计算机体系结构基础

胡伟武、苏孟豪

第03章 特权指令系统

- 特权指令系统简介
- 异常与中断
  - 异常分类
  - 异常处理
  - 中断
- 存储管理
  - 虚拟存储的基本原理
  - MIPS处理器对虚存系统的支持
  - LINUX操作系统的存储管理
  - TLB的性能分析和优化

特权指令系统简介

操作系统专用的指令系统结构

- 用户程序可“看到”：
  - 用户态指令系统结构
  - 操作系统提供的系统调用
  - 以上两部分合称为ABI（Application Binary Interface）
- 操作系统可“看到”：
  - 用户态指令系统结构
  - 特权态指令系统结构
  - 编译器编不出特权态指令系统结构

特权态指令系统结构的作用

- 运行模式定义及切换
  - 如MIPS的kernel和user模式，X86的Ring0-Ring3模式
  - 计算机刚启动时处于核心态，用户程序通过例外可进入核心态
  - 操作系统通过设置控制寄存器或执行ERET指令等进入用户态
- 虚拟存储管理
  - TLB对用户程序不可见，对操作系统可见
  - 与存储管理相关的控制寄存器如页表起始地址
- 异常与中断处理
  - 用户程序需要操作系统服务时发出系统调用
  - 用户程序执行过程中出现的异常（如非法指令）
  - 外部中断（如敲击键盘）

特权态专用的控制寄存器

- 用户态可见的寄存器
  - 通用寄存器
  - 浮点寄存器
  - 部分状态寄存器，如X86的EFLAG、MIPS的FCR
- 核心态还可以访问控制寄存器
  - 控制系统状态
  - 存储管理
  - 例外和中断处理
  - 调试和性能分析
  - 等等

MIPS的控制寄存器

助记符	编号	说明
SR	12	状态寄存器，包含CPU特权等级、中断使能和其他模式配置。
Cause	13	标记异常和中断发生的原因。
EPC	14	异常程序计数器，异常和中断处理结束后的重新执行地址。
Count	9	组成一个简单但有用的高精度内部计数器。
Compare	11	
BadVAddr	8	存放导致地址相关异常的程序地址。
Context	4	存储管理（TLB）相关寄存器，将在第五章进行详细介绍。
EntryHi	10	
EntryLo0-1	2-3	
Index	0	
PageMask	5	
Random	1	
Wired	6	
PRId	15	CPU类型和版本标识。
Config	16	CPU设置参数，依据实现而不同。
Config1-3	16.1-3	
EBase	15.1	异常入口指针基地址，多处理器时作为CPU标识。
IntCtl	12.1	向量中断和中断优先级相关设置。
SRSCtl	12.2	影子寄存器相关控制。
SRSMap	12.3	
CacheErr	27	用于分析内存错误的寄存器。
ECC	26	
ErrorEPC	30	
TagLo	28.0	cache操作相关寄存器。
DataLo	28.1	
TagHi	29.0	
DataHi	29.1	
Debug	23.0	EJTAG调试单元相关寄存器。
DEPC	24.0	
DESAVE	31.0	
WatchLo	18.0	数据观测点寄存器，当CPU对该地址进行访存时会触发异常。
WatchHi	19.0	
PerfCtl	25.0	性能计数器寄存器。
PerfCnt	25.1	
LLAddr	17.0	存放LL指令的地址。
HWREna	7.0	决定哪些硬件寄存器对用户特权程序可访问。

异常与中断

什么是异常

使处理器从软件的*正常执行流*中脱离的事件，也称例外

异常与中断

---异常分类

异常分类 – 依据来源

- 外部事件
- 指令执行中的错误
- 数据完整性问题
- 地址转换异常
- 系统调用和陷入
- 需要软件修正的运算

外部事件

- 来自CPU流水线外部的事件，也称中断
  - 输入输出设备
  - USB、网卡、键盘、鼠标
- 特点
  - 独立于CPU指令流，纯异步
- 使用中断模式后CPU计算可与I/O处理并行
  - 在等待键盘输入时，CPU可以解视频，或者进入低功耗模式
  - 对比轮询模式，为了得知I/O状态，CPU需要主动地查询
  - 绝大多数外设都采用中断模式（*其它定时查询*）

指令执行中的错误

- 保留指令（Reserved Instruction）
- 浮点除以0
- 整数运算溢出
- 地址不对齐, AdE
- 用户态下的无权限访问
  - 只能在核心态下执行的指令，CpU
  - 非法地址空间访问（Kseg2）, AdE

数据完整性问题

- 存储器发生软错误，导致ECC或奇偶校验错误
- 软错误：可恢复的电位翻转
  - 存储器或触发器都可能产生，但由于存储器的信息密度较高，更容易产生软错误
  - 通常来源于高能粒子冲击，如宇宙射线、核辐射。因此卫星中使用的芯片尤其要注意减少软错误的发生。
  - 减少软错误的方法：使用抗辐照的工艺（如SOI）；使用纠错机制（如ECC校验、三模冗余校验）

地址转换异常

- 从虚地址到物理地址转换时，没有有效的TLB项
- TLB（Translation Lookaside Buffer）是内存中页表在处理器中的缓存（局部性）
- TLB异常
  - TLB refill
  - TLB invalid
  - TLB modify

系统调用和陷入

- 软件故意产生的异常
- 系统调用：操作系统为用户态程序访问核心态资源准备的接口
- 陷入：软件预设的断点，如MIPS的TRAP指令、整数除零检查

```
        li      v0, ID
        syscall

        bnez    v0, 1f
        div     zero, v1, v0
        break   0x7
1:

        teq     v0, $0, 0x7
        div     zero, v1, v0
```

需要软件修正的运算

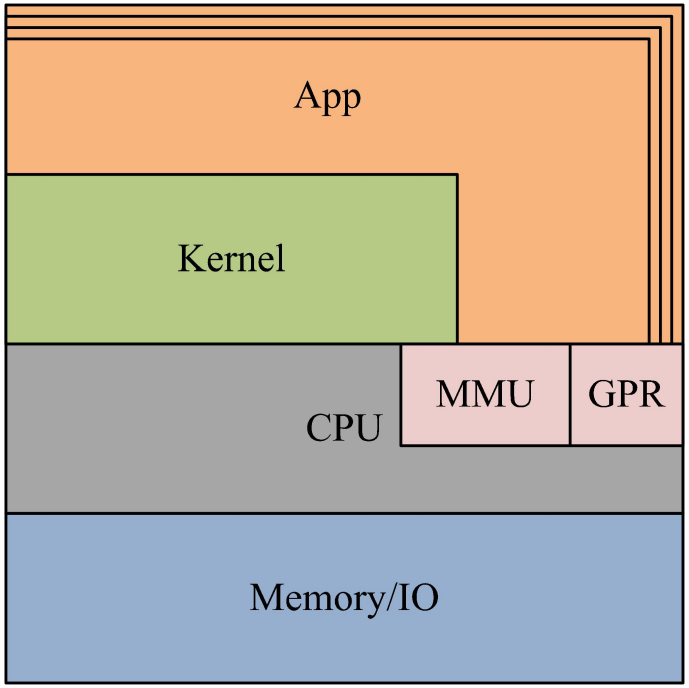
- 通常来源于浮点运算指令
- 如操作数特别小(非规格化非0数)
  - 正常浮点数：11位指数，53位有效数字(1.xxxx)
  - 特别小的数：指数为0(-1022)，52位有效数字(0.xxx)
- 没有硬件浮点支持时
  - 操作系统模拟

MIPS定义的例外

例外号	名称	说明
0x00	Int	中断
-	TLBRefill	TLB中找不到映射项，专用例外入口
0x01	Mod	TLB中找到映射项，但其中D位为0
0x02,0x03	TLBL/S	TLB中找到映射项，但其中V位为0
0x04,0x05	AdEL/AdES	非对齐、无权访问、非法64位
0x06,0x07	IBE/DBE	总线错
0x08	Sys	系统调用
0x09	Bp	指令断点(BREAK、SDBBP)
0x0A	RI	保留指令
0x0B	CpU	协处理器不可用
0x0C	Ov	整数溢出
0x0F	FPE	浮点例外
0x17	WATCH	地址监视例外
0x18	MCheck	内部一致性问题，如TLB中一个VPN有多项映射
0x30	CacheErr	Cache数据校验错

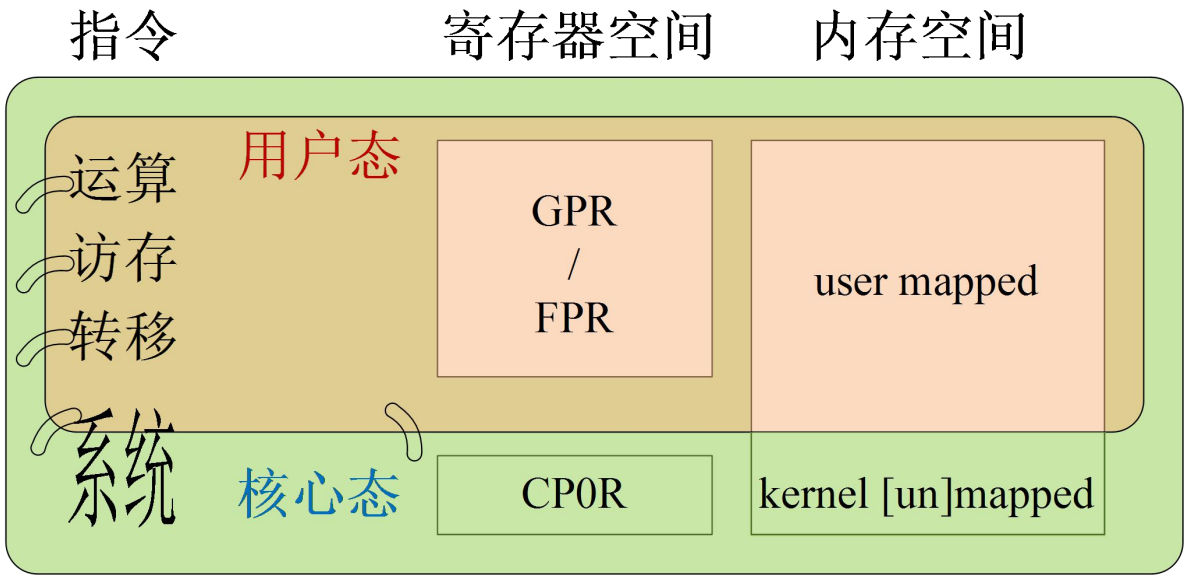
有了例外机制

- 结合操作系统，支撑应用软件运行
  - 统一、“无限”的存储空间
  - 方便的系统调用
  - “独占”CPU*的假象*
- 与运行级别、访问控制结合
  - 安全隔离



