

## Section 1 实验概述

在本次实验中，将会学习并用到 **x86** 汇编相关知识、计算机的启动过程、**IA-32** 处理器架构和字符显示原理。根据所学的知识，自己编写 **x86** 汇编程序，然后使用 **qemu** 模拟计算机启动，计算机启动后加载程序运行，完成本次实验能够增进对计算机启动过程的理解，为后面编写操作系统加载程序奠定基础。同时，本次实验将继续学习使用 **gdb** 来调试程序以及 **gdb** 的相关调试指令。

## Section 2 预备知识与实验环境

- 预备知识：**x86** 汇编语言、**IA-32** 处理器体系结构、字符显示的原理
- 实验环境：
  - 虚拟机版本/处理器型号：**Virtualbox7.0.6**、**Ubuntu18.04**
  - 代码编辑环境：**gedit** 文本编辑器、**gdb** 调试器、**qemu**
  - 代码编译工具：**nasm**

## Section 3 实验任务

- 实验任务 1：编写 **MBR** 用以操作系统的启动 **Hello World**
- 实验任务 2：编写一个字符弹射汇编小程序
- 实验任务 3：实模式中断
- 实验任务 4：汇编代码的编写

## Section 4 实验步骤与实验结果

### ----- 实验任务 1 -----

- 任务要求：编写 **MBR** 的代码，在 **MBR** 被加载到内存地址 **0x7c00** 后，向屏幕输出青色的 **Hello World**
- 思路分析：了解了 **MBR** 程序在计算机启动之后就会开始执行，同时 **IA-32** 处理器将 **25×80** 的显示像素点映射到了内存地址 **0xB8000~0xBFFFF** 处，其中每一个字符由两个字节来表示，其中低字节表示要输出的字符，然后高字节表示字符的颜色和属性，所以为了向屏幕输出字符，我们只需要从 **0x8000** 开始，按照字符的顺序依次将字符和他的属性赋值到地址种就可以显示字符。

- 实验步骤:

**Step1:** 在开始我们的正式实现功能之前我们需要进行一些初始化工作:

`org` 是一个伪代码指令表示我们的下面的代码是从 `0x7c00` 开始的

`[bits 16]` 也是一个伪指令表示代码是按照 16 位格式编译的

由于段寄存器不能够进行用立即数赋值, 我们借助 `ax` 进行初始化

```
org 0x7c00
[bits 16]
xor ax,ax
;初始化段寄存器, 段地址全部设为 0
mov ds,ax
mov ss,ax
mov es,ax
mov fs,ax
mov gs,ax
```

**Step2:** 同时借助 `ax` 对段寄存器 `gs` 赋值为 `0xB800`, 这里需要注意的是并没有将 `gs` 赋值为 `0xB8000` 是因为 `x86` 架构中通过段寄存器访问地址的时候会先将 `gs` 中的值左移 4 位, 所以这里的初始值给了 `0xB800`

```
mov sp, 0x7c00
mov ax, 0xb800
mov gs, ax
```

**Step3:** 使用 `ax` 寄存器对 `0xB8000` 显存区域进行赋值用于打印字符, 首先对 `ax` 寄存器的高 8 位赋值成 `03H` 表示我们输出的字符颜色是青色的, 然后直接将我们需要显示的字符赋值到 `ax` 的第八位 `al`, 这里的赋值是字符的 `ASCII` 码, 然后我们将 `ax` 赋值到 `0xB8000` 处, 表示我们在屏幕第一个位置上输出青色的 `H`。

```
mov ah, 0x03 ;青色
mov al, 'H'
mov [gs:2 * 0], ax
```

后面的代码, 我们只要修改上述代码的 `al` 的值, 将其改成我们的需要显示的字符, 而我们修改 `gs` 后面的偏移量, 修改我们需要显示的位置, 就可以实现在我们想要的位置输出我们想要的字符。

