«به نام خدا»

نام: فاطمه زهرا بخشنده استاد: دکتر میثم عبداللهی

شماره دانشجویی: 98522157

گزارش تمرین پنجم:

توضیحات:

فایل مربوط به هر سوال تمرین، در این فایل زیپ قرار داده شده است. هر سه سوال با emu8086 نوشته شده اند.

توضیحات الگوریتم هر سوال:

سوال اول:

در این سوال از وقفه های مربوط به موس استفاده می کنیم. برنامه دارای یک LOP اصلی است که تا فشرده شدن کلید ESC کیبورد ادامه دارد. به محض فشرده شدن کلید ESC برنامه متوقف میشود.

در ابتدا GRAPHIC MODE را تنظیم میکنیم. در هر مرحله در حلقه با استفاده از وقفه 33، موقعیت موس را دریافت میکنیم، سپس ابتدا موقعیت جدید موس را روی صفحه چاپ می کنیم.

```
CHECK:   mov ax, 3
         int 33h

         cmp cx, curX
         jne PRINTP
         cmp dx, curY
         jne PRINTP

CON:     shr cx, 1
         cmp bx, 1
         jne CHANGE:
         JE SET_CLR
CO:      mov al, COLOR          ;color
         jmp DRAW


PRINTP:  print 0,0,0000_1111b,"x="
         mov ax, cx
         call print_ax
         print_space 4
         print 0,1,0000_1111b,"y="
         mov ax, dx
         call print_ax
```

```
print_ax proc
cmp ax, 0
jne print_ax_r
    push ax
    mov al, '0'
    mov ah, 0eh
    int 10h
    pop ax
    ret
print_ax_r:
    pusha
    mov dx, 0
    cmp ax, 0
    je pn_done
    mov bx, 10
    div bx
    call print_ax_r
    mov ax, dx
    add al, 30h
    mov ah, 0eh
    int 10h
    jmp pn_done
pn_done:
    popa
    ret
endp
```

```
print macro x, y, attrib, sdat
LOCAL   s_dcl, skip_dcl, s_dcl_end
    pusha
    mov dx, cs
    mov es, dx
    mov ah, 13h
    mov al, 1
    mov bh, 0
    mov bl, attrib
    mov cx, offset s_dcl_end - offset s_dcl
    mov dl, x
    mov dh, y
    mov bp, offset s_dcl
    int 10h
    popa
    jmp skip_dcl
    s_dcl DB sdat
    s_dcl_end DB 0
    skip_dcl:
endm

print_space macro num
    pusha
    mov ah, 9
    mov al, ' '
    mov bl, 0000_1111b
    mov cx, num
    int 10h
    popa
endm
```

و سپس چک میکنیم آیا کلیکی انجام شده یا نه...در صورتی که انجام شده باشد، رنگ پیکسل را تنظیم می کنیم. اگر رنگ قبلی قرمز بود رنگ جدید را آبی، اگر رنگ قبلی آبی بود رنگ جدید را سبز، و اگر رنگ قبلی سبز بود رنگ جدید را قرمز می گذاریم...به این ترتیب طبق خواسته سوال نقطه هایی که رسم می شوند به ترتیب قرمز، آبی و سبز خواهند بود.

