

ארגון המחשב ושפת סף (203.1130)

סמסטר ב' תשס"ד
בחינה סופית - מועד א'

הוראות לנבחן:

- משך הבחינה שלוש שעות.
- מותר להשתמש בכל חומר עזר, למעט מחשבים ומחשבוניס מכל סוג.
- יש להשיב על כל השאלות.
- יש לרשום את התשובות בגוף השאלון במקומות המיועדים לכך.
- נא לכתוב בכתב יד ברור ונקי. מומלץ להשתמש בעפרון ומחק.
- בשאלון זה 15 דפים, כולל דף זה. ודא כי כל הדפים נמצאים.

ב ה צ ל ח ה !

ניקוד	ציון	
25		שאלה 1
25		שאלה 2
25		שאלה 3
25		שאלה 4
100		סה"כ

שאלה מס' 1 (25 נקודות)

א. לגבי תהליך האסמבלי של Borland Turbo Assembler, אילו מהטענות הבאות נכונות? תתכן יותר מתשובה אחת נכונה. הקף בעגול את כל התשובות הנכונות.

$$\frac{1\frac{1}{2}}{2}$$

- i. המעבר השני מתבצע תמיד. ☐
- ii. המעבר השני מתבצע כברירת מחדל. ☐
- iii. המעבר השני מופעל על ידי אופציה של הפקודה .tasm. ☒
- iv. אפשר להסתפק תמיד במעבר יחיד. ☒
- v. המעבר השני יכול לשפר את קוד המכונה של התכנית. ☒

ב. לגבי סגמנטים וקבוצות (groups) בתכניות בשפת אסמבלי, אילו מהטענות הבאות נכונות? תתכן יותר מתשובה אחת נכונה. הקף בעגול את כל התשובות הנכונות.

$$\frac{1\frac{1}{2}}{2}$$

- i. קבוצה מורכבת מסגמנטים. ☒
- ii. כל סגמנט חייב להיות חלק מקבוצה. ☒
- iii. קבוצה חייבת לכלול יותר מסגמנט אחד. ☒
- iv. קבוצות נחוצות רק לצורך קישור עם תכניות בשפה עילית. ☐
- v. איסוף סגמנטים לקבוצות מאפשר הרכבת מודלים שונים של תכניות בשפת מכונה. ☒

ג. עבור כל אחד משלושת קטעי הקוד שלהלן, רשום את תוכנו של האוגר dx בגמר ביצוע הקטע. רשום את התשובה בבסיס 10.

<pre>.data string1 db 1,45,-2,-1,8,"h",8ch,7ch,-4ch,0 string2 db 2,-1,76,-8,6,"H",9dh,8dh,+5dh,0 .code xor dx,dx mov ax,seg string1 mov ds,ax mov es,ax lea si,string1 lea di,string2 cld loop3: cmpsb je exit3 jl skip3 inc dx skip3: jb loop3 inc dx jmp loop3 exit3: nop</pre>	<pre>xor dx,dx xor ax,ax add ax,132 call proc2 jmp exit2 proc2: jz ret2 shr ax,1 call proc2 ret2: inc dx ret exit2: nop</pre>	<pre>xor dx,dx mov cx,-1 loop1: add dx,1 rcr cx,1 loopnz loop1</pre>
dx = <u>10</u>	dx = <u>0</u>	dx = <u>0</u>

$$\frac{0}{8}$$

שאלה מס' 1 (המשך)

ד. תרגם את הכתובת הלוגית (segment:offset) שלהלן לכתובת פיזית, כשהמעבד במצב real. רשום את התוצאה בבסיס 16.

1c54h:07b2h

1 C C 2

0/1

ה. בצע את פעולות החיבור והחסור שלהלן בשיטת המשלים ל-2 ברוחב של 16 ביטים. כל המספרים נתונים בבסיס 16. רשום גזז את התוצאות בבסיס 16. ציין את ערכי הדגלים CF ו-OF בגמר כל פעולה, כפי שהיו נקבעים על ידי ביצוע במעבד X86.

6f46
+
b1e1
2127

CF=1
OF=0

2/2

6f46
-
b1e1
b2c5

CF=1
OF=1

2/2

8174
+
2f8e
b102

CF=0
OF=0

2/2

8174
-
2f8e
51e6

CF=0
OF=1

1/2

ו. תרגם את המספרים שבטבלה מבסיס 10 לייצוג סטנדרטי בשיטת הנקודה הצפה בבסיס 2. רשום את החזקה ללא bias (ראה דוגמא).

Decimal	Sign	Exponent	Mantissa
9.0	0	+3	1.001
-16.5	1	+4	1.00001
0.875	0	-1	1.11

2/2

ז. להלן משתנים של תכנית אסמבלי, המכילים ערכים בשיטת הנקודה הצפה. תרגם את הערכים לבסיס 10 בייצוג ללא חזקה (ראה דוגמא). תזכורת: בשיטת הנקודה הצפה במחשב, גזז החזקה הוא מספר ללא סימן הכולל bias.

float1 dd 0c1180000h

-9.5

float2 dd 0c1840000h

-32.5

double1 dq 3fec000000000000h

+ 17 27/32
269.875

0/2

!

שאלה מס' 2 (25 נקודות)

- הגדרה: "מחרוזת ספרות עשרונית" היא רצף של בתים שבו כל בית מכיל ספרה עשרונית (כלומר, ערך בתחום 0-9). אורך המחרוזת הוא מספר הבתים.
- "מחרוזת תווים עשרונית" היא רצף של בתים שבו כל בית מכיל קוד ASCII של ספרה עשרונית (כלומר, ערך בתחום 30h-39h). אורך המחרוזת הוא מספר הבתים.

לדוגמא: המחרוזת `str1` שלהלן היא מחרוזת ספרות עשרונית, ואילו המחרוזת `str2` היא מחרוזת תווים עשרונית. אורך כל אחת מהמחרוזות הוא 8.

```
str1 db 0,5,1,9,2,0,3,9
str2 db "05192089"
```

כידוע, מספר תעודת הזהות במדינת ישראל הוא בן תשע ספרות. הספרה הימנית ביותר היא ספרת ביקורת, אשר ניתנת לחישוב באופן חד ערכי משמונה הנפרות האחרות.

האלגוריתם למישוב ספרת הביקורת נתון להלן ע"י השגרה `control` (בשפת C).

הפרמטר לשגרה הוא מצביע למחרוזת ספרות עשרונית באורך 8, המכילה את שמונה הספרות הראשונות של מספר זהות (משמאל לימין). המחרוזת עצמה נמצאת בסגמנט הנתונים. השגרה מחזירה את ספרת הביקורת המתאימה (בית המכיל ערך בתחום 0-9).

לדוגמא: עבור המחרוזת `str1` לעיל, תוחזר ספרת הביקורת 0.

```
char control(char *id)
{
    char temp, sum, i;
    sum=0;
    for (i=0; i<=7; i++)
    {
        temp=id[i];
        if (i&1) /* if i is an odd number */
        {
            temp=temp*2;
            if (temp>=10)
                temp=temp-9;
        }
        sum=sum+temp;
    }
    temp=sum%10; /* remainder of the division sum/10 */
    if (temp>0)
        temp=10-temp;
    return temp;
}
```

Handwritten notes:
 div 2
 1
 1x2
 temp=4
 temp=8

המשך שאלה מס' 2 בדף הבא

שאלה מס 2 (המשך)

א. עליך לממש בשפת אסמבלי את השגרה control לפי האלגוריתם שבדף הקודם.
הנה כי מערכת מחזיקה מספרים עשרוניים תמינה (כלומר, אין צורך לטפל בשגיאות).
הקפד על כללי התכנות הנכון של שגרות.

control proc

push bp

mov bp, sp

push dx ; temp

push dx ; i

push cx ; sum

push di ; id

~~push dx ; temp~~

~~mov cx, cx ; sum~~

~~mov dx, [bp+2] ; id~~

~~xor dx, dx ; [i=i]~~

bigloop:

cmp dx, 7

je finishloop

mov dx, [di+bx]

mov ax, dx

div 2

cmp dx, 1

jne aftercalc

mul 2

cmp ax, 10

if aftercalc

sub ax, 9

After calc:

Add cx, dx ; sum = sum + temp

jmp bigloop

bigloop:

mov dx, ax

xor dx, dx

div 10

cmp dx, 10

jle finish

המשך שאלה מס' 2 בדף הבא

mov bx, dx

mov dx, 10

sub dx, bx

finish:

xor ax, ax

mov al, dl

~~mov ax, dx~~

~~pop dx~~

~~pop cx~~

~~pop dx~~

~~pop di~~

~~ret~~

-3/6

שאלה מס 2 (המשך)

- ב. כתוב בשפת אסמבלי שגרה בשם `checkId`, שמקבלת כפרמטר במחסנית מצביע למחרת ספרות עשרונית באורך 9 המכילה מספר זהות. המחרת עצמה נמצאת בסגמנט הנתונים. השגרה בודקת האם ספרות הביקורת של מספר הזהות נכונה, ומחזירה באוגר AX את הערך 0 אם ספרות הביקורת נכונה, ואחרת את הערך 1.
 חובה להשתמש בקריאה לשגרת האסמבלי `control` מסעיף א'.
 הנת כי מועברת מחרת ספרות עשרונית תקינה (אין צורך לטפל בשגיאות).

`checkId proc`

`push bp`
`mov bp, sp`
`push dx`
`mov dx, [bp+2] + 4`

כדי נוסף - control?

`finish: pop bx`
`pop bp`
`ret`

`call control`
`add sp, 2`
`add al, 30h`
`cmp [dx+8], al`
`JAE False`
`xor AX, AX`
`jmp finish`
`False:`
`mov AX, 1`

הקווי "C" (האחרון-הראשון)
 code-אחרון

- 1/4

- i. מדוע לא ניתן לקרוא לשגרה `checkId` שבסעיף ב' מתוך תכנית בשפת C?

כי היא אינה מוגדרת כ-`public`

אם היה קובץ

- ii. בדצוננו לאפשר קריאות לשגרה `checkId` הן מתכניות בשפת C, והן מתכניות בשפת אסמבלי שכתובות בקובץ אחר. לשם כך נוסיף שתי שורות לקובץ האסמבלי בו כתובה השגרה, חזאת מיד לפני שורת הקוד הראשונה מסעיף ב'. רשום להלן שתי שורות אלו.

`public checkId`
`extern int`
`checkId proc`

`checkId` - גורם
 - 1/4

- iii. רשום את כותרת השגרה `checkId` כפי שתופיע בתכנית בשפת C שקודאת לשגרה.

`public checkId`
`extern int checkId(char *str)`
 המשך שאלה מס' 2 בדף הבא

← אישורוניג אפססמ

שאלה מס 2 (המשך)

ז. נתונה שגרה בשם a2d, שכותרתה (בשפת C) היא:

void a2d(char far *outStr, char far *inStr, int len)

הפרמטרים outStr ו-inStr הם מצביעים רחוקים, כלומר כתובות לוגיות מלאות (segment:offset). הפרמטר outStr מצביע על מחרוזת ספרות עשרונית, ואילו הפרמטר inStr מצביע על מחרוזת תווים עשרונית. שתי המחרוזות באורך זהה הנתון ע"י הפרמטר len. השגרה מתרגמת את המחרוזת עליה מצביע inStr למחרוזת ספרות עשרונית, ורשמת את התוצאה במחרוזת עליה מצביע outStr.

