

امیرحسین احمدی ۹۷۵۲۲۲۹۲ - تمرین سوم

-۱

ابتدا آرایه grades شامل نمرات که به صورت bcd packed ذخیره شده اند را تعریف میکنیم. سپس متغیرهای مورد نیاز در طول الگوریتم را تعریف میکنیم.

```
.data
grades db 09H, 03H, 10H, 14H, 18H, 12H, 16H, 08H, 09H, 13H, 15H,
size dw 20

sum dw 0
avg dw 0
new_sum dw 0
new_avg dw 0

median db 0
max db 0
diff db 0
```

ابتدا بایست مجموع اعداد آرایه را برای محاسبه ی میانگین اولیه حساب کنیم. برای این کار عدد را از حالت bcd packed به hex تبدیل میکنیم و با متغیر sum جمع میکنیم.

```
; sum
mov si, 0
mov ah, 0
sum_loop:
    mov al, grades[si]
    and al, 0F0h
    ror al, 4
    mov dl, 10
    mul dl

    mov bl, grades[si]
    and bl, 0Fh

    add al, bl
    add sum, ax

    inc si
    cmp si, size
    jl sum_loop
```

سپس برای محاسبه میانگین، sum را گرفته و بر سائز آرایه که ۲۰ است تقسیم میکنیم. از آنجایی که sum به صورت hex است، آن را به صورت bcd packed در میاوریم.

```

;avg
mov ax, sum
div size

mov ah, 0
mov cl, 10
div cl
ror al, 4
add al, ah
mov ah, 0
mov avg, ax

```

حال می‌خواهیم آرایه را سورت کنیم تا بتوانیم میانه را حساب کنیم. برای این کار دو بار روی آرایه فور می‌زنیم. برای این کار از روش selection sort استفاده می‌کنیم.

```

;sort
mov si, 0
sort_loop_i:
    mov di, si
    inc di

    sort_loop_j:
        inc di
        cmp di, size
        jge break

        mov al, grades[si]
        mov bl, grades[di]
        cmp al, bl
        jg swap
        jmp sort_loop_j

    swap:
        mov grades[si], bl
        mov grades[di], al
        jmp sort_loop_j

    break:
        inc si
        cmp si, size
        jl sort_loop_i

```

از آنجا که طول آرایه ما برابر با ۲۰ است، میانه برابر با میانگین دو عضو وسط آن است. پس دو عدد را گرفته و جمع می‌کنیم (برای جمع bcd packed از دستور daa استفاده می‌کنیم) و آن را بر ۲ تقسیم می‌کنیم.

```

;median
mov ax, 0
mov al, grades[9]
add al, grades[10]
daa
mov ah, 0
mov bl, 2
div bl
mov median, al

```

حال که آرایه سورت شده است، عدد آخر آرایه همان ماکسیموم است. آن را از ۲۰ کم کرده تا میزانی که باید به اعضای آرایه اضافه کنیم بدست بیاید.

```

;max
mov al, grades[19]
mov max, al

;diff
mov al, 20h
sbb al, max
das
mov diff, al

```

سپس روی آرایه سورت زده و اختلاف را با اعضای بای میانه جمع میکنیم.

```

;increment
mov si, 0
increment_loop:

    mov al, grades[si]
    cmp al, median
    jg increase
    jmp end_loop

    increase:
        add al, diff
        daa
        mov grades[si], al

    end_loop:
        inc si
        cmp si, size
        jl increment_loop

```

در آخر نیز مانند همان اول کار new_sum و new_avg را محاسبه میکنیم.

```

;new_sum
mov si, 0
mov ah, 0
new_sum_loop:
    mov al, grades[si]
    and al, 0F0h
    ror al, 4
    mov dl, 10
    mul dl

    mov bl, grades[si]
    and bl, 0Fh

    add al, bl
    add new_sum, ax

    inc si
    cmp si, size
    jl new_sum_loop

;new_avg
mov ax, new_sum
div size

mov ah, 0
mov cl, 10
div cl
ror al, 4
add al, ah
mov ah, 0
mov new_avg, ax

```

نتیجه ران کردن کد را نیز میتوانید در زیر ببینید.

