הטכניון – מכון טכנולוגי לישראל

ארגון ותכנות המחשב

תרגיל 2 - חלק יבש

המתרגל האחראי על התרגיל: תומר כץ.

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.

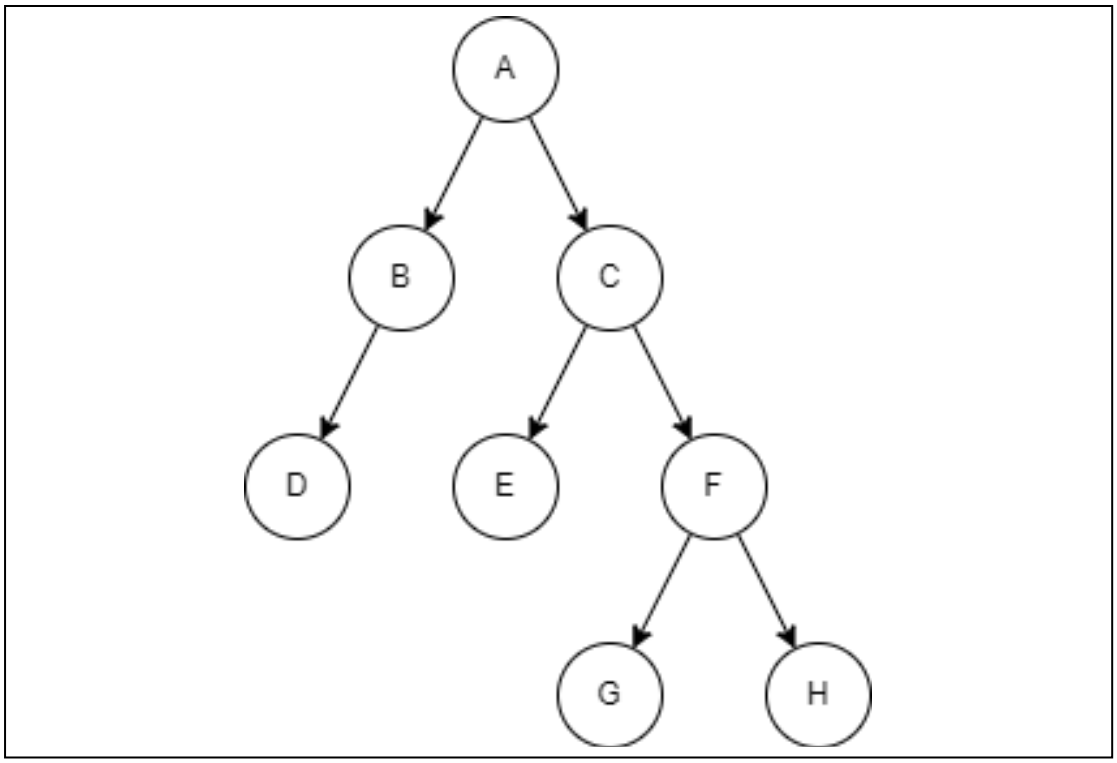
מגישים:
אנטוני סלבין
נועם גולדשטיין

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
```

1. 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:

```
.section .text
.global _start
_start:
    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, 4(%rdi)
    je next
    pushq %rdi
    mov 4(%rdi), %rdi
    call func
    pop %rdi
    cmp $1, %eax
    je finish
next:
    cmpq $0, 12(%rdi)
    je fail
    pushq %rdi
    mov 12(%rdi), %rdi
    call func
    pop %rdi
    cmp $1, %eax
    je finish

fail:
    mov $0, %eax
finish:
    leave
    ret
```

2. 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). (5 נקודות)

מינימום : $x - 0x40$

מקסימום : x

3. רשמו מה יהיה פלט הפונקציה עבור קטע הקוד הנוכחי (7 נקודות)

פלט הפונקציה יהיה 1 מכיוון הערך 8 נמצא בעץ (בצומת E)

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

הstruct 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!

```
int func (Node* root, int x){

      If (root->data == x)

            return 1;

      if (root.left != null)

         if (func(root.left, x) == 1)

            return 1;

      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.

5. 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 הint 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 ב-quad במקום ה-int.

רשמו את השינויים שצריכים להיות בקוד על מנת שיעבוד בצורה תקינה עם מקטע הנתונים החדש (5 נקודות)

בכל מקום בו רשום $\%esi$ צריך לשנות ל- $\%rsi$.

בכל מקום עם שיטת המיעון $4(\%rdi)$, יש להחליף ל- $8(\%rdi)$.

בכל מקום עם שיטת המיעון $12(\%rdi)$, יש להחליף ל- $16(\%rdi)$.

6. 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 בקצרה: (5 נקודות)

- התוכנית תסתיים ופלט הפונקציה יהיה 1
- התוכנית תסתיים ופלט הפונקציה יהיה b
- התוכנית תכנס ללולאה אינסופית
- התוכנית תקרוס במהלך ריצה
- התוכנית כלל לא תבנה

נימוק: התוכנית תיכנס לרקורסיה אינסופית כאשר בכל קריאה רקורסיבית $\%rsp$ גדל, ולכן התוכנית תקרוס עקב שגיאת stack overflow.

