הטכניון – מכון טכנולוגי לישראל

ארגון ותכנות המחשב

תרגיל 2 - חלק יבש

המתרגל האחראי על התרגיל: איתי אילת.

שאלותיכם במייל בעניינים מנהלתיים בלבד, יופנו רק אליו.

כתבו בתיבת **subject**: יבש **2** את"ם.

שאלות בעל-פה ייענו על ידי כל מתרגל.

הוראות הגשה:

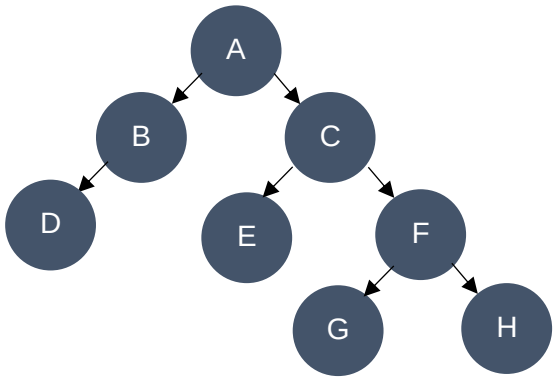
- לכל שאלה יש לרשום את התשובה במקום המיועד לכך.
- יש לענות על גבי טופס התרגיל ולהגיש אותו באתר הקורס כקובץ PDF.
- על כל יום איחור או חלק ממנו, שאינו בתיאום עם המתרגל האחראי על התרגיל, יורדו 5 נקודות.
- הגשות באיחור יש לשלוח למייל של אחראי התרגיל בצירוף פרטים מלאים של המגישים (שם+ת.ז).
- שאלות הנוגעות לתרגיל יש לשאול דרך הפיאצה בלבד.
- ההגשה בזוגות.

שאלה 1 (45 נק') – שגרות:

ג'וני סטודנט אחראי כל בוקר בשש ו30 כותב קוד אסמבלי. לפניכם מקטע הנתונים שג'וני כתב:

```
.section .data
A:  .long 3
    .quad B
    .quad C
B:  .long 4
    .quad D
    .quad 0
C:  .int 5
    .quad E
    .quad F
D:  .int 7
    .quad 0
    .quad 0
E:  .int 8
    .quad 0
    .quad 0
F:  .int 9
    .quad G
    .quad H
G:  .int 10
    .quad 0
    .quad 0
H:  .int 11
    .quad 0
    .quad 0
```

א. ציירו את הגרף המתקבל מפירוש מקטע הנתונים (מומלץ להסתכל בתרגול 3 תרגיל 1 ולהיזכר שם על אופן פירוש הזיכרון כרשימה מקושרת). בכל צומת בגרף ציינו את התווית המתאימה לו בלבד (אין צורך לציין ערכים נוספים) (3 נקודות)



ג'וני לא מפחד משגרה שוחקת ולכן כותב את השגרה func וקוד שמתשמש בה (שימו לב סוף השגרה בעמוד הבא):

```
27  .section .text
28  .global _start
29  _start:
30      mov $8, %esi
31      mov $A, %rdi
32      call func
33      movq $60, %rax
34      movq $0, %rdi
35      syscall
36
37  func:
38      pushq %rbp
39      movq %rsp, %rbp
40      cmp (%rdi), %esi
41      jne continue
42      mov $1, %eax
43      jmp finish
44
45  continue:
46      cmpq $0, 4(%rdi)
47      je next
48      pushq %rdi
49      mov 4(%rdi), %rdi
50      call func
51      pop %rdi
52      cmp $1, %eax
53      je finish
54  next:
55      cmpq $0, 12(%rdi)
56      je fail
57      pushq %rdi
58      mov 12(%rdi), %rdi
59      call func
60      pop %rdi
61      cmp $1, %eax
62      je finish
66  fail:
67      mov $0, %rax
68  finish:
69      leave
70      ret
```

ב. נתון שבתחילת התוכנית ערך של rsp הוא x. כאשר x הוא מספר בקסדצימלי. מה הוא הערך המקסימלי ומה הערך המינימלי שrsp יכיל לאורך ריצת התוכנית? תנו נוסחא שהמספרים בה הם בבסיס הקסדצימלי (בטאו את התשובה בהאמצעות x). (7 נקודות)

