

第三次作业

目录

- 第三次作业
 - 目录
 - 第一题
 - 题目
 - 思路
 - 结果
 - 心得
 - 问题
 - 解决
 - 第二题
 - 题目
 - 思路
 - 结果
 - 心得
 - 问题
 - 解决

第一题

题目

功能：接收年、月、日信息并进行显示。

目的：掌握响铃符方法；掌握年、月、日的输入方法；掌握过程调用方法。

内容：先显示提示信息“WHATISTHEDATE(MM/DD/YY)?”并响铃一次然后接收键盘输入的月/日/年信息，并显示。

思路

首先，声明一个栈段，用来存储进制转换的结果；声明一个数据段，用来存储年月日数据和输出提示语。响铃在这里是 07H，作为字符串的一部分输出。

调用三次 GetNum 过程，分别读取月、日、年的数值。

调用三次 Disp 过程，分别显示年、月、日数值。

以下是部分程序代码：

```
stk segment

stk ends

data segment
num      dw  ?,?,?
msg      db  "WHAT IS THE DATE?(MM DD YYYY)",07h, 0ah, 0dh, '$';响铃07h
data ends
```

开头这一部分声明了栈段和数据段，其中数据段中声明了有3个字形大小的数组，提示语也在这里，其中响铃符用ASCII码07h表示。

```
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
mov ax, stk     ;将栈段的地址存储在寄存器AX中
mov ss, ax      ;将栈段寄存器SS设置为栈段的地址，这样程序可以访问栈段中的数据

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

mov si, 0       ;访问数组第0个元素
call GetNum     ;调用GetNum，接收键入的月值
mov si, 2       ;访问数组第1个元素
call GetNum     ;调用GetNum，接收键入的日值
mov si, 4       ;访问数组第2个元素
call GetNum     ;调用GetNum，接收键入的年值
mov si, 4       ;访问数组第2个元素
call Disp      ;调用Disp显示年值
mov dx, "-"     ;输出短横线
mov ah, 02h     ;将寄存器AH设置为02h，这是DOS的功能号，表示要执行字符输出
int 21h         ;触发DOS中断21h，以执行字符输出操作，输出短横线
mov si, 0       ;访问数组第0个元素
call Disp      ;调用Disp显示月值
mov dx, "-"     ;输出短横线
mov ah, 02h     ;将寄存器AH设置为02h，这是DOS的功能号，表示要执行字符输出
int 21h         ;触发DOS中断21h，以执行字符输出操作，输出短横线
mov si, 2       ;访问数组第1个元素
call Disp      ;调用Disp显示日值

mov ah, 4ch     ;功能号4CH:结束正在运行的程序，并返回DOS操作系统。
int 21h         ;中断
```

主程序执行的过程，首先，是地址初始化；其次，打印提示语和响铃符；然后，分三次调用 GetNum 过程，将月、日、年分别存放到 Num 数组三个不同位置；最后，分三次调用 Disp 过程，分别取出 Num 数组三个位置的 年、月、日数据，显示到屏幕上。

```
GetNum:         ;读取数字过程
init:          ;初始化
    xor ax, ax  ;清零
    xor bx, bx  ;清零
    xor cx, cx  ;清零
    xor dx, dx  ;清零
    mov ah, 01h ;读一个字符方式
    int 21h     ;中断
input:         ;循环输入开始
    cmp al, 30h ;判断'0'
    jb check    ;比0小，跳转
    cmp al, 39h ;判断'9'
    ja check    ;比9大，跳转
    sub al, 30h ;ASCII处理
    shl bx, 1   ;bx左移
    mov cx, bx  ;赋值
```

```

shl bx, 1      ;bx左移
shl bx, 1      ;bx左移
add bx, cx     ;加和
add bl, al     ;加和
mov ah, 01h    ;读一个字符方式
int 21h        ;中断
jmp input      ;循环输入结束
check:         ;检查输入
cmp al, 0dh    ;输入回车
je save        ;跳转保存结果
cmp al, 20h    ;输入空格
je save        ;跳转保存结果
cmp al, "/"    ;输入/
je save        ;跳转保存结果
jmp init       ;输入错误，跳转初始化
save:          ;保存结果
mov ax, bx     ;ax是结果
mov num[si], ax;存入数据段内存
ret            ;用栈中的数据，修改IP的内容，从而实现近转移

