```
;*****************************************************************************
;* This file contains PID functions with interrupt.
;*****************************************************************************
;*File name:    PIDInt.asm
;*Dependencies: p18f452.inc (change to specific application requirements)
;*Processors:   PIC18
;*Assembler:    MPASMWIN 02.70.02 or higher
;*Linker:       MPLINK 2.33.00 or Higher
;*Company:      Microchip Technology, Inc.
;*
;* Software License Agreement
;*
;* The software supplied herewith by Microchip Technology Incorporated
;* (the "Company") for its PICmicro® Microcontroller is intended and
;* supplied to you, the Company's customer, for use solely and
;* exclusively on Microchip PICmicro Microcontroller products. The
;* software is owned by the Company and/or its supplier, and is
;* protected under applicable copyright laws. All rights are reserved.
;* Any use in violation of the foregoing restrictions may subject the
;* user to criminal sanctions under applicable laws, as well as to
;* civil liability for the breach of the terms and conditions of this
;* license.
;*
;* THIS SOFTWARE IS PROVIDED IN AN "AS IS" CONDITION. NO WARRANTIES,
;* WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
;* TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
;* PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
;* IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
;* CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
;*
;*
;*
;* Author               Date            Comment
;*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
;* C.Valenti              June 29, 2004    Initial Release (V1.0)
;*
;* Revisions:
;*      7/8/04  -Removed unused variables a_Err1Lim & a_Err2Lim
;*                      Modified code after the "restore_limit" label to reflect using
;*                      aErr1Lim & aErr2Lim defined constants.
;*                      -Changed constant: #define derivCount to #define derivCountVal
;*                      -pidStat1 bit comments were corrected to the correct bit #
;*                      -In the PidInterrupt routine, the " movlw derivCountVal " was added
;*                      for loading the derivCount variable.
;*
;*      10/20/04 -Added bra statment to the Derivative routine
;*                       Amended code for checking the a_Error2 limits.
;*
;*****************************************************************************

;PID Notes:
;       PROPORTIONAL = (system error * Pgain )
;       System error = error0:error1
;
;       INTEGRAL = (ACUMULATED ERROR  * Igain)
;       Accumulated error (a_error) = error0:error1 + a_Error0:a_Error2
;
;       DERIVATIVE = ((CURRENT ERROR  - PREVIOUS ERROR) * Dgain)
;       delta error(d_error) = errro0:error1 - p_error0:p_error1
;
;       Integral & Derivative control will be based off sample periods of "x" time.
;       The above sample period should be based off the PLANT response
;       to control inputs.
;               SLOW Plant response = LONGER sample periods
;               FAST Plant response = SHORTER sample periods
;
;       If the error is equal to zero then no PID calculations are completed.
;
;       The PID routine is passed the 16- bit errror data by the main application
;       code through the error0:error1 variables.
;       The sign of this error is passed through the error sign bit:
;                       pidStat1,err_sign
;       The PID outputs a 24-bit vaule in pidOut0:pidOut2 and the sign of this
;       result is the pid_sign bit in the pidStat1 register.
;-----------------------------------------------------------------------------

        list            p=18F452
        #include        <p18f452.inc>

;***** SYSTEM CONSTANTS
#define aErr1Lim                0x0F    ;accumulative error limits (4000d)
#define aErr2Lim                0xA0

#define timer1Hi                0x3E    ;Timer1 timeout defined by timer1Lo & timer1Hi
#define timer1Lo                0x0D    ;this timout is based on Fosc/4

#define derivCountVal   .10             ;determies how often the derivative term will be executed.

;#define        pid_100                 ;comment out if not using a 0 - 100% scale


        EXTERN  FXM1616U,FXD2416U,_24_BitAdd,_24_bit_sub
        EXTERN  AARGB0,AARGB1,AARGB2,AARGB3
        EXTERN  BARGB0,BARGB1,BARGB2,BARGB3

        GLOBAL  error0, error1, pidStat1

;***** VARIABLE DEFINITIONS
```

```
pid_data        UDATA
#ifdef  pid_100
percent_err     RES     1                       ;8-bit error input, 0 - 100% (0 - 100d)
percent_out     RES     1                       ;8-bit output, 0 - 100% (0 - 100d)
#endif

derivCount      RES 1                   ;This value determins how many times the Derivative term is
                                                ;calculated based on each Integral term.
