**EEL 4742 LABORATORY EXPERIMENT # 4**

**Section:** 0011 **Points:** 100

**Name:** Kelsey Cameron

**Introduction:**

The purpose of this lab was to understand how to use the on-chip LCD screen to display various numbers and letters. Many challenges I faced, including remembering to subtract from the tens column when decrementing the counter in part 4, or figuring out exactly which bits to turn on and which to turn off in order to print each of the numbers and letters. This assignment is valuable because in the real world, everything has a screen and needs to be turned on or off. My program is unique because it sets the bits at a configurable location. In C, I use a variable to store this index (0-7) but in assembly I have to add and subtract to the register that corresponds to the LCD address, in this case, R5.

Regardless of challenges, I managed to successfully complete this lab. My code is listed below.

**Complete C Code:**

// kelsey cameron

**#include** "msp430fg4618.h"

**#include** "stdio.h"

**void** **showEights**(**void**);

**void** **showRightMostTwo\_part1**(**void**);

**void** **showLeftMostTwo\_part2**(**void**);

// Defined functions for each character, pass in row value.

**void** **zero**(**int** row);

**void** **one**(**int** row);

**void** **two**(**int** row);

**void** **three**(**int** row);

**void** **four**(**int** row);

**void** **five**(**int** row);

**void** **six**(**int** row);

**void** **seven**(**int** row);

**void** **eight**(**int** row);

**void** **nine**(**int** row);

**void** **showA**(**int** row);

**void** **showB**(**int** row);

**void** **showC**(**int** row);

**void** **showD**(**int** row);

**void** **showE**(**int** row);

**void** **showF**(**int** row);

**void** **part3CountToF**(**void**);

**void** **part4Count**(**void**);

**void** **reset**(**int** row);

// 1.0 1.2 SW1 SW2

**void** **Init\_LCD**(**void**);

// setup a pointer to the area of memory of the TMS430 that points to

// the segments

// of the softbaugh LCD LCDM3 = the starting address

// each of the seven segments for each display is store in memory

// starting at address LCDM3

// which is the right most seven segment of the LCD

// The bit order in each byte is

// dp, E, G, F, D, C, B, A or

// :, E, G, F, D, C, B, A

// after the seven segments these memory locations are used to turn on

// the special characters

// such as battery status, antenna, f1-f4, etc.

// there are 7 seven segment displays

**unsigned** **char** \*LCDSeg = (**unsigned** **char** \*) &LCDM3; // a char pointer is used because default for char \* is +1 incrementation

// there are 11 locations that are needed for the softbaugh LCD

// only 7 used for the seven segment displays

**int** LCD\_SIZE=11;

**int** **main**(**void**){

**volatile** **unsigned** **char** a;

WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer

// configure each input and output

P2DIR |= 0x06; // 0110

P1DIR = 0x00;

Init\_LCD();

//showRightMostTwo\_part1();

//showLeftMostTwo\_part2();

//part3CountToF();

//part4Count();

}

//---------------------------------------------------------------------

// Initialize the LCD system

//---------------------------------------------------------------------

**void** **Init\_LCD**(**void**){

// Using the LCD A controller for the MSP430fg4618

// the pins of the LCD are memory mapped onto the mp430F4xxx

// memory bus and

// are accessed via LCDSeg[i] array

// See page 260 of Davie's text

// LCD\_SIZE-4 only gives the 7 segment displays plus DP, and

// (colons are the same bit setting)

// LCD\_SIZE-4 only gives the 7 segment displays plus DP, and

// colons (colons / dp)

// Right most seven segment display is at LCDSeg[0];

// Display format

// AAA

// F B

// X F B

// GGG

// X E C

// E C

// DP DDD

// bit order

// dp, E, G, F, D, C, B, A or

// :, E, G, F, D, C, B, A

**int** n;

**for** (n=0;n<LCD\_SIZE;n++){

// initialize the segment memory to zero to clear the LCD

// writing a zero in the LCD memory location clears turns

// off the LCD segment

// Including all of the special characters

// This way or

\*(LCDSeg+n) = 0;

}

// Port 5 ports 5.2-5.4 are connected to com1, com2, com3 of LCD and

// com0 is fixed and already assigned

// Need to assign com1 - com3 to port5

P5SEL = 0x1C; // set port5 to be in charge of the LED // BIT4 | BIT3 |BIT2 = 1 P5.4, P.3, P5.2 = 1

// Used the internal voltage for the LCD bit 4 = 0 (VLCDEXT=0)

// internal bias voltage set to 1/3 of Vcc, charge pump disabled,

// page 26-25 of MSP430x4xx user manual

LCDAVCTL0 = 0x00; // set the voltage of the LEDS to zero

// LCDS28-LCDS0 pins LCDS0 = lsb and LCDS28 = MSB need

// LCDS4 through LCDS24

// from the experimenter board schematic the LCD uses S4-S24,

// S0-S3 are not used here

// Only use up to S24 on the LCD 28-31 not needed.