**Step4:** 然后使用 `nasm` 将我们的代码编译成二进制文件, 打开命令行, 进入到我们代码文件所在的目录, 输入下面的代码,

```
nasm -f bin mbr.asm -o mbr.bin
```

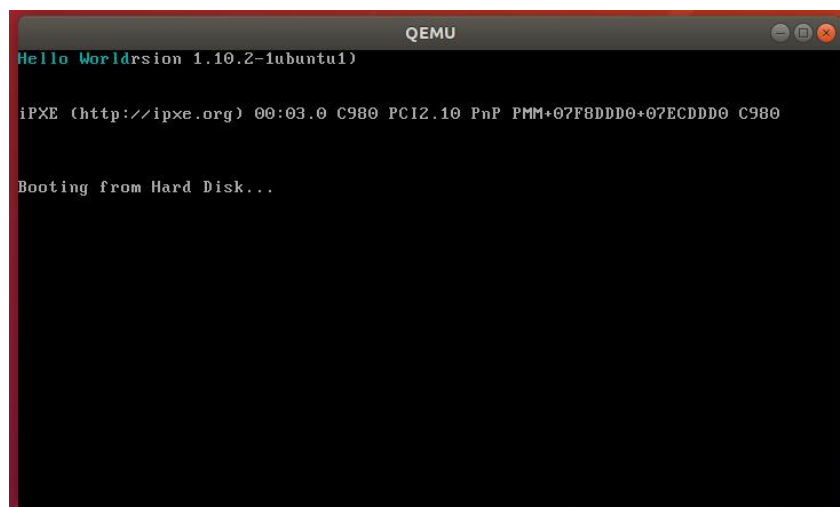
**Step5:** 使用 `qemu` 创建一个虚拟的磁盘，然后使用 `dd` 指令将我们所写 `mbr` 的写入我们创建的虚拟磁盘的首扇区，这里创建了一个 `10m` 的磁盘 `hd.img`

```
qemu-img create hd.img 10m
dd if=mbr.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
```

**Step6:** 最后似乎用 `qemu` 模拟计算机启动，运行我们写于的 `mbr` 可以看到我们写的代码的打印 `hello world` 程序的效果。

```
qemu-system-i386 -hda hd.img -serial null -parallel stdio
```

- 实验结果展示：通过执行前述代码，可得下图结果。可以看到第一行打印了一行青色的 `Hello World`,



## ----- 实验任务 2 -----

- 任务要求：编写一个字符弹射程序，其从点 $(2,0)$ 处开始向右下角  $45^\circ$  开始射出，遇到边界反弹，反弹后按  $45^\circ$  角射出，方向视反弹位置而定。同时，加入变色，双向射出的效果。

- 思路分析：

在屏幕上输出字符如果使用实验任务 1 的方式通过往 `0x8c000` 写入地址的方式可能会比较复杂，这里我们通过 `int 10H` 的实模式中断实现字符的显示，实模式中断的好处就是方便我们控制字符的位置，我们并不需要计算地址，而直接使用坐标就能够找到屏幕的位置。

同样的我们可以使用 `03H` 功能获取光标的位置，然后在这基础上对光标的坐标进行改变，然后直接调用 `02H` 中断移动光标的位置，使用 `09H` 打印字符串，对于颜色和坐标的变化，可以使用自增自减的方式改变。

- 实验步骤:

**Step1:** 首先使用 06H 功能中断实现清屏效果, 然后我们需要初始化的我们的颜色和方向, `dirc_row` 表示行增量, 表示我们的 `y` 轴的坐标是不断增加, 即向右移动, 若为 -1 那么就是坐标向左移动, 同样的 `dirc_col` 为 1 时表示 `x` 会向右移动, 为 -1 时会想左移动, 初始的颜色我们设置为 00, 初始的字符也设置为 '0'

```
;初始化方向和显示的数字及其颜色
dirc_row db 1
dirc_col db -1
mov al,'0'
mov bl,0x00
```

**Step2:** 实现双向射出, 为了实现双向射出, 我们需要一次循环中同时打印两个坐标的值, 这里我们利用栈来保存我们的现场的所有寄存器的值, 然后计算当前位置的中心对成的坐标, 使用 02H 功能移动光标到当前坐标, 使用 09H 功能输出当前的字符, 然后使用 `popad` 恢复我们寄存器之前的状态, 重复我们的打印字符的操作, 具体代码如下:

```
loop:
;将寄存器存入栈中
pushad
;计算与当前中心对称的位置坐标并输出字符
sub dl,79
neg dl      ;dl=79-dl
mov ah,0x02
mov bh,0
int 0x10    ;dh=24-dh
mov ah,0x09
mov cx,1
int 0x10
;出栈恢复之前的寄存器中的值
popad
;2H 移动光标到新位置,9H 显示字符
mov ah,0x02
mov bh,0
int 0x10

mov ah,0x09
mov cx,1
int 0x10
```

