中断与异常设计实现

ZJUNIX

目录

MIPS32异常机制简介

- 异常种类
- 系统调用
- 中断
- 异常向量表

异常种类

- MIPS32规范规定了数十种异常(exception)
  - 中断(interrupt)属于一种异常
- 大多数异常使CPU跳转到同一个地址(0x0180)
  - 有少数例外，例如中断(0x0200)
- 通过COP0(协处理器0)的寄存器可以获得异常的种类
- 常见异常种类：
  - 中断(interrupt)
  - 系统调用(syscall)

系统调用

- 系统调用(syscall)是应用程序和操作系统之间的重要接口
- 系统调用异常由syscall指令产生
  - 内核态下执行syscall指令也会产生该异常
- 系统调用属于一种一般性异常(General exception)
  - 由于很多其他异常也使用该异常向量，处理前需要判断异常种类

中断

- 中断(interrupt)由指令序列以外的外部事件触发
- 驱动程序和进程调度系统都依赖于中断
- 中断属于一种异常，但可以使用专用的异常向量
- 所有中断使用同一个异常向量，因此需要判断中断号
  - MIPS32规范规定了6个中断
  - 其中一个中断用于内部定时器：进程调度依赖于此中断

异常向量表

- 异常向量与COP0部分寄存器的设置有关

Exception	Status[BEV]	Status[EXL]	Cause[IV]	Vector
TLB Refill	0	0	X	0x8000 0000
	0	1	X	0x8000 0180
	1	0	X	0xBFC0 0200
	1	1	X	0xBFC0 0380
Interrupt	0	0	0	0x8000 0180
	0	0	1	0x8000 0200
	1	0	0	0xBFC0 0380
	1	0	1	0xBFC0 0400
General exception	0	X	X	0x8000 0180
	1	X	X	0xBFC0 0380

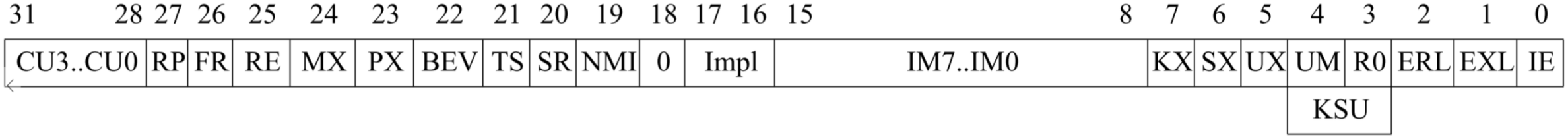
- ZJUNIX系统将配置COP0寄存器使用以上异常向量

异常机制初始化

- 相关COP0寄存器
- 初始化代码

相关COP0寄存器

- Status寄存器：12号COP0寄存器



- 所有位全部清零
  - BEV=0: 异常向量基址0x80000000
  - IM=0x00: 屏蔽所有中断(注册中断处理函数时打开对应中断)
  - UM=0: 处理器处于内核态
  - ERL=0, EXL=0: 清除所有异常状态
  - IE=0: 屏蔽所有中断

相关COP0寄存器

- Cause寄存器：13号COP0寄存器

31	30	29 28	27 24	23	22	21 16	15 8	7	6 2	1 0
BD	0	CE	0	IV	WP	0	IP7:IP0	0	Exc Code	0

- IV=1: 中断使用0x8000 0200异常向量
- Exc Code: 异常号，只读
- IP: 中断号
  - IP0和IP1是软件中断
  - IP2至IP7对应6个外部中断，只读

初始化代码

- 使用内联汇编对COP0寄存器进行初始化
- 12号寄存器(Status)=0x0000 0000
- 13号寄存器(Cause)=0x0080 0000

```
asm volatile(
    "mtc0 $zero, $12\n\t"
    "li $t0, 0x800000\n\t"
    "mtc0 $t0, $13\n\t"
);
```

arch/mips32/exc.c

异常入口及处理流程

- 异常入口与出口
- 异常处理流程
- 系统调用处理流程
- 中断处理流程

异常入口与出口

- 异常和中断入口需要保护异常发生时的上下文
  - 主要包括CPU内的通用寄存器
- 还需要根据发生异常时处于用户态/内核态相应调整堆栈
  - 如果异常发生时处于用户态，则将sp寄存器指向内核栈
- 异常出口处恢复上下文