// Also LCDACTL1 not required since not using S32 - S39

// Davie's book page 260

// page 26-23 of MSP430x4xx user manual

LCDAPCTL0 = 0x7E; // set the clock with a divider of 128, scan freq = 256hz

// The LCD uses the ACLK as the master clock as the scan rate for

// the display segments

// The ACLK has been set to 32768 Hz with the external

// 327768 Hz crystal

// Let's use scan frequency of 256 Hz (This is fast enough not

// to see the display flicker)

// or a divisor of 128

// LCDFREQ division(3 bits), LCDMUX (2 bits), LCDSON segments on,

// Not used, LCDON LCD module on

// 011 = freq /128, 11 = 4 mux's needed since the display uses for

// common inputs com0-com3

// need to turn the LCD on LCDON = 1

// LCDSON allows the segments to be blinked good for blinking but

// needs to be on to

// display the LCD segments LCDSON = 1

// Bit pattern required = 0111 1101 = 0x7d

// page 26-22 of MSP430x4xx user manual

LCDACTL = 0x7d; // control signal for 4 multiplexors (2^4 = 16 min you need for 11)

}

**void** **showRightMostTwo\_part1**(**void**){

// At row 0, set the pattern equal to 2

**volatile** **unsigned** **int** i; // volatile to prevent optimization

LCDSeg[0] |= BIT0; // 11111111

LCDSeg[0] |= BIT1;

LCDSeg[0] |= BIT3;

LCDSeg[0] |= BIT6;

LCDSeg[0] |= BIT5;

}

**void** **showLeftMostTwo\_part2**(**void**){

// At row 6, set the pattern equal to 2

**volatile** **unsigned** **int** i; // volatile to prevent optimization

LCDSeg[6] |= BIT0; // 11111111

LCDSeg[6] |= BIT1;

LCDSeg[6] |= BIT3;

LCDSeg[6] |= BIT6;

LCDSeg[6] |= BIT5;

}

**void** **showEights**(**void**){

**volatile** **unsigned** **int** i; // volatile to prevent optimization

**for** (i=0;i<LCD\_SIZE;i++){

LCDSeg[i]=0xff; // 1111111

}

**for** (;;){

P2OUT ^= 0x02; // Toggle P1.0 using exclusive-OR

i = 20000; // SW Delay

**do** i--;

**while** (i != 0);

}

}

**void** **reset**(**int** row){

// turns off all bits at a certain row

LCDSeg[row] = 0x00;

}

**void** **zero**(**int** row){

// resets, and then makes zero pattern

reset(row);

LCDSeg[row] |= BIT0;

LCDSeg[row] |= BIT1;

LCDSeg[row] |= BIT2;

LCDSeg[row] |= BIT3;

LCDSeg[row] |= BIT6;

LCDSeg[row] |= BIT4;

}

**void** **one**(**int** row){

// resets, and then makes one pattern

reset(row);

LCDSeg[row] |= BIT1;

LCDSeg[row] |= BIT2;

}

**void** **two**(**int** row){

// resets, and then makes two pattern

reset(row);

LCDSeg[row] |= BIT0;

LCDSeg[row] |= BIT1;

LCDSeg[row] |= BIT3;

LCDSeg[row] |= BIT6;

LCDSeg[row] |= BIT5;

}

**void** **three**(**int** row){

// resets, and then makes three pattern

reset(row);

LCDSeg[row] |= BIT0;

LCDSeg[row] |= BIT1;

LCDSeg[row] |= BIT3;

LCDSeg[row] |= BIT2;

LCDSeg[row] |= BIT5;

}

**void** **four**(**int** row){

// resets, and then makes four pattern

reset(row);

LCDSeg[row] |= BIT4;

LCDSeg[row] |= BIT5;

