**EEL 4742 LABORATORY EXPERIMENT # 3**

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

**Name:** Kelsey Cameron

**Introduction:**

The purpose of this lab was to understand how to convert from ASCII to both other ASCII values and other numbers, and to integrate this into assembly. This lab also taught us how to program in assembly step by step, like how certain functions must be repeated within a loop. Also, we learned how to sort ASCII values and use conditional statements within assembly, as well as manipulating arrays. This assignment was quite extensive and covered more concepts than any other lab thus far, therefore the code below will be more helpful in understanding than any description I could provide.

**Complete C Code:**

/\*

\* main.c

\*/

**#include** "msp430fg4618.h"

// uncomment this to submit

// comment this to submit

//#include "msp430f5529.h"

**#include** "stdio.h"

**void** **Init\_UART**(**void**);

**void** **printOpposite**(**char** c);

**void** **OUTA\_UART**(**unsigned** **char** A);

**unsigned** **char** **getASCIIValue**(**char** c);

**unsigned** **char** **getHexVal**(**char** c);

**void** **part4**();

**void** **part5**();

**unsigned** **char** **INCHAR\_UART**(**void**);

**void** **printHexValue**(**char** c);

**void** **printUART**(**char**\* firstletter);

// prints one character at a time via the UART from a string of characters to

//the hyperterminal display. This function should continue printing characters until the

//null character is reached (0x00) indicating an end of a string.

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

WDTCTL = WDTPW + WDTHOLD; // passes the locking password to the watchdog and then holds it.

Init\_UART();

**for**(;;){// 110 sets both P2 bit 1 and P2 bit 2 to an output

// Part 1 - uncomment to test ; kelsey cameron

/\*

volatile unsigned char first = INCHAR\_UART();

volatile unsigned char second = INCHAR\_UART();

first = getASCIIValue(first);

second = getASCIIValue(second);

first = first <<4;

OUTA\_UART(first+second);

OUTA\_UART(0x0A);

OUTA\_UART(0x0D);

\*/

// Part 2 - uncomment to test

//volatile unsigned char c = INCHAR\_UART();

//printHexValue(c);

// Part 3 - uncomment to test

//printOpposite(INCHAR\_UART());

//OUTA\_UART(0x0A);

//OUTA\_UART(0x0D);

// Part 4 - uncomment to test kelsey cameron

//part4();

// Part 5 - uncomment to test

//part5();

}

}

**void** **part4**(){

**int** b=0;

**int** first = 0; // stores the first asci code

**int** y = 0;

**char** a; // stores the binary char

**int** second = 0; // stores the second asci code

**for**(y=0;y<8;y++){

a = INCHAR\_UART();

OUTA\_UART(a);

a=a-0x30; // subtracts 30 to convert it to a 1 or 0

a=a<<(7-y); // shifts the first over by 7, the second over by 6, etc

b=b|a; // ors with b to make sure the binary is in the correct order

}

OUTA\_UART(0x0A); //newline

OUTA\_UART(0x0D);

first= b & 0xF0; // look at only the first four bits

second= b & 0x0F; // look at only the last four bits

first=first>>4; // shift first by 4 right so the extra zero goes away

// converts the actual letters and numbers to asci letters and numbers

**if**(first>=0xA && first <=0xF)

{

first+=0x37;

}

**else** **if** (first>=0x0 && first<=0x9)

{

first+=0x30;

}

**if**(second>=0xA && second<=0xF)

{

second+=0x37;

}

**else** **if** (second>=0x0 && second<=0x9)

{

second+=0x30;

}

// prints

OUTA\_UART(first);

OUTA\_UART(second);

OUTA\_UART(0x0A); //newline

OUTA\_UART(0x0D);

}

**void** **printOpposite**(**char** c){

**if**(c >= 0x41 && c <= 0x5A){

c+=0x20;

}

**else** **if**(c >= 0x61 && c <= 0x7A){

c-=0x20;

}

OUTA\_UART(c);

}

**void** **printHexValue**(**char** c){

**char** second = c & 0x0F; //0x15 = 1111

**char** first = c;

first = c & 0xF0;

first = first >> 4;

OUTA\_UART(getHexVal(first));

OUTA\_UART(getHexVal(second));

}

// 3F = 3\*16 + F

**unsigned** **char** **getASCIIValue**(**char** c){

**if**(c >= 0x30 && c <= 0x39){

c-=0x30;

}

**else** **if**(c >= 0x41 && c <= 0x46){

c-=0x37;

}

**return** c;

}

**unsigned** **char** **getHexVal**(**char** c){

**if**(c >= 0x00 && c <= 0x09){

c+=0x30;

}

**else** **if**(c >= 0x0A && c <= 0x0F){

c+=0x37;

}

**return** c;

}

**void** **part5**(){

**char** array[33];

**int** num = 0;

**int** x = 0;

**for**(x = 0; x < 32; x++){

**char** c = INCHAR\_UART();

OUTA\_UART(c);

**if**((c == '\0') || (c == 0x0D) || (c==0x0A) || (**int**)(c==10)){

**break**;

}

num++;

array[x] = c;

}

OUTA\_UART(0x0D);

**int** i, j, temp;

**for**(i = 0; i < num; i++){

**for**(j = 0; j<num; j++){

**int** num1 = array[i];

**int** num2 = array[j];

**if**(num1 < num2){

temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

}

**for**(i = 0; i < num; i++){

**if**(array[i]==0x0D || array[i]==0x0A || array[i]=='\0'){

**break**;

}

OUTA\_UART(array[i]);

}

}

**void** **printUART**(**char** \* firstletter){

**while**(\*firstletter!='\0'){

OUTA\_UART(\*firstletter);

firstletter++;

}

OUTA\_UART(0x0D);

}

// 1.0 SW1 1.2 SW2

// remember that 0 means button is pressed.

// This code below is for the lab chip. Uncomment it before submitting, and delete your own chip uart code

**void** **OUTA\_UART**(**unsigned** **char** A){

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

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

// IFG2 register (1) = 1 transmit buffer is empty,

// UCA0TXBUF 8 bit transmit buffer

// wait for the transmit buffer to be empty before sending the

// data out

**do**{

}**while** ((IFG2&0x02)==0);

// send the data to the transmit buffer

UCA0TXBUF =A;

}

// while bit 2 of the IFG2 register is set to zero,

// this is a transmitter flag that sees if the transmitter is free to transfer or not.

// wait for the transmission to be complete. This flag = 1 when bits are on the output lines.

// send the data to the transmit buffer

// so this takes the data from A, and sends it

// the buffer that stores data. This UCA0TXBUF is a variable

// that stores the address of a buffer register on the chip.

// this data transfer is hard wired, and data from A is placed on the

// output lines that are placed into the register.

// basically UART works by shifting the bits right, and the rightmost bit is sent to the reciever

**unsigned** **char** **INCHAR\_UART**(**void**){

**do**{

}**while**((IFG2&0x01) == 0); // IFG2 = Interrupt FlaG Second Register

// First bit of IFG2 recieves, and the second bit transfers.

**return** (UCA0RXBUF);

}

**void** **Init\_UART**(**void**){

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

// Initialization code to set up the uart on the experimenter

// board to 8 data,

// 1 stop, no parity, and 9600 baud, polling operation

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

P2SEL=0x30; // transmit and receive to port 2 bits 4 and 5

// sets Bits p2.4 transmit and p2.5 receive, similar to how we set bits as outputs

// or inputs

// 30 = 110000

UCA0CTL0=0; // 8 bits of data, no parity 1 stop, uart, async

// (7)=1 (parity), (6)=1 Even, (5)= 0 lsb first,

// (4)= 0 8 data / 1 7 data,

// (3) 0 1 stop 1 / 2 stop, (2-1) -- UART mode,

// (0) 0 = async

// this basically initializes the flag.