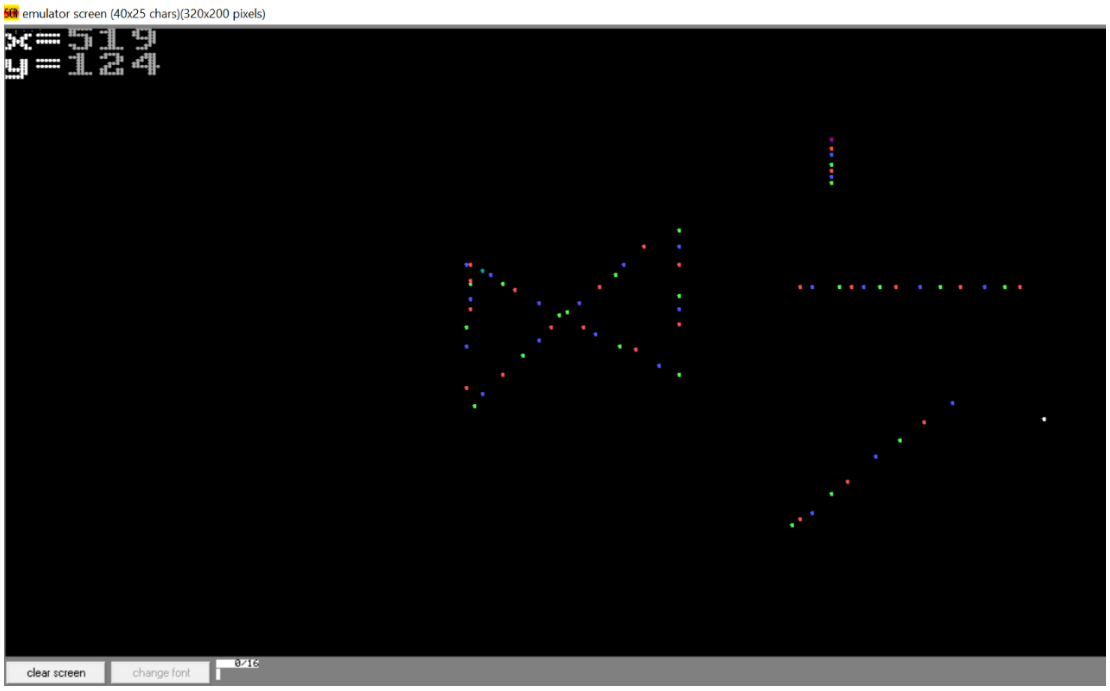
اگر کلیک انجام نشده باشد نباید پیکسل دارای رنگ شود.

```
CHANGE: cmp oldX, -1
        je SET
        push cx
        push dx
        mov cx, oldX
        mov dx, oldY
        mov ah, 0dh
        int 10h
        xor al, 1111b
        mov ah, 0ch
        int 10h
        pop dx
        pop cx

SET:    mov ah, 0DH
        int 10h
        xor al, 1111B
        mov oldX, cx
        mov oldY, dx

DRAW:   mov ah, 0CH
        int 10H
```

```
SET_CLR:CMP COLOR, 0CH
        JNE NOTRED
        MOV COLOR, 09H
        JMP CO

NOTRED: CMP COLOR, 09H
        JNE NOTBLUE
        MOV COLOR, 0AH
        JMP CO

NOTBLUE:MOV COLOR, 0CH
        JMP CO


CHANGE: cmp oldX, -1
        je SET
        push cx
        push dx
        mov cx, oldX
        mov dx, oldY
```

با اجرا کردن برنامه، میبینیم که با تکان دادن موس روی پنجره گرافیکی، موقعیت آن که در بالای صفحه نوشته شده است، آپدیت میشود. همچنین با کلیک کردن میتوانیم نقطه های رنگی بکشیم.

چون در این برنامه هر لحظه موقعیت موس را چک کرده و روی صفحه چاپ می کنیم، کشیدن نقطه ها کمی کندتر انجام خواهد شد و باید کمی موس را بیشتر روی صفحه نگه داشت تا نقطه بکشد. پس هنگام کلیک کردن، با موس سریع حرکت نکنید😊



با فشردن دکمه CLEAR SCREEEN صفحه پاک میشود.

سوال دوم:

در این سوال، ابتدا DATA های مورد نیاز را تعریف میکنیم. باید دو دیتا برای نگه داری شماره میله های مبدا و مقصد داشته باشیم.

سپس یک MACRO مینویسیم که تعداد دیسک هارا دریافت کرده، مقادیر مورد نیاز را به STACK پوش میکند و تابع بازگشتی هانوی را صدا میزند.

```
;------------------------
.DATA
    from DB "mov from "
    F    DB ?
    to   DB " to "
    T    DB ?
    line DB 0AH, 0DH, '$'
    c    DB 0
;------------------------
.CODE
Tower MACRO num
        MOV AX, 1
        PUSH AX
        MOV AX, 3
        PUSH AX
        MOV AX, 2
        PUSH AX
        MOV AX, num
        PUSH AX
        CALL hanoi
      ENDM
```

