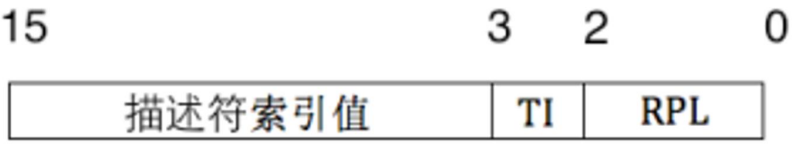
1. **什么是实模式，什么是保护模式?**

实模式：基地址+偏移量可以直接获得物理地址的模式

保护模式：不能直接拿到物理地址,需要进行地址转换

2. **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
- 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



3. **什么是描述符?**

描述一个段是否在内存中；保护模式下引入描述符来描述各种数据段，描述符为 8 字节，有 第五个字节说明描述符的类型。

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。

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。

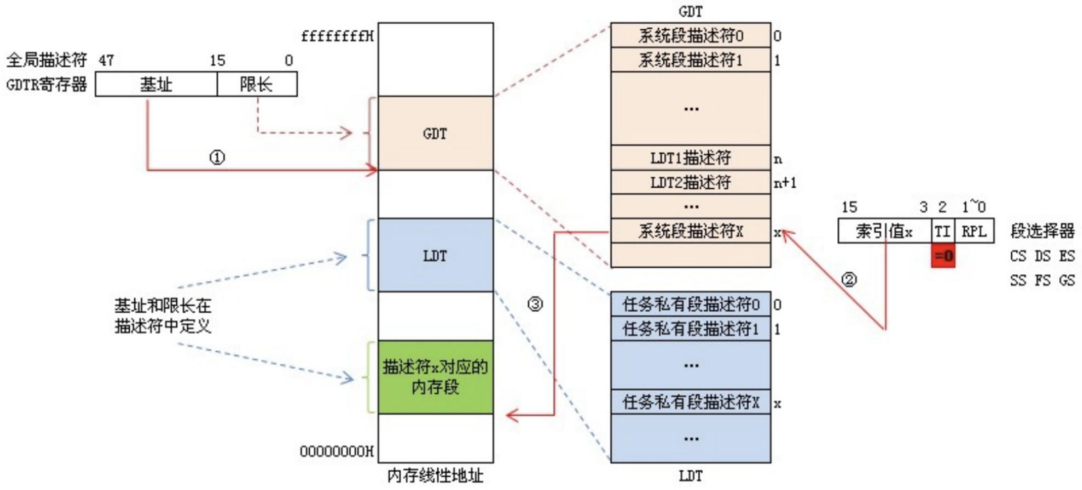
5. **请分别说明 GDTR 和 LDTR 的结构。**

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。

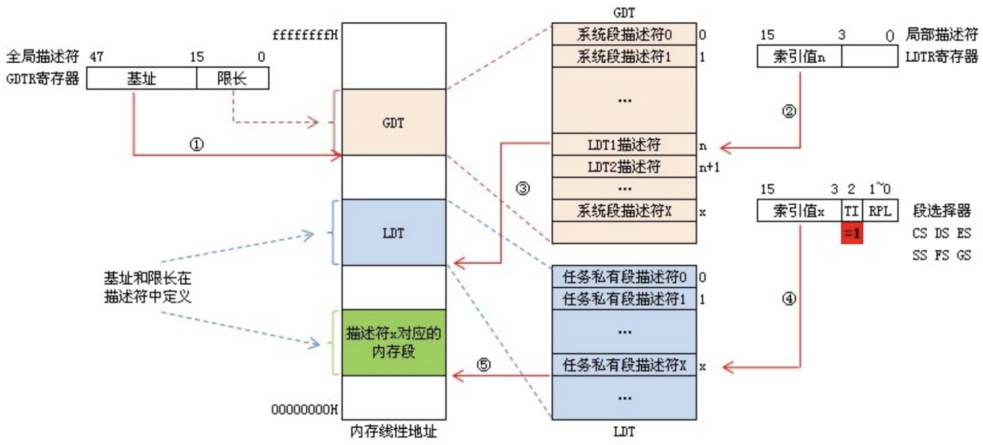


1）给出段选择子（放在段选择寄存器里）+ 偏移量

2）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

3）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。



1）给出段选择子（放在段选择寄存器中）+ 偏移量

2）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

3）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

4）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

8. 根目录区大小一定么？扇区号是多少？为什么？

不一定；19；1(引导扇区) + 9(FAT1) + 9(FAT2) = 19

9. 数据区第一个簇号是多少？为什么？

是 2，因为 FAT 中序号为 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？

文件分配表被划分为紧密排列的若干个表项，每个表项都与数据区中的一个簇相对应，而且表项的序号也是与簇号一一对应的。

FAT 项的值代表文件的下一个簇号

用于描述文件系统内的簇分配状态，说明文件系统内数据所分配的连续簇的顺序关系

11. 解释静态链接的过程。

1）空间和地址分配（相似段合并）

2）符号解析和重定位

12. 解释动态链接的过程。

1）动态链接器自举

动态链接器本身也是一个不依赖其他共享对象的共享对象，需要完成自举。

2）装载共享对象

将可执行文件和链接器自身的符号合并成为全局符号表，开始寻找依赖对象。加载对象的过程可以看做图的遍历过程；新的共享对象加载进来后，其符号将合并入全局符号表；加载完毕后，全局符号表将包含进程动态链接所需全部符号。

3）重定位和初始化

链接器遍历可执行文件和共享对象的重定位表，将它们 GOT/PLT 中每个需要重定位的位置进行修正。完成重定位后，链接器执行.init 段的代码，进行共享对象特有的初始化过程（例如 C++里全局对象的构造函数）。

