```
;*****************************************************************************************
;* Final Project: Robot Guidance Program (9S32C)                                         *
;*****************************************************************************************

; export symbols
        XDEF Entry, _Startup            ; export 'Entry' symbol
        ABSENTRY Entry                  ; for absolute assembly: mark this as application entry
point

; Include derivative-specific definitions
                INCLUDE 'derivative.inc'

;*****************************************************************************************
;* EQUATES SECTION                                                                       *
;*****************************************************************************************
LCD_DAT       EQU   PORTB               ; LCD data port, bits - PB7,...,PB0
LCD_CNTR      EQU   PTJ                 ; LCD control port, bits - PJ6(RS),PJ7(E)
LCD_E         EQU   $80                 ; LCD E-signal pin
LCD_RS        EQU   $40                 ; LCD RS-signal pin

;Guidance Variables (change depending on robot)
;-----------------------------------------------------------------------------------------
PTH_A_INT     EQU   $C0                 ; Path detection threshold
PTH_B_INT     EQU   $CA                 ; ""
PTH_C_INT     EQU   $CA                 ; ""
PTH_D_INT     EQU   $CA                 ; ""

PTH_E_INT     EQU   $70                 ; If SENSOR_LINE < PTH_E_INT robot must shift
right
PTH_F_INT     EQU   $72                 ; If SENSOR_LINE > PTH_F_INT robot must shift left

INC_DIS       EQU   300                 ; INCREMENT distance
FWD_DIS       EQU   2000                ; FORWARD distance
REV_DIS       EQU   1000                ; REVERSE distance
STR_DIS       EQU   1000                ; STRAIGHT distance
TRN_DIS       EQU   10000               ; TURN distance
UTRN_DIS      EQU   12000               ; U-TURN distance
;-----------------------------------------------------------------------------------------

PRI_PTH_INT   EQU   0                   ; Primary path value
SEC_PTH_INT   EQU   1                   ; Secondary path value

START         EQU   0                   ; START state value
```

```
FWD          EQU   1                ; FORWARD state value
REV          EQU   2                ; REVERSE state value
RT_TRN       EQU   3                ; RIGHT TURN state value
LT_TRN       EQU   4                ; LEFT TURN state value
BK_TRK       EQU   5                ; BACKTRACK state value
SBY          EQU   6                ; STANDBY state value

;*****************************************************************************************
;* VARIABLE SECTION                                                                      *
;*****************************************************************************************
            ORG   $3850

CRNT_STATE     DC.B  6              ; Current state register

COUNT1         DC.W  0              ; initialize 2-byte COUNT1 to $0000
COUNT2         DC.W  0              ; initialize 2-byte COUNT2 to $0000

A_DETN         DC.B  0              ; SENSOR A detection (PATH = 1, NO PATH = 0)
B_DETN         DC.B  0              ; SENSOR B detection (PATH = 1, NO PATH = 0)
C_DETN         DC.B  0              ; SENSOR C detection (PATH = 1, NO PATH = 0)
D_DETN         DC.B  0              ; SENSOR D detection (PATH = 1, NO PATH = 0)
E_DETN         DC.B  0              ; SENSOR E detection (PATH = 1, NO PATH = 0)
F_DETN         DC.B  0              ; SENSOR F detection (PATH = 1, NO PATH = 0)

RETURN         DC.B  0              ; RETURN (TRUE = 1, FALSE = 0)
NEXT_D         DC.B  1              ; Next direction instruction

TEN_THOUS      DS.B  1              ; 10,000 digit
THOUSANDS      DS.B  1              ; 1,000 digit
HUNDREDS       DS.B  1              ; 100 digit
TENS           DS.B  1              ; 10 digit
UNITS          DS.B  1              ; 1 digit
BCD_SPARE      DS.B  10
NO_BLANK       DS.B  1              ; Used in 'leading zero' blanking by BCD2ASC

SENSOR_LINE    DC.B  $0             ; (LINE ) Storage for guider sensor readings
SENSOR_BOW     DC.B  $0             ; (FRONT) Initialized to test values
SENSOR_PORT    DC.B  $0             ; (LEFT )
SENSOR_MID     DC.B  $0             ; (MIDDLE)
SENSOR_STBD    DC.B  $0             ; (RIGHT)
SENSOR_NUM     DS.B  1              ; The currently selected sensor
TEMP           DS.B  1              ; Temporary location
```

```
;*************************************************************************************
;* CODE SECTION                                                                      *
;*************************************************************************************
            ORG   $4000                ; Where the code starts ----------------------+
Entry:                                 ;                                             |
_Startup:                               ;                                            |
            LDS   #$4000               ; Initialize the stack pointer                I
                                    ;                                                N
            JSR   initPORTS             ;                                           I
                                    ;                                                T
            JSR   initAD               ; Initialize ATD converter                    I
                                    ;                                                A
            JSR   initLCD              ; Initialize the LCD                          L
            JSR   clrLCD               ; Clear LCD & home cursor                      I
                                    ;                                                Z
            JSR   initTCNT             ; Initialize the TCNT                         A
                                    ;                                                T
            CLI                      ; Enable interrupts                            I
                                    ;                                                O
            LDX   #msg1                ; Display msg1                                N
            JSR   putsLCD               ; ""                                        |
                                    ;                                                |
            LDAA  #$8A                 ; Move LCD cursor to the end of msg1          |
            JSR   cmd2LCD                ; ""                                       |
