امیرحسین احمدی ۹۷۵۲۲۲۹۲ – تمرین پنجم

۱- برای این سوال ابتدا نیاز است که برای استفاده از رنگ ها به ویدیو مود برویم.

```
; Video Mode
MOV     AH, 00H
MOV     AL, 13H
INT     10H
```

سپس برای استفاده از موس نیاز است تنظیمات زیر را انجام دهیم.

```
MOV     AX, 0       ; Star Mouse
INT     33H



MOV     AX, 1H      ; Display Mouse Cursor
INT     33H
```

سپس در یک while هر دفعه چک میکنیم که آیا از صفحه کلید دکمه ای (هر دکمه ای) زده شده است، در این صورت الگوریتم پایان میابد و صفحه نمایش را پاک میکنیم.

```
; Check Left Button State
MOV     AX, BX
AND     AX, 0000000000000001B
CMP     AX, 1B      ; If Bit 0 == 0 : No Left Button


JNE     CHECK_KEY
```

```
        MOV     AH, 0BH
        INT     21H
        CMP     AL, 0       ; AL==0 : No Key.
        JZ      WHILE

        MOV     AX, 0600H   ; Reset Screen.
        MOV     BH, 07
        MOV     CX, 0
        MOV     DX, 184FH
        INT     10H
```

همچنین همیشه در ابتدای CHECK_KEY مختصات خانه ای که در حال حاضر در آن هستیم را چاپ میکنیم.

```
CHECK_KEY:
    ;Check If Reset Key Was Pressed : Works With Any Key

        MOV     X, CX
        MOV     Y, DX
        MOV     AX, CX
        CALL    PRINT

        MOV     AH, 09H
        MOV     DX, OFFSET STRINGVIR
        INT     21H

        MOV     AX, Y
        CALL    PRINT
        CALL    NEWLINEPOINTER
```

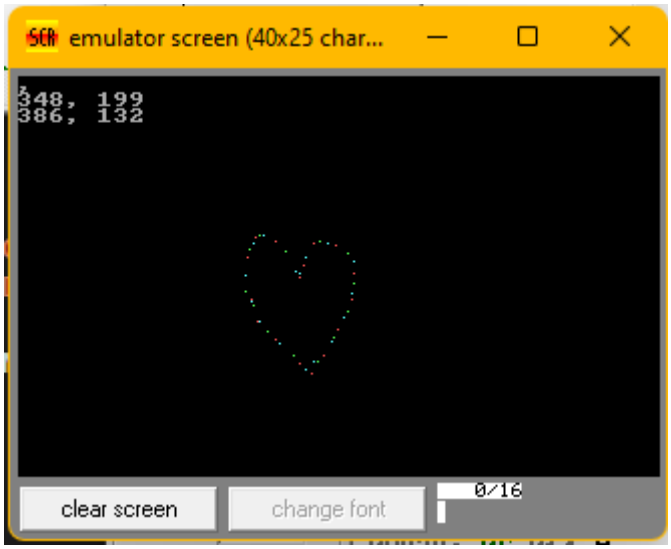
در ادامه در هر مرحله محله موس را دریافت کرده و نگه داشته تا در ادامه آن را رنگ کنیم.

```
; CX Is Doubled So We / 2
MOV     TEMP_Y, DX
MOV     TEMP_X, CX

MOV     AX, CX
MOV     BL, 2
DIV     BL

MOV     CX, AX
MOV     DX, TEMP_Y
MOV     AH, 02h     ; Set Cursor Position
INT     10h         ; Actually Does It
```

برای رنگ کردن نیز ابتدا کد رنگ قرمز را در color قرار داده و در هر مرحله با رنگ کردن یک نقطه، رنگ ها را تغییر میدهیم.

```
        ; Change Color
        MOV     AH, 0CH
        MOV     AL, COLOR

        CMP     AL, 0CH
        JE      RED

        CMP     AL, 0BH
        JE      BLUE

        CMP     AL, 0AH
        JE      GREEN


RED:
        MOV     COLOR, 0BH
        INT     10H
        JMP     WHILE

BLUE:
        MOV     COLOR, 0AH
        INT     10H
        JMP     WHILE

GREEN:
        MOV     COLOR, 0CH
        INT     10H
        JMP     WHILE
```

خروجی کد را نیز میتوان در زیر دید.