```

GetNum 过程负责读取缓冲区中的内容，并且将读取到的值经过ASCII码处理，存放到 Num 数组相应位置。

```

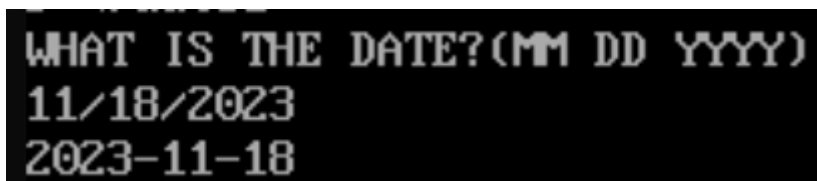
Disp:          ;显示数值过程
xor cx, cx     ;cx清空
mov bx, num[si];num放入bx
mov ax, bx     ;赋值
mov bx, 10     ;十进制
divide:        ;进制转换
xor dx, dx     ;清空
div bx         ;除以10取余数，dx:ax / bx = ax.....dx
add dl, 30h    ;ASCII转换
push dx        ;商入栈
inc cx         ;数字位数+1
cmp ax, 0      ;商为0，结束
jne divide     ;商不为0，继续除
output:        ;循环输出开始
pop dx         ;依次弹出结果
mov ah, 2      ;输出方式
int 21h        ;中断
loop output    ;循环输出结束
ret            ;用栈中的数据，修改IP的内容，从而实现近转移

```

Disp 过程负责读取 Num 数组相应位置的值，并进制转换和ASCII码处理后，显示到屏幕上。

结果

程序运行正确，结果正常，如图：

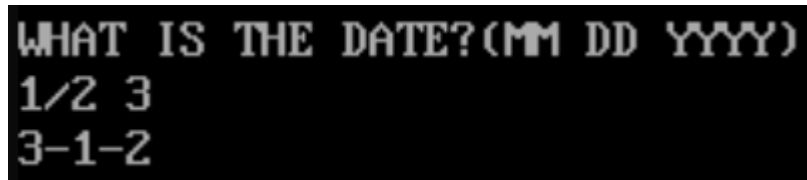


```

WHAT IS THE DATE?(MM DD YYYY)
11/18/2023
2023-11-18

```

另外一些格式也可以，比如：



心得

经过这次作业，熟悉了数组中的元素寻址，同时解决了上次悬而未决的问题，通过指针+2的方式，对字型的数据进行下一个寻址。也对过程调用有了初步的认识。

问题

在过程结束时，通常会写上 `ret` 指令，来结束一段过程调用，并用栈中的数据，修改IP的内容，从而实现近转移。既然是用栈操作的，那就有可能被过程调用中的 `push`、`pop` 等指令干扰。比如，在过程调用中 `push` 了一个数字1，那么栈顶元素就会变成1，执行到 `ret` 时就会把栈顶元素弹出，赋值给IP，跳到错误的地方，而不是上次 `call` 的地方。

解决

避免使用栈存放数据，防止与地址产生冲突。或者在 `ret` 执行前，保证 `push` 和 `pop` 的数量一致，确保栈顶是上次 `call` 的地址。

第二题

题目

汇编语言（第4版）（王爽）P160 实验7

思路

按照行顺序，处理每一行。使用 `bp`、`si` 和 `di` 三个寄存器寻址，将 `DATA` 数据段中的数据搬运到 `TABLE` 数据段中，并且计算人均收入。按照格式，处理内存数据即可。

以下是部分程序代码：

```
mov    cx, 21                ;循环次数
mov    bx, data              ;将数据段DATA的地址存储在寄存器BX中
mov    ds, bx                ;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
mov    bx, table             ;将数据段TABLE的地址存储在寄存器BX中
mov    es, bx                ;将数据段TABLE的地址存储在段地址寄存器ES中
xor    bx, bx                ;清零
xor    di, di                ;清零
```

