

בפסקית מספר 16h הינה פסקית התוכנה של המקלדת. תוכניות שמעווגינות באינפורמציה מהקלדת פונוט אליה, בדרך כלל בעקביפין דרך מערכת הפעלה והיא-caillo התוכנית הסטנדרטית לקרוא מידע מהקלדת. זהה פסקית תוכנה של ה-BIOS ובעיקרו של דבר היא מהויה מעין קוד המכיר את מבנה הנתונים של ה-BIOS ומתקדמת התאם.

בשתח זיכרונו מיוחד בהתחלה של הזיכרון של המחשב (מיד אחרי השתח של IV, ככלומר מכתובת 1024 וайлך) ישנו שתח זיכרונו של ה-BIOS שמכיל בין השאר שטח המשקף את מצב המקלדת, איזה מקשים נלחצו וטרם נקרו ע"י שם תוכנית, מצב הנורדות (Caps Lock, Insert, Scroll, Num), מצב מקשי הסטטוס (ה-Shift, Alt, Ctrl) וכו'. למשל בזמנים כתובות מוחלטות 417h (1041) ו- 418h (1042) ו- 419h (1043) וכו'. ישנים בתים המהווים משתני דגלים של מצב הנורדות ומקשי הסטטוס. נוסף לכך יש שם, חוץ לשמירת מידע על עד ל-20 לחיצות מקשים שלא נקרו עדין (מעבר לכך ע"י התגובה לחיצות נספנות יהיה אפסוף). השתח הזה מתרזוק ומעודכן בעיקר בערך ע"י ISR של פסקית מספר 9, פסקית החומרה של המקלדת. המידע שנשמר שם הוא על לחיצות על (כמעט) כל המקשים של המקלדת, כולל המקשים F1 - F12, Page Down, Esc, Ins, Del וכו'.

כאשר תוכנית מבקשת מ-16h INT אינפורמציה על לחיצת מקש, האינפורמציה מגיעה בשתי צורות: קוד ASCII וקוד Scan. קוד ASCII הוא בדרך כלל מה שאנו חנו צרייכים, למשל אם נלחץ על המקש המסומן A נקבל או את הקוד ז'א' או 'A' בהתאם למצב נורנת Caps Lock ומקשי ה-Shift וכו'. קוד Scan הוא מספר ייחודי לכל מקש, בתחום 127 - 1. לצורך זכרור שיש מקשים שאין להם קוד ASCII (כמו Page Down, Esc, F1) ויש קוד ASCII שיש להם יותר ממקש אחד (למשל הספרות העשויות, הסימנים +, -, *, /).

ל- 16h INT אופציות רבות, הנבחרות לפי הערך של האוגר AH ברגע הקראיה, השימושות ביותר מבחןתינו הם:

, INT 16h
AH = 0, קראית מקש:

קראייה ל-16h INT כאשר 0 = AH פירושו בקשה לקבל מידע על לחיצת מקש של המשתמש. אם אין מידע מסווג זה במתנה, החזרה מהקראייה תتابע עד שחייצה כזו מתבצע (ככלומר התוכנית מעצר בזמן בלתי מוגבל). עם החזרה (בין עט אחריו המתנה ובין שלא) יהיו באוגר AX האינפורמציה המבוקשת: ב-AL יהיה קוד ASCII של המקש שנלחץ וαιלו ב-AH יהיה קוד Scan של המקש. אותה לחיצת מקש שהמידע עליה מוחזרת לקרוא ל-16h INT נחשבת לנקראה" קרי "מボטלת" ככלומר שקריאה נוספת ל-16h INT AH אופציה 0 = AH לא תתייחס אליה ותקרה את האינפורמציה

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

, INT 16h

1 = AH, עיון במקש:

האופציה הזו דומה לאופציה 0 = AH, אבל במספר הבדלים מהותיים. ראשית, הקריאה תמיד חוזרת מיד לקורא ל-16h INT, גם אם אין אינפורמציה על לחיצת מקש שטרם נקרה. התוכנית הקוראת יכולה להבחן אם הייתה אינפורמציה על לא לפני הערך של הדגל ZF, ZF = 1 אם לא הייתה אינפורמציה של לחיצת מקש בחמתנה לקריאה, אם הייתה 0 = ZF. במידה והיתה אינפורמציה לחיצת מקש ממתינה, היא תוחזר ב-AX ממש כמו ב-16h INT אופציה 0 = AH, אך בנסיבות למקרה הקודם הקריאה ל-16h INT אופציה 1 = AH איינה "מבטלת" את הלחיצה. הקריאה הבאה ל-16h INT באופציה 0 = AH או 1 = AH יקרה את אותו מקש עצמו. ככלומר קריאה ל-16h INT אופציה 1 = AH היא קריאה "לא חייבות" של המקש, רק מעין הצעה לבדוק מה יש שם, אם בכלל. למשל, אם נרצה לבדוק אם יש אינפורמציה על לחיצת מקש ממתינה ובמקרה ויש לקרוא אותה אבל מבלי להמתין לחיצת כזו במידה ואין, אנחנו נקרא קודם ל-16h INT אופציה 1 = AH, נבדק ש-0 = ZF ובמקרה וזה המצב, נקרא ל-16h INT אופציה 0 = AH.

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

2 = AH, סטטוס המקלדת. מחזיד ב-AL אחד שני בתים הדגמים של המקשים המיוחדים (נורות ה-Alt, Caps Lock, Num, Scroll, Ctrl). מצב מקשי ה-Shift.). מידע נוסף ניתן לקרוא מכוחבת מוחלטת 418h או ע"י קריאה ל-16h INT אופציה AH = 12h.

, INT 16h

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

Type 16 Interrupt Operations

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

- If AL=0, KEYBOARD_I/O reads the scan code of the next key in the buffer pointer, then advances the buffer pointer. If the buffer is empty, KEYBOARD_I/O waits for a key to be pressed before proceeding.
- If AL=1, KEYBOARD_I/O returns the status of the keyboard buffer in the ZF 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 AL=2, KEYBOARD_I/O returns a keyboard status byte in AL. A description of this byte follows.

The KEYBOARD_I/O routine affects only AX and the flags.

The upper half of Figure 6-6 shows the arrangement of the status byte returned by the AL=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_I/O 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 41171 (KB_FLAG is at 41170).

The first KEYBOARD_I/O option, AL=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 Yes 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:

```
STI          ;Enable interrupts
MOV AL,0    ;Read a key
INT
```

