内部资料

LS232用户手册

龙芯技术服务中心

2009年5月14日

版 本 记 录

版本号	编者	日期	概要

目　录

图 目 录

表 目 录

1 微体系结构

LS232核是一款实现 MIPS32兼容且支持 DSP 扩展和 EJTAG 调试的双发射处理器，通过采用转移预测、寄存器重命名、乱序发射、路预测的指令CACHE、非阻塞的数据CACHE、写合并收集等技术来提高流水线的效率，形成了一款具有突出的性能价格比及性能功耗比的 32 位嵌入式处理器 IP 。

LS232处理器 IP 具有如下主要特点：

- MIPS32兼容的 32 位处理器，包含可配置的 DSP 部件和 64 位浮点部件（ FPU ）。
- 高效的双发射 + 五级流水结构（取指、译码、发射、执行并写回、提交）
- 支持寄存器重命名、动态调度、转移预测等乱序发射、乱序执行技术。其中包含 6 项的转移队列、 16 项重命名队列，采用 Gshare 转移猜测， BHT 为 256项。
- 包含两个定点、一个浮点、一个访存四个功能部件。
- 定点部件支持 DSP 扩展指令的运算。
- 浮点部件支持全流水的 64 位浮点加法和浮点乘法运算，硬件实现浮点除法运算。
- 专门的 SIMD 型多媒体加速指令。
- 32 项全相联 JTLB，每项映射两页，页大小可变，并具有可执行位设置以防止缓冲区溢出攻击。
- 4 项的指令 TLB 和 8 项的数据 TLB ，每项映射一页，页大小可变。

- 分离的一级指令 Cache 和数据 Cache，可配置为 1 路 /2路 /4路组相联，每路大小各为 4KB 。
- 支持非阻塞的 Cache 访问技术， 4 项 load 队列、 2 项 store队列、 3 项 miss 队列，最多容忍 5 条 store 指令 Cache 不命中和 4 条 load 指令 Cache 不命中。
- 支持 cached store 指令的写合并和 uncache 写加速技术。
- 支持 cache lock 技术和预取指令。
- 软 IP 级可配置：是否包含 DSP 指令支持、浮点部件、指令和数据 Cache 的大小（ 4KB/8KB/16KB）、定点乘法规模（半规模 / 全规模 / 串行）等可配置以满足不同应用的要求。
- 支持向量中断、中断优先级，软件可设置实时中断响应模式，此时最多 8 个时钟周期进入中断处理程序。
- 标准的 EJTAG 调试标准， 6 个指令断点， 2 个数据断点，支持快速下载，方便软硬件调试。
- 标准的 32 位 AMBA AHB接口和 32/64 位 AMBA AXI 接口。

上述特点使得LS232处理器 IP 具有执行效率高，功耗和成本低的优势，可在面向嵌入式的应用领域中得到广泛采用。

1.1 整体结构概述

LS232核是一款实现 MIPS32兼容且支持 DSP 扩展和 EJTAG 调试的双发射处理器，通过采用转移预测、寄存器重命名、乱序发射、路预测的指令CACHE、非阻塞的数据CACHE、写合并收集等技术来提高流水线的效率。

LS232核还可实现微体系结构的灵活配置，包括是否需要浮点功能部件，是否需要 TLB 和 CACHE部件，以及指令CACHE和数据 CACHE分别的 4KB 、 8KB 、 16KB大小配置。 LS232核的微体系结构如图 1–1所示，各部分参数说明如表 1–1。处理器采用 5 级流水线实现，下面按照流水线的顺序介绍 LS232核的处理过程及各个部件的功能。

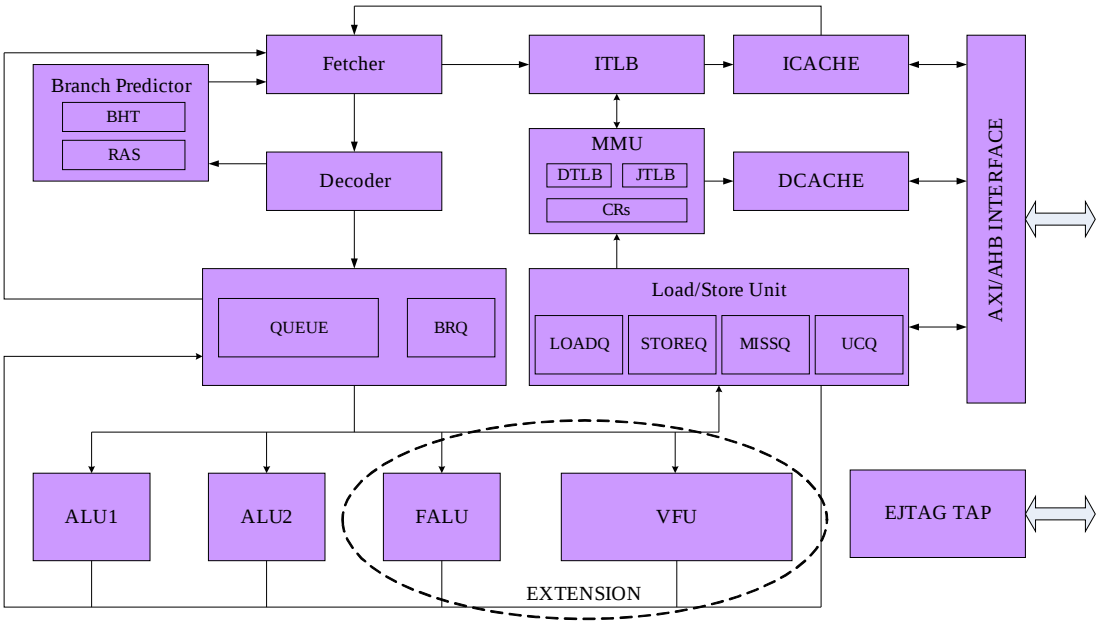


图 1–1 LS232 核微体系结构图

表 1–1 LS232 微体系结构参数

ITLB	4 项，每项对应 1 页，页大小从 4K 到 16M 可配置
ICACHE	4 路组相联，16KB 大小，每行 32B
BHT	256 项，采用 Gshare 算法
RAS	4 项
QUEUE	16 项
BRQ	6 项
GR	32 个 32 位通用寄存器，
FR	32 个单精度浮点寄存器或 16 个双精度浮点寄存器
ALU1	执行定点（/DSP)加减、移位、逻辑运算，转移指令
ALU2	包括 ALU1 所有指令和定点（/DSP)的乘除法指令； 1 组（或 4 组）HI/LO 寄存器

FALU	单精度和双精度浮点指令
DTLB	8 项，每项对应 1 页，页大小从 4K 到 16M 可配置
JTLB	32 项，每项对应奇偶页，页大小从 4K 到 16M 可配置
DCACHE	4 路组相联，16KB 大小，每行 32B
LOAD QUEUE	4 项
STORE QUEUE	2 项
MISS QUEUE	3 项

1) 取指流水级

LS232核的指令执行从取指开始，每拍可以同时取两条指令。根据 PC 的值，访问 ITLB 将虚地址转换成物理地址，如果发生 ITLB 查找不命中，则需要 1 拍的延迟从 JTLB 中选择相应项填充 ITLB 。同时根据路预测的结果访问指令 CACHE，读取相应 CACHE行的所有 TAG 和相应预测的路的指令，将 TAG 和转换后的物理地址高位比较，如果路预测正确，则将指令送往译码部件，否则需要 1 拍延迟读取正确的路的指令，同时修正路预测。如果发生 CACHE不命中，则发出内存访问请求。取指流水线还检查 PC 的合法性，例如核心态、用户态、地址错等，还有是否是 EJTAG 所设的指令断点，以及 TLB 例外等。

2) 译码流水级

取来的指令经过译码部件，翻译成 32 位 LS232核的内部操作。这主要是将原先不完全相同格式的指令，翻译成完全相同的规整格式的内部操作，方便内部的各个部分识别处理相应的操作。译码好的内部操作将送往操作队列

QUEUE，在进入操作队列的同时，查找之前的操作是否与当前操作存在数据相关，如果存在数据相关，则将原操作数指向操作队列中对应的操作，并记录相应信息，此过程是为了将来的乱序执行的重命名。

译码流水线还进行转移预测处理。当译码时遇到条件转移指令，LS232核采用 Gshare 算法通过 BHT 进行转移预测，并且计算出跳转的目标地址，修改 PC 的值。动态转移预测还可以进行配置，通过控制寄存器 config6可配置不同的转移预测策略，包括总是跳转、总是不跳转、总是向前跳转、总是向后跳转、转移历史表（ BHT ）动态预测、 Gshare 动态预测等六种策略，默认的是 Gshare。对于 likely的条件转移，LS232中总是预测 taken。当译码时遇到 BLINK 类指令（此为函数调用指令），则把 PC + 8压入地址返回栈 RAS 中。当译码时遇到 JR31指令（此为函数返回指令），则将 RAS 栈顶的值弹出用于预测该指令的跳转目标地址。

译码后的指令进入 QUEUE时，如果是一个基本块的开始，则需要同时占用 BRQ 的一项，并且记录该指令的 PC 。如果是一条转移指令，则与转移预测相关的信息记录到对应基本块的 BRQ 中，并且记录转移指令相对基本块的开始指令的偏移。

3) 发射流水级

LS232核采用乱序发射策略，将操作数准备好的指令送往功能部件，其中访存指令间仍维持原有的顺序发射执行。每拍可以同时发射两条指令，不过这两条指令中包含的定点（ /DSP ）乘除法指令、访存指令、浮点指令只能有一条。发射的指令从 GR 和 FR 中读取相应的源操作数，送往各个功能部件的 BUFFER，等待执行。发射策略是， issue_bus0优先发射准备好的访存

指令， issue_bus1优先发射准备好的第一条 alu指令（包括转移），另外再找一条准备好的浮点指令，选择空闲的总线发射，最后找一条准备好的乘法类（ acc）指令或第二条alu指令（不包括转移），选择空闲的总线发射。其中的访存指令和乘法类指令都是顺序发射，乘法类指令包括乘除指令和所有访问 hi/lo寄存器的指令。LS232中还将一些指令定义为 wait issue指令和 stall issue指令， wait issue指令需要等到队列头才能发射，stall issue指令将阻塞其后的所有指令的发射。

4) 执行流水级

各个功能部件根据指令执行相应的操作并且把结果写回操作队列。

定点（ /DSP）加减、移位、逻辑运算、转移指令一拍即执行写回，乘法指令需要 3拍执行写回，除法指令 3 – 35 拍。访问 hi/lo 寄存器的指令也走乘法流水线，也要 3拍写回。

浮点指令乘法 4拍，除法单精度 24 拍、双精度 56 拍，加减 3拍，定浮点转换 3拍，单双精度转换 2拍，分支 2拍，取负、绝对值等 2拍。

访存指令执行写回需要 3拍。第一拍通过 ADDR 模块计算虚地址。第二拍，访问DTLB将虚地址转换成物理地址，如果发生 DTLB查找不命中，则需要从 JTLB中选择相应项填充 DTLB。这级流水线也检查各种例外，还有EJTAG的数据断点。同时访问数据 CACHE，读取相应 CACHE行的 TAG和数据，将 TAG和转换后的物理地址高位比较，选择正确的数据送往 LOAD/STORE队列。第三拍，如果在数据 CACHE中命中，则 LOAD 结果写回 QUEUE， STORE操作写入数据CACHE。如果发生数据CACHE不命中，则物理地址送往 MISS队列，访问内存，填充数据 CACHE。 LS232核

还支持写合并的收集技术，即当STORE操作发生数据CACHE不命中后，STORE操作进入 MISS队列时，不会立即发起访存，会在 MISS队列中继续收集相同 CACHE行的 STORE操作，如果可以收集满完整的CACHE行，则不需要访问内存，直接将这行收集好的数据填充数据 CACHE，以此节约访存带宽。 LS232的写合并只要一项会在 MISS队列中等待合并，当发生两项写 miss ，则较早的那项将结束等待，开始发起内存请求。通过以上技术和各个访存队列， LS232核最多可以容忍5条不同 CACHE行的 STORE指令不命中（相同 CACHE行的 STORE指令不论多少条都能接收）和 4条在 3个CACHE行内的 LOAD 指令不命中。

LS232的功能部件，除了有结果总线 resbus ，还有 forward总线来表示下一拍写回的指令。这样，在发射时，选择准备好的指令，可以通过上一拍的forward总线来确定本拍要写回的指令，这种指令也可以发射。

乘法类指令、存数指令、 mtc0类指令由于都是在执行时修改寄存器或内存，所以这些指令的执行需要确保不会被取消，即这些指令前没有未执行的转移指令、会引起例外的指令等。因此这些指令的执行都需要等待一个 store_ok ，或 acc_write_ok等这样的信号。

转移指令经过功能部件计算后，先写回 brq，由 brq进行预测错的转移取消后，再写回 queue中。转移指令的target地址存在 queue的 imm 域中，而转移指令的地址则根据基本块的起始地址和转移指令相对基本块的偏移计算所得，或者根据 queue第一条指令的 pc 和转移指令相对queue_head的偏移计算所得。

5) 提交流水级

指令提交就是将操作队列中已经执行完的指令的运算结果写入到定点寄存器堆中，同时将相应的操作队列项释放，每拍可以提交两条指令。如果指令发生异常，则置异常信号，取消处理器中所有指令的执行，并且到相应的异常入口地址取指执行。如果发生转移预测错误，同样取消后续指令的执行，重新开始取指。

2 指令集概述

MIPS公司开发的 MIPS体系结构（ ISA）已经发展了 6个版本，依次分别为 MIPS I， MIPS II， MIPS III， MIPS IV， MIPS V和 MIPS 32/64，版本之间向前兼容。目前最新的是 MIPS 32/64体系结构的 Release2版本，其中，MIPS32体系结构基于 MIPSII体系结构的指令集，并补充了 MIPS III， IV 和 V中的部分指令增强其生成代码和移动数据的效率；MIPS64体系结构基于 MIPS V体系结构的指令集，兼容 MIPS 32 体系结构。

MIPS32是 MIPS公司为了统一 MIPS指令系统的不同版本，在 MIPS II的基础上进行扩充，并对系统态的指令进行规范而定义的。MIPS32指令系统的用户态指令融合了不同指令系统的优点，系统态指令也更加规范。

LS232处理器兼容 MIPS32的 Release2体系结构。MIPS32主要包括 ISA（Instruction Set Architecture）、PRA（Privileged Resource Architecture）、ASE（Application Specific Extensions）和 UDI（User Defined Instructions）四个部分，LS232处理器实现了 ISA、 PRA，以及 DSP ASE。并且 LS232还实现了龙芯多媒体扩展。

本章从 CPU， PRA 两个方面描述LS232指令结构系统，并分别给出指令列表，以及LS232实现的 CP0 寄存器的定义，关于 DSP 、浮点和多媒体部分可以参见 LS232 DSP扩展手册、LS232浮点扩展手册、LS232多媒体扩展手册等。LS232与 MIPS32 Release2兼容，具体的指令说明以及寄存器描述可以参见 MIPS32用户手册。

2.1 CPU寄存器

MIPS32体系结构定义了如下 CPU 寄存器:

(1) 32 个 32 位的通用寄存器。 GPRs (general purpose registers)。其中有两个被赋予了特殊含义: R0 , 0号通用寄存器,值永远为 0 ; R 3 ,1 3 1号通用寄存器, 被JAL, BLTZAL , BLTZALL, BGEZAL , 和 BGEZALL指令隐式的用作目标寄存器,存放返回地址。

(2)一对用于保存乘法、除法和乘加操作的结果的特殊寄存器。 HI 和 LO 。 HI 寄存器用于存放乘或者除运算结果的高位; LO 寄存器用于存放乘或者除运算结果的低位。

(3)一个特殊的程序计数器(PC)。这个寄存器程序不可直接访问。

2.2 CPU指令集

MIPS32体系结构的定义把 CPU 指令集按照功能划分为以下几组:存取、计算、跳转分支、协处理器和杂项指令。① Load 和 Store 访存指令在主存和通用寄存器之间移动数据。访存指令都是立即数指令(I 型),因为该指令模式所支持的唯一访存模式就是基址寄存器加上 16 位的对齐的偏移量。② Computational 计算型指令完成寄存器值的算术、逻辑、移位、乘法和除法操作。计算型指令包含了寄存器指令格式(R型,操作数和运算结果均保存在寄存器中)和立即数指令格式(I 型,其中一个操作数为一个 16 位的立即数)。③ Jump and Branch 跳转和分支指令改变程序的控制流。绝对地址跳转被称为“Jump(跳转)”(J 型或者 R型), PC (指令计数器)相

关的跳转指令被称为“ Branch(分支)”(I型)。④ Coprocessor协处理器指令完成协处理器内部的操作。协处理器的访存操作是I型指令。在MIPS32定义了 0号协处理器(系统处理器)和1号协处理器(浮点协处理器),用户可以自定义 2号协处理器的功能。⑤ Special特殊指令完成系统调用和断点操作。这些指令通常是 R型的。⑥ Exception异常指令引起跳转,根据异常号比较结果跳转到通用异常处理向量。这些指令包括 R型和 I型指令格式。

LS232 IP虽然与 MIPS32 Release2版本兼容,从功能上实现了MIPS32体系结构规定的所有 CPU指令,但是有些指令在实现了有细微的并不影响兼容性的差别,以下值得编程人员注意。

(1) pref、 prefx指令。

表 2-2到表 2-11列出了 LS232IP实现的 MIPS32指令中的 CPU指令。 MIPS32的浮点和 CP0协处理器指令在后面章节介绍。

表 2-2 CPU指令集:访存指令

OpCode	Description	MIPS ISA
LB	取字节	I
LBU	取无符号字节	I
LH	取半字	I
LHU	取无符号半字	I
LW	取字	I
LWU	取无符号字	I
LWL	取字左部	I
LWR	取字右部	I
LL	取标志处地址	I
SB	存字节	I

OpCode	Description	MIPS ISA
SH	存半字	I
SW	存字	I
SWL	存字左部	I
SWR	存字右部	I
SC	满足条件下存	I
SYNC	同步	I
PREF	预取	MIPS32

表 2-3 CPU 指令集：算术指令 (ALU 立即数)

OpCode	Description	MIPS ISA
ADDI	加立即数	I
ADDIU	加无符号立即数	I
SLTI	小于立即数设置	I
SLTIU	无符号小于立即数设置	I
ANDI	与立即数	I
ORI	或立即数	I
XORI	异或立即数	I
LUI	取立即数到高位	I

表 2-4 CPU 指令集：算术指令 (2 操作数)

OpCode	Description	MIPS ISA
CLO	计算前导 0 的个数	MIPS32
CLZ	计算前导 1 的个数	MIPS32

表 2-5 CPU 指令集：算术指令 (3 操作数 , R- 型)

OpCode	Description	MIPS ISA
ADD	加	I
ADDU	无符号加	I
SUB	减	I
SUBU	无符号减	I
SLT	小于设置	I

SLTU	无符号小于设置	I
AND	与	I
OR	或	I
XOR	异或	I
NOR	或非	I

表 2-6 CPU 指令集：乘法和除法指令

OpCode	Description	MIPS ISA
MADD	乘加	MIPS32
MADDU	无符号乘加	MIPS32
MSUB	乘减	MIPS32
MSUBU	无符号乘减	MIPS32
MUL	乘（结果放到 GPR）	MIPS32
MULT	乘（结果放到 HI,LO）	I
MULTU	无符号乘	I
DIV	除	I
DIVU	无符号除	I
MFHI	从 hi 寄存器取数到通用寄存器	I
MTHI	从通用寄存器存数到 hi 寄存器	I
MFLO	从 lo 寄存器取数到通用寄存器	I
MTLO	从通用寄存器存数到 lo 寄存器	I

表 2-7 CPU 指令集：跳转和分支指令

Opcode	Description	MIPS ISA
J	跳转	I
JAL	立即数调用子程序	I
JR	跳转到寄存器指向的指令	I
JALR	寄存器调用子程序	I
BEQ	相等则跳转	I
BNE	不等则跳转	I
BLEZ	小于等于 0 跳转	I
BGTZ	大于 0 跳转	I
BLTZ	小于 0 跳转	I
BGEZ	大于或等于 0 跳转	I

Opcode	Description	MIPS ISA
BLTZAL	小于 0 调用子程序	I
BGEZAL	大于或等于 0 调用子程序	I
BEQL	相等则 Likely 跳转	II
BNEL	不等则 Likely 跳转	II
BLEZL	小于或等于 0 则 Likely 跳转	II
BGTZL	大于 0 则 Likely 跳转	II
BLTZL	小于 0 则 Likely 跳转	II
BGEZL	大于或等于 0 则 Likely 跳转	II
BLTZALL	小于 0 则 Likely 调用子程序	II
BGEZALL	大于或等于 0 则 Likely 调用子程序	II

表 2-8 CPU 指令集：移位指令

OpCode	Description	MIPS ISA
SLL	逻辑左移	I
SRL	逻辑右移	I
SRA	算术右移	I
SLLV	可变的逻辑左移	I
SRLV	可变的逻辑右移	I
SRAV	可变的算术右移	I

表 2-9 CPU 指令集：特殊指令

OpCode	Description	MIPS ISA
SYSCALL	系统调用	I
BREAK	断点	I
MOVZ	等于 0 时移动	MIPS32
MOVN	不等于 0 时移动	MIPS32
MOVT	浮点真时移动	MIPS32
MOVF	浮点假时移动	MIPS32

表 2-10 CPU 指令集：异常指令

OpCode	Description	MIPS ISA
TGE	大于或等于陷入	II

TGEU	无符号数大于或等于陷入	II
TLT	小于陷入	II
TLTU	无符号数小于陷入	II
TEQ	等于陷入	II
TNE	不等陷入	II
TGEI	大于或等于立即数陷入	II
TGEIU	大于或等于无符号立即数陷入	II
TLTI	小于立即数陷入	II
TLTIU	小于无符号立即数陷入	II
TEQI	等于立即数陷入	II
TNEI	不等于立即数陷入	II

表 2-11 CPU 指令集： CP0 指令

OpCode	Description	MIPS ISA
MFC0	从 CP0 寄存器取	I
MTC0	往 CP0 寄存器写	I
TLBR	读索引的 TLB 项	III
TLBWI	写索引的 TLB 项	III
TLBWR	写随机的 TLB 项	III
TLBP	在 TLB 中搜索匹配项	III
CACHE	Cache 操作	III
ERET	异常返回	III

2.3 CP0 指令集

下面给出 LS232定义的 CP0 指令：

表 2-12 LS232 的 CP0 指令

OpCode	Description	MIPS ISA
MFC0	从 CP0 寄存器取	MIPS32
MTC0	写 CP0 寄存器	MIPS32
ERET	异常返回	MIPS32
DERET	Debug 返回	EJTAG
SDBBP	软件中断	EJTAG

CACHE 指令		
OpCode	Description	Target Cache
CACHE0	Index Invalidate	Instruction Cache
CACHE8	Index Store Tag	Instruction Cache
CACHE16	Hit Invalidate	Instruction Cache
CACHE28	Hit lock	Instruction Cache
CACHE1	Index WriteBack Invalidate	Data Cache
CACHE5	Index Load Tag	Data Cache
CACHE9	Index Store Tag	Data Cache
CACHE17	Hit Invalidate	Data Cache
CACHE21	Hit WriteBack Invalidate	Data Cache
Cache29	Hit lock	Data Cache

MIPS32只对 CACHE指令进行了必要的规范，其中有很多 CACHE指令是推荐或可选的， LS232IP 只实现了 CACHE指令中 MIPS32规定必须要实现的部分。具体可参见上表。

3 内存管理

LS232IP 提供了一个功能完备的内存管理单元（ MMU ），它利用片上的 TLB 实现虚拟地址到物理地址的转换。

本章节描述了处理器的虚拟地址和物理地址空间，虚拟地址到物理地址的转换， TLB 在实现这些转换时的操作， Cache ，以及为 TLB 提供软件接口的系统控制协处理器（ CP0 ）寄存器。

快速查找表 TLB

把虚拟地址映射成物理地址是由 TLB 来实现的， LS232IP 包括一个联合 TLB(JTLB)和独立的指令 TLB和数据 TLB各一个 (ITLB和 DTLB)。小 TLB 能缓解访存部件和取指部件对 JTLB的访问竞争，加快地址转换的速度。

JTLB

为了能够快速地进行虚拟地址到物理地址的映射， LS232IP 采用了集成在片上的全相联映射机制的 TLB ， JTLB用于指令和数据的地址映射，用它们的名字进行索引。

