Day 5

Phase 1

TASK：使用结构体的方式来获取先前保存的启动信息。

先要获取先前保存的启动信息非常简单，只要直接读取asmhead当中保存启动信息的那些位置就行。

例如我们想获取解析度和VRAM地址，我们只需要执行

```
short xsize, ysize;
char *vram;
xsize = *((short*)0x0ff4);
ysize = *((short*)0x0ff6);
vram = (char*)(*((unsigned int*)0x0ff8));
```

但这样看起来一点也不优雅。

我们之前保存启动信息时，启动信息的保存位置时连贯的，而c语言将结构体也解读成连贯的内存地址，我们可以利用这个特性来更优雅的书写这段程序。

```
struct bootinfo {
    char cyls, leds, vmode, _;
    short scrnx, scrny;
    char* vram;
};
```

由于保存的时候，VMODE和SCRNX中间间隔了一个字节的位置，所以我们随意写一个char，来避免c语言错误的计算接下来的数据的地址。

然后我们就可以更加优雅的获取这些启动信息了！如下：

```
struct bootinfo *binfo = (bootinfo*)0x0ff0;
short xsize = binfo->scrnx;
short ysize = binfo->scrny;
```

> 书上的关于->符号的部分早在大一的时候就已经学过了，所以我们跳过他

Phase 2

TASK：显示字符、字符串、变量值

进入到32位模式之后，我们无法再调用BIOS的打印字符函数，所以我们要自己搞。打印点阵字，我们只需要将对应的像素置黑就行了。

我们先搞一个对单像素进行操作的宏函数，来方便我们写程序

```
#define setpixel(x, y, c) (vram[(y) * xsize + (x)] = (c))
```

```
void putfont8(char *vram, int xsize, int x, int y, char c, char *font) {
    int i, d;
    for (i = 0; i < 16; i++) {
        d = font[i];
        if (d & 0x80) setpixel(x, y + i, c);
        if (d & 0x40) setpixel(x + 1, y + i, c);
        if (d & 0x20) setpixel(x + 2, y + i, c);
        if (d & 0x10) setpixel(x + 3, y + i, c);
        if (d & 0x08) setpixel(x + 4, y + i, c);
        if (d & 0x04) setpixel(x + 5, y + i, c);
        if (d & 0x02) setpixel(x + 6, y + i, c);
        if (d & 0x01) setpixel(x + 7, y + i, c);
    }
}
```

不过感觉还是书的作者写的程序更加的高校一些喔，不过作者的程序也有改动空间，做乘法的速度比加法慢很多，稍加修改就可以减少乘法运算和加法运算的次数。

```
void putfont8(char *vram, int xsize, int x, int y, char c, char *font)
{
    int i;
    char *p, d /* data */;
    p = vram + y * xsize + x;
    for (i = 0; i < 16; i++, p += xsize) {
        // p = vram + (y + i) * xsize + x;
        d = font[i];
        if ((d & 0x80) != 0) { p[0] = c; }
        if ((d & 0x40) != 0) { p[1] = c; }
        if ((d & 0x20) != 0) { p[2] = c; }
        if ((d & 0x10) != 0) { p[3] = c; }
        if ((d & 0x08) != 0) { p[4] = c; }
        if ((d & 0x04) != 0) { p[5] = c; }
        if ((d & 0x02) != 0) { p[6] = c; }
        if ((d & 0x01) != 0) { p[7] = c; }
    }
    return;
}
```

