

בפסיקה מספר 16h הינה פסיקת התוכנה של המקלדת. תוכניות שמעונינות באינפורמציה מהמקלדת פונות אליה, בדרך כלל בעקיפין דרך מערכת ההפעלה והיא כאילו התוכנית הסטנדרטית לקרוא מידע מהמקלדת. זוהי פסיקת תוכנה של ה-BIOS ובעיקרו של דבר היא מהווה מעין קוד המכיר את מבנה הנתונים של ה-BIOS ומתפקדת התאם.

בשטח זיכרון מיוחד בהתחלה של הזיכרון של המחשב (מיד אחרי השטח של IV, כלומר מכתובת 1024 ואילך) ישנו שטח זיכרון של ה-BIOS שמכיל בין השאר שטח המשקף את מצב המקלדת, איזה מקשים נלחצו וטרם נקראו ע"י שום תוכנית, מצב הנורות (Caps Lock, Insert, Scroll, Num), מצב מקשי הסטטוס (Shift, Alt), Ctrl וכו'. למשל בכתיים בכתובות מוחלטות 417h (1041) ו- 418h (1042) ישנם בתיים המהווים משתני דגלים של מצב הנורות ומקשי הסטטוס. נוסף לכך יש שם, חוצץ לשמירת מידע על עד ל-20 לחיצות מקשים שלא נקראו עדיין (מעבר לכך התגובה ללחיצות נוספות יהיה צפוף). השטח הזה מתוחזק ומעודכן בעיקר ע"י ה-ISR של פסיקה מספר 9, פסיקת החומרה של המקלדת. המידע שנשמר שם הוא על לחיצות על (כמעט) כל המקשים של המקלדת, כולל המקשים Page Down, F1 - F12, Esc, Ins, Del ... וכו'.

כאשר תוכנית מבקשת מ-INT 16h אינפורמציה על לחיצת מקש, האינפורמציה מגיעה בשתי צורות: קוד Ascii וקוד Scan. קוד ה-Ascii הוא בדרך כלל מה שאנחנו צריכים, למשל אם נלחץ על המקש המסומן A נקבל או את הקוד Ascii 'a' או 'A' בהתאם למצב נורת ה-Caps Lock ומקשי ה-Shift וכו'. קוד ה-Scan הוא מספר יחודי לכל מקש, בתחום 1 - 127. צריך לזכור שיש מקשים שאין להם קוד Ascii (כמו F1, Esc, Page Down) ויש קודי Ascii שיש להם יותר ממקש אחד (למשל הספרות העשרוניות, הסימנים +, -, *, /).

ל-INT 16h אופציות רבות, הנבחרות לפי הערך של האוגר AH ברגע הקריאה, השימושים ביותר מבחינתנו הם:

INT 16h,

AH = 0, קריאת מקש:

קריאה ל-INT 16h כאשר AH = 0 פירושו בקשה לקבלת מידע על לחיצת מקש של המשתמש. אם אין מידע מסוג זה בהמתנה, החזרה מהקריאה תתעקב עד שחיצה כזו תתבצע (כלומר התוכנית תעצר לזמן בלתי מוגבל). עם החזרה (בין עם אחרי המתנה ובין שלא) יהיו באוגר AX האינפורמציה המבוקשת: ב-AH יהיה קוד ה-Ascii של המקש שנלחץ ואילו ב-AH יהיה קוד Scan של המקש. אותה לחיצת מקש שהמידע עליה מוחזרת לקורא ל-INT 16h נחשבת ל"נקראה" קרי "מבוטלת" כלומר שקריאה נוספת ל-INT 16h אופציה AH = 0 לא תתיחס אליה ותקרא את האינפורמציה

על הלחיצה הבאה. זה נראה מובן מאליו, אבל זה לא נכון לאופציה הבאה.

