MIPS异常和中断处理机制研究

SMB产品线 沈浩

版 本 历 史

版本/状态	责任人	起止日期	备注
1.0/正式	沈浩	2010-10-25	➢ 初始发布。

目录

1. 摘要

在MIPS体系结构中，中断、自陷、系统调用以及其他打断程序正常运行的流的事件称为异常，都采用同一种机制处理。比如系统复位后MIPS重新启动的方式就是当做是一种异常来实现的，借用了异常处理的功能。本文将会在MIPS体系结构下看看MIPS CPU是怎样决定接受异常以及软件要怎样做去正确处理异常。

2. 异常类型

MIPS的异常和中断服务的目的主要有三：

1. 提供一个合法地从用户态到内核态的切换通道，使得程序能够访问如CP0、kuseg等平时被保护的资源；
2. 处理一些非法的操作，如TLB Miss/Address Error等；
3. 处理外部和内部的中断。

在研究异常之前，我们对异常做一个大致的分类。这样我们既对异常有了初步的了解，又对异常发生的条件可以有一个初步的认识。

- 外部事件：在CPU核之外的事件。也就是中断事件。中断可以让CPU注意到某些外部事件，这是同时处理多个事件的操作系统所必不可少的特性。中断是唯一由CPU正常指令流之外的事件引起的异常条件。因为中断不是我们小心就可以避免的，所以必须有某种机制在必要的时候禁止中断。
- 存储器地址转换异常：当某个地址需要转换但是硬件不能有效转换时，或当写一个有写保护的页时，就会发生异常。软件必须确定这个异常到底是不是错误。如果异常症状是由于应用程序访问了允许的地址空间之外的非法内存，问题的解决是软件决定终止以保护其他的部分。
- 其他需要内核干预的非常情况：其中最为显著地一个例子是浮点指令导致的条件，此时硬件无法处理由于某些困难和少见的操作符的组合而寻求软件仿真服务。这类情况由于不同的内核会有不同的处理。
- 程序或硬件检测到的错误：这包括不存在的指令、在用户权限下非法的指令、在相应的SR位被禁止时执行的协处理指令、整数溢出、地址对齐出错、用户态中访问kuseg以外的地址。
- 系统的调用和自陷：这些指令的整个目的就是产生定义好的异常，它们用来以一种安全的方式提供软件服务（系统调用、精心植入的条件自陷代码以及断点）。

3. 异常发生条件

MIPS 4KEc 处理器核的异常来源于很多方面，包括转换旁视缓冲（TLB）未命中、算术溢出、I/O中断、系统调用（system calls）。当CPU检测到其中一个异常时，指令流将会挂起处理器将进入内核态。

在内核态下处理器禁止中断并且强迫进入位于特定地址的软件异常处理程序，软件异常处理将会首先保存处理器的内容，包括PC指针的内容、当前的操作模式以及中断的状态，以便在异常返回时可以恢复这些内容。

《See MIPS Run》中，将MIPS的异常机制称为“精确异常”（precise exception）。用通俗的语言解释之，由于异常是执行指令时同步发生，因此，在造成异常的指令之前执行的指令，无疑均是有效的。然而，由于MIPS的高度流水体系结构，在引发异常的指令执行时，后面一条指令已经完成了读取和译码的预备工作，万事俱备，只待ALU部件空闲即执行之。当异常产生时，这些预备工作便被废弃。CPU从异常中返回时，再重新做读取和译码的工作。

从程序员角度看来异常发生的时间就是没有歧义的：异常之前执行的最后一条指令就是异常受害指令的钱一条。如果该异常不是中断，受害指令即是引发异常的指令。在典型的MIPS CPU上中断发生后，在中断处理开始前完成的最后一条指令就是当检测到中断时正好结束MEM阶段的指令。异常受害指令就是那条刚刚结束ALU阶段的指令。但是，要小心注意：MIPS体系结构并不承诺精确地中断延时，中断信号在到达CPU核之前可能要花一个或者几个时钟阶段重新同步。

因此，我们就可以保证，在异常发生时，异常指令之后所有的指令均不会被执行。这样，就不需要在MIPS的异常处理例程(Exception Handler)中为延迟槽(Delay Slot)指令而烦恼了。

4. 异常相关寄存器

所有的异常、中断行为都是处理器根据CP0控制寄存器内的相关寄存器的有关域实现的，同时在后面的描述中将会涉及到很多与寄存器相关的功能和描述。本节将介绍与异常和中断有关的寄存器。

4.1 状态寄存器（SR）

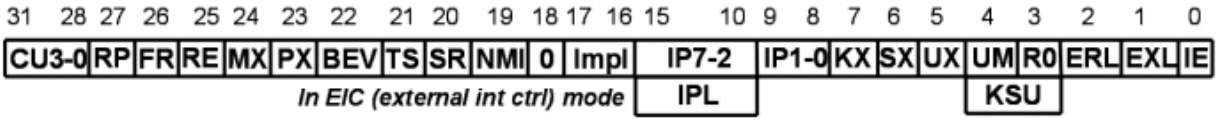


图 4-1 状态寄存器的几个域

以下是对与异常有关的域的介绍：

BEV：启动时的异常向量。当BEV=1时，CPU采用ROM（kseg1）空间的异常地址入口点。运行的操作系统里，BEV通常设置为0。

SR、NMI：软复位或者不可屏蔽的中断发生了：MIPS CPU提供几种不同方式的复位，与硬件信号区别。特别是，配置寄存器Config在软复位过程中保持不变——当然软复位或者硬复位是CPU永远也不会返回的异常。SR(SR)域在硬件复位（所有运行参数都要重新装入）后清零，而在软复位或NMI后置位。SR（NMI）位只在NMI异常之后才置位。

