```
/*

Common header file

Designed by Shane Tolmie of www.microchipC.com corporation.  Freely
    distributable.
Questions and comments to webmaster@microchipC.com
Lots of Hi-Tech C FAQ and sample source code at www.microchipC.com.

For Microchip PIC16Fx.

Compiled with Hitech-C v7.85
[jo:130816] tested with XC8 and works fine.

Usage: #include in all ".c" files in project along with "pic.h"
       [jo:130816] change "pic.h" by "xc.h" with XC8 compiler

[jo:160206] incluido #define para OFF e ON
[jo:160227] incluido #define para ANALOG e DIGITAL

*/

//warning about #if statements: if any of the arguments are spelled wrong or
//unrecognised by the compiler, it will not generate a warning, but not
    include code

#ifndef ALWAYS_H
#define ALWAYS_H

/*
Turning bits on/off according to mask
use ~0 instead of 0xFF, etc, because this ensures machine independence
if int changes from 16-bit to 32-bit
Example C:
x=0b001;
bits_on(x,0b100)  //now x=0b101
*/

#define bits_on(var,mask) var |= mask
#define bits_off(var,mask) var &= ~0 ^ mask

//defines
#define INPUT 1   //port directions, ie: TRISA0=INPUT;
#define OUTPUT 0
#define TRUE 1
#define FALSE 0
#define HIGH 1
#define LOW 0
#define hi 1
#define lo 0
#define OFF 0     // [jo:160207] definiÁ„o para desligado
#define ON 1      // [jo:160207] definiÁ„o para ligado
#define ANALOG 1  // [jo:160227] definiÁ„o para analÛgico
#define DIGITAL 0 // [jo:160227] definiÁ„o para digital

#define b asm("nop")  //convenient point for breakpoint (debugging)
#define l while(1)    //loop for ever (debugging)

//see AM576.  If interrupt occurs just when gie gets set to zero, it won't
    be cleared
```

```
#define gie_on      GIE=1
#define gie_off     while(GIE==1) GIE=0

/*
Reading an 8-bit byte in a 16-bit int

With Hi-Tech C, this method is better than using pointers, as using pointers
    in
different banks needs different #defines

It is just as efficient - the optimizer picks out the correct
byte.  Of course, >>7 requires 7 shifts.

This method cannot be used to alter a hi/lo byte, this needs pointers (as
    below)

Example C:
unsigned int x;
unsigned char y;
x=0x1234;
y=hibyte(x);      //now y=0x12 - works for variables in any bank 0 to 3
y=lobyte(x);        //now y=0x34

lobyte(x)=0xaa;   //will not work :( - use pointers
*/

#define hibyte(x) (unsigned char)(x>>8)
#define lobyte(x) (unsigned char)(x & 0xFF)

/*
given variable of any type (char, uchar, int, uint, long) it modifies
    the unsigned char residing at that memory location
    for ints, byte1 is msb, byte0 is lsb (least significant byte)
    for longs   byte3 is msb, byte0 is lsb

    ie: sample C code

    unsigned int myint=0x4321;
    long mylong=0x87654321;

    //for myint     byte1(myint)=0x43; (msb) and byte0(myint)=0x21; (lsb)
    //for mylong    byte3(mylong)=0x87; (msb), byte2(mylong)=0x65;
                                byte2(mylong)=0x43; and byte0(mylong)=0x21;
                                    (lsb)

    note:   to avoid fixup overflow errors add bankX if the target variable
                resides in banks 1, 2 or 3
*/

#define byte0(x) (unsigned char)(*(((unsigned char *)&x)+0))
#define byte1(x) (unsigned char)(*(((unsigned char *)&x)+1))
#define byte2(x) (unsigned char)(*(((unsigned char *)&x)+2))
#define byte3(x) (unsigned char)(*(((unsigned char *)&x)+3))

#define lobyte_atbank0 byte0   //another way of saying it
#define hibyte_atbank0 byte1

#define byte0_atbank1(x) (unsigned char)(*(((bank1 unsigned char *)&x)+0))
#define byte1_atbank1(x) (unsigned char)(*(((bank1 unsigned char *)&x)+1))
```

```
#define byte2_atbank1(x) (unsigned char)(*(((bank1 unsigned char *)&x)+2))
