

计算机学院课程

---

# MIPS体系结构概述

高小鹏

北京航空航天大学计算机学院

# 基本说明

- See MIPS Run Linux
  - 第2.8节：基本地址空间
  - 第3章：CP0
    - ◆ 与PPT相关的内容
  - 第5章：中断
    - ◆ 5.1～5.5, 5.7, 5.8

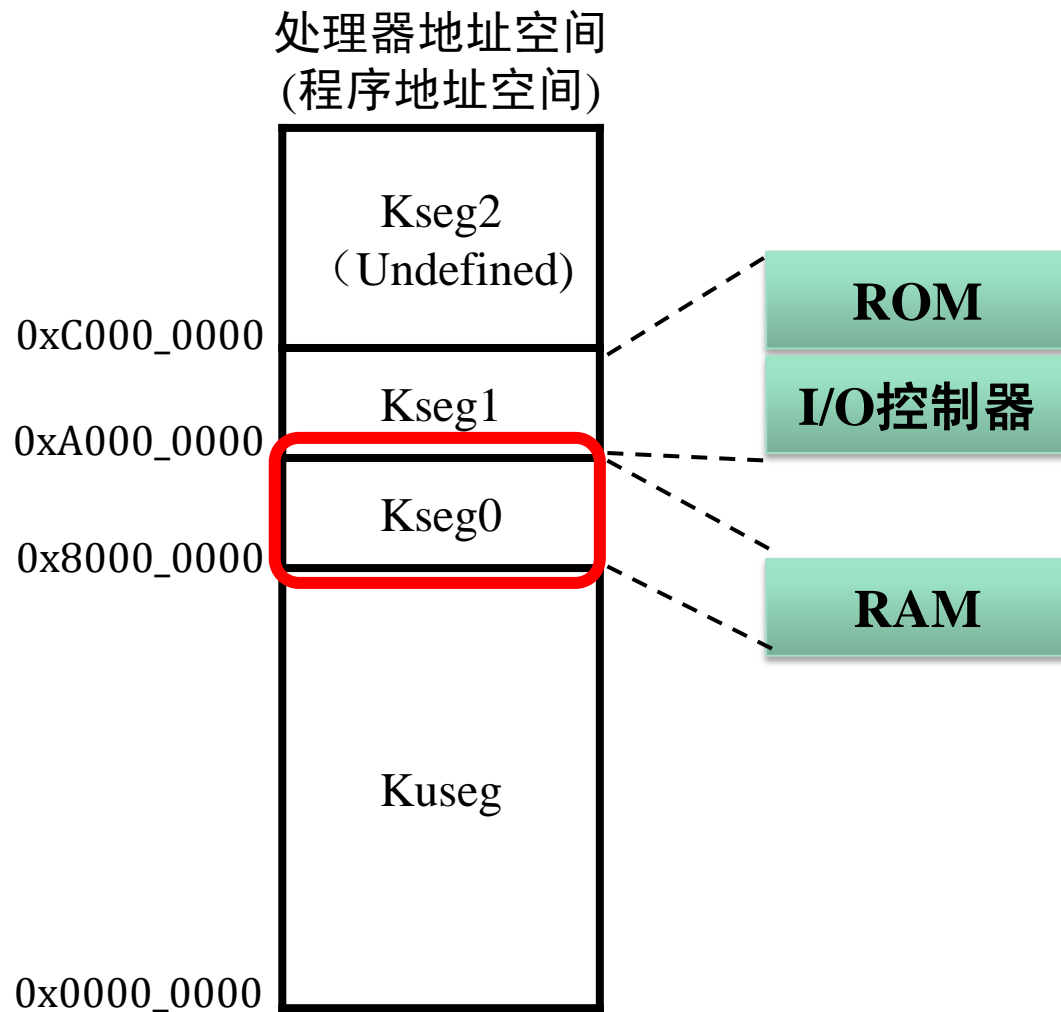
# 提纲

- 存储器及存储器视图
- 乘除法部件
  - MFHI/MFLO, MTHI/MTLO
- 异常/中断
- 协处理器
- 异常/中断通用框架
- 高级话题：多任务切换核心机制