- הערך המקסימלי של rsp הוא הערך ההתחלתי x, שכן בכל פעולת push אנו מקטינים את ערכו של rsp (מחסנית גדלה מטה). לכן $x_{\max} = x$.
- מציאת הערך המינימלי:
  - בקריאה לפונ' func, עושים call יחיד, שדוחף למחסנית את rip – כלומר 8 בתים (0x8)
  - בכל כניסה לפונ' func, מתבצע push של rbp – כלומר 8 בתים (0x8).
  - בכל כניסה לפונ' continue או next, מתבצע push של-rdi , ומתבצע call יחיד, שדוחף למחסנית את rip. כלומר 16 בתים (0x10).
  - בקוד אנחנו מבצעים חיפוש pre-order לערך 8 בעץ. כל כניסה לבן מלווה בcontinue או next, וכל יציאה מבן מלווה ב- ret, leave ו- pop %rdi , כלומר "מחזירים" 24 בתים. לכן, נסיק כי נקבל את הערך המינימאלי כאשר נהיה בשלב הכי "עמוק" בעץ. כלומר, בחיפוש pre-order בעץ הנתון, נגיע לעומק 3 (ב-D וב-E).
  - כלומר, סה"כ ניכנס ל- func 3 פעמים (A – C – E), וניכנס לבנים פעמיים (C – E).

- ולכן הנוסחה היא:

$$x_{min} = x - (0_x8 + 3 * 0_x8 + 2 * 0_x10) = x - (0_x8 + 0_x18 + 0_x20) =$$
$$= x - 0_x40$$

ג. רשמו מה יהיה פלט הפונקציה עבור קטע הקוד הנוכחי (5 נקודות)

- כנכתב לעיל, הפונקציה מחפשת את הערך 8 בעץ, ואם היא מוצאת אותו, היא מכניסה ל- eax את הערך 1, דהיינו true. ולכן, זה יהיה פלט הפונקציה משום שקיים צומת עם ערך 8 בעץ (E).

ד. המירו את הפונקצייה לשפת c על ידי כך שתשלימו את המקומות החסרים בקוד. העיזרו בהגדרת structה שנתונה לכם (10 נקודות):
structה הנתון:

```
typedef struct _Node {
    int data;
    struct _Node *left;
    struct _Node *right;
} Node;
```

הערה1: שני הפרמטרים צריכים להיות תואמים לשני הפרמטרים של פונקצית האסמבלי גם מבחינת תפקיד וגם מבחינת סדר. כלומר, root צריך להתאים בתפקידו לפרמטר הראשון שמועבר לפוקנציה בשפת אסמבלי גם מבחינת הקונבנצייה שלמדנו.
הערה2: אורך הקו לא מלמד על אורך האיבר שצריך להשלים. **מותר להשלים יותר ממילה אחת בכל קו אך לא יותר מפקודה אחת!**

```
bool func ( Node *root, int x){
        If (root->data == x )
                return true;
        if (root->left != null)
          if (func(root->left,x))
                return true;
        if (root->right != null)
           return func(root->right,x);
        return 0;
}
```

הערה: בסעיפים הבאים יש כל מיני שינויים בקוד. כל שינוי מתקיים רק בסעיף בו מופיע. זאת אומרת הסעיפים לא תלויים אחד בשני.

ה. מוני חבר של ג'וני הוא לא כמו ג'וני. הוא אוהב לעשות שינויים רבים בקוד. הוא מחליט לקחת את המקטע הנתונים של ג'וני ולשנות בכל struct את הintה בquad. כלומר מקטע הנתונים ישתנה כך:

```
.data
A:  .quad 3
    .quad B
    .quad C
B:  .quad 4
    .quad D
    .quad 0
```

ובאופן דומה כל שאר האותיות יחליפו את הנתון הראשון במקום בint בquad.
רשמו את השינויים שצריכים להיות בקוד על מנת שיעבוד בצורה תקינה עם מקטע הנתונים החדש (5 נקודות)

מה שיצטרך להשתנות בקוד הוא הפנייה לבנים של צמתים. בגלל ששינינו ל-quad במקום long, נוסיף 4 בתים עבור קריאה לבן ימני ולבן שמאלי. כלומר:

```
Mov $8, %esi
Mov $A, %rdi
Call func
Movq $60, %rax
Movq $0,%rdi
Syscall
func:
pushq %rbp
movq %rsp,%rbp
cmp (%rdi),%esi
jne continue
mov $1,%eax
jmp finish
continue:
cmpq $0,8(%rdi)
je next
pushq %rdi
mov 8(%rdi),%rdi
call func
pop %rdi
cmp $1,%eax
je finish
next:
cmpq $0,16(%rdi)
je fail
pushq %rdi
mov 16(%rdi),%rdi
call func
pop %rdi
cmp $1,%eax
je finish
fail:
mov $0,%rax
finish:
leave
ret
```

