分析 bootloader 进入保护模式的过程

0.关中断并清除寄存器

通过阅读 bootasm.S 代码，可以看到在进入 A20 之前先进入一个 start 函数

```
# start address should be 0:7c00, in real mode, the beginning address of the running bootloader
.globl start
start:
.code16                                             # Assemble for 16-bit mode
    cli                                             # Disable interrupts
    cld                                             # String operations increment

    # Set up the important data segment registers (DS, ES, SS).
    xorw %ax, %ax                                   # Segment number zero
    movw %ax, %ds                                   # -> Data Segment
    movw %ax, %es                                   # -> Extra Segment
    movw %ax, %ss                                   # -> Stack Segment
```

cli 指令关闭中断，cld 指令注释的解释是“字符串增加操作”
下面的一系列 xor 和 mov 指令则是把寄存器清 0

1. 为何开启 A20，以及如何开启 A20

为何开启 A20：

早期 8086 CPU 只有 20 位地址线，才用了段地址加偏移地址的地址转换机制，但这种方式下的地址表示能力超过了 20 位地址线的物理寻址能力，所以当寻址到超过 1MB 的内存时会发生“回卷”。

但是下一代的 80286 CPU 提供了 24 根地址线和保护模式，这样就可以访问 1MB 以上的内存了，系统也不会再“回卷”。但这就造成了向下不兼容。

为了保持向下的兼容性，IBM 决定加个硬件逻辑（A20 Gate）来模仿回卷。这个 Gate 可以控制 A20 地址线的打开和关闭，初始状态为 0（关闭的）。

在实模式下如果要访问高端内存区，也就是 1MB 以上的内存区，这个开关必须打开；而在保护模式下，如果不打开这个开关系统就只能访问奇数兆的内存，无法访问所有可用内存，所以保护模式下这个开关也必须打开。

如何开启 A20：

根据 bootasm.S 中的代码：

`seta20.1:`	
`inb $0x64, %al`	从输入缓冲端口读取内容到 al 寄存器
`testb $0x2, %al`	测试输入缓冲区是否为空
`jnz seta20.1`	如果不为空就跳转回开头直到为空

movb $0xd1, %al outb %al, $0x64	al 寄存器赋值为 1101 0001（写输出端口的指令）通过 al 寄存器向 0x64 端口写入数据
seta20.2: inb $0x64, %al testb $0x2, %al jnz seta20.2	同理 20.1，等待输入缓冲端口不忙
movb $0xdf, %al outb %al, $0x60	通过 al 寄存器向 0x60 端口写入指令：1101 1111，即将 A20 设置为 1

2.如何初始化 GDT 表以及如何使能和进入保护模式

lgdt gdtdesc	一个简单的 GDT 表和其描述符已经静态储存在引导区中，载入即可。

movl %cr0, %eax
orl $CR0_PE_ON, %eax
movl %eax, %cr0
	将 cr0 寄存器的第 0 位（PE 位）置 1

ljmp$PROT_MODE_CSEG,$protcseg	长跳转到 32 位代码段
.code32 protcseg:	初始化段寄存器

movw $PROT_MODE_DSEG, %ax
movw %ax, %ds
movw %ax, %es
movw %ax, %fs
movw %ax, %gs
movw %ax, %ss
	$PROT_MODE_DSEG 的值为 0x8

movl $0x0, %ebp
movl $start, %esp
call bootmain
	建立堆栈，设置栈指针
	栈顶在 start 处，也就是 0x7c00 处
	调用 main.c 中的 bootmain 函数