

ארגון המחשב ושפת סף (203:1130)

סמסטר ב' תשס"ה  
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 14 דפים, כולל דף זה. ודא כי כל הדפים נמצאים.

ב ה צ ל ח ה !

ציון	ניקוד	
21	25	שאלה 1
19	25	שאלה 2
19	25	שאלה 3
17	25	שאלה 4
76	100	סה"כ



שאלה מס' 1 (25 נקודות)

א. לגבי סגמנטים בתכנית שרצה על מעבד X86 במצב real, אילו מהטענות הבאות נכונות? תתכן יותר מתשובה אחת נכונה. הקף בעגול את כל התשובות הנכונות.

- i. כתובת התחלת סגמנט הקוד מיושרת (aligned) על כפולה של 16.
- ii. בתכנית שרצה על מעבד 8086 יש לכל היותר ארבעה סגמנטים.
- iii. הגודל המקסימלי של סגמנט הוא  $2^{20}$  בתים.
- iv. כתובות התחלת הסגמנטים ידועות בזמן הקישור של התכנית.
- v. בכל תכנית יש סגמנט מחסנית אחד ויחיד.
- vi. כל סגמנט חייב להיות חלק מקבוצה (group).

-2/3

ב. לגבי תוויות (labels) בקוד שכתוב בשפת אסמבלי של X86, אילו מהטענות הבאות נכונות? תתכן יותר מתשובה אחת נכונה. הקף בעגול את כל התשובות הנכונות.

- i. תווית חייבת להיות מוגדרת לפני השימוש הראשון בה.
- ii. לכל תווית יש ערך מספרי.
- iii. תווית אינה יכולה להיות מוגדרת יותר מפעם אחת בכל קובץ מקור של התכנית.
- iv. התוויות משמשות בתהליך האסמבלי בלבד.
- v. באופרנד של פקודה (בשיטת המיעון) יכולה להופיע לכל היותר תווית אחת.
- vi. בקוד אסמבלי אין אבחנה בין אותיות קטנות וגדולות בשמות התוויות.

-2/2

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

<pre>.data list db 0,1,2,3,4,5,6,7,8,9,10,11,12,13  .code xor dx,dx lea bx,list mov sp,bx add bx,13 mov cx,7 mov ax,@data mov ds,ax mov ss,ax c: dec bx pop ax cmp ax,[bx] loopne c mov dx,cx</pre>	<pre>xor dx,dx xor bx,bx b: inc dx dec bl push bx popf jbe b</pre>	<pre>xor dx,dx n = 4 rept n local a mov cx,n a: inc dx n = n - 1 loop a endm</pre>
dx = <u>2</u>	dx = <u>64</u>	dx = <u>10</u>

x

-2/7

המשך שאלה מס' 1 בדף הבא

# שאלה מס' 1 (המשך)

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

9c660 h  
00b25 h  
9d185

9c66h:0b25h

9d185

✓ 11

ה. בצע את פעולות החיבור והחסור 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.

9c  
abbb  
+  
3562  
E1AD

CF= 0

OF= 0

abbb  
-  
3562  
7657

CF= 0

OF= 1

5f63  
+  
b2c2  
1225

CF= 1

OF= 0

5f63  
-  
b2c2  
ACAT

CF= 1

OF= 1

✓ 8/8

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
-51.625	1	+5	1.1001101
0.21875	0	-3	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, שדה החזקה הוא מספר ללא סימן הכולל bias.

float1 dd 0c1180000h

-9.5

float2 dd 0c2568000h

-53.625

double1 dq 3fe3000000000000h

+0.59375

✓ 2/2



## שאלה מס' 2 (25 נקודות)

שפת המכונה של המעבד x86 כוללת פקודות להזזה וסיבוב של אופרנד באורך 1, 2 או 4 בתים. אין בשפת המכונה פקודות להזזה וסיבוב של רצף בתים באורך אחד. שאלה זו עוסקת בהזזה וסיבוב של רצף (מערך) בתים בזיכרון, באורך  $n \geq 1$  כלשהו. המגבלה היחידה על האורך  $n$  היא שרצף הבתים כולו חייב להיכלל בסגמנט אחד. בכל הסעיפים בשאלה זו אנו מניחים כי האורך  $n$  הוא חוקי, ואין צורך לבדוק זאת.

