Git：<https://github.com/Soiaoder/x86homework.git>

年月日输出

第一道题比较简单，利用call和ret，能一定程度上减少重复的代码。

1. 子程序问题

在最初的版本里，依照之前的格式，在

main proc far

…

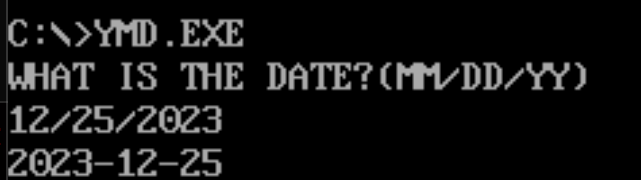
…

main end

codeseg end

end main

中编写代码,在运行过后会发现程序卡死，在不断调试中发现与call和ret。删掉call和ret程序能够正常运行。在深入了解子程序后，发现问题在于main本身就是一个子程序,在子程序中定义其他子程序并调用这是行不通的



工资计算：

这道题需要用到之前所学的所有知识，并且还要自学显示器缓冲区的一些知识

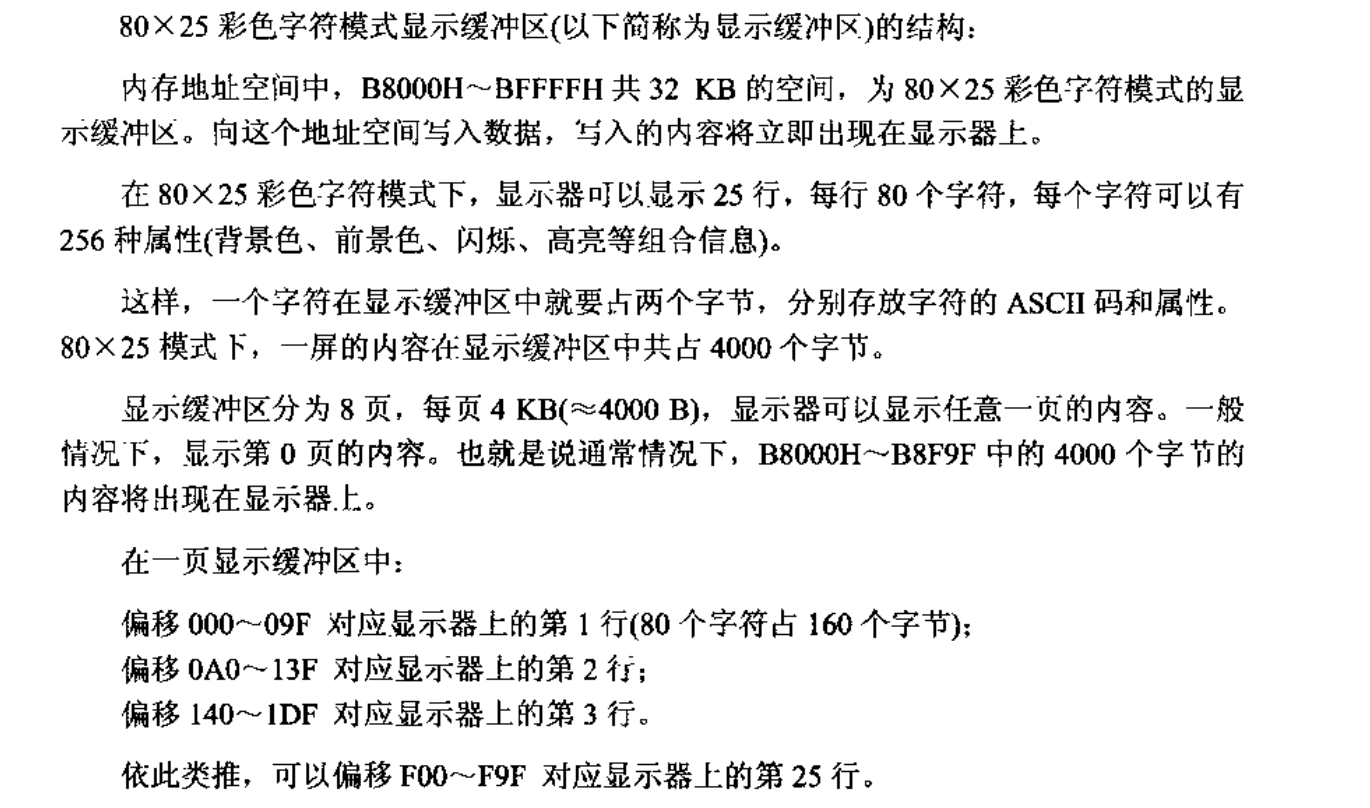
1. 对于存储器的赋值问题

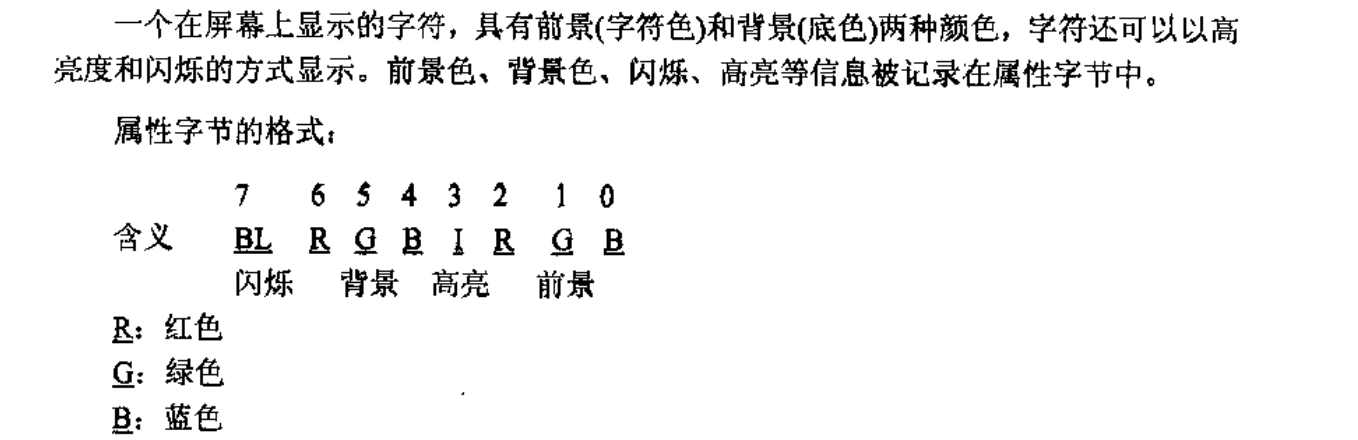
最初尝试使用(举例)mov es:[bx],1，编译器会报错，显示必须指定大小，查阅资料后了解到：

寄存器有明确的字节或字类型，有寄存器参与的指令其操作数类型就是寄存器的类型。对于存储器单元与立即数同时作为操作数的情况，必须显式指明。用byte ptr指示字节类型，用word ptr指示字类型

1. 显示器缓冲区的一些心得

重要资料：





总结：我们可以想象显示器为80\*25的网格，每一格对应显示的一个字符

下面即一格：

|  |  |
| --- | --- |
| 0 1 2 3 4 5 6 7 | 8 9 10 11 12 13 14 15 |
| Ascii码对应字符 | 字符属性（闪烁，背景，高亮，颜色） |

想要精准定位到目标格，可以使用行，列，

举例：

DL为行数，CL为列数，已知每一行A0H，令BL=DL\*A0H+CL\*2即为目标格的偏移地址。

1. 子程序的小技巧

在看别人写的代码过程中，发现许多人都有一个习惯，在该子程序的开头，将用到的寄存器全部push，在ret之前又全部pop。这能有效的保护寄存器。也许某些情况下没有push，pop寄存器程序也能成功运行，但这个行为添加了一层保险。