تابع بازگشتی هم الگوریتم برج هانوی را اجرا میکند.

```
hanoi proc
        push bp
        mov bp, sp
        cmp word ptr ss:[bp+4], 0
        je down

        push word ptr ss:[bp+0AH]
        push word ptr ss:[bp+6]
        push word ptr ss:[bp+8]
        mov ax, word ptr ss:[bp+4]
        dec ax
        push ax
        call hanoi

        push word ptr ss:[bp+0AH]
        push word ptr ss:[bp+08]
        call print

        push word ptr ss:[bp+06H]
        push word ptr ss:[bp+8]
        push word ptr ss:[bp+0AH]
        mov ax, word ptr ss:[bp+4]
        dec ax
        push ax
        call hanoi

        pop bp
        ret 8
    down:
        pop bp
        ret 8

hanoi endp
```

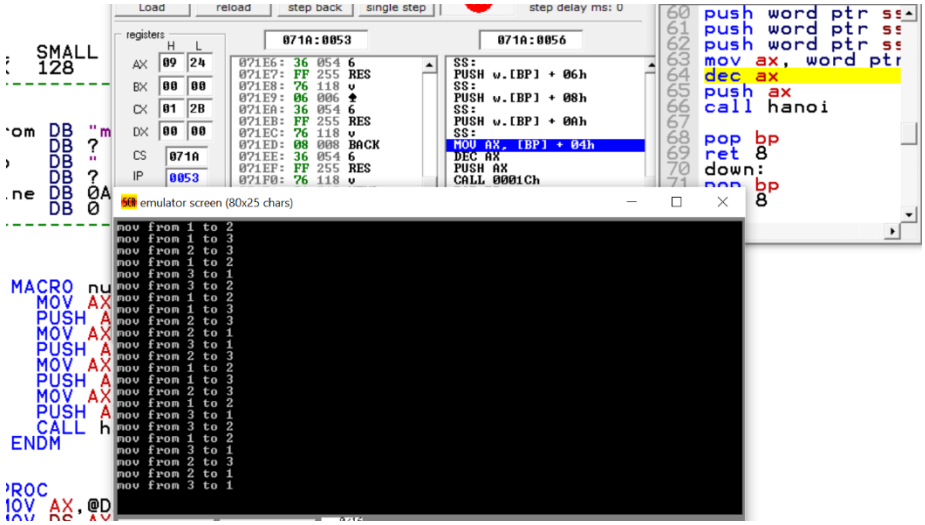
در هر مرحله تابع PRINT را صدا زده و جزئیات مربوط به آن مرحله چاپ میشود:

```
print proc
        push bp
        mov bp, sp
        mov F, '0'
        mov al, byte ptr ss:[bp+06]
        add F, al
        mov T, '0'
        mov al, byte ptr ss:[bp+4]
        add T, al
        lea dx, from
        mov ah, 09
        int 21h
        pop bp
        ret 4

print endp
end
```

برای اجرا شدن الگوریتم برای ده دیسک، در قسمت MAIN برنامه MACRO مان را با عدد 10 فراخوانی میکنیم:

```
MAIN PROC           ;program entry point
     MOV AX,@DATA   ;load the data segment address
     MOV DS,AX      ;assign value to DS

     Tower 10D      ;macro

EXIT: MOV AH,4CH    ;set up to
      INT 21H       ;return to the OS
MAIN  ENDP
```

با اجرای برنامه، میتوانیم مراحل حل را مشاهده کنیم:



سوال سوم:

در این سوال، باید از چندین عملیات 16 بیتی برای انجام عملیات 48 بیتی استفاده کنیم. با نوشتن فرمول ضرب و تقسیم، میتوان فهمید که الگوریتم آن به چه صورت است.

