操作系统原理实验报告

- **实验名称**：实模式和保护模式下的OS启动
- **授课教师**：张青
- **学生姓名**：林隽哲
- **学生学号**：21312450

实验要求

- 熟悉nasm汇编语言，并能够编写简单的nasm汇编程序。简单的编写mbr引导程序，并启动测试。

实验过程

Assignment 1 MBR

内容

- 1.1 复现Example 1。说说你是怎么做的，并将结果截图。
- 1.2 修改Example 1代码，使得MBR被加载到 `0x7c00` 后在 `(12, 12)` 处开始输出你的学号。注意，你的学号显示的前景色和背景色必须和教程中的不同。说说你是怎么做的，并将结果截图。

1.1 实验步骤

- 编写MBR如下：

```
; [filename] mbr.asm
org 0x7c00
[bits 16]
xor ax, ax ; clear ax
; initialize segments
mov ds, ax
mov ss, ax
mov es, ax
mov fs, ax
mov gs, ax

; set stack pointer
mov sp, 0x7c00
mov ax, 0xb800
mov gs, ax

mov ah, 0x01 ; blue
mov al, 'H'
mov [gs:2*0], ax

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 $ ; jump to current address (infinite loop)

; times, an assembly pseudo-instruction, used to repeat the specified number of operations
; $ is the current address, $$ is the start of the current section
; fill the rest of the sector with 0s
times 510-($-$$) db 0
db 0x55, 0xaa ; boot signature, meaning this is a bootable mbr
```

- 使用nasm汇编器来将代码编译成二进制文件：

```
nasm -f bin mbr.asm -o mbr.bin
# -f 指定输出的文件格式
# -o 指定输出的文件名
# mbr.bin 中保存的是机器可以识别的机器指令，可以使用命令xdd查看其中的内容
```

- 生成了mbr.bin文件后，我们将其写入到硬盘的首扇区。我们需要先创建一个虚拟磁盘：

```
# qemu-img create filename [size]
qemu-img create hd.img 10M
```

- 然后将mbr.bin写入到hd.img的首扇区：

```
dd if=mbr.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
# if 表示输入文件
# of 表示输出文件
# bs 表示块大小，以字节表示
# count 表示写入的块数
# seek 表示越过输出文件中多少块之后再写入
# conv=notrunc 表示不截断输出文件，如果不加上这个参数，那么硬盘在写入后多余的部分会被截断
```

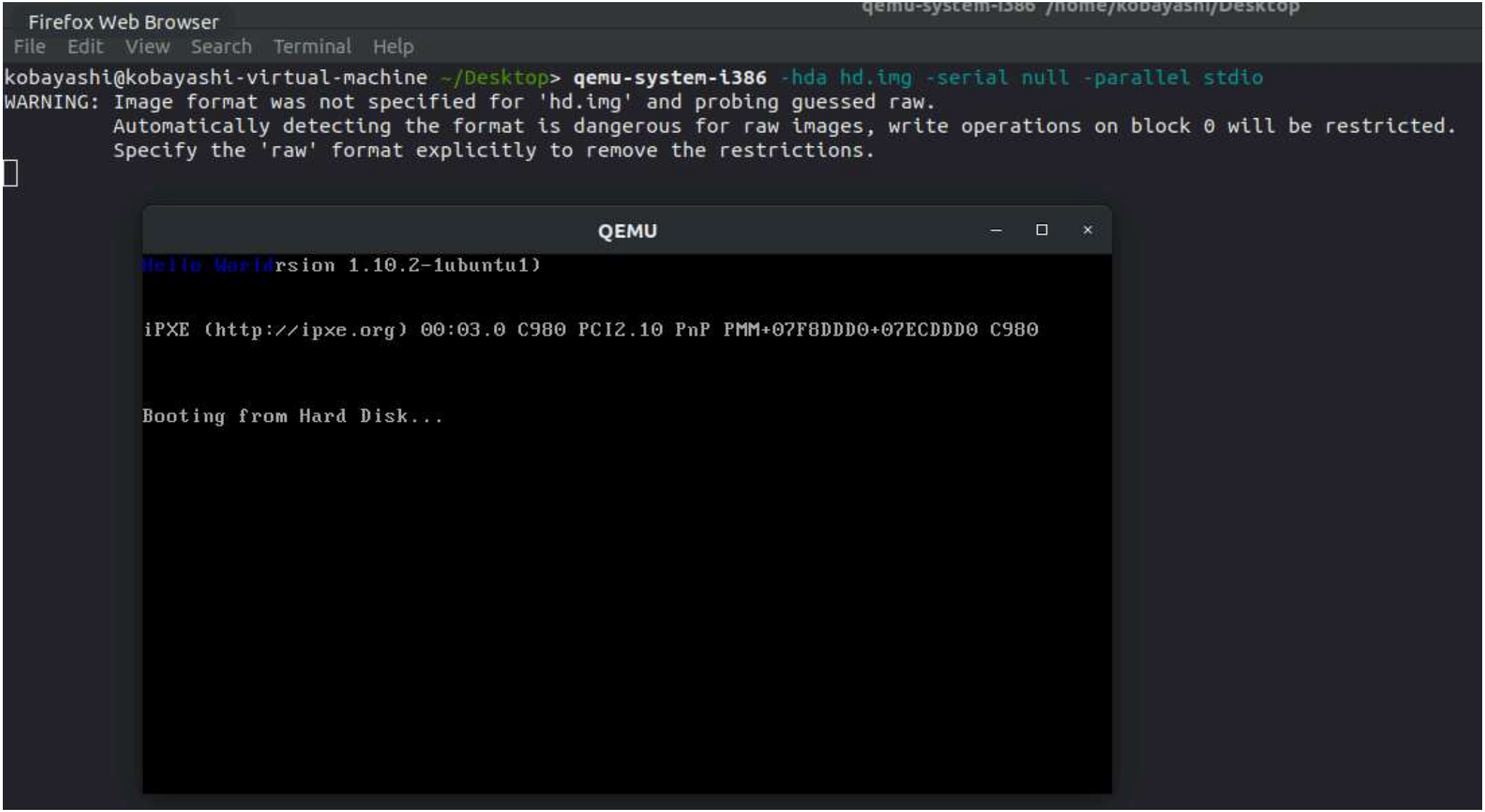
```
kobayashi@kobayashi-virtual-machine ~/Desktop> nasm -f bin mbr.asm -o mbr.bin
kobayashi@kobayashi-virtual-machine ~/Desktop> qemu-img create hd.img 10m
Formatting 'hd.img', fmt=raw size=10485760
kobayashi@kobayashi-virtual-machine ~/Desktop> dd if=mbr.bin of=hd.img bs=512 count=1 seek=0 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000198291 s, 2.6 MB/s
kobayashi@kobayashi-virtual-machine ~/Desktop> xxd mbr.bin
00000000: 31c0 8ed8 8ed0 8ec0 8ee0 8ee8 bc00 7cb8  1.............|.
00000010: 00b8 8ee8 b401 b048 65a3 0000 b065 65a3  .......He....ee.
00000020: 0200 b06c 65a3 0400 b06c 65a3 0600 b06f  ...le....le....o
00000030: 65a3 0800 b020 65a3 0a00 b057 65a3 0c00  e.... e....We...
00000040: b06f 65a3 0e00 b072 65a3 1000 b06c 65a3  .oe....re....le.
00000050: 1200 b064 65a3 1400 ebfe 0000 0000 0000  ...de...........
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000100: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000120: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000130: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000150: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000160: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000170: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000180: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000190: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000001f0: 0000 0000 0000 0000 0000 0000 0000 55aa  ..............U.
```

```
kobayashi@kobayashi-virtual-machine ~/Desktop>
```

