赵明泽 2151974

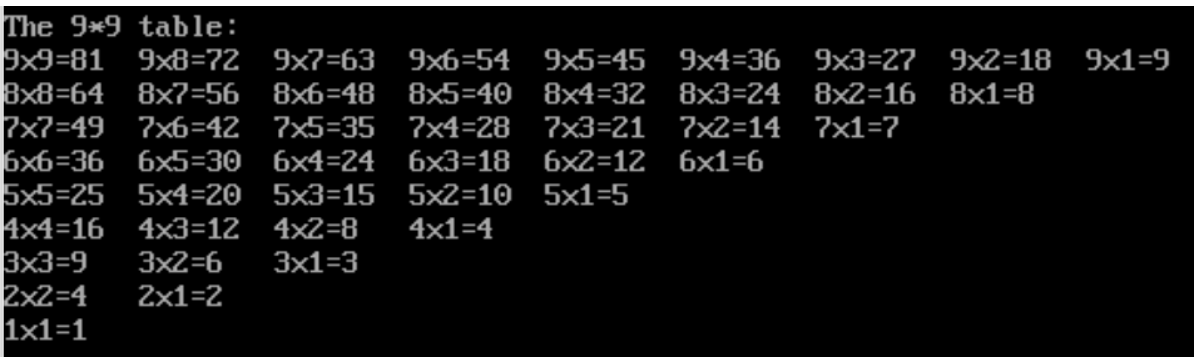
目录

九九乘法表的输出

要求：要求双循环都用 `LOOP` 指令。

目的：熟悉栈的使用。

结果输出：



分析：

首先声明一个数据段，用来存放提示语、乘数和结果

```
data segment
        msg  db "The 9*9 table:", 0ah, 0dh, '$'
        num  dw ?,?,?,?
        res  db ?,?,?,?
data ends
```

接下来，定义外圈循环的次数，初始化数据段地址，打印提示语

```
mov  cx, 9
mov  ax, data
mov  ds, ax
lea  dx, msg
mov  ah, 09h
int  21h
```

程序分为两层循环，外层循环控制打印行数，也就是第一个乘数；内圈循环控制打印列数，也就是第二个乘数。二者相乘后，通过进制转换和ASCII码转换，最终将数码打印到屏幕上。

下面是参考的C语言代码。与汇编不同，C语言乘数从小到大，而汇编从大到小。

```
#include <stdio.h>
int main()
{
    for(int i=1;i<=9;i++){
        for(int j=i;j<=9;j++){
            printf("%d*%d=%d\t",i,j,i*j);
        }
        printf("\n");
    }
    return 0;
}
```

参考在线编译器https://godbolt.org/给出的汇编结果，不难发现，从小到大的顺序使用了大量跳转语句，与本题要求不符。故从大到小比较简单。

```
.LC0:
        .string "%d*%d=%d\t"
main:
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     DWORD PTR [rbp-4], 1
        jmp     .L2
.L5:
        mov     eax, DWORD PTR [rbp-4]
        mov     DWORD PTR [rbp-8], eax
        jmp     .L3
.L4:
        mov     eax, DWORD PTR [rbp-4]
        imul    eax, DWORD PTR [rbp-8]
        mov     ecx, eax
        mov     edx, DWORD PTR [rbp-8]
        mov     eax, DWORD PTR [rbp-4]
        mov     esi, eax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        add     DWORD PTR [rbp-8], 1
.L3:
        cmp     DWORD PTR [rbp-8], 9
        jle     .L4
        mov     edi, 10
```

```
        call    putchar
        add     DWORD PTR [rbp-4], 1
.L2:
        cmp     DWORD PTR [rbp-4], 9
        jle     .L5
        mov     eax, 0
        leave
        ret
```

程序执行的大致流程：

```
loop1:
        ;更新内外圈乘数
loop2:
        ;显示乘数
        ;显示x号
        ;显示第二个乘数
        ;显示=号
        ;计算两数相乘的结果，并显示
        ;把各位转换为数值，如AX中的81，转换为8,1存在内存中
        ;输出内存中存放的转换数值数
        ;输出水平制表栏
        loop loop2      ;内层循环结束
        ;输出换行
        ;还原外层计数
        loop loop1      ;外层循环结束
```

