内存故障检测与处理

前置科普 - 可跳过

AVX512指令集

1、指令集与扩展指令集

指令是指计算机执行某种操作的命令。一台计算机的所有指令的集合构成该机的指令系统，也称指令集。指令系统是计算机的主要属性，位于硬件和软件的交界面上。目前英特尔和**AMD**的绝大部分处理器都使用的是**X86**指令集。

扩展指令集，英特尔®指令集扩展是可在多个数据对象上执行相同操作时可提高性能的附加指令。**X86**指令也只能一次处理一个数据，这样效率就很低，在很多应用中，数据都是成组出现的，比如点的坐标（**X**，**Y**，**Z**轴）或者 颜色（**RGB**）等。为了提高**CPU**在某些方面的性能，就必须增加一些特殊的指令满足时代进步的需求，这些新增的指令就构成了扩展指令集。

指令集扩展可包括：

（**1**）单指令多数据 （**SIMD**）

（**2**）英特尔® **Streaming SIMD** 扩展（英特尔® **SSE**、英特尔® **SSE2**、英特尔® **SSE3** 和英特尔® **SSE4**）

（**3**）英特尔® **Advanced Vector Extensions**（英特尔® **AVX**、英特尔® **AVX2**和英特尔® **AVX-512**）

2、扩展指令集的演变过程

SIMD（Single Instruction Multiple Data，单指令多数据）

英特尔在**1996**年率先引入了**MMX**（**Multi Media eXtensions**）多媒体扩展指令集，也开创了**SIMD**（**Single Instruction Multiple Data**，单指令多数据）指令集之先河，即在一个周期内一个指令可以完成多个数据操作，**MMX**指令集的出现让当时的**MMX Pentium**处理器大出风头。

SEE（Streaming SIMD Extensions，流式单指令多数据扩展）

SSE 是一种支持单指令多数据的过程或技术。旧款处理器每个指令只处理一个数据元素。**SSE** 使指令能够处理多个数据元素。它用于 **3D** 显卡等密集型应用程序，以实现更快的处理速度。**SSE** 旨在取代 **MMX™** 技术。它的数量扩展到了英特尔®处理器的代次，包括 **SSE2**、**SSE3/SSE3S** 和 **SSE4**。每次迭代都带来了新的指令并提高了性能。

1999年英特尔在**Pentium III**处理器中率先推出的，并将矢量处理能力从**64**位扩展

到了128位。在Willamette核心的Pentium 4中英特尔又将扩展指令集升级到SSE2（2000年），而SSE3指令集（2004年）是从Prescott核心的Pentium 4（第一次接触电脑时就是奔4）开始出现。

SSE2（流式传输SIMD扩展2）

SSE2 通过添加 144 条指令扩展了 MMX 技术和 SSE 技术，可在各种应用中提高性能。以 MMX 技术引入的 SIMD 整数指令从 64 位扩展到 128 位。这使 SIMD 整数运算的有效执行率翻倍。

双精度浮点 SIMD 指令允许以 SIMD 格式同时执行两个浮点运算。这种对双精度运营的支持有助于加快内容创建、金融、工程和科学应用的速度。

增强了原始 SSE 指令，以支持灵活、更高动态的计算功率范围。这是通过支持多种数据类型的算法操作来完成的。示例包括双词和四词。SSE2 指令帮助软件开发人员充分灵活。它们在运行 MPEG-2、MP3 和 3D 显卡等软件时可以实施算法并提供性能增强。

SSE3（流式传输SIMD扩展3）

基于 90 纳米工艺的英特尔®奔腾® 4 处理器发布，引入了流式传输 SIMD 扩展 3（SSE3），其中包括比 SSE2 多 13 个 SIMD 指令。这 13 个新指令主要用于改进线程同步和特定的应用区域，如媒体和游戏。。

SSE4

指令集是自SSE以来最大的一次指令集扩展，它实际上分成Penryn中出现的SSE4.1和Nehalem中出现的SSE4.2，其中SSE4.1占据了大部分的指令，共有47条，Nehalem中的SSE4指令集更新很少，只有7条指令，这样一共有54条指令，称为SSE4.2。

2007年8月，通用CPU领域Intel友商AMD抢先宣布了SSE5指令集(SSE到SSE4均为英特尔出品)，英特尔当即表示不玩SSE了也不再支持SSE5。

AVX指令集

1、AVX和AVX2

英特尔® AVX是一种面向英特尔® SSE 的 256 位指令集扩展，专为浮点 （FP） 密集型应用而设计。英特尔 AVX由于矢量更宽、新的可扩展语法和丰富的功能而提高性能。英特尔 AVX2于 2013 年发布，扩展了跨浮点和整数数据域的矢量处理能力。这样就可以在各种应用程序上实现更高的性能和更高效的数据管理。例如图像和音频/视频处理、科学模拟、金融分析以及 3D 建模和分析。

2、AVX-512

英特尔® AVX-512 一条指令就能处理两倍于英特尔 AVX/AVX2 可处理的数据元件，是英特尔 SSE 功能的四倍。英特尔 AVX-512指令非常重要，因为它们为最苛刻的计算任务提供了更高的性能功能。英特尔 AVX-512指令在设计指令功能时可为编译器提供最高程度的支持。

CPU故障检测寄存器

1、MCA 与 MCE 基本概念

Intel引入了 **MACHINE-CHECK ARCHITECTURE(MCA)** 和 **machine-check exception(MCE)** 机制用来对服务器硬件进行自检，并在发现硬件错误的时候发出中断或异常。系统软件收到中断或异常后，会对其进行响应，进行相应的修复、告警或其他策略等。通过**Intel**的这个**RAS**特性（可靠性、可用性和诊断功能），保证在发生**crash**等错误前，服务器可以有机会做一些容错处理，大大提升了**Intel**在数据中心高可靠服务器领域的竞争实力。

MCA

MCA——Machine Check Architecture，它用来检测硬件（这里的**Machine**表示的就是硬件）错误，比如系统总线错误、**ECC**错误等。这套系统通过一定数量的**MSR**（**Model Specific Register**）来实现，这些**MSR**分为两个部分，一部分用来进行管理，另一部分用来记录发生的硬件错误。