7. 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).

הסבירו בקצרה את תשובתכם! (10 נקודות)

- מחיקת הפקודת push popו שבשורות 60 ו 57. – לא יפגע
- מחיקת הפקודה pop בשורה 60 – לא יפגע
- מחיקת push popו שבשורות 48 51. – יפגע
- הוספת פקודה push %rdi אחרי continue בשורה 45. – לא יפגע
- הוספת הפוקדה push %rdi אחרי continue בשורה 45, שינוי פוקדת הpop שבשורה 51 לפקודה: mov (%rsp), %rdi ומחיקת הפקודת push popו בשורות 60 57. – לא יפגע

נימוק:

1. מחיקת הפקודות הללו גורמת לכך שבריצה מסוימת של הפונקציה, כשקוראים רקורסיבית לפונקציה על הבן הימני של צומת, המצביע לצומת (rdi) אינו נשמר ומשתנה להצביע על הבן הימני. בחזרה מהקריאה הרקורסיבית, המצביע עדיין מצביע לבן הימני ולא לצומת עצמו. אולם, הפונקציה אינה משתמשת יותר ב $\%rdi$ ולכן נכונות השגרה אינה תיפגע.
2. מחיקת הפקודה pop תגרום לאותו המצב שקרה ב-[1] (rdi אינו חוזר לערכו המקורי) שכאמור אינו משפיע על השגרה. בנוסף, בסוף השגרה, לפני האפילוג, $\%rsp$ אינו חוזר להצביע על תחילת מסגרת הפונקציה במחסנית, אך הפקודה $leave$ הכוללת את הפקודה $mov\ \%rbp, \%rsp$ מטפלת בזה. (הוזזת $\%rsp$ לבסיס המסגרת שקול לשחרור המקום במחסנית שהוקצה לפונקציה)
3. מחיקת הפקודות הללו גורמת לכך שבריצה מסוימת של הפונקציה, כשקוראים רקורסיבית לפונקציה על הבן השמאלי של צומת, המצביע לצומת (rdi) אינו נשמר ומשתנה להצביע על הבן השמאלי. בחזרה מהקריאה הרקורסיבית, המצביע עדיין מצביע לבן השמאלי ולא לצומת עצמו. ולכן כשנסיים את החיפוש בתת העץ השמאלי, ונרצה לחפש בתת העץ הימני, לא נוכל לעשות זאת, כי אנחנו ניגש לבן הימני של צומת אחר ולא לשלנו.
   * למשל, על הדוגמה שמצוירת בסעיף א', בצומת A כשנחזור מחיפוש בתת העץ השמאלי, $\%rdi$ יצביע על צומת D (לא B כי אותה הבעיה קורה שם). וכשננסה לחפש את הערך 8 בצד ימין אנחנו נחפש מימין לD ונחזיר 0 - שהערך לא קיים בעץ. זאת למרות שהפלט הרצוי הוא 1.
4. הוספת $push$ לא פוגעת באף pop בהמשך הפונקציה שאמור לקבל ערך אחר. הדבר היחיד שמשתנה הוא שעבשיו אנו לא משחרים את כל הזיכרון במחסנית כמו ב-[2]. כאמור, הפקודה $leave$ פותרת את זה.
5. הוספת $push$ אינה משנה כאמור ב-[4] . החלפת pop ב-$mov\ (\%rsp), \%rdi$ טוענת את הערך בראש המחסנית ל $\%rdi$ בדיוק כמו ש $pop\ \%rdi$ עושה. ההבדל הוא ש $\%rsp$ לא משתנה והמחסנית לא קטנה. טעינת ראש המחסנית ל $\%rdi$ מחזיר אותו לערכו המקורי ולכן הבעיה מ-[3] אינה קוראת. הבעיה שהמחסנית לא קטנה נפתרת עם $leave$ כאמור ב-[2]. מחיקת הפקודות בשורות 57,60 אינה משנה כאמור ב-[1].

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.

1. 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 1msg, בmsg2_len יהיה האורך של 2msg ובמשתנה all_msg_len יהיה שווה לסכום אורכי המחרוזות 1msg ו2msg. שימו לב עליכם לעשות זאת בצורה כזו שהאורכים יהיו נכונים בעת ריצת התוכנית ללא קשר לאיזה מחרוזות ג'ואי ישים ב1msg וב2msg. (3 נקודות)

msg1_len: .quad msg2 - msg1

msg2_len: .quad msg1_len - msg2

all_msg_len: .quad msg1_len - msg1

2. 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)

HWYOUDON

3. כעת מורידים את הסולמית שנמצאת בפונקציה (ובעת הפקודה חלק מהקוד) בנוסף מחליפים את השורה 9movq msg1_len, %r בשורה: 9movq all_msg_len, %r.
הערה: שינויים אלו ילוו אותנו גם בסעיפים הבאים (בסעיפים ד - ו השינויים בסעיף ג עדיין תקפים).
מה יודפס כעת בסיום ריצת הקוד? (5 נקודות)
hwyoudonje@osG@hr@od

