**第八章**

**8.2.1**

文本

AI 生成的内容可能不正确。

在段定义中使用align=子句，用于指定某个SECTION的汇编地址对齐方式。align=16表示段是16字节对齐。

段的汇编地址是段内第一个元素(数据、指令)的汇编地址。为了方便取得段的汇编地址，NASM编译器提供了表达式**section.段名称.start**。

段语句还可以包含vstart=子句，定义了该语句后，标号的汇编地址要从它所在段的开头计算，并且从0开始计算。

如果段语句中不包含vstart=子句，标号的汇编地址要从整个程序开头计算。

图示, 示意图

AI 生成的内容可能不正确。

**8.2.2**

图片包含 图示

AI 生成的内容可能不正确。



头部起码要包含下列信息：

1. 用户程序的尺寸，以字节为单位。
2. 应用程序的入口点，包括段地址和偏移地址。
3. 段重定位表。程序加载到内存后，每个段的地址必须重新确定一下。段的重定位是加载器的工作，它需要知道每个段在用户程序内的位置，即它们分别位于用户程序内的多少字节处。为此，需要在用户程序头部建立一张段重定位表。图片包含 文本

   AI 生成的内容可能不正确。

比如上一张图中，第14行，声明并初始化了段重定位表的项目数。每个表项占用4个字节。紧接着表项数的，是实际的段重定位表，每个表项用伪指令dd声明并初始化为1个双字。

**8.3.1**

图片包含 日程表

AI 生成的内容可能不正确。

用equ声明的数值不占用任何汇编地址，也不在运行时占用任何内存位置，仅仅代表一个数值。



代码8-1用dd伪指令声明了一个内存物理地址，用于加载用户程序。

图示

AI 生成的内容可能不正确。

文本

AI 生成的内容可能不正确。

**8.3.5**

使用LBA28访问硬盘。个人计算机上的主硬盘控制器被分配了8位端口，端口号从0x1f0到0x1f7。

第一步，写入0x1f2端口设置要读取的扇区数量，这是个8位端口，因此每次只能读写255个扇区。(如果写入的值为0，则表示要读取256个扇区)。每读一个扇区，这个数值就减一。因此，如果在读写过程中发生错误，该端口包含着尚未读取的扇区数。

第二步，设置起始LBA扇区号。扇区的读写是连续的，因此只需要给出第一个扇区的编号就可以了。扇区号28位，分成4段分别写入0x1f3、0x14、0x15、0x16号端口。最后4位在0x1f6号端口。

图示

AI 生成的内容可能不正确。

第三步，向端口0x1f7写入0x20，请求硬盘读。

第四步，等待读写操作完成。端口0x1f7既是命令端口，又是状态端口。内部操作期间，将bit7设置1表示忙状态，一旦准备就绪，将bit7清零，同时将bit3置1，表示准备好，请求主机发送或者接收数据。

图示

AI 生成的内容可能不正确。

第五步，连续读出数据。0x1f0是硬盘接口的数据端口，而且还是16位端口。一旦硬盘控制器空闲且准备就绪，就可以连续从这个端口写入或者读取数据。

**8.3.7**

图示

AI 生成的内容可能不正确。

**8.3.8**

用户程序加载到内存后，从物理地址phy\_base开始。

图示

AI 生成的内容可能不正确。

在用户程序头部内，偏移地址为0x6的双字，存放的是入口点代码段的汇编地址。

**第十章**

**10.1.1**

在32位模式下，处理器要求在加载程序时，先定义该程序拥有的段，然后允许使用这些段。定义段时，除了基地址(起始地址)外，还附加了段界限、特权级别、类型等属性。当程序访问一个段时，处理器将用固件实施各种检查工作，以防止对内存的违规访问。

图示, 表格

AI 生成的内容可能不正确。

在32位模式下，传统的段寄存器，CS/SS/DS/ES保存的不再是16位段基地址，而是段的选择子，用于选择要访问的段。除了段选择子外，每个段寄存器还包括一个不可见部分，称为描述符高速缓存器，里面有段的基地址和各种访问属性。这部分内容程序不可访问，由处理器自动使用。

