LAB3 Keypoints

OS LAB

2008-4-16

主要内容

0011

- 进程（ENVIRONMENT）
- 中断和异常
- 系统调用
- GCC内联汇编简介

进程

- 定义：Process

  进程是具有独立功能的程序关于某个数据集合上的一次运行活动，是系统进行资源分配和调度的独立单位，又称任务（Task）

- lab3中对应于：Environment

```
struct Env {
        struct Trapframe env_tf;        // Saved registers
        LIST_ENTRY(Env) env_link;       // Free list link pointers
        envid_t env_id;                 // Unique environment identifier
        envid_t env_parent_id;          // env_id of this env's parent
        unsigned env_status;            // Status of the environment
        uint32_t env_runs;              // Number of times environment has run

        //  Address space
        pde_t *env_pgdir;       // Kernel virtual address of page dir
        physaddr_t env_cr3;             // Physical address of page dir

};
```

进程的管理

- 系统为了**管理进程**设置一个专门的数据结构：**进程控制块（PCB：Process Control Block）**，用它来记录进程的外部特征，描述进程的运动变化过程（又称进程描述符、进程属性）
- Lab3中使用数据结构**Env作为PCB**

```
struct Env *envs = NULL;          // All environments
struct Env *curenv = NULL;        // The current env
static struct Env_list env_free_list;    // Free list
```

进程的管理(续)

- PCB的内容
  - （1）进程描述信息（PID等）
  - （2）进程控制信息（当前状态、优先级等）
  - （3）所拥有的资源和使用情况（打开文件等）
  - （4）CPU现场保护信息（寄存器信息、段页指针等）
- Lab3的Env结构中定义的内容

（1）

```
envid_t env_id;
envid_t env_parent_id;
```

（2）

```
unsigned env_status;        // Status of the environment
uint32_t env_runs;          // Number of times environment has ru
```

（4）

```
struct Trapframe env_tf;    // Saved registers
// Address space
pde_t *env_pgdir;           // Kernel virtual address of page dir
physaddr_t env_cr3;         // Physical address of page dir
```

进程的管理(续)

- PCB表
  - 系统把所有PCB组织在一起，并把它们放在内存的固定区域，就构成了PCB表
  - PCB表的大小决定了系统中最多可同时存在的进程个数，称为系统的并发度
- Lab3中的Environment数组
  - 等价于PCB表
  - 由envs指针指向，共有1024（NENV）个表项，即JOS系统并发度为1024
  - 在*i386_vm_init()*中完成Environment数组的内存分配和映射

进程状态

- 进程（process）的三种基本状态
  - 运行态、就绪态、等待态
  - 进程在消亡前处于且仅处于三种基本状态之一
- Lab3中的进程（Env）状态
  - unsigned env_status; //在Env结构中定义
  - 三个状态：FREE, RUNNABLE, NOT_RUNNABLE

```
// Values of env_status in struct Env
#define ENV_FREE        0
#define ENV_RUNNABLE    1
#define ENV_NOT_RUNNABLE 2
```

进程映像（要素）

- 进程（process）要素
  - 代码段（用户程序）
  - 数据段（用户数据）
  - 用户栈（堆栈）
  - 进程控制块PCB （进程属性）
- ELF可执行文件格式
  - Lab3中加载一个用户进程对应的代码和数据时读取的对象是ELF格式文件
  - 在Lab3中*由于没有文件系统*，因此ELF可执行映像是内嵌在内核中的
  - 在Lab3中加载ELF二进制映像主要由load_icode()完成

进程的地址空间

- 在JOS中，每个进程都有**4GB的虚拟地址空间**，其中的**内核部分**都是相同的
- 在进程运行之前，kernel要为其设置好地址空间，也就是建立相应的**页目录表和页表**
- 进程描述符Env结构的**env_pgdir**就指向该进程的***页目录表***

进程的地址空间（续）

- JOS中，*env_setup_vm()*为进程初始化虚拟地址空间
  - 以boot_pgdir**为模板**分配页目录表
  - **修改**进程地址空间的内核部分
- *segment_alloc（）*为进程**分配物理页**，并且完成向虚拟地址空间的**映射**

进程环境的创建和运行

- env_create()完成进程环境的创建
  - env_alloc（）分配一个Env结构
  - load_icode（）加载ELF映像
    - segment_alloc（）
- env_run() 完成进程环境的运行
  - *env_pop_tf*（）进程上下文切换