异常入口与出口

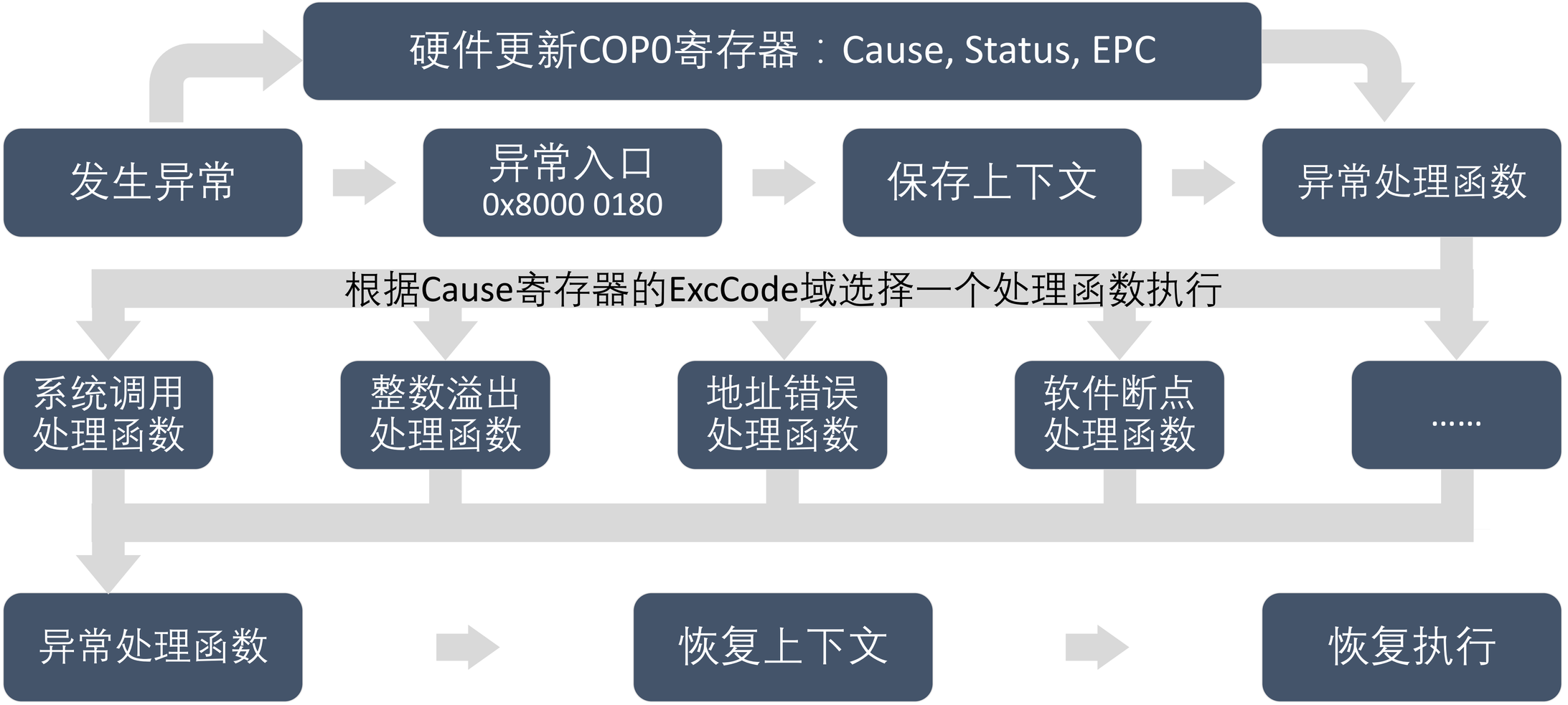
```
.org 0x0180
        lui $k0, 0x8000
        sltu $k0, $sp, $k0
        beq $k0, $zero, exception_save_context
        move $k1, $sp
        la  $k0, kernel_sp
        j exception_save_context
        lw  $sp, 0($k0)

.org 0x0200
        lui $k0, 0x8000
        sltu $k0, $sp, $k0
        beq $k0, $zero, interrupt_save_context
        move $k1, $sp
        la  $k0, kernel_sp
        lw  $sp, 0($k0)
```

```
interrupt_save_context:
        addiu $sp, $sp, -128
        sw $at, 4($sp)
        ...
        sw $ra, 124($sp)
        move $a2, $sp
        addi $sp, $sp, -32
        jal do_interrupts
        nop
        addi $sp, $sp, 32
        lw $at, 4($sp)
        ...
        lw $ra, 124($sp)
        move $sp, $k1
        eret
```

