第 5 章 TMS320DM6437 流水线与中断

CHAPTER 5

5.1 流水线

5.1.1 流水线概述

在冯·诺依曼结构中，程序中各条机器指令都是按照顺序执行的，只有在前一条指令的各过程段都全部完成后，才从存储器取出下一条指令，即机器各部件在某些周期内进行操作，而在某些周期内是空闲的，导致机器各部分的利用率不高。如果用控制器进行适当调度，可以让机器的各个部件在每个周期内都在工作，这样可以提高计算机各功能部件的工作效率和计算机的运行速度，这就需要流水线(Pipeline)技术。

流水线技术是指在程序执行时多条指令重叠进行操作的一种准并行处理技术。它是将一个重复的过程分解为若干个子过程，每个子过程由专门的功能部件来实现。流水线中的每个子过程及其功能部件称为流水线的级或段，段与段相互连接形成流水线。流水线的段数称为流水线的深度。把多个处理过程在时间上错开，依次通过各功能段，这样，每个子过程就可以与其他的子过程并行进行。对微处理器的每个部件来说，每隔 1 个时钟周期即可进入一条新指令，这样在同一时间内，就有多条指令交叠在不同部件内处理，使 CPU 运算速度提高。但这种流水线工作方式控制较为复杂，很难全速运行。

TMS320DM6437 的 DSP 具有独特的特点，它可以通过消除流水线交错，简化流水线的控制，并通过增加流水线消除程序提取、数据访问和乘法运算等传统结构的瓶颈，提高单周期的吞吐量。并且，利用流水线提供的灵活性可以简化编程，提高性能。

5.1.2 流水线操作

流水线操作以 CPU 周期为单位，一个 CPU 周期是指特定的执行包在流水线特定阶段的时间。CPU 周期的边界总是发生在时钟周期的边界，随着节拍代码流流经各个部件，各个部件根据指令代码进行不同处理。一个指令的取出和执行过程可以分为多个阶段。TMS320DM6437 的 DSP 指令集流中所有的指令都通过取指(Fetch)、译码(Decode)和执行(Execute)3 个阶段。各阶段的任务如下。

(1) 取指：取出一条指令送到指令寄存器。所有指令的取指阶段都有 4 个节拍，即 PG、PS、PW、PR 节拍，每个节拍的具体功能如下。

① PG 程序地址产生(Program Address Generate)：CPU 上取指包的地址确定。

② PS 程序地址发送(Program Address Send)：取指包的地址送至内存。

③ PW 程序访问等待(Program Access Ready Wait)：访问程序存储空间。

④ PR 程序取指包接收(Program Fetch Packet Receive)：取指包送至 CPU 边界。

TMS320DM6437 的 DSP 取指阶段的每个节拍采用 8 个字的取指包。四个节拍从左到右依次进行，所有的这 8 个字同时通过 PG、PS、PW 和 PR 进行取指。PR 取指包有四个执行包，PW 和 PS 各包括两个执行包，PG 包含八个指令的一个执行包。流水线取指级的四个节拍功能如图 5-1 所示。

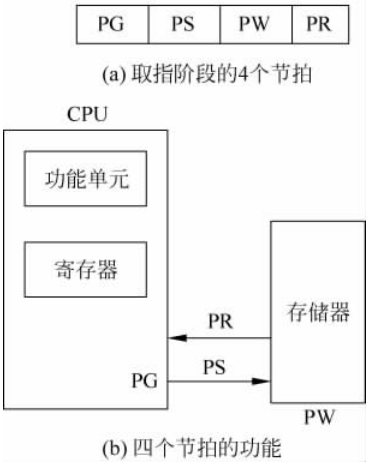

图 5-1　流水线取指级的四个节拍功能图

(2) 译码：对指令操作码进行译码，读取操作数。所有指令译码阶段都包括 2 个节拍，即 DP 和 DC 节拍。每个节拍的具体功能如下。

① DP 指令分配(Instruction Dispatch)：确定取指包的下一个执行包，并将其送至适当的功能单元准备译码。

② DC 指令译码(Instruction Decode)：指令在功能单元进行译码。

在流水线的 DP 阶段，取指包被分为执行包。执行包包括一个指令或两到八个平行指令。且在此阶段，执行包的指令被分配合适的功能单元。在 DC 阶段，源寄存器、目标寄存器和相关路径被解码，以执行功能单元的指令。

(3) 执行：根据操作码的要求，完成指令规定的操作，并把运算结果写到指定的存储或缓冲单元中。流水线的执行阶段节拍的数量不同，这取决于指令的类型。

流水线的执行部分被分为 5 个节拍。大多数 DSP 指令是单周期的，所以它们只有一个执行节拍(E1)。只有少数指令需要多个执行节拍。不同类型的指令需要不同数量的这些节拍来完成执行。这些流水线的阶段对理解 CPU 周期边界设备状态具有重要作用。流水线执行级的 5 个节拍如下。

① 执行节拍 E1：测试指定执行条件及读取操作数，对所有的指令适用。对于读取和存储指令，假定指令的条件被评估为真时，地址产生，其修正值写入寄存器；若指令的条件为假，则指令在 E1 后不写入任何结果或进行任何流水线操作；对于转移指令，程序转移目的地址取指包处于 PG 节拍；对于单周期指令，结果写入寄存器；对于双精度(DP)比较指令、ADDDP 和 MPYDP 等指令，读取源操作数的低 32 位；对于其他指令，读取操作数；对于双周期双精度(DP)指令，结果的低 32 位写入寄存器。

② 执行节拍 E2：读取指令的地址送至内存。存储指令的地址和数据送至内存。对结果进行饱和处理的单周期指令，若结果饱和，置 SRC 的 SAT 位；对于单个 16×16 乘法指令、乘法单元和非乘法操作指令，结果将写入寄存器文件。TMS320C64x 的 M 单元的非乘法操作指令，对于 DP 比较指令和 ADDDP/SUBDP 指令，读取源操作数的高 32 位；对于 MPYDP 指令，读取源操作数 1 的低 32 位和源操作数 2 的高 32 位；对于 MPYI 和 MPYID 指令，读取源操作数。

③ 执行节拍 E3：进行数据存储空间访问。对结果进行饱和处理的乘法指令在结果饱和时置 SAT 位；对于 MPYDP 指令读取源操作数 1 的高 32 位和源操作数 2 的低 32 位；对

于 MPYI 和 MPYID 指令，读取源操作数。

④ 执行节拍 E4：对于读取指令，把所读的数据送至 CPU 边界；对于乘法扩展，结果将被写入寄存器；对于 MPYI 和 MPYID 指令，读取源操作数；对于 MPYDP 指令，读取源操作数的高 32 位；对于 4 周期指令，结果写入寄存器；对于 INTDP 指令，结果的低 32 位写入寄存器。

