东莞理工学院

操作系统课程设计报告

院　　系：计算机学院

班　　级：14 软卓

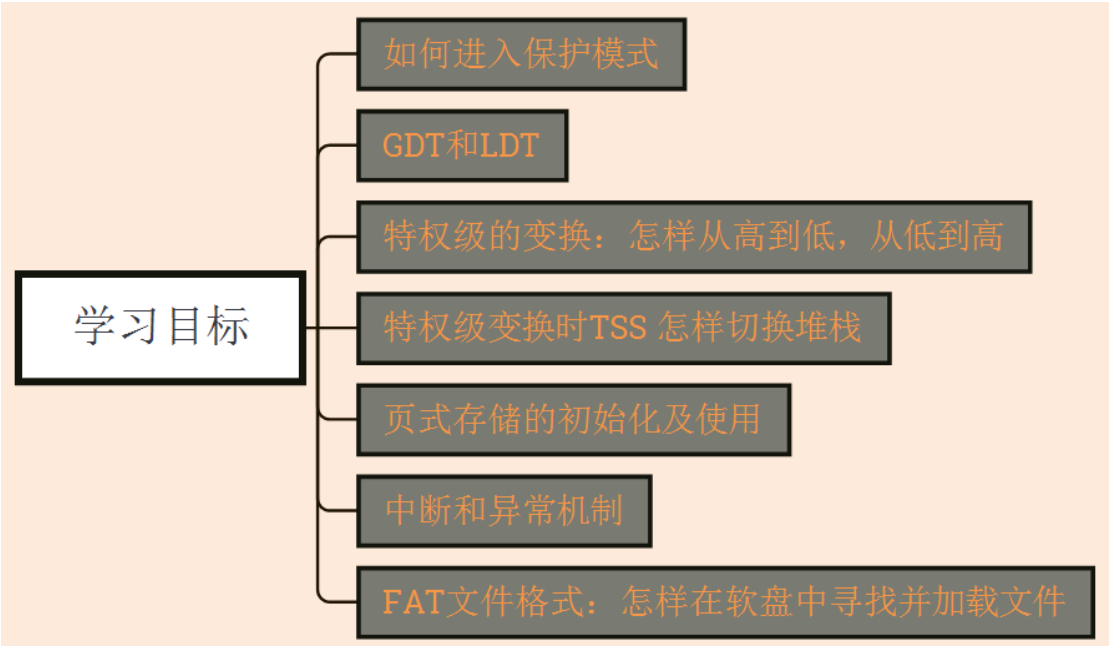
姓　　名：赖键锋

学　　号：201441402130

指导老师：李伟

日　　期：2016.6 - 2016.7

一、 相关说明



二、 相关知识的记录和说明

1. GDT:

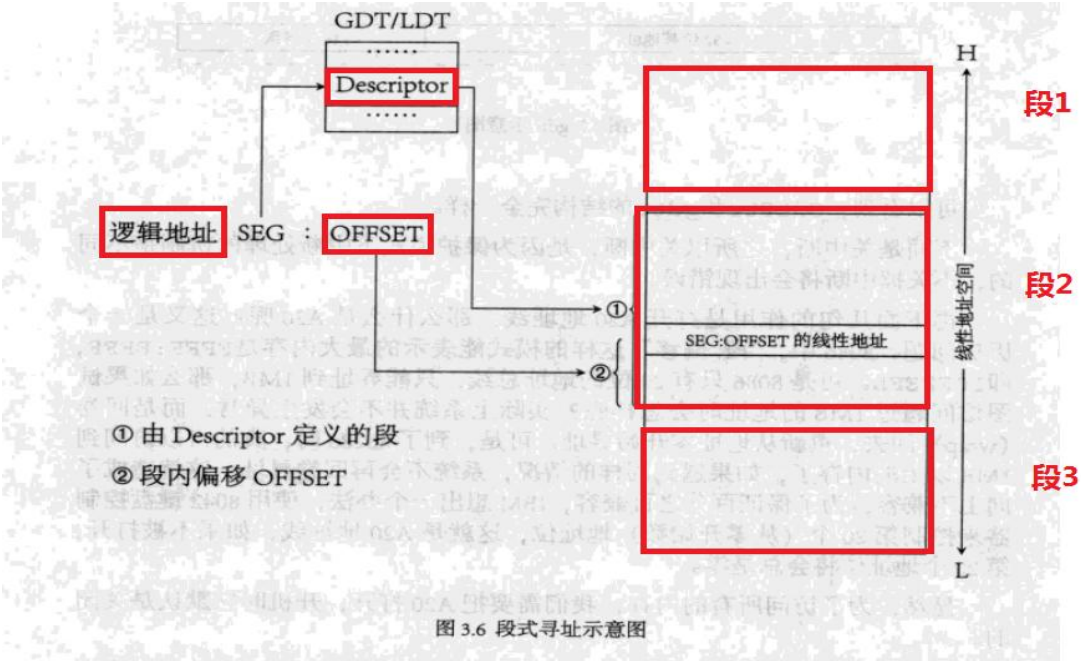


图 3.6 段式寻址示意图

- 物理地址 = 段值 *16 + 偏移
- GDT 是一个描述符表，是一个数组，每个表项称为一个描述符，

记录的是一个 段 的基地址，段的偏移范围和特权级等；

- 选择子除去开始的 3bit,剩余的 bit 表示的是描述符在 GDT 或 LDT 中的位置索引。一个选择子指向一个描述符，选择子的存在便于对描述符（段）的选择。

2. 选择子

   在实模式下,段值可以看做是地址的一部分,而在保护模式下,段值变成了指向 GDT 或 LDT 中一个表项的段选择符了,表项中才详细定义了段的起始地址,界限,属性等内容；

