```
/*****************************************************************
 * Monitor til Radio Amatør Computer                             *
 * OZ1BOV - Karsten Frahm                                        *
 *****************************************************************/

#include <reg51.h>      /* interne CPU register */
#include <stdio.h>      /* standar i/o - seriel kommunikation */
#include <ctype.h>
#include <absacc.h>     /* memory */

#define true 1
#define false 0
#define RTC 0xFFE0                  /* read / write to RTC */
#define wd 0xFFFC                   /* write, reset watch dog */
#define output_0 0xFFFD             /* write output, U20 */
#define centronic_out 0xFFFE        /* write data out to centronic printer */
#define output_1 0xFFFF             /* write output, U19 - centronic control */
#define input_0 0xFFFD              /* read input, U16 */
#define dip_switch_adr 0xFFFE       /* read dip-switch */
#define input_1 0xFFFF              /* read input, U15 - centronic control */
#define ESC 0x1B                    /* ESC char, 27 dec */

int i, c;
unsigned char rtc_power_on; /* vent 1 sek efter power on til rtc initaliseres */
unsigned int adresse;       /* adresse pointer, modtaget fra seriel */
unsigned int byte_x;        /* byte data, modtaget fra seriel */
unsigned int crc_sum;       /* crc sum, bruges af get_word & get_byte */
unsigned int up_bin_timeout; /* timeout counter for binær upload */
bit reset_wd;               /* reset wd i interrupt routine, hvis tilladt */

/*********************************************************************
 * 4 byte ascii data modtages fra den serielle linie og converteres  *
 * til word, gemmes i adresse. Hvis første modtagne ascii byte er CR *
 * returneres blot fra routinen, variablen adresse røres ikke        *
 *********************************************************************/
get_word()
{
  c = _getkey();
  if ( c != 13 )
  {
    adresse = 4096 * ((int)toint(c));
    c = _getkey();
    adresse += (256 * ((int)toint(c)));
    c = _getkey();
    adresse += (16 * ((int)toint(c)));
    c = _getkey();
    adresse += ((int)toint(c));
  }
} /* get_word end */

/*********************************************************************
 * Modtager 2 byte ascii data fra den serielle linie, converter til  *
 * byte, resultatet er i byte_x.                                     *
 *********************************************************************/
get_byte()
{
  c = _getkey();        /* 2 byte ascii data til hex byte, gemmes i int */
  if ( c != 13 )
  {
    byte_x = (16 * ((int)toint(c)));
    c = _getkey();
    byte_x += ((int)toint(c));
  }
} /* get_byte end */

/*****************************************************************
 * Se lige efter om der er kommet en char på den serielle linie, *
 * hvis så returner den modtagne char, ellers returner 0         *
 *****************************************************************/
char look_key()
{
  char c;

  if ( RI )   /* RI true, der er modtaget en char fra den serielle linie */
  {
    c = SBUF;
    RI = false;
  }
  else
    c = 0;
```

```
  return (c);
}

/*************************************************************
 * frigør rtc, livet går vidre                               *
 *************************************************************/
frigor_rtc()
{
  unsigned char i;

  i = XBYTE[RTC+13];     /* control register D */
  i = i & 0xFE;          /* clear hold bit */
  XBYTE[RTC+13] = i;     /* control register D */
}

/*************************************************************
 * stop RTC så vi kan læse/skrive til den                    *
