Delivery Plane Final Project

Created by: Kwesi Kyei-Fordjour, Garrett Thompson

Fall 2023

Table of Contents:

List of Figures:

Introduction:

Purpose:

The main objective of this project was to have a comprehensive task that involves all aspects of the Fall 2023 ECE 362 lab. This project tested our knowledge of each individual concept and also tested how we are able to combine these concepts such that they all function in harmony. This project was meant to simulate the functions of a delivery airplane that would travel to different destinations, play a radio, have variable motor control, rudder control, instrument display panel, panic system, refueling, and indicator lights. The program was created using the Freescale CodeWarrior IDE along side a predefined base file that included .C files that initialized the on board LCD, read the potentiometer, and send and play notes on the piezo speaker. The hardware includes the Motorola 68HC12 microcontroller and a peripheral board that will be described in greater detail later in the project description.

Assumptions:

This project contained a few minor assumptions. The first, and the most impactful, was the RTI period. This was set to a period of 1.024 ms to simplify programs timing. This in turn

would effect pulse width modulation driven components such as the DC motor and the piezo speaker radio. The next assumption, was the predefined maximum speed of the plane. This was defined in the project assignment to be 50 mph even though planes typically fly at speeds around 500 mph. Similarly, another assumption was the total capacity of the fuel tank. This was arbitrarily set to be approximately 70 miles as this was the distance of the longest delivery route. An assumption that accompanied the previous was that the fuel would decrease at a rate of roughly 1% of the tank capacity per second of engine run time. Finally, the last assumption was that the fuel consumption was constant regardless of engine speed.

Design:

Peripherals:

Figure 1: This image displays the Motorola 68HC12 microcontroller along with the peripheral board designed by IUPUI ECE department (2009)

The board displayed above, in figure 1, contains the Motorola 68HC12 microcontroller which is what handles the logic of the project. The microcontroller has expanded wide and

narrow mode capabilities to access memory efficiently. It as two 8-bit registers A and B, which can concatinate to a 16-bit register that is referred to as register D. It also contains two 16-bit address registers X and Y, as well as 16-bit registers for the program counter and stack pointer. Finally to control logic, it contains an 8-bit register denoted as the CCR. All peripherals will be described in figures below.

Figure 2: This image displays the onboard LCD screen.

The LCD panel is responsible for displaying the plane data such as the plane destination number, radio station, speed, fuel level, password entry screen, success and fail messages, refueling animation, and panic mode messages.

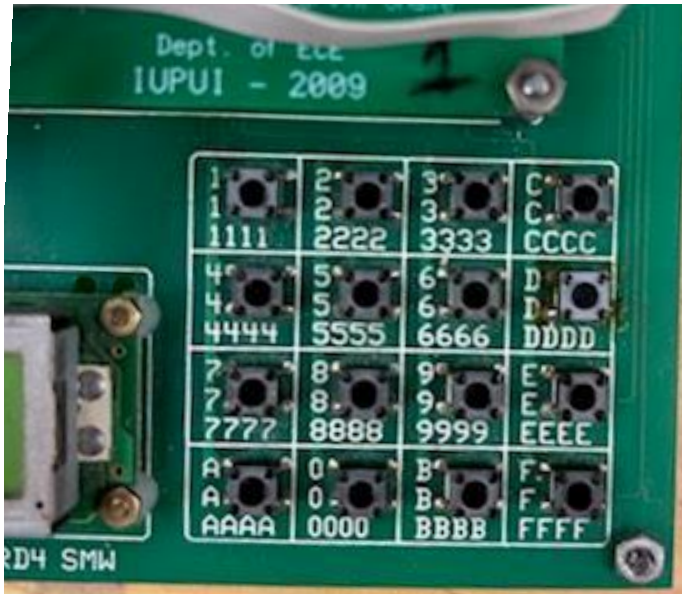

Figure 3: This image displays the onboard hexidecimal keypad, labeled with numbers 1-9 and A-F.

The hex keypad is used to take user input to the program. During start-up and panic mode, the keypad is used to enter a password to access the plane controls. Once the password has been entered, buttons labeled 4 and 5 are used for rudder control, and buttons 1 and 2 control radio station up and down. Additional features include buttons A and B for clearing plane checklist messages.

Figure 4: This image displays the onboard piezo speaker (large black circle in top left corner), potentiometer (white circle in center of photo labeled with letters "POT"), switch board (red rectangle in bottom left corner), eight LED indicator lights (above switch board), designated push button (small grey square with black circle to the right of the switch board), and IRQ push button (second push button to the right of the switch board).

The piezo speaker was utilized in this project to play different tones in various sequences to create songs for the radio, as well as alarm tones during the panic sequence. The volume of the speaker is manipulated via the dial next to the speaker labeled "AMP".

The potentiometer was incorporated into the program to modulate the DC motor speed as well as obtaining a speed value that is displayed on the information screen on the LCD.

The switch board was responsible for enabling the DC motor engine as well as toggling between information screens on the LCD. Engine enable was controlled via switch one's state, and the LCD information screens were toggled via switch 8.

The LED array had three functions, wing tip safety lights, turning indicators, and panic mode strobe sequence. The wing tip safety lights would continuously flash the leftmost and rightmost LEDs during flight. During the turning sequence depending on the direction (left or right) the corresponding four LEDs would strobe from center LED to the outer LED (again left or right). Finally, during panic sequence all LEDs strobe simultaneously until the password is

entered. The push button was used to initiate the refuel sequence while the engine is stopped. Similarly, the IRQ push button would trigger the panic mode.

Figure 5: This image displays the on board stepper motor (grey oval with gear in the center located at the bottom left corner) and DC motor (grey cylinder located above the stepper motor).

The stepper motor was used to represent the steering of the plane's rudder. This would only be active once a direction was selected (either left or right) at a stop. The rudder sequence would turn the stepper motor 90 degrees in the opposite direction of the selected direction just as a plane's rudder would function.

During the flight sequences, the DC motor was used to represent the plane's engine speed. This was controlled via pulse width modulation from the value read in from the potentiometer.

Software Implementation:

Discussion

All of the software implementation is described per the flowcharts displayed below. Overall, most of the major functions follow a variation of previous lab assignments with minor adjustments to allow for flags controlling the state of active functions. All variables are declared and initialized in the main. The RTI controls the timing of these state events via counters and flags. All user inputs are controlled via the hex keypad, the refuel pushbutton, IRQ button, and potentiometers. The user interface is exclusively portrayed using the LCD screen.