完整代码：

```
data segment
        msg  db "The 9*9 table:", 0ah, 0dh, '$'
        num  dw ?,?,?,? ;声明一个字（16位）数组num，该数组有四个元素，但初始化为问号?，表示这些值将在程序运行时被填充
        res  db ?,?,?,? ;声明一个字节（8位）数组
data ends

code segment
        assume cs:code, ds:data
start:
        mov  cx, 9      ;外圈循环次数
        mov  ax, data   ;将数据段的地址存储在寄存器AX中
        mov  ds, ax     ;将数据段寄存器DS设置为数据段的地址，这样程序可以访问数据段中的数据
        lea  dx, msg    ;打印提示语
        mov  ah, 09h    ;将寄存器AH设置为09h，这是DOS的功能号，表示要执行字符串输出
        int  21h        ;触发DOS中断21h，以执行字符串输出操作，输出提示语
loop1:
        mov  [num], cx  ;存放乘数
        push cx         ;保存外层计数
        push cx         ;乘数进栈
loop2:
        ;显示乘数
        mov  dx, [num]  ;将num数组的第一个元素（乘数）加载到寄存器DX中
        add  dx, 30h    ;转换到ASCII
        mov  ah, 02h    ;将寄存器AH设置为2，表示要执行字符输出操作
        int  21h        ;触发DOS中断21h，以输出当前乘数
```

```
        ;显示x号
        mov  dl, 78h     ;将DL寄存器设置为78h，表示要输出字符'x'
        mov  ah, 02h     ;将寄存器AH设置为2，表示要执行字符输出操作
        int  21h         ;触发DOS中断21h，以输出'x'
        ;显示第二个乘数
        mov  [num+1], cx;存放第二个乘数
        push cx          ;第二个乘数进栈
        mov  dx, cx      ;将第二个乘数加载到寄存器DX中
        add  dx, 30h     ;转换到ASCII
        mov  ah, 02h     ;将寄存器AH设置为2，表示要执行字符输出操作
        int  21h         ;触发DOS中断21h，以输出第二个乘数
        ;显示=号
        mov  dl, 3dh     ;将DL寄存器设置为3dh，表示要输出字符'='
        mov  ah, 02h     ;将寄存器AH设置为2，表示要执行字符输出操作
        int  21h         ;触发DOS中断21h，以输出'='
        ;计算两数相乘的结果，并显示
        pop  dx          ;取出第二个乘数
        pop  ax          ;取出第一个乘数
        push ax          ;第一个乘数再次进栈，在下次内层循环中推出再次使用
        mul  dx          ;相乘，结果在AX中
        mov  bx, 10      ;准备除以10
        mov  si, 2       ;循环2次，最大到十位 (乘法表最大为81,所以最大到十位)
toDec:                   ;把各位转换为数值，如AX中的81，转换为8,1存在内存中
        mov  dx, 0       ;清空DX寄存器
        div  bx          ;除10法得到各个位上的数值
        mov  [res+si], dl;余数为该位上的值，第一次循环为个位，第二次为十位，存到内存中
        dec  si          ;计数器自减
        cmp  ax, 0       ;商是否为0，为0算法结束
        ja   toDec       ;跳转，继续处理下一位
output:                  ;输出内存中存放的转换数值数
        inc  si          ;计数器自增
        mov  dl, [res+si];结果移动到DX寄存器低八位
        add  dl, 30h     ;转为ASCII码
        mov  ah, 02h     ;将寄存器AH设置为2，表示要执行字符输出操作
        int  21h         ;触发DOS中断21h，以输出分解后的数码
        cmp  si, 2       ;将计数器与2比较
        jb   output      ;跳转，把所有结果数码输出
        mov  dl, 09h     ;把水平制表栏移动到DX寄存器低八位
        mov  ah, 02h     ;将寄存器AH设置为2，表示要执行字符输出操作
        int  21h         ;触发DOS中断21h，以输出水平制表栏
        loop loop2       ;内层循环结束
        mov  dx, 0ah     ;把换行移动到DX寄存器低八位
        mov  ah, 02h     ;将寄存器AH设置为2，表示要执行字符输出操作
        int  21h         ;触发DOS中断21h，以输出换行
        pop  cx          ;内层计数
        pop  cx          ;还原外层计数
        loop loop1       ;外层循环结束
        mov ah, 4ch
        int 21h
code ends
        end start
```