LCDSeg[row] |= BIT1;

LCDSeg[row] |= BIT2;

}

**void** **five**(**int** row){

// resets, and then makes five pattern

reset(row);

LCDSeg[row] |= BIT0;

LCDSeg[row] |= BIT4;

LCDSeg[row] |= BIT2;

LCDSeg[row] |= BIT3;

LCDSeg[row] |= BIT5;

}

**void** **six**(**int** row){

// resets, and then makes six pattern

reset(row);

LCDSeg[row] |= BIT0;

LCDSeg[row] |= BIT4;

LCDSeg[row] |= BIT5;

LCDSeg[row] |= BIT6;

LCDSeg[row] |= BIT2;

LCDSeg[row] |= BIT3;

}

**void** **seven**(**int** row){

// resets, and then makes seven pattern

reset(row);

LCDSeg[row] |= BIT0;

LCDSeg[row] |= BIT1;

LCDSeg[row] |= BIT2;

}

**void** **eight**(**int** row){

// resets, and then makes eight pattern

reset(row);

LCDSeg[row] |= BIT0;

LCDSeg[row] |= BIT1;

LCDSeg[row] |= BIT2;

LCDSeg[row] |= BIT3;

LCDSeg[row] |= BIT4;

LCDSeg[row] |= BIT5;

LCDSeg[row] |= BIT6;

}

**void** **nine**(**int** row){

// resets, and then makes nine pattern

reset(row);

LCDSeg[row] |= BIT0;

LCDSeg[row] |= BIT1;

LCDSeg[row] |= BIT2;

LCDSeg[row] |= BIT3;

LCDSeg[row] |= BIT4;

LCDSeg[row] |= BIT5;

}

**void** **showA**(**int** row){

// resets, and then makes A pattern

reset(row);

LCDSeg[row] |= BIT0;

LCDSeg[row] |= BIT1;

LCDSeg[row] |= BIT2;

LCDSeg[row] |= BIT4;

LCDSeg[row] |= BIT5;

LCDSeg[row] |= BIT6;

}

**void** **showB**(**int** row){

// resets, and then makes b pattern

reset(row);

LCDSeg[row] |= BIT2;

LCDSeg[row] |= BIT3;

LCDSeg[row] |= BIT4;

LCDSeg[row] |= BIT5;

LCDSeg[row] |= BIT6;

}

**void** **showC**(**int** row){

// resets, and then makes c pattern

reset(row);

LCDSeg[row] |= BIT3;

LCDSeg[row] |= BIT5;

LCDSeg[row] |= BIT6;

}

**void** **showD**(**int** row){

// resets, and then makes d pattern

reset(row);

LCDSeg[row] |= BIT1;

LCDSeg[row] |= BIT2;

LCDSeg[row] |= BIT3;

LCDSeg[row] |= BIT5;

LCDSeg[row] |= BIT6;

}

**void** **showE**(**int** row){

// resets, and then makes E pattern

reset(row);

LCDSeg[row] |= BIT0;

LCDSeg[row] |= BIT3;

LCDSeg[row] |= BIT4;

LCDSeg[row] |= BIT5;

LCDSeg[row] |= BIT6;

}

**void** **showF**(**int** row){

// resets, and then makes F pattern

reset(row);

LCDSeg[row] |= BIT0;

LCDSeg[row] |= BIT4;

LCDSeg[row] |= BIT5;

LCDSeg[row] |= BIT6;

}

**void** **part3CountToF**(**void**){

**volatile** **int** i = 0;

zero(0);

**for**(i=0;i<20000;i++);

one(0);

**for**(i=0;i<20000;i++);

two(0);

**for**(i=0;i<20000;i++);

three(0);

**for**(i=0;i<20000;i++);

four(0);

**for**(i=0;i<20000;i++);

five(0);

**for**(i=0;i<20000;i++);

six(0);

**for**(i=0;i<20000;i++);

seven(0);

**for**(i=0;i<20000;i++);

eight(0);

**for**(i=0;i<20000;i++);

nine(0);

**for**(i=0;i<20000;i++);

showA(0);

**for**(i=0;i<20000;i++);

showB(0);

**for**(i=0;i<20000;i++);

showC(0);

**for**(i=0;i<20000;i++);

showD(0);

**for**(i=0;i<20000;i++);

showE(0);

**for**(i=0;i<20000;i++);

showF(0);