⑤ 执行节拍 E5：对于读取指令，把所读的数据写入寄存器；对于 INTDP 指令，结果写入寄存器。

TMS320DM6437 中所有指令均按照以上 3 级流水线运行，具体流水线结构如图 5-2 所示。3 级流水线各节拍的功能描述如表 5-1 所示。

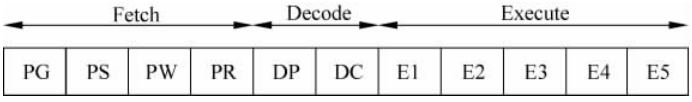

图 5-2　TMS320DM6437 3 级流水线

表 5-1　3 级流水线各节拍功能描述

流水线阶段		描　　述
Fetch	PG	Program Address Generate，程序地址产生
	PS	Program Address Send，程序地址发送
	PW	Program Access Ready Wait，程序访问等待
	PR	Program Fetch Packet Receive，程序取指包接收
Decode	DP	Instruction Dispatch，指令分派
	DC	Instruction Decode，指令译码
Execute	E1	执行阶段的第一个节拍
	…	…

流水线流程图如图 5-3 所示，连续的各个取值包都包含 8 条并行指令，各个指令包以每个时钟一个节拍的方式通过流水线。

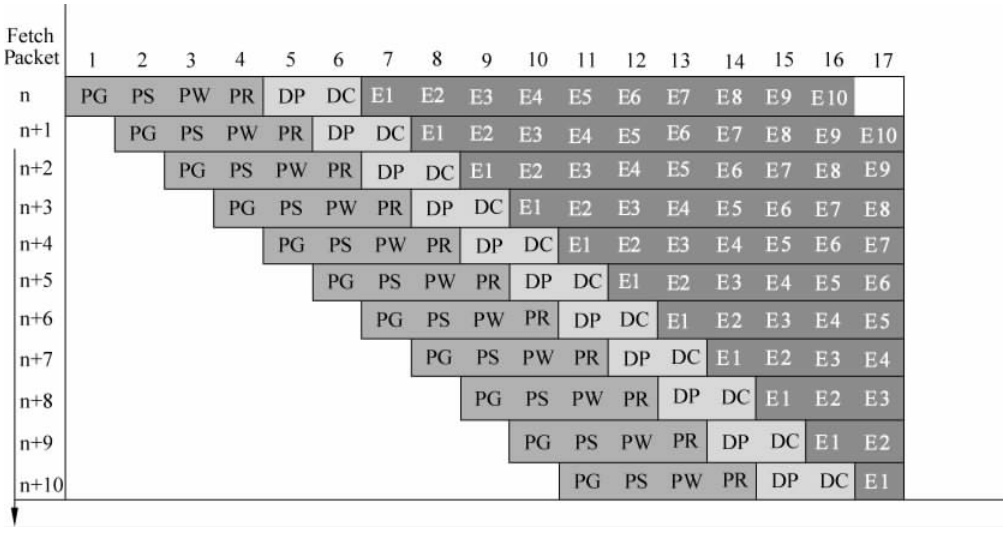

图 5-3　TMS320DM6437 流水线流程图

流水线的实现步骤如下：

(1) 从存储器中取出一条指令。

(2) 判断该指令是否支持并行执行，如果支持，则执行步骤(1)，不支持则执行步骤(3)。

(3) 对指令包中的每一条指令进行解析，解析过程中运用缓存机制提高指令的解析效率。

(4) 判断延时队列中是否有指令，如果有指令则执行步骤(5)，没有则执行步骤(6)。

(5) 对延时队列中的每一条指令进行如下操作：将指令的延时间隙减 1；此时，如果延时间隙为 0，则执行指令在 En 节拍的操作。

(6) 判断指令包中是否有未执行的指令，如果有，则执行操作(7)，没有则结束。

(7) 对执行包中的每一条指令进行如下操作：如果指令有延时间隙，则执行步骤(6)，如果没有执行延时，则执行指令 E1 节拍的操作并跳到步骤(6)。

(8) 执行指令 E1 节拍的操作，指令的延时间隙减 1，将指令加入到延时队列中，并跳到步骤(6)。

5.1.3 指令对流水线性能的影响

TMS320C64x+系列的 DSP 的流水线操作执行指令分为 7 种，分别是单周期指令(Single-cycle Instruction)、双周期或多周期指令(Two-cycle or Multiply Instruction)、存储指令(Store Instruction)、扩展乘法指令(Extended Multiply Instruction)、读入指令(Load Instruction)、分支指令(Branch Instruction)和 NOP 指令。

单周期指令在流水线的 E1 节拍完全执行；双周期或多周期指令在 E1 和 E2 节拍完成；存储指令要在 E1 到 E3 节拍完成其操作；扩展乘法指令是使用 E1 和 E4 节拍；读入指令需要用 E1 到 E5 节拍完成；分支指令只使用 E1 节拍。其中，当要将同一指令读入和存储到相同的内存位置时，有如下规则(i 为周期)：

(1) 在存储之前执行读入指令，则以前的值被读入，新的值被存储。

```
i     LDW
i+1   STW
```

(2) 读入指令前执行存储指令，则新的值被存储和读入。

```
i     STW
i+1   LDW
```

(3) 当读入指令和存储指令同时执行，以前的值首先被读入，新的值被存储，但是都在同一节拍。

```
i   STW
i   ||LDW
```

TMS320DM6437 指令集中每一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。由于指令复杂程度的不同，各种指令的执行周期也不相同，多周期指令具有延时间隙。延时间隙在数量上等于从读取指令的源操作数到可以访问执行的结果所需要的指令周期数。对于单周期指令类型的指令如 ADD 而言，源操作数读取数据在第 i 个指令周期执行，计算结果在第 i+1 个指令周期可以被访问，等效于无延时。就乘法指令 MPY 来说，如源操作数读取数据在第 i 个指令周期执行，计算结果在第 i+2 个指令

周期才能被访问，延时周期为1。对于转移类指令，如果是转移到标号地址的指令或是中断IRP和NRP引起的转移，没有读操作。对于Load指令，在第i个周期读地址指针并且在该周期内修改基地址，在第i+4个周期写入寄存器。

CPU运行时，每次取八条指令组成一个指令包，取指包一定在地址的256位边界定位。每条指令的最后一位是并行标志位P，P标志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从低地址到高地址依次判读P标志位：如果为1，则该指令将与下一条指令并行执行；如果为0，则下一条指令要在本指令执行完以后才能执行。指令执行过程会占用一定的资源，并行执行的指令所需资源不能冲突。造成冲突的因素有：使用相同的功能单元、使用交叉通路、使用长型数据、对寄存器的读取与存储等。表5-2是除NOP指令外的六种类型指令的每个执行阶段中发生的操作的映射。

表5-2　六种类型指令的每个执行阶段发生的操作

执行阶段	指令类型					
	单周期指令	双周期或多周期指令	存储指令	扩展乘法指令	读入指令	分支指令
E1	计算结果并写入寄存器	读取操作数，开始计算	计算地址	读取操作数，开始计算	计算地址	PG中的目标代码
E2		计算结果，写入寄存器	把地址和数据发送给存储器		把地址发送给存储器	
E3			通过存储器		通过存储器	
E4				将结果写入寄存器	发送数据返回到CPU中	
E5					将数据写入寄存器	

