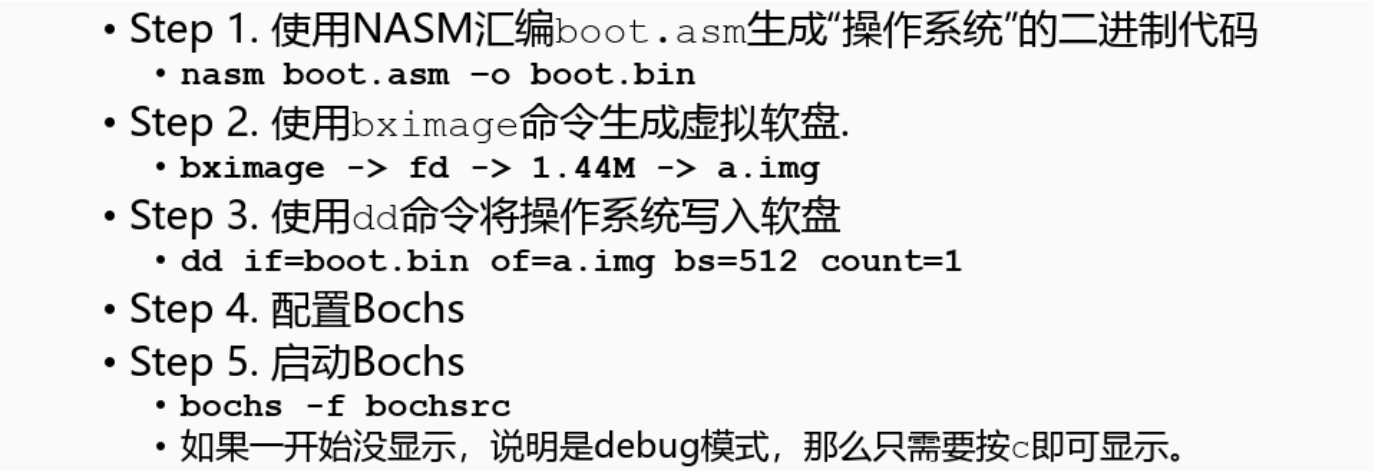
第一次实验

实验任务一：Hello OS

- 任务。选择任意平台，参考讲义搭建NASM+Bochs实验平台，在该实验平台上汇编boot.asm，并用Bochs虚拟机运行显示Hello,OS。
- 要求。理解boot.asm、bochsrc中的内容，理解实验过程中涉及的所有命令选项参数的含义。在Moodle上提交代码与运行截图。在不依赖助教提示查看讲义的情况下复现实验（包括制作镜像启动虚拟机等步骤）。

实验步骤

- 安装bochs虚拟机以及bochs-x图形库。`sudo apt-get install bochs bochs-x。`（tip：apt-get下载安装后的文档一般在/usr/share中、可执行文件在/usr/bin中、配置文件在/etc中、lib文件在/usr/lib中……）
- 制作一个软盘，作为装载操作系统的存储设备。命令行输入`bximage`进入交互页面，根据提示输入：创建软盘还是硬盘，选软盘fd；软盘大小，默认值1.44MB；名称，默认a.img。（tip：在哪执行bximage，软盘就创建在了哪里，我使用共享文件夹，放在了/mnt/hgfs/OSLAB/LAB1中）
- 安装NASM汇编器。`sudo apt-get install nasm`
- 写一个基本的hello,OS程序boot.asm。（代码略）
- 使用NASM来汇编boot.asm，生成boot.bin二进制代码。`nasm boot.asm -o boot.bin`
- 将boot.bin写入“软盘”。使用dd命令（用于读取、转换并输出数据）。`dd if=boot.bin of=a.img bs=512 count=1`，is代表输入文件，of代表输出设备，bs代表一个扇区大小，count代表扇区数。
- 配置并启动虚拟机。在当前位置创建名为bochsrc的配置文件（内容略），然后`bochs -f bochsrc`进入bochs的debug模式，再输入c即可。
- 验收时的步骤。