4）转交控制权

完成所有工作，将控制权转交给程序的入口开始执行。

13. 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

14. 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

1. BPB 指定字段的含义

Offset (decimal)	Offset (hex)	Size (in bytes)	Meaning
0	0x00	3	The first three bytes EB 3C 90 disassemble to JMP SHORT 3C NOP. (The 3C value may be different.) The reason for this is to jump over the disk format information (the BPB and EBPB). Since the first sector of the disk is loaded into ram at location 0x0000:0x7c00 and executed, without this jump, the processor would attempt to execute data that isn't code. Even for non-bootable volumes, code matching this pattern (or using the E9 jump opcode) is required to be present by both Windows and OS X. To fulfil this requirement, an infinite loop can be placed here with the bytes EB FE 90.
3	0x03	8	OEM identifier. The first 8 Bytes (3 - 10) is the version of DOS being used. The next eight Bytes 29 3A 63 7E 2D 49 48 and 43 read out the name of the version. The official FAT Specification from Microsoft says that this field is really meaningless and is ignored by MS FAT Drivers, however it does recommend the value "MSWIN4.1" as some 3rd party drivers supposedly check it and expect it to have that value. Older versions of dos also report MSDOS5.1, linux-formatted floppy will likely to carry "mkdosfs" here, and FreeDOS formatted disks have been observed to have "FRDOS5.1" here. If the string is less than 8 bytes, it is padded with spaces.
11	0x0B	2	The number of Bytes per sector (remember, all numbers are in the little-endian format).
13	0x0D	1	Number of sectors per cluster.
14	0x0E	2	Number of reserved sectors. The boot record sectors are included in this value.

每个扇区的字节数

每个簇的扇区数

Boot record占用的扇区数

16	0x10	1	Number of File Allocation Tables (FAT's) on the storage media. Often this value is 2.
17	0x11	2	Number of directory entries (must be set so that the root directory occupies entire sectors).
19	0x13	2	The total sectors in the logical volume. If this value is 0, it means there are more than 65535 sectors in the volume, and the actual count is stored in the Large Sector Count entry at 0x20.
21	0x15	1	This Byte indicates the media descriptor type.
22	0x16	2	Number of sectors per FAT. FAT12/FAT16 only.
24	0x18	2	Number of sectors per track.
26	0x1A	2	Number of heads or sides on the storage media.
28	0x1C	4	Number of hidden sectors. (i.e. the LBA of the beginning of the partition.)
32	0x20	4	Large sector count. This field is set if there are more than 65535 sectors in the volume, resulting in a value which does not fit in the *Number of Sectors* entry at 0x13.

FAT的数量，一般为2

根目录文件数的最大值

扇区数

一个FAT表的扇区数

36	0x024	1	Drive number. The value here should be identical to the value returned by BIOS interrupt 0x13, or passed in the DL register; i.e. 0x00 for a floppy disk and 0x80 for hard disks. This number is useless because the media is likely to be moved to another machine and inserted in a drive with a different drive number.
37	0x025	1	Flags in Windows NT. Reserved otherwise.
38	0x026	1	Signature (must be 0x28 or 0x29).
39	0x027	4	VolumeID 'Serial' number. Used for tracking volumes between computers. You can ignore this if you want.
43	0x02B	11	Volume label string. This field is padded with spaces.
54	0x036	8	System identifier string. This field is a string representation of the FAT file system type. It is padded with spaces. The spec says never to trust the contents of this string for any use.
62	0x03E	448	Boot code.
510	0x1FE	2	Bootable partition signature 0xAA55.

Boot代码

Magic number 0xAA55

Boot record占据了第一个扇区！

2. 如何进入子目录并输出（说明方法调用）

```
//注意，有的32是byte，并不是32位，注意区分！！！！！！！！！！！！ 一个entry是32byte
/**
 * 处理32byte的entry，但这个entry可能是file的信息，也可能是dir的信息，注意区分【开始的入口，给出一个entry，准备处理】
 * @param father 父目录
 * @param entries 32*8的倍数 一整个目录的字符串 若是root directory 则为512byte 一个二进制串
 */
void traverseDirectoryNode(DirNode* father,string entries){
```

每次按 entry 取一个目录项，并找到对应的数据区。如果目录项指明该数据区是子目录，则递归的再次调用该函数

3. 如何获得指定文件的内容，即如何获得数据区的内容（比如使用指针等）

```
/**
 * 遍历文件链表（若发现子目录，则转过去给traverseDirectory    要改！！！！！！！！！！！！！！！！！！！！
 * @param secNum 起始cluster(sec)号
 * @param root  父文件夹          在调用之前就要把文件的起始node放在root里面
 * @param last 该文件的上一块（链表里）
 */
void traverseFileNode(File* root,FileNode* last,int secNum){
```

有 file 和 fileNode 两个 class

4. 如何进行 C 代码和汇编之间的参数传递和返回值传递

```
//********************* 与nasm联合的部分

extern "C" {
void printStr(char *, int);
}



/**
 * 输出到控制台的函数
 * @param s
 */

void printInNasm(string s) {
    //把string转化为char*
    char* str=new char [s.length()];
    int i=0;
    for(;i<s.length();i++) str[i]=s[i];
    printStr(str, s.length());
}



//******************* end
```

5. 汇编代码中对 I/O 的处理方式，说明指定寄存器所存值的含义

```
global printStr

section .text
printStr:

    mov edx,[esp+8]
    mov ecx,[esp+4];
    mov ebx,1
    mov eax,4
    int 80h
    ret
```