pidOut0         RES     1                       ;24-bit Final Result of PID for the "Plant"
pidOut1         RES     1
pidOut2         RES     1
error0          RES     1                       ;16-bit error, passed to the PID
error1          RES     1
a_Error0        RES     1                       ;24-bit accumulated error, used for Integral term
a_Error1        RES     1
a_Error2        RES     1
p_Error0        RES     1                       ;16-bit previous error, used for Derivative term
p_Error1        RES     1
d_Error0        RES     1                       ;16-bit delta error (error - previous error)
d_Error1        RES     1

prop0           RES     1                       ;24-bit proportional value
prop1           RES     1
prop2           RES     1
integ0          RES 1                   ;24-bit Integral value
integ1          RES     1
integ2          RES     1
deriv0          RES 1                   ;24-bit Derivative value
deriv1          RES     1
deriv2          RES     1

kp                      RES     1                       ;8-bit proportional Gain
ki                      RES     1                       ;8-bit integral Gain
kd                      RES     1                       ;8-bit derivative Gain

pidStat1        RES     1                       ;PID bit-status register
pidStat2        RES     1                       ;PID bit-status register2
tempReg         RES     1                       ;temporary register

;                                   pidStat1 register
;       ____________________________________________________________________________________
;       |  bit 7   |   bit 6    |  bit 5 |    bit 4   |   bit 3    |  bit 2   |   bit 1    |  bit 0   |
;       | pid_sign | d_err_sign |   mag  | p_err_sign | a_err_sign | err_sign |  a_err_z   |  err_z   |
;       |__________|____________|________|____________|____________|__________|____________|__________|

err_z           equ     0                       ;error zero flag, Zero = set
a_err_z         equ     1                       ;a_error zero flag, Zero = set
err_sign        equ     2                       ;error sign flag, Pos = set/ Neg = clear
a_err_sign      equ     3                       ;a_error sign flag, Pos = set/ Neg = clear
p_err_sign      equ     4                       ;a_error sign flag, Pos = set/ Neg = clear
mag                     equ     5                       ;set = AARGB magnitude, clear = BARGB magnitude
d_err_sign      equ     6                       ;d_error sign flag, Pos = set/ Neg = clear
pid_sign        equ     7                       ;PID result sign flag, Pos = set/ Neg = clear

;       _________________________________ pidStat2 register_____________________________________
;       | bit 7 |  bit 6  |  bit 5   |    bit 4   |   bit 3    | bit 2 |   bit 1    |  bit 0   |
;       |       |         |          |            |            |       |            | d_err_z  |
;       |_______|_________|__________|____________|____________|_______|____________|__________|

d_err_z         equ     0                       ;d_error zero flag, Zero = set


_PIDCODE        CODE                    ;start PID code here
;*********************************************************************;
; Function: PidInit                                                   ;
;                                                                     ;
; PreCondition: Called by the application code for PID initalization  ;
;                                                                     ;
; Overview: PID variables are cleared, PID gains are given values,    ;
;                       flags are initialized.                                                                  ;
;                                                                                                                       ;
; Input:                                                                                                                ;
;                                                                     ;
; Output: none                                                        ;
;                                                                     ;
; Side Effects: W register is changed                                                   ;
;                                                                     ;
; Stack requirement: 1 levels deep                                    ;
;                                                                     ;
;*********************************************************************;
PidInitalize:
        GLOBAL  PidInitalize
        clrf    error0
        clrf    error1
        clrf    a_Error0
        clrf    a_Error1
        clrf    a_Error2
        clrf    p_Error0
        clrf    p_Error1
        clrf    d_Error0
        clrf    d_Error1

        clrf    prop0
        clrf    prop1
        clrf    prop2
        clrf    integ0
        clrf    integ1
        clrf    integ2
        clrf    deriv0
        clrf    deriv1
        clrf    deriv2

        clrf    kp
        clrf    ki
        clrf    kd

        clrf    pidOut0
        clrf    pidOut1
        clrf    pidOut2

        clrf    AARGB0
```

```
        clrf    AARGB1
        clrf    AARGB2
        clrf    BARGB0
        clrf    BARGB1
        clrf    BARGB2

        movlw   .160                            ;10 x 16, Kp, Ki & Kd are 8-bit vlaues that cannot exceed 255
        movwf   kp                                      ;Enter the PID gains scaled by a factor of 16, max = 255

        movlw   .160                            ;10 x 16
        movwf   ki

        movlw   .160                            ;10 x 16
        movwf   kd

        movlw   .10
        movwf   derivCount                      ;derivative action = TMR1H:TMR1L * derivCount
        bcf             pidStat1,err_z          ;start w/error not equal to zero
        bsf             pidStat1,a_err_z        ;start w/a_error equal to zero
        bsf             pidStat2,d_err_z        ;start w/d_error equal to zero