- 最后，我们使用qemu来模拟计算机启动：

```
qemu-system-i386 -hda hd.img -serial null -parallel stdio
# -hda hd.img 表示将文件hd.img作为0号磁盘映像。
# -serial dev 表示重定向虚拟串口到指定设备，null表示重定向到空设备。
# -parallel stdio 表示重定向虚拟并口到主机标准输入输出设备中。
```

- 启动后的效果如下。可以看到屏幕的第一行已经输出了 `Hello World` 。



1.2 实验步骤

- 修改mbr.asm如下：

```
; mbr_id.asm
[bits 16]
xor ax, ax ; clear ax
; initialize segments
mov ds, ax
mov ss, ax
mov es, ax
mov fs, ax
mov gs, ax

; set stack pointer
mov sp, 0x7c00
mov ax, 0xb800
mov gs, ax

mov ah, 0x14 ; red on blue

mov al, '2'
mov [gs:0x0a*160+0x0c*2], ax
mov al, '1'
mov [gs:0x0a*160+0x0d*2], ax
mov al, '3'
mov [gs:0x0a*160+0x0e*2], ax
mov al, '1'
mov [gs:0x0a*160+0x0f*2], ax
mov al, '2'
mov [gs:0x0a*160+0x10*2], ax
mov al, '4'
mov [gs:0x0a*160+0x11*2], ax
mov al, '5'
mov [gs:0x0a*160+0x12*2], ax
```

```
mov al, '0'
mov [gs:0x0a*160+0x13*2], ax

jmp $ ; jump to current address (infinite loop)

; times, an assembly pseudo-instruction, used to repeat the specified number of operations
; $ is the current address, $$ is the start of the current section
; fill the rest of the sector with 0s
times 510-($-$$) db 0
db 0x55, 0xaa ; boot signature, meaning this is a bootable mbr
```

- 重复1.1中的步骤，最终启动后的效果如下：

```
File  Edit  View  Search  Terminal  Help
kobayashi@kobayashi-virtual-machine ~/Desktop> nasm -f bin mbr_id.asm -o mbr_id.bin
kobayashi@kobayashi-virtual-machine ~/Desktop> dd if=mbr_id.bin of=mbr_id.img bs=512 count=1 seek=0 conv=notrunc
1+0 records in
1+0 records out
512 bytes copied, 0.000112719 s, 4.5 MB/s
kobayashi@kobayashi-virtual-machine ~/Desktop> qemu-system-i386 -hda mbr_id.img -serial null -parallel stdio
WARNING: Image format was not specified for 'mbr_id.img' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.
         Specify the 'raw' format explicitly to remove the restrictions.
```