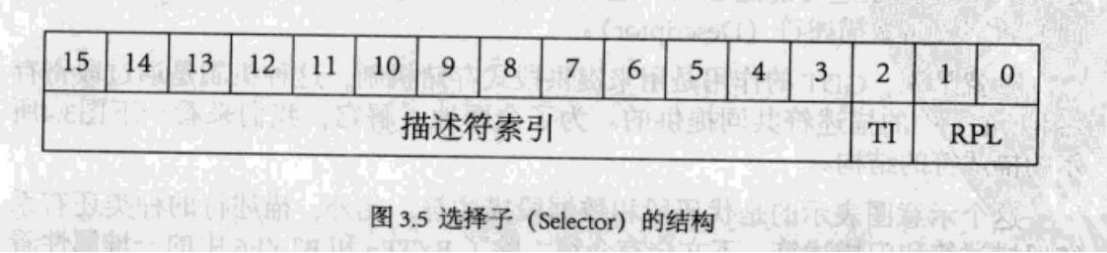


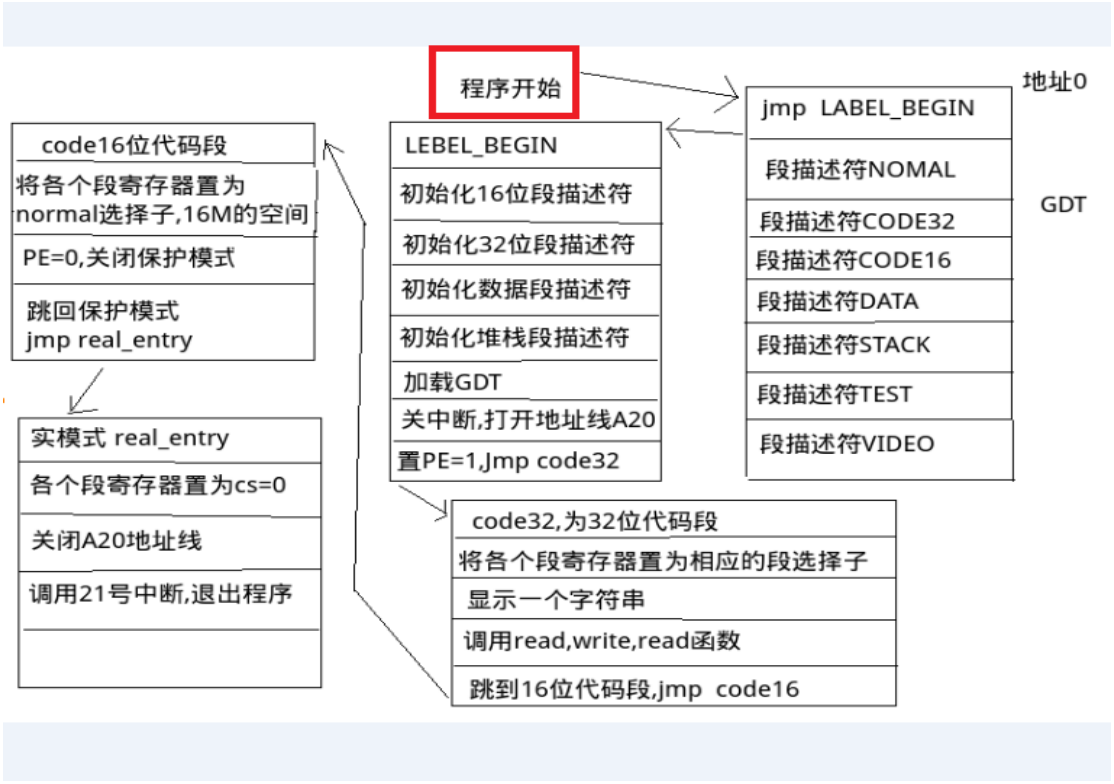
图 3.5 选择子（Selector）的结构

当 TI = 0 时,选择子指向 GDT,此时索引是相对于 GDT 的偏移;当 TI = 1 时,选择子指向 LDT,表示相对于 LDT 的偏移;RPT 是特权级；

3. 三种描述符区别：

- TSS descriptor 提供硬件级的进程切换机制；
- LDT descriptor 供进程使用多个 descriptor
- Gate descriptor 提供 processor 权限级别的切换机制。

4. 进入保护模式后，段之间的跳转都是通过 GDT，先进行特权级的比较，合法才设置寄存器，进行跳转。下图是 pmtest2.asm 流程：



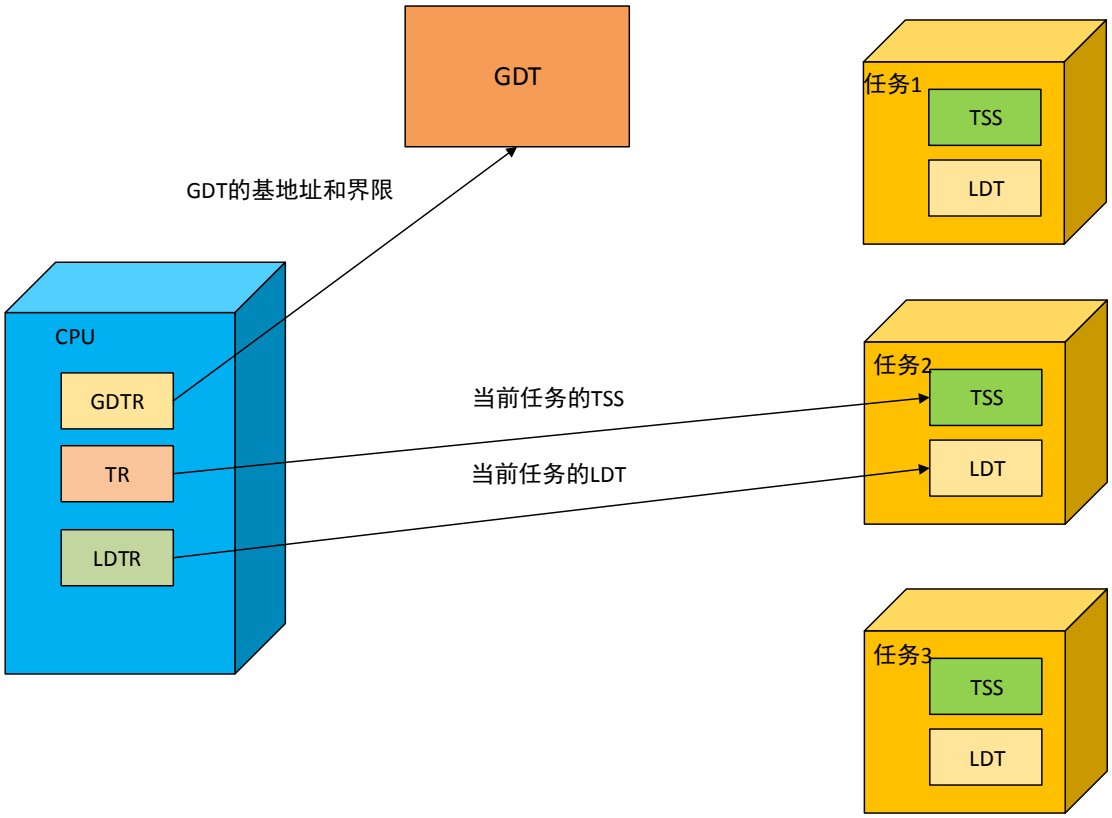
5. TSS(任务状态段)：每个任务都有一个 TSS；

在一个多任务环境中，当任务发生切换时，必须保存现场(比如通用寄存器，段寄存器，栈指针等)。为了保存被切换任务的状态，并且在下次执行它时恢复现场，每个任务都应当有一片内存区域，专门用于保存现场信息，这就是任务状态段(Task State Segment)。

6. 切换任务时：

处理器将要挂起的任务的现场信息保存到 TR 指向的 TSS;然后，使 TR 指向新任务的 TSS，并从这个 TSS 中恢复现场。

多任务系统的组成示意图：



7. 特权级的转移:

程序控制的转移,可以是由 jmp,call, ret , sysenter , sysexit , int n , iret 或中断和异常机制引起. 在转移控制之前,处理器会检查描述符的界限,类型,特权级等内容,如果检查成功,cs 将被加载,程序控制将转移到新的代码段中,从 eip 指示的位置开始执行.