**for**(i=0;i<20000;i++);

part3CountToF();

}

**void** **displayBasedOnNum**(**int** row, **int** num){

// displays a number based on integer

**if**(num == 0){

zero(row);

}

**else** **if**(num == 1){

one(row);

}

**else** **if**(num == 2){

two(row);

}

**else** **if**(num == 3){

three(row);

}

**else** **if**(num == 4){

four(row);

}

**else** **if**(num == 5){

five(row);

}

**else** **if**(num == 6){

six(row);

}

**else** **if**(num == 7){

seven(row);

}

**else** **if**(num == 8){

eight(row);

}

**else** **if**(num == 9){

nine(row);

}

**volatile** **int** i = 0;

**for**(i=0;i<20000;i++);

}

**void** **part4Count**(**void**){

**int** count = 0;

**while**(1==1){

**if**(P1IN == BIT0){

// count up

count++;

}

**else** **if**(P1IN == BIT1){

// count down

count--;

}

**if**(count < 10){

// reset rows 1 and 2, display the count

displayBasedOnNum(0, count);

reset(1);

reset(2);

}

**else** **if**(count >= 10 && count < 100){

// reset row 2, display row 1 and row 0 for tens and ones place

reset(2);

displayBasedOnNum(1, (**int**)(count/10));

displayBasedOnNum(0, count%10); //11

}

**else** **if**(count > 100){ // 317

**int** remain100 = count%100; // 317 %100 = 17 // 317 / 100 = 3 // 317 / 10 = 31 317%10 = 7

**int** remain10 = count%10;

displayBasedOnNum(2, (**int**)(count/100));

displayBasedOnNum(1, (**int**)((count%100) - count%10)/10);

displayBasedOnNum(0, (**int**)(count%10));

}

// keep counting at max and min

**if**(count == 1000){

count = 0;

}

**if**(count == -1){

count = 999;

}

}

}

**Complete Assembly Code:**

; kelsey cameron

.cdecls C,LIST,"msp430fg4618.h" ; cdecls tells assembler

; to allow

; the device header file

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

; #LCDM3 is the start of the area of memory of the TMS430 that points

; to the segments

; of the softbaugh LCD LCDM3 = the starting address

; each of the seven segments for each display is store in memory

; starting at address LCDM3

; which is the right most seven segment of the LCD

; The bit order in each byte is

; dp, E, G, F, D, C, B, A or

; :, E, G, F, D, C, B, A

; after the seven segments these memory locations are used to turn on

; the special characters

; such as battery status, antenna, f1-f4, etc.

; there are 7 seven segment displays

; data area ram starts 0x1100

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

; the .sect directives are defined in lnk\_msp430f4618.cmd

; .sect ".stack" ; data ram for the stack

; .sect ".const" ; data rom for initialized data

; constants

; .sect ".text" ; program rom for code

; .sect ".cinit" ; program rom for global inits

; .sect ".reset" ; MSP430 RESET Vector

**.sect** ".sysmem" ; data ram for initialized

; variables

; there are 11 locations that are needed for the softbaugh LCD

; only 7 used for the seven segment displays

LCD\_SIZE **.byte** 11 ; eleven bytes needed by the LCD

; This is the code area

; flash begins at address 0x3100

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

; Main Code

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

**.text** ; program start

**.global** \_START ; define entry point

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

**START** **mov.w** #300h,SP ; Initialize 'x1121

; stackpointer

StopWDT **mov.w** #WDTPW+WDTHOLD,&WDTCTL ; Stop WDT

SetupP1 **bis.b** #04h,&P2DIR ; P2.2 output

; go initialize the LCD Display

main

**call** #Init\_LCD

**mov.b** #000b, &P1DIR

**mov.b** #0000000b, R7

**mov.w** #LCDM3, R5

; Throughout the entire program, R5 will be used to store the index of where the number should be displayed

; R13 will store the number you want to display

; R14 will be used as the offset

; clears whatevers in there.