ו. ג'וני מתחיל להתעייף מהשגרה ומחליט לקום ולשנות את מבנה הנתונים באופן הבא:

```
D:  .int 7
    .quad A
```

מה יהיה פלט התוכנית? יש לסמן תשובה מבין התשובות הבאות ולנמק במשפט **אחד**: (5 נקודות)

a. התוכנית תסתיים ופלט הפונקציה יהיה 1
b. התוכנית תסתיים ופלט הפונקציה יהיה b
c. התוכנית תכנס ללולאה אנסופית - הבן השמאלי של D יהיה כעת ראש העץ A, ולכן מאופי החיפוש בעץ, שבודק קודם כל את הבן השמאלי, נקבל לולאה אין סופית של כניסות לבן השמאלי A-B-D-A. נציין שבסופו של דבר התכנית תקרוס, בין אם בגלל stackOverflow , ובין אם תחרוג מהזמן שהוקצב לה ע"י מערכת ההפעלה.
d. התוכנית תקרוס במהלך ריצה
e. התוכנית כלל לא תבנה

ז. פתאום ג'וני כמו מוני! מחליט לבצע שינויים נוספים ולא שגרתיים בקוד מול כל שינוי שג'וני מציע עליכם לכתוב האם נכונות השגרה תיפגע (האם יש קלט עבורו השגרה לאחר השינוי שונה מהשגרה לפני השינוי). הסיברו **בקצרה** את תשובתכם!: (10 נקודות)

a. מחיקת הפקודת push וpop שבשורות 60 ו 57.
- השגרה לא תשתנה. שורות אלה תפקידן לשחזר בחזרה מהבן הימני, את ערך הצומת שממנו הגענו. אולם, במעבר pre-order כפי שמתבצע בקוד, בחזרה מהבן הימני, אנו גם מסיימים את הבדיקה של כלל הצמתים שמעליו. לכן, אין צורך לשמור את הערכים שעשויים לחזור מהבן הימני ולכן אין קלט עבורו פעולת השגרה תשתנה או לא תעבוד כמתוכנן.

b. מחיקת הפקודה pop בשורה 60
- האלגוריתם יעבוד. למעשה, בתחילת כל frame של בן ימני, אנו דוחפים את rdi למחסנית. מחיקת השורה תוביל לכך שלא נשחזר את rdi כשחוזרים מהבן הימני. לפי מה שנכתב בסעיף הקודם, אין צורך לגבות את האב כשיורדים לבן הימני. בנוסף, כאשר נבצע את הפקודה leave, אנו מחזירים את rsp להיות בסיס הframe, וכך נדרוס את הערך השמור והחזרה לא תיפגע.

c. מחיקת push וpop שבשורות 48 51
- השגרה כן תשתנה אם נמחק שתי שורות אלו. זאת מכיוון שלפי אופן פעולת האלגוריתם, לאחר ירידה לבנים שמאליים אנו חוזרים אחורה לאבות שלהם ברקורסיה. אם הערכים של האבות לא יהיו מעודכנים בעת ההשוואה, האלגוריתם ימשיך לבן הימני של הבן השמאלי (ולא לבן הימני של האב). למשל, נחפש את הערך 5 בעץ (אשר שוכן בצומת C). האלגוריתם ירד בענף השמאלי עד לצומת D ויחזיר false עד כה משום שאף אחד מהערכים אינו תואם ל-5. יחד עם זאת, אין עדכון של הצומת שממנה נרצה לצאת לבנים הימניים ולכן בעת העלייה למעלה, כאשר אנו בצומת B נבדוק שוב את הבן הימני של צומת D שהוא null, וגם בחזרה הבאה ברקורסיה נבדוק שוב את הבן הימני של D במקום את הבן הימני של הצומת האמיתית- A. לפיכך נקבל שהבן הימני של A הוא בעצם null (למרות שזו תוצאה שגויה) וכך האלגוריתם יפסיק ולא יחזיר את התוצאה הרצויה.

d. הוספת פקודה push %rdi אחרי continue בשורה 45
- האלגוריתם לא ישתנה. כל כניסה לבן מלווה בקריאה ל-func ולכן נפתח frame חדש. הוספת הפקודה תשמור פעמיים את ערך rdi הנוכחי, כאשר פונים לבן השמאלי שלו. לאחר מכן, מתבצע שחזור תקין ונכון לערך האב המקורי (בשונה מהסעיף הקודם), ומיד עוברים לבן הימני, שעושה גם push ו-pop ל-rdi. לאחר מכן נבצע leave ו- ret, שישחזרו את rsp וrbp הקודמים, באופן שידרוס את השמירה הכפולה, ולכן השמירה הכפולה לא תוביל לשינוי.