指令	Jmp	Call + 调用门	ret
转移方向	同级之间	低级 → 高级	高级 → 低级

8. Jmp 和 call :

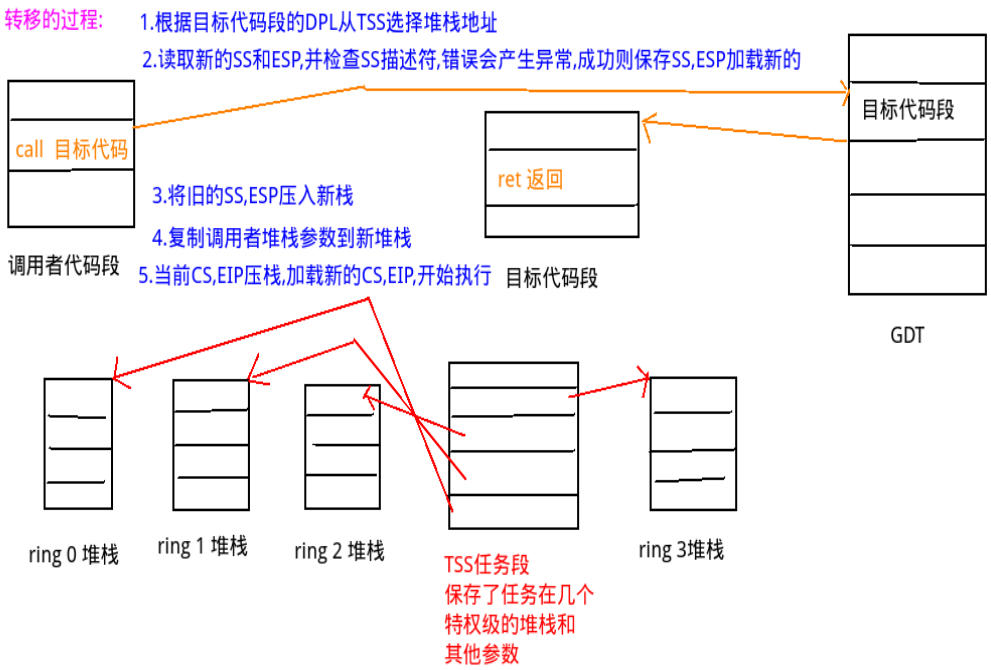
- Jmp 指令仅仅进行执行流程的跳转，不会保存返回地址。

- Call 指令在进行流程跳转前会保存返回地址，以便在跳转目标代码中可以使用 ret 指令返回到 call 指令的下一条指令处继续执行。执行段内跳转时，只保存 EIP；如果是段间跳转，还保存 CS。
- 执行 call 指令前必须准备好任务状态段 TSS。
- 执行 Ret 指令前，堆栈中应该准备好了

9. Ret 指令

a) 检查保存的 cs 中的 RPL 以判断返回时是否要变换特权级
b) 加载被调用者堆栈上的 cs 和 eip（此时会进行代码段描述符和选择子类型和特权级检查
c) 如果 ret 指令含有参数，则增加 esp 的值以跳过参数，然后 esp 将指向被保存过的调用者的 ss 和 esp
d) 加载 ss 和 esp，切换到调用者堆栈，被调用者的 ss 和 esp 被丢弃。
e) 如果 ret 指令含有参数，增加 esp 的值以跳过参数（此时已经在调用者堆栈中，也就是在 ring3 堆栈中了
f) 检查 ds、es、fs、gs 的值，如果其中哪一个寄存器指向的段的 DPL 小于 CPL，那么一个空描述符被加载到该寄存器

10.TSS 堆栈转移的过程



11. PDE：页目录表的表项；PTE：页表的表项；

CR3：页目录寄存器 PDBR；

CR0 的 PG 位:分页机制的开关，PG=1 表示分页机制生效；

12.中断和异常机制

- 中断通常在程序执行时因为硬件而随机发生，它们通常是用来处理处理器外部的事件，如外围设备的请求。软件通过执行 int n 指令也可以产生中断。
- 异常通常是在处理器执行指令时检查到错误时发生，比如遇到零除的情况。处理器检测的错误条件有很多，比如保护违例，页错误等。
- 中断和异常的两个问题：

（一）处理器可以对何种类型的通知做出反应?