            LDX   #msg2                ; Display msg2                                |
            JSR   putsLCD               ; ""                                       |
                                    ;                                                |
            LDAA  #$C0                 ; Move LCD cursor to the 2nd row              |
            JSR   cmd2LCD                ; ""                                       |
            LDX   #msg3                ; Display msg3                                |
            JSR   putsLCD               ; ""                                       |
                                    ;                                                |
            LDAA  #$C7                 ; Move LCD cursor to the end of msg3          |
            JSR   cmd2LCD                ; ""                                       |
            LDX   #msg4                ; Display msg4                                |
            JSR   putsLCD               ; ""                                       |
                                    ;----------------------------------------------+

                                    ;----------------------------------------------+
        MAIN: JSR   UPDT_READING           ;                                        M
            JSR   UPDT_DISPL             ;                                        A
            LDAA  CRNT_STATE               ;                                      I
            JSR   DISPATCHER             ;                                        N
```

```
            BRA   MAIN                ;                                |
                                      ;----------------------------------------+

;*******************************************************************************************
;* DATA SECTION                                                                 *
;*******************************************************************************************
        msg1: dc.b  "S:",0              ; Current state label
        msg2: dc.b  "R:",0              ; Sensor readings label
        msg3: dc.b  "V:",0              ; Battery voltage label
        msg4: dc.b  "B:",0              ; Bumper status label

         tab: dc.b  "START  ",0           ; States
            dc.b  "FWD    ",0           ; ""
            dc.b  "REV    ",0           ; ""
            dc.b  "RT_TRN ",0           ; ""
            dc.b  "LT_TRN ",0           ; ""
            dc.b  "RETURN ",0           ; ""
            dc.b  "STANDBY",0           ; ""

;*******************************************************************************************
;* SUBROUTINE SECTION                                                           *
;*******************************************************************************************
;+--------------------------------------------------------------------------------------+
;| Starboard (Right) Motor ON                                                           |
;+--------------------------------------------------------------------------------------+
STARON      BSET  PTT,%00100000
            RTS

;+--------------------------------------------------------------------------------------+
;| Starboard (Right) Motor OFF                                                          |
;+--------------------------------------------------------------------------------------+
STAROFF     BCLR  PTT,%00100000
            RTS

;+--------------------------------------------------------------------------------------+
;| Starboard (Right) Motor FWD                                                          |
;+--------------------------------------------------------------------------------------+
STARFWD     BCLR  PORTA,%00000010
            RTS

;+--------------------------------------------------------------------------------------+
;| Starboard (Right) Motor REV                                                          |
;+--------------------------------------------------------------------------------------+
```

```
STARREV         BSET  PORTA,%00000010
                RTS

;+--------------------------------------------------------------------------------------+
;| Port (Left) Motor ON                                                                 |
;+--------------------------------------------------------------------------------------+
PORTON          BSET  PTT,%00010000
                RTS

;+--------------------------------------------------------------------------------------+
;| Port (Left) Motor OFF                                                                |
;+--------------------------------------------------------------------------------------+
PORTOFF         BCLR  PTT,%00010000
                RTS

;+--------------------------------------------------------------------------------------+
;| Port (Left) Motor FWD                                                                |
;+--------------------------------------------------------------------------------------+
PORTFWD         BCLR  PORTA,%00000001
                RTS

;+--------------------------------------------------------------------------------------+
;| Port (Left) Motor REV                                                                |
;+--------------------------------------------------------------------------------------+
PORTREV         BSET  PORTA,%00000001
                RTS

;*******************************************************************************************
;* STATES SECTION                                                                          *
;*******************************************************************************************
DISPATCHER      CMPA  #START                    ; If it's the START state ----------------+
                BNE   NOT_START                 ;                                        |
                JSR   START_ST                  ; then call START_ST routine             D
                RTS                             ; and exit                               I
;-----------------------------------------------;                                        S
NOT_START       CMPA  #FWD                      ; Else if it's the FORWARD state         P
                BNE   NOT_FORWARD               ;                                        A
                JMP   FWD_ST                    ; then call the FWD_ST routine           T
;-----------------------------------------------;                                        C
NOT_FORWARD     CMPA  #RT_TRN                   ; Else if it's the RIGHT_TURN state      H