Terminal

QEMU

```
SeaBIOS (version 1.10.2-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDD0+07ECDDD0 C980

Booting from Hard Disk...

            21312450
```

Assignment 2 实模式中断

内容

- 2.1 请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。说说你是怎么做的，并将结果截图。
- 2.2 请修改1.2的代码，使用实模式下的中断来输出你的学号。说说你是怎么做的，并将结果截图。
- 2.3 请在2.1和2.2的知识的基础上，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。

2.1 实验步骤

- 编写cursor.asm如下：

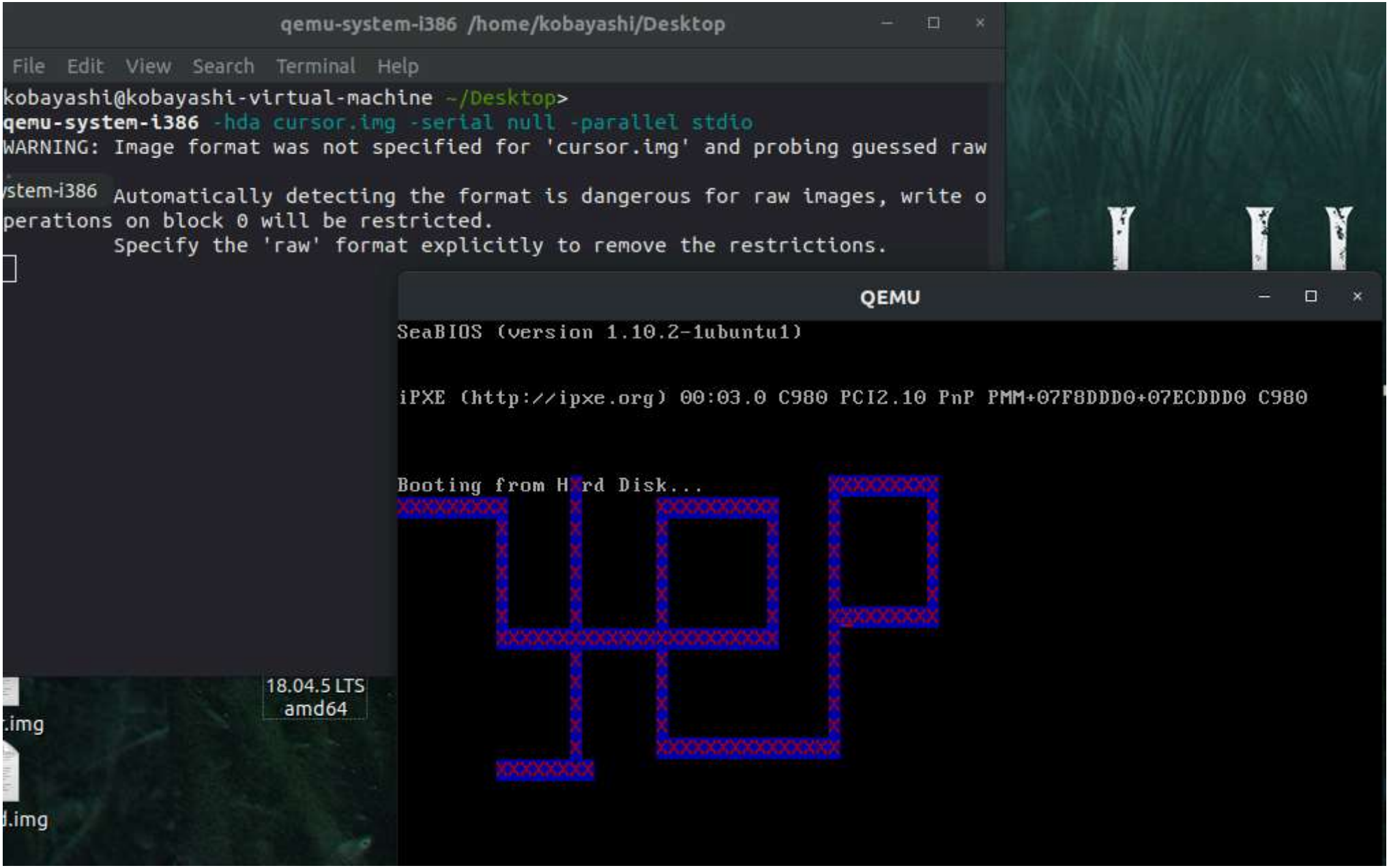
```
; cursor.asm
[bits 16]
xor ax, ax ; clear ax
; initialize segments
mov ds, ax
mov ss, ax
mov es, ax
mov fs, ax
mov gs, ax

; set stack pointer
mov sp, 0x7c00
mov ax, 0xb800
mov gs, ax

; get the current cursor position
mov ah, 0x03
int 0x10

; press h, j, k, l to move the cursor
; press q to quit
read_key:

; write down a 'X'
mov ah, 0x09
mov al, 'X'
mov bh, 0
mov bl, 0x14
mov cx, 1
int 0x10
```

```
mov ah, 0
int 0x16
cmp al, 'h'
je move_left
cmp al, 'j'
je move_down
cmp al, 'k'
je move_up
cmp al, 'l'
je move_right
cmp al, 'q'
je quit
jmp read_key

move_left:
mov ah, 0x02
int 0x10
sub dl, 1
mov ah, 0x0c
int 0x10
jmp read_key

move_down:
mov ah, 0x02
int 0x10
add dh, 1
mov ah, 0x0c
int 0x10
jmp read_key

move_up:
mov ah, 0x02
```

```
int 0x10
sub dh, 1
mov ah, 0x0c
int 0x10
jmp read_key

move_right:
mov ah, 0x02
int 0x10
add dl, 1
mov ah, 0x0c
int 0x10
jmp read_key

quit:

jmp $ ; jump to current address (infinite loop)

; times, an assembly pseudo-instruction, used to repeat the specified number of operations
; $ is the current address, $$ is the start of the current section
; fill the rest of the sector with 0s
times 510-($-$$) db 0
db 0x55, 0xaa ; boot signature, meaning this is a bootable mbr
```