# MIPS存储视图

- kseg0

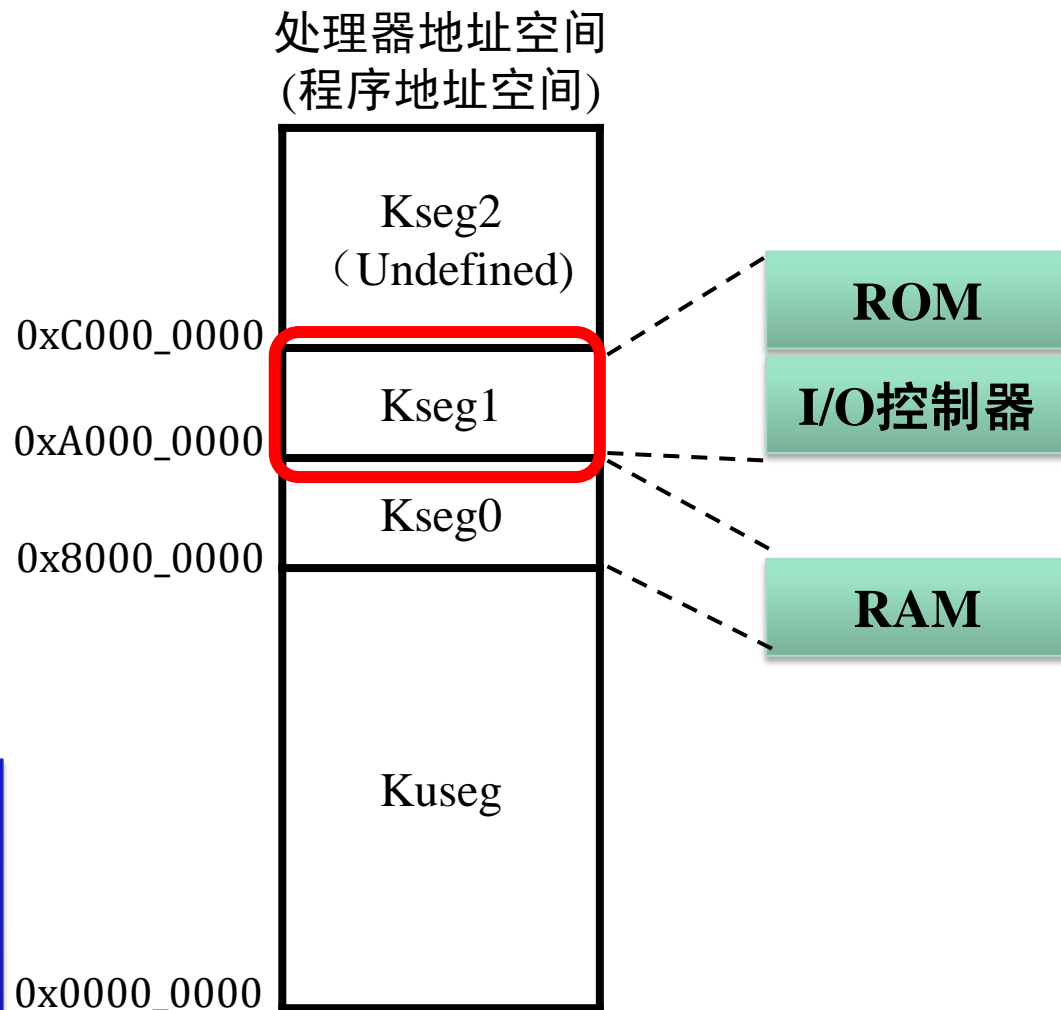
- 512MB空间
- 映射RAM
- 可以被Cache



# MIPS存储视图

## ■ kseg1

- 512MB空间
- 映射ROM与I/O设备
- 不被Cache
- PC复位地址  
0xBFC00000



S/H

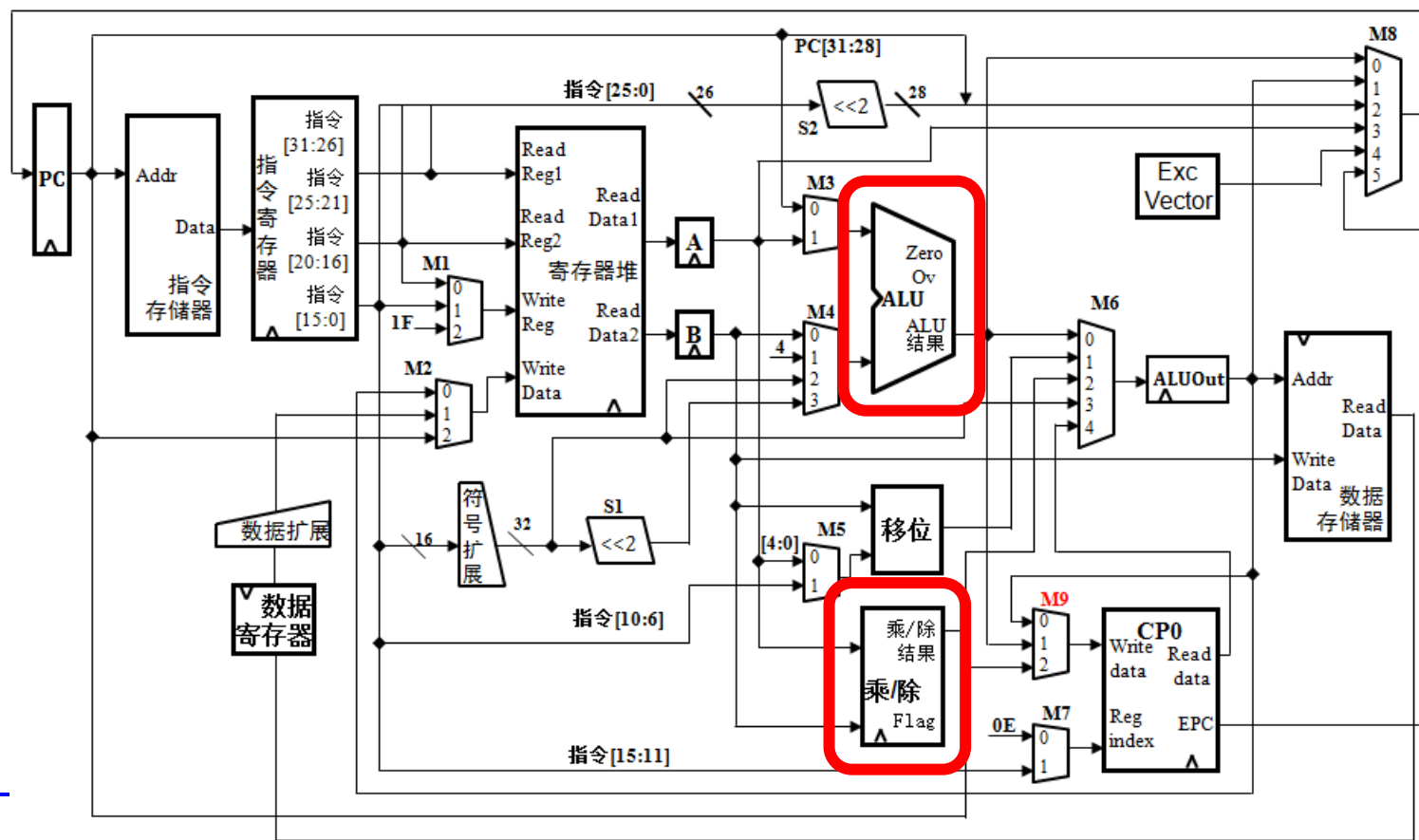
⇒ ROM占用地址空间为  
0xBFC0\_0000~0xBFF  
F\_FFFF, 大小4MB

# 提纲

- 存储器及存储器视图
- 乘除法部件
  - MFHI/MFLO, MTHI/MTLO
- 异常/中断
- 协处理器
- 异常/中断通用框架
- 高级话题：多任务切换核心机制

# 乘法/除法

- 硬件：独立的乘除法部件
  - 不集成在ALU内部
  - 防止慢速乘除运算影响快速ALU



# 乘除法单元接口寄存器及专用指令

- 寄存器：HI、LO
  - 乘法：HI/LO分布存放乘法结果的高/低32位
  - 除法：HI存放余数，LO存放商
- 专用指令：MFHI/MFLO、MTHI/MTLO
  - MFHI/MFLO：将HI/LO传输至GPR
  - MTHI/MTLO：将GPR传输至HI/LO
    - ◆ 唯一用途：中断返回时恢复现场

**S/H**

⇒ 在读出HI和LO之前，不能写入HI和LO  
为什么？



# 乘法指令、除法指令

## ■ 乘法指令

- MULT/MULTU: 将2个操作数送入乘除单元, 并启动乘法计算

## ■ 除法指令

- DIV/DIVU: 将2个操作数送入乘除单元, 并启动除法计算

## ■ 指令特点

- 送操作数, 启动计算
- 结果保存在HI/LO中
- 紧跟的MFHI/MFLO未必能立刻得到结果

**S/H**

⇒ 1)乘除法指令只是写入乘除单元内部寄存器; 2)写入内部寄存器的动作应触发内部计算流程; 3)乘除部件需要设置计算完成标志

# 提纲

- 存储器及存储器视图
- 乘除法部件
  - MFHI/MFLO, MTHI/MTLO
- 异常/中断
- 协处理器
- 异常/中断通用框架
- 高级话题：多任务切换核心机制