**Step3: 实现延迟等待**，为了是我们打印字符的动态效果更加的明显，这里我们再每次打印了一个字符之后就利用循环，使得这一段时间内 CPU 并没有进行实际的操作达到延迟效果，由于 16 位的数字所产生的延迟效果还是不够，这里使用了双重循环的方法来增加延迟等待时间

```
;双重循环使 CPU 空转以达到延迟效果
    mov cx, 0x300    ;外层循环次数
out_delay:
    mov si, 0xFFFF   ;内层循环次数

in_delay:
    dec si
    jnz in_delay
    dec cx
    jnz out_delay
```

**Step4: 调整下一次打印字符的方向和字符以及字符的颜色**，使用条件判断语句，来判断下一次循环的所有参数，如果达到了上下边界，那么就令 `dirc_row` 的值取反，如果达到了左右边界，就令 `dirc_col` 的值取反来改变方向，然后使得我们的字符在 0 到 9 之间不断循环，颜色从 00 到 FF 间不断的循环，具体如下：

```
;调整方向和字符以及颜色
adjust_char:
    cmp al, '9'
    je reset_char
adjust_color:
    cmp bl, 0xFF
    je reset_color
adjust_row:
    cmp dh, 24
    je bounce_row
    cmp dh, 0
    je bounce_row
adjust_col:
    cmp dl, 79
    je bounce_col
    cmp dl, 0
    je bounce_col
    jmp continue
reset_char:
    mov al, '0'
```

```

        jmp adjust_color
reset_color:
        mov bl,0x00
        jmp adjust_row
bounce_row:
        neg byte [dirc_row]
        jmp adjust_col
bounce_col:
        neg byte [dirc_col]
        jmp continue

```

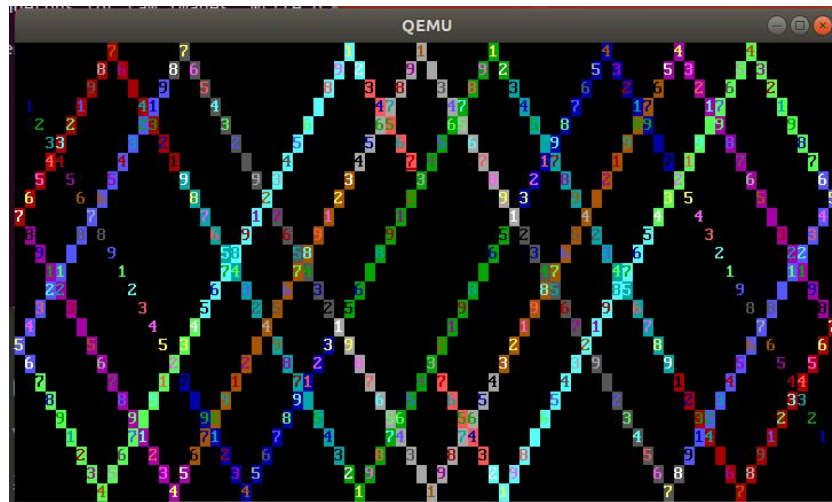
**Step5: 实现颜色和字符的变换，更新下一次循环所需要的参数，包括 dh 行坐标，dl 列坐标，bl 字符颜色，al 输出的字符**

```

;更新新的位置和颜色
continue:
        add dh,[dirc_row]
        add dl,[dirc_col]
        add bl,00000001b
        add al,1
        jmp loop

```

- 实验结果展示：通过执行前述代码，可得下图结果。在提供的源代码 Code 文件夹中还提供了程序执行的动态效果



### ----- 实验任务 3 -----

- 任务要求：
  - ①探索实模式下的光标中断，利用中断实现光标的位置获取和光标的移动。
  - ②请修改 Hello World 的代码，使用实模式下的中断来输出你的学号。
  - ③探索实模式的键盘中断，利用键盘中断实现键盘输入并回显

- 思路分析:

①实模式下光标的中断中主要通过 03H 功能号获取光标的位置，通过 02H 功能号实现光标的移动

②实模式中断下的输出学号，我们需要重复使用光标的移动功能和使用 09H 功能号实现字符的输出功能，通过查阅资料，发现了一种更加方便的方法，就是使用 0EH 功能号来显示字符，它能够在显示字符之后自动将光标移动到下一个位置，这样比用 02H 功能自己移动光标的方式简单很多。

③实模式下的键盘中断主要通过 int 16H 中断功能中的 00H 号功能进行从键盘中获取键盘的输入，需要注意的是这里默认的键盘扫描码是小键盘的扫描码也就是我们输入键盘右边的九宫格数字键可能会出现问题。