        bsf             pidStat1,p_err_sign     ;start w/ previous error = positive
        bsf             pidStat1,a_err_sign     ;start w/ accumulated error = positive

        bcf             PIR1,TMR1IF                     ;clear T1 flag
        bsf             PIE1,TMR1IE                     ;enable T1 interrupt

        movlw   b'00000001'                     ;configure T1 for Timer operation from Fosc/4
        movwf   T1CON
        movlw   timer1Hi                        ;load T1 registers with 5ms count
        movwf   TMR1H
        movlw   timer1Lo
        movwf   TMR1L

        return                                          ;return back to the main application code


;*********************************************************************;
; Function: PidMain                                                   ;
;                                                                     ;
; PreCondition: error0:erro1 are loaded with the latest system error  ;
;                                                                     ;
; Overview: This is the routine that the application code will call   ;
;                       to get a PID correction value. First, the error is checked      ;
;                       to determine if it is zero, if this is true, then the PID       ;
;                       code is complete.                                                                                       ;
;                                                                                                                               ;
; Input: error0:error1, sign of the error: pidStat1,err_sign          ;
;                                                                     ;
; Output: prop0:prop2                                                 ;
;                                                                     ;
; Side Effects: W register is changed                                                   ;
;                                                                     ;
; Stack requirement: 5 levels deep                                    ;
;                                                                     ;
;*********************************************************************;
PidMain:
        GLOBAL  PidMain
        bcf             PIE1,TMR1IE                     ;disable T1 interrupt
#ifdef          pid_100                         ;if using % scale then scale up PLANT error
        movlw   .40                                     ; 0 - 100% == 0 - 4000d
        mulwf   percent_err,1           ;40 * percent_err --> PRODH:PRODL
        movff   PRODH,error0
        movff   PRODL,error1            ;percentage has been scaled and available in error0:error1
#endif
        movlw   0
        cpfseq  error0                          ;Is error0 = 00 ?
        bra             call_pid_terms          ;NO, done checking

        cpfseq  error1                          ;YES, Is error1 = 00 ?
        bra             call_pid_terms          ;NO, start proportional term
        bsf             pidStat1,err_z          ;YES, set error zero flag
        bsf             PIE1,TMR1IE                     ;enable T1 interrupt
        return                                          ;return back to the main application code

call_pid_terms
        call    Proportional            ;NO, start with proportional term
        call    Integral                        ;get Integral term
        call    Derivative                      ;get Derivative term

        call    GetPidResult            ;get the final PID result that will go to the system
        bsf             PIE1,TMR1IE                     ;enable T1 interrupt
        return                                          ;return back to the main application code


;*********************************************************************;
; Function: Proportional                                              ;
;                                                                     ;
; PreCondition: error0:erro1 are loaded with the latest system error  ;
;                                                                     ;
; Overview: This routine will multiply the system's 16-bit error by the ;
;           proportional gain(Kp) --> error0:error1 * Kp                              ;
;                                                                                                                               ;
; Input: error0:error1, sign of the error: pidStat1,err_sign          ;
;                                                                     ;
; Output: prop0:prop2                                                 ;
;                                                                     ;
; Side Effects: W register is changed                                                   ;
;                                                                     ;
; Stack requirement: 2 levels deep                                    ;
;                                                                     ;
;*********************************************************************;
Proportional:
        clrf    BARGB0
        movff   kp,BARGB1
        movff   error0,AARGB0
        movff   error1,AARGB1
        call    FXM1616U                        ;proportional gain * error

        movff   AARGB1,prop0            ;AARGB2 --> prop0
        movff   AARGB2,prop1            ;AARGB3 --> prop1
        movff   AARGB3,prop2            ;AARGB4 --> prop2
```

```
        return                                  ;return to mainline code


;*********************************************************************;
; Function: Integral                                                  ;
;                                                                     ;
; PreCondition:  error0:erro1 are loaded with the latest system error ;
;                                                                     ;
; Overview: This routine will multiply the system's 16-bit accumulated;