该实现中，我通过中段 `0x10` 的功能获取光变的位置与移动光标，通过中断 `0x16` 来获取键盘输入。在键盘输入为 `h` 、 `j` 、 `k` 、 `l` 时，分别移动光标的位置。在键盘输入为 `q` 时，退出程序。

- 最终启动后的效果如下：



2.2 实验步骤

- 编写output.asm如下所示：

```
; output.asm
org 0x7c00 ; origin, the start of the boot sector
[bits 16]

msg db '21312450', 0 ; the string to be output

xor ax, ax ; clear ax
; initialize segments
mov ds, ax
mov ss, ax
mov es, ax
mov fs, ax
mov gs, ax

; set stack pointer
mov sp, 0x7c00
mov ax, 0xb800
mov gs, ax

; init cursor position
mov bh, 0x00 ; page number
mov dh, 0x0c ; row
mov dl, 0x0c ; column

; set cursor position
mov ah, 0x02
int 0x10

; set output attributes
mov bl, 0x14 ; blue color
mov cx, 0x01 ; print one character at a time
```

```
mov si, msg ; load the address of the string into si

print_string:
    mov al, [si] ; load the character into al
    or al, al ; check if al is 0
    jz end_print_string ; if al is 0, jump to halt

    mov ah, 0x09 ; print character
    int 0x10 ; call video interrupt

    mov ah, 0x02 ; move cursor to next position
    inc dl ; increment column
    int 0x10 ; call video interrupt

    inc si ; move to next character

    jmp print_string ; repeat the process
end_print_string:

jmp $ ; jump to current address (infinite loop)

; times, an assembly pseudo-instruction, used to repeat the specified number of operations
; $ is the current address, $$ is the start of the current section
; fill the rest of the sector with 0s
times 510-($-$$) db 0
db 0x55, 0xaa ; boot signature, meaning this is a bootable mbr
```

上示代码中，主要用到了中断 `0x10` 来设置光标的位置和输出字符。在输出字符时，通过循环来输出字符串中的每一个字符。

- 最终启动后的效果如下：

fish /home/kobayashi/Desktop

File Edit View Search Terminal Help

```
kobayashi@kobayashi-virtual-machine ~/Desktop>
nasm -f bin output.asm -o output.bin
kobayashi@kobayashi-virtual-machine ~/Desktop>
dd if=output.bin of=output.img bs=512 count=1 seek=0 conv=notrunc
0+0 records in
1+0 records out
512 bytes copied, 0.000136553 s, 3.7 MB/s
kobayashi@kobayashi-virtual-machine ~/Desktop> xxd output.bin
00000000: 3231 3331 3234 3530 0031 c08e d88e d08e  21312450.1......
00000010: c08e e08e e8bc 007c b800 b88e e8b7 00b6  .......|........
00000020: 0cb2 0cb4 02cd 10b3 14b9 0100 be00 7c8a  ..............|.
00000030: 0408 c074 0db4 09cd 10b4 02fe c2cd 1046  ...t...........F
00000040: ebed ebfe 0000 0000 0000 0000 0000 0000  ................
00000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
000000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

eOffice Writer

qemu-system-i386 /home/kobayashi/Desktop

File Edit View Search Terminal Help

```
kobayashi@kobayashi-virtual-machine ~/Desktop>
qemu-system-i386 -hda output.img -serial null -parallel stdio -s -S
WARNING: Image format was not specified for 'output.img' and probing guessed raw
.
        Autom
perations on b
        Speci
```

QEMU [Stopped]

```
SeaBIOS (version 1.10.2-1ubuntu1)

iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DDD0+07ECDDD0 C

Booting from Hard Disk...

21312450_
```

gdb /home/kobayashi

File Edit View Search Terminal Help

```
Remote debugging using :1234
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0x0000fff0 in ?? ()
(gdb) b *0x7c00
Breakpoint 1 at 0x7c00
(gdb) c
Continuing.