一般地，中断的调用方式如下，

|  |
|--|
| 将功能号写入 ah，将参数写入其他参数<br>int 中断号<br>从寄存器中取出返回值 |
|--|

- 实验步骤:

①实模式光标的移动和位置的获取:

**Step1:** 首先使用 02H 功能号实现光标的移动，这里移动到(8, 8)的位置，给 dh 赋值为 8，表示第八行，将 dl 赋值为 8，表示第八列，实现如下，

|  |
|--|
| <pre>mov ah,0x02 mov bh,0 mov dh,8 mov dl,8 int 0x10</pre> |
|--|

**Step2:** 使用 03H 功能号获得光标的位置，这时候光标的行坐标会存到 dh，列坐标会存到 dl，我们需要将他取出，用于后面的输出

|  |
|--|
| <pre>mov ah,0x03 mov bh,0 int 0x10</pre> |
|--|

**Step3:** 使用 0E 功能号一次输出光标的行和列 8 和 8

|   |
|---|
| <pre>mov al,dh add al,'0' mov ah,0x0e mov bl,10001111b mov cx,1</pre> |
|---|

```
int 0x10

mov al,dl
add al,'0'
mov ah,0x0e
mov bl,10001111b
mov cx,1
int 0x10
```

## ②实模式中断下字符输出：

**Step1:** 使用 **0E** 功能号实现字符的打印以及光标的移动，正如前面所说，**0E** 功能号相对于 **09** 功能号的好处是 **0E** 在输出完字符之后，会自动的将光标移动到下一个位置，其他的与 **09** 完全一致，下面是一个打印 2 的例子

```
mov al,'2'
mov ah,0x0e
mov bl,10001111b
mov cx,1
int 0x10
```

**Step2:** 只要将上面的 **al** 的值替换成我们想要打印的字符就可以实现其他字符的输出，我们重复上面的代码依次输出 **23320104** 即可

## ③实模式键盘中断：

与前面的光标中断所用的 **int 10H** 中断服务不同，通过查阅资料 **BIOS** 通过 **int 16H** 中断服务实现关于键盘中断的

**Step1:** 首先使用 **int 10H** 中的 **6H** 功能实现清屏效果

```
;功能号 6H 实现清屏操作
mov ah, 0x06
mov al, 0
mov bh, 0x07
mov ch, 0    ;左上角行号
mov cl, 0    ;左上角列号
mov dh, 24   ;右下角行号
mov dl, 79   ;右下角列号
int 0x10
```

**Step2:** 使用 **int 16H** 中的 **0** 号功能实现从键盘上读入字符，这时候键盘上读入的字符会保存在 **al** 寄存器中

```
mov ah,0x0
int 16H
```

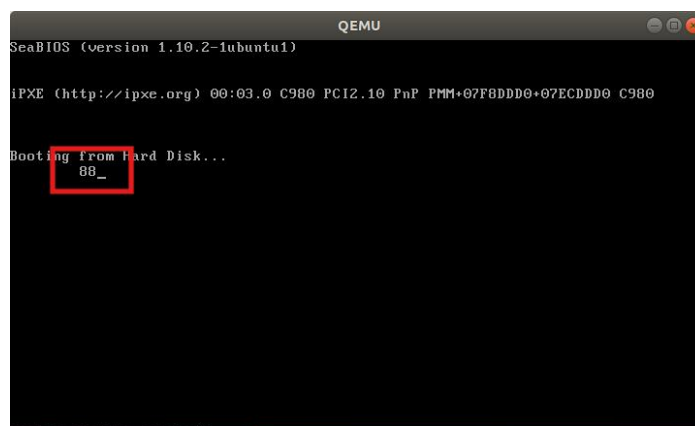


**Step3:** 用 int 10H 中的 0E 功能实现读入字符的回显

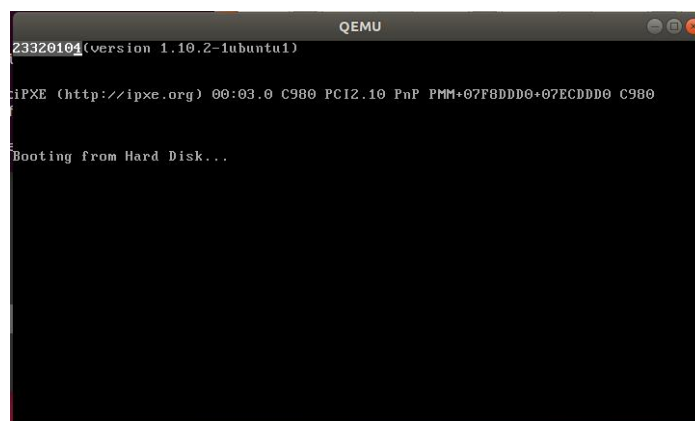
```
mov ah,0x0e
mov bl,10001111b
mov cx,1
int 0x10
jmp loop
```