# Exceptions and Interrupts

- “Unexpected” events requiring change in flow of control
  - Different ISAs use the terms differently
- *Exception*
  - Arises within the CPU  
(e.g. undefined opcode, overflow, syscall, TLB Miss)
- *Interrupt*
  - From an external I/O controller

# Handling Exceptions (1/2)

- In MIPS, exceptions managed by a System Control Coprocessor (CP0)
- Save PC of offending (or interrupted) instruction
  - In MIPS: save in special register called *Exception Program Counter (EPC)*
- Save indication of the problem
  - In MIPS: saved in special register called *Cause* register
  - In simple implementation, might only need 1-bit (0 for undefined opcode, 1 for overflow)
- Jump to *exception handler code* at address 0x???00180

# Handling Exceptions (2/2)

- Operating system is also notified
  - Can kill program (e.g. segfault)
  - For I/O device request or syscall, often switch to another process in meantime
    - This is what happens on a TLB misses and page faults

# Exception Properties

- Re-startable exceptions
  - Pipeline can flush the instruction
  - Handler executes, then returns to the instruction
    - Re-fetched and executed from scratch
- PC+4 saved in EPC register
  - Identifies causing instruction
  - PC+4 because it is the available signal in a pipelined implementation
    - Handler must adjust this value to get right address

# Handler Actions

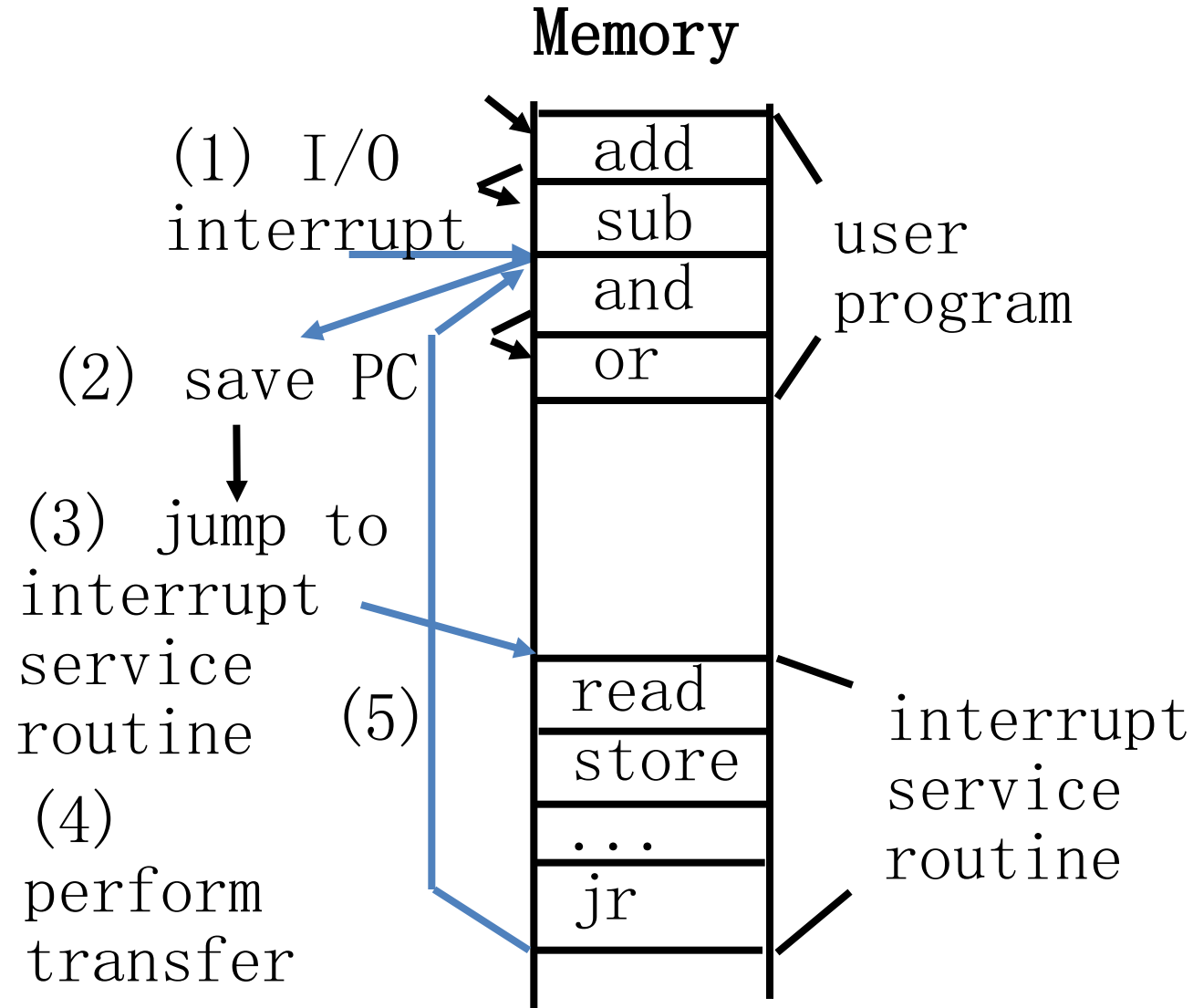
- Read Cause register, and transfer to relevant handler
- OS determines action required:
  - If restartable exception, take corrective action and then use EPC to return to program
  - Otherwise, terminate program and report error using EPC, Cause register, etc.  
(e.g. our best friend the segfault)