JTLB按奇 / 偶表项成对组织，把虚拟地址空间和地址空间标识符映射到 4G 的物理地址空间。在默认的情况下， JTLB有 32 对奇 / 偶表项，允许最多记录 64 项页映射关系。

有两个机制分别用来协助控制映射空间的大小和内存不同区域的替换策略。

第一，每一页的大小可以是 4KB 到 256MB，但必须是按 4 倍递增。

TLB 中每项的 pagemask 域用于记录映射的页的大小，该记录在写一个新的表项载入 TLB 中。因此操作系统可以支持不同大小的页表项以适用于不同的目的，但是同一虚地址所对应的相邻奇偶两页的页大小必须一致。

第二， LS232IP 在 TLB 缺失的时候采用随机替换的策略来选择要被替换的 TLB 表项。

虚实地址转换也有不经过 TLB 完成的，比如 KSEG0和 KSEG1内核地址空间段就是不进行页面映射的，其中的物理地址是由虚拟地址减去一个基址得到的。在一个新的映射中操作系统会把一定数量的页面驻留在 TLB 中，而不致于被随机替换出去，这种机制有利于使操作系统提高性能，避免死锁。这种机制也使实时系统比较方便地为某一关键软件提供特定入口。

对每个页来说，JTLB还维护该页面的 Cache 一致性属性，每个页都有特定的位来标记：不经过 Cache（ Uncached），非一致性 Cache（ Cacheable Noncoherent 或者是非 Cache 加速（ Uncached Accelerated ）。

数据 TLB

LS232IP 的数据 TLB(DTLB)有 8个表项，每个DTLB表项只能映射一页，页面大小由每一项的 pagemask 域来指定。数据地址的映射和指令地址的映射能并行执行，从而提高了性能。当 DTLB中的表项失效时，从 JTLB中查找相应的表项，随机选择一个 DTLB表项进行替换，DTLB的操作对用户是完全透明的。处理器保证 DTLB与 JTLB的一致，如果 JTLB被修改时 DTLB将被刷新。

指令 TLB

LS232IP 的指令 TLB(ITLB) 有 4 个表项，每个 ITLB 表项只能映射一页，页面大小由每一项的 pagemask 域来指定。数据地址的映射和指令地址的映射能并行执行，从而提高了性能。当 ITLB 中的表项失效时，从 JTLB 中查找相应的表项，随机选择一个 ITLB 表项进行替换，ITLB 的操作对用户是完全透明的。硬件保证 ITLB 与 JTLB 的一致，如果 JTLB 被修改时 ITLB 将被刷新。

命中和失效

如果虚拟地址与 TLB 中某个表项的虚拟地址一致（即 TLB 命中），则物理页面号就从 TLB 中取出，并和偏移连接组成物理地址。

如果虚拟地址与 TLB 中任何表项的虚拟地址都不一致（即 TLB 失效），则 CPU 产生一个异常，并由软件根据内存中存放的页表重新填写 TLB 。软件既可以重写一个指定的 TLB 表项，也可以使用硬件提供的机制重写任意一个 TLB 表项。

多项命中

LS232IP 对 TLB 中虚地址不只与一个表项的虚地址一致的情况，没有提供任何的探测和禁用机制。多项命中并不会物理地破坏 TLB ，因此多项命中的探测机制是不必要的。多项命中的情况没有被定义，因此软件要控制不要让多项命中的情况发生。

3.1 处理器模式

LS232IP 有 3 种工作模式，支持调式模式，但是与其它 MIPS IP 不同的是， LS232IP 只支持一种地址模式和一种尾端模式。

处理器工作模式

以下三种模式的处理器优先级依次降低：

- **内核模式**（最高的系统优先级）：在这种模式下处理器可以访问和改变任何寄存器，操作系统最内层的内核运行在内核模式；
- **管理模式**：处理器的优先级降低，操作系统的一些不太关键的部分运行在该模式；
- **用户模式**（最低的系统优先级）：该模式下使不同的用户间不致互相干扰。

三种模式的切换是由操作系统（在内核模式）置位状态寄存器 KSU 的相应位来实现的。此外，当出现一个错误（ ERL 位置位）或出现一个例外（ EXL 位置位）时，处理器被强制切换到内核模式。表 3-13 列出了三种模式切换时 KSU,EXL,ERL 的置位情况，空的表项可以不必关心。

表 3-13 处理器工作模式

KSU 4:3	ERL 2	EXL 1	描述
10	0	0	用户模式
01	0	0	管理模式
00	0	0	内核模式
	0	1	例外级别
	1		错误级别

调试模式

LS232IP 实现了完全兼容MIPS公司 EJTAG规范的调试功能，当满足一定的触发条件时，处理器会进入调试模式。更具体的介绍请参看第 章。

地址模式

LS232IP 只支持 64 位的虚拟地址模式。

尾端模式

LS232IP 只工作在小尾端模式。

3.2 地址空间

本节叙述的是虚拟地址空间，物理地址空间，和经过 TLB 进行虚实地址转换的方法。

虚拟地址空间

LS232IP 虚拟地址为 32 位，共 4G 的空间依据处理器处于不同的模式而划分成不同的段。具体的虚拟地址空间分布如下表所示，空项表示处于该模式下该段地址空间不可用：

表 3-14 LS232IP 地址空间的分配

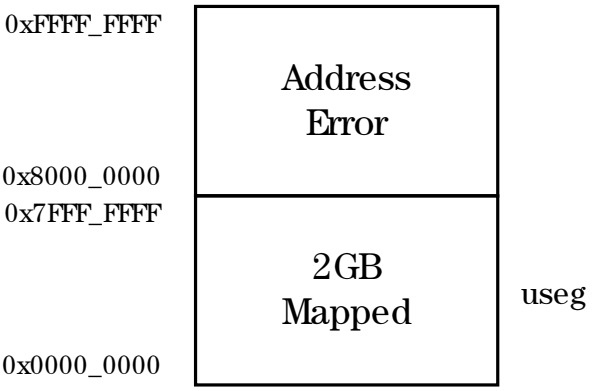
<table>
<tr><th rowspan="2">虚拟地址范围</th><th colspan="4">从不同模式下访问</th></tr>
<tr><th>用户模式</th><th>管理模式</th><th>核心模式</th><th>调试模式</th></tr>
<tr><td rowspan="2">0xFFFF_FFFF-0xFF40_0000</td><td rowspan="8"></td><td rowspan="3"></td><td rowspan="2">kseg3</td><td>kseg3</td></tr>
<tr><td>dseg</td></tr>
<tr><td>0xFF3F_FFFF-0xFF20_0000</td><td>kseg3</td></tr>
<tr><td>0xFF1F_FFFF-0xE000_0000</td><td>sseg</td><td>ksseg/ kseg2</td><td>ksseg/ kseg2</td></tr>
<tr><td>0xDFFF FFFF-0xC000 0000</td><td rowspan="3"></td><td>kseg1</td><td>kseg1</td></tr>
<tr><td>0xBFFF FFFF-0xA000 0000</td><td rowspan="2">kseg0</td><td rowspan="2">kseg0</td></tr>
<tr><td>0x9FFF FFFF-0x8000 0000</td></tr>
</table>

0x7FFF FFFF–0x0000 0000	useg	suseg	kuseg	kuseg

用户模式地址空间

在用户模式下，只有一个称为用户段 (User Segment)的单独、统一的虚拟地址空间，其大小为 2G(2^{31}) 字节，名字为 useg 。图 3-2 显示了用户虚拟地址空间。

当处理器的 Status寄存器的值同时满足三个条件：KSU=$1_2$0、 EXL=0、 ERL=0时，处理器工作在用户模式下。同时，



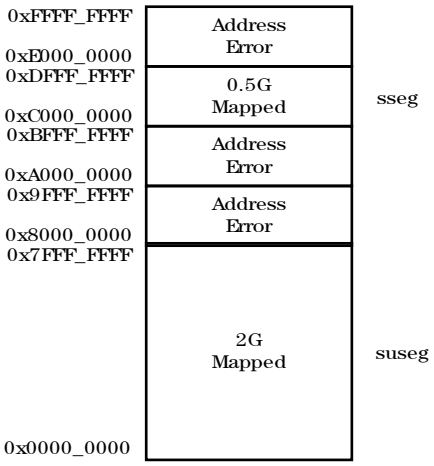
Debug 寄存器的 DM位必须为 0 。

图 3–2 用户模式下虚拟地址空间

用户模式下可用的虚拟地址的范围从 0x0000_0000到0x7FFF_FFFF，在用户模式下访问其它地址范围都将导致地址错误异常。

管理模式地址空间

管理模式是为分层结构的操作系统设计的。在分层结构的操作系统中，真正的内核运行在内核模式下，操作系统的其余部分运行在管理模式下。管理地址空间提供了管理模式下程序访问的代码和数据空间。图 3-3 显示了管理模

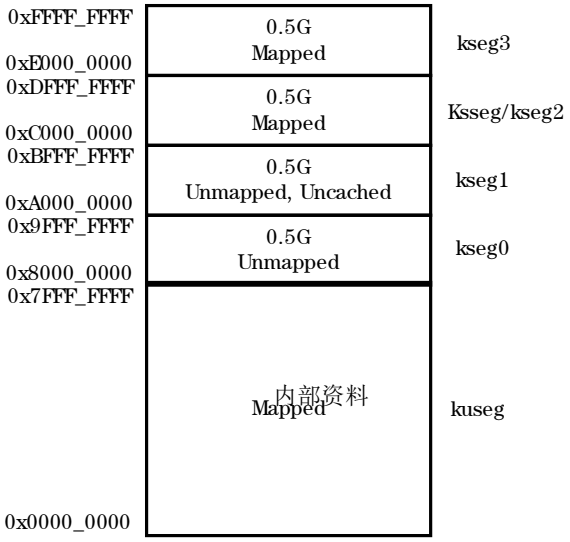


式下地址空间的分布概况

图 3–3 管理模式下虚拟地址空间

当处理器的 Status 寄存器的值同时满足三个条件：KSU=0b01、EXL=0、ERL=0时，处理器工作在管理模式下。同时，Debug 寄存器的 DM位必须为 0。

管理模式下可用的地址空间分为两段，用户段的范围从 0x0000_0000 到 0x7FFF_FFFF，管理段的范围从 0xC000_0000到 0xDFFF_FFFF。在管



理模式下任何对其它地址范围的访问都将导致地址错误异常。

核心模式地址空间

当处理器的 Status寄存器的值满足下述条件： KSU=00_2 或 EXL=1 或 ERL=1，且 Debug 寄存器的 DM=0 时，处理器工作在内核模式下。同时，每当处理器检测到一个非调试例外时便进入内核模式，并一直保持到执行例外返回指令（ ERET）。 ERET指令会将 ERL 或 EXL 位清 0，处理器返回到例外发生之前的模式。显示了核心模式下虚拟地址空间的分布概况。

图 3–4 核心模式下虚拟地址空间

核心模式下整个 4G 的虚拟地址空间都可以被访问，共分为五段：用户段的范围从 0x0000_0000到 0x7FFF_FFFF，当 Status寄存器的 ERL=0时，该段地址空间通过 TLB 进行地址转换，当 ERL=1 时，该段地址空间变为 Unmapped Uncached空间，它的物理地址直接等于虚拟地址；核心段 0 的范围从 0x8000_0000到 0x9FFF_FFFF，这段空间为 Unmapped空间，由 Config寄存器的 K0 域定义该段空间的 cache属性；核心段 1 的范围从 0xA000_0000到 0xBFFF_FFFF，这段空间为 Unmapped Uncached空间；管理段 / 核心段 2 的范围从 0xC000_0000到 0xDFFF_FFFF，核心段 3 的范围从 0xE000_0000到 0xFFFF_FFFF，这两段地址空间都通过 TLB 进行地址转换。

调试模式地址空间

调试模式下地址空间的分布与核心模式下的分布基本一致，不同之处在于从核心段 3（ kseg3）中划分出一段空间作为调式段，范围从 0xFF20_0000到 0xFF3F_FFFF，为一段 Unmapped 空间。在具体使用时，

这一段空间被进一步均分为 dmseg和 drseg两段，分别用于不同的访问，进一步的描述请参见“第 节 ”。

虚实地址转换

根据 MIPS32的规范， LS232的虚拟地址转换包括以下几种情况。①凡是 mapped的段，都通过访问 TLB 获得物理地址以及相应页的状态。② kseg0段的物理地址为虚地址减去 0x80000000得到，该段是否 cached根据 Config寄存器的 K0 字段决定。③ kseg1的物理地址为虚地址减去 0xA0000000得到，该段为 uncached段。④当 ERL 为 1时，最低 2G 空间 useg 的物理地址低 31 位直接从虚拟地址得到，并成为uncached访问方式。否则的话， useg 为 mapped,cached的访问模式。

（1）用虚页号（VPN）表示的虚拟地址（VA）与 TLB 中的对应域作比较；

（2）如果有一致的情况，则表示物理地址（PA）高位的页框号（PFN）从 TLB 中输出；

（3）偏移量 Offset 不经过 TLB，而是和 PFN 合并形成物理地址

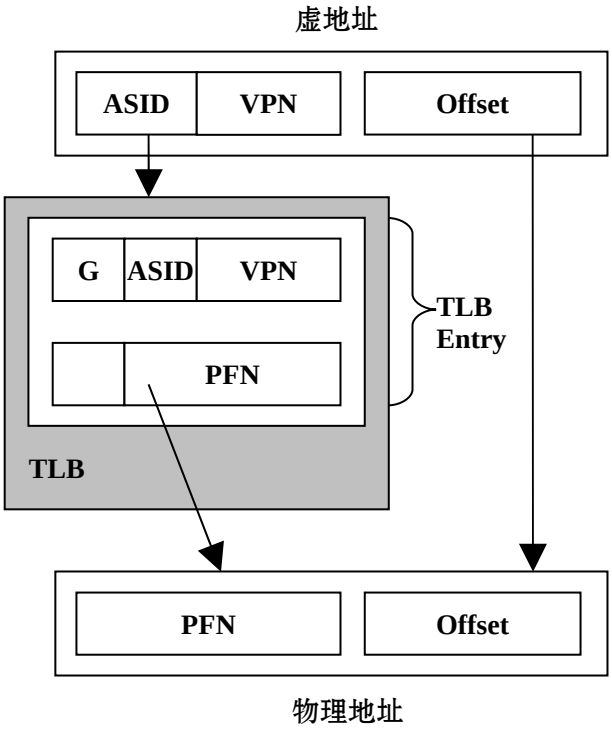


图 3-5 虚实地址转换概览

图 3-5给出了通过 TLB 进行虚实地址转换的概览。首先比较处理器给

出的虚拟地址和 TLB 中存放的虚拟地址。当虚页号（ VPN ）等于某个 TLB 表项的 VPN 域，并且如果下面两种情况中的任何一种成立：

- TLB 表项的 Global 位为 1
- 虚拟地址的 ASID 域一样。

TLB 就命中了。如果不满足以上条件，那么 CPU 会产生 TLB 失效异常，以使软件能够根据内存中存放的页表重新填写 TLB 。

如果 TLB 命中了，则物理页号将从 TLB 中取出，并与页内偏移量 Offset合并，形成物理地址。页内偏移量 Offset在虚实地址转换的过程中不经过 TLB 。

TLB 是一个全相联存储器，包括 32 项，提供 32 对奇 / 偶页面映像。页的大小为 4KB–16MB(以 4 的倍数递增)。 TLB 的每一项共有 96 位。其主要内容如下表所示，其中 NE 为表示不可执行位，是 LS232中为了防止缓冲区溢出攻击对每页增设的 .NE位为 1 时表示该页不可执行。

MASK		
VPN2	G	ASID

NE 0	PFN0	C0	D 0	V 0

NE 1	PFN1	C1	D 1	V 1

图 3–6 TLB 表项内容

由于 MIPS的 TLB 每一项包括一个奇数页和一个偶数页，因此进行虚地址匹配时使用 VPN2，即虚地址的页号除以 2。 MIPS 的页大小是可以变化的，由 TLB 中的 MASK 域决定，地址匹配时要考虑这个因素。此外，ASID域也应该匹配，除非 G域为 1。如果进行上述匹配后发现 TLB 中没有与虚地址匹配的 TLB 项，则产生 TLBLR或 TLBSR例外。如果匹配成功但相应的 TLB 有效位 V为 0，则产生 TLBLI 或 TLBSI 例外。对于存数操作，如果匹配成功且有效位为 1，但是 D为 0，则产生 MOD例外。图 3-7给出 TLB 地址转换过程。

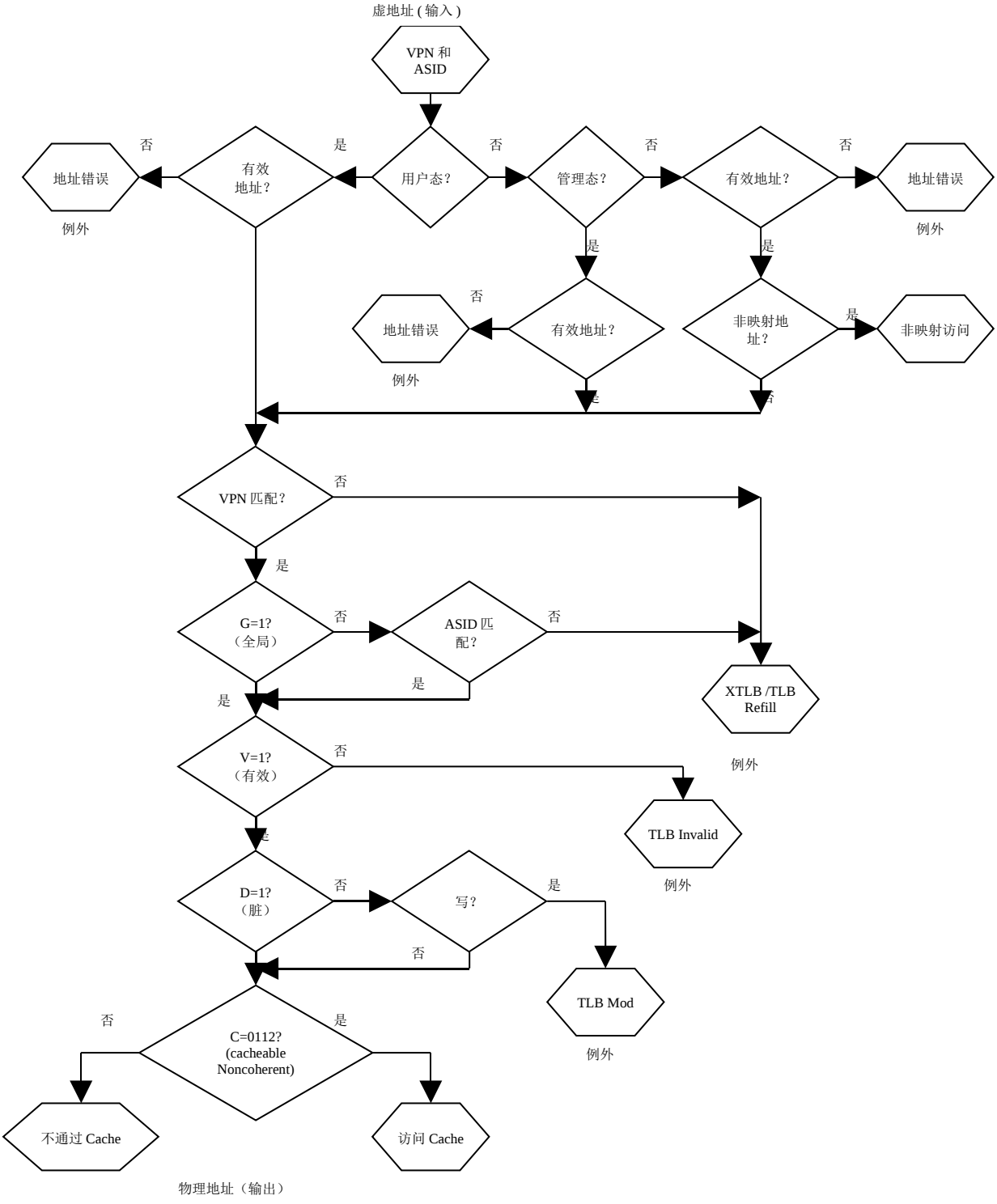

虚地址 (输入)
VPN 和 ASID
用户态？
否
是
有效地址？
否
地址错误
例外
管理态？
否
是
有效地址？
否
是
地址错误
例外
非映射地址？
是
否
非映射访问
VPN 匹配？
否
是
G=1?（全局）
否
是
ASID 匹配？
否
是
XTLB /TLB Refill
例外
V=1?（有效）
否
是
TLB Invalid
例外
D=1?（脏）
否
是
写？
是
否
TLB Mod
例外
C=0112?(cacheable Noncoherent)
否
是
不通过 Cache
访问 Cache
物理地址（输出）

图 3–7 TLB 地址转换

3.4 地址过滤窗口

LS232IP 为提升性能采用了推测执行技术，因此位于错误推测路径上的取指操作和取数操作都可能产生程序员不可控的处理器接口总线访问地址。另一方面，在一些系统中，存在一些设备包含有会引起系统死机的地址空洞。当错误推测路径引发的不可控地址恰好落在这些地址空洞的范围内，就会引起系统的死机。为解决该问题， LS232IP 提供了地址过滤窗口机制。该机制保证落在地址过滤窗口内的地址访问永远不会出现在处理器接口总线上。

现介绍地址过滤窗口的具体工作机制。 LS232IP 可最多配置 4 个地址过滤窗口，依次称作 0 号、 1 号、 2 号、 3 号地址过滤窗口。每个地址过滤窗口包含 3 个域，第 1 个域为使能域，用于该地址过滤窗口过滤功能的开启或关闭；第 2 个域为窗口基址 (base address)，记录的是过滤窗口的起始地址，特别注意的是这里记录的地址是物理地址；第 3 个域为过滤掩码 (mask)，用于控制地址窗口的大小。例如 64K字节大小的窗口，其过滤掩码为 32'hFFFF_0000。需要注意的是窗口的大小必须为 2 的 n 次方字节（ n 的取指范围从 8 到 31，包括 8 和 31），同时地址过滤窗口的起始地址仅从窗口边界开始计算。再以 64K字节大小的窗口为例，如果软件将窗口基址设为 32'h0001_0000，处理器则将视该窗口的范围为 32'h00010000~32'h0001FFFF。

地址过滤窗口的使能通过 CP0 第 22，select 7号寄存器的 [3:0] 位控制，第 i 位对应第 i 个过滤窗口的使能（ i=0,1,2,3），为 1 表示开启，为 0 表示关闭。地址过滤窗口的其他域按照 0 号窗口的基址、0 号窗口的掩码、 1 号窗口的基址、……、 3 号窗口的掩码的顺序组织成一个数组，即 0

号窗口的基址为数组的第 0个元素， 0号窗口的掩码为数组的第 1个元素，……余下各项可依次类推。对地址过滤窗口非使能控制域的访问即通过对该数组各元素的读写完成。其中，读操作的执行顺序是，先向 CP0第 22， select 5号寄存器写入需访问元素的数组下标，再读取 CP0第 22， select 6号寄存器，取回内容即为指定数组项的内容。写操作的执行顺序类似，区别在于当数组下标写入 CP0 第 22， select 5号寄存器后，需要将写入的内容写入到 CP0 第 22， select 6号寄存器。下面给出一段设置第 N个地址过滤窗口的参考代码，代码中 N、 BASE和 MASK 均为常数，代表窗口号，窗口的基址，窗口的掩码。

```
li      a1, 2×N                 # 设置 N 号窗口的基址
mtc0    a1, $22, 5              #
li      t0, BASE                #
mtc0    t0, $22, 6              #
li      a1, 2×N+1               # 设置 N 号窗口的掩码
mtc0    a1, $22, 5              #
li      t0, MASK                #
mtc0    t0, $22, 6              #
mfc0    t0, $22, 7
li      t1, 1                   # 这里可以直接将第 N 位置为 1
sll     t1, t1, N               #
or      t0, t0, t1              #
mtc0    t0, $22, 7              # 使能 N 号过滤窗口
```

4 Cache 的组织与操作

LS232IP 实现了分离的一级数据 cache和一级指令 cache。数据 cache 和指令 cache各自均可以按需求配置成 1 路、 2 路和 4 路，采用写回策略，替换采用随机替换策略，两者的cache行大小均为 32 字节，同时实现了以 cache行为单位的 cache锁机制。给出了两个 cache的一些参数。