- 实验结果展示：通过执行前述代码，可得下图结果，

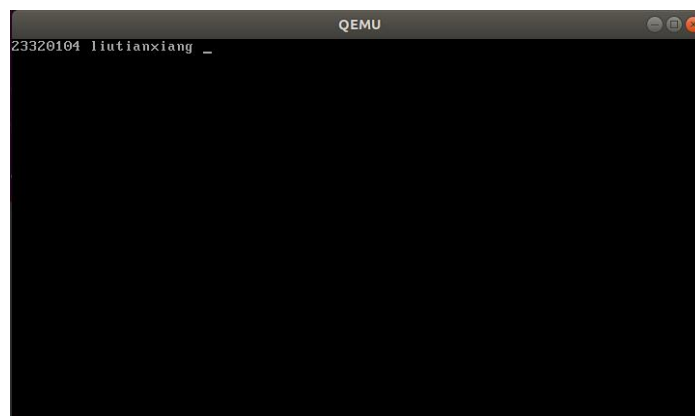
①实模式光标中断：



②实模式中断下字符输出：



③实模式键盘中断：在提供的源代码中还有程序执行的动态效果



## ----- 实验任务 4 -----

- 任务要求：汇编代码的编写寄存器，使用 32 位的寄存器，实现的代码文件中的 `assignment/student.asm` 中的分支逻辑、循环逻辑、函数逻辑的实现，编写好代码之后，在目录 `assignment` 下使用命令 `make run` 即可测试，不需要使用 `qemu` 启动，附上 `make run` 的截图

- 思路分析：

①分支逻辑的实现：为了实现三分支的 `if` 语句，我们需要使用 `cmp` 语句以及相关的跳转语句，以及设置合适的 `if`、`elif`、`else`、`end` 标号，实现不同代码块的跳转

②循环逻辑的实现，主要使用使用 `loop ... jmp loop` 的结构实现循环，同时要设计好循环跳出的条件，避免死循环的产生。

③函数的实现，为了实现不同函数的调用，我们需要使用到栈来保存与恢复我们的现场，从而实现函数的调用，记得最后使用 `ret` 指令将栈顶的内容弹出栈

- 实验步骤：

### ①分支逻辑的实现：

**Step1: 实现 if 分支**，由于 `cmp` 无法进行立即数和内存地址的比较，所以首先需要将 `a1` 地址中的值取到寄存器中，然后与 `12` 进行比较，如果大于 `12` 那么就要跳转到 `else_if` 段，如果没跳转说明 `a1` 小于 `12`，则执行实现 `if_flag = a1 / 2 + 1`，然后结束分支，跳转到 `end_if`

```
your_if:
    mov eax,[a1]
    cmp eax,12
    jge else_if
    mov edx,0
    mov eax,[a1]
    mov ebx,2
    idiv ebx
    mov dword [if_flag],eax
    add dword [if_flag],1
    jmp end_if
```

**Step2: else\_if 分支的实现**，首先比较 a1 如果大于 24，那么就跳转到最后的 else 分支，如果小于 24，就实现  $if\_flag = (24 - a1) * a1$  计算，然后结束分支，跳转到 end\_if

```
else_if:
    cmp eax,24
    jge else
    mov eax,24
    sub eax,[a1]
    imul eax,[a1]
    mov [if_flag],eax
    jmp end_if
```

**Step3: else 分支的实现**，在 else 分支中无需进行跳转，实现算数左移 4 位的运算操作

```
else:
    mov eax,[a1]
    shl eax,4
    mov [if_flag],eax
```

## ②循环逻辑的实现:

**Step1:** 将地址 a2 中的值取出放入到寄存器 ebx 中方便后续计算，然后设置循环退出的条件，如果 a2 小于 12，就跳转到 end\_while，退出循环

```
your_while:
    mov ecx,[while_flag]
while_loop:
    mov ebx,[a2]
    cmp ebx,12
    jl end_while
```

**Step2:** 调用函数 my\_random，使用栈保存现场后使用 call 调用 call\_random 函数，然后出栈恢复调用函数前的状态

```
push ecx
call my_random
pop ecx
sub ebx,12
```