遇到问题1：

在程序开始时，并没有显式声明栈段，在不事先声明栈段的情况下，发现后续程序中直接执行 `push` 和 `pop` 操作，发现可行不会报错，这是什么原理？在内存的哪里存放该栈段？在执行 `push` 和 `pop` 操作时又动用了哪里的栈指针和起始地址？

心得体会1：

在8086汇编语言中，如果不显式声明栈段（stack segment），程序会默认使用一个栈段，通常会将栈指针（stack pointer）初始化为适当的内存地址。这是因为栈是一个非常重要的数据结构，几乎所有的汇编程序都需要使用栈来存储临时数据、函数参数和返回地址等。

默认情况下，栈通常被初始化为一个合适的内存区域，这个内存区域可能在操作系统或编译器级别进行设置。具体的栈段地址通常存储在 SS 寄存器中，而栈指针存储在 SP 寄存器中。程序可以直接使用 `push` 和 `pop` 指令来操作栈，而不需要显式声明栈段。

当程序使用 `push` 指令时，它将数据推送（压入）到栈中，同时 SP 寄存器会递减，指向栈中的下一个位置。当程序使用 `pop` 指令时，它将数据弹出栈，同时 SP 寄存器会递增，指向栈中的前一个位置。

这种默认的栈机制允许汇编程序员方便地使用栈来保存和恢复寄存器状态、传递参数和执行函数调用等操作。但需要注意的是，如果栈溢出或栈下溢，这可能会导致程序出现错误或崩溃，因此程序员需要小心管理栈的使用，确保不会超出栈的合法范围。

栈段的具体位置和大小通常由操作系统或编译器进行设置，而程序员不需要直接干预。在DOS环境下，栈通常会位于程序的数据段之上，且通常是以地址0xFFFF为起点向下增长的。不同的操作系统和环境可能会有不同的栈管理方式，但栈的基本原理是通用的。