指令系统全貌

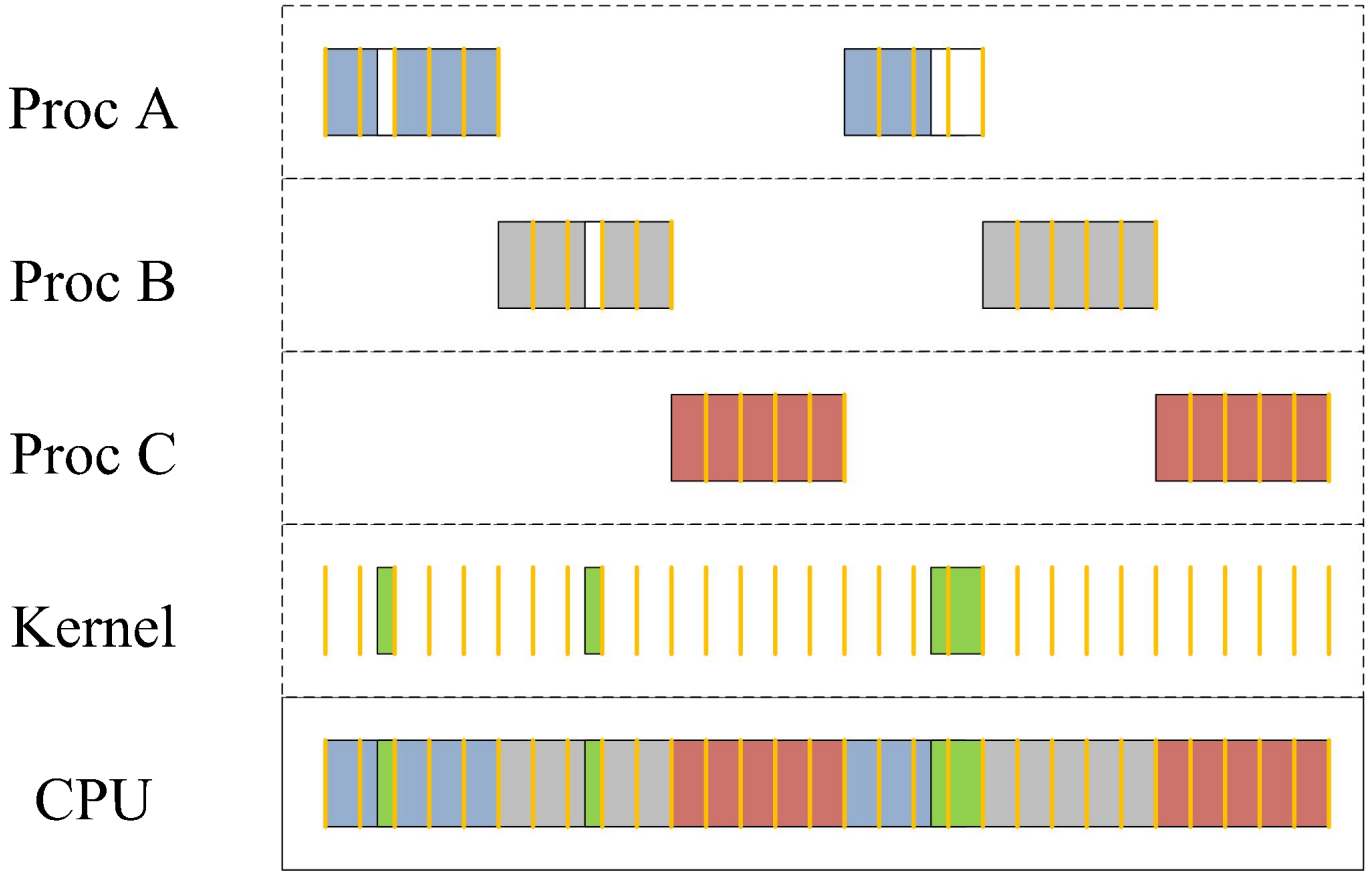
- 指令集、运行级别、地址空间、存储管理
- 异常与中断发挥了纽带作用



进程调度的例子

- 基于时间片的调度
  - 每次调度，从就绪状态的进程中选一个，指定时间片长度
  - 在中断/例外时检查时间片是否用光
  - 如果是，则根据调度算法选另一个进程运行
- MIPS处理器中时间的维护
  - CP0_COUNT: 每两个时钟周期加1
  - CP0_COMPARE: 比较值，与COUNT相等时发时钟中断
  - 中断处理中往COMPARE加一个时间单位

进程调度的例子



异常与中断

---异常处理

异常处理的流程

A. 异常处理准备

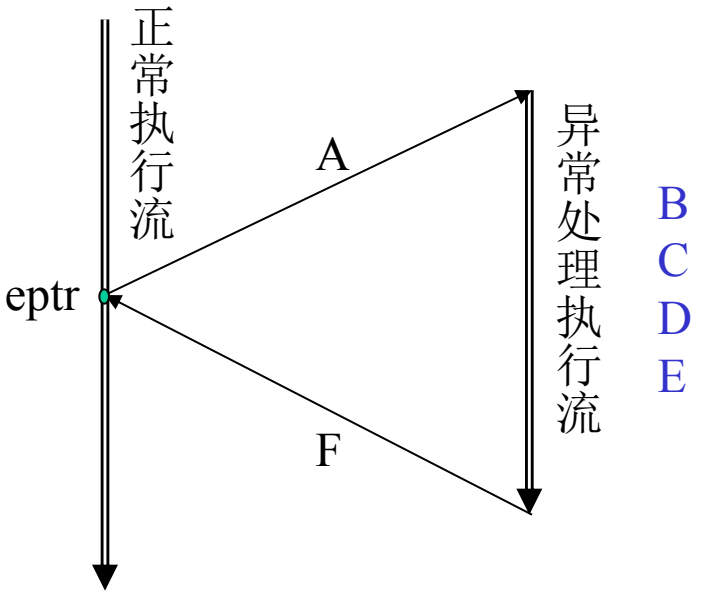
B. 确定异常来源

C. 保存执行状态

D. 执行异常处理

E. 恢复执行状态

F. 返回正常执行流



异常处理准备

- 精确异常：异常产生时，被异常打断的指令（eptr）前的指令都已经执行完，eptr后的指令都未执行
- 硬件负责完成精确异常，并将eptr保存到协处理器寄存器中以供恢复
  - MIPS：EPC寄存器
  - Power：SRR0、CSRR0
  - SPARC：TPC[TL]
  - X86：CS、EIP

确定异常来源

- **X86：由硬件确定异常编号，并查询存放在内存中的中断描述符表（IDT），得到异常处理地址。**
- **MIPS：将异常相关状态存放于Cause寄存器中，由软件进一步查询并区分处理，但一些常用异常有专用入口**

内存区域	入口地址	异常处理
片上调试	0xff200200	EJTAG调试，映射到探测内存区域时
	0xbfc00480	EJTAG调试，用普通ROM内存映射时
重启	0xbfc00000	重启和NMI入口点
ROM中的入口点	0xbfc00400	中断专用，只当Cause[IV]有效时
	0xbfc00380	所有其他异常
	0xbfc00300	缓存错误
	0xbfc00200	简单的TLB重填
RAM中的入口点	BASE+0x200+…	多中断入口
	BASE+0x200	特殊的中断，当Cause[IV]有效时
	BASE+0x180	所有其他异常
	BASE+0x100	缓存错误，但通过非缓存的kseg1访问
	BASE+0x000	简单的TLB重填

保存执行状态

- 将被打断程序的状态保存到栈中供恢复
  - 除**k0/k1**外的所有通用寄存器
  - 其他需要保存的协处理器寄存器（如何优化？）
- 根据中断优先级要求关闭或保留部分中断使能

处理异常

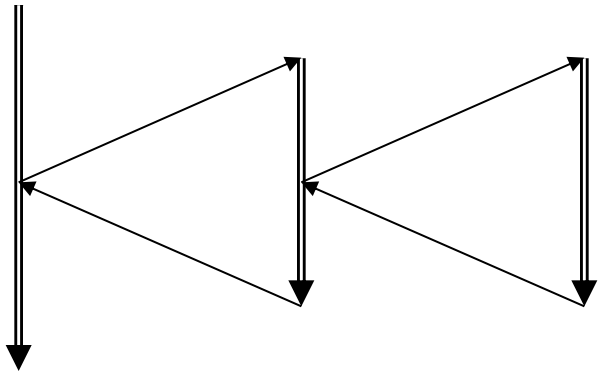
- 根据确定好的异常处理入口地址跳转到对应的处理程序
- 中断处理需要清除中断源状态
- 当处理程序很长时，还需要考虑只处理较为紧急的部分，防止影响对其它中断的响应

恢复执行状态并返回

- 从栈中恢复相关寄存器的值
- 从异常返回正常的执行流
  - 异常返回往往包含从核心态变为用户态的模式切换，模式切换和地址变化需要同时完成（原子地），否则可能导致保护模式失效
  - **MIPS中的ERET指令，X86中的IRET指令**

异常嵌套

- 在异常处理中又产生了新的异常，如何处理？
- 无限的异常嵌套不可取
- 将异常分为多个不同的优先级，更高优先级的异常可以抢占处理，同优先级或低优先级只能等待



异常与中断

---中断

中断

- 嵌入式系统中，CPU的主要作用之一就是处理中断
- 两类中断：可屏蔽（INT）、不可屏蔽（NMI）
  - MIPS中有八个可屏蔽中断位INT0-INT7，通常INT2-INT7可供外部的中断请求使用，INT0/1用于软中断
  - 使用处理器外的中断控制器来提供更多的中断输入
  - NMI来源于硬件预设的致命错误，其处理入口地址与复位相同，均为0xbfc0000，但会在寄存器中保存NMI状态位

中断优先级

- MIPS在内的许多指令系统将中断和异常一视同仁，但事实上确实需要有优先级的区别
- 一个软件的中断优先级方案：
  - 软件维护一个中断优先级（IPL），对每个中断源赋予特定的优先级
  - 正常执行状态，CPU在最低优先级，所有中断可触发
  - 处于最高中断优先级时，所有中断被屏蔽
  - 高优先级中断可抢占低优先级中断

中断处理的原子性

- 在MIPS中，对中断的使能和屏蔽都需要通过读写协处理器寄存器SR来进行
- 若在标号1和2之间被中断，可能导致SR被修改了不符合预期的值
  - 原子指令：如MIPS的DI指令
  - 禁用中断
  - 约定中断退出时恢复原SR
  - 通用的临界区处理

```
    mfc0 t0, SR
1:
    or   t0, <SET>
    and  t0, <CLR>
2:
    mtc0 t0, SR
    ehb
```

通用的原子性处理

- 信号量、锁
- 基本步骤：等待锁并加锁、做原子操作、打开锁
- 原子性的保障在于“测试并设置”的原子性
- 68000系列的测试并设置指令、x86的lock类指令
- 另一种思路：先设置，只在原子性未被破坏时生效
  - MIPS的LL（Load Linked）、SC（store conditional）指令
  - Transactional memory

向量化中断

- 在嵌入式CPU中，中断处理是CPU重要的工作内容
- MIPS向量化中断：
  - 为每个可屏蔽中断分配独立的入口地址（最多6个外部中断）
  - IntCtl寄存器的VS域定义入口地址之间的间隔
- MIPS EIC模式中断：
  - 面向更多的中断源，减少确定中断来源的代价
  - INT2-INT7形成编码，从而支持63个中断输入（0表示没有中断）

X86的原生向量化中断

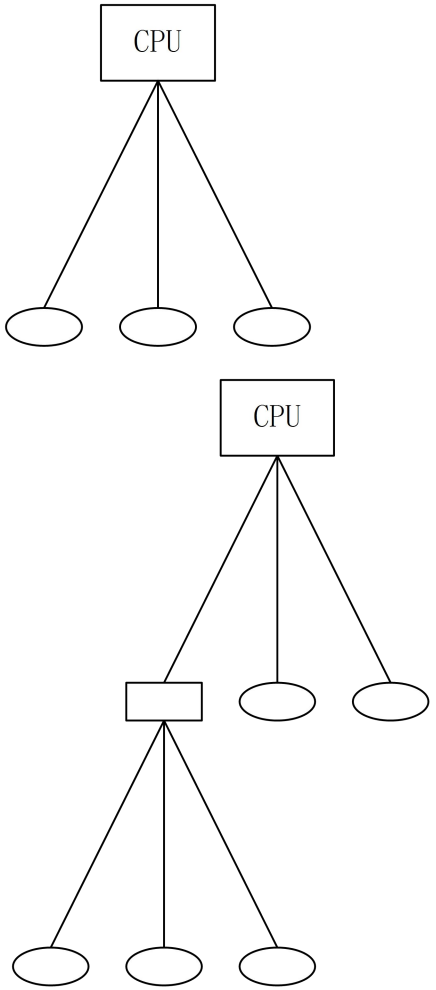
- 地址空间的特定位置存放中断向量表或中断描述符表
- 硬件完成中断号的识别和入口地址查表
- 中断向量表中保存中断入口地址的段地址和偏移
- 中断描述符表还包含特权等级和描述符类别等信息
- 最多支持256个中断和异常
  - 0-19号为系统预设的异常和NMI
  - 20-31号为Intel保留的编号
  - 32号开始可用于外部中断