我们还不能立即开始打印我们的字符，我们还没有字库呢！我们这就不要自造轮子了，使用作者提供的字库吧。

```
extern char hankaku[4096];
putfont8(binfo->vram, binfo->scrnx,  8, 8, COL8_FFFFFF, hankaku + 'H' * 16);
putfont8(binfo->vram, binfo->scrnx, 16, 8, COL8_FFFFFF, hankaku + 'E' * 16);
putfont8(binfo->vram, binfo->scrnx, 24, 8, COL8_FFFFFF, hankaku + 'L' * 16);
putfont8(binfo->vram, binfo->scrnx, 32, 8, COL8_FFFFFF, hankaku + 'L' * 16);
putfont8(binfo->vram, binfo->scrnx, 40, 8, COL8_FFFFFF, hankaku + 'O' * 16);
putfont8(binfo->vram, binfo->scrnx, 56, 8, COL8_FFFFFF, hankaku + 'W' * 16);
putfont8(binfo->vram, binfo->scrnx, 64, 8, COL8_FFFFFF, hankaku + 'O' * 16);
putfont8(binfo->vram, binfo->scrnx, 72, 8, COL8_FFFFFF, hankaku + 'R' * 16);
putfont8(binfo->vram, binfo->scrnx, 80, 8, COL8_FFFFFF, hankaku + 'L' * 16);
putfont8(binfo->vram, binfo->scrnx, 88, 8, COL8_FFFFFF, hankaku + 'D' * 16);
```

执行结果如下

然后我们尝试打印字符串吧！由于字符串长度有可能超出屏幕边界，我们先不考虑换行，我们就进行简单的截断吧！

```
void putfont8(char *vram, int xsize, int ysize, int x, int y, char c, char *font)
{
    int i;
    char *p, d /* data */;
    p = vram + y * xsize + x;
    for (i = 0; i < 16 && y + i < ysize; i++, p += xsize) {
        // p = vram + (y + i) * xsize + x;
        d = font[i];
        if (x < xsize && (d & 0x80) != 0) { p[0] = c; }
        if (x + 1 < xsize && (d & 0x40) != 0) { p[1] = c; }
        if (x + 2 < xsize && (d & 0x20) != 0) { p[2] = c; }
        if (x + 3 < xsize && (d & 0x10) != 0) { p[3] = c; }
        if (x + 4 < xsize && (d & 0x08) != 0) { p[4] = c; }
        if (x + 5 < xsize && (d & 0x04) != 0) { p[5] = c; }
        if (x + 6 < xsize && (d & 0x02) != 0) { p[6] = c; }
        if (x + 7 < xsize && (d & 0x01) != 0) { p[7] = c; }
    }
    return;
}
void pustring8(char *vram, int xsize, int ysize, int x, int y, char c, char *fontku, char
*str) {
    for (; *str; str++, x += 8) {
```

```
            putfont8(vram, xsize, ysize, x, y, c, hankaku + (*str) * 16);
        }
    }
    // 以下为测试代码
    pustring8(binfo->vram, binfo->scrnx, binfo->scrny, 8, 8, COL8_FFFFFF, hankaku, "A Warm
    Welcome From Wallace CHU. Hello World with News!");
    // 加个阴影？弄个背景吧，不然看不出来
    init_screen(binfo->vram, binfo->scrnx, binfo->scrny);
    pustring8(binfo->vram, binfo->scrnx, binfo->scrny, 9, 9, COL8_000000, hankaku, "A Warm
    Welcome From Wallace CHU. Hello World with News!");
    pustring8(binfo->vram, binfo->scrnx, binfo->scrny, 8, 8, COL8_FFFFFF, hankaku, "A Warm
    Welcome From Wallace CHU. Hello World with News!");
```

