תרגול 10
פסיקות

מבוסס על שקפים מאת יאן ציטרין

1

בעיות של קלט / פלט סינכרוני

המהירות של ה-CPU גדולה בהרבה מהמהירות של ההתקנים החיצוניים

TPB

TKB

b_w: tstb @#tks
bpl b_w
ה-CPU מחכה עד שהתו יהיה מוכן בחוצץ.

b_w: tstb @#tps
bpl b_w
ה-CPU מחכה לסיום ההדפסה הקודמת.

הפתרון: שימוש במנגנון פסיקות

- **פסיקה : אירוע חיצוני שגורם למעבד בתנאים מסוימים להפסיק לבצע את קטע הקוד הנוכחי ולעבור לביצוע של שיגרה מיוחדת הנקראת *שיגרת שרות הפסיקה*.**
- **דוגמה לאירוע כזה: *הקשת תו על המקלדת*.**

יתרונות:

- התוכנית יכולה לבצע פקודות אחרות, שלא תלויות בקלט שהיא מחכה לו.
- התוכנית יכולה להגיב בקלות לסוגים שונים של אירועים. (בניגוד לשיטה הסינכרונית שבה בדיקה של מספר אירועים יותר מסורבלת).

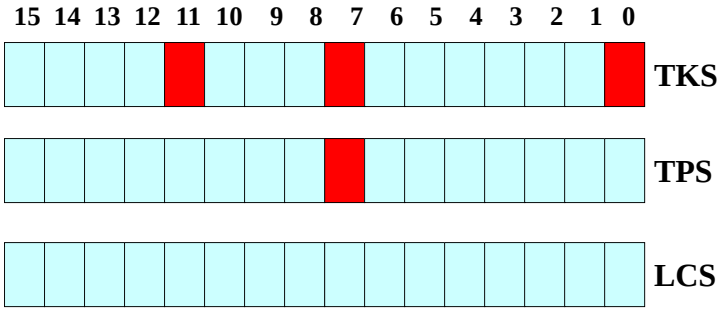
התקנים חיצוניים

התקן	מילת Status	מילת Buffer
מקלדת	TKS = 177560	TKB = 177562
מדפסת	TPS = 177564	TPB = 177566
שעון	LCS = 177546	אין (חסר משמעות)

כל התקן חיצוני יכול ליזום פסיקה:

- **המקלדת: התו שהקליד המשתמש הועבר ל- TKB והביט Done עלה מ-0 ל-1**
- **המדפסת: התו שנשלח ל-TPB הודפס והביט Ready עלה מ-0 ל-1**
- **השעון: יוזם פסיקה כל פרק זמן מוגדר ($\frac{1}{60}$ sec)**

Interrupt Enable



- הביט מספר 6 נקרא **Interrupt Enable (IE)**
- IE אומר למערכת האם ההתקן יכול להפסיק את פעולתו של ה-CPU, כלומר הדלקת הביט הזה הנה תנאי הכרחי לפסיקה

האם צריכים להיות הבדלים בין מנגנון הקפיצה לשגרת פסיקה לבין מנגנון הקפיצה לשגרה רגילה?

ההבדל נובע מעיתוי הקפיצה:
בשגרה רגילה - הקפיצה יזומה ע"י התוכנית (סינכרונית).
בשגרת פסיקה – הקפיצה אינה תלויה במצב התוכנית (אסינכרונית).

רמז: מה ההבדל בין הפקודות בקטעי הקוד הבאים:

mov r1, r2	cmp r1, r2
bic #70, r2	beq label1
add r1, r2	mov r1, r2
inc r1	sxt r1

האם צריכים להיות הבדלים בין מנגנון הקפיצה לשגרת פסיקה לבין מנגנון הקפיצה לשגרה רגילה?

ההבדל נובע מעיתוי הקפיצה:
בשגרה רגילה - הקפיצה יזומה ע"י התוכנית (סינכרונית)
בשגרת פסיקה – הקפיצה אינה תלויה במצב התוכנית (אסינכרונית).

תשובה:
יש לשמור את הPSW לפני הקפיצה לשגרת הפסיקה, ולשחזר אותו כאשר חוזרים משגרת הפסיקה.
(כמו כן, אין שימוש ברגיסטר קישור – כתובת החזרה נשמרת על המחסנית).

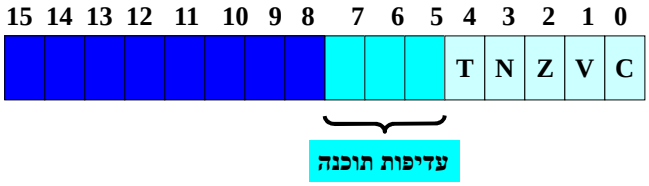
עדיפות חומרה

- לכל התקן חיצוני מוגדרת עדיפות חומרה.
- זהו מספר מ-0 עד-7 שנקבע ע"י יצרן החומרה ולא ניתן לשינוי (Hardcoded).
- משמעות של עדיפות החומרה: האם ביצוע של התוכנית הנוכחית יכול להפסיק בעקבות פסיקה שיוזם ההתקן, כלומר באיזו מידה חשוב למערכת לטפל בבקשתו של ההתקן.