表 4-15 cache 参数

参数	指令 Cache	数据 Cache
Cache 大小	4KB,8KB,16KB	4KB,8KB,16KB
相联度	1 路,2 路,4 路组相联	1 路,2 路,4 路组相联
替换策略	随机法	随机法
块大小(line size)	32 字节	32 字节
索引(Index)	虚地址 11:5 位	虚地址 11:5 位
标志(Tag)	物理地址 31:12 位	物理地址 31:12 位
写策略	不可写	写回法
读策略	阻塞	非阻塞
读顺序	关键字优先	关键字优先
写顺序	不可写	顺序式

访问一次一级 cache需要 3 个时钟周期。每个一级 cache都有其自身的数据通路，从而两个 cache可以同时访问。指令 cache和数据 cache的读通路是 64bit，回填通路为 256 位。

一级 Cache 采用虚地址索引和物理地址标志，虚地址索引可能会引起不一致问题，因此 LS232IP 不支持 4KB 以下大小的页。

LS232IP 的指令 cache采用阻塞式，即指令 cache失效会阻塞后续的取指；但数据 cache采用了非阻塞技术。非阻塞 Cache 是通过允许 Cache 失效访存操作后面的多个 Cache 失效或命中的访存操作继续进行，来提高系统的整体性能。在非阻塞 Cache 设计中， Cache 并不会在某个失效上暂停。

LS232IP 支持多重失效下的命中，最多容忍 5条 store指令 cache失效和4条 load指令 cache失效。当一级cache失效时，需要访问主存储器。为了尽可能最大限度地发挥 cache的优势，在使用访存数据的指令之前，尽可能早的执行相应的 load 操作。

针对那些需要顺序存取的 I/O系统， LS232IP 的默认设置是采用阻塞式的 Uncached 访问方式。

一级指令 cache

一级指令 Cache可配置为 1路、 2路、 4路，大小分别为 4KB 、 8KB 、 16KB。 Cache 块大小（通常也被称作 Cache 行）为 32 字节，可以存放8条指令。由于 LS232IP 采用的 64 位的读通路，所以每个时钟周期可以取两条指令送到超标量调度单元。

指令cache的组织

给出了一级指令 Cache 的组织结构。该 Cache 采用四路组相联的映射方式，其中每组包括 128个索引项。根据索引 (Index)选择相应的标志(Tag)和数据 (Data)。从 Cache 读出 Tag后，它被用来和虚地址中的被转换的部分进行比较，从而确定包含正确数据的组。

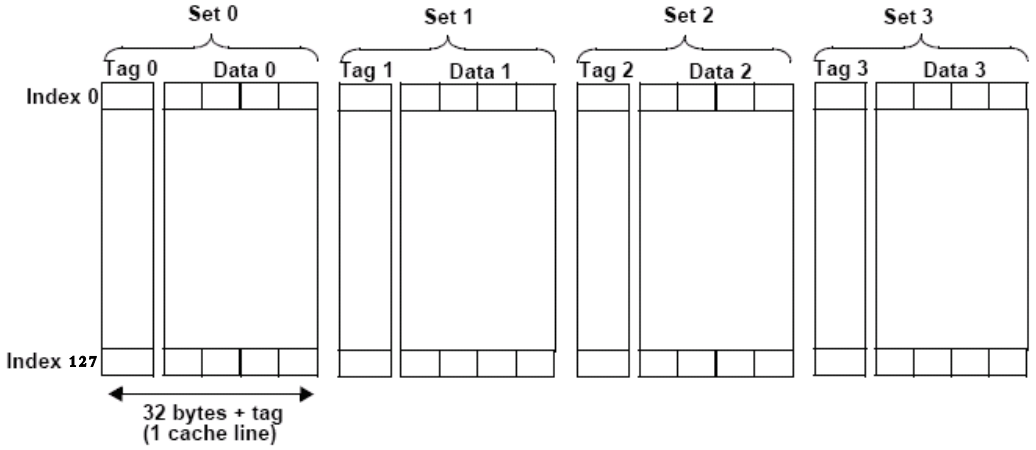
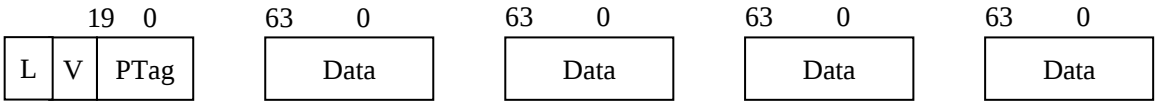


图 4–8指令 cache的组织



V……. Tag 有效位
L……. 锁标志位
PTag……20 位的物理地址 Tag （物理地址的位 31:12 ）
Data….. Cache Data

图 4–9指令 cache行格式

当一级指令 Cache 被索引时，四个组都会返回它们相应的 Cache 行，Cache 行大小为 32 字节， Cache 行采用了 28 位作为标志和 1 位作为有效位。图 4–9 描述了指令 Cache 行格式。

指令cache的访问

LS232IP 指令 Cache 采用虚地址索引和物理地址标志的组相联结构。图 4–10 给出了一个四路组相联指令 Cache 在一次访问时，虚地址如何被分解。

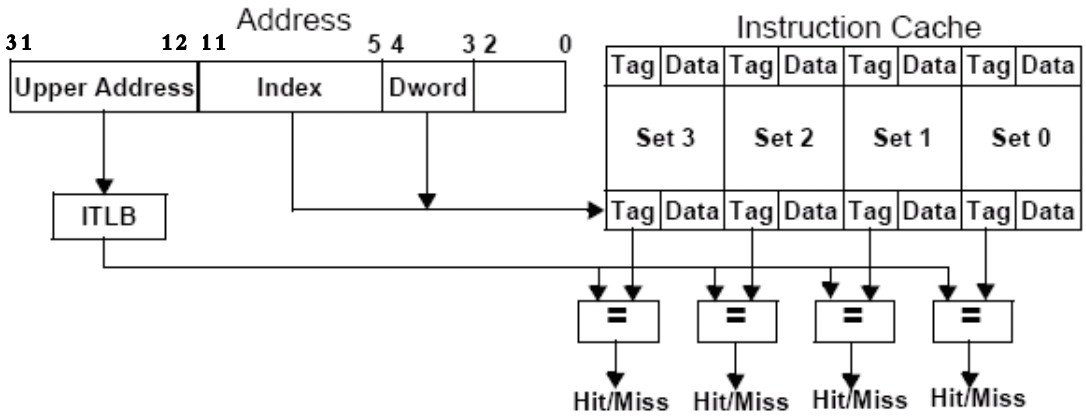


图 4–10 指令 cache访问

如图 4–10 所示，地址的低 12 位被用作指令 Cache 的索引。其中 11:5 位用于索引 128个项。其中每个项中又包含四个 64 位的双字，使用 4:3 位在这四个双字中进行选择。

当对 Cache 索引时，从 Cache 中取出四个块中的 Data 和相应的物理地址 Tag，同时，高位地址通过指令 TLB(ITLB) 进行转换，将转换后的地址与取出的四个组中的 Tags 进行比较，若存在一个 Tag与其匹配，则使用该组中的数据。这就被称为一次“一级 Cache 命中 (Hit)”。若四组的 Tag 都不与其匹配，那么中止操作，并开始访问主存储。这就被称为“一级 Cache 失效 (miss)”。

一级数据 cache

一级数据 Cache可配置为 1 路、 2 路、 4 路，大小分别为 4KB 、 8KB 、 16KB。 Cache 块大小（通常也被称作 Cache 行）为 32 字节，即可以存放 8 个字。数据 Cache 的读写数据通路都是 64 位。

数据 Cache 采用的写策略是写回法，即写数据到一级 Cache 的操作不会引起主存的更新。只有在数据 Cache 行被替换出去时，数据才会被写

到主存中。

数据 cache的组织

给出了一级数据 Cache 的组织结构。该 Cache 采用四路组相联的映射方式，其中每组包括 128个索引项。根据索引 (Index)选择相应的标志(Tag)和数据 (Data)。从 Cache 读出 Tag后，它被用来和虚地址中的被转换的部分进行比较，从而确定包含正确数据的组。

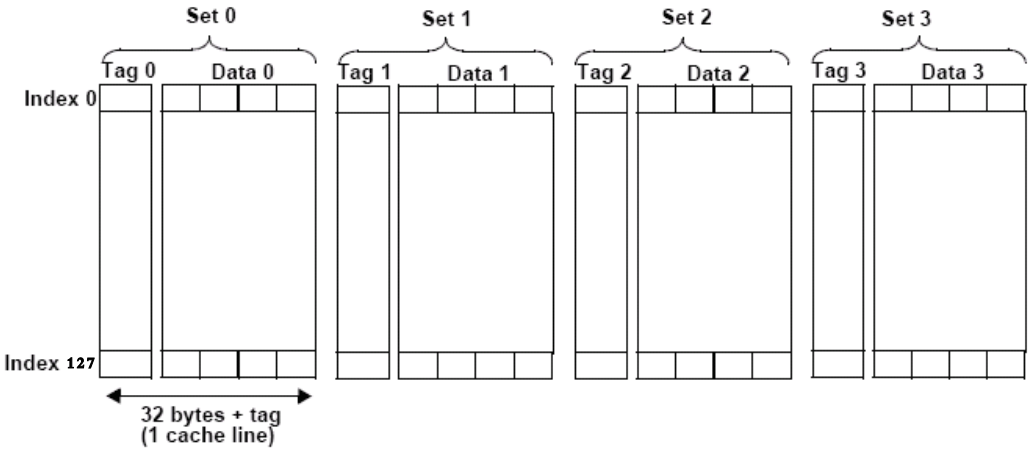
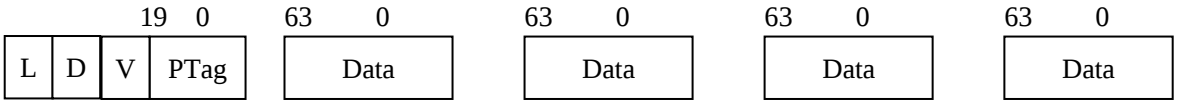


图 4–11 数据 cache的组织

当索引数据 Cache 时，四个组中都会返回它们各自相应的 Cache 行。Cache 块大小为 32 字节， Cache 行使用了 28 位作为物理标志地址 1



V……. Tag 有效位
D……. 脏位
L……. 锁标志位
PTag……20 位的物理地址 Tag （物理地址的位 31:12 ）
Data…… Cache Data

位脏位、 1 位状态位和 1 位锁标志位。图 4-12 给出了一个数据 Cache 行的格式。

图 4-12 数据 cache行格式

数据 cache 的访问

LS232IP 数据 Cache 采用虚地址索引和物理地址标志的组相联结构。图 4-13 给出了一个四路组相联数据 Cache 在一次访问时，虚地址如何被分解。

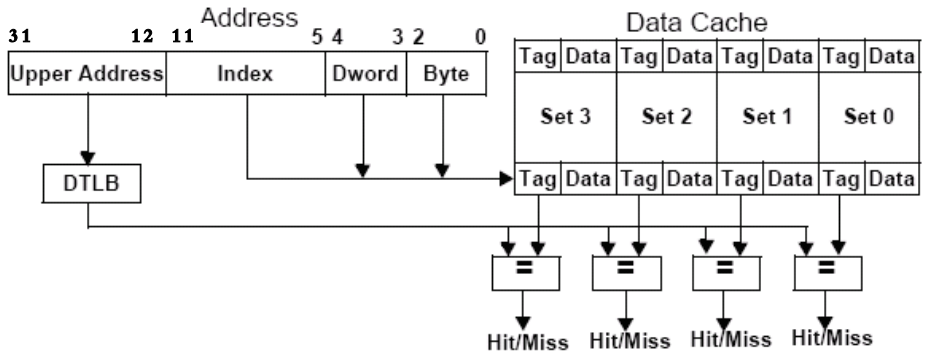


图 4-13 数据 cache访问

数据 cache 失效处理

数据 Cache 访问失效的取数指令，将访问主存，用从内存取回的值填充数据 Cache 。数据 Cache 访问失效的存数指令的普通处理类似于失效的取数指令，但是对于从 cache行起始位置开始的地址连续的失效存数指令采用了存填充缓冲区（ Store Fill Buffer ）优化策略。

Cache 失效的存数操作不需要留在访存队列中等待 Cache 块的填充。

如果 Cache 不命中，在该存数操作执行后把相应的写操作送到访存失效队列后立即退出访存队列以防止访存队列堵塞。如果失效队列的某项一直只有失效store写入，那么这一项延迟访问主存，等待收集为全修改 Cache 块。如果收集为全修改 Cache 块，直接填充数据 Cache 。如果等待收集过程中发生访存失效队列满或处理器执行同步指令、 Cache 指令需要清空访存失效队列等操作，则将未收集满的存数操作的值和从内存取回的 Cache 块的值进行合并送回数据 Cache 。该方法实现了 Store Fill Buffer 的功能，而且不需要设计独立的存数指令收集缓冲区，避免了增加额外的硬件开销，又避免了存数指令收集缓冲区与访存失效队列互相查询以保证数据一致性的开销。通过 Store Fill Buffer 的优化，有效地提高了处理器的带宽利用率。

uncached 存数操作加速

LS232的 uncached存数操作加速支持所有的存数指令，存数操作的收集动作可以从任何合法地址开始，最多可以收集从一个 cache行对齐地址开始的连续的 32 个字节。 uncached存数加速缓冲将一直收集数据直到满足下列触发条件时，才会将已经收集的内容写入到主存中。

uncached存数加速缓冲触发写的条件：

- 一行已经收集满
- uncached加速存数操作落在另一 32B cache 行地址上
- 普通 uncache 操作
- SYNC指令和 CACHE指令

Cache 算法和 Cache 一致性属性

LS232实现表 4-2所示的 Cache 算法和 Cache 一致性属性。

表 4-1 LS232 Cache 的一致性属性

属性分类	一致性代码
保留	0
保留	1
非高速缓存（Uncached）	2
非一致性高速缓存（Cacheable Noncoherent）	3
保留	4
保留	5
保留	6
非高速缓存加速（Uncached Accelerated）	7

非高速缓存 （Uncached，一致性代码 2）

如果某个页采用非高速缓存算法时，那么对于在该页中任何位置的 Load 或 Store 操作，处理器都直接发射一个双字，部分双字，字，部分字的读或写请求给主存，而不通过一级 Cache 。非高速缓存算法采用阻塞的方式实现。

非一致性高速缓存 （Cacheable Noncoherent，一致性代码 3）

一个具有该属性的行可以驻留在 Cache 中，相应的存数和取数操作都只访问一级 Cache 。当一级 Cache 失效时，处理器会从主存中取回数据，并将其写入一级 Cache 。

LS232的 Cache 采用写回策略，因此只有当 Cache 行发生替换或软件执行 Cache 操作主动将 Cache 行的内容写回时，被修改的 Cache 行内容才写回下一级 Cache 或主存中。

由于系统中存在多个主设备可以访问主存，因此需要使用一种机制来保证 Cache 和主存中的数据一致性，这种机制被称为 Cache 一致性协议。而非一致性高速缓存机制（ Cacheable Noncoherent ）是指处理器没有提供硬件机制来解决 Cache 一致性的问题，需要通过软件采用 Cache 指令来主动维护 Cache 的一致性。

非高速缓存加速 （Uncached Accelerated，一致性代码 7）

非高速缓存加速属性用于优化在一个连续的地址空间中完成的一系列 Uncached 存数操作。该优化方法是通过设置缓冲区来收集这种属性的存数操作。 LS232的非高速缓存加速支持所有的存数指令，存数操作的收集动作可以从任何合法地址开始，最多可以收集从一个cache行对齐地址开始的连续的 32 个字节。只要缓冲区尚允许写入，就可以把这些存数操作的数据存入缓冲区中。把数据存储到缓冲区中就和存储到 Cache 中一样。当缓冲区满的时候，将缓冲区的数据以块方式写回内存。若缓冲区尚未收集满时就遇到写回的触发条件，则收集工作中止，缓冲区中保存的数据按字节写方式逐个写回内存。

非高速缓存加速缓冲区触发写的条件：

- 一行已经收集满
- 非高速缓存加速属性存数操作落在另一 32B cache 行地址上
- 普通 uncache 操作
- SYNC指令和 CACHE指令

非高速缓存加速属性可以加速顺序的 Uncached 访问，它适用于对显示设备存储的快速输出访问。

Cache 的维护

在片上多级存储器结构中，必须要保证，在进程切换前所有修改的数据已经更新到外部存储器。为了刷新片上写缓冲区，软件上使用 SYNC指令。这条指令在所有悬挂的存数操作已经到达引脚外部总线前，和在所有悬挂的取数操作已经完成写相应目的寄存器前，将一直停顿处理器。

可以通过使用 Cache 指令来维护 Cache 。 LS232在一级数据 Cache 中使用了两条“Hit”型 Cache操作： Hit_Invalidate和 Hit_Writeback_Invalidate。前者将相关的 Cache 行内容无效，一般在处理器读取 DMA操作刚刚完成的设备输入缓存区内容前使用；后者将相关的 Cache 行内容写回下一级存储后再设置为无效，一般在处理器写即将开始 DMA 操作的设备输出缓冲区后使用。

5 CP0控制寄存器

MIPS 定义了包括 CP0 在内的特权体系结构作为规范的一部分。为了适应所有的应用，MIPS提供了规范的子集，允许根据需要只实现那些必要的特征，同时与 MIPS体系结构兼容。 LS232IP 实现了 MIPS32 Release1特权体系结构中必须实现部分的全部特征，和可选择实现的部分特征：如 DEBUG 寄存器， Performance Counter寄存器等。

本小节描述 LS232IP实现的 CP0 指令以及 CP0 的寄存器定义。 CP0 寄存器用于控制处理器的状态改变并报告处理器的当前状态。这些寄存器通过 MFC0指令来读或 MTC0指令来写。

当处理器运行在核心模式时或状态寄存器（ Status 寄存器）中的第 28

位（ CU0 ）被设置时，可以使用 CP0 指令。否则，执行 CP0 指令将产生“ CP0 协处理器不可用例外”。

表 5-16 列出了 LS232IP 实现的所有 CP0 寄存器。

表 5-16 LS232IP 实现的 CP0 寄存器

寄存器号	寄存器名字	描述
0	Index	可写的寄存器，用于指定需要读/写的 TLB 表项
1	Random	用于 TLB 替换的伪随机计数器
2	EntryLo0	TLB 表项低半部分中对应于偶虚页的内容（主要是物理页号）
3	EntryLo1	TLB 表项低半部分中对应于奇虚页的内容（主要是物理页号）
4	Context	32 位寻址模式下指向内核的虚拟页转换表（PTE）
5	Page Mask	设置 TLB 页大小的掩码值
6	Wired	固定连线的 TLB 表项数目（指不用于随机替换的低端 TLB 表项）
7	HWREna	读硬件寄存器时用到的 mask 位(R2)
8	BadVaddr	错误的虚地址
9	Count	计数器
10	EntryHi	TLB 表项的高半部分内容（虚页号和 ASID）
11	Compare	计数器比较
12	Status(select0)	处理器状态寄存器
	IntCtl(select1)	控制扩展的中断功能（R2）
	SRSCtl(select2)	控制对影子寄存器的操作（R2）
	SRSMap(select3)	影子寄存器与中断向量的对应关系(R2)
13	Cause	最近一次例外的原因
14	EPC	例外程序计数器
15	PRID	处理器修订版本标识号
	Ebase	保存异常当 BEV 为 0 时的向量入口的基址
16	Config0	配置寄存器（Cache 大小等）
	Config1	配置寄存器
	Config2	在没有二级 cache 的 R2 实现中，仅表示实现 Config3
	Config3	配置寄存器 （R2）
	Config6	配置分支预测的策略
17		

寄存器号	寄存器名字	描述
18		
19		
20		
21		保留
22		保留
23	Debug	EJTAG 使用的控制寄存器
24	DEPC	EJTAG debug 例外程序计数器
25	Control R0（s0）	性能计数器 0 的控制寄存器
	Counter R0（s1）	性能计数器 0 的计数器
	Control R1（s2）	性能计数器 1 的控制寄存器
	Control R1（s3）	性能计数器 1 的计数器
26		保留
27		保留
28	TagLo	CACHE TAG 寄存器的低半部分
29		
30	ErrorEPC	错误例外程序计数器
31	DESAVE	EJTAG debug 例外保存寄存器

5.1 Index 寄存器 (0,select 0)

Index寄存器是个 32 位可读 / 写的寄存器，其中最后六位的值用于索引 TLB 的表项。寄存器的最高位表示 TLB 探测 (TLBP)指令执行是否成功。

Index寄存器的值指示 TLB 读 (TLBR)和 TLB 索引写 (TLBWI)指令操作的 TLB 表项。

图 5-14 表示 Index寄存器的格式，表 5-17 描述了 Index寄存器各域的含义。

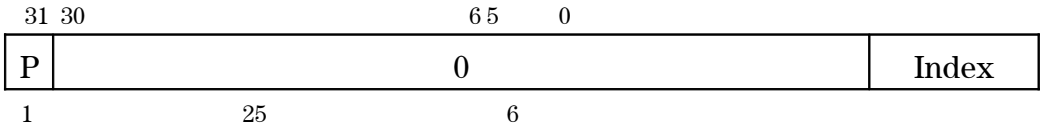


图 5-14 Index 寄存器

表 5-17 Index 寄存器各域描述

域	描述
P	探测失败。上一次 TLB 探测指令（TLBP）没有成功时置 1
Index	指示 TLB 读指令和 TLB 索引写指令操作的 TLB 表项的索引值
0	保留。必须按 0 写入，读时返回 0。

5.2 Random 寄存器 (1,select1)

Random 寄存器是个只读寄存器，其中低六位索引 TLB 的表项。每执行完一条指令，该寄存器值减 1。同时，寄存器值在一个上界和一个下界之间浮动，上下界具体是：

- 下界等于保留给操作系统专用的 TLB 项数（即 Wired寄存器的内容）。
- 上界等于整个 TLB 的项数减 1（最大为 32-1）。

Random 寄存器指示将由 TLB 随机写指令操作的 TLB 项。从这个目的来说，无需读此寄存器。但该寄存器是可读的，以验证处理器相应的操作是否正确。

为了简化测试，Random 寄存器在系统重起时置为上界。另外，当Wired寄存器被写时，该寄存器也要置为上界。

图 5-15 表示 Random 寄存器的格式，而表 5-18 描述 Random 寄存器各域的含义。

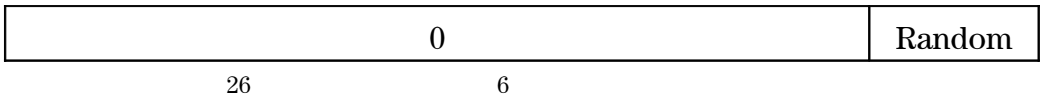


图 5-15 Random 寄存器

表 5-18 Random 寄存器各域

域	描述
Random	随机 TLB 索引值
0	保留。必须按 0 写入，读时返回 0。

5.3 EntryLo0 (2,select 0)以及 EntryLo1 (3, select 0) 寄存器

EntryLo 寄存器包括两个相同格式的寄存器：

- EntryLo0 用于偶虚页
- EntryLo1 用于奇虚页

EntryLo0和 EntryLo1寄存器都是可读 / 写寄存器。当执行 TLB 读和写操作时，它们分别包括 TLB 项中奇偶页的物理页号（ PFN ）。图 5-16 表示这些寄存器的格式。

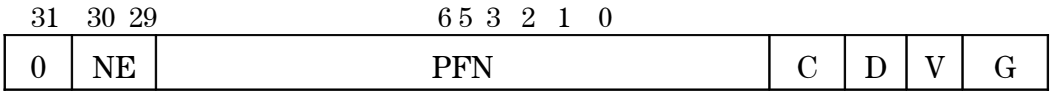


图 5-16 EntryLo0和 EntryLo1 寄存器

EntryLo0和 EntryLo1寄存器的 PFN 域是 32 位物理地址中的高 20 位（ 31 ： 12 ）。

表 5-19 EntryLo 寄存器域

域	描述
NE	不可执行位。1 表示不可执行，0 表示可执行。

PFN	页号，是物理地址的高位。
C	TLB 页的 Cache 一致性属性。
D	脏位。如果该位被设置，页面则标记为脏，也就是可写的。实际上这一位在软件中作为防止数据被更改的写保护使用。
V	有效位。当该位被设置时，说明 TLB 表项是有效的，否则将产生一个 TLBL 或 TLBS 例外。
G	全局位。当 EntryLo0 和 EntryLo1 中的 G 位都被设置为 1 时，处理器将在 TLB 查找时忽略 ASID。
0	保留。必须按 0 写入，读时返回 0。