 *************************************************************/
hold_rtc()
{
  unsigned char i, q;

  do
  {
    i = XBYTE[RTC+13];     /* control register D */
    i = i | 1;             /* set hold bit, or */
    XBYTE[RTC+13] = i;     /* control register D */
    i = XBYTE[RTC+13];     /* control register D */
    i = i & 0x02;          /* mask busy bit ud, and */
    if ( i )
    {
      frigor_rtc();              /* clear hold flag */
      for ( q=0; q<150; q++);    /* delay loop, ca 200 µS */
    }
  }
  while ( i );
}

/*************************************************************
 * restart RTC efter power on                                *
 *************************************************************/
restart_rtc()
{
  XBYTE[RTC+15] = 4;       /* register F, set til 24 timers ur */
  XBYTE[RTC+13] = 0;       /* register D */
  hold_rtc();
  XBYTE[RTC+15] = 7;
  XBYTE[RTC+14] = 0;       /* register E */
  XBYTE[RTC+15] = 4;       /* register F */
  XBYTE[RTC+13] = 0;       /* register D */
  frigor_rtc();
} /* restart_rtc end */

/*************************************************************************
 * udskriver klokken ud på den serielle linie, sender nyt data           *
 * hver sekundt, indtil der modtages en char på den serielle linie       *
 * sender kun CR ud i starten af hver udskrift, derved skrives           *
 * hele tiden på samme linie på skærmen                                  *
 *************************************************************************/
clock()
{
  unsigned char sek, forige_sek, min, tim, dat, mdr, aar;

  printf("\n\nklokken og dato:\n");
  do
  {
    hold_rtc();
    sek = 0x0F & XBYTE[RTC+0];
    sek = sek + 10 * ( 0x0F & XBYTE[RTC+1] );      /* sekunder */
    min = 0x0F & XBYTE[RTC+2];
    min = min + 10 * ( 0x0F & XBYTE[RTC+3] );      /* minutter */
    tim = 0x0F & XBYTE[RTC+4];
    tim = tim + 10 * ( 0x0F & XBYTE[RTC+5] );      /* timer */
    dat = 0x0F & XBYTE[RTC+6];
    dat = dat + 10 * ( 0x0F & XBYTE[RTC+7] );      /* dato */
    mdr = 0x0F & XBYTE[RTC+8];
    mdr = mdr + 10 * ( 0x0F & XBYTE[RTC+9] );      /* måned */
    aar = 0x0F & XBYTE[RTC+10];
    aar = aar + 10 * ( 0x0F & XBYTE[RTC+11] );     /* år */
```

```
    frigor_rtc();
    if ( sek != forige_sek )
    {
      printf("%c %2bu:%2bu:%2bu", 13, tim, min, sek);
      printf(" - %2bu-%2bu-%2bu", dat, mdr, aar);
      forige_sek = sek;
    }
  }
  while ( look_key() == 0 );
} /* clock end */

/*********************************************************************
 * Set rtc med tidspunkt og dato der modtages fra den serielle linie *
 * Format TTMMSS-DDMMÅÅ, skal overholdes, ellers !!!!                *
 *********************************************************************/
set_rtc()
{

  hold_rtc();
  XBYTE[RTC+15] = 0x07;                                  /* stop RTC */
  XBYTE[RTC+5] = (unsigned char)toint(_getkey());        /* 10 time   */
  XBYTE[RTC+4] = (unsigned char)toint(_getkey());        /*  1 time   */
  XBYTE[RTC+3] = (unsigned char)toint(_getkey());        /* 10 minut  */
  XBYTE[RTC+2] = (unsigned char)toint(_getkey());        /*  1 minut  */
  XBYTE[RTC+1] = (unsigned char)toint(_getkey());        /* 10 sekund */
  XBYTE[RTC+0] = (unsigned char)toint(_getkey());        /*  1 sekund */
  _getkey();                                             /* hent bindestreg */
  XBYTE[RTC+7] = (unsigned char)toint(_getkey());        /* 10 dato   */
  XBYTE[RTC+6] = (unsigned char)toint(_getkey());        /*  1 dato   */
  XBYTE[RTC+9] = (unsigned char)toint(_getkey());        /* 10 måned  */
  XBYTE[RTC+8] = (unsigned char)toint(_getkey());        /*  1 måned  */