MCE

当**CPU**检测到不可纠正的**MCE**（**Machine Check Error**）时，就会触发**#MC**（**Machine Check Exception**），通常软件会注册相关的函数来处理**#MC**，在这个函数中会通过读取**MSR**来收集**MCE**的错误信息，然后重启系统。当然由于发生的**MCE**可能是非常致命的，**CPU**直接重启了，没有办法完成**MCE**处理函数；甚至有可能在**MCE**处理函数中又触发了不可纠正的**MCE**，也会导致系统直接重启。

当然**CPU**还会检测到可纠正的**MCE**，当可纠正的**MCE**数量超过一定的阈值时，会触发**CMCI**（**Corrected Machine Check Error Interrupt**），此时软件可以捕捉到该中断并进行相应的处理。**CMCI**是在**MCA**之后才加入的，算是对**MCA**的一个增强，在此之前软件只能通过轮询可纠正**MCE**相关的**MSR**才能实现相关的操作。

CMCI

Corrected machine-check error interrupt (CMCI) 是**MCA**的增强特性。在原来的芯片里面，都是使用一种叫做 **threshold-based error reporting**的机制来处理 **corrected error.** 但是 **threshold-based error reporting**需要系统软件周期性的轮询检测硬件的 **corrected MC errors**，造成**CPU**的浪费。 **CMCI** 提供了一种机制，当 **corrected error**发生次数到达阀值的时候，就会发送一个信号给本地的**CPU**来通知系统软件。当然，系统软件可以通过 **IA32_MCi_CTL2 MSRs**来控制该特性的开关。

默认的情况下，**CMCI**是被禁止的。 在 **IA32_MCG_CAP[10] = 1**的情况下，系统软件需要使能每一个**bank**的 **IA32_MCi MSR**的 **CMCI**位来让芯片通过中断报告 **hardware corrected errors**。

可以通过向 **IA32_MCi_CTL2[14:0]**中写入希望的阀值的方法，来探 知某个**bank**中是否已经设置了阀值。如果可以写入并保存，则该**bank**支持设置阀值（同时**CMCI**也一定是支持的）。如果写入后，**IA32_MCi_CTL2[14:0]**都是**0**，那么就不支持设置阀值。同样，软件向 **MCi_CTL2[30]**写入**1**，来判断是否可以发送**CMCI**信号。如果写入后， **MCi_CTL2[30]=0**，那么该**BANK**就不支持**CMCI**，如果 **MCi_CTL2[30]=1**，那么

CMCI就是支持并使能的。

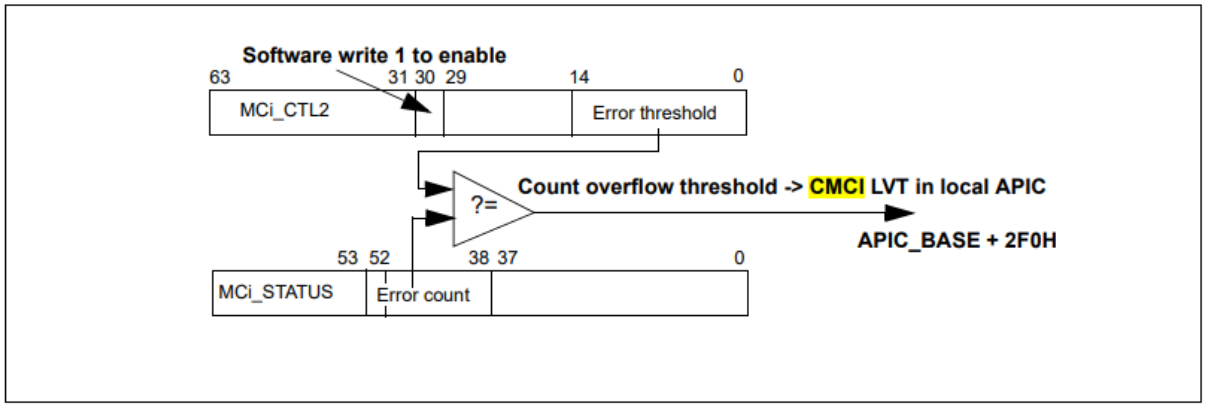


Figure 15-10. CMCI Behavior

2、Machine Check MSR 实现 MCA

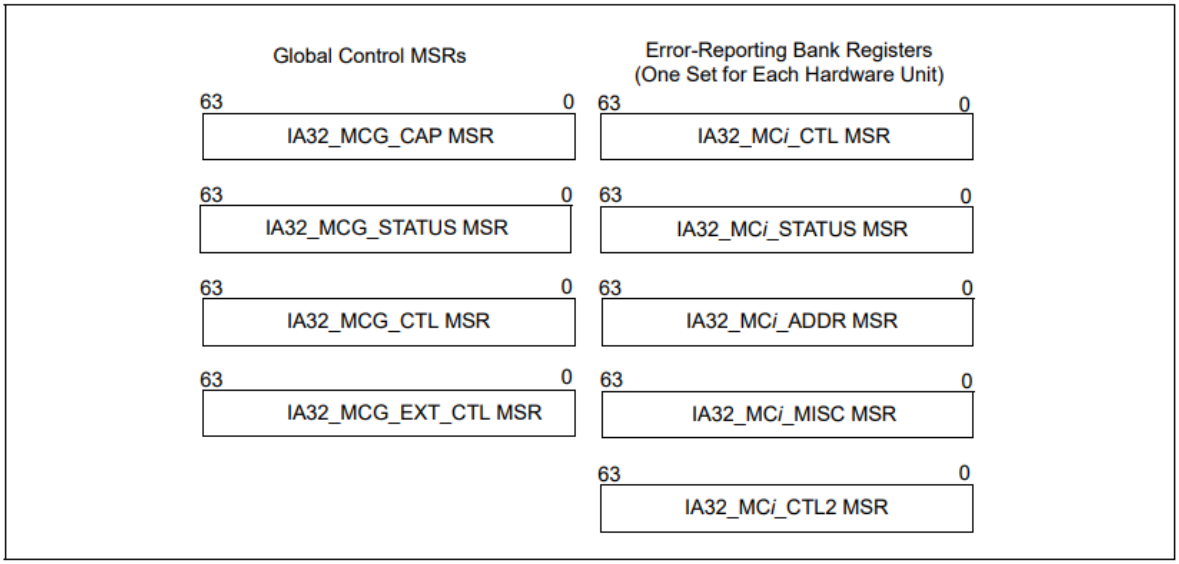


Figure 15-1. Machine-Check MSRs

上图包含**MCA**相关的所有**MSR**。左边的是全局寄存器，右边的是多组寄存器。其中的 **i** 表示各个组的 **index**，组的称呼是 **Error Reporting Register Bank**。**MCA**通过若干**Bank**的**MSR**寄存器来表示各种类型的**MCE**。

