```
*****************************************************************************
*****************************************************************************
**   M E G A S Q U I R T - 2 0 0 1 - V3.000
**
**   (C) 2002 - B. A. Bowling And A. C. Grippo
**
**   This header must appear on all derivatives of this code.
**
*****************************************************************************
*****************************************************************************

;-----------------------
; Version 3.000
;-----------------------
; Version 2.986 upgraded to Version 3.000
;
;-----------------------
; Version 2.986
;-----------------------
; Modified boot_r12 file to raise the LVI trip point.
; Fill unused RAM locations with $32, an illegal opcode to force a reset. Done
;  to prevent flash erasure if code ever gets into a runaway state - for protection
only.
; Put in interlocks to prevent VE/Constant section flash erasure unless
specifically
;  invoked by a flash burn command.
; Put in Odd-fire RPM averaging (Willitte) for odd-fire engines.
; Added new calculation structure to code - same calculation as HI-RES code, but
results
;  are in 0.1 millisecond units. Calculation overflow fixed.
; Flyback damping code installed - jumper port X0 for INJ1 and jumper port X1 for
INJ2.
; Fixed display of WARMCOR variable (no flickering).
; Inhibit PWM mode while cranking.
; Made "ADC" into "ADD in ADC interrupt section (Tom D) to properly perform the sum
;  without a carry.
; Perform 12 ms time compare for re-enabling of tach IRQ interrupt (fix for random
tach problem) -
;  original trial time of 20 ms too long (Magnus Bjelk fix - also for high/low byte
check).
; Moved after-start enrichment to increment after "N" tach pulses have occurred,
;  with N being the number of cylinders. This will lengthen the afterstart
enrichment
;  time by an amount scaled by the number of cylinders (Tom).
; Fix display of barometer correction (no screen flicker if 100% barometer is
selected).
; Fixed IRQ counter to work properly for NCYL above 8 cylinders.
;
;-----------------------
; Version 2.0
;-----------------------
; Acceleration Enrichment Cold Multiplier added - now cold accel enrichment in form
of a + b * x
; Barometer Correction Selection Mode put in Config13
; Bootloader Bug (lda vs ldx) fixed for this version
; RPM Calc made more efficient and bug fix for RPM exclusion values (Konstantin)
; Changed Flood clear value to 0.3 ms
; Added the "Guy ROM Offset" to fix the bootloader erase problem
; Added Embedded Code revision number SCI query command ("Q")
```

```
; Fixed SCI "C" transfer mode
; Added Alpha-N mode
; Added O2 target voltage constant and DIY-WB O2 sensor support, and bit for Odd-
Fire Mode
;  (no odd-code mode in embedded code yet).
; Fixed Battery voltage correction roll-over problem.
; Fixed Lininterp problem with large negative numerator spans
; Fixed "sticky" decel bit
; Added both MPX4115 and MPX4250 KPA and Barofac tables in flash - selected by
config register
; Prime Pulses and Adjustable RPM for entry into closed-loop added 4/2/02
;
;-----------------------
; Version 1.5
;-----------------------
; O2 closed-loop mode added 11/1/01
;
;-----------------------
; Version 1.01
;-----------------------
; Inverted Injector driver MC33151 implemented:
; To change output from inverted to non-inverted, change
;  all of the bset and bclr calls for inject1 and inject2 to their inverse.

.header 'MegaSquirt'
;.base 10t
.pagewidth 130
.pagelength 90
;.set simulate

.nolist
      include "gp32.equ"
.list
      org   ram_start
      include "megasquirt.h"


*************************************************************************
**
** Main Routine Here - Initialization and main loop
**
** Note: Org down 128 bytes below the "rom_start" point
**       because of erase bug in bootloader routine
**
** Note: Items commented out after the Start entry point are
**       taken care of in the Boot_R12.asm code
**
*************************************************************************
      org   {rom_start + 128}
Start:
;       *** Note - uncomment this code if you do not use the Bootloader to
initilize ***
;        clra
;     sta   copctl
;     mov   #%00000001,config2
;     mov   #%00001001,config1
;     mov   #%00000001,config1
;     ldhx  #ram_last+1       ; Set the stack Pointer
;     txs                     ;  to the bottom of RAM
```

```
;PllSet:
;     bclr  BCS,pctl          ; Select external Clock Reference
;     bclr  PLLON,pctl        ; Turn Of PLL
;     mov   #$02,pctl         ; Set P and E Bits
;     mov   #$C0,pmrs         ; Set L
;     mov   #$03,pmsh         ; Set N (MSB)
;     mov   #$84,pmsl         ; Set N (LSB)
;     bset  AUTO,pbwc
;     bset  PLLON,pctl        ; Turn back on PLL
;PLLwait:
;        brclr   LOCK,pbwc,PLLwait
;        bset    BCS,pctl



;
;   Set all RAM to known value - for code runaway protection.
;   If there is ever a code runaway, and processor tries
;    executing this as an opcode ($32) then a reset will occur.
;
      ldhx   #ram_start ;Point to start of RAM
ClearRAM:
      lda   #$32        ; This is an illegal op-code - cause reset if executed
      sta    ,x         ;Set RAM location
      aix    #1         ;advance pointer
      cphx   #ram_last+1      ;done ?
      bne    ClearRAM         ;loop back if not




; Set up the port data-direction registers
        lda     #%00000000
        sta     ddrb                    ; Set as inputs (ADC will select which
channel later)
        lda     #%00110000              ; Turn off injectors (inverted output)
        sta     portd
        lda     #%11110000
        sta     ddrd                  ; Outputs for injector
        clr     porta
        lda     #%11111111
        sta     ddra                    ; Outputs for Fp and Idle
        lda     #$00
        sta     portc
        lda     #%00011111              ; ** Was 11111111
        sta     ddrc                    ; Outputs for LED
        lda     #%00000001              ; Serial Comm Port
        sta     ddre

; Set up the Real-time clock Timer (TIM2)
        MOV     #Timerstop,t2sc         ; Stop Timer so it can be set up
        mov     #$00,T2MODH
        mov     #$B8,T2MODL             ; set timer modulus register to 184 decimal
        mov     #T2SC0_No_PWM,T2SC0     ; make this normal port output (PWM MODE is
#$5E)
        mov     #$00,T2CH0H
        mov     #$00,T2CH0L
```

```
        MOV     #Timergo,T2SC           ;  set timer prescaler to divide by 4

; Set up the PWM for the Injector (for current limit mode)
        MOV     #Timerstop,t1sc         ; Stop Timer so it can be set up
        mov     #$00,T1MODH
        mov     #$64,T1MODL             ; set timer modulus register to 100 decimal
        mov     #T1SCX_NO_PWM,T1SC0     ; make this normal port output (PWM MODE is
#$5E)
        mov     #T1SCX_NO_PWM,T1SC1     ; make this normal port output (PWM MODE is
#$5E)
        mov     #$00,T1CH0H
        mov     INJPWM,T1CH0L
        mov     #$00,T1CH1H
        mov     INJPWM,T1CH1L


; Set up SCI port
      lda   #$12              ; This is 9600 baud w/ the osc frequency selected
      sta   scbr
      bset  ensci,scc1        ; Enable SCI
      bset  RE,SCC2                ; Enable receiver
        bset    SCRIE,SCC2             ; Enable Receive interrupt
      lda   SCS1              ; Clear SCI transmitter Empty Bit
        clr     txcnt
        clr     txgoal

; Set up Interrupts
        mov     #%00000100,INTSCR   ;Enable IRQ



; Set up RAM Variable
        clr     mms
        clr     ms
        clr     tenth
        clr     secl
        clr     sech
        lda     #$FF
        clr     squirt
        lda     #$32
        sta     pw
        clr     engine
        clr     rpmph
        clr     rpmch
        clr     rpm
      clr   old_rpm1
      clr   flocker
        lda     #$00
        sta     pwcalc
        sta     pw
        clr     pwrun1
        clr     pwrun2
        lda     #$FF
        sta     last_tps
        clr     egocount

        lda     #$BB
        sta     baro
        sta     map
```

```
        sta     mat
        sta     clt
        sta     tps
        sta     batt

        lda     #$64
        sta     aircor
        sta     vecurr
        sta     barocor
        sta     warmcor
        sta     egocorr
      sta   tpsfuelcut
        clr     gammae
        lda     #$46
        sta     map
        lda     #$65
        sta     baro
      clr   tpsaccel

        clr     igncount

; Fire up the ADC, and perform one conversion to get the baro value
      lda   #%01110000  ; Set up divide 8 and internal bus clock source
      sta   adclk
      lda   #%00000000  ; Select one conversion, no interrupt, AD0
      sta   adscr
      brclr   coco,adscr,*    ; wait until conversion is finished

      lda   adr
      sta   baro        ; Store value in Barometer


      clr   adsel       ; Clear the channel selector

TURN_ON_INTS:
        cli       ; Turn on all interrupts now

; Load the constants (VE Table, etc) from Flash to RAM - the program uses the RAM
values

        clrh
        clrx
XXVV:
        lda     VETABLE,x       ; This is the FLASH table for all inputs
        sta     VE,x            ; This is the RAM-based table - this is what is
used by pgm.