Breakpoint 1, 0x00007c00 in ?? ()
(gdb) x/30i $pc
=> 0x7c00:      xor    (%ecx),%dh
   0x7c02:      xor    (%ecx),%esi
   0x7c04:      xor    -0x3fceffd0(,%esi,1),%dh
   0x7c0b:      mov    %eax,%ds
   0x7c0d:      mov    %eax,%ss
   0x7c0f:      mov    %eax,%es
   0x7c11:      mov    %eax,%fs
   0x7c13:      mov    %eax,%gs
   0x7c15:      mov    $0xb87c00,%esp
   0x7c1a:      mov    $0xb7e88e,%eax
   0x7c1f:      mov    $0xc,%dh
   0x7c21:      mov    $0xc,%dl
   0x7c23:      mov    $0x2,%ah
   0x7c25:      int    $0x10
   0x7c27:      mov    $0x14,%bl
   0x7c29:      mov    $0xbe0001,%ecx
   0x7c2e:      jl     0x7bba
   0x7c30:      add    $0x8,%al
   0x7c32:      shlb   $0x9,-0x4c(%ebp,%ecx,1)
   0x7c37:      int    $0x10
   0x7c39:      mov    $0x2,%ah
   0x7c3b:      inc    %dl
   0x7c3d:      int    $0x10
   0x7c3f:      inc    %esi
   0x7c40:      jmp    0x7c2f
   0x7c42:      jmp    0x7c42
   0x7c44:      add    %al,(%eax)
   0x7c46:      add    %al,(%eax)
   0x7c48:      add    %al,(%eax)
   0x7c4a:      add    %al,(%eax)
(gdb) b *0x7c42
Breakpoint 2 at 0x7c42
(gdb) c
Continuing.

Breakpoint 2, 0x00007c42 in ?? ()
(gdb) s
```

2.3 实验步骤

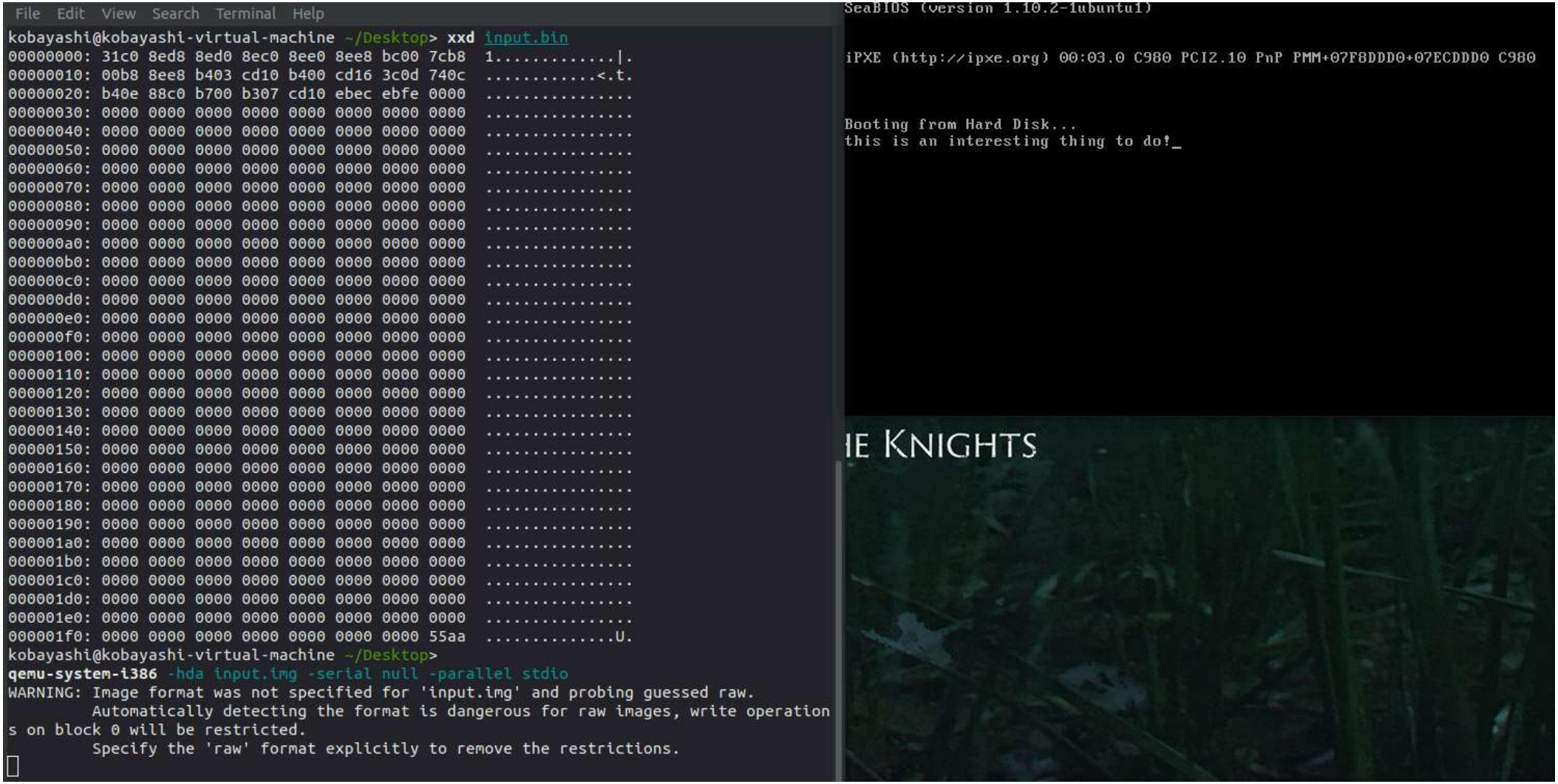
- 编写input.asm如下：