,INT 16h

AH = 1, עיון במקש:

האופציה הזו דומה לאופציה AH = 0, אבל במספר הבדלים מהותיים. ראשית, הקריאה תמיד חוזרת מיד לקורא ל-INT 16h, גם אם אין אינפורמציה על לחיצת מקש שטרם נקראה. התוכנית הקוראת יכולה להבחין אם היתה לחיצת מקש או לא לפי הערך של הדגל ZF, $ZF = 1$ אם לא היתה אינפורמציה של לחיצת מקש בהמתנה לקריאה, אם היתה $ZF = 0$. במידה והיתה אינפורמציה ללחיצת מקש ממתינה, היא תוחזר ב-AX ממש כמו ב-INT 16h אופציה AH = 0, אך בניגוד למקרה הקודם הקריאה ל-INT 16h אופציה AH = 1 אינה "מבטלת" את הלחיצה. הקריאה הבאה ל-INT 16h באופציה AH = 0 או AH = 1 יקראו את אותו מקש עצמו. כלומר קריאה ל-INT 16h אופציה AH = 1 היא קריאה "לא מחייבת" של המקש, רק מעין הצצה לבדוק מה יש שם, אם בכלל. למשל, אם נרצה לבדוק אם יש אינפורמציה על לחיצת מקש ממתינה ובמידה ויש לקרוא אותה אבל מבלי להמתין ללחיצה כזו במידה ואין, אנחנו נקרא קודם ל-INT 16h אופציה AH = 1, נבדוק ש- $ZF = 0$ ובמידה וזה המצב, נקרא ל-INT 16h אופציה AH = 0.

הקוד יכול להראות כך:

```
MOV AH,1
INT 16h
JZ No_Infol
MOV AH,0
INT 16h
JMP With_Infol
No_Infol:
```

....

With_Infol:

,INT 16h

AH = 2, סטטוס המקלדת. מחזיר ב-AL אחד משני בתי הדגלים של המקשים המיוחדים (נורות ה-Caps Lock, Num, Scroll, מצב מקשי ה-Shift, Alt, Ctrl). מידע נוסף ניתן לקרוא מכתובת מוחלטת 41h או ע"י קריאה ל-INT 16h אופציה AH = 12h.

,INT 16h

AH = 5, הדמית מקש. אפשר בתוכנית ליצור מצב שבו "כאילו" נלחץ מקש כלשהוא, כאשר האינפורמציה של לחיצת המקש אותו רוצים לדמות מועברת דרך CX: CL יהיה קוד ה-Ascii ו-CH קוד ה-Scan.

Type 16 Interrupt Operations

The Type 16 (Keyboard I/O) interrupt calls `KEYBOARD_IO`, which starts at location `F000:E82E`. This routine lets you select from three different operations, based on a value in `AX`:

- If `AX=0`, `KEYBOARD_IO` reads the scan code of the next key in the buffer into `AX` and its character code into `AL`, then advances the buffer pointer. If the buffer is empty, `KEYBOARD_IO` waits for a key to be pressed before proceeding.
- If `AX=1`, `KEYBOARD_IO` returns the status of the keyboard buffer in the Zero Flag (ZF). If the buffer is empty, ZF is 1; if any key codes are available for reading, ZF is 0. If ZF is 0, the next available character is in `AX`, and the entry remains in the buffer.
- If `AX=2`, `KEYBOARD_IO` returns a keyboard status byte in `AL`. A description of this byte follows.

The `KEYBOARD_IO` routine affects only `AX` and the flags.

The upper half of Figure 6-6 shows the arrangement of the status byte returned by the `AX=2` option. In this byte (KB_FLAG in the BIOS listing) the upper four bits tell you whether various keyboard modes are on (1) or off (0), and the lower four bits tell you whether the `Alt`, `Ctrl`, or shift key is being pressed.

The lower half of Figure 6-6 shows a companion byte to `KB_FLAG` that gives additional keyboard status information. The `KEYBOARD_IO` routine uses this second byte (`KB_FLAG_1`) internally but provides no way to read it into a register. However, `KB_FLAG_1` follows `KB_FLAG` in the BIOS, so to find out how `KB_FLAG_1` is configured, you read the contents of location `4171H` (`KB_FLAG` is at `4170H`).