遇到问题2：

将以下程序：

```
pop  dx
pop  ax
push ax
```

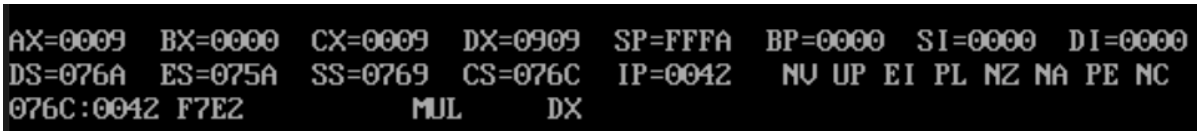
替换成以下程序：

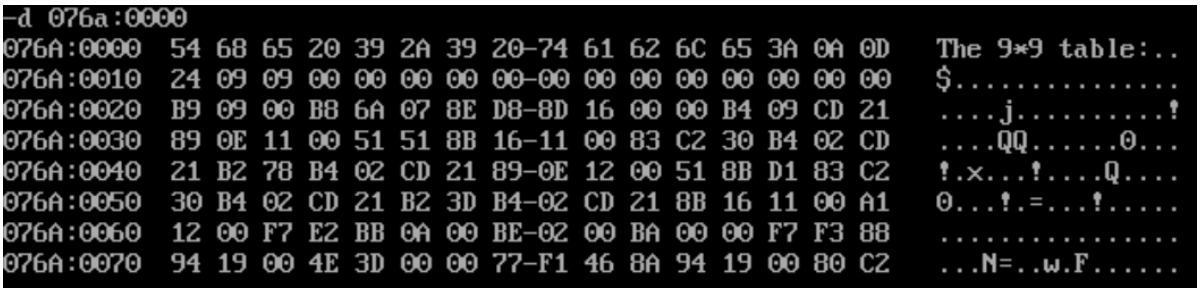
```
mov  dx, [num]
mov  ax, [num+1]
```

想法是直接用内存单元里面放的数据来相乘，但是结果不对，上面打印[num]和[num+1]的值显示结果正确，为什么在下面的指令中使用值就不对了呢？

心得体会2：

这个问题暂时无法解决。查看了内存，发现DX寄存器中的数值有问题





询问了人工智能，没有得到满意的答案

> 您的想法是使用`mov dx, [num]`和`mov ax, [num+1]`指令来直接从内存中加载数据到寄存器中，然后进行相乘。从您提供的代码看，这理论上是可行的，但有一些可能的问题和注意事项：
>
> 1. 数据的存储顺序：首先确保`[num]`和`[num+1]`中的数据存储在内存中的顺序是正确的。如果数据的存储顺序相反，那么`mov dx, [num]`和`mov ax, [num+1]`将加载错误的数据到寄存器中，导致结果不正确。
> 2. 数据类型：另一个潜在的问题是数据类型。如果`[num]`和`[num+1]`中的数据不是16位整数，而是其他数据类型（例如字节），那么将导致不正确的结果。确保数据类型匹配。
> 3. 前置条件：在执行`mov dx, [num]`和`mov ax, [num+1]`之前，确保内存中的数据已经正确初始化。如果数据没有正确初始化，那么加载到寄存器中的数据也将不正确。
>
> 如果所有这些条件都得到满足，那么直接从内存中加载数据进行相乘应该是可行的。如果结果仍然不正确，可能需要进一步检查代码和数据以查找问题的根本原因。可以使用调试工具来检查寄存器的内容和内存中的数据，以帮助排除问题。

目前的想法是在源程序的

```
;显示第二个乘数
mov  [num+1], cx;存放第二个乘数
```

第34行这一句代码处出现了问题。这里用默认的数据段基址，变址是数组首地址`num`，偏移量是1，于是指针只偏移了8位一个字节，导致产生了`09 09`这样的数据分布。但是在尝试过诸多方法后，比如把偏移量改成2，程序依然存在bug，实在不清楚原因。

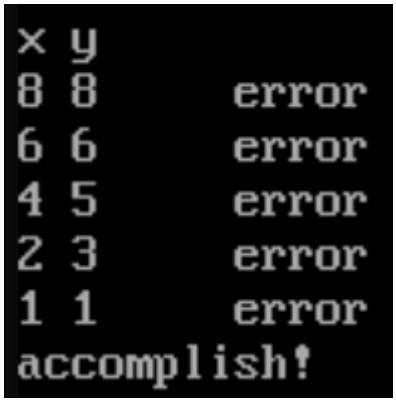
九九乘法表纠错

要求：检查9*9乘法表内数据是否正确，将不正确位置确定下来并显示在屏幕上。

如数据部分：

```
data segment
table   db 7,2,3,4,5,6,7,8,9            ;9*9表数据
        db 2,4,7,8,10,12,14,16,18
        db 3,6,9,12,15,18,21,24,27
        db 4,8,12,16,7,24,28,32,36
        db 5,10,15,20,25,30,35,40,45
        db 6,12,18,24,30,7,42,48,54
        db 7,14,21,28,35,42,49,56,63
        db 8,16,24,32,40,48,56,7,72
        db 9,18,27,36,45,54,63,72,81
......
data ends
```

结果输出：



分析：

首先，在数据段把提示语和相应的变量、数组存储好。

```
data segment
cnt     db 80
table   db 7,2,3,4,5,6,7,8,9        ;9*9表数据
        db 2,4,7,8,10,12,14,16,18
        db 3,6,9,12,15,18,21,24,27
        db 4,8,12,16,7,24,28,32,36
        db 5,10,15,20,25,30,35,40,45
        db 6,12,18,24,30,7,42,48,54
        db 7,14,21,28,35,42,49,56,63
        db 8,16,24,32,40,48,56,7,72
        db 9,18,27,36,45,54,63,72,81
msg     db "x y", 0ah, 0dh, '$'
err     db 09h, "error", 0ah, 0dh, '$'
acc     db "accomplish!", '$'
data ends
```