```
; input.asm
[bits 16]
xor ax, ax ; clear ax
; initialize segments
mov ds, ax
mov ss, ax
mov es, ax
mov fs, ax
mov gs, ax

; set stack pointer
mov sp, 0x7c00
mov ax, 0xb800
mov gs, ax

; get the current cursor position
mov ah, 0x03
int 0x10

; read the keyboard, and output the character to the screen
read_key:
    ; read the key from the keyboard, and store it in al
    mov ah, 0
    int 0x16

    cmp al, 0x0d ; check if the character is enter
    jz end_read_key

    ; show the character in al on the screen
    mov ah, 0x0e
    mov al, al
```

```
    mov bh, 0
    mov bl, 0x07
    int 0x10
    jmp read_key
end_read_key:

jmp $ ; jump to current address (infinite loop)

times 510-($-$$) db 0
db 0x55, 0xaa ; boot signature, meaning this is a bootable mbr
```

上述代码主要通过中断先实现键盘的按键读取，然后实现按键的回显。

- 最终启动后的运行效果如下：



Assignment 3 汇编

内容

- 3.1 **分支逻辑的实现**。请将下列伪代码转换为汇编代码，并放置在标号 `your_if` 之后。

```
if a1 < 12 then
        if_flag = a1 / 2 + 1
else if a1 < 24 then
        if_flag = (24 - a1) * a1
else
        if_flag = a1 << 4
end
```

- 3.2 **循环逻辑的实现**。请将下列伪代码转换为汇编代码，并放置在标号 `your_while` 之后。

```
while a2 >= 12 then
        call my_random      // my_random将产生一个随机数放到eax中
        while_flag[a2 - 12] = eax
        --a2
end
```

- 3.3 **函数的实现**。请编写函数 `your_function` 并调用之，函数的内容是遍历字符数组 `string` 。

```
your_function:
        for i = 0; string[i] != '\0'; ++i then
                pushad
                push string[i] to stack
                call print_a_char
                pop stack
                popad
        end
        return
end
```

3.1 实验步骤

- 分支逻辑的实现，编写if.asm如下：

```
; if.asm
; load the value of a1 into eax
mov eax, [a1]

; if a1 < 12, then goto if1
mov ebx, 12
cmp eax, ebx
jl if1

; else if a1 < 24, then goto if2
mov ebx, 24
cmp eax, ebx
jl if2

; else, if_flag = a1 << 4
shl eax, 4
mov [if_flag], eax
jmp endif

; if1: if_flag = (a1 / 2) + 1
if1:
shr eax, 1
inc eax
mov [if_flag], eax
jmp endif

; if2: if_flag = (24 - a1) * a1
if2:
mov ecx, 24
sub ecx, eax
imul ecx, eax
```

```
mov [if_flag], ecx
jmp endif

endif:
```

3.2 实验步骤

- 循环逻辑的实现，编写while.asm如下：

```
; while.asm
; load the value of a2 into ebx
mov ebx, [a2]

while_loop:
    ; if a2 < 12, end the loop
    cmp ebx, 12
    jl end_while_loop

    ; push and save ebx
    push ebx
    ; generate a random character and store it in al
    call my_random
    ; pop and restore ebx
    pop ebx

    ; read the value of char array pointer while_flag
    mov edx, [while_flag]
    ; calculate the address of the current character,
    ; and store the random character in it
    mov [edx + ebx - 12], al

    dec ebx
    mov [a2], ebx

    jmp while_loop
end_while_loop:
```

3.3 实验步骤

- 函数的实现，编写function.asm如下：

```
; function.asm
mov eax, 0 ; i = 0
mov ebx, [your_string]

print_loop:
    movzx ecx, byte [ebx + eax]
    test ecx, ecx
    jz end_print_loop

    push eax
    push ebx
    push ecx ; the argument to print_a_char
    call print_a_char
    pop ecx
    pop ebx
    pop eax

    inc eax
    jmp print_loop
end_print_loop:
```

最终测试效果

fish /home/kobayashi/Desktop/assignment

File Edit View Search Terminal Help

