1. 什么是实模式，什么是保护模式？

实模式（Real Mode）：实模式是处理器的一种工作模式，通常在早期的x86架构的处理器中使用。在实模式下，处理器以简单的16位模式工作，可以访问1MB的物理内存。实模式下的处理器可以直接访问内存中的物理地址，但没有内存保护、虚拟内存和多任务处理的功能。实模式主要用于早期的操作系统和一些特定的应用程序。

保护模式（Protected Mode）：保护模式是处理器的一种工作模式，通常在现代x86架构的处理器中使用。在保护模式下，处理器可以访问高于1MB的物理内存，支持虚拟内存、内存保护和多任务处理等高级功能。保护模式下的处理器使用段描述符和页表来管理内存访问权限，并且可以同时运行多个任务或进程。现代操作系统如Windows、Linux等都在保护模式下运行

CPU复位（reset）或加电（power on）的时候以实模式启动，处理器以实模式工作。在实模式下，内存寻址方式和8086相同，由16位段寄存器的内容乘以16（10H）当做段基地址，加上16位[偏移地址](https://baike.baidu.com/item/%E5%81%8F%E7%A7%BB%E5%9C%B0%E5%9D%80?fromModule=lemma_inlink" \t "https://baike.baidu.com/item/%E4%BF%9D%E6%8A%A4%E6%A8%A1%E5%BC%8F/_blank)形成20位的物理地址，最大寻址空间1MB，最大分段64KB。

1. 什么是选择子？

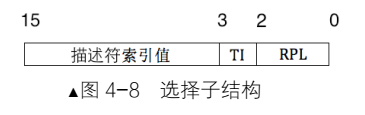
选择子共16位，放在段选择寄存器里

低2位表示请求特权级

第3位表示选择GDT还是LDT方式（TI为0表示描述符在GDT表中，为1表示描述符在LDT表中。）

高13位表示在描述符表中的偏移

故描述符表的项数最多是2的13次方



1. 什么是描述符？

段描述符是GDT/LDT表中的一个数据结构项，包含了一个段的位置、大小、特权级别等信息。每个段描述符占用8字节（64位），包含了段基地址、段限长、段属性。

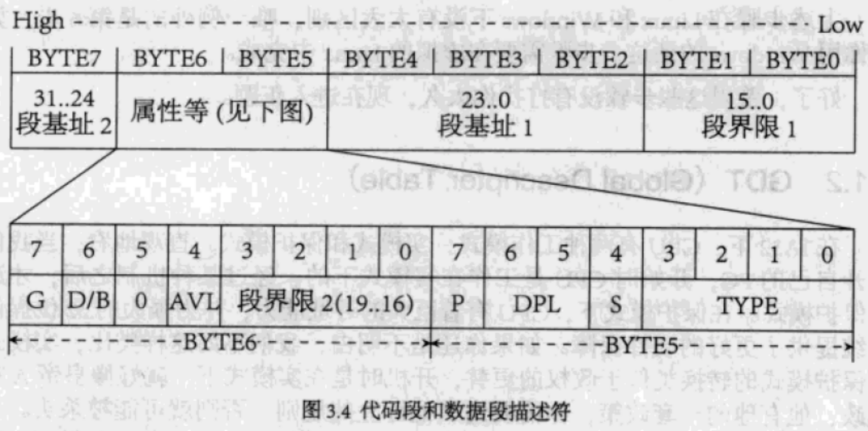
段基址（Base address），指定段在线性地址空间中的开始地址。基地址是线性地址，对应于段中偏移0处。是由三段地址拼接而成的，一共32位。

段界限，段限长（limit），是虚拟地址空间中段内最大可用偏移位置。它定义了段的长度。段内到limit的地址范围对应线性地址中base~base+limit的范围，当偏移量大于limit则会异常。由两段地址拼接而成的。一共20位。补充：需要和G配合使用。G置位是以4KB作为颗粒度，此时地址范围是4kb-4gb；

G复位是以字节作为颗粒度，此时地址范围是1b-1mb

DPL:描述符特权级。硬件会比较CPL（当前的特权级）和DPL（目标段的特权级）进行特权级的检查。

S：段描述符的类型，置位是代码、数据段，复位是数据段



4、什么是GDT，什么是LDT?