;           error by the integral gain(Ki)--> a_Error0:a_Error1 * Ki  ;
;                                                                                                                                   ;
; Input: a_Error0:a_Error1, sign of a_Error: pidStat1,a_err_sign      ;
;                                                                     ;
; Output: integ0:integ2                                               ;
;                                                                     ;
; Side Effects: W register is changed                                                 ;
;                                                                     ;
; Stack requirement: 2 levels deep                                    ;
;                                                                     ;
;*********************************************************************;
Integral:
        btfsc   pidStat1,a_err_z        ;Is a_error = 0
        bra             integral_zero           ;Yes

        clrf    BARGB0                          ;No
        movff   ki,BARGB1                       ;move the integral gain into BARGB1
        movff   a_Error1,AARGB0
        movff   a_Error2,AARGB1
        call    FXM1616U                        ;Integral gain * accumulated error

        movff   AARGB1,integ0           ;AARGB1 --> integ0
        movff   AARGB2,integ1           ;AARGB2 --> integ1
        movff   AARGB3,integ2           ;AARGB3 --> integ2
        return                                          ;return
integral_zero
        clrf    integ0                          ;a_error = 0, clear Integral term
        clrf    integ1
        clrf    integ2
        return


;*********************************************************************;
; Function: Derivative                                                ;
;                                                                     ;
; PreCondition:  error0:erro1 are loaded with the latest system error ;
;                                                                     ;
; Overview: This routine will multiply the system's 16-bit delta              ;
;           error by the derivative gain(Kd) --> d_Error0:d_Error1 * Kd ;
;                   d_Error0:d_Error1 = error0:error1 - p_Error0:p_Error1           ;
;                                                                                                                                   ;
; Input: d_Error0:d_Error1, pidStat2,d_err_z                                                ;
;                                                                     ;
; Output: deriv0:deriv2                                               ;
;                                                                     ;
; Side Effects: W register is changed                                                 ;
;                                                                     ;
; Stack requirement: 2 levels deep                                    ;
;                                                                     ;
;*********************************************************************;
Derivative:
        btfsc   pidStat2,d_err_z        ;Is d_error = 0?
        bra             derivative_zero         ;YES

        movff   d_Error1,BARGB1         ;result ---> BARGB1
        movff   d_Error0,BARGB0         ;result ---> BARGB0
        movff   kd,AARGB1
        clrf    AARGB0
        call    FXM1616U                        ;Derivative gain * (error_l - prv_error1)

        movff   AARGB1,deriv0           ;AARGB1 --> deriv0
        movff   AARGB2,deriv1           ;AARGB2 --> deriv1
        movff   AARGB3,deriv2           ;AARGB3 --> deriv2
        return                                          ;return
derivative_zero
        clrf    deriv0                          ;d_error = 0, clear Derivative term
        clrf    deriv1
        clrf    deriv2
        return



;*********************************************************************;
; Function: GetPidResult                                              ;
;                                                                     ;
; PreCondition: Proportional, Integral & Derivative terms have been           ;
;                               calculated. The Timer1 interrupt is disabled within           ;
;                               this routine to avoid corruption of the PID result.           ;
;                                                                     ;
; Overview: This routine will add the PID terms and then scale down   ;
;                       the result by 16. This will be the final result that is       ;
;                       calcualted by the PID code.                                                                 ;
;                                                                                                                                   ;
; Input: prop0:prop2, integ0:integ2, deriv0:deriv2                                          ;
;                                                                     ;
; Output: pidOut0:pidOut2                                             ;
;                                                                     ;
; Side Effects: W register is changed                                                 ;
;                                                                     ;
; Stack requirement: 4 levels deep max.                                                       ;
;                                                                     ;
;*********************************************************************;
GetPidResult:
        movff   prop0,AARGB0                    ;load Prop term & Integral term
        movff   prop1,AARGB1
        movff   prop2,AARGB2
        movff   integ0,BARGB0
        movff   integ1,BARGB1
        movff   integ2,BARGB2

        call    SpecSign                                ;YES, call routine for add/sub sign numbers
        btfss   pidStat1,mag                    ;which is greater in magnitude ?