                BNE   NOT_RT_TRN                ;                                        E
                JSR   RT_TRN_ST                 ; then call the RT_TRN_ST routine        R
                RTS                             ; and exit                               |
```

```
;-----------------------------------------------;                                 |
NOT_RT_TRN      CMPA  #LT_TRN               ; Else if it's the LEFT_TURN state      |
            BNE   NOT_LT_TRN              ;                                       |
            JSR   LT_TRN_ST             ; then call LT_TRN_ST routine             |
            RTS                       ; and exit                                  |
;-----------------------------------------------;                                 |
NOT_LT_TRN      CMPA  #REV                ; Else if it's the REVERSE state        |
            BNE   NOT_REVERSE             ;                                       |
            JSR   REV_ST                ; then call the REV_ST routine            |
            RTS                       ; and exit                                  |
;-----------------------------------------------;                                 |
NOT_REVERSE     CMPA  #BK_TRK               ; Else if it's the BACKTRACK state      |
            BNE   NOT_BK_TRK              ;                                       |
            JMP   BK_TRK_ST             ; then call the BK_TRK_ST routine         |
;-----------------------------------------------;                                 |
NOT_BK_TRK      CMPA  #SBY                ; Else if it's the STANDBY state        |
            BNE   NOT_SBY               ;                                       |
            JSR   SBY_ST                ; then call the SBY_ST routine            |
            RTS                       ; and exit                                  |
;-----------------------------------------------;                                 |
NOT_SBY         NOP                       ; Else the CRNT_STATE is not defined, so  |
DISP_EXIT       RTS                       ; Exit from the state dispatcher ----------+
;*******************************************************************************************
START_ST        BRCLR PORTAD0,$04,NO_FWD        ; If "NOT" FWD_BUMP
            JSR   INIT_FWD              ; Initialize the FORWARD state and
            MOVB  #FWD,CRNT_STATE         ; Set CRNT_STATE to FWD
            BRA   START_EXIT            ; Then exit
                                ;
NO_FWD          NOP                       ; Else
START_EXIT      RTS                             ; return to the MAIN routine
;*******************************************************************************************
FWD_ST          PULD                      ;
            BRSET PORTAD0,$04,NO_FWD_BUMP   ; If FWD_BUMP then
            LDAA  SEC_PTH_INT           ; Correct NEXT_D value
            STAA  NEXT_D                ; ""
            JSR   INIT_REV              ; Initialize the REVERSE routine
            MOVB  #REV,CRNT_STATE         ; Set CRNT_STATE to REV
            JMP   FWD_EXIT              ; and return

NO_FWD_BUMP     BRSET PORTAD0,$08,NO_REV_BUMP   ; Else if REV_BUMP, then
            JMP   INIT_BK_TRK           ; Initialize the BACKTRACK state
            MOVB  #BK_TRK,CRNT_STATE        ; Set CRNT_STATE to BK_TRK
            JMP   FWD_EXIT              ; and return
```

```
NO_REV_BUMP     LDAA  D_DETN                ; Else if D_DETN equals 1 then
            BEQ   NO_RT_INTXN             ; The robot should make a RIGHT turn
            LDAA  NEXT_D                  ; Push direction for the previous
            PSHA                        ; Intersection to the stack
            LDAA  PRI_PTH_INT             ; Then store direction taken to NEXT_D
            STAA  NEXT_D                  ; ""
            JSR   INIT_RT_TRN             ; Initialize the RT_TRN state
            MOVB  #RT_TRN,CRNT_STATE      ; Set CRNT_STATE to RT_TRN
            JMP   FWD_EXIT                ; Then exit

NO_RT_INTXN     LDAA  B_DETN                ; Else if B_DETN equals 1
            BEQ   NO_LT_INTXN             ; Check if A_DETN equals 1
            LDAA  A_DETN                  ; If A_DETN equals 1 a FORWARD path exists
            BEQ   LT_TURN                 ; The robot should continue forward
            LDAA  NEXT_D                  ; Push direction for the previous
            PSHA                        ; Intersection to the stack
            LDAA  PRI_PTH_INT             ; Then store direction taken to NEXT_D
            STAA  NEXT_D                  ; ""
            BRA   NO_SHFT_LT              ; Else if A_DETN equals 0
LT_TURN       LDAA  NEXT_D                  ; Push direction for the previous
            PSHA                        ; Intersection to the stack
            LDAA  SEC_PTH_INT             ; Then store direction taken to NEXT_D
            STAA  NEXT_D                  ; ""
            JSR   INIT_LT_TRN             ; The robot should make a LEFT turn
            MOVB  #LT_TRN,CRNT_STATE      ; Initialize the LT_TRN state
            JMP   FWD_EXIT                ; Set CRNT_STATE to LT_TRN and exit

NO_LT_INTXN     LDAA  F_DETN                ; Else if F_DETN equals 1
            BEQ   NO_SHFT_RT              ; The robot should shift RIGHT
            JSR   PORTON                  ; and turn on the LEFT motor
RT_FWD_DIS      LDD   COUNT2                ;
            CPD   #INC_DIS                ;
            BLO   RT_FWD_DIS              ; If Dc>Dfwd then
            JSR   INIT_FWD                ; Turn motors off
            JMP   FWD_EXIT                ; and exit

NO_SHFT_RT      LDAA  E_DETN                ; Else if E_DETN equals 1
            BEQ   NO_SHFT_LT              ; The robot should shift LEFT
            JSR   STARON                  ; and turn on the RIGHT motor
LT_FWD_DIS      LDD   COUNT1                ;
            CPD   #INC_DIS                ;
            BLO   LT_FWD_DIS              ; If Dc>Dfwd then
```

```
            JSR   INIT_FWD              ; Turn motors off
            JMP   FWD_EXIT               ; and exit

NO_SHFT_LT     JSR   STARON                 ; Turn motors on
            JSR   PORTON                ; ""
FWD_STR_DIS     LDD   COUNT1                 ;
            CPD   #FWD_DIS               ;
            BLO   FWD_STR_DIS             ; If Dc>Dfwd then
            JSR   INIT_FWD              ; Turn motors off

FWD_EXIT       JMP   MAIN                   ; return to the MAIN routine