e. הוספת הפוקדה push %rdi אחרי continue בשורה 45, שינוי פוקדת הpop שבשורה 51 לפקודה: mov (%rsp), %rdi ומחיקת הפקודת push וpop בשורות 60 57.
- האלגוריתם לא ישתנה. ראינו כי שורות 45, 60, 57 לא ישנו את הפונקציונליות של האלגוריתם. הפקודה mov (%rsp), %rdi לא תבצע rsp++ לעומת pop לאחר שחוזרים ברקורסיה מהבן השמאלי, אך לא מתבצעים pop-ים נוספים המסתמכים על כך ש-rsp

השתנה, וכן צומת האב עדיין תשוחזר לערך הנכון ברגיסטר rdi. ולכן לא תהיה פגיעה באלגוריתם.

שאלה 2 (30 נק') – קריאות מערכת:

ג'ואי מרגיש מתוסכל מכך שחבריו חושבים שהוא פחות חכם מהם. לכן, הוא מחליט להרשים אותם בעזרת כתיבת קוד אסמבלי.

א. לפניכם מקטע הנתונים שג'ואי כתב מבלי ערכי הנתונים עצמם:

```
.section .data
msg1: .ascii ???????
msg2: .ascii ???????
msg1_len: .quad ____
msg2_len: .quad ____
all_msg_len: .quad ____
```

ג'ואי לא יודע עדיין אילו מחרוזות הוא יכתוב. עליכם להשלים את המקומות הריקים שקשורים לאורכי המחרוזות כך שמשתנה msg1_len יהיה האורך של msg1, בmsg2_len יהיה האורך של msg2 ובמשתנה all_msg_len יהיה שווה לסכום אורכי המחרוזות msg1 וmsg2. שימו לב עליכם לעשות זאת בצורה כזו שהאורכים יהיו נכונים בעת ריצת התוכנית ללא קשר לאיזה מחרוזות ג'ואי ישים בmsg1 ובmsg2. (3 נקודות)

```
msg1_len:  .quad msg2-msg1
msg2_len:  .quad msg1_len-msg2
all_msg_len:  .quad msg1_len-msg1
```

ב. כעת נתון מקטע הנתונים שכולל את המחרוזות:

```
.section .data
msg1: .ascii "HOW YOOOU DOOIN?"
msg2: .ascii "JOEY DOESN'T SHARE FOOD!"
msg1_len: .quad ____
msg2_len: .quad ____
all_msg_len: .quad ____
```

לפניכם נתונה התוכנית שג'ואי כתב:

```
.section .text
.global _start
_start:
    mov $msg1, %rsi
    mov $1, %rdi
    mov $1, %rdx
    mov $1, %rax
    xor %rbx, %rbx

    movq msg1_len, %r9
    call Joey_func
```

ומוצגת כאן גם הפונקציה שכתב:

```
Joey_func:
        cmp %rbx, %r9
        je end
        # addb $0x20, (%rsi)
        test $1, %rbx
        jnz skip
        syscall
skip:   inc %rsi
        inc %rbx
        call Joey_func
end:    ret
```

מה יודפס בסיום ריצת הקוד? (שימו לב השורה השלישית בפונקציה נמצאת בהערה ולא רלוונטית לסעיף). (5 נקודות)

הקוד רץ על המחרוזת הראשונה, ומדפיס כל אות שניה. לכן, נקבל: HWYOUDON

ג. כעת מורידים את הסולמית שנמצאת בפונקציה (וכעת הפקודה חלק מהקוד) בנוסף מחליפים את השורה movq msg1_len, %r9 בשורה: movq all_msg_len, %r9.
הערה: שינויים אלו ילוו אותנו גם בסעיפים הבאים (בסעיפים ד - ו השינויים בסעיף ג עדיין תקפים).
מה יודפס כעת בסיום ריצת הקוד? (5 נקודות)
הפעם הקוד רץ על 2 המחרוזות, וגם הפעם ידפיס כל אות שנייה. ללא השורה בסולמית, יודפס: HWYOUDONJE OS' HR OD.
השורה שנמצאת בסולמית מוסיפה את הערך $0x20$ לכל תו ascii – ולכן כל אות גדולה במחרוזת תהפוך לאות קטנה, כל רווח שיודפס יהפוך ל-@, והתו ' יהפוך ל-G. כלומר יודפס:
hwyoudonje@osG@hr@od