进程运行前的调用图

- 用户代码执行前的**调用图**

- start (kern/entry.S)
- i386_init
  - cons_init
  - i386_detect_memory
  - i386_vm_init
  - page_init
  - env_init
  - idt_init (still incomplete at this point)
  - env_create
  - env_run
    - env_pop_tf

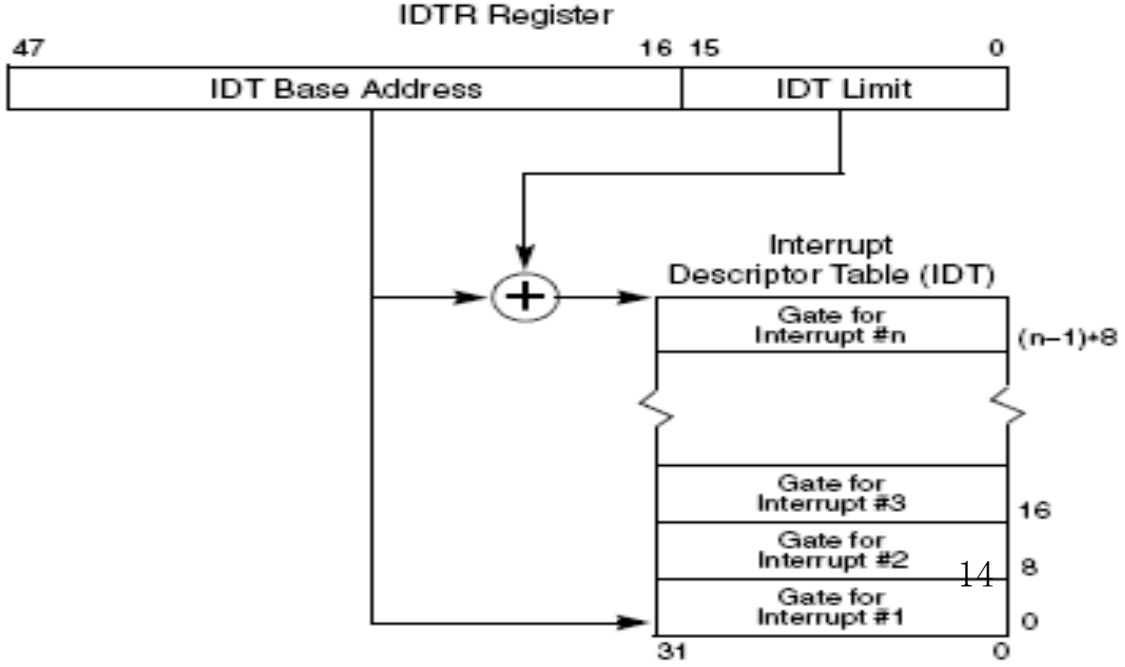
- 通过该调用图掌握JOS Kernel启动流程

中断和异常

- 一个进程的执行不能对**内核（kernel）和其他进程**产生干扰
- 当一个**系统调用（syscall）**发生时，处理器要从用户态**切换**到内核态
- **中断和异常**是**保护控制转移**机制
- 在X86体系下，保护控制转移由两个特殊的机制实现：IDT&TSS