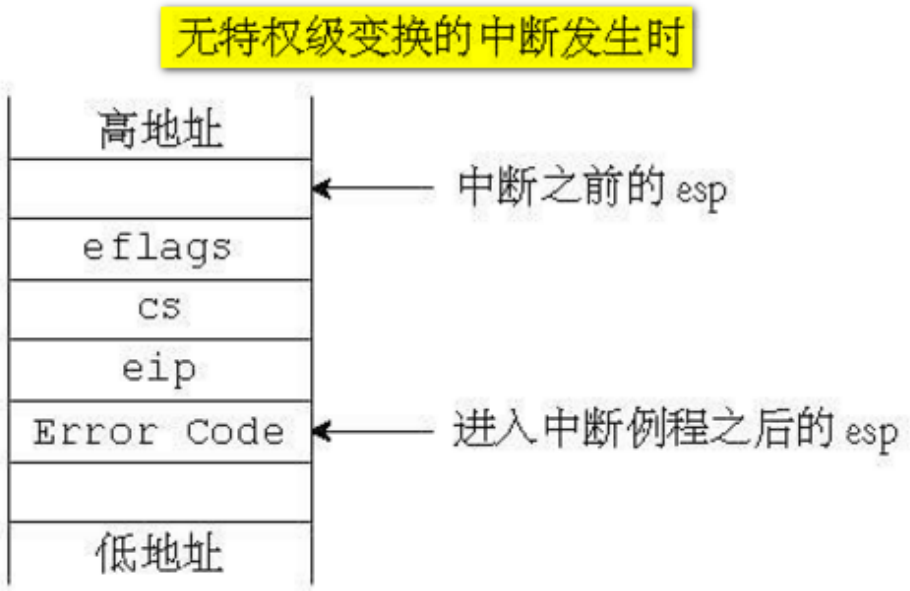
处理器能处理的中断和异常如下图所示

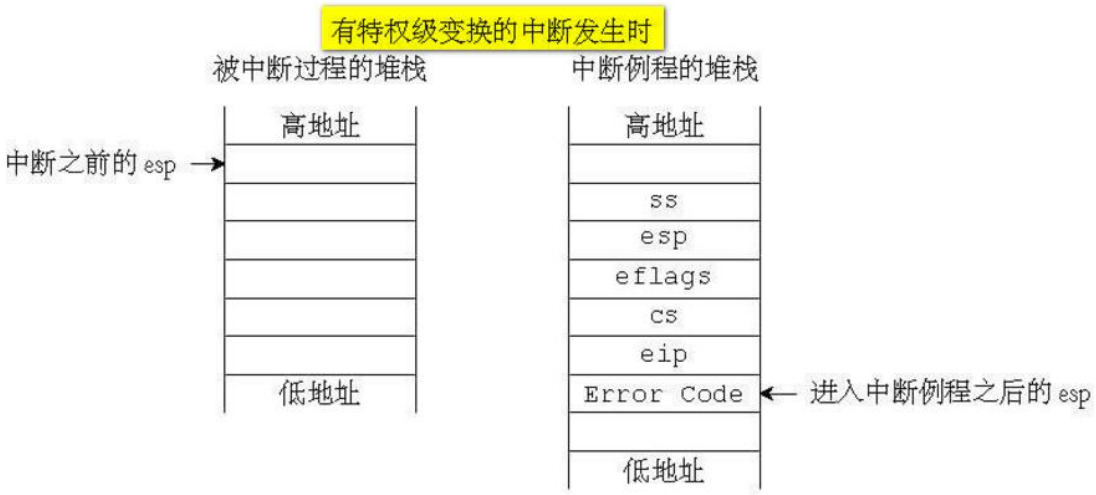
向量号	助记符	描述	类型	出错码	源
0	#DE	除法错	Fault	无	DIV 和 IDIV 指令
1	#DB	调试异常	Fault/Trap	无	任何代码和数据的访问
2	—	非屏蔽中断	Interrupt	无	非屏蔽外部中断
3	#BP	调试断点	Trap	无	指令 INT 3
4	#OF	溢出	Trap	无	指令 INTO
5	#BR	越界	Fault	无	指令 BOUND
6	#UD	无效（未定义的）操作码	Fault	无	指令 UD2 或者无效指令
7	#NM	设备不可用（无数学协处理器）	Fault	无	浮点或 WAIT/FWAIT 指令
8	#DF	双重错误	Abort	有 (0)	所有能产生异常或 NMI 或 INTR 的指令
9		协处理器段越界（保留）	Fault	无	浮点指令（386 之后的 IA32 处理器不再产生此种异常）
10	#TS	无效 TSS	Fault	有	任务切换或访问 TSS 时
11	#NP	段不存在	Fault	有	加载段寄存器或访问系统段时
12	#SS	堆栈段错误	Fault	有	堆栈操作或加载 SS 时
13	#GP	常规保护错误	Fault	有	内存或其他保护检验
14	#PF	页错误	Fault	有	内存访问
15	—	Intel 保留，未使用			
16	#MF	x87FPU 浮点错（数学错）	Fault	无	x87FPU 浮点指令或 WAIT/FWAIT 指令
17	#AC	对齐检验	Fault	有 (0)	内存中的数据访问（486 开始支持）
18	#MC	Machine Check	Abort	无	错误码（如果有的话）和源依赖于具体模式（奔腾 CPU 开始支持）
19	#XF	SIMD 浮点异常	Fault	无	SSE 和 SSE2 浮点指令（奔腾 III 开始支持）
20~31	—	Intel 保留，未使用			
32~255	—	用户定义中断	Interrupt		外部中断或 int n 指令

（二） 当接到某种通知时做出何种处理?

解决：每一种中断向量（异常）都会对应一个中断向量号，而这个中断向量号通过 IDT 就与相应的中断处理程序对应起来。

13. 从中断或异常返回时必须使用指令 iretd,它与 ret 很相似,只是它同时会改变 eflags 的值。

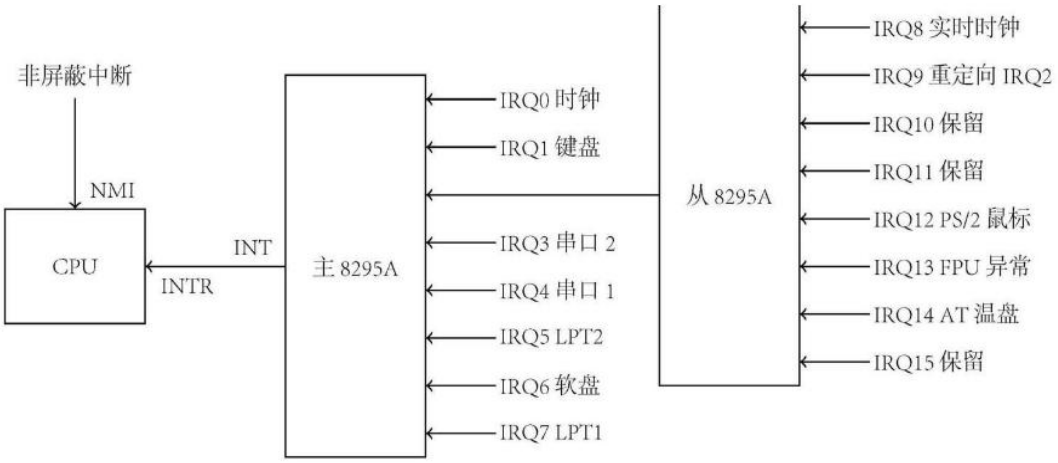




14. 8259A 是可编程中断控制器，对它的设置是通过向相应的端口写入特定的 ICW（Initialization Command Word）来实现的，有 4 个 ICW；8259A 的初始化顺序是固定的；

OCW（Operation Control Word）有三个，当要屏蔽或打开外部中断，或发送 EOI 给 8259A 以通知它中断处理结束是要用到 OCW。

15. 8259A 芯片是一个中断管理芯片，中断的来源除了来自于硬件自身的 NMI 中断和来自于软件的 INT n 指令造成的软件中断之外，还有来自于外部硬件设备的中断，这些中断是可屏蔽的。

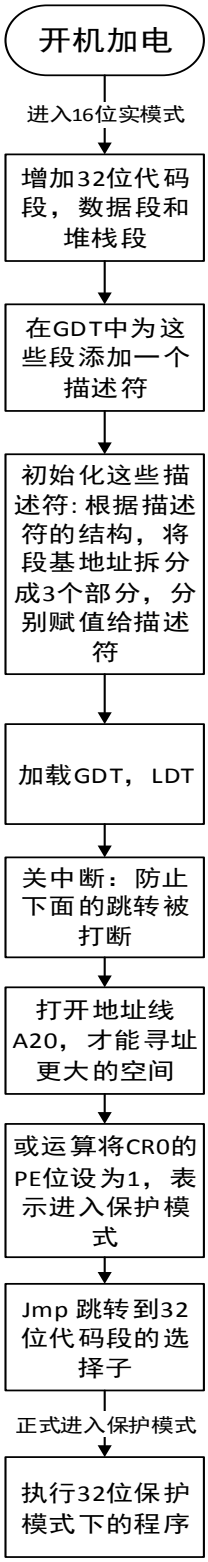


时钟中断就是通过 IR0 发出中断请求的。

三、 程序关系图或流程图

1） 进入保护模式最简单的代码理解

1. 保护模式其实就是通过描述符中段基址、段界限、段属性和特权级来实现保护；所以进入保护模式前要准备好保护模式的环境，包括 GDT；



要进入保护模式：

1) 增加 32 位保护模式下的代码段，数据段，和堆栈段；
2) 初始化选择子；
3) 加载 GDTR；
4) 关中断；
5) 打开地址线 A20；
6) 设置 CR0 的 PE 位为 1；
7) 跳转；

返回实模式，可以把实模式看成保护模式下的一个段，段基址是 0，段界限是 1M：

1) 将寄存器设置为实模式段的段地址，恢复实模式的环境；
2) 将 CR0 的 PE 为设置为 0，关闭保护模式；
3) 跳转回实模式段；

2. 段描述符的结构，每个描述符 8 个字节。

```
%macro Descriptor 3
    dw  %2 & 0FFFFh             ; 段界限1
    dw  %1 & 0FFFFh             ; 段基址1
    db  (%1 >> 16) & 0FFh           ; 段基址2
    dw  ((%2 >> 8) & 0F00h) | (%3 & 0F0FFh) ; 属性1 + 段界限2 + 属性2
    db  (%1 >> 24) & 0FFh           ; 段基址3
%endmacro ; 共 8 字节
```

