ПРИСТАП ДО ТАСТАТУРАТА ОД BIOS-ОТ

Прекинот 16h предвидува три функции за читање на тастатурата и статусот на тастатурата. BIOS функциите за тастатурата се многу ограничени: на пример не постои BIOS функција за бришење на знак од баферот на тастатурата или функција за преименување на применият знак. DOS функциите може да го направат ова.

Извесни комбинации на тастери не можат да се прочитаат од BIOS-от како код на тастери бидејќи тие извршуваат соодветни акции. На пример, притискање на <PrtSc> тастерот го повикува BIOS прекинот 5h кој активира процедура што го испраќа тековниот екран на печатачот.

<Ctrl> - <Num Lock> комбинација од тастери го стопира целиот систем се додека корисникот не притисне некој тастер.

Притискање на <Ctrl> - <Break> комбинација на тастерите го повикува прекинот 1Bh. Нормално тековната програма се стопира и се враќа на DOS. За да се избегне ова овој прекин може да се пренасочи во процедура која го продолжува изведувањето на програмата.

AT компјутерите имаат тастер <Sys Req> со чие притискање се повикува прекинот 15h при што во AX регистарот се пренесува вредност 8500h. При отпуштање на овој тастер во AX регистарот се пренесува вредност 8501h. Вредноста 85h во AH регистарот го дава бројот на функцијата од прекинот 15h. Функцијата 85h од BIOS прекинот 15h содржи само IRET инструкција - што значи притискање на <Sys Req> нема видливи ефекти.

Функција 0: Читање на тастатурата

Прекинот 16h го прифаќа повикот кога програмата очекува кориснички внес од еден или повеќе знаци. Ако знакот е веќе внесен пред повикот на функцијата, баферот на тастатурата го празни овој знак и го пренесува до програмата. Ако во баферот на тастатурата нема знак, функцијата 0 чека додека не се внесе знак и потоа се враќа на повикувачкиата програма. Повикувачот може да детектира знак или активност на тастер во зависност од содржината на AH и AL регистрите.

ASCII кодови на тастатурата

Ако AL регистарот содржи вредност различна од 0, тогаш го содржи ASCII кодот од знакот. AH регистарот го содржи "scan" кодот од активниот тастер. Некои разлики има во контролните тастери:

Code	Key (s)
8	<Backspace>
9	<Tab>
10	<Ctrl><Return>
13	<Return>
27	<Esc>

Проширени кодови на тастатурата

Ако после повикот на прекинот во AL регистарот се содржи вредност 0, тогаш AH регистарот го содржи проширениот код на тастатурата. Разликата помеѓу ASCII кодот и проширениот код е во тоа што извесни тастери (на пример стрелките за движење на покажувачот) не можат да се сместат во 256-знаковното множество. Следната табела дава преглед на проширеното множество на кодови на тастатурата:

Code(s)	Key(s)
15	<Shift> <Tab>
16 - 25	<Alt> <Q>, <W>, <E>, <R>, <T>, <Y>, <U>, <I>, <O>, <P>
30 - 38	<Alt> <A>, <S>, <D>, <F>, <G>, <H>, <J>, <K>, <L>
44 - 50	<Alt> <Z>, <X>, <C>, <V>, <B>, <N>, <M>
59 - 68	<F1> - <F10>
71	<Home>
72	<Cursor Up>
73	<Page Up>
75	<Cursor Left>
77	<Cursor Right>
79	<End>
80	<Cursor Down>
81	<Page Down>
82	<Insert>
83	<Delete>
84 - 93	<Shift> <F1> - <F10>
94 - 103	<Ctrl> <F1> - <F10>
104 - 113	<Alt> <F1> - <F10>
115	<Ctrl><Cursor Left>
116	<Ctrl><Cursor Right>
117	<Ctrl><End>
118	<Ctrl><Page Down>
119	<Ctrl><Home>
120 - 131	<Alt> <1>, <2>, <3>, <4>, <5>, <6>, <7>, <8>, <9>, <0>
132	<Ctrl><Page Up>

Функција 1: Читање на тастатурата

Функцијата 1 чита знак од тастатурата. За разлика од функцијата 0, функцијата 1 го остава знакот во баферот на тастатурата. Исто како и во функцијата 0, AL содржи вредност различна од 0 ако корисникот внесе проширен код, или вредност 0 ако е внесен нормален код од тастатурата.