  XBYTE[RTC+11] = (unsigned char)toint(_getkey());       /* 10 år */
  XBYTE[RTC+10] = (unsigned char)toint(_getkey());       /*  1 år */

  do
  {
    /* vent her indtil der modtages CR */
  }
  while ( look_key() == 0);
  XBYTE[RTC+15] = 4;                                     /* start RTC igen */
  frigor_rtc();
  clock();      /* vis lige hvad klokken blev sat til */
} /* set_rtc end */


/*********************************************************************
 * initalicering af CPU registre                                     *
 *********************************************************************/
start()
{
  TL0  = 0x65;            /* timer 0 - ca 5 mS ved 11.0592 Mc */
  TH0  = 0xEA;            /* 65535 - 5530 = 60005 = 0xEA65 */
  TH1  = 0xFD;            /* timer 1 - 9600 Baud ved 11.0592 Mc */
  SCON = 0x52;            /* seriel  : 8 bit uart, variabel baud rate */
  TMOD = 0x21;            /* timer 0, mode 1 : 16 bit timer, 5 mS timer interrupt */
                          /* timer 1, mode 2 : 8 bit med autoreload, seriel timer */
  TCON = 0x50;            /* start timer 0 & 1 */
  rtc_power_on = 200;     /* 200 * 5 = 1 Sekundt, RTC power on time */
  IE   = 0x82;            /* enable timer 0 interrupt */
  XBYTE[output_1] = 0xC0; /* /strobe og /reset til centronic high */
  reset_wd = true;        /* reset WD er tilladt */
} /* start end */


/*********************************************************************
 * timer 0 interrupt. interrupt for hver 5 mS, resetter watch dog *
 *********************************************************************/
void timer0() interrupt 1 using 1
/* timer 0, register bank 1 */
{
  TL0  = 0x65;          /* timer 0 - ca 5 mS ved 11.0592 Mc */
  TH0  = 0xEA;          /* 5530 = 0x159A */
  if (reset_wd)         /* reset WD hvis det er tilladt */
    XBYTE[wd] = 1;      /* vov */
  if ( rtc_power_on > 0 )
    rtc_power_on--;
  if ( up_bin_timeout < 65000 ) /* biniær upload timeout counter */
    up_bin_timeout++;
} /* timer 0 end */
```

```
/*****************************************************************
 * seriel interrupt.                                             *
 * Bruges kun til at vække CPU'en under Idle mode                *
 *****************************************************************/
void seriel() interrupt 4
/* seriel input / output */
{
} /* seriel end */


/*********************************************************************
 * skriv hjælpe tekst ud til den serielle linie                      *
 *********************************************************************/
help()
{
  printf("\n\nMonitor for Radio Amatør Computer");
  printf("\nMonitor ver 1.0 - 22.09.1991");
  printf("\nHjælpe oversigt :");
  printf("\ne,Exxxx     : vis Eprom indhold fra adresse xxxx hex");
  printf("\nu,U         : Upload hex programfil til ram, *.HEX");
  printf("\nb,B         : Upload biniær programfil til ram, *.BIN");
  printf("\nf,Fxx       : Fyld ram memory med xx hex");
  printf("\nh,H         : denne Hjælpe oversigt");
  printf("\np,P         : seriel til Printer, ESC returner til monitor");
  printf("\nm,Mxxxx     : vis Memory indhold fra adresse xxxx hex");
  printf("\nt,T         : ram Test");
  printf("\nw,W         : dip sWitch stilling");
  printf("\nc,C         : vis Clock, tid & dato");
  printf("\ns,S         : Set clock [TTMMSS-DDMMÅÅCR] - CR (return) indlæser");
  printf("\no,O         : power dOwn");
  printf("\ni,I         : Idle mode, indtil i,I modtages igen");
  printf("\ng,G         : afprøv watch doG");
  printf("\nr,Rxxxx.xx : Ret i ram memory [adresse.data]");
} /* help end */