段描述符的定义：

```
%include    "pm.inc"    ; 常量, 宏, 以及一些说明

org 07c00h
    jmp LABEL_BEGIN

[SECTION .gdt]
; GDT
;                                段基址,       段界限     , 属性
LABEL_GDT:      Descriptor       0,                0, 0           ; 空描述符
LABEL_DESC_CODE32: Descriptor       0, SegCode32Len - 1, DA_C + DA_32; 非一致代码段
LABEL_DESC_VIDEO:  Descriptor 0B8000h,          0ffffh, DA_DRW      ; 显存首地址
; GDT 结束

GdtLen      equ $ - LABEL_GDT   ; GDT长度
GdtPtr      dw  GdtLen - 1  ; GDT界限
        dd  0       ; GDT基地址

; GDT 选择子
SelectorCode32      equ LABEL_DESC_CODE32   - LABEL_GDT
SelectorVideo       equ LABEL_DESC_VIDEO    - LABEL_GDT
; END of [SECTION .gdt]
```

定义了3个描述符，
分别指向3个段

程序开头GDT的定义，定义了 3 个描述符，分别指向 3 个段；

3.

```
LABEL_SEG_CODE32:
    mov ax, SelectorVideo
    mov gs, ax          ; 视频段选择子(目的)

    mov edi, (80 * 11 + 79) * 2 ; 屏幕第 11 行, 第 79 列。
    mov ah, 0Ch         ; 0000: 黑底    1100: 红字
    mov al, 'P'
    mov [gs:edi], ax

    ; 到此停止
    jmp $

SegCode32Len    equ $ - LABEL_SEG_CODE32
; END of [SECTION .s32]
```

这是进入保护模式后要运行的 32 位代码段，它的描述符是 LABEL_DESC_CODE32，它的作用是让 gs:edi 指向屏幕的某个位置，

然后输出 ax 。

4. 进入保护模式前的准备：

```
27
28  [SECTION .s16]
29  [BITS   16]
30  LABEL_BEGIN:
31      mov ax, cs
32      mov ds, ax
33      mov es, ax
34      mov ss, ax
35      mov sp, 0100h
36      ; 初始化 32 位代码段描述符
37      xor eax, eax
38      mov ax, cs
39      shl eax, 4
40      add eax, LABEL_SEG_CODE32
41      mov word [LABEL_DESC_CODE32 + 2], ax
42      shr eax, 16
43      mov byte [LABEL_DESC_CODE32 + 4], al
44      mov byte [LABEL_DESC_CODE32 + 7], ah
45      ; 为加载 GDTR 作准备
46      xor eax, eax
47      mov ax, ds
48      shl eax, 4
49      add eax, LABEL_GDT      ; eax <- gdt 基地址
50      mov dword [GdtPtr + 2], eax ; [GdtPtr + 2] <- gdt 基地址
51      ; 加载 GDTR
52      lgdt    [GdtPtr]
53      ; 关中断
54      cli
55      ; 打开地址线A20
56      in  al, 92h
57      or  al, 00000010b
58      out 92h, al
59      ; 准备切换到保护模式
60      mov eax, cr0
61      or  eax, 1
62      mov cr0, eax
63      ; 真正进入保护模式
64      jmp dword SelectorCode32:0  ; 执行这一句会把 SelectorCode32 装入 cs,
65                                  ; 并跳转到 Code32Selector:0  处
```

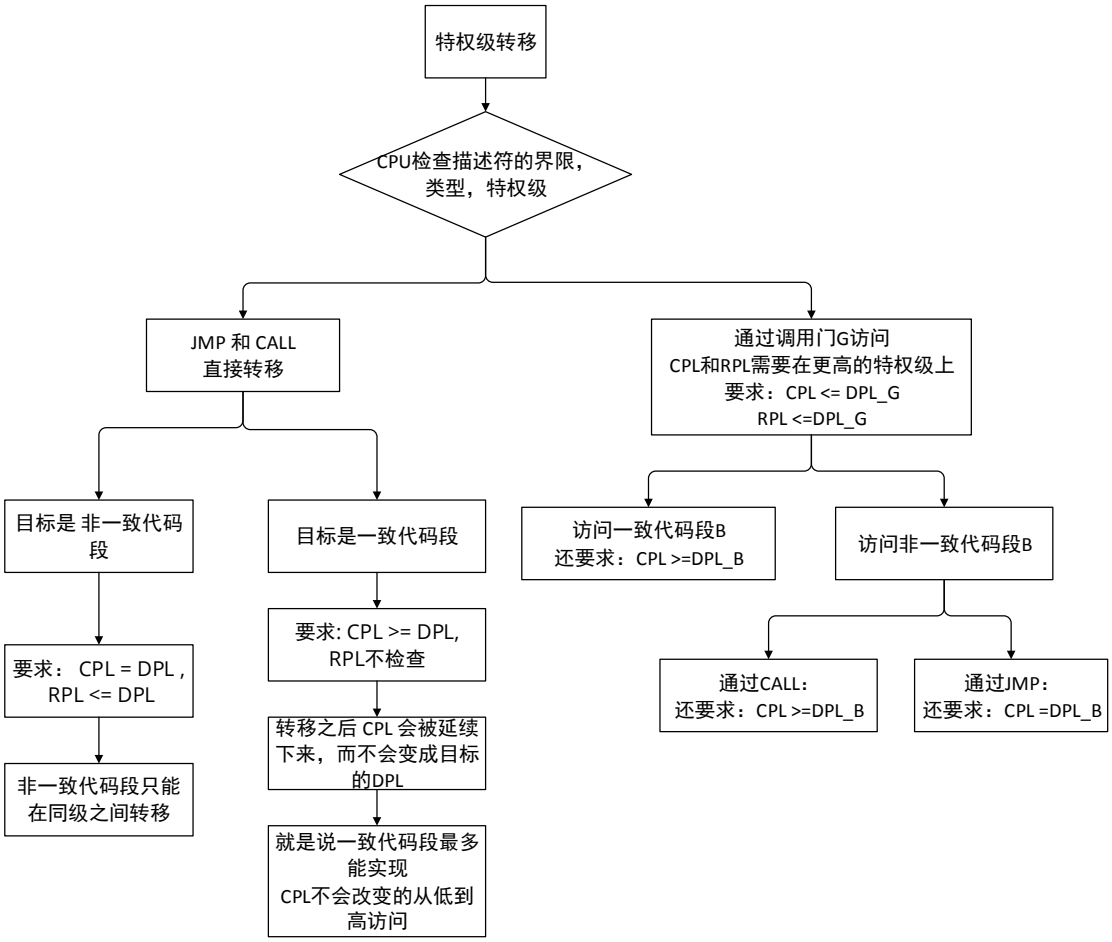
- 因为要使用到描述符，而描述符现在只是给出了定义，还没有指向具体的段，所以要先初始化；
- 初始化描述符：(在第 38 到 45 行),先获得 32 位代码段的首地址 LABEL_SEG_CODE32，然后按照描述符的结构，将该地址拆成 3 个部分分别填进它的段描述符。
- 第 48 到 55,行，加载 GDT，要先把 GDT 的长度，界限和基地

址算出来，存放到 GdtPtr 结构中，然后用 lgdt 指令将这些信息加载到进村器 GDTR 中，

- A20 地址线关闭则总是 0，访问不到所有的内存。
- 关中断 cli，是为了避免过程被中断打扰，还要做或运算，将 cr0 的 PE 位设为 1，表示打开保护模式；
- 长跳转到 32 位代码段的选择子处，即跳转到代码段，开始执行该程序，显示一个红色 P