א. להלן השגרה `shr_long1` שכותרתה (בשפת C) היא:

```
void shr_long1(char * byteSequence, unsigned int n);
```

הפרמטר `byteSequence` הוא מצביע (היסט) לרצף של בתים שנמצא בסגמנט ה-`.data`. הפרמטר  $n$  הוא אורך הרצף ( $n \geq 1$ ). השגרה מזיזה ימינה בביט אחד את רצף הבתים, בדומה לפקודת המכונה `shr`. עם החזרה מהשגרה, הדגל `cf` מכיל את הביט שיצא מקצהו הימני של הרצף. אין חשיבות למצב שאר דגלי התנאי עם החזרה מהשגרה.

```
1 shr_long1 proc
2     push bp
3     mov bp, sp
4     push bx
5     push cx
6     mov bx, [bp+4]
7     mov cx, [bp+6]
8     cld
9     loop_shr: ror byte ptr [bx], 1
10    inc bx
11    loop loop_shr
12    pop cx
13    pop bx
14    pop bp
15    ret
16    endp
```

i. האם ניתן להחליף את שורה 10 בפקודה: `add bx, 1`? נמק.

לא, כי `add` יסמך את סיבוב של ה-`CF` שלילי. תוצאה של `add` תהיה `CF=0` במקום `CF=1` (כפי שצויין).

ii. מה תבצע השגרה אם נחליף את שורה 8 בצמד הפקודות הבא?

```
8.1 sar byte ptr [bx], 1
8.2 rcl byte ptr [bx], 1
```

שנים `CF` יתו עיט של הביט השמאלי. כיוון וסי' כן `CF` יתו עיט של הביט `SAR` של ימני של `CF`.

iii. נשנה את הגדרת הפרמטר `byteSequence`, כך שמעתה יהיה זה היסט לרצף בתים בתוך הסגמנט אליו מצביע האוגר `es` (במקום `ds` בהגדרה המקורית).

עליך להכניס שינויים בקוד השגרה `shr_long1` כדי להתאימה להגדרה החדשה. רשום את מספרי השורות בשגרה שיש לשנות, ואת קוד האסמבלי החדש של שורות אלה. בצע שינויים הכרחיים בלבד.

מס' שורה

קוד אסמבלי חדש

`mov esi, [bp+4]`

לא נכון

7/1

המשך שאלה מס' 2 בדף הבא

## שאלה מס 2 (המשך)

ב. כתוב בשפת אסמבלי שגרה בשם `shl_long1`, שמקבלת פרמטרים זהים לאלו של השגרה `shr_long1` מסעיף א' (ללא השינוי בתת סעיף iii). השגרה `shl_long1` מזיזה שמאלה בביט אחד את רצף הבתים שהועבר כפרמטר, בדומה לפקודת המכונה `shl`. עם החזרה מהשגרה, הדגל `cf` מכיל את הביט שיצא מקצהו השמאלי של הרצף. מותר לשגרה להרוס את שאר דגלי התנאי.

```

shl_long1 proc
    push bp
    mov bp, sp
    push bx
    push cx
    mov cx, [bp+4]
    mov cx, [bp+6]
    add bx, cx
    dec bx
    loop_shl: rcl byte ptr [bx], 1
    dec bx
    loop loop_shl
    pop cx
    pop bx
    ret
endp

```

*Handwritten notes:*  
 - `dec bx` points to the line `dec bx` in the code.  
 - `loop_shl` is written next to the `loop` instruction.  
 - `-1/2/4` is written on the right side of the code block.