在每个 TLB表项中只有一个全局位，在 TLB写操作中根据 EntryLo0[0]和 EntryLo1[0]的值写入。

5.4 Context (4, select 0)

Context寄存器是一个读 / 写寄存器，它包含指向页表中某一项的指针。该页表是一个操作系统数据结构，存储虚拟地址到物理地址的转换。

当 TLB 缺失时， CPU 将根据缺失转换从页表中加载 TLB 。一般情况下，操作系统使用 Context寄存器寻址页表中当前页的映射。 Context寄存器复制 BadVAddr 寄存器中的部分信息，但是该信息被安排成一种利于软件 TLB 例外处理程序处理的形式。

图 5-17 显示了 Context寄存器的格式；表 5-20 描述了上下文寄存器字段。

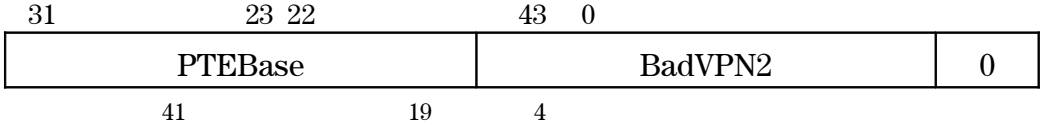


图 5-17 Context 寄存器

表 5-20 Context 寄存器域

域	描述
BadVPN2	当缺失时这一字段被硬件写。它包含最近不能进行有效转换的虚地址的虚页号(VPN)。
PTEBase	这一字段是操作系统使用的读/写字段。该字段写入的值允许操作系统将 Context 寄存器作为一个指向内存中当前页表的指针。
0	保留。必须按 0 写入，读时返回 0。

19 位的 BadVPN2字段包含导致 TLB 缺失的虚地址的 31:13位；第 12 位被排除是因为一个单一的 TLB 项映射到一个奇偶页对。对于一个 4K 字节的页尺寸，这一格式可以直接寻址 PTE 表项为 8 字节长且按对组织的页表。对于其它尺寸的页和 PTE ，移动和屏蔽这个值可以产生合适的地址。

5.5 PageMask 寄存器 (5, select 0)

PageMask 寄存器是个可读写的寄存器，在读写 TLB 的过程使用；它包含一个比较掩码，可为每个 TLB 表项设置不同的页大小，如表 5-21 。该寄存器的格式如图 5-18 。

TLB 读写操作使用该寄存器作为一个源或目的；当进行虚实地址转换时， TLB 中对应于 PageMask 寄存器的相应位指示虚地址位 24:13 中哪些位用于比较。当 MASK 域的值不是表 5-21 中的值时， TLB 的操作为未定义。 0 域为保留，必须按 0 写入，读时返回 0 。

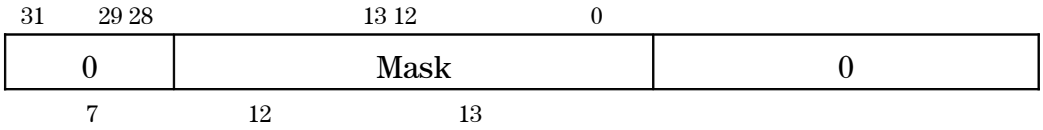


图 5-18 PageMask 寄存器

表 5-21 不同页大小的掩码（ Mask ）值

页大小	位															
	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13
4Kbytes	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16 Kbytes	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
64 Kbytes	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
256 Kbytes	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
1 Mbytes	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
4 Mbytes	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
16M bytes	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
64M bytes	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
256Mbytes	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

5.6 Wired 寄存器 (6, select 0)

Wired 寄存器是一个可读 / 写的寄存器，该寄存器的值指定了 TLB 中固定表项与随机表项之间的界限，如图 5–19 所示。 Wired 表项是固定的不可替换的表项，这些表项的内容不会被 TLB 写操作修改。而随机表项的内容可以被修改。

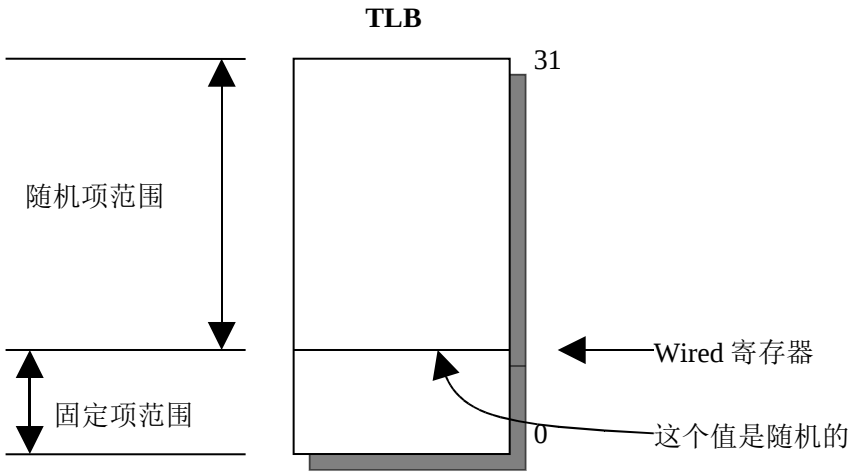


图 5-19 Wired 寄存器界限

Wired 寄存器在系统复位时置 0。写该寄存器的同时， Random 寄存器值要置为上限值（参阅前面的 Random 寄存器）。

图 5-20 表示 Wired 寄存器的格式；表 5-22 描述了寄存器的域。

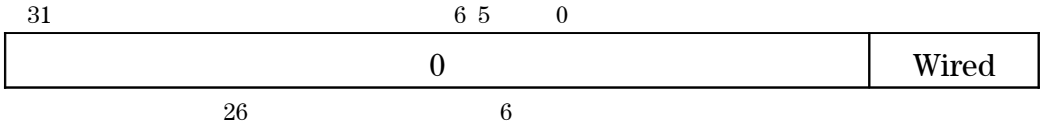


图 5-20 Wired 寄存器

表 5-22 Wired 寄存器域

域	描述
Wired	TLB 固定表项边界
0	保留。必须按 0 写入，读时返回 0。

5.7 HWREna 寄存器 (7, select0)

硬件寄存器读使能寄存器（ HWREna ）包含一位的掩码用来决定当使用 RDHWR 指令时，对应的硬件寄存器是否可读。

图 5-21 显示了 HWREna寄存器的格式，表 5-23 描述了 HWREna寄存器各个域的意义。

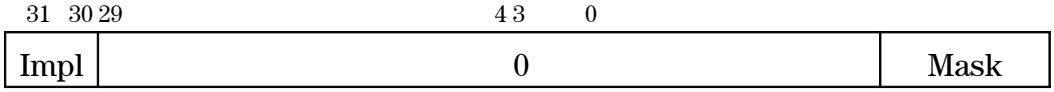


图 5-21 HWREna寄存器

表 5-23 HWREna寄存器域

域	描述
Impl	保留域，必须写为0，且读时返回0
Mask	读硬件寄存器的掩码位，若为1，则对应的硬件寄存器可读，否则不可读
0	保留。必须按0写入，读时返回0。

5.8 BadVAddr寄存器 (8, select0)

错误虚地址寄存器（ BadVAddr ）是一个只读寄存器，它记录了最近一次导致 TLB 或寻址错误例外的虚拟地址。除非发生软件复位， NMI 或 Cache 错误例外， BadVAddr 寄存器将一直保持不变。否则这个寄存器就是未定义的。

图 5-22 显示了错误虚地址寄存器的格式。

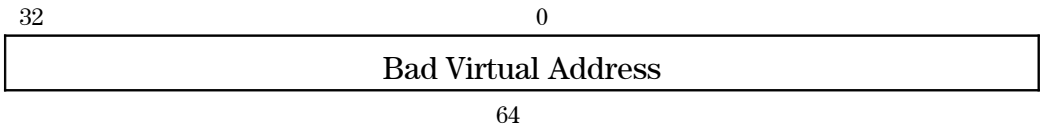


图 5-22 BadVAddr 寄存器

5.9 Count寄存器 (9,select0)以及 Compare寄存器 (11)

Count 寄存器和 Compare寄存器都是 32 位读写寄存器，他们的模式如所示。

Count 寄存器作为一个实时的定时器工作，每两个时钟周期增 1。

Compare寄存器用来在特定的时刻生成一个中断，该寄存器被写入一个值，图 5-23 并且不断地与 Count寄存器中的值比较。一旦这两个值相等，Cause 寄存器里的中断位 IP[7]被设置。当 Compare寄存器被再次写时这个中断位才被重置。

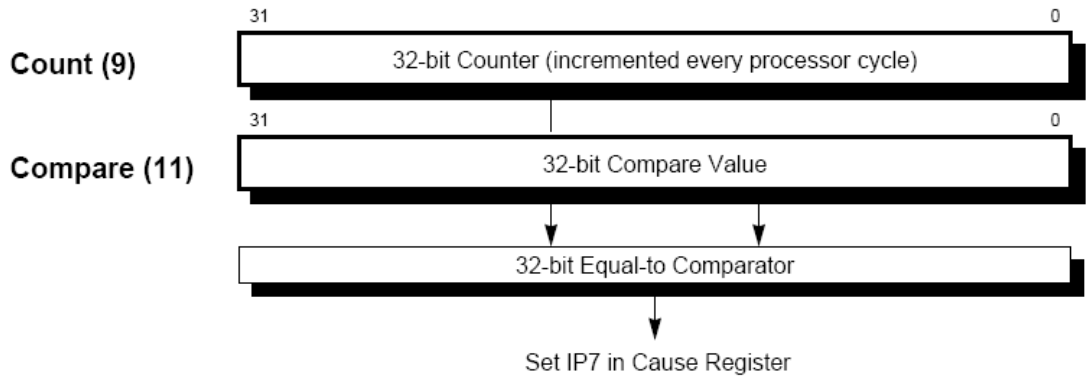


图 5-23 Count 寄存器和 Compare寄存器

5.10 EntryHi寄存器 (10)

EntryHi寄存器用于 TLB 读写时存放 TLB 表项的高位。

EntryHi寄存器可以被 TLB Probe, TLB Write Random, TLB Write Indexed, 和 TLB Read Indexed 指令访问。

图 5-24 表示 EntryHi寄存器的格式。表 5-24 表示 EntryHi寄存器的域。

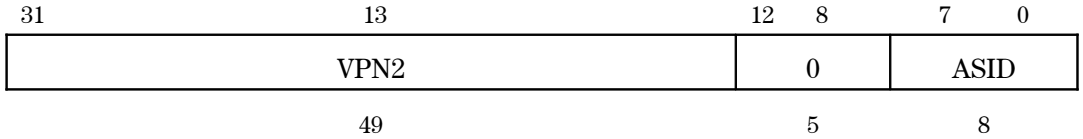


图 5-24 EntryHi 寄存器

表 5-24 EntryHi 寄存器域

域	描述
VPN2	虚页号除2（映射到双页）；虚拟地址的高位。
ASID	地址空间标识域。一个8位的域；用于让多个进程共享TLB；对于相同的虚页号，每个进程都与其他进程有不同的映射。
0	保留。必须按0写入，读时返回0。

VPN2域包含 32 位虚拟地址的 31:13 位。

当一个 TLB Refill ， TLB Invalid，或 TLB Modified例外发生时，没有匹配 TLB 表项的虚拟地址中虚拟页号（ VPN2）和 ASID将被加载到EntryHi寄存器。

5.11 Status寄存器 (12, select 0)

Status 寄存器 (SR) 是一个读写寄存器，它包括操作模式，中断允许和处理器状态诊断。下面列表描述了一些更重要的 Status 寄存器字段；图 5-25 显示了整个寄存器的格式，包括域的描述。其中重要的域有：

- 8位的中断屏蔽 (IM) 域控制 8个中断条件的使能。中断在被触发之前必须被使能，在 Status 寄存器的中断屏蔽域和 Cause 寄存器的中断待

定域相应的位都应该被置位。更多的信息，请参考 Cause 寄存器的中断待定（ IP ）域。

- 4 位的协处理器可用性（ CU ）域控制 4 个可能的协处理器的可用性。不管 CU0 位如何设置，在内核模式下 CP0 总是可用的。

31 28	27	26	25	24	23	22	21	20	19 16	15 8	7 5	4 3	2	1	0
CU (cu3:cu0)	0	0	RE	0	0	BEV	0	SR	0	IM7–IM0	0	KSU	ERL	EXL	IE
4	1	1	1	1	1	1	1	1	4	8	3	2	1	1	1

图 5–25 Status 寄存器

图 5–25 显示了 Status 寄存器的格式，表 5–25 描述了 Status 寄存器的域。

表 5–25 Status 寄存器域

域	描述
CU	控制 4 个协处理器单元的可用性。不管 CU0 位如何设置，在内核模式下 CP0 总是可用的。 1– 可用 0– 不可用 CU 域的初值是0011
0	保留。必须按 0 写入，读时返回 0。
BEV	控制例外向量的入口地址 0 – 正常 1 – 启动
SR	1 表示有软复位例外发生
IM	中断屏蔽：控制每一个外部、内部和软件中断的使能。如果中断被使能，将允许它触发，同时 Cause 寄存器的中断 Pending 字段相应的位被置位。 0 – 禁止 1 – 允许
KSU	模式位 11 ¶L 未定义

	10 □□ 普通用户 01 □□ 超级用户 00 □□ 核心
ERL	错误级。当发生复位，软件复位，NMI 或 Cache 错误时处理器将重置此位。 0 □□正常 1 □□错误
EXL	例外级。当一个不是由复位，软件复位或 Cache 错误引发的例外产生时，处理器将设置该位。
IE	中断使能。 0 □□禁用所有中断 1 □□使能所有中断

Status 寄存器模式和访问状态

下面描述 Status 寄存器中用于设置模式和访问状态的域：

- **中断使能：**当符合以下条件时，中断被使能：
  - IE = 1且
  - EXL = 0且
  - ERL = 0。

如果遇到这些条件， IM 位的设置允许中断。

- **操作模式：**当处理器处于普通用户、内核和超级用户模式时需要设置下述位域。
  - 当 KSU = 10_2, EXL = 0 和 ERL = 0时处理器处于普通用户态模式下。
  - 当 KSU = 01_2, EXL = 0 和 ERL = 0时处理器处于超级用户态模式下。
  - 当 KSU = 00_2, or EXL = 1或者 ERL = 1时处理器处于内核态模式下。

- **内核地址空间访问：** 当处理器处在内核模式时，可以访问内核地址空间。

- **超级用户地址空间访问：** 当处理器处在内核模式或超级用户模式时，可以访问超级用户地址空间。

- **用户地址空间访问：** 处理器在这三种操作模式下都可以访问用户地址空间。

Status 寄存器复位

复位时， Status 寄存器的值是 0x00400004。

5.12 IntCtl 寄存器 (12,select 1)

在 Rlease2的版本中， IntCtl寄存器用来控制扩展的中断特性。包括向量中断和外部中断。图 5-26 列出了 IntCtl寄存器的格式，表 5-26 描述了各个域的意义。

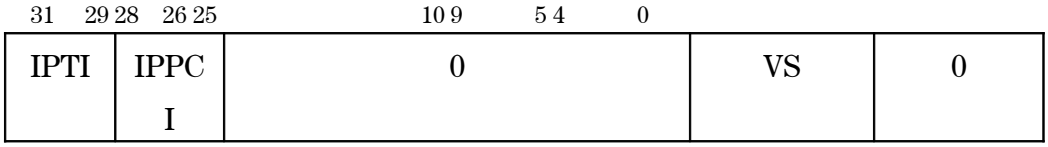


图 5-26 IntCtl 寄存器

表 5-26 IntCtl 寄存器域

域	描述
IPTI	对于向量中断而言，这个域表示时钟中断的中断号

<table>
<tr><td></td><td>
<table>
<tr><th>Encoding</th><th>IP bit</th><th>Hardware Interrupt Source</th></tr>
<tr><td>2</td><td>2</td><td>HW0</td></tr>
<tr><td>3</td><td>3</td><td>HW1</td></tr>
<tr><td>4</td><td>4</td><td>HW2</td></tr>
<tr><td>5</td><td>5</td><td>HW3</td></tr>
<tr><td>6</td><td>6</td><td>HW4</td></tr>
<tr><td>7</td><td>7</td><td>HW5</td></tr>
</table>
</td></tr>
<tr><td>IPPCI</td><td>表示 Performance Counter 的中断号。
<table>
<tr><th>Encoding</th><th>IP bit</th><th>Hardware Interrupt Source</th></tr>
<tr><td>2</td><td>2</td><td>HW0</td></tr>
<tr><td>3</td><td>3</td><td>HW1</td></tr>
<tr><td>4</td><td>4</td><td>HW2</td></tr>
<tr><td>5</td><td>5</td><td>HW3</td></tr>
<tr><td>6</td><td>6</td><td>HW4</td></tr>
<tr><td>7</td><td>7</td><td>HW5</td></tr>
</table>
</td></tr>
<tr><td>VS</td><td>在实现向量中断的版本中，这个域表示中断向量之间偏移差
<table>
<tr><th>Encoding</th><th>Spacing Between Vectors(hex)</th><th>Spacing Between Vectors(decimal)</th></tr>
<tr><td>0x00</td><td>0x000</td><td>0</td></tr>
<tr><td>0x01</td><td>0x020</td><td>32</td></tr>
<tr><td>0x02</td><td>0x40</td><td>64</td></tr>
<tr><td>0x04</td><td>0x80</td><td>128</td></tr>
<tr><td>0x08</td><td>0x100</td><td>256</td></tr>
<tr><td>0x10</td><td>0x200</td><td>512</td></tr>
</table>
</td></tr>
<tr><td>0</td><td>保留。必须按 0 写入，读时返回 0。</td></tr>
</table>

5.13 SRSCtl 寄存器 (12,select 2)

控制影子寄存器的读写。 LS232未实现影子寄存器。固所有域都写为 0 。

31 30 29 26 25 22 21 18 17 16 15 12 11 10 9 6 5 4 3 0

0	HSS	0	EICSS	0	ESS	0	PSS	0	CSS

图 5-27 SRSCtl 寄存器

表 5-27 SRSCtl 寄存器域

域	描述
HSS	表示实现的影子寄存器组数；若为 0 表示没有影子寄存器实现
EICSS	EIC 中断的影子寄存器组
ESS	异常的影子寄存器组
PSS	前一个影子寄存器组
CSS	当前影子寄存器组
0	保留。必须按 0 写入，读时返回 0。

5.14 SRSMap 寄存器 (12,select 3)

用来反映影子寄存器与异常向量的对应关系。LS232未实现影子寄存器。固所有域都写为 0 。

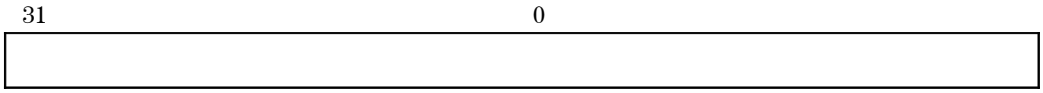


图 5-28 SRSMap寄存器

5.15 Cause 寄存器 (13,select 0)

32 位的可读写 Cause 寄存器描述了最近一个例外发生的原因。

图 5-29 显示了这一寄存器的域，表 5-28Cause 寄存器域 描述了 Cause 寄存器的域。一个 5 位例外码（ ExcCode ）指出了原因之一，如表 5-28 所示。

31	30	29	28	27	26	25	24	23	22	16	15	8	7	6	2	1	0
BD		TI		CE		DC		PCI		0	IV		0		IP7~IP0	0	Exc-Cod		0

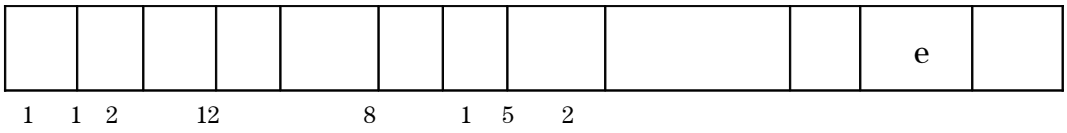


图 5-29 Cause 寄存器

表 5-28 Cause 寄存器域

域	描述
BD	指出最后采用的例外是否在分支延时槽中。 1 – 延时槽 0 – 正常
CE	当发生协处理器不可用例外时协处理器的单元编号。
DC	关掉 Count 寄存器。DC=1 时关掉 count 寄存器
PCI	Performance counter 中断,用来指示是否有待处理的 PC 中断
IV	指示中断向量是否用普通的异常向量（0 表示是，1 表示用特殊向量）
IP	指出等待的中断。该位将保持不变直到中断撤除。IP0~IP1 是软中断位，可由软件设置与清除。 1 – 中断等待 0 – 没有中断
ExcCode	例外码域 (见表5-11)
0	保留。必须按 0 写入，读时返回 0。

表 5-29 Cause 寄存器的 ExcCode 域

例外代码	Mnemonic	描述
0	INT	中断
1	MOD	TLB 修改例外
2	TLBL	TLB 例外（读或者取指令）
3	TLBS	TLB 例外（存储）
4	ADEL	地址错误例外（读或者取指令）
5	ADES	地址错误例外（存储）
6	IBE	总线错误例外（取指令）
7	DBE	总线错误例外（数据引用：读或存储）
8	SYS	系统调用例外
9	BP	断点例外
10	RI	保留指令例外
11	CPU	协处理器不可用例外

12	OV	算术溢出例外
13	TR	陷阱例外
14	–	保留
15	FPE	浮点例外
16 – 22	–	保留
23	WATCH	WATCH 例外
24 – 30	–	保留
31	–	保留

5.16 Exception Program Counter 寄存器 (14, select0)

例外程序计数器（ Exception Program Counter，简称 EPC ）是一个读 / 写寄存器，它包括例外处理结束后的继续处理地址。

对于同步例外， EPC 寄存器的内容是下面之一：

- 指令虚地址，这是导致例外的直接原因，或者
- 之前的分支或者跳转指令（当指令在分支延时槽中，指令延时位在 Cause 寄存器中被置位）的虚地址。

当 Status 寄存器中的 EXL 位被置 1 时，处理器不写 EPC 寄存器。

图 5–30 显示了 EPC 寄存器的格式。

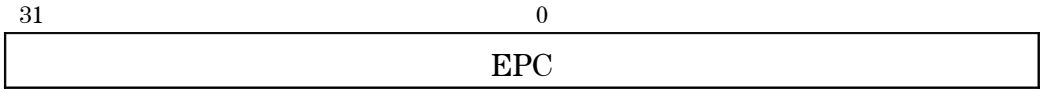


图 5–30 EPC 寄存器

5.17 Processor Revision Identifier (PRID) 寄存器 (15)

PRId寄存器是个 32 的只读寄存器，该寄存器包含了标定处理器和

CP0 版本的实现版本和修订版本的信息。图 5-31 表示了该寄存器的格式；表 5-30 描述了该寄存器的域。

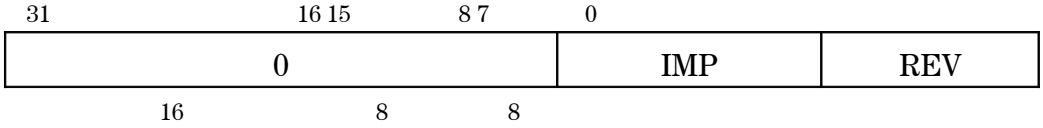


图 5-31 Processor Revision Identifier 寄存器

表 5-30 PRId 寄存器域

域	描述
IMP	实现版本号
REV	修订版本号
0	保留。必须按 0 写入，读时返回 0。

PRID寄存器的低位（ 7： 0 位）可用作修订版本的号码，而高位（ 15： 8）位可用作实现版本的号码。LS232实现版本号为 0x42，修订版本号为 0x00 。

版本号码的表示格式为 Y.X，其中 Y（ 7： 4 位）为主要版本号，而 X（ 3： 0 位）为小版本号。

版本号码可以区分一些处理器的版本，但不能保证处理器的任何改动要体现在 PRID 寄存器中，换句话说，不能保证版本号的改动必须体现处理器的修改。因为这个原因，寄存器的值没有给出，而软件也不能依赖 PRID 寄存器中的版本号来标识处理器。