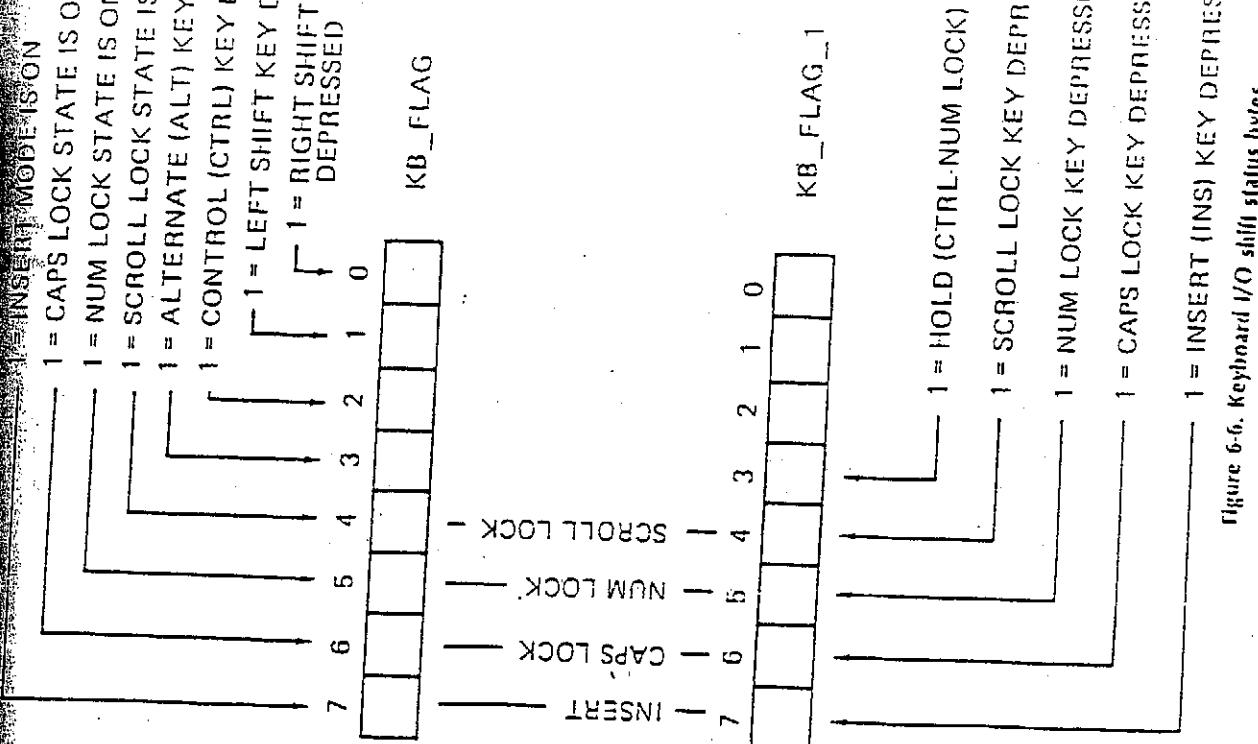


Figure 6-6. Keyboard I/O status bytes.

תוכניות דוגמא dem_key1.c, dem_key5.c

התוכניות הללו מפקידן להמיחש שימוש ב-INT 16h dem_key1.c. שימוש באופציה 0 = AH קריאת מקש מהמקלדת עם המתנה אם יש צורך בכך. אופציה 2 = AH בדיקת סטטוס המקלדת. dem_key5.c ממחישה שימוש באופציה 2 = AH 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) ומצבת Scan code לתוכן משתנה ה-C scan_temp ומצבת ה- Ascii code(inp_char) לתוכן משתנה ה-C. משם תוכנית ה-C מדפיסה אותן בצורה נוחה למשתמש.

התוכנית הראשית ממשת לולאה החזרת על עצמה עד לרגע שהמשתמש לוחץ על המKeySpecה Esc שהתוכנית מזזה לפוי 1 = Scan code. בדוגמה הריצה הספציפית זו נלחצו

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

שימוש לב שללחיצות עצם אין השתקפות בפלט (echo) - זה אף פעם לא אוטומטי ומישם בתוכנה ברובד זה או אחר. מהפלט אפשר לראות שללחיצות '9' ו-'9-Shift' אותו (10) Scan code ('9') ASCII code ('9-Shift') (19) אך הם נבדלים ב- ('9') לעומת ('9-Shift'). מי שיריץ את התוכנית יראה שמספרי ה-Scan code ניתנו למקשים העיקריים של דבר לפי מיקומם במקלדת (ביחוד לאותם מקלדות של ה-PC מראשית שנות השמונים).

למקשים שאין להם Ascii code בדרך כלל מוחזר כ-AL אפס, ונסيون להדפיס אפס מ恰恰טא ברווח. לחיצת Esc יש משום מה Ascii code = 27 שמשורם מה מוחק תר מהפלט, בגלל זה נמחק גרש בשורה האחרונה בפלט.

