گزارش قسمت کدی دستورکار دهم ریزپردازنده و زبان اسمبلی

امیرحسین ادواری ۹۸۲٤۳۰۰٤- زهرا حیدری ۹۸۲٤۳۰۲۰

سوال اول)

قسمت اول:

ابتدا رشته مورد پردازش را تعریف و یک بایت نیز در انتهای آن تعریف میکنیم. پوینتر STRING_START واضحا به آغاز رشته اشاره می‌کند. پوینتر STRING_END نیز به اولین بایت بعد از آخرین بایت رشته ورودی اشاره می‌کند.

دو رشته برای چاپ عبارت YES و NO نیز تعریف می‌کنیم.

```
.DATA

    STRING_START  DB 'tryyourbestsebruoyyrt$'
    STRING_END DB 0 ; Points to the end of above string.

    YES DB 'YES', 13, 10, '$'
    NO DB 'NO', 13, 10, '$'
```

در قسمت کد، پس از مقدار دهی پوینتر دیتاسگمنت، آدرس آغاز رشته را به DI و آدرس پایان رشته را (با کسر ۲ از STRING_END آدرس پایان رشته به‌دست می‌آید) به SI می‌دهیم.

در یک حلقه متوالیا کاراکتری را که DIبه آن اشاره می‌کند با کاراکتری که SI به آن اشاره می‌کند مقایسه می‌کنیم. در صورت تساوی پیش می‌رویم در غیر اینصورت NO چاپ می‌شود.

اگر دو پوینتر همدیگر را رد کنند (SI>DI) و همه مقایسه‌ها منجر به تساوی شوند. YES چاپ می‌شود.

```
27  START_PALINDROME:
28
29      CMP DI, SI
30      JA PRINT_YES
31
32      MOV AL, [SI]
33      MOV BL, [DI]
34      CMP AL, BL
35      JNE PRINT_NO
36      INC DI
37      DEC SI
38      JMP short START_PALINDROME
39
40  PRINT_YES:
41      ; Load address of string :
42      LEA DX, YES
43
44      ; Print String loaded in DX:
45      MOV AH, 09H
46      INT 21H
47      JMP short END_PALINDROME
48
49  PRINT_NO:
50      ; Load address of string :
51      LEA DX, NO
52
53      ; Print String loaded in DX:
54      MOV AH, 09H
55      INT 21H
56
57  END_PALINDROME:
```

قسمت دوم:

```
62      MOV BX, OFFSET STRING_START
63      MOV AX, '$'
64      MOV CL, 'y'
65      MOV DL, 0
66
67  START_COUNT:
68      CMP [BX], AX
69      JE  END_COUNT
70
71      CMP CL, [BX]
72      JE INCREMENT
73      INC BX
74      JMP short START_COUNT
75  INCREMENT:
76      INC DL
77      INC BX
78      JMP short START_COUNT
79
80  END_COUNT:
81
82      MOV AH, 2
83      INT 21H
84      JMP short FINISH
85
86  FINISH:
87      ;get back to DOS
88      MOV AH, 4CH
89      INT 21H
90
```

برای شمارش تعداد y از ابتدای استرینگ (توسط BX که آدرس ابتدای رشته را نگه می-دارد) پیمایش می‌کنیم. اگر مقدار کاراکتری که BX به آن اشاره میکند معادل $ بود، به انتهای رشته رسیده‌ایم و شمارش تمام است. در غیر اینصورت اگر کاراکتر y بود مقدار DL و نیز BX را increment می‌کنیم. و اگر کاراکتر خوانده شده y و $ نبود، صرفا BX را جلوتر می‌بریم.

سوال دوم)

```
.DATA

    DATA1 DD 0EH        ; Div/Mult first Operand
    DATA2 DD 07H        ; Div/Mult Second Operand

    DATA3 DD 0,0        ; Multiplication res
    DATA4 DD 0          ; Division res

    OUT_NUMBER DD 10000000H    ; for testing print32


    OUT_STR DB 11 DUP ('$')    ; used to cast int32 to string

    BUFF DB 8
         DB ?
         DB 8 DUP(0)           ; read string from input

    NEW_LINE DB 13,10, '$'

    IN_NUMBER DD 00000000H    ; store input number in binary format
```

در ابتدا حافظه مورد نیاز برای عملکردهایی که در ادامه شرح داده می‌شود را تعریف می‌کنیم.

GETNUM:

```
GETNUM      PROC FAR

    MOV AH, 0AH
    MOV DX, OFFSET BUFF
    INT 21H

    MOV SI, OFFSET BUFF+1
    MOV CL, [SI]
    MOV CH, 0
    INC CX
    ADD SI, CX
    MOV AL, '$'
    MOV [SI], AL

    MOV AH, 9
    MOV DX, OFFSET NEW_LINE
    int 21h

    mov ah, 9
    mov dx, offset BUFF + 2 ;START OF READ VALUE
    int 21h

    DEC CX ; now cx holds the length
    MOV SI, OFFSET BUFF+1
    ADD SI, CX

    MOV DI, OFFSET IN_NUMBER
CHAR_TO_INT:
    CMP CX, 0
    JLE END_CONV
    MOV AL, [SI]

    CMP AL, '9'
    JG  CHAR
    SUB AL, 30H
    JMP short SET
```

برای دریافت عدد از ورودی ابتدا آنرا به صورت رشته دریافت می‌کنیم. سپس در انتهای آن $ قرار می‌دهیم. (لازم به‌ذکر است طبق هایلایتهای این اینتراپت، بعد از خواندن استرینگ طول آن را نیز در ابتدای بافر قرار میدهد. این مقدار را در CX قرار داده و از آن استفاده می‌کنیم.