arch/mips32/start.s

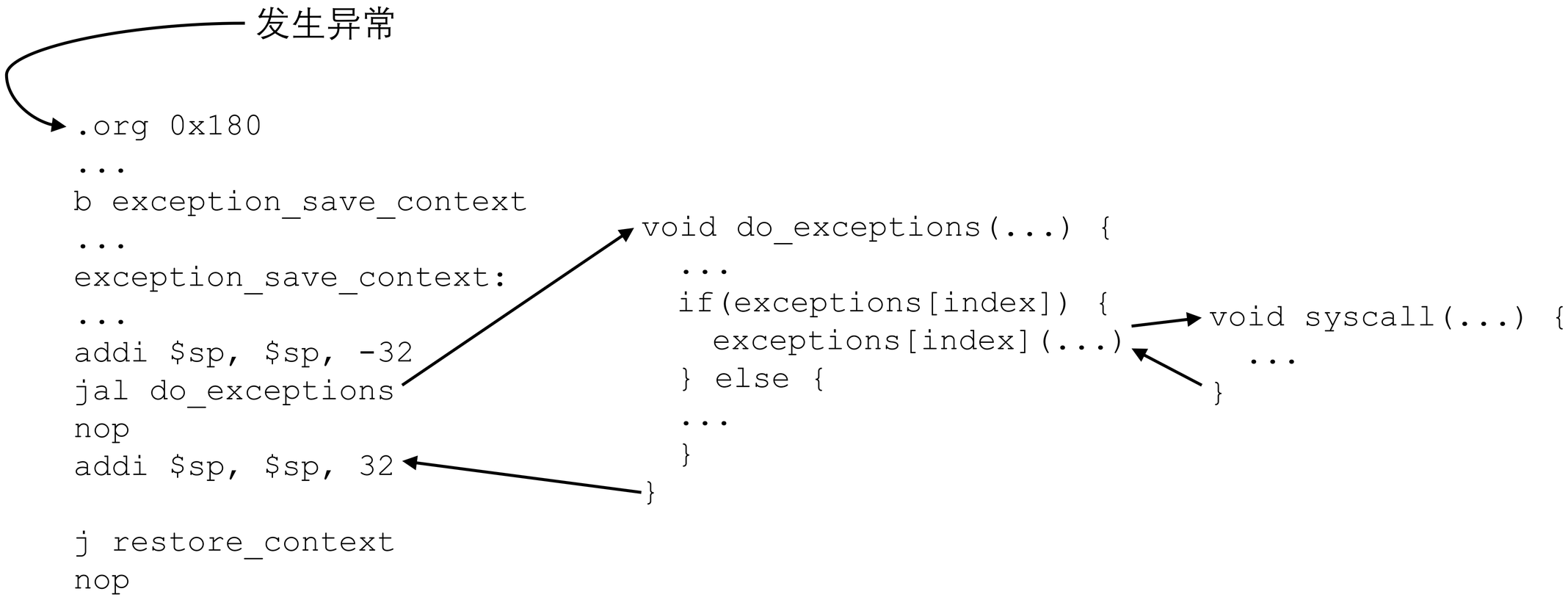
异常处理流程


硬件更新COP0寄存器：Cause, Status, EPC
发生异常
异常入口
0x8000 0180
保存上下文
异常处理函数
根据Cause寄存器的ExcCode域选择一个处理函数执行
系统调用
处理函数
整数溢出
处理函数
地址错误
处理函数
软件断点
处理函数
……
异常处理函数
恢复上下文
恢复执行