Flow Charts

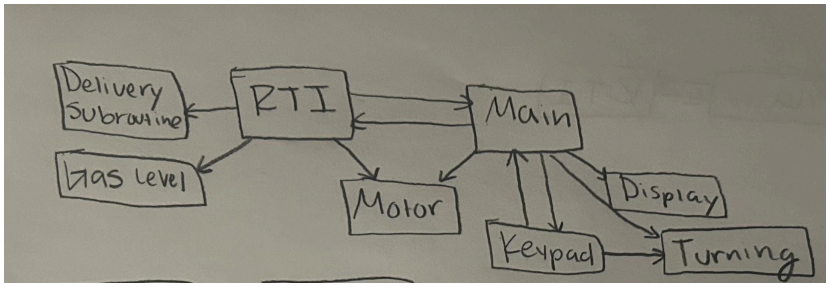

Figure 6: Main flow chart

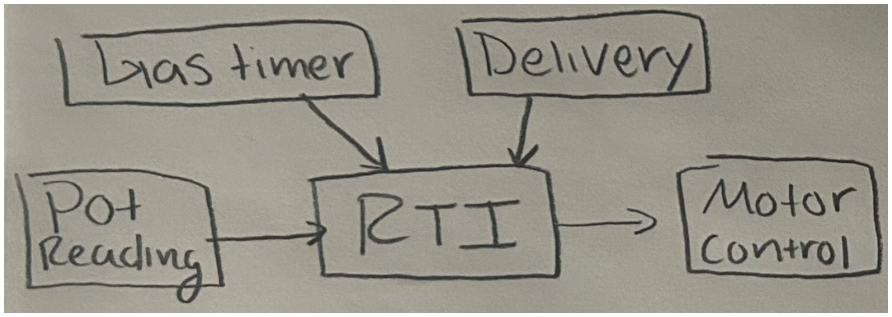

Figure 7: Motor control flow chart

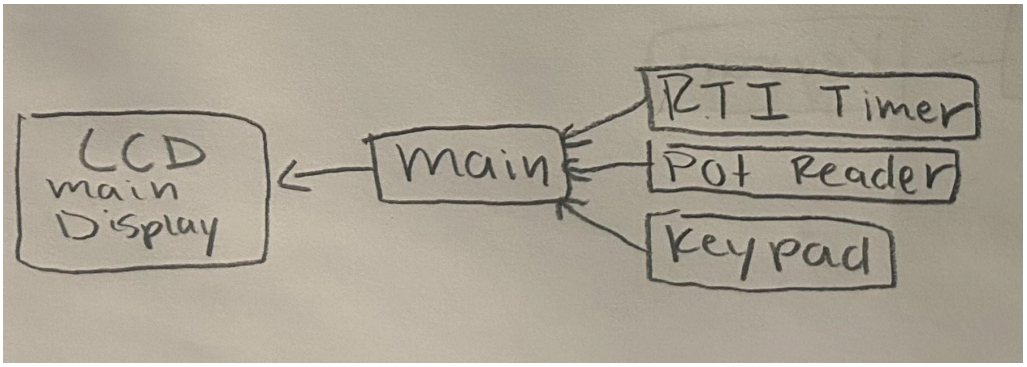

Figure 8: LCD flow chart

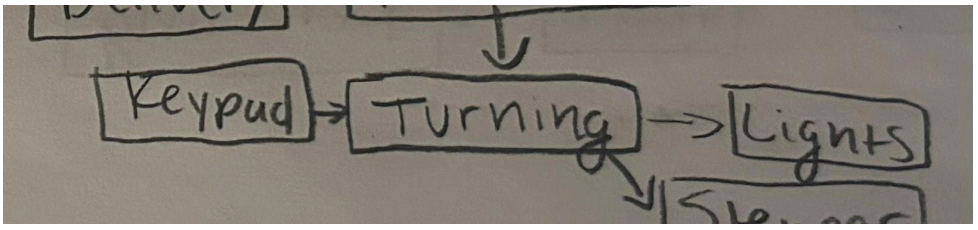

Figure 9: Turning flow chart

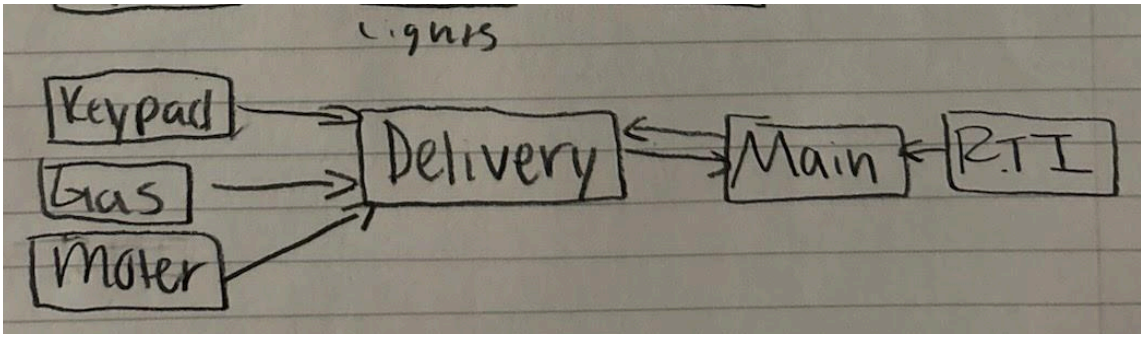

Figure 10: Delivery flow chart

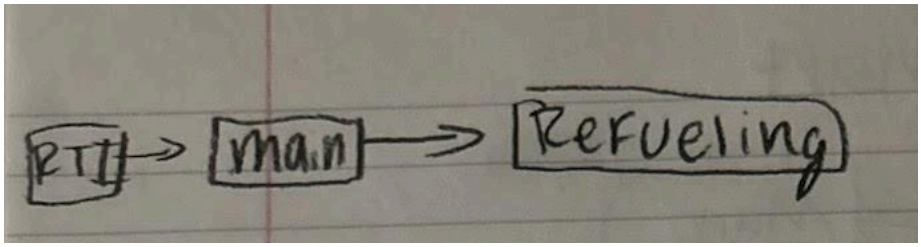

Figure 11: Refueling flow chart

Error Handling and Fail Safes

Errors were handled in the form of messages displayed on the LCD screen. This would come in the form of a failure message when an incorrect turn was inputted when the plane was considered to be on the runway. A failure message is also displayed if the plane runs out of fuel. A message would also displayed on the LCD to inform the user if an incorrect password was entered on the keypad during startup or during the panic sequence.

Failsafes were implemented as per the project requirements. These include engine enable switch (controlled by switch 1), password clearance (as referenced above), halting engine function during turning procedures, and take off checklist implemented as an additional features.

Design Changes:

The vast majority of the project directly followed the project requirements. The only change that was implemented that mildly deviated from the project requirements was simply that the fuel level was displayed as a percentage to make the display within the character limit on the information screen.

Project Additions:

- Password security: Keys are entered in during the password sequence and are read into the software. On the LCD display the password is hidden by displaying a “*” symbol.
- After the password is entered the program runs a preflight checklist on the LCD. The “A” key must be pressed to confirm that the potentiometer is zeroed.

- The second checklist item prompts the user via the LCD screen to press the “B” key to ensure the rear view mirrors have been checked.
- And additional message is displayed on the LCD screen after the second turn in the delivery sequence to inform the user that the delivery is over halfway completed.
- The delivery destination is randomly determined via any key press from the hex keypad.

Division of Labor:

Due to scheduling conflicts and multiple projects from other classes Garrett was limited to writing the code associated with the piezo speaker and song selection. This consisted of a small portion of the RTI as well as the two .asm files named “PlaySong” and “SongSelect”. Kwesi was able to complete all other sections of the code. To compensate for the imbalance of work Garrett was responsible for writing the majority of this report.

Functionality:

As far as project functionality goes, every system worked as required with the exception of the piezo speaker. When creating the code each person worked on separate project files, and the issues arose when attempting to combine the code handling the speaker with the rest of the code that Kwesi had written. Prior to adding the speaker code to the rest of the program, the speaker functioned as intended with flags that would trigger the radio station to increase or decrease and would sound the alarm when the IRQ was executed. The theory for why it failed when the code was merged together, was that Kwesi’s code took too long to execute before it was able to change notes at the correct timing and also slowed down the RTI enough to where the tone of the notes were shifted. Both programs used the same RTI period.

Conclusion:

In conclusion, this project challenged our abilities to program in assembly quickly and efficiently. It was able to bring all of the concepts learned throughout the semester into one cohesive task that ranged from basic conditional logic to complex tasks like interfacing with .C files and stack manipulation. This project greatly improved each of our abilities to find errors and debug complex systems.

Future Improvements:

The first aspect of the project that would need to be improved would be the piezo speaker code. There would need to be a considerable amount of debugging to determine the source of the timing error.

Appendix I: User Manual

Start-up:

1. Upon starting the system, the LCD display will prompt the user to enter a 4-digit password.
2. Clear checklist by pressing prompted button on the hex keypad
3. Press any key on the hex keypad to generate a random destination
4. Flip switch one to the top position to enable the engine power

Navigation:

1. Turn the potentiometer all the way clockwise to bring the engine to full power
2. Once the first stop has been reached press “4” on the hex keypad to turn left and “5” to turn right according to the delivery map for the displayed destination
3. At each turn prompt it is recommended to press the push button to refuel the plane to avoid running out of fuel and failing the delivery
4. When the correct path has been taken, a message will display on the LCD screen indicating a successful delivery and will restart the delivery process from the home airport prompting the user to repeat the process for a new random destination

Radio:

1. While in the flight state, the radio will begin to play a song, if the user wishes to change songs press “1” on the hex keypad to increase the radio station and “2” to decrease the radio station.
2. If the program is at the maximum radio station number and station increase is inputted the radio station will reset to 1; and if the radio is at 1 and the station down button is pressed, the station number will transition to the last radio station value.

Information Panel:

1. Toggle switch 8 to change between information screens on the LCD

Panic Mode:

1. Press the IRQ button to initiate panic mode, the LEDs will strobe and the radio will play a two tone alarm
2. Enter the password to reset the program and return to making deliveries

