|  |
| --- |
| Phong Nguyen |
| Project 1 |
| ECE 371 |

|  |
| --- |
| 11/5/2017 |

**Table of contents**

**Black box 2**

**Pseudocode 2**

**Flow chart 3**

**Truth tables 4**

**Algorithm 5**

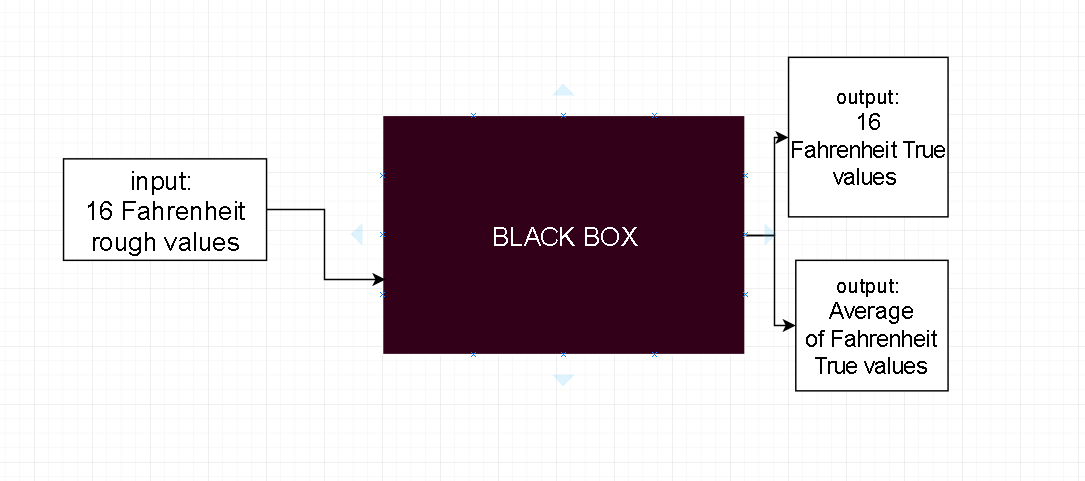
**Part 1 Assembly code 7**

**Part 2 Assembly code 9**

**Assembly code in disassembly 11**

**Screen capture of running program 13**

**BLACK BOX**

First, I have to recognize input and output of the project. Here is are simplify the project into black box 

Clearly, I can see there are 2 outputs. It’s my goal to design the system calculates the output that I want. In order to obtain these outputs, I must need 2 sub routines. They are Conversion and Average.

Once, I realize the subroutine I need to design each sub routine in pseudocode so it can be designed in any language, not just assembly langue.

**PSEUDOCODE**

Conversion

if rough value adding into true value 0

. If rough value > 20 adding 1 to into rough value save in true value

If rough value > 39 adding 3 to into rough value save in true value

If rough value > 59 adding 7 to into rough value save in true value

If rough value > 79 adding 12 to into rough value save in true value

If rough value > 99 adding 20 to into rough value save in true value

Average:

Counter is 16

While the counter is > 0

Previous Sum add true value save into sum

Until counter is 0 next step

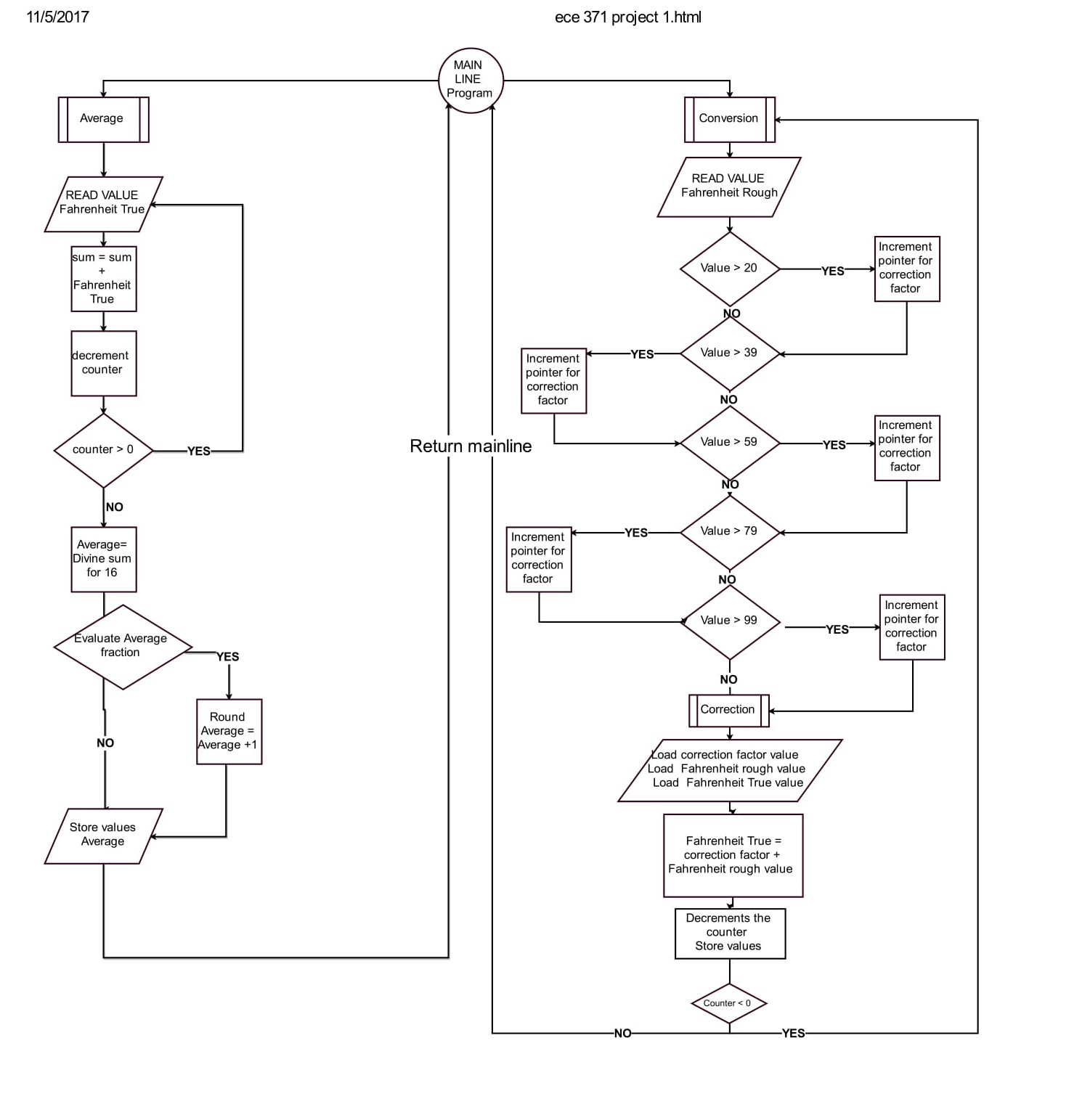
Sum dive it into 16

Check if rounding

Round by adding 1

Else return to main program

**FLOWCHART**



**TRUTH TABLE**

In order to design a perfect system working smoothly without glitch or mistake, I give a trials value and calculate them beforehand. I make a table of these trials value by the time I finish coding I can run them and compare the values from my table to the values output of the program. It’s a really nice technique I figure out when coming across the project problems. This is similar to truth table.