ZJUNIX

异常处理流程

以系统调用为例



arch/mips32/start.s　　arch/mips32/exc.c　　kernel/syscall.c

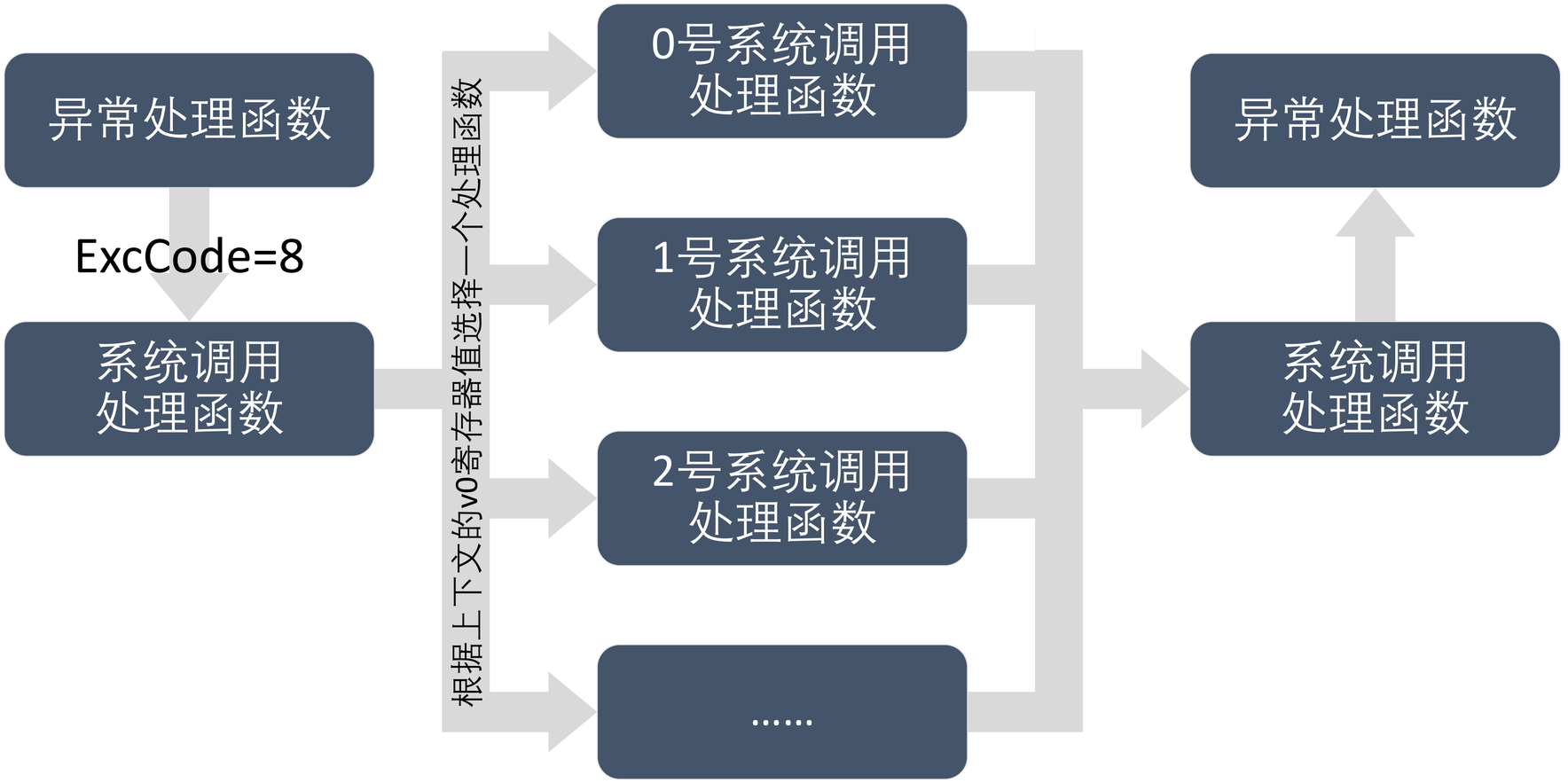
异常处理流程

- 异常处理函数：判断异常种类，调用对应的处理函数

```
typedef void (*exc_fn)(unsigned int, unsigned int, unsigned int*);
...
exc_fn exceptions[32];
...
void do_exceptions(unsigned int status,unsigned int cause,unsigned int *context) {
    int index = cause >> 2;
    index &= 0x1f;
    if (exceptions[index]) {
        exceptions[index](status, cause, context);
    } else {
        ...
    }
}
```