5.18 Config0寄存器（16， select0)

Config 寄存器规定了 LS232处理器中各种配置选择项；表 5-31 列出了这些选项。

由 Config 寄存器的位 31:3所定义的一些配置选项，在复位时由硬件设置，而且作为只读状态位包括在 Config 寄存器中，用于软件的访问。其他配置选项（ Config 寄存器的位 2:0 ）是可读 / 写的并且由软件所控制。在复位时这些域是没有定义的。

Config 寄存器的配置是受限的。 Config 寄存器在 Cache 被使用之前应该由软件来初始化，并且，在做了任何改变后 Cache 应该重新初始化。

图 5-32 表示了 Config 寄存器的格式；表 5-31 Config 寄存器域描述了 Config 寄存器的域。 Config 寄存器的初值为 0x00030932 。

Config 寄存器

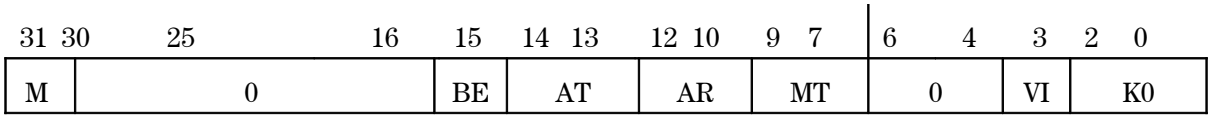


图 5-32 Config 寄存器

表 5-31 Config 寄存器域

域	描述
M	表示 config1 寄存器是否实现
BE	1：大尾端 0 ： 小尾端
AT	0：MIPS32 兼容 1 ： 仅能访问 3 2 位地址的 MIPS64 兼容 2 ： MIPS64 兼容 3 ： 保留
AR	0: release1 1:release2 2-7: 保留
MT	MMU 类型
VI	指令 cache 是否是虚 cache
K0	Kseg0 的 Cache 一致性算法。 7 – Uncached Accelerated

	3 – Cachable 2 – Uncached

5.19 Config1寄存器（16， select1)

Config1寄存器辅助 Config0寄存器规定了 LS232 处理器中其他各种配置选择项；其中包括 Icache, Dcache的各项配置参数，比如每路的组数，cache行的大小，相关联数。如果 cache行大小为 0，则表示 cache没有实现。

31	30 25	24 22	21 19	18 16	15 13	12 10	9 7	6	5	4 3	2	1	0
M	MMU Size –1	IS	IL	IA	DS	DL	DA	C2	0	PC	WR	CA	EP	FP

图 5–33 Config 寄存器

表 5–32 Config 寄存器域

域	描述
M	用来指示是否实现了 config2 寄存器
MMU Size–1	TLB 表项大小减一
IS	Icache 每路的组数
IL	Icache 每行的大小
IA	Icache 组相联数
DS	Dcache 每路的组数
DL	Dcache 每行的大小
DA	Dcache 组相联数
C2	Coprocessor2 是否实现
PC	Performance Counter 是否实现
WR	Watch 寄存器是否实现
CA	代码压缩是否实现
EP	EJTAG 是否实现
FP	浮点功能单元是否实现
0	保留域，必须写 0

5.20 Config2寄存器（16， select2)

Config2定义了二级缓存的参数，因为LS232没有实现二级缓存，固此仅用此寄存器的最高位表示实现了Config3寄存器。

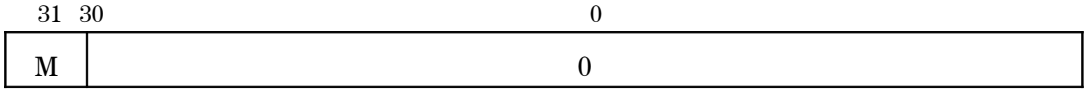


图 5-34 Config 寄存器

表 5-33 Config 寄存器域

域	描述
0	保留。必须按 0 写入，读时返回 0。
M	Config3 寄存器是否实现

5.21 Config3寄存器（16， select3)

Config3寄存器辅助 Config0寄存器规定了 LS232 处理器中其他各种配置选择项；

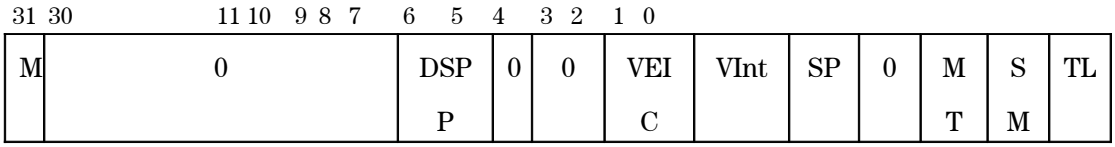


图 5-35 Config 寄存器

表 5-34 Config 寄存器域

域	描述
M	用来指示是否实现了 config2 寄存器
DSPP	DSP 是否实现
VEIC	是否实现了外部中断
VInt	是否实现了向量中断
SP	是否支持小物理页
MT	是否实现了 MTASE
SM	是否实现了 Smart ASE
TL	是否实现了 Trace 逻辑

5.22 Config6寄存器（16， select6)

Config6寄存器为 LS232 自己定义使用的控制寄存器，用来配置各种分支预测方式，以及表示是否实现实时中断。

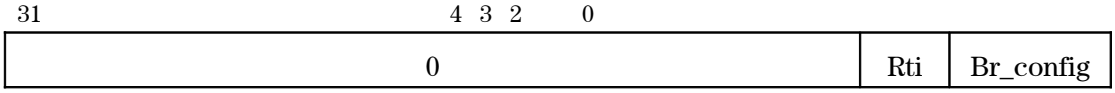


图 5-36 Config6 寄存器

表 5-35 Config6 寄存器域

域	描述
0	保留域
Rti	是否实现实时中断
Br_config	分支预测方式，具体如下： 编码 / 分支预测方式 3’b000 / Gshare 索引 bht 3’b001 / Pc 索引的 bht 3’b010 / 总是跳转 3’b011 / 总是不跳 3’b100 / 向前跳转 3’b101 / 向后跳转

5.23 Config7寄存器（16， select7)

Config6寄存器为 LS232可靠性增强版本中定义的控制寄存器，用来控制一级指令 Cache 和一级数据 Cache 的校验功能的开启与关闭。

31	6	5	4	3	0
0				Parity_EN	ECC_EN	0

图 5-37 Config7寄存器

表 5-36 Config7 寄存器域

域	描述
0	保留域
ECC_EN	一级数据 Cache ECC 校验功能的使能，1 表示开启该功能
Parity_EN	一级指令 Cache 奇偶校验功能的使能，1 表示开启该功能

5.24 Debug寄存器（ 23 ）

Debug 寄存器保存最近发生的调试例外以及调试模式例外的原因。同时，单步调试例外的使能也在这个寄存器控制。

在非调试模式下，只有 DM位与 EJTAGver 域是可读的，其他位的值都是没有意义的。 当发生调试例外或 / 和调试模式例外时，只需更新这些域：

- 发生调试例外或调试模式例外时，根据需要更新 DSS,DBp,DDBL, DDBS,DINT, DDBLImpr, DDBSImpr 域；
- 若在调试模式下发生例外，更新 DExcCode 域；发生调试例外时，这个域的值无意义；
- 若发生调试例外，更新 Halt 与 Doze 位；但若发生调试模式例外，则这两位的值没有意义；
- 发生调试例外或调试模式例外时，都需要更新 DBD 域。

图 5-38 显示了 Debug 寄存器的格式，而表 5-37 则列出了各个域的含义

31	30	29	28	27	26	25	24	20	19	18	17:16

DBD	DM	No DCR	LSNM	0	0	Cout DM	0				DD BSI mp r	DDBLI mpr	0	
15	14 10					9	8	7 6	5	4	3	2	1	0
0	DexcCode					No S St	S St	0	DI NT	D IB	DD BS	DDBL	D Bp	0S S

图 5–38 Debug 寄存器格式

表 5–37 Debug 寄存器域

域名	所在位	描述	读/写
DBD	32	指示发生例外的指令是否是延迟槽指令 0： no 1： yes	只读
DM	30	是否处于调试模式： 0： no 1： yes	只读
NoDCR	29	指示 Dmseg 是否存在： 0：存在 1：不存在	只读
LSNM	28	当有 dmseg 时，控制地址在 dmseg 的访存操作是否对 dmseg： 0：访问 Dmseg 1：访问系统内存	只读
CountDM	25	控制 Count 寄存器在调试模式下是否还计数： 0：停止计数	可读 可写

		1：继续计数	
DDBSImpr	19	Store 操作发生了带值比较的精确数据例外	只读
DDBLImpr	18	Load 操作发生了带值比较的精确数据例外	只读 可写
DExcCode	14:10	调试模式例外的例外编码	只读
NoSSt	9	指示是否实现单步调试例外 0：实现了单步调试例外 1：没有实现单步调试例外	只读
SSt	8	单步调试例外的使能位 0：单步调试例外不可用 1：单步调试例外可用	可读 可写
DINT	5	是否发生了调试中断例外	只读
DIB	4	是否发生了指令断点调试例外	只读
DDBS	3	store 操作发生了不带值比较的精确数据例外	只读
DDBL	2	Load 操作发生了不带值比较的精确数据例外	只读
DBp	1	是否发生了调试断点例外	只读
DSS	0	是否发生了单步调试例外	只读
0		必须写入 0；读出时返回 0。	只读

5.25 DEPC寄存器（ 24 ）

调试例外程序计数器（ Debug Exception P Counter，简称 DEPC ）是一个读 / 写寄存器，它包括例外处理结束后的继续处理地址。

当发生调试例外时， DEPC 寄存器的内容是下面之一：

- 指令虚地址，这是导致例外的直接原因，或者
- 之前的分支或者跳转指令（当指令在分支延时槽中，指令延时位在 Cause 寄存器中被置位）的虚地址。

DERET从调试模式返回时，从 DEPC 保存的地址继续执行。

图 5-39 显示了 DEPC 寄存器的格式。

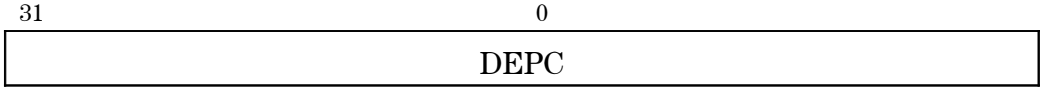


图 5-39 DEPC 寄存器

5.26 Performance Counter 寄存器 (25 , select0,1,2,3)

LS232处理器定义了两个性能计数器（ Performance Counter），他们分别映射到 CP0 寄存器 25 号的 select 1与 select 3寄存器。对应的关联控制寄存器分别影射到 P0 寄存器 25 号的 select 0与 select 2寄存器。每个计数器都是 32 位的读 / 写寄存器，并且在每次关联控制域中可数事件发生时自增。每个计数器都可以独立对一种事件计数。

31	11	10	5	4	3	2	1	0

0	Event	IE	U	S	K	EXL

图 5-40 控制寄存器性能计数寄存器

31 0
Counter0

图 5-41 性能计数器寄存器

当计数器的首位（ 31 位）变成 1（计数器溢出）时，计数器将触发一个中断 IP[6]，关联控制域使能中断。在计数器溢出后无论中断是否被告知，计数都将继续。表 5-38 控制域格式 描述 24 号寄存器控制域的格式。表 5-39 描述计数使能位的定义。表 5-40 与表 5-41 分别描述计数器 0 和计数器 1 各自的事件。

表 5-38 控制域格式

[10:5]	[4]	[3:0]
Event 1 Select	IP[6] Interrupt Enable	计数使能位 (K/S/U/EXL)

表 5-39 计数使能位定义

计数使能位	Count Qualifier(CP0 Status 寄存器域)
K	KSU = 0 (内核模式), EXL = 0, ERL = 0
S	KSU = 1 (超级用户模式), EXL = 0, ERL = 0
U	KSU = 2 (普通用户模式), EXL = 0, ERL = 0
EXL	EXL = 1, ERL = 0

表 5-40 计数器 0 事件

事件	内部信号	描述
0x0	Clock	时钟
0x1	Brbus_valid	执行过的分支总数
0x2	Reserved	保留
0x3	Brbus_jr31	jr31 指令的总数

事件	内部信号	描述
0x4	Imemread_valid	取指读访存的次数
0x5	Qissuebus_valid0	发射总线 0 有效的次数
0x6	Qissuebus_valid1	发射总线 1 有效的次数
0x7	Reserved	
0x8	Brbus_bht	用 bht 做分支预测的分支总数
0x9	Mreadreq_valid	访存的总次数（包括指令与数据）
0xa	Stalled_fetch	没有取指的时钟数
0xb	Queue_full	操作队列满的次数
0xc	Flush_pipeline_cycles	因为各种原因引起的清空流水线的时钟数
0xd	Ex_tlbr	tlbr 例外的次数
0xe	Ex_int	发生中断的次数
0xf	Inst_queue_write_cycle	往队列写指令的 cycle 数
0x10	Icache_access	读 icache 的次数
0x11	Dcache_access	读 dcache 的次数
0x12	Brbus_static	静态预测的分支总数
0x13	Loadq_full	Load queue 满而产生阻塞的时钟数
0x14	Storeq_full	Store queue 满而产生阻塞的时钟数
0x15	Missq_full	Miss queue 满而产生阻塞的时钟数
0x16	Miss_req_queue_full	Miss request queue 满而产生阻塞的时钟数
0x17	Dtlb_access	访问小 dtlb 的次数
0x18	Itlb_access	访问小 itlb 的次数
0x19	Insts_executed_alu1	在定点功能单元 1 执行的指令数
0x1a	Insts_executed_alu2	在定点功能单元 2 执行的指令数
0x1b	Insts_executed_addr	在访存功能单元 1 执行的指令数
0x1c	Insts_executed_falu	在浮点功能单元 1 执行的指令数
0x1d	Data_inst_conflict	数据和指令因访存接口而产生的冲突次数
0x1e	Not_store_ok_stall	Store_queue 因 head 项 store_ok!=1 处于满状态而产生阻塞效果的时钟数
0x1f	St_ld_conflict	Store 与 load 在 dcache 同一 bank 上产生访问冲突

表 5–41 计数器 1 的事件

事件	内部信号	描述
0x0	Pc_cmmit	提交的指令总数
0x1	Brbus_brerr	分支预测错总数
0x2	clock	时钟
0x3	Brbus_jr31mis	jr31 预测错总数
0x4	Dmemread_valid	数据读访存总数
0x5	reserved	

事件	内部信号	描述
0x6	reserved	
0x7	Duncache_valid	数据 uncached 总数
0x8	Brbus_bhtmiss	用 bht 做分支预测的分支预测错的总数
0x9	Mwrite_req_valid	访存写访存次数
0xa		
0xb	Brq_full	brq 满的次数
0xc		
0xd	Exbus_ex	发生异常的总数
0xe		
0xf	Inst_queue_write	写入队列的指令总数
0x10	Icache_hit	Icache 命中的次数
0x11	Dcache_hit	Dcache 命中的次数
0x12	Brbus_static_miss	Static 预测分支错误的次数
0x13	Icache_way_hit	Icache 路预测命中的次数
0x14	Icache_update	Icache 路预测失效更新的次数
0x15	Insts_fetched	所有取来的指令
0x16	Insts_stall_cycles	因为 icache miss 而阻赛的时钟数
0x17	Dtlb_miss	小 dtlb 失效但是大 tlbhit 的次数
0x18	Itlb_miss_tlb_hit	小 itlbmiss 但是大 tlbhit 的次数
0x19	Commitbus0_valid	提交总线 0 提交的指令数
0x1a	Commitbus1_valid	提交总线 1 提交的指令数
0x1b	Has_commit	有提交指令的时钟数
0x1c	Commit_two	同时提交两条指令的时钟数
0x1d	Data_inter_conflict	不同类型数据访问同时访存接口产生冲突
0x1e	Commit_ld_st	正常提交的 load 和 store 的指令数

5.27 ErrCtl(26，select 0)寄存器

（仅出现于可靠性扩展版本中，进一步了解请参考“ LS232可靠性扩展手册”）

5.28 CacheErr0寄存器（27， select0)

（仅出现于可靠性扩展版本中，进一步了解请参考“ LS232可靠性扩展手册”）

5.29 CacheErr1寄存器（27， select1)

（仅出现于可靠性扩展版本中，进一步了解请参考“ LS232可靠性扩展

手册”）

5.30 TagLo(28)寄存器

TagLo 寄存器是 32 位读 / 写寄存器，用于保存一级 Cache 的标签和状态，使用 CACHE和 MTC0指令往 Tag寄存器写。

图 5-42 显示了这些寄存器用于一级 Cache （ P-Cache ）操作的格式。

表 5-42 列出了 TagLo 和 TagHi寄存器中域的定义。

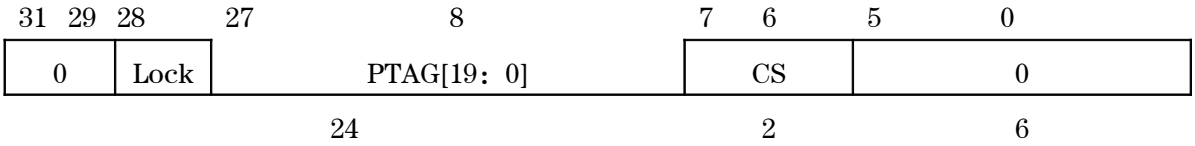


图 5-42 TagLo 寄存器 (P-Cache)

表 5-42 Cache Tag 寄存器域

域	描述
PTAG	指定物理地址的 31:12 位。
CS	指定 Cache 的状态。
Lock	该 cache 行是否被 lock 住，即不可被替换
0	保留。必须按 0 写入，读时返回 0。

5.31 TagLo(29)寄存器

TagHi寄存器是 32 位读 / 写寄存器。图 5-43 显示了这些寄存器的格式

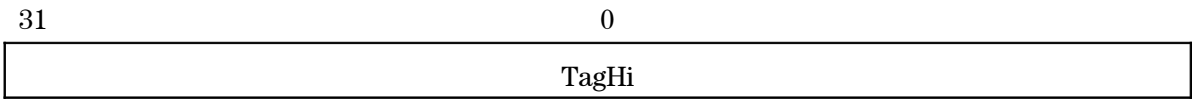


图 5-43 TagLo 寄存器 (P-Cache)

5.32 ErrorEPC 寄存器 (30)

除了用于 ECC 和奇偶错误例外外， ErrorEPC寄存器与 EPC 寄存器类似。它用于在复位、软件复位、和不可屏蔽中断（ NMI ）例外时存储程序计数器。

ErrorEPC是一个读写寄存器，它包括处理一个错误后指令重新开始执行的虚拟地址。图 5-44 显示了 ErrorEPC寄存器的格式。

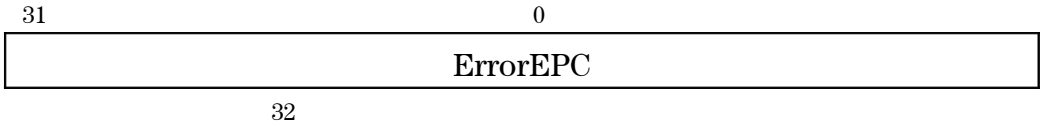


图 5-44 ErrorEPC 寄存器

6 处理器例外

本章介绍 LS232处理器例外，内容包括：中断，例外的产生及返回；例外向量位置和所支持的例外类型。其中对每一类支持的例外类型，介绍内容包括例外的原因，处理和服务。

LS232 IP 的例外处理遵循 MIPS32 R2 规范，且支持向量中断。表 6-43 给出了 LS232IP 实现的例外以及例外时的寄存器修改。

表 6-43 例外编码及寄存器修改

	status_EXL	status_ERL	status_BEV	status_NMI	status_SR	cause_exco	debug_DM	bebug_5:0	EPC	ErrorEPC	DEPC	cacheErr	FCR_cause	Entry_hi	Context	Badvaddr
reset		1	1	0	0					*						
soft reset		1	0	0	1					*						
NMI		1	0	1	0					*						
INT		1				0			*					*	*	*
MOD		1				1			*					*	*	*
TLBL		1				2			*					*	*	*
TLBS		1				3			*							*
ADEL		1				4			*							*
ADES		1				5			*							
IBE		1				6			*							
DBE		1				7			*							
SYS		1				8			*							

BP		1				9			*							
RI		1				10			*							
CPU		1				11			*							
OV		1				12			*							
TRAP		1				13			*							
FPE		1				15			*				*			
DSS							1	1		*						
DBP							1	2		*						
DDBL							1	4		*						
DDBS							1	8		*						
DIB							1	16		*						
DINT							1	32		*						

6.1 中断

LS232支持两个软件中断及六个硬件中断，六个硬件中断中包含两个特殊的中断：时钟中断和性能计数中断。这里需要注意，虽然 NMI（不可屏蔽中断）字面上有“中断”二字，但是更准确的描述应该是 NMI 异常，因为 NMI 异常并不在中断处理系统处理。

6.1.1 中断发生的条件

当以下条件都满足时，才可发生中断：

- 有中断请求
- 控制寄存器 Status 的 IE 位为 1

- 控制寄存器 Debug 的 DM位为 0
- 控制寄存器 Status 的 EXL 与 ERL 位都为 0

以上条件表明只有当处理器处于非例外，非错误，非调式的模式下，若中断被使能，那么相应的中断才可发生。

6.1.2 向量中断模式

LS232处理器支持向量中断模式。向量中断模式是在普通的中断模式基础上，增加了对8个中断做优先级排序，且给各个中断分配特定中断向量地址的功能。虽然 MIPS32 R2的向量中断模式允许使用影子寄存器组来加快中断处理进程，但是232处理器没有提供影子寄存器组，于是中断处理进程采用普通模式既可。当以下条件都满足时，表示处理器采用的是向量中断模式：

- 控制寄存器 Config3的 VInt 位为 1；
- 控制寄存器 Config3的 VEIC 位为 0；
- 控制寄存器 IntCtl的 VS 位不等于 0；
- 控制寄存器 Cause 的 IV 位为 1；
- 控制寄存器 Status 的 BEV 位为 0；

在向量中断模式下，六个硬件中断能被独立地解释，且有优先级。时钟中断和性能计数中断可以被设置在任意的六个硬件中断上，这通过控制寄存器IntCtl的 IPTI 与 IPPCI位来设置。LS232在实现上，统一将他们编码在硬件中断 5上。

当中断被使能时，若有中断发生（ CauseIP & StatusIM != 0 ） ，则按以下的优先级表来发出优先级最高的中断：

表 6-44 中断优先级

优先级	中断类型	中断源	中断请求的计算来源	中断向量号
最高	硬件中断	HW5	CauseIP7 & StatusIM7	7
		HW4	CauseIP6 & StatusIM6	6
		HW3	CauseIP5 & StatusIM5	5
		HW2	CauseIP4 & StatusIM4	4
		HW1	CauseIP3 & StatusIM3	3
		HW0	CauseIP2 & StatusIM2	2
最低	软件中断	SW1	CauseIP1 & StatusIM1	1
		SW0	CauseIP0 & StatusIM0	0

表 6-44 给出了 8 个中断的相对优先级以及其对应的中断向量号，其中软件中断的优先级低于所有的硬件中断。当有中断发生时，仅处理具有最高优先级的中断，且根据其对应的中断向量号来计算这个中断向量的偏移地址，如下图所示：