Appendix II: Code

Main.asm:

```
            INCLUDE 'derivative.inc'
;********************** EXPORT REFERENCES *************************
            XDEF Entry, _Startup

            XREF __SEG_END_SSTACK      ; symbol defined by the linker for the
end of the stack
            ; LCD and Pot References
            XREF
SendsChr,PlayTone,display_string,init_LCD,IRQ_ISR,RTI_ISR,pot_value,read_pot,de
lay1, gasempty
            xref keypad,wait1
            xdef song1,
songcounter,SongOfStorms,note_count,song_flag,song_num,song_list,song_up,song_d
own
            xdef keypadpress
            xdef lut
            xdef gas
            xdef array
            xdef Port_U,timer
            xdef keypress
            xdef Port_S,numsuc,stop3
            xdef counter100
            xdef counter, time1, time2, time3, choice1, choice2
            xdef stateflagg
            xdef motorstate, gasflg,stop1,halfway
            xdef ton
            xdef val1
            xdef Port_T
            xdef CRGFLG,start2, random
            xdef turnval
            xref turnlights, deliveryin
            xdef RTICTL,counter2
            xdef leftarray, rightarray, stop1flag, stop3flag
            xref panicseq
            xdef panicfg, dispwrong, dispright
            xref refuel
            xdef fuel, fueled, stop2flag,stop2
            xdef disp, disfuel, rearray, disptemp,dispspeed, dispgas
            xdef delnum, timing, timingflag

;********************** VARIABLE DECLARATION  **********************
my_variable: SECTION
songcounter:      ds.b    1
note_count:   ds.b    1
song_flag:    ds.b    1
```

```
song_num:     ds.b    1
song_up:      ds.b    1
song_down     ds.b    1
disp:     ds.b  33
stop1: ds.b 1
stop1flag: ds.b 1
disptemp:       ds.b  33
dispspeed:      ds.b  33
counter:  ds.b  1
counter2: ds.b 1
keypadpress:   ds.b  1
val1:      ds.b  1
turnval: ds.b 2
counter100: ds.b 1
ton:      ds.b 1
stateflagg:  ds.b 1
motorstate:   ds.b 1
radio:    ds.b 1
panicfg: ds.b 1
fuel:   ds.b 1
disfuel: ds.b 2
var1: ds.b 1
var2: ds.b 1
var3: ds.b 1
vari: ds.b 1
place: ds.b 1
speed: ds.b 1
gas: ds.b 1
dispcheck1: ds.b 33
dispcheck2: ds.b 33
dispgas:  ds.b  33
fueled: ds.b 1
var4: ds.b 1
var5: ds.b 1
gasflg: ds.b 1
timer: ds.b 1
dispstop1: ds.b 33
dispright: ds.b 33
dispwrong: ds.b 33
timing: ds.b 1
timingflag: ds.b 1
stop2: ds.b 1
stop2flag: ds.b 1
start2: ds.b 1
numsuc: ds.b 1
stop3: ds.b 1
stop3flag: ds.b 1
time1: ds.b 1
time2: ds.b 1
```

```
time3: ds.b 1
choice1:  ds.b  1
choice2:  ds.b  1
random: ds.b 1
radionum: ds.b 1
delnum: ds.b 1
halfway:  ds.b  33

my_constant:  SECTION
Port_T:      equ     $240
T_DDR:       equ     $242
Port_U:      equ     $268
U_DDR:       equ     $26A
U_PPS:       equ     $26D
U_PER:       equ     $26C
Port_S:      equ     $0248
S_DDR:       equ     $024A
Port_P:      equ     $258
P_DDR:       equ     $25A
;RTICTL:     equ     $003B ;not needed
RTIENA:      equ     $0038 ;CRGINT
RTIFLG:      equ     $0037 ;CRGFLG
lut:  dc.b $eb, $77, $7b, $7d, $b7, $bb, $bd, $d7, $db, $dd, $e7, $ed, $7e,
$be, $de,$ee, $0
array: dc.b $70, $B0, $D0, $E0, $0
keypress: dc.b $07, $0B, $0D, $0E, $0
steparray: dc.b $a ,$12, $14, $c,$0
leftturnarr: dc.b $c ,$14, $12, $a,$0
leftarray: dc.b $10, $20, $40, $80, $0
rightarray: dc.b $08, $04, $02, $01, $0
rearray: dc.b $2b, $2b,$2b,$2b,$2b,$2b,$2b,$2b
song1: dc.b $24, $24, $27, $27, $30, $30, $30, $30, $24, $24, $27, $27, $30,
$30, $30, $30,$00

HotCrossBuns:  dc.b
$1,$1,$1,$1,$2,$2,$2,$2,$3,$3,$3,$3,$FF,$FF,$1,$1,$1,$1,$2,$2,$2,$2,$3,$3,$3,$3
,$FF,$FF,$3,$FF,$3,$FF,$3,$FF,$3,$FF,$2,$FF,$2,$FF,$2,$FF,$2,$FF,$1,$FF,$1,$FF,
$1,$FF,$1,$FF,$FF,$FF,$FF,$0
SongOfStorms:  dc.b
$6,$5,$4,$4,$4,$4,$6,$5,$4,$4,$4,$4,$3,$3,$2,$2,$3,$2,$3,$4,$5,$5,$FF,$FF,$3,$3
,$4,$4,$3,$2,$3,$3,$3,$3,$3,$3,$3,$3,$3,$3,$5,$5,$4,$3,$5,$5,$5,$5,$5,$5,$5,$5,
$FF,$FF,$0
Doom:   dc.b
$5,$FF,$5,$FF,$2,$FF,$5,$FF,$5,$FF,$2,$FF,$5,$FF,$5,$FF,$2,$FF,$5,$FF,$5,$FF,$6
,$6,$6,$6,$FF,$5,$FF,$5,$FF,$2,$FF,$5,$FF,$5,$FF,$2,$FF,$5,$FF,$5,$FF,$2,$FF,$5
,$FF,$5,$FF,$5,$5,$4,$4,$3,$3,$FF,$0
Alarm:          dc.b   $1,$3,$0
song_list:      dc.w   HotCrossBuns,SongOfStorms,Doom,$0000,Alarm
```

```
;********************** INITIALIZATIONS  ************************
MyCode:     SECTION
Entry:
_Startup:
;-------------------- Data initializations -----------------------
           movb     #0,counter
           movb #0,timer
           movb #0, stop1
           movb #0, stop2
           movb #0, stop2flag
           movb #0, stop1flag
           movb #0, stop3flag
           movb      #0,songcounter
            movb      #0,note_count
            movb      #0,song_flag
            movb      #0,song_num
            movb      #0,song_up
            movb      #0,song_down
            movb #0, start2
            movb #0, numsuc
            movb #1, random
            movb #1, delnum


;------------------- String initializations -----------------------
           ;intializing string "disp" to be:

           ;"The value of the pot is:      ",0

           movb     #'H',halfway            ;initializing different displays
for the LCD
           movb     #'a',halfway+1
           movb     #'l',halfway+2
           movb     #'f',halfway+3
           movb     #'w',halfway+4
           movb     #'a',halfway+5
           movb     #'y',halfway+6
           movb     #' ',halfway+7
           movb     #'t',halfway+8
           movb     #'h',halfway+9
           movb     #'e',halfway+10
           movb     #'r',halfway+11
           movb     #'e',halfway+12
           movb     #'.',halfway+13
           movb     #' ',halfway+14
           movb     #' ',halfway+15
```

```
movb    #'K',halfway+16
movb    #'e',halfway+17
movb    #'e',halfway+18
movb    #'p',halfway+19
movb    #' ',halfway+20
movb    #'g',halfway+21
movb    #'o',halfway+22
movb    #'i',halfway+23
movb    #'n',halfway+24
movb    #'g',halfway+25
movb    #'!',halfway+26
movb    #' ',halfway+27
movb    #' ',halfway+28
movb    #' ',halfway+29
movb    #' ',halfway+30
movb    #' ',halfway+31
movb    #0,halfway+32


movb    #'T',disp
movb    #'h',disp+1
movb    #'e',disp+2
movb    #' ',disp+3
movb    #'v',disp+4
movb    #'a',disp+5
movb    #'l',disp+6
movb    #'u',disp+7
movb    #'e',disp+8
movb    #' ',disp+9
movb    #'o',disp+10
movb    #'f',disp+11
movb    #' ',disp+12
movb    #'t',disp+13
movb    #'h',disp+14
movb    #'e',disp+15
movb    #' ',disp+16
movb    #'p',disp+17
movb    #'o',disp+18
movb    #'t',disp+19
movb    #' ',disp+20
movb    #'i',disp+21
movb    #'s',disp+22
movb    #':',disp+23
movb    #' ',disp+24
movb    #' ',disp+25
movb    #' ',disp+26
movb    #' ',disp+27
movb    #' ',disp+28
```

```
movb    #' ',disp+29
movb    #' ',disp+30
movb    #' ',disp+31
movb    #0,disp+32


movb    #'C',dispcheck1
movb    #'h',dispcheck1+1
movb    #'e',dispcheck1+2
movb    #'c',dispcheck1+3
movb    #'k',dispcheck1+4
movb    #' ',dispcheck1+5
movb    #'t',dispcheck1+6
movb    #'h',dispcheck1+7
movb    #'r',dispcheck1+8
movb    #'o',dispcheck1+9
movb    #'t',dispcheck1+10
movb    #'t',dispcheck1+11
movb    #'l',dispcheck1+12
movb    #'e',dispcheck1+13
movb    #' ',dispcheck1+14
movb    #' ',dispcheck1+15
movb    #'A',dispcheck1+16
movb    #' ',dispcheck1+17
movb    #'w',dispcheck1+18
movb    #'h',dispcheck1+19
movb    #'e',dispcheck1+20
movb    #'n',dispcheck1+21
movb    #' ',dispcheck1+22
movb    #'r',dispcheck1+23
movb    #'e',dispcheck1+24
movb    #'a',dispcheck1+25
movb    #'d',dispcheck1+26
movb    #'y',dispcheck1+27
movb    #':',dispcheck1+28
movb    #' ',dispcheck1+29
movb    #' ',dispcheck1+30
movb    #' ',dispcheck1+31
movb    #0,dispcheck1+32

movb    #'C',dispcheck2
movb    #'h',dispcheck2+1
movb    #'e',dispcheck2+2
movb    #'c',dispcheck2+3
movb    #'k',dispcheck2+4
movb    #' ',dispcheck2+5
movb    #'r',dispcheck2+6
movb    #'e',dispcheck2+7
movb    #'a',dispcheck2+8
```

```
movb     #'r',dispcheck2+9
movb     #' ',dispcheck2+10
movb     #'v',dispcheck2+11
movb     #'i',dispcheck2+12
movb     #'e',dispcheck2+13
movb     #'w',dispcheck2+14
movb     #'s',dispcheck2+15
movb     #'B',dispcheck2+16
movb     #' ',dispcheck2+17
movb     #'w',dispcheck2+18
movb     #'h',dispcheck2+19
movb     #'e',dispcheck2+20
movb     #'n',dispcheck2+21
movb     #' ',dispcheck2+22
movb     #'r',dispcheck2+23
movb     #'e',dispcheck2+24
movb     #'a',dispcheck2+25
movb     #'d',dispcheck2+26
movb     #'y',dispcheck2+27
movb     #':',dispcheck2+28
movb     #' ',dispcheck2+29
movb     #' ',dispcheck2+30
movb     #' ',dispcheck2+31
movb     #0,dispcheck2+32

movb #'W',dispwrong
movb #'r',dispwrong+1
movb #'o',dispwrong+2
movb #'n',dispwrong+3
movb #'g',dispwrong+4
movb #' ',dispwrong+5
movb #'t',dispwrong+6
movb #'u',dispwrong+7
movb #'r',dispwrong+8
movb #'n',dispwrong+9
movb #'!',dispwrong+10
movb #' ',dispwrong+11
movb #' ',dispwrong+12
movb #' ',dispwrong+13
movb #' ',dispwrong+14
movb #' ',dispwrong+15
movb #'R',dispwrong+16
movb #'e',dispwrong+17
movb #'t',dispwrong+18
movb #'r',dispwrong+19
movb #'y',dispwrong+20
movb #' ',dispwrong+21
movb #'d',dispwrong+22
movb #'e',dispwrong+23
```

```
            movb #'l',dispwrong+24
            movb #'i',dispwrong+25
            movb #'v',dispwrong+26
            movb #'e',dispwrong+27
            movb #'r',dispwrong+28
            movb #'y',dispwrong+29
            movb #'!',dispwrong+30
            movb #' ',dispwrong+31
            movb #0,dispwrong+32

            movb #'S',dispright
            movb #'u',dispright+1
            movb #'c',dispright+2
            movb #'c',dispright+3
            movb #'e',dispright+4
            movb #'s',dispright+5
            movb #'s',dispright+6
            movb #' ',dispright+7
            movb #'c',dispright+8
            movb #'l',dispright+9
            movb #'i',dispright+10
            movb #'c',dispright+11
            movb #'k',dispright+12
            movb #' ',dispright+13
            movb #' ',dispright+14
            movb #'A',dispright+15
            movb #'-',dispright+16
            movb #'F',dispright+17
            movb #' ',dispright+18
            movb #'t',dispright+19
            movb #'o',dispright+20
            movb #' ',dispright+21
            movb #'b',dispright+22
            movb #'e',dispright+23
            movb #'g',dispright+24
            movb #'i',dispright+25
            movb #'n',dispright+26
            movb #'!',dispright+27
            movb #' ',dispright+28
            movb #' ',dispright+29
            movb #' ',dispright+30
            movb #' ',dispright+31
            movb #0,dispright+32  ;string terminator, acts like '\0'in C

;------------------- Device initialization ------------------------
;dipswitches Port T (b7 - b0)           ;WHERE B [ 7 ~ 0 ]
;LEDs Port S (b7 - b0)
;Stepper Motor Port P (b5 - b1)
;Keypad Port U (b7 - b0)
```

```
;Pushkeypadpress Port P (b6)
;DC Motor Port T (b5)              No, its actually b3[7~0]
;Speaker Port T (b3)
;j6 lower 2 pins for speaker,port t b5,b3[7~0] must be STARTED up.

            LDS  #__SEG_END_SSTACK     ;initialize the stack pointer
            movb      #$FF,S_DDR       ;LEDS output
            movb      #$1E,P_DDR       ;stepper and PB b5[7~0]
            movb      #$28,T_DDR       ;b5[7~0] speaker output, b3 out dc MUST
BE FLIPPED UP
            ;movb      #$80,INTCR       ;edge trigger - not needed
            movb      #$40,INTCR       ;enable IRQ - must have vector
            movb      #$80,RTIENA      ;enable rti

;YOU DECIDE THE DIV FOR YOUR RTI, .128MS IS RECCOMMENDED FOR THE SPEAKER
;UNCOMMENT THE LINE BELOW IF THAT IS WHAT YOU WANT TO USE
            ;movb      #$10,RTICTL      ;set div: .128ms for speaker

            movb      #$F0,U_DDR
            movb      #$F0,U_PPS
            movb      #$0F,U_PER       ;init registers of U port
            jsr       init_LCD

           movb #'P',disp
           movb #'l',disp+1                         ;initial display
           movb #'e',disp+2
           movb #'a',disp+3
           movb #'s',disp+4
           movb #'e',disp+5
           movb #' ',disp+6
           movb #'e',disp+7
           movb #'n',disp+8
           movb #'t',disp+9
           movb #'e',disp+10
           movb #'r',disp+11
           movb #' ',disp+12
           movb #'p',disp+13
           movb #'i',disp+14
           movb #'n',disp+15
           movb #' ',disp+16
           movb #' ',disp+17
           movb #' ',disp+18
           movb #' ',disp+19
           movb #' ',disp+20
           movb #' ',disp+21
           movb #' ',disp+22
           movb #' ',disp+23
           movb #0,disp+32
           ldd #disp
```

```
           jsr display_string


;************************ MAIN CODE ****************************
            ;CLI                        ;officially in operational mode (move
if you need a startup state), enable interrupts

 ;----------------- MAIN FLAG/STATE MACHINE --------------------

            ;read above! you still must decide on your rti frequency divider

alwayscheck:
           movb #$40, RTICTL
           movb #5, gasflg
           movb #$0, counter
           movb #$0, keypadpress
           movb #$0, val1
           movb #$0, counter100
           movb #$0, ton
           movb #$0, stateflagg
           movb #$0, motorstate


    password1:
          jsr panicseq
          jsr keypad     ;jump to keypad subroutine
          ldaa keypadpress    ;load value from keypad
          cmpa #1
          bne password1
    password2:
           movb #'*',disp+21
           ldd #disp
           jsr display_string
          jsr panicseq
          jsr keypad
          ldaa keypadpress    ;load value from keypad
          cmpa #2
          bne password2
    password3: movb #'*',disp+22
           ldd #disp
           jsr display_string
          jsr panicseq
          jsr keypad
          ldaa keypadpress    ;load value from keypad
          cmpa #3
          bne password3
   password4: movb #'*',disp+23
           ldd #disp
```

```
     jsr display_string
    jsr panicseq
    jsr keypad
    ldaa keypadpress    ;load value from keypad
    cmpa #4
    bne password4
    movb #'*',disp+24
     ldd #disp
     jsr display_string
     jsr wait1
     jsr wait1
     jsr wait1
     jsr wait1
    clr panicfg

password5: ldd  #dispcheck1     ;secondary display
       jsr display_string
       jsr keypad
       ldaa keypadpress
       cmpa #$a
       bne password5

password6: ldd  #dispcheck2
       jsr display_string        ;third display
       jsr keypad
       ldaa keypadpress
       cmpa #$b
       bne password6


     movb #$01, radio
     SEI ;stops interrupts by setting I in CCR
     ;ldab random
     ;stab delnum
     movb     #'S',disp
     movb     #'t',disp+1          ;waiting for first switch to be set
     movb     #'a',disp+2
     movb     #'r',disp+3
     movb     #'t',disp+4
     movb     #' ',disp+5
     movb     #'t',disp+6
     movb     #'h',disp+7
     movb     #'e',disp+8
     movb     #' ',disp+9
     movb     #'e',disp+10
     movb     #'n',disp+11
     movb     #'g',disp+12
     movb     #'i',disp+13
     movb     #'n',disp+14
```

```
          movb     #'e',disp+15
          movb     #' ',disp+16
          movb     #'t',disp+17
          movb     #'o',disp+18
          movb     #' ',disp+19
          movb     #'b',disp+20
          movb     #'e',disp+21
          movb     #'g',disp+22
          movb     #'i',disp+23
          movb     #'n',disp+24
          movb     #'.',disp+25
          movb     #'.',disp+26
          movb     #'.',disp+27
          movb     #' ',disp+28
          movb     #' ',disp+29
          movb     #' ',disp+30
          movb     #' ',disp+31
          ldd #disp
          jsr display_string
 start:
         brclr Port_T, #$01, start            ;starts engine
         lbra startradio



startradio:                                  ;radio should be playing
         ;movb #$10, RTICTL
         CLI
         movb #$81, Port_S

    nofg1:                                    ;setting flags
         ldaa stateflagg
         cmpa #1
         bne nofg1
         movb #$0, Port_S
         movb #$0 , stateflagg
   nofg2:
         ldaa stateflagg
         cmpa #1
         bne nofg2
         movb #$0, Port_S
         movb #$0 , stateflagg


   keychecker:                                ;checks if keypad has been
pressed and does something based off the press
         sei
         ldab panicfg
```

```
          cmpb #$01
          lbeq password1
          movb #$0, keypadpress
          jsr keypad
          ldaa keypadpress
          cmpa #1
          lbeq radio1show
          cmpa #2
          lbeq radio2show
          cmpa #3
          lbeq radio3show1
          cmpa #4
          lbeq leftturn
          cmpa #15
          lbeq suc
          cmpa #12
          lbeq suc
          cmpa #13
          lbeq suc
          cmpa #14
          lbeq suc
          cmpa #5
          lbeq righturn
          brclr Port_P, #$20, motoron12
          brset Port_T, #$80, motoron12


  starting:        ldab radio
          cmpb #$01
          beq radio1
          cmpb #$02
          beq radio1
          cmpb #$03
          beq radio1
 motoron:                                                  ;branch checkpoint
        lbra motoron22
  radio1show: movb #'1', dispspeed+22                       ;changes radio
display based on kepad
            ldab radio
            cmpb #$01
            beq radio1
            decb

            lbra motoron22
   radio2show:  movb #'2', dispspeed+22
            ldab radio
            cmpb #$03
            beq radio1
            incb
```

```
            lbra starting
  radio3show1: movb #'3', dispspeed+22
              lbra starting
  radio1:
          cli
          lbra motoron22              ;on successful delivery it resets flags
and picks new destination based off of keypad press
 suc:       jsr keypad
            ldd keypadpress
            ldx #3
            idiv
            addb #1
            stab delnum
            movb #0, stop1
            movb #0, stop2
            movb #0, start2
            movb #0, stop1flag
             movb #0, stop2flag
           inc numsuc
        movb #0, stop3flag
        lbra motoron
  motoron12: lbra motoron22
 leftturn:                                  ;left turn sets flags of the stops
and flickers the lights
           movb #1, stop1flag
           movb #1, stop2flag
          movb #$13, counter100

          jsr turnlights
          movb #$f0, RTICTL
          movb #$00, counter100
          cli
          ldx #leftturnarr
     loop:                                   ;turns stepper motor
          ldaa 1, x+
          cmpa #$0
          lbeq  keychecker
    ltich:
          ldab stateflagg
          cmpb #$1
          lbne ltich
          staa Port_P
          clr  stateflagg
          lbra loop
 motorempty:                                   ;if gas is empty it displays
it

          brclr Port_P, #$20, refuelanim
          lbra motoron22
```

```
  next2:
          lbra startradio

 righturn:                                  ;right turn sets flags of the stops
and flickers the lights
          movb #1, stop2flag
          movb #1, stop1flag
          movb #$13, counter100
          jsr turnlights

          movb #$f0, RTICTL
          movb #$00, counter100
          cli
          ldx #steparray
loop2:

          ldaa 1, x+                        ;turns stepper motor
          cmpa #$0
          lbeq  next2
    rtich:
          ldab stateflagg
          cmpb #$1
          lbne rtich
          staa Port_P
          clr  stateflagg
          lbra loop2

  refuelanim:

          jsr refuel

          lbra next2
 motoron22:

           cli


          movb #$01, motorstate
          movb #$40, RTICTL
         movb  #0,counter      ;checking of multiple flags
     ldaa stop3flag
     cmpa #$6
     lbeq succ
      lbra timingg

     succ: movb #$00, motorstate
           bclr Port_T, #$8
           ldd #dispright
           jsr display_string
```

```
          lbra keychecker


  timingg: ldaa timingflag        ;check to see if its time to make first turn
        cmpa #0
       lbeq end1
        ldaa stop1flag
         cmpa #$6
         beq stoper
         cmpa #1
         beq check2
         lbra sy

check2:   ldaa stop2flag          ;check to see if its time to make second turn
       cmpa #$6
        lbne sy
 stoper:        movb #$00, motorstate        ;lcd message for making turn
           bclr Port_T, #$8

          movb #$f0, RTICTL
           movb #'M',dispstop1
           movb #'a',dispstop1+1
           movb #'k',dispstop1+2
           movb #'e',dispstop1+3
           movb #' ',dispstop1+4
           movb #'a',dispstop1+5
           movb #' ',dispstop1+6
           movb #'t',dispstop1+7
           movb #'u',dispstop1+8
           movb #'r',dispstop1+9
           movb #'n',dispstop1+10
           movb #' ',dispstop1+11
           movb #'n',dispstop1+12
           movb #'o',dispstop1+13
           movb #'w',dispstop1+14
           movb #',',dispstop1+15
           movb #'l',dispstop1+16
           movb #'e',dispstop1+17
           movb #'f',dispstop1+18
           movb #'t',dispstop1+19
           movb #' ',dispstop1+20
           movb #'o',dispstop1+21
           movb #'r',dispstop1+22
           movb #' ',dispstop1+23
           movb #'r',dispstop1+24
           movb #'i',dispstop1+25
           movb #'g',dispstop1+26
           movb #'h',dispstop1+27
           movb #'t',dispstop1+28
```

```
          movb #'.',dispstop1+29
          movb #'.',dispstop1+30
          movb #'.',dispstop1+31
          movb #0,dispstop1+32
          ldd #dispstop1
          jsr display_string

        lbra  keychecker
                   ; range of 0-255
sy:        ldaa gasflg                    ;lcd message for empty gas
         cmpa #$6
         lbne stat
          movb #$00, motorstate
          bclr Port_T, #$8

          movb #$f0, RTICTL
          movb #'G',dispgas
          movb #'a',dispgas+1
          movb #'s',dispgas+2
          movb #' ',dispgas+3
          movb #'e',dispgas+4
          movb #'m',dispgas+5
          movb #'p',dispgas+6
          movb #'t',dispgas+7
          movb #'y',dispgas+8
          movb #' ',dispgas+9
          movb #'p',dispgas+10
          movb #'l',dispgas+11
          movb #'e',dispgas+12
          movb #'a',dispgas+13
          movb #'s',dispgas+14
          movb #'e',dispgas+15
          movb #'r',dispgas+16
          movb #'e',dispgas+17
          movb #'f',dispgas+18
          movb #'u',dispgas+19
          movb #'e',dispgas+20
          movb #'l',dispgas+21
          movb #' ',dispgas+22
          movb #'n',dispgas+23
          movb #'o',dispgas+24
          movb #'w',dispgas+25
          movb #'!',dispgas+26
          movb #' ',dispgas+27
          movb #' ',dispgas+28
          movb #' ',dispgas+29
          movb #' ',dispgas+30
          movb #' ',dispgas+31
          movb #0,dispgas+32
```

```
           ldd #dispgas
           jsr display_string
           lbra motorempty
                                                          ;takes in pot value
    stat:      jsr read_pot
          ldd pot_value
         ;; subb #5
          ;ldaa gas
          ;cmpa #70
         ; lbeq jump

                                                          ;converts pot value to
a range of 0-50
         ldx #5
           idiv
           tfr x,d
           ldx #100
             idiv
             xgdx
             addb  #$30
             stab  var1
             tfr x,d
            ldx #10
            idiv
            xgdx
             addb  #$30
            stab var2
             tfr  x,d
             addb  #$30
             stab  var3
             ldab var2

       ldd timer                                                  ;based on
how long the motor has been running there is a timer that increases
       ldx #100
          idiv
          tfr x,d
        cpd #17
        bhi hi1
        movb #'1',dispspeed+28
        movb #'0',dispspeed+29
        movb #'0',dispspeed+30
        lbra sar

   hi1:cpd #32                                                ;displays
percentage based on the value of the timer with respect to gas level
       bhi hi2
      movb #' ',dispspeed+28
       movb #'9',dispspeed+29
```

```
   movb #'0',dispspeed+30
   movb #' ',dispspeed+31
   lbra sar
hi2:cpd #48
   bhi hi3
   movb #' ',dispspeed+28
   movb #'8',dispspeed+29
   movb #'0',dispspeed+30
   movb #' ',dispspeed+31
   lbra sar
hi3:cpd #64
   bhi hi4
  movb #' ',dispspeed+28
   movb #'7',dispspeed+29
   movb #'0',dispspeed+30
   movb #' ',dispspeed+31
   lbra sar
hi4:cpd #80
   bhi hi5
  movb #' ',dispspeed+28
   movb #'6',dispspeed+29
   movb #'0',dispspeed+30
   movb #' ',dispspeed+31
   lbra sar
 hi5:cpd #96
   bhi hi6
  movb #' ',dispspeed+28
   movb #'5',dispspeed+29
   movb #'0',dispspeed+30
   movb #' ',dispspeed+31
   lbra sar
hi6:cpd #112
   bhi hi7
  movb #' ',dispspeed+28
   movb #'4',dispspeed+29
   movb #'0',dispspeed+30
   movb #' ',dispspeed+31
   lbra sar
hi7:cpd #128
   bhi hi8
 movb #' ',dispspeed+28
   movb #'3',dispspeed+29
   movb #'0',dispspeed+30
   movb #' ',dispspeed+31
   lbra sar
hi8:cpd #144
   bhi hi9
 movb #' ',dispspeed+28
   movb #'2',dispspeed+29
```

```
       movb #'0',dispspeed+30
       movb #' ',dispspeed+31
       lbra sar
   hi9: movb #' ',dispspeed+28
   movb #'1',dispspeed+29
       movb #'0',dispspeed+30
       movb #' ',dispspeed+31
       lbra sar

sar:
        ldaa delnum                                          ;main lcd display
     cmpa #1
     beq first
     cmpa #2
     beq secondd
     cmpa #3
     beq third
     bra num

     first:  movb #'1', dispspeed+5
             bra num
     secondd: movb #'2', dispspeed+5
             bra num

     third: movb #'3', dispspeed+5
      bra num

    num:      movb #'D',dispspeed
          movb #'e',dispspeed+1
          movb #'s',dispspeed+2
          movb #'t',dispspeed+3
          movb #':',dispspeed+4

          movb #' ',dispspeed+6
          movb #'M',dispspeed+7
          movb #'P',dispspeed+8
          movb #'H',dispspeed+9
          movb #':',dispspeed+10
          movb #' ',dispspeed+11
          movb var2,dispspeed+12
          movb var3,dispspeed+13
          movb #' ',dispspeed+14
          movb #' ',dispspeed+15
          movb #'R',dispspeed+16
          movb #'a',dispspeed+17
          movb #'d',dispspeed+18
          movb #'i',dispspeed+19
          movb #'o',dispspeed+20
          movb #':',dispspeed+21
```

```
           movb #' ',dispspeed+23
           movb #'G',dispspeed+24
           movb #'a',dispspeed+25
           movb #'s',dispspeed+26
           movb  #':',dispspeed+27
           movb #'%',dispspeed+31
           movb #0,dispspeed+32



          ldd #dispspeed
           jsr display_string

 end1:         movb #$0, motorstate

          lbra next2

 delivery:
          movb #$02, Port_S
          lbra keychecker

  end
```