ג. כתוב בשפת אסמבלי שגרה בשם `ror_long1`, שמקבלת פרמטרים זהים לאלו של השגרה `shr_long1` מסעיף א' (ללא השינוי בתת סעיף iii). השגרה `ror_long1` מסובבת ימינה בביט אחד את רצף הבתים שהועבר כפרמטר, בדומה לפקודת המכונה `ror`. עם החזרה מהשגרה, הדגל `cf` מכיל את הביט שעבר מקצהו הימני של הרצף אל קצהו השמאלי. מותר לשגרה להרוס את שאר דגלי התנאי.

חובה לממש את השגרה `ror_long1` בעזרת קריאה לשגרה `shr_long1`.

```

ror_long1 proc
    push bp
    mov bp, sp
    push bx
    push cx
    mov bx, [bp+4]
    mov cx, [bp+6]
    push cx
    push bx
    call shr_long1
    mov cx, [bp+4]
    add byte ptr [bx], 10000
    mov bx, [bp+4]
    pop cx
    pop bx
    ret
endp

```

*Handwritten notes:*  
 - `shr_long1` is circled and has an arrow pointing to it from the text "קריאה לשגרה".  
 - `add byte ptr [bx], 10000` is circled and has an arrow pointing to it from the text "הרוס את CF".  
 - `mov bx, [bp+4]` is circled and has an arrow pointing to it from the text "הרוס את CF".  
 - `mov cx, [bp+6]` is circled and has an arrow pointing to it from the text "הרוס את CF".  
 - `push cx` and `push bx` are circled and have an arrow pointing to them from the text "הרוס את CF".  
 - `pop cx` and `pop bx` are circled and have an arrow pointing to them from the text "הרוס את CF".  
 - `ret` is circled and has an arrow pointing to it from the text "הרוס את CF".  
 - `endp` is circled and has an arrow pointing to it from the text "הרוס את CF".  
 - `-1/2/4` is written on the right side of the code block.

המשך שאלה מס 2 בדף הבא

```
void shift_long(char * byteSequence, unsigned int n, int k);
```

השגרה `shift_long` מבצעת כדלקמן: אם  $k > 0$ , רצף הביתים שהועבר כפרמטר מוזז ימינה ב-  $k$  ביטים. אם  $k < 0$ , רצף הביתים מוזז שמאלה ב-  $-k$  ביטים. עם החזרה מהשגרה, הדגל `cf` מכיל את הביט האחרון שיצא מקצה הרצף (מימין או משמאל, לפי המקרה). מותר לשגרה להרוס את שאר דגלי התנאי.

publ. - shift - long;  
- shift - long piece

push bp  
mov bp, sp  
push bx  
push cx  
push ax  
mov bx, [bp+4]  
mov cx, [bp+6]  
mov cx, [bp+8]  
mov ax  
push bx

cup  $x, 0$   
 ing ~~sh~~ ie - exit  
 loop1: call sh - loop1  
 loop loop1

- shl: jmp exit  
neg cx  
call shl\_large  
trap shl

- exit:  
 jmp .exit  
 pop cx  
 pop cx  
 pop ax  
 pop cx  
 pop bx  
 pop dx  
 ret  
 endp

}  
 add sp, 4  
 pop ax

b7

$$-\frac{1}{2} \mid 6$$



## שאלה מס 2 (המשך)

ה. להלן השגרה `shr_long8`, שמקבלת פרמטרים זהים לאלו של השגרה `shr_long1` מסעיף א' (ללא השינוי בתת סעיף iii). השגרה `shr_long8` מזיזה ימינה ב-8 ביטים את רצף הבתים שמועבר כפרמטר. עם החזרה מהשגרה, הדגל `cf` מכיל את הביט האחרון שיצא מקצהו הימני של הרצף. אין חשיבות למצב שאר דגלי התנאי עם החזרה מהשגרה.

```

1  shr_long8 proc
2      push    bp
3      mov     bp, sp
4      push    8
5      push    word ptr [bp+6]
6      push    word ptr [bp+4]
7      call    shift_long
8      mov     sp, bp
9      pop     bp
10     ret
11     endp

```

i. מה עושה שורה 8?