2） 特权级转移

1. 特权级的检查规则：



2. 关于特权级变化的关键的代码：

- **高特权级转移到低特权级**，参数可以由自己压栈，然后 retf 弹出。

```
314     ; Load TSS
315     mov ax, SelectorTSS
316     ltr ax  ; 在任务内发生特权级变换时要切换堆栈。
317             ;而内层堆栈的指针存放在当前任务的TSS中，
318             ;所以要设置任务状态段寄存器 TR。
319
320     push    SelectorStack3
321     push    TopOfStack3
322     push    SelectorCodeRing3
323     push    0
324     retf        ; Ring0 -> Ring3，历史性转移！将打印数字 '3'。
```

- **低特权级到高特权级**，用 call + 调用门：

```
443     call    SelectorCallGateTest:0
444         ; 测试调用门（有特权级变换），将打印字母 'C'。
```

3） 页式存储

4）中断和异常

IDT 描述符的定义中：先重复定义 32 个中断门描述符，再重复定义 95 个中断门描述符，最后再定义一个中断描述符，总共有 3 个中断处理例程。

```
105 LABEL_IDT:
106 ; 门                        目标选择子,            偏移, DCount, 属性
107 %rep 32
108         Gate    SelectorCode32, SpuriousHandler,      0, DA_386IGate
109 %endrep
110 .020h:     Gate    SelectorCode32,    ClockHandler,      0, DA_386IGate
111 %rep 95
112         Gate    SelectorCode32, SpuriousHandler,      0, DA_386IGate
113 %endrep
114 .080h:     Gate    SelectorCode32,  UserIntHandler,      0, DA_386IGate
115
116 IdtLen      equ $ - LABEL_IDT
117 IdtPtr      dw  IdtLen - 1  ; 段界限
118         dd  0       ; 基地址
119 ; END of [SECTION .idt]
```

加载 IDT：步骤与加载 GDT 一样，

```
210
211     ; 为加载 IDTR 作准备
212     xor eax, eax
213     mov ax, ds
214     shl eax, 4
215     add eax, LABEL_IDT      ; eax <- idt 基地址
216     mov dword [IdtPtr + 2], eax ; [IdtPtr + 2] <- idt 基地址
217
218     ; 保存 IDTR
219     sidt    [_SavedIDTR]
220
```

要使始终中断处理例程工作，需要打开 8259A 的时钟中断 IRQ0，就要设置 IMR（中断屏蔽寄存器）和 IF 为，设置 IMR 通过 OCW2，设置 IF 为通过 sti 指令。

```
225     ; 加载 GDTR
226     lgdt    [GdtPtr]
227
228     ; 关中断
229     ;cli
230
231     ; 加载 IDTR
232     lidt    [IdtPtr]
```

时钟中断处理程序：

```
386 ; int handler ---------------------------------------------------------
387 _ClockHandler:
388 ClockHandler    equ _ClockHandler - $$
389     inc byte [gs:((80 * 0 + 70) * 2)]   ; 屏幕第 0 行, 第 70 列。
390     mov al, 20h
391     out 20h, al             ; 发送 EOI
392     iretd
```

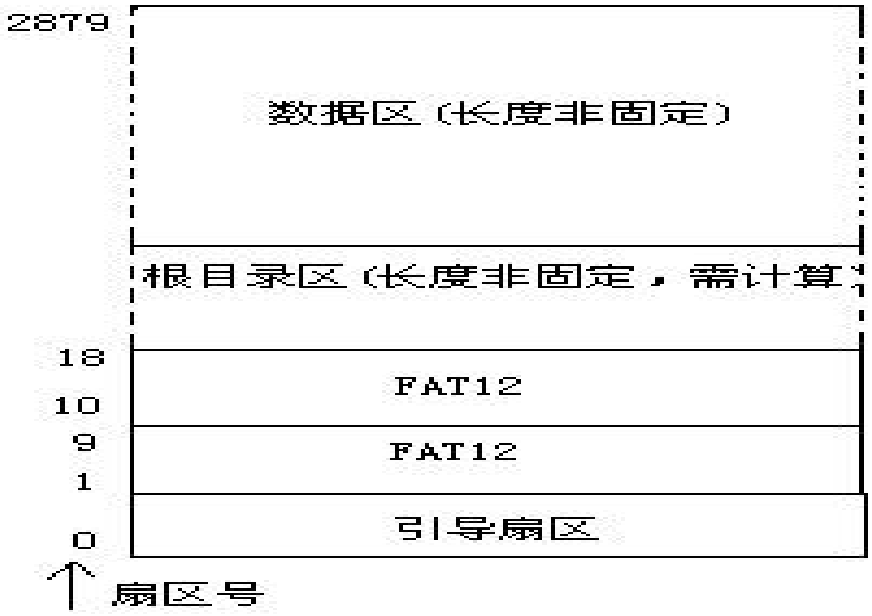
打开时钟中断后，机器的时钟中断会不断产生，并通过 8259A 中断控制器将中断向量号送给 CPU，引起时钟中断处理例程，所以例程中的那个字符会不断的发生变化。

5） 突破 512 字节的限制，自己编写内核加载程序

1. FAT12

由于文件的大小不同和不停的增删操作，同一个文件的数据往往不能完整地存放在磁盘的一段连续的扇区内，而会分成若干小段，像一条链子一样存放，这种存储方式称为**文件的链式存储**。为了实现文件的链式存储，磁盘上必须准确地记录哪个文件占用了哪些扇区，哪些扇区还没有被占用等信息。用来记录磁盘中文件如何被分散存储在不同扇区的信息的表格就叫**文件分配表**（File Allocation Table，简写为 **FAT**）。可以采用多种办法记录文件占用存储单元的信息，FAT 表相应的也有很多种格式，3.5 英寸软盘采用的是 FAT12 格式。

2. FAT12 的软盘空间存储图：



- FAT 表中每个表项的值代表的就是某文件占用的下一个扇区的序号！如果值大于或等于 4088，则表示当前扇区已经是本文件的最后一个扇区。两个 FAT 表是一样的。

- 根目录表位于第二个 FAT 表之后。根目录表由最多 224 个表项组成（DOS 的规定），每个表项对应一个文件，记录了该文件的文件名、文件属性和占用的第 1 个扇区号。根目录表的表项格式是这样的：

名称	偏移	长度	描述
DIR_Name	0	11	文件名8字节，扩展名3字节
DIR_Attr	0xB	1	文件属性
	0xC	10	保留
DIR_WrtTime	0x16	2	最后修改时间
DIR_WrtDate	0x18	2	最后修改日期
DIR_FstClus	0x1A	2	此条目对应的开始扇区号
DIR_FileSize	0x1C	4	文件大小