;*******************************************************************************************
REV_ST         LDD   COUNT1                 ; If Dc>Drev then
            CPD   #REV_DIS               ; The robot should make a U TURN
            BLO   REV_ST                ; so
            JSR   STARFWD                ; Set STBD Motor to FWD direction
            LDD   #0                   ; Reset timer
            STD   COUNT1                ; ""

REV_U_TRN      LDD   COUNT1                 ; If Dc>Dutrn then
            CPD   #UTRN_DIS              ; The robot should stop
            BLO   REV_U_TRN              ; so
            JSR   INIT_FWD              ; Initialize the FWD state
            LDAA  RETURN                 ; If RETURN equals 1
            BNE   BK_TRK_REV             ;
            MOVB  #FWD,CRNT_STATE          ; Then set state to FWD
            BRA   REV_EXIT              ; and exit
BK_TRK_REV     JSR   INIT_FWD               ;
            MOVB  #BK_TRK,CRNT_STATE       ; Else set CRNT_STATE to BK_TRK

REV_EXIT       RTS                        ; return to the MAIN routine
;*******************************************************************************************
RT_TRN_ST      LDD   COUNT2                 ; If Dc>Dfwd then
            CPD   #STR_DIS               ; The robot should make a TURN
            BLO   RT_TRN_ST              ; so
            JSR   STAROFF               ; Set STBD Motor to OFF
            LDD   #0                   ; Reset timer
            STD   COUNT2                ; ""

RT_TURN_LOOP    LDD   COUNT2                 ; If Dc>Dfwdturn then
            CPD   #TRN_DIS               ; The robot should stop
            BLO   RT_TURN_LOOP            ; so
            JSR   INIT_FWD              ; Initialize the FWD state
```

```
            LDAA  RETURN                ; If RETURN equals 1
            BNE   BK_TRK_RT_TRN         ;
            MOVB  #FWD,CRNT_STATE       ; Then set state to FWD
            BRA   RT_TRN_EXIT           ; and exit
BK_TRK_RT_TRN   MOVB  #BK_TRK,CRNT_STATE    ; Else set state to BK_TRK

RT_TRN_EXIT     RTS                     ; return to the MAIN routine
;*******************************************************************************************
LT_TRN_ST       LDD   COUNT1                ; If Dc>Dfwd then
            CPD   #STR_DIS              ; The robot should make a TURN
            BLO   LT_TRN_ST             ; so
            JSR   PORTOFF               ; Set PORT Motor to OFF
            LDD   #0                ; Reset timer
            STD   COUNT1                ; ""

LT_TURN_LOOP    LDD   COUNT1                ; If Dc>Dfwdturn then
            CPD   #TRN_DIS              ; The robot should stop
            BLO   LT_TURN_LOOP          ; so
            JSR   INIT_FWD              ; Initialize the FWD state
            LDAA  RETURN                ; If RETURN equals 1
            BNE   BK_TRK_LT_TRN         ;
            MOVB  #FWD,CRNT_STATE       ; Then set state to FWD
            BRA   LT_TRN_EXIT           ; and exit
BK_TRK_LT_TRN   MOVB  #BK_TRK,CRNT_STATE    ; Else set state to BK_TRK

LT_TRN_EXIT     RTS                     ; return to the MAIN routine
;*******************************************************************************************
BK_TRK_ST       PULD                    ;
            BRSET PORTAD0,$08,NO_BK_BUMP    ; If REV_BUMP, then we should stop
            JSR   INIT_SBY              ; Initialize the STANDBY state
            MOVB  #SBY,CRNT_STATE       ; set the state to SBY
            JMP   BK_TRK_EXIT           ; and exit

NO_BK_BUMP      LDAA  NEXT_D                ; If NEXT_D equals 0
            BEQ   REG_PATHING           ; Use regular pathing mode
            BNE   IRREG_PATHING         ; Else use irregular pathing mode
;-------------------------------------------------------------------------------------
REG_PATHING     LDAA  D_DETN                ; If D_DETN equals 1
            BEQ   NO_RT_TRN             ; The robot should make a RIGHT turn
            PULA                    ; Pull the next direction value from the stack
            PULA                    ; and store it in NEXT_D
            STAA  NEXT_D                ; ""
            JSR   INIT_RT_TRN           ; Initialize the RT_TRN state
```

```
            MOVB  #RT_TRN,CRNT_STATE       ; Set CRNT_STATE to RT_TRN
            JMP   BK_TRK_EXIT              ; Then exit

NO_RT_TRN       LDAA  B_DETN                ; If B_DETN equals 1
            BEQ   RT_LINE_S                ; Check if A_DETN equals 1
            LDAA  A_DETN                   ; If A_DETN equals 1 a FORWARD path exists
            BEQ   LEFT_TURN                ; The robot should continue forward
            PULA                        ; Pull the next direction value from the stack
            PULA                        ; and store it in NEXT_D
            STAA  NEXT_D                   ; ""
            BRA   NO_LINE_S                ; Else if A_DETN equals 0
LEFT_TURN       PULA                       ; The robot should make a LEFT turn
            PULA                        ; Pull the next direction value from the stack
            STAA  NEXT_D                   ; and store it in NEXT_D
            JSR   INIT_LT_TRN              ; Initialize the LT_TRN state
            MOVB  #LT_TRN,CRNT_STATE       ; Set CRNT_STATE to LT_TRN
            JMP   BK_TRK_EXIT              ; and exit
;-------------------------------------------------------------------------------------------
IRREG_PATHING   LDAA  B_DETN                ; If B_DETN equals 1
            BEQ   NO_LT_TRN                ; The robot should make a LEFT turn
            PULA                        ; Pull the next direction value from the stack
            STAA  NEXT_D                   ; and store it in NEXT_D
            JSR   INIT_LT_TRN              ; Initialize the LT_TRN state
            MOVB  #LT_TRN,CRNT_STATE       ; Set CRNT_STATE to LT_TRN
            JMP   BK_TRK_EXIT              ; and exit

NO_LT_TRN       LDAA  D_DETN                ; If D_DETN equals 1
            BEQ   RT_LINE_S                ; Check if A_DETN equals 1
            LDAA  A_DETN                   ; If A_DETN equals 1 a FORWARD path exists
            BEQ   RIGHT_TURN               ; The robot should continue forward
            PULA                        ; Pull the next direction value from the stack
            STAA  NEXT_D                   ; and store it in NEXT_D
            BRA   NO_LINE_S                ; Else if A_DETN equals 0
RIGHT_TURN      PULA                       ; The robot should make a RIGHT turn