UCA0CTL1= 0x41;

// select ALK 32768 and put in

// software reset the UART

// (7-6) 00 UCLK, 01 ACLK (32768 hz), 10 SMCLK,

// 11 SMCLK

// (0) = 1 reset

UCA0BR1=0; // upper byte of divider clock word

UCA0BR0=3; // clock divide from a clock to bit clock 32768/9600

// = 3.413

// UCA0BR1:UCA0BR0 two 8 bit reg to from 16 bit

// clock divider

// for the baud rate

UCA0MCTL=0x06;

// low frequency mode module 3 modulation pater

// used for the bit clock

UCA0STAT=0; // do not loop the transmitter back to the

// receiver for echoing

// (7) = 1 echo back trans to rec

// (6) = 1 framing, (5) = 1 overrun, (4) =1 Parity,

// (3) = 1 break

// (0) = 2 transmitting or receiving data

UCA0CTL1=0x40;

// take UART out of reset

IE2=0; // turn transmit interrupts off

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

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

// IFG2 register (0) = 1 receiver buffer is full,

// UCA0RXIFG

// IFG2 register (1) = 1 transmit buffer is empty,

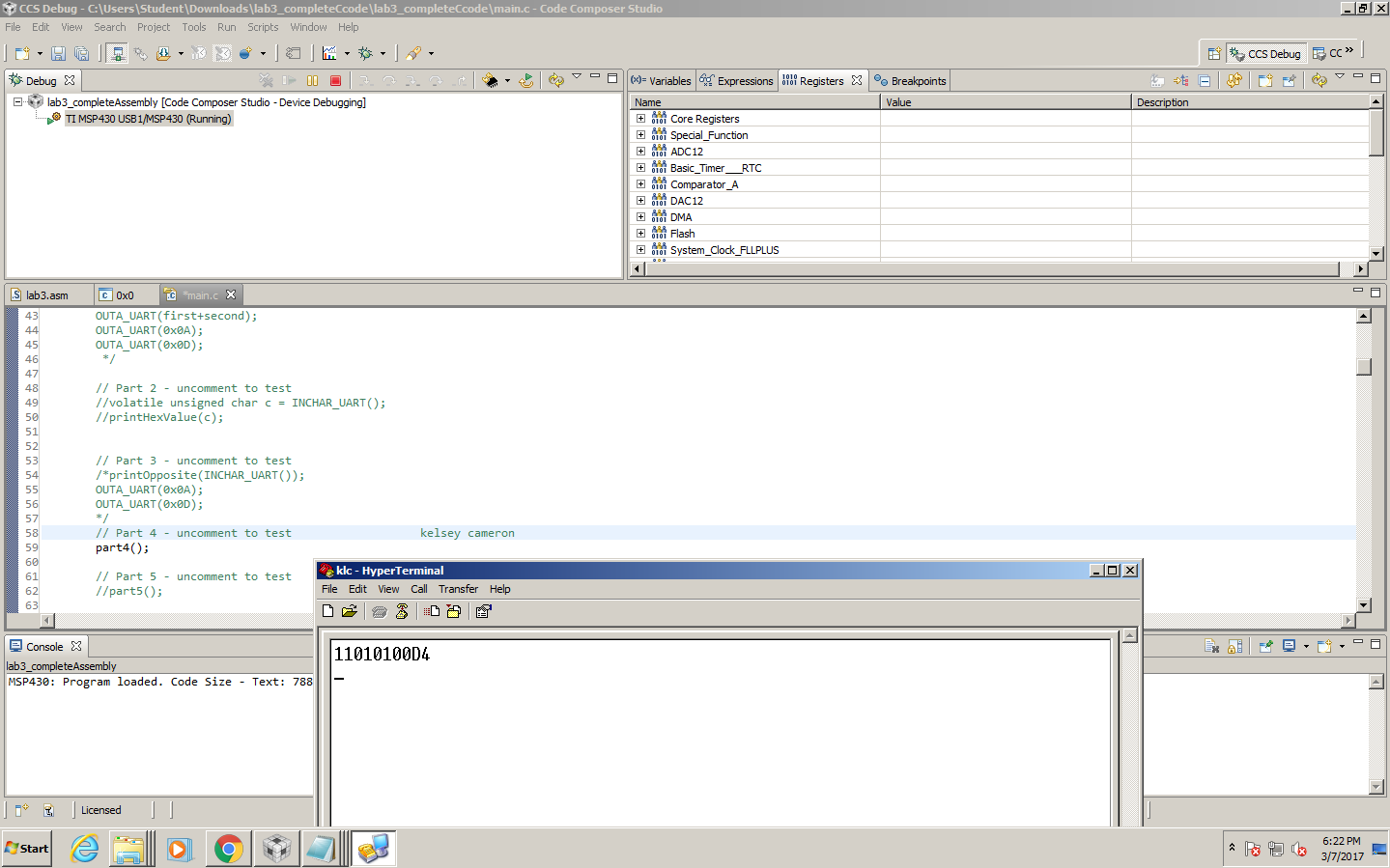
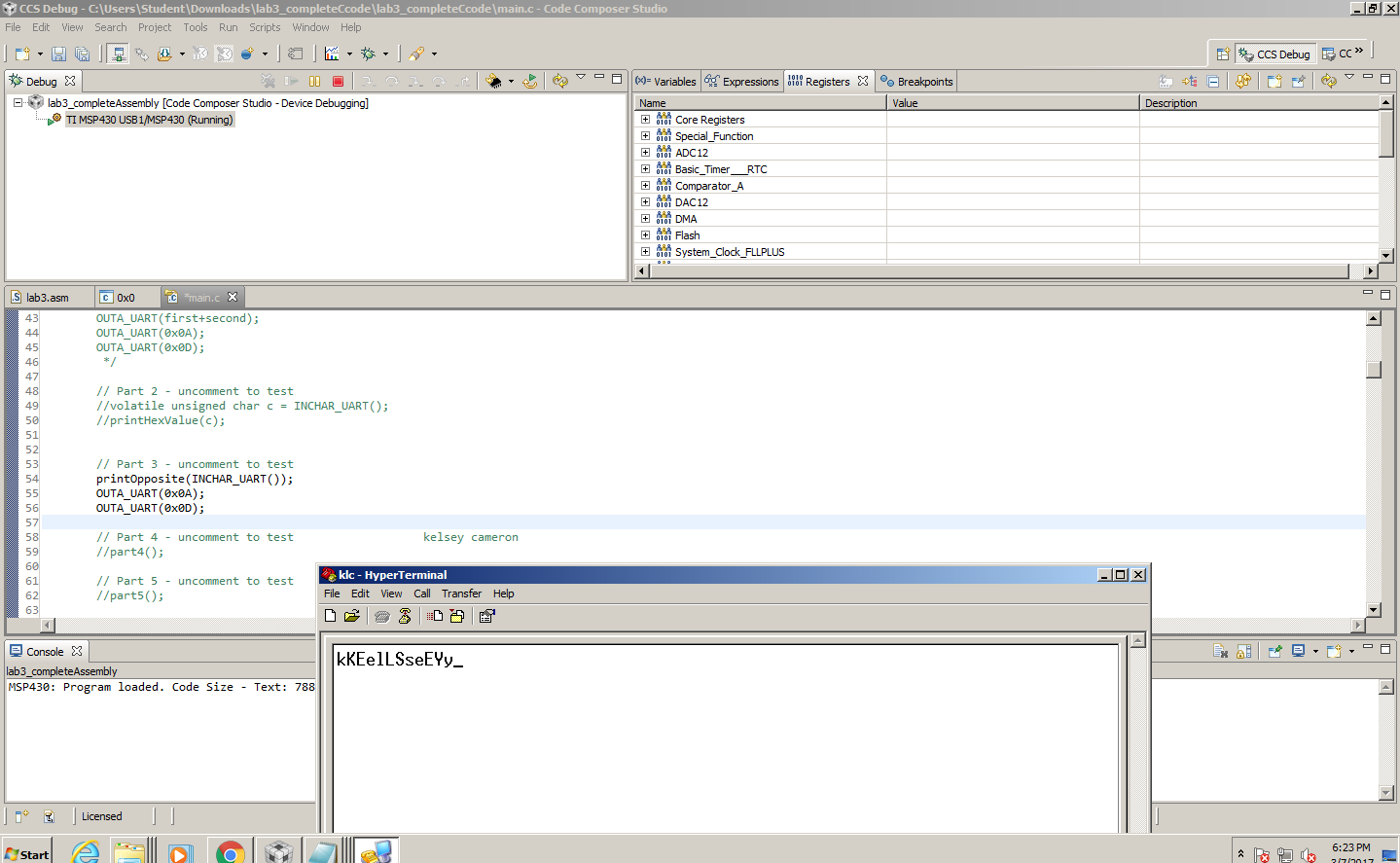
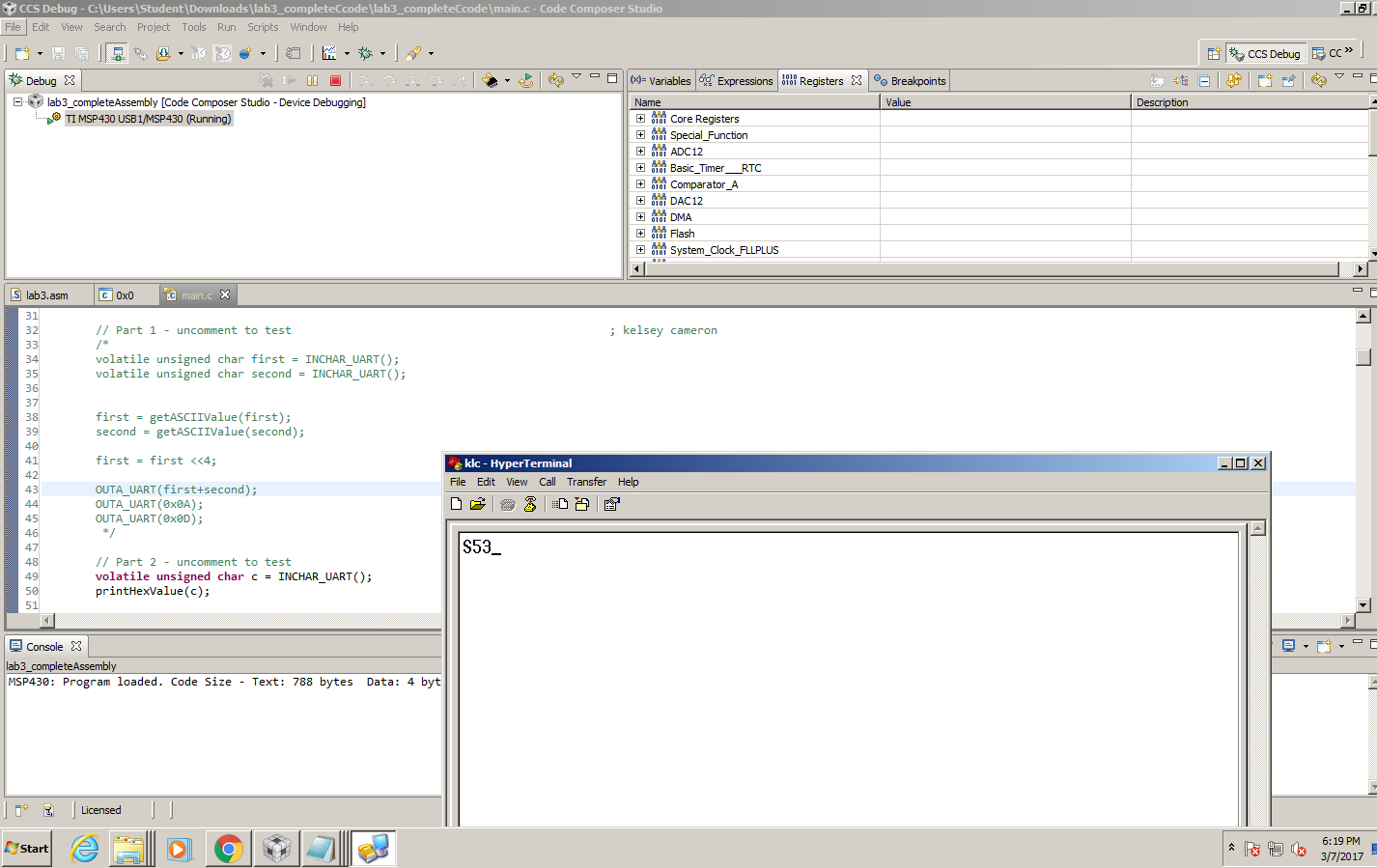
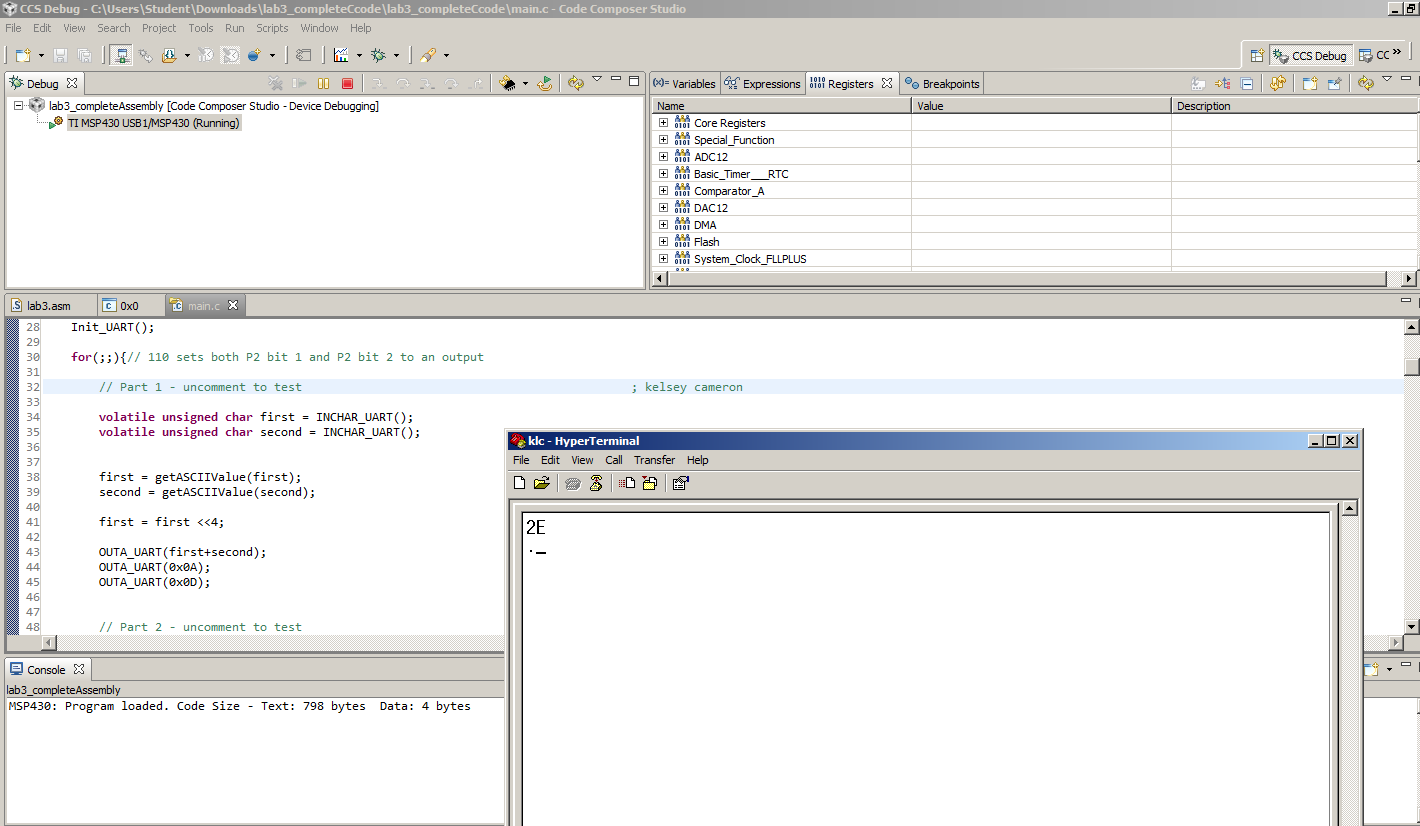
// UCA0RXIFG

