第 2 章 MCU

目录

本章包括下列主题：

2.1 简介

PIC32MX 单片机 （MCU）是一款基于 MIPS® Technologies 的 M4K™ 内核的复杂片上系统。M4K™ 是最新型的 32 位低功耗 RISC 处理器内核，采用了增强型 MIPS32® Release 2 指令集架构。本节概述 PIC32MX 系列单片机的 CPU 特性和系统架构。

主要特性

- 最高可达到 1.5 DMIPS/MHz 的性能
- 可编程预取高速缓存存储器，以增强闪存中的执行效率
- 16 位指令模式 （MIPS16e），用于紧凑型代码
- 带有 63 个优先级的向量中断控制器
- 可编程的用户和内核工作模式
- 可对外设寄存器执行原子级位操作 （单周期）
- 乘法 / 除法单元，最高指令发出速率为每个时钟一条 32 × 16 乘法指令
- 高速 Microchip ICD 端口，具有基于硬件的非侵入式数据监视和应用程序数据流功能
- EJTAG 调试端口，支持广泛的第三方调试、编程和测试工具
- 指令控制的功耗管理模式
- 5 级流水线指令执行
- 内部代码保护，以帮助保护知识产权

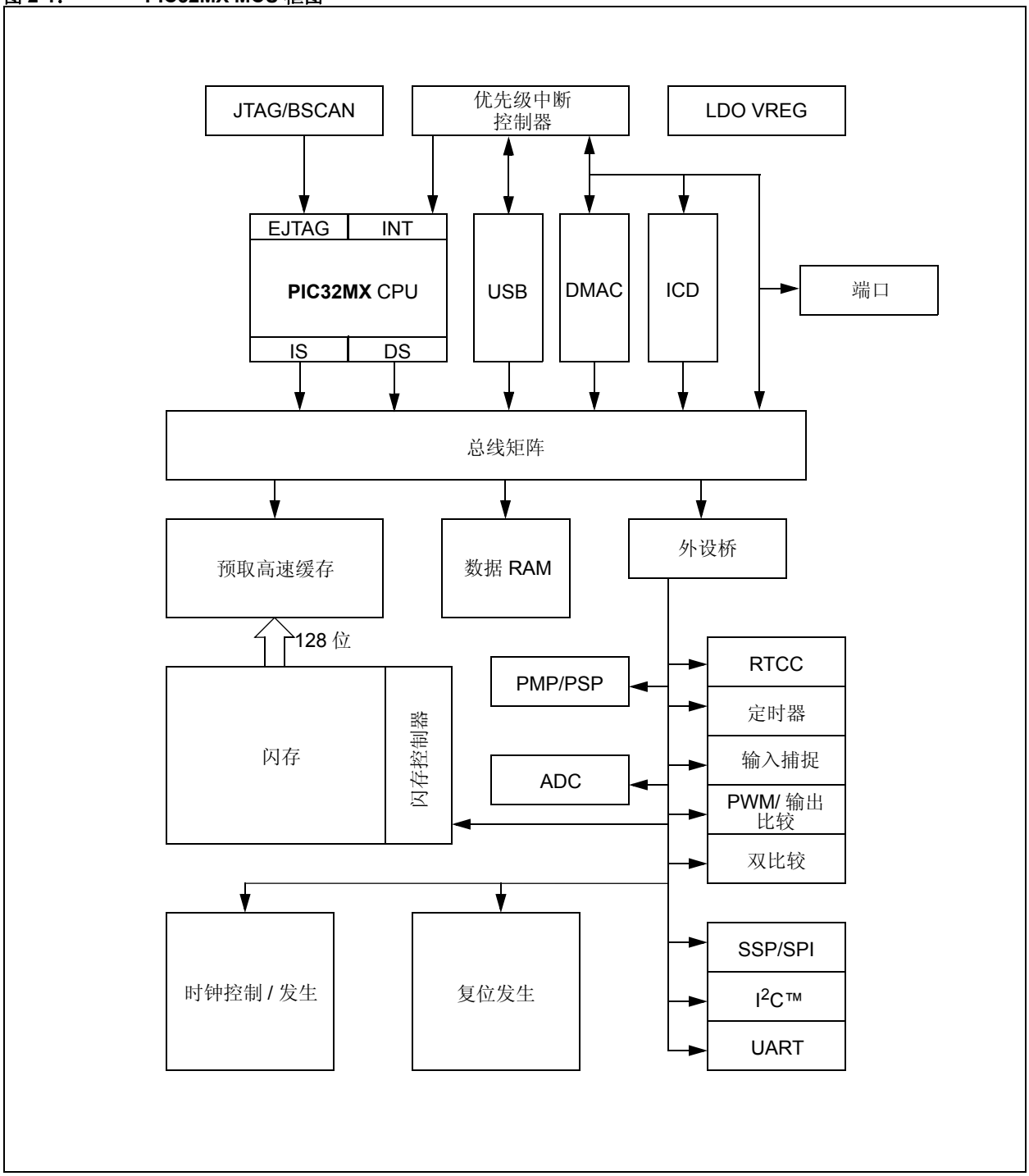
相关的 MIPS® 文档

- MIPS M4K™ Software User's Manual – MD00249-2B-M4K-SUM
- MIPS® Instruction Set – MD00086-2B-MIPS32BIS-AFP
- MIPS16e™ – MD00076-2B-MIPS1632-AFP
- MIPS32® Privileged Resource Architecture – MD00090-2B-MIPS32PRA-AFP

2.2 架构概述

PIC32MX 系列处理器是复杂的片上系统，包含许多特性。PIC32MX 系列的所有处理器中都包含了高性能 RISC CPU，可以使用 32 位、16 位模式，乃至混合模式进行编程。PIC32MX MCU 包含了高性能中断控制器、DMA 控制器、USB 控制器、在线调试器、用于对外设进行高速数据访问的高性能开关矩阵，以及用于保存数据和程序的片上数据 RAM 存储器。对于闪存，采用了独特的预取高速缓存和预取缓冲区，无需闪存访问延时，提供相当于 0 个等待状态的访问性能。

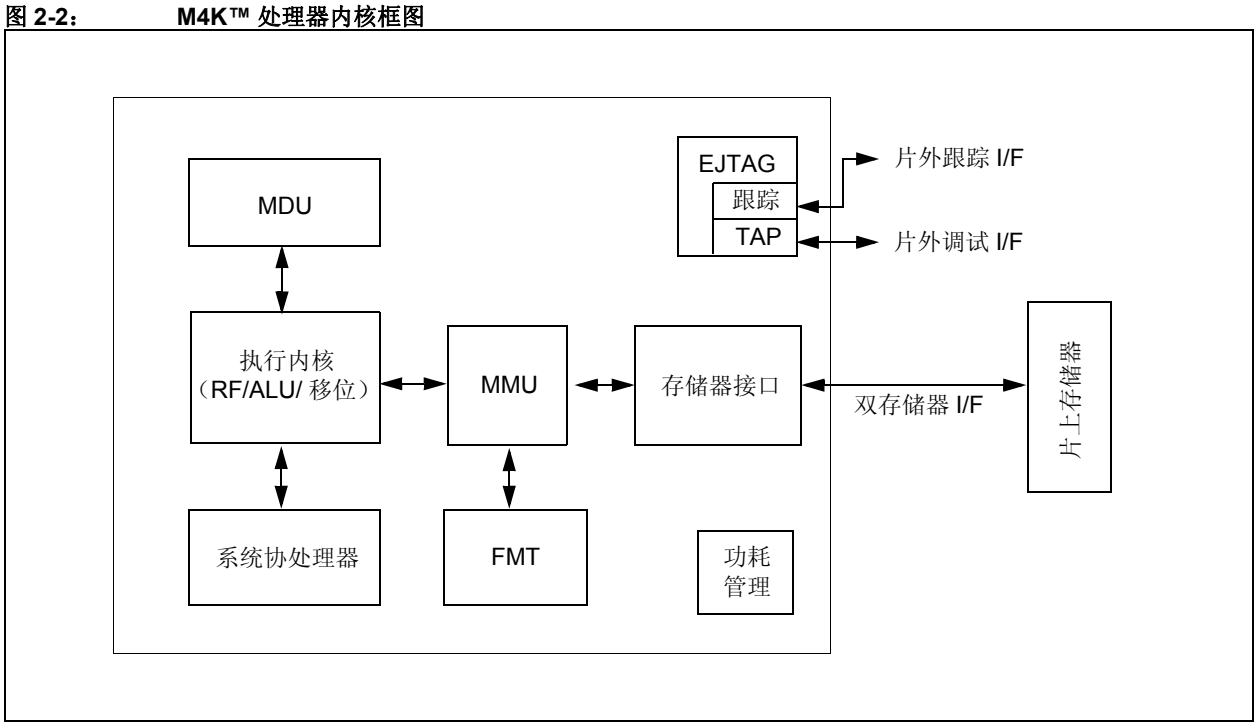
图 2-1： PIC32MX MCU 框图



芯片中有两条内部总线，用于连接所有外设。主外设总线通过外设桥将大部分外设单元与总线矩阵进行连接。此外，还有连接中断控制器、DMA 控制器、在线调试器和 USB 外设的高速外设桥。PIC32MX MCU 的核心是 M4K CPU 内核。 CPU 在程序控制下执行操作。 CPU 会同时进行指令的取指、解码和执行。指令位于程序闪存存储器或数据 RAM 存储器中。

PIC32MX CPU 基于装载 / 存储架构，大多数操作都是对一组内部寄存器执行。它使用特定的装载和存储指令在这些内部寄存器和外界之间传送数据。

图 2-2： M4K™ 处理器内核框图



2.2.1 总线

PIC32MX MCU 上有两条独立的总线。一条总线负责为 CPU 取指，另一条总线是装载和存储指令的数据路径。指令总线（或 I 侧总线）和数据总线（或 D 侧总线）连接到总线矩阵单元。总线矩阵是一个开关，支持在系统中同时进行多个访问。通过总线矩阵，不访问同一目标设备的多个不同总线主设备可以同时进行访问。多个不同主设备访问同一目标设备时，总线矩阵可以通过仲裁算法将这些访问串行。

由于 CPU 到总线矩阵具有两条不同的数据路径，所以对于系统来说， CPU 实际上是两个不同的总线主设备。从闪存中运行代码时，对 SRAM 和内部外设的装载和存储操作将与对闪存的取指操作并行进行。

除了 CPU 之外，在 PIC32MX MCU 中还有三个其他总线主设备——DMA 控制器、在线调试器单元和 USB 控制器。

2.2.2 编程模型简介

PIC32MX 处理器具有以下特性：

- 5 级流水线
- 32 位地址和数据路径
- 类似于 DSP 的乘加和乘减指令 （MADD、 MADDU、 MSUB 和 MSUBU）
- 目标乘法指令 （MUL）
- 0/1 检测指令 （CLZ 和 CLO）
- 等待指令 （WAIT）
- 条件传送指令 （MOVZ 和 MOVN）
- 实现了 MIPS32 增强型架构 （Release 2）
- 向量中断
- 可编程异常向量基址
- 原子级中断允许 / 禁止
- 通用寄存器 （General Purpose Register， GPR）影子集
- 位域操作指令
- MIPS16e 应用特定扩展，可提高代码密度
- 相对于 PC 的特殊指令，用于有效装载地址和常量
- 数据类型转换指令 （ZEB、 SEB、 ZEH 和 SEH）
- 紧凑型跳转 （JRC 和 JALRC）
- 堆栈帧建立和拆除 “宏” 指令 （SAVE 和 RESTORE）
- 存储器管理单元，带有简单固定映射转换 （Fixed Mapping Translation， FMT）功能
- 处理器与协处理器寄存器之间的数据传输
- 存储器与协处理器寄存器之间的直接数据传输
- 性能优化的乘法 / 除法单元 （高性能构建时选项）
- 最高指令发出速率为每个时钟一条 32 × 16 乘法指令
- 最高指令发出速率为每隔一个时钟一条 32 × 32 乘法指令
- 提早除法控制——将除法周期缩短 11 至 34 个时钟
- 低功耗模式 （由 WAIT 指令触发）
- 通过 SDBBP 指令设置软件断点

2.2.3 内核定时器

PIC32MX 架构包含了一个可供应用程序使用的内核定时器。该定时器通过两个协处理器寄存器实现——计数寄存器 （CP0_COUNT）和比较寄存器 （CP0_COMPARE）。计数寄存器每两个系统时钟 （SYSCLK）周期递增一次。在 Debug （调试）模式期间，可以选择暂停计数递增。比较寄存器用于产生定时器中断 （如需要）。当比较寄存器与计数寄存器匹配时，将产生中断。只有在中断控制器中允许某个中断时，才会执行该中断。

关于内核定时器的更多信息，请参见**第 2.12 节 “CP0 寄存器”**和**第 8 章 “中断”**（DS61108）。

2.3 PIC32MX CPU 详细信息

2.3.1 流水线分级

流水线包含 5 级：

- 指令 （I）级
- 执行 （E）级
- 存储 （M）级
- 对齐 （A）级
- 回写 （W）级

2.3.1.1 I 级——取指

在 I 级期间：

• 从指令 SRAM 取指。

• MIPS16e 指令转换为类似于 MIPS32 的指令。

2.3.1.2 E 级——执行

在 E 级期间：

- 从寄存器文件中取操作数。
- M 和 A 级的操作数旁路传递到此级。
- 对于寄存器 - 寄存器指令，算术逻辑单元（Arithmetic Logic Unit， ALU）开始执行算术或逻辑运算。
- 对于装载和存储指令， ALU 会计算数据虚拟地址， MMU 执行虚拟地址到物理地址的固定转换。
- ALU 确定转移条件是否为真，并计算转移指令的虚拟转移目标地址。
- 指令逻辑选择指令地址，并且 MMU 执行虚拟地址到物理地址的固定转换。
- 所有乘法 / 除法运算都在此级开始。

2.3.1.3 M 级——从存储器取数据

在 M 级期间：

- 算术或逻辑 ALU 运算完成。
- 对于装载和存储指令，将执行数据 SRAM 访问。
- 阵列中的 16 × 16 或 32 × 16 乘法运算完成，并在 M 级停留一个时钟，以在 M 级中完成进位传输加法。
- 32 × 32 乘法运算会在 M 级停留两个时钟，以在 M 级中完成阵列第二个周期和进位传输加法。
- 乘法和除法计算在 MDU 中进行。如果计算在 IU 将指令推移到 M 级之前完成，那么 MDU 会将结果保存在临时寄存器中，直到 IU 将指令推移到 A 级 （因此将不会被中止）。

2.3.1.4 A 级——对齐

在 A 级期间：

- 独立的对齐器将装载数据与其字边界对齐。
- 乘法运算产生可供回写的结果。实际的寄存器回写在 W 级执行。
- 从此级开始，装载数据或来自 MDU 的结果可通过旁路传递到 E 级。