The first `KEYBOARD_IO` option, `AX=0`, is convenient for setting up interactive operations with the computer. In such operations you essentially tell the computer to wait for an operator to type something at the keyboard before proceeding. Let's take a look at some possible applications.

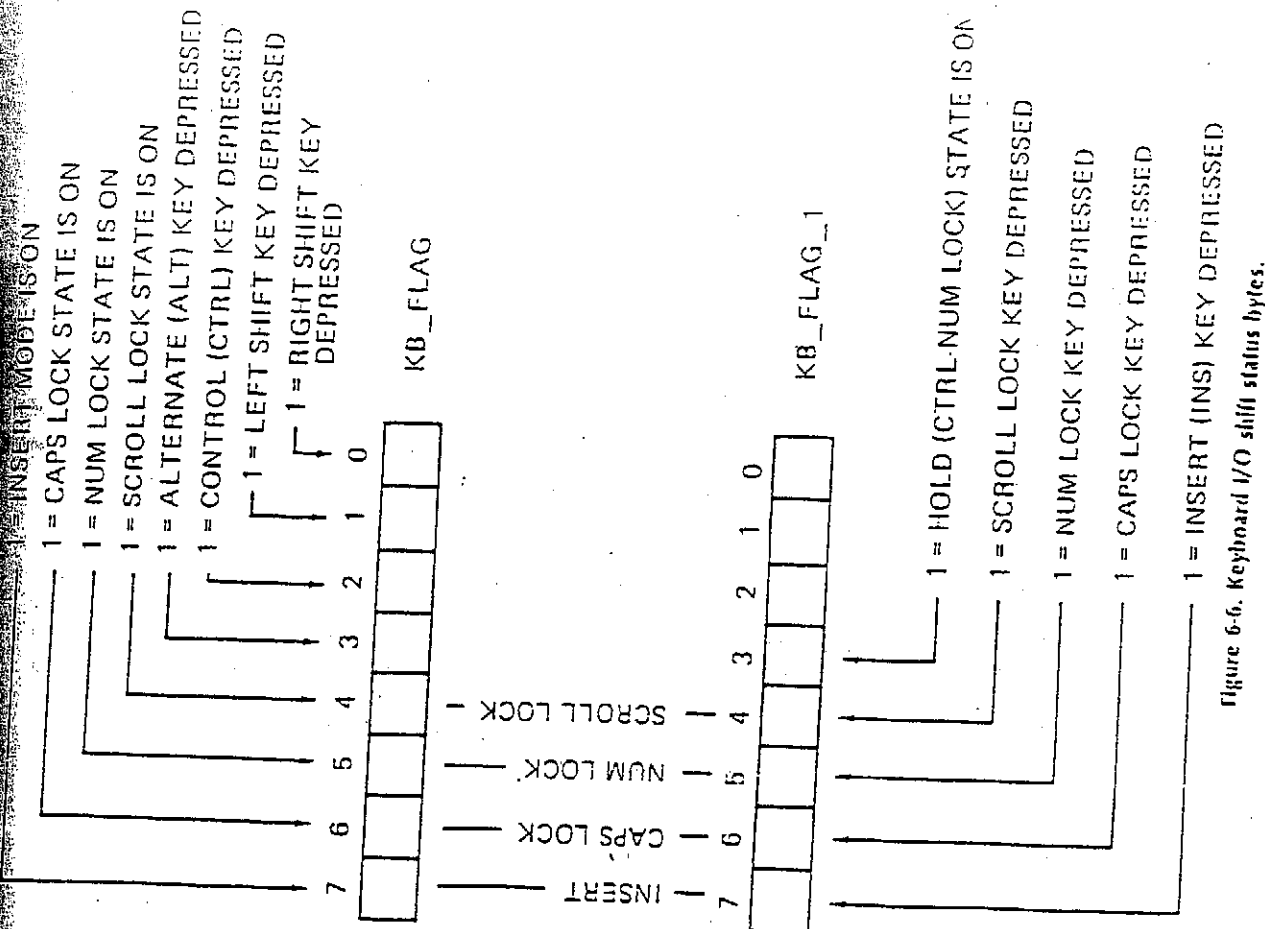
A Single-Key Read Operation

Suppose your program has displayed a message that instructs the operator to press either "Y" or "N" to signify Yes or No. A response makes the program jump to a set of instructions labeled YES, and an N makes it jump to instructions labeled NO. Any other key makes the program wait until it receives either Y or N. This sequence should do the job:

```

GET_KEY:  STI
          MOV     AX,0
          INT     16H
          Enable Interrupts
          Read a key

```



תוכניות דוגמא dem_key1.c, dem_key5.c

התוכניות הללו תפקידן להמחיש שימוש ב-INT 16h. dem_key1.c ממחישה שימוש באופציה AH = 0 קריאת מקש מהמקלדת עם המתנה אם יש צורך בכך. dem_key5.c ממחישה שימוש באופציה AH = 2 בדיקת סטטוס המקלדת.

התוכנית dem_key1.c

קטע ה-inline assembler

```
MOV AH,0
INT 16h
MOV scan_temp,AH
MOV inp_char,AL
```

הינו קריאת מקש (עם המתנה) והצבת ה-Scan code לתוך משתנה ה-C scan_temp והצבת ה-Ascii code לתוך משתנה ה-C inp_char. משם תוכנית ה-C מדפיסה אותם בצורה נוחה למשתמש.

התוכנית הראשית מממשת לולאה החוזרת על עצמה עד לרגע שהמשתמש לוחץ על המקש Esc שהתוכנית מזהה לפי Scan code = 1. בדוגמת הריצה הספציפית הזו נלחצו

```
'9',
'Shift-9',
'r',
'Shift-r',
F1,
F10,
Shift-F1,
Shift-F10
```