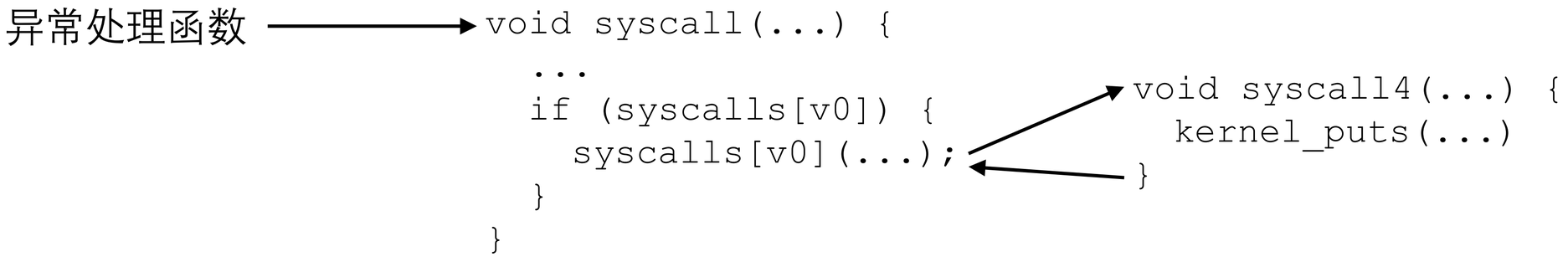
代码文件：arch/mips32/exc.c

系统调用处理流程


异常处理函数
ExcCode=8
系统调用
处理函数
根据上下文的v0寄存器值选择一个处理函数
0号系统调用
处理函数
1号系统调用
处理函数
2号系统调用
处理函数
……
异常处理函数
系统调用
处理函数

ZJUNIX

系统调用处理流程



kernel/syscall.c

kernel/syscall.c

系统调用处理流程

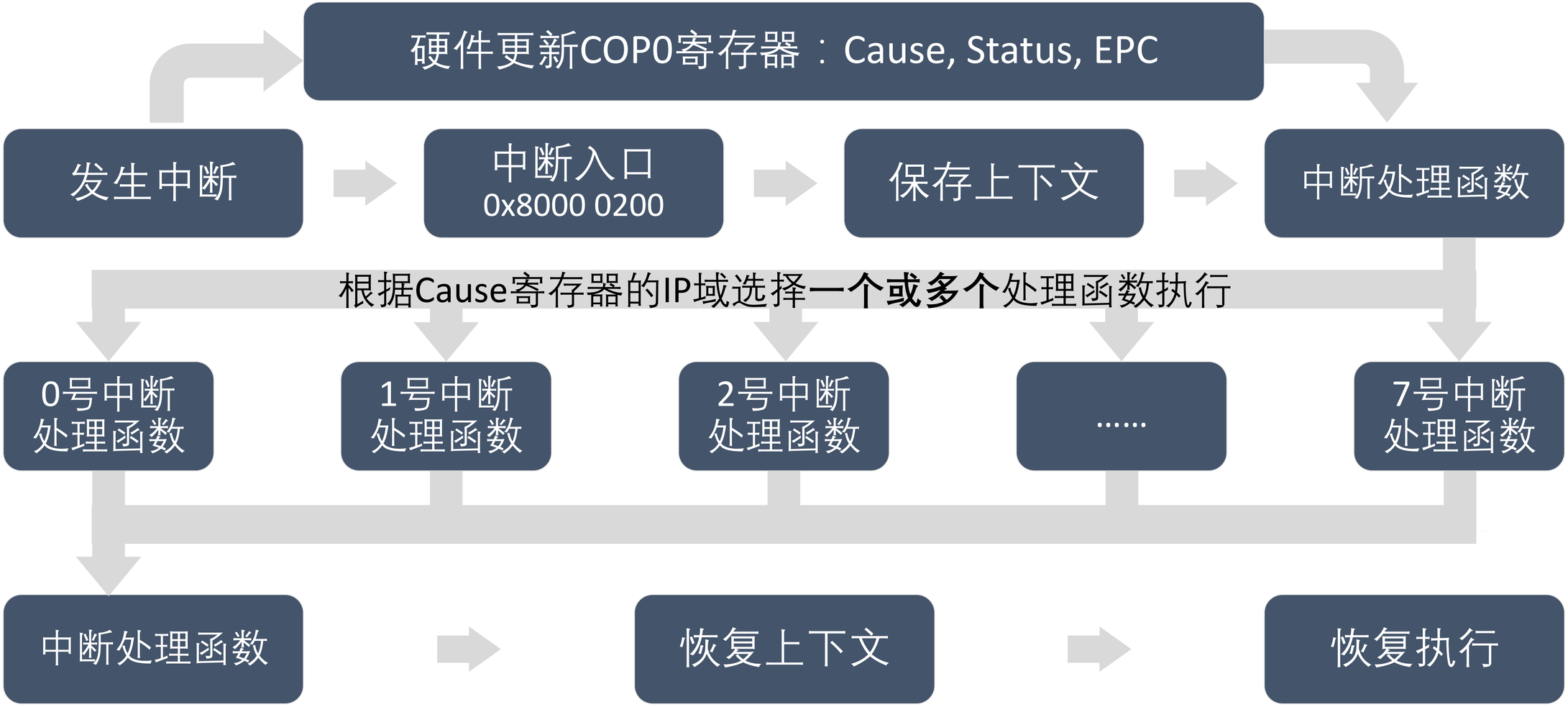
- 系统调用属于一种异常，对应的异常号为8
- 系统调用号保存在通用寄存器v0中
  - 通过异常上下文获取该值

```
typedef void (*sys_fn)(unsigned int, unsigned int, unsigned int, unsigned int);
...
sys_fn syscalls[256];
...
void syscall(unsigned int status, unsigned int cause, unsigned int* context) {
    unsigned int v0 = context[2] & 255;
    context[0] += 4;
    if (syscalls[v0]) {
        syscalls[v0](context[4], context[5], context[6], context[7]);
    }
}
```