/*********************************************************************
 * Printer port styring, simpel                                      *
 *********************************************************************/
centronic(out_data_byte)
char out_data_byte;

{
  XBYTE[centronic_out] = out_data_byte;   /* out_data_byte; */
  XBYTE[output_1] = 0x40;                 /* low strobe out */
  XBYTE[output_1] = 0xC0;                 /* high strobe out */
} /* centronic(out_data_byte) end */

/*********************************************************************
 * Modtaget data fra den serielle linie printes ud på Centronic port *
 * ESC returner til monitoren                                        *
 *********************************************************************/
print_out()
{
char print_char;

  do
  {
    print_char = _getkey();
    if ( print_char != ESC )
    {
      centronic(print_char);
      if (print_char == 0x0D) /* test om CR */
      {
        for ( print_char=0; print_char <= 250; print_char++);   /* delay */
        print_char = 0x0A;    /* LF, line feed */
        centronic(print_char);
      }
    }
  }
  while ( print_char != ESC );
} /* print_out end */

/*********************************************************************
 * Sætter CPU i power down mode                                      *
 * Reset bringer CPU tilbage til normal mode, WD                     *
 *********************************************************************/
Power_down()
{
PCON |= 0x02;
} /* Power down end */
```

```
/*********************************************************************
 * Sætter CPU i Idle mode                                            *
 * Interrupt bringer CPU tilbage til normal mode, timer køre vidre   *
 * Bliver i denne routine indtil det modtages samme char             *
 *********************************************************************/
Idle_mode()
{
  char key;

  printf("\n\nidle on ");
  do
  {
    PCON |= 0x01;      /* set idle flag */
    key = look_key();
  }
  while ( key != 'i' && key != 'I' );
  printf("\nidle off ");
} /* Idle mode end */

/*********************************************************************
 * Stopper reset af watch dog                                        *
 * CPU bliver resat af WD efter max 100 mS hvis MAX691 er monteret   *
 *********************************************************************/
Stop_wd()
{
reset_wd = false;      /* WD skal ikke resettes */
} /* Stop_WD end */

/*********************************************************************
 * Upload program fra PC via seriel til ram                          *
 * format intel hex                                                  *
 *********************************************************************/
up_load()
{
  bit stop_record = false;
  bit crc_fejl = false;
  char c;
  int antal_byte;
  int linier = 0, fejl_linie = 0;

  printf("\n\nUpload hex program til ram");
  do
  {
    do                 /* vent her indtil der modtages ':'  */
      c = _getkey();
    while ( c != ':' );
    linier++;
    get_byte();        /* antal data byte der skal modtages, er i byte_x */
    antal_byte = byte_x;
    crc_sum = byte_x; /* beregn crc sum, clear crc sum før start af ny record */
    get_word();        /* start adresse for denne record, er i adresse */
    crc_sum += (char)adresse;    /* beregn crc sum, low byte */
    crc_sum += adresse / 256;    /* beregn crc sum, high byte */
    get_byte();                  /* hent record type, bruges ikke */
    crc_sum += byte_x;           /* beregn crc sum */
    if ( antal_byte > 0 )
    {                            /* hvis det er data */