שימו לב שללחיצות עצמם אין השתקפות בפלט (echo) - זה אף פעם לא אוטומטי ומיושם בתוכנה ברובד זה או אחר. מהפלט אפשר לראות שללחיצות '9' ו-'Shift-9' אותו Scan code (10) וכן"ל" ללחיצות 'r' ו-'Shift-r' (19) אך הם נבדלים ב-Ascii code ('9' לעומת 'r', 'R' לעומת 'R'). מי שיריץ את התוכנית יראה שמספרי ה-Scan code ניתנו למקשים בעיקרו של דבר לפי מיקומם במקלדת (ביחוד לאותם מקלדות של ה-PC מראשית שנות השמונים).

למקשים שאין להם Ascii code בדרך כלל מוחזר ב-AL אפס, ונסיון להדפיס אפס מתבטא בריוח. ללחיצת Esc יש משום מה Ascii code = 27 שמשום מה מוחק תו מהפלט, בגלל זה נמחק גרש בשורה האחרונה בפלט.

למקש F1 יש את Scan code = 59 ול-F10 יש את Scan code = 68 למעשה כל המקשים F1 - F10 יש Scan code = 59 - 68 בהתאמה. לחיצת המקשים הללו ביחד עם Shift גורמת לתוספת של 25 ל-Scan code אולי בגלל שאין למקשים הללו Ascii code ולא ניתן להבדיל ביניהם באמצעותו.

```

/* dem_key1.c - demonstrate use of int 16h, AH = 0 */

#include <stdio.h>

void main(void)
{
    unsigned int scan_code;
    char scan_temp, inp_char;

    printf("\nPress ESC to exit program\n");

    do {
        printf("Press any key (almost)\n:");

        asm {
            MOV AH,0          ; /* BIOS read char from buffer option */
            INT 16h           ; /* BIOS read char from buffer */
            MOV scan_temp,AH  ; /* Transfer scan code to program */
            MOV inp_char,AL   ; /* Transfer char to program */
        }

        scan_code = (unsigned int) scan_temp;

        if (scan_code == 1)
            printf("You pressed ESC, \n");

        printf("You pressed key assigned"
               " scan code = %d, char_value= '%c'\n",
               scan_code, inp_char);

    } while(!(scan_code == 1));

} /* main */