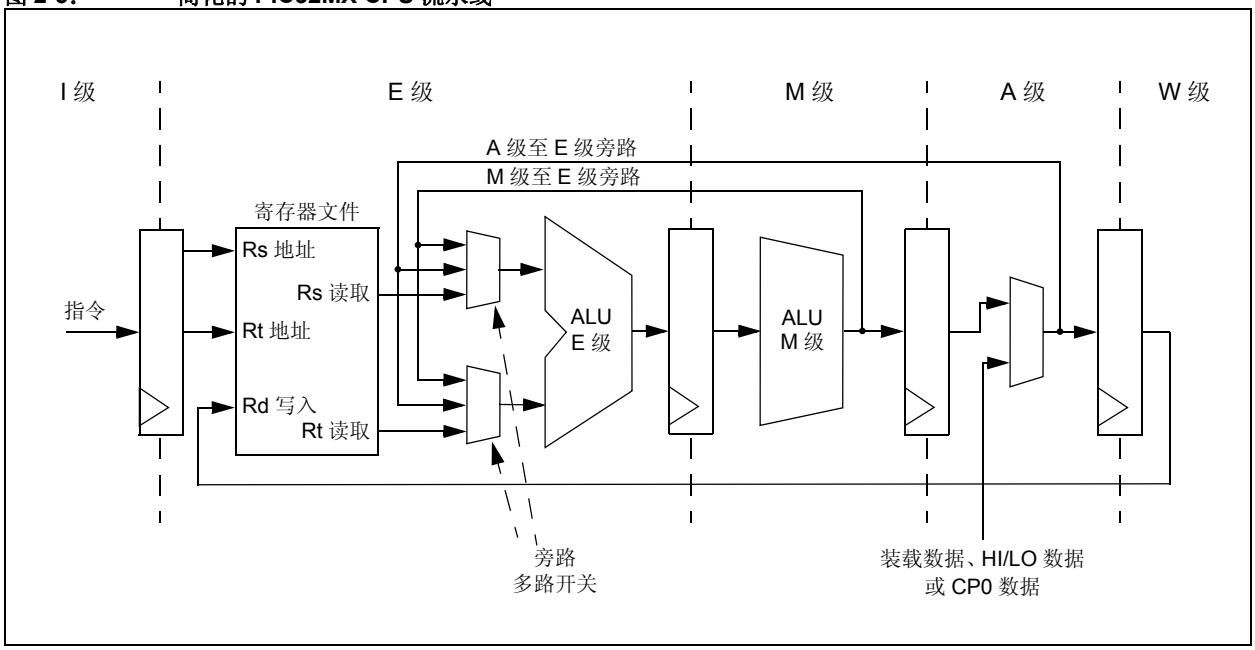
2.3.1.5 W 级——回写

在 W 级期间：

对于寄存器 - 寄存器指令或装载指令，结果回写到寄存器文件中。

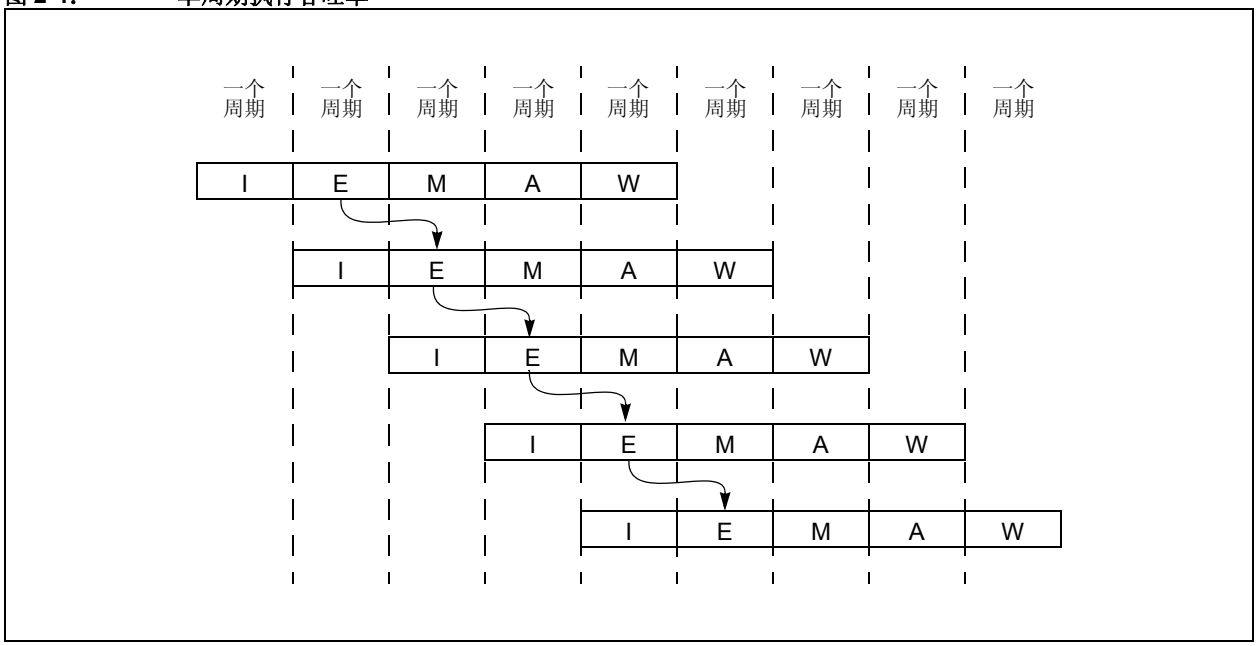
M4K 内核实现了 “旁路” 机制，通过该机制可以将运算结果直接送到需要它的指令处，而无需先将结果写入寄存器，然后再进行回读。

图 2-3： 简化的 PIC32MX CPU 流水线



在 PIC32MX 内核中使用指令流水线实现了一个快速的单周期指令执行环境。

图 2-4： 单周期执行吞吐率



2.3.2 执行单元

PIC32MX 执行单元负责执行 MIPS 指令集大部分指令的处理。执行单元通过流水线执行的方式，为大部分指令提供了单周期吞吐率。流水线执行将复杂的操作拆分为称作 “级”的小片段。这些操作级在多个时钟周期中执行。

执行单元包含以下特性：

- 32 位加法器，用于计算数据地址
- 地址单元，用于计算下一条指令的地址
- 逻辑单元，用于进行转移判断和转移目标地址计算
- 装载对齐器
- 旁路多路开关，用于避免执行指令流时 （当数据生成指令后紧跟使用其结果的指令时）出现停顿
- 前导 0/1 检测单元，用于实现 CLZ 和 CLO 指令
- 算术逻辑单元 （ALU），用于执行按位逻辑运算
- 移位器和存储对齐器

2.3.3 MDU

乘法 / 除法单元用于执行乘法和除法运算。 MDU 包含 32 × 16 乘法器、结果累加寄存器（HI 和 LO） 、乘法和除法状态机，以及执行这些功能所需的所有多路开关和控制逻辑。高性能流水线 MDU 支持在每个时钟周期执行一次 16 × 16 或 32 × 16 乘法运算； 32 × 32 乘法运算可以每隔一个时钟周期发出一次。它采用适当的互锁机制来阻止发出背靠背 32 × 32 乘法运算。除法运算通过简单的每时钟 1 位迭代算法实现，最坏情况下需要 35 个时钟周期才能完成。提早算法会检测被除数的符号扩展，如果它的实际大小为 24、 16 或 8 位，则除法器将会跳过 32 次迭代中的 7、 15 或 23 次迭代。在除法器仍然工作时尝试发出后续的 MDU 指令会导致流水线停顿，直到除法运算完成为止。

M4K 还另外实现了一条乘法指令 MUL，该指令规定乘法结果的低 32 位放入寄存器文件中，而不是 HI/LO 寄存器对中。通过避免显式的从 LO 传送（MFLO）指令（使用 LO 寄存器时需要），并通过支持多个目标寄存器，乘法密集运算的吞吐率可以提高。乘加 （MADD/MADDU）和乘减（MSUB/MSUBU）这两条指令用于执行乘加和乘减运算。MADD 指令可以将两个数字相乘，然后将乘积与 HI 和 LO 寄存器的当前内容相加。类似地，MSUB 指令可以将两个操作数相乘，然后从 HI 和 LO 寄存器内容中减去乘积。MADD/MADDU 和 MSUB/MSUBU 运算常用于数字信号处理器（Digital Signal Processor， DSP）算法。

2.3.4 影子寄存器集

PIC32MX 处理器实现了通用寄存器 （GPR）的一个副本，供高优先级中断使用。这个额外的寄存器存储区称为影子寄存器集。当发生高优先级中断时，处理器可以无需软件干预而自动切换到影子寄存器集。这可以降低中断处理程序中的开销，并降低实际响应延时。

影子寄存器集由位于系统协处理器 （CP0）中的寄存器和位于 CPU 内核之外的中断控制器硬件控制。

关于影子寄存器集的更多信息，请参见 “中断”章节。

2.3.5 流水线互锁处理

当某个流水线级中的指令由于数据相依性或类似外部条件而无法前移时，流畅的流水线流程会被中断。流水线中断完全在硬件中进行处理。这些相依性称为*互锁*。在每个周期，都会为所有执行中的指令检查互锁条件。互锁条件的一个示例就是一条指令依赖于前一条指令的结果。

通常， MIPS 处理器支持两种类型的硬件互锁：

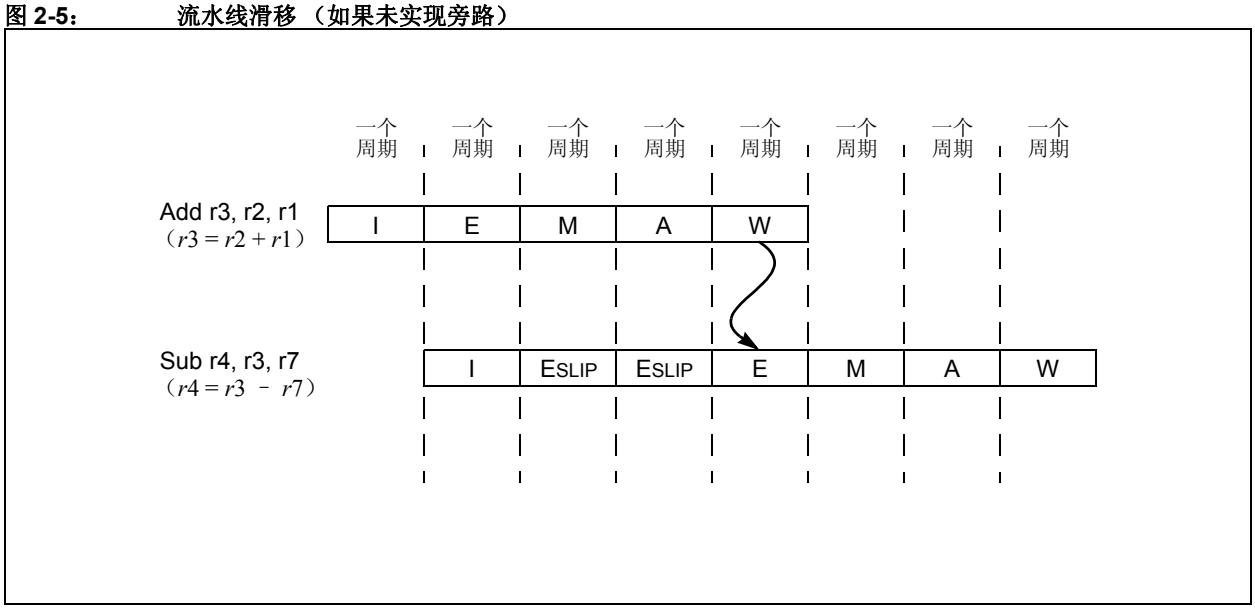
- 停顿 （Stall）
  停顿通过暂停整个流水线来解决。当前正在每个流水线级中执行的所有指令都会受到停顿的影响。
- 滑移 （Slip）
  滑移允许流水线中有一部分前移，而流水线的其他部分保持静态。

在 PIC32MX 处理器内核中，所有互锁都以滑移形式进行处理。为了最大程度减少滑移，通过使用称作寄存器旁路的方法 （下面将介绍该方法）来从其他流水线级抓取结果。

注：	为了说明流水线滑移的概念，以下示例给出了 PIC32MX 内核未实现寄存器旁路时所发生情况的图示。

如图 2-5 所示，对于寄存器 r3， SUB 指令与前一条 ADD 指令存在源操作数相依性。 SUB 指令会滑移两个时钟，等待 ADD 的结果回写到寄存器 r3 中。在 PIC32MX 系列处理器上，*不会*发生这种滑移情况。

图 2-5： 流水线滑移 （如果未实现旁路）



2.3.6 寄存器旁路

如前面所述， PIC32MX 处理器实现了一种称作寄存器旁路的机制，帮助减少执行期间的流水线滑移。当某条指令处于流水线 E 级时，只有在操作数可用的情况下，该指令才能继续。如果指令的某个源操作数需要通过执行流水线中的另一条指令计算获得，通过寄存器旁路可以产生一条快捷路径，直接从流水线获取源操作数。处于 E 级的指令可以从在流水线 M 级或 A 级中执行的另一条指令处获取源操作数。如图 2-6 所示，三条指令组成的序列存在相依性，但它们在执行期间完全未发生滑移。该示例使用了 A 级至 E 级和 M 级至 E 级的寄存器旁路。图 2-7 显示了一条利用 A 级至 E 级旁路的装载指令的操作。因为装载指令的结果直到流水线 A 级才可用，所以不需要 M 级至 E 级旁路。

寄存器旁路的性能优点是，即使存在寄存器相依性，ALU 运算的指令吞吐率也可以上升为每个时钟一条指令。

图 2-6： IU 流水线 M 级至 E 级旁路

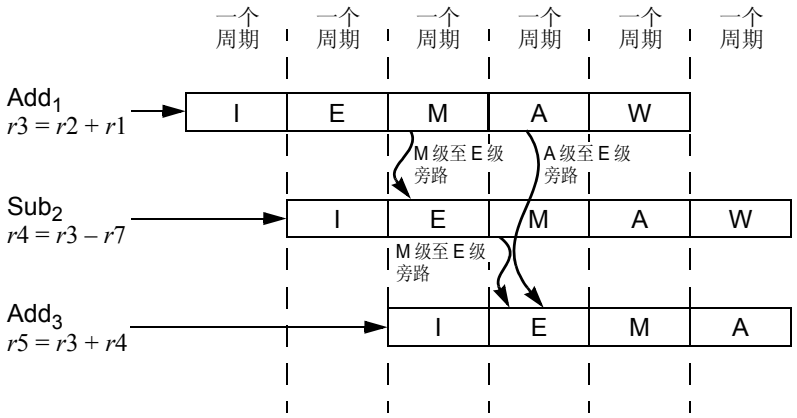
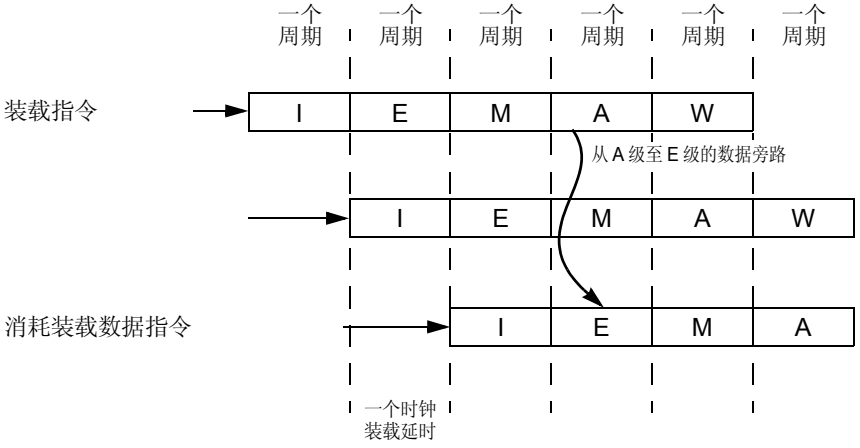


图 2-7： IU 流水线 A 级至 E 级数据旁路



2.4 写入 CP0 寄存器时的特殊注意事项

通常， PIC32MX 内核会确保指令按照完全顺序性的程序模型执行。程序中的每条指令都可以看到前一条指令的结果。但也存在一些背离该模型的情况。这些背离情况称为*冒险（hazard）*。

在特权软件中，存在两种不同的冒险：

- 执行冒险
- 指令冒险

2.4.1 执行冒险

执行冒险在执行一条指令时产生，另一条指令在执行时看到。表 2-1 列出了这些执行冒险。

表 2-1： 执行冒险

产生方	=	消耗方	冒险源自	间距 （指令数）
MTC0	=	依赖于 $Status_{CU}$ 新值的协处理器指令执行	$Status_{CU}$	1
MTC0	=	ERET	EPC DEPC ErrorEPC	1
MTC0	=	ERET	Status	0
MTC0, EI, DI	=	被中断的指令	$Status_{IE}$	1
MTC0	=	被中断的指令	$Cause_{IP}$	3
MTC0	=	RDPGPR WRPGPR	$SRSCtl_{PSS}$	1
MTC0	=	未看到定时器中断的指令	清除定时器中断的比较更新	4
MTC0	=	受更改影响的指令	任何其他 CP0 寄存器	2