如果流水线中的指令相互独立，则可以充分发挥流水线的性能。但在实际中，指令间可能会相互依赖，这会降低流水线的性能。相邻或相近的两条指令因存在某种关联，后一条指令不能在原指定的时钟周期开始执行，造成流水线中出现“阻塞”的情况。

1. 在一个取指包中有多个执行包的流水线操作

一个FP包含8条指令，可分成1～8个EP，每个EP是并行执行的指令，每条指令在1个独立的功能单元内执行。当一个取指包包含多个执行包时，这时将出现流水线阻塞。

例如，取指包n包含3个执行包，n+1～n+6只有一个执行包。n～n+6的所有指令代号依次按A、B、C……排列，取指包n的8条指令中，A和B，C、D和E，F、G和H分别形成一个执行包。

```
  instruction A; EP k          FP n
|| instruction B;

  instruction C; EP k + 1      FP n
|| instruction D;
|| instruction E;
```

```
  instruction F; EP k + 2        FP n
|| instruction G;
|| instruction H;

  instruction I; EP k + 3        FP n + 1
|| instruction J;
|| instruction K;
|| instruction L;
|| instruction M;
|| instruction N;
|| instruction O;
|| instruction P;
```

这时，取指包包含 3 个执行包的流水线操作，且取指包在指令周期 1～4 时，通过取指级的 4 个节拍，同时 1～4 的每个指令周期都有 1 个新取指包进入 PG 节拍；在第 5 个指令周期的 DP 节拍，CPU 扫描 FPn 的 P 位，检测出有 3 个执行包 EPk～EPk＋2，迫使流水线阻塞，允许 EPk＋1 和 EPk＋2 在第 6 和 7 个指令周期进入 DP 节拍，一旦 EPk＋2 进入 DC 节拍，流水线阻塞也被释放；另外，FPn＋1～FPn＋4 在第 6 和 7 个指令周期被阻塞，FPn＋5 在第 6 和 7 个指令周期也被阻塞，直到第 8 个指令周期后才进入 PG 节拍：第 8 个指令周期后流水线将连续操作，直至又有多个执行包进入 DP 节拍或有中断发生。图 5-4 是程序执行时的流水线阻塞图。

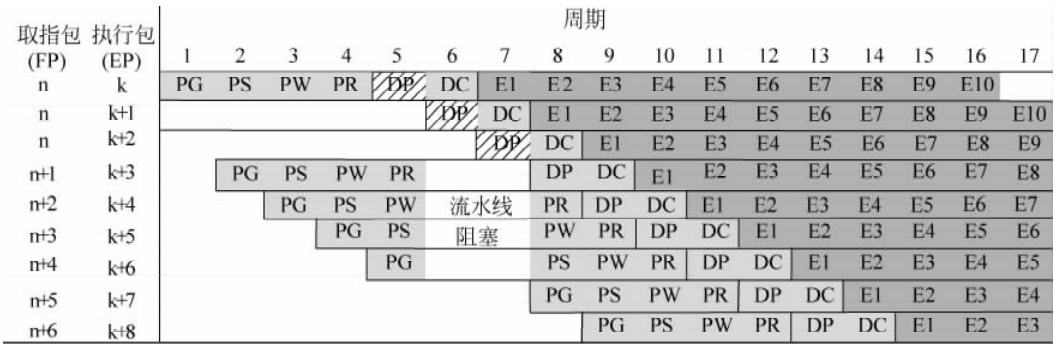

图 5-4　流水线阻塞图

2. 多周期 NOP 指令对流水线运行的影响

一个 FP 中的 EP 的数量是影响指令通过流水线运行方式的一种因素，另一种因素就是 EP 中指令的类型，例如 NOP 指令。NOP 是不使用功能单元的空操作，空操作的指令周期数由该指令的操作数决定，如果 NOP 与其他指令并行使用，将给其他指令加入额外的延迟间隙。

图 5-5 表示一个单周期 NOP 指令与其他代码在一个执行包中。LD、ADD 和 MPY 指令的结果在适当指令周期是可用的，NOP 指令对执行包无影响。图 5-6 用一个多周期 NOP 5 代替单周期 NOP 指令。NOP 5 将产生除它的执行包内部指令操作之外的空操作。在 NOP 5 指令周期完成之前，任何其他指令不能使用 LD、ADD 和 MPY 指令的结果。

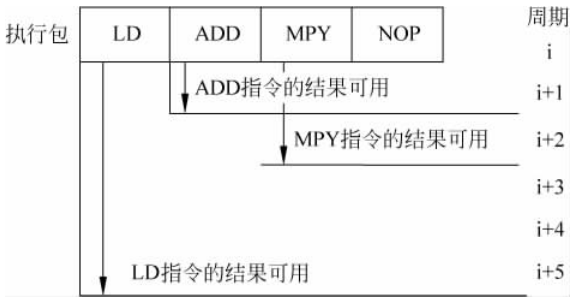

图 5-5　NOP 与其他代码在一个执行包

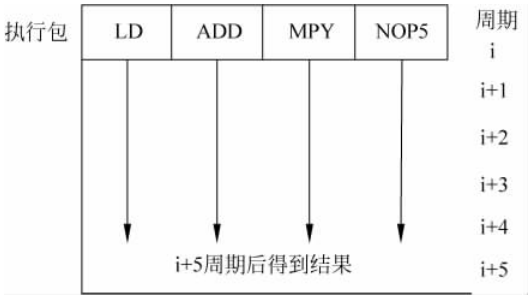

图 5-6　用多周期 NOP 指令代替单周期 NOP 指令

另外，跳转指令可以影响多周期 NOP 指令的执行，当一个跳转指令延迟间隙结束时，多周期 NOP 指令不管是否结束，这时跳转都将废弃多周期 NOP 指令。图 5-7 中，在发出跳转指令的 5 个延迟间隙后，跳转目标进入执行操作。若 EP1 中没有跳转指令，跳转延迟间隙为指令周期 2 至指令周期 6，一旦目标代码在指令周期 7 到达 E1 阶段，就会立即执行目标代码。

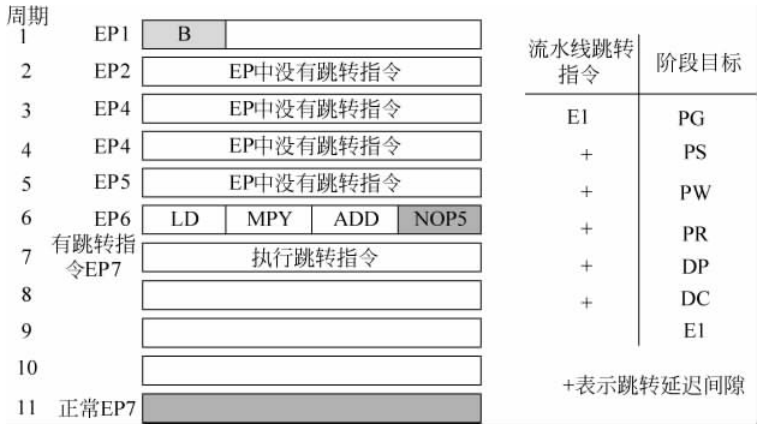

图 5-7　跳转指令对多周期 NOP 指令的影响

5.1.4　存储器对流水线性能的影响

TMS320DM6437 片内为哈佛结构，即存储器分为程序指令存储空间和数据存储空间，是一种并行体系结构。它将程序和数据存储在不同的存储空间中，每个存储器独立编址、独