RTI_ISR.asm

```
              XDEF RTI_ISR
              xref counter
              xref ton, start2
              xref Port_T
              xref CRGFLG
              xdef RTI_ISR
              xref stateflagg
              xref motorstate
              xref counter100
              xref PlayTone
              xref SendsChr
              xref song1,delnum, random
              xref read_pot
              xref pot_value,time1, time2, time3
              xref counter2,songcounter,note_count,song_flag, stop3flag
              xref gas,gasflg,stop2flag, timer, stop1,stop3, stop1flag,
timingflag, timing, stop2

RTI_ISR:
             ldaa delnum
              cmpa #1
              beq varii1
              cmpa #2
```

```
          beq varii2
          cmpa #3
          beq varii3
   varii1:  movb #17, time2
     movb #5, time3
     bra check1


varii2:  movb #17, time2
     movb #22, time3
     bra check1

varii3:  movb #13, time2
     movb #9, time3
     bra check1


   check1:      ldaa timingflag
         cmpa #0
         lbeq time
        ldaa motorstate
        cmpa #$1

        beq next
        inc timing

        inc counter100
        ldaa counter100
        cmpa #$20
        lbne exit
        movb #$1, stateflagg
        movb #$0, counter100
        lbra exit


  next:
       ldd pot_value
       cpd #01
       lblo less1      ;if motor isn't running skip the fuel counter

      inc counter2      ;<---\
      ldx counter2     ;    |
       cpx #1000           ;    |
       bls  skip         ;    |
       clr  counter2     ;    |
      inc gas         ;    counts every second while motor is spinning
       ldab gas
       cmpb #255
       beq timerinc
```

```
        bra skip        ;    |

  timerinc: inc timer
            inc stop1
            inc stop2
            inc stop3
            ldab timer
            cmpb #63
            beq flagset
            ldaa stop1
            cmpa #22
            beq stop1fg
            ldaa stop1flag
            cmpa #1
            lbne skip
            ldaa stop2
            cmpa time2
            beq stop2fg
            ldaa stop2flag
            cmpa #1
            lbne exit
            ldaa stop3
            cmpa time3
            beq stop3fg
            bra exit

   flagset: movb #$6, gasflg
            ldab #0
            stab gas
            stab timer
            ldy #0
            sty counter2


             bra exit

 stop1fg: movb #$6, stop1flag

       bra exit

 stop2fg: movb #$6, stop2flag
 bra exit

stop3fg: movb #$6, stop3flag
bra exit

   skip: ldd pot_value
         cpd #0
         lbeq less1
```

```
          ldx #5
          idiv
          tfr x,d

          stab ton
          ldab counter
          cmpb ton
          ;beq less1
          bls less2
          ;ldab ton
          ;bls less2
          cmpb #51
          bls less1
          bra greater

    less1: bclr Port_T, #$8
        inc counter
         bra exit
    less2: bset Port_T, #$8
       inc counter
         bra exit
  greater: movb #0, counter
         bra exit

    time: ldx timing
          cpx #2000
          inc timing
          bne exit
          movb #1, timingflag
    exit:
         bset CRGFLG, #$80
   rti
```

