题。

① 用较长的字长来存放运算结果，使目的操作数字长超出源操作数的字长。超出源操作数字长的部分称为保护位。例如，两个 32 位整型数的加、减，用 40 位存放结果，有 8 个保护位。但增加保护位要占用系统资源，还可能会降低运算速度(例如，有些系统存储 40 位整数要比存 32 位整数占用较多时间)，使用时必须综合考虑。

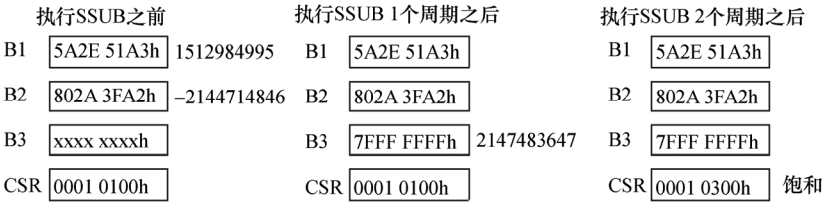
② 用带饱和的加减运算指令 SADD、SSUB 做补码数加、减运算。当产生溢出时这类指令将使目的操作数置为同符号的最大值(绝对值)，即保持运算结果的符号不变，同时使 CPU 的状态寄存器 CSR 内的 SAT 位置 1，提醒用户注意。

③ 对整个系统乘一个小于 1 的比例因子，即将所有参与运算的数值减小，以保持运算过程不产生溢出，但这种方法会降低计算精度。

【例 2-4】 减法运算举例，对相同的 src1、src2 使用不同的减法指令说明饱和减法指令与减法指令的差别以及保护位对防止溢出的作用。

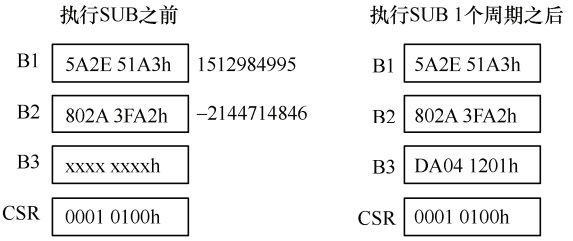
```
SSUB.L2    B1, B2, B3
```

计算结果如下所示：



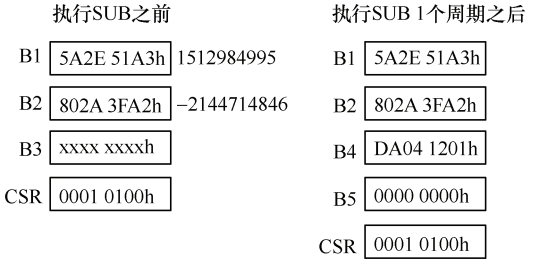
这里用带饱和的减法运算保证了结果的符号不变。存放到目的寄存器 B3 中的内容为 32 位正最大值。CSR 寄存器的饱和位置 1，提示已产生了溢出。若改用普通减法指令，将产生溢出。

```
SUB.L2    B1, B2, B3
```



此时存入 B3 寄存器的内容为 0xDA041201，它是一个负数(−637267455)，得数的正负号及数值全错了。如果把目的操作数改用长型整数，即把指令改成：

```
SUB.L2    B1, B2, B5:B4
```



则得数为 B5:B4＝0x00DA041201，存放在 B5 寄存器中的内容为 0x00，存放在 B4 寄存器中的内容为 0xDA041201。它是一个 40 位有符号数(+3657699841)，结果正确。

在数字信号处理中，经常要计算累加和，这时更要特别注意溢出问题。例 2-4 是一个简单的求多个整型数累加和的汇编程序，它用 40 位的长型数存放和数，防止溢出。

【例 2-5】 计算累加和的程序，用长型数存放和数，有 8 位保护位。

下述程序在进入 loop 循环前，已使寄存器 A4 指向存放数据的基地址，寄存器 B1 存放欲累加的个数，寄存器组 A3:A2 用来存放累加和，进入循环前已清零。

```
Loop: LDW.D1    *A4++, A0
      NOP    4
      ADD.L1    A3:A2, A0, A3:A2
      SUB.L2    B1, 1, B1
[B1]  B.S1    Loop
      NOP    5
```

在求多个同字长数的累加和时，如果存放结果的字长增加 N 位，可以保证 2^N-1 次累加运算不溢出。例 2-5 中，源操作数字长 32 位，用 40 位字长存放累加结果，有 8 个保护位，可以确保 255 次 32 位字的累加运算无溢出。这个估计是比较保守的，只有在所有源操作数同符号，且绝对值都较大时，才会达到限度。如果两个源操作数有不同符号，或绝对值都较小，可以保证更多次累加运算不产生溢出。

一般而言，用与源操作数相同字长的数据类型来保存累加和是非常危险的。通常的选择是在计算过程(循环)内用较长的数据类型保存和数，最后根据具体情况选取适当字长。总之，应根据源操作数及运算次数，谨慎选择数据类型和运算方法，防止溢出。