立访问。与两个存储器相对应的是体系中设置了程序总线和数据总线两条总线，从而使数据的吞吐率提高了数倍。这种结构允许在一个机器周期内同时获得指令字（来自程序存储器）和操作字（来自数据存储器），从而提高了执行速度，提高了流水线数据的吞吐率。又由于程序和数据存储器在两个分开的空间中，因此取指和执行能完全重叠。为了进一步提高运行速度和灵活性，许多芯片在哈佛结构的基础上作了改进，一是允许数据存放在程序存储器中，并被算术运算指令直接使用，增强了芯片的灵活性；二是将指令暂存在高速缓冲器中，当执行此指令时，不需要再从存储器中读取指令，节省了取指令周期。根据内存类型和完成访问所需的时间，流水线可能会中断，以确保数据和指令的正确协调。数据读取和程序读取在流水线中有相同的操作，它们使用不同的节拍完成操作。由于数据读入和程序取指，内存访问被分为多个阶段，这保证了 TMS320C64x 系列的 DSP 能进行内存访问。表 5-3 为数据读取和指令读取的流水线操作。数据读取和指令读取在内部存储器中以相同的速度进行，且执行同种类型操作。

表 5-3　数据读取和指令读取的流水线操作

操　　作	读取指令阶段	加载数据阶段
计算地址	PG	E1
将地址发送到存储器	PS	E2
存储器读/写	PW	E3
指令读取：在 CPU 边界得到取指包 数据读取：在 CPU 边界得到数据	PR	E4
指令读取：将指令发送给功能单元 数据读取：将数据发送到寄存器	DP	E5

存储器对流水线性能的影响主要为存储器阻塞，即当存储器没有做好响应 CPU 访问的准备时，流水线将产生存储器阻塞。对于程序存储器，存储器阻塞发生在 PW 节拍，对于数据存储器则发生在 E3 节拍。存储器阻塞会使处于该流水线的所有节拍延长 1 个指令周期以上，从而使执行增加额外的指令周期。程序执行的结果等同于是否有存储器阻塞发生。具体如图 5-8 所示。

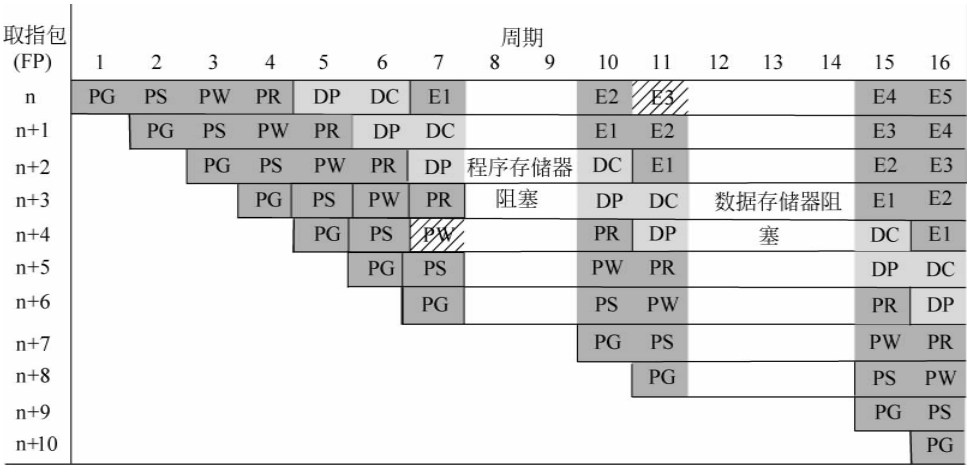

图 5-8　存储器阻塞图

5.2 DSP 的中断系统

5.2.1 中断的基础知识

1. 中断类型和优先级

中断是由硬件或软件驱动的信号，为使 CPU 具有对外界异步事件的处理能力而设置的。中断信号使 DSP 暂停正在执行的程序，接受并响应中断请求，进入中断服务程序。而 CPU 则要完成当前指令的执行，并冲掉流水线上还未解码的指令。中断源可以在芯片内或片外，如定时器、模数转换器或其他外围设备。中断过程包括保存当前进程的上下文、完成中断任务、恢复寄存器和进程上下文、恢复原始进程。中断一旦被正确启用，CPU 将开始处理中断并将程序流重新定向到中断服务程序。通常 DSP 工作在包含多个外界异步事件环境中，当这些事件发生时，DSP 应及时执行这些事件所要求的任务。TMS320DM6437 的 CPU 有 3 种类型中断，即 RESET（复位）、不可屏蔽中断（NMI）和可屏蔽中断（INT4～INT15）。3 种中断的优先级别不同，复位 RESET 具有最高优先级，不可屏蔽中断为第二优先级，响应信号为 NMI 信号，最低优先级中断为 INT15。中断优先级别见表 5-4。

表 5-4 中断优先级别

优 先 级	中 断 类 型
最高级	RESET
第二优先级	NMI
	INT4
	INT5
	INT6
	INT7
	INT8
	INT9
	INT10
	INT11
	INT12
	INT13
	INT14
最低级	INT15

（1）复位 RESET 具有最高级别中断，复位中断时正在执行的指令中止，所有的寄存器返回到默认状态。复位是低电平有效信号，必须保证低电平 10 个时钟周期，然后再变高才能正确重新初始化 CPU，这点较为特殊。其他的中断则是在转向高电平的上升沿有效。复位中断服务的取指包必须位于特定设备的特定地址。此外，复位中断不受分支指令的影响。

（2）不可屏蔽中断（NMI）：不可屏蔽中断优先级为 2，它通常用来向 CPU 发出严重硬件问题的警报。出现在 NMI 线上的请求，不受中断标志位 IF 的影响，在当前指令执行完以后，CPU 立即无条件响应。为实现此中断，在中断使能寄存器中的不可屏蔽中断使能位（NMIE）必须置 1。NMIE 为 0 时，所有可屏蔽中断（INT4～INT15）均被禁止。

（3）可屏蔽中断（INT4～INT15）：可被 CPU 通过指令限制某些设备发出中断请求的中断。有 12 个可屏蔽中断，它们被链接到芯片外部或片内外设，也可由软件控制或者不用。I/O 设备发出的所有中断都可以产生可屏蔽中断，受标志位 IF 的影响，根据中断循环标志的设置来判断 CPU 是否响应中断请求。中断发生时将中断标志寄存器（IFR）的相应位置 1。