        incx
        cbeqx  #$80,XXXV
        bra    XXVV
XXXV:



; Load the Flash Programming Routine into RAM - it will sit there until invoked

        clrh
        clrx
NEXTRAM:
        lda    ERASE_N_BURN,x   ; Programmer routine to be copied
```

```
        sta     BURNER,x         ; RAM Location that keep the burner code
        incx
        cbeqx   #$FF,DONE_LOAD_BURN
        bra     NEXTRAM

DONE_LOAD_BURN:

*******************************************************************************
**
** Prime Pulse - Shoot out one priming pulse of length PRIMEP
**
*******************************************************************************
        lda     PRIMEP
        beq     LOOPER           ; Branch around priming pulse if zero

        sta     pw
        sta     pw2
        bset    fuelp,porta
        clr     pwrun1
        clr     pwrun2
        bset    sched1,squirt
        bset    inj1,squirt
        bset    sched2,squirt
        bset    inj2,squirt
        bset    running,engine

*******************************************************************************
*******************    M A I N  E V E N T  L O O P    ******************
*******************************************************************************

LOOPER:

*******************************************************************************
**
**  Correction Factor Lookup Table Access
**
**   Perform table lookup for barometer and air density correction factors,
**    and performs coolant temperature conversion from counts to degrees F
**
**   All tables are pre-computed for all 256 different values
**    and stored in FLASH
**
**   Note: Coolant temperature is in degrees F plus 40 - this allows
**    unsigned numbers for full temperature range
**
**
*******************************************************************************

        clrh

      lda   config11
      and   #$01
      bne   TurboMap

NaMap:
        lda     baro
        tax
        lda     BAROFAC4115,x
        sta     tmp11         ; Barometer Correction Gamma
```

```
        lda     map
        tax
        lda     KPAFACTOR4115,x
        sta     kpa             ; Manifold Air Pressure in Kilopascals

        lda     config13                ; Check if we use baro correction mode
(bit=1) or not (bit=0)
        bit     #$08                    ; Use BIT instead of brset because outside
of zero-page
      bne     NaBaroUse               ; Branch if the bit is set - we use the MAP-
obtained barometer

        lda     #$64
        sta     barocor                 ; Store 100% barometer correction
(correction not used)
      bra   CltCalc

NaBaroUse:
      lda   tmp11             ; Use the saved MAP-derived barometer value
      sta   barocor
      bra   CltCalc

TurboMap:
        lda     baro
        tax
        lda     BAROFAC4250,x
        sta     tmp11           ; Barometer Correction Gamma

        lda     map
        tax
        lda     KPAFACTOR4250,x
        sta     kpa             ; Manifold Air Pressure in Kilopascals

        lda     config13                ; Check if we use baro correction mode
(bit=1) or not (bit=0)
        bit     #$08                    ; Use BIT instead of brset because outside
of zero-page
      bne     TurboBaroUse            ; Branch if the bit is set - we use the MAP-
obtained barometer

        lda     #$64
        sta     barocor                 ; Store 100% barometer correction
(correction not used)
      bra   CltCalc

TurboBaroUse:
      lda   tmp11             ; Use the saved MAP-derived barometer value
      sta   barocor


CltCalc:
        lda     clt
        tax
        lda     THERMFACTOR,x
        sta     coolant         ; Coolant temperature in degrees F + 40

        lda     mat
        tax
```

```
        lda     AIRDENFACTOR,x
        sta     aircor          ; Air Density Correction Factor

*****************************************************************************
**
** Fast Idle Comparison - fast idle set is coolant below FAST IDLE value
**
*****************************************************************************
        lda     fastidle
        cmp     coolant
        bhi     SLOW_IDLE_SET
FAST_IDLE_SET:
        bclr    iasc,porta
        bra     RPM_COMP
SLOW_IDLE_SET:
        bset    iasc,porta

*****************************************************************************
**
** Computation of RPM
**
**   rpmk:rpmk+1
**   ----------- = rpm
**   rpmph:rpmpl
**
**  rpmk:rpmK+1 = RPM constant = (6,000 * (stroke/2))/ncyl
**  rpmph:rpmpl = period count between IRQ pulsed lines, in 0.1 millisecond
resolution
**
*****************************************************************************
RPM_COMP:

        brclr   running,engine,LOOPER
      ldhx    rpmph
        beq     LOOPER

        pshh
        pula
        tsta
        beq        FAST_RPM_CALC

SLOW_RPM_CALC:
        clr       intacc1
        clr       intacc1+1

        sthx      intacc2

        ldhx      rpmk
        sthx      intacc1+2

        jsr       udvd32              ; 32 x 32 divide
        lda       intacc1+3           ; get 8-bit RPM result
        bra       RPM_CALC_DONE

FAST_RPM_CALC:
        lda       rpmk
        psha
        pulh
        lda       rpmk+1
```

```
        div                             ; A = (H:A) / X

RPM_CALC_DONE:
      sta     tmp1            ; Will move this later on...

; If odd-fire is set (bit zero of Config13), then average RPM values
        lda       config13
        and       #$01
        beq       NO_ODD_FIRE

YES_ODD_FIRE:
      lda     tmp1
      add     old_rpm1    ; this variable is updated in the IRQ section to latch
last RPM
      rora
      sta     rpm
      bra     CHK_FOR_WENRCH

NO_ODD_FIRE:
      lda     tmp1
        sta       rpm

CHK_FOR_WENRCH:
        cmp       #$03                ; Check if we are cranking
        bhi       WARM_UP_ENRICH


*****************************************************************************
**
** Cranking Mode
**
** Pulsewidth is directly set by the coolant temperature value of
**  CWU (at -40 degrees) and CWH (at 165 degrees) - value is interpolated
**
*****************************************************************************

CRANKING_SET:
        bset    crank,engine
        bclr    startw,engine
        bclr    warmup,engine

        lda     #155T   ; 3 Volts comparison value for TPS - flood clear trigger
        cmp     tps
        bhi     INTERP_CRANK_PW

FLOOD_CLEAR:
        lda     #$03    ; 0.3 ms pulsewidth - just opening...
        sta     pwcalc
        jmp     LOOPER

INTERP_CRANK_PW:
        lda     #$00
        sta     tmp1
        lda     #205T   ; 165 + 40 degrees (because of offset in lookup table)
        sta     tmp2
        mov     cwu,tmp3
        mov     cwh,tmp4
        mov     coolant,tmp5
        jsr     lininterp
```

```
        mov     tmp6,pwcalc
        jmp     LOOPER

*****************************************************************************
**
** Warm-up and After-start Enrichment Section
**
** The Warm-up enrichment is a linear interpolated value from WWU (10 points)
**  which are placed at different temperatures
**
** Method:
**
** 1) Perform ordered table search of WWU (using coolant variable) to determine
**  which bin.
** 2) Perform linear interpolation to get interpolated warmup enrichment
**
** Also, the after-start enrichment value is calculated and applied here - it
**  is an added percent value on top of the warmup enrichment, and it is applied
**  for the number of ignition cycles specified in AWC. This enrichment starts
**  at a value of AWEV at first, then it linearly interpolates down to zero
**  after AWC cycles.
**
** 3) If (startw, engine is set) then:
** 4)  compare if (awc < asecount) then:
** 5)   x1=0, x2=AWC, y1=AWEV, y2=0, x=asecount, y=ASEenrichment
** 6)  else clear startw bit in engine
**
*****************************************************************************
WARM_UP_ENRICH:
        brclr   crank,engine,WUE1
        bclr    crank,engine
        bset    startw,engine
        bset    warmup,engine
        clr     asecount
WUE1:
        ldhx    #WWURANGE
        sthx    tmp1
        lda     #$09
        sta     tmp3
        lda     coolant
        sta     tmp4
        jsr     ORD_TABLE_FIND

WUE4:
        clrh
        lda     tmp5
        tax
        lda     WWU,x
        sta     tmp4
        decx
        lda     WWU,x
        sta     tmp3
        mov     coolant,tmp5
        jsr     lininterp

        mov     tmp6,tmp12
;        mov     tmp6,warmcor
;        lda     warmcor
        lda     tmp12
```

```
        cmp     #$64
        bne     ASE1

; Outside of warmup range - clear warmup enrichment mode
        lda     #$64
        sta     warmcor
        bclr    startw,engine
        bclr    warmup,engine
        bclr    wled,portc
        bra     ASE_FINAL

ASE1:
        bset    wled,portc
        bset    warmup,engine
        brclr   startw,engine,ASE_FINAL
        lda     asecount
        cmp     awc
        bhi     ASE_END
ASE2:
        clr     tmp1
        mov     AWC,tmp2
        mov     AWEV,tmp3
        clr     tmp4
        mov     asecount,tmp5
        jsr     lininterp
        lda     tmp6
        bcs     ASE_RAIL
        add     tmp12
        sta     warmcor
        bra     TAE

ASE_RAIL:
        lda     #$FE
        sta     warmcor         ; Rail warmcor value
        bra     TAE

ASE_END:
        bclr    startw,engine

ASE_FINAL:
        mov     tmp12,warmcor