IM7~0中断屏蔽：一个8位的域允许定义哪些中断源活动时产生异常。其中六个中断源由CPU核外部的信号产生（一个可能由FPU使用）；其余两个是Cause寄存器中软件可写的中断位。

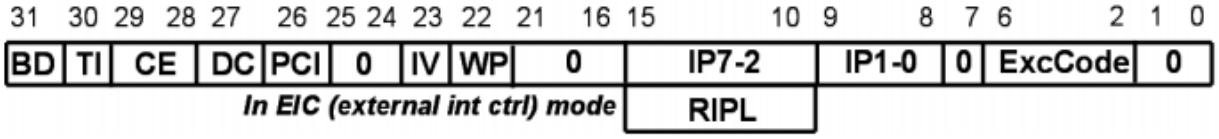
KSU CPU特权级：0是核心态，1是管理级，2是用户级。如果紧跟异常之后的EXL和ERL被置位，不管这个域是什么值CPU都会自动处于核心态。在MIPS 4KEc处理器中把该域记载为两个单独的域，高位叫做UM。

ERL 错误级：在CPU发现收到错误的数据时该位置位。

EXL 异常级：任何异常发生时置位，这会强行进入核心态并禁止中断：目的是把EXL位维持足够长的时间以便软件决定新的CPU特权级和中断屏蔽位应该设成什么。

IE 全局的中断使能位：不论这一位是什么值，ERL和EXL总是禁止所有中断。

4.2 原因寄存器（Cause）



Cause寄存器用来找出发生的一擦很难过类型，决定调用哪个异常处理例程。以下是有关的域的介绍：

BD 分支延迟：只要是发生在延迟槽中的指令，Cause（BD）就会置位，EPC就指向分支指令。如果想要分析异常受害指令，只要看看Cause（BD）（如果Cause（BD）==1，那么该指令位于EPC + 4 ）就知道了。

CE：协处理器错误：如果异常是由于相应的SR（CUx）位没有使能某个协处理器格式的指令引起的，那么Cause（CE）保存这条指令的协处理器号。

IV 该位写入1表示中断将使用一个特殊的异常入口点。

IP7-0 待决的中断：给出待发生的中断。Cause（IP7-2）照抄CPU硬件的输入信号，Cause（IP1-0）（软件中断位）可读可写包含你最近写入的值。当相应的SR（IM）位（还要受其他禁止中断的条件约束）允许时，这八位中的任意一位活动都会导致一个中断。Cause（IP）和Cause寄存器的其他域有着微妙的不同：它并不告诉你异常触发时发生了什么，而是告诉你现在在发生什么。

ExcCode：编码不同类型的异常：

表 4-1 ExcCode 异常的不同类型

ExcCode	助记符	描述
0	Int	中断
1	Mod	存储操作时，该页在TLB中被标记为只读，修改该页发生异常
2	TLBL	没有TLB转换（读写）。也就是TLB中没有和程序地址匹配的有效入口。当根本没有匹配项（连无效的匹配项都没有）并且CPU尚处于异常模式——SR（EXL）置位——即TLB失效时，为高效平滑处理这种常见的事件采用的特殊异常入口点
3	TLBS	
4	AdEL	地址错误 (取数、取指或存数时)：这要么是在用户态试图存取kuseg以外的空间，或者是试图从未对齐的地址读取一个双字、字或者半字
5	AdES	
6	IBE	总线错误 （取指）
7	DBE	总线错误 (读取数据)
8	Sys	执行了一条syscall指令，引发的异常
9	Bp	执行了一条break断点指令，由调试程序使用。
10	RI	不认识的或非法指令码
11	CpU	使用未使能的协处理指令异常
12	Ov	自陷形式的整数算术指令导致的溢出
13	Tr	Trap 异常
14-15	-	保留
16	IS1	定制的异常类型与CPU的具体实现相关
17	CEU	CorExtend 不可用
18	C2E	来自协处理器2的异常
19-22	-	保留
23	WATCH	load/store的物理地址匹配了使能的WatchLo/WatchHi寄存器（调试）
24	MCheck	机器检查（CPU检测到了CPU控制系统中的灾难性错误）
25-31	-	保留

4.3 其他相关寄存器

EPC 异常返回地址寄存器：保存异常返回点的寄存器。导致（或者遭受）异常的指令地址存入EPC，除非Cause寄存器的BD位置位了，这种情况下EPC指向前一条（分支指令）。