GDT：全局描述符表，是全局唯一的。

存放一些公用的描述符，和包含**各进程局部描述符**表首地址的描述符。

GDT是线性地址空间中的一个数据结构

LDT：局部描述符表，每个进程都可以有一个。

存放本进程内使用的描述符。

5、请分别说明GDTR和LDTR的结构。

GDTR：48位寄存器，高32位放置GDT首地址，低16位放置GDT限长

限长决定了可寻址的大小，注意低16位放的不是选择子

LDTR：16位寄存器，放置一个特殊的选择子，用于查找当前进程的LDT首地址。

6. 请说明GDT直接查找物理地址的具体步骤。

给出段选择子（放在段选择寄存器里）+ 偏移量

若选择了GDT方式，则从GDTR获取GDT首地址，用段选择子中的13位做偏移，拿到GDT中的描述符

如果合法且有权限，用描述符中的段首地址加上1.中的偏移量找到物理地址,寻址结束

7. 请说明通过LDT查找物理地址的具体步骤。

给出段选择子（放在段选择寄存器中）+ 偏移量

若选择了LDT方式，则从GDTR获取GDT首地址，用LDTR中的偏移量做偏移，拿到GDT中的描述符1

从描述符1中获取LDT首地址，用段选择子中的13位做偏移，拿到LDT中的描述符2

如果合法且有权限，用描述符2中的段首地址加上1.中的偏移量找到物理地址。寻址结束

1. 根目录区大小一定么？扇区号是多少？为什么？

不是，需要计算。扇区号是（根目录最大文件数\*32+每扇区字节数-1）/每扇区字节数得到根目录区占用的真正扇区数。扇区号从第19（1+9+9）个扇区开始，长度是前面计算的到的真正扇区数。每一个根目录的大小一定要是扇区字节数的倍数，所以可能有空白的情况。

9、数据区第一个簇号是多少？为什么？

在1.44M软盘上，FAT前三个字节的值是固定的0xF0、0xFF、0xFF，用于表示这是一个应用在1.44M软盘上的FAT12文件系统。本来序号为0和1的FAT表项应该对应于簇0和簇1，但是由于这两个表项被设置成了固定值，簇0和簇1就没有存在的意义了，所以数据区就起始于簇2。

10. FAT表的作用？

FAT表中每12位成为一个FAT项(FATEntry)，代表一个簇。所以2个FAT项会占用3个字节。FAT项的值代表文件的下一个簇号，值大于或等于0xFF8，表示当前簇已经是本文件的最后一个簇，值为0xFF7，表示它是一个坏簇。通过FAT表的值，可以索引到文件的下一簇的位置。

1. 解释静态链接的过程。

相似段合并；重定位。

相似段合并：链接器将所有的目标文件中的相似段（如函数、全局变量等）合并成一个单一的可执行文件。这样，多个目标文件中的相似段只需要在最终的可执行文件中存在一份，避免了代码的冗余。

重定位：链接器会对合并后的目标文件进行符号解析，将函数和全局变量的引用连接到其定义处，以解决跨文件的引用问题。同时，链接器还会对目标文件中的地址（包括绝对地址和相对地址）进行重定位，将其调整为在最终可执行文件中的正确地址。

最终生成独立的可执行文件，无需依赖外部的库文件，便于分发和部署。

1. 解释动态链接的过程。

动态链接器⾃举；装载共享对象；重定位和初始化

动态链接器自举：在程序启动时，操作系统的动态链接器会自动加载到内存中，并负责加载和链接共享库。

装载共享对象：动态链接器根据程序中的链接信息和共享库的依赖关系，将共享库加载到程序的地址空间中。这包括将共享库的二进制代码和数据加载到内存，并分配适当的地址空间。

重定位：在共享库被加载到内存后，动态链接器需要对共享库中的符号进行重定位。

初始化：在共享库被加载到内存后，动态链接器会调用共享库中的初始化函数（如构造函数）进行初始化。

1. 静态链接相关PPT中为什么使用ld链接⽽不是gcc？

为了避免gcc进⾏glibc的链接

使用ld而不是GCC进行链接的一些常见情况包括：

链接其他的C库：有些情况下，用户可能需要链接除了glibc之外的其他C库，如musl libc、uclibc等。这可以通过使用ld手动链接目标文件和相应的C库来实现。

自定义的C库：用户可能希望使用自己编写的C库，而不是系统默认的glibc。这时可以使用ld手动链接自定义的C库。

链接选项设置：使用ld可以手动设置链接选项，如链接时的优化选项、链接器脚本等

15. linux下可执行文件的虚拟地址空间默认从哪里开始分配。

从0x08048000开始分配