

IRQSTAT.C

```

/*****
/*          I R Q S T A T . C          */
/**-----**/
/* task      : indicates the status of the IRQ controller */
/**-----**/
/* author    : Michael Tischer / Bruno Jennrich          */
/* developed on : 3/12/1994                                */
/* last update : 4/06/1995                                */
/**-----**/
/* COMPILER OPT. : 1 Byte Struct Member Alignment!      */
/*               : No optimizations!                      */
/* COMPILER      : Borland C++ 3.1, Microsoft Visual C++ 1.5 */
/*****
#include <dos.h>
#include <stdio.h>
#include "types.h"
#include "win.h"
#include "irqutil.h"

/*- Remove the following lines within a project: -----*/
#include "win.c"
#include "irqutil.c"

/*- Global variables -----*/
WINDOW IRQStat1, IRQStat2, IRQCnt;
INT      iCounts[ 16 ];

/*****
/* PrintIRQ : print out all pending (Request) and up to the present */
/*          : executed (In Service) interrupts.                      */

```

```

/**-----**/
/* print out : number of executed IRQ's with the highest */
/* priority. */
/**-----**/
/* Info : The IRQ being presently executed having the highest */
/* priority is transmitted over the ISR (In Service Register) */
/* which is at the disposal of every PIC (Master und Slave). */
/* Take note that the priorities are divided */
/* as follows : */
/* PRI: 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00 */
/* IRQ: 0 1 2 8 9 10 11 12 13 14 15 3 4 5 6 7 */
/* The task of the IRR und ISR flags in this function */
/* proceed in this order! */
/*****
int _FP PrintIRQ( void )
{ static BYTE bIRRM, bIRRs, bISRM, bISRs, i;
  static INT iIRQ;
  static CHAR szIRR[ 17 ],
              szISR[ 17 ],
              szCntone [ 17 ],
              szCntten[ 17 ];

  bIRRM = irq_ReadIRR( MASTER_PIC );
  bIRRs = irq_ReadIRR( SLAVE_PIC );
  bISRM = irq_ReadISR( MASTER_PIC );
  bISRs = irq_ReadISR( SLAVE_PIC );

  win_GotoXY( &IRQStat1, 0, 0 );
  win_Print ( &IRQStat1, "IRR ");
  win_GotoXY( &IRQStat2, 0, 0 );
  win_Print ( &IRQStat2, "ISR ");
  win_GotoXY( &IRQCnt, 0, 0 );

```

```

win_Print ( &IRQCnt,    "CNT ");

iIRQ = -1;
for( i = 0; i < 2; i++ )                                /* MASTER 0,1 */
{
    szIRR[ i ] = bIRRM & 0x01 ? 'X' : '-';
    bIRRM >>= 1;
    if( bISRM & 0x01 )
        if( iIRQ == -1) iIRQ = i;
    szISR[ i ] = bISRM & 0x01 ? 'X' : '-';
    bISRM >>= 1;
}

for( i = 8; i < 16; i++ )                                /* SLAVE 8-15 */
{
    szIRR[ i ] = bIRRs & 0x01 ? 'X' : '-';
    bIRRs >>= 1;
    if( bISRs & 0x01 )
        if( iIRQ == -1)
        {
            iIRQ = i;
            iCounts[ 2 ]++;      /* increment cascading MASTER-IRQ */
        }

    szISR[ i ] = bISRs & 0x01 ? 'X' : '-';
    bISRs >>= 1;
}

for( i = 2; i < 8; i++ )                                /* MASTER 2 - 7 */
{
    szIRR[ i ] = bIRRM & 0x01 ? 'X' : '-';
    bIRRM >>= 1;
}

```

```

    if( bISRM & 0x01 )
        if( iIRQ == -1)
            iIRQ = i;
    szISR[ i ] = bISRM & 0x01 ? 'X' : '-';
    bISRM >>= 1;
}
szISR[ 16 ] = '\0';
szIRR[ 16 ] = '\0';

iCounts[ iIRQ ]++;
iCounts[ iIRQ ] %= 100;
for( i = 0; i < 16; i++ )
{
    szCntone[ i ] = '0' + iCounts[ i ] / 10;
    szCntten[ i ] = '0' + iCounts[ i ] % 10;
}
szCntone[ 16 ] = '\0';
szCntten[ 16 ] = '\0';

win_Print ( &IRQStat1, szIRR );
win_Print ( &IRQStat2, szISR );
win_GotoXY( &IRQCnt, 4, 0 );
win_Print ( &IRQCnt, szCntone );
win_GotoXY( &IRQCnt, 4, 1 );
win_Print ( &IRQCnt, szCntten );

return iIRQ;
}

/*****
/* IRQ : interrupt routine for all hardware interrupts */

```

```

/**-----**/
/* Info : By searching the IS register, the IRQ is established */
/*         with the highest priority. The old IRQ routine is */
/*         then called up. This is only possible, */
/*         when the PIC's are programed in the AT at */
/*         "established priority". */
/*****
void (_interrupt _FP *lpOldIRQ[16])( void );
void _interrupt _FP IRQ()
{ INT iIRQ;
  void (_interrupt _FP *lpFun)( void );

  _disable();
  iIRQ = PrintIRQ();
  __asm pushf
  lpFun = lpOldIRQ[ iIRQ ];
  lpFun();
  PrintIRQ();
  _enable();
}

/*****
/* InstallIRQs : Set all hardware IRQ routines */
/*               at the above mentioned handler */
/*****
void InstallIRQs( void )
{ BYTE i;

  win_Init( &IRQStat1, 80-20, 0, 20, 1, 0x1F, 0x1F, 0);
  win_Clr( &IRQStat1 );
  win_Init( &IRQStat2, 80-20, 1, 20, 1, 0x1F, 0x1F, 0);
  win_Clr( &IRQStat2 );

```

```

win_Init( &IRQCnt , 80-20, 2, 20, 2, 0x1F, 0x1F, 0);
win_Clr( &IRQCnt );
for( i = 0; i < 16; i++ )
{
    iCounts[ i ] = 0;
    lpOldIRQ[i] = irq_SetHandler( i, IRQ );
}
}

/*****
/* UninstallIRQs : Set hardware IRQ routines at the old handler.      */
*****/
void UninstallIRQs( void )
{
    BYTE i;
    for( i = 0; i < 16; i++ ) irq_SetHandler( i, lpOldIRQ[i] );
}

/*****
/*              M A I N   P R O G R A M              */
*****/
void main( void )
{
    InstallIRQs();
    _dos_keep(0, 20 * 64 );
}
/* ca. 20K allocate */

```