      while ( antal_byte > 0 )
        {
          get_byte();
          XBYTE[adresse] = byte_x;
          crc_sum += byte_x;     /*beregn crc sum */
          antal_byte--;
          adresse++;
        }
      crc_sum = 256 - (unsigned char)crc_sum; /* converter crc_sum til 2's complement
*/
      get_byte();                             /* hent crc sum */
      if ( byte_x != (unsigned char)crc_sum ) /* er modtaget crc lig med beregnet crc
*/
      {
        crc_fejl = true;
        fejl_linie = linier;
      }
    }
    else
    {                                         /* hvis det er end record, stop */
      crc_sum = 256 - (unsigned char)crc_sum; /* converter crc_sum til 2's complement
*/
```

```
      get_byte();                              /* hent check sum */
      if ( byte_x != (unsigned char)crc_sum ) /* er modtaget crc lig med beregnet crc
*/
      {
        crc_fejl = true;
        fejl_linie = linier;
      }
      stop_record = true;
    }
  }
  while ( ! stop_record );
  if ( crc_fejl )
  {
    printf("\ncrc fejl ved modtagelse af program");
    printf("\nantal modtagne hex data linier : %u",linier);
    printf("\nfejl i hex data linie          : %u",fejl_linie);

  }
  else
  {
    printf("\nprogram modtaget uden crc fejl");
    printf("\nantal modtagne hex data linier : %u",linier);
  }
} /* up_load end */

/*********************************************************************
 * Upload program fra PC via seriel til ram                          *
 * format biniær                                                     *
 * 5 sekunder uden modtaget data returneres til kommando mode        *
 *********************************************************************/
up_bin()
{
  unsigned char hex_byte;
  unsigned int adresse_pointer;

  printf("\n\nUpload biniær program til ram");
  hex_byte = _getkey();
/*  get_byte();                 /* hent, bruges ikke */
  adresse_pointer = 0;     /* reset adresse pointer */
  up_bin_timeout = 0;      /* clear timeout counter */
  do
  {
    if ( RI )    /* RI true, der er modtaget en char fra den serielle linie */
    {
      hex_byte = SBUF;
      RI = false;
      XBYTE[adresse_pointer] = hex_byte;
      adresse_pointer++;
      up_bin_timeout = 0;        /* clear timeout */
    }
  }
  while ( up_bin_timeout < (200 * 5) );      /* 5 sekundteres timeout */
  printf("\nantal modtagne hex data byte : %u",adresse_pointer);
} /* up_bin end */

/*********************************************************************
 * Læser indhold af eprom fra adresse og 8 linier a' 16 byte frem,   *
 * skriver først alle hex værdier, der næst ascii tegnet for værdien *
 * Kaldes read_eprom uden adresse parameter forsætter den fra sidste *
 * adresse                                                           *
 *********************************************************************/
read_eprom()
{
  char c, indhold_c;
  unsigned int adr;
  int indhold;
  int linier = 0;
  int byte = 0;

  get_word();
  printf("\n\nvis eprom indhold");
  printf("\n      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    ASCII");
  for ( linier = 0; linier < 8; linier++)
  {
    printf("\n%4X ",adresse," ");
    adr = adresse;
    for ( byte = 0; byte <= 15; byte++)
    {
      indhold = CBYTE[adresse];      /* hent indhold fra eprom */
      printf("%2X ",indhold);
```

```
      adresse++;
    }
    printf(" - ");
    for ( byte = 0; byte <= 15; byte++)
    {
      indhold_c = CBYTE[adr];
      if (isgraph(indhold_c))
        printf("%c",indhold_c);
      else
        printf(".");
      adr++;
    }
  }
} /* read_eprom */

/*************************************************************************