*************************************************************************
**
**  Throttle Position Acceleration Enrichment
**
**   Method is the following:
**
**
**   ACCELERATION ENRICHMENT:
**   If (tps < last_tps) goto DEACCELERATION_ENRICHMENT
**   If (tps - last_tps) > tpsthresh and TPSAEN = 0 then (acceleration enrichemnt):
**   {
**    1) Set accelration mode
**    2) Continuously determine rate-of-change of throttle, and perform
**        interpolation of TPSAQ values to determine acceleration
**        enrichment amount to apply.
**   }
**   If (TPSACLK > TPSACLKCMP) and TPSAEN is set then:
```

```
**   {
**    1) Clear TPSAEN bit in engine
**    2) Set TPSACCEL to 100%
**    3) Go to EGO Delta Step Check Section
**   }
**
**
**   DEACCELERATION ENRICHMENT:
**   If (last_tps - tps) > tpsthresh then (deacceleration fuel cut)
**   {
**    If (TPSAEN = 1) then:
**    {
**      1) TPSACCEL = 100 percent (no acceleration)
**      2) Clear TPSAEN bit in ENGINE
**      3) Go to EGO Delta Step
**    }
**    If (RPM > 15 (corresponding to 1500 RPM)) then (fuel cut mode):
**    {
**      1) Set TPSACCEL value to TPSDQ
**      2) Set TPSDEN bit in ENGINE
**      3) Go to EGO Delta Step Check Section
**    }
**   }
**   else
**   {
**    If (TPSDEN = 1) then
**    {
**     1) Clear TPSDEN bit in ENGINE
**     2) TPSACCEL = 100%
**     3) Go to EGO Delta Step Check Section
**    }
**   }
**
*************************************************************************
TAE:
        sei
        mov     tps,tmp1
        mov     last_tps,tmp2
        cli
        lda     tmp1
        cmp     tmp2
        blo     TDE2
AE_CHK:
        lda     tmp1
        sub     tmp2
        cmp     tpsthresh
        blo     B_LO_CONT
        brset   TPSAEN,ENGINE,AE_COMP_SHOOT_AMT

; Add in accel enrichement
        lda     TPSAQ           ; start out using first element - will determine
actual next time around
        sta     TPSACCEL        ; Acceleration percent amount - used in later
calculations
        clr     TPSACLK
        lda     TPSASYNC
        sta     tpsaclkcmp      ; Shoot time comparison value
        bset    TPSAEN,ENGINE
        bclr    TPSDEN,ENGINE
```

```
        bset    aled,portc
        bclr    wled,portc
        jmp     MAE

;TDE2:   bra     TDE             ; Only to extend branch for assembler


; First, calculate Cold temperature add-on enrichment value from coolant value
TPSACOLD,
; from -40 degrees to 165 degrees.
;
; Then determine cold temperature multiplier value ACCELMULT (in percent), from -40
degrees to 165 degrees.
;
; Next, Calculate Shoot amount (quantity) for acceleration enrichment from table.
;  Find bins (between) for corresponding TPSDOT, and linear interpolate
;  to find enrichment amount (from TPSAQ). This is continuously
;  checked every time thru main loop while in acceleration mode,
;  and the highest value is latched and used.
;
; The final acceleration applied is AE = Alookup(TPSDOT) * (ACCELMULT/100) +
TPSACOLD


AE_COMP_SHOOT_AMT:
; First, the "added" amount based on cold temperatures
        lda     #$00    ; 0 -> - 40 degrees
        sta     tmp1
        lda     #205T   ; 165 + 40 degrees (because of offset in lookup table)
        sta     tmp2
        mov     TPSACOLD,tmp3   ; This is the amount at coldest
        mov     #$00,tmp4       ; no enrichemnt addon at warm temperature
        mov     coolant,tmp5
        jsr     lininterp
        mov     tmp6,tmp13      ; result - save here temporarily

; Second, find the multiplier (ACCELMULT) amount based on cold temperatures
        lda     #$00    ; 0 -> - 40 degrees
        sta     tmp1
        lda     #205T   ; 165 + 40 degrees (because of offset in lookup table)
        sta     tmp2
        lda     ACMULT
        sta     tmp3   ; This is the amount at coldest
        mov     #100T,tmp4       ; 1.00 multiplier at 165 degrees
        mov     coolant,tmp5
        jsr     lininterp
        mov     tmp6,tmp14      ; result - save here temporarily


; Now the lookup table amount based on TPSDOT
        ldhx    #tpsdotrate
        sthx    tmp1
        lda     #$03
        sta     tmp3
        lda     tps
        sub     last_tps
        sta     tmp4    ; TPSDOT
        sta     tmp10   ; Save away for later use below
        jsr     ord_table_find
```

```
        clrh
        lda     tmp5
        tax
        lda     TPSAQ,x
        sta     tmp4
        decx
        lda     TPSAQ,x
        sta     tmp3
        lda     tmp10
        sta     tmp5
        jsr     lininterp       ; tmp6 has the result
        bra     FIND_TOTAL_A

; This is here to extend the jump range for the checks at the beginning (long-
jumps)
B_LO_CONT:
        bra     TAE_CHK_TIME
TDE2:   bra     TDE             ; Only to extend branch for assembler


; Now, the final applied acceleration enrichment amount is ((TMP6 * tmp14)/100) +
tmp13
FIND_TOTAL_A:
        lda     tmp6
        tax
        lda     tmp14
        mul
        pshx
        pulh
        ldx     #$64
        div
        bcs     UPPER_RAIL_AE
        psha
        pshh
        pula
        cmp     #$32
        ble     NDA1
        pula
        inca
        bra     ADD_TO_AE
NDA1:
        pula
        bra     ADD_TO_AE

UPPER_RAIL_AE:
        lda     #200T           ; Set to 20 milliseconds railed
        bra     ADD_TO_AE

ADD_TO_AE:

        add     tmp13           ; Add on the amount computed in cold temperature
enrich above
        sta     tmp6
        cmp     TPSACCEL
        blo     TAE_CHK_TIME
        lda     tmp6            ; Replace with this higher value
        sta     TPSACCEL
```

```
; Check if acceleration done
TAE_CHK_TIME:
        brset   TPSDEN,ENGINE,RST_ACCEL
        lda     tpsaclk
        cmp     tpsaclkcmp
        blo     MAE
RST_ACCEL:
        bclr    TPSAEN,ENGINE
        lda     #$64
      sta   tpsfuelcut
        clr     TPSACCEL
        bclr    aled,portc
      bclr  TPSDEN,ENGINE

        bra     MAE

; deaccel
TDE:
        lda     tmp2
        sub     tmp1

        cmp     tpsthresh
        blo     TDE_CHK_DONE
        brclr   TPSAEN,ENGINE,TDE_CHK_FUEL_CUT
        lda     #$64
      sta   TPSFUELCUT
        clr     TPSACCEL
        bclr    TPSAEN,ENGINE
        bclr    aled,portc
      bclr  TPSDEN,ENGINE
        bra     MAE
TDE_CHK_FUEL_CUT:
        lda     rpm
        cmp     #$0F
        blo     MAE
        lda     TPSDQ
        sta     TPSFUELCUT
        bset    TPSDEN,ENGINE
        bclr    TPSAEN,ENGINE
        bclr    aled,portc
        bra     MAE
TDE_CHK_DONE:
        brclr   TPSDEN,ENGINE,MAE
        bclr    TPSDEN,ENGINE
        lda     #$64
        sta     TPSFUELCUT
        clr     TPSACCEL
        bra     MAE

*****************************************************************************
**
**  Exhaust Gas Oxygen Sensor Measurement Section
**
**   Steps are the following:
**
**   If egodelta = 0 then goto skipo2
**   If RPM < RPMOXLIMIT then goto skipo2
**   If TPSAEN in ENGINE or TPSDEN in ENGINE are set, then goto skipo2
**   If coolant < egotemp then goto skipo2
```

```
**   If sech = 0 and secl < 30 seconds then got skipo2 (skip first 30 seconds)
**   If tps > 3.5 volts then goto skipo2
**
**   If egocount > egocountcmp
**   {
**      egocount = 0
**      If ego > 26 (counts, or 0.5 Volts) then (rich)
**      {
**         tmp = egocurr - egodelta
**         if tmp < egolimit then goto VETABLELOOKUP
**         egocorr = tmp
**         goto VETABLELOOKUP
**      }
**      else (lean)
**      {
**         tmp = egocorr + egodelta
**         if tmp > egolimit then goto VETABLELOOKUP
**         egocorr = tmp
**         goto VETABLELOOKUP
**      }
**   }
**
**   skipo2:
**   egocorr = 100%
**   goto VETABLELOOKUP
**
**
**
*************************************************************************
MAE:
        lda     egodelta
      beq   SKIPO2
      lda   rpm
      cmp   RPMOXLIMIT  ; Low-end of RPM
      blo   SKIPO2
      brset TPSAEN,ENGINE,SKIPO2
      brset TPSDEN,ENGINE,SKIPO2
      lda   coolant
      cmp   egotemp
      blo   SKIPO2
      lda   tps
      cmp   #$B2
      bhi   SKIPO2
        lda     sech
        bne     chk_o2_lag      ; if high seconds set then we can check o2
        lda     secl
        cmp     #$1E    ; 30 seconds threshold
        blo     SKIPO2

; Check if exceeded lag time - if so then we can modify egocorr
CHK_O2_LAG:
      lda   egocount
      cmp   egocountcmp
      blo   VETABLELOOKUP

; Check if rich/lean
      clr   egocount

        lda     config13                ; Check if Narrow-band (bit=0) or DIY-WB
```

```
(bit=1)
        bit     #$02                    ; Use BIT instead of brset because outside
of zero-page
      bne     WBO2TYPE                ; Branch if the bit is set
NBO2TYPE:
      lda   ego
      cmp   VOLTOXTARGET
      blo   O2_IS_LEAN
        bra     O2_IS_RICH
WBO2TYPE:
      lda   ego
      cmp   VOLTOXTARGET
      blo   O2_IS_RICH
        bra     O2_IS_LEAN

; rich o2 - lean out egocorr
O2_IS_RICH:
        lda     #$64
        sub     egolimit    ; Generate the lower limit rail point
        sta     tmp2
      lda   egocorr
      sub   egodelta
      sta   tmp1
      cmp   tmp2
      blo   VETABLELOOKUP ; railed at egolimit value
      lda   tmp1
      sta   egocorr
      bra   VETABLELOOKUP

; lean o2 - richen egocorr
O2_IS_LEAN:
        lda     #$64
        add     egolimit    ; Generate the upper limit rail point
        sta     tmp2

      lda   egocorr
      add   egodelta
      sta   tmp1
      cmp   tmp2
      bhi   VETABLELOOKUP ; railed at egolimit value
      lda   tmp1
      sta   egocorr
      bra   VETABLELOOKUP

; reset egocorr to 100%
SKIPO2:
      lda   #$64
      sta   egocorr
      bra   VETABLELOOKUP


*****************************************************************************
**
**  VE 3-D Table Lookup
**
**   This is used to determine value of VE based on RPM and MAP
**   The table looks like:
**
**      105 +....+....+....+....+....+....+....+
```

