Interrupts

Interrupts can be easily handled by means of reserved word `interrupt`. mikroC PRO for PIC implictly declares function `interrupt` which cannot be redeclared. Its prototype is:

```
void interrupt(void);
```

For P18 low priorty interrupts reserved word is `interrupt_low`:

```
void interrupt_low(void);
```

You are expected to write your own definition (function body) to handle interrupts in your application.

mikroC PRO for PIC saves the following SFR on stack when entering interrupt and pops them back upon return:

- PIC12 family: `W`, `STATUS`, `FSR`, `PCLATH`
- PIC16 family: `W`, `STATUS`, `FSR`, `PCLATH`
- PIC18 family: `FSR` (fast context is used to save `WREG`, `STATUS`, `BSR`)

Use the `#pragma disablecontexsaving` to instruct the compiler not to automatically perform context-switching. This means that no regiser will be saved/restored by the compiler on entrance/exit from interrupt service routine. This enables the user to manually write code for saving registers upon entrance and to restore them before exit from interrupt.

P18 priority interrupts

Note: For the P18 family both low and high interrupts are supported.

1. function with name `interrupt` will be linked as ISR (interrupt service routine) for high level interrupt
2. function with name `interrupt_low` will be linked as ISR for low level interrupt_low

If interrupt priority feature is to be used then the user should set the appropriate SFR bits to enable it. For more information refer to datasheet for specific device.

Function Calls from Interrupt

Calling functions from within the interrupt() routine is now possible. The compiler takes care about the registers being used, both in "interrupt" and in "main" thread, and performs "smart" context-switching between the two, saving only the registers that have been used in both threads.Check functions reentrancy.

Interrupt Examples

Here is a simple example of handling the interrupts from `TMR0` (if no other interrupts are allowed):

```
void interrupt() {
  counter++;
  TMR0 = 96;
  INTCON = $20;
}
```

In case of multiple interrupts enabled, you need to test which of the interrupts occurred and then proceed with the appropriate code (interrupt handling):

```
void interrupt() {
  if (INTCON.TMR0IF) {
    counter++;
    TMR0 = 96;
    INTCON.TMR0F = 0;
  }
  else if (INTCON.RBIF) {
    counter++;
    TMR0 = 96;
    INTCON.RBIF = 0;
  }
}
```