传递中断事件

- 把来自系统中各个设备的各种中断信号传递到CPU
- 怎么传？

传递中断事件

- 中断线
  - 直接拉线，送到CPU引脚
  - 汇总到中断控制器，再转接到CPU
- 中断消息
  - 通过系统总线，发中断消息到CPU

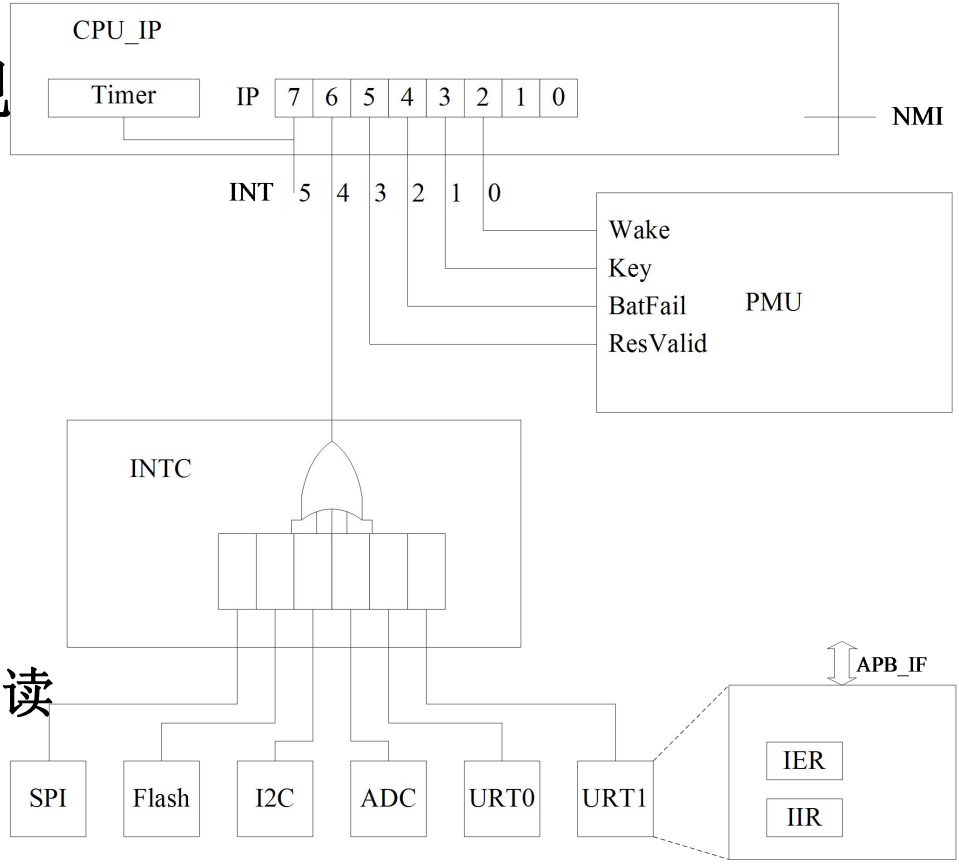


中断线

- 专用信号线传输
  - 带外信号(side-band)
- 中断类型
  - 边沿中断：由信号变化的边沿表示，需要在中断控制器记录
  - 电平中断：由中断线电平高低表示，信号传递途中无存储

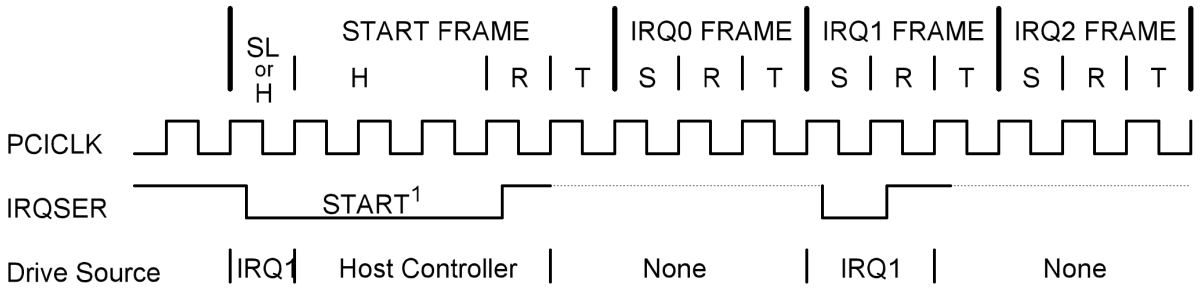
中断线-实例

- 典型的MIPS系统中断实现
  - CPU核内部定时器中断
  - CPU核引脚输入的中断
  - 由中断控制器汇总的中断
- 中断来源判定过程
  - 先读CP0_Cause.IP
  - 如果来自中断控制器则再去读相应状态



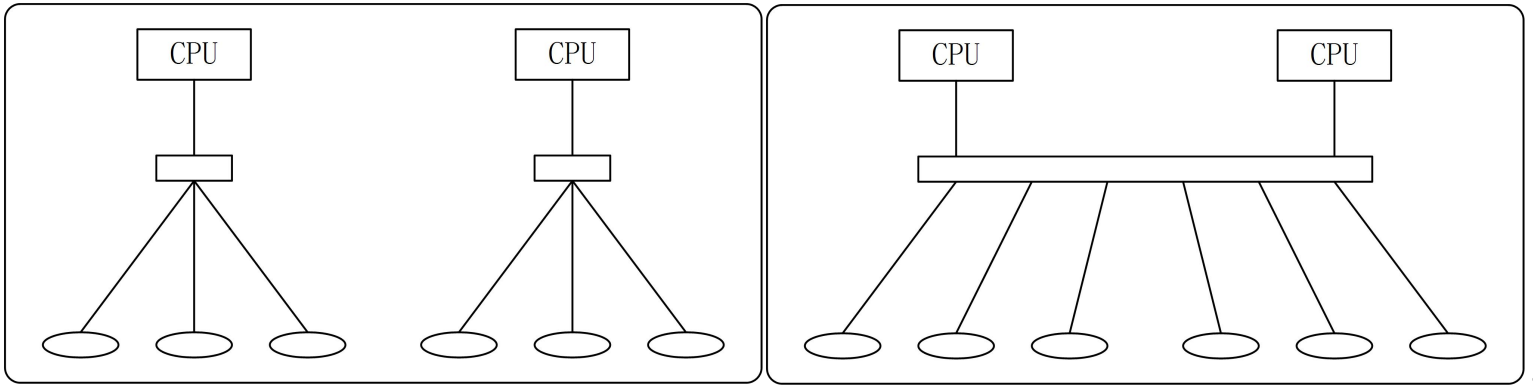
中断线

- 中断线共享问题
  - 片内设备可以有独立中断线
  - 片外资源有限，多个外设挂在同一条中断线上
- 怎么知道谁发了中断？
  - 逐个设备查询，Linux中一个IRQ号上可以挂载多个ISR
  - 串行中断(serirq)，硬件查询



中断线

- 如果有多个CPU，中断哪一个？
  - 由硬连线决定
  - 在中断控制器重定向
  - 如何做到负载均衡？
  - 如何让存在前后关联的中断在同一个CPU处理？



中断线

- **CPU被中断后，能否看到更新后的硬件状态?**
  - 连线传递近乎“光速”（如果不考虑中间的触发器）
  - 外设内部寄存器在发出中断时已更新
  - 外设发出中断前的内存写可能还没到达目的地
  - 解决方法
    - 先确保内存写完成，再发中断
    - 把中断也变成一个内存写

中断消息

- **Message Signaled Interrupt (MSI)**
- **通过数据通道的写传递**
  - **带内传递**
  - **无需额外连线资源，扩展性强**
  - **一个设备可以申请N个中断**
- **具体实现**
  - **操作系统给每一个中断分配MSI向量，即发中断的写地址**
  - **在需要发中断时发出对应MSI向量的写**
  - **写到CPU内部的中断控制器上**

中断消息

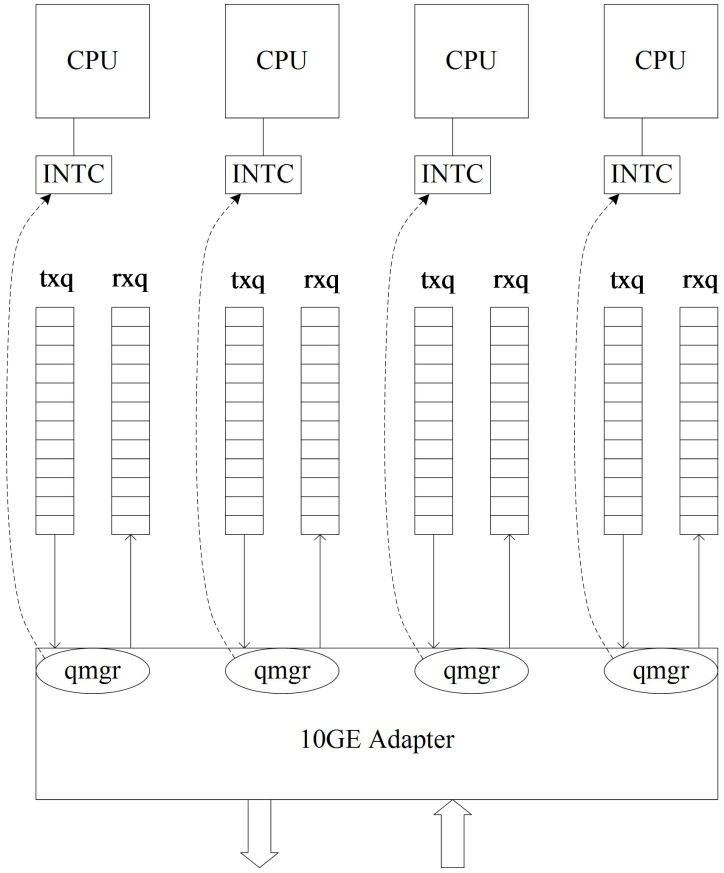
- 降低中断处理延迟
  - CPU被中断后
    - 读中断控制器的状态，确定中断号
    - 逐个执行挂在该中断号下的ISR
    - ISR去访问设备寄存器，确定是否发生中断
    - 如果有再进一步处理，清中断
  - 可降低的延迟
    - 不用读远在桥片上的中断控制器
    - 不用读可能更远的外设寄存器
    - 避免中断号共享

中断消息

- 实现负载均衡
  - 某些高速外设中断频度非常高，一个CPU不堪重负
  - 例如10G以太网
    - 全是大包：每秒有10G/8/1500约83万个包
    - 全是小包：每秒有10G/8/64约2千万个包
    - 如果每个包都对应一个中断，则一个2GHz处理器需要每100~2000个周期处理一个中断，及其对应的包
    - 一般内存访问延迟为100ns，200周期!

中断消息

- 实现负载均衡
  - 充分利用多处理器的特性
  - 在设备端进行中断分配
    - 多队列，对应不同的CPU
    - 接收到的包，分析包头确定网络流，相同流的包放到同一个队列，优化局部性



描述符DMA（*相当于网卡的指令*）
txdesc：发送长度、内存缓冲区位置
rxdesc：内存缓冲区位置、缓冲区大小
正常工作过程中CPU与网卡主要通过内存交互

存储管理

一个访存例子

- 一个程序片段
  - **array = (int*)malloc(0x1000);**
  - **for (i=0;i<1024;i++) array[i] = 0;**
- 软件功能
  - **分配一个1024项的一维整数数组array，并初始化为0；**
- 硬件过程？
  - **数组地址分配在什么地址？**
  - **数组存在什么地方（内存/硬盘）？**
  - **什么时候分配？什么时候存？**
  - **虚地址和物理地址如何转换？**