- 根目录表后面就是数据区了，数据区从 33 号逻辑扇区开始：

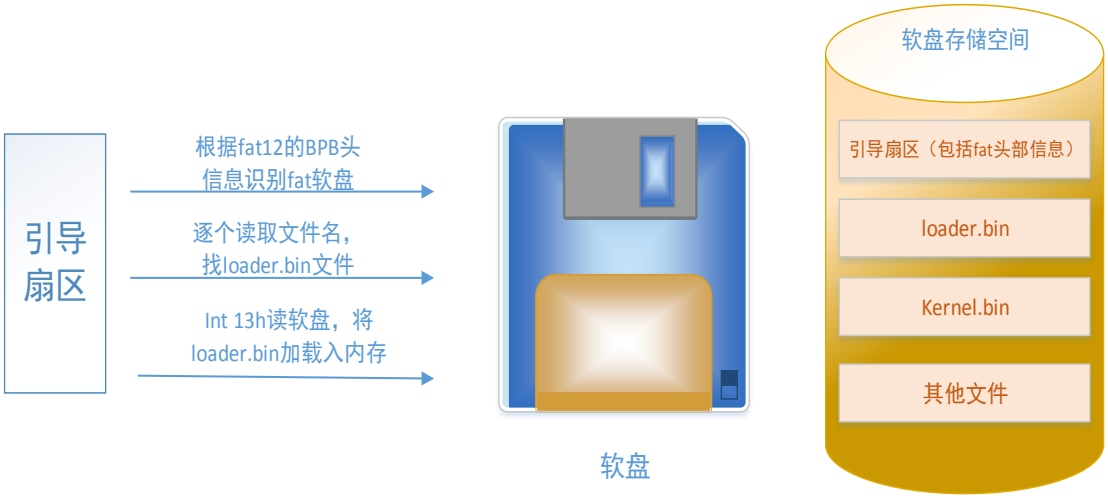
引导扇区数 1＋FAT 表占用扇区数 18＋ 根目录表占用扇区数 14＝33。

3. 操作系统又是怎么知道 FAT12 有 2 个 FAT，每个 FAT 占用 9 扇区，根目录表最多有 224 个表项的呢？——因为这些信息都记录在**磁盘第一个扇区**里了！第一个扇区的 512 字节，每个都有明确的用途，表格：

名称	偏移	长度	内容	软盘参考值
BS_jmpBoot	0	3	跳转指令，指向程序入口	jmp _main nop
BS_OEMName	3	8	厂商名	自己定义
BPB_BytsPerSec	11	2	每扇区字节数	512
BPB_SecPerClus	13	1	每簇扇区数	1
BPB_RsvdSecCnt	14	2	保留扇区数（用作引导）	1
BPB_NumFATs	16	1	FAT 表数	2
BPB_RootEntCnt	17	2	根目录中能容纳文件的最大数量	DOS为224
BPB_TotSec16	19	2	扇区总数（FAT12、16 用）	2880
BPB_Media	21	1	介质描述符	0xF0
BPB_FATSz16	22	2	每 FAT 表占用扇区数	9
BPB_SecPerTrk	24	2	每磁道扇区数	18
BPB_NumHeads	26	2	磁头数	2
BPB_HiddSec	28	4	隐藏扇区数	0
BPB_TotSec32	32	4	扇区总数（FAT32 用）	2880
BS_DrvNum	36	1	驱动器号	0
BS_Reserved1	37	1	未使用	0
BS_BootSig	38	1	扩展引导标记	0x29
BS_VolD	39	4	卷标序列号	0
BS_VolLab	43	11	卷标	自己定义
BS_FileSysType	54	8	文件系统类型	'FAT12'
引导代码	62	448	引导代码、数据及其他填充字符	
结束标志	510	2		0xAA55