2.4.2 指令冒险

指令冒险在执行一条指令时产生，另一条指令在取指时看到。表 2-2 列出了这些指令冒险。

表 2-2： 指令冒险

产生方	=	消耗方	冒险源自
MTC0	=	看到新值的取指操作 （包括对 ERL 进行更改，然后从 useg 段取指）	Status

2.5 RELEASE 2 架构的详细信息

PIC32MX CPU 采用了 MIPS 32 位的 Release 2 架构。PIC32MX CPU 实现了以下 Release 2 特性：

- 使用外部至内核的中断控制器实现向量中断
  能够将向量中断直接跳转到该中断的处理程序。
- 可编程异常向量基址
  对于在 $Status_{BEV}$ 为 0 时发生的异常，允许移动异常向量的基址。因此，所有系统都可以将异常向量放在适合于系统环境的存储器中。
- 原子级中断允许 / 禁止
  增加了两条指令，可在原子级允许 / 禁止中断，并返回 *Status* 寄存器的先前值。
- 对于功耗高度敏感的应用，可以禁止 *Count* 寄存器。
- GPR 影子寄存器
  提供了附加的 GPR 影子寄存器，并且可以将这些寄存器与向量中断或异常绑定。
- Field、 Rotate 和 Shuffle 指令
  另外增加了处理寄存器位域的功能。
- 显式冒险管理
  提供了一组指令来显式地管理冒险，代替基于周期的 SSNOP 冒险处理方法。

2.6 拆分式 CPU 总线

PIC32MX CPU 内核具有两条不同总线，用于帮助实现优于单总线系统的系统性能。这种性能改善通过并行操作实现。装载和存储操作在执行取指操作的同时发生。这两条总线称为 I 侧总线（用于为 CPU 输送指令）和 D 侧总线 （用于数据传输）。

CPU 在流水线 I 级期间取指。取指请求发送到 I 侧总线，并由总线矩阵单元进行处理。根据地址，BMX 将执行以下操作之一：

- 将取指请求转发到预取高速缓存单元
- 将取指请求转发到 DRM 单元，或
- 产生异常

无论取指地址为何，取指操作总是使用 I 侧总线。 BMX 根据地址和 BMX 寄存器中的值来确定要为每个取指请求执行的操作。（请参见 BMX 章节）。

D 侧总线负责处理由 CPU 执行的所有装载和存储操作。在执行装载或存储指令时，该请求将通过 D 侧总线传给 BMX。该操作在流水线 M 级期间发生，并传给以下几个目标设备之一：

- 数据 RAM
- 预取高速缓存 / 闪存
- 快速外设总线 （中断控制器、 DMA、调试单元、 USB 和 GPIO 端口）
- 通用外设总线（UART、SPI、闪存控制器、EPMP/EPSP、TRCC 定时器、输入捕捉、PWM/输出比较、 ADC、双比较、 I^2C、时钟 SIB 和复位 SIB）

2.7 内部系统总线

PIC32MX 处理器内部总线将外设与总线矩阵单元相连接。总线矩阵会利用芯片中的几条数据路径，将来自 5 个不同发起器的总线访问请求传给一组目标，以帮助消除性能瓶颈。

总线矩阵使用的路径中有一部分是专用路径，而其他路径则由若干个目标共用。

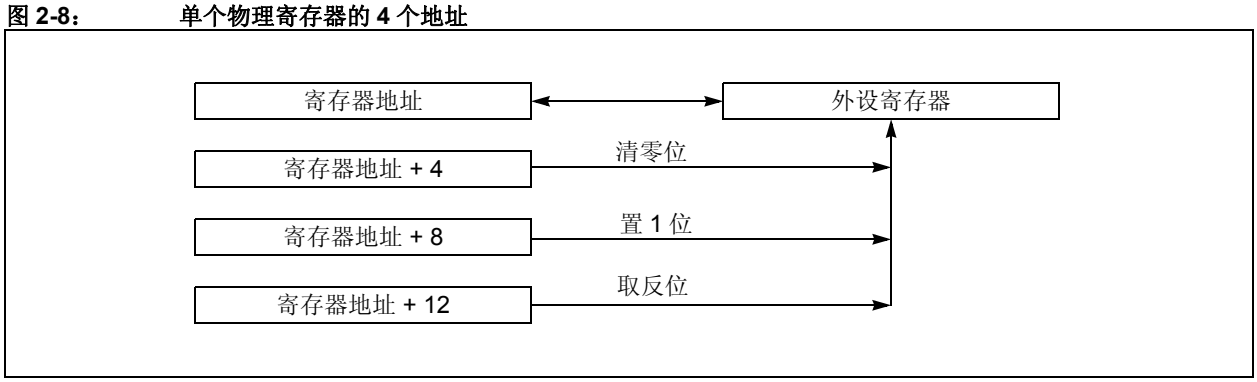
数据 RAM 和闪存读取路径是专用路径，通过它们，对于存储器资源的访问不会被外设总线活动延迟，实现很短的访问延时。高带宽外设放置在高速总线上。它们包括中断控制器、调试单元、DMA 引擎和 USB 主机 / 外设单元。

不需要高带宽的外设位于独立的外设总线上，以节省功耗。

2.8 置 1/ 清零 / 取反

为了对外设进行单周期位操作，可以根据外设地址，使用三种不同方式访问外设单元中的寄存器。每个寄存器都具有 4 个不同的地址。虽然这 4 个不同地址显现为不同寄存器，它们实际上只是同一物理寄存器的 4 种不同寻址方法。

图 2-8： 单个物理寄存器的 4 个地址



基址寄存器地址用于正常的读 / 写访问，其他三个地址用于提供特殊的只写功能。

1. 正常访问
2. 置 1 位原子级 RMW 访问
3. 清零位原子级 RMW 访问
4. 取反位原子级 RMW 访问

外设读取必须通过每个外设寄存器的基址进行。读取置 1/ 清零 / 取反地址的含义是未定义的，对于每个外设可能不同。

对基址执行写操作时，将向外设寄存器中写入完整的值。所有的位都会被写入。例如，假设在写入 0x000000ff 之前，某个寄存器包含 0xaaaa5555。写入之后，寄存器将包含 0x000000ff（假定所有位均为读 / 写位）。

对任意外设寄存器的置 1 地址执行写操作时，只有对于写入 1 的位，目标寄存器中的对应位才会置 1。例如，假设在向置 1 寄存器地址中写入 0x000000ff 之前，某个寄存器包含 0xaaaa5555。写入置 1 寄存器地址之后，外设寄存器的值将包含 0xaaaa55ff。

对任意外设寄存器的清零地址执行写操作时，只有对于写入 1 的位，目标寄存器中的对应位才会清零。例如，假设在向清零寄存器地址中写入 0x000000ff 之前，某个寄存器包含 0xaaaa5555。写入清零寄存器地址之后，外设寄存器的值将包含 0xaaaa5500。

对任意外设寄存器的取反地址执行写操作时，只有对于写入 1 的位，目标寄存器中的对应位才会取反 （或翻转） 。例如，假设在向取反寄存器地址中写入 0x000000ff 之前，某个寄存器包含 0xaaaa5555。写入取反寄存器之后，外设寄存器的值将包含 0xaaaa55aa。

2.9 ALU 状态位

不同于大多数其他 PIC® 单片机，PIC32MX 处理器不使用 STATUS 寄存器标志。许多处理器中都使用条件标志，帮助在程序执行期间执行判定操作。标志是否置 1 取决于比较操作或一些算术运算的结果。然后，这些单片机上的条件转移指令会根据一组条件代码的值作出判定。

但 PIC32MX 处理器改为使用一些指令，这些指令执行比较操作，然后将标志或值存储到通用寄存器中。然后，使用该通用寄存器作为操作数来执行条件转移。

2.10 中断和异常机制

PIC32MX 系列处理器实现了高效灵活的中断和异常处理机制。中断和异常的相似之处在于，当前指令流会临时改变，以执行特殊的过程来处理中断或异常。两者的区别在于，中断通常是正常操作产生的结果，而异常则是错误条件 （例如总线错误）产生的结果。

当发生中断或异常时，处理器会执行以下操作：

1. 将要在处理程序返回之后执行的下一条指令的 PC 保存到协处理器寄存器中
2. 更新 Cause 寄存器，以反映异常或中断的原因
3. 将 Status 寄存器的 EXL 或 ERL 置 1，以在内核模式执行
4. 根据 EBASE 和 SPACING 值计算处理程序 PC
5. 处理器从新 PC 处开始执行

这是对中断和异常机制的简单概述。关于中断和异常处理的更多信息，请参见**第 8 章 “中断”**（DS61108）。

2.11 编程模型

PIC32MX 系列处理器设计为与高级语言 （例如 C 编程语言）配合使用。它支持多种数据类型，并使用高级语言所需的简单而灵活的寻址模式。它有 32 个通用寄存器，以及两个用于乘法和除法的特殊寄存器。

对于 PIC32MX 处理器上的机器语言指令，有三种不同的格式：

- 立即数 （I 类） CPU 指令
- 跳转 （J 类） CPU 指令，以及
- 寄存器 （R 类） CPU 指令

大多数操作都在寄存器中执行。寄存器类型的 CPU 指令具有三个操作数；即两个源操作数和一个目标操作数。

具有三个操作数和很大的寄存器集可以让汇编语言编程器和编译器高效地使用 CPU 资源。中间结果可以保留在寄存器中，不需要常常从存储器来回传送数据，从而可以得到速度更快和长度更短的程序。

立即数格式指令具有一个立即操作数、一个源操作数和一个目标操作数。跳转指令具有一个 26 位相对指令偏移字段，用于计算跳转目标。

2.11.1 CPU 指令格式

CPU 指令是单个 32 位对齐的字。以下显示了 CPU 指令格式：

- 立即数 （见图 2-9）
- 跳转 （见图 2-10）
- 寄存器 （见图 2-11）

表 2-3 列出了在这些指令中使用的字段。

表 2-3： CPU 指令格式字段

字段	说明
opcode	6 位主操作码
rd	目标寄存器的 5 位说明符
rs	源寄存器的 5 位说明符
rt	目标 （源 / 目标）寄存器的 5 位说明符，或用于指定主操作码 REGIMM 中的功能
immediate	16 位有符号立即数，用作逻辑操作数、算术有符号操作数、装载 / 存储地址字节偏移和 PC 相对转移指令的有符号指令位移
instr_index	26 位索引，左移 2 位来提供跳转目标地址的低 28 位
sa	5 位移位量
function	6 位功能字段，用于指定主操作码 SPECIAL 中的功能

图 2-9： 立即数 （I 类） CPU 指令格式

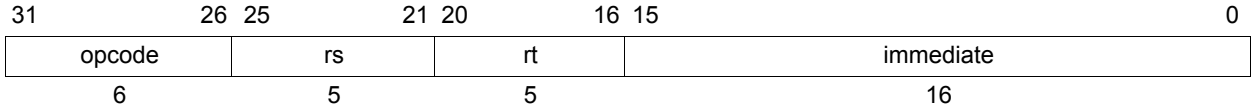


图 2-10： 跳转 （J 类） CPU 指令格式

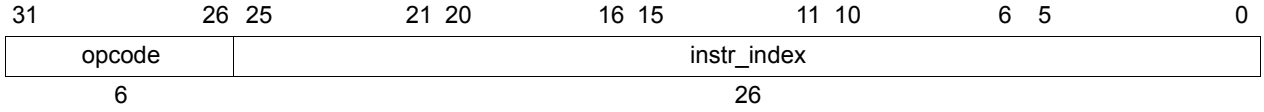
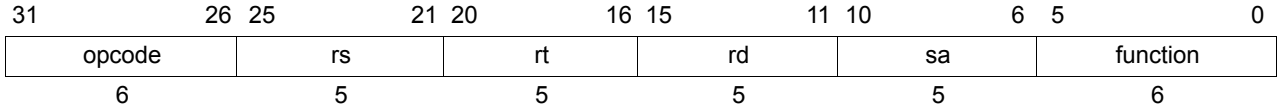


图 2-11： 寄存器 （R 类） CPU 指令格式



2.11.2 CPU 寄存器

PIC32MX 架构定义了以下 CPU 寄存器：

- 32 个 32 位通用寄存器 （GPR）
- 2 个特殊用途寄存器，用于保存整数乘法、除法和乘法累加运算的结果 （HI 和 LO）
- 1 个特殊用途程序计数器 （Program Counter， PC），它只会受一些特定指令间接影响——它在架构中是一个不可见的寄存器。

2.11.2.1 CPU 通用寄存器

CPU 通用寄存器中的两个寄存器具有指定的功能：

- r0

  r0 硬连接到值 0，可以用作结果将被丢弃的任何指令的目标寄存器。在需要值 0 时，r0 也可以用作源寄存器。

- r31

  r31 是 JAL、BLTZAL、BLTZALL、BGEZAL 和 BGEZALL 使用的目标寄存器，无需在指令字中明确指定。其他情况下， r31 用作一般寄存器。

其他寄存器可供一般用途使用。

2.11.2.2 寄存器约定

虽然 PIC32MX 架构中的大部分寄存器都指定为通用寄存器，但为了通过高级语言正确执行软件操作 （例如 Microchip C 编译器），建议按下表方式使用寄存器。

表 2-4： 寄存器约定

CPU 寄存器	符号寄存器	用法
r0	zero	总是为 0(1)
r1	at	汇编器临时量
r2 - r3	v0-v1	函数返回值
r4 - r7	a0-a3	函数参数
r8 - r15	t0-t7	临时量——调用程序不需要保留内容
r16 - r23	s0-s7	保存的临时量——调用程序必须保留内容
r24 - r25	t8 - t9	临时量——调用程序不需要保留内容
r26 - r27	k0 - k1	内核临时量——用于中断和异常处理
r28	gp	全局指针——用于快速访问常用数据
r29	sp	堆栈指针——软件堆栈
r30	s8 或 fp	保存的临时量——调用程序必须保留内容，*或者* 帧指针——指向堆栈中过程帧的指针
r31	ra	返回地址 (1)

注 1： 硬件强制实现，不仅仅是约定。

2.11.2.3 CPU 特殊用途寄存器

CPU 包含三个特殊用途寄存器：

- PC——程序计数器寄存器
- HI——乘法和除法寄存器高位字结果
- LO——乘法和除法寄存器低位字结果
  - 在乘法运算期间， HI 和 LO 寄存器存储整数乘法的乘积。
  - 在乘加或乘减运算期间， HI 和 LO 寄存器存储整数乘加或乘减的结果。
  - 在除法运算期间，HI 和 LO 寄存器存储整数除法的商（在 LO 中）和余数（在 HI 中）。
  - 在乘法累加运算期间， HI 和 LO 寄存器存储运算的累加结果。

图 2-12 给出了 CPU 寄存器的布局。

表 2-5： CPU 寄存器

31	0
r0 （zero）	
r1 （at）	
r2 （v0）	
r3 （v1）	
r4 （a0）	
r5 （a1）	
r6 （a2）	
r7 （a3）	
r8 （t0）	
r9 （t1）	
r10 （t2）	
r11 （t3）	
r12 （t4）	
r13 （t5）	
r14 （t6）	
r15 （t7）	
r16 （s0）	
r17 （s1）	
r18 （s2）	
r19 （s3）	
r20 （s4）	
r21 （s5）	
r22 （s6）	
r23 （s7）	
r24 （t8）	
r25 （t9）	
r26 （k0）	
r27 （k1）	
r28 （gp）	
r29 （sp）	
r30 （s8 或 fp）	
r31 （ra）	

通用寄存器

31	0
HI	
LO	

31	0
PC	

特殊用途寄存器

表 2-6： MIPS16e 寄存器用法

MIPS16e 寄存器编码	32 位 MIPS 寄存器编码	符号名称	说明
0	16	s0	通用寄存器
1	17	s1	通用寄存器
2	2	v0	通用寄存器
3	3	v1	通用寄存器
4	4	a0	通用寄存器
5	5	a1	通用寄存器
6	6	a2	通用寄存器
7	7	a3	通用寄存器
N/A	24	t8	MIPS16e 条件代码寄存器；由 BTEQZ、BTNEZ、CMP、CMPI、SLT、SLTU、SLTI 和 SLTIU 指令隐式引用
N/A	29	sp	堆栈指针寄存器
N/A	31	ra	返回地址寄存器

表 2-7： MIPS16e 特殊寄存器

符号名称	用途
PC	程序计数器。 PC 相对寻址 Add 和 Load 指令可以将该寄存器作为操作数进行访问。
HI	包含乘法或除法结果的高位字。
LO	包含乘法或除法结果的低位字。