מחזרת את הסמן למצבו המקורי (כלומר לא 1)  $CF$

ii. כדי לאפשר קריאה לשגרה `shr_long8` הן משפת C והן משפת אסמבלי באותו שם בדיוק, יש להוסיף לקוד האסמבלי של השגרה שתי שורות, לפני שורה מס' 1. רשום שתי שורות אלה.

```

public  shr_long8
shr_long8:

```

iii. השגרה `shr_long8` אינה יעילה, וניתן לכתוב שגרה יעילה יותר באופן משמעותי. נמק טענה זו.

1. כתוב מימוש אחר לשגרה `shr_long8` מסעיף ה', תוך שימוש חובה בפקודת מכונה על מחרוזות. במימוש החדש מותר לשגרה להרוס את כל הדגלים, כולל הדגל `cf`. אסור להשתמש באף פקודת מכונה להזזה או סיבוב, ואסור לקרוא לשגרות מהסעיפים הקודמים.

```

shr_long8 proc
    push    bp
    mov     bp, sp
    push    8
    push    word ptr [bp+6]
    push    word ptr [bp+4]
    mov     cx, [bp+8]
    add     dx, cx
    mov     cx, [bp+8]
    push    ds
    pop     es
    mov     di, bx
    mov     si, bx
    stc
    dec     di
    test    byte ptr [bx], 1
    jnz     1
    mov     dx, 1
    xor     dx, dx
    dec     cx
    rep     movsb

```

```

cmp     dx, 0
jnz     13
cld
jnz     14
13: stc
14: pop     cx
    pop     si
    pop     dx
    pop     es
    pop     bx
    pop     cx
    pop     bp
    jmp     endp

```

א. פוסט

(הא. גניזוני?)

### שאלה מס' 3 (25 נקודות)

א. להלן הגדרת המאקרו `xtosr` המקבל פרמטר אחד שהוא אוגר רב תכליתי בכל רוחב אפשרי. המאקרו מחליף בין תוכן האוגר המועבר כפרמטר לבין תוכן ראש המחסנית.

```
xtosr macro reg
    push    bp
    mov     bp, sp
    xchg    reg, [bp+2]
    pop     bp
endm
```

i. האם המאקרו `xtosr` עובד נכון כאשר הפרמטר המועבר למאקרו הוא האוגר `bp`? הסבר.

לא, בתחילת המקרו אין שום דבר איתו של `bp` וסעיף אותו לזמן של `sp` - לכן במקום המחסנית ציורה `sp`.

ii. הגדר מאקרו בשם `xtosbp` ללא פרמטרים, אשר מחליף בין תוכן האוגר `bp` לבין המילה שבראש המחסנית. אסור למאקרו לשנות את הדגלים. אפשר להשתמש בקריאה למאקרו `xtosr`.

```
xtosbp macro
    push    ax
    mov     ax, bp
    xosr    ax
    mov     bp, ax
    pop     ax
endm
```

להחליף את  
הראשונה  
הזל  
האפקט יורשתי  
 $bp \leftarrow ax$

-3/3

iii. איזו בעיה עלולה להיווצר כאשר הפרמטר המועבר למאקרו `xtosr` הוא האוגר `sp`? הסבר.

במקורו אין הסעה, `sp` יצאה לסוף האם המחסנית (אם סעיף לזמן הקריאה למקרו) ולכן השימוש במחסנית יסוג.

iv. איזו בעיה תיווצר כאשר הפרמטר המועבר למאקרו `xtosr` אינו אוגר רב תכליתי?