```

```
        bra             integ_mag                       ;BARGB is greater in magnitude
        bra             prop_mag                        ;AARGB is greater in magnitude

integ_mag                                               ;integ > prop
        bcf             pidStat1,pid_sign               ;PID result is negative
        btfsc   pidStat1,a_err_sign
        bsf             pidStat1,pid_sign               ;PID result is positive
        bra     add_derivative                  ;(Prop + Integ) + derivative

prop_mag                                                ;integ < prop
        bcf             pidStat1,pid_sign               ;PID result is negative
        btfsc   pidStat1,err_sign
        bsf             pidStat1,pid_sign               ;PID result is positive

add_derivative
        movff   deriv0,BARGB0                   ;YES, AARGB0:AARGB2 has result of Prop + Integ
        movff   deriv1,BARGB1                   ;load derivative term
        movff   deriv2,BARGB2

        movff   pidStat1,tempReg                ;pidStat1 ---> tempReg
        movlw   b'11000000'                             ;prepare for sign check of bits 7 & 6
        andwf   tempReg,f

        movf    tempReg,w                               ;check error sign & a_error sign bits
        sublw   0x00
        btfsc   STATUS,Z
        bra             add_neg_d                       ;bits 7 & 6 (00) are NEGATIVE, add them
        bra             other_combo_d                   ;bits 7 & 6 not equal to 00
add_neg_d
        call    _24_BitAdd                              ;add negative sign values
        bra             scale_down                      ;scale result

other_combo_d
        movf    tempReg,w
        sublw   0xC0
        btfsc   STATUS,Z
        bra             add_pos_d                       ;bits 7 & 6 (11) are POSITIVE, add them
        bra             find_mag_sub_d                  ;bits 7 & 6 (xx) are different signs , subtract them
add_pos_d
        call    _24_BitAdd                              ;add positive sign values
        bra             scale_down                      ;scale result
find_mag_sub_d
        call    MagAndSub                               ;subtract unlike sign numbers
        btfss   pidStat1,mag                    ;which is greater in magnitude ?
        bra             deriv_mag                       ;BARGB is greater in magnitude
        bra             scale_down                      ;derivative term < part pid term, leave pid_sign as is

deriv_mag                                               ;derivative term > part pid term
        bcf             pidStat1,pid_sign               ;PID result is negative
        btfsc   pidStat1,d_err_sign
        bsf             pidStat1,pid_sign               ;PID result is positive

scale_down
        clrf    BARGB0                                  ;(Prop + Integ + Deriv) / 16 = FINAL PID RESULT to plant
        movlw   0x10
        movwf   BARGB1
        call    FXD2416U
        movff   AARGB2,pidOut2                  ;final result ---> pidOut2
        movff   AARGB1,pidOut1                  ;final result ---> pidOut1
        movff   AARGB0,pidOut0                  ;final result ---> pidOut0

#ifdef          pid_100                                 ;Final result needs to be scaled down to  0 - 100%
        movlw   0x06                                    ;% ratio for propotional & integral & derivative
        movwf   BARGB0
        movlw   0x40
        movwf   BARGB1

        call    FXD2416U                                ;pidOut0:pidOut2 / % ratio = 0 - 100% value
        movf    AARGB2,W                                ;AARGB2 --> percent_out
        movwf   percent_out                             ;error has been scaled down and is now available in a 0 -100% range
#endif

        return                                                  ;return to mainline code


;*****************************************************************;
; Function: GetA_Error                                          ;
;                                                               ;
; PreCondition: Proportional term has been calculated                           ;
;                                                               ;
; Overview: This routine will add the current error with all of the             ;
;                       previous errors. The sign of the accumulated error will         ;
;                       also be determined.     After the accumulated error is                  ;
;                       calculated then it is checked if it = 00 or as exceeded         ;
;                       the defined limits.                                                                                     ;
;                                                                                                                               ;
; Input: a_Error0:a_Error1, error0:error1                                                       ;
;                                                               ;
; Output: a_Error0:a_Error1 (updated value)                     ;
;                                                               ;
; Side Effects: W register is changed                                                   ;
;                                                               ;
; Stack requirement: 4 levels deep max.                                                         ;
;                                                               ;
;*****************************************************************;
GetA_Error:
        movff   a_Error0,BARGB0                 ;load error & a_error
        movff   a_Error1,BARGB1
        movff   a_Error2,BARGB2
        clrf    AARGB0
        movff   error0,AARGB1
        movff   error1,AARGB2

        call    SpecSign                                ;call routine for add/sub sign numbers
        btfss   pidStat1,mag                    ;which is greater in magnitude ?