```

E:\>dem_key1

```

Press ESC to exit program
Press any key (almost)
:You pressed key assigned scan code = 10, char_value= '9'
Press any key (almost)
:You pressed key assigned scan code = 10, char_value= '('
Press any key (almost)
:You pressed key assigned scan code = 19, char_value= 'r'
Press any key (almost)
:You pressed key assigned scan code = 19, char_value= 'R'
Press any key (almost)
:You pressed key assigned scan code = 59, char_value= ' '
Press any key (almost)
:You pressed key assigned scan code = 84, char_value= ' '
Press any key (almost)
:You pressed key assigned scan code = 68, char_value= ' '
Press any key (almost)
:You pressed key assigned scan code = 93, char_value= ' '
Press any key (almost)
:You pressed ESC,
You pressed key assigned scan code = 1, char_value= '

```

E:\>

התוכנית dem_key5.c משתמשת ברוטינה ה-BIOS AH = 2, INT 16h לבדיקת סטטוס המקלדת. הרוטינה הזו מחזירה את בית הדגלים של המקלדת KB_FLAG ב-AL. התוכנית גם קוראת 2 הבתים בכתובות אבסולוטיות 417h (1041) ו-418h (1042) כלומר קצת מעבר ל-IV. כפי שאנחנו רואים בית 417h הינו בעצם KB_FLAG ואילו בית 418h הוא בעצם KB_FLAG_1. התוכנית מדפיסה את הסיביות באמצעות טכניקה בשפת C המאפשר גישה לסיביות של בתי זכרון הנקרא bit fields. שימו לב שתוכנית יכולה באמצעות KB_FLAG ו-KB_FLAG_1 לדעת אם המקשים Shift, Ctrl, Alt לחוצים (אפילו להבדיל בין 2 ה-Shift-ים), לדעת אם הנוריות Caps lock, Num lock, Scroll Lock דלוקות או לא ואפילו לדעת אם המקשים הללו לחוצים כרגע להבדיל ממצב הנוריות (הנוריות יכולות להיות כבויים גם אם המקש לחוץ). ניתן גם להבחין אם המקלדת נעולה ע"י מפתח (Hold) דבר שנתמך בחלק מה-PC-ים.


```

/* dem_key5.c - demonstrate use of int 16h, AH = 2, including aux byte. */

#include <stdio.h>

typedef struct status_byte
{
    unsigned int right_shift_key : 1;
    unsigned int left_shift_key : 1;
    unsigned int ctrl_key : 1;
    unsigned int alt_key : 1;
    unsigned int scroll_lock_on : 1;
    unsigned int num_lock_on : 1;
    unsigned int caps_lock_on : 1;
    unsigned int insert_mode_on : 1;
} STATUS_BYTE;

typedef struct aux_byte
{
    unsigned int unused : 3;
    unsigned int hold_on : 1; /* ctrl-num lock */
    unsigned int scroll_lock_depressed : 1;
    unsigned int num_lock_depressed : 1;
    unsigned int caps_lock_depressed : 1;
    unsigned int insert_key_depressed : 1;
} AUX_BYTE;

void main(void)
{
    STATUS_BYTE sbyte, sbyte1;
    char *p, *p1;
    AUX_BYTE abyte;

    asm {
        MOV AH,2          ; /* BIOS read keyboard status option */
        INT 16h           ; /* BIOS read keyboard status */
        MOV BYTE PTR sbyte,AL ; /* Transfer char to program */
        PUSH ES
        MOV AX,0
        MOV ES,AX          /* ES = 0 */
        MOV AL,ES:[418h]    /* read phisical locations 417h, 418h */
        MOV BYTE PTR abyte,AL
        MOV AL,ES:[417h]
        MOV BYTE PTR sbyte1,AL
        POP ES
    }

    p = (char *) &sbyte;
    p1 = (char *) &sbyte1;

    printf("\n sbyte = %d, sbyte1 = %d\n", (int) *p, (int) *p1);
}