התקן	עדיפות חומרה
מקלדת	4
מדפסת	4
שעון	6

עדיפות תוכנה

- המשמעות: מה החשיבות של הקוד שרץ כרגע
- שמורה בביטים 5-7 בPSW (הערכים יכולים להיות 0-7).



- עדיפות התוכנה של התוכנית הראשית הנה 0, אם לא נקבע אחרת.

וקטור הפסיקה

- איך המערכת יודעת איפה נמצאת שגרת שרות של פסיקה מסוימת?
- התשובה: לכל פסיקה מוגדר *וקטור הפסיקה* – זוג מילים בזיכרון :
  - ✓ כתובת של שגרת שרות הפסיקה
  - ✓ PSW מילת המצב של המעבד
- המתכנת אחראי על אתחול וקטור הפסיקה

וקטור הפסיקה - המשך

התקן	עדיפות חומרה	וקטור פסיקה
מקלדת	4	(60,62)
מדפסת	4	(64,66)
שעון	6	(100, 102)

מדוע מספקים ערך חדש עבור PSW בווקטור הפסיקה?

- **לקבוע את עדיפות התוכנה בזמן ביצוע שגרת הפסיקה**

בד"כ קובעים את עדיפות התוכנה של שגרת שרות הפסיקה של התקן כלשהו, בהתאם לעדיפות החומרה שלו.

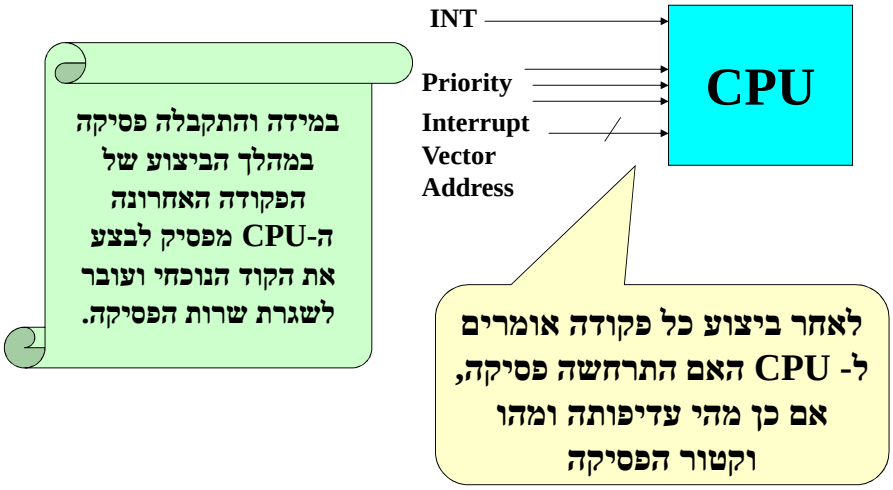
תנאים להתרחשות הפסיקה

1. IE במילת הסטטוס של ההתקן דלוק.
2. Done/Ready עולה מ-0 ל-1.
3. עדיפות החומרה של ההתקן גדולה ממש מעדיפות התוכנה שרצה כרגע.

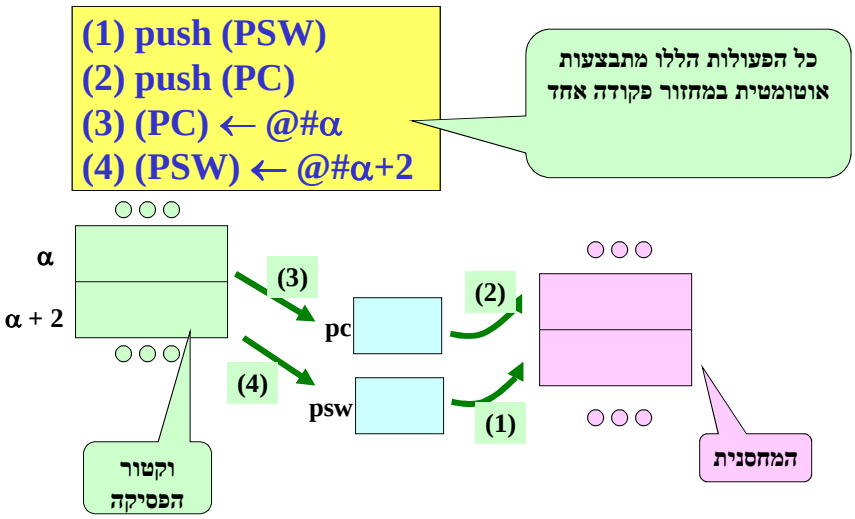
אם עדיפות החומרה לא מספיקה אז הפסיקה נדחית וממתינה לתורה

ל-CPU יש "בקר פסיקות" שמנהל תור פסיקות לפי עדיפויותיהן. בתור נשמרת לכל היותר פסיקה אחת מכל התקן. כלומר אם בתור יש כבר פסיקה מהמקלדת, מתעלמים מכל פסיקה מהמקלדת עד שלא נטפל בפסיקה הראשונה.