        bra             a_err_zero                              ;bargb, keep sign as is or both are same sign

        bcf             pidStat1,a_err_sign             ;aargb, make sign same as error, a_error is negative
        btfsc   pidStat1,err_sign
        bsf             pidStat1,a_err_sign             ;a_error is positive
```

```
a_err_zero
        bcf             pidStat1,a_err_z                ;clear a_error zero flag
        movlw   0
        cpfseq  AARGB0                                  ;is byte 0 = 00
        bra             chk_a_err_limit                 ;NO, done checking

        cpfseq  AARGB1                                  ;is byte 1 = 00
        bra             chk_a_err_limit                 ;NO, done checking

        cpfseq  AARGB2                                  ;is byte 2 = 00
        bra             chk_a_err_limit                 ;NO, done checking
        bsf             pidStat1,a_err_z                ;YES, set zero flag

        movff   AARGB0,a_Error0                 ;store the a_error
        movff   AARGB1,a_Error1
        movff   AARGB2,a_Error2
        return                                                  ;a_error = 00, return

chk_a_err_limit
        movff   AARGB0,a_Error0                 ;store the a_error
        movff   AARGB1,a_Error1
        movff   AARGB2,a_Error2

        movlw   0                                               ;a_error reached limits?
        cpfseq  a_Error0                                ;Is a_Error0 > 0 ??, if yes limit has been exceeded
        bra             restore_limit                   ;YES, restore limit value

        cpfseq  a_Error1                                ;Is a_Error1 = 0 ??, if yes, limit not exceeded
        bra             chk_a_Error1                    ;NO
        return                                                  ;YES
chk_a_Error1
        movlw   aErr1Lim
        cpfsgt  a_Error1                                ;Is a_Error1 > aErr1Lim??
        bra             equal_value                             ;NO, check for a_Error1 = aErr1Lim ?
        bra             restore_limit                   ;YES, restore limit value
equal_value
        cpfseq  a_Error1                                ;a_Error1 = aErr1Lim?
        return                                                  ;no, done checking a_error
chk_a_Error2
        movlw   aErr2Lim                                ;Yes, a_Error1 = aErr1Lim
        cpfsgt  a_Error2                                ;Is a_Error2 > aErr2Lim ??
        return                                                  ;NO, return to mainline code

restore_limit
        clrf    a_Error0                                ;YES, a_error limit has been exceeded
        movlw   aErr1Lim
        movwf   a_Error1
        movlw   aErr2Lim
        movwf   a_Error2
        return                                                  ;return to mainline code


;*********************************************************************;
; Function: GetDeltaError                                           ;
;                                                                   ;
; PreCondition: The derivative routine has been called to calculate the ;
;                               derivative term.                                                                                ;
;                                                                   ;
; Overview: This routine subtracts the previous error from the current  ;
;                       error.                                                                                                  ;
;                                                                                                                               ;
; Input: P_Error0:p_Error1, error0:error1                                                       ;
;                                                                   ;
; Output: d_Error0:d_Error1, d_Error sign                                               ;
;                                                                   ;
; Side Effects: W register is changed                                                           ;
;                                                                   ;
; Stack requirement: 3 levels deep max.                                                                 ;
;                                                                   ;
;*********************************************************************;
GetDeltaError:
        clrf    AARGB0                                  ;load error and p_error
        movff   error0,AARGB1
        movff   error1,AARGB2
        clrf    BARGB0
        movff   p_Error0,BARGB1
        movff   p_Error1,BARGB2

        movf    pidStat1,w                              ;pidStat1 ---> tempReg
        movwf   tempReg                                 ;prepare for sign check of bits 4 & 2
        movlw   b'00010100'
        andwf   tempReg,f

        movf    tempReg,w                               ;check error sign & a_error sign bits
        sublw   0x00
        btfsc   STATUS,Z
        bra             p_err_neg                               ;bits 4 & 2 (00) are NEGATIVE,
        bra             other_combo2                    ;bits 4 & 2 not equal to 00
p_err_neg
        call    MagAndSub
        bcf             pidStat1,d_err_sign             ;d_error is negative
        btfsc   pidStat1,p_err_sign             ;make d_error sign same as p_error sign
        bsf             pidStat1,d_err_sign             ;d_error is positive
        bra             d_error_zero_chk                ;check if d_error = 0

other_combo2
        movf    tempReg,w
        sublw   0x14
        btfsc   STATUS,Z
        bra             p_err_pos                               ;bits 4 & 2 (11) are POSITIVE
        bra     p_err_add                               ;bits 4 & 2 (xx) are different signs

p_err_pos
        call    MagAndSub
        bcf             pidStat1,d_err_sign             ;d_error is negative
        btfsc   pidStat1,p_err_sign             ;make d_error sign same as p_error sign
        bsf             pidStat1,d_err_sign             ;d_error is positive
        bra             d_error_zero_chk                ;check if d_error = 0
p_err_add