```
kobayashi@kobayashi-virtual-machine ~/D/assignment> ls
assignment-4-example.mp4*  head.include*  mbr.bin*      student.o  test.cpp*
end.include*               makefile*      student.asm*  test*      test.o
kobayashi@kobayashi-virtual-machine ~/D/assignment> make run
>>> begin test
>>> LibreOffice Writer
>>> while test pass!
Mr.Chen, students and TAs are the best!
kobayashi@kobayashi-virtual-machine ~/D/assignment>
```

vim /home/kobayashi/Desktop/assignment

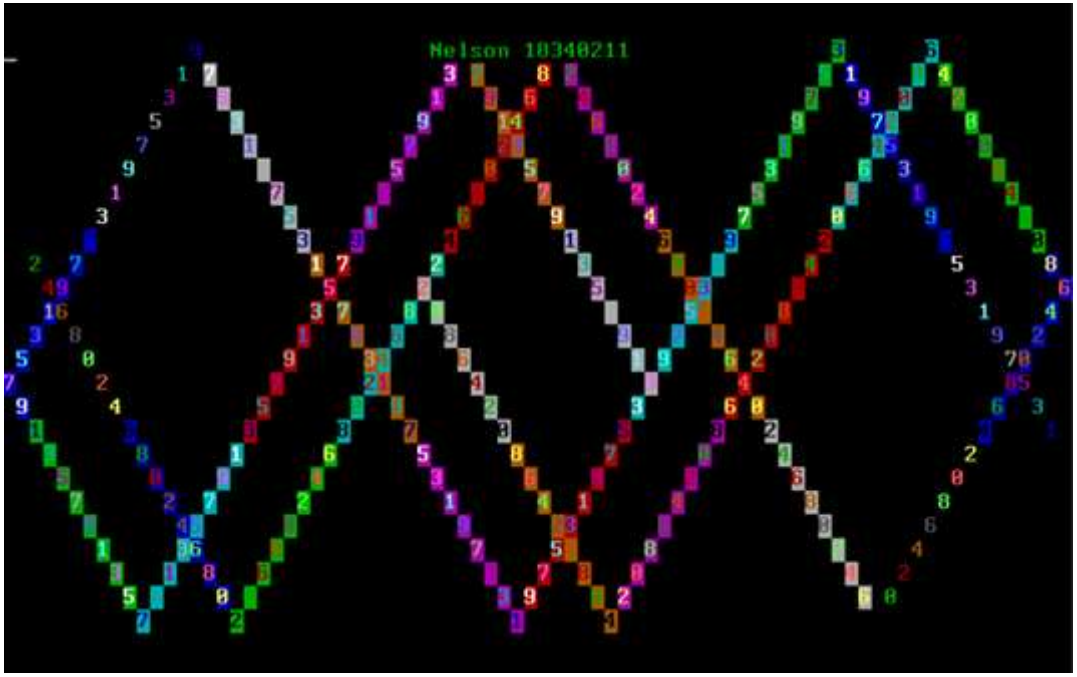
File Edit View Search Terminal Help

```
; If you meet compile error, try 'sudo apt install gcc-multilib g++-multilib' first

%include "head.include"
; you code here

your_if:
; put your implementation here
; load the value of a1 into eax
mov eax, [a1]

; if a1 < 12, then goto if1
mov ebx, 12
cmp eax, ebx
jl if1

; else if a1 < 24, then goto if2
mov ebx, 24
cmp eax, ebx
jl if2

; else, if_flag = a1 << 4
shl eax, 4
mov [if_flag], eax
jmp endif

; if1: if_flag = (a1 / 2) + 1
if1:
shr eax, 1
inc eax
mov [if_flag], eax
jmp endif

; if2: if_flag = (24 - a1) * a1
if2:
mov ecx, 24
sub ecx, eax
imul ecx, eax
mov [if_flag], ecx
jmp endif

endif:

your_while:
; put your implementation here
"student.asm" 96L, 1575C                                    8,1          Top
```

Assignment 4 汇编小程序

内容

- **字符弹射程序**。请编写一个字符弹射程序，其从点（2，0）处开始向右下角45度开始射出，遇到边界反弹，反弹后按45度角射出，方向视反弹位置而定。同时，你可以加入一些其他效果，如变色，双向射出等。注意，你的程序应该不超过510字节，否则无法放入MBR中被加载执行。



实验步骤

- 编写bounce.asm如下：

```
; bounce.asm
org 0x7c00
[bits 16]
; initialize segments
xor ax, ax
mov ds, ax
mov ss, ax
mov es, ax
mov fs, ax
mov gs, ax

; set stack pointer
mov sp, 0x7c00
mov ax, 0xb800
mov gs, ax

; set cursor position to (2,0)
mov ah, 0x02
mov bh, 0x00
mov dh, 0x02
mov dl, 0x00
int 0x10