האפקט שרק מציגון לאחד מאין רב תכליתי  
עם שאר קואסיציה (מקום ה) (אחד)



המשך שאלה מס' 3 בדף הבא



### שאלה מס' 3 (המשך)

- i. הגדר מאקרו בשם `isDenorm` שכותרתו נתונה להלן. הפרמטר `float` הוא מספר ממשי בייצוג סטנדרטי בנקודה צפה ברוחב 32 ביטים. הפרמטר יכול להיות אוגר רב תכליתי של המעבד X86, או ערך בזיכרון בכל שיטת מעון אפשרית. המאקרו בודק האם המספר `float` הוא בלתי מנורמל. אם המספר בלתי מנורמל, המאקרו מדליק את הדגל `cf` באוגר הדגלים, ואחרת המאקרו מכבה את הדגל `cf`. אסור להשתמש בפקודות של המעבד המתמטי. מותר למאקרו להרוס את שאר דגלי התנאי.

```
isDenorm macro float
    local a
    local b
    mov     eax, float
    jz      a
    cld
    jmp     b
a: stc
b:
endm
```

יש להניח אם לא  
(המכונה הנכונה)

- ii. הגדר מאקרו בשם `jDenorm` שכותרתו נתונה להלן. הפרמטר `float` זהה לפרמטר של המאקרו `isDenorm`, ואילו הפרמטר `dest` זהה לאופרנד של פקודת מכונה להסתעפות מותנית. המאקרו מבצע הסתעפות מותנית לכתובת `dest` אם ורק אם המספר `float` בלתי מנורמל. חובה להשתמש בקריאה למאקרו `isDenorm`. אסור למאקרו לשנות את הדגלים.

```
jDenorm macro float, dest
    isDenorm float
    jz      dest
endm
```

- ג. הגדר מאקרו בשם `fldreg`, המקבל פרמטר אחד שהוא אוגר רב תכליתי ברוחב 32 ביטים במעבד X86. המאקרו טוען את תוכן האוגר שמועבר כפרמטר לתוך האוגר `st(0)` של המעבד המתמטי. אם הפרמטר אינו אחד האוגרים הרב תכליתיים ברוחב 32 ביטים, המאקרו אינו פורש אף פקודה בקוד המכונה.

```
fldreg macro reg
    IRP    reg, st, st(0), st(1), st(2), st(3), st(4), st(5), st(6), st(7)
        fld     reg
    ENDIRP
endm
```

יש 4 אוגרים כאלה

אוגרים  
st, st(0), st(1), st(2), st(3), st(4), st(5), st(6), st(7)

הפקודה `fld`  
לוקח מקלט  
אל אוגר

2/3



### שאלה מס' 3 (המשך)

ד. להלן הגדרת המאקרו `sum`. המאקרו מקבל לכל היותר ששה פרמטרים, כאשר כל פרמטר הוא בגודל מילה ובכל שיטת מעון. המאקרו מחשב את סכום כל הפרמטרים ומציב את התוצאה באוגר `ax`.

```

1 sum macro n1,n2,n3,n4,n5,n6
2   k = 0
3   irp n,<&n1,&n2,&n3,&n4,&n5,&n6>
4     ifidni <&n>,<ax>
5       k = k + 1
6     endif
7   endm

8   if k eq 0
9     xor ax,ax
10  elseif k ge 2
11    push dx
12    mov dx,k
13    imul dx
14    pop dx
15  endif

16  irp n,<&n1,&n2,&n3,&n4,&n5,&n6>
17    ifnb <&n>
18      ifdifi <&n>,<ax>
19        add ax,n
20      endif
21    endif
22  endm
23 endm

```

י. להלן מספר דוגמאות של קריאות למאקרו `sum`. עבור כל דוגמא, רשום את הקוד בשפת אסמבלי שמתקבל מפרישת הקריאה למאקרו. יש לרשום רק שורות שיוצרות קוד מכוּנה (אין לרשום את ההנחיות לאסמבלי מותנה, וכד').