```

```

printf("\nKeyboard status:\n\n");
printf("Right shift: %d,\nLeft shift: %d,\nCtrl key: %d,\nAlt key: %d,\n",
    (unsigned int) sbyte.right_shift_key,
    (unsigned int) sbyte.left_shift_key,
    (unsigned int) sbyte.ctrl_key,
    (unsigned int) sbyte.alt_key);

printf("Scroll lock on: %d,\nNum lock on: %d,"
    "\nCaps Lock on: %d,\nInsert mode: %d",
    (unsigned int) sbyte.scroll_lock_on,
    (unsigned int) sbyte.num_lock_on,
    (unsigned int) sbyte.caps_lock_on,
    (unsigned int) sbyte.insert_mode_on);
putchar('\n');

printf("\nAuxiliary byte:\n\n");
printf("Hold on: %d,\nScroll lock dep: %d,\nNum lock dep: %d,"
    "\nCaps Lock dep: %d,\nInsert key dep: %d",
    (unsigned int) abyte.hold_on,
    (unsigned int) abyte.scroll_lock_depressed,
    (unsigned int) abyte.num_lock_depressed,
    (unsigned int) abyte.caps_lock_depressed,
    (unsigned int) abyte.insert_key_depressed);
putchar('\n');

} /* main */

```

E:\>DEM_KEY5

sbyte = 97, sbyte1 = 97

Keyboard status:

Right shift: 1,
 Left shift: 0,
 Ctrl key: 0,
 Alt key: 0,
 Scroll lock on: 0,
 Num lock on: 1,
 Caps Lock on: 1,
 Insert mode: 0

Auxiliary byte:

Hold on: 0,
 Scroll lock dep: 1,
 Num lock dep: 1,
 Caps Lock dep: 0,
 Insert key dep: 0

E:\>

התוכנית המשולבת הזו ממחישה איך ב-DOS תוכנית אפליקציה יכולה להתשמש במנגנון הפסיקות לצרכיה ובאופן עקרוני איך מערכות הפעלה עושות זאת היום. העיקר בתוכניות הדוגמא הללו הוא מימוש mysleep, מעין מימוש פרטי של הרוטינה sleep של שפת C. כמו ב-sleep הסטנדרטי mysleep מקבלת פרמטר שלם אותו היא מפרשת כזמן ההרדמה בשניות ועוצרת את התוכנית הקוראת לה לזמן הזה. זהו שירות שימושי למדי המאפשרת לתוכנית לממש אלמנטים זמניים (למשל תצוגה זמנית על המסך).

המימוש של השרות הזה כאן הוא באמצעות השתלטות על פסיקה 8 (פסיקת הזמן) ומתוך הנחה שהפסיקה מתרחשת 18.2065 פעמים בשניה.

מה שהמימוש של mysleep עושה למעשה הוא קביעת רוטינה פנימית שלו Timer בתור הרוטינה מטפלת בפסיקה 8 (תוך שימור כתובת רוטינת הטיפול המקורית), התמדת זמן ההרדמה משניות לפסיקות שעון ע"י הכפלה ב-182065 וחילוק ללא שארית ב-10000. את התוצאה mysleep שומר ב-AX. זהו דיוק מספיק בכדי לממש עיכוב בדיוק גבוה למספר רב למדי של שניות.

mysleep מאתחל מונה בשם counter באפס. הרוטינה Timer מקדם את המונה הזה פעם אחת לכל פסיקת שעון, ו-mysleep ממש לולאה המשווה את זמן העיכוב בפסיקות שעון לערך של המונה בלולאה שהוא יוצא ממנה רק כאשר המונה עובר את זמן ההרדמה. למעשה העיכוב בזמן מתבצע בפועל כאן.