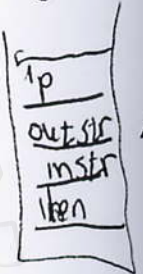
לדוגמא (ראה דף 4): מחרוזת התווים העשרונית str2 תתורגם למחרוזת הספרות העשרונית str1.

להלן מימוש השגרה a2d בשפת אסמבלי. עליך להשלים את הקוד במקומות החסרים המסומנים בקו.

```

1      .386
2      a2d  proc
3          <push bp
4          <mov bp,sp
5          pushf
6          pusha
7          push ds
8          push es
9      outStr- les di, [bp+2]
10     instr- lds si, [bp+6]
11     mov cx, [bp+10]
12     cld
13     nextc: lodsb
14     sub al, 30h
15     mov al, [di]
16     loop nextc
17     pop es
18     pop ds
19     popa
20     popf
21     pop bp
22     ret
23     endp a2d
    
```

אולימ אפססמ
subsb = סרימ
סרימ



נשהי 300
הנחית
לא הבוק דינס
נא אפססמ
למנה לא
לא אפססמ
מנה לא

מנה לא
stosb

-2/4

al
ds:si → al

ה. i. מה עושה ההנחיה 386, ומדוע היא נחוצה בקוד האסמבלי בסעיף ד' (בשורה 1)?

מכיון שסרימ מנה אפססמ double word למכיון בשורה 9
(di:es) ←

מה עושה 30h.

ii. האם, לדעתך, שורות הקוד 5 ו-20 בניסיון ד' נחוצות, או שאפשר לוותר עליהן? נמק את תשובתך.

שורות (חוצות) מנה סרימ אפססמ אפססמ קסרימ
(כיון dF)

שאלה מס 2 (המשך)

1. כתוב בשפת אסמבלי תכנית ראשית כדלקמן. התכנית מקבלת ארגומנט אחד (המוקלד בשורת הפקודה) והוא מספר זהות בן 9 ספרות. התכנית בודקת האם ספרת הביקורת של מספר הזהות נכונה. אם ספרת הביקורת נכונה, התכנית מדפיסה את התו "0" (בקוד ASCII). אחרת, התכנית מדפיסה את התו "1".

הנחיות:

- חובה להשתמש בקריאות לשגרות `checkId` ו-`a2d` מסעיפים ב' ו-ד', הנח כי השגרות והתכנית הראשית נמצאים כולם בקובץ מקור אחד.
- הנח כי הארגומנט תקין (אין צורך לטפל בשגיאות), וכי הוא מוקלד אחרי תו רווח אחד משם התכנית.
- מכורות: בתחילת ביצוע התכנית, הארגומנט נמצא בסגמנט ה-PSP החל מהיסט `82h`, והמצביע לסגמנט ה-PSP נמצא באוגר `ES`.
- אין להדפיס יותר מתו בודד, כמפורט לעיל.

```
.model small use16
.stack 100h
.data
```

```
.code
```

-5/3-

שאלה מס' 3 (25 נקודות)

א. כידוע, בעת המעבר לשגרת השרות של פניקה, המעבד מכבה את הדגל IF באוגר הדגלים. בתוך שגרת הפסיקה ניתן להדליק את הדגל IF לפי הצורך. אולם, ללא בדיקות נוספות, שגרת הפסיקה אינה יכולה לדעת האם הדגל IF היה במצב זלוק או כבוי בעת קבלת הפסיקה.

הגדר מאקרו בשם irif, ללא פרמטרים, אשר משחזר את הדגל IF באוגר הדגלים למצב בו היה בעת שהפסיקה התקבלה. הנח שהמאקרו וקרא מייד לאחר הכניסה לשגרת הפסיקה. המאקרו אינו משנה את שאר הדגלים.

```

irif macro
    push bp
    mov bp, sp
    pushf
    push ax
    check equ 10000000h
    mov AX, [bp+6]
    AND AX, check
    or [bp+2], AX
    pop ax
    popf
    pop bp
ENDM

```

4/5

ב. לפקודת החזרה משגרה ret אופרנד אופציונאלי n, שהוא קבוע (אופרנד מיידי) ללא סימן ברוחב 16 ביטים. אם האופרנד מופיע, הפקודה מנילקת מהמחסנית n בתים נוספים אחרי סילוק כתובת החזרה.

לפקודה ret מספר opcodes שונים, המותאמים, בין השאר, לסוג כתובת החזרה במחסנית (far או near) מגבלות שפת האסמבלי אינן מאפשרות למזכנת לציין במפורש בפקודה ret עצמה, האם מדובר בחזרה מסוג far או near. אפשר להתגבר על מגבלה זו ע"י הגדרת מאקרו נפרד לכל סוג של חזרה משגרה, אשר יפרוש ישירות את קוד המכונה המתאים של הפקודה ret.

הגדר מאקרו בשם retfar שמבצע חזרה נושגרה מסוג far. המאקרו מקבל פרמטר אופציונאלי n הזהה לאופרנד של הפקודה ret. קוד המכונה של הפקודה ret עבור סוג חזרה זה תופס 3 בתים. ה-opcode הוא 0CAh. הערך n תופס את שני הבתים הנותרים. אם לא מופיע פרמטר בקריאה, על המאקרו להתנהג כאילו מופיע הפרמטר 0.

```

retfar macro n
    db 0CAh
    if n > 0
        db n
    endif
    ELSE dw 0
endm

```

2/3

שאלה מס' 3 (המשך)

לפקודת החרדה מפסיקה `iret` אין אופרנדים, וזאת בסתירה מפקודת החרדה משגרה `ret` (ראה סעיף ב'). נגזיר מאקרו בשם `iretn`, שמקבל פרמטר אופציונלי `n` הזהה לאופרנד של הפקודה `ret`. המאקרו `iretn` מבצע חזרה מפסיקה, ומשנותם בפרמטר `n` בדומה לפקודה `ret`. דהיינו, אם הפרמטר מופיע, המאקרו מסלק מהמחסנית `n` בתים נוספים, אחרי סילוק כתובת החרדה והדגלים.

להלן מימוש המאקרו `iretn`.

```
1  iretn macro n
2      ifb <n>
3          ired
4          exitm
5      endif
6      push bp
7      mov bp, sp
8      irp x, <6,4,2>
9          push [bp+x]
10         pop [bp+x+1]
11     endm
12     pop bp
13     add sp, n
14     ired
15 endm
```

i. להלן שתי דוגמאות של קריאות למאקרו `iretn`. עבור כל דוגמא, רשום את הקוד בשפת אסמבלי שמתקבל מפרישת הקריאה למאקרו. יש לרשום רק שורות שיוצרות קוד מכונה (אין לרשום את ההנחיות לאסמבלי מנותנה, וכד').

<code>iretn 4</code>	<code>iretn</code>
<pre>push bp mov bp, sp push [bp+6] pop [bp+10] push [bp+4] pop [bp+8] push [bp+2] pop [bp+6] pop bp add sp, 4 ired</pre>	<pre>ired</pre>

ii. איזו בעיה עלולה להיווצר אם נחליף את שורה 8 של המאקרו `iretn` בשורה שלהלן?

`irp x, <2,4,6>`

1/2
 לא יתבצע הפופ הנכון, ולכן יישאר `bp` שגוי. כתוצאה מכך, הפופים הבאים יתבצעו על בסיס `bp` שגוי, ויפגעו הדגלים. (הוא יתחיל)
 הם לא יתחלפו, הם יפגעו (חלקם)

iii. כתוב מימוש חלופי למאקרו `iretn`, לפי הדרישות הבאות: תוכל להשתמש בקריאה למאקרו `retfar` מסעיף ב', ואסור להשתמש בפקודה `ired`. רשום להלן את המימוש החלופי במלואו. חשוב: כיצד נשחרר את אוגרי הדגלים?

`iretn macro n`

```
push bp
mov bp, sp
push ax
xchg [bp+2], ax
xchg [bp+6], ax
xchg [bp+10], ax
```

המשך שאלה מס' 3 בדף הבא

```
xchg [bp+4], ax
xchg [bp+6], ax
xchg [bp+8], ax
pop ax
pop bp
popf
retfar 4, n
```

2/3

שאלה מס' 3 (המשך)

מהי אסמבלי? מהי תוכנית?

ד. הגדר מאקרו בשם \log_2 , המקבל פרמטר אחד n , שהוא קבוע מספרי שלם. המאקרו מקצה בית אחד בזיכרון ומאתחל אותו לערך $\lfloor \log_2(n) \rfloor$ (החלק השלם של ה- \log לפי בסיס 2 של n). עבור $n \leq 0$, נגדיר $\log_2(n) = -1$.

לדוגמא: הקריאה $\log_2 27$ תפרוש און הקוד db 4
הקריאה $\log_2 -6$ תפרוש און הקוד db -1

אין צורך לבדוק בתוך המאקרו האם הערך $\lfloor \log_2(n) \rfloor$ גולש מקיבולת של בית. במקרה כזה האסמבלר יודיע ממילא על שגיאה בקוד שנפרש.

על המאקרו לבצע את החישוב כולו בזמן אסמבלי, ולפרוש קוד שתופס בית אחד בלבד. יש לבנות את המאקרו באופן יעיל, תוך שימוש במשתני אסמבלי ומבני אסמבלי מותנה מתאימים.

\log_2 Macro n

```

x=1
count=0
if n le 0
    db -1
    ExitM
endif
else
    REP: N
    if n le x
        db count-1
        ExitM
    endif
    x=x*2
    count=count+1
    end if
    endm
endm log2
    
```

4/5

ה. הגדר מאקרו בשם $\log_2\text{tab}$, המקבל פרמטר אחד m , שהוא קבוע מספרי שלם. המאקרו מקצה בזיכרון מערך בן m בתים, ומאתחל את הגיית במקום ה- i במערך ($0 \leq i \leq m-1$) לערך $\lfloor \log_2(i) \rfloor$. אם $m \leq 0$, המאקרו אינו מקצה שום מקום בזיכרון. לדוגמא, הקריאה $\log_2\text{tab } 5$ תפרוש את הקוד שלהלן.

```

db -1
db 0
db 1
db 1
db 1
db 2
    
```

על המאקרו לבצע את החישוב כולו בזמן אסמבלי. יש לבנות את המאקרו באופן יעיל, תוך שימוש במשתני אסמבלי ומבני אסמבלי מותנה מתאימים. חובה להשתמש בקריאות למאקרו \log_2 מסעיף ד'.

$\log_2\text{tab}$ Macro M

```

if M le 0
    ExitM
endif
    
```

2/3

```

Num=0
REP M
    log2 Num
    Num=Num+1
endm log2tab
    
```

לחומיד טורה

שאלה מס' 4 (25 נקודות)

א. ברצוננו לעדכן את שדות מס' 25h של פסיקה 21h. כידוע, שדות זה משמש לטעינת וקטור פסיקה לתוך טבלת וקטורי הפסיקה. ברצוננו ששגרת השדות תדפיס על המסך את התו "!" בכל פעם שמתבצעת טעינת וקטור פסיקה. אין שנוי בשאר השדות של פסיקה 21h.