# I/O Interrupt

- An I/O interrupt is like an exception except:
  - An I/O interrupt is “asynchronous”
  - More information needs to be conveyed
- “Asynchronous” with respect to instruction execution:
  - I/O interrupt is not associated with any instruction, but it can happen in the middle of any given instruction
  - *I/O interrupt does not prevent any instruction from running to completion*



# Interrupt-Driven Data Transfer



# 异常入口

复位 (仅有 ROM)	0xBFC0 0000	复位及不可屏蔽中断入口点。
ROM 入口点	0xBFC0 0000	中断专用——仅当 Cause(IV) 置位时使用，并非所有的 CPU 都有。
	0xBFC0 0380	所有其它异常。
	0xBFC0 0300	高速缓存错误。
	0xBFC0 0200	简单的 TLB 重填 (SR(EXL)为零)。
RAM 入口	BASE+0x200+...	多个中断入口点—— VI 模式多七个，EIC 模式 62 个。
	BASE+0x200+...	中断特殊/专用 ((Cause(IV)) 为一)。
	BASE+0x180	所有其它异常。
	BASE+0x100	高速缓存错误——在 RAM 中但是永远经过不经高速缓存的 kseg1 窗口。
	BASE+0x000	简单的 TLB 重填 (SR(EXL)为零)。

# 提纲

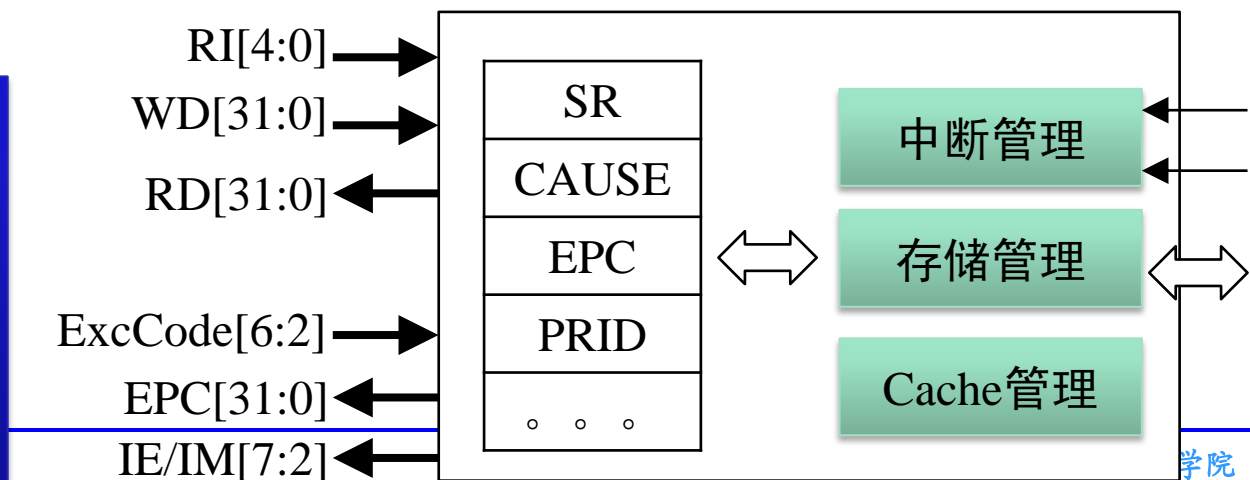
- 存储器及存储器视图
- 乘除法部件
  - MFHI/MFLO, MTHI/MTLO
- 异常/中断
- 协处理器
- 异常/中断通用框架
- 高级话题：多任务切换核心机制