; call #part1

; call #part2

; call #part3

;;;;;;;;;;;;;;;;;; uncomment all of this to test part 4

; mov.w #0x0, R14

; mov.w #0x0, R11

; mov.w #0x0, R13

; mov.w #0x0, R12

; add.b R14, R5

; call #zero

; sub.b R14, R5

; mov.w #0x01, R14

; add.b R14, R5

; call #zero

; sub.b R14, R5

; add.b #0x01, R14

; add.b R14, R5

; call #zero

; sub.b R14, R5

; mov.w #0x00, R14

;call #part4

**jmp** main

part1

**mov.w** #LCDM3, R5 ; LCDM3 is the base address of the LCD display

**add.b** #0, R5

**call** #two

**call** #delay

**ret**

part2

**mov.w** #LCDM3, R5

**add.b** #6, R5

**call** #two

**jmp** part2

part3

**mov.w** #LCDM3, R5

**add.b** #0, R5

**call** #zero

**call** #delay

**call** #one

**call** #delay

**call** #two

**call** #delay

**call** #three

**call** #delay

**call** #four

**call** #delay

**call** #five

**call** #delay

**call** #six

**call** #delay

**call** #seven

**call** #delay

**call** #eight

**call** #delay

**call** #nine

**call** #delay

**call** #showA

**call** #delay

**call** #showB

**call** #delay

**call** #showC

**call** #delay

**call** #showD

**call** #delay

**call** #showE

**call** #delay

**call** #showF

**call** #delay

**jmp** part3

; R11 will store the counter

part4 ; 10

; 1.0 and 1.2 are the buttons

**cmp.b** #0x02, &P1IN

**jeq** firstison

**jmp** checkSec

; checkBoth, give precedence to counting down, since it doesn't matter

checkSec

**cmp.b** #0x01, &P1IN

**jeq** secondison

**jmp** after

; left button s2b - if button is pressed, count up

firstison

**add.b** #0x01, R11

**jmp** after

; right button s1 - if button is pressed, count down

secondison

**cmp** #0x00, R12

**jne** checkZeroPlace

**jmp** dontrid

checkZeroPlace

**cmp** #0x0, R11

**jeq** getRidTens

**jmp** dontrid

getRidTens

**sub.b** #0x01, R12

**mov.b** #0xA, R11

dontrid

**cmp** #0x00, R13

**jne** checkTensPlace

**jmp** dontridHund

checkTensPlace

**cmp** #0x0, R12

**jeq** checkOnesPlace

**jmp** dontridHund

checkOnesPlace

**cmp** #0x0, R11

**jeq** getRidHund

**jmp** dontridHund

getRidHund

**sub.b** #0x01, R13

**mov.b** #0x9, R12

**mov.b** #0xA, R11

dontridHund

**cmp** #0x00, R11

**jeq** testNextDigit

**jmp** dontset

testNextDigit

**cmp** #0x00, R12

**jeq** testFinalDigit

**jmp** dontset

testFinalDigit

**cmp** #0x00, R13

**jeq** setto999

**jmp** dontset

setto999

**mov.b** #0x09, R11

**mov.b** #0x09, R12

**mov.b** #0x09, R13

**jmp** after

dontset

**sub.b** #0x01, R11

**jmp** after

after

**cmp** #0x3E8, R11

**jeq** resetCountToZero

**jmp** after2

; if count == 1000, count = 0

resetCountToZero

**mov.b** #0x00, R11

**jmp** after2

after2

; display what is in R11, the counter

**mov.b** #0x00, R14

**call** #displayBasedOnR11

**add.b** #0x01, R14

**mov.b** R11, R15

**mov.b** R12, R11

**add.b** #0x01, R5

**call** #displayBasedOnR11

**sub.b** #0x01, R5

**add.b** #0x02, R5

**mov.b** R13, R11

**add.b** #0x02, R14

**call** #displayBasedOnR11

**sub.b** #0x02, R5

**mov.b** #0x00, R14

**mov.b** R15, R11

; loop infinitely

**jmp** part4

; R12 will store the 10's place val, R13 will store the 100's place val

displayBasedOnR11

**cmp** #0x0A, R11

**jge** incrementTens

**jmp** dontinc

incrementTens

**add.b** #0x01, R12

**mov.b** #0x00, R11

dontinc

**cmp** #0x0A, R12

**jge** incrementHund

**jmp** dontincHun

incrementHund

**add.b** #0x01, R13

**mov.b** #0x00, R12

**mov.b** #0x00, R11

dontincHun

**cmp** #0x0A, R13

**jge** setAllZero

**jmp** continue

setAllZero

**mov.b** #0x00, R12

**mov.b** #0x00, R11

**mov.b** #0x00, R13

continue

**call** #delay

**ret**

printStoredNum

**cmp** #0x0, R11

**jeq** printZero

**jmp** checkNext0

printZero

**call** #zero

checkNext0

**cmp** #0x01, R11

**jeq** print1

**jmp** checkNext1

print1

**call** #one

checkNext1

**cmp** #0x02, R11

**jeq** printTwo

**jmp** checkNext2

printTwo

**call** #two

checkNext2

**cmp** #0x03, R11

**jeq** printThree

**jmp** checkNext3

printThree

**call** #three

checkNext3

**cmp** #0x04, R11

**jeq** printFour

**jmp** checkNext4

printFour

**call** #four

checkNext4