2.1 MCG

2.1.1、 IA32_MCG_CAP MSR (MCG)

IA32_MCG_CAP MSR 是一个只读寄存器，提供关于处理器的**MCA**信息。

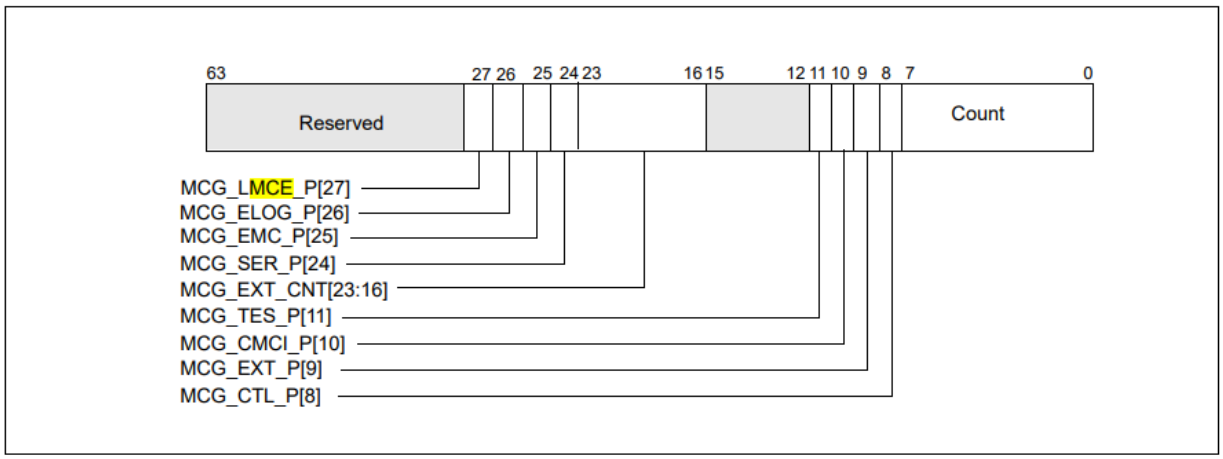


Figure 15-2. IA32_MCG_CAP Register

BIT0-7：表示的是CPU支持的Bank的个数；

BIT8：1表示IA32_MCG_CTL有效，如果是0的话表示无效，读取该IA32_MCG_CTL这个MSR可能发生Exception（至少在UEFI下是这样）；

BIT9：当前处理器是否实现了 extended machine-check state寄存器（该寄存器地址为180H）

BIT10：1表示支持CMCI，但是CMCI是否能用还需要通过IA32_MCi_CTL2这个MSR的BIT30来使能；

BIT11：1表示IA32_MCi_STATUS这个MSR的BIT56-55是保留的，BIT54-53是用来上报Threshold-based Error状态的；

BIT16-23：表示存在的Extended Machine Check State寄存器的个数；

BIT24：1表示CPU支持Software Error Recovery；

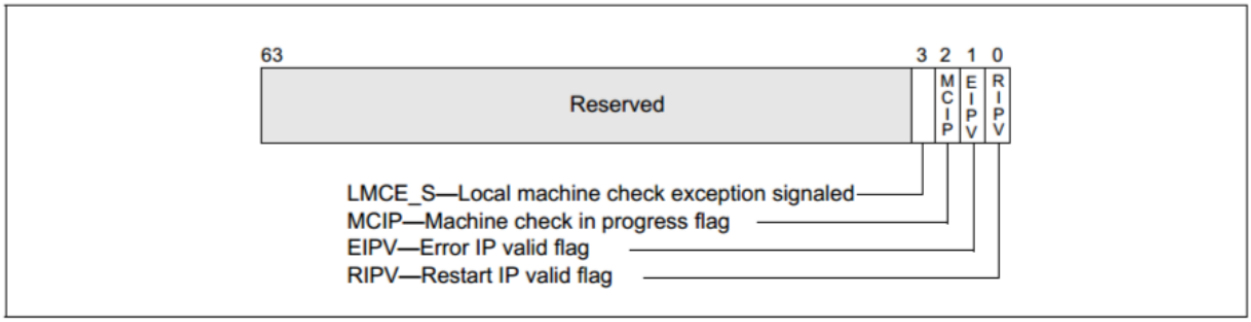
BIT25：1表示CPU支持增强版的MCA；

BIT26：1表示支持更多的错误记录（需要UEFI、ACPI的支持）；

BIT27：1表示支持Local Machine Check Exception；

2.1.2、IA32_MCG_STATUS MSR (MCG)

IA32_MCG_STATUS MSR 描述计算机检查异常后处理器的当前状态发生。



- `RIPV (restart IP valid) flag, bit 0` - 是否可重启指令指针所引用指令（restart IP valid)标志：这个寄存器位设置时，程序可以可靠地重启这个指令指针引用的压入堆栈的指令。当这个位寄存器位清除时，程序不能可靠地重启压入指令指针所引用指令。
- `EIPV (erro IP valid) flag, bit 1` - 是否准确表示指令指针引用的指令：这个寄存器位设置时，表示MCE发生时，这个指令指针指向的堆栈中的指令是和错误直接相关的。如果这个标志位没有设置，则指令指针有可能和错误无关。
- `MCIP (machine check in progress) flag, bit 2` - 标记是否产生了machine-check exception(MCE)。软件可以设置或清除这个标志位。当发生第二次Machine-Check Event(MCE)时候，如果这个MCIP被设置了，则会导致处理器进入shutdown状态。
- `LMCE_S (local machine check exception signaled), bit 3` - 标记是否发生了一个本地machine-check exception(MCE)。这个标志位设置的时候就宝石这个MCE异常只发生在本逻辑处理器上。