以下为运行结果

截断成功！没有显示异常

阴影看起来挺自然

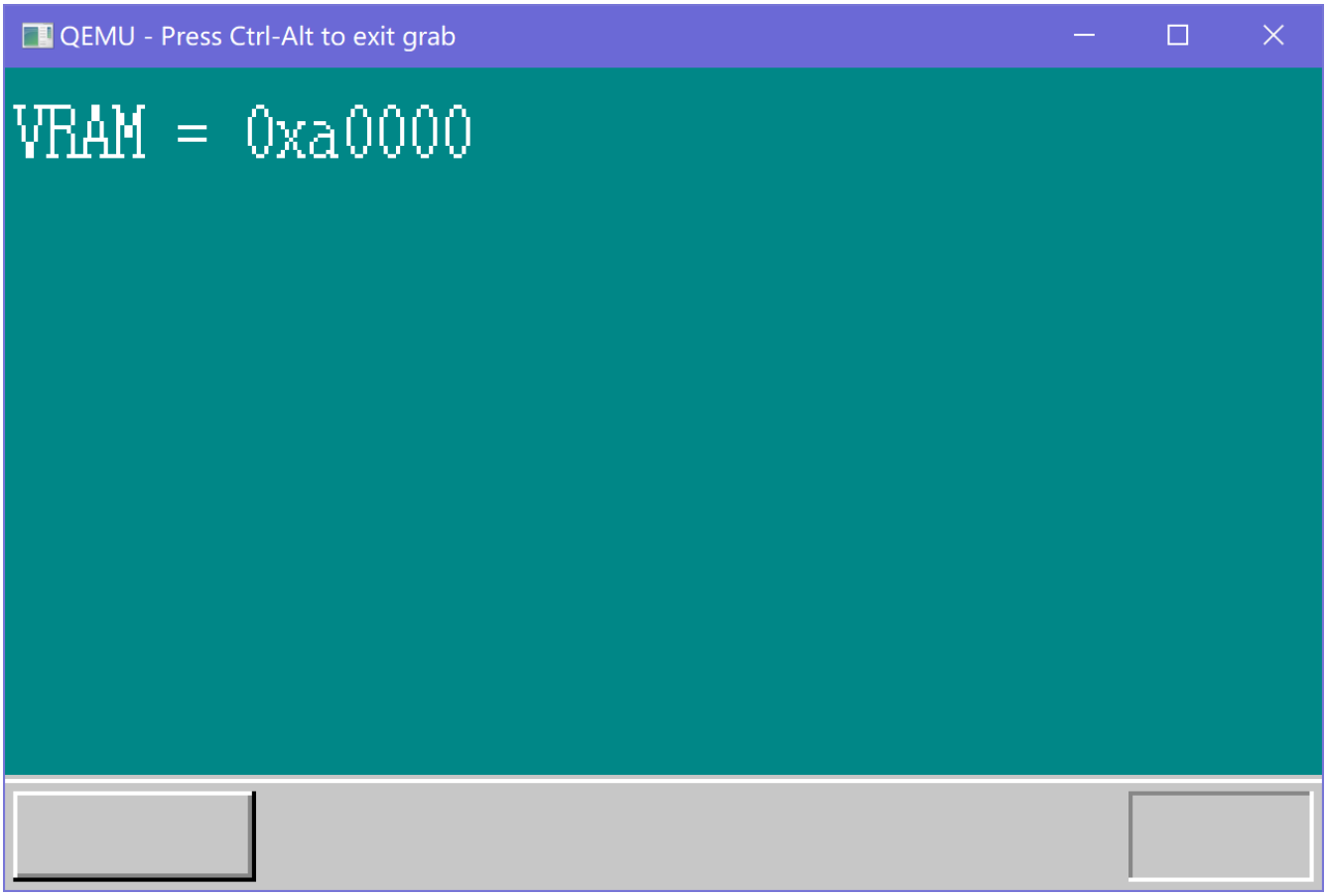
显示变量值。

先用sprintf把变量值打印到字符串当中，然后再输出字符串就好咯。

稍加修改

```
char tmp[81];
sprintf(tmp, "VRAM = 0x%x", (unsigned int)binfo->vram);
pustring8(binfo->vram, binfo->scrnx, binfo->scrny, 2, 8, COL8_FFFFFF, hankaku, tmp);
```