sum bx,y,[bp+2],,4	sum bx,ax,y	sum ax,bx,ax,ax	sum bx	sum ax
<pre> xor ax,ax add ax,bx add ax,y add ax,[bp+2] add ax,4 </pre>	<pre> add ax,bx add ax,y </pre>	<pre> push dx mov dx,3 imul dx pop dx add ax,0x </pre>	<pre> xor ax,ax add ax,bx </pre>	<pre> xor ax,ax </pre>

המשך שאלה מס' 3 בדף הבא

Πολύτιμο δόρυ σε σπασμό των οφθαλμών

1000, 2000, 4000, 6000, 8000, 10000, 12000, 14000, 16000, 18000, 20000, 22000, 24000, 26000, 28000, 30000, 32000, 34000, 36000, 38000, 40000, 42000, 44000, 46000, 48000, 50000, 52000, 54000, 56000, 58000, 60000, 62000, 64000, 66000, 68000, 70000, 72000, 74000, 76000, 78000, 80000, 82000, 84000, 86000, 88000, 90000, 92000, 94000, 96000, 98000, 100000, 102000, 104000, 106000, 108000, 110000, 112000, 114000, 116000, 118000, 120000, 122000, 124000, 126000, 128000, 130000, 132000, 134000, 136000, 138000, 140000, 142000, 144000, 146000, 148000, 150000, 152000, 154000, 156000, 158000, 160000, 162000, 164000, 166000, 168000, 170000, 172000, 174000, 176000, 178000, 180000, 182000, 184000, 186000, 188000, 190000, 192000, 194000, 196000, 198000, 200000, 202000, 204000, 206000, 208000, 210000, 212000, 214000, 216000, 218000, 220000, 222000, 224000, 226000, 228000, 230000, 232000, 234000, 236000, 238000, 240000, 242000, 244000, 246000, 248000, 250000, 252000, 254000, 256000, 258000, 260000, 262000, 264000, 266000, 268000, 270000, 272000, 274000, 276000, 278000, 280000, 282000, 284000, 286000, 288000, 290000, 292000, 294000, 296000, 298000, 300000, 302000, 304000, 306000, 308000, 310000, 312000, 314000, 316000, 318000, 320000, 322000, 324000, 326000, 328000, 330000, 332000, 334000, 336000, 338000, 340000, 342000, 344000, 346000, 348000, 350000, 352000, 354000, 356000, 358000, 360000, 362000, 364000, 366000, 368000, 370000, 372000, 374000, 376000, 378000, 380000, 382000, 384000, 386000, 388000, 390000, 392000, 394000, 396000, 398000, 400000, 402000, 404000, 406000, 408000, 410000, 412000, 414000, 416000, 418000, 420000, 422000, 424000, 426000, 428000, 430000, 432000, 434000, 436000, 438000, 440000, 442000, 444000, 446000, 448000, 450000, 452000, 454000, 456000, 458000, 460000, 462000, 464000, 466000, 468000, 470000, 472000, 474000, 476000, 478000, 480000, 482000, 484000, 486000, 488000, 490000, 492000, 494000, 496000, 498000, 500000, 502000, 504000, 506000, 508000, 510000, 512000, 514000, 516000, 518000, 520000, 522000, 524000, 526000, 528000, 530000, 532000, 534000, 536000, 538000, 540000, 542000, 544000, 546000, 548000, 550000, 552000, 554000, 556000, 558000, 560000, 562000, 564000, 566000, 568000, 570000, 572000, 574000, 576000, 578000, 580000, 582000, 584000, 586000, 588000, 590000, 592000, 594000, 596000, 598000, 600000, 602000, 604000, 606000, 608000, 610000, 612000, 614000, 616000, 618000, 620000, 622000, 624000, 626000, 628000, 630000, 632000, 634000, 636000, 638000, 640000, 642000, 644000, 646000, 648000, 650000, 652000, 654000, 656000, 658000, 660000, 662000, 664000, 666000, 668000, 670000, 672000, 674000, 676000, 678000, 680000, 682000, 684000, 686000, 688000, 690000, 692000, 694000, 696000, 698000, 700000, 702000, 704000, 706000, 708000, 710000, 712000, 714000, 716000, 718000, 720000, 722000, 724000, 726000, 728000, 730000, 732000, 734000, 736000, 738000, 740000, 742000, 744000, 746000, 748000, 750000, 752000, 754000, 756000, 758000, 760000, 762000, 764000, 766000, 768000, 770000, 772000, 774000, 776000, 778000, 780000, 782000, 784000, 786000, 788000, 790000, 792000, 794000, 796000, 798000, 800000, 802000, 804000, 806000, 808000, 810000, 812000, 814000, 816000, 818000, 820000, 822000, 824000, 826000, 828000, 830000, 832000, 834000, 836000, 838000, 840000, 842000, 844000, 846000, 848000, 850000, 852000, 854000, 856000, 858000, 860000, 862000, 864000, 866000, 868000, 870000, 872000, 874000, 876000, 878000, 880000, 882000, 884000, 886000, 888000, 890000, 892000, 894000, 896000, 898000, 900000, 902000, 904000, 906000, 908000, 910000, 912000, 914000, 916000, 918000, 920000, 922000, 924000, 926000, 928000, 930000, 932000, 934000, 936000, 938000, 940000, 942000, 944000, 946000, 948000, 950000, 952000, 954000, 956000, 958000, 960000, 962000, 964000, 966000, 968000, 970000, 972000, 974000, 976000, 978000, 980000, 982000, 984000, 986000, 988000, 990000, 992000, 994000, 996000, 998000, 1000000, 1002000, 1004000, 1006000, 1008000, 1010000, 1012000, 1014000, 1016000, 1018000, 1020000, 1022000, 1024000, 1026000, 1028000, 1030000, 10320