۲- برای سوال دو ابتدا متغییر های src, des, mid و size را تعریم میکنیم که به ترتیب برابر میله ی مبدا، میله ی مقصر، میله ی وسط و تعداد دیسک ها هستند.

```
src     db  1
des     db  3
mid     db  2
size    db  10
```

سپس متغیر های بالا را درون bl, bh, cl, ch میریزیم که قرار است در هر مرحله از الگریتم استیت مارا نگه دارند.

```
; state
mov     bl, size
mov     bh, src
mov     cl, des
mov     ch, mid
```

حال تابع بازگشتی hanoi را صدا میزنیم. در این تابع ابتدا چک میکنیم که اگر تعداد دیسک ها برابر ۱ بود، همان دیسک را از میله ی مبدا به مقصد چاپ کند و الگوریتم پایان بپزیرد.

```
cmp     bl, 1
jne     recursive

print   bh, cl

ret
```

در غیر این صورت به این صورت باید عمل کنیم. ابتدا باید ۹ دیسک اول را به میله ی وسط برده و سپس دیسک آخر را به میله ی مقصد ببریم و در ادامه ۹ دیسک را به میله ی مقصد ببریم. حال برای پیاده سازی ابتدا باید تعداد میله ها را یکی کم کنیم، همچنین جای میله ی وسط و مقصد را عوض کنیم تا بتوان با صدا زدن دوباره تابع hanoi، ۹ دیسک اول را به میله وسط برد.

```
dec     bl
mov     al, cl
mov     cl, ch
mov     ch, al
call    hanoi
```

سپس میله آخرین دیسک میله مبدا را به میله مقصد میبریم.

```
print   bh, cl
```

و بعد دوباره تعداد دیسک ها را ۹ در نظر گرفته و این بار جای میله مبدا را با میله وسط عوض میکنیم تا بتوانیم ۹ دیسک را از وسط به مقصد ببریم و الگوریتم ما تمام است.

```
dec     bl
mov     al, bh
mov     bh, ch
mov     ch, al
call    hanoi
```

به این نکته توجه داشته باشید که از آنجایی که ممکن است در صدا زدن توابع متغییر ها تغییر کنند، هر بار قبل از صدا کردن تابعی متغییر هایی که استیت مارا نگه میداشتند را درون استک پوش کرده و بعد از انجام آن پاپ میکنیم.

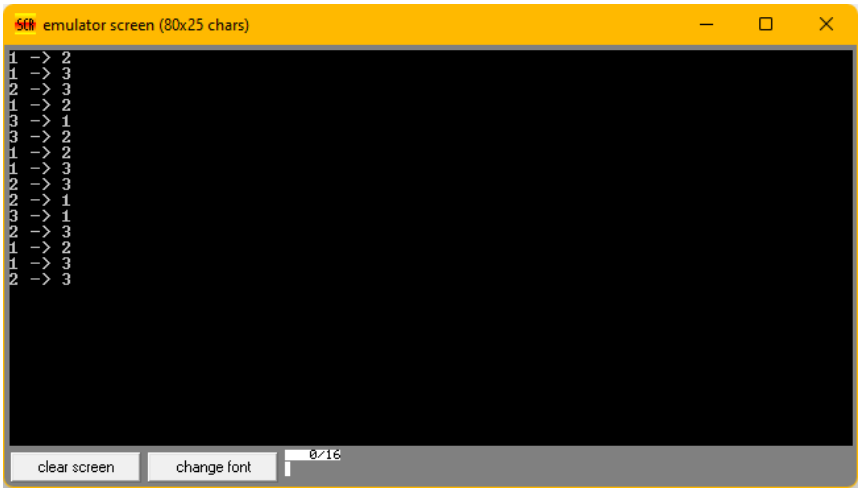
```
push    bx
push    cx
```

```
pop     cx
pop     bx
```

همچنین تابع print با استفاده از ماکرو پیاده سازی شده است به این صورت که میله ی مبدا و مقصد را برای جا به جایی یک دیس میگیرد و جا به جایی آن را با استفاده از یک استرینگ چاپ میکند.

```
print   macro   s, d

        mov     from, '0'
        add     from, s
        mov     to, '0'
        add     to, d

        lea     dx, from
        mov     ah, 09
        int     21h

        endm
```

خروجی سوال دو برای ۱۰ دیسک بسیار طولانی است، در زیر خروجی را برای ۴ دیسک مشاهده میکنید.



