操作系统实验报告

221900180 田永铭

2024 年 3 月 28 日

一、实验要求

使用 nasm 汇编编程实现一个 MBR 程序，每隔 1 秒钟左右输出一个 (组）字符（自定），具体内容包括：
1）实现一个时钟中断处理例程，按规定时间间隔输出字符（可使用 BIOS 提供的 int 10h 实现字符输出）；
2）修改中断向量表，将时钟中断处理例程起始地址填入中断向量表对应表项；
3）设置 8253/4 定时器芯片，每隔 20ms 左右（自定 $<$ 1 秒）产生一次时钟中断

二、实验环境

- 操作系统：Windows 11
- 编程语言：汇编
- 使用工具：qemu;vscode Hex Editor
- 虚拟系统版本:windows-7;linux-20-desktop

三、实验原理

时钟中断: 时钟中断是系统本身提供的重要中断服，8086 系统还提供了可以给用户自定义修改的时钟中断。

修改中断向量表: 中断向量表是一个存储中断处理例程起始地址的表格。为了让系统在时钟中断发生时执行我们编写的中断处理例程，需要修改中断向量表，将时钟中断处理例程的起始地址填入对应的表项。

8253/8254 定时器芯片: 这是一个可编程定时器芯片，通常用于在计算机中生成定时中断。在本实验中，可以设置 8253/8254 芯片，使其每隔约 20 毫秒产生一次时钟中断，以满足输出字符的要求。

四、实验过程

作业视频演示链接如下 (上传至南大云盘，如果查看不了则需要下载查看)(作业附件中也有演示视频):
221900180_ 田永铭 _ 实验 3 视频演示

实验主要分为以下几个步骤：

1. 修改中断向量表
2. 设置 8253/4 定时器芯片
3. 按规定时间间隔输出字符
4. 整合代码
5. 写入磁盘，查看运行效果
6. 其他各种尝试和探索过程

4.1 修改中断向量表

参考8086 中断向量表，我们可以得到关键信息: 定时器 (IRQ0) 的 I/O 地址为 20h-23h, 中断号为 8h；同时，提供给用户的定时器控制的软中断，地址为 70h-73h, 中断号为 1C。我们需要用到的是 4*1C 和 4*1C+2 的数字，即 70h 和 72h, 我们需要把中断服务程序入口地址偏移量存放到前者中，把中断服务程序入口地址的段基址放到后者中即可。

```
mov word [0x70], clock_handler ;修改中断，跳转clock_handler处理中断
mov word [0x72], 0

clock_handler:
;to do
```

4.2 设置 8253/4 定时器芯片

为了让定时器每 20ms 就产生一个中断，我们需要修改定时器的频率，并且将计数值发送到计数器的端口。实现代码如下

```
; 设置定时器0（8253/8254）以产生时钟中断

; 发送命令字节到控制寄存器端口0x43
mov al, 00110110b ; 方式3，用于定时产生中断
```

```
out 0x43, al

; 计算计数值，产生20毫秒的时钟中断，时钟频率为1193180赫兹
; 计数值 = (时钟频率 / 每秒中断次数) - 1
; = (1193180 / (1 / 0.02)) - 1= 23863
mov ax, 23863

; 将计数值分为低字节和高字节，发送到计数器0的数据端口（端口0x40）
out 0x40, al          ; 低字节
mov al, ah
out 0x40, al          ; 高字节
```

4.3 按规定时间间隔输出字符

定时器每 20ms 就产生一个中断，要想每 1s 输出一个字符，我们需要进行计数，每中断 50 次才打印。这里采用的是一个数据段，每次用 cx 从数据段中读计数，自增后写回，如果到了 50，则打印并且重新计数，否则直接返回时钟中断。实现代码如下：

```
lock_handler:
        ;每次中断将数据段中值加1，加到50后才打印并且清零
        ;实现1秒打印一次
        inc word [counter]
        mov cx, [counter]
        cmp cx, 50
        jne r
        mov si,msg
        call print
        mov word [counter], 0
        r:iret

exit:
        ret

print:
        ; 打印字符
        lodsb
```

```
        or al, al
        jz exit
        mov ah, 0x0E
        int 0x10            ; 调用INT 10h中断
        jmp print

msg db "Yong-Ming Tian ", 0
```

4.4 整合代码

为了使得程序运行符合预期，我们还需要：

1. 整合代码，在代码开头设置起点，初始化
2. 注意开始程序时用 cli 关中断，完成时钟设置后用 sti 开中断
3. 在程序的最后添加 jmp $，使得程序不断运行
4. 在代码最后填充剩余的 512 字节使得代码符合 MBR 要求