2.1.3、IA32_MCG_CTL MS MSR (MCG)

如果在**IA32_MCG_CAP MSR** 中设置了功能标志**MCG_CTL_P**，则存在**IA32_MCG_CTL MSR**。**IA32_MCG_CTL**控制机器检查异常的报告。如果存在，则向此寄存器写入 **1** 并启用机器检查功能和写入所有 **0** 将禁用机器检查功能。所有其他值均未定义和/或具体实施。

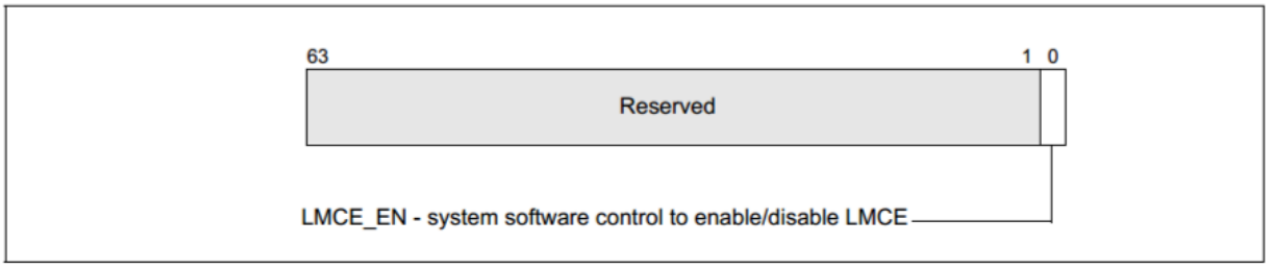
这个寄存器的存在依赖于**IA32_MCG_CAP**这个**MSR**的**BIT8**。主要用来**Disable**（写**1**）或者**Enable**（写全**0**）**MCA**功能。

2.1.4、IA32_MCG_EXT_CTL MSR (MCG)

如果**IA32_MAG_CAP MSR**中设置了能力标志 **MCG_LMCE**，则**IA32_MCG_EXT_CTL_MSR**存在。

IA32_MCG_EXT_CTL在系统中，**LMCE_EN**（位**0**）准许处理器向单个逻辑处理器发送一些**MCE**信号。

如果在**IA32_MCG_CAP**中，**MCG_LMCE_P**没有被设置，或者系统软件没有通过设置**IA32_FEATURE_CONTROL**使能**LMCE**，**LMCE_ENABLED**（位**20**），任何写入或者读取**IA32_MCG_EXT_CTL**的尝试都将导致 **#GP**（保护模式下的异常）。



系统软件将此设置为准许硬件想单个逻辑处理器发送一些**MCE**的信号。只有当平台软件已经按照下方所述配置**IA32_FEATURE_CONTROL**时，系统软件才能设置**LMCE_EN**。

> *LMCE*的预期用途需要平台软件和系统软件进行适当适配，平台软件可以通过设置*IA32_FEATURE_CONTROL MSR (MST*地址*3AH)* 中的*bit20 (LMCE_ENABLED)* 来打开*LMCE*。
>
> 系统软件必须确保 *IA32_FEATURE_CONTROL*。锁定（*bit0*）和*IA32_FEATURE_CONTROL*，在尝试设置 *IA32_MCG_EXT_CTL*之前，设置*LMCE_ENABLED*（*bit20*）。 *LMCE_EN*（*bit0*）。当系统软件启用*LMCE*时，硬件将确定特定错误是否只能传递给单个逻辑处理器。软件不应该判断硬件选择作为*LMCE*交付的错误类型。

2.2、MCi

每个**Bank**包含**IA32_MCi_CTRL**，**IA32_MCi_STATUS**，**IA32_MCi_ADDR**，**IA32_MCi_MISC**，**IA32_MCi_CTRL2**寄存器。**Bank**的数量在**IA32_MCG_CAP MST**（地址**0179H**）的**[7:0]**位设置。第一个错误报告寄存器（**IA32_MCO_CTL**）总是以地址**400H**开头。

2.2.1、IA32_MCi_CTL MSRs

IA32_MCi_CTL MSR控制特定硬件单元（或者一组硬件单元）的 **#MC**中断信号。**64**个标志（**EEj**）中的每一个都代表一个潜在的错误。设置**EEj**标志启用相关错误的**#MC**中断信号，清除该标志禁用该信号。处理器会丢弃对未实现位的写入。

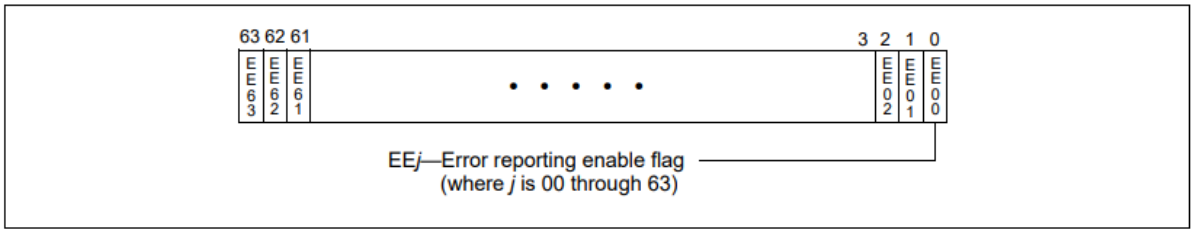


Figure 15-5. IA32_MCi_CTL Register

NOTE

2.2.2、IA32_MCi_STATUS MSRS

如果**IA32_MCi_STATUS MSRS** 的**VAL**（**valid**）标志被设置，则每一个**IA32_MCi_STATUS MSRS**包含了与**MCE**相关的信息。软件通过显式地写入全**0**来清除**IA32_MCi_STATUS MSRs**。对任何位写入**1**将会导致**#GP**