存储管理

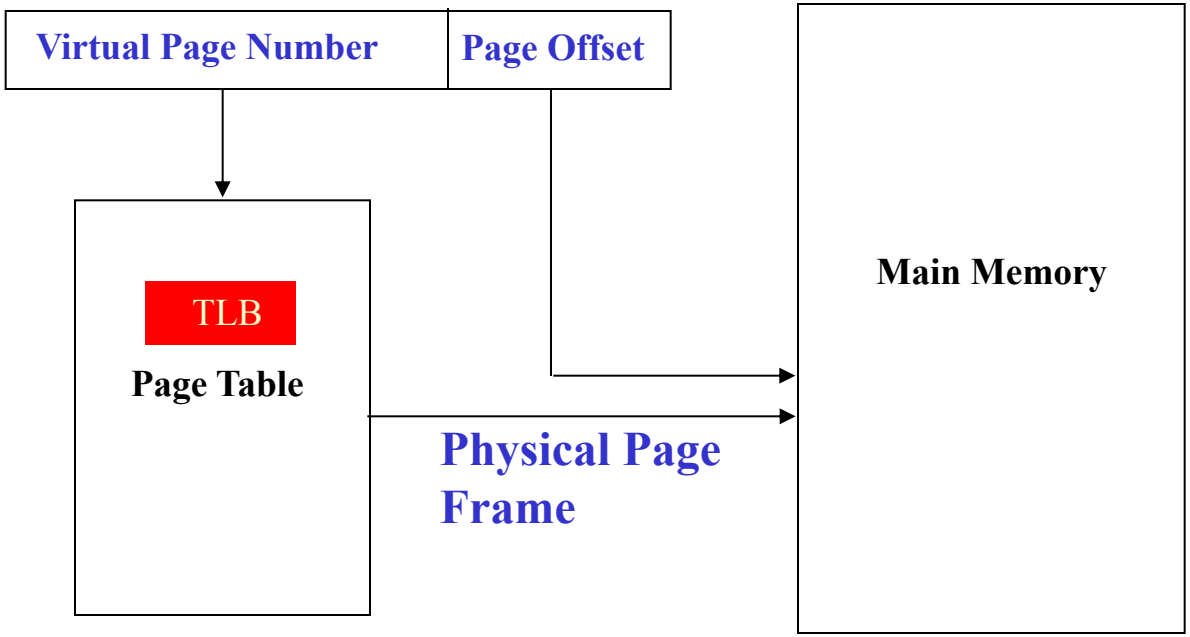
---虚拟存储的基本原理

虚拟存储原理

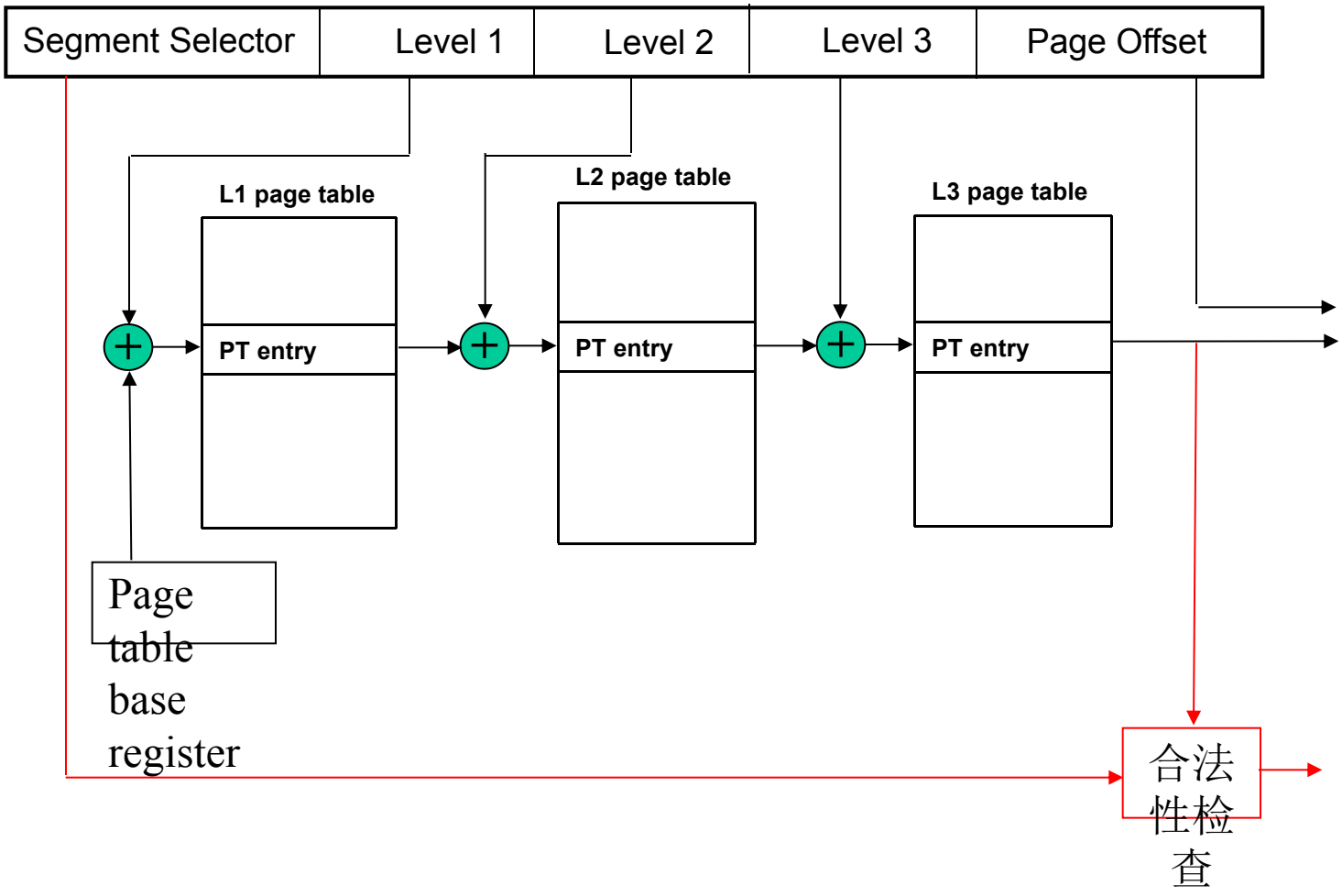
- 虚拟存储是计算机系统发展过程中有里程碑作用的事件
  - 多进程环境下统一的编程空间
  - 多进程环境下的共享与保护
  - 支持大于实际物理内存的编程空间
- 虚实地址分开，建立一种从虚地址空间映射到物理内存的机制
  - 把两个层次的存储转换为一个层次的存储
  - 物理内存实际上是磁盘的一个Cache

虚实地址转换与页表

- 在页的范围内，虚实地址相等
- **TLB是页表的cache**



多级页表


Segment Selector
Level 1
Level 2
Level 3
Page Offset
L1 page table
L2 page table
L3 page table
PT entry
PT entry
PT entry
Page table base register
合法性检查

TLB

- **TLB实际上是操作系统中页表的Cache**
  - **TLB主要负责完成用户空间到物理空间的转化**
  - **一般与Cache访问同时进行**
  - **TLB内容：虚地址（Cache的Tag），物理地址（Cache的Data），保护位（Cache的状态）**
- **TLB失效处理**
  - **TLB失效时需要把相应页表内容从内存取到TLB**
  - **TLB失效时硬件（如X86的page walker）和软件（如MIPS的特殊例外）来填充TLB**

Cache和虚拟存储

比较内容	Cache	虚拟存储
调度单位	块（16B-128B）	页（4KB-64KB）
命中延迟	1-3时钟周期	50-150时钟周期
失效延迟	8-150时钟周期	1,000,000-10,000,000时钟周期
失效率	0.1-10%	0.00001-0.001%
映射前地址	25-45位物理地址	32-64位虚地址
映射后地址	14-20位cache地址	25-45位物理地址
映射者	硬件	硬件（TLB）+操作系统
组织方式	直接相联、组相联、全相联	全相联、组相联（Page Coloring）
替换方式	随机、FIFO、LRU	LRU
写回方式	Write-back、Write-through	Write-back

分清两种映射关系

- **TLB是页表的Cache**
  - 负责地址转换，一页管4KB以上数据
  - 页表由操作系统管理，存在系统空间
- **内存是硬盘的Cache**
  - 负责数据缓存，一字节就是一字节
  - 用户程序的数据在用户空间

存储管理

---MIPS处理器对虚存系统的支持

MIPS处理器对虚拟存储的支持

- 分段，段内分页
- **TLB**
- 特殊的控制寄存器
- 特殊指令
- 专用的例外入口

MIPS的访问权限

- **User mode**
  - **EXL=0 and ERL=0 and KSU=10**
- **Supervisor mode**
  - **EXL=0 and ERL=0 and KSU=01**
- **Kernel mode**
  - **EXL=1 or ERL=1 or KSU=00**

MIPS存储空间分段情况

- **32位模式**

地址范围	容量	映射方式	Cached	访问权限
0xe0000000-0xffffffff	0.5GB	查找TLB	Yes (TLB)	Kernel
0xc0000000-0xdfffffff	0.5GB	查找TLB	Yes (TLB)	Kernel, Supervisor
0xa0000000-0xbfffffff	0.5GB	地址-0xa0000000	No	Kernel
0x80000000-0x9fffffff	0.5GB	地址-0x80000000	Yes (Config)	Kernel
0x00000000-0x7fffffff	2GB	查找TLB	Yes (TLB)	Kernel, Supervisor, User

MIPS的TLB及相关控制寄存器

- **32位模式**
- **全相联**
- **32-64项**

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0													MASK																		
0																															
VPN2																						G	0						ASID		
0																															
PFN																												C	D	V	G
0																															
PFN																												C	D	V	G

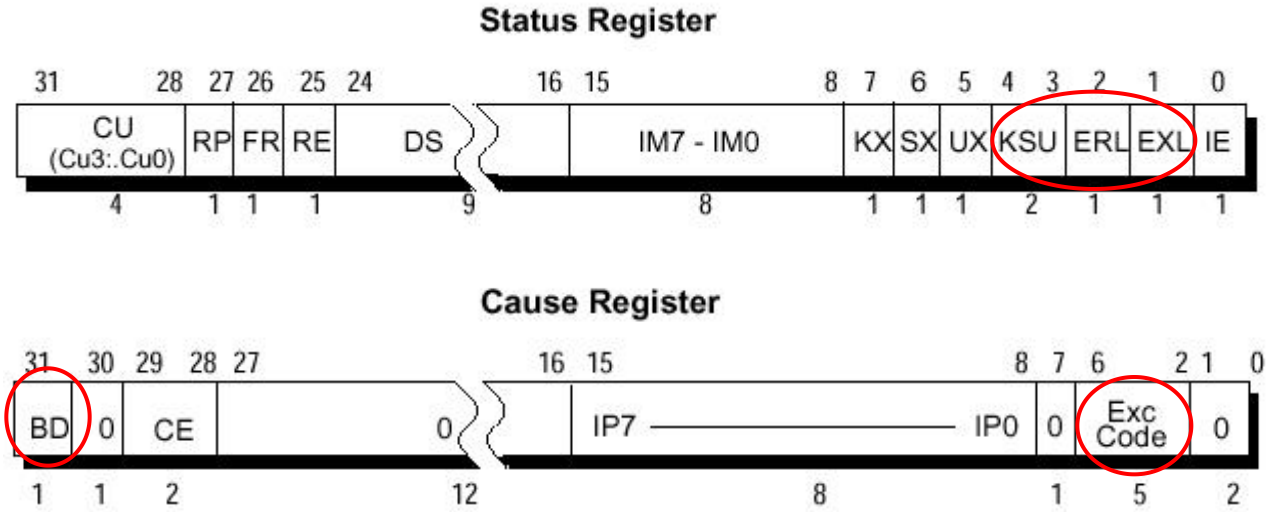
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Pagemask	0												MASK																			
EntryHi	VPN2																						0							ASID		
EntryLo0	0																													C	D	V	G
EntryLo1	0																													C	D	V	G
Index	P	0																												Index		
Random	0																													Random		
Wired	0																													Wired		
EPC	EPC																															
BadVAddr	Bad Virtual Address																															
Context	PTEBase												BadVPN2																		0	

与TLB管理有关的指令

- **MFC0, MTC0**
  - **在通用寄存器和控制寄存器之间搬运数据**
- **TLBR**
  - **以Index寄存器为索引把TLB内容读到PageMask、EntryHi和EntryLo0/1等寄存器**
- **TLBP**
  - **检查EntryHi中指定的虚页是否在TLB中**
- **TLBWR, TLBWI**
  - **分别以Random和Index寄存器为索引把Pagemask、EntryHi和EntryLo0/1寄存器的内容写入TLB**

发生TLB例外时硬件处理过程

- **置BadVaddr, Context, EntryHi,**
- **PC=例外入口地址**
  - **TLB Refill入口=0x80000000**
  - **其它入口=0x80000180**
- **置Status, Cause**



TLB例外类型

- **Refill**
  - **如果查找TLB没有找到一个虚地址匹配（VPN2+ASID/G）**
  - **例外入口：80000000 （除非exl=1）**
- **TLB invalid**
  - **如果找到一个虚地址匹配项，但其v=0**
  - **例外入口：80000180**
  - **细分为两种：TLBL for loads, TLBS for stores**
- **TLB modify**
  - **如果找到一个虚地址匹配项，其v=1，但D=0且访问为store**
  - **例外入口80000180**