**10.1.3**

图示

AI 生成的内容可能不正确。

段部件产生的地址是物理地址，但当页功能开启时，段部件产生的地址就不再是物理地址了，而是线性地址，线性地址还要经页部件转换后，才是物理地址。

**10.2.5**

在处理器内部，有一个小容量的高速缓存器，叫分支目标缓存器(Branch Target Buffer，BTB)。当指令实际执行时，预测失败，需要清空流水线，同时刷新BTB中的记录。

**10.3.1**

16位指令和32位指令共用相同的指令码，但通过不同的指令前缀，结合处理器当前的运行状态来决定该指令的寻址方式。比如，当处理器运行在16位模式时，如果没有指令前缀0x66，则认为指令是传统的16位寻址方式；如果有指令前缀0x66，则指令是新的32位寻址方式。如果处理器当前运行在32位模式下，且没有前缀0x66，则视为默认的32位寻址方式，否则就是传统的16位寻址方式。

**10.3.2**

为了指明程序的默认运行环境，编译器提供了伪指令bits，用于指明其后的指令应该是被编译成16位的还是32位的。

**第十一章**

**11.2**

和一个段有关的信息需要8个字节来描述，所以称为段描述符，每个段都需要一个描述符。

最主要的描述符表是全局描述符表(Global Descriptor Table，GDT)，所谓全局，意味着该表是为整个软硬件系统服务的。在进入保护模式前，必须要定义全局描述符表。

为了跟踪全局描述符表，处理器内部有一个48位的寄存器，称为全局描述符表寄存器(GDTR)。该寄存器由32bit线性地址和16bit边界组成。32bit线性地址保存的是全局描述符表在内存中的起始线性地址，16bit边界部分保存的是全局描述符表的边界，数值上等于表大小(总字节数)-1。

文本

AI 生成的内容可能不正确。

该表最大是2^16字节，最多可以定义8192个描述符。

图示

AI 生成的内容可能不正确。

**11.3**

在保护模式下，想访问哪个段，在访问之前，必须先在GDT内定义要访问的内存段。

图片包含 表格

AI 生成的内容可能不正确。

段界限20位，用来限制段的扩展范围。

G位是粒度位，用于解释段界限含义。为0时，以字节为单位，为1时，以4KB为单位，前者扩展范围是1字节到1MB，后者扩展范围是4KB到4GB。

S用于指定描述符的类型，为0时，表示是一个系统段，为1时，表示是一个代码段或者数据段。

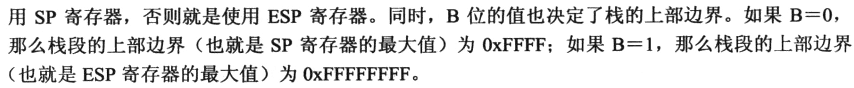
DPL表示描述符的特权级。0~3，其中0是最高特权级别，3是最低特权级别。

P是段存在位，用于指示描述符对应的段是否存在。0表示描述符虽然建立，但是对应的内存空间并不存在，比如内存空间紧张的时候，会将段换出到硬盘中。

D/B位是默认的操作数大小或者默认的栈指针大小，又或者上部边界标志。该标志位主要是为了能够在32位处理器上兼容运行16位保护模式的程序。

文本

AI 生成的内容可能不正确。



L是64位代码段标志。

表格

AI 生成的内容可能不正确。

X是可执行，数据段不可执行，代码段可执行。数据段的E指示段扩展方向，0向上扩展，1向下扩展。W表示段是否可写。

代码段的C指示特权级依从，0表示非依从，只能从与其特权级相同或者门调用，1允许从低特权级转移到该段执行。R表示代码段是否允许读出。

**11.4**

ldgt m48

该指令操作数是一个48位的内存区域，在16位模式下，地址是16位的，32位模式下，地址是32位的。实模式和保护模式下都可以执行。这6字节的区域中，低16bit是GDT的界限值是，高32bit是GDT的基地址。