IRQ_ISR.asm

```
              XDEF      IRQ_ISR
              xref panicfg, song_num

IRQ_ISR:
         movb #$01, panicfg
         movb #4, song_num

        rti
```

delay.asm

```
    xdef keypad
    xdef delay1ms
    xref stateflagg
```

```
    xref keypadpress
    xref lut
    xref array
    xref Port_U
    xref keypress
    xref Port_S
    xref  val1
    xdef wait1
    xref timing, timingflag
    xref display_string
    xref RTICTL
    xref dispgas
    xref fueled
    xref Port_T
    xref motorstate,gasflg


 wait1:    cli
           movb #0, timingflag


           rts



delay1ms:     ldy  #4000
        loop:
              dey
              bne loop
              rts



keypad:   ldx #array

 loop2:   ldaa 1, x+
          cmpa #$0
          beq  last
          staa Port_U
          jsr delay1ms
          ldaa Port_U
          staa val1
          brset Port_U, $0f, loop2

          ;Look up table
          ldab #$0
          ldx #lut ;load array to reg x
 loopback:
```

```
          ldaa 0,x
          cmpa #0
          beq done
          cmpa val1
          beq found
          inx
          incb
          bra loopback
    done:
          bra last
   found: stab keypadpress
          bra  last
    last:
          rts
```

panicseq.asm

```
  xdef panicseq
  xref RTICTL
  xref panicfg
  xref SendsChr, PlayTone
  xref stateflagg, Port_S, disp, display_string
  xref motorstate
  xref Port_T


panicseq:
        movb #$00, motorstate
        bclr Port_T, #$8
        ldaa panicfg
        cmpa #$01

        lbne exit
        movb #$20, RTICTL
        cli
        ldaa #8    ;plays b flat/ b
        psha
        ldd #0
        jsr SendsChr
        leas 1,sp
        jsr PlayTone
        movb #$40 , RTICTL
        movb #$ff, Port_S
  nofg3:
        ldaa stateflagg
        cmpa #1
        bne nofg3
        movb #$0, Port_S
        movb #$0 , stateflagg
```

```
 sei
movb #'P',disp
 movb #'l',disp+1
 movb #'e',disp+2
 movb #'a',disp+3
 movb #'s',disp+4
 movb #'e',disp+5
 movb #' ',disp+6
 movb #'e',disp+7
 movb #'n',disp+8
 movb #'t',disp+9
 movb #'e',disp+10
 movb #'r',disp+11
 movb #' ',disp+12
 movb #'p',disp+13
 movb #'i',disp+14
 movb #'n',disp+15
 movb #' ',disp+16
 movb #' ',disp+17
 movb #' ',disp+18
 movb #' ',disp+19
 movb #' ',disp+20
 movb #' ',disp+21
 movb #' ',disp+22
 movb #' ',disp+23
 movb #' ',disp+24
 movb #' ',disp+25
 movb #' ',disp+26
 movb #' ',disp+27
 movb #' ',disp+28
 movb #' ',disp+29
 movb #' ',disp+30
 movb  #' ',disp+31
 movb #0,disp+32
 ldd #disp
 jsr display_string

 movb #$10, RTICTL
 cli
 ldaa #08
 psha
 ldd #0
 jsr SendsChr
 leas 1, sp
 jsr PlayTone
 sei
```

```
   exit:
        rts
```