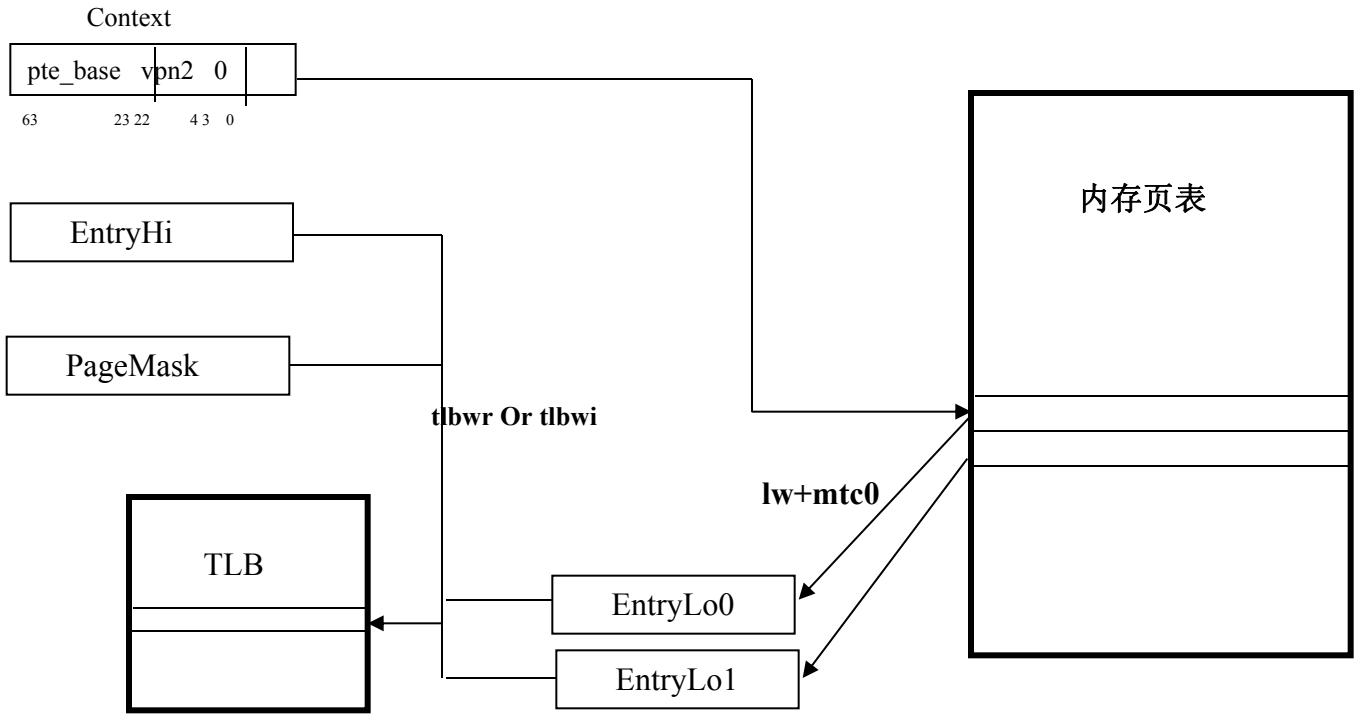
例外返回

- 例外处理器在核心态下进行
  - 不允许在核心态下执行一条用户指令
  - 不允许在用户态下执行指令核心指令
- 例外返回的两种方式
  - **jr+mtc0: mtc0必须在jr的延迟槽中**
  - **eret: eret没有延迟槽**

一种虚拟存储实现方式

- 使得发生例外时context寄存器指向页表中相应项
  - 一维线性页表
  - 内存页表每项8个字节，每对页面占用16字节页表
  - 进程切换时操作系统更改context寄存器中pte_base域，使其指向该进程的页表基地址（pte_base是进程上下文的一部分）
- 一维线性页表需要很大的空间，不能全部分配物理内存
  - 放在kernel mapped的kseg2/kseg3段
  - 需要解决TLB refill重入问题

TLB refill过程



Pagemask在操作系统初始化时由OS设置（固定页操作系统）
Pte_base在进程切换时由OS设置
Vpn2和EntryHi在缺页时由CPU设置

TLB refill代码

```
set  noreorder
.set  noat
TLBmissR4K:
DMFC0    k1,C0_CONTEXT      # (1)
NOP                         # (2)
LW       k0,0(k1)           # (3)
LW       k1,8(k1)           # (4)
MTC0     k0,C0_ENTRYLO0     # (5)
MTC0     k1,C0_ENTRYLO1     # (6)
NOP                         # (7)
TLBWR                       # (8)
ERET                        # (9)
.set  at
.set  reorder
```

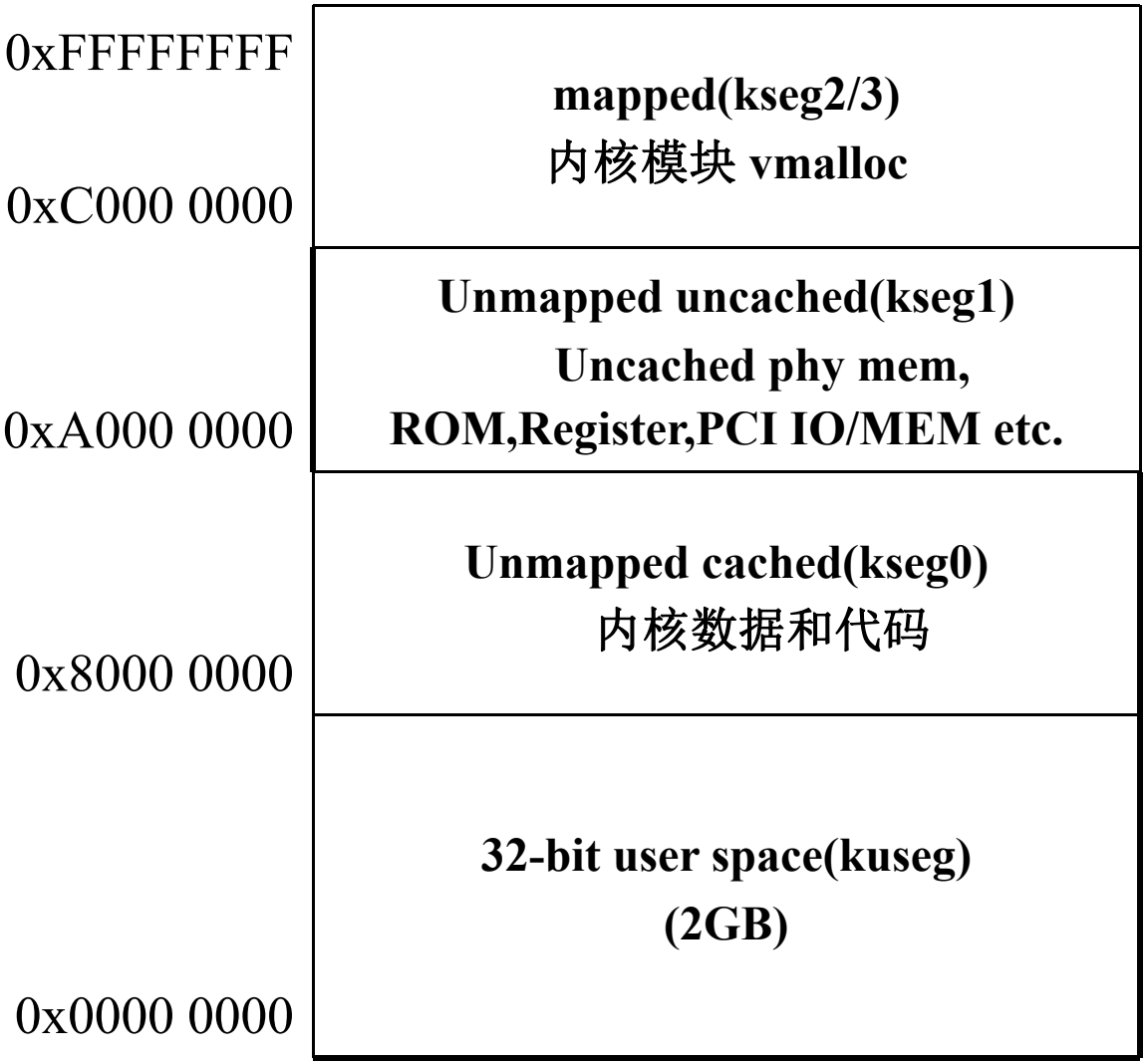
存储管理

---LINUX操作系统的存储管理

Linux/mips虚拟存储管理

- 虚拟地址空间
- 页表和**TLB**
  - 页表组织
  - 例外处理函数
  - 一些优化

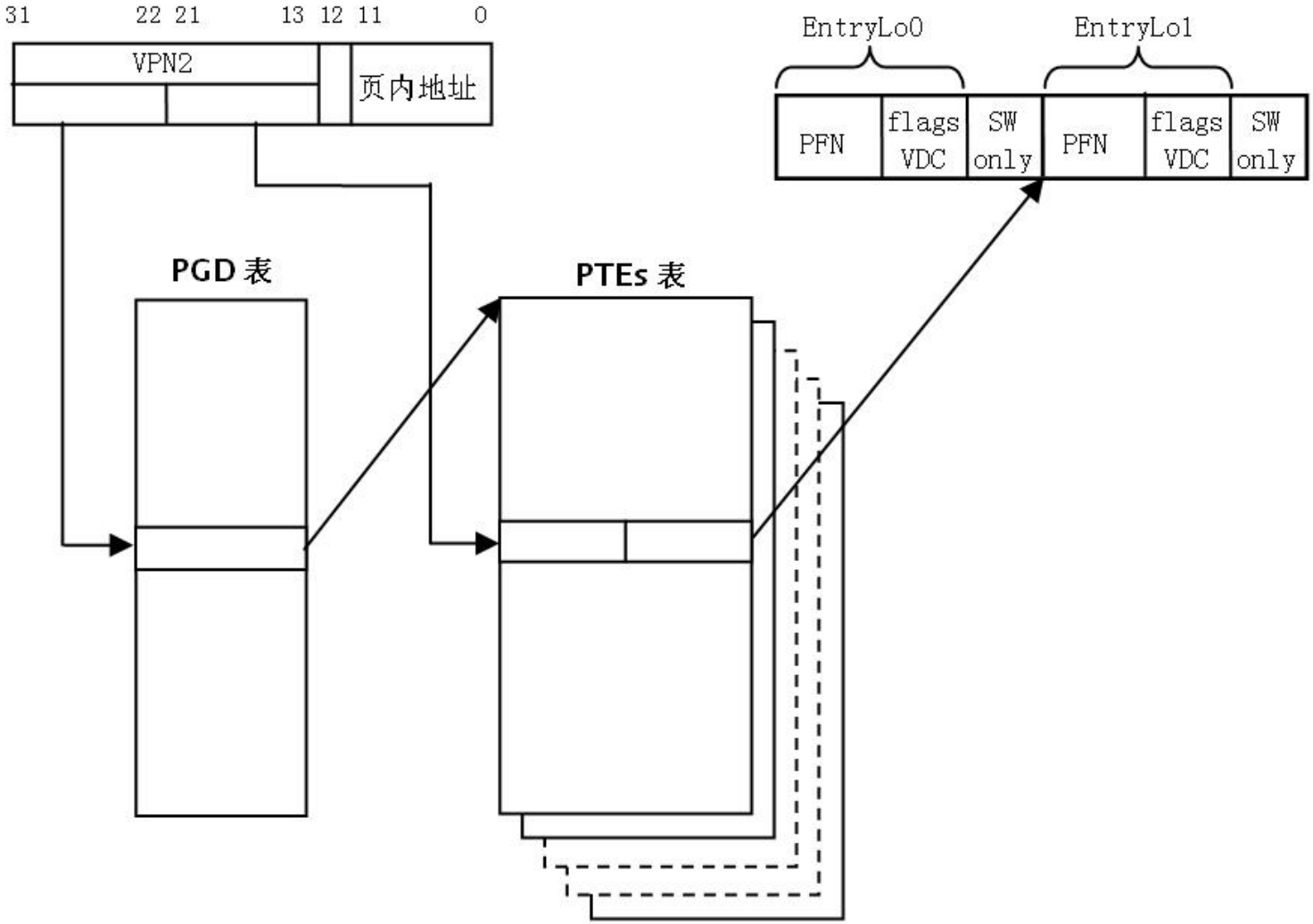
Linux/MIPS虚拟地址空间安排



内存中的页表组织（32位情况）

- 两级页表，每项4个字节
  - PFN：物理帧号
  - Flags：V、C、D
  - Exts：软件扩展位，用于维护一些硬件没有实现的功能，例如ref位，modified位
- 页表存放在kseg0
  - 页表访问不引起TLB例外
  - 页表存储空间在使用到的时候分配
- 每个进程的页表基地址存放在进程上下文中

