ASM.S

Hongjie

主要处理 cpu 发生的各种异常，也是必须要做的工作。
异常表如下：

异常号	异常处理函数
0	divide_error
1	debug
2	nmi
3	overflow
4	bounds
5	invalid_op
6	device_not_available
7	double_fault
8	coprocessor_segment_overrun
9	invalid_TSS
10	segment_not_present
11	stack_segment
12	general_protection
13	page_fault
14	reserved
15	coprocessor_error

这些函数的处理过程大致相同，先将实际处理任务的函数地址入栈，然后使用汇编语句 jmp 跳到 no_error_code 处或 error_code 处执行。那我们先看看 error_code。

```
error_code:
    xchgl %eax,4(%esp)   # error code <-> %eax  //eax 为错误代码
    xchgl %ebx,(%esp)    # &function <-> %ebx//ebx 为处理任务的函数地址
    pushl %ecx
    pushl %edx
    pushl %edi
    pushl %esi
    pushl %ebp
    push %ds
    push %es
    push %fs                  //顺序保存以上寄存器
    pushl %eax        # error code  //错误代码入栈
    lea 44(%esp),%eax    # offset     //偏移量入栈
    pushl %eax
    movl $0x10,%eax     //0x10 为内核数据段选择子
```

```
mov %ax,%ds         //设为内核数据段
mov %ax,%es         //设为内核数据段
mov %ax,%fs         //设为内核数据段
call *%ebx           //调用处理任务的函数
addl $8,%esp
pop %fs
pop %es
pop %ds
popl %ebp
popl %esi
popl %edi
popl %edx
popl %ecx
popl %ebx
popl %eax          //顺序出栈
iret                //中断，异常返回
```

在顺序保存寄存器之后有三行语句：

```
pushl %eax    //错误代码入栈
lea 44(%esp),%eax  //偏移量
 pushl %eax
```

这里连续两次入栈是为了调用下面的实际处理函数做准备，实际上相当于他们的参数。错误代码好理解，那偏移量是什么呢？在发生异常或中断之后，系统自动将 cs,eip,eflags,esp 入栈。44(%esp)就是这时的栈顶。这主要是为了报告那里出现了问题而设的。

这里为什么非要把函数地址入栈，然后再调用呢？如果直接在 error_code 中调用函数，那就要书写 13 次这样的代码。原来这样做， error_code 相当于一个子函数，函数地址相当于参数，使整个代码结构化了。

No_error_code 和 error_code 区别在于 No_error_code 将常数 0 作为错误代码入栈。