ECE 3731: Microproc & Embedded Sys Lab

Final Project: Musical Calculator

August 8th, 2018

Summer 2018

Honor Code: I have neither given nor received unauthorized assistance on this graded report.

x Brian Vera-Burgos x Srinivas Simhan

Table of Contents

Introduction

The purpose of this program is to write an assembly language program that asks the user to enter two digits, implement an operation using the two digits, and then display the output on the computer and on the HCS Dragon12-Light board.

Both input digits are between 0 and 9, and in the case of an invalid input, then the program will prompt for another number to be entered.

For each entered input (first digit, second digit, and operation), the program generates two tones, each for half a second. For example, if the user enters 5, it generates a tone of 770Hz for half a second, and the immediately generates a tone of 1336Hz for another half a second.

Once the third value (which is the operation value) is executed, the HCS Dragon12-Light board then displays the generated value onto it's seven segment display.

Once this motion has been completed, the program then starts the cycle over again and asks for a first digit value and repeats the cycle. If the user inputs 'C' as an input, the program breaks and starts over, asking for the first digit value to be inputted again.

Objective

- To become familiar with CodeWarrior
- To become familiar with HCS Dragon12-Light
    - To understand how to utilize and output to the seven segment display on the HCS Dragon12-Light board
- To become familiar with Tera Term
- To understand and create flowcharts to be used in explanations in terms of documentation for our code
- To implement Dual Tone Multi Frequency using keyboard inputs
- To implement the speaker output on the HCS Dragon12-Light board
- To implement addition, subtraction, multiplication, and division using inputs