> *NOTE*
> 当 *IA32_MCG_CAP[24] = 1*，*IA32_MCG_CAP[11] = 1* 和 *IA32_MCG_CAP[10] = 1* 时，下图描述了 *IA32_MCi_STATUS MSR* 。 当 *IA32_MCG_CAP[24] = 0* 和 *IA32_MCG_CAP[11] = 1*，*bit 56:55* 被保留，*bit 54:53*可以设置 *threshold-based error* 报告。 当*n IA32_MCG_CAP[11] = 0*，*bit 56:53* 是其他信息字段的一部分。 *bit 54:53* 用于*threshold-based error*报告，始于*Intel Core Duo*处理器，目前用于缓存信息。当 *IA32_MCG_CAP[10]=0*时，*bit 52:38*是“其他信息”字段的一部分。*Intel 64*处理器引入了位*52:38*用于校正*MC*错误计数，*CPUID*将*DisplayFamily_DisplayModel*报告为 *06H_1AH*。

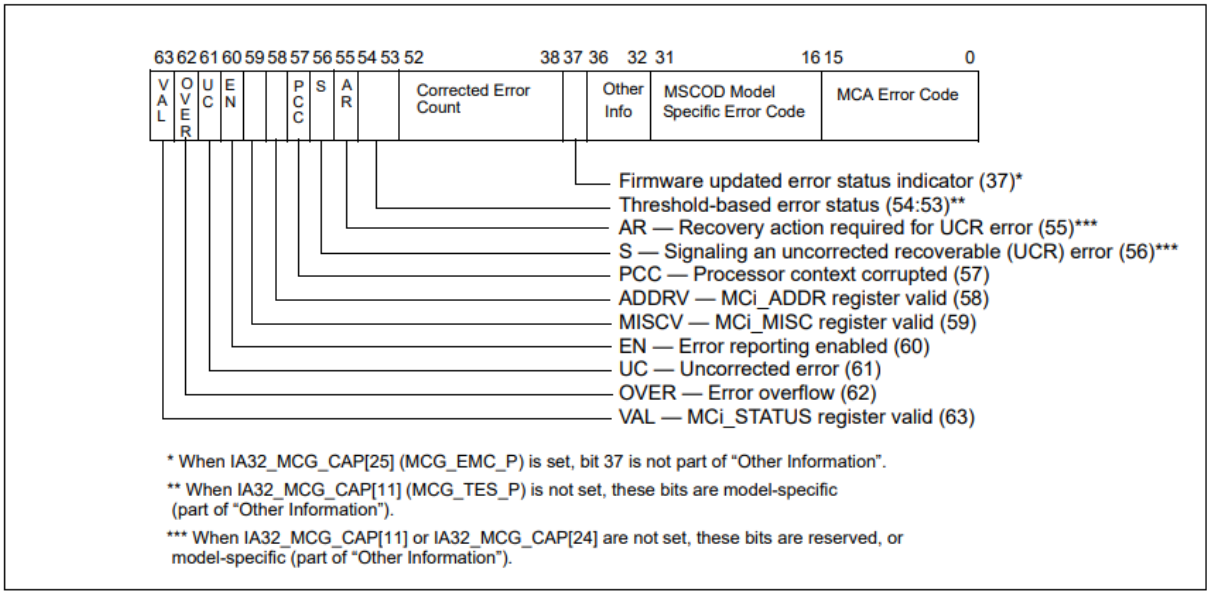


Figure 15-6. IA32_MCi_STATUS Register

MCA (machine-check architecture) error code field, bits 15:0

当 machine-check error被探测到，这里就写入了通用架构相关的 machine-check error code。既然是架构相关，那么所有种类的CPU都会理解这些error code的含义。注意这里也会保证和IA32的兼容。

Model-specific error code field, bits 31:16

指定了 model-specific error code，这种error code是每种CPU都有不同解释的，唯一的标识了一种错误。

Reserved, Error Status, and Other Information fields, bits 56:32

这部分很乱，主要是Other Information用来提供额外信息和error发生的counter累加器用来判断是否超过阀值需要告警， 有兴趣的可以参考SDM。 最重要的是两个位。当IA32_MCG_CAP[11] = 1时， S (Signaling) flag, bit 56 为1 表示需要针对该error report需要发送#MC， AR (Action Required) flag, bit 55为1表明是否需要软件采用recovery Action，这也是区分SRAR和SRAO的地方。

PCC (processor context corrupt) flag, bit 57

当设定了，就意味着整个处理器都被探测到的错误污染了，没法进行修复或者重新执行指令。当没有设置，表明处理器当前还没有被错误污染，软件有可能通过恢复动作让系统正常运行

ADDRV (IA32_MCi_ADDR register valid) flag, bit 58

当设定了，就代表 IA32_MCi_ADDR寄存器中含有发生错误时候的内存地址

MISCV (IA32_MCi_MISC register valid) flag, bit 59

当设定了，表明 IA32_MC*i*_MISC寄存器中包含了错误的额外信息

EN (error enabled) flag, bit 60

当设定了，就说明了改错误在 IA32_MC*i*_CTL寄存器中对应的 EEj位被设定了

UC (error uncorrected) flag, bit 61

当设定了，表明处理器不能依靠硬件来矫正这个错误（如多bit的内存错误），当清除则该错误可以被处理器硬件纠正（如单bit内存错误）

OVER (machine check overflow) flag, bit 62

当设定了，就表明发生了MC嵌套。也就是说 error-reporting register bank 里面还有着前一个错误信息（VAL=1，软件正在处理）时，另外一个错误就发生了。处理器会设置OVER位，软件负责清除。通常按照一定的规则对错误内容进行覆盖。通常来说， enabled errors 覆盖 disabled errors， uncorrected errors 覆盖 corrected errors。 Uncorrected errors不会覆盖前一个有效的 Uncorrected errors

VAL (IA32_MCi_STATUS register valid) flag, bit 63

当设定了，就意味着 IA32_MCi_STATUS中的错误信息是有效的，系统软件正在处理。当该位被设定时，处理器按照覆写规则来处理更多的错误信息。处理器负责设定VAL，系统软件负责清除这个位（当处理完成）。参考SDM。