**cmp** #0x05, R11

**jeq** printFive

**jmp** checkNext6

printFive

**call** #five

checkNext6

**cmp** #0x06, R11

**jeq** print6

**jmp** checkNext7

print6

**call** #six

checkNext7

**cmp** #0x07, R11

**jeq** print7

**jmp** checkNext8

print7

**call** #seven

checkNext8

**cmp** #0x08, R11

**jeq** print8

**jmp** checkNext9

print8

**call** #eight

checkNext9

**cmp** #0x09, R11

**jeq** print9

**jmp** end

print9

**call** #nine

end **ret**

zero

**mov.b** #0000000b, R7

**bis.b** #1011111b, R7 ; bit 0,1,2,3,4,6

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

one

**mov.b** #0000000b, R7

**bis.b** #00000110b, R7 ; bit 12

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

two

**mov.b** #0000000b, R7

**bis.b** #1101011b, R7 ; bit 0,1,3,5,6

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

three

**mov.b** #0000000b, R7

**bis.b** #0101111b, R7 ; bit 0,1235

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

four

**mov.b** #0000000b, R7

**bis.b** #0110110b, R7 ; bit 1245

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

five

**mov.b** #0000000b, R7

**bis.b** #0111101b, R7 ; bit 02345

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

six

**mov.b** #0000000b, R7

**bis.b** #1111101b, R7 ; bit 023456

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

seven

**mov.b** #0000000b, R7

**bis.b** #0000111b, R7 ; bit 023456

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

eight

**mov.b** #0000000b, R7

**bis.b** #1111111b, R7 ; bit all

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

nine

**mov.b** #0000000b, R7

**bis.b** #0111111b, R7

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

showA

**mov.b** #0000000b, R7

**bis.b** #1110111b, R7

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

showB

**mov.b** #0000000b, R7

**bis.b** #1111100b, R7 ; bit all

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

showC

**mov.b** #0000000b, R7

**bis.b** #1101000b, R7 ; bit all

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

showD

**mov.b** #0000000b, R7

**bis.b** #1101110b, R7 ; bit all

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

showE

**mov.b** #0000000b, R7

**bis.b** #1111001b, R7 ; bit all

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

showF

**mov.b** #0000000b, R7

**bis.b** #1110001b, R7 ; bit all

**mov.b** R7, 0(R5) ; R7 stores the LCD values

**ret**

clearLCD

**mov.b** #0x00, R6

; R5 points to the beginning memory for the LCD

**mov.w** #LCDM3, R5

; move 0 into R7 to clear the LCD memory

**mov.b** #0x00, R7

lpt2 **mov.b** R7, 0(R5)

; Increment R5 to point to the next seven segment display

; Increment R6 for the next count in the loop

**inc.w** R5

**inc.b** R6

; See if the loop is finished

**cmp.b** LCD\_SIZE, R6

**jnz** lpt2

**ret**

delay

**mov** #0xff00, R10

L77 **dec.w** R10

**jnz** L77

**ret**

showEight

; LCD\_SIZE-4 only gives the 7 segment displays plus DP, and

; colons (colons = dp)

; Right most display is at LCDSeg[0];

; R6 is a loop counter to cover all of the segments. This count

; counts up from 0

**mov.b** #0x00, R6

; R5 points to the beginning memory for the LCD

; Turn on all of the segments

; LCD\_SIZE-4 only gives the 7 segment displays plus DP, and

; colons: colons = dp

; Right most display is at LCDSeg[0];

; To turn on a segment of the LCD a one is written in the

; the appropriate location in the LCD memory

; Setting all the bits to 1 for all memory locations turns on

; all of the display elements

; including all special characters

**mov.w** #LCDM3, R5

; move 0xff into R7 to turn on all LCD segments the LCD memory

**mov.b** #0xFF, R7

lpt1 **mov.b** R7, 0(R5)

; Increment R5 to point to the next seven segment display

; Increment R6 for the next count inthe loop

**inc.w** R5

**inc.b** R6

; See if the loop is finished / finish writng to the last display

; element

**cmp.b** LCD\_SIZE, R6

**jnz** lpt1

; Blink the green LED to make sure the code is running

;

Mainloop **xor.b** #04h,&P2OUT ; Toggle P2.2

Wait **mov.w** #0A000h,R15 ; Delay to R15

L1 **dec.w** R15 ; Decrement R15 jnz L1 ; Delay over?