4. בזמן שג'ואי אכל בסלון סנדוויץ, חיית המחמד שלו (אפרוח) טיילה על המקלדת והוסיפה את הפקודה: 9inc %r. הפקודה נוספה שורה לפני הקריאה לפונקציה של ג'ואי בתוכנית הראשית.
מה יהיה פלט התוכנית כעת? (2 נקודות)
hwyoudonje@osG@hr@od0

5. חברה טובה של ג'ואי פיבי אמרה לו ששימוש ברגיסטר 9r 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 ב9r בשימוש בrcx.
לא יעזור – הפקודה $syscall$ שומרת את rip ב- rcx ודורסת אותו. ג'ואי אינו משחזר את rcx לערכו הקודם ולכן פעולת הפונקציה תיפגע.
- מוניקה מציעה להחליף את השימוש ב9r בשימוש ב11r.
לא יעזור – הפקודה $syscall$ שומרת את rflags ב- $r11$ ודורסת אותו. ג'ואי אינו משחזר את $r11$ לערכו הקודם ולכן פעולת הפונקציה תיפגע.
- פיבי מציעה להחליף את השימוש ב9r בשימוש בrdi.
לא יעזור – אנו קוראים ל $syscall$ עם מספר שירות (הערך בrax) 1 שזה sys_write. קריאת מערכת זו מקבלת את rdi כפרמטר שמחזיק את ה descriptor פלט ולכן אם נשתמש בו, הפונקציה לא תדפיס את הפלט למסך.
- רייצ'ל מציעה להחליף את השימוש ב9r בשימוש ב12r.
יעזור – כשקוראים לפקודה syscall, מערכת ההפעלה מגבה את כל הרגיסטרים. הפקודה syscall עצמה לא משנה את רגיסטר r12 ולכן ג'ואי יכול להשתמש בו.
- רוס מציע להחליף את השימוש ב9r בשימוש בrbp.
יעזור – כשקוראים לפקודה syscall לא מתבצעת החלפת מחסניות. אבל מתבצעת החלפת מחסניות **בגרעין** על ידי מערכת ההפעלה וכל הרגיסטרים מגובים, הפקודה syscall עצמה לא משנה את רגיסטר rbp ולכן ג'ואי יכול להשתמש בו.

6. 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)

הפקודה $syscall$ שומרת את rip ב- rcx ודורסת אותו. ג'ואי אינו משחזר את rcx לערכו הקודם ולכן הפקודה $jmp * \%rcx$ תקפוץ לשורה שאחרי $syscall$ במקום לתחילת הפונקציה. מכיוון שמדלגים על ההשוואה בהתחלה הפונקציה לא תעצור. לכן יודפס "h" בכניסה לפונקציה ולאחר מכן נכנס ללולאה אינסופית שבכל לולאה מגדילים את rsi ואת rbx בלבד.

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

1. 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.

נדחוף את רגיסטר הדגלים למחסנית באמצעות $pushfq$. לאחר מכן נבצע
orq $0x801, (%rsp)
כאשר 0x801 הוא מספר שבבסיס בניארי כולו אפסים מלבד המקומות של CF ו-OF שבהם יהיה 1. כך נשנה רק את מקומות הדגלים האלה ל- 1 על המחסנית. לבסוף נבצע $popfq$ ובכך נטען הערכים החדשים ל $rflags$.

2. 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

הולי צריכה לשנות את הדגל $IOPL$ ל- 3.

3. 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 בבך שהיא לא מצליחה לשנות את IOPL? התייחסו לצורך בקריאות מערכת (4 נקודת)

לא ניתן לשנות את דגל $IOPL$ מקוד המשתמש בלי רמת ההרשאה הנכונה ($CPL = 0$).
ההיגיון הוא שאנו לא מעוניינים שכל משתמש יוכל לגשת להתקני קלט פלט כרצונו ורק מערכת ההפעלה תוכל לעשות זאת. ולכן אנו רוצים להכריח את המשתמש להשתמש בקריאות מערכת אשר מעבירות את השליטה למערכת ההפעלה (ואת CPL ל-0 בקוד הגרעין) שתיגש להתקנים בשבילנו.

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

4. 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 וגם לנמק בקצרה (5 נקודות)
   - כיבוי דגל IF באוגר הדגלים
   - הדלקת דגל IF באוגר הדגלים
   - שינוי CPL ל00
   - לא ניתן לחסום פסיקות תוכנה.

נימוק: לא ניתן לחסום פסיקות תוכנה כי הן אינן תלויות בדגל הפסיקות IF.

5. 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? יש לסמן את האפשרות הנכונה וגם לנמק בקצרה (5 נקודות)
   - כיבוי דגל IF באוגר הדגלים
   - הדלקת דגל IF באוגר הדגלים
   - עליו לחבר את הפסיקות לכניסת NMI ואז לכבות את דגל IF
   - לא ניתן לחסום פסיקות חומרה ולכן לא יצליח.

נימוק: פסיקות חומרה שאינן מועברות דרך כניסת NMI מועברות למעבד דרך APIC אשר מעביר אותם למעבד רק אם דגל הפסיקות IF דולק. לכן כיבוי IF יחסום את פסיקות אלו.