**ECE222 Lab2&3 Post-lab Report**

**Lab2 Dairy**

February 15th, 2012

* Main menu was completed.
* Only the calculator mode could be chosen.
* An error message would be shown on screen if user chose other modes.
* Some messages (prompts) are inserted into the object program using the DC (data constant) directive. We could simply refer back to the messages using some labels, namely PROMPT, PROMPT1, PROMPT2, PROMPT3, PROMPT4 and PROMPT5.
* The calculator mode subroutine would be designed later.

February 16th, 2012

* We have already designed the “bin to dec” and “dec to bin” subroutine on the lab1 post-lab report, so we can use this subroutine for character conversions.
* Input/output subroutines, namely in\_string, out\_string, in\_char, out\_crlf and out\_char, can be taken from the source code given in lab1.

February 17th, 2012

* The calculator mode subroutine was almost completed. However, the subtract operation could only handle subtractions with positive answers. If the answer was negative, the answer outputted on the screen would be “63356/”; therefore, we came up with an idea to handle this situation.
* Idea: We have to first compare which number is greater. If the first number is greater than the second number, we will proceed to the normal subtraction operation. If the first number is less than the second number, then we have to subtract the first number from the second number and prepend a negative sign in front of the answer (push the sign into the stack), and the output would be a negative answer.

February 18th, 2012

* Overflow/underflow checking was added to the character conversion subroutine. Before doing any “bin to dec” conversion, we have to check if the number is less than 9999 first. We do this by using

cmp.l #9999,D6

bgt handle\_overflow

And use the handle\_overflow subroutine to print out an overflow/underflow message (PROMTP3).

* We have changed the in\_string subroutine a little bit, so that when user presses enter without providing any input, the program would direct the user back to the main menu. We do this by testing the character count (tst.l D2), and if the character count is zero, the program goes back to menu (beq menu).

**Lab3 Dairy**

March 8th, 2012

* Since we cannot use Trap #15 in the in\_char subroutine anymore, we have replace it with

WAIT\_RECEIVER move.b USR0,D0

btst.b #0,D0 ; test if a key is pressed

beq WAIT\_RECEIVER ; wait for key to be pressed

move.b URB0,D1 ; record key press into D1

* Since we cannot use Trap #15 in the out\_char subroutine anymore, we have replace it with

WAIT\_TRANSMITTER

move.b D0,UCR0 ; set “Transmitter Enable” bit in UCR0 to 1

move.b USR0,D2

btst.b #2,D2 ; test if a key is pressed by checking if bit2 is zero

beq WAIT\_TRANSMITTER ; wait for key to be pressed

move.b D1,UTB0 ; take content of D1 and store it into UTBO

* This is tested by trying all operations and see if input/output data is received/transmitted properly.

March 9th, 2012

* A separate input subroutine is written for keypad.
* Delay is performed by looping around the DELAYTIME loop a few thousand times.

DELAYTIME subq.l #01,D0 ; loop appropriate times to create a delay

cmp.l #0,D0

beq DELAYTIME

* A character look up table is created to determine which button is pressed.
* In order to stop auto repeat, the program has to loop around the STOP\_AUTO\_REPEAT loop until no input is received from PADAT.

STOP\_AUTO\_REPEAT clr.l D7

move.w PADAT,D7

cmp.l #$FF0F,D7

bne STOP\_AUTO\_REPEAT

**Discussion**

We used our testing strategy described in the pre-lab to check if keys of the keyboard and the keypad were detected properly. Key debouncing was checked by pressing on one of the keys and making sure that only one input is taken. In lab 2, we learned to use cfasm, the Coldfire assembler program, to “translate” our source code into the machine language that the computer understands. We also learned more about subroutines by creating several subroutines ourselves. In lab 3, we learned to write our own version of I/O subroutines for the small calculator program and to configure serial ports for input and output and use a parallel port to communicate with the keypad. We also learned how to deal with mechanical switch rebouncing problem by creating a delay.

**Possible Improvements**

1. Include deleting function

* A delete key could possibly be added to our program so that users can delete a character and type in a new one anytime.

1. Shorten the program

* Since sometimes we are using the same piece of code more than once in the program, the program could possibly be shortened by using more subroutines.