这部分代码通过键盘控制器的端口0x64发送命令0xd1来访问端口0x60，然后向0x60写入0xdf来启用A20地址线，这使得CPU能够访问超过1MB的内存。waitkbdout函数用于等待键盘控制器不忙。

### **禁用中断**

**asm**复制代码

|  |  |
| --- | --- |
|  | MOV AL,0xff |
|  | OUT 0x21,AL |
|  | NOP ; 不断执行OUT指令（这里NOP可能是误加，因为通常不需要重复OUT） |
|  | OUT 0xa1,AL |
|  | CLI ; 进一步中断CPU |

这段代码通过向PIC（可编程中断控制器）的端口0x21和0xa1写入0xff来禁用所有中断。然后，通过CLI指令禁用CPU的中断。

### **设置A20地址线**

**asm**复制代码

|  |  |
| --- | --- |
|  | CALL waitkbdout |
|  | MOV AL,0xd1 |
|  | OUT 0x64,AL |
|  | CALL waitkbdout |
|  | MOV AL,0xdf ; enable A20 |
|  | OUT 0x60,AL |
|  | CALL waitkbdout |

这部分代码通过键盘控制器的端口0x64发送命令0xd1来访问端口0x60，然后向0x60写入0xdf来启用A20地址线，这使得CPU能够访问超过1MB的内存。waitkbdout函数用于等待键盘控制器不忙。

### **进入保护模式**

**asm**复制代码

|  |  |
| --- | --- |
|  | [INSTRSET "i486p"] ; 说明使用486指令 |
|  | LGDT [GDTR0] ; 设置临时GDT |
|  | ... |
|  | MOV CR0,EAX |
|  | JMP pipelineflush |
|  | pipelineflush: |
|  | ... |

这部分代码首先设置全局描述符表（GDT），然后修改CR0寄存器的位来禁用分页（通过AND操作）并启用保护模式（通过OR操作）。JMP pipelineflush用于清除CPU的指令流水线，确保接下来的指令在保护模式下执行。

### **加载和设置bootpack**

**asm**复制代码

|  |  |
| --- | --- |
|  | MOV ESI,bootpack ; 源 |
|  | MOV EDI,BOTPAK ; 目标 |
|  | ... |
|  | CALL memcpy |
|  | ... |
|  | MOV ESP,[EBX+12] ; 堆栈的初始化 |
|  | JMP DWORD 2\*8:0x0000001b |

这段代码使用memcpy函数将bootpack（操作系统核心）从某个位置复制到内存中的另一个位置（BOTPAK）。然后，它设置堆栈指针（ESP）并跳转到bootpack的入口点（0x0000001b），这里假设bootpack的入口点位于某个特定的段偏移量。

### **辅助函数**

* waitkbdout：等待键盘控制器不忙，确保可以安全地向其发送命令。
* memcpy：一个简单的内存复制函数，用于在内存中复制数据块。

### **GDT设置**

**asm**复制代码

|  |  |
| --- | --- |
|  | GDT0: |
|  | RESB 8 ; 初始值 |
|  | DW 0xffff,0x0000,0x9200,0x00cf ; 写32bit位段寄存器 |
|  | DW 0xffff,0x0000,0x9a28,0x0047 ; 可执行的文件的32bit寄存器（bootpack用） |
|  | ... |
|  | GDTR0: |
|  | DW 8\*3-1 |
|  | DD GDT0 |

这部分代码定义了全局描述符表（GDT）的内容，包括两个段描述符：一个用于数据段，另一个用于代码段（bootpack）。GDTR0结构体包含了GDT的基地址和大小，这些信息通过LGDT指令加载到GDT寄存器中。

整体而言，这段代码展示了从实模式到保护模式的转换过程，以及如何在保护模式下加载和启动一个简单的操作系统核心。