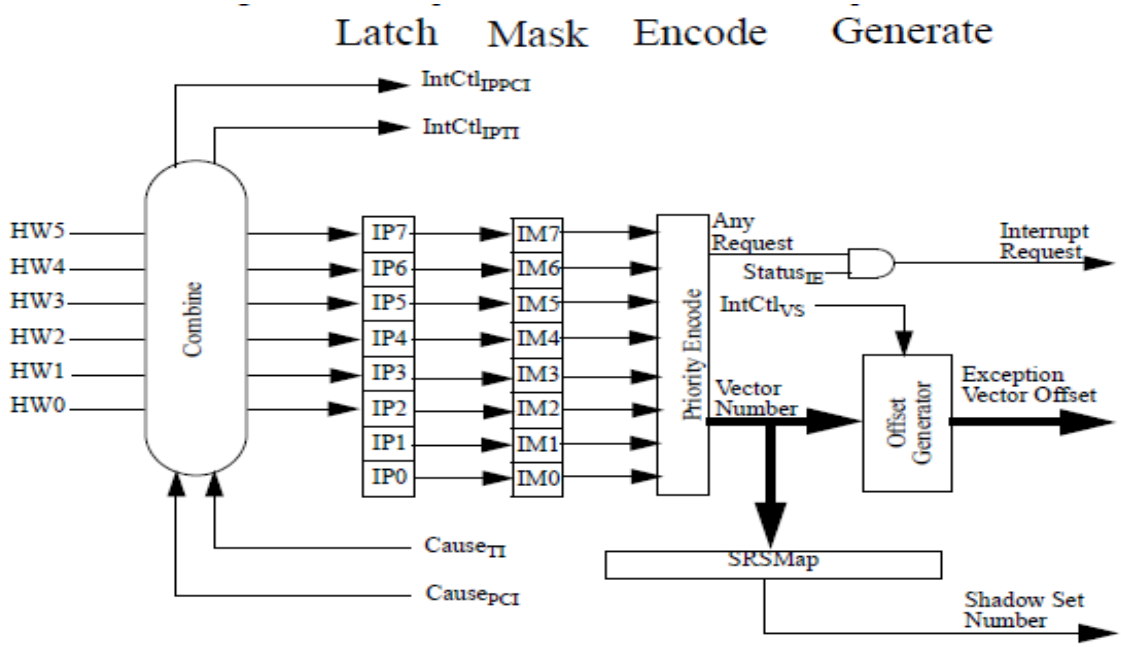


图 6-45　向量中断模式下中断的产生

6.1.3　中断向量的偏移地址计算

向量中断模式下，各个中断的中断向量号由中断控制逻辑产生，如图 6-45 所示。这个中断号结合控制寄存器 IntCtl的 VS 域，再加上中断向量偏移的基址从而算出中断向量的偏移地址。在向量中断模式下，中断向量偏移地址的基址是0x200。控制寄存器IntCtl的 VS 域用来指定两个中断向量之间的偏移。如果 VS 为 0，说明 8个中断的向量位于同一个地址，即处理器采用的是通常的中断模式；若不为 0 则说明处于向量中断模式。下表具体列出各个中断向量的偏移地址。

表 6-45　中断向量的偏移地址

中断向量号	控制寄存器 IntCtl 的 VS 域				
	0b00001	0b00010	0b00100	0b01000	0b10000
0	0x200	0x200	0x200	0x200	0x200
1	0x220	0x240	0x280	0x300	0x400

2	0x240	0x280	0x300	0x400	0x600
3	0x260	0x2c0	0x380	0x500	0x800
4	0x280	0x300	0x400	0x600	0xa00
5	0x2A0	0x340	0x480	0x700	0xc00
6	0x2C0	0x380	0x500	0x800	0xe00
7	0x2E0	0x3c0	0x580	0x900	0x1000

中断向量偏移地址的具体计算公式如下：

Vectoffset <– 0x200 + (vectoerNumber * (IntCtl.VS || 0b00000))

6.2 例外向量位置

冷重置、软重置和非屏蔽中断 (NMI) 例外的向量地址是专用的冷重置例外向量地址，这个地址既不通过 Cache 进行存取，也无需地址映射。而所有其它例外向量地址的形式都是基地址加上向量偏移地址。

启动时 (Boot–Time)例外向量 (状态寄存器中的 BEV 位 =1)地址位于既不通过 Cache 进行存取，也无需地址映射的地址空间。在正常操作期间 (BEV位=0)，普通例外的向量地址位于需要通过 Cache 进行存取的地址空间。

LS232是 MIPS32 R2 版本兼容，所以当 BEV 为 0 时，例外向量入口的基址由 EBase 寄存器中的地址指定，具体如表 6–46 所示，其列出了LS232的例外向量的基址。表 6–47 列出了 LS232例外向量的偏移地址，其中对中断而言，因为LS232实现了向量中断，表 6–47 仅列出其向量入口偏移地址的基址，其具体的偏移地址则根据控制寄存器 INTCtl中的 VS 域指定，具体参见前述中断章节。而表 6–48 则列出了 LS232的异常向量地址。

表 6–46 LS232 的例外向量基址

BEV 位	例外类型	例外向量地址
x	Cold Reset/ NMI/soft reset	0xBFC00000

x	EJTAG Debug(ProbEn ==0)	0XBFC00480
	EJTAG Debug(ProbEn==1)	0XFF200200
BEV = 0	others	Ebase31..12‖0x000
BEV = 1	others	0xBFC00200

表 6-47 LS232 例外向量的偏移

例外类型	例外向量地址
TLB Refil，EXL=0	0x000
Cold Reset/ NMI/softrest	None
General Exception	0x180
Interrupt,CauseIv==1	0x200（当 bev=0 时，这是 offset 的基址）
Others	0xBFC00200

表 6-48 LS232 异常向量入口地址

Exception	Statuts_BEV	Status_EXL	Cause_IV	EJTAG ProbEn	Vector
Reset, Soft reset, NMI	X	X	X	X	0xBFC0 0000
EJTAG Debug	X	X	X	0	0xBFC0 0480
EJTAG Debug	X	X	X	1	0xFF20 0200
TLB Refill	0	0	X	X	0x8000 0000
TLB Refill	0	1	X	X	0x8000 0180
TLB Refill	1	0	X	X	0xBFC0 0200
TLB Refill	1	1	X	X	0xBFC0 0380
Interrupt	0	0	0	X	0x8000 0180
Interrupt	0	0	1	X	0x8000 0200

Interrupt	1	0	0	X	0xBFC0 0380
Interrupt	1	0	1	X	0xBFC0 0400
All others	0	X	X	X	0x8000 0180
All others	1	X	X	X	0xBFC0 0380
‘x’ denotes don’t care					

6.3 例外的产生及返回

当处理器开始处理某个例外，状态寄存器的 EXL 位是被置为 1 的，这意味着系统运行在内核模式。在保存了适当的现场状态之后，例外处理程序通常将状态寄存器的 KSU 字段设定为内核模式，同时将 EXL 位置回为 0。当恢复现场状态并且重新执行时，处理程序则会把 KSU 字段恢复回上次的值，同时置 EXL 位为 1。

从例外返回也会将 EXL 位置为 0。

6.4 例外优先级

这一章的剩余部分将表 6-49 按照中给出的优先顺序依次介绍各个例外（对某些特定例外，如 TLB 例外和指令 / 数据例外，为了方便而放在一起介绍）。当一条指令同时产生一个以上例外时，只向处理器报告其中优先级最高的例外。有些例外并不是由当时正在执行的指令产生的，而有些例外可能被推迟处理。更多细节请查看本章对各个例外的单独介绍。

表 6-49 例外优先顺序

例外优先顺序
冷重置(最高优先级)
不可屏蔽中断 (NMI)

地址错误 – – 取指
TLB 重填 – – 取指
TLB 无效 – – 取指
总线错误 – –取指
整型溢出，陷阱，系统调用，端点，保留指令，协处理器不可用，浮点例外
地址错误 – – 数据存取
TLB 重填 – – 数据存取
TLB 无效 – – 数据存取
TLB 修改 – – 写数据
总线错误 – –数据存取
中断(最低优先级)

一般来说，下面各节介绍的例外先由硬件来处理，然后由软件来服务。

6.5 冷重置例外

原因

当系统第一次上电或者冷重置时，产生冷重置例外。该例外不可屏蔽。

处理

CPU 为这个例外提供了一个特殊的中断向量 0xBFC0 0000 。

冷重置向量地址属于无需地址映射和不通过 Cache 存取数据的 CPU 地址空间，因此处理这个例外不必初始化 TLB 或 Cache 。这也意味着即使 Cache 和 TLB 处于不确定状态，处理器也可以取出并执行指令。

当例外发生时， CPU 中所有寄存器内容是不确定的，但下列寄存器域除外：

- 状态（ Status ）寄存器的初值为 0x00400004 SR 位被清为 0， ERL 位和 BEV 位被置为 1 。
- 配置（ Config0 ）寄存器的初值为 0x80000480 。
- Config1

- Config2
- Config3
- 随机（ Random ）寄存器初始化为它的最大值。
- Wired寄存器初始化为 0 。
- ErroEPC 寄存器初始化为 PC 的值。
- Performance Counter的中断使能被清空。
- 所有等待处理的 Watch 例外和外部中断都被清除。
- PC 被置为 0xBFC0 0000

Cause 控制寄存器里的 Excode

无

入口向量

Reset（ 0xBFC0 0000 ）

服务

冷重置例外服务包括：

- 初始化所有的处理器寄存器，协处理器， Cache 和存储系统。
- 执行诊断测试。
- 引导自举操作系统。

6.6 NMI 例外

原因

NMIn为低产生 NMI 例外。该例外不可屏蔽。

处理

当发生 NMI 例外时，状态寄存器的 SR 位被置为 1，用以区分冷重置。

NMI 例外只能在指令的边界被提取。它并不抛弃任何机器的状态，而是保留处理器的状态用于诊断。 Cause 寄存器内容保持不变，而系统则跳到 NMI 例外处理程序开始处。

NMI 例外保留除下列寄存器外的所有寄存器值：

- 包含 PC 值的 ErrorEPC寄存器。
- 置为 1 的状态寄存器 ERL 位。
- 软重置或 NMI 置为 1，冷重置置为 0 的状态寄存器 SR 位。
- 置为 1 的状态寄存器 BEV 位。
- PC 寄存器重置为 0xBFC0 0000

Cause 控制寄存器里的 Excode

无

入口向量

Reset（ 0xBFC0 0000 ）

服务

NMI 例外可以用于除“重置处理器，同时保持 Cache 和内存内容”之外的情形。例如，当检测到电源故障时，系统可以通过 NMI 例外立即、可控地关闭系统。

由于 NMI 例外在另外一个错误例外中发生，因此从例外返回后，通常不太可能继续执行程序。

6.7 地址错误例外

原因

当执行以下情况时，会发生地址错误例外：

- 引用非法地址空间。
- 在用户模式下引用超级用户地址空间。
- 在用户或超级用户模式下引用内核地址空间。
- 取 (Load) 或存 (Store) 一个双字，但双字不对齐于双字边界。
- 取 (Load， Fetch) 或存 (Store) 一个字，但字不对齐于字的边界。
- 取或存一个半字，但半字不对齐于半字的边界。
- 从一个不可以执行的地址上取指
- 取指的地址不在字边界对齐

该例外不可屏蔽。

Cause 控制寄存器里的 Excode

AdEL： load 操作或者取址

AdES： store 操作

入口向量

基址：

BEV = 1 : BFC00200

BEV = 0 : Ebase31:12||0x00

偏移地址 ：

0x180

处理

共用例外向量用于地址错误例外。 Cause 寄存器的 ExcCode字段值被设为 ADEL或 ADES编码值，连同 EPC 寄存器和 Cause 寄存器的 BD 位一起，指明引起例外的指令以及例外的起因是指令引用、取操作指令还是存操作指令。

例外发生时， BadVAddr 寄存器保存了没有正确对齐的虚地址，或者受保护的地址空间的虚地址。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则， EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1 。

服务

此时，正在运行的导致例外发生的进程会收到 UNIX SIGSEGV(段违例) 信号，这个错误对该进程来说通常是致命的。

6.8 TLB 例外

可能发生三种 TLB 例外：

- 当 TLB 中没有项与欲引用的映射地址空间的地址匹配时，会导致 TLB 重填例外。
- 当虚地址引用与 TLB 中某一项匹配，但该项被标示为无效时， TLB

无效例外发生。

● 当写内存操作的虚地址引用与 TLB 中某项匹配，但该项并没有被标示为“脏”时（表示该项不可写）， TLB 修改例外发生。

下面三节将介绍这些 TLB 例外。

注： TLB 重填向量选择已经在本章前面作了介绍，具体章节见6.9“ TLB 重填例外”。

6.9 TLB 重填例外

原因

当 TLB 中没有项匹配映射地址空间的引用地址 ,且控制寄存器status的 EXL 位为 0 时， TLB 重填例外发生，该例外是不可屏蔽的。

Cause 控制寄存器里的 Excode

TLBL： load 操作或者取址

TLBS： store 操作

入口向量

基址：

BEV = 1 : BFC00200

BEV = 0 : Ebase31:12||0x00

偏移地址 :

EXL = 1 : 0x180

EXL = 0 ： 0x200

处理

当状态寄存器中的 EXL 位被设为 0时，所有的地址引用使用这些例外向量。这个例外设置 Cause寄存器中 ExcCode字段的值为 TLBL或 TLBS编码。这个编码与 EPC 寄存器以及 Cause 寄存器的 BD一起，指明引起例外的指令以及例外的起因是指令引用、取操作指令还是存操作指令。

发生这个例外时， BadVAddr 、 Context和 EntryHi寄存器保存了那条地址转换失败的虚地址。 EntryHi寄存器也保存了转换失败时的 ASID。 Random寄存器通常保存了用于放置被替换 TLB 项的合法位置。 EntryLo 寄存器的内容是不确定的。如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了导致例外的指令的地址；否则， EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD位被置为 1。

服务

为了服务这个例外， Contex寄存器的内容被作为虚地址以取得某些内存位置，这些位置包含了一对 TLB 项的物理页地址和访问控制位。这一对 TLB 项被放入了 EntryLo0/EntryLo1寄存器； EntryHi和 EntryLo 寄存器被写入 TLB 。

用于获得物理地址和访问控制信息的虚地址有可能位于一个没有驻留在 TLB 中的页面上。如果出现这种情况，则在 TLB 重填处理程序允许另外一个 TLB 重填例外来解决。由于 Status 寄存器的 EXL 位被置为 1，第二个 TLB 重填例外被传递的是共用例外向量。

6.10 TLB 无效例外

原因

当一个虚地址引用匹配到一项被标记为无效的 TLB 项 (TLB 有效位被清掉) 时， TLB 无效例外发生。这个例外是不可屏蔽的。

这里需要注意的是，因为 TLB 无效例外和 TLB 重填例外在 cause寄存器里的例外编码是一样的，所以这两个例外难以区分。唯一区分的方法就是用 TLBP，看是否在 TLB 里找到匹配的表项。

Cause 控制寄存器里的 Excode

TLBL： load 操作或者取址

TLBS： store 操作

入口向量

基址：

BEV = 1 : BFC00200

BEV = 0 : Ebase31:12||0x00

偏移地址 :

0x180

处理

共用例外向量用于处理这个例外。 Cause 寄存器的 ExcCode字段值被设为 TLBL或 TLBS，连同 EPC 寄存器和 Cause 寄存器的 BD 位一起，指明引起例外的指令以及例外的起因是指令引用、取操作指令还是存操作指令。

发生这个例外时， BadVAddr 、 Context和 EntryHi寄存器保存了那

条地址转换失败的虚地址。EntryHi寄存器也保存了转换失败时的ASID。 Random寄存器通常保存了用于放置被替换 TLB 项的合法位置。EntryLo 寄存器的内容是不确定的。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则， EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

当发生下面情况之一时， TLB 项被标记为无效：

- 虚地址不存在
- 虚地址存在，但是不在主存中（缺页 ）
- 引用这个页而引发一个陷阱（例如 ，维护引用位 ）

在服务完 TLB 无效例外的起因之后，通过 TLBP指令来定位 TLB 项（探测 TLB 来找匹配的项），然后用标记位有效的一项来替换该 TLB 项。

6.11 TLB 修改例外

原因

当写内存操作的虚地址引用与 TLB 中某项匹配，但该项并没有被标示为“脏”，因此该项不可写时， TLB 修改例外发生。该例外不可屏蔽。

Cause 控制寄存器里的 Excode

Mod

入口向量

基址：

BEV = 1 : BFC00200

BEV = 0 : Ebase31:12||0x00

偏移地址 ：

0x180

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器中的 ExcCode 字段值被设置为 MOD 。

发生这个例外时， BadVAddr 、 Context和 EntryHi寄存器保存了那条地址转换失败的虚地址。 EntryHi寄存器也保存了转换失败时的 ASID 。 EntryLo 寄存器的内容是不确定的。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则， EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1 。

服务

内核使用失败的虚地址或虚页号来识别相应的访问控制信息。被识别的页可能允许或者不允许写访问；如果写访问不允许，那么写保护违例发生。

如果写访问是允许的，那么内核在其自己的数据结构内将页标记为可写。TLBP指令把必须改变的 TLB 项的索引放入到Index寄存器中。包含物理页和访问控制位 (D 位被设置) 的一个字被取出放入 EntryLo 寄存器中，然后，EntryHi和 EntryLo 寄存器被写入 TLB 中。

6.12 总线错误例外

原因

当处理器进行数据块的读取、更新或双字 / 单字 / 半字的读请求时收到外部的 ERR 完成应答信号，或者处理器双字 / 单字 / 半字的读请求时收到外部的 ACK 完成应答信号，并且与之相关联的双字 / 单字 / 半字数据应答包含了不可改正的错误，总线错误例外发生。该例外不可屏蔽。

Cause 控制寄存器里的 Excode

IBE：取指发生总线错

DBE ：访存发生总线错

入口向量

基址：

BEV = 1 : BFC00200

BEV = 0 : Ebase31:12||0x00

偏移地址 ：

0x180

处理

共用中断向量用于处理总线错误例外。 Cause 寄存器的 ExcCode字段值被设为 IBE或 DBE ，连同 EPC 寄存器和 Cause 寄存器的 BD 位一起，指明引起例外的指令以及例外的起因是指令引用、取操作指令还是存操作指令。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器

保存了该指令的地址；否则， EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

发生错误的物理地址可以通过 CP0 寄存器中的信息计算出来。

如果 Cause 寄存器中的 ExcCode字段值被设置为IBE编码 （表示是取指引用 ），那么导致例外发生的指令虚地址保存在 EPC 寄存器中（如果 Cause 寄存器的 BD 位被置为 1，则该指令的虚地址为 EPC 寄存器内容加 4）。

如果 Cause 寄存器中的 ExcCode字段值被设置为 DBE 编码 （表示是读取或存储引用 ），那么导致例外发生的指令虚地址保存在 EPC 寄存器中（如果 Cause 寄存器的 BD 位被置为 1，则该指令的虚地址为 EPC 寄存器内容加 4）。

于是，读取和存储引用的虚地址就可以通过解释这条指令来获得。而物理地址可以通过TLBP指令以及读取 EntryLo 寄存器内容计算物理页号来获得。导致例外发生的正在运行的进程会收到 UNIX SIGBUS(总线错误) 信号，对该进程来说这通常是致命的。

6.13 整型溢出例外

原因

当一条 ADD、 ADDI、 SUB、 DADD、 DADDI或 DSUB 指令执行，导致结果的补码溢出时，整型溢出例外发生。这个例外是不可屏蔽的。

Cause 控制寄存器里的 Excode

Ov

入口向量

基址：

BEV = 1 : BFC00200

BEV = 0 : Ebase31:12||0x00

偏移地址 :

0x180

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode字段被置为 OV 编码值。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则， EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1 。

服务

导致例外发生的正在执行的进程会收到一个 UNIX SIGFPE/FPE_INTOVE_TRAP(浮点例外 / 整型溢出) 信号。对该进程来说，这个错误通常是致命的。

6.14 陷阱例外

原因

当

TGE、 TGUE、 TLT、 TLTU、 TEQ、 TNE、 TGEI、 TGEUI、 TLTI、 TLTUI、 TEQI、 TNEI指令执行，条件结果为真时，陷阱例外发生。这个例外是不可屏蔽的。

Cause控制寄存器里的 Excode

Tr

入口向量

基址：

BEV = 1 : BFC00200

BEV = 0 : Ebase31:12||0x00

偏移地址 :

0x180

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode字段被置为 TR 编码值。

如果引发例外的指令不是位于分支延迟槽内的指令，那么 EPC 寄存器保存了该指令的地址；否则， EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务

导致例外发生的正在执行的进程会收到一个 UNIX SIGFPE/FPE_INTOVE_TRAR（浮点例外 / 整型溢出）信号。对该进程来说，这个错误通常是致命的。

6.15 系统调用例外

原因

当执行 SYSCALL指令的时候，系统调用例外发生。这个例外是不可屏蔽的。

Cause 控制寄存器里的 Excode

Sys

入口向量

基址：

BEV = 1 : BFC00200

BEV = 0 : Ebase31:12||0x00

偏移地址 :

0x180

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode字段被置为 SYS编码值。

如果 SYSCALL指令没有在分支延迟槽中，则 EPC 寄存器保存这条指令的地址；否则，保存之前的分支指令的地址。

如果 SYSCALL指令在分延迟槽中，则状态寄存器中的 BD 位被置为 1，否则该位被清 0 。

服务

当这个例外发生时，控制权被转到适当的系统例程。进一步的系统调用区分可以分析 SYSCALL指令的 Code 字段（位 25 ： 6），以及载入 EPC 寄存器中所存地址的指令的内容。

为了恢复进程的执行，必须改变 EPC 寄存器的内容，这样 SYSCALL 指令才不会再次被执行；这可以通过在返回之前使 EPC 寄存器的值加 4 来完成。

如果 SYSCALL指令处在分支延迟槽中，则需要更复杂的算法，这已经超出了本节所能描述的范围。

6.16 断点例外

原因

当执行一条 BREAK指令时，发生断点例外。这个例外是不可屏蔽的。

Cause 控制寄存器里的 Excode

Bp

入口向量

基址：

BEV = 1 : BFC00200

BEV = 0 : Ebase31:12||0x00

偏移地址 ：

0x180

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode字

段被置为 BP 编码值。

如果 BREAK指令没有在分支延迟槽中，则 EPC 寄存器保存这条指令的地址；否则，保存之前的分支指令的地址。

如果 BREAK指令在分延迟槽中，则状态寄存器中的 BD 位被置为 1，否则该位清 0。

服务

当这个例外发生时，控制权被转到适当的系统例程。进一步的区分可以分析 BREAK指令的 Code 字段（位 25： 6），以及载入 EPC 寄存器中所存地址的指令的内容。如果这条指令在分支延迟槽中，那么 EPC 寄存器中的内容必须加上 4 以定位到该指令。

为了恢复进程的执行，必须改变 EPC 寄存器的内容，这样 BREAK指令才不会再次被执行；这可以通过在返回之前使 EPC 寄存器的值加 4 来完成。

如果 BREAK指令在分支延迟槽中，那么为了恢复进程的继续执行，需要解释这条分支指令。

6.17 保留指令例外

原因

当以下条件之一成立时，保留指令例外发生：

- 试图执行一条主操作码 (位 31:26) 没有定义的指令。
- 试图执行一条次操作码 (位 5:0)没有定义的 SPECIAL 指令。
- 试图执行一条次操作码 (位 20:16) 没有定义的 REGIMM 指令。

这个例外是不可屏蔽的。

Cause 控制寄存器里的 Excode

RI

入口向量

基址：

BEV = 1 : BFC00200

BEV = 0 : Ebase31:12||0x00

偏移地址 :

0x180

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode字段被置为 RI 编码值。

如果保留指令指令没有在分支延迟槽中，则 EPC 寄存器保存这条指令的地址；否则，保存之前的分支指令地址。

服务

此时， MIPS32 ISA中没有指令被解释执行。正在执行的导致例外发生的进程会收到 UNIX SIGILL/ILL_RESOP_FAULT(非法指令 / 保留的操作错误) 信号。对该进程来说，这个错误通常是致命的。

6.18 协处理器不可用例外

原因

试图执行以下任意一条协处理器指令，将会导致协处理器不可用例外发

生：

● 相应的协处理器单元（ CP1 或 CP2 ）没有被标记为可用，并且进程执行相应的协处理器指令。

● CP0 单元没有被标记为可用，并且进程执行在用户或者超级用户的模式下的 CP0 指令。

这个例外是不可屏蔽的。

Cause 控制寄存器里的 Excode

CpU

入口向量

基址：

BEV = 1 : BFC00200

BEV = 0 : Ebase31:12||0x00

偏移地址 ：

0x180

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode字段被置为 CPU 编码值。 Cause 寄存器的 CE 域指示四个协处理器的哪个被引用。如果这条指令不是在分支延迟槽中， EPC 寄存器保存了不可使用协处理器指令的地址；否则， EPC 寄存器保存了之前的分支指令的地址。

服务

有以下的几种情形：

如果进程被授权访问协处理器，协处理器被标记为可用，那么相应的用户状态被恢复以便协处理器执行。

如果进程被授权访问协处理器，但是协处理器不存在或者有故障，则需要解释 / 模拟这条协处理器指令。

如果在 Cause 寄存器中的 BD 位被设置了，分支指令必须被解释；然后协处理器指令被模拟。例外返回时跳过例外的协处理器指令继续执行。

如果进程没有被授权访问协处理器，这时执行的进程收到 UNIX SIGILL/ ILL_PRIVIN_FAULT(非法指令 / 特权指令错误) 信号。这个错误通常是致命的。