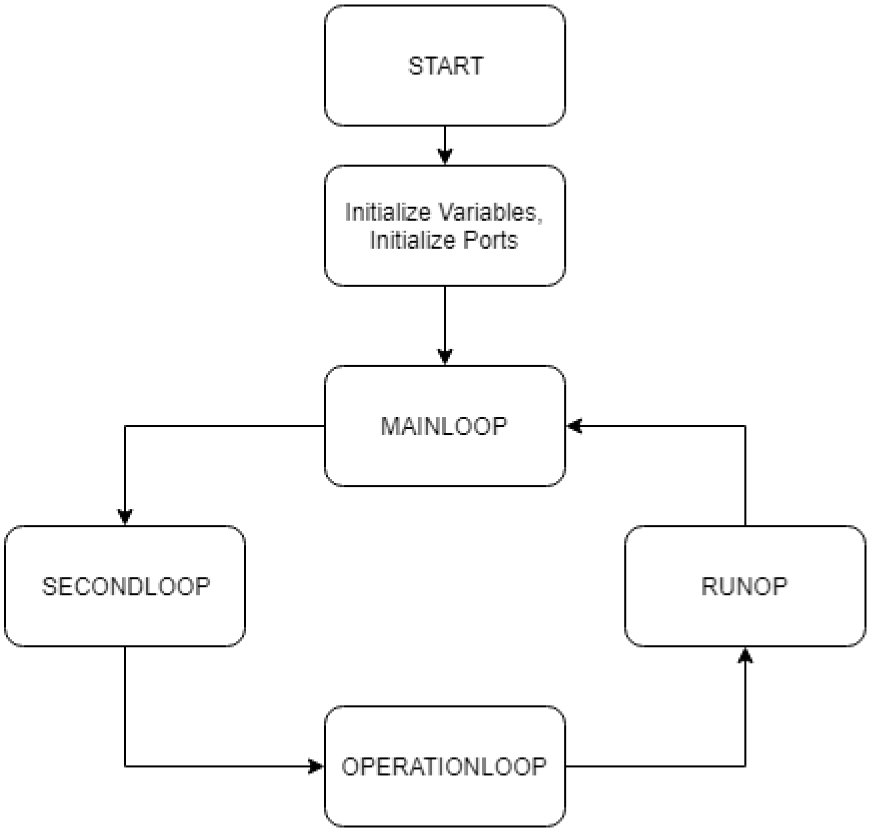
Equipment Used

- CodeWarrior
- Tera Term
- HCS Dragon12-Light

Flowcharts

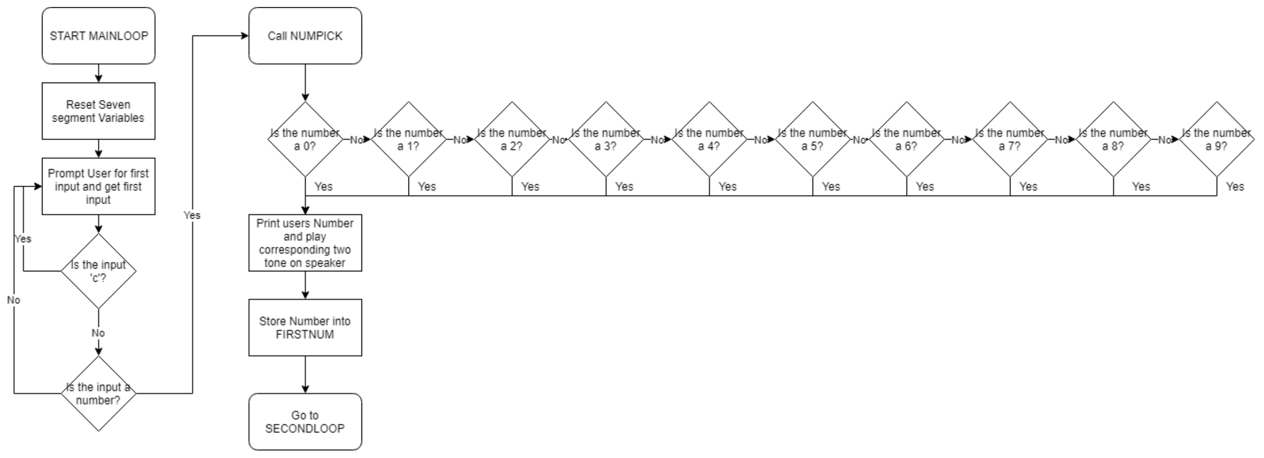
1) Total Flowchart

This program works by running through 4 major labels which loop back to mainloop. There is a flowchart for each major label, a general overview is as shown below:



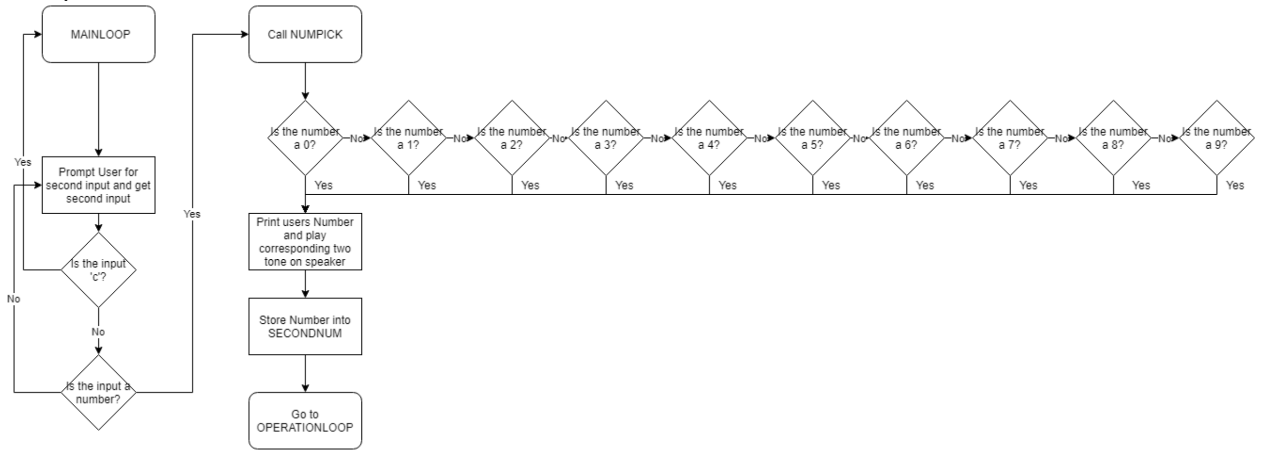
2) Mainloop Flowchart

The MAINLOOP label is used as the starting point of the program. Whenever the program restarts, it comes to this point. Its primary function is to prompt the user for the first variable and to identify, store, and play the two tone sound for the number. This label will go to SECONDLOOP upon completion.