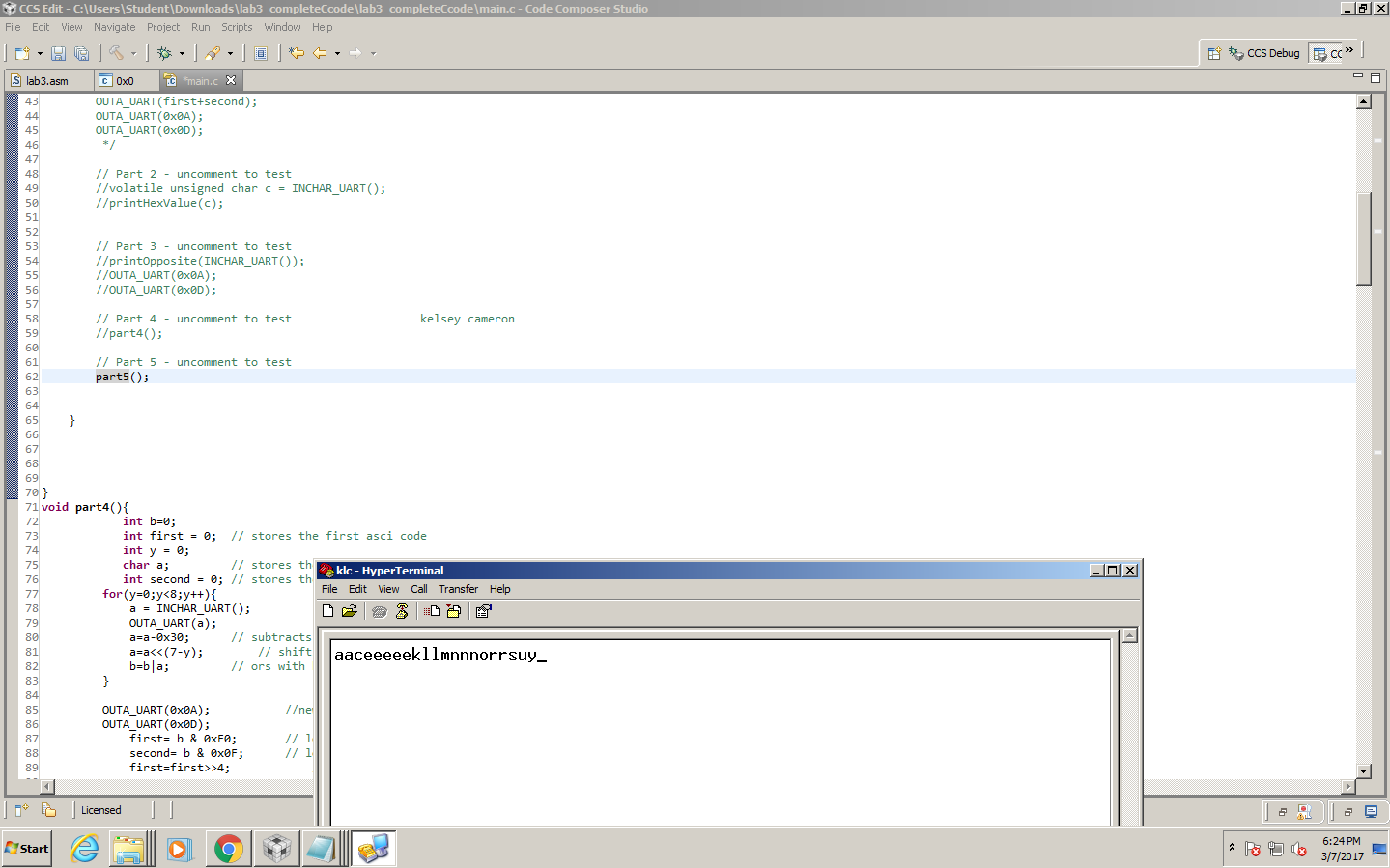
// UCA0RXBUF 8 bit receiver buffer

// UCA0TXBUF 8 bit transmit buffer

}

**C Code Screenshots:**





**Complete Assembly Code:**

; final assembly code for lab #3 kelsey cameron

.cdecls C,LIST,"msp430fg4618.h"

**.sect** ".const"

.bss label, 4

**.word** 0x1234

**.text**

**.global** \_START

.bss myarray, 32

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

**START** **mov.w** #300h,SP

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

; uncomment the lab stuff above this line

; delete below this line for lab

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

; MSP430 Assembler Code Template for use with TI Code Composer Studio

;

;

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

; .cdecls C,LIST,"msp430f5529.h" ; Include device header file

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

; .def RESET ; Export program entry-point to

; make it known to linker.

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

; .text ; Assemble into program memory.

; .retain ; Override ELF conditional linking

; and retain current section.

; .retainrefs ; And retain any sections that have

; references to current section.