2．乘法运算指令

C67xx 公共指令集内的乘法指令以 16×16 位的硬件乘法器为基础，可分为以下两大类：

- 适用于整数乘法的指令(见表 2-18 中以 MPY 为首字母的 22 条指令)；
- 适用于 Q 格式数相乘的 3 条指令(SMPY/SMPYLH/SMPYHL)。

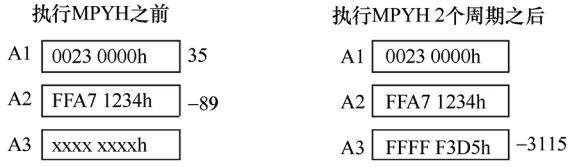
整数乘法的两个源操作数都是 16 位字长，目的操作数为 32 位的寄存器。根据源操作数为有/无符号数以及源操作数是寄存器的低/高半字，可以组合出 16 种不同的乘法指令。除了两个无符号源操作数相乘外，只要有一个源操作数是有符号数，其结果就认定是有符号数。由于目的操作数为 32 位，乘法指令不存在溢出问题。

【例 2-6】 整数乘法运算举例。

下面两条指令的源操作数形式相同，MPYH 指令认定两个源操作数为有符号数，其结果是有符号数。MPYHU 指令认定两个源操作数为无符号数，其结果是无符号数。

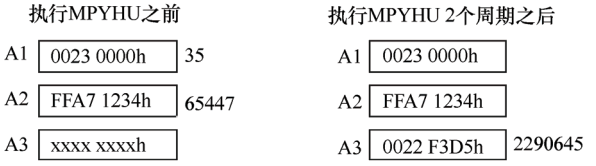
```
MPYH.M1    A1, A2, A3
```

运算结果如下所示：




```
MPYHU.M1    A1, A2, A3
```

运算结果如下所示：



在两个 Q 格式数相乘时，用有符号整数乘法指令，其结果有两个符号位，其小数点位置也需重新判定。C67xx 提供了一类带左移及饱和的乘法指令 SMPY/SMPYLH/SMPYHL，它将两个有符号数相乘的结果左移 1 位，使之只有 1 位符号位。如果原来是两个 Q0 格式数，则该类乘法指令运算结果为 Q1 格式数。

SMPY/SMPYLH/SMPYHL 指令有一个特殊情况，那就是当两个源操作数都是 8000h 时，按上述处理将出现错误。C67xx 规定，当 SMPY/SMPYLYLH/SMPYHL 指令的两个源操作数都是 8000h 时，则将运算结果置为 32 位有符号数的最大正值，并将 CSR 寄存器的饱和位(SAT)置位。

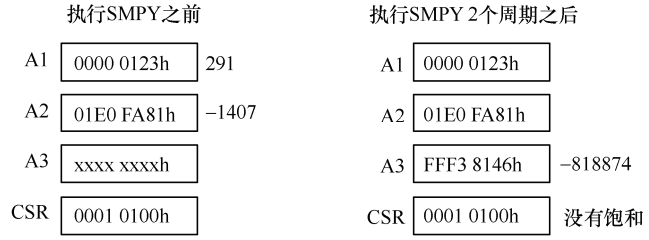
【例 2-7】 SMPY 类指令的例子。

本例中如果用不带左移的乘法指令 MPY，则 A3 寄存器的内容应为 0xFFF9C0A3 (−409437)。采用带左移的乘法指令，结果是 0xFFF38146 (−818874)。

A1 和 A2 本来是 Q0 定标，因此乘积也应是 Q0 定标，由于 SMPY 指令自动将乘积左移一位，使结果 A3 成为 Q1 定标，所以结果就是−409437×2 =−818874。在使用 A3 时必须将其右移一位，即除以 2。

```
SMPY.M1    A1, A2, A3
```

运算结果如下所示：



3．其他算术运算类指令

C67xx CPU 还提供了 ABS(取绝对值)和 SAT(将 40 位长型有符号数转换为 32 位有符号数)等算术运算指令。SAT 指令在转换时，如果被转换数超出了 32 位有符号数所能表示的范围，则取 32 位的饱和值，并将 CSR 寄存器中的 SAT 位置 1。

2.3.5 逻辑及字段操作类指令

1．逻辑运算指令

C67xx CPU 支持典型的布尔代数运算指令 AND、OR 和 XOR。这类指令都是对两个操作数按位做“与”、“或”和“异或”运算，并将结果写入目的寄存器。C67xx 还提供求补码的指令 NEG，可用于对 32 位、40 位有符号数求补码。

2．移位指令

C67xx 公共指令集共有 4 种移位指令：算术左移指令 SHL、算术右移指令 SHR、逻辑右移(无符号扩展右移)指令 SHRU 和带饱和的算术左移指令 SSHL。图 2-13 所示为一个长型数据(40 位)执行 SHR 指令操作的示意图。

```
SHR    src2, src1, dst
```