 * Ret ram med ønsket værdi                                              *
 *************************************************************************/
ret_ram()
{

  get_word();          /* adresse er i adresse */
  _getkey();           /* hent punktum */
  get_byte();          /* data er i byte_x */
  printf("\nadresse \%4X tilskrives værdien \%2X", adresse, byte_x);
  XBYTE[adresse] = byte_x;

} /* ret_ram end */

/*************************************************************************
 * Læser indhold af ram fra adresse og 8 linier a' 16 byte frem,         *
 * skriver først alle hex værdier, der næst ascii tegnet for værdien     *
 * Kaldes read_memory uden adresse parameter forsætter den fra sidste    *
 * adresse                                                               *
 *************************************************************************/
read_memory()
{
  char c, indhold_c;
  unsigned int adr;
  int indhold;
  int linier = 0;
  int byte = 0;

  get_word();
  printf("\n\nvis memory indhold");
  printf("\n      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    ASCII");
  for ( linier = 0; linier < 8; linier++)
  {
    printf("\n%4X ",adresse," ");
    adr = adresse;
    for ( byte = 0; byte <= 15; byte++)
    {
      indhold = XBYTE[adresse];
      printf("%2X ",indhold);
      adresse++;
    }
    printf(" - ");
    for ( byte = 0; byte <= 15; byte++)
    {
      indhold_c = XBYTE[adr];
      if (isgraph(indhold_c))
        printf("%c",indhold_c);
      else
        printf(".");
      adr++;
    }
  }
} /* read_memory */

/*************************************************************************
 * Vis dip switch stilling                                               *
 *************************************************************************/
vis_dip_switch()
{
  int dip_switch;

  dip_switch = XBYTE[dip_switch_adr];
  printf("\n\nDip switch : %2x \n", dip_switch);
}  /* vis_dip_switch end */
```

```
/*****************************************************************
 * Fyld memory med data, fortsætter indtil der ikke er mere ram  *
 *****************************************************************/
fill_memory()
{
  int ram_adr = 0;
  bit ram_fejl = false;

  get_byte();     /* hent byte som skal fyldes i ram memory */
  printf("\nfyld ram memory med %2X", byte_x);
  while (!ram_fejl)
  {
    XBYTE[ram_adr] = byte_x;
    if ( XBYTE[ram_adr] != byte_x)
      ram_fejl = true;
    ram_adr++;
  }
} /* fill_memory end */

/*****************************************************************
 * Tester ram med 55 og AA, fortsætter indtil første ram fejl,   *
 * skriver fejl adressen ud                                      *
 *****************************************************************/
ram_test()
{
  int ram_adr = 0;
  bit ram_fejl = false;

  printf("\n\nRam test");
  printf("\nRam start adresse : %5u", ram_adr);
  while (!ram_fejl)
  {
    XBYTE[ram_adr] = 0x55;
    if ( XBYTE[ram_adr] != 0x55)
      ram_fejl = true;
    XBYTE[ram_adr] = 0xAA;
    if ( XBYTE[ram_adr] != 0xAA)
      ram_fejl = true;
    if ( !ram_fejl )
      ram_adr++;
  }
  printf("\nRam stop adresse  : %5u", ram_adr);
}  /* ram_test end */

/*****************************************************************
 * Hoved program                                                 *
 *****************************************************************/
main()
{
  start();
  do
  {
  }
  while ( rtc_power_on > 0 ); /* vent her 1 sek, RTC skal lige vågne */
  restart_rtc();
  help();
  for (;;)
  {
    c = _getkey();
    if      ( c == 'e' || c == 'E' )
      read_eprom();