结果如下



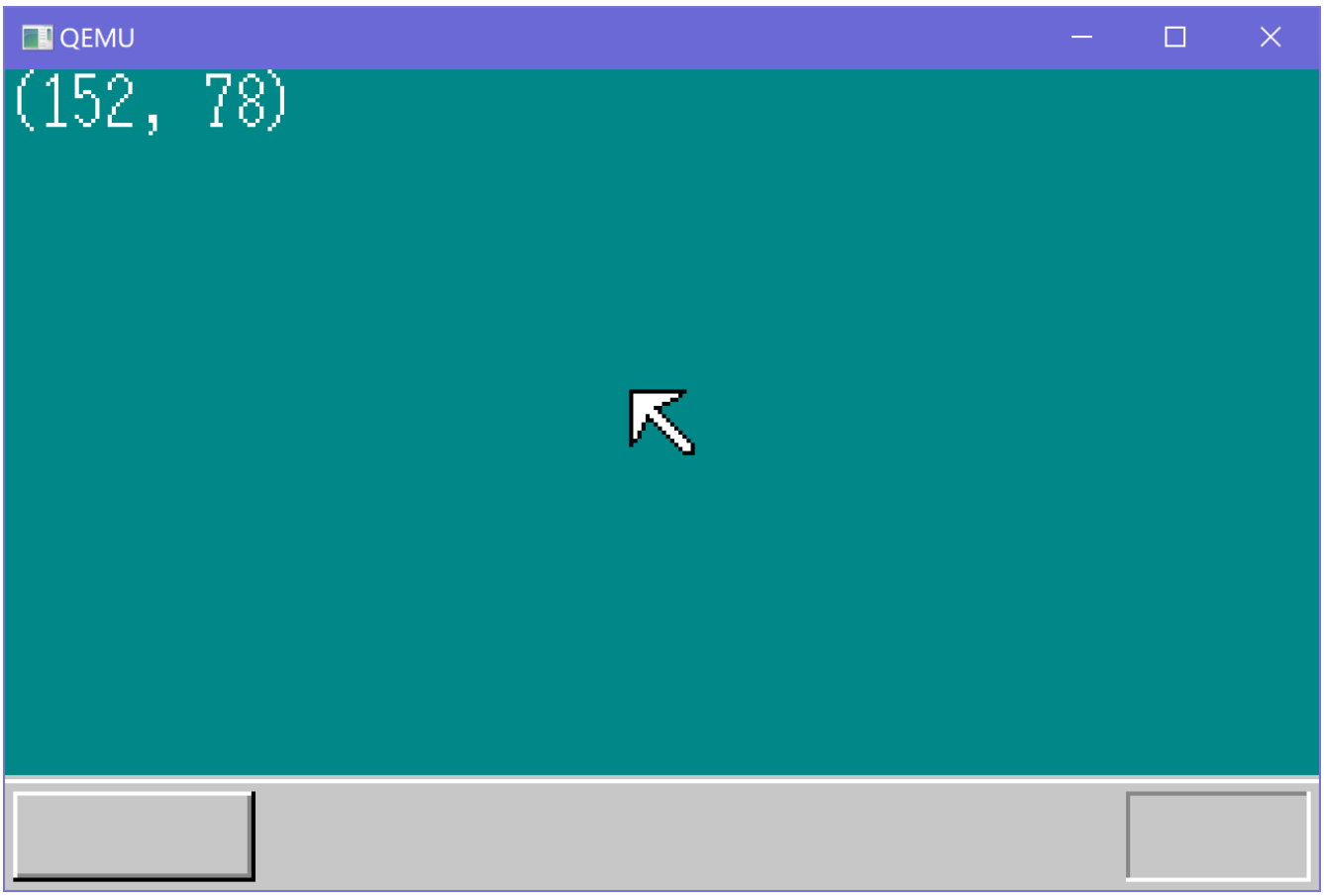
Phase 3

鼠标指针

首先先想办法打印一个指针吧！

和字符打印的方式大同小异，就用书上的代码吧！稍微改改指针的形状。

效果如下



Phase 4

TASK：GDT 与 IDT设定

GDT

当我们在学习计算机组成原理的时候，在存储系统这一章，我们学习并了解了主存-外存层次的基本信息映射单位——段。段是按照程序的逻辑结构划分的多个相对独立的逻辑信息单位.段的长度不固定，但通常有一个允许的最大长度。

段的优点是逻辑独立，易于编译、管理、修改和保护，便于多道程序共享。段的动态可变段长，允许自由调度，有效利用主存空间。

段的缺点是段长不固定，主存分配算法复杂，容易在段间留下碎片，造成浪费。

段的大小、起始地址管理属性等信息存储在主存中的GDT（global segment descriptor table）中。 我们需要先设置GDT的位置等信息。

IDT

IDT是记录中断对应关系的表。在我理解，就是可以用她来设定不同中断号码对应的中断服务程序。我们将来将会借助它来实现对键盘、鼠标等输入设备的信息获取。

至此我们已经完成了Day5内容的学习与实验