6.19 浮点例外

原因

浮点协处理器使用浮点例外。这个例外是不可屏蔽的。

Cause 控制寄存器里的 Excode

FPE

入口向量

基址：

BEV = 1 : BFC00200

BEV = 0 : Ebase31:12||0x00

偏移地址 :

0x180

处理

共用例外向量用于处理这个例外，并且 Cause 寄存器的 ExcCode字段被置为 FPE 编码值。

浮点控制 / 状态寄存器的内容指示这个例外产生的原因。

服务

清除浮点 / 状态寄存器中的适当位可以清除这个例外。

6.20 中断例外

原因

当八个中断条件中的一个触发，中断例外发生。这些中断的重要性依赖于特定的系统实现。

通过清掉在状态寄存器中的 Interrupt-Mask (IM)域中的相应的位，八个中断中的任何一个都可以被屏蔽，并且，通过清掉状态寄存器的 IE 位，可以一次屏蔽所有的八个中断。

Cause 控制寄存器里的 Excode

Int

入口向量

基址：

BEV = 1 : BFC00200

BEV = 0 : Ebase31:12||0x00

偏移地址 :

0x180

处理

共用例外向量用于处理该例外，并且 Cause 寄存器的 ExcCode字段被置为 INT编码值。

Cause 寄存器中的 IP 域指明了当前的中断请求。不止一个的中断位可能同时被设置（如果中断触发并且在寄存器被读到之前被撤消，甚至没有位被设置 ），但仅具有最高优先级的中断被响应。

IP[7] 中断位既被配置为第六个硬件中断，也在 Count 寄存器内容与 Compare寄存器内容相等时被配置为内部中断； IP[6] 中断位既被配置为第五个外部中断，也被配置为 CP0 性能计数器溢出中断。

软件需要对每一个可能的中断源进行查询来判断中断产生的原因（一个中断同时可能有多个源）。

服务

如果中断是由两个软件产生例外之一导致的，则设置 Cause 寄存器中的相应位， IP[1 ： 0]，为 0来清除中断条件。

软件中断是非精确的。一旦软件中断触发，在例外被处理之前，程序还可能继续执行几条指令。定时器中断的清除通过向 Compare 寄存器写入值来完成。性能计数器中断的清除则是向计数器的溢出位，即位 31 ，写入 0来实现。

冷重置和软重置会清除所有未完成的外部中断请求， IP[2]至 IP[6] 。

如果中断是硬件产生的，那么撤消引起触发的中断管脚的条件，就可清除中断

条件。

6.21 EJTAG 调试例外

原因

当 EJTAG 相关的条件满足时，发生 EJTAG 调试例外，详细细节请参照第八章所述

入口向量

基址：

ProbTrap = 0 ： BFC0 0480

ProbTrap = 1 ： FF20 0200

7 特权指令

表 7–50 列出了 LS232处理器定义的 CP0 指令。

表 7–50 CP0 指令

指令	描述
CACHE	CACHE 操作
ERET	例外返回
MFC0	从 CP0 取数据
MTC0	将数据送到 CP0
TLBP	查询 TLB 项
TLBR	用索引读 TLB 表项
TLBWI	用索引填充 TLB 表项
TLBWR	随机填充 TLB 表项

7.1 CP0 传输指令

LS232处理器实现的 CP0 传输指令包括MTC0、 MFC0、 DMTC0和DMFC0。表 7–51 列出了 32/64 位 CP0 寄存器的 CP0 传输指令操作。

表 7–51 CP0 传输指令

指令	CP0 寄存器位数	操作
MFC0 rt, rd，sel	32	rt <– $rd_{31..0}$
MTC0 rt, rd, sel	32	rd <– $rt_{31..0}$

7.1.1 MFC0 指令

从 CP0 控制寄存器取数据指令

31 26	25 21	20 16	15 11	10 3	2 0
COP0 010000	MF 00000	rt	rd	000000000	Sel
6	5	5	5	8	3

指令格式

MFC0 rt, rd

MFC0 rt, rd, sel

- **指令描述**

将由 rd 与 sel 域制定的 CP0 寄存器的内容取进通用寄存器 rt。注意这里，若指令中未指定特定的 sel 域，那么默认为 0 。

- **指令操作**

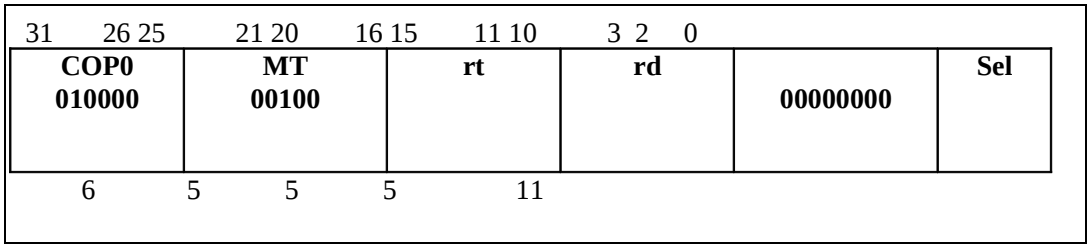
GPR[rt] <– CPR[0， rd ， sel]

- **引起的例外**

协处理器不可用例外 ；保留指令例外

7.1.2 MTC0 指令

将数据送到 CP0 控制寄存器指令



- **指令格式**

MTC0 rt, rd

MTC0 rt, rd, sel

- **指令描述**

将通用寄存器 rt 的内容送入由 rd 和 sel 指定的 CP0 寄存器中。注意这里，若指令中未指定特定的 sel 域，那么默认为 0 。

- **指令操作**

CPR[0 ， rd ， sel] <– GPR[rt]

- **引起的例外**

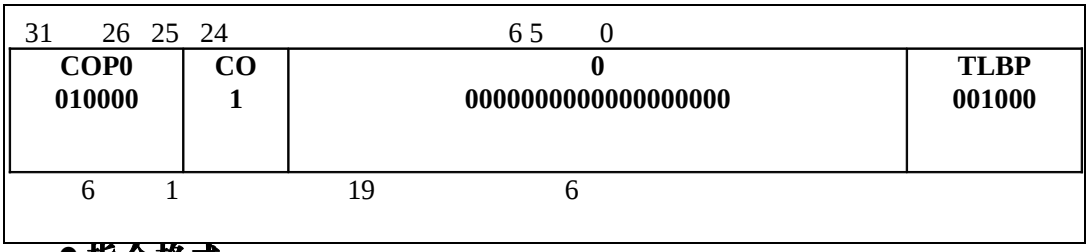
协处理器不可用例外；保留指令例外

7.2 TLB 控制指令

LS232处理器实现的 TLB 控制指令包括 TLBP、 TLBI 、 TLBWI 和 TLBWR。

7.2.1 TLBP 指令

查询 TLB 表项



- **指令格式**

TLBP

- **指令描述**

将内容与 EntryHi寄存器内容相同的 TLB 表项的地址送入Index寄存器。如果没有 TLB 表项与 EntryHi寄存器的内容相同， Index寄存器的最高位置为 1 。

- **指令操作**

Index<–1||0^{25}||undefined^{6}

for I in 0..TLBEntries–1

if(TLB[i]$_{167..141}$and not(0^{15}||TLB[i]$_{216..205}$))

=(EntryHi$_{39..13}$ and not(0^{15}||TLB[i]$_{216..205}$)) and

(TLB[i]$_{140}$ or (TLB[i]$_{135..128}$=EntryHi$_{7..0}$)) then

Index<=0^{26}||i$_{5..0}$

endif

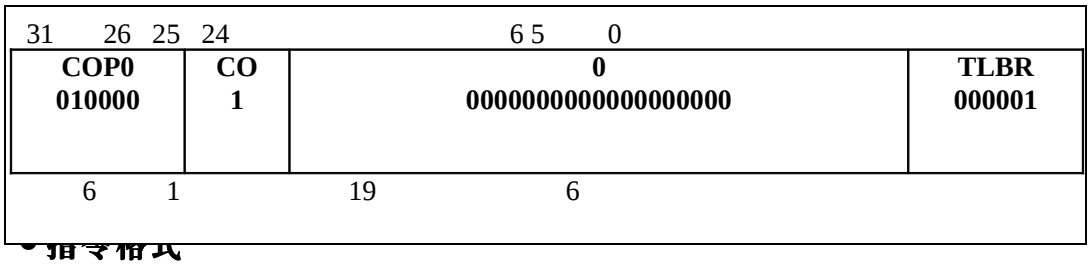
endfor

- **引起的例外**

协处理器不可用例外

7.2.2 TLBR 指令

按索引读 TLB 表项



- **指令格式**

TLBR

- **指令描述**

将从 TLB 中读出的 G位（控制 ASID匹配）写入 EntryLo0和 EntryLo1寄存器。将 Index寄存器索引的 TLB 表项的内容送入EntryHi 和 EntryLo 寄存器。如果 Index寄存器的值大于处理器中的 TLB 表项数，则该操作无效。

- **指令操作**

PageMask<–TLB[Index$_{5..0}$]$_{223..192}$

EntryHi<– TLB[Index$_{5..0}$]$_{191..128}$ and not TLB[Index$_{5..0}$]$_{255..192}$

EntryLo1<– TLB[Index$_{5..0}$]$_{127..65}$|| TLB[Index$_{5..0}$]$_{140}$

EntryLo0<– TLB[$Index_{5..0}$]$_{63..1}$|| TLB[$Index_{5..0}$]$_{140}$

- **引起的例外**

协处理器不可用例外

7.2.3 TLBWI 指令

按索引填充 TLB 表项

31 26	25 24	6 5 0	
COP0 **010000**	**CO** **1**	**0** **00000000000000000000**	**TLBWI** **000010**
6	1	19	6

- **指令格式**

TLBWI

- **指令描述**

用 EntryLo0和 EntryLo1寄存器的 G位相与的值设置 TLB 的 G 位。用 EntryH和 EntryLo 寄存器的内容设置 Index索引的 TLB 表项。如果 TLB Index寄存器的值大于处理器中的 TLB 表项数，则该操作无效。

- **指令操作**

TLB[Index PageMask||(EntryHi and not PageMask)

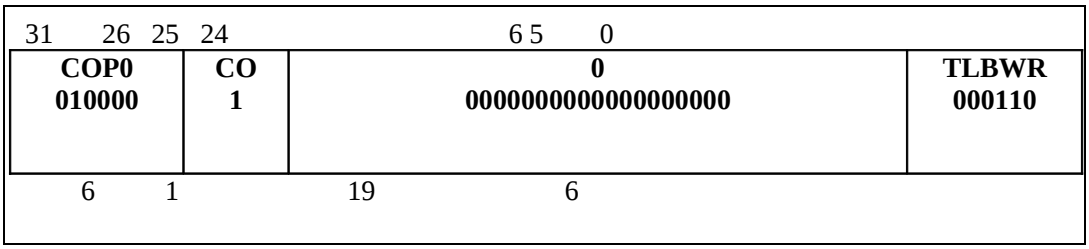
EntryLo0

- **引起的例外**

协处理器不可用例外

7.2.4 TLBWR 指令

随机填充 TLB 表项



- **指令格式**

TLBWR

- **指令描述**

用 EntryLo0和 EntryLo1寄存器的 G位相与的值设置 TLB 的 G 位。用 EntryHi和 EntryLo 寄存器的内容设置 TLB Random寄存器索引的 TLB 表项。

- **指令操作**

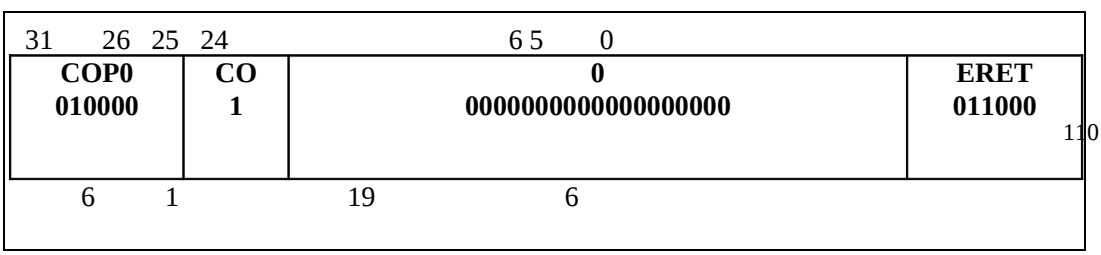
TLB[Random$_{5..0}$]<–PageMask||(EntryHi and not PageMask)||EntryLo1||EntryLo0

- **引起的例外**

协处理器不可用例外

7.3 ERET 指令

例外返回指令



- **指令格式**

ERET

- **指令描述**

ERET指令用来从中断、例外和错误陷入返回。与 Branch和 Jump 指令不同，ERET指令没有延迟槽。 ERET不能作为延迟槽指令。如果处理器在为错误陷入服务（ SR_2=1），则从 Error EPC 中取 PC 值，并且清 Status 寄存器（ SR_2 ）的 ERL 位。否则（ SR_2=0），从 EPC 寄存器中取 PC 值，并且清 Status寄存器（ SR_1 ）的 EXL 位。如果ERET指令在 LL 和 SC 指令之间执行，会导致 SC 失败。如果没有例外（ Status 寄存器的 EXL=0 并且 ERL=0 ），将 EXL 置为 0，并跳到 EPC 寄存器保存的地址处。

- **指令操作**

If SR_2=1 then

PC<–ErrorEPC

$SR<–SR_{31..3}||0||SR_{1..0}$

else

PC<–EPC

$SR<–SR_{31..2}||0||SR_0$

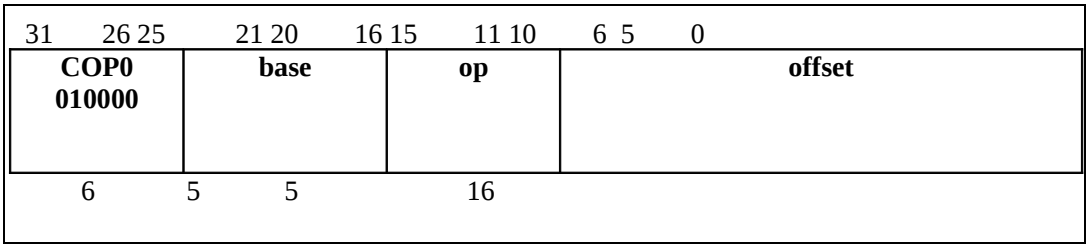
Endif

LLbit<–0

- **引起的例外**

协处理器不可用例外

7.4 CACHE指令



- **指令描述**

用符号扩展的 16 位 offset加上通用 base 寄存器的值形成 Cache 操作的虚地址（ VA ）。虚地址（ VA ）通过 TLB 转换成物理地址（ PA ）， 5位的操作码（如表 8–3所示）指定对这个地址的 Cache 操作。除下表以外的其他 Cache 指令未定义，若执行了那些未定义的指令，将会报保留指令里外。对 Uncached 地址空间的 Cache 指令操作则不会产生执行效果。

表 7–2 CACHE 指令

Op 域	描述	目标 Cache
00000	Index Invalidate	(I)
00001	Index WriteBack Invalidate	(D)
00101	Index Load Tag	(D)
01000	Index Store Tag	(I)
01001	Index Store Tag	(D)
10000	Hit invalidate	(I)
10001	Hit Invalidate	(D)
10101	Hit WriteBack	(D)

	Invalidate	
11100	Fetch and Lock	(I)
11101	Fetch and Lock	(D)

- **指令操作：**

vAddr<–((offset$_{15}$)^{48}||offset$_{15..0}$)+GPR[base]

(pAddr)<–AddressTranslation(vAddr)

CacheOp(op,vAddr,pAddr)

- **引起的例外**

TLB Refill 例外；

TLB Invalid例外；

协处理器不可用例外 ；

地址错例外；

Cache Error 例外；

Bus Error例外；

Index Invalidate (I)

Index Invalidate（ I ）指令将指令 Cache 中的对应行的某一路的 Cache 块置为 Invalid状态。其中 VA[11:5]定义行地址，VA[13: 12]定义组号。无效即将指令 Cache 状态位置为 0（ Invalid ）。

Index WriteBack Invalidate (D)

Index WriteBack Invalidate（ D ）指令将数据 Cache 中的对应块置为 Invalid 状态。 VA[11:5] 定义地址， VA[13:12]定义无效的组号。无效即数据 Cache 状态位置为 00 （ Invalid ）。如果数据 Cache 对应块的数据是脏的，则将数据写入内存。

Index Load Tag (D)

Index Load Tag(D)将数据 Cache的 Tag域写入 CP0的 TagLo 寄存器。VA[11:5] 定义地址， VA[13 ： 12] 定义读 Tag的组号。

操作的映射定义如下：

TagLo[7:6] = State bits

TagLo[27:8] = Tag[19 ： 0]

TagLo[28] = Lock bits

所有其他 CP0 TagLo 寄存器位置为 0 。

Index Store Tag (I)

Index Store Tag(I)将 CP0 的 TagLo 寄存器的值存入数据 Cache 的 Tag域。 VA[13:5] 定义地址， VA[1:0] 定义读 Tag的组号。

操作的映射定义如下：

Tag[31] = TagLo[7]&TagLo[6]

Tag[30] = TagLo[28]

Tag[29 ： 20] = 10’b0

Tag[19:0] = TagLo[27 ： 8]

Index Store Tag (D)

Index Store Tag(D)将 CP0 的 TagLo 寄存器的值存入数据 Cache 的 Tag域。 VA[13:5] 定义地址， VA[1:0] 定义读 Tag的组号。

操作的映射定义如下：

Tag[21] = TagLo[7]&TagLo[6]

Tag[20] = TagLo[28]

Tag[19:0] = TagLo[27 ： 8]

Hit Invalidate (I)

Hit Invalidate (I)无效数据 Cache 匹配地址 PA 的项。 VA[11:5]索引的所有的路都从指令 Cache 中读出。

如果 Istate的值不等于 0（ Invalid），并且 Cache 指令的 PA 与从指令 Cach中读出的 ITag匹配，将该项的 IStat置e为 0（ Invalid）。

Hit Invalidate (D)

Hit Invalidate (D)无效数据 Cache 匹配地址 PA 的项。 VA[11:5]索引的所有的路都从指令 Cache 中读出。

如果 Dstate的值不等于 0（ Invalid），并且 Cache 指令的 PA 与从指令 Cach中读出的 DTag匹配，将该项的 DStat置为 0（ Invalid）。

Hit WriteBack Invalidate (D)

Hit WriteBack Invalidate（ D）指令将数据 Cache 中与 PA 地址相匹配的块置为 Invalid 状态。数据 Cache 中的四路都用 VA[11:5] 做为索引读出。如果 DState不等于 0（ Invalid），并且 DTag与 Cache 指令的 PA 相匹配，则将数据 Cache 的该项状态位置为 0（ Invalid）。如果数据 Cache 对应块的数据是脏的，则将数据写入内存。

Fetch and Lock(I)

Fetch and Lock（ I）指令将指令 cache中与 PA 地址相匹配的 cache 行的 lock 位置为 1；若指令 cache中没有 PA 指定的那个 cache 行，那么从内存中取得这个 cache 行写入指令 cache中，并置其 lock 位与 valid

位都为 1。

Fetch and Lock(D)

Fetch and Lock(D)指令将数据 cache中与 PA 地址相匹配的 cache 行的 lock 位置为 1；若数据 cache中没有 PA 指定的那个 cache 行，那么从内存中取得这个 cache 行写入数据 cache中，并置其 lock 位与 valid 位都为 1。

7.5 CP0 指令相关的处理

LS232处理器能够处理硬件中的流水线相关，包括 CP0 相关和访存相关，因此 CP0 指令并不需要 NOP 指令来校正指令序列。

8 性能优化与实时中断

性能计数器

LS23定义了两对用来做性能分析的性能计数器（Performance Counter）。每对性能计数器都包含一个控制寄存器与计数器，其中性能控制寄存器用来指定性能计数器在何时可用以及指定与其对应的计数器计数的事件，性能计数器则在其对应的控制寄存器中指令事件发生时，开始计数；具体格式请参考第六章的相应章节。因为我们只有两个计数器，因此在同一时刻我们只能对两个事件计数，这两个事件分别有各自对应的控制寄存器指定。

LS232为每个性能计数器定义了 32 个事件，总共 64 个事件。通过对这些事件的计数，我们可以知道这些事件在程序执行过程中发生的频率，以此作为我们分析LS232处理器性能的依据。下面我们将具体对这些事件分析，以期能够指导对LS232的性能分析。

基本事件

表 8-52 列出了 LS232的性能技术器定义的基本事件，包括 cpu的时钟以及提交的指令总数。从这两个事件我们可以得到程序执行的IPC。

这里我们需要注意的是，我们在两个计数器里分别定义了一个 clock。之所以重复定义了两个时钟，是为了能够对每个counter中定义的事件，都能得出相对于时钟的效果，就像计算 IPC一样。

表 8-52 基本事件

事件	内部信号	描述	所在的性能计数器
0x0	Clock	Cpu 的时钟数	Counter0
0x0	Pc_cmmit	提交的指令总数	Counter1
0x2	Clock	Cpu 的时钟数	Counter1

8.1.1 转移指令相关事件

针对对转移指令预测效率的统计，性能计数器规定了以下几个，如表8-53 所示：

表 8-53 分支指令相关事件

事件	内部信号	描述	所在的性能计数器
0x1	Brbus_valid	执行过的分支总数	Counter0
0x1	Brbus_brerr	分支预测错总数	Counter1
0x3	Brbus_jr31	jr31 指令的总数	Counter0
0x3	Brbus_jr31mis	jr31 预测错总数	Counter1
0x8	Brbus_bht	用 bht 做分支预测的分支总数	Counter0
0x8	Brbus_bhtmiss	用 bht 做分支预测的分支预测错的总数	Counter1
0x12	Brbus_static	静态预测的分支总数	Counter0
0x12	Brbus_static_miss	Static 预测分支错误的次数	Counter1

对以上事件的统计，我们可以得出各类分支预测，以及所有分支预测的失效率。比如对同时统计两个counter的第 0x1号事件，我们可以得到所有执行过的分支指令的总数，以及其中分支预测错误的指令数，从而我们可以得到分支预测的失效率。正如我们所了解的那样，当分支预测错时，需要将流水线清空，然后从正确的目标地址继续执行。而清空流水线相当于在流水线的每一级流水都加入一个气泡，对于LS232的五级流水而言，这就相当于在流水线中至少加入了 5 个气泡（因为对于访存，乘除法之类的操作，其真正执行起来可能需要更多拍数），从而导致流水线至少停顿五拍。所以若分支失效率越大，流水线的效率就越差，从而导致LS232的性能也就越差。

LS232在设计过程中，权衡分支预测代价以及分支预测的效果后，针对不同类型的分支指令采用了不同的分支预测策略。比如对目标地址是 31 号

寄存器的间接跳转指令（ jr31），我们采用 4 项的 RAS 来对目标地址做分支预测。在性能计数器的事件中，我们也对这类分支指令的总数以及预测失效的总数作了统计，他们分别在 counter0， counter1 的第 0x3 号事件。通过对这两个事件的统计，我们可以计算出 jr31类指令的分支预测失效率。

LS232对非 link和非 likely的条件分支指令采用了 gshare 的预测器，来预测其跳转方向。同样，这类指令的总数以及预测失效的总数也能通过性能计数器得到，他们分别在 counter0， counter1 的第 0x8 号事件。

对于 link类与 likely类的条件分支指令， LS232采用了静态预测跳转的策略，这类指令的分支预测失效率，我们可以通过统计 counter0和 counter1 的第 0x12 号事件得到。

8.1.2 取指相关的事件

表 8-54 列出了 LS232的性能技术器定义的有关取指相关的事件，大致有两类事件，一是 icache 访问相关，一类是 itlb 访问相关，具体如下表所示：

表 8-54 取指相关的事件

事件	内部信号	描述	所在的性能计数器
0x4	Imemread_valid	取指读访存的次数	Counter0
0x10	Icache_access	读 icache 的次数	Counter0
0x10	Icache_hit	Icache 命中的次数	Counter1
0x18	Itlb_access	访问小 itlb 的次数	Counter0
0x18	Itlb_miss_tlb_hit	小 itlbmiss 但是大 tlbhit 的次数	Counter1
0x13	Icache_way_hit	Icache 路预测命中的次数	Counter1
0x14	Icache_update	Icache 路预测失效更新的次数	Counter1
0x15	Insts_fetched	所有取来的指令	Counter1

