ECE371 - PROGRAMMING PROJECT #1, FALL 2018

ARRAYS AND PROCEDURES

Yusme Pradera

**Part 1 of Project\_1**

* To implement the first part of the project the textbook was a great help. In particular the example (Program to average elements of two arrays) page 154.

My first attempt was to as described by the lab report to using standard program structures to write an algorithm for the program section.

Average of 16 items from Fahrenheit\_Temps array

* Get a Fahrenheit\_Temps Array byte
* Add Fahrenheit\_Temps to AVERAGE
* Decrement element counter, set flags
* Repeat loop until it reaches 16 added items
* Divide by 16 by shifting left
* Add contents of carry flag
* Store the resulted byte in AVERAGE
* Return to mainline Program

**Algorithm for the first version of the program:**

**AVE:**

LDRB R6, [R1], #1 @ Get a Fahrenheit\_Temps byte, increment pointer

ADDS R2, R2, R6 @ Add Fahrenheit\_Temps to AVERAGE

SUBS R3, R3, #1 @ Decrement element counter, set flags

BNE AVE @ Repeat loop until it reaches 16 times

MOVS R6, R6, LSR #4 @ Divide by 16, LSB to carry

ADC R6, R6, #0 @ Add contents of carry flag to sum in r6

STRB R6, [R2] @ Copy result byte to AVERAGE

* Problems encountered by creating this part of the project was in the dividing I had dividing by 256 not by 16 by having #8 which in return wasn’t dividing by the amount of correct numbers in the array in this case #4 since its (2^number) will give the number divided by .
* Needed to delete a line of code that I forgot from the previous time I was implementing something different (MOV PC, R14 @ Return to mainline program),

**PART 2 Project\_1**

The second part of the project took a little more research than the first part. The finding the MAX and Min values from the array.

* The use of BGT and BLT helped me accomplish what I wanted which was to branch when greater or when less than and store the value in registers assigned for those values.

**Part 2 – Finding Max and Min values in the Fahrenheit\_Temps**

Algorithm to find the Max and Min Value in the Fahrenheit\_Temps.

* Take the first element in the Fahrenheit\_Temps Array.
* Get the value from Max\_Value
* Compare value to determine Max\_Value.
* If the Value in Max\_Value its greater than the compared value than branch and save the result in Max\_Value
* Get the value from Min\_Value
* Compare value to determine Min\_Value.
* If the Min\_Value its greater than the compared value than branch and save the result in Min\_Value

This process will repeat by using a second counter until reaches 16 times

**Algorithm for the second version of the program:**

**COMPARE:**

LDRB R6, [R1], #1 @ Get the first value from the Fahrenheit\_Temps

LDRB R8, [R4] @ Get the value from Max\_Value

CMP R6, R8 @ Compare values to determine Max\_Value

BGT Max\_V @ Branch if greater

LDRB R9, [R5] @ Get Value from Min\_Value

CMP R6, R9 @ Compare the value from array with the value in R9

BLT Min\_V @ Branch if less

NOP

**Max\_V:** STRB R6, [R4]

SUBS R7, R7, #1 @ Decrement second counter

BNE COMPARE

**Min\_V:** STRB R6, [R5]

SUBS R7, R7, #1 @ Decrement second counter

BNE COMPARE

* One of the biggest time consuming things within this part of the project was translating the section with conditional when greater than and less than to also store the value.
* Early approaches involved thinking in using a dummy register to store the first item in the array and comparing each other items until it discovered the desired value but then the comparison would have had to be to many times and accomplishing it this way was not a good approach.
* Figuring out what was the problem with the statement “No source available for "0x80000074" which the T.A. explained it was just a bug from CCS.

**Putting everything together**

The program its split into two sections averaging 16 values taken from Fahrenheit’s Temp Array and finding Min and Max Values from the Fahrenheit’s Temp Array.

The implementation of this program could be a great thing for testing purposes. For example getting the Min and Max temp values from testing an engine under stress. And also a range of temperatures from beginning to end of the testing period.

The first thing done was to use a paper and pencil to calculate the values just to know the expected result that needed to be seen.

The array contains 16 numbers and they sum to 752. Since there are 16 items in the array dividing by 16 this will give us a result of 47. After number expected were known it was time to start writing the program. First declaring all the variables that will be used in the program. This program as stated above uses an array of 16 numbers called Fahrenheit\_Temps that uses R1, tree variables called Average using R2 to save the obtained number when the Average procedure gets accomplished, Max\_Value, using R4, to store the maximum value obtained from the comparison of the Fahrenheit\_Temps Array as well as a Min\_Value variable, using R5, to store the minimum value from the Fahrenheit\_Temps Array. There are two counters. Register R7 was used to implement the counter that decrements the 16 values of the array to do calculations in a loop until we complete going over all the contents in the array. The other counter was used as a dummy variable initialized to 0 to add the numbers and such in R3.