另外，除了以上三种，还有一种中断响应信号，IACK 和 INUMx 信号用来通知 C6000 的片外硬件：在 CPU 内有一个中断已经发生且正在进行处理时，会有 IACK 信号指出 CPU 已经开始处理一个中断，INUMx 信号指出正在处理的是哪一个中断。例如：

```
INUM3 = 0(MSB)
INUM2 = 1
```

```
INUM1 = 1
INUM0 = 1(LSB)
```

这些信号提供的 4bit 的数值是 0111，因此 INT7 被中断。

2. 中断服务表(IST)

IST 是包含中断服务代码的取指包的一个地址表。当 CPU 开始处理一个中断服务时，它要参照 IST 进行。IST 包含 16 个连续取指包，每个中断服务取指包都包含 8 条指令。图 5-9 给出了 IST 的地址和内容。

由于每个取指包都有 8 条 32 位指令字，故中断服务表内的地址以每个地址 20h 增长。

3. 中断服务取指包(ISFP)

ISFP 是用于服务中断的取指包，当中断服务程序很小时，可以把它放在一个单独的取指包内，如图 5-10 所示。其中，为了中断结束后能够返回主程序，FP 中包含一条跳转到中断返回指针所指向地址的指令。接着是一条 NOP 5 指令，它使跳转目标能够有效地进入流水线的执行级。若无这条指令，CPU 将会在跳转前执行下一个 ISFP 中的 5 个执行包。

000h	RESET ISFP
020h	NMI ISFP
040h	Reserved
060h	Reserved
080h	INT 4 ISFP
0A0h	INT 5 ISFP
0C0h	INT 6 ISFP
0E0h	INT 7 ISFP
100h	INT 8 ISFP
120h	INT 9 ISFP
140h	INT 10 ISFP
160h	INT 11 ISFP
180h	INT 12 ISFP
1A0h	INT 13 ISFP
1C0h	INT 14 ISFP
1E0h	INT 15 ISFP

图 5-9　中断服务表

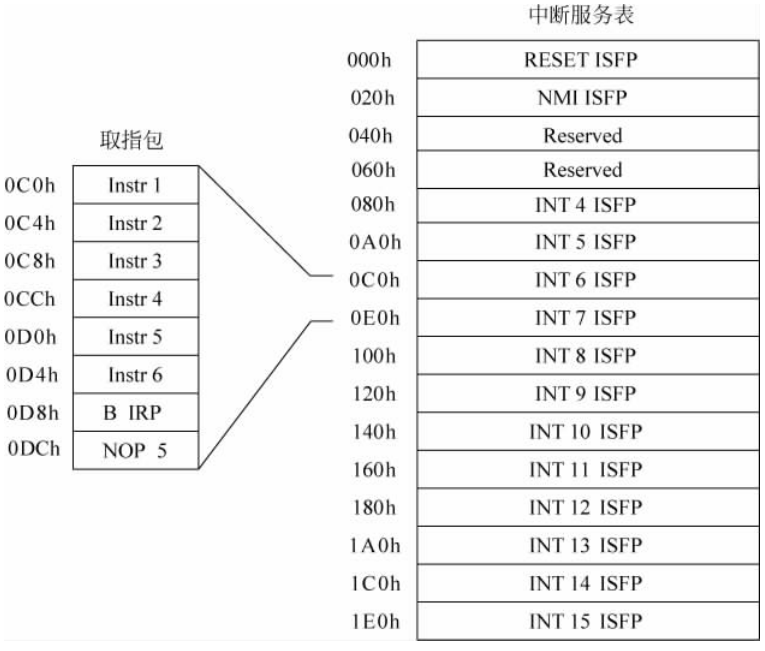

图 5-10　单独的中断取指包

若中断服务程序太长，不能放在单一的 FP 内，则需要跳转到另外的中断服务程序的位置上，如图 5-11 所示。

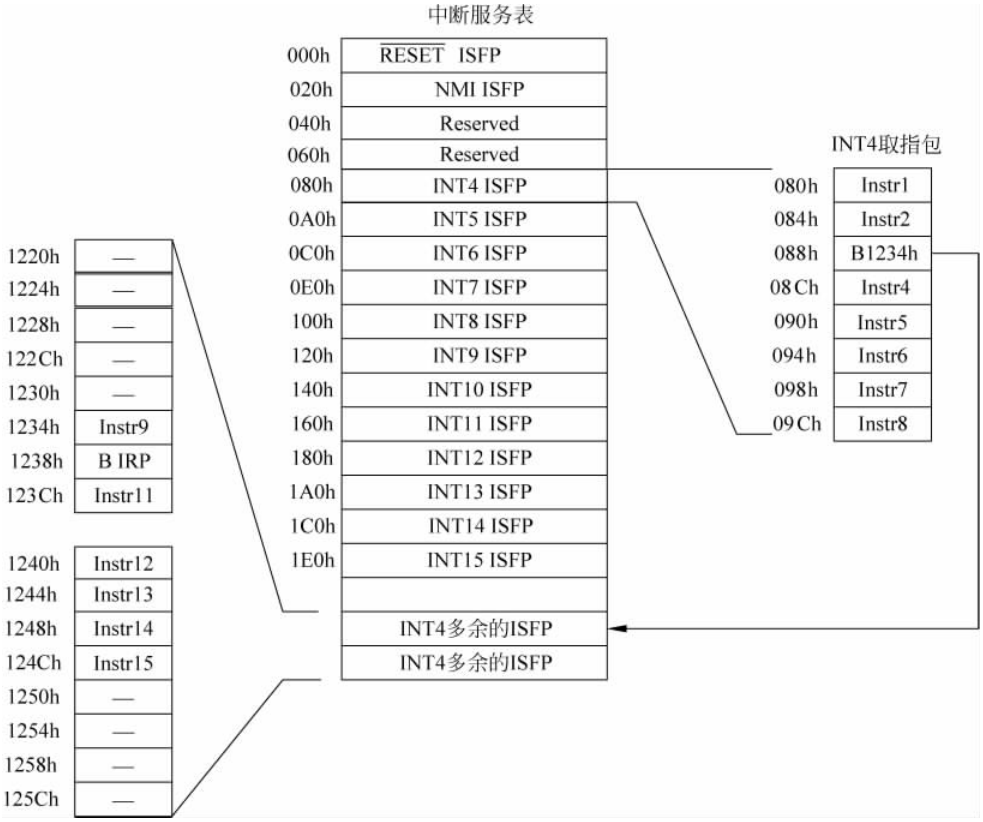

图 5-11　中断服务程序过长时的处理

例如，INT4 的中断服务程序太长而将部分程序放在以地址 1234h 开始的内存内。

在 INT4 的 ISFP 内有一条跳转到 1234h 的跳转指令，因跳转有 5 个延迟间隙，故将 B 1234h 放在了 ISFP 中间，尽管 1220h～1230h 与 1234h 指令并行，但 CPU 不执行 1220h～1230h 内的指令。