variables	
size: word	elements: 1
edit	show as: hex
GRADES	03h, 03h, 04h, 06h, 08h, 08h, 09h, 09h, 10h, 10h, 13h, 14h, 15h, 16h, 17h, 18h, 18h, 19h, 20h, 20h
SIZE	0014h
SUM	00DC h
AUG	0011h
NEW SUM	00F0h
NEW AVG	0012h
MEDIAN	10h
MAX	18h
DIFF	02h

ابتدا یک آرایه شامل ۲۰ اسم تعریف کردم، از آنجایی که لیست استرینگ نداریم، انتهای هر اسم \$ گذاشته تا از هم تفکیک شوند. یک آرایه از اعداد به اسم ans نیز تعریف کردم که ۲۰ درایه دارد و تعداد تکرار هر کلمه را در آن نگه میدارم که در ابتدا صفر است. متغیر size نیز برابر با سایز arr یعنی ۲۰ است.

```
arr db "amirh$", "amirh$", "mmd$", "dani$", "arma
size equ $-arr
ans db 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

الگوریتم کلی به این شکل است که من روی آرایه اسم‌ها حرکت میکنم، به اول هر اسمی که رسیدم، از اول روی آرایه حرکت میکنم و تمام کلمات را با کلمه ی فعلی مقایسه میکنم و در صورت تشابه خانه متناظر در ans را افزایش میدهم.

ابتدا دو متغیر bx برای حرکت رو arr و متغیر bp برای حرکت روی ans را برابر صفر (اول آرایه) میگذارم. سپس وارد for میشوم. در هر فور ابتدا چک میکنم که اگر bx به انتهای arr رسیده بود (بزرگتر از size بود) از for خارج شود و الگوریتم تمام شود.

```
mov bx, 0
mov bp, 0

for:

    cmp bx, size
    jge the_end
```

حال برای مقایسه di را روی اول arr گذاشته که بتوانم همه کلمات را به کلمه ای که bx روی آن قرار دارد مقایسه کنم و متغیر si را برابر bx گذاشته که برای هر بار مقایسه روی کلمه فعلی حرکت کرده بدون آن که bx را گم کنم. در هر بار مقایسه ابتدا چک میکنم که اگر di به انتهای arr رسیده بود مقایسه ها تمام است و باید به کلمه بعدی برویم.

```
mov di, 0
mov si, bx

compare:

    cmp di, size
    jge next_word
```

حال si و di را یکی یکی جلو برده و با هم مقایسه میکنیم. اگر شبیه به هم بودند انقدر این کار را تکرار میکنیم تا به \$ برسیم و کلمه تمام شود که در این صورت به similar میرویم. در similar خانه مربوط به

کلمه فعلی در `ans` را افزایش داده، `di` که روی `$` است را هم یکی افزایش میدهیم تا روی حرف اول کلمه بعد برود و `si` را دوباره به اول کلمه فعلی میبریم تا دوباره عمل مقایسه را آغاز کنیم. اگر به کاراکتر غیر مشابه رسیدیم به `dissimilar` میرویم.

```
mov al, arr[si]
cmp al, arr[di]
jne dissimilar

cmp al, '$'
je similar

inc si
inc di
jmp compare

similar:
    inc ans[bp]
    inc di

    mov si, bx
    jmp compare
```

در `dissimilar` چون کلمه مشابه نبوده کاری به `ans` نداریم. از آنجایی که ممکن است وسط کلمه باشیم، `di` را آنقدر زیاد کرده تا به `$` برسیم و به کلمه بعدی برویم. `si` را نیز دوباره به اول کلمه برده و دوباره مقایسه را آغاز میکنیم.

```
dissimilar:
    inc di
    cmp arr[di-1], '$'
    jne dissimilar

    mov si, bx
    jmp compare
```

در انتها وقتی که مقایسه ها تمام شد، `bx` را که اول کلمه فعلیست آنقدر زیاد میکنیم تا به `$` برسیم و به اول کلمه بعدی برویم، `bp` را نیز افزایش داده تا در خانه متناظر `ans` قرار بگیرد و دوباره `for` را تکرار میکنیم.

```
next_word:
    inc bx
    cmp arr[bx-1], '$'
    jne next_word
    inc bp
    jmp for
```