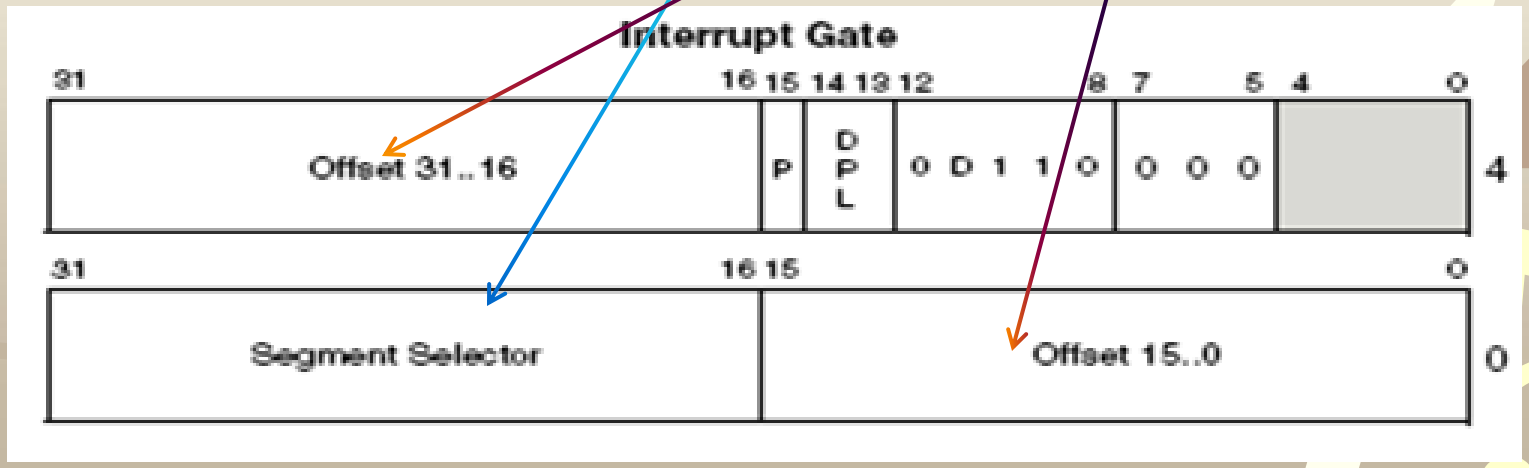
中断和异常

- 中断描述符表(IDT)
  - IDT中每个表项存放一个**门描述符**
  - **门描述符**就是每一个中断或异常处理程序的入口地址



中断和异常

- 中断门(Interrupt Gate)
  - 指向目标代码：SegSelector＋Offset
  - 权限：DPL
  - 具体定义见inc/mmu.h 的 struct Gatedesc



中断和异常

- X86允许256种不同的中断或者异常入口
- **中断向量**
  - 每个中断和异常都由一个唯一的整数值标识，称之为**中断向量**（*Interrupt Vector*）
  - 中断向量由**中断源**提供
  - 中断向量被CPU用来作为**IDT的索引**访问对应的门描述符
  - 跟GDT一样，IDT也被安装在Kernel私有区域
  - 在JOS中IDT表的初始化： trapentry.S和idt_init(void)

中断和异常

- 中断或异常的特权级别
  - 由中断门描述符中的DPL约束了可以通过该门的优先级别
  - 门描述符中的段选择子（Seg Selector）中的CPL说明中断处理程序运行的特权级别
  - JOS中异常都是在内核模式下处理的
  - 门描述符中的DPL的设置在trapentry.S中
  - PartB前的Questions 2

中断和异常

- 任务状态段（TSS）
  - TSS是一个**特殊的数据结构**，一个任务的所有状态信息存储在其中
  - TSS是个段，由**TSS段选择子**定义
  - 定义在mmu.h

```
struct Taskstate {
    uint32_t ts_link;   // Old ts selector
    uintptr_t ts_esp0; // Stack pointers
    uint16_t ts_ss0;    //  after an incre
    uint16_t ts_padding1;
    uintptr_t ts_esp1;
    uint16_t ts_ss1;
    uint16_t ts_padding2;
    uintptr_t ts_esp2;
    uint16_t ts_ss2;
    uint16_t ts_padding3;
    physaddr_t ts_cr3;// Page directory
    uintptr_t ts_eip;   // Saved state fr
    uint32_t ts_eflags;
    uint32_t ts_eax;   // More saved sta
    uint32_t ts_ecx;
    uint32_t ts_edx;
```

31	15	0	
I/O Map Base Address		T	100
	LDT Segment Selector		96
	GS		92
	FS		88
	DS		84
	SS		80
	CS		76
	ES		72
EDI			68
ESI			64
EBP			60
ESP			56
EBX			52
EDX			48
ECX			44
EAX			40
EFLAGS			36
EIP			32
CR3 (PDBR)			28
	SS2		24
ESP2			20
	SS1		16
ESP1			12
	SS0		8
ESP0			4
	Previous Task Link		0

Reserved bits. Set to 0.