            STAA  NEXT_D                   ; Pull the next direction value from the stack
            JSR   INIT_RT_TRN              ; Initialize the RT_TRN state
            MOVB  #RT_TRN,CRNT_STATE       ; Set CRNT_STATE to RT_TRN
            JMP   BK_TRK_EXIT              ; Then exit
;-------------------------------------------------------------------------------------------
RT_LINE_S       LDAA  F_DETN                ; Else if F_DETN equals 1
            BEQ   LT_LINE_S                ; The robot should shift RIGHT
            JSR   PORTON                   ; and turn on the LEFT motor
RT_FWD_D        LDD   COUNT2                ;
```

```
            CPD   #INC_DIS              ;
            BLO   RT_FWD_D                ; If Dc>Dfwd then
            JSR   INIT_FWD              ; Turn motors off
            JMP   BK_TRK_EXIT             ; and exit

LT_LINE_S   LDAA  E_DETN                  ; Else if F_DETN equals 1
            BEQ   NO_LINE_S             ; The robot should shift RIGHT
            JSR   STARON                ; and turn on the LEFT motor
LT_FWD_D    LDD   COUNT1                    ;
            CPD   #INC_DIS              ;
            BLO   LT_FWD_D                ; If Dc>Dfwd then
            JSR   INIT_FWD              ; Turn motors off
            JMP   BK_TRK_EXIT             ; and exit

NO_LINE_S   JSR   STARON                  ; Turn motors on
            JSR   PORTON                ; ""
FWD_STR_D   LDD   COUNT1                    ;
            CPD   #FWD_DIS                ;
            BLO   FWD_STR_D               ; If Dc>Dfwd then
            JSR   INIT_FWD              ; Turn motors off

BK_TRK_EXIT JMP   MAIN                    ; return to the MAIN routine
;*******************************************************************************************
SBY_ST      BRSET PORTAD0,$04,NO_START    ; If FWD_BUMP
            BCLR  PTT,%00110000           ; Initialize the START state
            MOVB  #START,CRNT_STATE       ; Set CRNT_STATE to START
            BRA   SBY_EXIT              ; Then exit
                              ;
NO_START    NOP                           ; Else
SBY_EXIT    RTS                           ; return to the MAIN routine

;*******************************************************************************************
;* STATE INITIALIZATION SECTION                                                            *
;*******************************************************************************************
INIT_FWD    BCLR  PTT,%00110000           ; Turn OFF the drive motors
            LDD   #0                    ; Reset timer
            STD   COUNT1                  ; ""
            STD   COUNT2                  ; ""
            BCLR  PORTA,%00000011         ; Set FWD direction for both motors
            RTS
;*******************************************************************************************
INIT_REV    BSET  PORTA,%00000011         ; Set REV direction for both motors
            LDD   #0                    ; Reset timer
```

```
            STD   COUNT1                ; ""
            BSET  PTT,%00110000          ; Turn ON the drive motors
            RTS
;*******************************************************************************************
INIT_RT_TRN    BCLR  PORTA,%00000011     ; Set FWD direction for both motors
            LDD   #0                    ; Reset timer
            STD   COUNT2                ; ""
            BSET  PTT,%00110000         ; Turn ON the drive motors
            RTS
;*******************************************************************************************
INIT_LT_TRN    BCLR  PORTA,%00000011     ; Set FWD direction for both motors
            LDD   #0                    ; Reset timer
            STD   COUNT1                ; ""
            BSET  PTT,%00110000         ; Turn ON the drive motors
            RTS
;*******************************************************************************************
INIT_BK_TRK    INC   RETURN             ; Change RETURN value to 1
            PULA                        ; Pull the next direction value from the stack
            STAA  NEXT_D                ; and store it in NEXT_D
            JSR   INIT_REV              ; Initialize the REVERSE routine
            JSR   REV_ST                ; Jump to REV_ST
            JMP   MAIN
;*******************************************************************************************
INIT_SBY       BCLR  PTT,%00110000      ; Turn off the drive motors
            RTS

;*******************************************************************************************
;* SENSOR SUBROUTINE SECTION                                                              *
;*******************************************************************************************
UPDT_READING    JSR   G_LEDS_ON         ; Turn ON LEDS
            JSR   READ_SENSORS          ; Take readings from sensors
            JSR   G_LEDS_OFF            ; Turn OFF LEDS

            LDAA  #0                    ; Set sensor A detection value to 0
            STAA  A_DETN                ; Sensor A
            STAA  B_DETN                ; Sensor B
            STAA  C_DETN                ; Sensor C
            STAA  D_DETN                ; Sensor D
            STAA  E_DETN                ; Sensor E
            STAA  F_DETN                ; Sensor F

CHECK_A        LDAA  SENSOR_BOW         ; If SENSOR_BOW is GREATER than
            CMPA  #PTH_A_INT            ; PTH_A_INT
```

```
            BLO   CHECK_B               ;
            INC   A_DETN                ; Set A_DETN to 1

CHECK_B     LDAA  SENSOR_PORT           ; If SENSOR_PORT is GREATER than
            CMPA  #PTH_B_INT            ; PTH_B_INT
            BLO   CHECK_C               ;
            INC   B_DETN                ; Set B_DETN to 1

CHECK_C     LDAA  SENSOR_MID            ; If SENSOR_MID is GREATER than
            CMPA  #PTH_C_INT            ; PTH_C_INT
            BLO   CHECK_D               ;
            INC   C_DETN                ; Set C_DETN to 1

CHECK_D     LDAA  SENSOR_STBD           ; If SENSOR_STBD is GREATER than
            CMPA  #PTH_D_INT            ; PTH_D_INT
            BLO   CHECK_E               ;
            INC   D_DETN                ; Set D_DETN to 1

CHECK_E     LDAA  SENSOR_LINE           ; If SENSOR_LINE is LESS than
            CMPA  #PTH_E_INT            ; PTH_E_INT
            BHI   CHECK_F               ;
            INC   E_DETN                ; Set E_DETN to 1

CHECK_F     LDAA  SENSOR_LINE           ; If SENSOR_LINE is GREATER than
            CMPA  #PTH_F_INT            ; PTH_F_INT