להלן תכנית המיישמת את שגרת השדות החדשה של פסיקה 21h כמתואר לעיל. בנוסף, התכנית מיישמת שגרת שדות חדשה עבור פסיקה 0.

```

1      .model small
2      .stack 100h

3      .code
4      oldInt21 dd ?
5      newInt21: cmp ah,25h
6                je do25
7                jmp [oldInt21]

8      do25: push ax
9            push dx
10           mov dl,"!"
11           mov ah,2
12           int 21h
13           pop dx
14           pop ax
15           jmp [oldInt21]

16     oldint0 dd ?
17     newInt0: lds dx,oldInt:21
18             mov ax,2521h
19             int 21h

20           [ lds dx,oldInt:0
21             mov ax,2500h
22             int 21h ]

23           mov ah,4ch
24           int 21h
25           iret

26     main: mov ax,3521h
27           int 21h
28           mov word ptr oldInt21,bx
29           mov word ptr oldint21+2,es

30           mov ax,3500h
31           int 21h
32           mov word ptr oldInt0,bx
33           mov word ptr oldint0+2,es

34           mov ax,cs
35           mov ds,ax
36           [ mov dx,offset newInt21
37             mov ax,2521h
38             int 21h ]

39           [ mov dx,offset newInt0
40             mov ax,2500h
41             int 21h ]

42           [ xor bx,bx
43             div bx ]

44           mov ah,4ch
45           int 21h

46           end main

```

Handwritten notes and annotations:

- Next to line 12: $int\ 21h \leftarrow \text{!}$
- Next to line 19: $int\ 21h$
- Next to line 22: $int\ 21h$
- Next to line 24: $int\ 21h$
- Next to line 38: $ah=2, al=71$
- Next to line 41: 25
- Next to line 43: $?$
- Next to line 45: $int\ 21h$
- Next to line 46: $end\ main$
- Next to line 26: $main:$
- Next to line 27: $int\ 21h$
- Next to line 28: $mov\ word\ ptr\ oldInt21,bx$
- Next to line 29: $mov\ word\ ptr\ oldint21+2,es$
- Next to line 30: $mov\ ax,3500h$
- Next to line 31: $int\ 21h$
- Next to line 32: $mov\ word\ ptr\ oldInt0,bx$
- Next to line 33: $mov\ word\ ptr\ oldint0+2,es$
- Next to line 34: $mov\ ax,cs$
- Next to line 35: $mov\ ds,ax$
- Next to line 36: $mov\ dx,offset\ newInt21$
- Next to line 37: $mov\ ax,2521h$
- Next to line 38: $int\ 21h$
- Next to line 39: $mov\ dx,offset\ newInt0$
- Next to line 40: $mov\ ax,2500h$
- Next to line 41: $int\ 21h$
- Next to line 42: $xor\ bx,bx$
- Next to line 43: $div\ bx$
- Next to line 44: $mov\ ah,4ch$
- Next to line 45: $int\ 21h$
- Next to line 46: $end\ main$

שאלה מס' 4 (המשך)

השאלות x-i שלהלן מתייחסות לתכנית שבדף הקודם. מומלץ לעקוב אחרי התכנית לפי סדר השאלות, ולעבור על כל השאלות מראש, לפני כתיבת התשובות.

- i. מה עושות שורות 26-29? מה עושות שורות 30-33?
 סוגיות הכתובות מופיעות שם. עקרו הפסיקה המקורית של ה-20!
 26-29 - סוגיות הכתובות מופיעות שם.
 30-33 - סוגיות הכתובות מופיעות שם.
- ii. מה עושות שורות 34-38? מה עושות שורות 39-41?
 34-38 - סוגיות הכתובות מופיעות שם.
 39-41 - סוגיות הכתובות מופיעות שם.
- iii. מה עושות שורות 7-4? באילו מקרים מונבעת ההסתעפות בשורה 7?
 סוגיות הכתובות מופיעות שם. אלו הן הסוגיות המקוריות המקרה והסוגיות.
 25 - סוגיות הכתובות מופיעות שם.
- iv. האם הדפסת התו "!" בשורה 12, מבצעת לפני או אחרי טעינת וקטור הפסיקה? הסבר.
 הדפסת התו "!" מבצעת לפני או אחרי טעינת וקטור הפסיקה. (שורות 34-38)
 כי תבצע יפסק הקטע (שורה 12) והדפסת "!"
- v. כמה פעמים יודפס התו "!" במהלך ביצוע קטע הקוד בשורות 34-41? הסבר.
 סוגיות הכתובות מופיעות שם. אלו הן הסוגיות המקוריות המקרה והסוגיות.
- vi. איזו שורה תבצע מיד אחרי שורה 43? הסבר.
 שורה 12 - הסוגיות המקוריות מופיעות שם. אלו הן הסוגיות המקוריות המקרה והסוגיות.
- vii. מהו מספר השורה האחרונה שתבצע לפני שהתכנית תסתיים? הסבר.
 שורה 24 - סוגיות הכתובות מופיעות שם. אלו הן הסוגיות המקוריות המקרה והסוגיות.
- viii. כמה פעמים בסה"כ יודפס התו "!" במהלך ביצוע התכנית, מראשיתה ועד סיומה? הסבר.
 סוגיות הכתובות מופיעות שם. אלו הן הסוגיות המקוריות המקרה והסוגיות.
- ix. נחליף את שורה 38 בצמד השורות שלהלן.
 pushf
 call far ptr newInt21
 מהי כעת התשובה לסעיף v? מדוע יש הבדל בין התשובות בשני המקרים?
 סוגיות הכתובות מופיעות שם. אלו הן הסוגיות המקוריות המקרה והסוגיות.
- x. איזו בעיה עלולה להיווצר אם נמחק את שורה 43?
 סוגיות הכתובות מופיעות שם. אלו הן הסוגיות המקוריות המקרה והסוגיות.

המשך שאלה מס' 4 בדף הבא

שאלה מס' 4 (המשך)

ב. נגדיר שורות חדש של פסיקה 21h, שמספרו יהיה 0ffh. אין שינוי בשאר השרותים של פסיקה 21h. השרות החדש משמש לקליטת תו מהמקלדת, והוא משלב מאפיינים של מספר שרתי קלט ופלט קיימים.

השרות החדש מקבל כפרמטרים את האוגר DL ואת הדגל DF באוגר הדגלים. השרות פועל כדלקמן.

- קוראים תו אחד מחוצץ המקלדת, תוך המתנה להקלדה אם החוצץ ריק.
- אם הדגל DF כבוי, לא מתבצע echo. אחרת, מתבצע echo אל המסך באופן הבא:
- אם האוגר DL מכיל 0, מדפיסים את התו שנקרא מהמקלדת. אחרת, מדפיסים את התו שבאוגר DL.
- מחזירים באוגר AL את קוד ה-ASCII של התו שנקרא, ובאוגר AH את קוד ה-SCAN שלו.
- אין להרוס את ערכי שאר האוגרים.

כתוב להלן את שגרת השרות של פסיקה 21h שתיישם את השרות החדש. השגרה תתחיל בתווית newInt21. הנח כי וקטור הפסיקה של שגרת השרות הקודמת נשמר במשתנה oldInt21 בסגמנט הקוד של שגרת השרות החדשה.

.code
oldInt21
newInt21:

```
dd ?  
cmp ah, 0ffh  
je do0ffh  
jmp [oldInt21]  
do0ffh: mov ah, 0  
push bp  
mov bp, sp  
mov ax, [bp+2]  
and ax, 00000000h  
cmp ax, 0  
jne DFnotclear  
DFnotclear: cmp dl, 0  
jz BFromBuffer  
mov ah, 2  
jmp [oldInt21]  
BFromBuffer: mov dl, al  
mov ah, 2  
jmp [oldInt21]  
Finish: pop bp  
pop f  
pop ax  
pop ax  
iret
```

Notecho:
jmp [oldInt21]
jmp finish

ג. כתוב קטע קוד שקולט תו ומוקלדת 8 תווים לתוך חוצץ בשם input שנמצא בסגמנט הנתונים, כאשר תוך כדי קלט מתבצע echo של נוכביות אל המסך. חובה להשתמש בשרות החדש מסעיף ב'. אין להשתמש בשרותי קלט או פלט אחרים.

.data
input db 8 dup(?)

```
.code  
mov ax, 0d0ba  
mov ls, ax  
mov bl, 1  
mov bx, offset input  
mov cx, 8  
mov ah, 0ffh
```

```
myloop: int 21h  
mov byte ptr [bx], al  
inc bx  
loop myloop  
mov ax, 4ch  
int 21h
```


ארגון המחשב ושפת סף (203.1130)

סמסטר ב' תשס"ג
בחינה סופית - מועד א'

הוראות לנבחן:

- משך הבחינה שלוש שעות.
- מותר להשתמש בכל חומר עזר, למעט מחשבים ומחשבוניס מכל סוג.
- יש להשיב על כל השאלות.
- יש לרשום את התשובות בגוף השאלון במקומות המיועדים לכך.
- נא לכתוב בכתב יד ברור ונקי. מומלץ להשתמש בעפרון ומחק.
- בשאלון זה 15 דפים, כולל דף זה. ודא כי כל הדפים נמצאים.

ב ה צ ל ח ה !

ציון	ניקוד	
13	25	שאלה 1
20	25	שאלה 2
12	25	שאלה 3
19	25	שאלה 4
64	100	סה"כ

שאלה מס' 1 (25 נקודות)

א. נתון שדגל הסימן דלוק (sf=1), והאוגר ax מכיל ערך שלילי. עבור כל אחת מהפקודות שלהלן, ציין מתי היא גורמת לכיבוי דגל הסימן: תמיד, לפעמים כן ולפעמים לא, או אף פעם. הקף בעגול את התשובה הנכונה.

תמיד / לפעמים / אף פעם	sar ax,1
תמיד / לפעמים / אף פעם	sal ax,1
תמיד / לפעמים / אף פעם	shr ax,1
תמיד / לפעמים / אף פעם	add ax,ax
תמיד / לפעמים / אף פעם	neg ax
תמיד / לפעמים / אף פעם	or ax,bx
תמיד / לפעמים / אף פעם	mov ax,0
תמיד / לפעמים / אף פעם	sbb ax,ax

$-\frac{1}{2}$

1001
1001
0

ב. לגבי האסמבלר, הקשר (linker) והטען (loader), אילו מהטענות שלהלן נכונות? תתכן יותר מתשובה אחת נכונה. הקף בעגול את כל התשובות הנכונות.

- הקשר מכין מידע שימש את הטען.
- האסמבלר מכין מידע שימש את הקשר אך לא את הטען.
- האסמבלר מכין מידע שימש גם את הקשר וגם את הטען.
- הטען אינו מסוגל לתפקד ללא מידע שהוכן על ידי האסמבלר.

$-\frac{1}{4}$

ג. הכרנו את כלל התכנות הבא עבור שפת אסמבלי: כאשר קוראים לשגרה עם פרמטרים במחסנית, המקום הקורא הוא האחראי להוצאת הפרמטרים מן המחסנית לאחר החזרה מן השגרה. אילו מהטענות שלהלן נכונות לגבי כלל זה? תתכן יותר מתשובה אחת נכונה. הקף בעגול את כל התשובות הנכונות.

- הכלל הכרחי תמיד, כי השגרה אינה מסוגלת להוציא את הפרמטרים מן המחסנית.
- הכלל משמש לצורך קישור עם תכניות הכתובות בשפה עילית.
- הכלל מאפשר לתת לתכנית מבנה מסודר ומודולרי.
- הכלל הכרחי כאשר משתמשים בשגרה שמקבלת מספר משתנה של פרמטרים במחסנית.
- הכלל מיותר כאשר גם השגרה וגם המקום הקורא כתובים בשפת אסמבלי.