המחיר של המוצר הוא 100 ש"ח. המחיר של המוצר הוא 100 ש"ח.

$\sim 8 \text{ ns} \rightarrow \text{odd } \text{ex. ip}$

100 04.04 1998

(הנה' יא) ארבע' יא  
 . ארבע' יא ארבע' יא  
 (א' יא ארבע' יא)



## שאלה מס' 4 (25 נקודות)

א. כידוע, שגרת השרות של פסיקה מס' 21h מספקת מספר תתי שירות שונים לביצוע קלט של תו בודד מן המקלדת. בפרט, תת שירות מס' 1 מבצע echo של התו שנקלט, ואילו תת שירות מס' 8 אינו מבצע echo.

ברצוננו לשנות את שגרת הפסיקה 21h כך שתתי השירות 1 ו-8 יתחלפו ביניהם. כלומר, לאחר השינוי, קריאה לתת שירות 1 תגרום לביצוע קלט ללא echo, ואילו קריאה לתת שירות 8 תגרום לבצוע קלט עם echo. אין שינוי בתתי שירות אחרים של פסיקה 21h.

כתוב להלן שגרת שירות חדשה לפסיקה 21h שתממש את השינוי המתואר לעיל. השגרה תתחיל בתווית newInt21. הנח כי וקטור הפסיקה של שגרת השרות הקודמת נשמר במשתנה oldInt21 בסגמנט הקוד של שגרת השירות החדשה.

.code

oldInt21 dd ?

newInt21:

pushd  
 mov ah, 8  
 mov al, 18  
 cmp al, 1  
 jbe oldInt21  
 popd

mov ah, 8  
 jmp oldInt21

mov ah, 8  
 jmp oldInt21

לשנות את AH  
 למקור

-2/6



המשך שאלה מס' 4 בדף הבא

## שאלה מס' 4 (המשך)

ב. i. כתוב שגרת שרות חדשה לפסיקת שרותי השעון (פסיקה מס 1ch), אשר תבצע כדלקמן:  
כל 5 דקות בדיוק תושמע סדרה של 10 צפופים, במרווחים של 2 שניות בין צפוף לצפוף.  
שים לב: משך הזמן בו מושמעת סדרת צפופים נכלל ב- 5 הדקות שעוברות עד לסדרה הבאה.