            BLO   UPDT_DONE             ;
            INC   F_DETN                ; Set F_DETN to 1

UPDT_DONE   RTS
;*******************************************************************************************
G_LEDS_ON   BSET  PORTA,%00100000       ; Set bit 5
            RTS
;*******************************************************************************************
G_LEDS_OFF  BCLR  PORTA,%00100000       ; Clear bit 5
            RTS
;*******************************************************************************************
READ_SENSORS  CLR   SENSOR_NUM          ; Select sensor number 0
            LDX   #SENSOR_LINE          ; Point at the start of the sensor array
 RS_MAIN_LOOP: LDAA  SENSOR_NUM         ; Select the correct sensor input
            JSR   SELECT_SENSOR         ; on the hardware
            LDY   #400                  ; 20 ms delay to allow the
            JSR   del_50us              ; sensor to stabilize
            LDAA  #%10000001            ; Start A/D conversion on AN1
```

```
            STAA  ATDCTL5
            BRCLR ATDSTAT0,$80,*          ; Repeat until A/D signals done
            LDAA  ATDDR0L                 ; A/D conversion is complete in ATDDR0L
            STAA  0,X                   ; so copy it to the sensor register
            CPX   #SENSOR_STBD            ; If this is the last reading
            BEQ   RS_EXIT                 ; Then exit
            INC   SENSOR_NUM              ; Else, increment the sensor number
            INX                         ; and the pointer into the sensor array
            BRA   RS_MAIN_LOOP            ; and do it again
      RS_EXIT: RTS
;*******************************************************************************************
SELECT_SENSOR   PSHA                        ; Save the sensor number for the moment
            LDAA  PORTA                   ; Clear the sensor selection bits to zeros
            ANDA  #%11100011
            STAA  TEMP                    ; and save it into TEMP
            PULA                        ; Get the sensor number
            ASLA                        ; Shift the selection number left, twice
            ASLA
            ANDA  #%00011100              ; Clear irrelevant bit positions
            ORAA  TEMP                    ; OR it into the sensor bit positions
            STAA  PORTA                   ; Update the hardware
            RTS

;*******************************************************************************************
;* UTILITY SUBROUTINE SECTION                                                  *
;*******************************************************************************************
del_50us:       PSHX                        ;2 E-clk Protect the X register
eloop:          LDX   #300                  ;2 E-clk Initialize the inner loop counter
iloop:          NOP                         ;1 E-clk No operation
            DBNE  X,iloop                 ;3 E-clk If the inner cntr not 0, loop again
            DBNE  Y,eloop                 ;3 E-clk If the outer cntr not 0, loop again
            PULX                        ;3 E-clk Restore the X register
            RTS                         ;5 E-clk Else return
;*******************************************************************************************
cmd2LCD:        BCLR  LCD_CNTR,LCD_RS         ; select the LCD Instruction Register (IR)
            JSR   dataMov                 ; send data to IR
              RTS
;*******************************************************************************************
putsLCD         LDAA  1,X+                    ; get one character from the string
            BEQ   donePS                  ; reach NULL character?
            JSR   putcLCD
            BRA   putsLCD
donePS              RTS
```

```
;*******************************************************************************************
putcLCD     BSET  LCD_CNTR,LCD_RS       ; select the LCD Data register (DR)
            JSR   dataMov               ; send data to DR
            RTS
;*******************************************************************************************
dataMov     BSET  LCD_CNTR,LCD_E        ; pull the LCD E-sigal high
            STAA  LCD_DAT               ; send the upper 4 bits of data to LCD
            BCLR  LCD_CNTR,LCD_E        ; pull the LCD E-signal low to complete the write
oper.
            LSLA                        ; match the lower 4 bits with the LCD data pins
            LSLA                        ; -"-
            LSLA                        ; -"-
            LSLA                        ; -"-
            BSET  LCD_CNTR,LCD_E        ; pull the LCD E signal high
            STAA  LCD_DAT               ; send the lower 4 bits of data to LCD
            BCLR  LCD_CNTR,LCD_E        ; pull the LCD E-signal low to complete the write
oper.
            LDY   #1                    ; adding this delay will complete the internal
            JSR   del_50us              ; operation for most instructions
            RTS
;*******************************************************************************************
int2BCD     XGDX                        ; Save the binary number into .X
            LDAA  #0                    ; Clear the BCD_BUFFER
            STAA  TEN_THOUS
            STAA  THOUSANDS
            STAA  HUNDREDS
            STAA  TENS
            STAA  UNITS
            STAA  BCD_SPARE
            STAA  BCD_SPARE+1

            CPX   #0                    ; Check for a zero input
            BEQ   CON_EXIT              ; and if so, exit

            XGDX                        ; Not zero, get the binary number back to .D as dividend
            LDX   #10                   ; Setup 10 (Decimal!) as the divisor
            IDIV                        ; Divide: Quotient is now in .X, remainder in .D
            STAB  UNITS                 ; Store remainder
            CPX   #0                    ; If quotient is zero,
            BEQ   CON_EXIT              ; then exit

            XGDX                        ; else swap first quotient back into .D
            LDX   #10                   ; and setup for another divide by 10
```

```
            IDIV
            STAB  TENS
            CPX   #0
            BEQ   CON_EXIT

            XGDX                    ; Swap quotient back into .D
            LDX   #10               ; and setup for another divide by 10
            IDIV
            STAB  HUNDREDS
            CPX   #0
            BEQ   CON_EXIT

            XGDX                    ; Swap quotient back into .D
            LDX   #10               ; and setup for another divide by 10