**Step3:** 最后将结果放到[while\_flag]所指向的内存中，并返回 loop 处

```
mov ecx,[while_flag]
mov byte [ecx+ebx],al
dec dword [a2]
jmp while_loop
```

### ③函数的实现:

**Step1:** 通过基址寻址的方式依次将 `your_string` 内存中存储的字符串的字符取出来放到 `al` 寄存器中, 如果 `al` 中的字符的 ASCII 码为 0 就结束循环

```
your_function:
    xor esi,esi
function_loop:
    mov al,[your_string+esi]
    cmp al,0
    je function_end
```

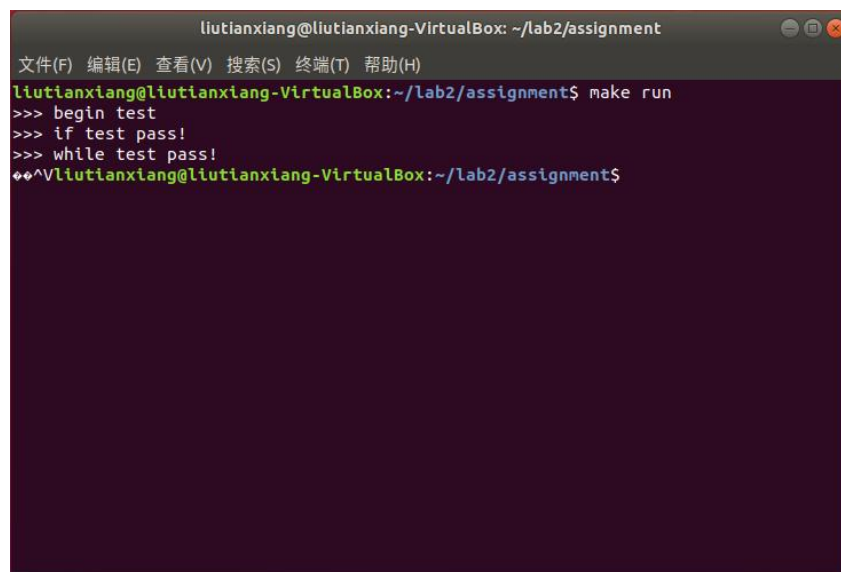
**Step2:** 使用 `pushad` 保存调用函数前的寄存器状态, 使用 `call` 函数调用函数 `print_a_char`, `popad` 恢复寄存器, `esi` 自增, 跳回循环

```
    pushad
    push ax
    call print_a_char
    pop ax
    popad
    inc esi
    jmp function_loop
```

**Step3:** 最后在函数的结尾需要 `ret` 函数返回栈顶数据, 退出函数

```
function_end:
    ret
```

- 实验结果展示: 通过执行前述代码, 可得下图结果,



```
liutianxiang@liutianxiang-VirtualBox: ~/lab2/assignment
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
liutianxiang@liutianxiang-VirtualBox:~/lab2/assignment$ make run
>>> begin test
>>> if test pass!
>>> while test pass!
♦♦^liutianxiang@liutianxiang-VirtualBox:~/lab2/assignment$
```

## Section 5 实验总结与心得体会

这次实验我们从计算机的开机启动过程出发，学习到了计算机启动过程主要包括加电开机、BIOS 启动、加载 MBR、硬盘启动和内核启动 5 个过程，然后我们学习到了如何使用编写 MBR 来启动我们的操作系统，计算机通过检查 MBR 最后两个字节是否为 `0x55, 0xAA` 来判断是否有操作系统。

在编写 MBR 的过程中学习到了 x86 汇编语法知识，比如 x86 汇编所使用的寄存器、运算和逻辑指令、内存的寻址方式，包括间接寻址和直接寻址、控制转移指令、栈以及函数调用的操作等等，同时在实验中通过补全一段汇编代码，加深了汇编中分支、循环、函数的编写以及使用。

同时为了实现计算机的 I/O 操作，还学习到了关于实模式下的中断操作，学习到了 `int 10H` 关于光标以及字符显示的相关操作、`int 16H` 关于键盘输入的相关中断功能，并利用这些操作编写了一些具有动态效果的程序，比如字符弹射程序，未来也可以通过这些中断服务实现操作系统的启动动画。

同时在本次实验进一步学习了 gdb 与 qemu 的一些相关指令，学习到了 gdb 调试指令，配合 qemu 对我们所编写代码进行调试，包括设置断点，打印当前地址、打印当前 PC 以及之后的汇编指令等等。