1. 32位双字的除法运算

汇编能够处理的商最多就16位，但是显然这道题在/10的过程中商会超范围，需要另寻算法。

假设：被除数为A，除数是N（本题中为10）

Int（A/N）为商 ， res（A/N）为余数

A=int（A/N）\*N+res（A/N）

现将A分为高16位H和低16位D

A=H\*65536+D，也可以理解为H左移4位+D

A/N=（H\*65536+D）/N

=( (Int（H/N）\*N+res(H/N)) \*65536 + D)/N

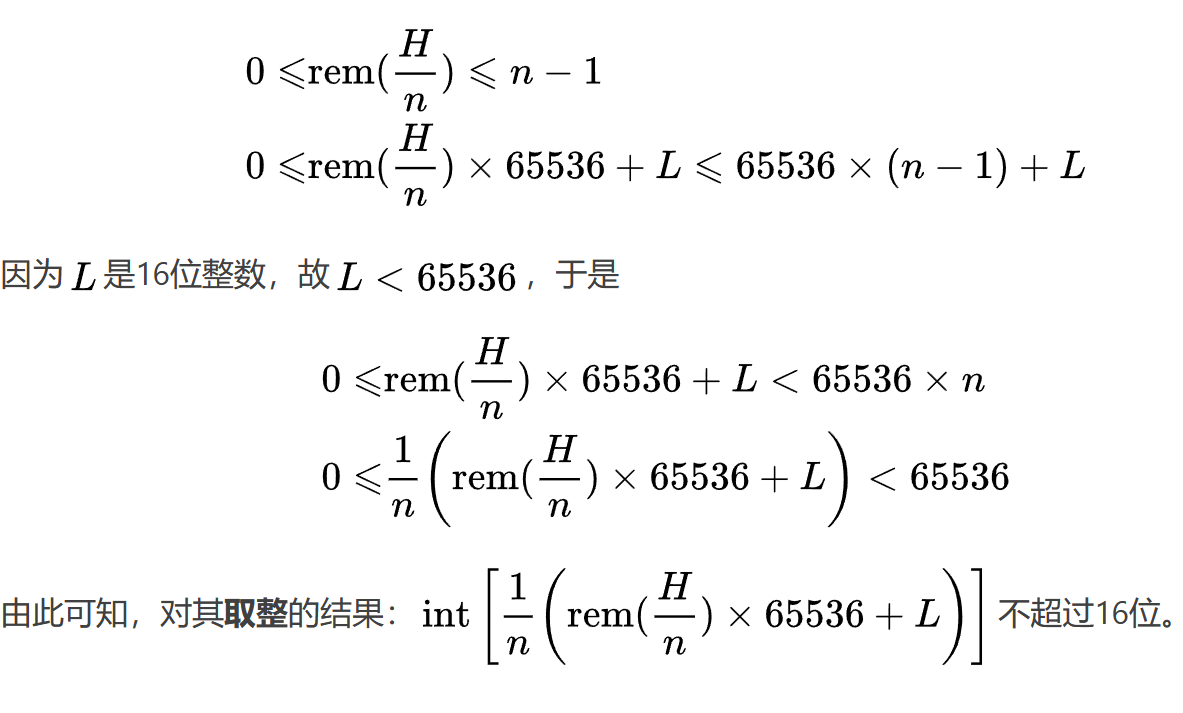
=int(H/N)\*N\*65536/N+ (res(H/N)\*65536+D)/N

=int(H/N)\*65536 + (res(H/N)\*65536+D)/N

因为N为10，即res（H/N）<10

* (res(H/N)\*65536+D)/N不会超过16位,且其余数同样也是A/N的余数

其实只要N是16位的数，都可以用上面的方法，



而且不是32位，16位使用该方法同样能得到正确结果

1. 倒叙打印

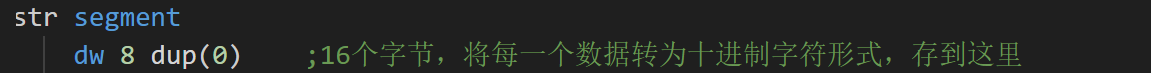
在看别人的代码中，他们16转10进制并输出的方式与我之前使用的方法略有不同，并且更加优秀。

采用子程序的方法，代码使用率提高，重复性降低。

大致思想为用分为三个步骤

1. 使用循环，在循环中不断使用divdw子程序，将每次余数push进栈中，当商为0时，退出循环（假设数为123456789，栈：底987654321顶，弹出后刚好为正序）

使用bl来计数

1. 将bl赋值给cl，使用循环，将栈顶依次取出，并将值+30h后，放入提前准备好的str

并在循环结束后末尾添0（不是’0’,就是0），标志结束

1. 最后再使用循环根据需求利用str进行输出即可

本题用到的数据随存随用，所以用一个str dw 8 dup（0）即可，如果想长期保存合理利用数据段即可

6.

显示器缓冲区存在直接覆盖当前画面的问题，会使画面重叠，能想到解决办法即先将80\*25全部黑色填满，再打印。比较简单。