۳- برای سوال ۳، دو عدد ۴۸ بیتی خود را به صورت یک لیست سه تایی از اعداد ۱۶ بیتی در نظر میگیریم که عدد اول نشان دهنده کم ارزش ترین ۱۶ بیت است. برای ریزالت نیز یک لیست ۶ تایی مانند دو عدد ورودی درنظر میگیریم.

```
first       dw      1234h, 5678h, 9012h ; 901256781234h

second      dw      3456h, 7891h, 1234h ; 1234789134456h

res         dw      0h, 0h, 0h, 0h, 0h, 0h
```

حال به این صورت عمل میکنیم که متغییر res را با شروع از اولی ۱۶ بیتی اش به ترتیب پر میکنیم تا نتیجه بدست بیاید. برای این کار متغییر si را در نظر میگیریم. فقط توجه کنید از آن جایی که لیست ها از اعداد ۱۶ بیتی هستند برای حرکت روی لیست باید si را دوتا دوتا افزایش دهیم که بتواند به عدد بعدی برود.

با توجه به توضیحات بالا از si مساوی صفر یک فور زده و هر وقت که si بزرگتر از ۸ شد الگوریتم تمام است.

```
            mov si, 0
for1:       cmp si, 8
            jg exit
```

برای انجام عمل ضرب به این صورت باید عمل کنیم که در هر مرحله res[i] باید برابر شود با مجموع first[j] و second[k] هایی که j به علاوه k برابر با i باشند و همچنین در صورت وجود کری باید آن را به خانه i + 1 ببریم.

برای این کار متغییر cx را که قرار است به خانه های عدد دوم اشاره کند را ابتدا برابر si قرار میدهیم و در هر مرحله bp را که قرار است روی عدد اول حرکت کند برابر با si منهای cx قرار میدهیم. همچنین در هر مرحله قرار است cx را یکی یکی کم کرده (و به طبع bp زیاد میشود) تا بتوانیم تمام j و k های گفته شده در بالا را بدست آوریم.

```
            mov cx, si
for2:       mov bp, si
            sub bp, cx
```

از آن جایی که cx میتواند فقط از ۲ تا ۰ مقدار بپزیرد، در صورت بالاتر بودن آن باید cx را کم کرده و الگوریتم را دوباره اجرا کنیم. همچنین bp نیز اگر مقداری بیشتر از ۲ پیدا کند، الگوریتم تمام است و باید به سراغ si بعدی برویم.

```
cmp bp, 4
jg break

cmp cx, 4
jg continue
```

حال هر بار first[bp] و second[cx] را درنظر گرفته و حاصل ضرب آن ها را با res[si] جمع میکنیم و باقی آن را با res[si + 1] جمع میکنیم.

```
mov dx, 0
mov ax, ds:first[bp]
mov di, cx
mov bx, second[di]
mul bx

add res[si], ax
adc dx, 0
add res[si + 2], dx
```

در آخر نیز چک میکنیم که در صورت صفر بودن cx به پایان الگوریتم رسیده ایم و باید به si بعدی برویم و در غیر این صورت با زیاد کردن cx به حالات دیگر میرویم.

```
            cmp cx, 0
            je break

continue:   dec cx
            dec cx
            jmp for2
```

خروجی الگوریتم را میتوانید در زیر ببینید.

```
FIRST   1234h, 5678h, 9012h
SECOND  3456h, 7891h, 1234h
RES     0AD78h, 1F7Ch, 0BCD4h, 520Ah, 0D1A8h, 0A3Eh
```