; the screen size is 80*25

; I use bl to save the color and the direction at the same time
; 0x01: right up, blue
; 0x02: right down, green
; 0x04: left up, red
; 0x08: left down, yellow
```

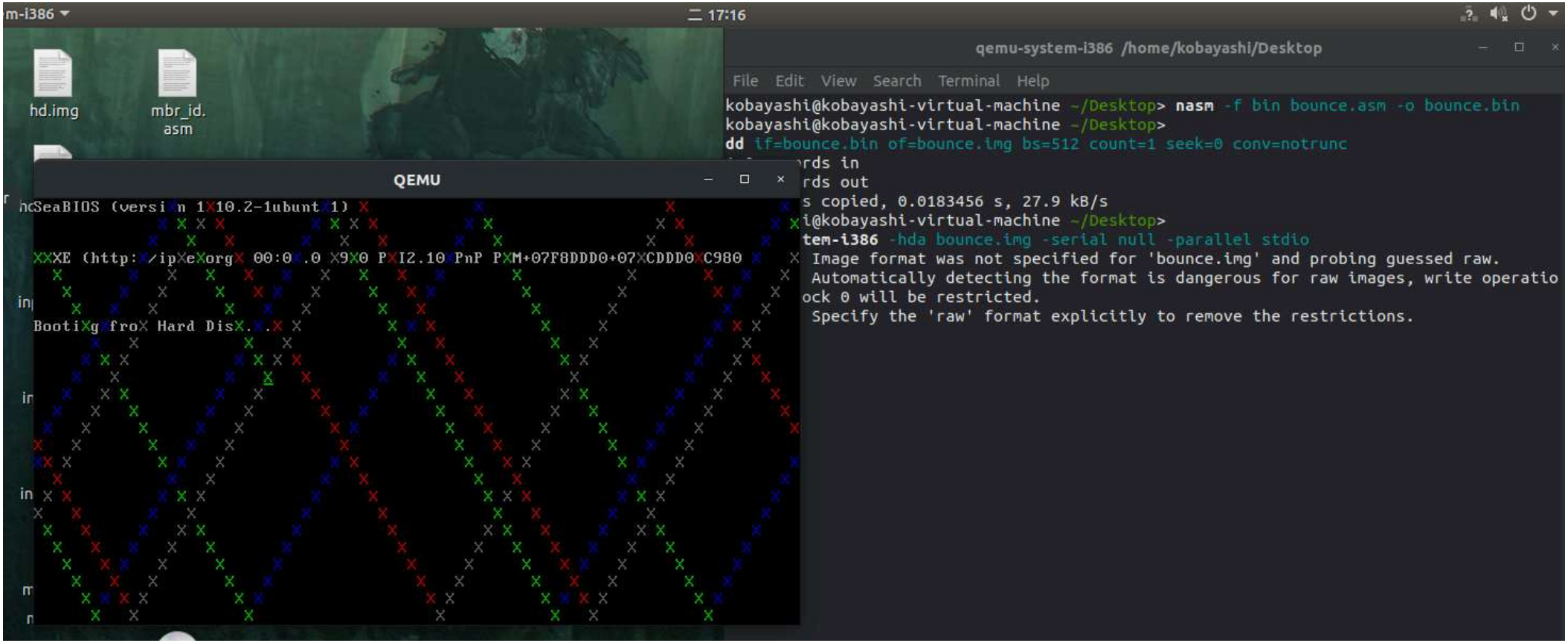
```
mov bl, 0x02 ; the initial direction is right down

move_loop:
    ; judge the current direction
    ; and choose the next move
    cmp bl, 0x01
    je move_right_up
    cmp bl, 0x02
    je move_right_down
    cmp bl, 0x04
    je move_left_up
    cmp bl, 0x08
    je move_left_down

move_right_up:
    ; judge if the cursor is at the end of top
    cmp dh, 0x00
    je change_right_down
    ; judge if the cursor is at the end of right
    cmp dl, 0x50
    je change_left_up
    ; move
    mov ah, 0x02
    inc dl
    dec dh
    int 0x10
    jmp move_next

move_right_down:
    ; judge if the cursor is at the end of bottom
    cmp dh, 0x18
    je change_right_up
```

```
    ; judge if the cursor is at the end of right
    cmp dl, 0x50
    je change_left_down
    ; move
    mov ah, 0x02
    inc dl
    inc dh
    int 0x10
    jmp move_next

move_left_up:
    ; judge if the cursor is at the end of top
    cmp dh, 0x00
    je change_left_down
    ;  judge if the cursor is at the end of left
    cmp dl, 0x00
    je change_right_up
    ; move
    mov ah, 0x02
    dec dl
    dec dh
    int 0x10
    jmp move_next

move_left_down:
    ; judge if the cursor is at the end of bottom
    cmp dh, 0x18
    je change_left_up
    ; judge if the cursor is at the end of left
    cmp dl, 0x00
    je change_right_down
    ; move
```

```
    mov ah, 0x02
    dec dl
    inc dh
    int 0x10
    jmp move_next

move_next:
    ; show the path
    mov ah, 0x09
    mov al, 'X'
    mov cx, 0x01
    int 0x10

    ; delay
    push cx
    push dx
    call delay
    pop dx
    pop cx

    jmp move_loop

jmp $ ; jump to current address (infinite loop)

; delay function
delay:
    mov cx, 0x0ff
delay_loop1:
    mov dx, 0x0ffff
delay_loop2:
    dec dx
    jnz delay_loop2
```

```
    dec cx
    jnz delay_loop1
    ret

; change direction
change_right_down:
    mov bl, 0x02
    jmp move_right_down
change_left_up:
    mov bl, 0x04
    jmp move_left_up
change_left_down:
    mov bl, 0x08
    jmp move_left_down
change_right_up:
    mov bl, 0x01
    jmp move_right_up

times 510-($-$$) db 0
db 0x55, 0xaa ; boot signature, meaning this is a bootable mbr
```

- 最终启动后的效果如下：



总结

通过这次实验我熟悉了汇编的编写以及mbr文件的制作流程，但在本次实验中编写的代码中仍有这许多需要改进的地方，我将会在今后的实验中尝试将代码写得更加简洁、高效。