|  |  |  |  |
| --- | --- | --- | --- |
| Fahreinth Rough |  | Expected Fahreinth True |  |
| Decimal | Hex | Decimal | Hex |
| 54 | 0036 | 57 | 0039 |
| 55 | 0037 | 58 | 003A |
| 66 | 0042 | 73 | 0049 |
| 77 | 004D | 84 | 0054 |
| 88 | 0058 | 100 | 0064 |
| 99 | 0063 | 111 | 006F |
| 78 | 004E | 85 | 0055 |
| 63 | 003F | 70 | 0046 |
| 82 | 0052 | 94 | 005E |
| 24 | 0018 | 25 | 0019 |
| 63 | 003F | 70 | 0046 |
| 74 | 004A | 81 | 0051 |
| 84 | 0054 | 96 | 0060 |
| 54 | 0036 | 57 | 0039 |
| 94 | 005E | 106 | 006A |
| 80 | 0050 | 92 | 005C |
|  | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Using offset to know when to rounded |  |  |  |  |
| TRUTH TABLE FOR ROUNDING |  |  |  |  |
| 1 | 1 | r<0.1 | +0 |  |
| 2 | 10 | r<0.5 | +0 |  |
| 3 | 11 | r<0.2 | +0 |  |
| 4 | 100 | r<0.25 | +0 |  |
| 5 | 101 | r<0.35 | +0 |  |
| 6 | 110 | r<0.4 | +0 |  |
| 7 | 111 | r<0.45 | +0 |  |
| 8 | 1000 | r<0.50 | +0 |  |
| 9 | 1001 | r<0.55 | +1 | <- rounding start from here |
| 10 | 1010 | r<0.60 | +1 |  |
| 11 | 1011 | r<0.65 | +1 |  |
| 12 | 1100 | r<0.70 | +1 |  |
| 13 | 1101 | r<0.75 | +1 |  |
| 14 | 1110 | r<0.80 | +1 |  |
| 15 | 1111 | r<0.85 | +1 |  |

|  |  |  |
| --- | --- | --- |
| SUM | 1259 | 04EB |
| Average | 78.6875 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sum in binary |  |  |  |  |
| 0000-0000-0000-0000-0000-0100-1110-1011 | | |  | offset |
| Average value After shift 4 bit | |  |  | 1011 |
| 0000-0000-0000-0000-0000-0000-0100-1110 | | |  |  |

For the rounding part, I write a truth table for associate offset bit. If I just dive the sum to average and ignore the offset bit, I don’t know when I should round the number up or down. This is when the offset bit comes to be part of the solution. I can take advantage of the offset bit to let the program know when it needs to round up the number or round down the number.

According to our flow chart, I can plan ahead by writing an algorithm for the project step by step. Not jump in coding yet because I need to assign the register as the right place to do the work. I can keep track of each register, so I don’t get lost when debugging or coding.

**ALGORTIHM**

Phong Nguyen

Algorithm Project 1 ECE 371

Mainline

Main:

Counter R3= 16

Load input pointer Fahrenheit Rough array into R0 half word

Load input pointer Fahrenheit True array into R1 half word

Load input pointer correction factor array into R2 half word

Load input pointer Average in to R7 word

Calling STEP1 (BL)

NOP

Calling STEP2 (BL)

NOP

STEP1: Save registers into the stack (R0-R9,R14)

Conversion:

Initialize pointer R2 = 0 correction factor

+Condition (Check what kind of A/D reading)

@ Initially R2 pointer is at 0

If (R4 > 20): increment pointer R2 for #2 @ R2= 1

If (R4 > 39): increment pointer R2 for #2 @ R2=2

If (R4 > 59): increment pointer R2 for #2 @ R2= 3

If (R4 > 79): increment pointer R2 for #2 @ R2=4

If (R4 > 99): increment pointer R2 for #2 @ R2=5

Now I have R2 pointer condition

+Correction (Add the correction factor into Rough value)

Load half word R0 into R4 (F rough)

Load half word R1 into R5 (F true)

Load half word R2 into R6 (correction factor)

R5 = R4 + R6 @ Adding correction

Store R5 back R1 array

R0 increment pointer into #2 @ R0++ for the next rough value

R1 increment pointer into #2 @ R2++ for the next true value

R3 = R3 – 1 @ decrement counter

Keep calling conversion BNE Conversion

Restore the register by popping from the stack (R0-R9, R14)

Move PC, R14

STEP2: Save registers into the stack (R0-R9,R14)

Average:

Load half word R1 into R5 (F true)

Load word R7 into R8 a word (average)

R6= R5+R6 @ adding for sum

R1 increment pointer into #2 @ R1 ++ for next true value

R3=R3-1 @decrement R3

If R3 = 0 then next otherwise keep calling Average

R6 is now the Sum of 16 F true Values

R9 will mask 4 loIr bit of R6 (R9 will decide how I should rounded for average number)

R8 = Shift the values to the right 4 bits (This is same as take the value dive by 16)

Compare R9 and 0x8 if update N flags and Z flags

If PL plus positive or zero (N clear) Called rounded

END: Restore the register by popping from the stack (R0-R9, R14)

Return to mainline program MOV PC, R14

Rounded:

R8 = R8+1 @ rounded R8 by adding one (round up)

Jump to end

Now it’s time for coding. The code will be attached at the end of the project slide, along with the disassembly code. As I run the program here is the screen capture for the running program. I can use the values from the memory dumb compare to the table I have after calculating

**PART 1 ASSEMBLY CODE (ONLY SUBROUTINE)**

CONVERSION:

LDR R2,=CORRECTION\_FACTOR @Reset the pointer for CORRECTION\_FACTOR

LDR R4,[R0] @Load half word into R0 into R4

CMP R4,#20 @if R4<=20 N and Z is set 1

BLE CORRECTION @Branch if less or equal

ADD R2,R2,#4 @increment pointer 1 R2=1

CMP R4,#39 @if R4<=39 N and Z is set 1

BLE CORRECTION @Load half word R0 into R4 (F rough)

ADD R2,R2,#4 @increment pointer 1 R2=2

CMP R4,#59 @if R4<=59 N and Z is set 1

BLE CORRECTION @Load half word R0 into R4 (F rough)

ADD R2,R2,#4 @increment pointer 1 R2=3

CMP R4,#79 @if R4<=79 N and Z is set 1

BLE CORRECTION @Load half word R0 into R4 (F rough)

ADD R2,R2,#4 @increment pointer 1 R2=4

CMP R4,#99 @if R4<=99 N and Z is set 1

BLE CORRECTION @Load half word R0 into R4 (F rough)

ADD R2,R2,#4 @increment pointer 1 R2=5

CMP R4,#120 @if R4<=120 N and Z is set 1

BLE CORRECTION

CORRECTION:

MOV R5,#0x0 @clear R5 buffer

MOV R6,#0x0 @clear R6 buffer

LDR R5,[R1] @ R5 (F True) get value from R1

LDR R6,[R2] @ R6 (Correction factor)get value from R2

ADD R5,R4,R6 @Add the correction into rough value

STR R5,[R1] @store R5 into R1 array

ADD R0,R0,#4 @increment R0 pointer

ADD R1,R1,#4 @increment R1 pointer

SUBS R3,R3,#1 @decrement R3 Counter

BNE CONVERSION

LDMFD R13!,{R0-R9,R14} @Pop the stack

MOV PC,R14

AVERAGE:

LDR R5,[R1] @Load half word R1 into R5

ADD R6,R6,R5 @Adding sum = sum + F true

ADD R1,R1,#4 @increment F true pointer

SUBS R3,R3,#1 @decrement the counter

BNE AVERAGE @branch if z is not set 1

LDR R8,[R7] @Load R8 into R7

AND R9,R6,#0x000f @Masking 4 lower bits and store in R9

MOV R8,R6,LSR #4 @shift 4 bit position in to the right

CMP R9,#0x8 @compare the mask bit

BPL ROUNDED @IF PL plus positive or z (N clear)

NOP

END:

STR R8,[R7] @store R8 into R7

LDMFD R13!,{R0-R9,R14} @Pop the stack

MOV PC,R14

ROUNDED:

ADD R8,R8,#1 @rounded R8

B END

**PART 2 ASSEMBLY CODE (COMPLETE PROGRAM)**

@Project 1

@ECE 371

@ Phong Nguyen October 2017

.text @start of the program code section

.global \_start

\_start:

@R0-R12 general purpose regs

@R13 is stack pointer (SP)

@R14 is link register (LR)

@R15 is used a program counter (PC)

.equ NUM,16

LDR R0,=FAHRENHEIT\_ROUGH

LDR R1,=FAHRENHEIT\_TRUE

LDR R2,=CORRECTION\_FACTOR

MOV R3,#NUM @set the counter 16

MOV R4,#0x0 @initialize R4 buffer

MOV R5,#0x0 @initialize R5 buffer

MOV R6,#0x0 @initialize R6 buffer

LDR R7,=ROUNDED\_AVERAGE

MOV R8,#0x0

MOV R9,#0x0

LDR R13,=STACK

ADD R13,R13,#0x100

BL STEP1

NOP

BL STEP2

NOP

@-----------------------------------------------------------------------------------------------

@-----------------------------------------------------------------------------------------------

