HCS12 Microcontroller Maze Solver for Robot Using Assembly

Authors: Abdullah Syed
Akash Kanagarajah
Zhu Feng Huang
Mosab Saleh
Eduardo Ibarra

Equates Section

```
***********************************************************************
; equates section
***********************************************************************
LCD_DAT          EQU PORTB
LCD_CNTR         EQU PTJ
LCD_E            EQU $80
LCD_RS           EQU $40
FWD_INT          EQU 69 ; 3 second delay (at 23Hz)
REV_INT          EQU 69 ; 3 second delay (at 23Hz)
FWD_TRN_INT      EQU 46 ; 2 second delay (at 23Hz)
REV_TRN_INT      EQU 46 ; 2 second delay (at 23Hz)
14
START            EQU 0
FWD              EQU 1
REV              EQU 2
ALL_STP          EQU 3
FWD_TRN          EQU 4
REV_TRN          EQU 5
; variable section
```

In the equates section labels are equated with a value when they are encountered in code. For example, the start, fwd, rev, all_stp, fw_trn, and rev_trn all equate to some decimal value. This means that when they are encountered in the code, at that instance, the variable will be replaced by the decimal value. However, if a memory location (indicated by $) equates to a variable, then when the variable is encountered to the code, at that instance the variable is equated to the memory location. If the variable is equated to another label, then the variable will point to the label. In this section, the FWD_INT and REV_INT labels are equated to 69 because that will give us a three second delay after initiating the subroutines.

Variables Section

```
; variable section
*******************************************************************
                ORG $3850 ; Where our TOF counter register lives
TOF_COUNTER     dc.b 0 ; The timer, incremented at 23Hz
CRNT_STATE      dc.b 3 ; Current state register
T_FWD           ds.b 1 ; FWD time
T_REV           ds.b 1 ; REV time
T_FWD_TRN       ds.b 1 ; FWD_TURN time
T_REV_TRN       ds.b 1 ; REV_TURN time
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
NO_BLANK        ds.b 1 ; Used in 'leading zero' blanking by BCD2ASC
```

In the variables section of the code the variables in the left hand column are either defined as dc.b, ds.b, RMB. In the very first line the ORG is followed by a memory location indicated by the dollar sign to indicate where in the memory the variables will be stored. When a variable is defined as a “ds.b”, that means a certain number of bytes is reserved for the variable. For example, if a variable was defined as TENS ds.b 1, then 1 byte would be reserved for the

TENS variable. If a variable is defined with “dc.b”, that means a constant byte is defined for the variable. For instance, dc.b 0 would define a constant byte at a certain memory location with the value of 0 at the location. Finally, RMB means reserve memory byte and in the variable section 10 bytes are reserved for the decimal point and string terminator.

Code Section

```
                ORG $4000 ; Where the code starts --------------------
Entry: ; |
_Startup: ; |
                CLI ; Enable interrupts |
                LDS #$4000 ; Initialize the stack pointer

                BSET DDRA,%00000011 ; STAR_DIR, PORT_DIR N
                BSET DDRT,%00110000 ; STAR_SPEED, PORT_SPEED I

                JSR initAD ; Initialize ATD converter I

                JSR initLCD ; Initialize the LCD L
                JSR clrLCD ; Clear LCD & home cursor I

                LDX #msg1 ; Display msg1 A
                JSR putsLCD ; " T

                LDAA #$C0 ; Move LCD cursor to the 2nd row O
                JSR cmd2LCD ; N
                LDX #msg2 ; Display msg2 |
                JSR putsLCD ; " |

                JSR ENABLE_TOF ; Jump to TOF initialization ---------------

MAIN            JSR UPDT_DISPL ; ----------------------------------------- M
                LDAA CRNT_STATE ; A
                JSR DISPATCHER ; I
                BRA MAIN ; ----------------------------------------- N
; data section
*******************************************************************
msg1            dc.b "Battery volt ",0
msg2            dc.b "State ",0
tab             dc.b "START ",0
                dc.b "FWD ",0
                dc.b "REV ",0
                dc.b "ALL_STP",0
                dc.b "FWD_TRN",0
                dc.b "REV_TRN",0
```

The main program continuously loops, updating the current state of the robot this way it detects if and when the robot has collided, and through the appropriate subroutine judging by the machine's current state, it changes the state to reverse, forward turn, reverse turn, etc.
The TOF_COUNTER increments at timely intervals, letting the robot know when to update its state status allowing for the execution of accurate turning maneuvers; the TOF_Counter is initialized before jumping into the main loop.