思路是：遍历数组，记录下当前位置行号和列号，行号和列号相乘得到九九乘积，乘积再和数组中的数值比较：相同，则继续下一个数；不相同，则打印行号、列号和错误信息。

```
loop1:  ;行循环
        ;列数每次初始化为9
loop2:  ;列循环
        ;计算两数相乘的结果，并比较
        ;取当前列数
```

```
        ;取当前行数
        ;行数、列数做乘法
        ;比较行、列乘积与表中数据
        ;相等跳转，不等打印
        ;打印行数
        ;显示空格
        ;打印列数
        ;打印“error”
here:   ;跳转到这里
        ;偏移量自减，下一个数
        loop loop2      ;列循环结束
        ;还原行数
        loop loop1      ;行循环结束
        ;打印“accomplish!”
```

完整代码：

```
data segment
cnt     db 80
table   db 7,2,3,4,5,6,7,8,9        ;9*9表数据
        db 2,4,7,8,10,12,14,16,18
        db 3,6,9,12,15,18,21,24,27
        db 4,8,12,16,7,24,28,32,36
        db 5,10,15,20,25,30,35,40,45
        db 6,12,18,24,30,7,42,48,54
        db 7,14,21,28,35,42,49,56,63
        db 8,16,24,32,40,48,56,7,72
        db 9,18,27,36,45,54,63,72,81
msg     db "x y", 0ah, 0dh, '$'
err     db 09h, "error", 0ah, 0dh, '$'
acc     db "accomplish!", '$'
data ends

code segment
        assume cs:code, ds:data
start:
        mov  cx, 9      ;行列数
        mov  ax, data   ;将数据段的地址存储在寄存器AX中
        mov  ds, ax     ;将数据段寄存器DS设置为数据段的地址，这样程序可以访问数据段中的数据
        lea  dx, msg    ;打印提示语
        mov  ah, 09h    ;将寄存器AH设置为09h，这是DOS的功能号，表示要执行字符串输出
        int  21h        ;触发DOS中断21h，以执行字符串输出操作，输出提示语
loop1:                  ;行循环
        push cx         ;乘数进栈
        mov  cx, 9      ;列数
loop2:                  ;列循环
        ;计算两数相乘的结果，并比较
        mov  di, cx     ;取当前列数
        pop  ax         ;取当前行数
        mov  bx, ax     ;换个寄存器
        push ax         ;当前行数再次进栈，在下次列循环中推出再次使用
        mul  cx         ;行数、列数做乘法
        mov  dl, cnt    ;存偏移量
        mov  si, dx     ;8位换16位
        cmp  al, [table+si];比较行、列乘积与表中数据
```

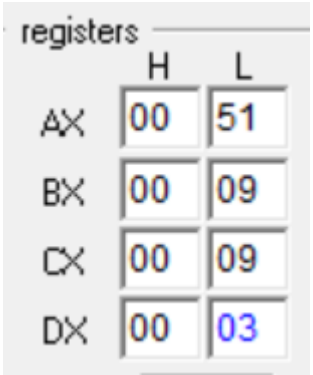
```
        je    here          ;相等跳转，不等打印
        ;打印行数
        add   bx, 30h       ;行数转化成ASCII码
        mov   dx, bx        ;放入DX寄存器
        mov   ah, 02h       ;将寄存器AH设置为2，表示要执行字符输出操作
        int   21h           ;触发DOS中断21h，以输出空格
        ;显示空格
        mov   dl, 20h       ;将DL寄存器设置为' '，表示要输出字符空格
        mov   ah, 02h       ;将寄存器AH设置为2，表示要执行字符输出操作
        int   21h           ;触发DOS中断21h，以输出空格
        ;打印列数
        add   di, 30h       ;列数转化成ASCII码
        mov   dx, di        ;放入DX寄存器
        mov   ah, 02h       ;将寄存器AH设置为2，表示要执行字符输出操作
        int   21h           ;触发DOS中断21h，以输出空格
        ;打印“error”
        lea   dx, err       ;打印“error”
        mov   ah, 09h       ;将寄存器AH设置为09h，这是DOS的功能号，表示要执行字符串输出
        int   21h           ;触发DOS中断21h，以执行字符串输出操作，输出提示语
here:
        dec   cnt           ;偏移量自减，下一个数
        loop loop2          ;列循环结束
        pop   cx            ;还原行数
        loop loop1          ;行循环结束
        ;打印“accomplish!”
        lea   dx, acc       ;打印“accomplish!”
        mov   ah, 09h       ;将寄存器AH设置为09h，这是DOS的功能号，表示要执行字符串输出
        int   21h           ;触发DOS中断21h，以执行字符串输出操作，输出提示语
        mov ah, 4ch
        int 21h
code ends
        end start
```