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

;RESET mov.w #\_\_STACK\_END,SP ; Initialize stackpointer

;StopWDT mov.w #WDTPW|WDTHOLD,&WDTCTL ; Stop watchdog timer

;;;;;;;;;;;;;;;;;;;

**call** #Init\_UART

Mainloop

;call #part1 ; kelsey cameron

;call #part2

;call #part3

;call #part4

**call** #part5

EXIT **jmp** Mainloop

part1

**call** #INCHAR\_UART

**call** #OUTA\_UART

**mov.b** R4, R7 ; R7 will represent first

**call** #INCHAR\_UART

**call** #OUTA\_UART

**mov.b** R4, R8 ; R7 will represent second

; call #OUTA\_UART

convertFirst **cmp.b** #0x30, R7

**jge** next11

next11 **cmp.b** #0x39, R7

**jl** addop1

**cmp.b** #0x39, R7

**jeq** addop1

**jmp** next21

addop1 **sub.b** #0x30, R7

**jmp** cont

next21 **cmp.b** #0x41, R7

**jge** next31

next31 **cmp.b** #0x46, R7

**jl** subtract1

next51 **cmp.b** #0x46, R7

**jeq** subtract1

subtract1 **sub.b** #0x37, R7

**jmp** cont

cont

convertSec **cmp.b** #0x30, R8

**jge** next112

next112 **cmp.b** #0x39, R8

**jl** addop12

**cmp.b** #0x39, R8

**jeq** addop12

**jmp** next212

addop12 **sub.b** #0x30, R8

**jmp** cont2

next212 **cmp.b** #0x41, R8

**jge** next312

next312 **cmp.b** #0x46, R8

**jl** subtract12

next512 **cmp.b** #0x46, R8

**jeq** subtract12

subtract12 **sub.b** #0x37, R8

**jmp** cont2

cont2

**rla** R7

**rla** R7

**rla** R7

**rla** R7

**mov.b** #0x0D, R4

**call** #OUTA\_UART

**mov.b** #0x0A, R4

**call** #OUTA\_UART

**add.b** R7, R8

**mov.b** R8, R4

**call** #OUTA\_UART

Exit1 **ret**

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; input S output 53

part2

;char second = c & 0x0F; //0x15 = 1111

; char first = c;

;first = c & 0xF0;

;first = first >> 4;

;OUTA\_UART(getHexVal(first));

;OUTA\_UART(getHexVal(second));

**call** #INCHAR\_UART

**call** #OUTA\_UART

**mov.b** R4, R7

**and.b** #0xF0, R7

**mov.b** R4, R8

**and.b** #0x0F, R8

**rra** R7

**rra** R7

**rra** R7

**rra** R7

checkLetter **cmp.b** #0x0A, R7

**jge** letter1

**jmp** checkNum

letter1 **cmp.b** #0x0F, R7

**jge** EXIT3

**jmp** letter2

letter2 **add.b** #0x37, R7

**jmp** shiftb

checkNum **cmp.b** #0x0, R7

**jge** number1

**jmp** EXIT3

number1 **cmp.b** #0x9, R7

**jge** Mainloop

**jmp** number2

number2 **add.b** #0x30,R7

**jmp** shiftb

shiftb **mov.b** R7, R4

;call #OUTA\_UART

NEXT **cmp.b** #0x0A, R8

**jge** letter3

**jmp** checkNum2

letter3 **cmp.b** #0x0F, R8

**jge** EXIT3

**jmp** letter4

letter4 **add.b** #0x37, R8

**jmp** PRINT

checkNum2 **cmp.b** #0x0, R8

**jge** number3

**jmp** EXIT3

number3 **cmp.b** #0x9, R8

**jge** EXIT3

**jmp** number4

number4 **add.b** #0x30,R8

**jmp** PRINT

PRINT **mov.b** R7, R4

**call** #OUTA\_UART

**mov.b** R8, R4

**call** #OUTA\_UART

EXIT3 **ret**

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

part3

**call** #INCHAR\_UART

**call** #OUTA\_UART

CHECKUPPER **cmp.b** #0x41, R4

**jge** next1

next1 **cmp.b** #0x5A, R4

**jl** addop

**cmp.b** #0x5A, R4

**jeq** addop

**jmp** next2

addop **add.b** #0x20, R4

**call** #OUTA\_UART

**jmp** Exit

next2 **cmp.b** #0x61, R4

**jge** next3

next3 **cmp.b** #0x7A, R4

**jl** subtract

next5 **cmp.b** #0x7A, R4

**jeq** subtract

subtract **sub.b** #0x20, R4

**call** #OUTA\_UART

**jmp** Exit

Exit **ret**

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; part4

part4

**mov.b** #0x0, R10 ; the counter for the loop

**mov.b** #0x07, R11 ; shift amount for each character

**mov.b** #0x0, R12 ; R12 stores the binary number

**mov.b** #0x0, R4

**mov.b** #0x0, R13

**mov.b** #0x0, R15

loop

**cmp.b** #0x08, R15

**jge** exitloop

**call** #INCHAR\_UART

**mov.b** R4, R7 ; R7 now stores each character

**call** #OUTA\_UART ; echo back what you recieve

**sub** #0x30, R7 ; subtract 30 from R7 to change ascii value into 1 or 0

; R7 contains number that you need to shift

**mov.b** #0x0, R10

shift

**cmp.b** R11, R10 ; compare counter to shift amount

**jge** stopshift ; stop shifting if R10 = R11

**rla** R7, #0x01 ; shift the 1 or zero over based on how many times you've looped

**add** #0x01, R10 ; increment counter by 1

**jmp** shift

stopshift

**add** R7, R12 ; add the 1 or zero to whatever was previously stored in R12

**add** #0x01, R15

**sub** #0x01, R11

**jmp** loop

exitloop

**mov.b** R12, R13 ; R12 will store upperhalf of number, R13 stores lower half.

**and.b** #0xF0, R12 ; first bits

**and.b** #0x0F, R13 ; second bits

**rra** R12

**rra** R12

**rra** R12

**rra** R12 ; shifts the bits in R12 to the right 4 times to get rid of zeros

convert **cmp.b** #0x0A, R12

**jge** checkNext

**jmp** checkNext2

checkNext **cmp.b** #0x0F, R12

**jl** addConvert

**cmp.b** #0x0F, R12

**jeq** addConvert

**jmp** checkNext2

addConvert **add.b** #0x37, R12

**jmp** dontadd ; finish and compare second num

checkNext2 **cmp.b** #0x00, R12

**jge** checkNext3

**jmp** dontadd

checkNext3 **cmp.b** #0x09, R12

**jl** convertAdd

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

**jeq** convertAdd

**jmp** dontadd

convertAdd **add.b** #0x30, R12

dontadd

**cmp.b** #0x0A, R13

**jge** checkNext23

**jmp** checkNext22

checkNext23 **cmp.b** #0x0F, R13

**jl** addConvert2

**cmp.b** #0x0F, R13

**jeq** addConvert2

**jmp** checkNext22

addConvert2 **add.b** #0x37, R13

**jmp** finish2

checkNext22 **cmp.b** #0x00, R13

**jge** checkNext33

**jmp** finish2

checkNext33 **cmp.b** #0x09, R13

**jl** convertAdd2

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

**jeq** convertAdd

**jmp** finish2

convertAdd2 **add.b** #0x30, R13

; Now you've converted everything and you can print

finish2 **mov.w** R12, R4

**call** #OUTA\_UART

**mov.w** R13, R4

**call** #OUTA\_UART

**mov.w** #0x0A, R4

**call** #OUTA\_UART

**mov.w** #0x0D, R4

**call** #OUTA\_UART

Exit2 **ret**

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

part5

;get or set first element myarray(0)

**mov.b** #0x0D, R9

**mov.b** #0x00, R8

**mov.b** #32, R7 ; value you will count to

**mov.b** #0x0, R4

**mov.b** #0x0, R14 ; a temporary variable to store the asci char

**mov.b** #0x0, R6 ; counter for array buffer

loop77

**cmp.b** R7, R6

**jge** stoploop77

**call** #INCHAR\_UART

**mov.b** R4, R8 ; R8 now stores each character

**cmp.b** R8, R9 ; if enter is pressed, stop taking in chars

**jeq** stoploop77

**mov.b** R8, myarray(R6) ; move character

**mov.b** myarray(R6), R4

**call** #OUTA\_UART

**add.b** #0x01, R6 ; increment counter

**jmp** loop77

stoploop77

**mov.b** R9, R4 ; prints enter

**call** #OUTA\_UART

;;;;;;;;;;;;;; now do the sorting

; R6 contains the # you count to. Use R7 as counter

**mov.b** #0x0, R7 ; R7 will be the counter for loopfirst

; R8 will be the counter for loopsecond

**mov.b** #0x0, R8

loopfirst

**cmp.b** R6, R7

**jeq** stoploopfirst

**mov.b** #0x0, R8

loopsecond

**cmp.b** R6, R8

**jeq** stoploopsecond

;;;;;;;;; the switch

**mov.b** myarray(R7), R15

**mov.b** myarray(R8), R15

;;; delete these two above

**cmp.b** myarray(R7), myarray(R8)

**jge** switch

**jmp** dontswitch

switch

**mov.b** myarray(R7), R14 ; R14 = R7

**mov.b** myarray(R8), myarray(R7) ; R7 = R8

**mov.b** R14, myarray(R8) ; R8 = R14

;

dontswitch

;;;;;;;;;;;;;