BadVaddr 无效虚拟地址寄存器：这个寄存器保存引发异常的地址，在发生MMU相关异常时，在用户程序试图访问kuseg以外的地址时，或者地址没有正确对齐时，该寄存器被设置。

Count/Compare寄存器：CPU的片上定时器。提供一个简单的间隔定时器，连续运行并且可以用编程中断。Count是一个连续计数的32位递增计数器，一般以CPU的流水线频率的同频或半频运行。Count计数到最大的32位无符号数值时，就会溢出而回到零。你可以读取Count获取当前的“时间”。Compare是一个32位的可写的寄存器。当Count计数增加到等于Compare中的值时，就会发出中断信号。中断信号会一直有效，直至下一次写入Compare才清除。

5. 异常优先级

下表详细列出了MIPS 4KEc所有可能的异常（优先级从上到下是从高到低），如果出现同时发生的异常，则先处理优先级别较高的异常。

异常	描述
Reset	重启
Soft Reset	软件重启
DSS	EJTAG 单步调试异常
DINT	EJTAG 调试中断。设置*ECR* 寄存器中的EjtagBrk位。
NMI	不可屏蔽中断（设置SR寄存器的NMI位）
Machine Check	TLB 写入冲突异常
Interrupt	unmasked 硬件和软件中断
Deferred Watch	Deferred Watch
DIB	EJTAG 硬件调试break指令中断
WATCH	内存断点异常
AdEL	取指地址对齐错误。 用户态访问内核态地址错误。
TLBL	TLB未命中。 TLB 命中页的V域 V=0
IBE	一般是指令cache出错
DBp	EJTAG 断点(指令SDBBP).
Sys	SYSCALL 指令.
Bp	BREAK 指令.
CpU	执行协处理器指令时协处理器没有使能.
CEU	CorExtend禁止时执行CorExtend 指令.
RI	执行到一条没有定义的指令时进入此异常
C2E	执行了一个引发通用异常的协处理器2指令
IS1	Execution of coprocessor 2 instruction which caused an Implementation Specific exception 1 in the coprocessor.（不明白）
Ov	算术溢出异常.
Tr	条件陷阱指令
Exception	Description
DDBL / DDBS	EJTAG 地址中断 (仅地址) or EJTAG 数据中断(地址和数据).
WATCH	内存断点异常
AdEL	地址对齐错误. 用户态下访问内核态地址
AdES	存储地址错误. 用户态下保存数据到内核态地址范围
TLBL	TLB载入未命中
TLBS	TLB数据保存未命中
TLB Mod	保存到TLB页上时D=0
DBE	读取或写入总线错误
DDBL	EJTAG数据硬件断点

6. 异常向量

异常向量是异常处理开始的地方。

大多数CISC处理器由硬件（或微代码）来分析异常，根据发生的异常类型把CPU发送到不同的入口点。在MIPS中，也不是所有的异常都是平等的。在体系结构发展的过程中也出现了差别。

用户地址 TLB（ translate look-aside buffer）填充。TLB 硬件只保存有有限的虚拟地址和物理地址的对照表，因此在使用虚拟地址的操作系统中，很复杂的程序所用的虚拟地址有可能不在 TLB 中。我们把这个事件称为 TLB miss（因为 TLB 是一种有软件管理的缓冲）。使用软件来管理这种情况在精简指令处理器刚被推出时是有争议的。MIPS 处理器为软件管理 TLB miss 提供了有力的支持。在这种异常处理中，硬件提供了足够的支持所以软件只需要13个时钟周期就足够了。这种经常有用到的程序被提供了特殊的入口地址，所以它能被优化的很好，不必判断到底是哪种异常发生了。

奇偶数据校验/ECC出错。R4000 和以后的处理器都检测数据错误并在检测到数据错误时产生一个陷入。数据错通常在数据从内存中读出的时候发生，在数据从缓存中被读出的时候被检测到。在数据出错时，用被缓存的入口地址显然是不明智的。所以此时无论 SR(BEV)位被设置与否，都应该用为被缓存的入口地址。

复位。从很多方面来说，把系统重置看作一种异常都是有道理的。特别是在考虑到 R4000及以后的处理器用同一个中断入口地址处理冷启动和热启动的时候。事实上，不可屏蔽中断应该被看作是一个较弱的热启动，和热启动相比，它的唯一区别就是，必须在当前指令被完成之后才生效。

所有异常的入口地址都位于 MIPS存储器映像中不作地址装换的区域，不要高速缓存的入口点位于kseg1，需要高速缓存的位于kseg0。不要高速缓存的入口点当SR（BEV）置位时是固定的，但是当SR（BEV）清零时，就可以对EBase寄存器进行编程来平移所有的入口点一起到别的内存块。

最初的异常向量间的距离缺省为128 个字节，因为 MIPS 的设计者认为对于基本的中断处理，32 条指令应该足够了。这样，我们就省掉了跳转指令，而有不浪费太多的内存。