            IDIV
            STAB  THOUSANDS
            CPX   #0
            BEQ   CON_EXIT

            XGDX                    ; Swap quotient back into .D
            LDX   #10               ; and setup for another divide by 10
            IDIV
            STAB  TEN_THOUS

    CON_EXIT: RTS                   ; We're done the conversion
;***************************************************************************************
BCD2ASC     LDAA  #$0               ; Initialize the blanking flag
            STAA  NO_BLANK

     C_TTHOU: LDAA  TEN_THOUS       ; Check the 'ten_thousands' digit
            ORAA  NO_BLANK
            BNE   NOT_BLANK1

    ISBLANK1: LDAA  #$20            ; It's blank
            STAA  TEN_THOUS         ; so store a space
            BRA   C_THOU            ; and check the 'thousands' digit

   NOT_BLANK1: LDAA  TEN_THOUS      ; Get the 'ten_thousands' digit
            ORAA  #$30              ; Convert to ascii
            STAA  TEN_THOUS
            LDAA  #$1               ; Signal that we have seen a 'non-blank' digit
            STAA  NO_BLANK
```

```
  C_THOU: LDAA  THOUSANDS          ; Check the thousands digit for blankness
          ORAA  NO_BLANK           ; If it's blank and 'no-blank' is still zero
          BNE   NOT_BLANK2

 ISBLANK2: LDAA  #$30              ; Thousands digit is blank
          STAA  THOUSANDS          ; so store a space
          BRA   C_HUNS             ; and check the hundreds digit

NOT_BLANK2: LDAA  THOUSANDS        ; (similar to 'ten_thousands' case)
          ORAA  #$30
          STAA  THOUSANDS
          LDAA  #$1
          STAA  NO_BLANK

  C_HUNS: LDAA  HUNDREDS           ; Check the hundreds digit for blankness
          ORAA  NO_BLANK           ; If it's blank and 'no-blank' is still zero
          BNE   NOT_BLANK3

 ISBLANK3: LDAA  #$20              ; Hundreds digit is blank
          STAA  HUNDREDS           ; so store a space
          BRA   C_TENS             ; and check the tens digit

NOT_BLANK3: LDAA  HUNDREDS         ; (similar to 'ten_thousands' case)
          ORAA  #$30
          STAA  HUNDREDS
          LDAA  #$1
          STAA  NO_BLANK

  C_TENS: LDAA  TENS               ; Check the tens digit for blankness
          ORAA  NO_BLANK           ; If it's blank and 'no-blank' is still zero
          BNE   NOT_BLANK4

 ISBLANK4: LDAA  #$20              ; Tens digit is blank
          STAA  TENS               ; so store a space
          BRA   C_UNITS            ; and check the units digit

NOT_BLANK4: LDAA  TENS             ; (similar to 'ten_thousands' case)
          ORAA  #$30
          STAA  TENS

  C_UNITS: LDAA  UNITS             ; No blank check necessary, convert to ascii.
          ORAA  #$30
          STAA  UNITS
```

```
            RTS                         ; We're done
;*******************************************************************************************
HEX_TABLE       FCC '0123456789ABCDEF'          ; Table for converting values
BIN2ASC         PSHA                        ; Save a copy of the input number on the stack
            TAB                         ; and copy it into ACCB
            ANDB #%00001111                 ; Strip off the upper nibble of ACCB
            CLRA                        ; D now contains 000n where n is the LSnibble
            ADDD #HEX_TABLE                 ; Set up for indexed load
            XGDX
            LDAA 0,X                    ; Get the LSnibble character
            PULB                        ; Retrieve the input number into ACCB
            PSHA                        ; and push the LSnibble character in its place
            RORB                        ; Move the upper nibble of the input number
            RORB                        ;  into the lower nibble position.
            RORB
            RORB
            ANDB #%00001111                 ; Strip off the upper nibble
            CLRA                        ; D now contains 000n where n is the MSnibble
            ADDD #HEX_TABLE                 ; Set up for indexed load
            XGDX
            LDAA 0,X                    ; Get the MSnibble character into ACCA
            PULB                        ; Retrieve the LSnibble character into ACCB
            RTS
;*******************************************************************************************
;* Update Display (Current State + Bumper Switches + Battery Voltage + Sensor Readings)    *
;*******************************************************************************************
UPDT_DISPL      LDAA  #$82                  ; Move LCD cursor to the end of msg1
            JSR   cmd2LCD               ;

            LDAB  CRNT_STATE                ; Display current state
            LSLB                        ; "
            LSLB                        ; "
            LSLB                        ; "
            LDX   #tab                  ; "
            ABX                         ; "
            JSR   putsLCD                   ; "
;---------------------------------------------
            LDAA  #$8F                  ; Move LCD cursor to the end of msg2
            JSR   cmd2LCD                   ; ""
            LDAA  SENSOR_BOW                ; Convert value from SENSOR_BOW to a
            JSR   BIN2ASC                   ; Two digit hexidecimal value
            JSR   putcLCD                   ; ""
```

```
            EXG   A,B                 ; ""
            JSR   putcLCD               ; ""

            LDAA  #$92                ; Move LCD cursor to Line position
            JSR   cmd2LCD               ; ""
            LDAA  SENSOR_LINE             ; Convert value from SENSOR_BOW to a
            JSR   BIN2ASC               ; Two digit hexidecimal value
            JSR   putcLCD               ; ""
            EXG   A,B                 ; ""
            JSR   putcLCD               ; ""

            LDAA  #$CC                ; Move LCD cursor to Port position on 2nd row
            JSR   cmd2LCD               ; ""
            LDAA  SENSOR_PORT             ; Convert value from SENSOR_BOW to a
            JSR   BIN2ASC               ; Two digit hexidecimal value
            JSR   putcLCD               ; ""
            EXG   A,B                 ; ""
            JSR   putcLCD               ; ""

            LDAA  #$CF                ; Move LCD cursor to Mid position on 2nd row
            JSR   cmd2LCD               ; ""
            LDAA  SENSOR_MID              ; Convert value from SENSOR_BOW to a
            JSR   BIN2ASC               ; Two digit hexidecimal value
            JSR   putcLCD               ; ""