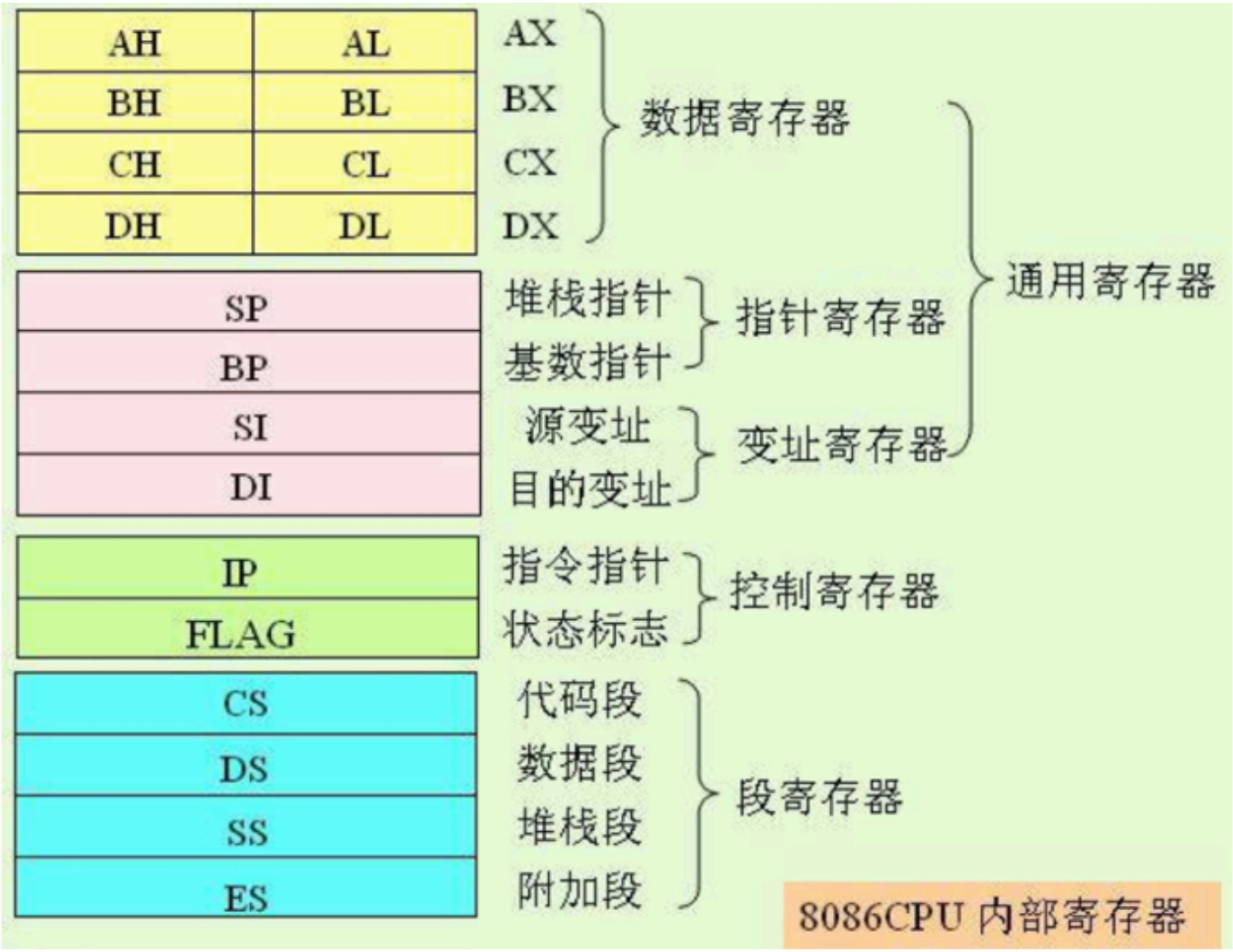
实验任务二：进制转换

- 使用NASM汇编语言实现进制转换。该任务要求原创，不得抄袭！
- 格式要求。输入格式：<十进制数> <转换类型>，两者中间有一个空格；<十进制数>的范围是[0, 10^30]，<转换类型> 包含二进制b、八进制o和十六进制h；输入q结束程序。输出格式：转换结果，二进制以0b开头，八进制以0o开头，十六进制以0x开头，字母均为小写，输出结果后继续接收下一个输入。
- 至少处理1种无效输入情况，如无效的数字、无效的转换类型或缺少转换类型等，保证程序不崩溃。

问题

Q1

- 问：8086有哪5类寄存器？ 请分别举例说明其作用。
- 答：



- 数据寄存器、指针寄存器、变址寄存器、控制寄存器、段寄存器，其中前三者属于通用寄存器。
- AX，BX，CX，DX 称作为数据寄存器。
  AX (Accumulator)：累加寄存器。BX (Base)：基地址寄存器。CX (Count)。计数器寄存器。DX

(Data)：数据寄存器；

- SP 和 BP 又称作为指针寄存器。SP (Stack Pointer)：堆栈指针寄存器。BP (Base Pointer)：基指针寄存器。
- SI 和 DI 又称作为变址寄存器。SI (Source Index)：源变址寄存器。DI (Destination Index)：目的变址寄存器。
- 控制寄存器。IP (Instruction Pointer)：指令指针寄存器。FLAG：标志寄存器。
- 段寄存器。CS (Code Segment)：代码段寄存器。DS (Data Segment)：数据段寄存器。SS (Stack Segment)：堆栈段寄存器。ES (Extra Segment)：附加段寄存器。

Q2

- 问：有哪些段寄存器， 它们的作用是什么?