ד. בזמן שג'ואי אכל בסלון סנדוויץ, חיית המחמד שלו (אפרוח) טיילה על המקלדת והוסיפה את הפקודה:
inc %r9. הפקודה נוספה שורה לפני הקריאה לפונקציה של ג'ואי בתוכנית הראשית.
מה יהיה פלט התוכנית כעת? (2 נקודות)
%r9 מכיל את אורך 2 המחרוזות, שעליו הקוד רץ. השורה תגדיל את האורך שנרוץ עליו ב-1. לכן, נזלוג לבית הראשון בכתובת msg1_len, ונדפיס את הערך שלו ב- ascii.
אורך המחרוזת הראשונה היא $0x10$, ולכן בבית הראשון בכתובת msg1_len מופיע המספר $0x10$.
כאשר נוסיף למספר $0x20$ נדפיס בסוף התוכנית את התו בערך $0x30$ – שהוא 0. כלומר יודפס:
hwyoudonje@osG@hr@od0

ה. חברה טובה של ג'ואי פיבי אמרה לו ששימוש ברגיסטר r9 מביא מזל רע. ג'ואי נלחץ נורא והחליט שיש לבצע שינוי בקוד מבלי לשנות את תוצאות הפעולה של הפונקציה (כלומר הפלט צריך להיות זהה). כיוון ולא ידע איך לשנות את הקוד הוא החליט לבקש את עזרת חבריו.
בסעיף הזה יופיעו העצות של כל החברים. עליכם לרשום ליד כל עצה האם היא לדעתכם תעזור לג'ואי. נמקו **בקצרה(!)** (10 נקודות)

צ'נדלר מציע להחליף את השימוש בr9 בשימוש בrcx.
הקוד לא יעבוד כי ה- syscall ידרוס את רגיסטר rcx וישמר בו rip.
מוניקה מציעה להחליף את השימוש בr9 בשימוש בr11.
הקוד לא יעבוד כי ה- syscall ידרוס את רגיסטר r11 וישמר בו rflags.

פיבי מציעה להחליף את השימוש בr9 בשימוש בrdi.
הקוד לא יעבוד כי ה- sys_write שקוראים לו בקוד, משתמש ב rdi כדי לזהות את אמצעי הפלט אליו צריך להדפיס. לכן, שינוי rdi מ-1 לערך של אורך המחרוזת, יגרור התנהגות לא צפויה, ובפרט לא ידפיס משהו על המסך.
רייצ'ל מציעה להחליף את השימוש בr9 בשימוש בr12.
הקוד צפוי לעבוד כי אין שימוש ב r12 כאשר קוראים ל sys_write, syscall והוא לא נדרס ע"י פעולות אחרות בקוד.
רוס מציע להחליף את השימוש בr9 בשימוש בrbp.
הקוד צפוי לעבוד. בקוד אין שימוש בפקודות הדורשות או משנות את הערך של בסיס המחסנית (לדוגמה leave, enter), ולכן נוכל להשתמש ב- rbp ע"מ לשמור ערכים.

ו. חבריו של ג'ואי מסבירים לו שהשימוש שלו ברקורסיה מיותר ובזבזני והוא יכול את אותו קוד בדיוק לכתוב בלולאות. ג'ואי מחליט לבצע את השינויים הבאים:
בתוכנית הראשית בשורה שלפני ביצוע הפקודה call ג'ואי מוסיף את הפקודה:
mov $Joey_func, %rcx
ובתוך הפונקציה ג'ואי מוחק את השורה בה יש שימוש בפקודה call והחליף אותה בפקודה:
jmp *%rcx
שימו לב שהתווית end נמצאת אחרי פקודה זו.
לצורך הבהרה הפונקציה נראת כך כעת:

```
Joey_func:
        cmp %rbx, %r9
        je end
        addb $0x20, (%rsi)
        test $1, %rbx
        jnz skip
        syscall
skip:   inc %rsi
        inc %rbx
        jmp *%rcx
end:    ret
```

כיצד שינוי זה ישפיע על אופן ריצת הפונקציה. מה יודפס אם נריץ את הפונקציה? (5 נקודות)
ג'ואי בחר לשמור את הכתובת של Joey_func ברגיסטר rcx. פקודת ה-syscall הראשונה תתבצע כנדרש, ותדפיס h למסך. לאחריה, ידרס הערך ב-rcx ובמקומו יושם ע"י מערכת ההפעלה את הrip לאחר פקודת ה-syscall, כלומר ברגיסטר תשמר הכתובת של הפקודה "inc %rsi", ואליה נקפוץ כל פעם. לכן, תיווצר לולאה אין סופית של רצף הפקודות:

```
skip:   inc %rsi
        inc %rbx
        jmp *%rcx
```

לסיכום, יודפס התו h.