Функција 2: Читање на контрола на тастатурата

Функцијата 2 има комплетно различна работа од претходните функции. Таа го чита статусот од извесни контролни копчиња и состојби (на пример <Insert>). За повикување на оваа функција во AH регистарот се сместува вредност 2. По повикот на функцијата статусот се вртаќа во регистарот AL. Изгледот на статусниот бајт е прикажан на сликата:

7	6	5	4	3	2	1	0

0 - десен SHIFT притиснат тастер
1 - лев SHIFT притиснат тастер
2 - CTRL притиснат тастер
3 - ALT притиснат тастер
4 - SCROLL LOCK поставен
5 - NUM LOCK поставен
6 - CAPS LOCK поставен
7 - INSERT поставен

С.Г.

Е.О.Т.

```
/*****************************************************/
/*                    K E Y C                        */
/*---------------------------------------------------*/
/*  Task    : Makes a function available for reading a character */
/*            from the keyboard. The upper-right corner of the   */
/*            screen lists the status of INSERT, CAPS LOCK and   */
/*            NUM LOCK.                                          */
/*---------------------------------------------------*/
/*  Memory model  : SMALL                            */
/*****************************************************/

/*== Add include files ==============================*/
#include <dos.h>
#include <bios.h>
#include <stdio.h>
/*== Type definitions ===============================*/
typedef unsigned char BYTE;                 /* Create a byte */
/*== Macros =========================================*/
#ifdef __TURBOC__                  /* Definitions for TURBO C */
 #define GetKbKey()      ( bioskey( 0 ) )
 #define GetKbReady()    ( bioskey( 1 ) != 0 )
 #define GetKbStatus()   ( bioskey( 2 ) )
#else                    /* Definitions for Microsoft C Compiler */
 #define GetKbKey()    ( _bios_keybrd( _KEYBRD_READ ) )
 #define GetKbReady()  ( _bios_keybrd( _KEYBRD_READY ) != 0 )
 #define GetKbStatus() ( _bios_keybrd( _KEYBRD_SHIFTSTATUS ) )
#endif
/*== Constants ======================================*/

/*-- Bit layout in BIOS keyboard status ---------------*/

#define SCRL  16                    /* SCROLL LOCK bit */
#define NUML  32                    /* NUM LOCK bit */
#define CAPL  64                    /* CAPS LOCK bit */
#define INS  128                    /* INSERT bit */

#define TRUE  ( 0 == 0 )            /* Constants make reading the */
#define FALSE ( 0 == 1 )            /* program code easier        */

#define FR   0   /* Row in which the flags should be displayed */
#define FC   65  /* Column in which the flags should be displayed */
#define FlagColour 0x70 /* Black characters on white background */

/*-- Codes of some keys as returned by GETKEY() -------*/
#define BEL   7                     /* Bell character code */
#define BS    8                     /* Backspace key code */
#define TAB   9                     /*    Tab key code */
#define LF    10                    /* Linefeed key code */
#define CR    13                    /* Carriage return code */
#define ESC   27                    /*    Escape key code */
#define F1    315                   /*    Function keys */
#define F2    316
#define F3    317
#define F4    318
#define F5    319
#define F6    320
#define F7    321
#define F8    322
#define F9    323
#define F10   324
#define CUP   328                   /* Cursor keys */
#define CLEFT 331
#define CRIGHT 333
#define CDOWN 328

/*== Global variables ===============================*/
BYTE Insert,                        /* INSERT flag status */
     Num,                           /* NUM flag status */
     Caps;                          /* CAPS flag status */

/*****************************************************/
/* GETPAGE : Gets the current screen page.           */
/* Input   : None                                    */
/* Output  : See below                               */
/*****************************************************/

BYTE GetPage( void )
{
 union REGS Register;     /* Register variables for interrupt call */

 Register.h.ah = 15;                    /* Function number */
 int86(0x10, &Register, &Register);     /* Call interrupt 10H */
 return(Register.h.bh);                 /* Current screen page number */
}

/*****************************************************/
/* SETPOS: Sets the cursor position in the current screen page. */
/* Input   : ScCol = New cursor column               */
/*           ScRow = New cursor row                  */
/* Output  : None                                    */
/* Info    : The screen cursor may change when you call this  */
/*           function if the current screen page is the specified */
/*           screen page.                            */
/*****************************************************/

void SetPos(BYTE ScCol, BYTE ScRow)
{
 union REGS Register;     /* Register variables for interrupt call */

 Register.h.ah = 2;                     /* Function number */
 Register.h.bh = GetPage();             /* Screen page */
 Register.h.dh = ScRow;                 /* Screen row */
 Register.h.dl = ScCol;                 /* Screen column */
 int86(0x10, &Register, &Register);     /* Call interrupt 10H */
}