开头做一些初始化的工作，把 `DATA` 数据段地址赋给数据段地址寄存器。使用段地址寄存器，寻址 `TABLE` 数据段的数据，并清空一些寄存器。

```

mov    cx, 21                                ;循环次数
row:                                       ;每一行循环开始
;-----0123-----;处理第0123字节
;-----4-----;处理第4字节
;-----5678-----;处理第5678字节
;-----9-----;处理第9字节
;-----AB-----;处理第AB字节
;-----C-----;处理第12字节
;-----DE-----;处理第DE字节
;-----F-----;处理第15字节
loop row                                   ;每一行循环结束

```

循环所有行数，分别处理年份、总收入、雇员人数和人均收入。

```

mov    bp, cx                                ;当前次数赋值栈基址指针寄存器
sub    bp, 21                                ;做差，结果<=0
neg    bp                                    ;取反，得一行中第几个索引
push   cx                                    ;入栈，防止过程中改变数值
mov    si, 0                                ;三种数据起始地址

```

开始前，先计算是第几个数据的索引，并且初始化三种数据的起始地址。

```

;-----0123-----;处理第0123字节
    shl    bp, 1                                ;*2
    shl    bp, 1                                ;*2
    mov    cx, 4                                ;年份的4个字节
year:                                       ;年份字节循环开始
    mov    dx, [si+bp]                          ;从DATA取到字符
    mov    es:[di], dx                          ;存入TABLE对应位置
    inc    si                                    ;DATA下一位
    inc    di                                    ;TABLE下一位
    loop   year                                ;年份字节循环结束

```

针对前四个字节处理，循环遍历存储年份的四个字节。

```

;-----4-----;处理第4字节
    mov    dl, " "                                ;存入空格
    mov    byte ptr es:[di], dl                  ;写入一个字节

```

后面的空格都是这么处理。由于空格是字节型数据，需要改变数据指针类型为字节。

```

;-----5678-----;处理第5678字节
    add    si, 80                                ;挪到总收入处
    inc    di                                    ;TABLE下一位
    mov    dx, word ptr [si+bp]                  ;从DATA取到低位字型数据
    mov    word ptr es:[di], dx                  ;存入TABLE对应位置
    add    si, 2                                ;挪到高位
    add    di, 2                                ;TABLE对应高位
    mov    dx, word ptr [si+bp]                  ;从DATA取到高位字型数据
    mov    word ptr es:[di], dx                  ;存入TABLE对应位置

```

这是处理双字型数据“总收入”的程序。由于普通寄存器放不下4字节的双字型数据，故需要按照高16位和低16位对双字进行拆分。程序中，指定了一个字型数据指针，为的是分别存储高低16位。按照8086的顺序，先存低16位，再存高16位。

```
;-----AB-----;处理第AB字节
shr bp, 1 ;/2
add si, 82 ;挪到雇员人数处
inc di ;TABLE下一位
mov dx, word ptr [si+bp] ;从DATA取到字型数据
mov word ptr es:[di], dx ;存入TABLE对应位置
```

这是处理字型数据“雇员数”的程序。大致思路与“总收入”一致，不过不用分别存了。

```
;-----DE-----;处理第DE字节
shl bp, 1 ;*2
mov ax, [si-84+bp] ;寄存器AX中存放被除数的低16位
mov dx, [si-82+bp] ;寄存器DX中存放被除数的高16位
shr bp, 1 ;/2
div word ptr [si+bp] ;除数是16位，做双字除法，结果在寄存器AX中
mov es:[di], ax ;写入人均收入
```

这是处理字型数据“人均收入”的程序。这部分需要用双字除以字，参考网络上的解释，如果除数是8位字节型数据，那么被除数做16位除法；如果除数是16位字型数据，那么被除数做32位双字除法，其中寄存器AX中存放被除数的低16位，寄存器DX中存放被除数的高16位，最终结果存放在寄存器AX中，余数存放在寄存器DX中。同样要注意的是，在寻址过程中，偏移量是4字节移动，还是2字节移动。

```
pop cx ;恢复循环次数
cmp cx, 1 ;比较是否是最后一行
je finish ;是，结束
inc di ;不是，进入下一行
loop row ;每一行循环结束
finish: ;结束位置
mov ah, 4ch ;功能号4CH:结束正在运行的程序，并返回DOS操作系
统。
int 21h ;中断
```