下表描述了异常入口点的规则：

内存区	入口点	异常处理
片上调试	0xFF20 0200	EJTAG调试，当映射到“probe”存储区时
	0xBFC0 0480	EJTAG调试，当使用正常的ROM时
复位（仅有ROM）	0xBFC0 0000	复位及不可屏蔽中断入口点
ROM入口点	0xBFC0 0000	中断专用——仅当Cause(IV)置位时使用，并非所有的CPU都有
	0xBFC0 0380	所有其他的异常
	0xBFC0 0300	高速缓存错误
	0xBFC0 0200	简单的TLB重填（SR（EXL）为零），TLB未命中异常
	0xBFC0 0280	XTLB未命中异常
RAM入口点 (BASE代表编程到EBase寄存器中的地址)	BASE+0x200+...	多个中断入口点 中断特殊/专用（Cause(IV)为一）
	BASE+0x180	所有其他异常
	BASE+0x100	高速缓存错误，在RAM中但是永远不经过高速缓存的kseg1窗口
	BASE+0x000	简单的TLB重填（SR（EXL为零））

MIPS CPU处理一个异常时通常所要做的：

1. 设置EPC指向重新开始的地址
2. 设置SR(EXL)位，强制CPU进入内核态（高特权级）并且禁止中断
3. 设置Cause寄存器这样软件可以看到发生异常的原因。在地址异常时，BadAddr也要设置，存储器管理系统异常还要设置某些MMU寄存器。
4. CPU从异常入口点开始取指

7. 异常处理过程

所有的MIPS异常处理例程都要经过以下相同阶段：

引导：在异常处理程序入口，被中断打断的程序只有很少的状态信息被保存了下来，所以第一件要做的事情是就是给自己腾出足够的空间能够做你想做的，而且不要覆盖被中断的程序的重要数据。

处理不同的异常：查询Cause（ExcCode），它告诉你为什么发生了异常，允许操作系统为不同的异常定义不同的函数。

构造异常处理环境：复杂的异常处理程序可能会是使用高级语言编写，也可能会使用到标准函数库。因此需要提供一块其他软件代码没有用到的，并且保存任何CPU寄存器可能被中断的程序和被调用的例程允许改变的寄存器。

处理异常：处理异常程序。任何允许的代码。

准备返回：高层函数通常当做子程序来调用，因而要返回到底层处理代码。在这里，保存的寄存器得到恢复，通过把SR改回到刚发生异常后的值CPU得以返回到其安全（内核模式，异常关闭）状态。

从异常返回：控制返回到异常受害指令并从内核态改变到低的特权级必须同时完成。即你在应用程序中和特权级运行一条指令也成为一个安全漏洞；另一方面，以用户特权级试图运行一条内核指令会导致致命的异常。指令eret可以同时清除SR（EXL）位并将控制返回到EPC中保存的地址。

8. 中断

8.1 MIPS CPU的中断资源

CPU是否愿意响应某个中断受SR中的位的影响。有三个相关的域：

- 全局中断使能位SR（IE）必须设置为1，否则不会服务于任何中断
- SR（EXL）（异常级）位和SR（ERL）（错误级）位，如果置位会禁止中断（因为紧接异常之后其中一位必然置位）
- 状态寄存器也有八个单个的中断屏蔽位SR(IM),Cause寄存器中的每个中断各一位。每个SR（IM）要设置成1以允许相应的中断，这样程序可以确切的知道那个中断可以发生，哪个不行。

查看Cause寄存器的内容可以知道当前是哪个中断正在活动。Cause寄存器活动的输入电平和SR（IM）的屏蔽对齐到了同样的位置，软件中断处于最低位，硬件中断依次递增。

一般来说（非向量化中断和EIC模式），所有的中断都是平等的。现在的CPU提供了可选的不同的异常地址入口点保留给中断使用。

中断处理开始于CPU接收到一个异常并从Cause（ExcCode）发现该异常是一个硬件中断。通过查询Cause（IM），可以找出哪个中断在活动，也就知道哪个设备请求服务/发出信号。通常的中断处理过程如下：

- 查询Cause寄存器的IP域，并和当前的中断屏蔽SR(IM)做逻辑“与”操作，以获得活动的、允许的、使能的中断请求位图。可能不止一个，其中的任何一个都有可能是导致中断的原因。
- 软件上可以在这里对中断处理优先级的问题，根据系统的需求分配不同的固定的优先级，并要求处理最高的优先级。
- 保存SR寄存器的IM域。修改SR（IM）以确保当前中断和你的软件认为具有相同或者较低优先级的所有中断被禁止。
- 置位全局中断使能位SR（IE）以允许处理高优先级别的中断。当清除异常级的时候，还需要修改CPU的特权级域SR（KSU）以保持CPU仍然处于核心态，还需要对SR（EXL）清零以离开异常模式。
- 调用中断例程。
- 返回时，需要再禁止中断以便能够恢复中断前的寄存器值并接着执行被中断的任务，也就是置位SR（EXL）。

8.2 实现中断优先级

在最新的向量化中断以前，MIPS CPU对于中断优先级有一个简单的办法：一切中断都是平等的。

如果CPU不支持向量化中断，要实现向量化中断必须：

- 在任何时候软件都要维护一个明确定义的中断优先级（IPL），CPU以该优先级运行。每个中断源分配到其中一个优先级。
- 如果CPU处于最低的中断优先级IPL，则允许任何中断。这时正常的程序的运行状态。
- 如果CPU处于最高优先级IPL，则禁止所有中断。