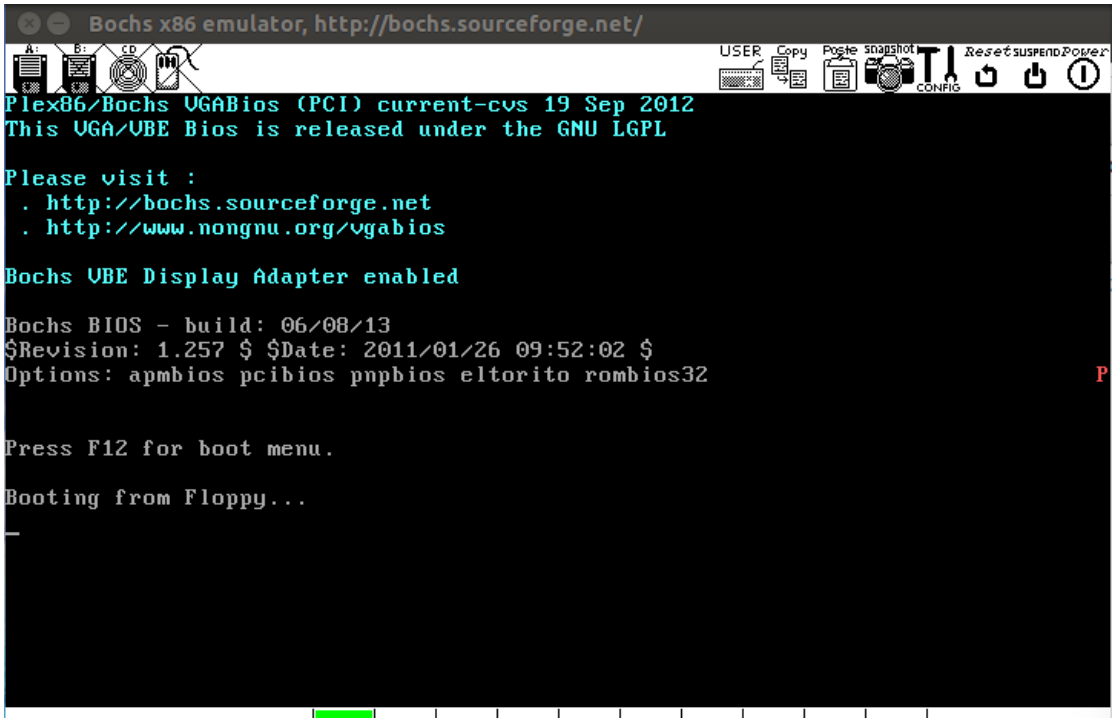
”？

4. 读取软盘，int 13h

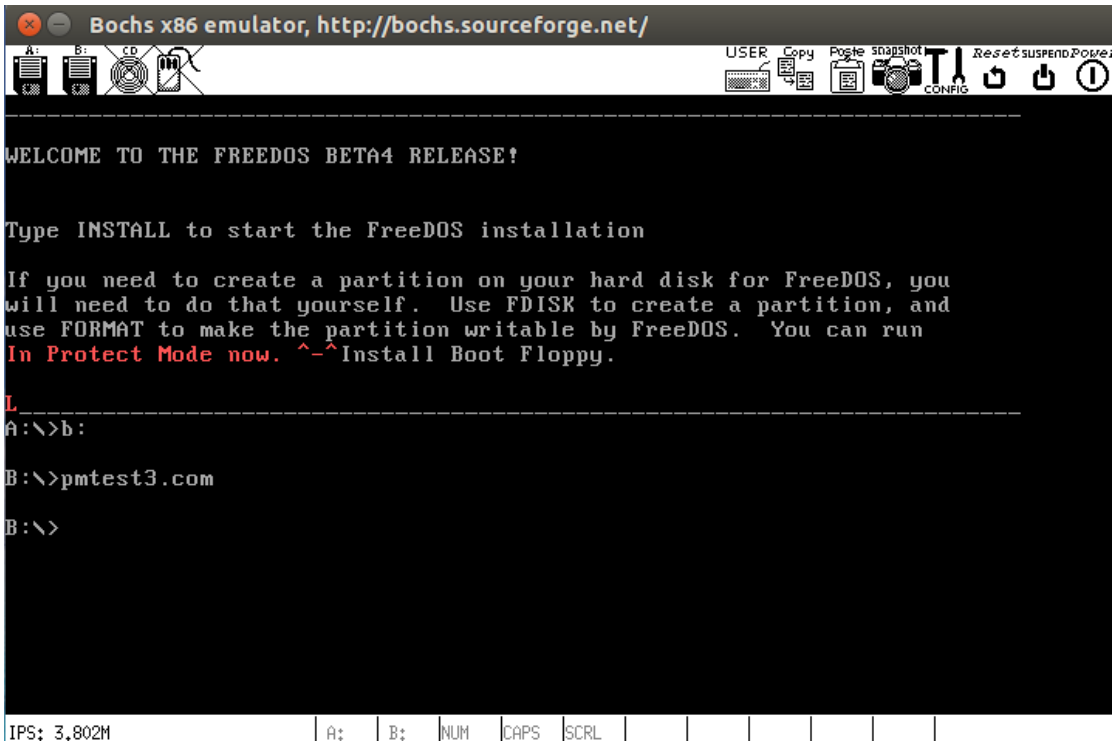


四、 运行过程及理解

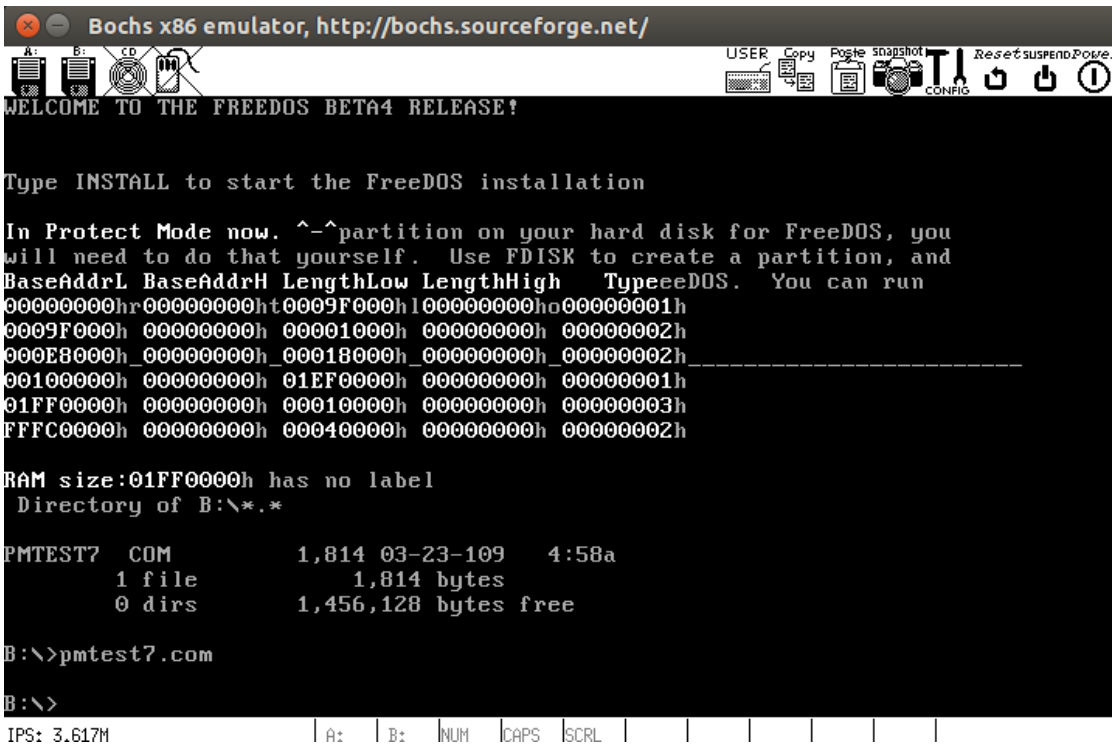
1. 第一次进入保护模式：在中间行最右边显示 P



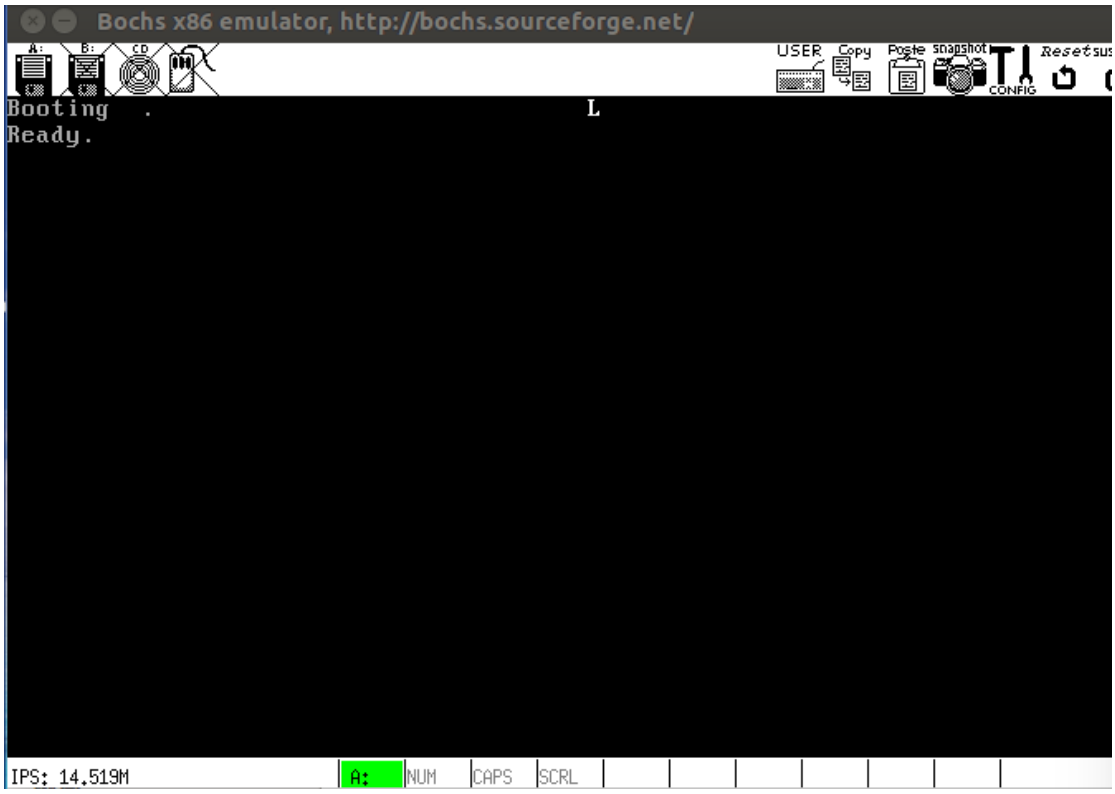
2. 借助 freedos 运行程序并进入保护模式。



3. 用 int 15 读内存信息并显示出来。



4. 引导扇区搜索 loader.bin，loader.bin 再搜索 kernel.bin.



五、 重点知识总结

1) GDT:

2) 特权级的转移

3) 分页机制

4) 中断和异常