سپس SI را در انتهای رشته خوانده شده قرار می‌دهیم و آنرا به صورت معکوس پیمایش می‌کنیم.

در هر مرحله، ۲ کاراکتر از رشته را خوانده، هرکدام را به معادل عددی شان (که بین ۰ تا F می‌باشد. روال تبدیل بدینگونه‌است که اگر کاراکتر بیشتر از مقدار ’9‘ بود ۳۷ را از آن کم می‌کنیم درغیر

اینصورت ۳۰ را از آن کم می‌کنیم) یکی از آنها را در نیبل پایینی و دیگری را در نیبل بالایی قرار می‌دهیم. سپس این مقدار ۸ بیتی را که در هر نیبل مقدار عددی یک کاراکتر قرارگرفته‌است را در آدرسی که DI به آن اشاره می‌کند قرار می‌دهیم. در نهایت DI را جلوتر برده و SI را عقبتر می‌بریم. (SI هر زمان که کاراکتر خوانده می‌شود بایستی یک بایت عقبتر بیاید) بدین‌ترتیب عدد ورودی کاراکتری را به صورت هگز (که قابلیت شرکت در محاسبات را دارد) ذخیره کردیم.

PRINTNUM:

```
PRINT32 PROC FAR
    PUSH BX
    ADD BX, 2
    CALL PRINT16
    POP BX
    CALL PRINT16

    LEA DX,OUT_STR
    MOV AH,9
    INT 21H
    RET
PRINT32 ENDP

PRINT16 PROC FAR
       MOV AX, [BX]
       MOV CX,4
       MOV BX, 16
LOOP1:
       MOV DX,0
       DIV BX

       CMP DL, 9
       JG CHAR
       ADD DL,30H
       JMP short PUSH_STACK
CHAR:
       ADD DL, 37H
PUSH_STACK:

       PUSH DX
       LOOP LOOP1

       MOV CX, 4
LOOP2:
       POP AX
       MOV [SI],AL
       INC SI
       LOOP LOOP2
       RET
PRINT16 ENDP
```

یک سابروتین PRINT16 داریم که در این سابروتین، آدرس یک عدد ١٦ بیتی را گرفته هر مرتبه در یک حلقه با ٤ تکرار، در هر مرحله عدد را تقسیم بر ١٦ کرده، باقیمانده را که مقداری عددی‌است به مقدار اسکی (کاراکتری) آن تبدیل می‌کنیم (روال کار در تبدیل بدین‌گونه است که اگر مقدار آن بزرگتر از ٩ بود با ٣٧ و در غیر اینصورت با ٣٠ جمع می‌کنیم) و سپس در استک قرار می‌دهیم. درنهایت ٤ مقدار وارد استک شده را یکی پس از دیگری از آن خارج می-کنیم و در یک استرینگ قرار می‌دهیم (توسط یک پوینتر SI که پس از نوشتن هر کاراکتر یک واحد جلو برده میشود (اضافه می‌شود))

در روتین PRINT32 دو مرتبه روتین PRINT16 را کال می‌کنیم. یکبار برای ١٦ بیت بالایی عدد ٣٢ بیتی مدنظر و بار دیگر برای ١٦ بیت پایین آن. بدین‌ترتیب عددی که در حافظه ذخیره کرده‌بودیم را به صورت یک رشته در فرمت هگز چاپ می‌کنیم.

MULNUMS:

```
MULTNUMS     PROC FAR
     MOV eax, DATA1
     IMUL DATA2

     MOV DATA1, eax ; LSB
     MOV DATA2, edx ; MSB

     MOV SI, OFFSET DATA3+4
     MOV [SI], EAX

     SUB SI,4
     MOV [SI], EDX


     LEA SI, OUT_STR
     LEA BX, DATA3
     CALL PRINT32

     LEA SI, OUT_STR
     LEA BX, DATA3+4
     CALL PRINT32

RET
MULTNUMS ENDP
```

برای انجام عملیات ضرب DATA1 را در EAX قرار می‌دهیم. سپس عملیات IMUL را با عملوند DATA2 انجام می‌دهیم. ۳۲ بیت پایین جواب در EAX و ۳۲ بیت بالایی آن در EDX قرار می‌گیرد. با این حساب هرکدام از این ۳۲ بیت را در نیمی‌از متغیر ٦٤ بیتی DATA3 قرار می‌دهیم. در نهایت آن‌هارا با روتین PRINT32 چاپ می‌کنیم.

DIVNUMS:

```
DIVNUMS      PROC FAR
    MOV EAX, DATA1
    IDIV DATA2

    MOV DATA4, EAX

    LEA SI, OUT_STR
    LEA BX, DATA4
    CALL PRINT32
    RET

DIVNUMS ENDP
```

برای عملیات تقسیم، DATA1 را در EAX قرار داده (مقسوم) سپس IDIV را با عملوند DATA2 (مقسوم علیه) اجرا می‌کنیم. در نهایت خارج‌قسمت در EAX قرار میگیرد، آنرا در DATA4 قرار می‌دهیم. در آخر نیز این دیتای ۳۲ بیتی را پرینت می‌کنیم.