- 答：8086有4个段寄存器，分别是CS、DS、SS、ES。它们的作用如下：
- CS（Code Segment）：存储当前执行指令所在代码段的起始地址。
- DS（Data Segment）：存储当前数据的地址，如变量、数组等。
- SS（Stack Segment）：存储堆栈的底部地址。
- ES（Extra Segment）：存储其他数据段的地址。

Q3

- 问：什么是寻址？ 8086有哪些寻址方式?
- 答：寻址是找到操作数的地址，从而能够取出操作数。
- 立即寻址：指令里直接给出操作数（立即数），MOV AX 1234H。直接寻址：指令里直接给出操作数的地址，根据地址取操作数，MOV AX [1234H]。
- 寄存器寻址：给出寄存器名，操作数在寄存器中，MOV AX BX。寄存器间接寻址：操作数地址在寄存器中，MOV AX [BX]。寄存器相对寻址：寄存器中的数作为基址，加上偏移量得到地址，然后取访问此地址，MOV AX [SI+3]。
- 基址加变址：基址寄存器(BX、BP)的内容加上变址寄存器(SI、DI)的内容得到地址，MOV AX [BX+DI]。相对基址加变址：基址+变址+偏移量，MOV AX [BX+DI+3]。

Q4

- 问：主程序与子程序之间如何传递参数?
- 答：用寄存器传参（能传递的参数有限）、用约定的地址传参、用堆栈传参（常用，将参数压入堆栈，然后在子程序中使用POP指令取出）

Q5

- 问：解释boot.asm文件中 org 07c00h 的作用。 如果去掉这一句， 整个程序应该怎么修改?
- 答：告诉汇编器，当前这段代码会放在7c00位置，后续绝对寻址就是该地址加上相对地址。org是伪指令，自身不生成机器码，即使没有它，BIOS也会将代码加载到7c00位置；它在编译期影响到内存寻址指令的编译，让编译器从相对地址7c00开始编译第一条指令，相对地址被编译加载后就