中断和异常

- 处理器需要一个地方来**保存旧的处理器状态**，以便在中断返回时**恢复**以前的工作，即**TSS**
- 当X86处理器发生中断或者陷入时，切换到**内核区域的一个堆栈，具体由TSS指出**
- 尽管TSS作用很大，但是JOS只是用它来定义**内核堆栈**

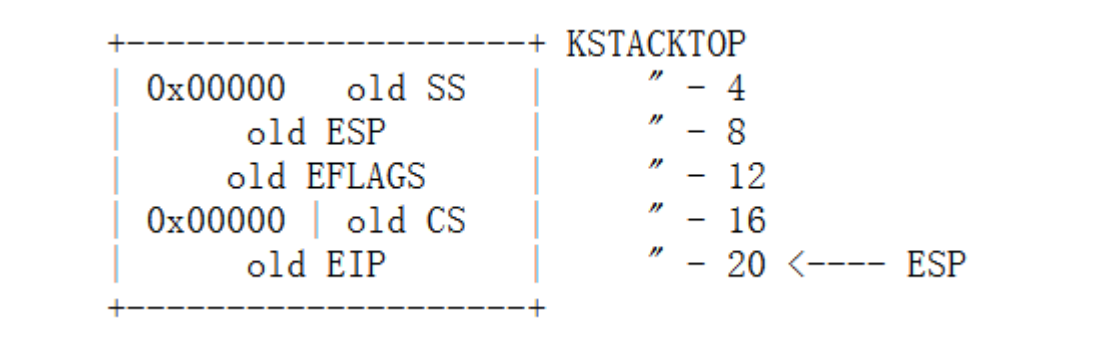
```
// Setup a TSS so that we get the right stack
// when we trap to the kernel.
ts.ts_esp0 = KSTACKTOP;
ts.ts_ss0 = GD_KD;
```

中断和异常

- JOS中断映射布局
  - 内部处理器**异常**： IDT[0--31]
  - 31以上的中断仅被用于**软件中断**或者异步**硬件中断**
  - 系统调用（T_SYSCALL）:**int $0x30**
- **堆栈切换**
  - 内核模式下的异常处理：**将异常参数压到当前堆栈**
  - 用户模式下的异常/中断处理：切换到TSS中SS0，ESP0**所指的堆栈**

中断和异常

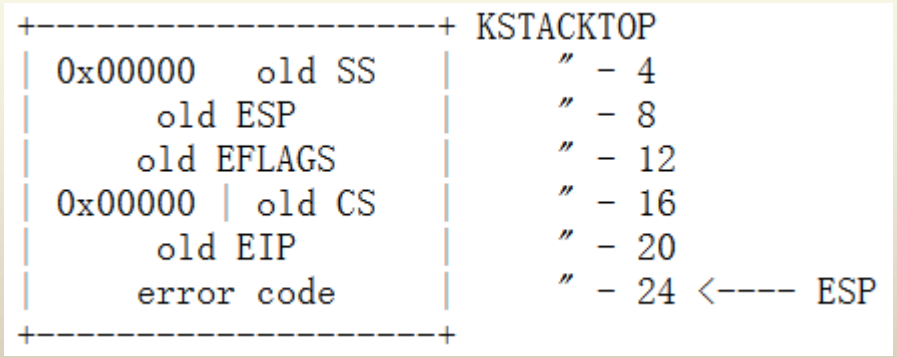
- 发生在用户模式下的除零中断(Int0)实例分析
  - 切换到TSS中SS0(GD_KD)，ESP0 (KSTACKTOP)所指的堆栈
  - 在**内核堆栈**压入必要信息