# 协处理器0 (CP0)

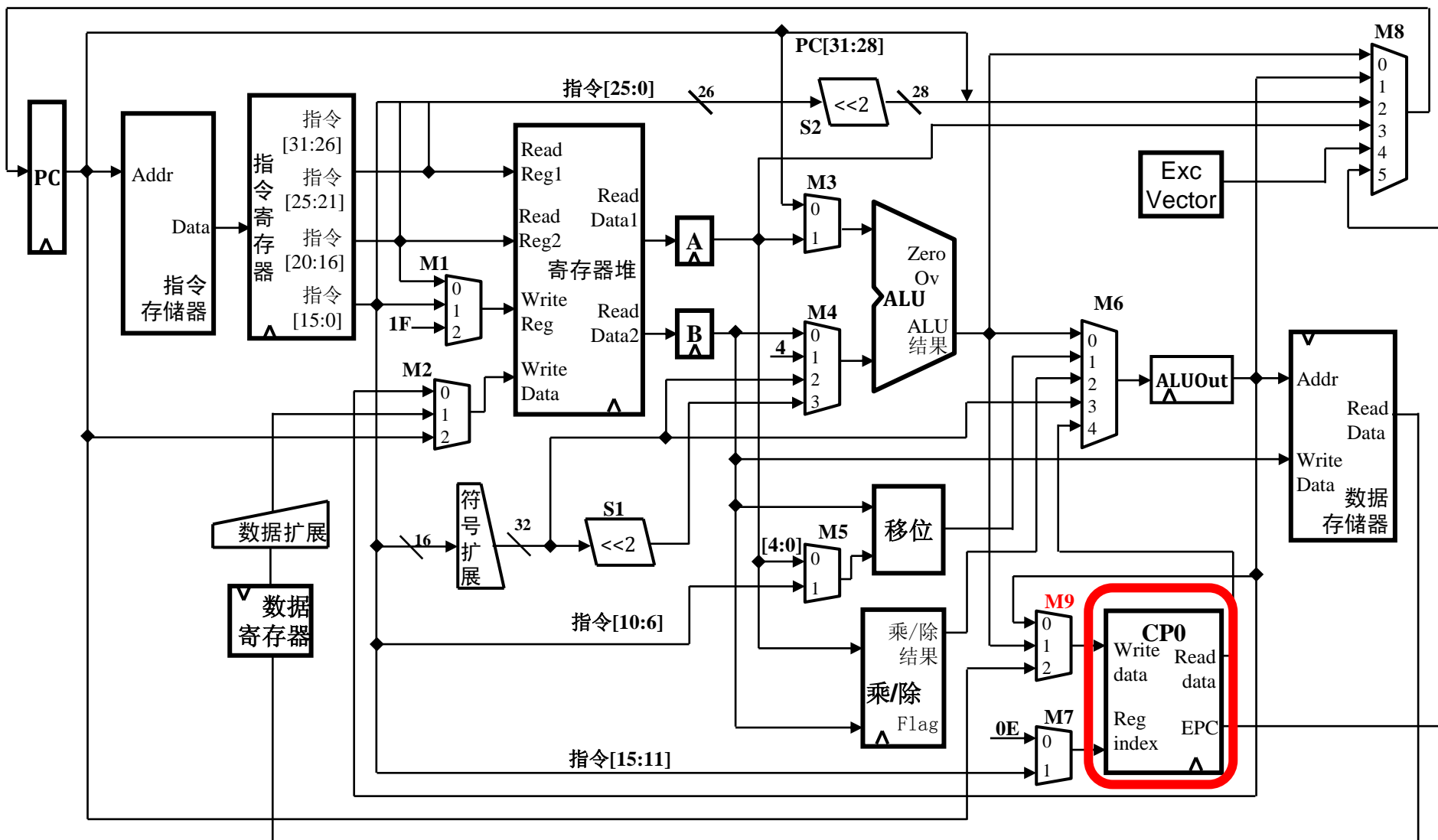
- 阅读内容：《See MIPS Run Linux》第3章
- CP0定位
  - ▣ 为OS的中断响应、存储管理、任务调度等提供必要的功能抽象
- CP0访问
  - ▣ 专用的CP0指令访问CP0内部寄存器：MFC0, MTC0

S/H

⇒ 程序员通过读写CP0寄存器来控制CP0内部功能



# CP0



# CP0重要寄存器

- SR: 用于对系统进行控制
  - 指令可读可写
- Cause: 指令只能读取, 硬件控制写入
  - IP[7:2]: 对应外部6个中断源
  - ExcCode: 异常/中断发生时控制器控制写入编码值
- EPC: 用于保存异常/中断发生时的PC
  - 保存PC: 硬件控制写入
  - 指令读取: 中断服务程序
- PRId: 处理器ID
  - 可以用于实现个性的编码☺