中断处理程序不仅可以按照分配给具体的中断源的优先级IPL运行，程序员可以升高和降低IPL。设备驱动程序应用端与硬件或中断处理程序通信的部分常常需要在临界区防止设备中断，所以程序员会临时升高IPL以匹配设备的中断输入位。

9. 异常处理和服务程序流程图

一般来说，CPU硬件首先会对异常做初步的处理，然后才是软件对异常的处理。本文将介绍以下异常处理和服务程序流程：

- 通用异常处理程序
- TLB未命中异常处理
- 复位、软重启和NMI（不可屏蔽中断）异常处理
- 调试异常

9.1 通用异常处理过程

除了复位、软件重启、NMI和TLB未命中外的异常符合以下异常处理流程。另外，中断可以被IE禁止和 IM域屏蔽，Watch异常也在EXL=1时被屏蔽。

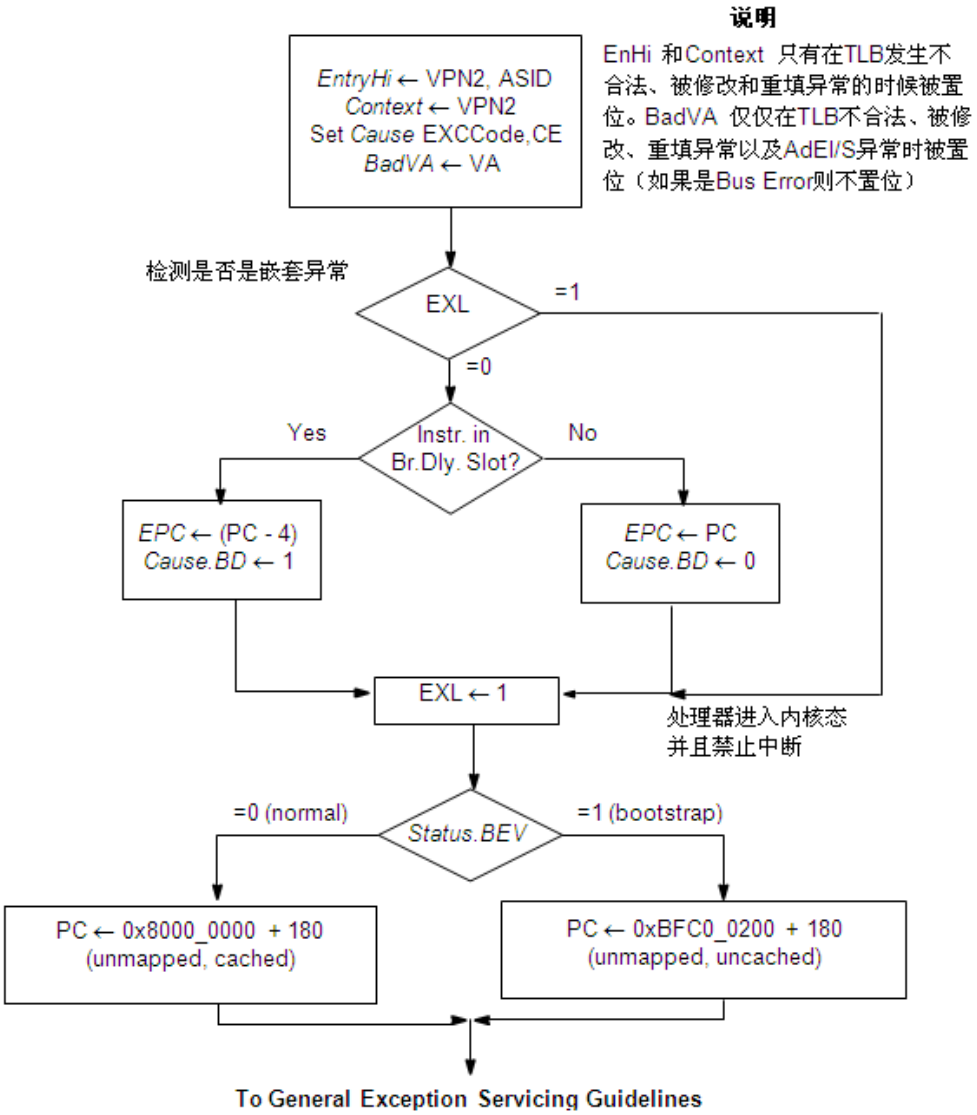


图 9-1 通用异常处理（硬件）

以下是软件处理过程：

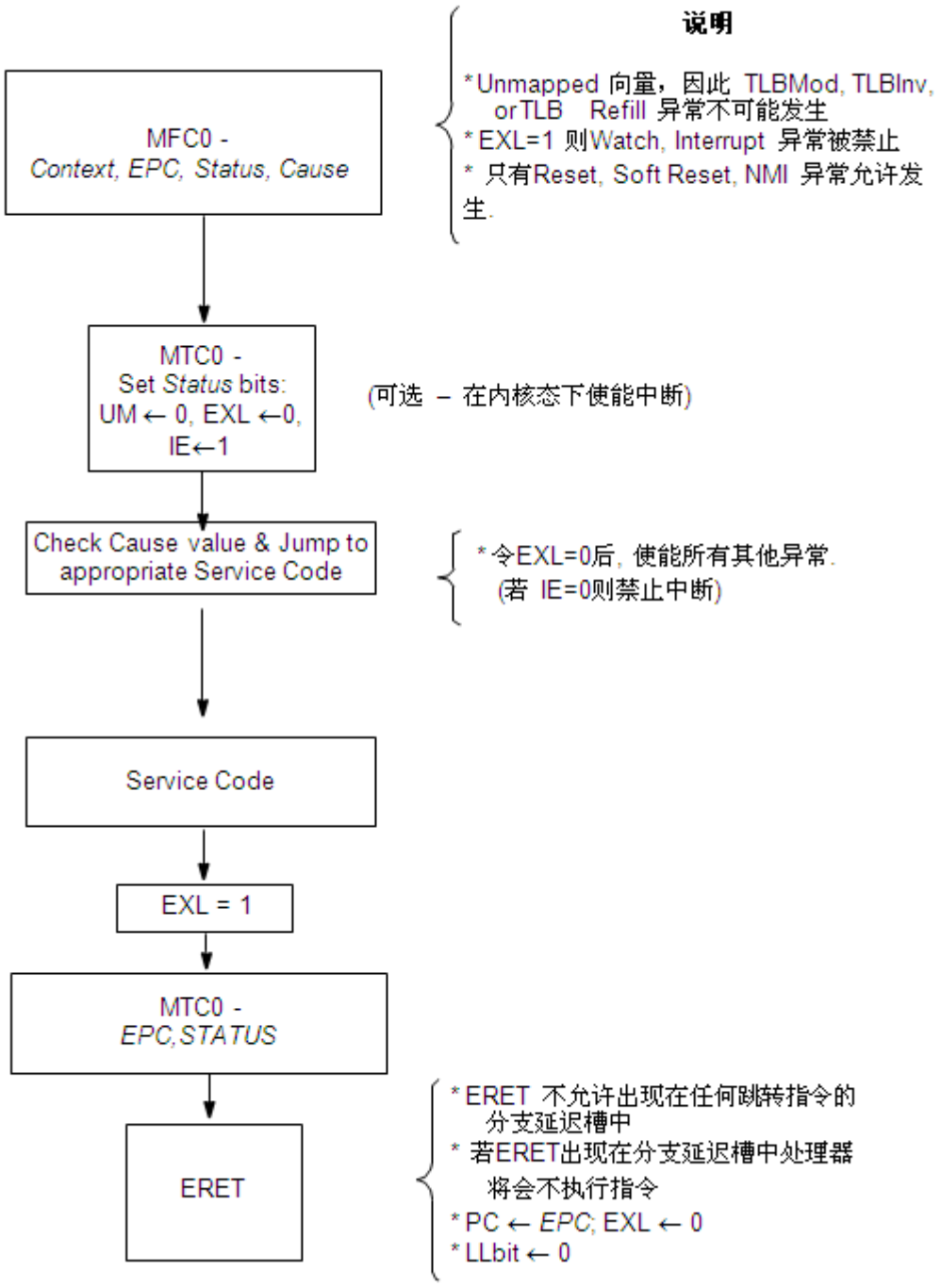


图 9-2 通用异常处理过程（软件）

9.2 TLB未命中异常处理过程

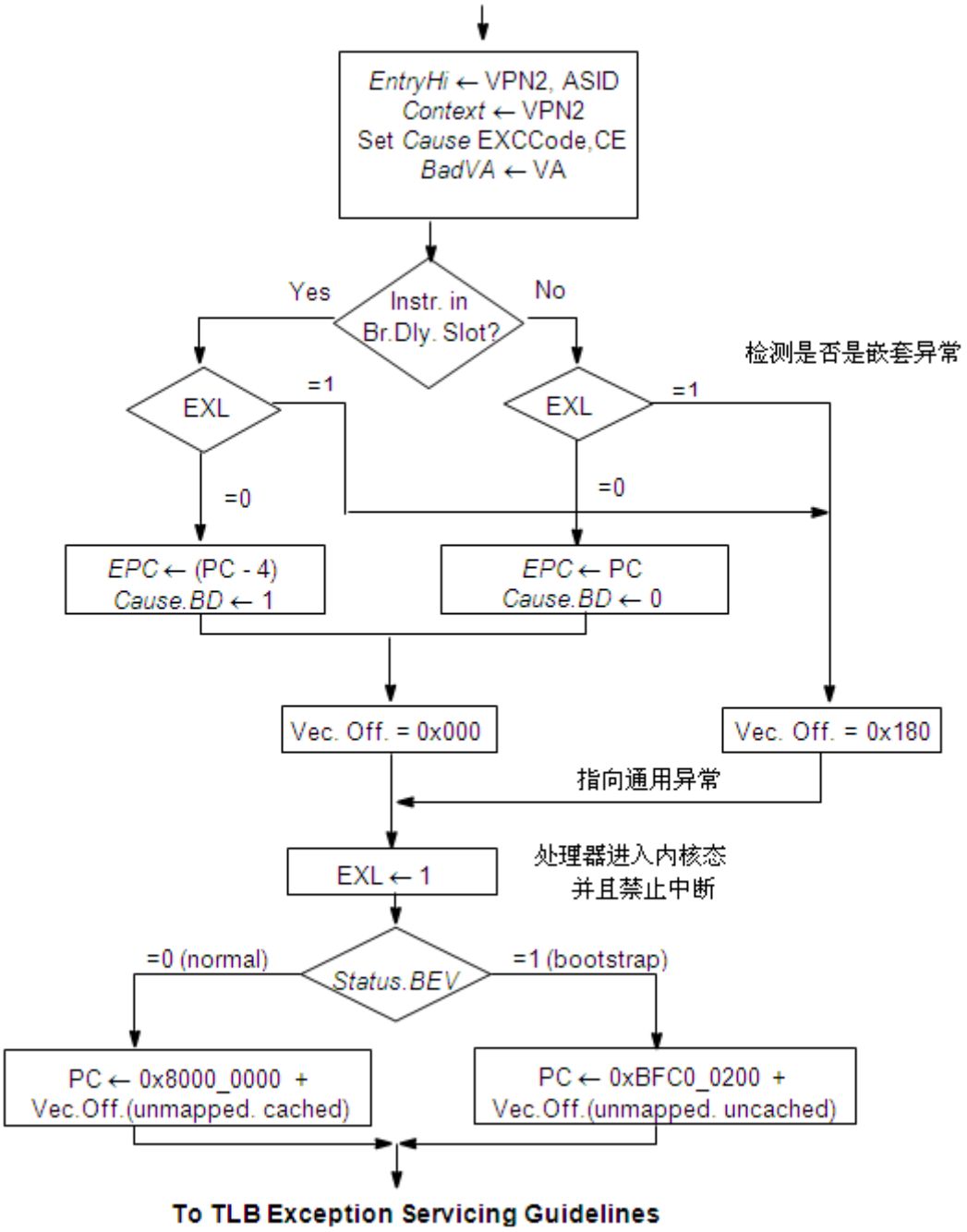