    else if ( c == 'u' || c == 'U' )
      up_load();
    else if ( c == 'c' || c == 'C' )
      clock();
    else if ( c == 'b' || c == 'B' )
      up_bin();
    else if ( c == 's' || c == 'S' )
      set_rtc();
    else if ( c == 'p' || c == 'P' )
      print_out();
    else if ( c == 'f' || c == 'F' )
      fill_memory();
    else if ( c == 'h' || c == 'H' )
      help();
    else if ( c == 'm' || c == 'M' )
      read_memory();
    else if ( c == 'o' || c == 'O' )
      Power_down();
    else if ( c == 'i' || c == 'I' )
```

```
      Idle_mode();
    else if ( c == 'g' || c == 'G' )
      Stop_wd();
    else if ( c == 't' || c == 'T' )
      ram_test();
    else if ( c == 'w' || c == 'W' )
      vis_dip_switch();
    else if ( c == 'r' || c == 'R' )
      ret_ram();
  }
}
```

```
/* ABSACC.H: direct access to 8051 memory areas */
/* Copyright KEIL ELEKTRONIK GmbH 1988 V1.0     */

#define CBYTE ((unsigned char *) 0x50000L)
#define DBYTE ((unsigned char *) 0x40000L)
#define PBYTE ((unsigned char *) 0x30000L)
#define XBYTE ((unsigned char *) 0x20000L)

#define CWORD ((unsigned int *) 0x50000L)
#define DWORD ((unsigned int *) 0x40000L)
#define PWORD ((unsigned int *) 0x30000L)
#define XWORD ((unsigned int *) 0x20000L)
```

```
/* CTYPE.H: prototypes for character functions */
/* Copyright KEIL ELEKTRONIK GmbH 1988 V1.0    */

extern bit isalpha (char);
extern bit isalnum (char);
extern bit iscntrl (char);
extern bit isdigit (char);
extern bit isgraph (char);
extern bit isprint (char);
extern bit ispunct (char);
extern bit islower (char);
extern bit isupper (char);
extern bit isspace (char);
extern bit isxdigit (char);
extern char tolower (char);
extern char toupper (char);
extern char toint   (char);

#define _tolower(c) ( (c)-'A'+'a' )
#define _toupper(c) ( (c)-'a'+'A' )
#define toascii(c)  ( (c) & 0x7F )
```

```
/*  Register Declarations for 8051 Processor */

/*  BYTE Register  */
sfr P0   = 0x80;
sfr P1   = 0x90;
sfr P2   = 0xA0;
sfr P3   = 0xB0;
sfr PSW  = 0xD0;
sfr ACC  = 0xE0;
sfr B    = 0xF0;
sfr SP   = 0x81;
sfr DPL  = 0x82;
sfr DPH  = 0x83;
sfr PCON = 0x87;
sfr TCON = 0x88;
sfr TMOD = 0x89;
sfr TL0  = 0x8A;
sfr TL1  = 0x8B;
sfr TH0  = 0x8C;
sfr TH1  = 0x8D;
sfr IE   = 0xA8;
sfr IP   = 0xB8;
sfr SCON = 0x98;
sfr SBUF = 0x99;


/*  BIT Register  */
/*  PSW   */
sbit CY   = 0xD7;
sbit AC   = 0xD6;
sbit F0   = 0xD5;
sbit RS1  = 0xD4;
sbit RS0  = 0xD3;
sbit OV   = 0xD2;
sbit P    = 0xD0;

/*  TCON  */
sbit TF1  = 0x8F;
sbit TR1  = 0x8E;
sbit TF0  = 0x8D;
sbit TR0  = 0x8C;
sbit IE1  = 0x8B;
sbit IT1  = 0x8A;
sbit IE0  = 0x89;
sbit IT0  = 0x88;

/*  IE   */
sbit EA   = 0xAF;
sbit ES   = 0xAC;
sbit ET1  = 0xAB;
sbit EX1  = 0xAA;
sbit ET0  = 0xA9;
sbit EX0  = 0xA8;

/*  IP   */
sbit PS   = 0xBC;
sbit PT1  = 0xBB;
sbit PX1  = 0xBA;
sbit PT0  = 0xB9;
sbit PX0  = 0xB8;

/*  P3  */
sbit RD   = 0xB7;
sbit WR   = 0xB6;
sbit T1   = 0xB5;
sbit T0   = 0xB4;
sbit INT1 = 0xB3;
sbit INT0 = 0xB2;
sbit TXD  = 0xB1;
sbit RXD  = 0xB0;

/*  SCON  */
sbit SM0  = 0x9F;
sbit SM1  = 0x9E;
sbit SM2  = 0x9D;
sbit REN  = 0x9C;
sbit TB8  = 0x9B;
sbit RB8  = 0x9A;
sbit TI   = 0x99;
sbit RI   = 0x98;
```

```
/* STDIO.H: prototypes for standard i/o functions */
/* Copyright KEIL ELEKTRONIK GmbH 1988 V1.1       */

#ifndef EOF
#define EOF -1
#endif

extern char _getkey ();
extern char getchar ();
extern char ungetchar (char);
extern putchar (char);
extern int printf (const char *, ...);
extern int sprintf (char *, const char *, ...);
extern char *gets (char *, int n);
extern int scanf (const char *, ...);
extern int sscanf (char *, const char *, ...);
extern int puts (const char *);
```