שאלה 3 (25 נק') – רמות הרשאה ואוגר הדגלים:

א. הפקודה pushfq דוחפת את הערך של אוגר הדגלים למחסנית. והפקודה popfq מוציאה את אוגר הדגלים מהמחסנית. הסבירו כיצד באמצעות שילוב של שתי פקודות אלו ניתן להדליק את הדגלים OF וCF. שימו לב במידה ואחד הדגלים כבר דלוק יש להשאירו דלוק כלומר, בסיום התהליך על שני הדגלים להיות דולקים. אין לשנות את שאר הביטים בריגסטר הדגלים. בנוסף, אין לשנות אף רגיסטר שהוא לא rflags, rip, rsp (גם לא באופן זמני). (7 נקודות)
הערה: במידה ובדקתם את עצמכם באמצעות דיבגר וראיתם שנדלק גם דגל TF זה בסדר תלמדו בהמשך מדוע הוא נדלק תוך כדי דיבוג.

נכניס את אוגר הדגלים למחסנית, נבצע or עם הביטים הדרושים להדלקה (במקרה זה 0 ו-11), ולאחר מכן נוציא את אוגר הדגלים מהמחסנית:

```
pushfq
orq $0b100000000001 , (%rsp)
popfq
```

ב. הולי התחמנית רוצה לאפשר לעצמה גישה ישירה אל התקני הקלט פלט ללא צורך בקריאות מערכת. איזה שינוי **באוגר הדגלים** יכול לעזור להולי במטרתה? (4 נקודות)
הערה: לא צריך לציין פקודה ספציפית, רק להגיד מה צריך לעשות ברמה התיאורטית
ברמה התיאורטית, השינוי הנדרש הוא הדלקת הדגל IOPL.

ג. הולי מחליטה לנסות את התעלול מסעיף א' רק שבמקום לשנות את CF וOF היא רוצה לשנות את IOPL. להפתעתה, היא לא מצליחה לשנות את הביטים הללו. הסבירו מה ההגיון בכך שהיא לא מצליחה לשנות את IOPL? התייחסו לצורך בקריאות מערכת (4 נקודת)
הצורך בקריאות מערכת הוא ליצור הפרדה בין חלקים רגישים של המחשב, שלא נרצה שלכל אחד תהיה גישה לשנות אותם כרצונו. בפרט, עבור התקני הקלט-פלט, יש צורך ברמת הרשאה של מערכת הפעלה כדי שרק היא תקבע את התקני הקלט-פלט. לכן, האפשרות לשנות את הביטים של IOPL דרך המחסנית נמנעת.
מאחר ורמת ההרשאה של הולי היא של משתמש, היא לא מצליחה לשנות את הביטים כרצונה.

CPL	IOPL	13:12 IOPL
0	0-3	S
1-3	<CPL	N
1-3	≥CPL	N

הערה: הסעיפים הבאים קשורים לפסיקות מומלץ לענות עליהם לאחר התרגול על פסיקות.

ד. וולי החבר המבולבל של הולי מתלבט כיצד ניתן לחסום פסיקות תוכנה לכן הוא שואל את הולי. אילו מבין התשובות הבאות על הולי לענות לו? יש לסמן את האפשרות הנכונה. (5 נקודות)
1. כיבוי דגל IF באוגר הדגלים
2. הדלקת דגל IF באוגר הדגלים
3. שינוי CPL ל00
4. לא ניתן לחסום פסיקות תוכנה.

ה. כעת נתון שוולי הצליח להגיע למצב שבו CPL שווה ל0. וולי מעוניין לחסום פסיקות חומרה שאינן מועברות דרך כניסת NMI. כיצד הוא יכול לעשות זאת? (5 נקודות)

1. כיבוי דגל IF באוגר הדגלים
2. הדלקת דגל IF באוגר הדגלים
3. עליו לחבר את הפסיקות לכניסת NMI ואז לכבות את דגל IF
4. לא ניתן לחסום פסיקות חומרה ולכן לא יצליח.