完整代码如下:

```
BITS 16
SECTION MBR vstart=0x7c00

;稳妥使用栈
xor ax, ax
mov ds, ax
mov es, ax
mov ss, ax
mov cx, ax
mov sp, 0x7c00

counter dw 0x00;定义数据段

start:
        cli ;关中断

        ;定时器提供的软中断地址是70-73，定时器地址是20-23
        mov word [0x70], clock_handler ;修改中断;
```

```
                ;跳转clock_handler处理中断
        mov word [0x72], 0

        ; 设置定时器0（8253/8254）以产生时钟中断

        ; 发送命令字节到控制寄存器端口0x43
        mov al, 00110110b ; 方式3，用于定时产生中断
        out 0x43, al

        ; 计算计数值，产生20毫秒的时钟中断，时钟频率为1193180赫兹
        ; 计数值 =（时钟频率 / 每秒中断次数）- 1
        ;= (1193180 / (1 / 0.02)) - 1= 23863
        mov ax, 23863

        ; 将计数值分为低字节和高字节，发送到计数器0的数据端口
        out 0x40, al          ; 低字节
        mov al, ah
        out 0x40, al          ; 高字节

        sti ;开中断
        jmp $    ;使程序不断停在这里执行等待时钟中断

; 时钟中断处理例程
clock_handler:
        ;每次中断将数据段中值加1，加到50后才打印并且清零
        ;实现1秒打印一次
        inc word [counter]
        mov cx, [counter]
        cmp cx, 50
        jne r
        mov si,msg
        call print
        mov word [counter], 0
        r:iret

exit:
```

```
        ret

print:
; 打印字符
        lodsb
        or al, al
        jz exit
        mov ah, 0x0E
        int 0x10            ; 调用INT 10h中断
        jmp print

msg db "Yong-Ming␣Tian␣", 0

times 510-($-$$) db 0   ; 填充剩余的512字节以使文件大小为512字节
dw 0xAA55               ; MBR结束标志
```

4.5 写入磁盘，查看运行效果

将 asm 文件编译并且写入磁盘，启动后查看效果，指令如下:

编译 (我的 asm 叫 Timer.asm, 编译生成的 bin 叫 Timer.bin)

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

写入虚拟磁盘

```
dd if=Timer.bin of=os.img bs=512 count=1
```

运行

```
qemu-system-x86_64 -bios D:\MyOwnFiles\qemu\share\bios.bin -drive
file=os.img,format=raw -m 2G -smp 2
```

查看运行效果: 在这一部分，我将通过视频详细展示我的代码运行结果，一共分为四部分:

- 设置时钟周期，使得 20ms 打印一次 (速度中等)
- 设置时钟周期，使得 45ms 打印一次 (速度慢)
- 设置时钟周期，使得 2ms 打印一次 (速度快)
- 保持时钟周期 20ms 一次，实现 50 次中断打印一次 (1s 一次)

视频演示链接如下 (上传至南大云盘，如果查看不了则需要下载查看)(作业附件中也有演示视频):
221900180_ 田永铭 _ 实验 3 视频演示
同时，也展示一张图片简要展示结果：