代码文件：kernel/syscall.c

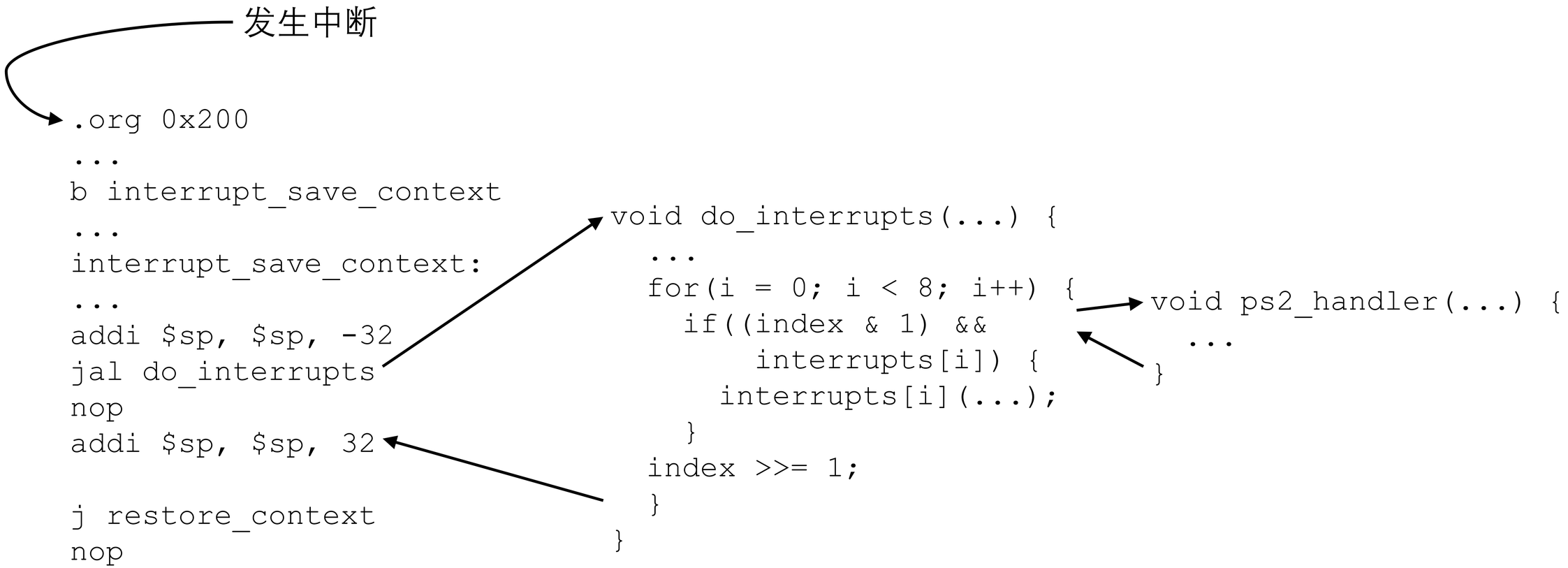
ZJUNIX

中断处理流程



ZJUNIX

中断处理流程



arch/mips32/start.s

arch/mips32/intr.c

kernel/driver/ps2.c

中断处理流程

- 中断处理函数：判断中断号，调用相应处理函数

```
typedef void (*intr_fn)(unsigned int, unsigned int, unsigned int*);
...
intr_fn interrupts[8];
...
void do_interrupts(unsigned int status,unsigned int cause,unsigned int *context) {
    int i;
    int index = cause >> 8;
    for (i = 0; i < 8; i++) {
        if ((index & 1) && interrupts[i] != 0) {
            interrupts[i](status, cause, context);
        }
        index >>= 1;
    }
}
```

代码文件：arch/mips32/intr.c

THANK YOU