פקודות הלולאה הם

```
MOV Counter,0
Delay:
CMP AX,Counter
JA Delay
```

מיד אחרי יציאה מהלולאה הזו mysleep משחזר את כניסה 8 ב-IV וחוזר לקוד הקורא.

נקודה טכנית היא העובדה שפסיקה 8 היא מאותם פסיקות שיש להודיע לחומרה שהיא טופלה ואחת הדרכים לדאוג לכך היא לקרוא רוטינת הטיפול בפסיקה 8 המקורית. לפיכך הקוד של Timer הוא

```

Timer PROC FAR
;
    PUSHF
    CALL DWORD PTR Timer_Offs
;
    INC Counter
;
    IRET
;
Timer ENDP

```

כאן אני מנצל את העובדה ש-Timer_Seg נמצא מיד אחרי Timer_Offs. השימוש בסגמנט הקוד לאכסון מידע הוא נוח במימוש פסיקות, כי על הערך של CS תמיד אפשר לסמוך.

שימו לב שכתובת אבסולוטית 32 ($4 \times 8 = 32$) עד 35 היא הכתובת כל כניסה מספר 8 ב-IV. הפקודות

```

MOV AX,ES:[32]
MOV Timer_Offs
MOV AX,ES:[34]
MOV Timer_Seg,AX

```

הם פקודות השימור של הכניסה המקורית ואילו הפקודות

```

CLI
MOV WORD PTR ES:[32],OFFSET Timer
MOV ES:[34],CS
STI

```

הם קביעת הרוטינה Timer כרוטינת הטיפול בפסיקה. כאן אני מנצל את העובדה שאני יודע ש-Timer ו-mysleep נמצאים באותו תאור סגמנט, דבר שלא היה בהכרח נכון אם הם היו בקבצים נפרדים.

```

/* testslp1.c - test mysleep */

#include <stdio.h>

extern void mysleep( int secs );

void main()
{
    int n=20;

    printf("Before mysleep(%d):\n", n);
    mysleep(n);
    printf("After mysleep(%d):\n", n);
} /* main */

```

```

C:\temp>tcc -ml -r- testslp1.c mysleep1.asm
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
testslp1.c:
mysleep1.asm:
Turbo Assembler Version 3.1 Copyright (c) 1988, 1992 Borland
International

Assembling file:    mysleep1.ASM
Error messages:     None
Warning messages:   None
Passes:             1
Remaining memory:   390k

Turbo Link Version 5.0 Copyright (c) 1992 Borland International

Available memory 4107872

```

```

C:\temp>testslp1
Before mysleep(20):
After mysleep(20):

C:\temp>

```

```

; mysleep1.asm - demonstrate Interrupt service routine
;
    .MODEL LARGE
    .STACK 100h
    .DATA
C182 DD 182065
C100 DD 10000
;
    .CODE
    .386
Timer_Offs DW 0
Timer_Seg DW 0
Counter DW 0
;
; My Ctrl-Break ISR
;
Timer PROC FAR
;
    PUSHF
    CALL DWORD PTR Timer_Offs
;
    INC Counter
Return:
    IRET ;
;
Timer ENDP

```

```

;
;
_mysleep PROC FAR
    PUBLIC _mysleep
    PUSH BP
    MOV BP,SP
    PUSH ES
;
    MOV AX,0
    MOV ES,AX
;
;
;
    MOV AX,ES:[32]          ; Preserve original ISR pointers
    MOV Timer_Offs,AX
    MOV AX,ES:[34]
    MOV Timer_Seg,AX
;
; Change Timer pointers
CLI
MOV WORD PTR ES:[32],OFFSET Timer
MOV ES:[34],CS
STI
MOV AX,DS
MOV ES,AX
;
;
;
    XOR EAX,EAX
    MOV AX,[BP+6]          ; Retrieve secs parameter
; Compute AX = secs * 18.2065 to convert to ticks
    MUL C182
    DIV C100

    MOV Counter,0          ; Init counter
Delay1:
;
    CMP AX,Counter          ; wait n secs
    JA Delay1
;
; Return to caller
; Restore old Timer
;
;
    MOV AX,0
    MOV ES,AX
    MOV AX,Timer_Offs
    CLI
    MOV ES:[32],AX
    MOV AX,Timer_Seg
    MOV ES:[34],AX
    STI
;
    POP ES
    POP BP
    RET                    ; return to caller
ENDP _mysleep
END