2.11.3 如何实现堆栈 /MIPS 调用约定

PIC32MX CPU 没有硬件堆栈。处理器依赖软件来提供此功能。由于硬件本身不会执行堆栈操作，因此系统中必须存在相关约定，让系统中的所有软件可以使用相同的机制。例如，堆栈可以向低地址方向延伸，也可以向高地址方向延伸。如果一段软件代码假定堆栈向低地址方向延伸，并调用了假定堆栈向高地址方向延伸的程序，则堆栈会损坏。

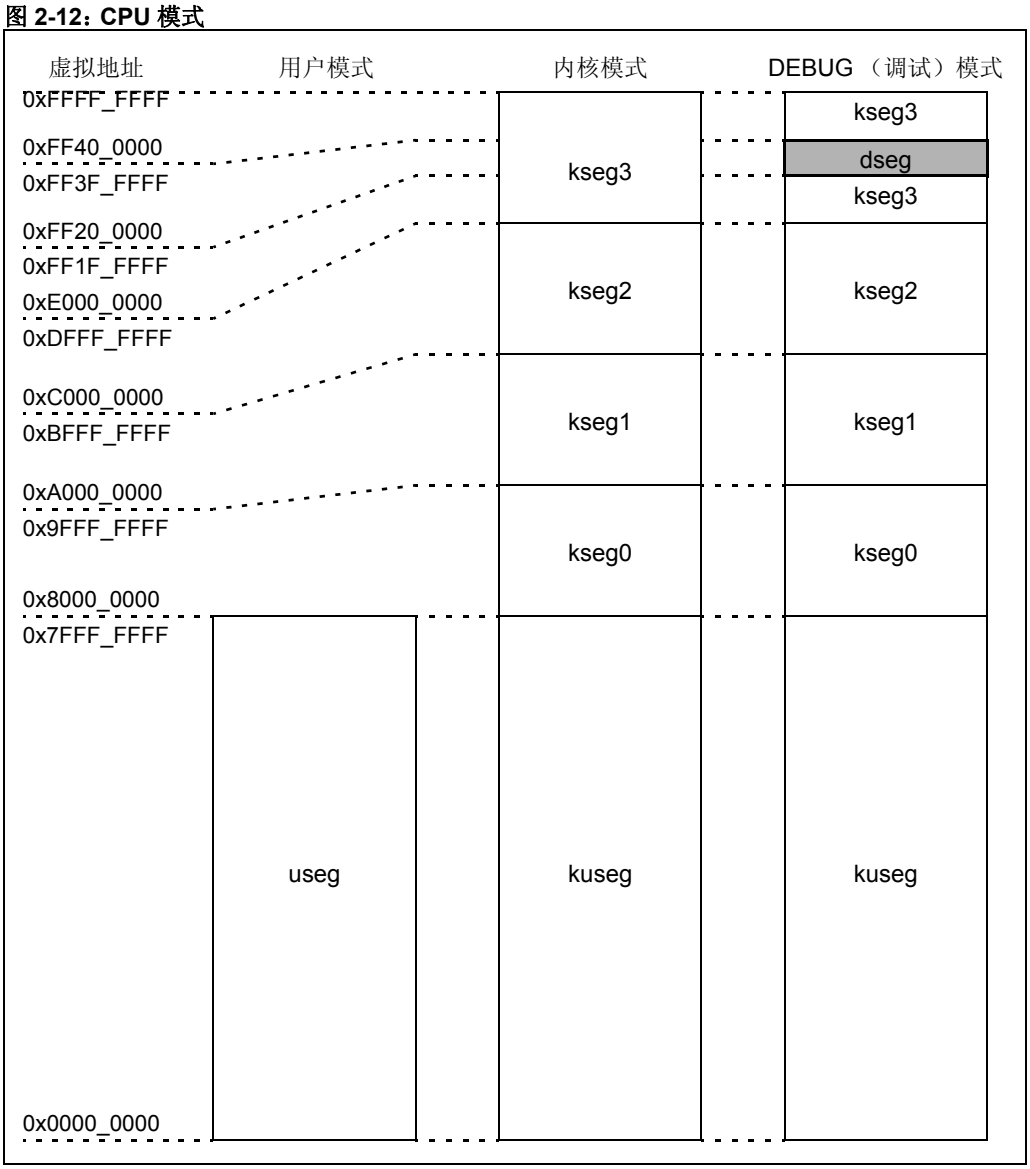
使用系统范围的调用约定可以防止发生该问题。Microchip C 编译器假定堆栈向低地址方向延伸。

2.11.4 处理器模式

PIC32MX系列CPU中有两种工作模式和一种特殊的执行模式；即用户模式、内核模式和DEBUG（调试）模式。处理器开始时以内核模式执行，如果需要，可以停留在内核模式下进行正常操作。用户模式是一种可选模式，系统设计人员可以利用该模式来划分特权软件和非特权软件。DEBUG （调试）模式通常仅供调试器或监视器使用。

两种工作模式之间的一个主要区别是允许软件访问的存储器地址。外设在用户模式下无法访问。图 2-12 给出了每种模式的不同存储器映射。关于处理器的存储器映射的更多信息，请参见**第 3 章“存储器构成”**（DS61115）。

图 2-12：CPU 模式



2.11.4.1 内核模式

为了访问许多硬件资源，处理器必须工作于内核模式。在内核模式下，软件可以访问处理器的整个地址空间，以及访问特权指令。

当 DEBUG 寄存器中的 DM 位为 0，并且 STATUS 寄存器包含一个或多个以下值时，处理器工作于内核模式：

UM = 0　　ERL = 1　　EXL = 1

检测到非调试异常时，EXL 或 ERL 将置 1，处理器将进入内核模式。在异常处理程序末尾，通常会执行异常返回 （ERET）指令。 ERET 指令会跳转到异常 PC （EPC 或 ErrorPC，具体取决于异常）、清零 ERL，并且如果 ERL= 0 则清零 EXL。

如果 UM = 1，则在 ERL 和 EXL 重新清零，并从异常返回之后，处理器会恢复为用户模式。

2.11.4.2 用户模式

在用户模式下执行时，软件仅限于使用处理器资源的一个子集。在许多情况下，都希望将应用程序级别的代码保持在用户模式下运行，在此模式下可控制所发生的错误，不允许错误影响内核模式代码。

应用程序可以通过受控接口 （例如 SYSCALL 机制）访问内核模式功能。

如图 2-12 所示，用户模式软件可以访问 USEG 存储区。

要工作于用户模式， STATUS 寄存器必须包含以下每个位值：

UM = 1　　EXL = 0　　ERL = 0

2.11.4.3 DEBUG （调试）模式

DEBUG （调试）模式是处理器的一种特殊模式，通常仅供调试器和系统监视器使用。通过调试异常进入 DEBUG （调试）模式，可以访问所有内核模式资源，以及用于调试应用程序的所有特殊硬件资源。

当 DEBUG 寄存器中的 DM 位为 1 时，处理器处于 DEBUG （调试）模式。

退出 DEBUG （调试）模式的方法通常是在调试处理程序中执行 DERET 指令。

2.12 CP0 寄存器

PIC32MX 使用特殊的寄存器接口在系统软件和 CPU 之间传送状态和控制信息。该接口称为协处理器 0。通过协处理器 0 可见的 CPU 功能为内核定时器、中断和异常控制、虚拟存储器配置、影子寄存器集控制、处理器标识和调试器控制。系统软件可以使用协处理器指令（例如 MFC0 和 MTC0）访问 CP0 中的寄存器。表 2-8 列出了 PIC32MX MCU 上的 CP0 寄存器。

表 2-8： CP0 寄存器

寄存器编号	寄存器名称	功能
0-6	保留	在 PIC32MX 内核中保留
7	HWREna	使能在非特权模式下通过 RDHWR 指令访问选定的硬件寄存器
8	BadVAddr	报告地址最近的地址相关异常
9	Count	处理器周期计数
10	保留	在 PIC32MX 内核中保留
11	Compare	定时器中断控制
12	Status/ IntCtl/ SRSCtl/ SRSMap	处理器状态和控制；中断控制；以及影子寄存器集控制
13	Cause	上一次异常的原因
14	EPC	上一次异常的程序计数器
15	PRId/ EBASE/	处理器标识和版本；异常基址
16	Config/ Config1/ Config2/ Config3	配置寄存器
17-22	保留	在 PIC32MX 内核中保留
23	Debug/ Debug2/	调试控制 / 异常状态和 EJTAG 跟踪控制
24	DEPC	上一次调试异常的程序计数器
25-29	保留	在 PIC32MX 内核中保留
30	ErrorEPC	上一次错误的程序计数器
31	DeSAVE	调试处理程序的中间结果寄存器

2.12.1 HWREna 寄存器 （CP0 寄存器 7，选择 0）

HWREna 包含决定哪些硬件寄存器可通过 RDHWR 指令访问的位掩码。

寄存器 2-1： HWREna：硬件访问使能寄存器； CP0 寄存器 7，选择 0

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
—	—	—	—	—	—	—	—
bit 31							bit 24

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
—	—	—	—	—	—	—	—
bit 23							bit 16

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
—	—	—	—	—	—	—	—
bit 15							bit 8

r-0	r-0	r-0	r-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	—	—	MASK<3:0>			
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-4 **保留：**写入 0；读取时返回 0

bit 3-0 **MASK<3:0>：**位掩码位

1 = 允许访问相应的硬件寄存器
0 = 禁止访问
该字段中的每个位用于允许 RDHWR 指令对特定硬件寄存器的访问（它可能不是实际的寄存器）。关于有效硬件寄存器的列表，请参见 RDHWR 指令。

2.12.2 BadVAddr 寄存器 （CP0 寄存器 8，选择 0）

BadVAddr 是一个只读寄存器，它捕捉最近导致地址错误异常的虚拟地址。可导致地址错误的事件有：在未对齐地址处执行装载、存储或取指操作，以及在用户模式下尝试访问内核模式地址。

BadVAddr 不会捕捉总线错误的地址信息，因为这不是寻址错误。

寄存器 2-2： BadVAddr：无效虚拟地址寄存器； CP0 寄存器 8，选择 0

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
BadVAddr<31:24>							
bit 31							bit 24

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
BadVAddr<23:16>							
bit 23							bit 16

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
BadVAddr<15:8>							
bit 15							bit 8

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
BadVAddr<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-0 **BadVAddr<31:0>：**无效虚拟地址位
捕捉导致最近地址错误异常的虚拟地址。

2.12.3 COUNT 寄存器 （CP0 寄存器 9，选择 0）

COUNT 用作一个定时器，无论指令是否执行、撤消，或是在流水线中有任何前移，它都以恒定速率递增。如果 CAUSE 寄存器中的 DC 位为 0，则计数器每隔一个时钟递增一次。

COUNT 可用于实现某种功能或用于进行诊断，包括在复位时使用或用于同步处理器。

通过写 DEBUG 寄存器中的 CountDM 位，可以控制在处理器处于 DEBUG（调试）模式时 COUNT 是否继续递增。

寄存器 2-3： COUNT：间隔计数器寄存器； CP0 寄存器 9，选择 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COUNT<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COUNT<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COUNT<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COUNT<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-0 **COUNT<31:0>：**间隔计数器位
该值每隔一个时钟周期递增一次。

2.12.4 COMPARE 寄存器（CP0 寄存器 11，选择 0）

COMPARE 与 COUNT 一起用于实现定时器和定时器中断功能。COMPARE 保持一个稳定值，不会自行更改。

当 COUNT 值等于 COMPARE 值时， CPU 会向系统中断控制器发出中断信号。该信号将一直保持有效，直到写入 COMPARE 为止。

寄存器 2-4： COMPARE：间隔计数比较寄存器； CP0 寄存器 11，选择 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COMPARE<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COMPARE<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COMPARE<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
COMPARE<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-0 **COMPARE<31:0>：**间隔计数比较值位

2.12.5 STATUS 寄存器 （CP0 寄存器 12，选择 0）

STATUS 是一个读 / 写寄存器，包含处理器的工作模式、中断允许和诊断状态。该寄存器的一些字段组合产生处理器的工作模式。

2.12.5.0.1 中断允许

当以下所有条件均为真时，将允许中断：

IE = 1　　EXL = 0　　ERL = 0　　DM = 0

如果满足这些条件，则 IPL 位的设置将允许中断。

2.12.5.0.2 工作模式

如果 Debug 寄存器中的 DM 位为 1，则处理器处于 DEBUG（调试）模式；否则，处理器处于内核或用户模式。

以下 CPU STATUS 寄存器位设置决定用户或内核模式：

表 2-9：　　决定处理器模式的 CPU 状态位

用户模式 （要求以下*所有*位和值）	UM = 1	EXL = 0	ERL = 0
内核模式 （要求一*个*或多个以下位值）	UM = 0	EXL = 1	ERL = 1

注：	STATUS 寄存器 CU 位 <31:28> 控制协处理器的可访问性。如果任意协处理器不可用，则访问它的指令会产生异常。

寄存器 2-5： STATUS：状态寄存器； CP0 寄存器 12，选择 0

R-0	R-0	R-0	R/W-x	R/W-0(1)	r-x	R/W-x	r-0
CU3	CU2	CU1	CU0	RP	FR	RE	—
bit 31							bit 24

r-0	R/W-1	r-0	R/W-0	R/W-0	r-0	r-0	r-0
—	BEV	保留	SR	NMI	—	—	—
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
IPL<15:10>						R<9:8>	
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
—	—	—	UM	—	ERL	EXL	IE
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31 **CU3**：协处理器 3 可用位
控制对协处理器 3 的访问
不支持 COP3。该位不能写入，并读为 0

bit 30 **CU2**：协处理器 2 可用位
控制对协处理器 2 的访问
不支持 COP2。该位不能写入，并读为 0

bit 29 **CU1**：协处理器 1 可用位
控制对协处理器 1 的访问
不支持 COP1。该位不能写入，并读为 0

bit 28 **CU0**：协处理器 0 可用位
控制对协处理器 0 的访问
0 = 不允许访问
1 = 允许访问
当处理器运行于内核模式时，无论 CU0 位的状态如何，协处理器 0 总是可用。

bit 27 **RP**：降低功耗位
使能降低功耗模式

bit 26 **FR**：FR 位
在 PIC32MX 处理器上保留

bit 25 **RE**：用于使能在处理器运行于用户模式时的反向字节序存储器引用
0 = 用户模式使用所配置的字节序
1 = 用户模式使用反向字节序
在 DEBUG （调试）模式、内核模式和监督模式下的引用不会受该位状态影响。

bit 24:23 **R<24:23>**：保留。写操作会被忽略，并读为 0。

寄存器 2-5： STATUS：状态寄存器； CP0 寄存器 12，选择 0 （续）

bit 22	**BEV**：控制位。控制异常向量的位置。 0 = 正常 1 = 引导
bit 21	**保留**
bit 20	**SR**：软复位位 指示进入复位异常向量是由于软复位。 0 = 非软复位 （NMI 或复位） 1 = 软复位 软件只能通过向该位写入 0 来清零该位，不能强制进行 0-1 转换。
bit 19	**NMI**：软复位位 指示进入复位异常向量是由于 NMI。 0 = 非 NMI （软复位或复位） 1 = NMI 软件只能通过向该位写入 0 来清零该位，不能强制进行 0-1 转换。
bit 18	**R**：保留。写操作会被忽略，并读为 0。
bit 17	**R**：保留。写操作会被忽略，并读为 0。
bit 16	**R**：保留。写操作会被忽略，并读为 0。
bit 15-10	**IPL<15:10>**：中断优先级位 该字段是当前 IPL 的编码值 （0..63）。只有所请求的 IPL 大于该值时，才会发出中断
bit 9-8	**R<9:8>**：保留 这些位可写，但对于中断系统没有任何作用。
bit 7-5	**R<7:5>**：保留。写操作会被忽略，并读为 0
bit 4	**UM**： 该位指示处理器的基本工作模式。该位的编码为： 0 = 基本模式为内核模式 1 = 基本模式为用户模式 **注：** 如果 ERL 或 EXL 置 1，则无论 UM 位状态如何，处理器也可处于内核模式。
bit 3	**R**：保留。写操作会被忽略，并读为 0
bit 2	**ERL**：错误级别位 当发生复位、软复位、 NMI 或高速缓存错误异常时由处理器置 1。 0 = 正常级别 1 = 错误级别 当 ERL 置 1 时： - 处理器运行于内核模式 - 中断被禁止 - ERET 指令将使用 ErrorEPC 而不是 EPC 中保存的返回地址 - kuseg 的低 2^{29} 字节视为非映射和非高速缓存区。这使得可以在存在高速缓存错误时访问主存储器。如果 ERL 位在处理器从 kuseg 中执行指令时置 1，则处理器的操作是*未定义的*。
bit 1	**EXL**：异常级别位 当发生除复位、软复位或 NMI 异常之外的任何其他异常时由处理器置 1。 0 = 正常级别 1 = 异常级别 当 EXL 置 1 时： - 处理器运行于内核模式 - 中断被禁止 如果捕捉到另一个异常， EPC、 CauseBD 和 SRSCtl 将不会被更新。