turnlights.asm

```
  xdef turnlights
  xref Port_S
  xref stateflagg
  xref RTICTL
  xref turnval
  xref keypadpress
  xref leftarray, rightarray
  xref counter100, stop1flag, deliveryin

turnlights:
          cli
          movb #$f0, RTICTL
          ldaa keypadpress
          cmpa #$05
          beq right

    left:
          ldx #leftarray
    lloop:
          ldaa 1, x+
          cmpa #$0
          beq  exit
    ltich:
          ldab stateflagg
          cmpb #$1
          bne ltich
          staa Port_S
          clr  stateflagg
          bra lloop

     right:
          ldx #rightarray
    rloop:
          ldaa 1, x+
          cmpa #$0
          beq  exit
    rtich:
          ldab stateflagg
          cmpb #$1
          bne rtich
          staa Port_S
          clr  stateflagg
          bra rloop
```

```
    exit: jsr deliveryin
          sei
          movb #$0, counter100
          movb #$0 , stateflagg
          rts
```

refuel.asm

```
    xdef refuel
    xref fuel
    xref disp
    xref display_string
    xref RTICTL, stateflagg
    xref disfuel
    xref rearray
    xref delay1
    xref disptemp
    xref dispspeed, refueling
    xref wait1, motorstate, Port_T, gas, fueled,gasflg,timer

 refuel:
           movb #$00, motorstate
           bclr Port_T, #$8
           ldaa #0
           staa gas
           staa timer
          movb #1, gasflg
           movb #$0, stateflagg
           movb #$f0, RTICTL
           movb #'G',disptemp
           movb #'a',disptemp+1
           movb #'s',disptemp+2
           movb #' ',disptemp+3
           movb #'i',disptemp+4
           movb #'s',disptemp+5
           movb #' ',disptemp+6
           movb #'r',disptemp+7
           movb #'e',disptemp+8
           movb #'f',disptemp+9
           movb #'u',disptemp+10
           movb #'e',disptemp+11
           movb #'l',disptemp+12
           movb #'i',disptemp+13
           movb #'n',disptemp+14
           movb #'g',disptemp+15
           movb #' ',disptemp+16
           movb #' ',disptemp+17
           movb #' ',disptemp+18
           movb #' ',disptemp+19
```

```
          movb #' ',disptemp+20
          movb #' ',disptemp+21
          movb #' ',disptemp+22
          movb #' ',disptemp+23
          movb #' ',disptemp+24
          movb #' ',disptemp+25
          movb #' ',disptemp+26
          movb #' ',disptemp+27
          movb #' ',disptemp+28
          movb #' ',disptemp+29
          movb #' ',disptemp+30
          movb #' ',disptemp+31
          movb #0,disptemp+32
          ldd #disptemp
          jsr display_string
          bra fueling
          CLI
          ldaa fuel
          cmpa #$08
          lbeq exit
          adda #1
    wait:
          ldab stateflagg
          cmpb #$01
          bne wait

fueling:
           movb #$f0, RTICTL
           movb #'-',disptemp+20
          ldd #disptemp
          jsr display_string

           jsr wait1
           jsr wait1
           movb #'-',disptemp+21
           ldd #disptemp
          jsr display_string
          jsr wait1
          jsr wait1
          movb #'-',disptemp+22
          ldd #disptemp
          jsr display_string
          jsr wait1
          jsr wait1
          movb #'-',disptemp+23
          ldd #disptemp
          jsr display_string
          jsr wait1
          jsr wait1
```

```
          movb #'-',disptemp+24
          ldd #disptemp
          jsr display_string
          jsr wait1
          jsr wait1
          movb #'-',disptemp+25
          ldd #disptemp
          jsr display_string
          jsr wait1
          jsr wait1
          movb #'-',disptemp+26
          ldd #disptemp
          jsr display_string
          jsr wait1
          jsr wait1
          movb #'-',disptemp+27
          ldd #disptemp
          jsr display_string
          jsr wait1


          ldd #dispspeed
          jsr display_string
          jsr wait1

    exit:
         SEI
         rts
```

deliveryin.asm

```
    xdef refuel
    xref fuel
    xref disp
    xref display_string
    xref RTICTL, stateflagg
    xref disfuel
    xref rearray
    xref delay1
    xref disptemp
    xref dispspeed, refueling
    xref wait1, motorstate, Port_T, gas, fueled,gasflg,timer

 refuel:
          movb #$00, motorstate
          bclr Port_T, #$8
          ldaa #0
          staa gas
          staa timer
```

```
      movb #1, gasflg
      movb #$0, stateflagg
      movb #$f0, RTICTL
      movb #'G',disptemp
      movb #'a',disptemp+1
      movb #'s',disptemp+2
      movb #' ',disptemp+3
      movb #'i',disptemp+4
      movb #'s',disptemp+5
      movb #' ',disptemp+6
      movb #'r',disptemp+7
      movb #'e',disptemp+8
      movb #'f',disptemp+9
      movb #'u',disptemp+10
      movb #'e',disptemp+11
      movb #'l',disptemp+12
      movb #'i',disptemp+13
      movb #'n',disptemp+14
      movb #'g',disptemp+15
      movb #' ',disptemp+16
      movb #' ',disptemp+17
      movb #' ',disptemp+18
      movb #' ',disptemp+19
      movb #' ',disptemp+20
      movb #' ',disptemp+21
      movb #' ',disptemp+22
      movb #' ',disptemp+23
      movb #' ',disptemp+24
      movb #' ',disptemp+25
      movb #' ',disptemp+26
      movb #' ',disptemp+27
      movb #' ',disptemp+28
      movb #' ',disptemp+29
      movb #' ',disptemp+30
      movb #' ',disptemp+31
      movb #0,disptemp+32
      ldd #disptemp
      jsr display_string
      bra fueling
      CLI
      ldaa fuel
      cmpa #$08
      lbeq exit
      adda #1
wait:
      ldab stateflagg
      cmpb #$01
      bne wait
```

```
fueling:
          movb #$f0, RTICTL
          movb #'-',disptemp+20
         ldd #disptemp
         jsr display_string

          jsr wait1
          jsr wait1
          movb #'-',disptemp+21
          ldd #disptemp
         jsr display_string
         jsr wait1
         jsr wait1
         movb #'-',disptemp+22
         ldd #disptemp
         jsr display_string
         jsr wait1
         jsr wait1
         movb #'-',disptemp+23
         ldd #disptemp
         jsr display_string
         jsr wait1
         jsr wait1
         movb #'-',disptemp+24
         ldd #disptemp
         jsr display_string
         jsr wait1
         jsr wait1
         movb #'-',disptemp+25
         ldd #disptemp
         jsr display_string
         jsr wait1
         jsr wait1
         movb #'-',disptemp+26
         ldd #disptemp
         jsr display_string
         jsr wait1
         jsr wait1
         movb #'-',disptemp+27
         ldd #disptemp
         jsr display_string
         jsr wait1


         ldd #dispspeed
         jsr display_string
         jsr wait1

   exit:
```

```
          SEI
          rts
```

PlaySong.asm

```
        xdef  PlaySong
        xref  note_count,song_flag,SendsChr


 PlaySong:    ldab  note_count
              abx
              ldaa  0,x
              bne   next
              clr   note_count
              bra   exit

      next:   cmpa  #$FF
              bne   play
              bclr  song_flag,#1
              bra   continue

      play:   bset  song_flag,#1
  continue:   psha
              ldd   #0
              jsr   SendsChr
              leas  1,sp

      exit:   rts
```

SongSelect.asm

```
        xdef    SongSelect
        xref    PlaySong,song_list,song_num,song_up,song_down


SongSelect:     pshx
                pshb

                ldab  song_up     ;check song_up flag
                beq   next1
                inc   song_num    ;if set, increment index
                clr   song_up

     next1:     ldab  song_down   ;check song_down flag
                beq   next2
                dec   song_num    ;if set, decrement index
                clr   song_down
```

```
next2:      ldab  song_num
            bpl   next3       ;check for invalid index
            addb  #3
            stab  song_num

next3:      lslb
            ldx   #song_list
            abx
            ldx   0,x
            bne   song        ;if not null element, play song
            clr   song_num    ;if null, reset to first array element
            bra   next3

 song:      jsr   PlaySong

            pulb
            pulx
            rts
```

Appendix III: Project and Report Requirements

Project Requirements:

Final Project ECE36200
Delivery Airplane

OBJECTIVE:
The goal of this project is to simulate a delivery plane that will take entire loads to airports across the country. It will have features like a dash and controls, wheels, gas, and a locking door handle. Your plane will receive a delivery destination, then you must fly the plane to the specific destination without messing up. If your plane runs out of gas, you must refill it. When the program first boots, you must first unlock the plane with a PIN code. If the panic button is set, an alarm will go off, and the PIN must be entered to unlock the plane.

FEATURES:

1. You can assume that for every delivery, you will start at the same airport, after each delivery is made, you may assume that you automatically return to that airport.
2. Delivering process
    a. Which destination you deliver to each time is random. There are multiple ways to do that. HINT: modulo & RTI counters/time are a good place to start, think of C code
    b. All delivery routes require two direction choices: one at the beginning when you leave the warehouse, and the other at a subsequent turn. You make the choice

via the keypad (remember this should trigger a brief turn signal flashing pattern, and wheel turn)

c. Once you choose a direction, you must use the potentiometer to speed up all the way, then slow down all the way to a stop at the destination. Then you will either make another decision or arrive at the delivery location.

d. Upon taking a wrong turn, the LCD should display that you failed to make the delivery. Then you start over with a new delivery. (Once again, think of edge cases with gas and other peripheral behavior. Does it make sense?)

e. Upon arrival, the LCD should display a success message, then you will start over with another assigned destination.

f. There are many ways to create bonus material here by making the system more sophisticated!