总之，通过本次的实验提高了我对于计算机启动和操作系统的理解，同时提高了我对于 x86 汇编程序的代码编写能力以及调试能力。

## Section 6 附录：参考资料清单

- [1] [INT10H 中断服务详解-CSDN 博客](#)
- [2] [汇编语言——一些中断的调用 - bling、 - 博客园](#)
- [3] [键盘 I/O 中断调用 \(INT 16H\) 和常见的 int 17H、int 1A H-CSDN 博客](#)
- [4] [GDB 是什么? - C 语言中文网](#)

## Section 7 附录：代码清单

【实验任务 1】完整的 mbr.asm 程序如下：

```
[bits 16]
xor ax, ax ; eax = 0
; 初始化段寄存器，段地址全部设为 0
mov ds, ax
mov ss, ax
mov es, ax
mov fs, ax
mov gs, ax
; 初始化栈指针
mov sp, 0x7c00
mov ax, 0xb800
mov gs, ax
mov ah, 0x03 ; 青色
mov al, 'H'
mov [gs:2 * 0],
mov al, 'e'
mov [gs:2 * 1], ax
mov al, 'l'
mov [gs:2 * 2], ax
mov al, 'l'
mov [gs:2 * 3], ax
mov al, 'o'
mov [gs:2 * 4], ax
mov al, ' '
mov [gs:2 * 5], ax
mov al, 'W'
mov [gs:2 * 6], ax
mov al, 'o'
mov [gs:2 * 7], ax
mov al, 'r'
mov [gs:2 * 8], ax
mov al, 'l'
mov [gs:2 * 9], ax
mov al, 'd'
mov [gs:2 * 10], ax
jmp $ ; 死循环
times 510 - ($ - $$) db 0
db 0x55, 0xaa
```

【实验任务 2】完整的思考题第 14 题 14.asm 程序如下：

```
org 0x7c00
```

```

[bits 16]
xor ax,ax
mov ds,ax
    mov ss,ax
    mov es,ax
    mov fs,ax
    mov gs,ax
    mov sp,0x7c00
    mov ax,0xb800
    mov gs,ax
;功能号 6H 实现清屏操作
    mov ah,0x06
    mov al,0
    mov bh,0x07
    mov ch,0    ;左上角行号
    mov cl,0    ;左上角列号
    mov dh,24   ;右下角行号
    mov dl,79   ;右下角列号
    int 0x10
;初始化方向和显示的数字及其颜色
    dirc_row db 1
    dirc_col db -1
    mov al,'0'
    mov bl,0x00
    mov ah,0x02
    mov bh,0
    mov dh,2
    mov dl,0
    int 0x10
loop:
;将寄存器存入栈中
    pushad
;计算与当前中心对称的位置坐标并输出字符
    sub dl,79
    neg dl      ;dl=79-dl
    mov ah,0x02
    mov bh,0
    int 0x10    ;dh=24-dh
    mov ah,0x09
    mov cx,1
    int 0x10
;出栈恢复之前的寄存器中的值
    popad
;2H 移动光标到新位置,9H 显示字符

```

```

    mov ah,0x02
    mov bh,0
    int 0x10
    mov ah,0x09
    mov cx,1
    int 0x10
;双重循环使 CPU 空转以达到延迟效果
    mov cx, 0x300    ;外层循环次数
out_delay:
    mov si, 0xFFFF   ;内层循环次数
in_delay:
    dec si
    jnz in_delay
    dec cx
    jnz out_delay
;调整方向以及颜色
adjust_char:
    cmp al,'9'
    je reset_char
adjust_color:
    cmp bl,0xFF
    je reset_color
adjust_row:
    cmp dh,24
    je bounce_row
    cmp dh,0
    je bounce_row
adjust_col:
    cmp dl,79
    je bounce_col
    cmp dl,0
    je bounce_col
    jmp continue
reset_char:
    mov al,'0'
    jmp adjust_color
reset_color:
    mov bl,0x00
    jmp adjust_row

bounce_row:
    neg byte [dirc_row]
    jmp adjust_col
bounce_col:

```



```

        neg byte [dirc_col]
        jmp continue
;更新新的位置和颜色
continue:
        add dh,[dirc_row]
        add dl,[dirc_col]
        add bl,00000001b
        jmp loop
jmp $ ; 死循环
times 510 - ($ - $$) db 0
db 0x55, 0xaa