$-\frac{1}{2}$

ד. עבור כל אחד משלושת קטעי הקוד שלהלן, רשום את תוכנו של האוגר dx בגמר ביצוע הקטע. רשום את התשובה בבסיס 10.

```
.data
o list db 5,4,3,2,1,0
6 count dw count-list-1

.code
xor dx,dx
xor bx,bx
x: mov cx,word ptr list[bx]
a: inc dx
loop a
inc bx
cmp bx,count
js x
```

dx=0
bx=0
cx=5
dx=1
cx=9
dx=7
bx=3
dx=9
cx=10

dx=5
 $-\frac{3}{2}$

המסר שאלה מס' 1 בדף הבא

```
b: xor dx,dx
xor ah,ah
inc dx
inc ah
sahf
jns b
jnz b
jnc b
```

dx=193

store ah into flags

dx=0
ah=0
dx=1

```
c: xor dx,dx
xor ax,ax
mov ax,8001h
stc
inc dx
adc ax,1
sar ax,1
jb c
```

dx=5
 $-\frac{2}{2}$

dx=0
ax=8001h/0000/0000/0000
CF=1

שאלה מס' 2 (25 נקודות)

א. הפונקציה הרקורסיבית $q(bx, cx)$ מוגדרת להלן על ידי אלגוריתם בשפת C.

```
int q(int bx, int cx)
{
    if ((bx<=0) || (cx<=0)) return 0;
    else if (bx==cx) return q(bx,bx-1)+1;
    else if (bx<cx) return q(bx,bx);
    else return q(bx,cx-1)+q(bx-cx,cx);
}
```

עליך לממש את הפונקציה q בשפת אסמבלר, בהתאם לדרישות הבאות:

- המימוש בעזרת שגרה אחת בלבד, בשם q . אין להשתמש בשגרות פנימיות נוספות.
 - הפרמטרים bx ו- cx מועברים באוגרים bx ו- cx בהתאמה. התוצאה מוחזרת באוגר ax .
 - מותר להשתמש באוגרים ax , bx , cx , bp ו- sp בלבד. אין להשתמש באוגרים אחרים.
 - אסור להשתמש במשתנים בזכרון.
 - מותר להשתמש במתמטית אך ורק לצורך שמירה ושחזור של אוגרים.
- הנח כי לא תהיה גלישה מהאוגרים במשך החישוב.
השלם את קוד השגרה q בתוך השלד שלהלן.

```
q
proc
push bp
mov bp, sp
push bx
push cx
xor ax, ax
cmp bx, 0
jle qexit
cmp cx, 0
jle qexit

cmp bx, cx
je l-bx-cx
jl l-bx-l-cx
push cx
push bx
dec cx
call q
pop bx
pop cx
```

sub bx, cx

call q

jmp qexit

```
l-bx-cx: mov cx, bx
dec cx
call q
inc ax
jmp qexit
```

```
l-bx-l-cx: mov cx, bx
call q
```

```
qexit: pop cx
pop bx
pop bp
ret
endp
q
```

המשך שאלה מס' 2 בדף הבא

שאלה מס 2 (המשך)

ב. i. כתוב שגרה בשם `cube`, שמקבלת במחסנית מספר x מטיפוס ממשי, המיוצג בשיטת הנקודה הצפה ברוחב 64 ביטים. השגרה מחזירה באוגר `ST(0)` את הערך x^3 (החזקה השלישית של x).

```
cube proc
    push bp
    mov bp, esp
    fld qword ptr [ebp+4]
    fld qword ptr [ebp+4]
    fmul st(1), st(0)
    fmulp st(1), st(0)
    pop bp
    ret
cube endp
```

אכזר פחית

ii. בסגנון הנתונים מוגדרים שני משתנים, `num` ו-`res`, שניהם מטיפוס ממשי בנקודה צפה ברוחב 64 ביטים. כתוב קטע קוד הקורא לשגרה `cube` עם הפרמטר `num`, ומציב את הערך המוחזר מן השגרה לתוך המשתנה `res`. הקפד על כללי התכנות הנכון של קריאה לשגרה.

```
.data
num    dq    123.456
res    dq    ?
.code
```

```
push num
call cube
add esp, 8
fstp res
```

אין צורך לכתוב
ב הוגר, במקום את

שאלה מס 2 (המשך)

הגדרה: במחרוזת חווים, "המקום הנבחר" הוא הבית הראשון (מצד שמאל) במחרוזת שמכיל את התו רווח (כלומר את הערך 20h בקוד ascii).

להלן נתונה השגרה findSpace. השגרה מקבלת כפרמטר באוגרים es:di כתובת לוגית מלאה (סגמנט והיסט) של מחרוזת תווים. אורך המחרוזת מועבר כפרמטר באוגר cx, בייצוג ללא סימן.

השגרה מחפשת את המקום הנבחר במחרוזת התווים שהועברה כפרמטר. אם המקום הנבחר נמצא, השגרה מחזירה באוגרים es:di את הכתובת הלוגית המלאה שלו, ובאוגר cx את האורך הנוסף של המחרוזת (כלומר, החל מן המקום הנבחר ועד סוף המחרוזת המקורית, כולל שני הקצוות). אם לא נמצא במחרוזת המקום הנבחר, השגרה מחזירה באוגר cx את הערך 0 והורסת את תוכן האוגר di.

```

1 findSpace: push ax
2             xor al,al
3             add al,20h
4             cld
5             repne scasb
6             jne endSpace
7             inc cx
8             dec di
9 endSpace:   pop ax
10            ret
    
```

i. השגרה findSpace מחזירה באוגר cx את הערך 0 אם ורק אם המקום הנבחר אינו נמצא במחרוזת. הסבר כיצד השגרה מקיימת טענה זו.
מה קורה כאשר המקום הנבחר הוא הבית האחרון במחרוזת? מה קורה כאשר השגרה מקבלת מחרוזת באורך 0?

הבקשה לפניה S מורה על המחרוזת של סוף 0! (אפשרות של חקירה)
 $2F=0$ (אם שניה אחת היא di: 20h). נקרא משתנה זה S. שמו, נקרא
 הוא שמשמש יקרא. אם לא $2F=0$ (זמן) המחרוזת יקרא, $cx=0$ וקצוות
 Rod. אם כן נקראי למחרוזת של הנקודים (אחרי) ונקראי "א" אחר של
 היא היתה

ii. כתוב בשפת אסמבלי שגרה בשם findNotSpace, הוזה בהגדרתה לשגרה findSpace, מלבד הבדל אחד: "המקום הנבחר" אותו מחפשת השגרה findNotSpace מוגדר כבית הראשון במחרוזת שאינו מכיל את התו רווח.
 רמז: נדרשים שינויים קלים בלבד בשגרה findSpace.

```

findspace:  push ax
            xor al,al
            add al,20h
            cld
            repne scasb
            jne endspace
            inc cx
            dec di
endspace:   pop ax
            ret
    
```

לפני
 $2F=0$
 שניה אחת היא di: 20h
 שמו, נקרא
 הוא שמשמש יקרא

בגוף המחרוזת הנבחרת הוא הבית הראשון, ואם יתכן שיהיה הקטנה
 אחר לא (אם לא) ונקראי "א" אחר של
 הוא שמשמש יקרא משתנה זה S. שמו, נקרא

המשך שאלה מס' 2 בדף הבא

שאלה מס 2 (המשך)

הגדרה: במחרוזת תווים, "אסימון" (token) הוא תת מחרוזת מקסימאלית שאיננה מכילה אף תו רווח.
 לדוגמא: במחרוזת "Hello World!" שני אסימונים: "Hello" ו-"World!".
 השגרה numTokens מקבלת במחסנית שני פרמטרים: כתובת לוגית מלאה (סגמנט והיסט) של מחרוזת תווים, ואורך המחרוזת (בגודל מילה בייצוג ללא סימן). השגרה מחזירה באוגר ax את מספר האסימונים במחרוזת.
 כתוב להלן את השגרה numTokens בשפת אסמבלי. חובה להשתמש בקריאות לשגרות findSpace ו-findNotSpace מסעיף ג'. הקפד על כללי התכנות הנכון של שגרה.

numTokens proc

```
push bp
mov bp, sp
push di
push si
push es
xor ax, ax
mov ax, [bp+6]
mov es, ax
mov di, [bp+4]
mov cx, [bp+8]
```

```
push di
call findSpace
pop si
cmp di, si
je equal
inc ax
```

```
equal: call findNotSpace
      cmp cx, 0
      jne l

pop es
pop si
pop di
pop bp
ret
endp
```

ה. כתוב בשפת אסמבלי שגרה בשם argc, ללא פרמטרים, שמחזירה באוגר ax את מספר האסימונים שבשורות הפקודה של DOS שהפעילה את התכנית, לא כולל שם הפקודה עצמה.
 חובה להשתמש בקריאה לשגרה numTokens מסעיף ד'.
 לדוגמא: שורת הפקודה שלהלן מפעילה את התכנית stam, שמבצעת קריאה לשגרה argc.
 הערך שמוחזר על ידי argc בדוגמא זו הוא 6.

C:\>stam yom shel hol im boker kahol

מחזור: המחרוזת של שורת הפקודה (לא כולל שם הפקודה) נמצאת בסגמנט ה-PSP, החל מהיסט 81h.
 אורך המחרוזת נמצא בבית שבהיסט 80h. הנת כי כתובת התחלת סגמנט ה-PSP נמצאת באוגר es.

argc proc

```
xor ax, ax
mov ax, es: [80h]
xor ah, ah
cmp ax, 0
je exit
push ax
push es
```

```
mov ax, [81h]
push ax
call numTokens
add sp, 6
cmp ax, 0
je exit
dec ax
ret
exit:
endp
```

כתוב קטע קוד בשפת אסמבלי, שקורא לשגרה argc מסעיף ה', ולאחר מכן מדפיס בצורה נאה את התוצאה המוחזרת מהשגרה, וזאת בעזרת קריאה לשגרה printf של שפת C. הגדר בסגמנט הנתונים את מחרוזת ההדפסה המועברת ל-printf.

```
.data
printStr dw " %d \n", 13, 10, 0
call argc
add sp, 4
mov cx, offset printStr
push bx
push ax
```

שאלה מס' 3 (25 נקודות)

א. כידוע, פקודת המכונה pop שולפת איבר מראש המחסנית, ומעבירה אותו לאופרנד של הפקודה. אין אפשרות, באמצעות הפקודה pop, לשלוף איבר מראש המחסנית ופשוט לזרוק אותו (דהיינו לא להעביר אותו לאף מקום).