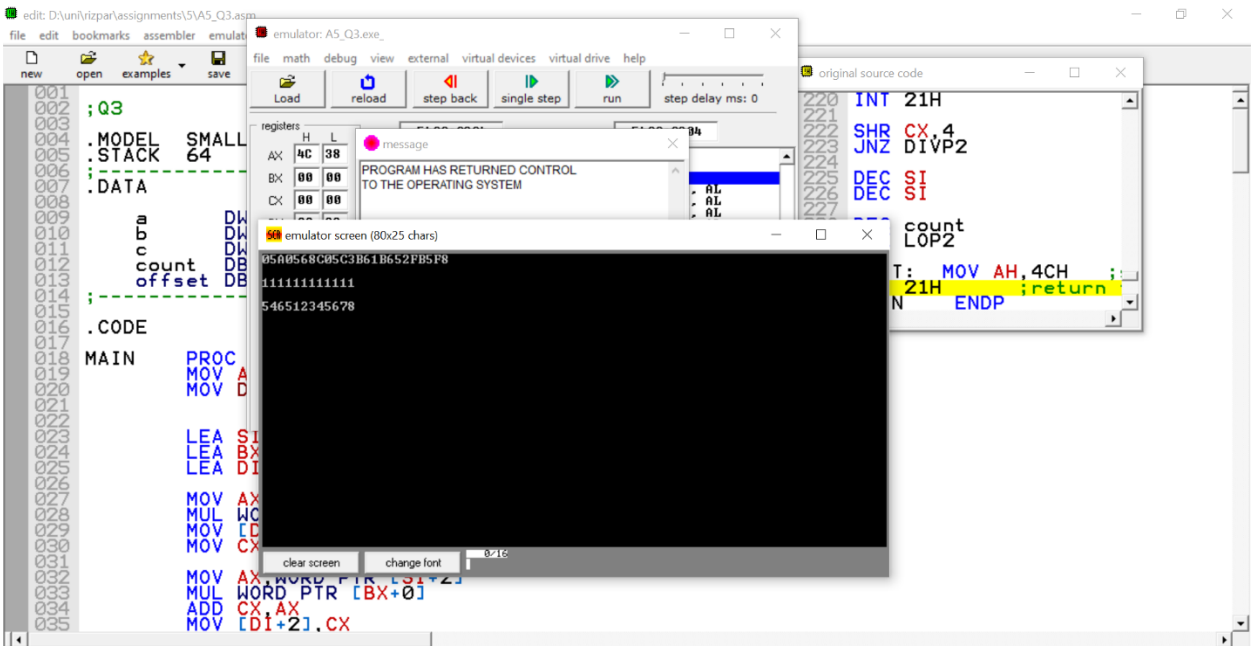
ابتدا دو عدد 48 بیتی را به صورت سه بخش 16 بیتی تعریف میکنیم.

```
.MODEL  SMALL
.STACK  64
;-------------------------------------------------
.DATA

    a       DW 5678h, 1234h, 5465h, 5 dup(0)  ;a = 5465 1234 5678
    b       DW 1111h, 1111h, 1111h, 5 dup(0)  ;b = 1111 1111 1111
    c       DW 6 dup(?)
    count   DB 6
    offset  DB 10
;-------------------------------------------------

.CODE

MAIN    PROC FAR        ;program entry point
        MOV AX,@DATA    ;load the data segment ADDress
        MOV DS,AX       ;asSIgn value to DS
```

```
MOV AX,WORD PTR [SI+4]
MUL WORD PTR [BX+2]
;ADD CX,AX
ADD WORD PTR [DI+6],AX
adc CX,DX

MOV WORD PTR [DI+8],CX

MOV CX,0

MOV AX,WORD PTR [SI]
MUL WORD PTR [BX+4]
ADD WORD PTR [DI+4],AX
adc CX,DX

ADD [DI+6],CX

MOV CX,0

MOV AX,WORD PTR [SI+2]
MUL WORD PTR [BX+4]
ADD WORD PTR [DI+6],AX
adc CX,DX

ADD WORD PTR [DI+8],CX
```

```
LEA SI,a
LEA BX,b
LEA DI,c

MOV AX,WORD PTR [SI]
MUL WORD PTR [BX+0]
MOV [DI],AX
MOV CX,DX

MOV AX,WORD PTR [SI+2]
MUL WORD PTR [BX+0]
ADD CX,AX
MOV [DI+2],CX

MOV CX,DX

MOV AX,WORD PTR [SI+4]
MUL WORD PTR [BX+0]
ADD CX,AX
MOV [DI+4],CX

MOV [DI+6],DX
MOV CX,0
```

الگوریتم انجام ضرب آن ها کمی طولانی هست اما مشابه نوشتن این اعداد زیر هم، و ضرب آن هاست. هر بخش 16 بیتی از عدد پایینی، هر دفعه ضربدر یک بخش 16 بیتی از عدد بالایی شده، و CARRY آن به خانه بالاتر میرود، به این ترتیب با ضرب اعداد 48 بیتی که تعریف کردیم، پس از تلاش بسیار زیاد:)) به نتیجه زیر میرسیم و آن را چاپ میکنیم.



اگر تقسیم انجام دهیم اعداد اولیه حاصل میشوند.