  - **设置%CS:%EIP**使其指向IDT中0号中断对应中断门中保存的中断处理函数地址
  - 由除零**中断处理函数**处理该中断并返回

中断和异常

- 对于某些X86异常，除了上述5个字段外，有时会加上一个error code



- 比如，缺页异常（page fault）

中断和异常

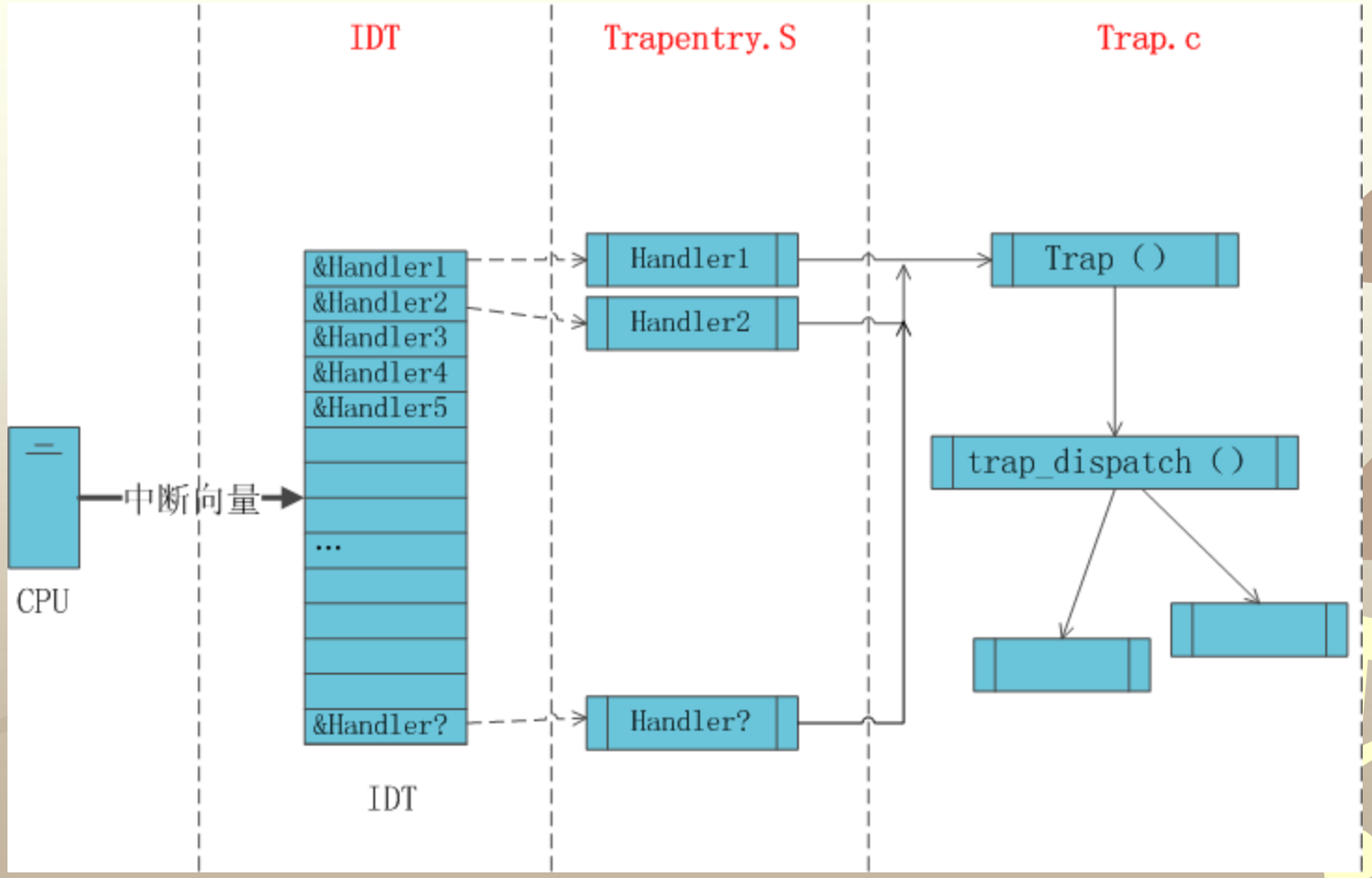
0011

Table 9-7. Error-Code Summary

Description	Interrupt Number	Error Code
Divide error	0	No
Debug exceptions	1	No
Breakpoint	3	No
Overflow	4	No
Bounds check	5	No
Invalid opcode	6	No
Coprocessor not available	7	No
System error	8	Yes (always 0)
Coprocessor Segment Overrun	9	No
Invalid TSS	10	Yes
Segment not present	11	Yes
Stack exception	12	Yes
General protection fault	13	Yes
Page fault	14	Yes
Coprocessor error	16	No
Two-byte SW interrupt	0-255	No

中断和异常

- 中断过程的**控制流**



中断和异常

- Part B前的Questions 1
- JOS中处理中断时，每个中断都有一个Handler，作用如下：
  - 区分不同的中断，中断发生时候，cpu并不传递中断号，只能通过其调用的Handler来区分中断号
  - 为了让栈中所存的数据的格式满足Trapframe结构，方便Handler统一调用trap（）函数；每个Handler会根据对应的中断是否有error code入栈来压入一个伪error code来保证压入堆栈的数据满足统一格式