        call    _24_BitAdd                              ;errors are different sign
        bcf             pidStat1,d_err_sign             ;d_error is negative
```

```
        btfsc   pidStat1,err_sign               ;make d_error sign same as error sign
        bsf             pidStat1,d_err_sign             ;d_error is positive

d_error_zero_chk
        movff   AARGB1,d_Error0
        movff   AARGB2,d_Error1

        movff   error0,p_Error0                 ;load current error into previous for next deriavtive term
        movff   error1,p_Error1                 ;load current error into previous for next deriavtive term
        bcf             pidStat1,p_err_sign             ;make p_error negative
        btfsc   pidStat1,err_sign               ;make p_error the same sign as error
        bsf             pidStat1,p_err_sign             ;make p_error positive

        bcf             pidStat2,d_err_z                ;clear delta error zero bit
        movlw   0
        cpfseq  d_Error0                                ;is d_error0 = 00
        return                                                  ;NO, done checking

        cpfseq  d_Error1                                ;YES, is d_error1 = 00
        return                                                  ;NO, done checking
        bsf             pidStat2,d_err_z                ;set delta error zero bit
        return                                                  ;YES, return to ISR

;*****************************************************************;
; Function: SpecSign                                             ;
;                                                                 ;
; PreCondition: The sign bits in pidStat1 have been set or cleared              ;
;                               depending on the variables they represent.                              ;
;                                                                 ;
; Overview: This routine takes the numbers loaded into the math               ;
;                               variables (AARGB, BARGB) and determines whether they    ;
;                               need to be added or subtracted based on their sign              ;
;                               which is located in the pidStat1 register.                              ;
;                                                                                                                                                               ;
; Input: pidStat1                                                                                                                               ;
;                                                                 ;
; Output: add/sub results in math variables (AARGB, BARGB)              ;
;                                                                 ;
; Side Effects: W register is changed                                                             ;
;                                                                 ;
; Stack requirement: 2 levels deep max.                                                   ;
;                                                                 ;
;*****************************************************************;
SpecSign
        movff   pidStat1,tempReg                ;pidStat1 ---> tempReg
        movlw   b'00001100'                             ;prepare for sign check of bits 3 & 2
        andwf   tempReg,f

        movf    tempReg,w                               ;check error sign & a_error sign bits
        sublw   0x00
        btfsc   STATUS,Z
        bra             add_neg                                 ;bits 3 & 2 are NEGATIVE (00), add them
        bra             other_combo                             ;bits 3 & 2 not equal to 00
add_neg
        call    _24_BitAdd                              ;add negative sign values
        return

other_combo
        movf    tempReg,w
        sublw   0x0C
        btfsc   STATUS,Z
        bra             add_pos                                 ;bits 3 & 2 are POSITIVE (11), add them
        bra             find_mag_sub                    ;bits 3 & 2 are different signs (xx), subtract them
add_pos
        call    _24_BitAdd                              ;add positive sign values
        return
find_mag_sub
        call    MagAndSub                               ;subtract unlike sign numbers
        return

;*****************************************************************;
; Function: MagAndSub                                            ;
;                                                                 ;
; PreCondition: This routine has been called by SpecSign because the    ;
;                               numbers being worked on are different in sign.                  ;
;                                                                 ;
; Overview: This routine will detemine which math variable                              ;
;                       (AARGB or BARGB) is greater in number manitude and then         ;
;                       subtract them, the sign of the result will be determined by     ;
;                       the values in the math variables and their signs.                       ;
;                                                                                                                                                               ;
; Input: pidStat1                                                                                                                               ;
;                                                                 ;
; Output: add/sub results in math variables (AARGB, BARGB)              ;
;                                                                 ;
; Side Effects: W register is changed                                                             ;
;                                                                 ;
; Stack requirement: 2 levels deep max.                                                   ;
;                                                                 ;
;*****************************************************************;
MagAndSub:
        movf    BARGB0,w
        subwf   AARGB0,w                                ;AARGB0 - BARGB0 --> W
        btfsc   STATUS,Z                                ;= zero ?
