Ryan Bentz

ECE 371

Design Project 1

11/05/17

Portland State University

Problem Statement

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 nonlinear. A correction value must be added to the value from the A/D converter to get the correct Fahrenheit temperature. 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. Calculate the rounded average of the 16 Fahrenheit_True values and put the rounded average in an appropriately named memory location.

List of Deliverables:

A. In-line Version Algorithm
B. Code for In-line Version
C. Procedural Version Algorithm
D. Code for Procedural Version
E. Discussion of Testing and Verification
F. Debugger Memory Results
G. Design Log
H. Statement of Authenticity

A. In-line Version Algorithm

Data Structures

ADC Rough Values: HWORD [16]
ADC True Values: HWORD [16]
ADC Average: HWORD

Correct the ADC rough values

Define constants for the correction factors
Get the address of the start of the rough value array
Get the address of the start of the true value array
REPEAT
    Load the value from the rough array and post index increment the address
    IF value less than 21
        Correct the ADC value with the correction factor +0
        Branch to storing the corrected ADC value
    IF value less than 40
        Correct the ADC value with the correction factor +1
        Branch to storing the corrected ADC value
    IF value less than 60
        Correct the ADC value with the correction factor +3
        Branch to storing the corrected ADC value

IF value less than 80
Correct the ADC value with the correction factor +7
Branch to storing the corrected ADC value
IF value less than 100
Correct the ADC value with the correction factor +12
Branch to storing the corrected ADC value
IF value less than 120
Correct the ADC value with the correction factor +20
Store the corrected ADC value into the true value array and increment pointer
UNTIL counter equals 0

Calculate the rounded average
Load the address of the true value array
Clear the sum register with 0
REPEAT
Load the first value into the register and post index increment
Add the true value item to the sum register
UNTIL counter equals 0
Divide the sum by 16
Save the result to the Average Value

B. Code for In-line Version

See attached.

C. Procedural Version Algorithm

Main
Get the address of the start of the rough value array
Get the address of the start of the true value array
Set the counter
Call procedure to correct the rough values and calculate the average
END

Procedure to calculate true values and the average

Save the needed register states and link register status onto the stack
Move a copy of the pointer to the true values array into a temp register

REPEAT
Load the value from the rough array and post index increment the address
IF value less than 21 THEN
Correct the ADC value with the correction factor +0
Branch to storing the corrected ADC value
IF value less than 40 THEN
Correct the ADC value with the correction factor +1

Branch to storing the corrected ADC value
IF value less than 60 THEN
Correct the ADC value with the correction factor +3
Branch to storing the corrected ADC value
IF value less than 80 THEN
Correct the ADC value with the correction factor +7
Branch to storing the corrected ADC value
IF value less than 100 THEN
Correct the ADC value with the correction factor +12
Branch to storing the corrected ADC value
IF value less than 120 THEN
Correct the ADC value with the correction factor +20
Store the corrected ADC value into the true value array and increment pointer
UNTIL counter equals 0

Get the address of the average value location in memory
Reset the loop counter
Clear the sum register

REPEAT
Add the value item to the sum register
UNTIL counter equals 0

Divide the sum by 16 to get the average
Round the average value
Store the average value in memory
Restore the states of the registers and load the program counter to return

D. Code for Procedural Version

See attached.

E. Discussion of Testing and Verification

In order to fully verify that the program works correctly the IF-ELSE conditionals to correct the rough values and the rounding average conditional need to be tested. The IF-ELSE conditionals need to be tested for the conditional cases and the edge cases for each range. Table 1 shows the values that need to be tested and what the expected values are.

The average should be rounded up of the tenths place is greater than or equal to 0.5 and should be rounded down if the tenths place is less than 0.5. This also corresponds to whether there is a 1 or a 0 in bit 5 of the byte. Tables 2 and 3 show two different sets of test values and what the expected rounded averages are. Table 2 shows the values to test the algorithm and make sure the average is rounded up when it is supposed to. Table 3 shows the values used to test the algorithm and make sure the average is rounded down when it is supposed to.