Subroutine Section

```
; subroutine section
*******************************************************************
DISPATCHER      CMPA #START ; If it's the START state ---------------
                BNE NOT_START ; |
                JSR START_ST ; then call START_ST routine D
                BRA DISP_EXIT ; and exit I

NOT_START       CMPA #FORWARD
                BNE NOT_FORWARD
                JSR FORWARD_STATE
                JMP DOSP_EXIT


NOT_FWD_TRN     CMPA #REV_TRN ; Else if it's the REV_TRN state C
                BNE NOT_REV_TRN ; H
                JSR REV_TRN_ST ; then call REV_TRN_ST routine E
                BRA DISP_EXIT ; and exit R

NOT_REV_TRN     SWI ; Else the CRNT_ST is not defined, so stop |
DISP_EXIT       RTS ; Exit from the state dispatcher ----------
*******************************************************************
START_ST        BRCLR PORTADO, $04, NO_FORWARD
                JSR INIT_FWD
                MOVB $FORWARD, CURRENT_STATE
                BRA START_EXIT

NO_FWD          NOP ; Else
START_EXIT      RTS ; return to the MAIN routine
*******************************************************************
FWD_ST          LDAA TOF_COUNTER
                CMPA T_FWD
                BNE NO_FWD
                JSR INIT_FWD_TRN
                MOVB #FWD_TRN, CRNT_STATE
                BRA FWD_EXIT

NO_FWD_TRN      NOP ; Else
FWD_EXIT        RTS ; return to the MAIN routine
*******************************************************************
REV_ST          LDAA TOF_COUNTER ; If Tc>Trev then
                CMPA T_REV ; the robot should make a FWD turn
                BNE NO_REV_TRN ; so
                JSR INIT_REV_TRN ; initialize the REV_TRN state
                MOVB #REV_TRN,CRNT_STATE ; set state to REV_TRN
                BRA REV_EXIT ; and return
NO_REV_TRN      NOP ; Else
REV_EXIT        RTS ; return to the MAIN routine
*******************************************************************
```

```
------------------------------------------------------------------------------------------------
INIT_FWD        BCLR PORTA,%00000011 ; Set FWD direction for both motors
                BSET PTT,%00110000 ; Turn on the drive motors
                LDAA TOF_COUNTER ; Mark the fwd time Tfwd
                ADDA #FWD_INT
                STAA T_FWD
                RTS
*******************************************************************
INIT_REV        BSET PORTA,%00000011 ; Set REV direction for both motors
                BSET PTT,%00110000 ; Turn on the drive motors
                LDAA TOF_COUNTER ; Mark the fwd time Tfwd
                ADDA #REV_INT
                STAA T_REV
                RTS
*******************************************************************
INIT_ALL_STP    BCLR PTT,%00110000 ; Turn off the drive motors
                RTS
*******************************************************************
INIT_FWD_TRN    BSET PORTA,%00000010 ; Set REV dir. for STARBOARD (right) motor
                LDAA TOF_COUNTER ; Mark the fwd_turn time Tfwdturn
                ADDA #FWD_TRN_INT
                STAA T_FWD_TRN
                RTS
*******************************************************************
INIT_REV_TRN    BCLR PORTA,%00000010 ; Set FWD dir. for STARBOARD (right) motor
                LDAA TOF_COUNTER ; Mark the fwd time Tfwd
                ADDA #REV_TRN_INT
                STAA T_REV_TRN
                RTS
; utility subroutines
*******************************************************************
initLCD BSET DDRS,%11110000 ; configure pins PS7,PS6,PS5,PS4 for output
        BSET DDRE,%10010000 ; configure pins PE7,PE4 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
*******************************************************************

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