2.2.3、IA32_MCi_ADDR MSRs

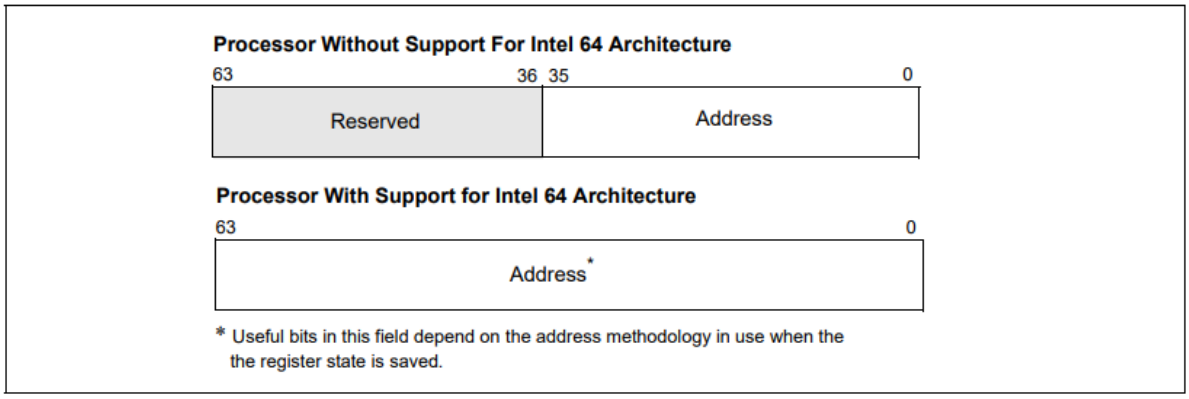


Figure 15-7. IA32_MC*i*_ADDR MSR

当 **IA32_MCi_STATUS. ADDRV = 1**的时候， **IA32_MCi_ADDR**中含有产生本次错误的指令或数据内存的内存地址。当 **IA32_MCi_STATUS. ADDRV = 0**，任何读写 **IA32_MCi_ADDR**的动作都会导致**#GP**

根据遇到的错误不同，返回的地址可能是一个段内偏移量、线性地址或物理地址。可以通过写入全**0**来清除该寄存器，如果在任何一位写入**1**都会导致**#GP**

2.2.4、IA32_MCi_MISC MSRs

当 **IA32_MCi_STATUS. MISCV = 1**时， **IA32_MCi_MISC MSR**中包含了本次 **machine-check error**的额外信息。如果 **IA32_MCi_STATUS. MISCV = 0**，任何读写 **IA32_MCi_MISC MSR**都会导致**#GP**

软件通过显示的写入全**0**来清除该寄存器，当写入任意一位**1**都会导致**#GP**

如果 **MISCV =1** 且 **IA32_MCG_CAP[24] = 1**， **IA32_MCi_MISC_MSR**的定义如下图所示，用来支持软件恢复 **uncorrected errors**

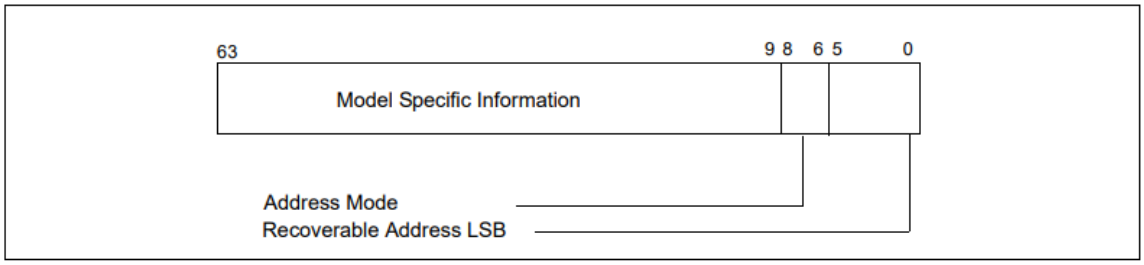


Figure 15-8. UCR Support in IA32_MCi_MISC Register

Recoverable Address LSB (bits 5:0)： 表明**error address**的最低有效位。例如，**IA32_MCi_MISC. LSB = 01001b (9),** 那么 **IA32_MCi_ADDR** 中记录的错误地址**[43:9]**是有效的，**[8:0]**直接忽略。如果是**12**，那么就说明页面是**4K**对其的**Address Mode (bits 8:6): IA32_MCi_ADDR**中记录的地址的类型，支持的类型如下所示

Table 15-3. Address Mode in IA32_MCi_MISC[8:6]

IA32_MCi_MISC[8:6] Encoding	Definition
000	Segment Offset
001	Linear Address
010	Physical Address
011	Memory Address
100 to 110	Reserved
111	Generic

Model Specific Information (bits 63:9): 非架构相关的

2.2.5、 IA32_MCi_CTL2 MSRs

IA32_MCi_CTL2 MSR提供了对于 **corrected MC error**发送信号能力的可编程接口，也同时意味着要求 **IA32_MCG_CAP[10] = 1**。系统软件检查每个**bank**的**IA32_MCi_CTL2**。

当 **IA32_MCG_CAP[10] = 1**，每**bank**的 **IA32_MCi_CTL2 MSR**都存在。但是对于**corrected MC error**不一定是每个**bank**都发送信号的，需要检查相应**bank**的标志位。寄存器如下所示：