4. 中断服务表指针寄存器(ISTP)

ISTP 用于确定中断服务程序在中断服务表中的地址。ISTP 中的 ISTB 确定 IST 的地址的基值，HPEINT 确定当前响应的中断，并给出这一特定中断取指包在 IST 中的位置。图 5-12 标出 ISTP 各字段的位置，表 5-5 给出了这些字段的描述。

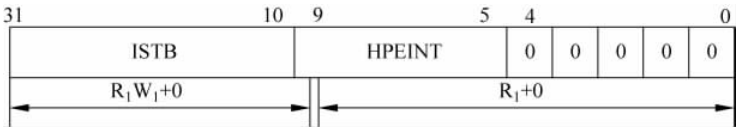

图 5-12　ISTP 格式描述

表 5-5 ISTP 字段描述

字段	字 段 名 称	描　　述
0～4		置 0，取指包必须界定在 8 个字(32 字节)范围内
5～9	HPEINT	最高级使能中断 该字段给出 IER 中使能的最高级挂起中断的序号(相应为 IFR 的位数)，因此可利用 ISTP 手动跳转到最高级使能中断，如果没有中断挂起和使能，HPEINT 的值为 00000b。这个相应的中断不需要靠 NMIE(除非是 NMI)或 GIE 使能
10～31	ISTB	IST 的基地址 复位时置 0，这样在开始时 IST 必须放在 0 地址，复位后，可对 ISTB 写新的数值重新定位 IST。如果重新定位，第一个 ISFP(对应 RESET)从不执行，因为复位使 ISTB 置 0

复位取指包必须放在地址为 0 的内存中，而 IST 中的其余取指包可放在符合 256 字边界调整要求的程序存储单元的任何区域内。IST 的位置由中断服务表基值(ISTB)确定。

下面给出了一个中断服务表重新定位的例子，图 5-13 给出了新的中断服务表基值 ISTB 与 IST 位置的关系。

(1) 中断服务表重新定位到 800h。

① 将 0h～200h 的 IST 复制到 800h～A00h 的内存中。

② 将 800h 写入 ISFP 寄存器：

MVK 800h，A2

MVC A2，ISTP

ISTP＝800h＝1000 0000 0000b

(2) 重新定位的 ISP 中，ISTP 如何将 CPU 指向正确的 ISFP。

① 假设：

IFR＝BBC0h＝1011 1011 1100 0000b

IER＝1230h＝0001 0010 0011 0001b

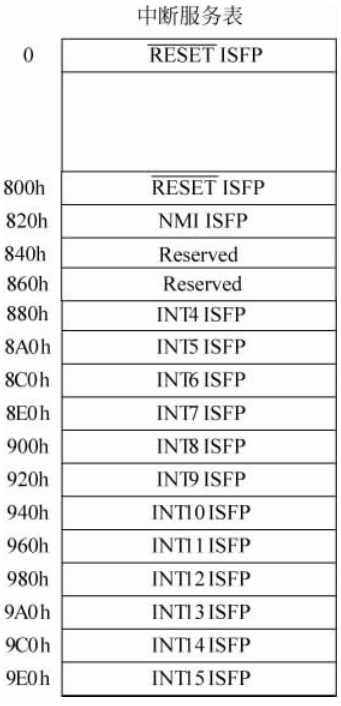

图 5-13　ISTB 与 IST 位置关系

② 启用中断等待：INT9 和 INT12。

在 IFR 中 1S 表明等待中断；在 IER 中 1S 表明中断已启用。与 INT12 相比 INT9 具有更高的优先级，所以 HPEINT 被编码为 INT9 的值，即 01001b。

HPEINT 对应 ISTP 的 9～5 位：

ISTP＝1001 0010 0000b＝920h＝INT9 的地址

5.2.2　中断控制寄存器

TMS320DM6437 芯片控制寄存器组中有下列 8 个寄存器涉及中断控制寄存器：

1. 控制状态寄存器(CSR)

控制全局使能或禁止中断。该寄存器用于标识一些状态，其中包括全局中断使能、高速

缓冲存储器的控制位和其他各种控制位和状态位。在控制状态寄存器中包含控制中断的两个域，即 GIE 和 PGIE。全局中断使能(GIE)允许通过控制单个位的值来使能或禁止所有的可屏蔽中断。GIE＝1 使能可屏蔽中断使之能够进行处理；GIE＝0 禁止可屏蔽中断使之不能处理。PGIE 在中断任务状态寄存器中与 GIE 具有一样的物理地址，PGIE＝1 从中断返回后将启用中断；PGIE＝0 中断返回后禁止中断。CSR 的格式如图 5-14 所示，字段描述如表 5-6 所示。

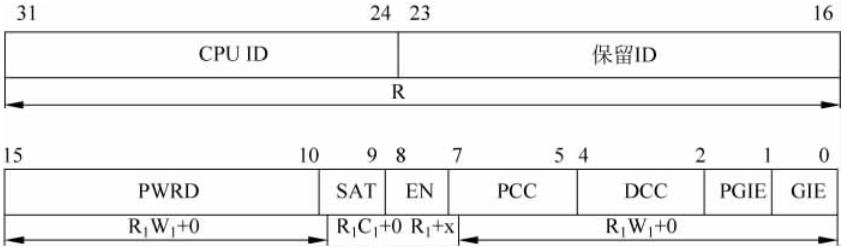

图 5-14　CSR 格式

表 5-6　CSR 字段描述

字段	字段名称	描　　述
0	GIE	全局中断使能。全局使能或禁止所有的可屏蔽中断 GIE＝0，全局禁止可屏蔽中断 GIE＝1，全局使能可屏蔽中断
1	PGIE	先前的 GIE，当一个中断发生时，PGIE 保存 GIE 的值，当从中断返回后要使用这个值

全局禁用可屏蔽中断的代码：

```
MVC   CSR,B0;
AND   -2,B0,B0;
MVC   B0,CSR;
```

全局使能可屏蔽中断的代码：

```
MVC   CSR,B0;
OR    1,B0,B0;
MVC   B0,CSR;
```

2. 中断使能寄存器(IER)

该寄存器用于标识某个中断是使能还是禁止。使能或禁止单一的中断处理，在用户模式下无法访问 IER，格式如图 5-15 所示。IER 的最低位对应于复位，只可读不可写，复位总能被使能；NMIE＝0 时，禁止所有非复位中断；NMIE＝1 时，GIE 和相应的 IER 位一起控制 INT15～INT4 中断使能。对 NMIE 写 0 无效，只有复位或 NMI 发生时它才清 0；NMIE 置 1 靠执行 B NRP 指令和写 1 完成。

使能一个中断(INT9)的代码：

```
MVK   200h,B1;
MVC   IER,B0;
OR    B1,B0,B0;
```

图 5-15　IER 格式

```
MVC  B0,IER;
```

禁用一个中断(INT9)的代码：

```
MVK  FDFFh,B1;
MVC  IER,B0;
AND  B1,B0,B0;
MVC  B0,IER;
```