寄存器 2-5：　STATUS：状态寄存器； CP0 寄存器 12，选择 0 （续）

bit 0　**IE**：中断允许位
用作软件和硬件中断的主允许控制：
0 = 禁止中断
1 = 允许中断
该位可以通过 DI 和 EI 指令单独修改。

2.12.6 Intctl：中断控制寄存器 （CP0 寄存器 12，选择 1）

Intctl 寄存器控制 PIC32MX 架构的向量间距。

寄存器 2-6： Intctl：中断控制寄存器； CP0 寄存器 12，选择 1

R-0	R-0	R-0	R-0	R-0	R-0	r-x	r-x
—	—	—	—	—	—	—	—
bit 31							bit 24

r-x	r-x	r-x	r-x	r-x	r-x	r-x	r-x
—	—	—	—	—	—	—	—
bit 23							bit 16

r-x	r-x	r-x	r-x	r-x	r-x	R/W-0	R/W-0
—	—	—	—	—	—	VS<9:8>	
bit 15							bit 8

R/W-0	R/W-0	R/W-0	r-x	r-x	r-x	r-x	r-x
VS<7:5>			—	—	—	—	—
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-29 **R：**保留

bit 28-26 **R：**保留

bit 25-10 **保留：**写入 0；忽略读操作
必须写为 0；读取时返回 0。

bit 9-5 **VS<9:5>：**向量间距位
该字段指定每个中断向量之间的间距。

编码	向量之间的间距 （十六进制）	向量之间的间距 （十进制）
16#00	16#000 0x	0
16#01	16#020	32
16#02	16#040	64
16#04	16#080	128
16#08	16#100	256
16#10	16#200	512

所有其他值均保留。如果向该字段中写入保留值，则处理器的操作是*未定义的*。

bit 4-0 **未实现：**读为 0
必须写为 0；读取时返回 0。

2.12.7 SRSCtl 寄存器 （CP0 寄存器 12，选择 2）

SRSCtl 寄存器控制处理器中的 GPR 影子寄存器集的操作。

表 2-10： 发生异常或中断时新 $SRSCtl_{CSS}$ 的来源

异常类型	条件	$SRSCtl_{CSS}$ 源	备注
异常	全部	$SRSCtl_{ESS}$	
非向量中断	$Cause_{IV}$ = 0	$SRSCtl_{ESS}$	视为异常
向量 EIC 中断	$Cause_{IV}$ = 1 且 $Config3_{VEIC}$ = 1	$SRSCtl_{EICSS}$	来源是外部中断控制器

寄存器 2-7： SRSCtl：寄存器； CP0 寄存器 12，选择 2

r-x	r-x	R-0	R-0	R-0	R-1	r-x	r-x
—	—	HSS<29:26>				—	—
bit 31							bit 24

r-x	r-x	R-x	R-x	R-x	R-x	r-x	r-x
—	—	EICSS<21:18>				—	—
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	r-x	r-x	R/W-0	R/W-0
ESS<15:12>				—	—	PSS<9:8>	
bit 15							bit 8

R/W-0	R/W-0	r-0	r-0	R-0	R-0	R-0	R-0
PSS<7:6>		0<5:4>		CSS<3:0>			
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-30 **保留：**写入 0；忽略读操作
必须写为 0；读取时返回 0。

bit 29-26 **HSS<29:26>：**影子寄存器集高编号位
该字段包含该处理器实现的最高影子寄存器集编号。该字段中的值为 0 时，指示仅实现了正常 GPR。对于 PIC32MX 处理器，该字段的可能值为：

0 = 存在 1 个影子寄存器集 （正常 GPR 集）
1 = 存在 2 个影子寄存器集
3 = 存在 4 个影子寄存器集
2, 3-15 = 保留

该字段中的值也代表可以写入该寄存器的 ESS、EICSS、PSS 和 CSS 字段，或者 SRSMAP 寄存器的任意字段的最大值。如果向其中的任何其他字段写入的值大于该字段的值，则处理器的操作是*未定义的*。

bit 25-22 **保留：**写入 0；忽略读操作
必须写为 0；读取时返回 0。

寄存器 2-7： SRSCtl：寄存器； CP0 寄存器 12，选择 2 （续）

bit 21-18	**EICSS<21:18>**：外部中断控制器影子集位 EIC 中断模式影子集。对于每个中断请求，该字段从外部中断控制器装载，并代替 SRSMAP 寄存器来选择中断的当前影子集。
bit 17-16	**保留**：写入 0；忽略读操作 必须写为 0；读取时返回 0。
bit 15-12	**ESS<15:12>**：异常影子集位 该字段指定由于向量中断之外的任何其他异常而进入内核模式时使用的影子集。 如果软件写入该字段的值大于 HSS 字段中的值，则处理器的操作是*未定义的*。
bit 11-10	**保留**：写入 0；忽略读操作 必须写为 0；读取时返回 0。
bit 9-6	**PSS<9:6>**：先前影子集位 由于实现了 GPR 影子寄存器，所以在发生异常或中断时，将从 CSS 字段复制该字段。如果 $Status_{BEV}$ = 0，则 ERET 指令会将该值复制回 CSS 字段中。 以下情况下，该字段不会被更新：发生将 $Status_{ERL}$ 设置为 1 的异常（即，复位、软复位、NMI 和高速缓存错误）时，进入 EJTAG DEBUG（调试）模式时，或在 $Status_{EXL}$ = 1 或 $Status_{BEV}$ = 1 时发生异常或中断时。在 $Status_{ERL}$ = 1 时发生异常时，该字段不会被更新。 如果软件写入该字段的值大于 HSS 字段中的值，则处理器的操作是*未定义的*。
bit 5-4	**保留**：写入 0；忽略读操作 必须写为 0；读取时返回 0。
3-0	**CSS<3:0>**：当前影子集位 由于实现了 GPR 影子寄存器，所以该字段是当前 GPR 集的编号。在发生任何中断或异常时，该字段会被更新为新值；在执行 ERET 后会从 PSS 字段中恢复。表 2-10 列出了在发生异常或中断时，CSS 字段的各种更新来源。 以下情况下，该字段不会被更新：发生将 $Status_{ERL}$ 设置为 1 的异常（即，复位、软复位、NMI 和高速缓存错误）时，进入 EJTAG DEBUG（调试）模式时，或在 $Status_{EXL}$ = 1 或 $Status_{BEV}$ = 1 时发生异常或中断时。在 $Status_{ERL}$ = 1 或 $Status_{BEV}$ = 1 时执行 ERET 之后，该字段也不会被更新。在 $Status_{ERL}$ = 1 时发生异常时，该字段不会被更新。 软件只能通过写 PSS 字段并执行 ERET 指令来直接更改 CSS 的值。

2.12.8 SRSMAP：寄存器 （CP0 寄存器 12，选择 3）

SRSMAP 寄存器包含 8 个 4 位字段，这些字段用于将向量编号映射到要在提供中断服务时使用的影子集编号。该寄存器的值不用于非中断异常或非向量中断 （$Cause_{IV}$ = 0 或 $IntCtl_{VS}$ = 0）。这些情况下，影子集编号来自 $SRSCtl_{ESS}$。

如果 $SRSCtl_{HSS}$ 为 0，则软件读或写该寄存器的结果是*不可预测的*。

如果写入该寄存器任何字段的值大于 $SRSCtl_{HSS}$ 的值，则处理器的操作是*未定义的*。

SRSMAP 寄存器包含对应于向量编号 7..0 的影子寄存器集编号。对于同一影子集编号可以设立多个中断向量，产生从向量到单个影子寄存器集编号的多对一映射。

寄存器 2-8： SRSMAP：寄存器； CP0 寄存器 12，选择 3

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SSV7<31:28>				SSV6<27:24>			
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SSV5<23:20>				SSV4<19:16>			
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SSV3<15:12>				SSV2<11:8>			
bit 15							bit 8

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
SSV1<7:4>				SSV0<3:0>			
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-28 **SSV7<31:28>：**影子集向量 7 位
对应于向量编号 7 的影子寄存器集编号

bit 27-24 **SSV6<27:24>：**影子集向量 6 位
对应于向量编号 6 的影子寄存器集编号

bit 23-20 **SSV5<23:20>：**影子集向量 5 位
对应于向量编号 5 的影子寄存器集编号

bit 19-16 **SSV4<19:16>：**影子集向量 4 位
对应于向量编号 4 的影子寄存器集编号

bit 15-12 **SSV3<15:12>：**影子集向量 3 位
对应于向量编号 3 的影子寄存器集编号

bit 11-8 **SSV2<11:8>：**影子集向量 2 位
对应于向量编号 2 的影子寄存器集编号

bit 7-4 **SSV1<7:4>：**影子集向量 1 位
对应于向量编号 1 的影子寄存器集编号

寄存器 2-8： **SRSMAP：寄存器； CP0 寄存器 12，选择 3 （续）**

bit 3-0 **SSV0<3:0>：**影子集向量 0 位
对应于向量编号 0 的影子寄存器集编号

2.12.9 CAUSE 寄存器 （CP0 寄存器 13，选择 0）

CAUSE 寄存器主要描述最近异常的原因。此外，一些字段还用于控制软件中断请求，以及向哪个向量分派中断。除 $IP_{1..0}$、DC、IV 和 WP 字段外，CAUSE 寄存器中的所有其他字段都是只读的。 $IP_{7..2}$ 解释为请求的中断优先级 （Requested Interrupt Priority Level， RIPL）。

表 2-11： CAUSE 寄存器 ExcCode 字段

异常编码值		助记符	说明
十进制	十六进制		
0	16#00	Int	中断
4	16#04	AdEL	地址错误异常 （装载或取指）
5	16#05	AdES	地址错误异常 （存储）
6	16#06	IBE	总线错误异常 （取指）
7	16#07	DBE	总线错误异常 （数据引用：装载或存储）
8	16#08	Sys	Syscall 异常
9	16#09	Bp	断点异常
10	16#0a	RI	保留的指令异常
11	16#0b	CPU	协处理器不可用异常
12	16#0c	Ov	算术溢出异常
13	16#0d	Tr	陷阱异常
14-18	16#0e-16#12	–	保留

寄存器 2-9： CAUSE：寄存器； CP0 寄存器 13，选择 0

R-x	R-x	R-x	R-x	R/W-0	R-0	r-x	r-x
BD	TI	CE<29:28>		DC	R	0<25:24>	
bit 31							bit 24

R/W-x	R/W-0	r-x	r-x	r-x	r-x	r-x	r-x
IV	R	0<21:16>					
bit 23							bit 16

R-x	R-x	R-x	R-x	R-x	R-x	R/W-x	R/W-x
RIPL<15:10>						IP1..IP0<9:8>	
bit 15							bit 8

r-x	R-x	R-x	R-x	R-x	R-x	r-x	r-x
0	EXCCODE<6:2>					0<1:0>	
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31 **BD**：转移延时位
指示捕捉的上一次异常是否是在转移延时时隙中发生的：
0 = 不处于延时时隙中
1 = 处于延时时隙中
发生异常时，只有在 $Status_{EXL}$ 为 0 的情况下，处理器才会更新 BD。

bit 30 **TI**：定时器中断位
定时器中断。该位指示是否有定时器中断在等待处理 （类似于其他中断类型的 IP 位）：
0 = 没有定时器中断在等待处理
1 = 有定时器中断在等待处理

bit 29-28 **CE<29:28>**：协处理器异常位
捕捉到协处理器不可用异常时引用的协处理器单元编号。该字段在每次发生异常时由硬件装载，但对于除协处理器不可用异常之外的所有其他异常，它是*不可预测的*。

bit 27 **DC**：禁止计数位
禁止 COUNT 寄存器。在一些对功耗敏感的应用中，将不使用 COUNT 寄存器，可以停止它以避免不必要的翻转计数
0 = 使能 COUNT 寄存器计数
1 = 禁止 COUNT 寄存器计数

bit 26 **R**：位

bit 25-24 **保留**：写入 0；忽略读操作
必须写为 0；读取时返回 0。

寄存器 2-9： CAUSE：寄存器； CP0 寄存器 13，选择 0 （续）

bit 23	**IV**：中断向量位 指示中断异常使用的是通用异常向量还是特殊中断向量 0 = 使用通用异常向量 （16#180） 1 = 使用特殊中断向量 （16#200） 如果 $Cause_{IV}$ 为 1 且 $Status_{BEV}$ 为 0，则特殊中断向量代表向量中断表的基址。
bit 22	**R**：位
bit 21-16	**保留**：写入 0 ； 忽略读操作 必须写为 0 ； 读取时返回 0。
bit 15-10	**RIPL<15:10>**：请求的中断优先级位 请求的中断优先级。 该字段是所请求中断的编码值 （0..63）。值为 0 指示未请求任何中断。
bit 9-8	**IP1..IP0<9:8>**： 控制软件中断的请求： 0 = 未请求任何中断 1 = 请求软件中断 这些位导出到系统中断控制器，用于在 EIC 中断模式下与其他中断源一起比较优先级
bit 7	**保留**：写入 0 ； 忽略读操作 必须写为 0 ； 读取时返回 0。
bit 6-2	**EXCCODE<6:2>**：异常编码位 异常编码请参见表 2-11
bit 1-0	**保留**：写入 0 ； 忽略读操作 必须写为 0 ； 读取时返回 0。

2.12.10 EPC 寄存器 （CP0 寄存器 14，选择 0）

异常程序计数器（Exception Program Counter，EPC）是一个读 / 写寄存器，包含处理完异常之后继续进行处理的地址。 EPC 寄存器的所有位均为有效位，均可写。

对于同步 （精确）异常， EPC 包含以下值之一：

- 直接导致异常的指令的虚拟地址。
- 紧接在 BRANCH 或 JUMP 指令之前的虚拟地址 （如果导致异常的指令处于转移延时时隙中，并且 CAUSE 寄存器中的转移延时位置 1）。

发生新异常时，如果 STATUS 寄存器中的 EXL 位置 1，则处理器不会写 EPC 寄存器；但是，仍然可以通过 MTC0 指令写该寄存器。

由于 PIC32 系列实现了 MIPS16e ASE，所以读 EPC 寄存器（通过 MFC0）会在目标 GPR 中返回以下值：

$$GPR[rt] \leftarrow ExceptionPC_{31..1} \;||\; ISAMode_0$$

即，异常 PC 的高 31 位与 ISAMode 字段的低位相结合，并写入 GPR。

类似地，写 EPC 寄存器（通过 MTC0）时会获取 GPR 的值，并将该值分配到异常 PC 和 ISAMode 字段中，如下

$$ExceptionPC \leftarrow GPR[rt]_{31..1} \;||\; 0$$
$$ISAMode \leftarrow 2\#0 \;||\; GPR[rt]_0$$

即， GPR 的高 31 位写入异常 PC 的高 31 位，异常 PC 的低位会被清零。 ISAMode 字段的高位会被清零，低位则装入 GPR 的低位。

寄存器 2-10： EPC：寄存器； CP0 寄存器 14，选择 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EPC<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EPC<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EPC<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
EPC<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-0 **EPC<31:0>**：异常程序计数器位

2.12.11 PRID 寄存器 （CP0 寄存器 15，选择 0）

处理器标识（Processor Identification， PRID）寄存器是一个 32 位只读寄存器，包含处理器的制造商、制造商选项、处理器标识和版本的标识信息。

寄存器 2-11： PRID：寄存器； CP0 寄存器 15，选择 0

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
R<31:24>							
bit 31							bit 24

R-1	R-1	R-1	R-1	R-1	R-1	R-1	R-1
COMPANY ID<23:16>							
bit 23							bit 16

R-0x87	R-0x87	R-0x87	R-0x87	R-0x87	R-0x87	R-0x87	R-0x87
PROCESSOR ID<15:8>							
bit 15							bit 8