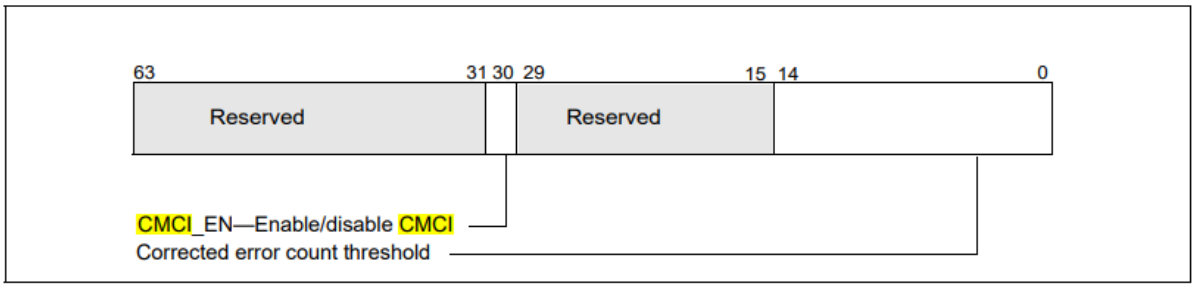


Figure 15-9. IA32_MCi_CTL2 Register

Corrected error count threshold, bits 14:0

系统软件负责初始化这个域，然会会同 IA32_MCi_STATUS[52:38]比较。当IA32_MCi_STATUS[52:38]等于阀值的时候，就会发送一个 overflow事件到APIC的 CMCI LVT entry（ APIC_BASE+02F0H ）。如果CMCI功能没有使能，但是 IA32_MCG_CAP[10] = 1，那么这个域总是0

CMCI_EN (Corrected error interrupt enable/disable/indicator), bits 30

系统软件设定这个位来使能发送 corrected machine-check error interrupt (CMCI) 的特性。如果某bank的CMCI没有使能， 但是 IA32_MCG_CAP[10] = 1，写1到这个bit会返回0，这个BIT也就意味着对应bank的CMCI是否被支持。

某些 微架构作为 corrected MC errors的源，肯能被多个逻辑处理器共享，那么这个bank的内容也就会被共享，软件需要负责 IA32_MCi_CTL2 MSR内容在多个逻辑处理器间的一致性问题。

在系统重启后 IA32_MC*i*_CTL2 MSRs统统被清零。

2.2.6 IA32_MCG Extended Machine Check State MSRs

对于**Intel**志强处理器，当 **IA32_MCG_CAP. MCG_EXT_P = 1** 时， 实现了更多的扩展 **machine-check state MSRs**，可以记录更加详细的错误信息。 **IA32_MCG_CAP. MCG_EXT_CNT**表明了这些扩展寄存器的数量。这些扩展寄存器如下所示

32位的扩展寄存器

Table 15-5. Extended Machine Check State MSRs in Processors Without Support for Intel 64 Architecture

MSR	Address	Description
IA32_MCG_EAX	180H	Contains state of the EAX register at the time of the machine-check error.
IA32_MCG_EBX	181H	Contains state of the EBX register at the time of the machine-check error.
IA32_MCG_ECX	182H	Contains state of the ECX register at the time of the machine-check error.
IA32_MCG_EDX	183H	Contains state of the EDX register at the time of the machine-check error.
IA32_MCG_ESI	184H	Contains state of the ESI register at the time of the machine-check error.
IA32_MCG_EDI	185H	Contains state of the EDI register at the time of the machine-check error.
IA32_MCG_EBP	186H	Contains state of the EBP register at the time of the machine-check error.
IA32_MCG_ESP	187H	Contains state of the ESP register at the time of the machine-check error.
IA32_MCG_EFLAGS	188H	Contains state of the EFLAGS register at the time of the machine-check error.
IA32_MCG_EIP	189H	Contains state of the EIP register at the time of the machine-check error.
IA32_MCG_MISC	18AH	When set, indicates that a page assist or page fault occurred during DS normal operation.

64位的扩展寄存器

Table 15-6. Extended Machine Check State MSRs
In Processors With Support For Intel 64 Architecture

MSR	Address	Description
IA32_MCG_RAX	180H	Contains state of the RAX register at the time of the machine-check error.
IA32_MCG_RBX	181H	Contains state of the RBX register at the time of the machine-check error.
IA32_MCG_RCX	182H	Contains state of the RCX register at the time of the machine-check error.
IA32_MCG_RDX	183H	Contains state of the RDX register at the time of the machine-check error.
IA32_MCG_RSI	184H	Contains state of the RSI register at the time of the machine-check error.
IA32_MCG_RDI	185H	Contains state of the RDI register at the time of the machine-check error.
IA32_MCG_RBP	186H	Contains state of the RBP register at the time of the machine-check error.
IA32_MCG_RSP	187H	Contains state of the RSP register at the time of the machine-check error.
IA32_MCG_RFLAGS	188H	Contains state of the RFLAGS register at the time of the machine-check error.
IA32_MCG_RIP	189H	Contains state of the RIP register at the time of the machine-check error.

Table 15-6. Extended Machine Check State MSRs
In Processors With Support For Intel 64 Architecture (Contd.)

MSR	Address	Description
IA32_MCG_MISC	18AH	When set, indicates that a page assist or page fault occurred during DS normal operation.
IA32_MCG_ RSERVED[1:5]	18BH-18FH	These registers, if present, are reserved.
IA32_MCG_R8	190H	Contains state of the R8 register at the time of the machine-check error.
IA32_MCG_R9	191H	Contains state of the R9 register at the time of the machine-check error.
IA32_MCG_R10	192H	Contains state of the R10 register at the time of the machine-check error.
IA32_MCG_R11	193H	Contains state of the R11 register at the time of the machine-check error.
IA32_MCG_R12	194H	Contains state of the R12 register at the time of the machine-check error.
IA32_MCG_R13	195H	Contains state of the R13 register at the time of the machine-check error.
IA32_MCG_R14	196H	Contains state of the R14 register at the time of the machine-check error.
IA32_MCG_R15	197H	Contains state of the R15 register at the time of the machine-check error.