```
**          ..................................
**      100 +....+....+....+....+....+....+....+
**                      ...
**   KPA                  ...
**                         ...
**       35 +....+....+....+....+....+....+....+
**          5    15   25   35   45   55   65   75 RPM/100
**
**
**  Steps:
**   1) Find the bracketing KPA positions via ORD_TABLE_FIND, put index in tmp8 and
**       bounding values in tmp9(kpa1) and tmp10(kpa2)
**   2) Find the bracketing RPM positions via ORD_TABLE_FIND, store index in tmp11
and
**       bounding values in tmp13(rpm1) and tmp14(rpm2)
**   3) Using the VE table, find the table VE values for tmp15=VE(kpa1,rpm1),
**       tmp16=VE(kpa1,rpm2), tmp17 = VE(kpa2,rpm1), and tmp18 = VE(kpa2,rpm2)
**   4) Find the interpolated VE value at the lower KPA range :
**       x1=rpm1, x2=rpm2, y1=VE(kpa1,rpm1), y2=VE(kpa1,rpm2) - put in tmp19
**   5) Find the interpolated VE value at the upper KPA range :
**       x1=rpm1, x2=rpm2, y1=VE(kpa2,rpm1), y2=VE(kpa2,rpm2) - put in tmp11
**   6) Find the final VE value using the two interpolated VE values:
**       x1=kpa1, x2=kpa2, y1=VE_FROM_STEP_4, y2=VE_FROM_STEP_5
**
*********************************************************************
VETABLELOOKUP:

; First, determine if in Speed-density or Alpha-N mode. If in Alpha-N mode, then
;  replace the variable "kpa" with the contents of "tps". This will not break
anything, since
;  this check is performed again when multiplying MAP against the enrichments, and
;  the SCI version of the variable is MAP, not kpa

        lda     config13                ; Check if in speed-density or Aplha-N mode
        bit     #$04                    ; Use BIT instead of brset because outside
of zero-page
      beq     VE_STEP_1               ; Branch if the bit is clear

        lda     tps
        sta     kpa

VE_STEP_1:
        ldhx    #KPARANGEVEFLASH
        sthx    tmp1
        lda     #$07
        sta     tmp3
        lda     kpa
        sta     tmp4
        jsr     ORD_TABLE_FIND
        lda     tmp1
        lda     tmp2
        mov     tmp5,tmp8    ;Index
        mov     tmp1,tmp9    ;X1
        mov     tmp2,tmp10    ;X2
VE_STEP_2:
        ldhx    #RPMRANGEVEFLASH
        sthx    tmp1
        lda     #$07
        sta     tmp3
```

```
        lda     rpm
        sta     tmp4
        jsr     ORD_TABLE_FIND
        mov     tmp5,tmp11      ;Index
        mov     tmp1,tmp13      ;X1
        mov     tmp2,tmp14      ;X2

VE_STEP_3:

;TABLEWALK:
        clrh
        lda     #$08
        psha
        pulx
        lda     tmp8
        deca
        mul
        add     tmp11
        deca
        tax
        lda     VE,x
        sta     tmp15
        incx
        lda     VE,x
        sta     tmp16
        lda     #$08
        psha
        pulx
        lda     tmp8
        mul
        add     tmp11
        deca
        tax
        lda     VE,x
        sta     tmp17
        incx
        lda     VE,x
        sta     tmp18
        jmp     VE_STEP_4


VE_STEP_4:
        mov     tmp13,tmp1
        mov     tmp14,tmp2
        mov     tmp15,tmp3
        mov     tmp16,tmp4
        mov     rpm,tmp5
        jsr     lininterp
        mov     tmp6,tmp19

VE_STEP_5:
        mov     tmp13,tmp1
        mov     tmp14,tmp2
        mov     tmp17,tmp3
        mov     tmp18,tmp4
        mov     rpm,tmp5
        jsr     lininterp
        mov     tmp6,tmp11
```

```
VE_STEP_6:
        mov     tmp9,tmp1
        mov     tmp10,tmp2
        mov     tmp19,tmp3
        mov     tmp11,tmp4
        mov     kpa,tmp5
        lda     kpa
        jsr     lininterp
        mov     tmp6,vecurr


*****************************************************************************
**
** Computation of Fuel Parameters
**
** Remainders are maintained for hi-resolution calculations - results
**  converted back to 100 microsecond resolution at end.
**
** (Warm * Tpsfuelcut)/100 = R1 + rem1/100
** (Barcor * Aircor)/100 = R2 + rem2/100
** ((R1 + rem1/100) * (R2 + rem2/100)) / 100 = R3 + rem3/100
** (EGO * MAP)/100 = R4 + rem4/100
** ((R3 + rem3/100) * (R4 + rem4/100)) /100 = R5 + rem5/100
** (VE * REQ_FUEL)/100 = R6 + rem6/100
** ((R5 + rem5/100) * (R6 + rem6/100))  = R7
**
**
**
** Note that GAMMAE only includes Warm, Tpsfuelcut, Barocor, and Aircor (EGO no
longer included)
**
** Rationle on ordering: to prevent calculation overflow for boosted operations,
**  the variables have been ordered in specific "pairs" in the calculation:
**     EGO * MAP - when at WOT, EGO is set to 100%, so MAP can run up to 255%
without overflow
**     VE * REQ_FUEL - for boosted applications, REQ_FUEL tends to be low (below 10
ms) due to the
**                     added fuel requirements (i.e. large injectors), so VE
entries can
**                     be well above 100%.
**
*****************************************************************************


WARMACCEL_COMP:

        mov     warmcor,tmp10       ; Warmup Correction in tmp10
        clr     tmp11               ; tmp11 is zero
        mov     tpsfuelcut,tmp12    ; tpsfuelcut in tmp12
        clr     tmp13               ; tmp13 is zero
        jsr     Supernorm           ; do the multiply and normalization
        mov     tmp10,tmp1          ; save whole result in tmp1
        mov     tmp11,tmp2          ; save remainder in tmp2

        mov     barocor,tmp10       ; tmp10 is barometer percent
        clr     tmp11               ; zero to tmp11
        mov     aircor,tmp12        ; air temp correction % in tmp12
        clr     tmp13               ; tmp13 is zero
        jsr     Supernorm           ; multiply and divide by 100
```

```
                            ; result in tmp10:tmp11
        mov     tmp1,tmp12          ; move saved tmp1 into tmp12
        mov     tmp2,tmp13          ; move saved tmp2 into tmp13
        jsr     Supernorm           ; multiply/divide
        mov     tmp10,tmp5          ; save whole result into tmp5
        mov     tmp11,tmp6          ; save remainder into tmp6
        lda     tmp10
        sta     gammae

        mov     egocorr,tmp10       ; closed-loop correction percent into tmp10
        clr     tmp11               ; remainder is zero
        mov     kpa,tmp12           ; MAP into tmp12
        clr     tmp13               ; no remainder
        jsr     Supernorm           ; do the multiply and divide

        mov     tmp5,tmp12          ; take saved result in tmp5 and put into tmp12
        mov     tmp6,tmp13          ; tmp6 into tmp13
        jsr     Supernorm           ; mult/div
        mov     tmp10,tmp3          ; result (whole) save in tmp3
        mov     tmp11,tmp4          ; remainder result save in tmp4

        mov     vecurr,tmp10        ; VE into tmp10
        clr     tmp11               ; no remainder value for VE
        mov     REQ_FUEL,tmp12           ; req-fuel into tmp12
        clr     tmp13               ; no remainder
        jsr     Supernorm           ; mult/div

        mov     tmp3,tmp12          ; take previous result and put in tmp12
        mov     tmp4,tmp13          ; again for remainder
        jsr     Supernorm           ; multiply/divide

        mov     tmp10,tmp11

*************************************************************************
**
** Calculation of Battery Voltage Correction for Injector Opening Time
**
** Injector open time is implemented as a linear function of
**  battery voltage, from 7.2 volts (61 ADC counts) to 19.2 volts (164 counts),
**  with 13.2 volts (113 counts) being the nominal operating voltage
**
** INJOPEN = injector open time at 13.2 volts in mms
** BATTFAC = injector open adjustment factor 6 volts from 13.2V in mms
**
**
** + (INJOPEN + BATTFAC)
** +   *
** +                     (INJOPEN)
** +                         *
** +                                        (INJOPEN - BATTFAC)
** +                                                *
** +
** ++++++++++++++++++++++++++++++++++++++++++++++++++++++
**    7.2V                 13.2V                19.2
**
*************************************************************************
BATT_CORR_COMP:
        lda     #61T
        sta     tmp1
```

```
        lda     #164T
        sta     tmp2
        lda     injopen
        add     battfac
        sta     tmp3
        lda     injopen
        sub     battfac
        sta     tmp4
        bpl     MBFF    ; Check if minus comdition
        clr     tmp4
MBFF:
        mov     batt,tmp5
        jsr     lininterp  ; injector open time in tmp6
**
;        clr     tmp6
**
*****************************************************************************
**
** Calculation of Final Pulse Width
**
**  The following equation is evaluated here:
**
**  PWCALC = TMP6 + TMP11 + TPSACCEL - INJOCFUEL
**
**  Note that INJOCFUEL (injected fuel during injector open and close) is currently
a
**    constant - eventually it will be a function
**    of battery voltage.
*****************************************************************************
ADD_INJ_OFFSET:
        lda     tmp11
        add     tmp6
        bcs     MAX_PWM_ALLOWED
      add   TPSACCEL
      bcs   MAX_PWM_ALLOWED
        sub     InjOCFuel
        sta     pwcalc
        jmp     FINISHED_PW_COMP

MAX_PWM_ALLOWED:
        lda     #$FE
        sta     pwcalc

FINISHED_PW_COMP:
        jmp       LOOPER

*****************************************************************************
**
** * * * * Interrupt Section * * * * *
**
** Following interrupt service routines:
**  - Timer Overflow
**  - ADC Conversion Complete
**  - IRQ input line transistion from high to low
**  - Serial Communication received character
**  - Serial Communications transmit buffer empty (send another character)
**
*****************************************************************************
```

```
*************************************************************************
**
** Timer Rollover - Occurs every 1/10 of a millisecond - main timing clock
**
**
** Generate time rates:
**  1/10 milliseconds
**  1 milliseconds
**  1/10 seconds
**  seconds
**
** Also, in 1/10 millisecond section, turn on/off injector and
**  check RPM for stall condition
** In milliseconds section, fire off ADC conversion for next channel (5 total),
**  and wrap back when all channels done
**
*************************************************************************

TIMERROLL:
;=======================================================================
;***************** 0.1 millesecond section ****************************
;=======================================================================
        inc      mms          ; bump up 0.1 millisec variable

;======== Injector Firing Control ========
;===== Main Injector Control Logic =======
        brset    sched1,squirt,NEW_SQUIRT1
INJF1:
        brset    sched2,squirt,NEW_SQUIRT2
INJF2:
        brset    firing1,squirt,CHK_DONE_1
INJF3:
        brset    firing2,squirt,CHK_DONE_2
        jmp      CHECK_RPM

;=== Injector #1 - Start New Injection ===
NEW_SQUIRT1:
        bset     firing1,squirt  ; Turn on "firing" bit
        bclr     sched1,squirt   ; Turn off schedule bit (is now current operation)
        bset     inj1,squirt
        bset     sled,portc      ; squrt LED is ON

        mov      #$00,T1CH0H
        mov      INJPWM,T1CH0L
      bset   7,PORTA    ; ** Flyback Damper - turn on X0 for Inj1
        bclr     inject1,portd  ;^* * * Turn on Injector #1 (inverted drive)
      bra    INJF1

;=== Injector #2 - Start New Injection ===
NEW_SQUIRT2:
        bset     firing2,squirt  ; Turn on "firing" bit
        bclr     sched2,squirt   ; Turn off schedule bit (is now current operation)
        bset     inj2,squirt
        bset     sled,portc     ; squrt LED is ON

        mov      #$00,T1CH1H
        mov      INJPWM,T1CH1L
```

```
      bset   6,PORTA    ; ** Flyback Damper - turn on X1 for Injector 2
        bclr     inject2,portd  ;^* * * Turn on Injector #1 (inverted drive)
      bra    INJF2

;=== Injector #1 - Check for end of Injection ===
CHK_DONE_1:
        inc      pwrun1
        lda      pwrun1
        cmp      pw
        beq      OFF_INJ_1
      brset  crank,engine,INJF3 ; do not perform PWM limiting when cranking
        lda      pwrun1
        cmp      INJPWMT
        beq      PWM_LIMIT_1
      bra    INJF3
OFF_INJ_1:
        bclr     firing1,squirt
        bclr     sched1,squirt
        bclr     inj1,squirt
      bclr   7,PORTA    ; ** Flyback Damper - turn off X0
        bset     inject1,portd  ;^* * * Turn Off Injector #1 (inverted drive)
        mov      #Timerstop,T1SC
        mov      #t1scx_NO_PWM,T1SC0
        mov      #Timergo_NO_INT,T1SC
        bra      INJF3
PWM_LIMIT_1:
        mov      #Timerstop,T1SC
        mov      #T1SCX_PWM,T1SC0
        mov      #Timergo_NO_INT,T1SC
        bra      INJF3

;=== Injector #2 - Check for end of Injection ===
CHK_DONE_2:
        inc      pwrun2
        lda      pwrun2
        cmp      pw2
        beq      OFF_INJ_2
      brset  crank,engine,CHECK_RPM ; do not perform PWM limiting when cranking
        lda      pwrun2
        cmp      INJPWMT
        beq      PWM_LIMIT_2
      bra    CHECK_RPM
OFF_INJ_2:
        bclr     firing2,squirt
        bclr     sched2,squirt
        bclr     inj2,squirt
      bclr   6,PORTA    ; ** Flyback Damper - turn off X1 (for Inj 2)
        bset     inject2,portd  ;^* * * Turn Off Injector #2 (inverted drive)
        mov      #Timerstop,T1SC
        mov      #t1scx_NO_PWM,T1SC1
        mov      #Timergo_NO_INT,T1SC
        bra      CHECK_RPM
PWM_LIMIT_2:
        mov      #Timerstop,T1SC
        mov      #T1SCX_PWM,T1SC1
        mov      #Timergo_NO_INT,T1SC
        bra      CHECK_RPM
```

```
;=======Check RPM Section=====
CHECK_RPM:
        brclr    running,engine,ENABLE_THE_IRQ ; Branch if not running right now
        brset    firing1,squirt,CHK_RE_ENABLE
        brset    firing2,squirt,CHK_RE_ENABLE
        bclr     sled,portc     ; squrt LED is OFF - nothing is injecting


CHK_RE_ENABLE:
;====== Check for re-enabling of IRQ input pulses


        lda      rpmph          ; Get high byte of last rpm interval
        beq      RPMLOWBYTECHK  ; If zero go ahead check for half interval
        lda      rpmcl          ; Check current rpm interval
        cmp      #128T          ; 12.8 milliseconds is maximum (half of 25.6 ms = 1
in rpmph)
        beq      REARM_IRQ      ; time to re-arm IRQ
        bra      INCRPMER       ; Jump around rpm half interval check


RPMLOWBYTECHK:
      lda    rpmpl            ; Load in the latched previous RPM value
        lsra
      cmp    rpmcl              ; Is it the same value as current RPM Counter?
      bne    INCRPMER   ; If not then jump around this

REARM_IRQ:
      bset   ACK,INTSCR ; clear out any latched interrupts
      bclr   IMASK,INTSCR     ; enable interrupts again for IRQ

INCRPMER:
        inc      rpmcl
        bne      CHECK_MMS
        inc      rpmch
        lda      rpmch
        cmp      #$64           ; If RPMPH is 100 (or RPMPeriod = 2.5 sec) then
engine stalled
        bne      CHECK_MMS

        clr      engine         ; Engine is stalled, clear all in engine
        bclr     fuelp,porta    ; Turn off fuel Pump
        bclr     iasc,porta     ; Turn off IAC ** Bug Fix By Guy Hill **
        clr      rpmch
        clr      rpmcl
        bclr     sled,portc     ; squrt LED is OFF
        bclr     wled,portc     ; warmup LED is off
        clr      pw             ; zero out pulsewidth
        clr      rpm
      bclr   IMASK,INTSCR     ; enable all IRQ interrupts
      bra    CHECK_MMS

ENABLE_THE_IRQ:
      bclr   IMASK,INTSCR   ; Enable IRQ

CHECK_MMS:
        lda      mms
        cmp      #$0A
```

```
        bne     RTC_DONE

;===============================================================================
;******************** millesecond section ********************************
;===============================================================================
MSEC:
        inc     ms          ; bump up millisec
        clr     mms

; Fire off another ADC conversion, channel is pointed to by ADSEL
      lda   adsel
      ora   #%01000000
      sta   adscr

MSDONE:
        lda     ms
        cmp     #$64
        bne     RTC_DONE

;===============================================================================
;******************** 1/10 second section ********************************
;===============================================================================
ONETENTH:
        inc     tenth
        inc     tpsaclk
        clr     ms

; Save current TPS reading in last_tps variable to ompute TPSDOT in acceleration
;  enrichment section

        lda     tps
        sta     last_tps

        lda     tenth
        cmp     #$0A
        bne     RTC_DONE

;===============================================================================
;********************** seconds section ********************************
;===============================================================================
SECONDS:

        clr     tenth
        inc     secl        ; bump up second count
        bne     RTC_DONE
        inc     sech

RTC_DONE:
      lda     T2SC
      bclr    7,T2SC
      rti


*******************************************************************************
**
** IRQ - Input trigger for new pulse event
**
** This line is connected to the input trigger (i.e TACH signal from ignition
**  system), and schedules a new injector shot (injector actually opened in
```

```
**  1/10 timer section above)
**
*************************************************************************
DOSQUIRT:

      lda     igncount
      bne     EGOBUMP   ; Only increment ase counter if cylinder count is zero
        inc       asecount      ; Increment after-start enrichment counter
EGOBUMP:
        inc       egocount      ; Increment EGO step counter



        lda       rpmch
        sta       rpmph
        lda       rpmcl
        sta       rpmpl
        clr       rpmch
        clr       rpmcl

            lda     rpm
      sta     old_rpm1      ; Used in odd-fire code - save the last computed RPM
for average

        bset      fuelp,porta   ; Turn on fuel Pump
        bset      running,engine ; Set engine running value

        brset     crank,engine,SCHED_SQUIRT     ;Squirt on every pulse if cranking

        inc       igncount      ; Check to see if we are to squirt or skip
        lda       igncount
        cmp       divider