עליך להגדיר מאקרו בשם popp, המקבל פרמטר בודד אופציונאלי הזהה לאופרנד של פקודת המכונה pop. כאשר מתבצעת קריאה למאקרו הכוללת פרמטר, המאקרו פועל באופן זהה לפקודה pop. ואילו כאשר הקריאה היא ללא פרמטר, המאקרו שולף מראש המחסנית מילה אחת וזורק אותה. המאקרו popp אינו משפיע על הדגלים, זאת בדיוק כפי שמתנהגת פקודת המכונה pop. יש להימנע מתופעות לוואי, כגון שינוי האוגרים (פרט לאוגר sp).

x macro pop
 ifnb < x >
 pop x
 exitm
 endif
~~mov esp, ax~~
 pop ax
 endm

ב. הגדר מאקרו בשם puship, ללא פרמטרים, הדוחף למחסנית את תוכן האוגר ip (instruction pointer) כפי שהוא בתחילת ביצוע המאקרו (כלומר, ההיסט אל הבית הראשון בקוד המכונה הנפרש על ידי המאקרו). על המאקרו לפרוש פקודת מכונה אחת בלבד. המאקרו אינו משפיע על הדגלים. הערה: ראה גם סעיף ג' להלן.

puship macro
 mov ~~esp, esp+2~~ ~~esp~~
 push ip
 endm

IP - 1
 ↑
 יש למחוק
 את כל

ג. הגדר מאקרו בשם pushipn, ללא פרמטרים, הדוחף למחסנית את תוכן האוגר ip כפי שהוא בגמר ביצוע המאקרו (כלומר ההיסט אל הבית הראשון שאחרי קוד המאקרו). על המאקרו לפרוש פקודת מכונה אחת בלבד. השונה מפקודת המכונה שפורש המאקרו puship מסעיף ב'. המאקרו אינו משפיע על הדגלים.

pushipn macro
 mov IP2, [esp+2]
 push ip
 endm

המשך שאלה מס' 3 בדף הבא

pivot macro p,n1,n2,n3,n4,n5,n6,n7,n8

המאקרו מקצה בזיכרון רצף של מילים, כמספר הפרמטרים m8-m1 שהועברו בפועל בקריאה. המילים הראשונות ברצף מכילות את אלה מהפרמטרים m8-m1 שערכם קטן או שווה לערך p (לפי סדר ההופעה של ערכים אלה בקריאה למאקרו). המילים הבאות ברצף מכילות את כל הפרמטרים הנותרים מבין m8-m1 (גם כאן לפי סדר הערכים בקריאה למאקרו).

```
dw 50
dw -100
dw 100
dw 50
dw 231
dw 100h
```

pivot element $P, n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8$

if nb < 0 then
if p < LE then
dw x

הקדמה: $x, \langle n_1, \dots, n_8 \rangle$
הקדמה: n_1, \dots, n_8
הקדמה: n_1, \dots, n_8

else

pivot $p, n_2, n_3, n_4, n_5, n_6, n_7$ preorder traversal

17p $x \in \{n_1, n_2, n_3, n_4, n_5, n_6, n_7, n_8\}$

cho x
endm
endm

שאלה מס' 3 (המשך)

ה. נגדיר זוג מאקרו-ים: while ו- endwhile המשמשים לכתיבת לולאה בשפת אסמבלי שדומה למבנה הבא בשפת C.

```
while (x cond y){ . . . };
```

להלן הגדרות המאקרו-ים.

```
while macro id,x,cond,y
    nextw&id: cmp x,y
                j&cond dow&id
                jmp     endw&id
dow&id:
endm
```

```
endwhile macro id
    jmp nextw&id
endw&id:
endm
```

להלן שלוש דוגמאות של לולאות המשתמשות במאקרו-ים לעיל. לכל דוגמא, רשום את הקוד בשפת אסמבלי שמתקבל לאחר פרישת כל הקריאות למאקרו. אין לרשום את כותרות המאקרו-ים.

<pre>xor si,si while 1,si,nge,len3 mov str3[si], ' ' inc si endwhile 1</pre>	<pre>xor ax,ax xor bx,bx while 2,ax,mp,100 add ax,bx inc bx endwhile 2</pre>	<pre>xor si,si while 3,si,b,len1 mov match[si],0 mov al,str1[si] xor di,di while 4,di,b,len2 cmp al,str2[di] jne skip inc match[si] skip: inc di endwhile 4 inc si endwhile 3</pre>
<p>XOR SI,SI</p> <pre>nextw1: cmp si,len3 jnge dow1 jmp endw1 dow1: mov str3[si], ' ' inc si jmp nextw1 endw1:</pre>	<p>XOR AX,AX XOR BX,BX</p> <pre>nextw2: cmp ax,100 jmp dow2 jmp endw2 dow2: add ax,bx inc bx jmp nextw2 endw2:</pre>	<p>XOR SI,SI</p> <pre>next3: cmp si,b jb dow3 jmp endw3 dow3: mov match[si],0 mov al,str1[si] xor di,di nextw4: cmp di,b jb dow4 jmp endw4 dow4: cmp al,str2[di] jne skip inc match[si] skip: inc di jmp nextw4 endw4: inc si jmp nextw3 endw3:</pre>

המשך שאלה מס' 3 בדף הבא

שאלה מס' 3 (המשך)

1. מהו תפקידו של הפרמטר id במאקרו-ים while ו- endwhile מסעיף ה'?

id - סמל ל- labels ב- מקומות

✓

2. איזו בעיה קיימת בקוד שנפרש בדוגמא האמצעית בסעיף ה'?

add ax, bx

ציר ה-ax לא נקבע כלל, אלא

(-1)

3. בעזרת המאקרו-ים מסעיף ה', נגדיר זוג מאקרו-ים: do ו- enddo, המשתמשים לכתיבת לולאה בשפת אסמבלי שדומה למבנה הבא בשפת C.

do { . . . } while (x cond y);

להלן הגדרת המאקרו enddo.

```
enddo macro id
    endwhile id
endm
```

השלם להלן את הגדרת המאקרו do במקומות המסומנים בקו. אין להוסיף פרמטרים או שורות.

```
do macro id,x,cond,y
    jmp down label
    while id, x, cond, y
endm
```

✓

שאלה מס' 4 (25 נקודות)

חלק 1

בגלל טעות בייצור, יצאה לשוק סדרה פגומה של מעבדי X86, בהם לא מוגדרת הפקודה pushf (אשר קוד המכונה שלה הוא הבית הבודד 9ch).

כידוע, כאשר המעבד מנסה לבצע פקודת מכונה שאינה מוגדרת, נוצרת חריגת "illegal opcode". שמטפורה הוא 6. הכתובת שנכנסת למחסנית בעת המעבר לשגרת השרות של חריגה 6 היא זו של הפקודה שגרמה לחריגה.

כדי להתגבר על הפגם במעבד, הוחלט לשכלל את שגרת השרות של חריגה 6 כדלקמן. אם הפקודה שגרמה לחריגה היא pushf, שגרת השרות תבצע סימולציה של פקודה זו. אין שינוי בטיפול בשאר המקרים של חריגה זו.

להלן שגרת השרות החדשה של פסיקה 6, המממשת שכלול זה. השגרה מתחילה בתווית newInt06. הנח כי וקטור הפסיקה של שגרת השרות המקורית נשמר במשתנה oldInt06 שמוגדר בסגמנט הקוד. כמו כן, הנח כי רוחב אוגר הדגלים הוא תמיד 16 ביטים.

```

1  oldInt06    dd ?
2  newInt06:   sub    sp,2
3              push   bp
4              mov    bp,sp
5              push   bx
6              push   es
7              les    bx,[bp+4]
8              cmp    byte ptr es:[bx],9ch
9              je     pushFlags
10             pop    es
11             pop    bx
12             pop    bp
13             add    sp,2
14             jmp    oldInt06
15  pushFlags:  mov    bx,[bp+8]
16             xchg   bx,[bp+6]
17             xchg   bx,[bp+4]
18             mov    [bp+2],bx
19             inc    word ptr [bp+2]
20             pop    es
21             pop    bx
22             pop    bp
23             iret

```


i. מה עושות שורות 9-77 באילו מקרים מתבצעת ההסתעפות בשורה 9?

מה עושה שורה 14? כיצד החרים משגרת החריגה כאשר מתבצעת שורה זו?

iii. רשום את תוכן המחסנית בתחילת ביצוע שורה 15, ובגמר ביצוע שורה 18. כל איבר במחסנית הוא בגודל מילה. צייר גם לאיזה איבר במחסנית מצביעים האוגרים sp ו- bp .

iv. מה תפקידה של שורה 72? מה עושות שורות 18-71?

מקדמה של 10 באתחול המוקדמות
מה עושה שורה 19?
מסומן לקידום של יציר קיים שזה באתחול כח. באתחול.

vi. מה נמצא בראש המחסנית לאחר ביצוע שורה 23?

שאלה מס' 4 (המשך)

ב. בשורה 8 בקוד שגרת השרות החדשה, האם השימוש ב-casting עבור האופרנד הראשון הכרחי? הסבר.

לא, כי לא ייתכן שיש פונקציה שמקבלת פונקציה כפרמטר, ולכן לא צריך casting.

ג. מדוע מוגדר המשתנה oldInt06 דווקא בסגמנט הקוד? כיצד, אם בכלל, ניתן להגדיר משתנה זה בסגמנט הנתונים? הסבר.

כי זהו זיכרון שייך למערכת ולא למשתמש, ולכן לא ניתן להשתמש בו. יש להשתמש ב-06h כדי להבדיל אותו מהזיכרון של המערכת.

ד. מסתבר כי במעבדים מהסדרה הפגומה לא מוגדרת גם הפקודה popf (אשר קוד המכונה שלה הוא הבית הבודד 9dh). עליך להוסיף לשגרה החדשה של חריגה 6 קוד אשר יבצע סימולציה של הפקודה popf, בדומה לסימולציה של הפקודה pushf.

רשום להלן את התוספות לקוד האסמבלי של שגרת החריגה. עליך למספר את השורות הנוספות בהתאמה למספרי השורות הקיימות. לדוגמא: שורות שיתווספו אחרי שורה 9 ימוספרו 9.1, 9.2 וכד'. אם יש צורך לשנות או למחוק שורה קיימת, רשום את מספר השורה ואת קוד האסמבלי החדש שלה.

9.1 `cmp byte ptr es:[bx], 9dh`

9.2 `je RopFlags`

24 `RopFlags: mov bx, [bp+8] ; bx = flags`
`xchg bx, [bp+2] ; bx = space`
`xchg bx, [bp+6]`
`xchg bx, [bp+4]`
`inc word ptr [bp+2]`
`pop es`
`pop bx`
`pop bp`

יש להוסיף את השורות הבאות:

`iret`
`add sp, 2`
`iret`

הקוד של `iret` הוא קוד שייבצע את הפונקציה `iret`.

המשך שאלה מס' 4 בדף הבא

חלק II

כידוע, פסיקת שרותי המקלדת 16h מספקת, בין השאר, את השרות הבא: כאשר ah=0, מתבצעת הוצאה של התו הבא מחוצץ המקלדת; קוד ה-ascii של התו מוחזר באוגר al, ואילו קוד ה-scan מוחזר באוגר ah.

ה. עליך לכתוב שגרת שרות חדשה לפסיקה 16h, שבה השרות עבור ah=0 ישתנה כדלקמן. אם התו שהוקלד נכנס לחוצץ המקלדת הוא אות אנגלית קטנה (lowercase), יוחזר באוגר al קוד ה-ascii של האות הגדולה (uppercase) המקבילה. ולהפך, אם התו בחוצץ המקלדת הוא אות גדולה, יוחזר באוגר al קוד ה-ascii של האות הקטנה המקבילה. עבור כל תו אחר, יוחזר באוגר al קוד ה-ascii המקורי שבחוצץ המקלדת. קוד ה-scan יוחזר באוגר ah, ללא שינוי מהשרות המקורי. אין שינוי בשאר השרותים של פסיקה 16h.