图 9-3 TLB未命中异常处理（硬件）

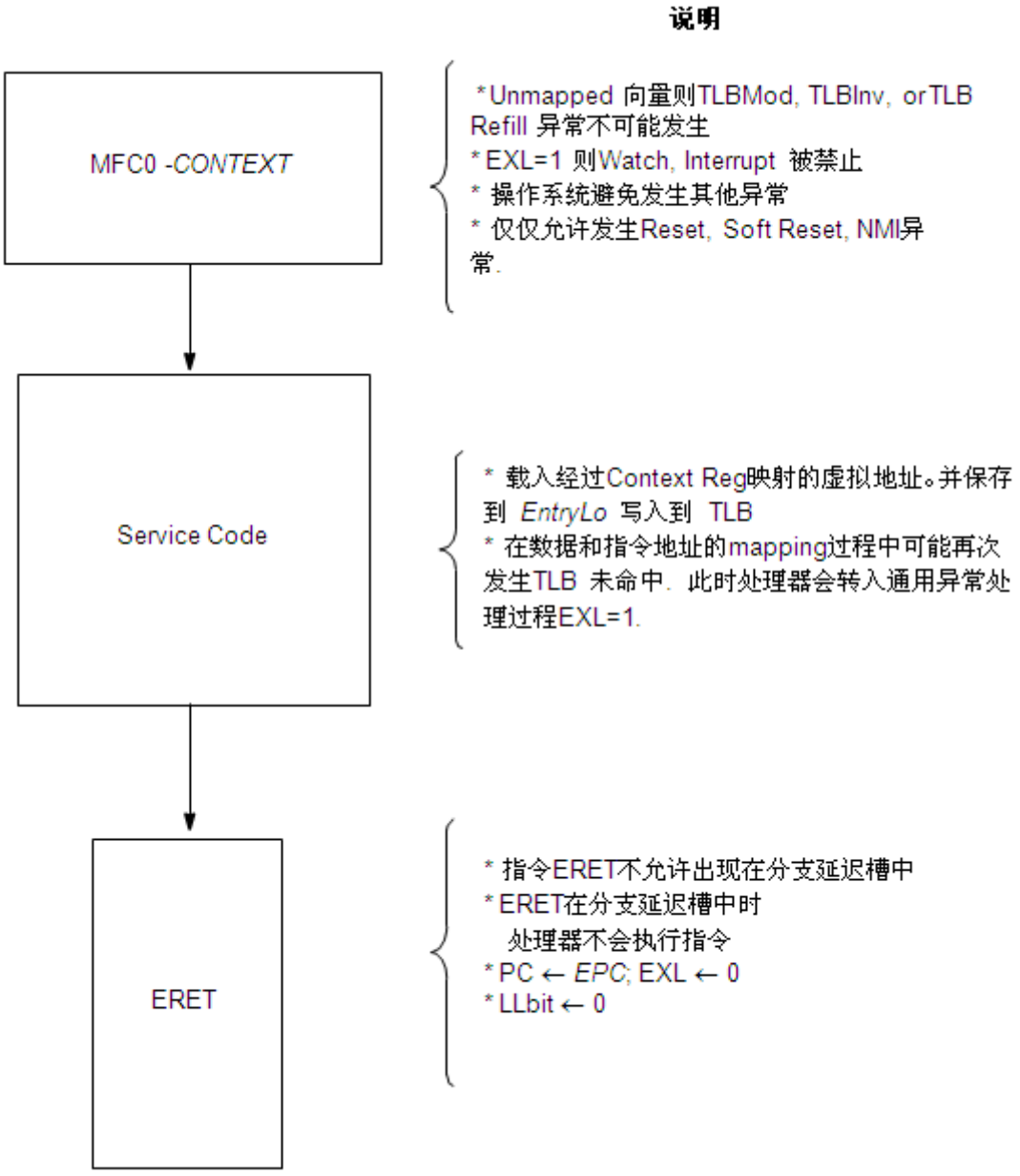


图 9-4 TLB未命中异常处理（软件）

9.3 复位、软重启和NMI中断处理过程

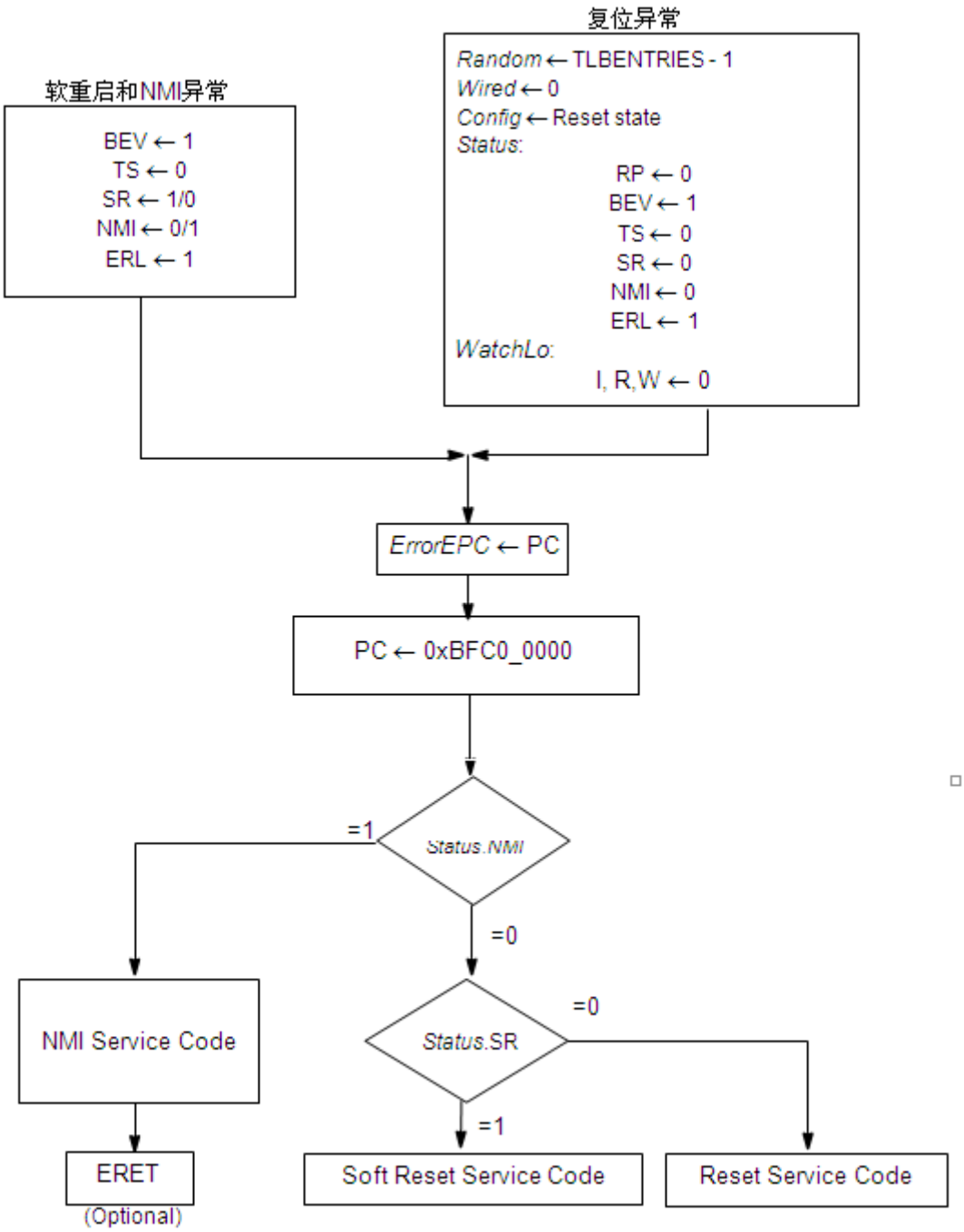


图 9-5 复位、软重启和NMI中断处理程序