        beq       SCHED_SQUIRT
        lda       igncount
        cmp       #$0F       ; The maximum allowed - reset if match
        bne       IRQ_EXIT
        clr       igncount
        bra       IRQ_EXIT

SCHED_SQUIRT:
        bset      fuelp,porta   ; Turn on fuel Pump
        bset      running,engine ; Set engine running value
        clr       igncount


        brset     crank,engine,SCHED_BOTH
        lda       alternate
        beq       SCHED_BOTH
        inc       altcount
        brset     0,altcount,SCHED_INJ2
SCHED_INJ1:
        lda       pwcalc
        sta       pw             ; latched calculated pulsewidth
        clr       pwrun1         ; Running counter variable set to zero
        bset      sched1,squirt
        bset      inj1,squirt
        bra       IRQ_EXIT
SCHED_INJ2:
        lda       pwcalc
```

```
        sta       pw2             ; latched calculated pulsewidth
        clr       pwrun2          ; Running counter variable set to zero
        bset      sched2,squirt
        bset      inj2,squirt
        bra       IRQ_EXIT

SCHED_BOTH:
        lda       pwcalc
        sta       pw              ; Same pulsewidth
        sta       pw2             ; for both
        clr       pwrun1          ; Running counter variable set to zero
        bset      sched1,squirt
        bset      inj1,squirt

        clr       pwrun2          ; Running counter variable set to zero
        bset      sched2,squirt
        bset      inj2,squirt



IRQ_EXIT:
      bset    ACK,INTSCR      ; Flush out any new interrupts pending
      bset    IMASK,INTSCR    ; Disable IRQ interrupts
        bset      IRQF,INTSCR
        rti


*************************************************************************
**
** ADC - Interrupt for ADC conversion complete
**
*************************************************************************
ADCDONE:
        pshh            ; Do this because processor does not stack H

        clrh
; Store previous values for derivative
        lda     adsel
        tax
        lda     map,x
        sta     lmap,x  ; Store the old value

      lda   adr     ; Load in the new ADC reading
        add     map,x   ; Perform (map + last_map)/2 averaging (for all ADC
readings) - bug fix
        rora
      sta   map,x       ; MAP is entry point, offset is loaded in index register

      lda     adsel
      inca
      cmp   #$06
      bne   ADCPTR
      clra
ADCPTR:
      sta   adsel

        pulh
      rti
```

```
*****************************************************************************
**
** SCI Communications
**
** Communications is established when the PC communications program sends
** a command character - the particular character sets the mode:
**
** "A" = send all of the realtime variables via txport.
** "V" = send the VE table and constants via txport (128 bytes)
** "W"+<offset>+<newbyte> = receive new VE or constant byte value and
**  store in offset location
** "B" = jump to flash burner routine and burn VE/constant values in RAM into flash
** "C" = Test communications - echo back SECL
** "Q" = Send over Embedded Code Revision Number (divide number by 10 - i.e. $21T
is rev 2.1)
**
*****************************************************************************
IN_SCI_RCV:
        pshh
        lda     SCS1    ; Clear the SCRF bit by reading this register

        lda     txmode  ; Check if we are in the middle of a receive new
VE/constant
        cmp     #$05
        beq     TXMODE_5
        cmp     #$06
        beq     TXMODE_6

        lda     SCDR    ; Get the command byte
        cmp     #$41    ; Is the recieve character a big "A" -> Download real-time
variables?
        beq     MODE_A
        cmp     #$42
        beq     MODE_B
        cmp     #$43
        beq     MODE_C
        cmp     #$56
        beq     MODE_V
        cmp     #$57
        beq     MODE_W
      cmp   #$51
      beq   MODE_Q
        bra     DONE_RCV

MODE_A
        clr     txcnt                   ; Send back all real-time variables
        lda     #$01
        sta     txmode
        lda     #$17
        sta     txgoal
        bset    TE,SCC2                 ; Enable Transmit
        bset    SCTIE,SCC2              ; Enable transmit interrupt
        bra     DONE_RCV

MODE_B:
        mov     #$CC,flocker
        jsr     BURNER
        clr     flocker
        clr     txmode
```

```
        bra     DONE_RCV

MODE_C:
        clr     txcnt                   ; Just send back SECL variable to test comm
port
        lda     #$01
        sta     txmode
        lda     #$2
        sta     txgoal
        bset    TE,SCC2                 ; Enable Transmit
        bset    SCTIE,SCC2              ; Enable transmit interrupt
        bra     DONE_RCV

MODE_V:
        clr     txcnt
        lda     #$03
        sta     txmode
        lda     #$7E
        sta     txgoal
        bset    TE,SCC2                 ; Enable Transmit
        bset    SCTIE,SCC2              ; Enable transmit interrupt
        bra     DONE_RCV
MODE_W:
        lda     #$05
        sta     txmode
        bra     DONE_RCV

MODE_Q:
        clr     txcnt                   ; Just send back SECL variable to test comm
port
        lda     #$05
        sta     txmode
        lda     #$2
        sta     txgoal
        bset    TE,SCC2                 ; Enable Transmit
        bset    SCTIE,SCC2              ; Enable transmit interrupt
        bra     DONE_RCV


TXMODE_5:
        mov     SCDR,rxoffset
        inc     txmode
        bra     DONE_RCV
TXMODE_6:
        clrh
        lda     rxoffset
        tax
        lda     SCDR
        sta     VE,x
        clr     txmode
        bra     DONE_RCV

DONE_RCV
        pulh
        rti


*** Transmit Character Interrupt Handler ***************
IN_SCI_TX:
```

```
        pshh
        lda     SCS1    ; Clear the SCRF bit by reading this register
        clrh
        lda     txcnt
        tax
        lda     txmode
      cmp   #$05
      beq   IN_Q_MODE
        cmp     #$01
        bne     IN_V_MODE
IN_A_OR_C_MODE:
        lda     secl,X
        bra     CONT_TX
IN_V_MODE
        lda     ve,x
      bra   CONT_TX
IN_Q_MODE
      lda   REVNUM,X

CONT_TX:
        sta     SCDR
        lda     txcnt
        inca
        sta     txcnt
        cmp     txgoal
        bne     DONE_XFER

        clr     txcnt
        clr     txgoal
        clr     txmode

        bclr    TE,SCC2                 ; Disable Transmit
        bclr    SCTIE,SCC2              ; Disable transmit interrupt

DONE_XFER
        pulh
        rti


*****************************************************************************
**
** Dummy ISR - just performs RTS
**
*****************************************************************************
Dummy:                  ; Dummy vector - there just to keep the assembler happy
      rti

*****************************************************************************
**
** Various functions/subroutines Follow
**
**  - Ordered Table Search
**  - Linear Interpolation
**  - 32 x 16 divide
*****************************************************************************


*****************************************************************************
**
```

```
** Ordered Table Search
**
**  X is pointing to the start of the first value in the table
**  tmp1:2 initially hold the start of table address, then they hold the bound
values
**  tmp3 is the end of the table (nelements - 1)
**  tmp4 is the comparison value
**  tmp5 is the index result - if zero then comp value is less than beginning of
table, and
**    if equal to nelements then it is rail-ed at upper end
**
*****************************************************************************
ORD_TABLE_FIND:
        clr     tmp5
        ldhx    tmp1
        lda     ,x
        sta     tmp1
        sta     tmp2
;        cmp     tmp4
;        bhi     GOT_ORD_NUM
REENT:
        incx
        inc     tmp5
        mov     tmp2,tmp1
        lda     ,x
        sta     tmp2

        cmp     tmp4
        bhi     GOT_ORD_NUM
        lda     tmp5
        cmp     tmp3
        bne     REENT

;        inc     tmp5
;        mov     tmp2,tmp1
GOT_ORD_NUM:
        rts

*****************************************************************************
**
** Linear Interpolation - 2D
**
**            (y2 - y1)
**  Y = Y1 +  --------- * (x - x1)
**            (x2 - x1)
**
**   tmp1 = x1
**   tmp2 = x2
**   tmp3 = y1
**   tmp4 = y2
**   tmp5 = x
**   tmp6 = y
*****************************************************************************
LININTERP:
        clr     tmp7    ; This is the negative slope detection bit
        mov     tmp3,tmp6
CHECK_LESS_THAN:
        lda     tmp5
        cmp     tmp1
```

```
        bhi     CHECK_GREATER_THAN
        bra     DONE_WITH_INTERP
CHECK_GREATER_THAN:
        lda     tmp5
        cmp     tmp2
        blo     DO_INTERP
        mov     tmp4,tmp6
        bra     DONE_WITH_INTERP

DO_INTERP:
        mov     tmp3,tmp6
        lda     tmp2
        sub     tmp1
        beq     DONE_WITH_INTERP
        psha
        lda     tmp4
        sub     tmp3
      bcc   POSINTERP
        nega
        inc     tmp7
POSINTERP:
        psha
        lda     tmp5
        sub     tmp1
        beq     ZERO_SLOPE
        pulx
        mul
        pshx
        pulh
        pulx

        div

        psha
        lda     tmp7
        bne     NEG_SLOPE
        pula
        add     tmp3
        sta     tmp6
        bra     DONE_WITH_INTERP
NEG_SLOPE:
        pula
        sta     tmp7
        lda     tmp3
        sub     tmp7
        sta     tmp6
        bra     DONE_WITH_INTERP
ZERO_SLOPE:
        pula            ;clean stack
        pula            ;clean stack
DONE_WITH_INTERP:
        rts


*******************************************************************************
*******************************************************************************
*
*     32 x 16 Unsigned Divide
```

```
*
*     This routine takes the 32-bit dividend stored in INTACC1.....INTACC1+3
*     and divides it by the 16-bit divisor stored in INTACC2:INTACC2+1.