R-Preset	R-Preset	R-Preset	R-Preset	R-Preset	R-Preset	R-Preset	R-Preset
REVISION<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-24 **R<31:24>：**保留
必须忽略写操作，并读为 0

bit 23-16 **COMPANY ID<23:16>：**
指示设计或制造处理器的公司。在 PIC32MX 中，该字段包含值 1，指示 MIPS Technologies, Inc.

bit 15-8 **PROCESSOR ID<15:8>：**
指示处理器的类型。通过该字段，软件可以区分各种类型的 MIPS Technologies 的处理器。

bit 7-0 **REVISION<7:0>：**
指定处理器的版本号。通过该字段，软件可以区分相同处理器类型的版本。
该字段分为以下 3 个子字段。

bit 7-5 **MAJOR REVISION<7:5>：**
该数字随处理器内核的主版本而增加。

bit 4-2 **MINOR REVISION<4:2>：**
该数字在处理器每次增量调整时增加，并在每次产生新的主版本时复位。

bit 1-0 **PATCH LEVEL<1:0>：**
如果发布了补丁来修改较旧的处理器版本，则该字段增加。

2.12.12 EBASE 寄存器 （CP0 寄存器 15，选择 1）

EBASE 寄存器是一个读 / 写寄存器，包含异常向量的基址 （在 $Status_{BEV}$ 等于 0 时使用）和一个只读 CPU 编号值 （软件可以使用它来区分多处理器系统中的不同处理器）。

EBASE 寄存器使得软件可以识别多处理器系统中的特定处理器，允许每个处理器的异常向量可以不同，特别是在由异构处理器组成的系统中。当 $Status_{BEV}$ 为 0 时，EBASE 寄存器的 bit 31..12 将与 0 组合，构成异常向量的基址。当 $Status_{BEV}$ 为 1，或者对于任何 EJTAG 调试异常时，异常向量基址将来自固定的默认值。EBASE 寄存器的 bit 31..12 的复位状态会将异常基址寄存器初始化为 16#8000.0000。

EBASE 寄存器的 bit 31..30 固定为值 2#10，以强制使异常基址处于 kseg0 或 kseg1 非映射虚拟地址段中。

如果要更改异常基址寄存器的值，则必须在 $Status_{BEV}$ 等于 1 时执行。如果 $Status_{BEV}$ 为 0 时，异常基址字段写入其他值，则处理器的操作是*未定义的*。

通过将 bit 31..20 与异常基址字段组合，异常向量的基址可以放置在任意 4 KB 页边界处。

寄存器 2-12： EBASE：寄存器； CP0 寄存器 15，选择 1

R-1	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
1	0	EXCEPTION BASE<29:24>					
bit 31							bit 24

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
EXCEPTION BASE<23:16>							
bit 23							bit 16

R/W-0	R/W-0	R/W-0	R/W-0	r-0	r-0	R-0	R-0
EXCEPTION BASE<15:12>				r		CPUNUM<9:8>	
bit 15							bit 8

R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0
CPUNUM<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31 **1**：1 位
该位的写操作会被忽略，读取时返回 1。

bit 30 **0**：0 位
该位的写操作会被忽略，读取时返回 0。

bit 29-12 **EXCEPTION BASE<29:12>：**
当 $Status_{BEV}$ 为 0 时，该字段与 bit 31..30 一起指定异常向量的基址。

bit 11-10 **保留：**
必须写为 0；读取时返回 0。

寄存器 2-12： EBASE：寄存器； CP0 寄存器 15，选择 1 （续）

bit 9-0 **CPUNUM<9:0>：**

该字段指定多处理器系统中的 CPU 数量，软件可以使用它来将特定处理器与其他处理器区分开来。在单处理器系统中，该值设置为 0。

2.12.13 CONFIG 寄存器 （CP0 寄存器 16，选择 0）

CONFIG 寄存器指定各种配置和功能信息。CONFIG 寄存器中的大部分字段由硬件在复位异常过程中初始化，或者为常量。

表 2-12： 高速缓存一致性属性

C(2:0) 值	高速缓存一致性属性
2	非高速缓存
3	可高速缓存

寄存器 2-13： CONFIG：寄存器； CP0 寄存器 16，选择 0

R-1	R-0	R-1	R-0	R/W-0	R/W-1	R/W-0	r-0
M	K23<30:28>			KU<27:25>			0
bit 31							bit 24

r-x	R-0	R-0	R-0	r-x	r-x	r-x	R-1
0	UDI	SB	MDU				DS
bit 23							bit 16

R-0	R-0	R-0	R-0	R-0	R-1	R-0	R-1
BE	AT<14:13>		AR<12:10>			MT<9:8>	
bit 15							bit 8

R-1	r-x	r-x	r-x	r-x	R/W-0	R/W-1	R/W-0
MT					K0<2:0>		
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31 **M：**
该位硬连线为 1，指示存在 CONFIG1 寄存器。

bit 30-28 **K23<30:28>：**kseg2 和 kseg3 位
该字段控制 kseg2 和 kseg3 地址段的高速缓存功能。
字段编码请参见表 2-12。

bit 27-25 **KU<27:25>：**kuseg 和 useg 位
该字段控制 kuseg 和 useg 地址段的高速缓存功能。
字段编码请参见表 2-12。

bit 24-23 **保留：**写入 0；忽略读操作
必须写为 0。读取时返回 0。

bit 22 **UDI：**用户定义位
该位指示实现了 CorExtend 用户定义指令。
0 = 未实现用户定义指令
1 = 实现了用户定义指令

寄存器 2-13： CONFIG：寄存器； CP0 寄存器 16，选择 0 （续）

bit 21	**SB**：SimpleBE 位 指示是否使能了 SimpleBE 总线模式。 0 = 内部总线接口上无保留字节使能 1 = 内部总线接口上仅允许简单字节使能
bit 20	**MDU**：乘法 / 除法单元位 该位指示存在的乘法 / 除法单元的类型 0 = 快速高性能 MDU
bit 19-17	**保留**：写入 0 ；忽略读操作 必须写为 0。读取时返回 0。
bit 16	**DS**：双 SRAM 位 0 = 统一的指令 / 数据 SRAM 内部总线接口 1 = 双指令 / 数据 SRAM 内部总线接口 **注：** PIC32MX 系列当前在内部使用双 SRAM 类型的接口。
bit 15	**BE**：大尾数位 指示处理器运行的字节序模式， PIC32MX 总是小尾数。 0 = 小尾数 （Little Endian） 1 = 大尾数 （Big Endian）
bit 14-13	**AT<14:13>**：架构类型位 处理器实现的架构类型。该字段总是为 00，以指示 MIPS32 架构。
bit 12-10	**AR<12:10>**：架构版本位 架构版本。该字段总是为 001，以指示 MIPS32 Release 2。 0：Release 1 1：Release 2 2-7：保留
bit 9-7	**MT<9:7>**：MMU 类型位 3：固定映射 0-2, 4-7：保留
bit 6-3	**保留**：写入 0 ；忽略读操作 必须写为 0 ；读取时返回 0
bit 2-0	**K0<2:0>**：Kseg0 位 Kseg0 一致性算法。字段编码请参见表 2-12。

2.12.14 CONFIG1 寄存器 （CP0 寄存器 16，选择 1）

CONFIG1 寄存器是 CONFIG 寄存器的辅助寄存器，保存关于内核所具备功能的附加信息。CONFIG1 寄存器中的所有字段都是只读的。

寄存器 2-14： CONFIG1：CONFIG1 寄存器； CP0 寄存器 16，选择 1

R-1	R-x	R-x	R-x	R-x	R-x	R-x	R-x
M	MMU Size<30:25>						IS
bit 31							bit 24

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
IS<23:22>		IL<21:19>			IA<18:16>		
bit 23							bit 16

R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x
DS<15:13>			DL<12:10>			DA<9:8>	
bit 15							bit 8

R-x	R-0	R-0	R-0	R-0	R-1	R-x	R-0
DA	C2	MD	PC	WR	CA	EP	FP
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31 **M：**位
该位硬连线为 1，指示存在 CONFIG2 寄存器。

bit 30-25 **MMU Size：**位
该字段包含的值等于 TLB 条目数减 1；因为 PIC32MX 没有 TLB，所以该字段为 0。

bit 24-22 **IS：**指令高速缓存集位
该字段包含每路的指令高速缓存集数量；因为 M4K 内核不包含高速缓存，所以该字段总是读为 0。

bit 21-19 **IL：**指令高速缓存线位
该字段包含指令高速缓存线大小；因为 M4K 内核不包含高速缓存，所以该字段总是读为 0。

bit 18-16 **IA：**指令高速缓存关联位
该字段包含指令高速缓存关联级别；因为 M4K 内核不包含高速缓存，所以该字段总是读为 0。

bit 15-13 **DS：**数据高速缓存集位
该字段包含每路的数据高速缓存集数量；因为 M4K 内核不包含高速缓存，所以该字段总是读为 0。

bit 12-10 **DL：**数据高速缓存线位
该字段包含数据高速缓存线大小；因为 M4K 内核不包含高速缓存，所以该字段总是读为 0。

bit 9-7 **DA：**数据高速缓存关联位
该字段包含设置的数据高速缓存关联的类型；因为M4K内核不包含高速缓存，所以该字段总是读为0。

寄存器 2-14： CONFIG1：CONFIG1 寄存器； CP0 寄存器 16，选择 1 （续）

bit 6 C2：协处理器 2 位
存在协处理器 2。
0 = COP2 接口上未连接协处理器
1 = COP2 接口上连接了协处理器
因为 PIC32MX 系列单片机中未实现协处理器 2，所以该位读为 0。

bit 5 **MD**：MDMX 位
实现了 MDMX。
该位总是读为 0，因为不支持 MDMX。

bit 4 **PC**：性能计数器位
实现了性能计数器寄存器。
总是为 0，因为 PIC32MX 内核不包含性能计数器。

bit 3 **WR**：监视寄存器位
实现了监视寄存器。
0 = 不存在任何监视寄存器
1 = 存在一个或多个监视寄存器

注： PIC32MX 未实现监视寄存器，因此该位总是读为 0。

bit 2 **CA**：代码压缩实现位
0 = 不存在 MIPS16e
1 = 实现了 MIPS16e

bit 1 **EP**：EJTAG 存在位
该位总是置 1，指示内核实现了 EJTAG。

bit 0 **FP**：FPU 实现位
该位总是为 0，因为内核不包含浮点单元。

2.12.15 CONFIG2 （CP0 寄存器 16，选择 2）

CONFIG2 寄存器是 CONFIG 寄存器的辅助寄存器，保留用于保存附加的功能信息。 CONFIG2 分配用于显示 2/3 级高速缓存的配置。这些字段复位为 0，因为 PIC32MX 内核不支持 L2/L3 高速缓存。 CONFIG2 寄存器中的所有字段都是只读的。

寄存器 2-15： CONFIG2：CONFIG2 寄存器； CP0 寄存器 16，选择 2

R-1	r-0	r-0	r-0	r-0	r-0	r-0	r-0
M	0	0	0	0	0	0	0
bit 31							bit 24

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 23							bit 16

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 15							bit 8

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31 **M**：位
该位硬连线为 1，指示存在 CONFIG3 寄存器。

bit 30-0 **保留**

2.12.16 CONFIG3 寄存器 （CP0 寄存器 16，选择 3）

CONFIG3 寄存器用于保存关于附加功能的信息。 CONFIG3 寄存器中的所有字段都是只读的。

寄存器 2-16： CONFIG3：CONFIG3 寄存器； CP0 寄存器 16，选择 3

R-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
M	0	0	0	0	0	0	0
bit 31							bit 24

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 23							bit 16

r-0	r-0	r-0	r-0	r-0	r-0	r-0	r-0
0	0	0	0	0	0	0	0
bit 15							bit 8

r-0	R-1	R-1	R-0	r-0	r-0	R-0	R-0
0	VEIC	VInt	SP	0	0	SM	TL
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31 **M**：保留
该位保留用于指示是否存在 CONFIG4 寄存器。对于当前的架构定义，该位应总是读为 0。

bit 30-7 **保留**：写入 0 ； 忽略读操作
必须写为 0 ； 读取时返回 0。

bit 6 **VEIC**：
实现了对外部中断控制器 （EIC）的支持。
0 = 未实现对 EIC 中断模式的支持
1 = 实现了对 EIC 中断模式的支持

注： PIC32MX 内部实现了 MIPS “外部中断控制器”，因此该位读为 1。

bit 5 **VINT**：向量中断位
实现了向量中断。该位指示是否实现了向量中断。
0 = 未实现向量中断
1 = 实现了向量中断
在 PIC32MX 内核上，该位总是为 1，因为实现了向量中断。

bit 4 **SP**：支持页位
实现了小 （1 KB）页支持，并且存在 PAGEGRAIN 寄存器。
0 = 未实现小页支持
1 = 实现了小页支持

注： PIC32MX 总是读为 0，因为 PIC32MX 未实现小页支持。

bit 3-2 **0**：
必须写为 0 ； 读取时返回 0。

寄存器 2-16： CONFIG3：CONFIG3 寄存器； CP0 寄存器 16，选择 3 （续）

bit 1 **SM**：SmartMIPS™ 位

实现了 SmartMIPS™ ASE。该位指示是否实现了 SmartMIPS ASE。由于 PIC32MX 内核上存在 SmartMIPS，所以该位总是为 0。

0 = 未实现 SmartMIPS ASE

1 = 实现了 SmartMIPS ASE

bit 0 **TL**：跟踪逻辑位

实现了跟踪逻辑。该位指示是否实现了 PC 或数据跟踪。

0 = 未实现片上跟踪逻辑 （PDTrace™）

1 = 实现了片上跟踪逻辑 （PDTrace™）

注： PIC32MX 未实现 PDTrace™ 片上跟踪逻辑，因此该位总是读为 0。

2.12.17 DEBUG 寄存器 （CP0 寄存器 23，选择 0）

DEBUG 寄存器用于控制调试异常，并提供关于调试异常原因的信息，以及何时由于 DEBUG（调试）模式下的一般异常而重新进入调试异常向量处。只读信息位在以下情况下更新：每次捕捉到调试异常时，或在已处于 DEBUG （调试）模式时捕捉到一般异常。

在 Non-DEBUG（非调试）模式下读取时，只有 DM 位和 EJTAGver 字段有效；所有其他位和字段的值是*不可预测的*。如果在 Non-DEBUG（非调试）模式下写 DEBUG 寄存器，则处理器的操作是*未定义的*。

一些位和字段只有在发生调试异常和 / 或 DEBUG（调试）模式下的异常时才会更新，如下所示：

- DSS、 DBp、 DDBL、 DDBS、 DIB 和 DINT 在发生调试异常和调试模式的异常时更新
- DExcCode 在发生 DEBUG （调试）模式下的异常时更新，发生调试异常之后它是未定义的
- Halt和Doze在发生调试异常时更新，在发生DEBUG（调试）模式下的异常之后值是未定义的
- DBD 在发生调试异常和调试模式下的异常时更新