Linux/MIPS的两层页表



Linux的tlb重填代码（共18条指令）

```
mfc0        k0, CP0_BADVADDR        #取发生tlb miss的地址
srl         k0, k0, 22              #最高10位是第一级页表
的索引
lw          k1, pgd_current          # 取页表入口指针
sll         k0, k0, 2               #每项4个字节,所以索引
*4=偏移
addu k1, k1, k0                  #*k1指向下一级页表入口
mfc0        k0, CP0_CONTEXT         #context包含失效的虚
页号
lw          k1, (k1)                #取出第二级页表
srl         k0, k0, 1
and         k0, k0, 0xff8           #算出第二级页表的偏移
addu k1, k1, k0
lw          k0, 0(k1)               #成对存放,一个偶数页
lw          k1, 4(k1)               #加一个奇数页
srl         k0, k0, 6               #移出6位软件用的位
mtc0        k0, CP0_ENTRYLO0        #写入偶数页表项
srl         k1, k1, 6
mtc0        k1, CP0_ENTRYLO1        #写入奇数页表项
tlbwr                            #写入TLB的一个随机项
eret                              #异常返回
```

例子

- **Array=(int*)malloc(0x1000)**
  - **用户程序malloc(0x1000)返回虚地址0x450000**
  - **操作系统在该进程的vma_struct链表中记录地址范围0x450000-0x4501000为已分配地址，可读可写**
- **For (I=0;I<1024;I++) Array[i] = 0**
  - **用户程序试图写0x450000，TLB查找失败，引起tlb refill例外**
  - **Tlb refill从相应页表位置取入页表内容填入TLB。但该页表还没初始化**
  - **例外返回到用户程序，重新开始访问**
  - **TLB表项找到，但是无效，发生TLB Invalid例外**
  - **操作系统查找vma_struct，判断该地址已分配，处于可写状态，因此为它分配物理页面，并将物理地址填入页表，更新TLB**
  - **例外返回，写操作再次重试，成功。**
  - **用户程序继续写0x450004,008…，因为TLB项已经存在，将全速运行，除非中间发生进程切换导致其TLB项被换出。如果发生被换出的情况，再次运行时将发生refill例外从页表取得有效内容，不会再发生invalid例外(因此,refill频率>>invalid)**
- **为什么要分成两次例外？**

Linux/MIPS中TLB例外的处理

- tlb refill exception(0x80000000):
  - (1) get badvaddr,pgd
  - (2) pte table ptr = badvaddr>>22 < 2 + pgd ,
  - (3) get context,offset = context >> 1 & 0xff8 (bit 21-13 + three zero),
  - (4) load offset(pte table ptr) and offset+4(pte table ptr),
  - *(5) right shift 6 bits,write to entrylo[01],
  - (6) tlbwr
- tlb modified exception(handle_mod):
  - (1) load pte,
  - *(2) if _PAGE_WRITE set,set ACCESSED | MODIFIED | VALID | DIRTY,
    reload tlb,tlbwi
  - else DO_FAULT(1)
- tlb load exception(handle_tlbl):
  - (1) load pte
  - (2) if _PAGE_PRESENT && _PAGE_READ, set ACCESSED | VALID
    else DO_FAULT(0)
- tlb store exception(handle_tlbs):
  - (1) load pte
  - *(2) if _PAGE_PRESENT && _PAGE_WRITE,set ACCESSED | MODIFIED | VALID | DIRTY
    else DO_FAULT(1)

例子

- 程序段
  - **array = (int*)malloc(0x10000);**
  - **for (i=0;i<16384;i++) array[i] = 0;**
- **页大小为4KB，发生了多少次例外？**
  - **地址空间为64KB，16页**
  - **发生8次TLB Refill（MIPS TLB一项两个连续虚页）和16次TLB Invalid**
- **页大小为16KB，发生了多少次例外？**
  - **发生2次TLB Refill和4次TLB Invalid例外**
  - **页大小为4KB，上述程序段执行完后发生进程切换**
  - **切换后再执行for (i=0;i<16384;i++) array[i] = i;**
  - **如果进程切换时TLB项被替换，只发生8次refill例外**
  - **如果进程切换时，array被调出内存，发生8次TLB Refill和16次TLB Invalid例外，invalid例外处理时把array从硬盘调到内存**
  - **如果页表也被调出内存？**

存储管理

---TLB的优化

TLB性能分析和一些优化

- 防缓冲区溢出攻击优化
- 硬件性能优化
- 软件性能优化

利用TLB的保护机制防范攻击

- 利用缓冲区溢出进行攻击的例子
- 龙芯处理器通过可执行保护防止缓冲区溢出攻击
  - **TLB增加可执行位**

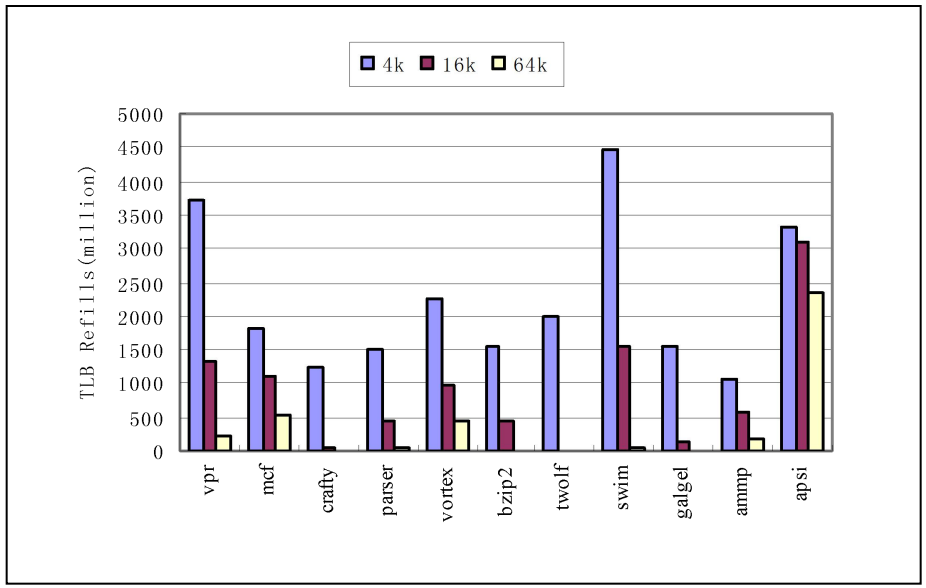
```
void fa(void){
  ...
  function fb(str);
  ...
}

void fb(char *str){
  ...
  char buffer[16];
  ...
  gets(buffer);
  ...
}
```

TLB相关性能数据

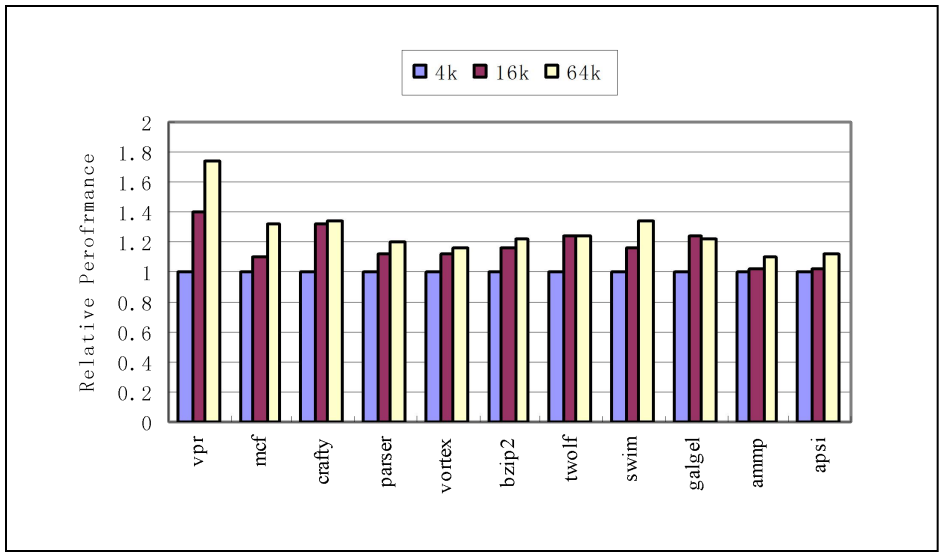
- **TLB miss处理的时间可以占到高达40%的运行时间，占40~90%的内核运行时间**
- **SPEC CPU2000大约1/4的程序有比较显著的TLB miss**
  - **早期多数CPU的TLB是全相连的32-128项，如果每项4KB，能映射的空间只有几百KB，越来越难满足现代程序的需求**
- **性能优化方法**
  - **增加TLB覆盖空间大小，降低TLB失效概率**
  - **降低TLB失效开销**