```
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 0d          ;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 #' '       ;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 #' '         ;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 #' '        ;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 #' '      ;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
```

```
*******************************************************************
ENABLE_TOF LDD #TOF_ISR ; Setup the interrupt vector for timer overflow
           STD $FFDE
           LDAA #%10000000
           STAA TSCR1 ; Enable TCNT by setting bit 7
* When enabling the timer overflow interrupt, it is prudent to clear
* the TOF flag so than an interrupt does not occur immediately, but
* rather on the next timer overflow.
           STAA TFLG2 ; Clear the TOF flag by writing to bit 7
           LDAA #%10000100 ; Turn timer overflow interrupt on by setting bit 7
           STAA TSCR2 ; in TSCR2 and select prescale factor equal to 16
           RTS
*******************************************************************
TOF_ISR    INC TOF_COUNTER ; Increment the overflow count
           LDAA #%10000000 ; Clear the TOF flag
           STAA TFLG2 ; --"--
           RTI
*******************************************************************
* Update Display (Battery Voltage + Current State) *
*******************************************************************
UPDT_DISPL      MOVB #$90,ATDCTL5 ; R-just., uns., sing. conv., mult., ch=0, start
                BRCLR ATDSTAT0,$80,* ; Wait until the conver. seq. is complete
                LDAA ATDDR0L ; Load the ch0 result - battery volt - into A
... ; Display the battery voltage
;-------------------------
                LDAA #$C6 ; Move LCD cursor to the 2nd row, end of msg2
                JSR cmd2LCD ;
                LDAB CRNT_STATE ; Display current state
                LSLB ; "
                LSLB ; "
                LSLB ; "
                LDX #tab ; "
                ABX ; "
                JSR putsLCD ; "
                RTS
```

The subroutine section puts together the functionality of the robot, such as the implementation of the robot's state machine in code. A state machine is a flow diagram that explains through use of arrows, how the robot will react under certain conditions. For example in the implementation, if the robot were to hit something in front of it while in FWD_ST, it would jump to subroutine (JSR) INIT_REV where the robot would reverse. In the INIT_REV subroutine, the robot will be programmed to reverse until the counter timer (Tc) is greater than the given time to reverse (Tc). The robot will also check to make sure it is being reversed for the correct time. Next, the robot will check the counter to see if the counter is greater than the reverse time (Tc>Trev). If so, then it will jump to the INIT_REV_TRN subroutine and a reverse turn will be initiated for a certain amount of time. Finally, once the reverse turn is completed it will return to the forward state.

The robot is programmed similarly to handle the other conditions it encounters such as if there was a reverse bump. The robot is also programmed to automatically turn every few seconds, provided there are no forward or rear bumps.

In addition, the dispatcher is a high-level overview of the robot's state machine: It is separate from the subroutine above, because it commands the robot which subroutines to jump to when it is in a certain state. For example, when it is in the forward state, the robot will constantly check if it is in the forward state, and if so then it will execute the forward state subroutine. The previous explanations above explain what happens in code when the dispatcher dispatches the robot to jump one of the subroutines.

The last part of the code consists of the INIT subroutines. These subroutines tell the motors to move forward, reverse, and stop. To move forward, PORT A is cleared by using %00000011 and bits at PORT T are set with %00110000 to turn on the motors. To reverse, PORT A is set with %00000011 and bits at PORT T are set with %00110000. Notice that to reverse, both PORT A and PORT T are set with the same bits as in the INIT_FWD subroutine. To initiate an all-stop, the motors are turned off by clearing the bits at PORT T with %00110000. Notice to turn on the motors we use BSET PTT %00110000 and to turn off the motors we use BCLR PTT %00110000.