תזכורת: צפוף נוצר על ידי הדפסת קוד אסקי 7 למסך.

שגרת השרות החדשה תתחיל בתווית `newIntlc`. הנח כי וקטור הפסיקה של שגרת השרות הקודמת נשמר במשתנה `oldIntlc` בסגמנט הקוד של שגרת השרות החדשה.  
הקפד על כללי התכנות המקובלים בשגרת שרותי השעון.

.386

.code

`oldIntlc dd ?`

`newIntlc dd 30000` *כדי להחליף את 30000 ב-10*

`beepers db 0`

`beepers db 0`

`count db 0`

`newIntlc:`

`push ax`

`push dx`

`add esp, 4`

`mov esi, newIntlc`

`mov edi, oldIntlc`

`mov ecx, 10`

`inc beepers`

`exit:`

`mov beepers, 0`

`mov ecx, 10`

`mov esi, newIntlc`

`mov edi, oldIntlc`

`13: mov esi, 0`

`mov edi, 0`

`int 21h`

`pop dx`

`pop ax`

`ret 4`

*איפה איפוס beepers?*

*איפוס איפה esi?*

*count de מה?*

*count קומ?*

*jmp 12*

*inc count*

*jmp 13*

*mov count, 0*

*jmp 11*

-4/9

i. האם, לדעתך, הצפופים בכל סדרת צפופים אכן יישמעו בקצב אחיד ומדויק? הסבר.

*לא, כי יש להמתין 2 שניות בין הצפוף לצפוף, ולכן יש להמתין 2 שניות בין הצפוף לצפוף.*

המשך שאלה מס' 4 בדף הבא



## שאלה מס' 4 (המשך)

ג. לצרכי דיבוג של תכנית המשתמש, ברצוננו לבנות מנגנון שיופעל עם כל ביצוע של פקודת המכונה add, ויבדוק האם פעולת החיבור גרמה לגלישה. במידה והייתה גלישה, תודפס מייד על המסך הודעת אזהרה כדלקמן: **Warning: Overflow on add**. הערה: עבור כל מקרה של גלישה בפקודת add תודפס הודעה נפרדת.

ניתן לממש את המנגנון המתואר לעיל על ידי שגרת שרות לפסיקת single-step (פסיקה מס' 1). כדי להפעיל מנגנון זה על קטעי קוד נבחרים בתכנית המשתמש, ניתן, לדוגמא, להשתמש במקרוים `traceOn` ו-`traceOff` שנלמדו בכיתה.

עליך כתוב שגרת שרות לפסיקה מס' 1 שתממש את המנגנון המתואר לעיל. אין צורך לבצע שרשור לשגרת הפסיקה הקודמת.

הנח שה- opcode של פקודת add (כלומר הבית הראשון בקוד המכונה) הוא ערך בין 0 ל-5 (כולל).

במקום בעת שנוצרת פסיקת single-step נשמרת במחסנית כתובת הפקודה הבאה, ולא כתובת הפקודה שהסתיימה כרגע. על שגרת הפסיקה "לזכור" את הפקודה הבאה עד לאחר ביצועה.

code

```

add: mov edi, 0
      print db "Warning: overflow on add", 10, 13, 0
      push edi
      cmp add, edi
      jao 1
      pop edi
      jmp 1
      print db "exit", 10, 13, 0
      jmp 1

exit1: mov esi, [ebp+8]
      mov eax, esi
      mov ax, [6x]
      cmp al, 5
      ja  exit1
      inc add, esi
      jmp  exit1

print:
exit1: mov add, esi
exit2: pop esi
      popa
      popa

```

2/9