3. Fuel

a. You can assume you can refuel at any point on the route.

b. If fuel runs out, you must display a message on the LCD, abruptly stop the plane, and refill the tank before completing the trip (by the usual speed up, then slow down process).

c. Think about the edge cases: running out at the destination/start? You must appropriately still handle this while executing the regular requirements.

d. You must consider the miles per gallon of the plane (100 miles for easy math) to keep track of the charge level. The power consumption can be calculated and applied using the distances shown in the map below.

e. This topic has a lot of potential for bonus features.

4. Upon program startup, you must get into the plane and start it via entering the password. (This should be prompted on the LCD)

5. After starting the plane, the radio should “play music” through the speaker, with the pilot able to change between 3 different “radio stations.”

6. The plane should not be able to be moved via the potentiometer when it does not make sense. I.e.: panic mode, choosing a turn.

7. EXTRA FEATURES

a. You are required to create additional feature(s) as part of the project. Part of the grade will be based on extras that you will create on your own. You are encouraged to be creative; the more complex, the more points are earned.

b. One feature is acceptable if it is advanced enough (A bit simpler than one of the main objectives; involving multiple steps/peripherals, etc. About the same difficulty as the IRQ is good.)

c. Otherwise, multiple smaller things would suffice. Many things on par with this are listed in italics throughout the instructions. Sufficient things to do multiple of usually enhance existing things.

d. If in doubt... ask a TA before you do it.

8. No delay loops are allowed, you must utilize the Real Time Interrupt (RTI). DELAY LOOPS ARE ONLY ALLOWED FOR HEX KEYPAD DE-BOUNCE.

PERIPHERALS:

1. Stepper Motor:

The stepper motor will be used to represent the rudder on the back of the plane. When turning left or right, it should turn a quarter of the way quickly (right is clockwise and vice-versa), then turn the other direction back to its original position. Remember, plane rudders are reverse to the turn direction (the rudder turning left will cause the plane to go right).

2. DC Motor and Potentiometer:

The airplane's propeller/engines will be represented by the DC motor. The speed can be changed using the potentiometer which acts as the thrust control. This should affect the speedometer display, in which the plane's speed should be the pot value scaled between 0-50 MPH. (I know this is unrealistic, but I don't want to take your valuable screen real-estate; if you want to maintain realism, commercial aircraft typically fly around 500mph).

3. Keypad:

- Two keys will be used to adjust the radio up and down. You should be able to change the radio at any time you are in the plane and it's not in panic.
- One key for each direction; a left turn and a right turn
- At times where only certain key presses make sense (i.e., choosing a direction), other key presses should do nothing.
- Keypad should also be able to enter a 4-digit password to unlock the plane upon startup and use the same password to activate your plane after panic mode.

4. LCD:

- The LCD displays all the important information for the operator to see. Every plane has a dash cluster that reports a myriad of information about the state of the vehicle. For the main screen, you should display current speed, radio station number (1,2, etc.), current destination, and finally fuel level. You have freedom to set this up, but it must make sense (fuel could be shown as %, etc.).
- It should display the animation referenced under 'push button.'
- It should display failure/success messages (briefly or until dismissed).
- It should tell the user the success/fail count as described under 'switches.'
- It should include other needed prompts for proper user interaction. Most have been mentioned elsewhere.

5. Switches:

- Switch 1 [8~1] is the ignition. The plane should not move if it is not flipped up, and the lcd should tell the user to turn the ignition first before continuing. This should be able to occur at any time while in a 'flying' state.
- Switch 8 [8~1] should show the regular dashboard when low. But when flipped

from low to high, should display how many successful and failed deliveries you have completed for 5 seconds before returning to the regular dash display. You must manually reset the switch (when going high to low, it shouldn't do anything).

6. Push Button:

REFER TO FUEL REQUIREMENTS (#3 in features). The push button is used to recharge the vehicle. Once pressed, your gas level should increase to full. Display a smooth animation on the LCD of a bar filling to represent the recharging. Something like this [###], becomes [####], etc.

(The animation can start from an empty tank each time, though an accurate fill would be a good bonus idea. I.e., if you have 40% gas, start the animation at 40%)

7. Speaker (Port T):

When the radio station is changed, the audible tone pattern should change. Each (of at least 3) radio stations should have a distinct song which plays on repeat. These should be very simple. A handful of different notes arranged in a pattern is fine. It does not need to represent any 'real' song. There should also be an alarm for the panic mode.

8. LEDs:

Use of LED is when the plane is flying. The left-most and right-most LED must flash on and off to warn other planes flying nearby. This is to represent the lights present on the tips of the wings. There should also be a sort of "turn-signal" with the left/right nibbles moments before the plane will be making a turn. LEDs will also be used during "panic mode."

9. IRQ:

When pressed, this simulates the plane's panic button and will make the lights operate like an alarm. All the LEDs should flash off and on, and the speaker should switch between two tones like an alarm. You need to enter the password to be operational again.

10. RTI controls the timing of the simulation.

(Note: you should only have one RTI for the entire project, and never call it directly)

UNIVERSAL REQUIREMENTS:

1. Each 'song' should be unique, repeating on a loop for the duration of the given event. "Songs" are truly just patterns of a handful of different notes.

2. The overall layout of your system should be easy to understand and make sense. The user should be able to operate the system with little to no training or explanation. If you are unsure if the layout of your system makes sense, ask one of your TAs or fellow students to try to move through your system.

3. No delay loops are allowed, you must utilize the Real Time Interrupt (RTI). DELAY LOOPS ARE ONLY ALLOWED FOR HEX KEYPAD DE-BOUNCE. HINT: If you are having trouble with the switches bouncing, maybe try only scanning them every X many RTIs.

4. Further statements about in which states of the system the user should be able to accomplish

certain tasks. I.e.: at any time, or only when X is happening
NOTE:
You are encouraged to be creative and make this project your own. You can make reasonable assumptions in the development of this project, but keep in mind that the assumptions must make sense to the user (and to the Lab TAs) so be ready to explain your choices. This is not an excuse to
cut corners.
FOR YOUR PROJECT, X% OF THE GRADE WILL BE BASED ON EXTRAS THAT YOU WILL
CREATE ON YOUR OWN. YOU ARE ENCOURAGED TO BE CREATIVE; THE MORE COMPLEX, THE MORE POINTS ARE EARNED.
If you have any questions pertaining to this project, please discuss them with your Lab TAs as early as possible. You may use any C code provided through the lab, but all other code must be written in assembly.

Additional Features Ideas:
1. Add password encryption, so that a '*' appears on the LCD rather than the number.
2. Implement a compass onto the LCD.
3. Add password protection by not allowing the user to have repeated values in the password.
4. Animate a message across the screen after a delivery is failed or completed.
5. Implementing a more robust tracking system on the map allowing you to manually return to the warehouse.
6. Create a check that you have enough gas to finish a flight before takeoff.
You could also come up with your own ideas but be sure to demonstrate them to the TA grading you clearly.

Report Requirements:

The final project should include the following sections:

1. Cover letter:
    a. The project title
    b. The name of the team members
    c. Project date (Fall 2013)
    d. Should include a picture of the equipment and/or a picture relevant to the project
2. Table of contents
3. List of figures and tables
4. Introduction:
    a. Describe the goal and the purpose of the project.
    b. Assumptions made

5. Design:
    a. Show the peripherals that are used in the project and what they are used for
    b. Software implementation of the project:
        i. Give a high level description and discussion
        ii. General system flow chart
        iii. Flow charts for each module
        iv. Error handling and fail safe techniques
    c. Changes made in the design
    d. Additions to the project
6. Description of the division of work between team members
7. Description of which parts of the proposed project is working and which part is not working
8. Conclusion
9. Discussion and suggestions for future improvements on your project
10. Appendixes:
    a. User Manual
    b. Code:
        i. The code should be commented (useful and meaningful comments)
        ii. Description of each file, subroutine and procedure:
            • Name
            • Inputs/Outputs and method of passing the parameters
            • General Description
11. References

Note:
• Use headings and subheadings throughout the report
• Pay attention to grammatical and spelling errors
• All figures and flow charts should be done using software (paint, Visio etc.)
• The report should be consistent in style
• CODE SHOULD BE SUBMITTED AS ENTIRE PROJECT IN ZIPPED FORMAT ALONG WITH REPORT.
• Fonts and sizes:
o Single spaced
o Use the “Times New Roman” font or any similar font
o Use font size 14 bold for headings
o Use font size 12 for subheadings
o Use font size 12 for text
o Use the “Courier New” font for the code and the size should be 10