# SR: 状态寄存器

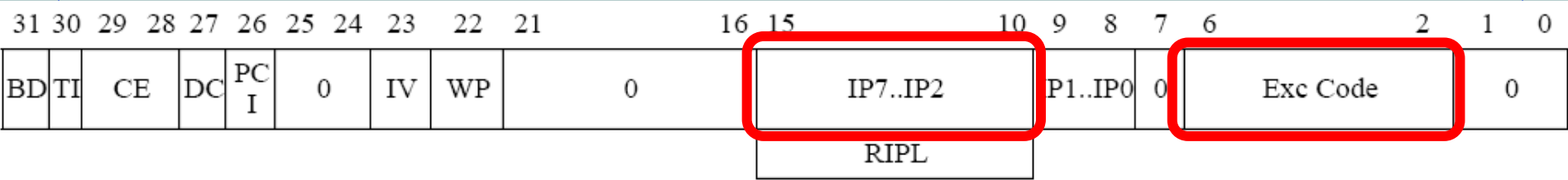
31	28	27	26	25	24	23	22	21	20	19	18	17	16	15	10	9	8	7	6	5	4	3	2	1	0
CU3..CU0	RP	FR	RE	MX	PX	BEV	TS	SR	NMI	0	Impl	IM7..IM2			IM1..IM0			KX	SX	UX	UM	R0	ER	EXL	IE

IPL

KSU

位含义	位序	描述	R/W	复位初值
IM7	15	硬件中断7中断屏蔽掩码。控制是否允许相应硬件中断。 0: 不允许中断; 1: 允许中断	R/W	未定义
IM6	14	硬件中断6中断屏蔽掩码。控制是否允许相应硬件中断。 0: 不允许中断; 1: 允许中断	R/W	未定义
IM5	13	硬件中断5中断屏蔽掩码。控制是否允许相应硬件中断。 0: 不允许中断; 1: 允许中断	R/W	未定义
IM4	12	硬件中断4中断屏蔽掩码。控制是否允许相应硬件中断。 0: 不允许中断; 1: 允许中断	R/W	未定义
IM3	11	硬件中断3中断屏蔽掩码。控制是否允许相应硬件中断。 0: 不允许中断; 1: 允许中断	R/W	未定义
IM2	10	硬件中断2中断屏蔽掩码。控制是否允许相应硬件中断。 0: 不允许中断; 1: 允许中断	R/W	未定义
EXL	1	异常级。任何异常发生时置位, 并且禁止中断。目的是把EXL位维持足够长时间以便软件决定中断屏蔽应设成什么。 0: 无异常发生; 1: 有异常发生	R/W	未定义
IE	0	全局中断使能。 0: 不允许中断; 1: 允许中断	R/W	未定义