正好和绝对地址吻合；如果去掉这句，boot.bin反编译出来的结果是一样的，但编译器是从0000开始计算偏移量，会因为访问的地址错误而出现乱码。修改：代码里的相对寻址改成绝对寻址，也就是原先代码里的 `mov ax, BootMessage` 修改为 `mov ax, BootMessage+07c00h`

Q6

- 问：解释 `int 10h` 的功能。
- 答：int 10h是一条系统调用指令，用于调用BIOS中提供的显示相关功能。通过传递不同的功能号和参数，可以实现屏幕显示、光标控制、颜色设置等操作。

Q7

- 问：解释boot.asm文件中 times 510-(-$) db 0 的作用。
- 答：这行代码的作用是填充代码的末尾，使整个文件的大小为 512 字节，即一个扇区的大小。`$` 表示当前位置，`$$` 表示代码的起始位置，`510-($-$$)` 表示需要填充的字节数，`db 0` 表示填充的数据为 0。

Q8

- 问：解释bochsrc中各参数的含义。
- 答：megs: 虚拟机的内存大小，这里设置为32MB。display_library: 显示模块，这里设置为sdl2，表示使用SDL库进行显示。floppya: 软盘A的设置，这里1_44=a.img表示使用1.44MB的软盘映像文件a.img作为软盘A，status=inserted表示将软盘插入虚拟机的软盘驱动器中。boot: 启动设备的类型，这里设置为floppy，表示从软盘启动。

Q9

- 问：boot.bin 应该放在软盘的哪一个扇区？ 为什么?
- 答：第一扇区。BIOS加电自检结束后，检查软盘的第0磁头第0磁道第1扇区，如果扇区以 `0xaa55` 结尾，则认定为引导扇区，将此扇区数据复制到物理内存地址 `0x7c00h` 处，然后设置PC，从 `0x7c00` 开始执行代码。因此把boot.bin放在第一扇区，就能直接将其加载到内存中执行。

Q10

- 问：为什么不让Boot程序直接加载内核，而需要先加载Loader再加载内核?
- 答：要做的事情除了加载内核，还要准备保护模式等；一个扇区大小只有512字节，Boot程序无法直接加载并启动整个操作系统。为了突破512字节的限制，我们让loader加载入内存、并把控制权交给它，由loader将内核加载入内存。

Q11

- 问：Loader的作用有哪些?

- 答：Loader程序是操作系统启动过程中的第二个程序，其主要作用是加载内核并跳转到内核的入口点开始执行。具体有：1.跳入保护模式：原先的x86处理器是16位、寄存器用ax、bx等表示，后来扩充成32位、寄存器为eax、ebx等，为了向前兼容而提出了保护模式，必须跳入保护模式才能访问1M以上的内存。2.启动内存分页：实现虚拟内存与物理内存之间的映射。3.加载并运行内核：从kernel.bin中读取内核并放入内存，然后跳转到内核所在的开始地址运行内核（有许多无关信息，不能把整个kernel.bin放在内存，如Linux下是以ELF文件的格式读取并提取代码）。

Q12

- 问：Kernel的作用有哪些?
- 答：Kernel是操作系统的核心部分，其主要作用是管理硬件资源和提供系统服务。具体有：1.进程管理：Kernel负责管理系统中的进程，包括创建、调度、销毁等操作。2.内存管理：Kernel负责管理系统中的内存资源，包括内存分配、回收等操作。3.文件系统：Kernel负责管理系统中的文件系统，包括文件读写、目录管理等操作。4.网络通信：Kernel负责管理系统中的网络通信，包括TCP/IP协议栈的实现、网络设备驱动等操作。5.设备驱动：Kernel负责管理系统中的设备驱动程序，包括硬盘、网络设备、打印机等设备的驱动程序实现。6.系统调用：Kernel提供一些系统调用接口，供应用程序调用，例如文件读写、网络通信等操作。