הנח כי טבלת קודי ה-ascii שבתוקף היא זו שבחוברת הקורס (עמ' 337-340 במהדורה הרביעית). שגרת השרות החדשה מתחילה בתווית newInt16. הנח כי וקטור הפסיקה של שגרת השרות המקורית נשמר במשתנה oldInt16 שמוגדר בסגמנט הקוד.

oldInt16 dd ?
newInt16:

```

    cmp ah, 0
    jne exit
    cmp al, 'A'
    jbe exit
    cmp al, 'a'
    jae exit
    cmp al, '0'
    jae lowCase
    cmp al, '9'
    jbe lowCase
    jmp exit
lowCase: sub al, 20h
    jmp exit
UpperCase: add al, 20h
    jmp exit
exit:

```

lowCase: sub al, 20h
UpperCase: add al, 20h
exit: oldInt16

הנח כי וקטור הפסיקה של שגרת השרות המקורית נשמר במשתנה oldInt16 שמוגדר בסגמנט הקוד.

הנח כי וקטור הפסיקה של שגרת השרות המקורית נשמר במשתנה oldInt16 שמוגדר בסגמנט הקוד.

ארגון המחשב ושפת סף (203.1130)

סמסטר ב' תשס"ג
בחינה סופית - מועד ב'

הוראות לנבחן:

- משך הבחינה שלוש שעות.
- מותר להשתמש בכל חומר עזר, למעט מחשבים ומחשבוניס מכל סוג.
- יש להשיב על כל השאלות.
- יש לרשום את התשובות בגוף השאלון במקומות המיועדים לכך.
- נא לכתוב בכתב יד ברור ונקי. מומלץ להשתמש בעפרון ומחק.
- בשאלון זה 15 דפים, כולל דף זה. ודא כי כל הדפים נמצאים.

ב ה צ ל ח ה !

ציון	ניקוד	
17	25	שאלה 1
16	25	שאלה 2
22	25	שאלה 3
18	25	שאלה 4
73	100	סה"כ

שאלה מס' 1 (25 נקודות)

א. נתון שדגל הגלישה כבוי (of=0), והאוגר al מכיל את הערך 128-. עבור כל אחת מהפקודות שלהלן, ציין מתי היא גורמת להדלקת דגל הגלישה (of=1): תמיד, לפעמים כן ולפעמים לא, או אף פעם. הקף בעיגול את התשובה הנכונה.

neg al	תמיד / לפעמים / אף פעם
add al,al	תמיד / לפעמים / אף פעם
sub al,al	תמיד / לפעמים / אף פעם
add al,bl	תמיד / לפעמים / אף פעם
rcr al,1	תמיד / לפעמים / אף פעם
sar al,1	תמיד / לפעמים / אף פעם
shl al,1	תמיד / לפעמים / אף פעם
sahf	תמיד / לפעמים / אף פעם

$$-128 + -128 = 0$$

$$-\frac{1}{2}$$

ב. באסמבלר של שני מעברים, אילו מהמשימות שלהלן מבוצעות במעבר הראשון? תתכן יותר מתשובה אחת נכונה. הקף בעגול את כל התשובות הנכונות.

- פריסת הקריאות למאקרו. (✓)
- הכנת קובץ קוד המכונה (object module).
- ניפוי שגיאות. (✓)
- פתרון ההתייחסויות לסמלים חיצוניים.
- בנית טבלת הסמלים. (✓)

preprocessor לסימנים (34/27)

$$-\frac{1}{2}$$

ג. לאילו מטרות משמשת ההנחיה model של האסמבלר? תתכן יותר מתשובה אחת נכונה. הקף בעגול את כל התשובות הנכונות.

- לציין את דגם המעבד שעליו אמורה התכנית לרוץ.
- לאפשר כתיבת תכניות גדולות התופטות יותר מסגמנט קוד אחד. (✓)
- לקבוע את הגודל המקסימאלי של הסגמנטים. (✓)
- להרשות שימוש באוגרים הרב תכליתיים המורחבים (ברוחב 32 ביטים).

$$-\frac{1}{4}$$

ד. עבור כל אחד משלושת קטעי הקוד שלהלן, רשום את תוכנו של האוגר dx בגמר ביצוע הקטע. רשום את התשובה בבסיס 10.

<pre>.data x db '1111222333334400',0 .code xor dx,dx mov di,@data mov es,di lea di,x cld mov cx,-1 a: inc dx mov ax,es:[di] repe scasd sub di,2 cmp byte ptr es:[di],0 ja a</pre>	<pre>xor dx,dx xor ax,ax mov bx,sp mov cx,100h mov es,ax mov word ptr es:4,offset y mov es:6,cs b: inc dx mov sp,bx push cx popf y: inc al jns b</pre>	<pre>xor dx,dx xor cl,cl mov ax,5555h mov cl,33 c: inc dx sub cl,2 ror ax,cl jno c</pre>
dx= _____	dx= _____	dx= _____

11
12
24
-16

A 10
B 11
C 12
D 13
E 14
F 15

-3-

שאלה מס' 1 (המשך)

ה. תרגם את הכתובת הלוגית (segment:offset) שלהלן לכתובת פיזית, כשהמעבד פועל במצב real. רשום את התוצאה בבסיס 16.

$$\begin{array}{r} 4d750 \\ + 0b10c \\ \hline 5885c \end{array}$$

4d75h:0b10ch

5	8	8	5	c
---	---	---	---	---

א. בצע את פעולות התיבור והחיסור שלהלן בשיטת המשלים ל-2 ברוחב של 16 ביטים. כל המספרים נתונים בבסיס 16. רשום גם את התוצאות בבסיס 16. ציין את ערכי הדגלים CF ו-OF בגמר כל פעולה, כפי שהיו נקבעים על ידי ביצוע במעבד x86.

$\begin{array}{r} 6f8f \\ - 7274 \\ \hline f01b \end{array}$ <p>CF = 1 OF = 0</p>	$\begin{array}{r} 6f8f \\ + 7274 \\ \hline e203 \end{array}$ <p>CF = 0 OF = 1</p>	$\begin{array}{r} df67 \\ - 9505 \\ \hline 4a62 \end{array}$ <p>CF = 0 OF = 0</p>	$\begin{array}{r} df67 \\ + 9505 \\ \hline 746c \end{array}$ <p>CF = 1 OF = 1</p>
---	---	---	---

א. תרגם את המספרים שבטבלה מבסיס 10 לייצוג סטנדרטי בשיטת הנקודה הצפה בבסיס 2. רשום את החזקה ללא bias (ראה דוגמא).

Decimal	Sign	Exponent	Mantissa
9.0	0	+3	1.001
24.125	0	-4	1.1000001
-0.25	1	-2	1.0

1.1000.001

-0.01

ה. להלן משתנים של תכנית בשפת אסמבלי, המכילים ערכים בשיטת הנקודה הצפה. תרגם את הערכים לבסיס 10 בייצוג ללא חזקה (ראה דוגמא). תזכורת: בשיטת הנקודה הצפה, שדה החזקה הוא מספר ללא סימן הכולל bias.

float1 dd 0c1180000h

float2 dd 41c10000h

float3 dd 0000000000000000h

-9.5

+ 24.125

-0.25

$$\begin{array}{r} 128 \\ + 3 \\ \hline 131 \\ - 127 \\ \hline 4 \end{array}$$

$$01000001110000010000000000000000$$

$$11000001 \times 2^4 = 11000.001$$

$$10111111111010000000000000000000$$

$$\begin{array}{r} 1023 \\ - 2 \\ \hline 1021 \\ - 1020 \\ \hline 1 \end{array}$$

$$100 \times 2^{-2} = 0.01$$

שאלה מס' 2 (25 נקודות)

להלן מספר הגדרות:

• "ריצה" היא רצף של בתים שכולם מכילים את אותו הערך. אורך הריצה הוא מספר הבתים בריצה. התחלת הריצה היא הבית שבכתובת הנמוכה ביותר בריצה.

• לצרכי שאלה זו, מחרוזת היא רצף של בתים המסתיים בבית שמכיל את הערך 0. אורך המחרוזת הוא מספר הבתים, לא כולל הבית המסיים. דהיינו, זוהי מחרוזת כמו בשפת C

לדוגמא, נתונה המחרוזת הבאה: `string1 db 'xxxxxxxxxxxxxxxxxy1111xx',0`

בדוגמא זו, הרצף '1111' הוא ריצה באורך 4, הרצף 'xxx' הוא ריצה באורך 3, והתו 'y' הוא ריצה באורך 1. לעומת זאת, הרצף 'xy1' אינו מהווה ריצה. הרצף 'xxxxxxxxxxxxxxxx' הוא ריצה באורך 15, וזוהי הריצה הארוכה ביותר המתחילה בכתובת `string1`.

א. כתוב בשפת אסמבלי שגרה בשם `getrun`, המקבלת כפרמטר באוגרים `es` ו-`di` כתובת לוגית מלאה (סגמנט והיסט). השגרה מחזירה באוגר `al` את האורך של הריצה הארוכה ביותר שמתחילה בכתובת `[di]` ב-`es`. האורך מוחזר בייצוג ללא סימן. אם אורך הריצה גדול מ-255, מוחזר הערך 255.

הקפד על כללי התכנות הנכון של שגרה.

```
getrun proc
    push bp
    mov bp, sp
    push bx
    push cx
    push si
    push dx
    mov cx, 255
    xor bx, bx
    xor ax, ax
    xor dx, dx
    mov si, di
    f1:
        inc di
        cmp byte ptr [esi+di], 0
        je exit
        mov al, [esi+di]
        mov dl, 1
        f2:
            cmp dl, bl
            jle f3
            mov bl, dl
        f3:
            inc si
            mov di, si
            cmp byte ptr [esi+di], 0
            je exit
            loop f1
    exit:
        mov al, bl
        pop dx
        pop si
        pop cx
        pop bx
        ret
endp
```

סימנים: זיגן, סימנים, סימנים, סימנים

המתחילה כתובת `es:di`

2

שאלה מס 2 (המשך)

ב. נתונה השגרה compress שכותרתה (בשפת C):

```
unsigned int compress(char *inStr, char *outStr)
```

הפרמטר inStr הוא מצביע למחרת בסגמנט הנתונים. הפרמטר outStr הוא מצביע לחוצץ בסגמנט הנתונים, שגודלו בלתי מוגבל (כלומר, מניחים כי יש תמיד מספיק מקום בחוצץ).

השגרה compress דוחסת את המחרות inStr. נוצרת מחרת חדשה (מחרת דחוסה), שהיא קצרה יותר מ-inStr, ובנויה באופן שמאפשר לשחזר ממנה את המחרות המקורית. השגרה כותבת את המחרות הדחוסה לתוך החוצץ outStr, ומחזירה את אורך המחרות הדחוסה.

הדחיסה נעשית בשיטה הבאה. ריצה באורך n בתים ($n \leq 255$) מוחלפת בשני בתים כדלקמן: הבית הראשון מכיל את התו שמרכיב את הריצה, והבית השני מכיל את המספר n (בייצוג ללא סימן).

לדוגמא, אם נדחס את המחרות string1 שלהלן לפי שיטה זו, נקבל מחרות הזזה ל-string2.

```
string1 db 'xxxxxxxxxxxxxxxxxy1111x',0
string2 db 'x',15,'y',1,'1',4,'x',2,0
```

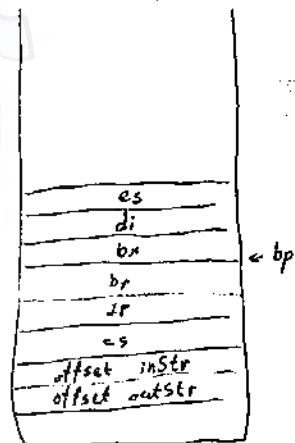
בדוגמא זו, אורך המחרות המקורית string1 הוא 22, ואילו אורך המחרות הדחוסה string2 הוא 8.