/*****************************************************/
/* GETPOS: Gets the cursor position in the current screen page. */
/* Input   : None                                    */
/* Output  : ScCol = Pointer to variable containing current column */
/*           ScRow = Pointer to variable containing current row */
/*****************************************************/

void GetPos(BYTE * ScCol, BYTE * ScRow)
{
 union REGS Register;     /* Register variables for interrupt call */

 Register.h.ah = 3;                     /* Function number */
 Register.h.bh = GetPage();             /* Screen page */
 int86(0x10, &Register, &Register);     /* Call interrupt 10H */
 *ScCol = Register.h.dl;                /* Read function result */
 *ScRow = Register.h.dh;                /*  from the registers */
}

/*****************************************************/
/* WRITECHAR: Writes a character with attribute to the current */
/*            cursor position in the current screen page. */
/* Input   : CharCode = ASCII code of character to be displayed */
/*           CAttr    = Character attribute          */
/* Output  : None                                    */
/*****************************************************/

void WriteChar(char CharCode, BYTE CAttr)
{
 union REGS Register;     /* Register variables for interrupt call */

 Register.h.ah = 9;                     /* Function number */
 Register.h.al = CharCode;              /* Character to be displayed */
 Register.h.bh = GetPage();             /* Screen page */
 Register.h.bl = CAttr;                 /* Color of char. to be displayed */
 Register.x.cx = 1;                     /* Display character once */
 int86(0x10, &Register, &Register);     /* Call interrupt 10H */
}
/*****************************************************/
/* WRITETEXT: Writes a character with constant color to a specific */
/*            position within the current screen page.  */
/* Input   : ScCol = Display column                  */
/*           ScRow = Display row                     */
/*           TEXT  = Pointer to the string to be displayed */
/*           CAttr = Attribute for character to be displayed */
/* Output  : None                                    */
/* Info    : Text is a pointer to a character vector, which contains */
/*           the text to be displayed, terminated with '\0'.  */
/*****************************************************/

void WriteText(BYTE ScCol, BYTE ScRow, char *Text, BYTE CAttr)
{
 union REGS InRegister,   /* Register variables for interrupt call */
            OutRegister;

 SetPos(ScCol, ScRow);                  /* Place cursor */
 InRegister.h.ah = 14;                  /* Function number */
 InRegister.h.bh = GetPage();           /* Screen page */
 while (*Text)                          /* Display text until '\0' */
 {
  WriteChar(*Text, CAttr);              /* Write character color */
  InRegister.h.al = *Text++;            /* for character */
  int86(0x10, &InRegister, &OutRegister);    /* Call interrupt */
 }
}

/*****************************************************/
/* CLS: Clear current screen page                    */
/* Input   : None                                    */
/* Output  : None                                    */
/*****************************************************/

void Cls( void )
{
 union REGS Register;     /* Register variables for interrupt call */

 Register.h.ah = 6;                     /* Function number: Scroll Up */
 Register.h.al = 0;                     /* 0 = Clear */
 Register.h.bh = 7;                     /* White text on black background */
 Register.x.cx = 0;                     /* Upper-left corner of screen */
 Register.h.dh = 24;                    /* Coordinates of lower- */
 Register.h.dl = 79;                    /* right corner of screen */
 int86(0x10, &Register, &Register);     /* Call BIOS video interrupt */
}

/*****************************************************/
/* NEGFLAG: Negates flag and displays corresponding text.  */
/* Input   : FLAG    = Last flag status              */
/*           FLAGREG = Current flag status (0 = off) */
/*           ScCol   = Screen column for flag names  */
/*           ScRow   = Screen row for flag names     */
/*           TEXT    = Flag name                     */
/* Output  : New flag status (TRUE = on, FALSE = off) */
/*****************************************************/

BYTE NegFlag(BYTE Flag, unsigned int FlagReg,
             BYTE ScCol, BYTE ScRow, char * Text)
{
 BYTE CurScRow,                         /*   Current row */
      CurScCol;                         /* Current column */

 if (!(Flag == (FlagReg != 0)))         /* Has flag been changed? */
 {                                      /* Yes */
  GetPos(&CurScCol, &CurScRow);         /* Get current cursor position*/
  WriteText(ScCol, ScRow, Text, (BYTE) ((Flag) ? 0 : FlagColour));
  SetPos(CurScCol, CurScRow);           /* Set old cursor position */
  return(Flag ^1);                      /* Change flag bit 0 */
 }
 else return(Flag);                     /* Change back to old status */
}
```

(Целосниот листинг можете да го превземете од ИнФорма ЕИС, од именикот информ)