在正常模式下读取时，除 EJTAGver 和 DM 外，其他所有位和字段都是未定义的。

寄存器 2-17： DEBUG：寄存器； CP0 寄存器 23，选择 0

R-U	R-0	R-0	R/W-0	R-U	R-U	R/W-1	R/W-0
DBD	DM	NODCR	LSNM	DOZE	HALT	COUNTDM	IBUSEP
bit 31							bit 24

R-0	R-0	R/W-0	R/W-0	R-0	R-0	R-0	R-1
MCHECKP	CACHEEP	DBUSEP	IEXI	DDBSIMPR	DDBLIMPR	VER<7:6>	
bit 23							bit 16

R-0	R-U	R-U	R-U	R-U	R-U	R-0	R/W-0
VER	DEXCCODE<14:10					NOSST	SST
bit 15							bit 8

R-0	R-0	R-U	R-U	R-U	R-U	R-U	R-U
R<7:6>		DINT	DIB	DDBS	DDBL	DBP	DSS
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31 **DBD：**
指示上一次调试异常或 DEBUG （调试）模式下的异常是否是在转移延时时隙中发生的：
0 = 不处于延时时隙中
1 = 处于延时时隙中

bit 30 **DM：**
指示处理器是否工作于 DEBUG （调试）模式：
0 = 处理器工作于 Non-DEBUG （非调试）模式
1 = 处理器工作于 DEBUG （调试）模式

bit 29 **NODCR：**
指示 dseg 存储器段是否存在，以及调试控制寄存器是否可访问：
0 = 存在 dseg
1 = 不存在 dseg

bit 28 **LSNM：**
控制 dseg 和主存储器之间的装载 / 存储访问：
0 = dseg 地址范围内的装载 / 存储访问送到 dseg
1 = dseg 地址范围内的装载 / 存储访问送到主存储器

bit 27 **DOZE：**
指示发生调试异常时处理器是否处于某种类型的低功耗模式：
0 = 发生调试异常时处理器不处于低功耗模式
1 = 发生调试异常时处理器处于低功耗模式

bit 26 **HALT：**
指示发生调试异常时内部系统总线时钟是否停止：
0 = 内部系统总线时钟停止
1 = 内部系统总线时钟在运行

寄存器 2-17： DEBUG：寄存器； CP0 寄存器 23，选择 0 （续）

bit 25 **COUNTDM：**
指示计数寄存器在 DEBUG （调试）模式下的行为。
0 = 计数寄存器在 DEBUG （调试）模式下停止
1 = 计数寄存器在 DEBUG （调试）模式下运行

bit 24 **IBUSEP：**
取指总线错误异常等待处理。在发生取指总线错误事件时或软件向该位写入 1 时置 1。在处理器捕捉到取指总线错误异常时清零，或通过复位清零。如果 IBUSEP 在 IEXI 清零时置 1，则处理器会捕捉取指总线错误异常，并清零 IBUSEP。

bit 23 **MCHECKP：**
指示有非精确机器检查异常等待处理。在 PIC32MX 处理器上，所有机器检查异常都是精确的，所以该位将总是读为 0。

bit 22 **CACHEEP：**
指示有非精确高速缓存错误等待处理。PIC32MX内核无法捕捉高速缓存错误，所以该位将总是读为0。

bit 21 **DBUSEP：**
数据访问总线错误异常等待处理。涵盖数据访问时发生的非精确总线错误，行为类似于 IBUSEP 在取指时发生非精确总线错误的行为。

bit 20 **IEXI：**
非精确错误异常禁止，它控制由于非精确错误指示而捕捉到的异常。在处理器捕捉到调试异常或 DEBUG（调试）模式下的异常时置 1。通过执行 DERET 指令清零；另外，可以通过 DEBUG（调试）模式软件修改。当 IEXI 置 1 时，由于总线错误（在取指或访问数据时发生）、高速缓存错误或机器检查而产生的非精确错误异常会被禁止和推迟，直到该位清零为止。

bit 19 **DDBSIMPR：**
指示捕捉到非精确调试数据中断存储异常。在 PIC32MX 内核上，所有数据中断都是精确的，所以该位将总是读为 0。

bit 18 **DDBLIMPR：**
指示捕捉到非精确调试数据中断装载异常。在 PIC32MX 内核上，所有数据中断都是精确的，所以该位将总是读为 0。

bit 17-15 **VER：**
EJTAG 版本

bit 14-10 **DEXCCODE：**
指示 DEBUG （调试）模式下的最近异常的原因。对于在 DEBUG （调试）模式下可能发生的那些一般异常，该字段的编码与 CAUSE 寄存器中 ExcCode 字段相同。
发生调试异常之后的值是未定义的。

bit 9 **NOSST：**
指示可通过 SST 位控制的单步功能在此实现中是否可用：
0 = 单步功能可用
1 = 单步功能不可用

bit 8 **SST：**
控制是否使能调试单步异常：
0 = 不使能调试单步异常
1 = 使能调试单步异常

bit 7-6 **保留：**
必须写为 0；读取时返回 0。

bit 5 **DINT：**
指示发生了调试中断异常。在发生 DEBUG （调试）模式下的异常时清零。
0 = 未发生调试中断异常
1 = 发生调试中断异常

寄存器 2-17： DEBUG：寄存器； CP0 寄存器 23，选择 0 （续）

bit 4 **DIB：**
指示发生了调试指令中断异常。在发生 DEBUG （调试）模式下的异常时清零。
0 = 未发生调试指令异常
1 = 发生调试指令异常

bit 3 **DDBS：**
指示在存储时发生了调试数据中断异常。在发生 DEBUG （调试）模式下的异常时清零。
0 = 在存储时未发生调试数据异常
1 = 在存储时发生了调试指令异常

bit 2 **DDBL：**
指示在装载时发生了调试数据中断异常。在发生 DEBUG （调试）模式下的异常时清零。
0 = 在装载时未发生调试数据异常
1 = 在装载时发生了调试指令异常

bit 1 **DBP：**
指示发生了调试软件断点异常。在发生 DEBUG （调试）模式下的异常时清零。
0 = 未发生调试软件断点异常
1 = 发生调试软件断点异常

bit 0 **DSS：**
指示发生了调试单步异常。在发生 DEBUG （调试）模式下的异常时清零。
0 = 未发生调试单步异常
1 = 发生调试单步异常

2.12.18 DEPC 寄存器 （CP0 寄存器 24，选择 0）

调试异常程序计数器 （Debug Exception Program Counter， DEPC）寄存器是一个读 / 写寄存器，包含在处理完调试异常或 DEBUG （调试）模式下的异常之后继续进行处理的地址。

对于同步 （精确）调试异常和 DEBUG （调试）模式下的异常， DEPC 包含以下值之一：

- 直接导致调试异常的指令的虚拟地址，或者
- 紧接在转移或跳转指令之前的虚拟地址（如果导致调试异常的指令处于转移延时时隙中，并且 Debug 寄存器中的调试转移延时 （Debug Branch Delay， DBD）位置 1）。

对于异步调试异常 （调试中断）， DEPC 包含一条指令的虚拟地址，在调试处理程序代码执行完毕之后将在该地址处继续执行。

由于 PIC32 系列实现了 MIPS16e ASE，所以读 DEPC 寄存器 （通过 MFC0）会在目标 GPR 中返回以下值：

$$GPR[rt] = DebugExceptionPC_{31..1} \;||\; ISAMode_{0}$$

即，调试异常 PC 的高 31 位与 ISAMode 字段的低位组合，并写入 GPR。

类似地，写 DEPC 寄存器 （通过 MTC0）时会获取 GPR 的值，并将该值分配到调试异常 PC 和 ISAMode 字段中，如下

$$DebugExceptionPC = GPR[rt]_{31..1} \;||\; 0$$
$$ISAMode = 2\#0 \;||\; GPR[rt]_{0}$$

即， GPR 的高 31 位写入调试异常 PC 的高 31 位，调试异常 PC 的低位会被清零。 ISAMode 字段的高位会被清零，低位则装入 GPR 的低位。

寄存器 2-18： DEPC：调试异常程序计数器寄存器； CP0 寄存器 24，选择 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DEPC<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DEPC<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DEPC<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DEPC<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-0 **DEPC<31:0>：**调试异常程序计数器位

DEPC 寄存器会更新为导致调试异常的指令的虚拟地址。如果指令处于转移延时时隙中，则紧接在转移或跳转指令之前的虚拟地址会被放入该寄存器中。

执行 DERET 指令会导致跳转到 DEPC 中的地址处。

2.12.19 ErrorEPC （CP0 寄存器 30，选择 0）

ErrorEPC 寄存器是一个类似于 EPC 寄存器的读 / 写寄存器，只是 ErrorEPC 是在发生错误异常时使用。 ErrorEPC 寄存器的所有位均为有效位，必须为可写。它也在发生复位、软复位和不可屏蔽中断 （Nonmaskable Interrupt， NMI）异常时用于存储程序计数器。

ErrorEPC 寄存器包含一个虚拟地址，在处理完错误之后在该地址处继续处理指令。该地址可能为：

- 导致异常的指令的虚拟地址
- 紧接在转移或跳转指令之前的虚拟地址 （当导致错误的指令处于转移延时时隙中时）

不同于 EPC 寄存器， ErrorEPC 寄存器不存在相应的转移延时时隙指示。

由于 PIC32 系列实现了 MIPS16e ASE，所以读 ErrorEPC 寄存器（通过 MFC0）会在目标 GPR 中返回以下值：

$$GPR[rt] = ErrorExceptionPC_{31..1} \;||\; ISAMode_0$$

即，错误异常 PC 的高 31 位与 ISAMode 字段的低位组合，并写入 GPR。

类似地，写 ErrorEPC 寄存器（通过 MTC0）时会获取 GPR 的值，并将该值分配到错误异常 PC 和 ISAMode 字段中，如下

$$ErrprExceptionPC = GPR[rt]_{31..1} \;||\; 0$$
$$ISAMode = 2\#0 \;||\; GPR[rt]_0$$

即， GPR 的高 31 位写入错误异常 PC 的高 31 位，错误异常 PC 的低位会被清零。 ISAMode 字段的高位会被清零，低位则装入 GPR 的低位。

寄存器 2-19： ErrorEPC：错误异常程序计数器寄存器； CP0 寄存器 30，选择 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ErrorEPC<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ErrorEPC<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ErrorEPC<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
ErrorEPC<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-0 **ErrorEPC<31:0>**：错误异常程序计数器位

2.12.20 DeSave 寄存器 （CP0 寄存器 31，选择 0）

调试异常保存 （DeSave）寄存器是一个读 / 写寄存器，用作简单的存储单元。调试异常处理程序使用该寄存器来保存一个 GPR，然后使用它将剩余的现场信息保存到预先确定的存储区 （例如保存在 EJTAG 探针中）。无法断定是否存在用于保存现场信息的有效堆栈时，可以通过该寄存器，对异常处理程序和其他类型的代码安全地进行调试。

寄存器 2-20： DeSave：调试异常保存寄存器； CP0 寄存器 31，选择 0

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DESAVE<31:24>							
bit 31							bit 24

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DESAVE<23:16>							
bit 23							bit 16

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DESAVE<15:8>							
bit 15							bit 8

R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x	R/W-x
DESAVE<7:0>							
bit 7							bit 0

图注：			
R = 可读位	W = 可写位	P = 可编程位	r = 保留位
U = 未实现位	-n = POR 时的值：（0， 1， x = 未知）		

bit 31-0 **DESAVE<31:0>：**调试异常保存位
由调试异常代码使用的中间结果寄存器。

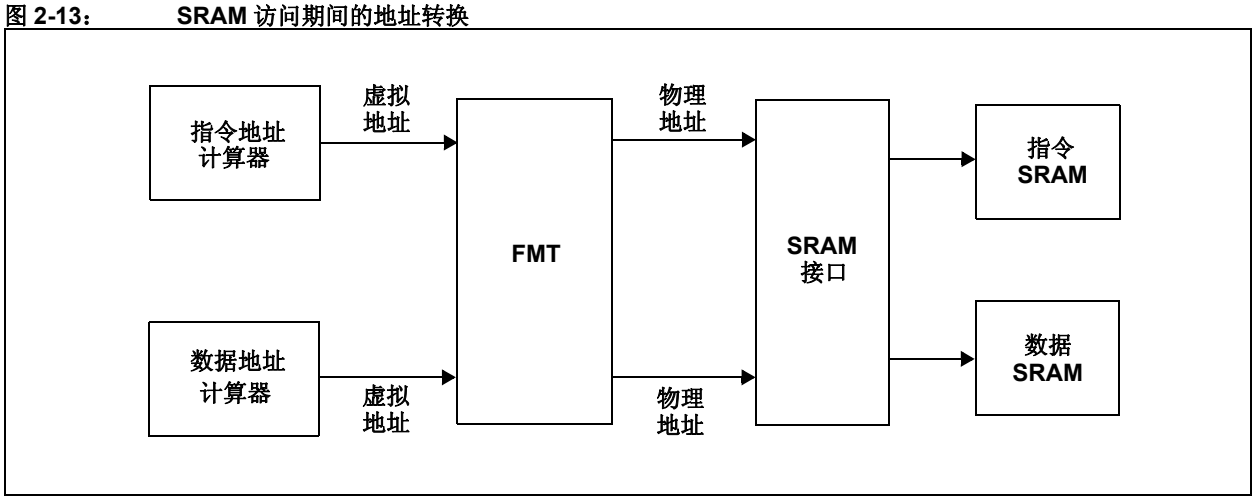
2.13 MIPS16e™ 执行

当内核工作于 MIPS16e 模式时，取指操作只需返回 16 位数据。但为了提高效率，每当地址为字对齐时，内核都会取 32 位的指令数据。因而，对于顺序执行的 MIPS16e 代码，每隔一条指令才会发生一次取指操作，这可提高性能和降低系统功耗。

2.14 存储器模型

软件使用的虚拟地址在发送给 CPU 总线之前，会通过存储器管理单元 （Memory Management Unit，MMU）转换为物理地址。对于该转换，PIC32MX CPU 使用固定映射方式。关于系统存储器模型的更多信息，请参见**第 3 章 “存储器构成”**（DS61115）。

图 2-13： SRAM 访问期间的地址转换



2.14.1 高速缓存功能

CPU 使用取指、装载或存储操作的虚拟地址来确定是否访问高速缓存。 kseg0 或 useg/kuseg 中的存储器访问可以进行高速缓存，而 kseg1 中的访问不能进行高速缓存。 CPU 使用 CONFIG 寄存器中的 CCA 位来确定存储器段的高速缓存功能。如果相应的 CCA = 011_2，则说明存储器访问可高速缓存。

关于高速缓存操作的更多信息，请参见**第 4 章 “预取高速缓存模块”**（DS61119）。

2.14.1.1 小尾数字节顺序

在以字节分辨率对存储器进行寻址的 CPU 上，对于多字节数据项存在一种约定，指定高字节到低字节的顺序。大尾数字节顺序规定最低地址包含最高有效字节 （MSB）。小尾数顺序规定最低地址包含多字节数据的最低有效字节 （LSB）。 PIC32MX CPU 系列支持小尾数字节顺序。

图 2-14：　　大尾数字节顺序

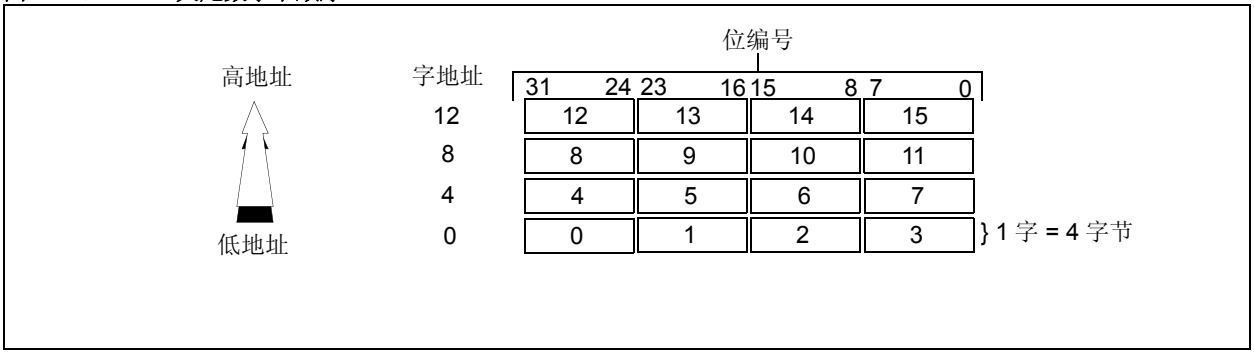
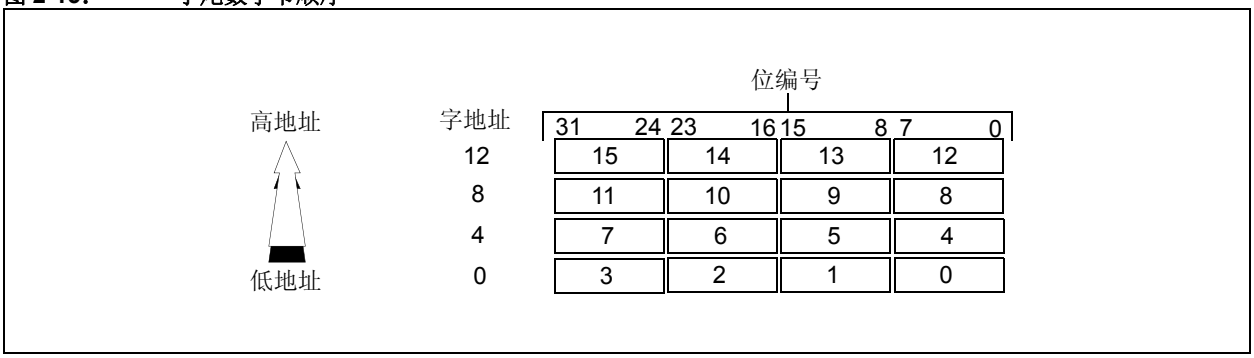