להלן מימוש השגרה compress בשפת אסמבלי. הפרמטרים מועברים לפי המוסכמות של שפת C. השגרה משתמשת בקריאה לשגרה getrun מסעיף א'.

```

1  compress  proc
2              push  bp
3              mov   bp,sp
4              push  bx
5              push  di
6              push  es
7
8              push  ds
9              pop   es          es = ds
10             mov   bx,[bp+6]    bx = offset outStr
11             mov   di,[bp+4]    di = offset inStr
12
13             cmp   byte ptr [di],0
14             je    exitcomp
15
16             mov   al,[di]      ; מציאת תו
17             mov   [bx],al      ; כתיבת תו
18             call  getrun       ; קריאת ריצה
19             mov   [bx+1],al    ; כתיבת אורך
20             add   bx,2         ; מעבר למיקום הבא
21             xor   ah,ah
22             add   di,ax        ; מעבר למיקום הבא
23             jmp   nextrun
24
25             mov   byte ptr [bx],0
26             mov   ax,bx
27             sub   ax,[bp+6]    ; חישוב אורך המחרות הדחוסה
28
29             pop   es
30             pop   di
31             pop   bx
32             pop   bp
33             ret
34             endp

```



שאלה מס 2 (המשך)

- i. כתוב בשפת אסמבלי קטע קוד שקורא לשגרה compress כדי לדחוס את המחרוזת string שלהלן לתוך החוצץ buffer. לאחר מכן, קטע הקוד מדפיס בצורה נאה את אורך המחרוזת הדחוסה, בעזרת קריאה לשגרה printf של שפת C. הגדר בסגמנט הנתונים את מחרוזת הפורמט המועברת ל-printf.

```
.data
string      db "aaaaaabbcbcdaa", 500 dup('e'), 0
buffer      db 100 dup (?)
outstring   db '%c; %d, %c, %d, %c, %d, %c, %d, %c, %d, %c, %d, %c, %d', 13, 10, '#'
```

```
.code
→ mov ax, @data
→ mov ds, ax
extern printf → push offset buffer
               push offset string
               call compress
               add sp, 2
               push offset outstring
               call _printf
               add sp, 4
               mov ah, 4ch
               int 21h
               end
```

י' (הקודם)
אורך המחרוזת



- ii. רשום את תוכן המחרוזת הדחוסה שנוצרת על ידי השגרה compress בקריאה שבסעיף i. השתמש במבנה הדומה למחרוזת הדחוסה string2 מהדוגמא בדף הקודם.

'a', 4, 'b', 3, 'c', 1, 'd', 1, 'e', 2, 255, 'e', 255, 0 ✓

- iii. מה עושה השגרה compress כאשר המחרוזת המועברת בפרמטר inStr היא באורך 0?

אלו 0 ב- outStr ומחזיר ב- ax 0 (אין מחרוזת) ✓

- iv. במקרים מסוימים, אורך המחרוזת הדחוסה שנבנית על ידי השגרה compress גדול מאורך המחרוזת המקורית. הסבר מה מאפיין את תוכן המחרוזת המקורית במקרים אלו.

כאשר יש אופי לקבוצה, כל מה שהיה מחרוזת מקורית, יש לה 2 ב- outStr ✓

- v. ציין אילו שנויים יש לבצע בקוד השגרה compress כדי שניתן יהיה לקרוא לה מתוך תכנית שכתובה בשפת C.

```
public _compress
_compress proc
```

בסוף:
אם מציין:

שאלה מס 2 (המשך)

ג. נתונה השגרה `uncompress` שכותרתה (בשפת C):

```
unsigned int uncompress(char *inStr, char *outStr)
```

הפרמטר `inStr` הוא מצביע למחרוזת בסגמנט הנתונים, והפרמטר `outStr` הוא מצביע לחוצץ בגודל בלתי מוגבל בסגמנט הנתונים. השגרה מבצעת טרנספורמציה הופכית לזו של השגרה `compress` מסעיף ב'. כלומר, השגרה `uncompress` מקבלת מחרוזת דחוסה `inStr`, משחזרת ממנה את המחרוזת המלאה המקורית, וכותבת את המחרוזת המלאה אל החוצץ `outStr`. השגרה מחזירה את אורך המחרוזת המלאה.

i. לא כל מחרוזות נתונה יכולה לשמש כפרמטר חוקי `inStr` לשגרה `uncompress`. הסבר טענה זו.

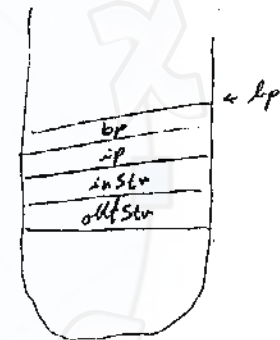
לחצוץ - מלאה מאלוים גודל 2 - לזכור לך יעלה דחוסה - אצ אצ כל יעצ חוצץ קטנים compress
ועל אם מחרוזת - כדחוסה לך כחץ כדחוסה - לא ס' לך כל טעם דחוסה - לך לא דחוסה.



ii. כתוב את השגרה `uncompress` בשפת אסמבלי. העבר את הפרמטרים לפי המוסכמות של שפת C. חובה להשתמש בפקודה `rep stosb` לבניית המחרוזת המלאה. אפשר להניח שהמחרוזת `inStr` היא חוקית. הקפד על כללי התכנות הנכון של שגרה.

```
public _uncompress
_uncompress proc
    push bp
    mov bp, sp
    push cx
    push dx
    push di

    push es
    xor ax, ax
    mov dx, [bp+4]
    mov di, [bp+6]
    mov cx, [bp+1]
    mov al, [bx]
    mov dx, ds
    mov es, ds
    rep stosb
    add bx, 2
    cmp byte ptr [bx], 0
    je exit
    jmp f1
exit:
    mov byte ptr [di], 0
    dec di
    push ax, [di]
    sub ax, [bp+6]
    pop es
    pop di
    pop dx
    pop cx
    pop bp
    ret
endp
```



אם נה? cmp byte ptr, 0
je exit



המשך שאלה חז' 2 דרך הרג

pop cx
pop bp
ret
endp

שאלה מס 2 (המשך)

ד. i. כתוב שגרה בשם average שמקבלת כפרמטרים במחשנית שני מספרים x, y מטיפוס ממשי, המיוצגים בשיטת הנקודה הצפה ברוחב 32 ביטים. השגרה מחזירה באוגר ST(0) את הממוצע של שני המספרים, כלומר את הערך $(x+y)/2$. הקפד על כללי התכנות הנכון של שגרה.

average Proc

fadd

fld 2,0

fexch st(1)

fldib

ret

endp

st 0



ii. בסגמנט הנתונים מוגדרים שלושה משתנים, num1, num2, ו-res, כולם מטיפוס ממשי בנקודה צפה, ברוחב 32 ביטים. כתוב קטע קוד הקורא לשגרה average עם הפרמטרים num1 ו-num2, ומציב את הערך המוחזר מן השגרה לתוך המשתנה avg. הקפד על כללי התכנות הנכון של קריאה לשגרה.

.data

num1 dd 12.34

num2 dd 56.78

avg dd ?

.code

finit

fld num1

fld num2

call average

fst avg

mov ah, 4ch

int 21h

end



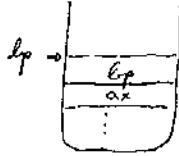
שאלה מס' 3 (25 נקודות)

א. כידוע, במעבד 8086, פקודת המכונה `push` אינה יכולה לקבל אופרנד מייד (קבוע). הגדר מאקרו בשם `pushw` שמקבל כפרמטר קבוע בגודל מילה. המאקרו מכניס את הקבוע לראש המחסנית. על המאקרו לעבוד נכון במעבד 8086. בדומה לפקודה `push`, אסור למאקרו לשנות את הדגלים.

`pushw macro x`

`push ax`
`mov ax, x`
`push bp`
`mov bp, sp`
`xchg ax, [bp+2]`
`pop bp`

`endm`



ב. בתכנון הדור הבא של מעבדי x86 הוחלט לוותר על מרבית פקודות ההסתעפות המותנית, זאת כדי לפנות קודי פעולה (opcode) עבור פקודות חדשות מתקדמות. נותרו רק ההסתעפויות המותנות שפעולות לפי דגל בודד, כדלקמן: `jno`, `jo`, `jnc`, `jnc`, `jns`, `js`, `jnz`, `jz`. כדלקמן: `jz`, `jnz`, `je`, `jne`, `jbe`, `jnb`, `jle`, `jge`.

הגדר מאקרו בשם `jle` שמקבל פרמטר יחיד הזהה לאופרנד של פקודת המכונה `jle`. המאקרו מבצע הסתעפות מותנית לפי ההגדרה של הפקודה `jle`. על המאקרו לעבוד נכון במעבדים מהדור הבא. בדומה לפקודת המכונה `jle`, אסור למאקרו לשנות את הדגלים.

`jle macro dest`

`local end, ck1, ck2, doj`

`push ax`

`pushf`

`pushf`

`pop ax`

`je doj` ← $ZF=0$ כל N ו LE $ZF=1$ 31 1000 1000 1000 0000

`and ax, 0000 1000 1000 0000b`

`test ax, 0` ← 31 1000 1000 0000 1000 1000 0000 0000

`je end` ← 31 1000 1000 0000 1000 1000 0000 0000

`test ax, 0000 1000b`

`je ck1`

`je ck2`

`ck1: test al, 1000 0000b`

`jnz doj`

`je end`

`ck2: test al, 1000 0000b`

`je doj`

`jnz end`

`doj: popf`

`pop ax`

`jmp dest`

`end: popf`

`pop ax`

`endm`

`jle (SF < OF) or (ZF = 1)`

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

$= \text{NOT}((SF < OF) \text{ and } (ZF = 0))$

ג. הגדר מאקרו בשם `addm` אשר כותבת נתונה להלן.

`addm macro x1, x2, x3, x4, x5, x6, x7, x8`

הפרמטר `x1` זהה לאופרנד הראשון של פקודת המכונה `add`. הפרמטרים האחרים זהים לאופרנד השני של הפקודה `add`. הנת כי הפרמטר `x1` תמיד מועבר בקריאה למאקרו. לגבי הפרמטרים האחרים, יתכן וחלקם לא מועברים בקריאה למאקרו.

המאקרו מבצע חיבור של כל הפרמטרים שהועברו בפועל בקריאה, ומציב את התוצאה בפרמטר `x1`. הנת כי כל המחברים באותו הרוחב.

יש לממש את המאקרו באופן יעיל, תוך שימוש במבני אסמבלי מותנה מתאימים. אין צורך לטפל במקרים מיוחדים, כגון גלישה באמצע שרשרת פעולות החיבור.

`addm macro x1, x2, x3, x4, x5, x6, x7, x8`

`irp arg, <x2, x3, x4, x5, x6, x7, x8>`

`ifnb <arg>`

`add x1, arg`

`endif`

`endm`

`endm`

מקרים מיוחדים: 31 1000 1000 0000 1000 1000 0000 0000

`addm [bx], [bp]`

31 1000 1000 0000 1000 1000 0000 0000

המשך שאלה מס' 3 בדף הבא

שאלה מס' 3 (המשך)

ד. כידוע, מנגנון הקישור לשגרה (הפקודות `call` ו-`ret`) אינו משמר את ערכי הדגלים, זאת לעומת מנגנון הקישור לפסיקה (הפקודות `int` ו-`iret`). ברצוננו לשכלל את מנגנון הקישור לשגרה, כך שהדגלים יישמרו בעת הקריאה לשגרה, וישוחזרו עם החזרה מהשגרה.