After declaring all the variables we can jump to start and under START: we can find AVE: that it’s the start of a loop designed to count and add all the elements of the array. First we load the first element in the array in register R6,( LDRB R6, [R1], #1), by using the #1 we command the program to increment to next value of the array that way the next time it goes over this line we can grab the next item in the array for the adding purpose. After using register 6 we add using the Register 3 and next we decrement the counter of 16 values used by Register R7 so that we keep a count of how many times we need to go over the loop.

One very helpful trick it’s the use of BNE AVE, first to specify keep going over the loop until done and also to link back to the beginning of the loop at AVE:. After all the addition from the elements in the Fahrenheit\_Temps Array was done the BNE AVE will come into effect and no longer will go over the loop jumping to the next step of the Average purpose. Unfortunately we don’t have a dividing and multiplying command, but shifting to the right we will be able to divide by 2 for every time we shift and since we want to shift by 16 then we need to shift 4 time. Just in case there is a carry ADC is used to add the carry to the result. Following, we conclude storing the average value into Register R2.

The next section of the program it’s to compare values from the Fahrenheit \_Temps Array to determine Min and Max values. COMPARE: was used to signal a loop that keeps going if not all the 16 values have been compared to. Proceeding with loading the first value of the array in R6 and incrementing by one so next time we approach the line of code again we can used the next item in the array (LDRB R6, [R1], #1).

Loading the contents of Register 4 which in this case is Max\_Value into Register 8 to proceed with a compare statement to determine if the value in Max\_Value is bigger. If the flag is activated than BGT (Branch in Greater than) continues to branch to Max\_V: where we store the new biggest value into Register 4 since it’s the one associated with the max values of the array. Next we decrement the counter keeping track of the 16 values in the array by using (SUBS R7, R7, #1), and to complete the branch we jump back to COMPARE to continue comparing the values of the array.

For the case of looking for the minimum value in the array we can observe that if the flag is not activated when comparing between R6 and R8 then the value in Max\_Value its smaller than the value compared with from the array so the program continues to load register 9 associated with Min\_Value in R5 and proceed with a comparison to determine if the value in Min\_Value it’s the smallest if the flag gets activated by doing the comparison the BLT(Branch if less than) Min\_V branches to Min\_V where it proceeds to store the value and decrement the 16 values counter to keep with the comparison. After The program have found the Min and Max values the program will reach its end with values of Average in R2, Min value in R5 and Max value in R4.

**Project\_1 Copy of the .s file**

@ Project 1 ECE 371

@ By Josh Pradera

@ Average of 16 items from Fahrenheit\_Temps Array, and choosing Max and Min Values

**.text**

**.global** \_start

**\_start:**

.EQU NUM, 16

LDR R13, =STACK @ Point stack pointer to bottom of stack space

ADD R13, R13, #0x100 @ Point stack pointer at top of stack

LDR R1, =Fahrenheit\_Temps @ Load Pointer to TEMP Array

LDR R2, =AVERAGE @ Load Pointer to AVERAGE

MOV R3, #0x0

LDR R4, =Max\_Value @ Load Pointer to Max\_Value

LDR R5, =Min\_Value @ Load Pointer to Min\_Value

MOV R7, #NUM @ Load second counter in R2

B START

NOP

**START:**

**AVE:**

LDRB R6, [R1], #1 @ Get a Fahrenheit\_Temps byte, increment pointer

ADD R3, R3, R6 @ Add Fahrenheit\_Temps to AVERAGE

SUBS R7, R7, #1 @ Decrement element counter, set flags

BNE AVE @ Repeat loop until it reaches 16 times

MOVS R3, R3, LSR #4 @ Divide by 16, LSB to carry

ADC R3, R3, #0 @ Add contents of carry flag to sum in r

STRB R3, [R2] @ Copy result byte to AVERAGE

**COMPARE:**

LDRB R6, [R1], #1 @ Get the first value from the Fahrenheit\_Temps

LDRB R8, [R4] @ Get the value from Max\_Value

CMP R6, R8 @ Compare values to determine Max\_Value

BGT Max\_V @ Branch if greater

LDRB R9, [R5] @ Get Value from Min\_Value

CMP R6, R9 @ Compare the value from array with the value in R9

BLT Min\_V @ Branch if less

NOP

**Max\_V:** STRB R6, [R4]

SUBS R7, R7, #1 @ Decrement second counter

BNE COMPARE

**Min\_V:** STRB R6, [R5]

SUBS R7, R7, #1 @ Decrement second counter

BNE COMPARE

**.data**

**.align** 2

**Max\_Value:** **.byte** 0x0

**Min\_Value:** **.byte** 0x0

**Fahrenheit\_Temps:** **.byte** 0x32, 0x34, 0x35, 0x42, 0x50, 0x54, 0x58, 0x78, 0x74, 0x60, 0x45, 0x41, 0x40, 0x39, 0x37, 0x33

**AVERAGE:** **.byte** 0x0

**STACK:** .rep 256 @ Reserve 256 bytes for stack and init with 0x00

**.byte** 0x00

.endr

.end @ End of the Program