        bra             check_1                                 ;YES
        btfsc   STATUS,C                                ;borrow ?
        bra             aargb_big                               ;AARGB0 > BARGB0, no borrow
        bra             bargb_big                               ;BARGB0 > AARGB0, borrow
check_1
        movf    BARGB1,w
        subwf   AARGB1,w                                ;AARGB1 - BARGB1 --> W
        btfsc   STATUS,Z                                ;= zero ?
        bra             check_2                                 ;YES
        btfsc   STATUS,C                                ;borrow ?
        bra             aargb_big                               ;AARGB1 > BARGB1, no borrow
        bra             bargb_big                               ;BARGB1 > AARGB1, borrow

check_2
        movf    BARGB2,w                                ;AARGB2 - BARGB2 --> W
        subwf   AARGB2,w
        btfsc   STATUS,C                                ;borrow ?
```

```
        bra             aargb_big                       ;AARGB2 > BARGB2, no borrow
        bra             bargb_big                       ;BARGB2 > AARGB2, borrow

aargb_big
        call    _24_bit_sub
        bsf             pidStat1,mag                    ;AARGB is greater in magnitude
        return

bargb_big
        movff   BARGB0,tempReg                  ;swap AARGB0 with BARGB0
        movff   AARGB0,BARGB0
        movff   tempReg,AARGB0

        movff   BARGB1,tempReg                  ;swap AARGB1 with BARGB1
        movff   AARGB1,BARGB1
        movff   tempReg,AARGB1

        movff   BARGB2,tempReg                  ;swap AARGB2 with BARGB2
        movff   AARGB2,BARGB2
        movff   tempReg,AARGB2

        call    _24_bit_sub                     ;BARGB > AARGB
        bcf             pidStat1,mag                    ;BARGB is greater in magnitude
        return

;*********************************************************************;
; Function: PidInterrupt                                                                        ;
;                                                                                               ;
; PreCondition: This Routine will be called by the application's main   ;
;                               code.                                                                                                   ;
;                                                                                               ;
; Overview: When Timer 1 overlfows, an updated value for the Integral   ;
;                       term will be calculated. An updated value for the derivative;
;                       term will be calculated if derivCount = 0. This routine         ;
;                       will check for error = 0, if this is true,then the routine      ;
;                       will return back to the main line code.                                 ;
;                                                                                                                                                       ;
; Input: pidStat1, a_error                                                                                                      ;
;                                                                                               ;
; Output: Integral & Derivative terms, Timer1 registers reloaded        ;
;                                                                                               ;
; Side Effects: W register is changed                                                   ;
;                                                                                               ;
; Stack requirement: 4 levels deep max.                                                 ;
;                                                                                               ;
;*********************************************************************;
PidInterrupt:
        GLOBAL  PidInterrupt

        btfsc   pidStat1,err_z                  ;Is error = 00 ?
        return                                                  ;YES, done.
        call    GetA_Error                              ;get a_error, is a_error = 00? reached limits?

derivative_ready?
        decfsz  derivCount,f                    ;is it time to calculate d_error ?
        bra             skip_deriv                              ;NO, finish ISR
        call    GetDeltaError                   ;error - p_error

        movlw   derivCountVal                   ;prepare for next delta error
        movwf   derivCount                              ;delta error = TMR1H:TMR1L * derivCount

skip_deriv
        movlw   timer1Hi                                ;reload T1 registers with constant time count (user defined)
        movwf   TMR1H
        movlw   timer1Lo
        movwf   TMR1L
        return                                                  ;return back to the application's ISR

        END                                                     ;directive 'end of program'
```