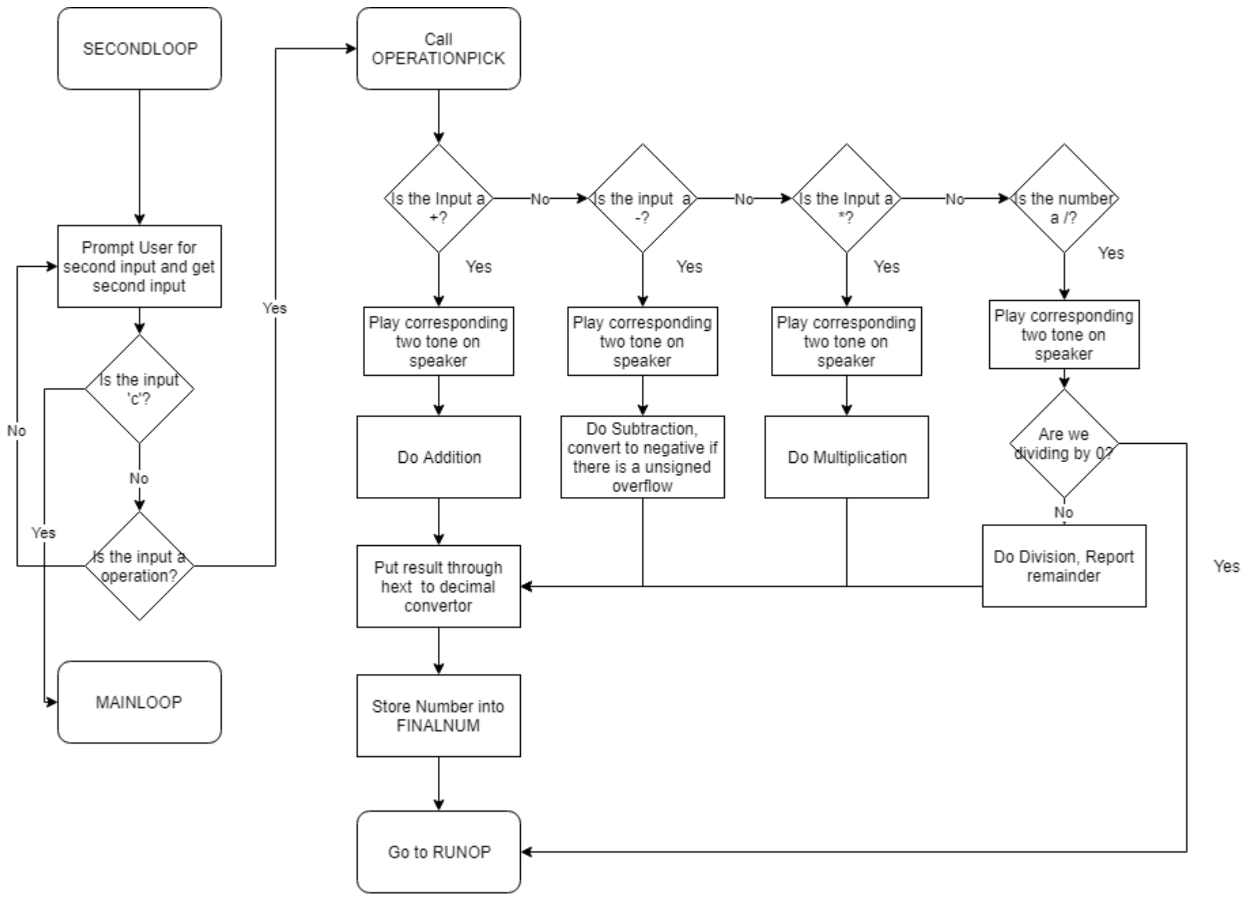
3) Secondloop flowchart

The SECONDLOOP label is used to prompt the user for the second value and to identify, store, and play the two tone sound for the number. This label will go to OPERATIONLOOP upon completion.