למקש F1 יש את Scan code = 59 ול-F10 יש את Scan code = 68 למעשה כל המKeySpecים Scan code = 59 - 68 F1 - F10 יש בההתאמה. לחיצת המKeySpecים הללו ביחד עם Shift גורמת לתוספת של 25 ל-Scan code=25 או לי בgalל שאין למקשיים 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

התוכנית dem_key5.c משתמשת בROUTINE ה-INT 16h, AH = 2 BIOS לבדיקה סטטוס המקלדת. הרוטינה הזאת מחזירה את בית הדגלים של המקלדת KB_FLAG ב-AL. התוכנית גם קוראת 2 הbiteם בכתובות אבסולוטיות 417h (1041) ו-418h (1042) כלומר קצר מעבר ל-IV. כפי שאנו 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 בתים זכרוּן הנקרא bit fields. שימוש לב שתוכנית יכולה באמצעות FLAG ו-KB_FLAG_1 לדעת אם המקלדים לחוצים (אפשר להבדיל בין Caps lock, Num lock, Scroll Lock, Shift, Ctrl, Alt 2 ה-Shift-ים), לדעת אם הנוריות כרגע מוצב דלוקות או לא ואיפילו לדעת אם המקלדים הללו לחוצים כרגע להבדיל מ מצב הנוריות (הנוריות יכולות להיות כבויות גם אם המקש לחוץ). ניתן גם להבחין אם המקלדת נעולה ע"י מפתח (Hold) דבר שנתמך בחלק מה-PC-ים.

```

/* dem_key5.c - demostrate 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_BYTEx;

void main(void)
{
    STATUS_BYTE sbyte, sbyte1;
    char *p, *p1;
    AUX_BYTEx 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, sbytel = 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:\>

תוכניות דוגמא 3m sleep1.asm

התוכנית המשולבת זו ממחישה איך ב-DOS תוכנית אפליקציה יכולה להתmesh במנגנון הפסיכות לצרכיה ובאופן עקרוני איך מערכות הפעלה עשוות זאת היום. העיקר בתוכניות הדוגמא הללו הוא מימוש mysleep, מעין מימוש פרטי של הרוטינה sleep של שפת C. כמו ב-קוד sleep הSTDRTI mysleep מקבל פרמטר שלם אותו היא מפרש בזמן הרדמה בשניות ועוצרת את התוכנית הקוראת לה זמן זהה. זהו שירות שימושי למדיום המאפשרת לתוכנית למש אלמנטים זמינים (למשל תצוגה זמנית על המסך).

המיושן של השירות הזה כאן הוא באמצעות השתלטת על פסיקה 8 (פסיקת הזמן) ומtower הנחיה שהפטיקה מתרחשת 2065.18.02. פעים בשניה.

מה שהמיושן של sleep מושן הוא קביעת רוטינה פנימית שלו Timer בתוך הרוטינה מטפלת בפסיקה 8 (Tower Shmord כתוכנת רוטינה הטיפול המקורי), התמרת זמן ההרדמה משניות לפסיקות שעון ע"י הכפלה ב-182065 וחילוק ללא שארית ב-10000. את התוצאה sleep שמור ב-AX. זהו דיקט מספיק בכדי למש עיצוב בדיקות גבורה למספר רב למדיום של שניות.

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

פקודות הלולאה הם

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

מיד אחרי יציאה מהלולאה הזר sleep משוחרר את כניסה 8 ב-IV ו חוזר לקוד הקורא.

נקודה טכנית היא העורבה שפסיקה 8 היא מאוחת פסיקות שיש להודיע לחומרה שהיא טופלה ואחת הדריכים לדאוג לכך היא לקרוא רוטינת הטיפול בפסיקה 8 המקורי. לפיכך הקוד של Timer הוא

```

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

```

כאן אני מנצל את הurbedה Seg_Timer ש-Timer_Offs נמצא מיד אחרי השימוש בסegment הקוד לאכסון מידע הוא נוח בימוש פסיקות, כי על הערך של CS חמיד אפשר לסמור.

שימוש לב שכחובת אבסולוטית 32 עד 35 היה כתובת כל כניסה מסטר 8 ב-VI. הפקודות

```

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

```

הן פקודות השימוש של הכניסה המקורית ואיילו הפקודות

```

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

```

הן קביעה הרוטינה Timer כרוטינה הטיפול בפסיקה. כאן אני מנצל את הurbedה שאני יודע ש-Timer ו-mysleep נמצאים באותו תאור Segment, 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>

```