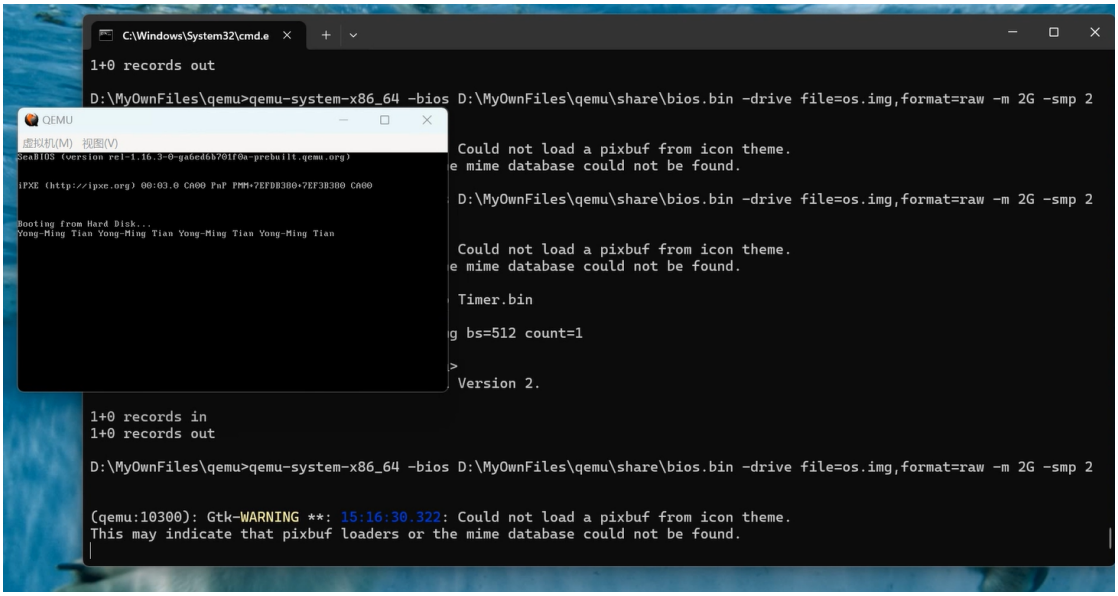

图 1: 打印 Yong-Ming Tian

4.6 其他各种尝试和探索过程

在一开始实现计数 50 次中断打印一次字符的时候，我采用的时寄存器方法，但是失败了。那么利用寄存器能不能实现全局计数呢？

```
xor cx,cx
sti
jmp $

; 时钟中断处理例程
clock_handler:
        inc cx
        cmp cx, 50
        jne no_print
        mov si,msg
```

```
        call print
        iret
```

以上代码显然是不合格的，在老师的帮助下，我进行了以下的思考：

4.6.1 防止 print 调用的 int 10h 修改 cx 的数值

在 call print 前后分别加上 push cx，pop cx 即可解决。但问题不在这。

4.6.2 考虑到中断也可能改变各种数值，需要更好的保存 cx 数值

```
xor cx,cx
push 0

sti ;开中断
jmp $   ;使程序不断停在这里执行等待时钟中断

; 时钟中断处理例程
clock_handler:
pop cx
inc cx
push cx
cmp cx, 50
jne no_print
mov si,msg
call print
mov word [counter], 0
xor cx, cx
iret

no_print:
iret
```

4.6.3 pop 出的并非 cx!

以上代码 pop 出来的不是 cx，而是中断发生后，硬件压进去的最后一个寄存器值，比如 flags，所以大部分情况是 0。

再次尝试在中断向量表的代码最后 push cx，同时保存 sp 的值到 bp，在中断处理程序中，直接用 bp 指向的内存位置读出值到 cx，代码如下:

```
push cx
mov bp, sp
je print

sti ;开中断
jmp $    ;使程序不断停在这里执行等待时钟中断

; 时钟中断处理例程
clock_handler:
inc word [bp]
mov cx, [bp]
cmp cx, 50
jne no_print
mov si,msg
call print
mov word [counter], 0
xor cx, cx
mov word [bp], 0
iret
```

仍然不能实现实验要求。

4.6.4 原因分析

实际情况不会用通用寄存器去存放一个全局变量，存在一个内存单元中是常用的做法。这种非常规的做法可能会带来一系列问题：比如不能正确获取寄存器应该有的值，在虚拟机环境下中断发生后可能会修改一些重要的数值，这些对硬件知识的更多了解使得这种方法并不太可行。

4.6.5 其他拓展

除了时钟中断，我们还可以玩一玩别的中断，比如除 0 就会引发 0 号中断处理，我们采用相同的办法重写该中断程序，也能实现在其上打印字符。

五、实验结果

本人完全完成了本次实验的要求，并且在此基础上思考了其他实现方法的可能性，又对其他中断进行了新的把玩，收获颇丰。

六、总结与思考

- 理论好理解不代表实验好做，实验需要非常多的细节，比如如何计数？利用寄存器来计数全局变量，显然不是一个好的选择
- STFW,RTFM 是计算机大类学生必备的素养，知识需要主动检索和学习
- 通过本次实验，我对中断向量表的理解更深了一步。在手写 MBR 的过程中，对时钟中断有了更直观的体验。尽管有些问题没有完全解决，但在解决问题过程中，我学到了更多的知识。

七、其它参考文献

除了正文给出的参考文献，我参考的文献还有:

修改中断向量表
中断向量表 0 号中断
微机原理——定时器 8253(8254) 学习 2 应用与设计
用汇编实现 8253 定时计数器应用实验
Microprocessor - 8086 Interrupts