这些寄存器通过写入全零来进行清除，写入其他值会导致**#GP**。当硬重启时寄存器被清除（ **power-up or RESET**），当软重启这些寄存器的值保留**(INIT reset)**

硬件故障在软件层面处理 **- kernel** 解读。

硬件故障监测工具 **mcelog**

1、mcelog简介

mcelog 是 **x86** 的 **Linux** 系统上用来检查硬件错误，特别是内存和**CPU**错误的工具。

可纠正和不可纠正的硬件错误统称为机器检查异常 **(MCE)**。**CPU** 自身能够纠正错误，并通知底层操作系统与 **CPU** 或缓存有关的问题。**CPU** 本身还能从某些错误中恢复。

mcelog 能捕获两类错误：已纠正的 和未纠正的。已纠正的错误是由 **CPU** 处理的事件，可用来识别可能预测更大问题的趋势。

未纠正的错误是关键异常，如果 **CPU** 无法恢复，往往会导致系统上的内核错误。

这会导致应用程序重置和中断。对于未纠正的错误，**mcelog** 捕获错误的能力取决于错误导致热重启还是硬重启。如果是热重启，信息会被 **mcelog** 捕获，恢复后可看到。硬重启会导致数据丢失，而且 **mcelog** 可能捕获不到该事件。

2、mcelog 安装与配置

2.1 mcelog的安装

源码地址 **URL**

```
$ yum install -y mcelog
```

2.2 mcelog的配置与启动

mcelog的启动方式有以下三种：

1、cron： 定时任务是一种比较老的方式。**Mcelog**每**5**分钟运行一次收集错误，这种方式的缺点就是延迟了错误的上报。

```
$ #!/bin/bash /usr/sbin/mcelog --ignorenodev --filter >>
/var/log/mcelog
```

以上脚本为：在每次定时任务到期执行时运行**/usr/sbin/mcelog**，并将记录的日志信息写到**/var/log/mcelog**文件中。

2、daemon：mcelog默认起来就是**daemon**守护进程的模式。这种模式下**mcelog**持续的作为后台的守护进程运行，并等待着错误的发生。此服务是运行脚本**/etc/init.d/mcelog**启动的。

```
$ mcelog --daemon
$ mcelog --client #查看守护进程是否检测到错误信息
```

3、trigger：触发模式是在错误发生时内核运行**mcelog**。

```
$ echo /usr/sbin/mcelog >
/sys/devices/system/machinecheck/machinecheck0/trigger
```

mce的配置文件

默认故障日志只记录在**/var/log/mcelog**，并不记录到系统日志中。可以通过修改**/etc/mcelog/mcelog.conf** 配置文件对**mcelog**的相关操作进行控制。

```
#
# config file for mcelog
# For further options, see the mcelog manpage and documentation
#

# Filter out known broken events by default
filter = yes
# don't log memory errors individually
#filter-memory-errors = yes

# output in undecoded raw format to be easier machine readable
#raw = yes

[server]
# An upstream bug prevents this from being disabled
# Only allow root to connect by default
client-user = root
# Path to socket client uses to connect
socket-path = /var/run/mcelog-client

[dimm]
# Enable DIMM-tracking
dimm-tracking-enabled = yes
.....................................
```

3、mcelog 日志信息

查看 **/var/log/message** 获取**mcelog**的日志信息。

```
Nov 28 13:33:51 localhost systemd[1]: systemd-hostnamed.service: Deactivated successfully.
Nov 28 13:33:59 localhost kernel: [16839.767719] mce: Machine check injector initialized
Nov 28 13:34:23 localhost kernel: [16861.233619] mce: Starting machine check poll CPU 0
Nov 28 13:34:23 localhost kernel: [16861.233965] mce: Machine check poll done on CPU 0
```

故障注入工具 **mce-inject**

1、mce-inject 简介

mce-inject用于测试**mcelog**能否正确的获取硬件错误信息，并进行正确解码，**mce-inject**可以向内核注入指定的错误信息，因此，可以很方便的了解到**mcelog**的功能是否正常。

这里需要注意的是，当用户利用**mce-inject**工具向内核注入不可恢复错误（如：**fatal**）时，会发生死机重新启动等现象，当然，可以通过更改**sys**文件系统下的**tolerate**文件来避免此现象的发生。

2、硬件故障模拟 - 故障编程

2.1 安装**mce-inject**工具

```
$ yum install -y ras-utils
```

2.2 挂载**mce-inject**内核模块

```
$ modprobe mce-inject
```

2.3 修改**tolerate**文件配置

位置：`/sys/devices/system/machinecheck/machinecheck*/`

machinecheck 中的*号由**CPU**的个数所决定的，如果是双核的，则存在**machinecheck0**和**machinecheck1**两个目录，对应目录里都有一个**tolerate**文件，**tolerate**中存放容忍程度值。

功能：向用户提供一个可选择的出现相应硬件错误时的容忍程度（**tolerate**），比如：当**tolerate**的值为**1**时，出现**fatal**错误时就会死机，重新启动，并且该错误信息并不被记录；当**tolerate**的值为**3**时（注意该值只用于测试），在出现**fatal**错误时，机器会容忍该错误不予响应，不会出现死机重新启动现象，并且会记录相关错误信息。

配置：

```
#cd  /sys/devices/system/machinecheck/machinecheck0
#echo 3 > tolerant
```

> *tolerate的取值可以为0、1、2、3。*
> *0: always panic on uncorrected errors, log corrected errors*
> *1: panic or SIGBUS on uncorrected errors, log corrected errors*
> *2: SIGBUS or log uncorrected errors (if possible), log corrected errors*
> *3: never panic or SIGBUS, log all errors (for testing only)*

3、mce-inject 使用

3.1 构造**cpu**故障代码

```
#cat mce
CPU 1 BANK 2
STATUS corrected
RIP 0x12341234
```

3.2 故障注入

```
$ mce-inject mce
```

3.3 查看日志信息

```
$ tail /var/log/mcelog
```

参考资料

Intel® AVX-512 Instructions
Do Intel® Processors Support MCE (Machine-Check Exception)...
Intel AVX-512 Brief Introduction: Intel AVX-512简介 (gitee.com)
英特尔®指令集扩展技术 (intel.cn)
AVX-512指令集的前世今生 - 知乎 (zhihu.com)
Intel MCA与MCE硬件机制概述 | CN-SEC 中文网
MCA？这是什么东西？我怎么没碰到过？ - 知乎 (zhihu.com)
MACHINE-CHECK 相关的MSR（三） - Error-Reporting Register Banks_leoufung的博客-程序员ITS301 - 程序员ITS301
理解mcelog如何工作 - 苏小北1024 - 博客园 (cnblogs.com)
MACHINE-CHECK 相关的MSR（三） - Error-Reporting Register Banks_leoufung的博客-程序员ITS301 - 程序员ITS301
(63条消息) Mcelog笔记*weijitao*的博客-*CSDN*博客mcelog