האינטרקציה בין המעבד לבין ההתקנים

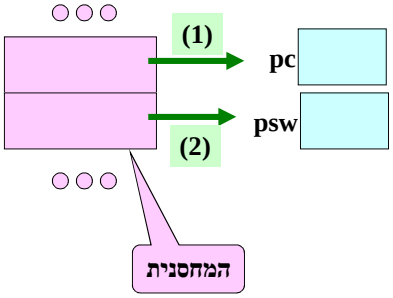


קפיצה לשגרת שרות הפסיקה



חזרה משגרת שרות הפסיקה

rti ⇔ (1) pop (PC)
(2) pop (PSW)



PSW ופסיקות - סיכום

- מדוע טוענים את ערך הPSW מווקטור הפסיקה בקפיצה לשגרת הפסיקה?

כדי לקבוע את עדיפות התוכנה בזמן ביצוע שגרת הפסיקה.

- מדוע שומרים את הPSW הקודם ומשחזרים אותו בעת החזרה משגרת הפסיקה?

כדי לא לפגוע בערכי הדגלים בקוד שהתבצע בזמן הפסיקה.
כדי לשחזר את עדיפות התוכנה המקורית.

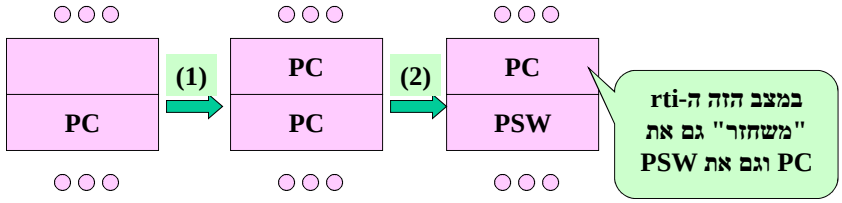
שינוי של עדיפות תוכנה

בעיה: לא ניתן לגשת ל-PSW כדי לשנות את עדיפות התוכנה

הרעיון: בשגרה רגילה לדמות את שגרת השרות של פסיקה

```
                 …
                 jsr     pc, ch_prio        ; at this point the programmer
                 …                          ; wants to change the priority

1. ch_prio:      mov     (sp), -(sp)        ;PSW simulating
2.               mov     #new_prio, 2(sp)   ;define priority
3.               rti                        ; return from the "interrupt
                                            ; routine"
```



במצב הזה ה-rti "משחזר" גם את PC וגם את PSW

דוגמה מהמבחן

```
1.   tks     =      177560
2.   tkb     =      177562
3.   tps     =      177564
4.   tpb     =      177566
5.   lcs     =      177546
6.  main:    mov    #inp, @#60
7.           mov    #200, @#62
8.           mov    #outp, @#64
9.           mov    #200, @#66
10.          mov    #clock, @#100
11.          mov    #300, @#102

12.          mov    #100, @#tps
13.          mov    #101, @#tks
14.  w:      wait
15.          br     w
16.  hlt:    halt
```

Interrupt Vector Initialization

IE ← 1, RE ← 1

דוגמה מהמבחן (המשך)

```
inp:    mov     r0, -(sp)
        movb    @#tkb, r0       } r0 contains ascii of
        bic     #177600, r0     } the typed character
        cmpb    #'$, r0         } if r0 = '$ goto seton
        beq     seton           }
        bic     #170, r0        }
        asl     alarm           }
        asl     alarm           } alarm = alarm*8 + r0
        asl     alarm           }
        add     r0, alarm       }
        mov     (sp)+, r0       }
        inc     @#tks           ; RE ← 1
        rti
seton:  mov     (sp)+, r0
        mov     #100, @#lcs     ; IE(Clock) ← 1
        rti
```


דוגמה מהמבחן (המשך)

```
outp:   clr     @#tps           ; IE(Printer) ← 0
        rti

clock:  dec     alarm           ; Every clock tick the alarm
        bne     ret             ; decreases by 1
        clr     @#lcs           ; When it reaches 0 the clock stops
b_w:    tstb    @#tps           ; and prints *
        bpl     b_w
movb    #'*, @#tpb
ret:    rti

alarm:  .word 0
```

```
1.   tks     =      177560
2.   tkb     =      177562
3.   tps     =      177564
4.   tpb     =      177566
5.   lcs     =      177546
6.   main:   mov    #inp, @#60
7.          mov    #200, @#62
8.          mov    #outp, @#64
9.          mov    #200, @#66
10.         mov    #clock, @#100
11.         mov    #300, @#102

12.         mov    #100, @#tps
13.         mov    #101, @#tks
14.  w:     wait
15.         br     w
16.  hlt:   halt
17.  inp:   mov    r0, -(sp)
18.         movb   @#tkb, r0
19.         bic    #177600, r0
20.         cmpb   #'$, r0
21.         beq    seton
22.         bic    #170, r0
23.         asl    alarm
24.         asl    alarm
25.         asl    alarm
26.         add    r0, alarm
27.         mov    (sp)+, r0
28.         inc    @#tks
29.         rti
30.  seton: mov    (sp)+, r0
31.         mov    #100, @#lcs
32.         rti
```