            EXG   A,B                 ; ""
            JSR   putcLCD               ; ""

            LDAA  #$D2                ; Move LCD cursor to Starboard position on 2nd row
            JSR   cmd2LCD               ; ""
            LDAA  SENSOR_STBD             ; Convert value from SENSOR_BOW to a
            JSR   BIN2ASC               ; Two digit hexidecimal value
            JSR   putcLCD               ; ""
            EXG   A,B                 ; ""
            JSR   putcLCD               ; ""
;---------------------------------------------
            MOVB  #$90,ATDCTL5            ; R-just., uns., sing. conv., mult., ch=0, start
            BRCLR ATDSTAT0,$80,*          ; Wait until the conver. seq. is complete
            LDAA  ATDDR0L               ; Load the ch0 result - battery volt - into A
            LDAB  #39                 ; AccB = 39
            MUL                     ; AccD = 1st result x 39
            ADDD  #600                ; AccD = 1st result x 39 + 600
            JSR   int2BCD
            JSR   BCD2ASC
```

```
            LDAA  #$C2                  ; move LCD cursor to the end of msg3
            JSR   cmd2LCD                 ; "
            LDAA  TEN_THOUS                 ; output the TEN_THOUS ASCII character
            JSR   putcLCD                 ; "
            LDAA  THOUSANDS                  ; output the THOUSANDS ASCII character
            JSR   putcLCD                 ; "
            LDAA  #$2E                  ; output the HUNDREDS ASCII character
            JSR   putcLCD                 ; "
            LDAA  HUNDREDS                  ; output the HUNDREDS ASCII character
            JSR   putcLCD                 ; "
;---------------------------------------------
            LDAA  #$C9                  ; Move LCD cursor to the end of msg4
            JSR   cmd2LCD

            BRCLR PORTAD0,#%00000100,bowON  ; If FWD_BUMP, then
            LDAA  #$20                  ;
            JSR   putcLCD                 ;
            BRA   stern_bump              ; Display 'B' on LCD
       bowON: LDAA  #$42                    ; ""
            JSR   putcLCD                 ; ""

   stern_bump: BRCLR PORTAD0,#%00001000,sternON; If REV_BUMP, then
            LDAA  #$20                  ;
            JSR   putcLCD                 ;
            BRA   UPDT_DISPL_EXIT           ; Display 'S' on LCD
     sternON: LDAA  #$53                    ; ""
            JSR   putcLCD                 ; ""
UPDT_DISPL_EXIT RTS                         ; and exit

;*******************************************************************************************
;* INITIALIZATION SUBROUTINE SECTION                                                       *
;*******************************************************************************************
initPORTS       BCLR  DDRAD,$FF               ; Set PORTAD as input
            BSET  DDRA, $FF               ; Set PORTA as output
            BSET  DDRT, $30               ; Set channels 4 & 5 of PORTT as output
            RTS
;*******************************************************************************************
initAD          MOVB  #$C0,ATDCTL2              ; power up AD, select fast flag clear
            JSR   del_50us                ; wait for 50 us
            MOVB  #$00,ATDCTL3              ; 8 conversions in a sequence
            MOVB  #$85,ATDCTL4              ; res=8, conv-clks=2, prescal=12
            BSET  ATDDIEN,$0C             ; configure pins AN03,AN02 as digital inputs
            RTS
```

```
;*******************************************************************************************
initLCD     BSET  DDRB,%11111111          ; configure pins PB7,...,PB0 for output
            BSET  DDRJ,%11000000          ; configure pins PJ7(E), PJ6(RS) for output
            LDY   #2000                   ; wait for LCD to be ready
            JSR   del_50us                ; -"-
            LDAA  #$28                    ; set 4-bit data, 2-line display
            JSR   cmd2LCD                 ; -"-
            LDAA  #$0C                    ; display on, cursor off, blinking off
            JSR   cmd2LCD                 ; -"-
            LDAA  #$06                    ; move cursor right after entering a character
            JSR   cmd2LCD                 ; -"-
            RTS
;*******************************************************************************************
clrLCD      LDAA  #$01                    ; clear cursor and return to home position
            JSR   cmd2LCD                 ; -"-
            LDY   #40                     ; wait until "clear cursor" command is complete
            JSR   del_50us                ; -"-
            RTS
;*******************************************************************************************
initTCNT    MOVB  #$80,TSCR1              ; enable TCNT
            MOVB  #$00,TSCR2              ; disable TCNT OVF interrupt, set prescaler to 1
            MOVB  #$FC,TIOS               ; channels PT1/IC1,PT0/IC0 are input captures
            MOVB  #$05,TCTL4              ; capture on rising edges of IC1,IC0 signals
            MOVB  #$03,TFLG1              ; clear the C1F,C0F input capture flags
            MOVB  #$03,TIE                ; enable interrupts for channels IC1,IC0
            RTS

;*******************************************************************************************
;* INTERRUPT SERVICE ROUTINE 1                                                             *
;*******************************************************************************************
ISR1        MOVB  #$01,TFLG1              ; clear the C0F input capture flag
            INC   COUNT1                  ; increment COUNT1
            RTI
;*******************************************************************************************
;* INTERRUPT SERVICE ROUTINE 2                                                             *
;*******************************************************************************************
ISR2        MOVB  #$02,TFLG1              ; clear the C1F input capture flag
            INC   COUNT2                  ; increment COUNT2
            RTI
;*******************************************************************************************
;* Interrupt Vectors                                                                       *
;*******************************************************************************************
            ORG   $FFFE
```

```
DC.W  Entry                ; Reset Vector

ORG   $FFEE
DC.W  ISR1                 ; COUNT1 INT

ORG   $FFEC
DC.W  ISR2                 ; COUNT2 INT
```