Required Tests for Verification

ADC value	Type	Expected Outcome
0	edge	rough val + 0
1 - 19	conditional	rough val + 0
20	edge	rough val + 0
21	edge	rough val + 1
22 - 38	conditional	rough val + 1
39	edge	rough val + 1
40	edge	rough val + 3
41-58	conditional	rough val + 3
59	edge	rough val + 3
60	edge	rough val + 7
61-78	conditional	rough val + 7
79	edge	rough val + 7
80	edge	rough val + 12
81-98	conditional	rough val + 12
99	edge	rough val + 12
100	edge	rough val + 20
101-119	conditional	rough val + 20
120	edge	rough val + 20

Table 1 – Test Values for IF-ELSE conditionals

Test Values and Expected Values When Carry Occurs

Rough Values	0F	1E	32	46	5A	6E	2	14	15	27	28	3B	3C	4F	50	63
Decimal	15	30	50	70	90	110	2	20	21	39	40	59	60	79	80	99
True Values	0F	1F	35	4D	66	82	2	14	16	28	2B	3E	43	56	5C	6F
Decimal	15	31	53	77	102	130	2	20	22	40	43	62	67	86	92	111

Sum	953	3B9
Average	59.6	3C

Table 2 – Test values for conditionals and rounding average up

Test Values and Expected Values When Carry Does Not Occur

Rough Values	0F	1E	32	46	5A	6E	0	14	15	27	28	3B	3C	4F	50	63
Decimal	15	30	50	70	90	110	0	20	21	39	40	59	60	79	80	99
True Values	0F	1F	35	4D	66	82	0	14	16	28	2B	3E	43	56	5C	6F
Decimal	15	31	53	77	102	130	0	20	22	40	43	62	67	86	92	111

Sum	951	3B7
Average	59.4	3B

Table 3 – Test values for conditionals and rounding average down

F. **Debugger Memory Results**

Figure 1 shows the debugger output of the memory state after execution of the program where the rounded average is rounded down or the carry is a zero. Table 2 shows the returned result matches the expected result.

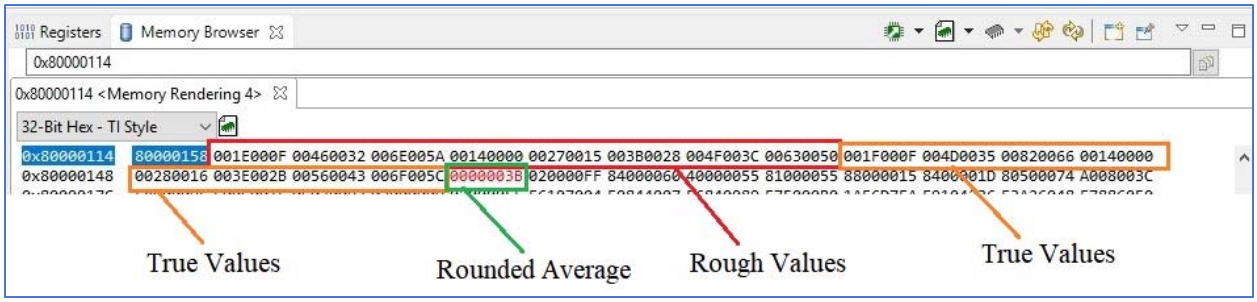

Figure 1 – Debugger Memory Readout for when the rounded average is rounded down

Figure 2 shows the debugger output of the memory state after execution of the program when the rounded average is rounded up or the carry is added to the average. Table 3 shows the returned result matches the expected result.