增加页大小后性能显著提高

- 增加页大小后，TLB失效明显减少
  - 16KB页时128页有2MB
  - 通过软件配置pagemask可以增加/减少页大小

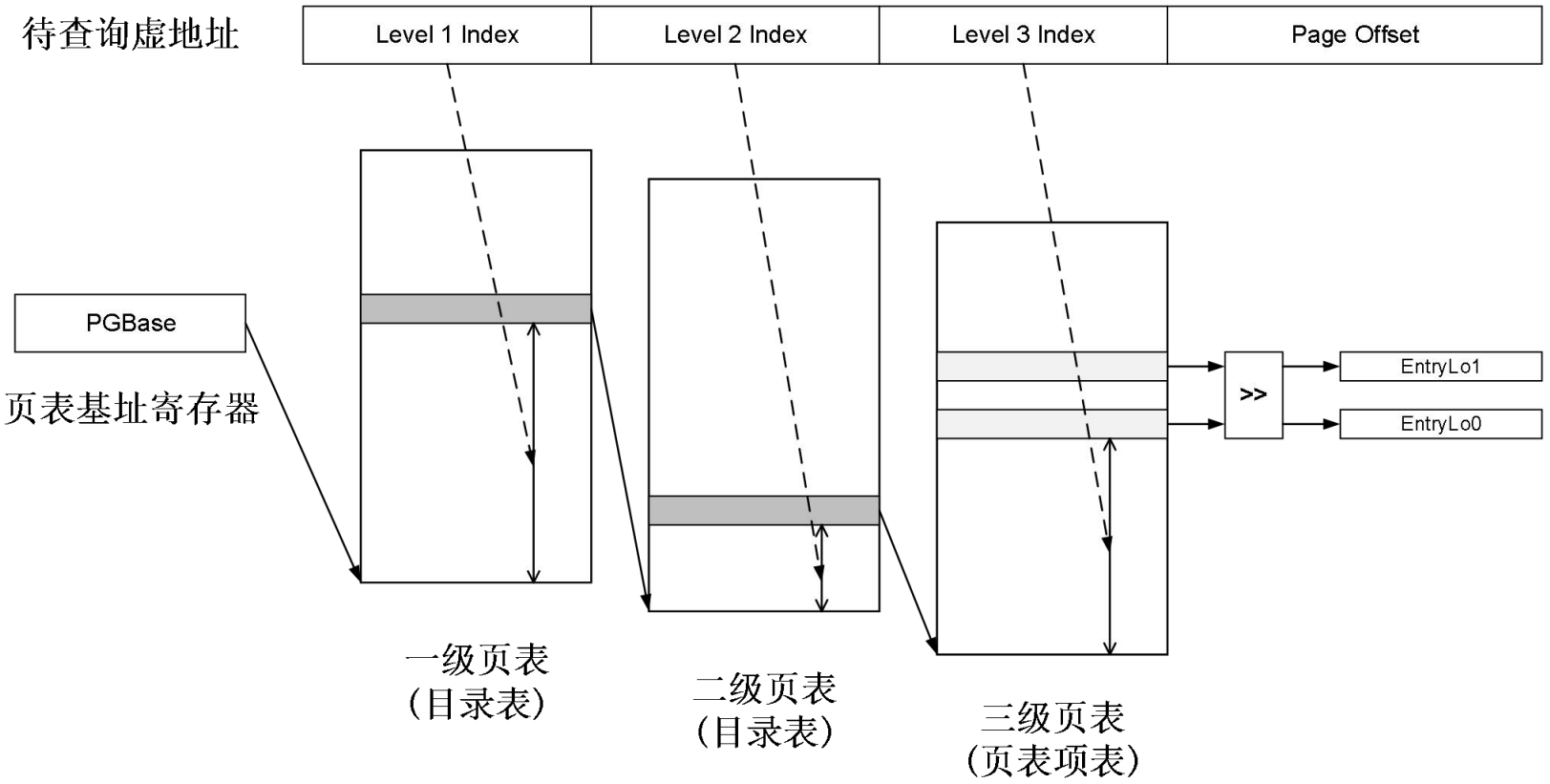


(a) TLB refills of 4KB, 16KB, and 64KB page size



(b) Performance of 4KB, 16KB, and 64KB page size

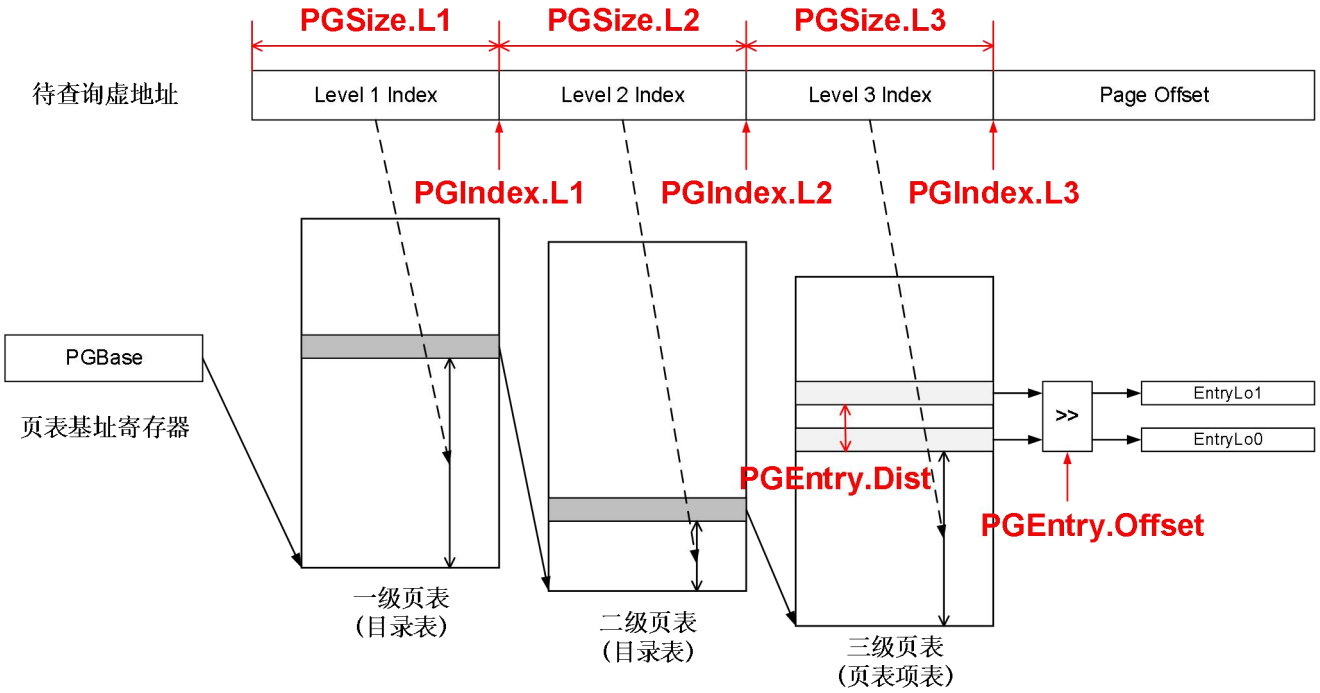
MIPS-Linux多级页表查找过程示意



- **MIPS原Context寄存器设计不再适合多级页表的快速查找**
  - 原设计针对一级页表

加速MIPS-Linux多级页表查找(1)

- 首先通过软件可配置寄存器来描述具体的页表结构
  - **PGBase**：页表基址寄存器
  - **PGSize**：描述各级页表的大小
  - **PGIndex**：描述各级页表索引值在待查询地址中的起始位置
  - **PGEntry**：描述奇偶页表项之间位置关系，以及页表中软件用的位的宽度
- **PGBase是进程上下文一部分，其余寄存器操作系统启动时配置**



加速MIPS-Linux多级页表查找(2)

- 其次定义专门的页表访存指令
  - **LDDIR dest, src, #N：将src寄存器中的值作为第N级目录页表的基址，同时根据CP0.BadVaddr中存放的待查询地址以及新增的PGSize、PGINdex配置寄存器信息，访存得到第N+1级页表的基址，写入到dest寄存器中。**
  - **LDPTE src, #0/#1：将src寄存器中的值作为页表项页表的基址，同时根据CP0.BadVaddr中存放的待查询地址以及新增的PGSize、PGINdex、PGEntry配置寄存器信息，访存得到偶数号/奇数号页表项的内容，将页表中纯粹软件用的位移除后，写入到CP0.EntryLo0/CP0.EntryLo1中。**

MIPS-Linux TLB重填代码优化

优化前

```
mfc0 k0, CP0_BADVADDR
srl  k0, k0, 22
lw   k1, pgd_current
sll  k0, k0, 2
adduk1, k1, k0
mfc0 k0, CP0_CONTEXT
lw   k1, (k1)
srl  k0, k0, 1
and  k0, k0, 0xff8
adduk1, k1, k0
lw   k0, 0(k1)
lw   k1, 4(k1)
srl  k0, k0, 6
mtc0 k0, CP0_ENTRYLO0
srl  k1, k1, 6
mtc0 k1, CP0_ENTRYLO1
tlbwr
eret
```

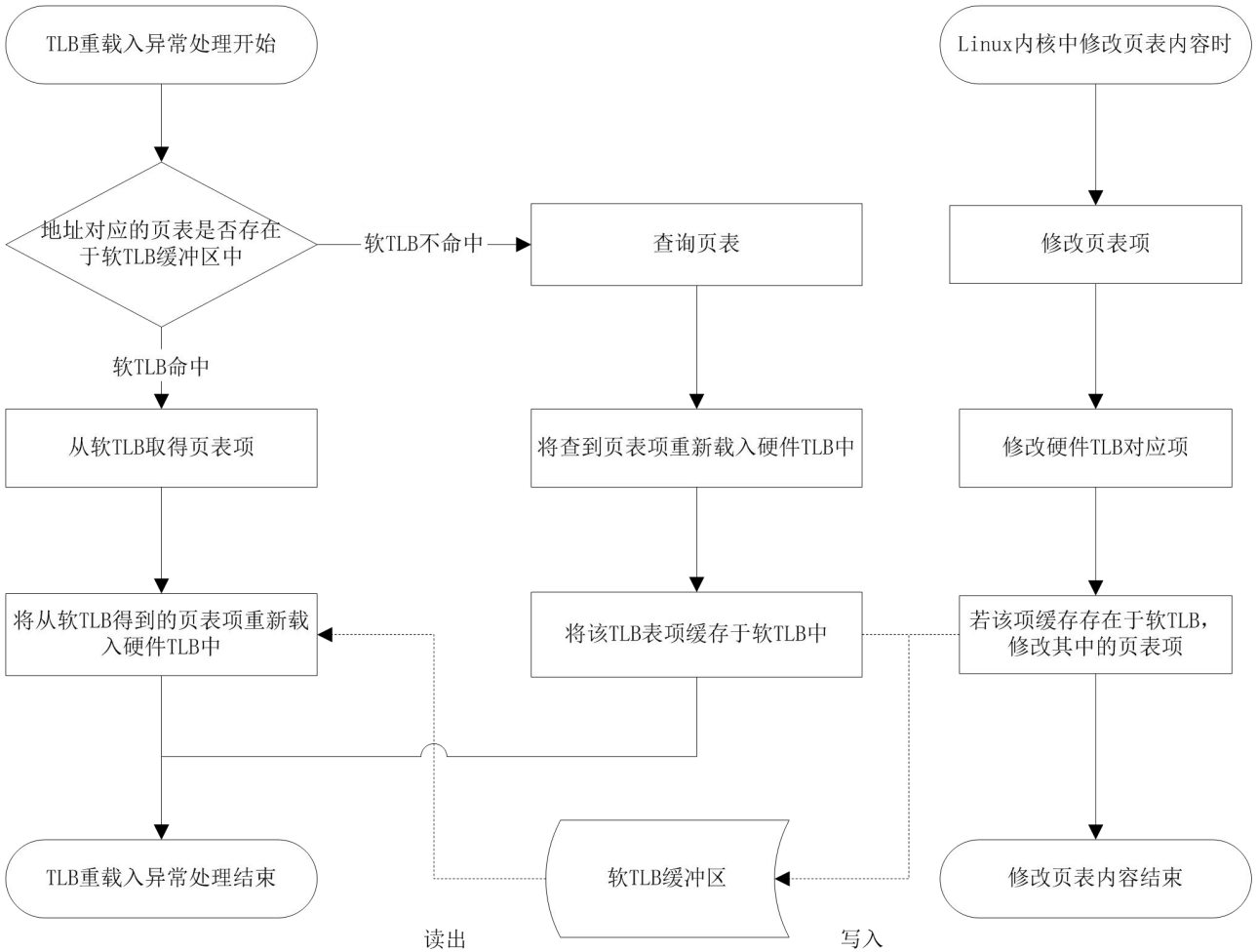
优化后

```
dmfc0   k0, pgbase

lddir   k0, k0, 1  （一级页表）



lddir   k0, k0, 2   （二级页表）









ldpte   k0, 0   三级页表/偶数页）

ldpte   k0, 1  （三级页表/奇数页）
tlbwr
eret
```

软TLB的方案

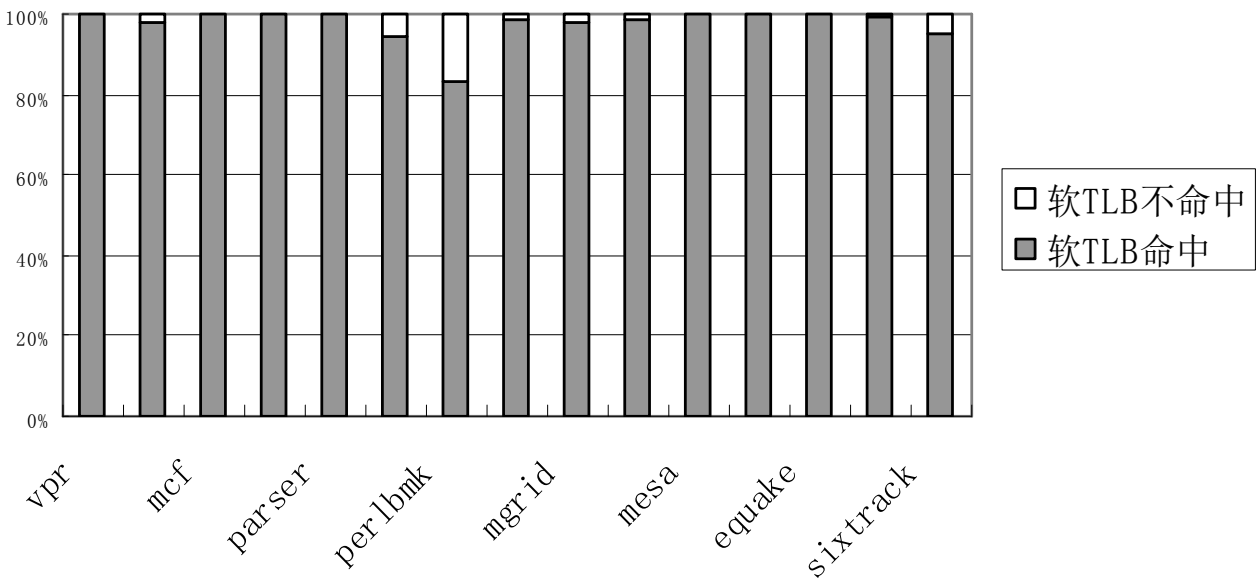
- 目的：主要减少TLB重载入异常处理的时间，提高TLB重载入异常处理的效率。
- 原理：通过减少TLB重载入异常处理过程Cache Miss的次数来减少TLB重载入异常处理的时间