3. 中断标志寄存器(IFR)、中断设置寄存器(ISR)和中断清除寄存器(ICR)

该寄存器用于显示中断标志。给出有中断请求但尚未得到服务的中断，存在于等待中断的状态中，包括 INT4～INT15 和不可屏蔽中断的状态。发生中断时，在 IFR 相应的每个位设置为 1，否则清除相应的位为 0。如果想要检查中断的状态，可以用 MVC 指令读 IFR。图 5-16 所示为中断 IFR 寄存器的格式。

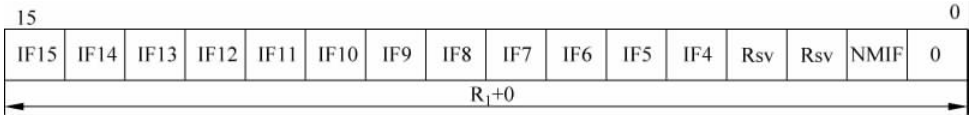

图 5-16　IFR 格式

设置一个中断(INT6)并读标志寄存器的代码：

```
MVK  40h,B3
MVC  B3,ISR
NOP
MVC  IFR,B4
```

清除一个中断(INT6)并读标志寄存器的代码：

```
MVK  40h,B3
MVC  B3,ICR
NOP
MVC  IFR,B4
```

4. 中断设置寄存器(ISR)

人工设置 IFR 中的标志位，使用它手动清除 IFR 中的可屏蔽中断(INT15～INT4)。该寄存器用于标识是否允许软件控制来设置中断。将 1 写入 ICR 中的任意位将导致 IFR 中相应的中断标志被清除；相反，将 0 写入则无影响。传入中断具有优先级，并覆盖所有对 ICR 的写入。此外，不能设置 ICR 中的任何位来影响 NMI 和 RESET。ISR 的格式如图 5-17 所示。

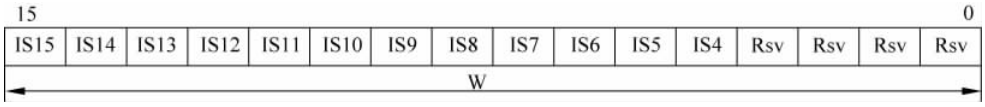

图 5-17　ISR 格式

5. 中断清零寄存器(ICR)

人工清除 IFR 中的标志位,使用它手动完成中断处理。该寄存器用于标识是否允许软件来清除中断。ICR 的格式如图 5-18 所示。

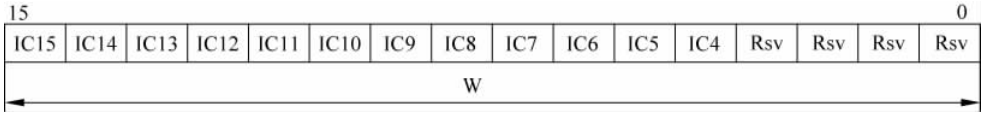

图 5-18　ICR 格式

另外,ISR 和 ICR 可用程序设置和清除 IFR 中的可屏蔽中断位,设置和清除操作不影响 NMI 和复位。

6. 不可屏蔽中断返回指针寄存器(NRP)

NRP 保存从不可屏蔽中断返回时的指针,该指针引导 CPU 返回到原来程序执行的正确位置。当 NMI 完成服务时,为返回到被中断的原程序中,在中断服务程序末尾必须安排一条跳转到 NRP 指令(B NRP)。

NRP 是一个 32 位可读写的寄存器,在 NMI 产生时,它将自动保存被 NMI 打断而未执行的程序流程中第 1 个执行包的 32 位地址。因此,虽然可以对这个寄存器写值,但任何随后而来的 NMI 中断处理将刷新该写入值。

从 NMI 返回的代码:

```
B    NRP;
NOP  5;     延迟间隙
```

7. 可屏蔽中断返回指针寄存器(IRP)

IRP 保存从可屏蔽中断返回时的指针,该指针引导 CPU 返回到原来程序执行的正确位置。当可屏蔽中断完成服务时,为返回到被中断的原程序中,在中断服务程序末尾必须安排一条跳转到 NRP 指令(B IRP)。

IRP 是一个 32 位可读写的寄存器,在可屏蔽中断产生时,它将自动保存被可屏蔽中断打断而未执行的程序流程中第 1 个执行包的 32 位地址。因此,虽然可以对这个寄存器写值,但任何随后而来的可屏蔽中断处理将刷新该写入值。

从一个可屏蔽中断返回的代码:

```
B    IRP;
NOP  5;     延迟间隙
```

8. 中断-复位状态

表明复位后 CPU 的状态。

复位后,控制寄存器及控制位的中值如下:

```
AMR,ISR,ICR,IFR 及 ISTP = 0h;
IER = 1h;
IRP 和 NRP 未定义;
CSR 的位 15～位 0 = 100h(小终端模式)
                  = 000h(大终端模式)
```

中断控制寄存器描述如表 5-7 所示。

表 5-7　中断控制寄存器描述

缩写	名　　称	描　　述
CSR	控制状态寄存器	控制全局使能或者禁止中断
IER	中断使能寄存器	使能或者禁止中断处理
IFR	中断标志寄存器	示出有中断请求但尚未得到服务的中断
ISR	中断设置寄存器	人工设置 IFR 中的标志位
ICR	中断清零寄存器	人工清除 IFR 中的标志位
ISTP	中断服务表指针	指向中断服务表的起始地址
NRP	不可屏蔽中断返回指针	包含从不可屏蔽中断返回的地址,该中断返回通过 B NRP 指令完成
IRP	中断返回指针	包含从可屏蔽中断返回的地址,该中断返回通过指令 B IRP 完成

5.2.3　中断响应过程

根据 TMS320DM6437 的中断机制,可以把中断分为三部分：中断请求、中断检查和中断响应。中断请求指中断源发出中断请求,如果中断处于开启状态以及该中断标志位使能,则处理器对该中断做出相应处理,否则忽略此次中断请求。通过中断检查,如果没有中断正在处理或者正在处理的中断的优先级比较低,则处理器对当前的中断请求做出响应,处理器会立刻跳转执行该中断任务。需要完成的工作是保存上下文,并将 PC 的值改为中断程序的入口地址。

TMS320DM6437 支持处理多个中断请求。当有一个中断正在执行的时候,如果产生了一个优先级更高的中断请求,这时处理器便会去响应优先级比较高的中断请求；如果中断请求的优先级比较低,则该中断请求被挂起,直到高优先级的中断执行完毕后才执行。

1. 中断请求

若有多个中断源,CPU 就需要判断其优先级：先响应优先级别高的中断请求；高优先级中断请求信号可中断低优先级中断服务。

2. 中断响应

CPU 要响应中断需要满足以下条件：无同级或高级中断正在服务；当前指令周期结束；若现行指令是 RETI、RET 或者访问 IE、IP 指令,则需要执行到当前指令及下一条指令方可响应。