遇到问题：

数组的访问，卡了很久。一开始以为是像高级语言那样，对二维数组进行访问，如下：

```
table[bx][si]
```

但是这样根本无法寻到正确的数，程序会打印一堆乱七八糟的东西，或者停不下来打印乱码，或者干脆什么都不打印。后来，更换了汇编器，使用了`debug.exe`，发现找到的值是错的！



```
AX=0051  BX=0009  CX=0009  DX=0000  SP=FFFE  BP=0000  SI=0009  DI=0000
DS=076A  ES=075A  SS=0769  CS=0771  IP=001F   NV UP EI PL NZ NA PO NC
0771:001F 8A901100      MOV     DL,[BX+SI+0011]                 DS:0023=03
-t

AX=0051  BX=0009  CX=0009  DX=0003  SP=FFFE  BP=0000  SI=0009  DI=0000
DS=076A  ES=075A  SS=0769  CS=0771  IP=0023   NV UP EI PL NZ NA PO NC
0771:0023 8B160D00      MOV     DX,[000D]                       DS:000D=0009
```

仔细观察DX寄存器，发现是03H，此时，行号、列号都为9，理应为81才对，为什么是3呢?

心得体会:

查阅王爽老师《汇编语言（第四版）》第164页，找到了寻址方式的不同。

表 8.2　寻址方式小结

寻址方式	含　义	名　称	常用格式举例
[idata]	EA=idata;SA=(ds)	直接寻址	[idata]
[bx]	EA=(bx);SA=(ds)	寄存器间接寻址	[bx]
[si]	EA=(si);SA=(ds)		
[di]	EA=(di);SA=(ds)		
[bp]	EA=(bp);SA=(ss)		
[bx+idata]	EA=(bx)+idata;SA=(ds)	寄存器相对寻址	用于结构体： [bx].idata 用于数组： idata[si],idata[di] 用于二维数组： [bx][idata]
[si+idata]	EA=(si)+idata;SA=(ds)		
[di+idata]	EA=(di)+idata;SA=(ds)		
[bp+idata]	EA=(bp)+idata;SA=(ss)		
[bx+si]	EA=(bx)+(si);SA=(ds)	基址变址寻址	用于二维数组： [bx][si]
[bx+di]	EA=(bx)+(di);SA=(ds)		
[bp+si]	EA=(bp)+(si);SA=(ss)		
[bp+di]	EA=(bp)+(di);SA=(ss)		
[bx+si+idata]	EA=(bx)+(si)+idata; SA=(ds)	相对基址变址寻址	用于表格(结构)中的数组项： [bx].idata[si] 用于二维数组： idata[bx][si]
[bx+di+idata]	EA=(bx)+(di)+idata; SA=(ds)		
[bp+si+idata]	EA=(bp)+(si)+idata; SA=(ss)		
[bp+di+idata]	EA=(bp)+(di)+idata; SA=(ss)		

此处，若是按照

```
table[bx][si]
```

寻址，则会变成

```
EA=(bx)+(si)+table
```

相当于9+9=18，寻到了第三行第一列的数字，当然是3.

结合课上讲的，终于有了深刻的领悟。