软TLB的方案（续）

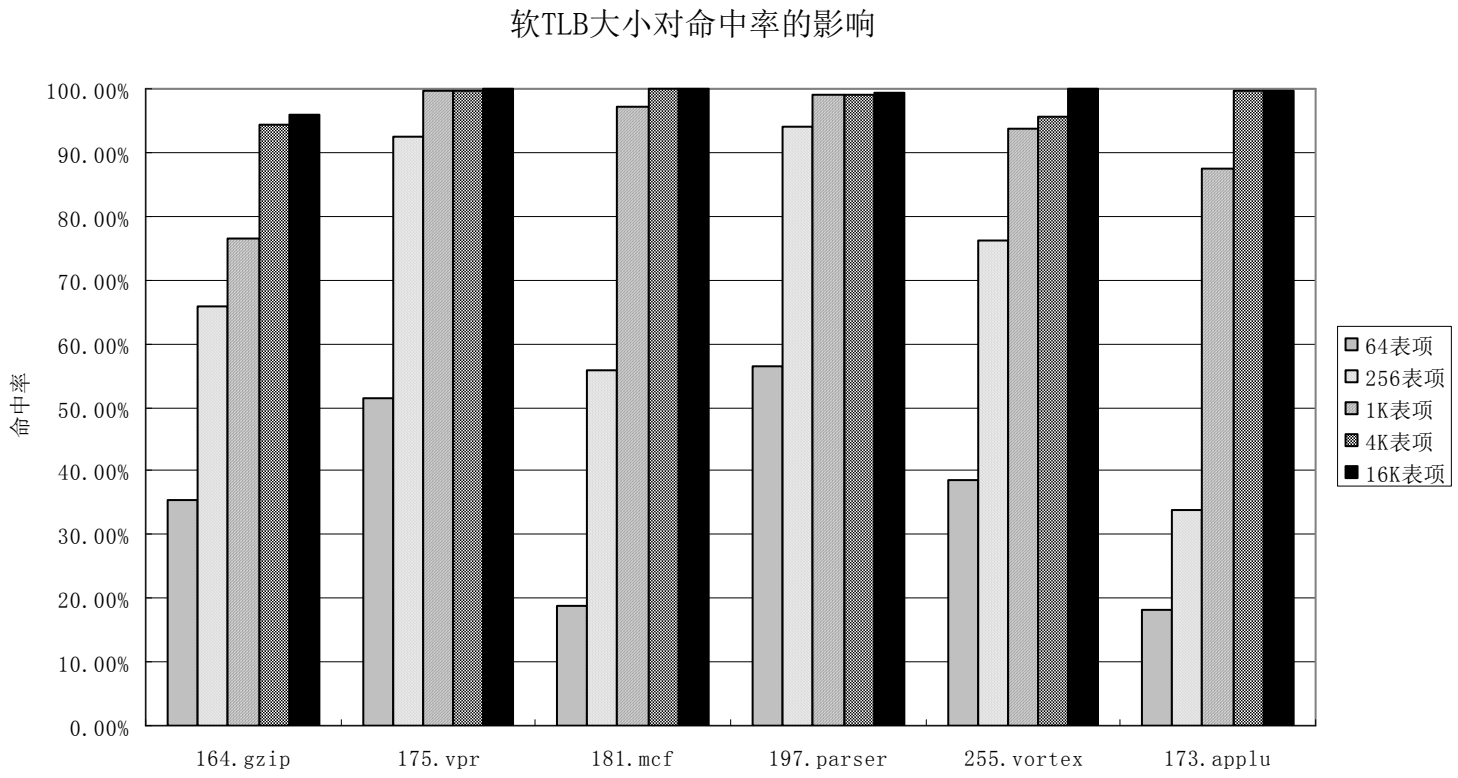

TLB重载入异常处理开始
地址对应的页表是否存在于软TLB缓冲区中
软TLB不命中
查询页表
软TLB命中
从软TLB取得页表项
将查到页表项重新载入硬件TLB中
将从软TLB得到的页表项重新载入硬件TLB中
将该TLB表项缓存于软TLB中
TLB重载入异常处理结束
软TLB缓冲区
读出
写入
Linux内核中修改页表内容时
修改页表项
修改硬件TLB对应项
若该项缓存存在于软TLB，修改其中的页表项
修改页表内容结束

软TLB的命中率

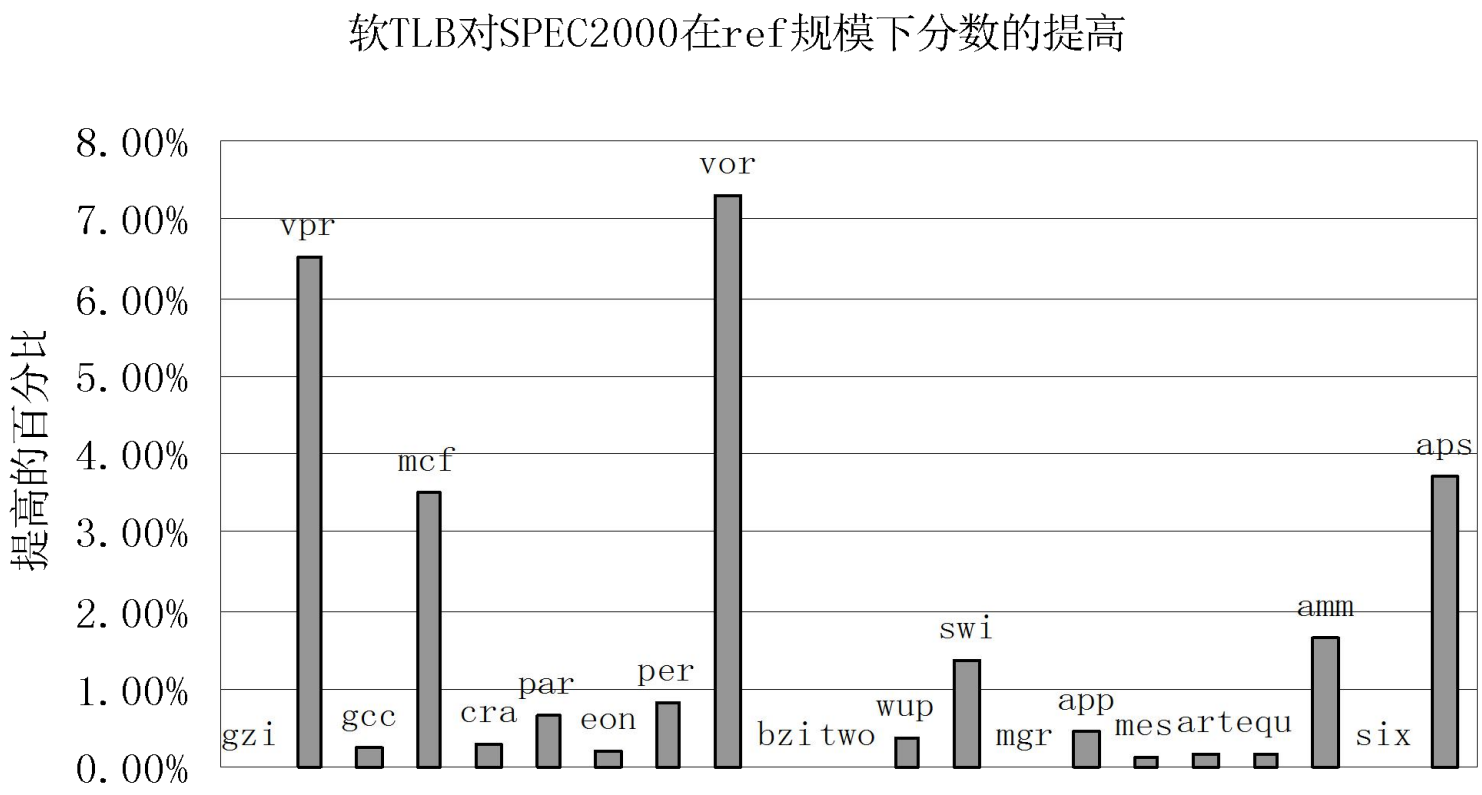
软TLB在ref规模下的命中率（4K的软TLB表项，直接映射）



缓冲区大小对软TLB命中率的影响



软TLB对SPEC性能的提高



常见处理器的结构参数（TLB）

		Intel Ivybridge	AMD BullDozer	IBM Power7	GS464E
前端	一级指令缓存	32KB，8 路，64B/行	64KB，2路，64B/行	32KB，4 路，128B/行	64KB，4 路，64B/行
	指令TLB	128 项，4 路，L1 ITLB	72 项，全相联，L1 ITLB 512 项，4 路，L2 ITLB	64 项，2 路，L1 ITLB	64 项，全相联，L1 ITLB
	分支预取	BTB （8K-16K?项） 间接目标队列 （? 项） RAS （? 项） 循环检测	512 项，4 路 L1 BTB 5120 项，5 路 L2 BTB 512 项 间接目标队列 24 项 RAS 循环检测	8K 项本地 BHT 队列， 16K 项全局BHT 队列， 8K 项 全局 sel 队列 128 项间接目标队列 16 项 RAS	8K项本地 BHT 队列 8K项全局BHT 队列 8K项 全局 sel 队列 1K项间接目标队列 16项 RAS 循环检测
乱序执行	ROB	168 项	128 项	120 项	128 项
	发射队列	54-项 统一	60-项 浮点(共享) 40-项 定点，访存	48-项 标准	32-项 浮点； 32-项 定点； 32-项 访存
	寄存器重命名	160 定点； 144 浮点	96 定点； 160 浮点(双核共享)	80 定点,浮点； 56 CR； 40 XER， 24 Link&Count	128 定点； 128 浮点/向量； 16 Acc； 32 DSPCtrl；32 FCR
运算部件	执行单元	ALU/LEA/Shift/128位 MUL/128位 Shift/256位FMUL/256位Blend + ALU/LEA/Shift/128位 ALU /128bit Shuffle/256位 FADD + ALU/Shift/Branch/ 128位 ALU/128bit Shuffle/256位 Shuffle/256位 Blend	ALU/IMUL/Branch + ALU/IDIV/Count + 128位FMAC/128位 IMAC + 128位FMAC/128位 XBAR + 128位 MMX + 128位 MMX/128位 FSTO	2 定点 + 2 浮点/向量 + 1 转移 + 1 CR	2 定点/转移/DSP + 2 浮点/向量
	向量宽度	256位	128位	128位	256位

常见处理器的结构参数（TLB）

		Intel Ivybridge	AMD BullDozer	IBM Power7	GS464E
访存	访存单元个数	2 取+ 1 存	2 取/存	2 取/存	2 取/存
	Load/Store队列	64-项 Load 队列, 36-项 Store队列	40-项 Load 队列, 24-项 Store 队列	32-项 Load队列, 32-项 Store 队列	64-项 取/存 队列
	Load/Store 宽度	128 位	128 位	256 位 load, 128 位 Store	256 位
	TLB	100项全相联，L1 DTLB, 512项4路，L2 TLB	32项全相联，L1 DTLB, 1024项8路，L2 TLB	64项全相联，L1 DTLB, 512项4路，L2 TLB	32-项全相联L1DTLB，每项两页 1024项8路L2 TLB，每项两页
	L1D	32KB，8 路，64B/行	16KB，4路，64B/行	32KB，8 路，128B/行	每核64KB，4 路，64B/行
	L2	每核256KB，8路，64B/行	双核共享2MB，16 路	每核256KB，8路，28B/行	每核256KB，16路,64B/行
	LLC	8个核20 MB	4个核8 MB	8个核32 MB	4个核4 MB～8MB
	L1失效队列	10	?	8	Unified 16
	L2失效队列	16	23	24	
	L1 Load-to-use	定点4时钟周期, 浮点/向量5时钟周期	4时钟周期	定点2时钟周期, 浮点/向量3 时钟周期	定点3时钟周期, 浮点/向量5 时钟周期
	L2 Load-to-use	12 时钟周期	18-20 时钟周期	8	22 时钟周期
	多层次硬件预取	有	有	有	有

作 业