程序结束部分，防止越界，做了特判。在这部分恢复循环次数的值。

结果

该程序并没有输出的结果，但是通过反汇编，可以看到在TABLE数据段，内存确实变化了，如图：

Random Access Memory

071e:00

update

☒ table

☐ list

071E:0000	31	39	37	35	20	10	00	00-00	20	03	00	20	05	00	20	1975	#...	U.	U.
071E:0010	31	39	37	36	20	16	00	00-00	20	07	00	20	03	00	20	1976	W...
071E:0020	31	39	37	37	20	7E	01	00-00	20	09	00	20	2A	00	20	1977
071E:0030	31	39	37	38	20	4C	05	00-00	20	0D	00	20	68	00	20	1978	L...	..	h.
071E:0040	31	39	37	39	20	56	09	00-00	20	1C	00	20	55	00	20	1979	V...	↑	U.
071E:0050	31	39	38	30	20	40	1F	00-00	20	26	00	20	D2	00	20	1980	g...	g.	?
071E:0060	31	39	38	31	20	80	3E	00-00	20	82	00	20	7B	00	20	1981	■>...	?	f.
071E:0070	31	39	38	32	20	A6	5F	00-00	20	DC	00	20	6F	00	20	1982	?...	?	o.

Random Access Memory

071e:0080

update

☒ table

☐ list

071E:0080	31	39	38	33	20	91	C3	00-00	20	DC	01	20	69	00	20	1983	??..	??	i.
071E:0090	31	39	38	34	20	C7	7C	01-00	20	0A	03	20	7D	00	20	1984	??	??	?
071E:00A0	31	39	38	35	20	81	24	02-00	20	E9	03	20	8C	00	20	1985	??	??	?
071E:00B0	31	39	38	36	20	8A	03	03-00	20	A2	05	20	88	00	20	1986	??	??	?
071E:00C0	31	39	38	37	20	7C	47	05-00	20	D2	08	20	99	00	20	1987	??	??	?
071E:00D0	31	39	38	38	20	EB	03	09-00	20	E9	0A	20	D3	00	20	1988	??	??	?
071E:00E0	31	39	38	39	20	CA	42	0C-00	20	C5	0F	20	C7	00	20	1989	??	??	?
071E:00F0	31	39	39	30	20	18	0D	12-00	20	03	16	20	D1	00	20	1990	??	??	?

Random Access Memory

071e:0100

update

☒ table

☐ list

071E:0100	31	39	39	31	20	38	1F	1C-00	20	22	20	20	E0	00	20	1991	8*↑.	"	?
071E:0110	31	39	39	32	20	58	19	2A-00	20	16	20	20	EF	00	20	1992	X *.	u-	?
071E:0120	31	39	39	33	20	28	44	39-00	20	5E	38	20	04	01	20	1993	(D9.	8	u
071E:0130	31	39	39	34	20	28	F0	46-00	20	C9	B0	20	66	00	20	1994	(?F.	??	f.
071E:0140	31	39	39	35	20	68	97	5A-00	20	88	45	20	4D	01	20	1995	h?Z.	?E	u
071E:0150	B9	15	00	BB	10	07	8E	DB-BB	1E	07	8E	C3	BB	00	00	??	??	???	???
071E:0160	8B	E9	83	ED	15	F7	DD	51-BE	00	00	D1	E5	D1	E5	B9	???	???	???	???
071E:0170	04	00	88	12	26	89	11	46-47	E2	F7	B2	20	26	88	11	u.	?#&?	FG???	&?

心得

通过这个作业，加深了对数据段寻址的认识，加深了对各种长度数据类型的处理方法认识，也增强了编程过程中的一些小技巧，比如反汇编查看内存，就比之前更加熟练。

问题

内存中的数据，无法准确定位到特定的数据段，从而查看内存情况。还有就是，对不同长度的数据，尤其是双字的特殊处理，很麻烦。

解决

首先说定位内存数据，后来我发现 ES 段地址寄存器的值指向的区域就是 TABLE 数据段，可以手动调节地址到特定区域查看。

其次是双字处理的问题，上网搜索，发现其实很简单，只要拆成两个字型数据，按照顺序放到对应的寄存器里就好。