```

【实验任务 3】完整的思考题第 16 题 16\_1.asm 程序如下：

```

org 0x7c00
[bits 16]
mov ah,0x02
mov bh,0
mov dh,8
mov dl,8
int 0x10
mov ah,0x03
mov bh,0
int 0x10
mov al,dh
add al,'0'
mov ah,0x09
mov bl,10001111b
mov cx,1
int 0x10
mov ah,0x02
mov bh,0
mov dh,8
mov dl,9
int 0x10
mov al,dl
add al,'0'
mov ah,0x09
mov bl,10001111b
mov cx,1
int 0x10
jmp $ ; 死循环
times 510 - ($ - $$) db 0
db 0x55, 0xaa

```

【实验任务 3】完整的思考题第 16 题 16\_2.asm 程序如下：

```
org 0x7c00
[bits 16]
mov ah,0x02
mov bh,0
mov dh,0
mov dl,0
int 0x10
mov al,'2'
mov ah,0x09
mov bl,10001111b
mov cx,1
int 0x10
mov ah,0x02
mov bh,0
mov dh,0
mov dl,1
int 0x10
mov al,'3'
mov ah,0x09
mov bl,10001111b
mov cx,1
int 0x10
mov ah,0x02
mov bh,0
mov dh,0
mov dl,2
int 0x10
mov al,'3'
mov ah,0x09
mov bl,10001111b
mov cx,1
int 0x10
mov ah,0x02
mov bh,0
mov dh,0
mov dl,3
int 0x10
mov al,'2'
mov ah,0x09
mov bl,10001111b
mov cx,1
int 0x10
mov ah,0x02
mov bh,0
```

```

mov dh,0
mov dl,4
int 0x10
mov al,'0'
mov ah,0x09
mov bl,10001111b
mov cx,1
int 0x10
mov ah,0x02
mov bh,0
mov dh,0
mov dl,5
int 0x10
mov al,'1'
mov ah,0x09
mov bl,10001111b
mov cx,1
int 0x10
mov ah,0x02
mov bh,0
mov dh,0
mov dl,6
int 0x10
mov al,'0'
mov ah,0x09
mov bl,10001111b
mov cx,1
int 0x10
mov ah,0x02
mov bh,0
mov dh,0
mov dl,7
int 0x10
mov al,'4'
mov ah,0x09
mov bl,10001111b
mov cx,1
int 0x10
jmp $ ; 死循环
times 510 - ($ - $$) db 0
db 0x55, 0xaa

```

【实验任务 3】完整的思考题第 16 题 16\_3.asm 程序如下：

```
org 0x7c00
```

```

[bits 16]
    mov ah, 0x06
    mov al, 0
    mov bh, 0x07
    mov ch, 0    ;左上角行号
    mov cl, 0    ;左上角列号
    mov dh, 24   ;右下角行号
    mov dl, 79   ;右下角列号
    int 0x10
    mov ah,0x02
    mov bh,0
    mov dh,0
    mov dl,0
    int 0x10
loop:
    mov ah,0x0
    int 16H
    cmp al,0x0D
    je end
    mov ah,0x0e
    mov bl,10001111b
    mov cx,1
    int 0x10
    jmp loop
end:
    jmp $ ; 死循环
    times 510 - ($ - $$) db 0
    db 0x55, 0xaa

```

【实验任务 4】完整的思考题第 17 题 17.asm 程序如下：

```

#include "head.include"
your_if:
    mov eax,[a1]
    cmp eax,12
    jge else_if
    mov edx,0
    mov eax,[a1]
    mov ebx,2
    idiv ebx
    mov dword [if_flag],eax
    add dword [if_flag],1
    jmp end_if
else_if:
    cmp eax,24

```

```

    jge else
    mov eax,24
    sub eax,[a1]
    imul eax,[a1]
    mov [if_flag],eax
    jmp end_if
else:
    mov eax,[a1]
    shl eax,4
    mov [if_flag],eax
end_if:
your_while:
    mov ecx,[while_flag]
while_loop:
    mov eax,[a2]
    cmp eax,12
    jl end_while
    push ecx
    call my_random
    pop ecx
    mov ebx,[a2]
    sub ebx,12
    mov byte [ecx+ebx],al
    dec dword [a2]
    jmp while_loop
end_while:
#include "end.include"
your_function:
    xor esi,esi
function_loop:
    mov al,[your_string+esi]
    cmp al,0
    je function_end

    pushad
    push ax
    call print_a_char
    pop ax
    popad

    inc esi
    jmp function_loop
function_end:
    ret

```