EE371 - PROGRAMMING PROJECT #1, FALL 2017

Douglas V. Hall
Electrical and Computer Engineering Department
Portland State University
Portland, OR 97207

ARRAYS AND PROCEDURES

REFERENCE: Hall Chapters 3 and 4 along with discussion.

IMPORTANT NOTE: Read ALL of the project description, the list of deliverables, and the required statement of individual effort before doing any work on the project.

Introduction:

This exercise will give you some experience using many of the basic techniques in ARM assembly language programming. Specifically, you will gain experience with processing arrays with pointers, auto-increment addressing, conditional branches, a procedure call and return, passing parameters to procedures converting an algorithm to assembly language, and the program development cycle. This assignment will also give you some practice following the "Fast is Slow" rule and convincing yourself that it really works.

Problem Statement:

For this project, assume that a system is set up with a temperature sensor and an Analog to Digital converter to determine the Fahrenheit temperature every 1.5 hours in each 24 hour period. The A/D converter produces 8-bit binary values that represent the temperature in Fahrenheit. Temperature values range from 0 – 120+ degrees Fahrenheit. The 8-bit binary values for the 16 temperatures taken each day are stored in an array of bytes named Fahrenheit_Rough. The only problem is that the temperature sensor is non-linear. A correction value must be added to the value from the A/D converter to get the correct Fahrenheit temperature. The Correction factors are as follows:

A/D Reading (decimal)	Correction Factor (decimal)
0-20	+ 0
21-39	+ 1
40-59	+ 3
60-79	+ 7
80-99	+ 12
100- 120	+ 20

Your first program task will be to generate the corrected value for each value in the Fahrenheit_Rough array and put the corrected value in the same numbered element location in a second array called Fahrenheit_True. Also, you will calculate the rounded

average of the 16 Fahrenheit_True values and put the rounded average in an appropriately named memory location.

You will develop the program in two major steps.

1. In the first step you systematically develop a simple program that does all the conversions, averaging , and writing the rounded average to a named memory location in a straight-line program that does not use a procedure.
2. Once you get the program working correctly, you generate a second version of the program that moves the corrections and averaging to a procedure called from a mainline. The mainline passes array pointers and a count to the procedure and stores the rounded average passed back to it from the procedure.

Development Suggestions:

1. The first step here is to understand the data structures for the problem. As part of this, you can write the ARM assembly language statements to declare the required data structures. Note that to test your program, you will have to initialize the Fahrenheit_Rough array with 16 trial byte values for which you can predict the correct correction needed. (If you need to declare an array that is longer than will fit on one line, you can simply put a dummy name on the next line and type more elements of the array on the next line.) Carefully determine reasonable test values to put in your Fahrenheit_Rough array, so you are testing a range of values.

2. Using the **standard program structures** shown in Hall Chapter 3, write the algorithm for the section of the program that reads each value from the Fahrenheit_Rough array, adds the needed correction factor, and writes the corrected value to the same element position in the Fahrenheit_True array. (The way you easily create an algorithm for a situation such as his is to just think how you would verbally tell a person in English or your native language the sequence of operations . In simplest terms, Get one of these, Add appropriate correction factor, Store result, Get next, etc. Then think how you would tell a person to determine the correct correction factor to add to each. One of the examples on page 140 gives you a big hint on how to do this.

3. Finally think how you would tell a person to determine the average of the 16 Fahrenheit_True values. The specification requires the “rounded average”, so you have to tell the person how to do this. It might help you to remember how you use the remainder when you round the result after doing decimal division.

4. As described in the ARM Tutorial, edit, assemble, link, run, test, and debug this first version of your assembly language program.

5. When your complete program works correctly, print out copies of the final source code and the appropriate Debugger memory display(s) to demonstrate that the program works correctly.

6. Now that the straight-line program is working correctly. Re-read the section of the text that teaches you how to work with procedures.

7. Write the algorithm for a program that moves the correction and averaging into a procedure and just calls the procedure from a mainline that sets up a stack, passes array pointers and a count to the procedure, and stores the average value returned from the procedure in a named memory location.

8. Write the assembly language mainline program that sets up the required data arrays, loads pointers to the source and destination arrays in the appropriate registers as specified by APCS, and calls the procedure (Use the examples given in class and the APCS to help with appropriate register usage.) To this, add the assembly language instructions to store the returned Average to a named memory location.

9. Modify the working conversion code to use the pointers passed from the mainline, save registers used in the procedure as specified by APCS. Restores registers at the end of the action, returns execution to the mainline with the Average in the appropriate register . Combine the mainline code and the conversion code

10. As described in the ARM Tutorial, edit, assemble, link, run, test, and debug your assembly program. At this point you will just be checking if the procedure is being called correctly, if the procedure is putting the correct binary values in the array, if average is passed back to the mainline, when execution returns to the mainline.

11. When the procedure call and the procedure are working correctly, save this version of your program and make sure it is thoroughly documented.

12. When your complete program works correctly, print out copies of the final source code and the appropriate Debugger memory display(s) to demonstrate that the program works correctly.

Deliverables:

The golden rule of documentation is, "Create the documentation you would like to receive, if you were assigned the task of adding features to the product."

Your documentation for the project should include:

- **A.** A design log that shows the steps you took, any problems you had, how you solved these problems, and the results at each step.
- **B.** For the first, straight-line version of the program, a clearly written algorithm
- **C.** For the second version of the program, a clearly written algorithm for the procedure and a clearly written algorithm /task list for the mainline.

D. A discussion of how you tested the program thoroughly so you can convince your boss that it is marketable. (Refer to Debugger printouts and specific cases. Remember comments about "boundary values")

E. A printout of the .s file for each program with header and full comments.

F. **Labeled** printouts of Debugger Memory Windows that show the program works.

G. **A signed statement that "I developed and wrote this program by myself with no help from anyone except the instructor and/or the T.A. and I did not provide help to anyone else"**

(Any evidence of joint work will result in project grades of zeros for all parties involved.)

GRADING KEY FOR ECE 371 DESIGN PROJECT #1 Doug Hall FALL 2017

LOG (DETAILED AND "AS YOU GO")	**20 MAX**	_____
TASK LIST/ALGORITHM (DETAILED AND COMPLETE)	**20 MAX**	_____
WORKING PROGRAM	**40 MAX**	_____
COMMENTS (CLEAR AND USEFUL)	**10 MAX**	_____
OVERALL ORGANIZATION	**10 MAX**	_____
TOTAL	**100 MAX**	_____