图 2-15：　　小尾数字节顺序



2.15 CPU 指令（按功能分组）

CPU 指令分为以下功能组：

- 装载和存储
- 计算
- 跳转和转移
- 其他
- 协处理器

每条指令的长度为 32 位。

2.15.1 CPU 装载和存储指令

MIPS 处理器使用装载 / 存储架构；所有操作均对保存在处理器寄存器中的操作数执行，主存储器仅通过装载和存储指令进行访问。

2.15.1.1 装载和存储的类型

有几种不同类型的装载和存储指令，分别针对不同用途而设计：

- 传送各种大小的字段 （例如， LB 和 SW）
- 将传送的数据交换为有符号或无符号整数 （例如， LHU）
- 访问未对齐字段 （例如， LWR 和 SWL）
- 原子级存储器更新 （读 - 修改 - 写：例如， LL/SC）

2.15.1.2 CPU 装载和存储指令的列表

CPU 装载和存储指令可以传送以下数据大小 （在 *AccessLength* 字段中定义）：

- 字节
- 半字
- 字

装载指令支持不同大小的有符号和无符号整数，它们可以对装载到寄存器中的数据进行符号扩展或零扩展。

未对齐字和双字可以通过使用一对特殊指令在两条指令中进行装载或存储。对于装载，一条 LWL 指令与一条 LWR 指令配对。装载指令从对齐字处读取左字节或右字节 （寄存器的左侧或右侧），并将它们合并为对应于目标寄存器的正确字节。

2.15.1.3 用于原子级更新的装载和存储

配对指令 （链接装载 （Load Linked）和条件存储 （Store Conditional））可以用于对字或双字高速缓存的存储单元执行原子级读 - 修改 - 写操作。这些指令在经过仔细编码的序列中使用，用以提供多种同步原语中的一种，包括测试并设置 （test-and-set）、位级别锁定、信号量，以及排序器和事件计数。

2.15.1.4 协处理器装载和存储

如果特定协处理器未使能，则无法执行对该处理器的装载和存储操作，尝试执行的装载或存储操作会导致协处理器不可用异常。使能协处理器是一种特权操作，通过系统控制协处理器 CP0 提供。

2.15.2 计算指令

对于以二进制补码表示的整数，将执行二进制补码算法。以下是有符号形式的补码运算：

- 加法
- 减法
- 乘法
- 除法

标为 “无符号”的加法和减法运算实际上是不带溢出检测的取模运算。

此外，还有无符号形式的*乘法*和*除法*运算，以及全部的*移位*和*逻辑*运算。逻辑运算对于寄存器宽度不敏感。

MIPS32 提供了 32 位整数和 32 位运算。

2.15.2.1 移位指令

ISA 定义了两种类型的移位指令：

- 从指令字中的 5 位字段获取固定移位量的指令 （例如， SLL 和 SRL）
- 从通用寄存器的低位获取移位量的指令 （例如， SRAV 和 SRLV）

2.15.2.2 乘法和除法指令

乘法指令执行 32 位 x 32 位乘法，产生 64 位或 32 位结果。除法指令将 64 位值除以 32 位值，产生 32 位结果。除一条指令外，其他所有指令均将它们的结果传到 HI 和 LO 特殊寄存器中。 MUL 指令会将结果的低半部分直接传到 GPR。

- 乘法会产生宽度为输入操作数两倍的全宽度乘积；低半部分装入 LO，高半部分装入 HI。
- 乘加和乘减产生宽度为输入操作数两倍的全宽度乘积，并将 HI 和 LO 的相连值加上或减去乘积。加法结果的低半部分装入 LO，高半部分装入 HI。
- 除法会产生一个商 （装入 LO）和一个余数 （装入 HI）。

结果通过在 HI/LO 和通用寄存器之间传送数据的指令进行访问。

2.15.3 跳转和转移指令

2.15.3.1 ISA 定义的跳转和转移指令的类型

架构中定义了以下跳转和转移指令：

- PC 相对条件转移
- PC 区域无条件跳转
- 绝对 （寄存器）无条件跳转
- 在通用寄存器中记录返回链接地址的一组过程调用。

2.15.3.2 转移延时和转移延时时隙

所有转移都具有一条指令的架构性延时。紧接在转移指令之后的指令处于**转移延时时隙**中。如果转移或跳转指令位于转移延时时隙中，则两种指令的操作都是未定义的。

根据约定，如果发生异常或中断，导致无法完成转移延时时隙中的指令，则会通过重新执行转移指令来继续指令流。为了允许这一点，转移必须可重新启动；过程调用不能使用存储返回链接的寄存器 （通常为 GPR *31*）来确定转移目标地址。

2.15.3.3 转移和可能转移

存在两种形式的条件转移；它们的区别在于未发生转移而向下执行时，对于延时时隙中指令的处理方式。

- 转移指令会执行延时时隙中的指令。
- 如果未发生转移，可能转移指令不会执行延时时隙中的指令（称作废弃延时时隙中的指令）。

  虽然本规范中包含了可能转移指令，但强烈建议避免在软件中使用可能转移指令，因为 MIPS 架构的未来版本中将会删除它们。

2.15.4 其他指令

2.15.4.1 指令序列化 （SYNC 和 SYNCI）

在正常操作中，架构并未规定对于执行处理器之*外*的观察者，其存储器装载和存储访问的顺序是如何的 （例如，在多处理器系统中）。

SYNC 指令可用于在执行指令流中建立一个同步点，该同步点可以决定一些装载和存储操作的相对顺序: 在 SYNC 之前执行的装载和存储操作完成之后，SYNC 之后的装载和存储操作才能开始。

SYNCI 指令可以将处理器高速缓存与先前写操作或对于指令流的其他修改进行同步。

2.15.4.2 异常指令

异常指令会将控制转移给内核中的软件异常处理程序。存在两种类型的异常: 条件和无条件异常。它们由以下指令产生：系统调用、陷阱和断点。

陷阱指令，它会根据比较结果产生条件异常

系统调用和断点指令，它们会产生无条件异常

2.15.4.3 条件传送指令

MIPS32 中包含了一些可以根据第三个通用寄存器值而条件性地将一个 CPU 通用寄存器的内容传送到另一个通用寄存器的指令。

2.15.4.4 NOP 指令

NOP 指令的编码实际上为一条全零指令。MIPS 处理器会将该编码特别处理为不执行任何操作，并优化指令的执行。此外，在任意处理器（包括架构的超标量实现形式）上， SSNOP 指令都占用一个指令发出周期。

2.15.5 协处理器指令

2.15.5.1 协处理器的作用

协处理器是一些备用执行单元，具有独立于 CPU 的寄存器文件。简单来说，MIPS 架构最多提供 4 个协处理器单元，编号为 0 至 3。 ISA 的每一级都定义了一些协处理器。协处理器 0 总是用于系统控制，协处理器 1 和 3 用于浮点单元。协处理器 2 保留用于特定于实现的用途。

协处理器可能具有两种不同的寄存器集：

- 协处理器通用寄存器
- 协处理器控制寄存器

每个寄存器集最多包含 32 个寄存器。协处理器计算指令可以使用任一寄存器集中的寄存器。

2.15.5.2 系统控制协处理器 0 （CP0）

所有 MIPS 处理器的系统控制器都实现为协处理器 0 （CP0），即**系统控制协处理器**。它提供了处理器控制、存储器管理和异常处理功能。

2.15.5.3 协处理器装载和存储指令

对于 CP0，未定义任何显式的装载和存储指令；仅对于 CP0，必须使用协处理器的传送指令来读写 CP0 寄存器。“协处理器装载和存储”（第 60 页）中概述了其他协处理器的装载和存储操作。

2.16 CPU 初始化

发生复位事件之后，软件需要初始化器件的以下部分。

2.16.1 通用寄存器

CPU 寄存器文件在上电时处于未知状态； r0 除外，它总是为 0。为了让硬件正确工作，并不需要初始化其他寄存器文件。但是根据软件环境，可能需要初始化几个寄存器。其中有：

- sp——堆栈指针
- gp——全局指针
- fp——帧指针

2.16.2 协处理器 0 状态

在退出引导代码之前，还需要初始化一些 CP0 状态。有一些异常会被 ERL = 1 或 EXL = 1 阻止，但复位时并不会清除这些异常。在退出引导代码时，可以将它们清除，以避免捕捉到虚假的异常。

表 2-13： CP0 初始化

CP0 寄存器	操作
CAUSE	WP （监视待处理）、 SW0/1 （软件中断）应清除。
CONFIG	通常， K0、 KU 和 K23 字段应先设置为所需的高速缓存一致性算法 （Cache Coherency Algorithm， CCA），然后再访问相应的存储区。
COUNT(1)	如果使用定时器中断，则应设置为已知值。
COMPARE(1)	如果使用定时器中断，则应设置为已知值。写入比较寄存器也会清除所有待处理的定时器中断（所以，应先设置 COUNT 寄存器，再设置 COMPARE 寄存器，以避免任何意外中断）。
STATUS	应设置所需的器件状态。
其他 CP0 状态	其他寄存器应先写入后再读取。一些寄存器并不是显式可写，它们的更新只是执行指令或捕捉异常的副产品。未初始化的位应在读取这些寄存器之后通过掩码操作去除。

注 1： 当 Count 寄存器等于 Compare 寄存器时，将会发出定时器中断。中断控制器中存在一个用于禁止向 CPU 传递该中断的屏蔽位 （如果需要）。

2.16.3 总线矩阵

在切换为用户模式之前或从 DRM 中执行之前，应先初始化 BMX。写入总线矩阵的值取决于要运行的应用程序的存储器布局。

2.17 复位的影响

2.17.1 $\overline{\text{MCLR}}$ 复位

硬件复位不会完全初始化 PIC32MX 内核。只有处理器状态中的一个最小子集会被清零。在非映射和非高速缓存代码空间中运行时，这已足以启动内核。软件可以在此后初始化所有其他处理器状态。上电复位会将器件置为一种已知状态。软复位可以通过将 MCLR 引脚置为有效来强制产生。实现该特性是为了与其他 MIPS 处理器保持兼容。在实际中，除 StatusSR 的设置外，两种复位的处理方式是相同的。

2.17.1.1 协处理器 0 状态

许多硬件初始化都在协处理器 0 中进行。

表 2-14： 复位清零或置 1 的位

位名称	清零或置 1	值	设置源	清零或置 1	值	设置源
StatusBEV	清零	1	复位或软复位			
StatusTS	清零	0	复位或软复位			
StatusSR	清零	0	复位	置 1	1	软复位
StatusNMI	清零	0	复位或软复位			
StatusERL	置 1	1	复位或软复位			
StatusRP	清零	0	复位或软复位			
与静态输入相关的配置字段	置 1	输入值	复位或软复位			
ConfigK0	置 1	010（非高速缓存）	复位或软复位			
ConfigKU	置 1	010（非高速缓存）	复位或软复位			
ConfigK23	置 1	010（非高速缓存）	复位或软复位			
DebugDM	清零	0	复位或软复位 (1)			
DebugLSNM	清零	0	复位或软复位			
DebugIBusEP	清零	0	复位或软复位			
DebugIEXI	清零	0	复位或软复位			
DebugSSt	清零	0	复位或软复位			

注 1： 除非使用 EJTAGBOOT 选项来引导进入 DEBUG （调试）模式。

2.17.1.2 总线状态机

当捕捉到复位或软复位异常时，所有待处理总线事务都会被中止，并且 SRAM 接口单元中的状态机会复位。

2.17.2 取指地址

在复位 / 软复位时，除非使用了 EJTAGBOOT 选项，否则取指操作会送到 VA 0xBFC00000（PA 0x1FC00000）。该地址处于 KSeg1 中，它是非映射和非高速缓存的。

2.17.3 WDT 复位

在 WDT 事件之后， CPU 寄存器的状态取决于 WDT 事件之前 CPU 的工作模式。

如果器件先前不是处于休眠模式， WDT 事件会将寄存器强制设为复位值。

2.18 相关应用笔记

本节列出了与手册本章内容相关的应用笔记。这些应用笔记可能并不是专为 PIC32MX 器件系列而编写的，但其概念是相近的，通过适当修改并受到一定限制即可使用。当前与 PIC32MX 系列的 CPU 相关的应用笔记包括：

标题	应用笔记编号
目前没有相关的应用笔记。	N/A

注： 如需获取更多 PIC32MX 系列器件的应用笔记和代码示例，请访问 Microchip 网站（www.microchip.com）。

2.19 版本历史

版本 A（2007 年 10 月）

这是本文档的初始版本。

版本 B（2008 年 4 月）

将状态修改为 “初稿”；修改了第 2.1 节 （主要特性）；修改了图 2-1；将 U-0 修改为 r-x。

版本 C（2008 年 5 月）

修改了图 2-1；增加了第 2.2.3 节 “内核定时器”；将保留位从 “保持为”修改为 “写入”。

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下， Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是 “牢不可破”的。

代码保护功能处于持续发展中。 Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案 （Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

ISBN: 978-1-60932-075-1

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 与位于俄勒冈州 Gresham 的全球总部、设计和晶圆生产厂及位于美国加利福尼亚州和印度的设计中心均通过了 ISO/TS-16949:2002 认证。公司在 PIC® MCU 与 dsPIC® DSC、KEELOQ® 跳码器件、串行 EEPROM、单片机外设、非易失性存储器和模拟产品方面的质量体系流程均符合 ISO/TS-16949:2002。此外，Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

全球销售及服务网点

美洲

公司总部 Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 1-480-792-7200
Fax: 1-480-792-7277
技术支持：
http://support.microchip.com
网址：www.microchip.com

亚特兰大 Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

波士顿 Boston
Westborough, MA
Tel: 1-774-760-0087
Fax: 1-774-760-0088

芝加哥 Chicago
Itasca, IL
Tel: 1-630-285-0071
Fax: 1-630-285-0075

克里夫兰 Cleveland
Independence, OH
Tel: 216-447-0464

Fax: 216-447-0643

达拉斯 Dallas
Addison, TX
Tel: 1-972-818-7423
Fax: 1-972-818-2924

底特律 Detroit
Farmington Hills, MI
Tel: 1-248-538-2250
Fax: 1-248-538-2260

科科莫 Kokomo
Kokomo, IN
Tel: 1-765-864-8360
Fax: 1-765-864-8387

洛杉矶 Los Angeles
Mission Viejo, CA
Tel: 1-949-462-9523
Fax: 1-949-462-9608

圣克拉拉 Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

加拿大多伦多 Toronto
Mississauga, Ontario,
Canada
Tel: 1-905-673-0699
Fax: 1-905-673-6509

亚太地区

亚太总部 Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200

Fax: 852-2401-3431

中国 – 北京
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

中国 – 成都
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

中国 – 重庆
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

中国 – 香港特别行政区
Tel: 852-2401-1200
Fax: 852-2401-3431

中国 – 南京
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

中国 – 青岛
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

中国 – 上海
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

中国 – 沈阳
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

中国 – 深圳
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

中国 – 武汉
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

中国 – 西安
Tel: 86-29-8833-7252

Fax: 86-29-8833-7256

中国 – 厦门
Tel: 86-592-238-8138

Fax: 86-592-238-8130

中国 – 珠海
Tel: 86-756-321-0040
Fax: 86-756-321-0049

台湾地区 – 高雄
Tel: 886-7-536-4818

Fax: 886-7-536-4803

台湾地区 – 台北
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

亚太地区

台湾地区 – 新竹
Tel: 886-3-6578-300
Fax: 886-3-6578-370

澳大利亚 Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

印度 India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4080

印度 India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

印度 India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

日本 Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

韩国 Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

韩国 Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 或
82-2-558-5934

马来西亚 Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

马来西亚 Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

菲律宾 Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

新加坡 Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

泰国 Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

欧洲

奥地利 Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

丹麦 Denmark-Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

法国 France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

德国 Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

意大利 Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

荷兰 Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

西班牙 Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

英国 UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

12/30/09