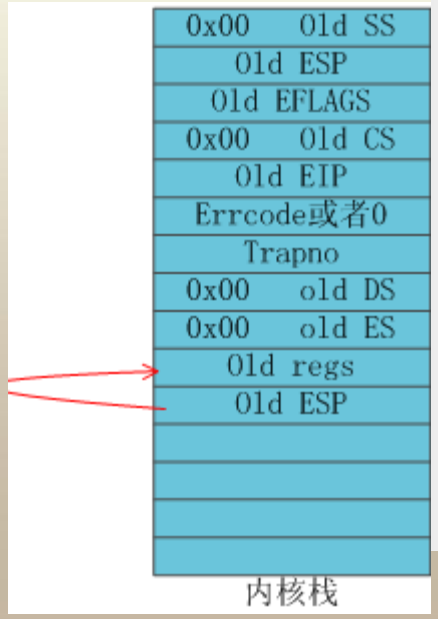
中断和异常

- 处理**缺页异常**（Page Faults）
  - trap_dispatch()函数将缺页中断(14号中断T_PGFLT)分发到**page_fault_handler()** 来处理
  - 将在Lab4中进一步处理
- 处理**断点异常**(Breakpoints Exception)
  - trap_dispatch()函数将断点异常（3，T_BRKPT）分发到**monitor()**来处理
  - 在JOS中可以把“int $3”作为一个**伪系统调用**来启动内核monitor

中断和异常

0011

- Trap（）函数输入参数struct Trapframe *tf的构造
  - 函数的输入参数是逆序压在堆栈上的
  - tf是指向Trapframe结构的一个指针，即下图的old ESP
  - “Errcode或者0”以上由CPU压入
  - 以下由Handler压入




```
struct Trapframe {
    struct PushRegs tf_regs;
    uint16_t tf_es;
    uint16_t tf_padding1;
    uint16_t tf_ds;
    uint16_t tf_padding2;
    uint32_t tf_trapno;
    /* below here defined by x86
    uint32_t tf_err;
    uintptr_t tf_eip;
    uint16_t tf_cs;
    uint16_t tf_padding3;
    uint32_t tf_eflags;
    /* below here only when cross
    uintptr_t tf_esp;
    uint16_t tf_ss;
    uint16_t tf_padding4;
};
```

系统调用

- 系统调用开启
  - #define T_SYSCALL 0x30 /* system call */
  - 设置idt[0x30]的dpl为3允许用户调用
- 系统调用流程(x86)
  - **系统调用是一个软件中断**
  - 传递**系统调用号**:EAX ； 5个参数:EDX, ECX, EBX, EDI, ESI
  - 返回值:EAX
- **理解**user/syscall.c中syscall()

系统调用

- 系统调用是由syscall.c实现的
- syscall（）函数按照syscallno进行派发
- **系统调用导致的内存保护**
  - 许多系统调用接口运行把指针传给kernel，这些指针指向用户buffer
  - 由于kernel拥有更高的权限，需要对用户传给kernel的指针进行**权限检查**
  - 对每个指针进行检查，由**user_mem_check（）**和**user_mem_assert（）**实现
  - Exercise 9 &10

GCC内联汇编

- GCC支持在C/C++代码中嵌入汇编代码，这些汇编代码被称作GCC Inline Assembly——GCC内联汇编
- 功用：
  - 将一些C/C++语法无法表达的指令直接潜入C/C++代码中
  - 允许我们直接在C/C++代码中使用汇编编写简洁高效的代码

GCC内联汇编——基本格式

- 基本内联汇编的格式

```
__asm__ __volatile__("Instruction List");
```

- 说明

1. __asm__是GCC 关键字asm 的宏定义
   #define __asm__ asm
   __asm__或asm 用来声明一个内联汇编表达式，所以任何一个内联汇编表达式都是以它开头的，是必不可少的。
2. Instruction List 是汇编指令序列：（1）每条指令都必须被双引号括起来 (2)两条指令必须用换行或分号分开
3. __volatile__是GCC 关键字volatile 的宏定义。如果用了它，则是向GCC 声明不允许对该内联汇编优化

GCC内联汇编——扩展格式

- 扩展的内联汇编格式为

```
__asm__ __volatile__("Instruction List"
: Output
: Input
: Clobber/Modify);
```

- 说明

1. Output 用来指定当前内联汇编语句的输出
2. Input 域的内容用来指定当前内联汇编语句的输入，Output和Input中，格式为形如“constraint”(variable)的列表（逗号分隔）
3. Clobber/Modify 声明当前内联汇编在Instruction List中对某些寄存器或内存进行修改。不能有Input或Output中限制的寄存器
4. 寄存器前必须使用两个百分号(%%)，而不是像基本汇编格式一样在寄存器前只使用一个百分号(%)

0011

END