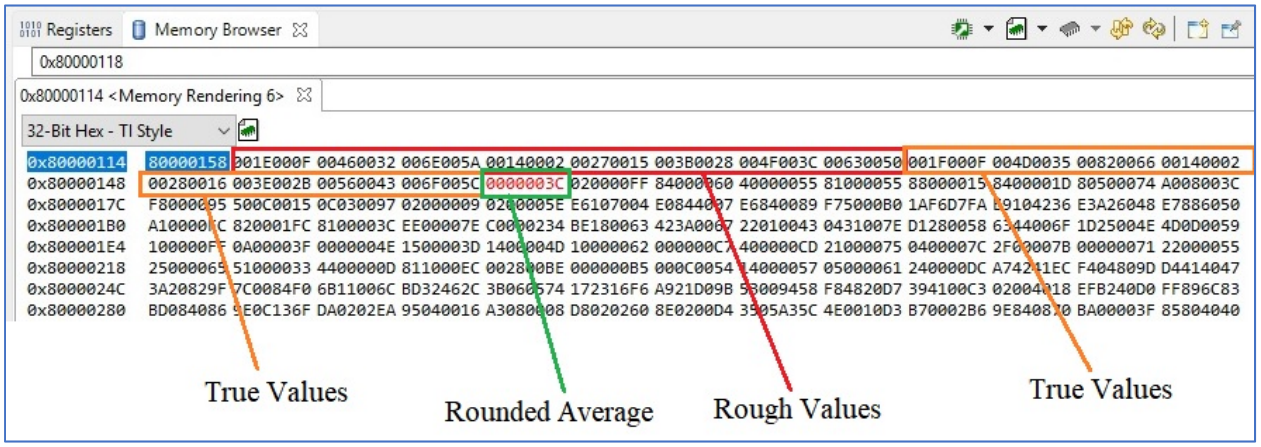

Figure 2 – Debugger Memory Readout for when the rounded average is rounded up

G. Design Log

10/29/17:

- Reviewed the assignment and determined the main tasks of the program.
- Wrote a rough sketch of the tasks and the algorithm to complete them.
- Wrote the first draft of the algorithm for conditioning the values
- Started translating the algorithm to assembly and planning what registers to use
- Researched how to best implement the IF-ELSE structure with minimal branching and translated algorithm to the match what the assembly code will eventually look like
- Converted the algorithm into assembly comments detailing what actions to take and where to put things
- Wrote the in-line assembly language to condition the values and tested the conditionals worked properly and on edge cases

10/30/17:

- Wrote the algorithm to average the values for the in-line version of the program
- Translated the algorithm into assembly language comments and line by line instructions of the order of operations

- Researched the best way to perform the rounding operation and looked at various instructions in the ARM programming manual. Tried to use the BCS and BCC functions with no luck and realized that there was an example in the textbook that showed how to do this. I abandoned the BCS/BCC instructions and switched to the ADC example used in the textbook.
- Had trouble with code composer crashing when I would upload my code to the beaglebone. This was very frustrating and wasted a lot of time until I finally gave up and ported my code to fresh project and the error went away. Something is happening with code composer and the makefile settings getting corrupted somehow because it would give me an error that the makefile had changed whenever I would build the solution.
- Began testing the code and still had trouble with the rounding portion of the code

11/2/2017

- Continued testing the code trying to figure out why the rounding portion was not adding the carry.
- Researched and thought some more about the shifting as a form of division and how that affected the carry flag and realized that the test average I was using was being rounded down because the tens place was less than 5. I switched some values to get an average with a tens place that was greater than .5 and saw that the rounding was indeed working as expected
- Did more tests to double check the carry procedure was working correctly
- Began planning of how to implement the functions as procedures and re-reading the project specifications
- Sketched a rough algorithm of what needed to be done in main and what needed to be done in the procedures and converted the sketch into an algorithm
- Translated the algorithm to assembly language comments and line by line order of operations
- Added the code to call the procedures and translated the in line version of the program to the procedural version
- Tested both carry and no carry conditions to make sure the procedural version of the

11/6/16

- Reviewed project instructions to make sure that the implementation of the program was within design specifications. I realized that I may have misinterpreted the procedural version of the program and had created two separate procedures to do the correction and the averaging. I also misinterpreted the requirements of how the procedure should be accessing memory. I e-mailed Dr. Hall for clarification.

11/15/17

- Fixed the operation of the procedural version given the input from Dr. Hall and a re-interpretation of what the procedure should accomplish. I combined the two procedures into a single procedure that did the correction and the averaging per the instructions.
- Double checked that the code worked correctly and adjusted the algorithm accordingly.
- Finalized report.

H. Statement of Authenticity

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.

Ryan Bentz