*     The quotient replaces the dividend and the remainder replaces the divisor.
*
UDVD32    EQU     *
*
DIVIDEND  EQU     INTACC1+2
DIVISOR   EQU     INTACC2
QUOTIENT  EQU     INTACC1
REMAINDER EQU     INTACC1
*
        PSHH                            ;save h-reg value
        PSHA                            ;save accumulator
        PSHX                            ;save x-reg value
        AIS     #-3                     ;reserve three bytes of temp storage
        LDA     #!32                    ;
        STA     3,SP                    ;loop counter for number of shifts
        LDA     DIVISOR                 ;get divisor msb
        STA     1,SP                    ;put divisor msb in working storage
        LDA     DIVISOR+1               ;get divisor lsb
        STA     2,SP                    ;put divisor lsb in working storage
*
*     Shift all four bytes of dividend 16 bits to the right and clear
*     both bytes of the temporary remainder location
*
        MOV     DIVIDEND+1,DIVIDEND+3   ;shift dividend lsb
        MOV     DIVIDEND,DIVIDEND+2     ;shift 2nd byte of dividend
        MOV     DIVIDEND-1,DIVIDEND+1   ;shift 3rd byte of dividend
        MOV     DIVIDEND-2,DIVIDEND     ;shift dividend msb
        CLR     REMAINDER               ;zero remainder msb
        CLR     REMAINDER+1             ;zero remainder lsb
*
*     Shift each byte of dividend and remainder one bit to the left
*
SHFTLP  LDA     REMAINDER               ;get remainder msb
        ROLA                            ;shift remainder msb into carry
        ROL     DIVIDEND+3              ;shift dividend lsb
        ROL     DIVIDEND+2              ;shift 2nd byte of dividend
        ROL     DIVIDEND+1              ;shift 3rd byte of dividend
        ROL     DIVIDEND                ;shift dividend msb
        ROL     REMAINDER+1             ;shift remainder lsb
        ROL     REMAINDER               ;shift remainder msb
*
*     Subtract both bytes of the divisor from the remainder
*
        LDA     REMAINDER+1             ;get remainder lsb
        SUB     2,SP                    ;subtract divisor lsb from remainder lsb
        STA     REMAINDER+1             ;store new remainder lsb
        LDA     REMAINDER               ;get remainder msb
        SBC     1,SP                    ;subtract divisor msb from remainder msb
        STA     REMAINDER               ;store new remainder msb
        LDA     DIVIDEND+3              ;get low byte of dividend/quotient
        SBC     #0                      ;dividend low bit holds subtract carry
        STA     DIVIDEND+3              ;store low byte of dividend/quotient
*
*     Check dividend/quotient lsb. If clear, set lsb of quotient to indicate
*     successful subraction, else add both bytes of divisor back to remainder
*
```

```
        BRCLR   0,DIVIDEND+3,SETLSB     ;check for a carry from subtraction
                                        ;and add divisor to remainder if set
        LDA     REMAINDER+1             ;get remainder lsb
        ADD     2,SP                    ;add divisor lsb to remainder lsb
        STA     REMAINDER+1             ;store remainder lsb
        LDA     REMAINDER               ;get remainder msb
        ADC     1,SP                    ;add divisor msb to remainder msb
        STA     REMAINDER               ;store remainder msb
        LDA     DIVIDEND+3              ;get low byte of dividend
        ADC     #0                      ;add carry to low bit of dividend
        STA     DIVIDEND+3              ;store low byte of dividend
        BRA     DECRMT                  ;do next shift and subtract

SETLSB  BSET    0,DIVIDEND+3            ;set lsb of quotient to indicate
                                        ;successive subtraction
DECRMT  DBNZ    3,SP,SHFTLP             ;decrement loop counter and do next
                                        ;shift
*
*     Move 32-bit dividend into INTACC1.....INTACC1+3 and put 16-bit
*     remainder in INTACC2:INTACC2+1
*
        LDA     REMAINDER               ;get remainder msb
        STA     1,SP                    ;temporarily store remainder msb
        LDA     REMAINDER+1             ;get remainder lsb
        STA     2,SP                    ;temporarily store remainder lsb
        MOV     DIVIDEND,QUOTIENT       ;
        MOV     DIVIDEND+1,QUOTIENT+1   ;shift all four bytes of quotient
        MOV     DIVIDEND+2,QUOTIENT+2   ; 16 bits to the left
        MOV     DIVIDEND+3,QUOTIENT+3   ;
        LDA     1,SP                    ;get final remainder msb
        STA     INTACC2                 ;store final remainder msb
        LDA     2,SP                    ;get final remainder lsb
        STA     INTACC2+1               ;store final remainder lsb
*
*     Deallocate local storage, restore register values, and return from
*     subroutine
*
        AIS     #3                      ;deallocate temporary storage
        PULX                            ;restore x-reg value
        PULA                            ;restore accumulator value
        PULH                            ;restore h-reg value
        RTS                             ;return
*******************************************************************************
```

```
*******************************************************************************
**
** FLASH Programming Routine
**  Copied into RAM and executed there
*******************************************************************************

; Erase VE and Constants in FLASH - 128 byte erase
ERASE_N_BURN:
        lda       flocker
```

```
        cmp       #$CC
        beq       ERASE_CONT
        rts
ERASE_CONT:
        ldhx      #FLCR
        lda       #%00000010
        sta       ,x     ; Set erase bit
        lda       FLBPR  ; Read block protect register
        sta       VETABLE ; Write to any address in page to be erased
        lda       #$0C    ; Delay 10 us
        bsr       DELAYE
        lda       #%00001010
        sta       ,x      ; set HVEN
        lda       #$F0    ; 200 us
        bsr       DELAYE
        bsr       DELAYE
        bsr       DELAYE
        bsr       DELAYE
        bsr       DELAYE   ; total of 1 ms
        lda       #%00001000
        sta       ,x      ; clear erase bit
        lda       #$06
        bsr       DELAYE   ; 5 us
        lda       #$00000000
        sta       ,x      ; clear HVEN
        lda       #$02
        bsr       DELAYE
        bra       VE_BURNER

DELAYE:
        deca
        bne       DELAYE
        rts

; Now Burn VE (you can only freakin burn 64 bytes at a time)
VE_BURNER
        lda       #%00000001
        sta       ,x      ; Set PGM bit
        lda       FLBPR   ; read from FLASH block protect register
        sta       VETABLE ; wrte to any address within pgmed range
        lda       #$0C
        bsr       DELAYVE
        lda       #%00001001
        sta       ,x      ; HVEN set
        lda       #$06    ;
        bsr       DELAYVE   ; 5 us
        clrh
        clrx
LOOP_TO_BURN_VE:
        lda       VE,x
        sta       VETABLE,x
        lda       #$22     ; 28 us, everything else takes 2 us => 30 us total
        bsr       DELAYVE
        incx
        cpx       #$40
        beq       SD1
        bra       LOOP_TO_BURN_VE
SD1:
        ldhx      #FLCR
```

```
        lda       #%00001000
        sta       ,x
        lda       #$06
        bsr       DELAYVE
        lda       #%00000000
        sta       ,x
        lda       #$01
        bsr       DELAYVE
        bra       BURN_C
DELAYVE:
        deca
        bne       DELAYVE
        rts

; Now Burn COnstants
BURN_C:
        lda       #%00000001
        sta       ,x      ; Set PGM bit
        lda       FLBPR   ; read from FLASH block protect register
        sta       CWUTABLE ; wrte to any address within pgmed range
        lda       #$0C
        bsr       DELAYC
        lda       #%00001001
        sta       ,x      ; HVEN set
        lda       #$06    ;
        bsr       DELAYC   ; 5 us
        clrh
        clrx
LOOP_TO_BURN_C:
        lda       CWU,x
        sta       CWUTABLE,x
        lda       #$22     ; 28 us, everything else takes 2 us => 30 us total
        bsr       DELAYC
        incx
        cpx       #$40
        beq       SD2
        bra       LOOP_TO_BURN_C
SD2:
        ldhx      #FLCR
        lda       #%00001000
        sta       ,x
        lda       #$06
        bsr       DELAYC
        lda       #%00000000
        sta       ,x
        lda       #$01
        bsr       DELAYC
        rts               ; Exit from all of the burn routine
DELAYC:
        deca
        bne       DELAYC
        rts
DONE_C:



**********************************************************************
**
```

```