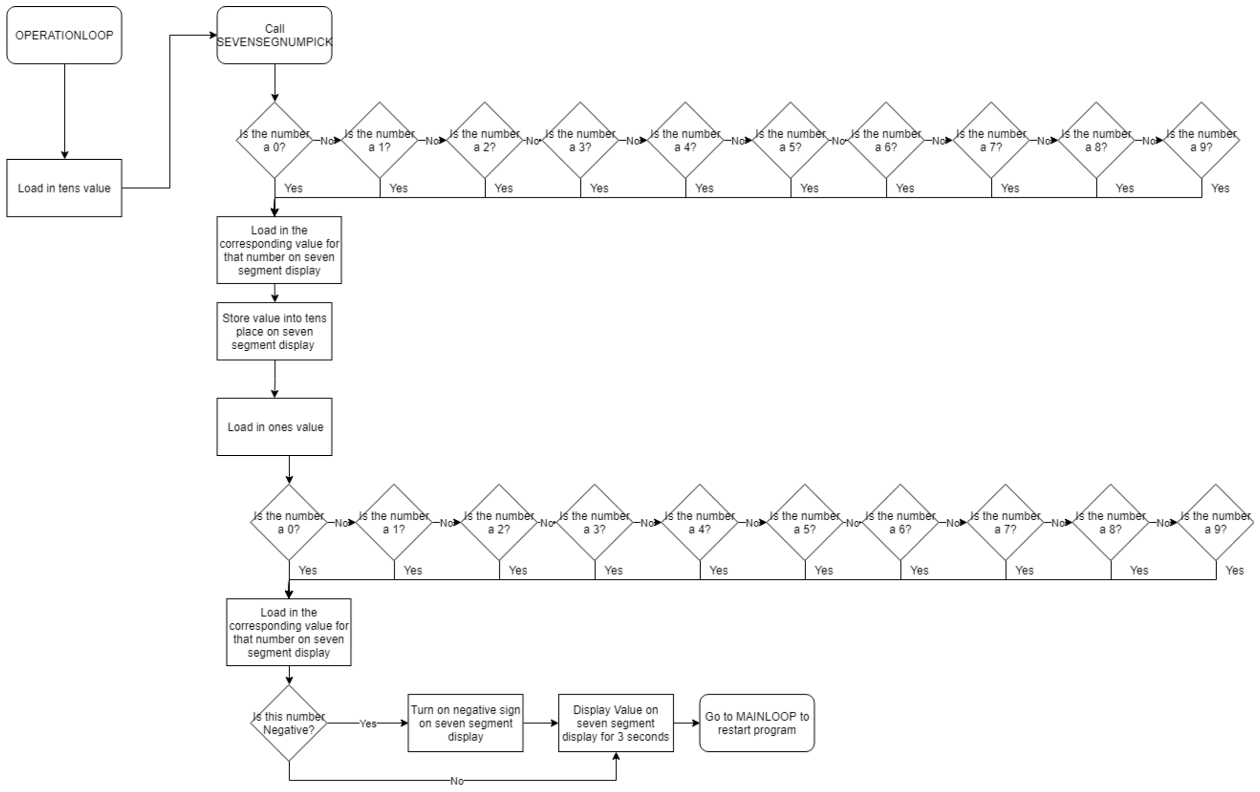
4) Operationloop flowchart

The OPERATIONLOOP label is used to determine what operation was requested to be performed from the user. This label will go to RUNOP upon completion.


SECONDLOOP
Prompt User for second input and get second input
Is the input 'c'?
Yes
No
Is the input a operation?
No
Yes
MAINLOOP
Call OPERATIONPICK
Is the Input a +?
No
Is the input a -?
No
Is the Input a *?
No
Is the number a /?
Yes
Play corresponding two tone on speaker
Do Addition
Do Subtraction, convert to negative if there is a unsigned overflow
Do Multiplication
Are we dividing by 0?
No
Yes
Do Division, Report remainder
Put result through hext to decimal convertor
Store Number into FINALNUM
Go to RUNOP

5) Runop flowchart

The RUNOP label finished the program by calling the stored values that the user inputted, and displaying them on the seven segment display briefly. After this label completes, the program returns to the mainloop and restarts.



Design Process

We initially figured that the best bet would be to use timer interrupts to make the two tone sound, since they are extremely accurate. However, we quickly found that this would be extremely difficult, as we need at least 10 timer subroutines to achieve all of the different subroutines. We also considered using the ms_delay function, since its easy to use and very accurate. This however, was impossible to use in this context since we needed to be accurate to tenths of milliseconds in order to generate the frequencies (ms_delay only goes to ones of milliseconds).

The simplest solution was to use the homemade timer loop that we made in lab. This is semi undesirable, since these timers are inaccurate, especially at tenths of milliseconds. Even with this inaccuracy, the method worked for our project. This is because the difference in each frequency relative to one another was good enough to hear the difference in each number, and thus worked for our needs.

Our homemade timer works by using a subroutine that delays for a tenth of a millisecond. We then call this function 65 times (for example) to create a delay that would make a 770hz frequency. Similar functions that call the unit function different numbers of times allow us to make the other frequencies. One minor difficulty with this method was that we had to calculate by hand the number of times we had to call each of the frequencies in order to create a half second delay in them.

Demo Link (YouTube Video)

- Due to a lack of editing software, the video is split into two parts, the introduction and the demonstration. Sorry for the inconvenience.
- Introduction: https://youtu.be/-kuHb-Pg1pY
- Demonstration: https://youtu.be/sH9VFC9QCrE

Summary

From this project, we learned how to write an assembly language program that asked the user to enter two digits, implement an operation using the two digits, and then display the output on the computer and on the HCS Dragon12-Light board. We implemented the required parameters as given by the original project requirements sheet. We handled inputs that were not from 0 to 9, and based on each of the numbers inputted, we generated two tones which played for half a second each, one after the other. Then after the operation was selected, we calculated and displayed the result onto the computer and the HCS Dragon12-Light board's seven segment display.

We also implemented a subroutine in which when we looked for an input, we checked if the user input was 'C', the program breaks and starts over, asking for the first digit value to be inputted again.

Conclusion

From this program, we were able to practice completing a full project using our assembly programming language skills that we developed over the past few months. This project required fluency of all basic programming skills, like subroutines and loops, as well as a good grasp of assembly specific concepts, like accumulators and memory. While we didn't use didn't use the full range of topics we covered, like interrupts and timers, this was because the nature of the project gave us the freedom to find the most effective way of completing the assignment. We did get to practice going out and learning how to use a new system on our own by implementing the seven segment convertor which was not covered in class. The freedom that this project offered, allowing us to pick and chose which methods would get us the best end result, made it an extremely educational and satisfying assignment to complete.