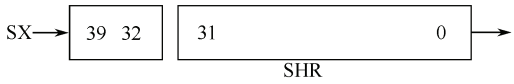


图 2-13　SHR 指令对 40 位长型数据的操作

在 SHR 指令里，源操作数 src2 可以是 32 或 40 位有符号数，src1 的低 6 位指定右移位数，将 src2 右移，得数放到 dst 中。移位时，最高位按符号位扩展。其他移位指令格式与 SHR 相似，其中 SHL 指令在左移时用 0 填补低位，SHRU 指令把 src2 视作无符号数，右移时，用 0 填补最高位。

算术左移指令 SHL 在左移过程中，其符号位有可能改变。带饱和的算术左移指令 SSHL 可用于防止这个问题产生。在 SSHL 指令情况下，src2 是 32 位有符号数，只要被 src1 指定移出的数位中有一位与符号位不一致，它就用与 src2 同符号的极大值填入 dst，并使 CSR 寄存器中的 SAT 位置位。

3．字段操作指令

在 C67xx 公共指令集内对定点数的字段操作指令可分为 3 类：

- 字段清零/置位指令 CLR/SET；
- 带符号扩展与无符号扩展的字段提取指令 EXT/EXTU；
- LMBD 与 NORM 指令。

CLR、SET、EXT 和 EXTU 等指令的操作及指定操作域的方法相似。下面以 CLR 指令为例说明。CLR 指令的汇编语言有两种形式：

```
CLR(.S)    src2, csta, cstb, dst
CLR(.S)    src2, src1, dst
```

前一种形式以常数 csta、cstb 指定位操作域，后一种形式以 src1 的位 0～4 代替 cstb，以 src1 的位 5～9 代替 csta，这样用 src1 可动态地指定操作域。CLR 指令操作内容可用图 2-14 表示，在指定范围内的位全被清零。

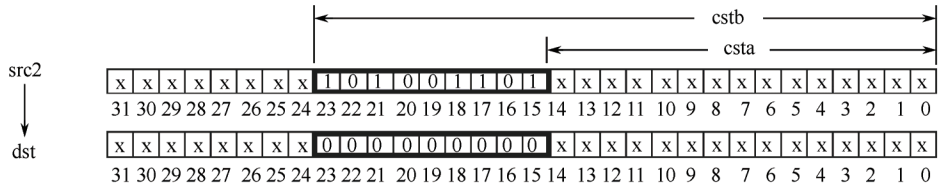


图 2-14　CLR 指令操作内容

LMBD 指令的汇编格式为：

```
LMSD[.L]　src1,src2,dst
```

其功能是寻找 src2 中与 src1 最低位(LSB)相同的最高位位置，并将其值(从左计起)返回送入 dst。图 2-15 是两个例子，两例都假设 src1 的 LSB 是 0，src2 分别如图 2-15(a)、(b)所示。在图 2-15(a)情况下，返回值为 0；在图 2-13(b)情况下，返回值为 32。

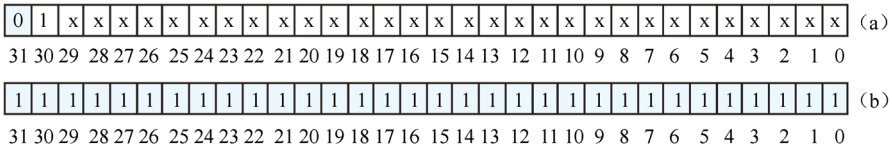


图 2-15　LMBD 指令举例

NORM 指令用来检测源操作数中有多少个冗余的符号位。其指令格式为：

```
NORM(.L)　src2,dst
```

图 2-16 中 NORM 指令的执行结果为 3。

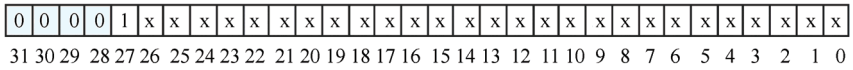


图 2-16　NORM 指令举例

4．比较及判别指令

CMPEQ/CMPGT(U)/CMPLT(U)指令用于比较两个有/无符号数是否相等、大于和小于。若为真，则目的寄存器置 1；反之，目的寄存器置 0。

注意：比较有符号数与无符号数大小的指令是不同的，要根据被比较对象的不同来选择不同的指令。

2.3.6　搬移类指令

搬移指令共有 3 种：

- MV 指令用于在通用寄存器之间传送数据；
- MVC 指令用于在通用寄存器与控制寄存器之间传送数据，此指令只能使用.S2 功能单元；
- MVK 类指令用于把 16 位常数送入通用寄存器。可使用 MVKL 指令向寄存器送入低 16 位常数，用 MVKH 或 MVKLH 指令向寄存器的高 16 位送数。

注意：必须先使用 MVKL 指令，再使用 MVKH 或 MVKLH 指令向寄存器搬移数据。

【例 2-8】 数据搬移指令的例子。