```

```

/* msleep2.c - Implement sleep myself, Pure C version */

#include <stdio.h>
#include <dos.h>

volatile int My_Sleep_Counter;

void interrupt (*Int8Save) (void); /* Pointer to function */

void interrupt My_Int8_Handler(void)
{
    asm {
        /* Notify hardware:
           Interrupt has been serviced */
        PUSHF
        CALL DWORD PTR Int8Save
    }

    My_Sleep_Counter--;
} /* My_Int8_Handler */

void my_sleep(int secs)
{
    Int8Save = getvect(8);          /* Preserve old pointer */
    setvect(8, My_Int8_Handler);    /* Set entry to new handler */

    My_Sleep_Counter = secs*182/10;

    while(My_Sleep_Counter > 0)
    {
        ;
        setvect(8, Int8Save);      /* Restore old pointer */
    } /* my_sleep */

    void main(void)
    {
        int secs;

        printf("Enter sleep time in seconds:\n");
        scanf("%d", &secs);

        system("time");
        printf("sleeping ---\n");

        my_sleep(secs);

        system("time");

        printf("Terminating ...\n");

    } /* main */
}

```

```

E:\USERS\EYTAN\ASM>tcc -ml -r- msleep2.c
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
msleep2.c:
Turbo Link Version 5.0 Copyright (c) 1992 Borland International

```

```

    Available memory 4121432

```

```

E:\USERS\EYTAN\ASM>msleep2
Enter sleep time in seconds:
60
Current time is 16:17:01.20
Enter new time:
sleeping ---
Current time is 16:18:02.72
Enter new time:
Terminating ...

```

```

E:\USERS\EYTAN\ASM>

```

```

/* msleep1.c - Implement sleep myself, Pure C version */

#include <stdio.h>
#include <dos.h>

volatile int My_Sleep_Counter;

void interrupt (*Int8Save) (void); /* Pointer to function */

void interrupt My_Int8_Handler(void)
{
    asm {
        /* Notify hardware:
           Interrupt has been serviced */
        PUSH AX
        MOV AL,20h
        OUT 20h,AL
        POP AX
    }

    My_Sleep_Counter--;
} /* My_Int8_Handler */

void my_sleep(int secs)
{
    Int8Save = getvect(8);          /* Preserve old pointer */
    setvect(8, My_Int8_Handler);    /* Set entry to new handler */

    My_Sleep_Counter = secs*182/10;

    while(My_Sleep_Counter > 0)
    ;
    setvect(8,Int8Save);            /* Restore old pointer */
} /* my_sleep */

void main(void)
{
    int secs;

    printf("Enter sleep time in seconds:\n");
    scanf("%d", &secs);

    system("time");
    printf("sleeping ---\n");

    my_sleep(secs);

    system("time");

    printf("Terminating ...\n");
} /* main */

```

```

E:\USERS\EYTAN\ASM>tcc -ml -r- msleep1.c
Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
msleep1.c:
Turbo Link Version 5.0 Copyright (c) 1992 Borland International

```

```

    Available memory 4121432

```

```

E:\USERS\EYTAN\ASM>msleep1.exe
Enter sleep time in seconds:
60
Current time is 16:14:47.13
Enter new time:
sleeping ---
Current time is 16:14:48.18
Enter new time:
Terminating ...

```

```

E:\USERS\EYTAN\ASM>

```