# Cause: 原因寄存器

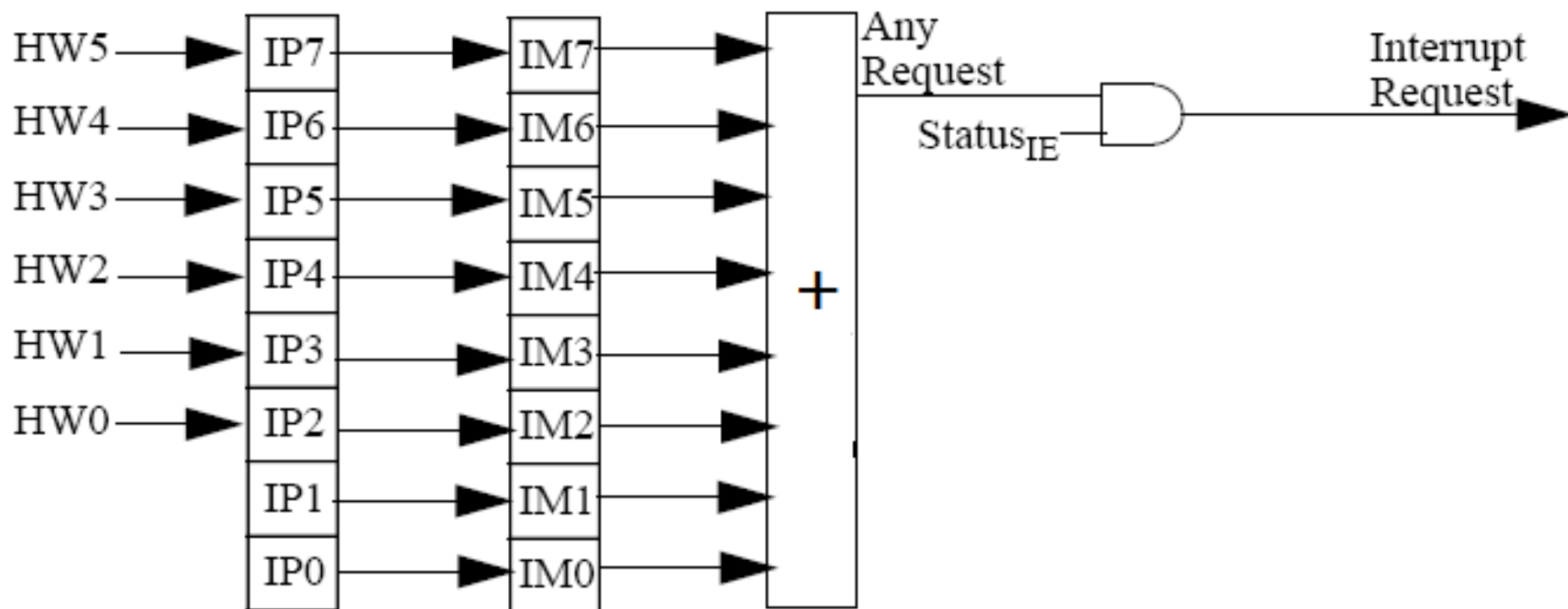


位含义	位序	描述	R/W	复位初值
IP7	15	指示发生的硬件中断7处于pending状态	R	未定义
IP6	14	指示发生的硬件中断6处于pending状态	R	未定义
<div><u>S/H</u> ⇒ 控制器产生ExcCode编码 ⇒ 异常服务程序读取ExcCode以判断具体原因</div>				未定义
ExcCode		<div><u>S/H</u> ⇒ pending: 外部中断仍保持有效</div> 10: 保留指令异常		未定义



# 简单的中断硬件机制

- Interrupt Request送入控制器
- 不设置优先级排队硬件
  - 中断服务程序决定应该优先响应哪个中断



# EPC: 异常返回地址寄存器

31

0

EPC

- 中断：硬件中断
  - EPC中保存的是遭受中断的指令的后继指令的地址
  - 中断返回时，不需要修改EPC的值
- 异常：溢出、除零、system call、break、未定义指令
  - EPC中保存的则是当前指令的地址
  - 因此从异常返回时，需要将EPC的值加4

## S/H

⇒ 发生异常/中断时，硬件自动完成PC→EPC

⇒ 异常/中断退出最后一条指令 ERET，完成  
EPC→PC

# PRId: 异常返回地址寄存器

31	24	23	16	15	8	7	0
Company Options		Company ID		Processor ID		Revision	

- 只读寄存器
- 我们自由使用☺
  - 0x42554141
  - 0x4E554141

# CP0指令

- 指令：MFC0、MTC0、ERET

- MFC0

- MFC0 rt, rd:  $GPR[rt] \leftarrow CP0[rd]$
- SR: 获取处理器的控制信息
- Cause: 获取处理器当前所处的状态
- EPC: 获取被异常/中断的指令地址
- PRId: 读取处理器ID（可以读取你的个性签名😊）

- MTC0

- MTC0 rt, rd:  $CP0[rd] \leftarrow GPR[rt]$
- SR: 对处理器进行控制，例如关闭中断
- EPC: 操作系统中将用于多任务切换

# CP0指令

- ERET: 从异常/中断中返回
  - 返回异常受害指令:  $PC \leftarrow EPC$
  - 允许异常/中断产生:  $CP0.SR[EXL] \leftarrow 0$

# 提纲

- 存储器及存储器视图
- 乘除法部件
  - MFHI/MFLO, MTHI/MTLO
- 异常/中断
- 协处理器
- 异常/中断通用框架
- 高级话题：多任务切换核心机制

# 异常/中断通用例程框架

- 曾老师的代码

# 提纲

- 存储器及存储器视图
- 乘除法部件
  - MFHI/MFLO, MTHI/MTLO
- 异常/中断
- 协处理器
- 异常/中断通用框架
- 高级话题：多任务切换核心机制



# 高级话题：多任务切换的核心机制

- 示例：P1和P2均执行无限循环，CPU在定时器中断驱动下轮流切换，实现P1和P2的时间片调度
  - 定时器固定间隔发出硬件中断
  - 中断服务程序完成切换
- 解决方案
  - 思路：eret返回前，先替换EPC为某个特定现场
  - 进入中断服务程序后，读取EPC并保存在对应任务的M\_EPC变量中
  - 读取另一任务的M\_EPC至EPC
  - ERET返回