**jmp** showEight ; Again

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

; Initialize the LCD system

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

Init\_LCD

; Using the LCD A controller for the MSP430fg4618

; the pins of the LCD are memory mapped onto the mp430F4xxx

; memory bus and

; are accessed via LCDSeg[i] array

; See page 260 of Davie's text

; LCD\_SIZE-4 only gives the 7 segment displays plus DP, and

; (colons are the same bit setting)

; LCD\_SIZE-4 only gives the 7 segment displays plus DP, and

; colons: colons / dp

; Right most seven segment display is at LCDSeg[0];

; Display format

; AAA

; F B

; X F B

; GGG

; X E C

; E C

; DP DDD

; bit order

; dp, E, G, F, D, C, B, A or

; :, E, G, F, D, C, B, A

; initialize the segment memory to zero to clear the LCD

; writing a zero in the LCD memory location clears turns off

; the LCD segment

; R6 is a loop counter to cover all of the segments

; including all special characters

**mov.b** #0x00, R6

; R5 points to the beginning memory for the LCD

**mov.w** #LCDM3, R5

; move 0 into R7 to clear the LCD memory

**mov.b** #0x00, R7

lpt **mov.b** R7, 0(R5)

; Increment R5 to point to the next seven segment display

; Increment R6 for the next count in the loop

**inc.w** R5

**inc.b** R6

; See if the loop is finished

**cmp.b** LCD\_SIZE, R6

**jnz** lpt

; Port 5 ports 5.2-5.4 are connected to com1,com2,com3 of LCD

; com0 fixed and already assigned

; Need to assign com1 - com3 to port5

; BIT4 | BIT3 |BIT2 = 1 P5.4, P.3, P5.2 = 1

**mov.b** #0x1C, &P5SEL

; Used the internal voltage for the LCD bit 4 = 0 (VLCDEXT=0)

; internal bias voltage set to 1/3 of Vcc, charge pump

; disabled,

; page 26-25 of MSP430x4xx user manual

**mov.b** #0x00, &LCDAVCTL0

; LCDS28-LCDS0 pins LCDS0 = lsb and LCDS28 = MSB need

; LCDS4 through LCDS24

; from the experimenter board schematic the LCD uses S4-S24,

; S0-S3 are not used here

; Only use up to S24 on the LCD 28-31 not needed.

; Also LCDACTL1 not required since not using S32 - S39

; Davie's book page 260

; page 26-23 of MSP430x4xx user manual

**mov.b** #0x7E, &LCDAPCTL0

; The LCD uses the ACLK as the master clock as the scan

; rate for the display segments

; The ACLK has been set to 32768 Hz with the external 327768 Hz

; crystal

; Let's use scan frequency of 256 Hz (This is fast enough not

; to see the display flicker)

; or a divisor of 128

; LCDFREQ division(3 bits), LCDMUX (2 bits), LCDSON segments

; on, Not used, LCDON LCD module on

; 011 = freq /128, 11 = 4 mux's needed since the display uses

; for common inputs com0-com3

; need to turn the LCD on LCDON = 1

; LCDSON allows the segments to be blanked good for blinking

; but needs to be on to

; display the LCD segments LCDSON = 1

; Bit pattern required = 0111 1101 = 0x7d

; page 26-22 of MSP430x4xx user manual

**mov.b** #0x7d, &LCDACTL

**ret**

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

; Interrupt Vectors

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

**.sect** ".reset" ; MSP430 RESET Vector

**.short** **START** ;

.end

**Conclusion:**

In conclusion, I learned how easy it is to overcomplicate things in this lab, and I also now understand the drawbacks of using too many functions in assembly. In retrospect, I would not have hardcoded the value of R5. I also would’ve incremented the tens digits and 100 digits in C, rather than using modulus division.