针对 icache 访问相关的事件，我们有 icache_hit 与 icache_access

事件，通过对这两个事件的统计，我们可以得到 icache 的命中率。但是这里需要注意的是，LS23的指令采用了路预测结构，当路预测命中时（icache_way_hit），取指一拍就可以返回有效的指令；若路预测失效，但是 icache 命中，那么需要两拍才能返回有效的指令，这就相当于取指的延迟为两拍；若 icache 失效，那么取指延迟将会更大。所以，在统计取指的延迟时，应统一考虑 icache_hit与 icache_way_hit两个事件。至于 icache_update 事件是为了统计路预测失效对功耗的影响。因为路预测失效需要更新路预测信息，这相当于写一次 icache 。

针对 itlb访问相关的事件，我们有 itlb_access与 itlb_miss_tlb_hit 两个事件。通过对这两个事件的统计，我们可以得到 itlb的命中率。当 itlb 命中时，我们一拍就可以返回转换后的物理地址；否则，若 itlb 失效，但是大 tlb 命中，那么我们至少需要两拍才能返回物理地址，这样相当于又将取指的延迟增加了一拍，即在取指级加入了一个气泡。

至于 imemread_valid事件，统计了因为取指而访存的次数，这包括对 uncache 空间取指以及 icache miss。通过对这个事件的计数，我们可以看出取指是否频繁地访问内存，造成取指的 IPC降低，从而分析具体的原因。

最后， Insts_fetched事件统计了取来指令的总数，LS232是双发射结构，所以这个事件每拍最多可以递增 2 。通过对这个事件的计数，再结合对 clock 的计数，我们可以得出取指的IPC。

8.1.3 译码级相关的事件

表 8-55 列出了译码级相关的事件。指令译码过后都会写入操作队列，为后续的执行指令流。通过对这两个事件的统计，我们可看出在往操作队列送

指令的过程中，是否能真正达到双发射。

而结合 Inst_queue_write与teclock事件，我们可以得到译码级的IPC，从而我们看出流水线的前端是否能为后端提供足够的指令流。

表 8-55 译码级相关的事件

事件	内部信号	描述	所在的性能计数器
0xf	Inst_queue_write_cycle	往队列写指令的cycle数	Counter0
0xf	Inst_queue_write	写入队列的指令总数	Counter1

8.1.4 发射与执行相关的事件

表 8-56 列出了发射与执行相关的事件，大致分为两类事件，一是发射相关，一直各个功能单元执行的指令相关，具体如下表所示：

表 8-56 发射与执行相关的事件

事件	内部信号	描述	所在的性能计数器
0x5	Qissuebus_valid0	发射总线0有效的次数	Counter0
0x6	Qissuebus_valid1	发射总线1有效的次数	Counter0
0x19	Insts_executed_alu1	在定点功能单元1执行的指令数	Counter1
0x1a	Insts_executed_alu2	在定点功能单元2执行的指令数	Counter1
0x1b	Insts_executed_addr	在访存功能单元1执行的指令数	Counter1
0x1c	Insts_executed_falu	在浮点功能单元1执行的指令数	Counter1

通过对事件 qissuebus_valid0与 qissuebus_valid两个事件的统计，我们可以观察到两条发射总线的效率，而将这两个事件与 clock事件相结合，我们可以得到发射的 IPC。

至于剩余的四个事件，分别统计了四个功能单元的执行频率，有利于我们观察到各类指令的频率。而若将他们分别与 clock结合，我们可以得到反映

各个功能单元执行效率的 IPC。

8.1.5 访存相关的事件

表 8-57 列出了访存相关的事件，这些事件大致可以分为三类，即 dcache 相关， dtlb相关以及访存接口相关，具体如下表所示：

表 8-57 访存相关的事件

事件	内部信号	描述	所在的性能计数器
0x11	Dcache_access	读 dcache 的次数	Counter0
0x11	Dcache_hit	Dcache 命中的次数	Counter1
0x17	Dtlb_access	访问小 dtlb 的次数	Counter0
0x17	Dtlb_miss	小 dtlb 失效但是大 tlbhit 的次数	Counter1
0x9	Mreadreq_valid	访存的总次数（包括指令与数据）	Counter0
0x4	Dmemread_valid	数据读访存总数	Counter1
0x1d	Data_inst_conflict	数据和指令因访存接口而产生的冲突次数	Counter0
0x1d	Data_inter_conflict	不同类型数据访问同时访存接口产生冲突	Counter1
0x1e	Not_store_ok_stall	Store_queue 因 head 项 store_ok!=1 处于满状态而产生阻塞效果的时钟数	Counter0
0x1f	St_ld_conflict	Store 与 load 在 dcache 同一 bank 上产生访问冲突	Counter0
0x7	Duncache_valid	数据 uncached 总数	Counter1
0x9	Mwrite_req_valid	访存写访存次数	Counter1
0x1e	Commit_ld_st	正常提交的 load 和 store 的指令数	Counter1

通过 Data cache 相关的两个事件： Dcache_access 与 dcache_hit，我们可以得出 dcache 的命中率，从而推出 dcache miss 对性能的影响。

针对 dtlb访问相关的事件，我们有 dtlb_access与 dtlb_miss两个事件。通过对这两个事件的统计，我们可以得到 dtlb的命中率。当dtlb命中时，我们一拍就可以返回转换后的物理地址；否则，若 dtlb 失效，但是大 tlb 命中，那么我们需要两拍才能返回物理地址，这样相当于将访存的延迟增加了一拍。如果 dtlb命中率较低，表示 8项的 DTLB不能满足该应用。

事件 Mreadreq_能够i反映程序对外部总线的压力。而 Dmemread_valid，反映数据访问对外部总线的压力。

而事件 Data_inst_conflic与 Data_inter_conflic可以在一定程度上反映了单一串行接口部件带来的冲突损失是否对程序影响严重。

对于事件 Not_store_ok_sta而言，如果其较大且次数和 Storeq_full次数接近，则反映 store之前的分支未完成或可能产生例外的指令对 store操作执行效率的影响较显著

事件 St_ld_conflic反映的问题是：当出现先 store而后 load 这样的操作序列，如果相隔一拍进入访存部件， store和 load 都在 cache中命中，但是它们落在同一 bank 上，那么后续的 load 将被阻塞一拍。如果可以，调整它们之间的间隔，可以避免这种冲突，提高程序性能。

事件 Duncache_valid，我们可以看出如果较高，反映程序中有大量的数据访问是 uncache 的。如果这些 uncache 操作中有大量连续地址的存数操作，则建议使用 uncache 加速技术对程序做改进。

通过统计事件Commit_ld_s与 clock，我们可以看出，如果它的比值较高，则反映这段程序是数据访问密集的程序。

8.1.6 指令提交相关的事件

表 8-58 列出了指令提交相关的事件，这些事件主要是对两条提交总线进行统计，从而可以观察到不同时间段，提交总线的效率。

表 8-58 指令提交相关的事件

事件	内部信号	描述	所在的性能计数器
0x19	Commitbus0_valid	提交总线 0 提交的指令数	Counter1
0x1a	Commitbus1_valid	提交总线 1 提交的指令数	Counter1
0x1b	Has_commit	有提交指令的时钟数	Counter1
0x1c	Commit_two	同时提交两条指令的时钟数	Counter1

8.1.7 流水线停顿相关的事件

表 8-59 列出了流水线停顿相关的事件 . 这些事件主要包括三类，一是对造成取指停顿的事件；一是因为访存内部的数据结构满而造成流水线停顿的事件；一是造成清空流水线的事件。具体如下表所示：

表 8-59 流水线停顿相关的事件

事件	内部信号	描述	所在的性能计数器
0xa	Stalled_fetch	没有取指的时钟数	Counter0
0xb	Queue_full	操作队列满的次数	Counter0
0xb	Brq_full	brq 满的次数	Counter1
0x16	Insts_stall_cycles	因为 icache miss 而阻赛的时钟数	Counter1
0x13	Loadq_full	Load queue 满而产生阻塞的时钟数	Counter0
0x14	Storeq_full	Store queue 满而产生阻塞的时钟数	Counter0
0x15	Missq_full	Miss queue 满而产生阻塞的时钟数	Counter0
0x16	Miss_req_queue_full	Miss request queue 满而产生阻塞的时钟数	Counter0
0xc	Flush_pipeline_cycles	因为各种原因引起的清空流水线的时	Counter0

		钟数	
0xd	Ex_tlbr	tlbr 例外的次数	Counter0
0xe	Ex_int	发生中断的次数	Counter0
0xd	Exbus_ex	发生异常的总数	Counter1

首先，对于第一类事件而言，LS232统计了取指停顿的总时钟数。分析 stalled_fetch 事件，我们可以看出前端取指的效率已经影响到流水线的效率。取指停顿可能有多个原因造成，LS232对其中最重要的三个进行统计，其中包括因为 icache miss造成的停顿，操作队列满以及分支队列满造成的停顿。当操作队列满过多时，我们可以看出前端的取指快于后端的执行，因而指令执行的瓶颈在后端。若分支队列满过多时，说明在程序执行的某个阶段的分支指令比较多。当 icache miss造成的停顿过多时，说明 icache 的容量已经达不到要求，或者这段时间 icache 内容替换过多。

LS232结构中能引起刷新流水线的事件只可能是分支预测错或发生了异常。而清空流水线相当于在流水线的每一级流水都加入一个气泡，对于 LS232 的五级流水而言，这就相当于在流水线中至少加入了 5 个气泡（因为对于访存，乘除法之类的操作，其真正执行起来可能需要更多拍数），从而导致流水线至少停顿五拍。所以一旦 flush_pipeline_cycles事件过多，那么处理器的效率会急剧下降。

分支预测错事件我们已经在前面叙述过，在这一章节就不再累述。这里主要分析一下异常事件。首先，LS232的性能计数器有能统计发生的异常总数 – Exbus_ex 事件，还有专门统计发生中断次数的事件 ex_int 以及统计 tlb 重填例外的次数的事件 –ex_tlbr. 通过观察这两个事件来分析引起异常事件过多的原因。比如若 ex_tlbr 事件过多，说明可能程序在频繁的换页或者说 32 项的 tlb 的容量或现有的页大小分配策略已经不能满足需要。

而通过观察统计访存部件的四个事件，我们也可以看出在某个事间段内访存操作是否密集，以及对流水线的影响。比如通过 Loadq_full ， Storeq_full事件能反映 load queue和 store queue的项数是否不够用， Loadq_full间接反映了 load miss率是否高； Missq_full 反映 miss queue 资源是否够用，或者反映落在不同 cacheline上的连续 miss 操作是否过多；Miss_req_queue_full 反映 miss request queue 资源是否够用。

8.2 访存性能优化建议

访存的性能对于处理器整体性能至关重要，这里介绍一些与 LS232IP 微结构相关的访存性能优化建议。

8.2.1 数据 Cache RAM 端口竞争冲突的优化

为保证 LS232IP 的高性价比特点，本处理器中的数据 Cache RAM采用单端口实现方案，因此，当一条写操作和一条读操作相隔一拍发射到访存部件执行，且它们的地址映射到同一个 Cache Bank上 ,那么虽然它们都会在 Cache 中命中，前面的写操作仍会阻塞后面的读操作一拍。软件人员在做代码的手工优化时，如果出现连续的访存读写序列，需要关注这一点。

8.2.2 写合并加速功能的优化

对于 Cache 空间的连续写操作， LS232IP 设计了写合并的优化，该优化能够显著提升连续的 Cache不命中的写操作的执行效率，诸如 memcpy 等操作。为充分利用该功能，建议软件将能够落在同一 Cache 行的写操作尽可能组织成为连续写的方式，且在该连续过程中尽量不引入两条及

以上落在其它 Cache 行的写操作。

8.2.3 UnCache 加速功能的优化

对于 UnCache空间的连续写操作，诸如显存的填充操作， LS232IP 设计了 UnCache加速优化，该优化将从 32 字节地址对齐开始的连续 32 字节写数据收集满之后再向处理器外发出写操作。该优化充分降低了连续 Uncache写操作对外部总线带宽的消耗，从而大幅提升系统性能。为充分利用该功能，建议软件将包含大量连续 Uncache 写操作的数据段所属页属性设置为 Uncache Acc，同时将尽可能将落在同一 32 字节范围内的写操作调度在一起。此外，同其它非龙芯系列 MIPS处理器的 Uncache加速机制相比，LS232IP 能收集字节和半字操作，进一步扩展了 UnCache加速优化使用的范围。同时 LS232IP 并不强制要求落在同一 32 字节范围内的写操作必须按照地址递增方式排列，落在 32 字节范围内的写操作可以以任意顺序排列，当它们收集满一行即可整行写出。该特性进一步增加了软件优化的自由度。

8.3 实时中断模式工作特性

LS232IP 为实时性要求严格的应用特别设计了实时中断模式，在该模式下，处理器在采样外部中断后，至多经过 12 个时钟周期（以处理器内部时钟计算）就可以进入中断处理程序，从而可以确保外部中断处理延迟上限的确立。在获得上述优点的同时，实时中断模式的实现进一步加强了访存操作的相关控制，从而在某些情况下会略微降低处理器的性能。因此，在软实时能够满足要求的应用场合，并不推荐使用该模式。

特别需要注意的是，在实时中断模式下，处理器不保证对外部读访问的

效果与处理器提交读操作序列所应产生的效果严格一致，因此对于外部会因为读操作而改变状态的设备或设备中的寄存器（例如， NAND Flash 的读操作、设备的读清寄存器），程序的执行结果可能会出错。因此在实时中断模式下，需确保处理器外部没有上述设备或设备中的寄存器，如果确实存在，请在操作过程中关闭外部中断，或者由软件进行正确性保证。

9 LS232与传统MIPS32 ISA的差异

总体来说，不带浮点指令的LS232与 MIPS32 ISA的 R2版本完全兼容。而 LS232的浮点指令是 R1版本兼容。以下内容从控制寄存器，定点指令与 CP0 指令，浮点指令三个方面来分析LS232与传统 MIPS32 ISA的差异。这里主要是一些 MIPS32 ISA中规定为实现相关的指令及控制寄存器在 LS232中的实现。

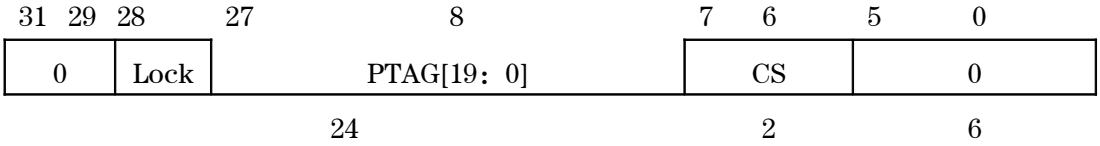
9.1 控制寄存器

控制寄存器与 MIPS32 R2版本完全兼容，但是在一些可选选项上有少许不同，具体如下：

- 所有实现的控制寄存器：所有针对 R2，标记为 Require的控制寄存器在LS232中都实现。而标记 Optipnal 的控制寄存器只实现了以下两个：
  - DEBUG：这个寄存器是在实现 EJTAG时所需要的，其格式严格按着 EJTAG标准而实现；
  - DEPC：也是在实现 EJTAG时所需要的，保存发生 EJTGAG例外的指令地址
- 所有规范中 require 的寄存器中 optional 域除了以下几个寄存中的个别域外，其它都为只读位，且读出 0：
  - 因为 LS232实现了向量中断（没有实现外部中断），所以实现了 IntCtl（ CP0 register12, select1 ）的 VS域（ 9..5 ），具体编码参照手册
- TAGLO与 TAG_HI 的格式： taglo和 taghi寄存器在实现 cache的处

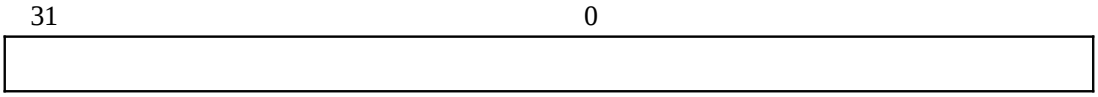
理器中是必需的，但是其格式时实现相关的，LS232中这两个寄存器的格式如下：

表 9-60 Taglo寄存器格式



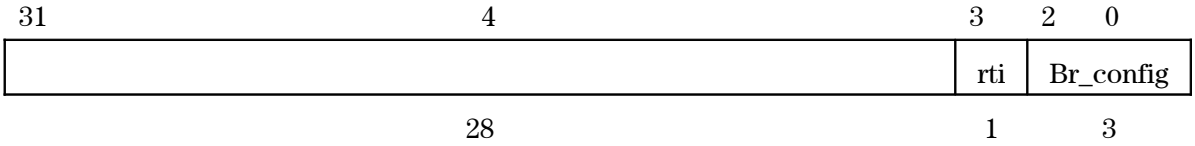
其中 lock， ptag， cs 为可读可写位。其它位保留，写忽略且读出 0

表 9-61 Taghi 寄存器格式



Taghi为 32bits的可读可写寄存器

- 实现了 config6(cp0 register16,select寄存器， config6在 mips32规范中为保留寄存器。 LS232使用这个寄存器，用来控制是否使用实时中断和分支预测策略的选择。具体格式如下：



其中 rti 与 br_config为可读可写位， rti 为 1表示使用实时中断。而br_config用来配置分支预测策略，具体编码如表 9-62 所示：

表 9-62 Config6 的分支预测配置定义

编码	意义
3’b00	Gshare 分支预测

0	
3’b001	用 pc 索引的 bht 预测
3’b010	Always taken
3’b011	Always not taken
3’b100	向前跳转 taken
3’b101	向后跳转 taken

- Entrylo 与 mips32 规范有少许不同： mips32 中，第 30 位为保留域。而 LS232利用这个 bits作为 NE 位。意为不可执行位。当这位为 1 时，表明这个物理页为不可执行页。所以当取指地址转换时，发现这位为 1，则报地址错例外。
- LS232不支持 1KB 小页。
- 性能计数器 0 与性能计数器 1 的事件定义分别如表 9-63 与表 9-64 所示：

表 9-63 计数器 0 事件

事件	内部信号	描述
0x0	Clock	时钟
0x1	Brbus_valid	执行过的分支总数
0x2	Reserved	保留
0x3	Brbus_jr31	jr31 指令的总数
0x4	Imemread_valid	取指读访存的次数
0x5	Qissuebus_valid0	发射总线 0 有效的次数
0x6	Qissuebus_valid1	发射总线 1 有效的次数
0x7	Reserved	
0x8	Brbus_bht	用 bht 做分支预测的分支总数
0x9	Mreadreq_valid	访存的总次数（包括指令与数据）
0xa	Stalled_fetch	没有取指的时钟数
0xb	Queue_full	操作队列满的次数
0xc	Flush_pipeline_cycles	因为各种原因引起的清空流水线的时钟数
0xd	Ex_tlbr	tlbr 例外的次数

事件	内部信号	描述
0xe	Ex_int	发生中断的次数
0xf	Inst_queue_write_cycle	往队列写指令的 cycle 数
0x10	Icache_access	读 icache 的次数
0x11	Dcache_access	读 dcache 的次数
0x12	Brbus_static	静态预测的分支总数
0x13	Loadq_full	Load queue 满而产生阻塞的时钟数
0x14	Storeq_full	Store queue 满而产生阻塞的时钟数
0x15	Missq_full	Miss queue 满而产生阻塞的时钟数
0x16	Miss_req_queue_full	Miss request queue 满而产生阻塞的时钟数
0x17	Dtlb_access	访问小 dtlb 的次数
0x18	Itlb_access	访问小 itlb 的次数
0x19	Insts_executed_alu1	在定点功能单元 1 执行的指令数
0x1a	Insts_executed_alu2	在定点功能单元 2 执行的指令数
0x1b	Insts_executed_addr	在访存功能单元 1 执行的指令数
0x1c	Insts_executed_falu	在浮点功能单元 1 执行的指令数
0x1d	Data_inst_conflict	数据和指令因访存接口而产生的冲突次数
0x1e	Not_store_ok_stall	Store_queue 因 head 项 store_ok!=1 处于满状态而产生阻塞效果的时钟数
0x1f	St_ld_conflict	Store 与 load 在 dcache 同一 bank 上产生访问冲突

表 9-64 计数器 1 的事件

事件	内部信号	描述
0x0	Pc_cmmit	提交的指令总数
0x1	Brbus_brerr	分支预测错总数
0x2	clock	时钟
0x3	Brbus_jr31mis	jr31 预测错总数
0x4	Dmemread_valid	数据读访存总数
0x5	reserved	
0x6	reserved	
0x7	Duncache_valid	数据 uncached 总数
0x8	Brbus_bhtmiss	用 bht 做分支预测的分支预测错的总数
0x9	Mwrite_req_valid	访存写访存次数
0xa		
0xb	Brq_full	brq 满的次数
0xc		
0xd	Exbus_ex	发生异常的总数
0xe		
0xf	Inst_queue_write	写入队列的指令总数
0x10	Icache_hit	Icache 命中的次数

事件	内部信号	描述
0x11	Dcache_hit	Dcache 命中的次数
0x12	Brbus_static_miss	Static 预测分支错误的次数
0x13	Icache_way_hit	Icache 路预测命中的次数
0x14	Icache_update	Icache 路预测失效更新的次数
0x15	Insts_fetched	所有取来的指令
0x16	Insts_stall_cycles	因为 icache miss 而阻赛的时钟数
0x17	Dtlb_miss	小 dtlb 失效但是大 tlbhit 的次数
0x18	Itlb_miss_tlb_hit	小 itlbmiss 但是大 tlbhit 的次数
0x19	Commitbus0_valid	提交总线 0 提交的指令数
0x1a	Commitbus1_valid	提交总线 1 提交的指令数
0x1b	Has_commit	有提交指令的时钟数
0x1c	Commit_two	同时提交两条指令的时钟数
0x1d		
0x1e	Data_inter_conflict	不同类型数据访问固访存接口产生冲突
0x1f	Commit_ld_st	正常提交的 load 和 store 的指令数

9.2 定点与 CP0 指令

LS232处理器的定点指令也完全与 MIPS32 R2 版本完全兼容，但是有些指令在实现上有着与 LS232结构相关的特点，具体如下：

- Wait 指令的实现：
  - 用户态的 wait 指令不产生作用。 Wait 指令在分支延迟槽时也不产生效果
  - Wait 指令使流水线停止，当发生中断等外部事件时，流水线重新开始执行。
- EHB， JR.HB,JALR.HB， SSNOP:

因为 LS232的结构本身就能保证不产生 cp0 hazards，所以这些指令就相当于普通指令。其中 EHB为普通的 SLL操作，执行延迟为 1cycle， JR.HB, JALR.HB分别对应于 JR， JALR，执行延迟也为 1cycle 。同样 SSNOP也是 SLL操作，执行延迟为 1cycle 。

- NOP 指令： nop指令并不真正送入功能单元执行，所以 nop指令的执行延迟为 0 ycle。
- SYNC：与规范仅有一点不同，既 SYNC指令仅能保证 store 已执行完，但是不能保证 store 的数据已经写入内存。
- SYNCI：相当于两条 cache指令，即 cache16， cache21。分别时 hit invalid on icache 已经 hit writback and invlid on dcache
- Cache指令除实现所有R2必需的指令 cache0,cache8,cache16,cache1,cache5,cache9,cache17,cache21 外，还增加两条cache指令， cache28， cache29，执行 lock on hit。
- Prefetch 与 Prefetchx指令：仅实现了 pretch 的功能，但并未使用 hint 域做为不同种类 pretch 的区分。 LS232仅有一种 prefetch 指令。

9.3 浮点指令

LS232处理器中的浮点指令是 MIPS32 的 R1 版本兼容， R2 版本不兼容。但也并不是所有 R2 版本的浮点指令都未实现，具体如下：

- 未实现的 R2 版的浮点指令：
  - 所有的 MADD，fmMSUB.fmt， NMADD.fm

RSQRT.fmt， RECIP.fmt;

- PLL.PS, PLU.PS, PUL.PS, PUU.PS；
- CVT. PS.S; CVT.S.PL ,CVT.S.PU;ALNV.PS；
- LDXC1, LUXC1,LWXC1, SDXC1, SUXC1, SWXC1 ；
- MFHC1,MTHC1；
- Fmt 中的 PS 格式的指令。

- 实现了的 R2 版的浮点指令：

C E I L L . f m t ;

TRUNC.L.fmt； ROUND.L.fmt中除去 p s 格式的指令