**11.6**

实模式和保护模式的切换是CR0寄存器。CR0是处理器内部的控制寄存器。包含了一系列用于控制处理器操作模式和运行状态的标志位。biit0是保护模式允许位。将该位置1，处理器进入保护模式。

图片包含 表格

AI 生成的内容可能不正确。

保护模式下的中断机制和实模式不同，原有的中断向量表不再适用。在保护模式下，BIOS中断都不能再用，在重新设置保护模式下的中断环境之前，必须关中断。

表格

AI 生成的内容可能不正确。

实模式下，访问内存使用逻辑地址，段地址\*16后加上偏移地址。

在32位处理器上，引用一个段时，处理器自动将段地址左移4位，并传送到描述符高速缓存器中。但是实模式下，向段处理器传送16位逻辑段地址(处理器不把它看成是描述符选择子)，只能访问1MB内存。在实模式下，段寄存器描述符高速缓存器的内容仅低20bit有效，高12bit全0。

保护模式下，传送到段选择器的内容不是逻辑段地址，而是段描述符在描述符表中的索引号。

段选择子由三部分组成，描述符索引号，TI描述符指示器，RPL请求特权级，表示给出当前选择子的那个程序的特权级别。

表格

AI 生成的内容可能不正确。

GDT的线性基地址在GDTR中，每个描述符占8个字节，描述符在表内的偏移地址是索引号\*8。

图示

AI 生成的内容可能不正确。

**11.7**

进入到保护模式前，使用转移指令清空流水线，并串行化执行，同时重新加载段选择器CS，并刷新描述符高速缓存器中的内容。code11-1中设置CR0的PE位之后，立即用jmp或者call转移到当前指令流的下一条指令上。

**14.1**

程序是记录在载体上的指令和数据，总是为了完成某个特定的工作，其正在执行中的一个副本，叫做任务。如果一个程序有多个副本正在内存中运行，那么，它对应多个任务，每一个副本都是一个任务。

为了有效地在任务之间实施隔离，处理器建议每个任务都应当具有自己的描述符表，称为局部描述符表LDT(Local Descriptor Table)，并且把专属于自己的那些段放在LDT中。LDT不同于GDT，只属于某个任务。

图示

AI 生成的内容可能不正确。

LDTR：局部描述符表寄存器，指向当前任务的LDT。发生任务切换时，LDTR的内容被更新，以指向新任务的LDT。和GDTR一样，LDTR包含了32位线性基地址字段和16位段界限字段，以指示当前LDT的位置和大小。图11-10中，段选择子bit2是表指示器TI，TI为0从GDT中加载描述符，TI为1从当前任务的LDT中加载描述符。

TSS：任务状态段(Task State Segment)，为了保存任务的状态，并在下次重新执行时恢复它们，每个任务都应当用一个额外的内存区域保存相关信息。

表格

AI 生成的内容可能不正确。

图11-4中，描述符有两bit的DPL字段，对应特权级0~3。DPL是每个描述符都有的字段，又称描述符特权级。描述符总是指向它所描述的目标对象，代表着该对象，因此，该字段实际上是目标对象的特权级。

只有在当前特权级CPL为0时才能执行的指令，称为特权指令。典型的特权指令包括加载全局描述符表的指令ldgt、加载局部描述符表的指令lldt、加载任务寄存器的指令ltr、读写控制寄存器的mov指令、停机指令hlt等十几条。

在处理器的标志寄存器EFLAGS中，bit13、bit12是IOPL位，也就是输入输出特权级(I/O Privilege Level)，它代表着当前任务的I/O特权级别。

图片包含 图示

AI 生成的内容可能不正确。

代码段的特权级检查是很严格的。一般控制转移只允许发生在两个特权级相同的代码段之间。不过，为了让特权级低的应用程序可以调用特权级高的操作系统例程，处理器也提供了相应的解决办法：

1. 将高特权级的代码段定义为依从的。可以从特权级比它低的程序调用并进入。该方式要求当前特权级CPL必须低于或等于目标代码段描述符的DPL。也就是控制从较低的特权级转移到较高的特权级，不允许反向操作。依从的代码段

图示

AI 生成的内容可能不正确。