OSlab1 实验报告

王亚楠　141220107

邮箱：wyn1996@hotmail.com

实验进度

已完成所有试验要求，并设计了拼图小游戏。
使用的 gcc 版本：32 位

试验心得

1. 原来实验使用的是标准 vga 显示，由于要实现拼图的功能，对显示的像素和色彩都有更高的要求，因此我采用了真彩色模式。在 start.S 中改动了切换到图形模式的汇编代码。如图所示，

```
#    movw    $0x13,    %ax
     movw    $0x4f02, %ax       #Enable 24 RGB
     movw    $0x0115, %bx
     int     $0x10              # 使?~T?BIOS中?~V??~H~G?~M??~T?~C容?~X~S??
~@~B?~\??~^?~N?OS?~W??~M?~\~@此步骤_____

     movw    $0x2000, %di       #About 24 RGB video mode
     movw    $0x4f01, %ax
     movw    $0x0115, %cx
     int     $0x10
```

其中通过’movw　　$4f02,　　%ax’来将显示模式变成所需要的真彩色模式，$0x0115 是 800*600＊24bit 对应的编号，’movw　$0x2000,　　%di’用来设定 ModeInfoBlock 所在的内存地址。
设置完真彩色模式后，我通过直接将图片对应的像素点映射到显存地址来实现图片的显示，通过查看 ModeInfoBlock 数组，得知显存地址的偏移量是 40bites，所以通过 0x2000+40 来得到显存的起始地址，然后便可以按照我的想法进行图像的显示了。

2. 关于将图片转换成像素点数组的过程中遇到了一些比较坑的问题。
原来我是通过写一个 shell 脚本来实现转换的，开始的代码如下：

```
#!/bin/bash

tar=puzzle/ren12.h
file=ren/12.bmp
name=ren12
w=`hexdump -s 18 -n 4 -e '4/4 "%d" ""' $file`
h=`hexdump -s 22 -n 4 -e '4/4 "%d" ""' $file`

bytes=$(( ((3 * $w + 3) / 4) * 4 ))

total_bytes=$(( $h * $bytes ))

echo total_bytes : $total_bytes

echo unsigned char $name[] = { > $tar
hexdump -s 54 -n $total_bytes -e " 1/1 \"%u,\" \"\n\"" $file >> $tar
echo }\; >> $tar
sed -i "s/,}/}/g" $tar
```

后来发现，图片的显示存在问题，部分像素点对应的位置发生了缺失和移位，然后我查看了像素点数组，发现其中有很多'*'，我一开始直接去掉了'*'，然后发现问题依旧存在。后来通过请教欧先飞后知道，因为这个脚本会将很多 0 看作一个'*'，所以缺失了很多像素点。然后将转换中最关键的一行代码化成了如下所示，

```
echo unsigned char $name[] = { > $tar
hexdump -s 54 -n $total_bytes -e " $bytes/1 \"%u,\" \"\n\"" $file >> $tar
echo }\; >> $tar
```

即通过整个图片大小 size 的转换来获得，而不是 1 个 bite1 个 bite 的获得，这样就可以很好的规避之前'*'的问题。

另外，通过和助教王诲喆沟通，他教给了我一个更加容易的方法，就是通过 imagemagick 工具对图片进行转换，先将要转换的图片变成.bgr 格式，然后通过 xxd 将.bgr 文件写到一个数组中去。

3. 关于键盘中断的话也想说些什么。
   当然，经过 PA 以后，设定键盘中断并不是什么难事。先注册中断号，然后在中断处理函数中对接受到的中断号(irq=1001，即键盘中断)进行相应的处理即可。
   但是，运行打字小游戏的时候发现，我电脑的键盘扫描码和实验所给的不一致，应该说是和 qemu 中内定的键盘扫描码不一致吧，暂时还没找到处理的方法。

4. 实现串口输出
   通过对输出的格式要求对%d,%x,%c,%s 依次进行判断然后按照规则调用 printer 函数进行输出。
   之前没有接触过 void **args 这样的存放字符串的指针数组，所以一开始不知道怎么获得%d 相对的字符信息，所以还是纠结了比较久的。

5. 关于拼图游戏
   其实这个游戏实现的话还是比较简单的，我是直接通过一个二维数组来表示每张图的位置。在 game.c 中获得当前按键的扫描码，然后判断扫描码对数组进行更新。然后更新完后，对拼图改变的两个位置重新写显存即可。

6. 对 lab1 整个的理解
   首先从 start.S 来说，先是关中断，设定真彩色显示模式、调用 BIOS，然后设置段寄存器，打开 A20 地址线，接着对 GDT 进行设置，接着进入保护模式，对段寄存器和栈进行设置。
   整个游戏的话，首先要对串号端口、时钟端口、中断符表以及我们所需要用到的时钟中断和键盘中断进行初始化，然后打开中断，进入游戏。
   lab1 加深了我对中断、模式切换的理解，同时自己实现一个小游戏也让人充满自豪感。