**add.b** #0x01, R8

**jmp** loopsecond

stoploopsecond

**add.b** #0x01, R7

**jmp** loopfirst

stoploopfirst

; R7 is loop counter to print array

; R6 stores # of chars

**mov.b** #0x0, R7

printArray

**cmp.b** R8, R7

**jge** stopprinting

**mov.b** myarray(R7), R4

**call** #OUTA\_UART

**add.b** #0x1, R7

**jmp** printArray

stopprinting

Exit7 **ret**

; LAB UART CODE UNCOMMENT TO TEST IN Lab

OUTA\_UART

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

; prints to the screen the ASCII value stored in register 4 and

; uses register 5 as a temp value

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

; IFG2 register (1) = 1 transmit buffer is empty,

; UCA0TXBUF 8 bit transmit buffer

; wait for the transmit buffer to be empty before sending the

; data out

**push** R5

lpa **mov.b** &IFG2,R5

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

**cmp.b** #0x00,R5

**jz** lpa

; ; send the data to the transmit buffer UCA0TXBUF = A;

**mov.b** R4,&UCA0TXBUF

**pop** R5

**ret**

INCHAR\_UART

**push** R5

lpb **mov.b** &IFG2,R5

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

**cmp.b** #0x00,R5

**jz** lpb

**mov.b** &UCA0RXBUF,R4

**pop** R5

;go get the char from the receive buffer

**ret**

Init\_UART

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

;Initialization code to set up the uart on the experimenter board to 8 data,

; stop, no parity, and 9600 baud, polling operation

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

**mov.b** #0x30,&P2SEL

**mov.b** #0x00,&UCA0CTL0

**mov.b** #0x41,&UCA0CTL1

**mov.b** #0x00,&UCA0BR1

**mov.b** #0x03,&UCA0BR0

**mov.b** #0x06,&UCA0MCTL

**mov.b** #0x00,&UCA0STAT

**mov.b** #0x40,&UCA0CTL1

**mov.b** #0x00,&IE2

**ret**

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

; Interrupt Vectors

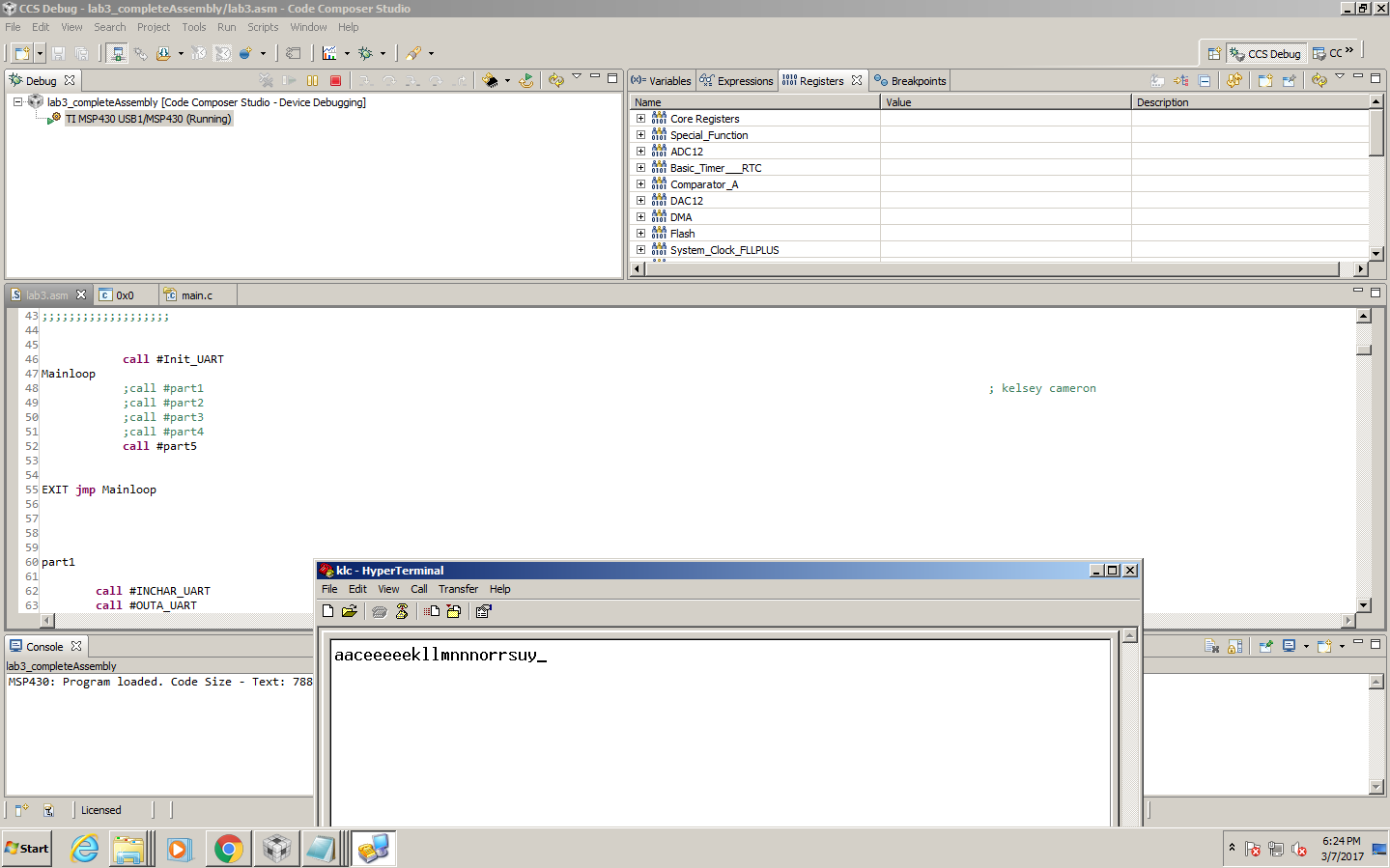
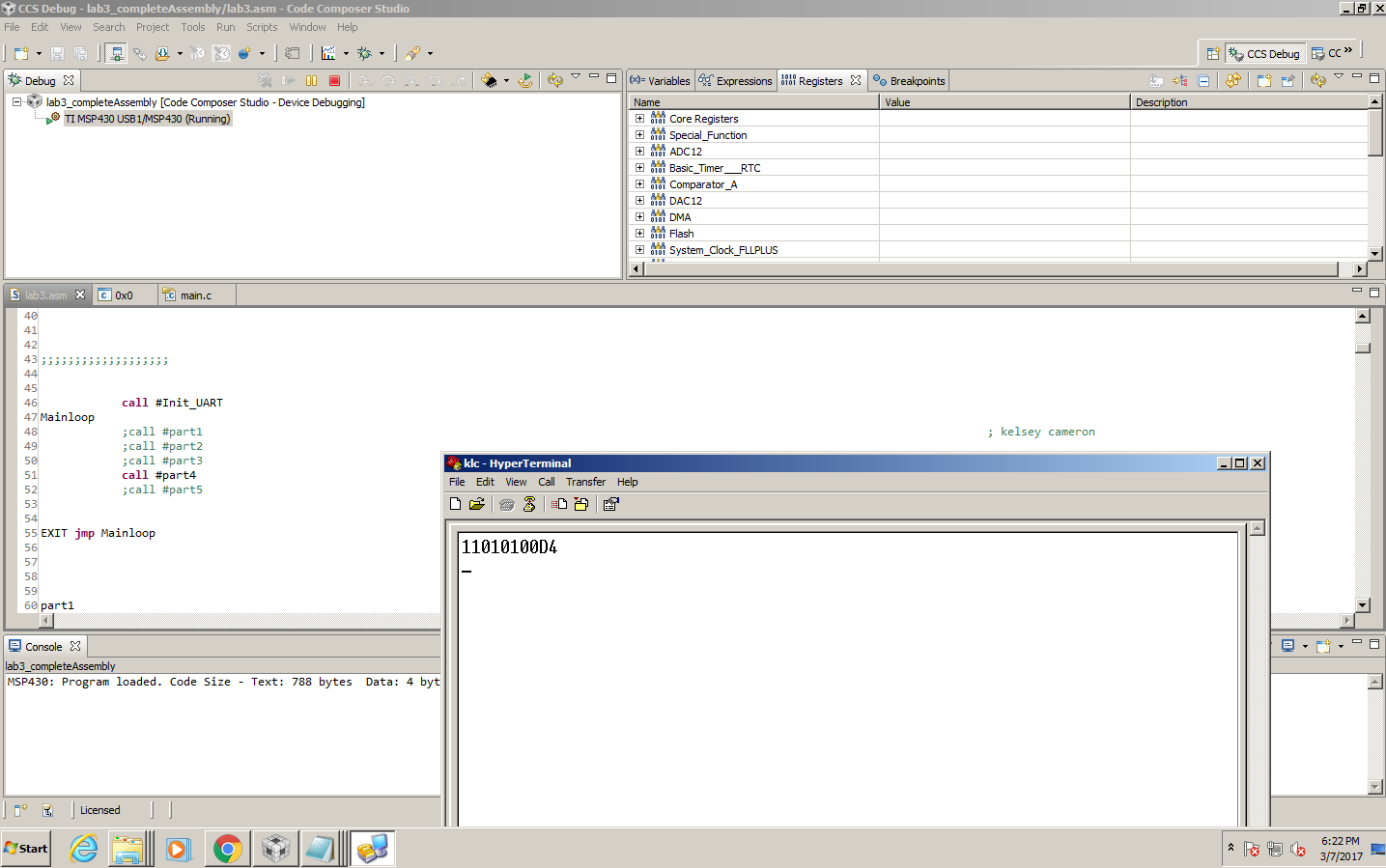