响应过程：置位中断优先级有效触发器,关闭同级和低级中断；调用入口地址,断电入栈；进入中断服务程序。

3. 中断处理

中断处理就是在确认中断后,执行中断服务程序,从中断入口地址开始执行,直到返回指令为止。一般包括：保护现场；执行中断服务程序；恢复现场。用户根据要完成的任务

编写中断服务程序，要注意将主程序中的需要保护的寄存器内的内容进行保护。保护和恢复现场可以通过堆栈操作或切换寄存器组完成。

4. 中断返回

中断返回是指中断服务完成后，CPU 返回到原程序的断点，继续执行原来的程序，通过中断执行指令 RETI 来实现。

中断响应的过程可以用图 5-19 所示的流程图来说明。

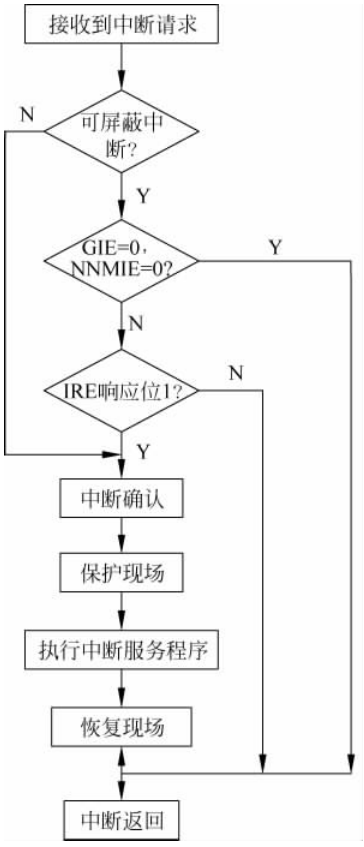

图 5-19　中断响应流程图

5.2.4　中断嵌套

中断嵌套是指中断系统正在执行一个中断服务时，如果事先设置了中断优先级寄存器 IP，那么另一个优先级更高的中断提出中断请求时发生中断嵌套，这时会暂时终止当前正在执行的级别较低的中断源的服务程序，去处理级别更高的中断源，待处理完毕，再返回到被中断了的中断服务程序继续执行；如果没有设置 IP，则不会发生任何嵌套。如果有同一优先级的中断触发，它并不是在“不断地申请”，而是将它相应的中断标志位置即 IE 寄存器的某位置位，当 CPU 执行完当前中断之后，按照查询优先级重新去查询各个中断标志位，进入相应中断。这种允许被“嵌套”的中断服务程序在软件上需要做额外的处理，中断服务函数开头首先得保存原始的中断返回指针 IRP 或 NRP、中断使能寄存器 IER 以及控制状态寄存器 CSR，一般保存在堆栈中，随后可以按要求重新设置 IER，使能比当前中断优先级更

高的中断，关闭低优先级的中断；然后重新开启全局中断，这之后的代码就可以被中断嵌套。

5.2.5 中断向量程序

```
interrupt void c_int14(void)                    //中断服务程序
{
   if(flag == 0)
   {
         GPIO_pinWrite(Hgpio,GPIO_PIN3,flag);
         flag = 1;
   }
   else
   {
       GPIO_pinWrite(Hgpio,GPIO_PIN3,1);  //点灯
       flag = 0;
    }
}
* ---------------------------------------------------------------------------
* 定义全局变量，并从此文件输出
* ---------------------------------------------------------------------------
.global _vectors                                //全局标号，可以在别处使用
.global _c_int00
.global _vector1
.global _vector2
.global _vector3
.global _vector4
.global _vector5
.global _vector6
.global _vector7
.global _vector8
.global _vector9
.global _vector10
.global _vector11
.global _vector12
.global _vector13
.global _c_int14; Hookup the c_int14 ISR in main()
.global _vector15
* ---------------------------------------------------------------------------
* 某些全局变量在本文件引用，但在别处定义
* 注意中断服务例程需在此引用
* ---------------------------------------------------------------------------
.ref _c_int00                                   //相当于 extern，在这里引用，在别处定义
* ---------------------------------------------------------------------------
* 该宏展示了中断服务表中的一个入口
* ---------------------------------------------------------------------------
VEC_ENTRY .macro addr                           //定义中断向量入口地址
STW B0, * -- B15;保存 B0 内容，中断产生后执行的第一条指令
```

```
MVKL addr,B0;
MVKH addr,B0;                                    //把地址装入 B0
B B0;跳转至 B0 中存储的地址
LDW  * B15++,B0;恢复 B0 内容; 由于 C6000 流水线的原因,跳转后仍可以执行多条指令
NOP 2
NOP
NOP
.endm
* ------------------------------------------------------------------------
* 该示例中断服务程序用于初始化 IST
* ------------------------------------------------------------------------
_vec_dummy:未定义中断服务程序
B B3;其他没有定义的中断跳转至 B3 存储的地址
NOP 5
* ------------------------------------------------------------------------
* 以下是真正的中断向量表,并在.text: vecs 中给出
* 如果读者不在连接命令文件中明确指出章节,将会默认连接到.text 处
* 另外,需设置 ISTP 寄存器指向该表
* ------------------------------------------------------------------------
.sect ".text:vecs";定义段
.align 1024;1024 字节对边界对齐
_vectors:
_vector0: VEC_ENTRY _c_int00;RESET 跳转到_c_int00 , _c_int00 是 C 语言程序的入口
_vector1: VEC_ENTRY _vec_dummy;NMI
_vector2: VEC_ENTRY _vec_dummy;RSVD
_vector3: VEC_ENTRY _vec_dummy; 所有未定义中断均跳转到同一地址
_vector4: VEC_ENTRY _vec_dummy
_vector5: VEC_ENTRY _vec_dummy
_vector6: VEC_ENTRY _vec_dummy
_vector7: VEC_ENTRY _vec_dummy
_vector8: VEC_ENTRY _vec_dummy
_vector9: VEC_ENTRY _vec_dummy
_vector10: VEC_ENTRY _vec_dummy
_vector11: VEC_ENTRY _vec_dummy
_vector12: VEC_ENTRY _vec_dummy
_vector13: VEC_ENTRY _vec_dummy
_vector14: VEC_ENTRY _c_int14; Hookup the c_int14 ISR in main() 定时中断向量
_vector15: VEC_ENTRY _vec_dummy
```

本章小结

本章一方面详细讲解了 TMS320DM6437 的流水线操作,包括流水线的具体流程、指令和存储器对流水线性能的影响和解决方案；另一方面叙述了 DSP 的中断系统,包括基础知识、中断控制寄存器和中断响应过程,并简要介绍了中断嵌套和中断向量程序。

思考与练习题

1. 请简述流水线原理。
2. 说明可屏蔽中断的响应过程和中断被响应后的操作。
3. 中断向量表的作用是什么？如何设置中断向量表？
4. INTR 中断和 NMI 中断有什么区别？
5. 在中断服务程序的入口处，为什么常常使用开中断指令？
6. CPU 满足什么条件时能够响应可屏蔽中断？