** Computation of Normalized Variables
**
**  The following is the form of the evaluation for the normalized variables:
**
**  (A rem A * B)
**  -------------  = C rem C
**      100
**
**  Where A = Whole part of the percentage,
**        rem A = Remainder of A from previous calculation (range 0 to 99)
**        B = Percentage multiplied (this always has a zero remainder)
**        C = Whole part of result
**        rem C = remainder of result
**
**
**  Calculation is preformed by the following method:
**
**     |(A * B) + (rem A * B)|
**     |           -----------|
**     |               100    |
**     ----------------------- = C rem C
**             100
**
**
**   Inputs:  tmp10 = A
**            tmp11 = rem A
**            tmp12 = B
**
**   Outputs: tmp10 = C
**            tmp11 = rem C
**            tmp13 = high order part of (A rem A) * B
**            tmp14 = low order part of (A rem A) * B
**
*************************************************************************
Supernorm:
      lda   tmp10 ;A
        tax
        lda     tmp12   ;B
        mul
        stx     tmp13   ;High order of A * B
        sta     tmp14   ;Low order of A * B

        lda     tmp11   ;rem A
        tax
        lda     tmp12   ;B
        mul
        pshx
        pulh
        ldx     #$64    ;100
        div

        adc     tmp14   ;Add to lower part
        sta     tmp14   ;Store back
        bcc     Roundrem ;Branch is no carry occurred
        inc     tmp13   ;Increment high-order part because an overflow occurred in
add

Roundrem:
        pshh
```

```
        pula
        cmp     #$32    ;Round if division remainder is greater than 50
        ble     FinalNorm
        lda     tmp14
        adc     #$01
        sta     tmp14
        bcc     FinalNorm
        inc     tmp13


FinalNorm:
        lda     tmp13
        psha
        pulh
        lda     tmp14
        ldx     #$64    ;100
        div
        bcs     RailCalc
        sta     tmp10
        pshh
        pula
        sta     tmp11

        cmp     #$32    ;Round if division remainder is greater than 50
        ble     ExitSN
        lda     tmp11
        adc     #$01
        sta     tmp11
        bcc     ExitSN
        lda     tmp10
        add     #$01
        sta     tmp10
        bne     ExitSN

RailCalc:
        mov     #$FF,tmp10      ;Rail value if rollover

ExitSN:
        rts


;--------------


        org     $FAC3   ; start of bootloader-defined jump table/vector
        db      $12     ; scbr regi init value
        db      %00000001 ; config1
        db      %00000001 ; config2
        dw      {rom_start + 128} ; megasquirt code start
        dw      $FB00    ; bootloader start

; Vector table
;     org   vec_timebase



        db      $CC
      dw    Dummy ;Timebase
        db      $CC
```

```
      dw    ADCDONE     ;ADC Conversion Complete
        db      $CC
      dw    Dummy ;Keyboard pin
        db      $CC
      dw    IN_SCI_TX   ;SCI transmission complete/transmitter empty
        db      $CC
      dw    IN_SCI_RCV  ;SCI input idle/receiver full
        db      $CC
      dw    Dummy ;SCI parity/framing/noise/receiver_overrun error
        db      $CC
      dw    Dummy ;SPI Transmitter empty
        db      $CC
      dw    Dummy ;SPI mode/overflow/receiver full
        db      $CC
      dw    TIMERROLL   ;TIM2 overflow
        db      $CC
      dw    Dummy ;TIM2 Ch1
        db      $CC
      dw    Dummy ;TIM2 Ch0
        db      $CC
      dw    Dummy ;TIM1 overflow
        db      $CC
      dw    Dummy ;TIM1 Ch1
        db      $CC
      dw    Dummy ;TIM Ch0
        db      $CC
      dw    Dummy ;CGM
        db      $CC
      dw    DOSQUIRT;IRQ
        db      $CC
      dw    Dummy ;SWI
        db      $CC
      dw    Start


; Lookup Tables
        org     $F100
      include "barofactor4115.inc"
      include "barofactor4250.inc"
        include "kpafactor4115.inc"
        include "kpafactor4250.inc"
        include "thermfactor.inc"
        include "airdenfactor.inc"

        org     $E000
; Initial VE table, copied into RAM at startup
VETABLE:
        db      39T     ; VE (0,0)
        db      40T     ; VE (0,1)
        db      41T     ; VE (0,2)
        db      44T     ; VE (0,3)
        db      44T     ; VE (0,4)
        db      44T     ; VE (0,5)
        db      45T     ; VE (0,6)
        db      45T     ; VE (0,7)
        db      47T     ; VE (1,0)
        db      47T     ; VE (1,1)
        db      51T     ; VE (1,2)
        db      51T     ; VE (1,3)
```

```
        db      50T     ; VE (1,4)
        db      50T     ; VE (1,5)
        db      50T     ; VE (1,6)
        db      50T     ; VE (1,7)
        db      52T     ; VE (2,0)
        db      55T     ; VE (2,1)
        db      55T     ; VE (2,2)
        db      57T     ; VE (2,3)
        db      60T     ; VE (2,4)
        db      61T     ; VE (2,5)
        db      61T     ; VE (2,6)
        db      65T     ; VE (2,7)
        db      59T     ; VE (3,0)
        db      60T     ; VE (3,1)
        db      60T     ; VE (3,2)
        db      65T     ; VE (3,3)
        db      66T     ; VE (3,4)
        db      70T     ; VE (3,5)
        db      70T     ; VE (3,6)
        db      70T     ; VE (3,7)
        db      61T     ; VE (4,0)
        db      63T     ; VE (4,1)
        db      65T     ; VE (4,2)
        db      65T     ; VE (4,3)
        db      68T     ; VE (4,4)
        db      70T     ; VE (4,5)
        db      72T     ; VE (4,6)
        db      75T     ; VE (4,7)
        db      65T     ; VE (5,0)
        db      72T     ; VE (5,1)
        db      72T     ; VE (5,2)
        db      74T     ; VE (5,3)
        db      74T     ; VE (5,4)
        db      75T     ; VE (5,5)
        db      75T     ; VE (5,6)
        db      77T     ; VE (5,7)
        db      70T     ; VE (6,0)
        db      74T     ; VE (6,1)
        db      74T     ; VE (6,2)
        db      75T     ; VE (6,3)
        db      75T     ; VE (6,4)
        db      77T     ; VE (6,5)
        db      77T     ; VE (6,6)
        db      78T     ; VE (6,7)
        db      75T     ; VE (7,0)
        db      77T     ; VE (7,1)
        db      79T     ; VE (7,2)
        db      82T     ; VE (7,3)
        db      82T     ; VE (7,4)
        db      82T     ; VE (7,5)
        db      82T     ; VE (7,6)
        db      85T     ; VE (7,7)
CWUTABLE:
        db      120T    ; CWU
        db      040T    ; CWH
        db      35T     ; AWEV
        db      250T    ; AWC
        db      180T    ; WWU (-40 F)
        db      180T    ; WWU (-20 F)
```

```
        db      160T    ; WWU (0 F)
        db      150T    ; WWU (20 F)
        db      135T    ; WWU (40 F)
        db      125T    ; WWU (60 F)
        db      113T    ; WWU (80 F)
        db      108T    ; WWU (100 F)
        db      102T    ; WWU (130 F)
        db      100T    ; WWU (160 F)
        db      20T      ; TPSAQ (tpsdot=0.1)
        db      50T      ; TPSAQ (tpsdot=0.4)
        db      105T      ; TPSAQ (tpsdot=0.8)
        db      150T     ; TPSAQ (tpsdot=1.5)
        db      90T     ; TPSACOLD (ms to add in when cold)
        db      03T     ; TPSTHRESH
        db      02T    ; TPSASYNC (accel enrich time in 1/10 second increments)
        db      100T    ; TPSDQ
        db      200T    ; EGOTEMP
        db      16T      ; EGOCOUNTCMP
        db      1T      ; EGODELTA
        db      15T     ; EGOLIMIT
        db      155T    ; REQFUEL
        db      4T      ; DIVIDER
        db      1T      ; Alternate
        db      10T     ; INJOPEN
        db      0T      ; INJOCFUEL
        db      75T    ; INJPWM
        db      255T    ; INJPWMT
        db      12T      ; BATTFAC
        db      $05     ; RPMK[0]
        db      $DC     ; RPMK[1]
RPMRANGEVEFLASH:
        db      5T      ; RPMRANGEVE[0]
        db      10T     ; RPMRANGEVE[1]
        db      15T     ; RPMRANGEVE[2]
        db      20T     ; RPMRANGEVE[3]
        db      28T     ; RPMRANGEVE[4]
        db      36T     ; RPMRANGEVE[5]
        db      44T     ; RPMRANGEVE[6]
        db      52T     ; RPMRANGEVE[7]
KPARANGEVEFLASH:
        db      20T     ; KPARANGEVE[0]
        db      30T     ; KPARANGEVE[1]
        db      40T     ; KPARANGEVE[2]
        db      50T     ; KPARANGEVE[3]
        db      60T     ; KPARANGEVE[4]
        db      75T     ; KPARANGEVE[5]
        db      90T     ; KPARANGEVE[6]
        db      100T    ; KPARANGEVE[7]
        db      113T    ; Config11
        db      112T    ; Config12
        db      00T     ; Config13
        db      20T     ; PRIMEP
        db      13T     ; RPMOXLIMIT
        db      185T    ; FAST IDLE TEMPERATURE
        db      26T     ; VOLTOXTARGET
        db      100T    ; ACCELMULT
        db      00T     ; BLANK[3]
        db      00T     ; BLANK[4]
        db      00T     ; BLANK[5]
```

```
        db      00T     ; BLANK[6]



; This is used to set the bin coolant range for WWU
WWURANGE:
        db      0T
        db      20T
        db      40T
        db      60T
        db      80T
        db      100T
        db      120T
        db      140T
        db      170T
        db      200T
tpsdotrate:
        db      05T     ; 0.1 volts delta
        db      20T     ; 0.4 volt delta
        db      40T     ; 0.8 volt
        db      77T     ; 1.5 volt

REVNUM: db      20T     ; Revision 3.000
Signature db 32T,' ** V3.000Embedded Code by B&G **'

        include "boot_r12.asm"

      end
```