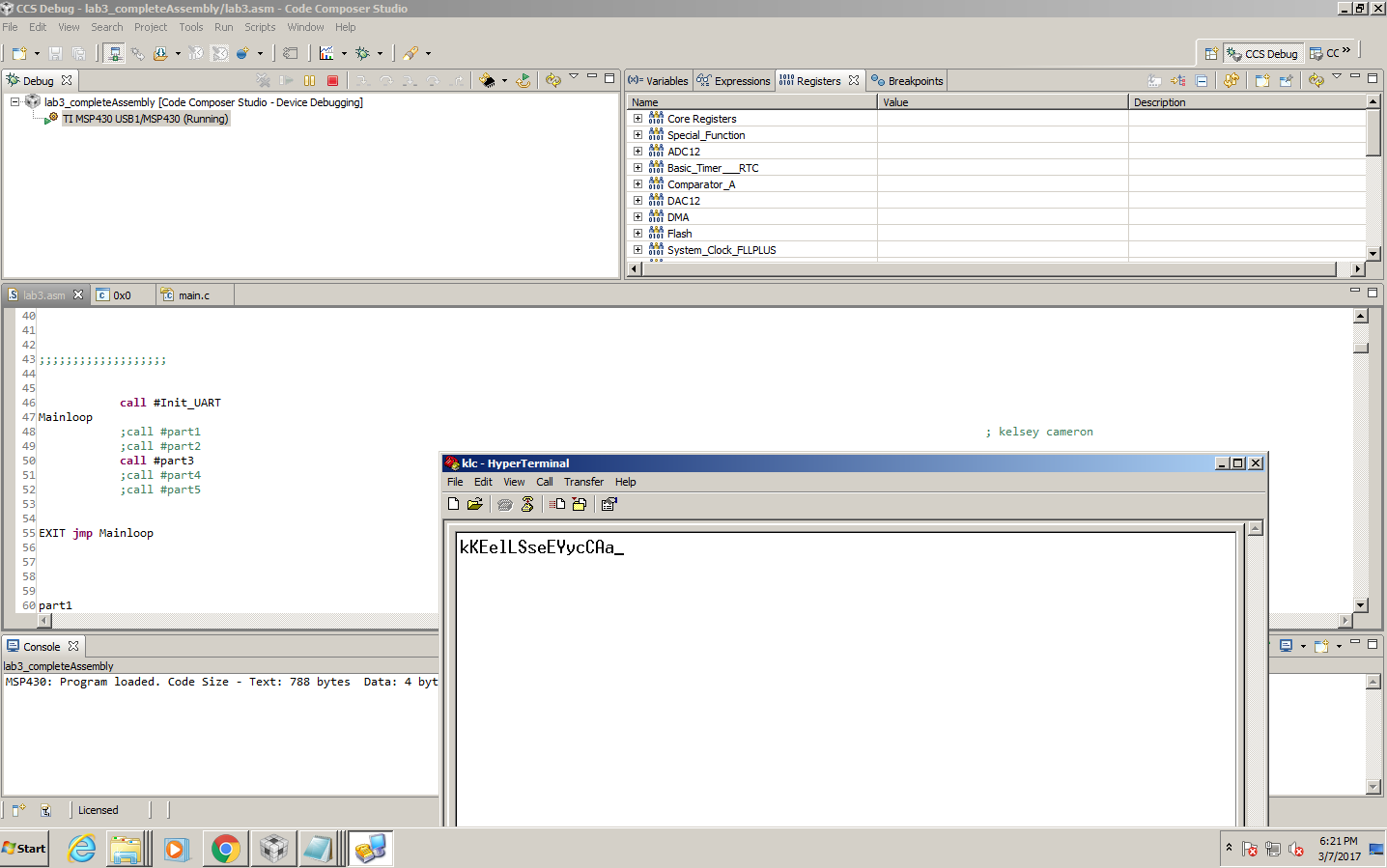
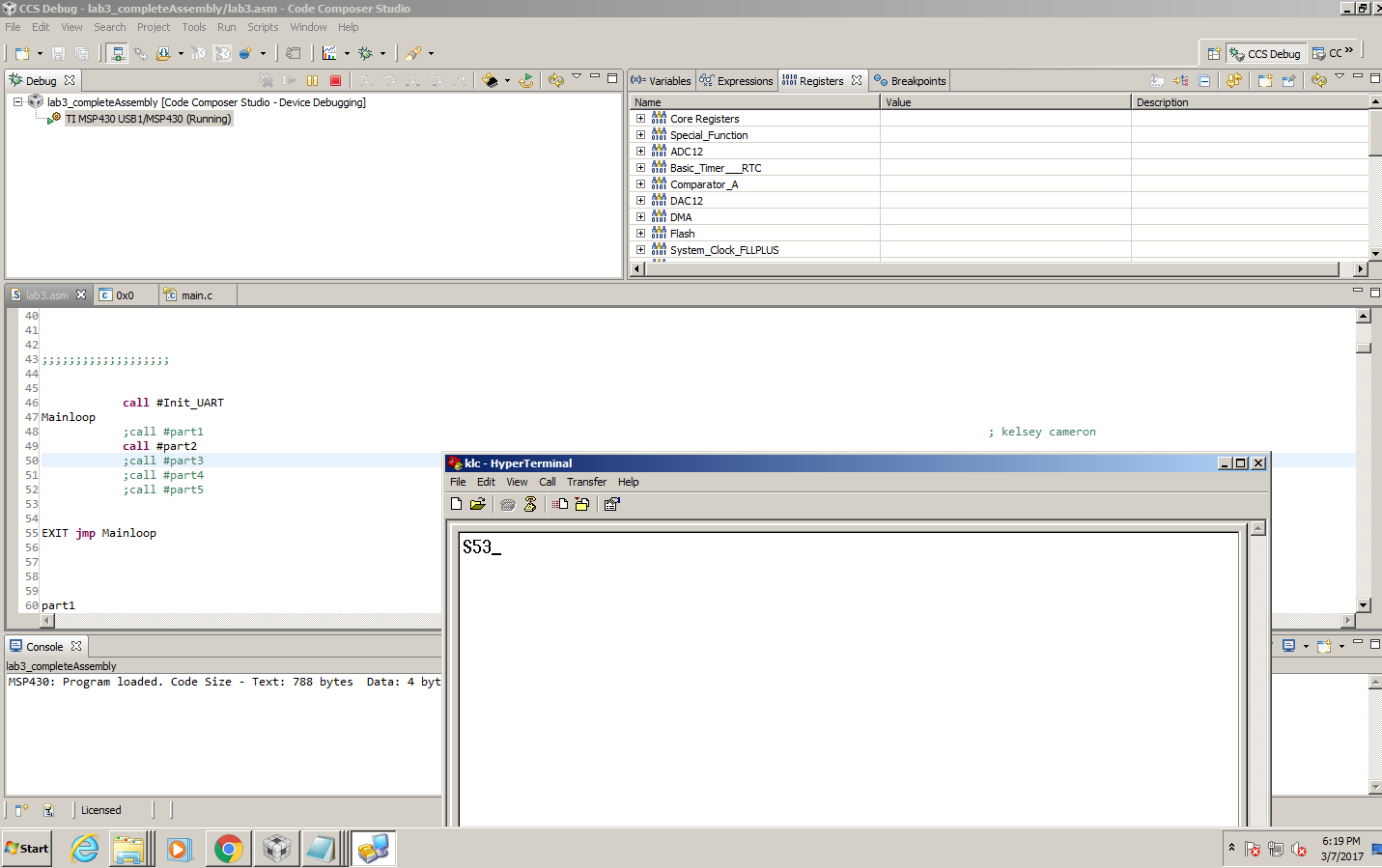
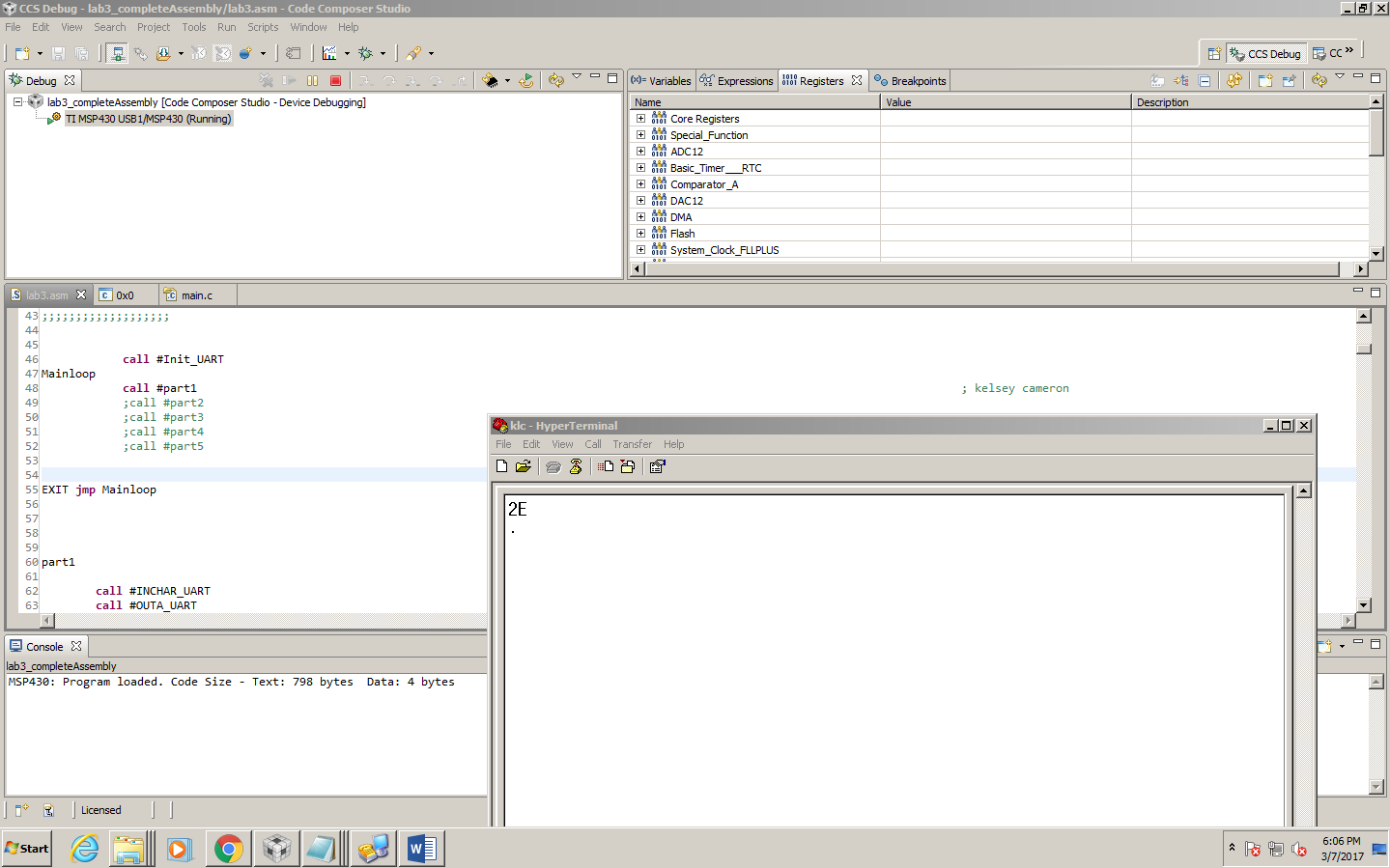
;----------------------------------------------------------------

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

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

.end

**Assembly Screenshots:**

****

**Conclusion:**

In conclusion, this experiment taught me a lot about ASCII values. Prior to this, I didn’t realize that you could add and subtract characters like numbers in Hex, and I never really realized that a character is stored in hex within the chip. I learned a lot about syntax for assembly in this experiment (like how I constantly forgot to write .b after almost everything, and how I often forgot the pound signs), but this extra experience has improved my ability to program in assembly code and enlightened my understanding of what is actually going on within the hardware.