#define byte3_atbank1(x) (unsigned char)(*(((bank1 unsigned char *)&x)+3))

#define lobyte_atbank1 byte0_atbank1  //another way of saying it
#define hibyte_atbank1 byte1_atbank1

#define byte0_atbank2(x) (unsigned char)(*(((bank2 unsigned char *)&x)+0))
#define byte1_atbank2(x) (unsigned char)(*(((bank2 unsigned char *)&x)+1))
#define byte2_atbank2(x) (unsigned char)(*(((bank2 unsigned char *)&x)+2))
#define byte3_atbank2(x) (unsigned char)(*(((bank2 unsigned char *)&x)+3))

#define byte0_atbank3(x) (unsigned char)(*(((bank3 unsigned char *)&x)+0))
#define byte1_atbank3(x) (unsigned char)(*(((bank3 unsigned char *)&x)+1))
#define byte2_atbank3(x) (unsigned char)(*(((bank3 unsigned char *)&x)+2))
#define byte3_atbank3(x) (unsigned char)(*(((bank3 unsigned char *)&x)+3))

/*
given variable of any type (char, uchar, int, uint, long) it modifies
    the int residing at that memory location
    ie: sample C code

    unsigned char array[4];
    unsigned int test;

    uint_atbyteaddr(&array[0])=0x4321;   //now array[0->3]={0x21,0x43,0,0};
    uint_atbyteaddr(&array[0+2])=0x8765; //now array[0-
        >3]={0x21,0x43,0x65,0x87};
    test=uint_atbyteaddr(&array[0+2]);   //now test=0x8765

    note: do NOT use &(array[0]+1) to reference the int stored at array[1]
        as it will
                reference the int after array[0] in pointer arithmetic.
                    This will
                result with the int at array[2].

                Instead use &array[0+1] to reference the int at uchar
                    array[1]

    note:   to avoid fixup overflow errors add bankX if the target variable
                resides in banks 1, 2 or 3
*/

#define uint_atbyteaddr(x)      (unsigned int)(*(((unsigned int *)x)))
#define uint_atbank1byteaddr(x) (unsigned int)(*(((bank1 unsigned int *)x)))
#define uint_atbank2byteaddr(x) (unsigned int)(*(((bank2 unsigned int *)x)))
#define uint_atbank3byteaddr(x) (unsigned int)(*(((bank3 unsigned int *)x)))

#define THE_BEER_IS_PLENTIFUL_AND_THE_PARTY_SWINGING TRUE

/*

NOTE: it is not recommended that unions are used to reference hi/lo bytes or
bits of a variable.  Use >>8 or &FF or pointers instead, as above.  It makes
passing variables to a function difficult, as the function must be defined
    to
accept variables of the same union.  Then, the function will no longer
    accept
normally defined variables.

these two structures allow access to 2 byte word, high and low bytes of
```

```
    variable
declaration:    union wordtype x;
usage:  x.word=0xABCD; x.byte.high=0xAB; x.byte.low=0xCD;
                x.part.bit15=1; (msb), x.part.bit0=1; (lsb)
declaration:    union chartype x;
usage:  x.byte=0xAB;
                x.part.bit7=1; (msb), x.part.bit0=1; (lsb)
*/

struct sixteen_bits {
    unsigned char bit0  :1;
    unsigned char bit1  :1;
    unsigned char bit2  :1;
    unsigned char bit3  :1;
    unsigned char bit4  :1;
    unsigned char bit5  :1;
    unsigned char bit6  :1;
    unsigned char bit7  :1;
    unsigned char bit8  :1;
    unsigned char bit9  :1;
    unsigned char bit10 :1;
    unsigned char bit11 :1;
    unsigned char bit12 :1;
    unsigned char bit13 :1;
    unsigned char bit14 :1;
    unsigned char bit15 :1;
};

struct eight_bits {
    unsigned char bit0 :1;
    unsigned char bit1 :1;
    unsigned char bit2 :1;
    unsigned char bit3 :1;
    unsigned char bit4 :1;
    unsigned char bit5 :1;
    unsigned char bit6 :1;
    unsigned char bit7 :1;
};

struct two_bytes {
    unsigned char low;
    unsigned char high;
};

union wordtype {
    unsigned int word;
    struct two_bytes byte;
    struct sixteen_bits part;
};

union chartype {
    unsigned char byte;
    struct eight_bits part;
};
#endif
```