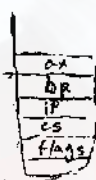

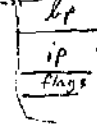
לשם כך נגדיר זוג מאקרו-ים בשם `callf` ו-`retf`. המאקרו `callf` מקבל פרמטר יחיד, הזהה לפרמטר של פקודת המכונה `call`. המאקרו `retf` הוא ללא פרמטרים. זוג המאקרו-ים מוגדר כך שהשימוש בהם יחדיו, כתחליף לזוג פקודות המכונה `call` ו-`ret`, יגרום לאפקט המבוקש של שימור הדגלים.

בטבלה שלהלן ארבע גרסאות שונות של הגדרת המאקרו `callf`. עבור כל גרסא, עליך להגדיר גרסא מתאימה של המאקרו `retf`. רשום כל הגדרה של `retf` מתחת לבן הזוג `callf` המתאים.

ראה דוגמא בטבלה.

אין לשנות את הגרסאות הנתונות של המאקרו `callf`.

הנח כי הקריאה לשגרה והשגרה עצמה נמצאים באותו סגמנט הקוד.

<pre>callf macro dest pushf call dest popf endm</pre>	<pre>callf macro dest pushf call far ptr dest endm</pre>	<pre>callf macro dest local lbl push offset lbl pushf jmp dest lbl: endm</pre>	<pre>callf macro dest pushf call dest endm</pre>
<pre>retf macro ret endm</pre>	<pre>retf macro push bp mov bp, sp push ax xchg ax, [bp+4] xchg ax, [bp+6] xchg ax, [bp+2] xchg ax, [bp+3] pop ax pop bp popf ret endm</pre> 	<pre>retf macro popf pop bx jmp [bx] endm</pre>  <p>ss = 3000, cs = 3000, bx = 2000</p>	<pre>retf macro push bp mov bp, sp push ax xchg ax, [bp+2] xchg ax, [bp+4] xchg ax, [bp+3] pop ax pop bp popf ret endm</pre> 

$ax = sp$ $[bp+2] = ax$
 $ax = flags$ $[bp+4] = sp$
 $ax = ax$ $[bp+3] = flags$

שאלה מס' 3 (המשך)

ה. להלן הגדרת המאקרו `mystery`. הפרמטרים x ו- y הם מספרים שלמים.

```

1  mystery macro x,y
2      if x lt 0
3          mystery -x/2,y
4          exitm
5      endif
6      n = y
7      rept x
8          if (n gt 16000) or (n lt -16000)
9              exitm
10             elseif (n le 127) and (n ge -128)
11                 db n
12             else
13                 dw n
14             endif
15             n = -n*2
16         endm
17     endm

```

i. להלן ארבע דוגמאות של קריאות למאקרו `mystery`. לכל דוגמא, רשום את הקוד בשפת אסמבלי שמתקבל לאחר פריסת הקריאה למאקרו. יש לרשום רק שורות שיוצרות קוד מכונה (אין לרשום הנחיות לאסמבלי מותנה, הצבות במשתני אסמבלר וכד').

mystery 6,10	mystery -9,-2	mystery 1000,-6000	mystery -1,3
db 10 db -80 db 40 db -80 dw 160 dw -320	db -2 db 4 db -8 db 16	dw -6000 dw 12,000	15 10 5 0

ii. רשום קריאה למאקרו `mystery` אשר גורמת לפריסת 100 בתים, שכולם מאותחלים לערך 0.

mystery 100,0

שאלה מס' 4 (25 נקודות)

לצורך בדיקות של תוכנה, נוח לפעמים לבצע סימולציה של קלט מהמקלדת על ידי קריאת תווים מהזכרון, במקום לעבוד עם המקלדת עצמה.

למימוש מנגנון הסימולציה, נקצה בזיכרון חוצץ אשר בו ימצאו תווי הקלט, ונשנה את שגרת השרות של פסיקת שרותי המקלדת 16h, כך שתשתמש בתווים מחוצץ זה במקום מהמקלדת.

כדי לאפשר יתר גמישות, חוצץ הסימולציה אינו קבוע, אלא ניתן לשינוי על ידי המשתמש. הגישה לתו הבא בחוצץ היא דרך מצביע מלא (סגמנט והיסט), הרשום במקום קבוע ומוסכם, בכתובת הפיזית 822h. מספר התווים שנותרו בחוצץ רשום גם הוא במקום קבוע ומוסכם, בכתובת הפיזית 820h, בייצוג ללא סימן ובגודל מילה. בעת אתחול מנגנון הסימולציה, המילה בכתובת הפיזית 820h מאותחלת ל-0.

הסימולציה מתבצעת על ידי שרות מס' 0 של פסיקה 16h (כלומר כשנכנסים לפסיקה עם ah=0). השרות מחזיר בכל פעם את התו הבא בחוצץ הסימולציה, ומעדכן את המצביע לתו הבא ואת מספר התווים שנותרו. התו מוחזר באוגר al. מכיוון שקוד ה-scan של המקש במקלדת אינו נגיש, מוחזר באוגר ah הערך 0.

במקרה שחוצץ הסימולציה התרוקן, השרות עובר לבצע קלט רגיל מן המקלדת האמיתית. השרות יחזור לבצע סימולציה אם וכאשר המשתמש יקצה חוצץ חדש (על ידי עדכון המצביע ומספר התווים).

להלן שגרת השרות החדשה של פסיקה 16h, המממשת את הסימולציה. השגרה מתחילה בתווית newInt16. הנח כי וקטור הפסיקה של שגרת השרות המקורית נשמר במשתנה oldInt06 שמוגדר בסגמנט הקוד.

```

1  inLen      equ 820h
2  inPtr      equ 822h

3  oldInt16   dd ?

4  newInt16:  push ds
5             push es
6             push bx

7             xor bx,bx
8             mov es,bx

9             cmp ah,0
10            je new0

11 chain16:   pop bx
12            pop es
13            pop ds
14            jmp oldInt16

15 new0:      cmp word ptr es:inLen,0
16            je chain16

17            lds bx,es:inPtr
18            mov al,ds:[bx]

19            mov ah,0

20            dec word ptr es:inLen
21            inc word ptr es:inPtr

22 exit16:    pop bx
23            pop es
24            pop ds
25            iret

```

int 16h - אמצעי
 : ah = 0 - סימולציה
 : ah != 0 - קלט רגיל

0 = מספר תווים
 : newInt16 - מצביע

→ 18 - ds = seg
 → 18 - bx = offset

→ 19 - scan = 0

→ 20 - inLen = מספר תווים
 → 21 - inPtr = כתובת

→ 24 - ds

המשך שאלה מס' 4 בעמוד 13

i. מה עושות שורות 10-19? באילו מקרים מתבצעת ההסתעפות בשורה 10?

ii. מה עושה שורה 14? כיצד תוודים משגרת הפסיקה כאשר מתבצעת שורה זו?

iii. להיכן מצביע האופרנד es:inLen בשורה 15?

iv. מה עושות שורות 15-16? באיזה מקרה מתבצעת ההסתעפות בשורה 16? מה קורה לגבי הקלט כאשר מתבצעת הסתעפות זו?

5. מה עושות שורות 17-18? מהו תוכן האוגר al אחרי בצוע שורה 18?

מכתב זה נשלח אל הנהלת המוסד לביטוח לאומי, תל אביב, ב-15.12.2018.

[illegible]

שאלה מס' 4 (המשך)

ב. כתוב קטע קוד בתכנית המשתמש, שמאתחל את חוצץ הסימולציה למחרת inputBuf, המוגדרת בסגמנט הנתונים להלן.

```
.data
inputBuf db "this is just some arbitrary input"
bufLen dw bufLen-inputBuf
```

```
.code
xor bx, bx
mov es, bx
mov ax, bufLen
mov es:inLen, ax
mov ax, seg inputBuf
mov [es:inPtr+2], ax
mov ax, offset inputBuf
mov es:inPtr, ax
```

הנחיה: $bufLen - inputBuf$ זהו
האורך של `inputBuf` ב-`ds`

ג. האם יכול המשתמש להפסיק בכל עת את הסימולציה, ולעבור לקלט רגיל מן המקלדת? הסבר.

כן, ניתן לעצור את ה-`counter` באמצעות `push ax`, `mov ax, 0`, `mov [es:inLen], ax`, `pop ax`.
(הערה: `push ax` מוסיף 2 בייט ל-`inLen`)

ד. ברצוננו להרחיב את הסימולציה גם לשורות מס' 1 של פסיקה 16h. כזכור, שורות זה בודק האם קיים תו בקלט, אך אינו מוציא את התו מהקלט. גם בשורות זה, כמו בשורות מס' 0, נשתמש בחוצץ הסימולציה. במקרה שחוצץ הסימולציה התרוקן, השורות עובר לבדיקת הקלט מהמקלדת האמיתית. לפיכך, השורות יציין שאין קלט רק כאשר גם חוצץ הסימולציה וגם חוצץ המקלדת ריקים.

עליך לממש את שורות מס' 1 החדש של פסיקה 16h. רשום להלן את התוספות לקוד האסמבלי של שגרת הפסיקה. עליך למספר את השורות הנוספות בהתאמה למספרי השורות הקיימות. לדוגמא: שורות שיתווספו אחרי שורה 10 ימוספרו 10.1, 10.2 וכד'. אם יש צורך לשנות או למחוק שורה קיימת, רשום את מספר השורה ואת קוד האסמבלי החדש שלה.

```
10.1 cmp ah, 1
10.2 je new1
```

```
14.1 new1: cmp word ptr es:inLen, 0
14.2       je chain16
14.3       lds bx, es:inPtr
14.4       mov al, ds:[bx]
14.5       mov ah, 0
14.6       jmp exit16
```

הוסף סימון
ZF=0

-2-

שאלה מס' 4 (המשך)

ה. קצב הקלט מהוצא הסימולציה מהיר בהרבה מקצב הקלט מהמקלדת האמיתית. כדי לדמות את המצב האמיתי יותר טוב, נוסיף לסימולציה קוצב זמן. הקוצב יגרום לכך שיחלפו בממוצע 200 מילי-שניות מרגע הוצאת תו מחוצא הסימולציה ועד שניתן יהיה להוציא את התו הבא.

למימוש קוצב הזמן, נגדיר דגל בגודל בית בכתובת הפיזית 826h. פסיקת שרותי השעון 1ch תרים את הדגל (תציב בו 1) כל 200 מילי-שניות. במקביל, שרות מס' 0 של פסיקה 16h ימתין בלולאה כל עוד הדגל אינו מודם, ולפני החזרה מן הפסיקה יוריד את הדגל ל-0.

להלן תוספת לקוד השגרה החדשה של פסיקה 16h, למימוש קוצב הזמן. מספרי השורות הנוספות הם בהתאמה למספרי השורות הקיימות.

```
16.1 inReady equ 826
16.2          sti
16.3 delay:   cmp byte ptr es:inReady,1
16.4          jne delay
16.5          mov byte ptr es:inReady,0
```

עליך לממש את שגרת השרות החדשה של פסיקת שרותי השעון 1ch, לפי התיאור לעיל. השגרה מתחילה בתווית newIntlc. הנח כי וקטור הפסיקה של שגרת השרות המקורית נשמר במשתנה oldIntlc שמוגדר בסגמנט הקוד.

```
.code
oldIntlc dd ?

newIntlc:
```