@-----------------------------------------------------------------------------------------------

STEP1: STMFD R13!, {R0-R9,R14}

CONVERSION:

LDR R2,=CORRECTION\_FACTOR @Reset the pointer for CORRECTION\_FACTOR

LDR R4,[R0] @Load half word into R0 into R4

CMP R4,#20 @if R4<=20 N and Z is set 1

BLE CORRECTION @Branch if less or equal

ADD R2,R2,#4 @increment pointer 1 R2=1

CMP R4,#39 @if R4<=39 N and Z is set 1

BLE CORRECTION @Load half word R0 into R4 (F rough)

ADD R2,R2,#4 @increment pointer 1 R2=2

CMP R4,#59 @if R4<=59 N and Z is set 1

BLE CORRECTION @Load half word R0 into R4 (F rough)

ADD R2,R2,#4 @increment pointer 1 R2=3

CMP R4,#79 @if R4<=79 N and Z is set 1

BLE CORRECTION @Load half word R0 into R4 (F rough)

ADD R2,R2,#4 @increment pointer 1 R2=4

CMP R4,#99 @if R4<=99 N and Z is set 1

BLE CORRECTION @Load half word R0 into R4 (F rough)

ADD R2,R2,#4 @increment pointer 1 R2=5

CMP R4,#120 @if R4<=120 N and Z is set 1

BLE CORRECTION

CORRECTION:

MOV R5,#0x0 @clear R5 buffer

MOV R6,#0x0 @clear R6 buffer

LDR R5,[R1] @ R5 (F True) get value from R1

LDR R6,[R2] @ R6 (Correction factor)get value from R2

ADD R5,R4,R6 @Add the correction into rough value

STR R5,[R1] @store R5 into R1 array

ADD R0,R0,#4 @increment R0 pointer

ADD R1,R1,#4 @increment R1 pointer

SUBS R3,R3,#1 @decrement R3 Counter

BNE CONVERSION

LDMFD R13!,{R0-R9,R14} @Pop the stack

MOV PC,R14

@-----------------------------------------------------------------------------------------------

@-----------------------------------------------------------------------------------------------

@-----------------------------------------------------------------------------------------------

STEP2: STMFD R13!, {R0-R9,R14}

AVERAGE:

LDR R5,[R1] @Load half word R1 into R5

ADD R6,R6,R5 @Adding sum = sum + F true

ADD R1,R1,#4 @increment F true pointer

SUBS R3,R3,#1 @decrement the counter

BNE AVERAGE @branch if z is not set 1

LDR R8,[R7] @Load R8 into R7

AND R9,R6,#0x000f @Masking 4 lower bits and store in R9

MOV R8,R6,LSR #4 @shift 4 bit position in to the right

CMP R9,#0x8 @compare the mask bit

BPL ROUNDED @IF PL plus positive or z (N clear)

NOP

END:

STR R8,[R7] @store R8 into R7

LDMFD R13!,{R0-R9,R14} @Pop the stack

MOV PC,R14

ROUNDED:

ADD R8,R8,#1 @rounded R8

B END

.data

FAHRENHEIT\_ROUGH: .word 0x36,0x37,0x42,0x4D,0X58,0X63,0X4E,0X3F,0X52,0X18,0X3F,0X4A,0X54,0X36,0X5E,0X50

FAHRENHEIT\_TRUE: .word 0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0,0x0

CORRECTION\_FACTOR: .word 0x0,0x1,0x3,0x7,0xc,0x14

ROUNDED\_AVERAGE: .word 0x0

STACK: .rept 256

.byte 0x00

.endr

.end

**ASSEMBLY CODES IN DISASSEMBLY**

\_start():

80000070: E59F00FC ldr r0, [pc, #0xfc]

14 LDR R1,=FAHRENHEIT\_TRUE

80000074: E59F10FC ldr r1, [pc, #0xfc]

15 LDR R2,=CORRECTION\_FACTOR

80000078: E59F20FC ldr r2, [pc, #0xfc]

16 MOV R3,#NUM @set the counter 16

8000007c: E3A03010 mov r3, #0x10

17 MOV R4,#0x0 @initialize R4 buffer

80000080: E3A04000 mov r4, #0

18 MOV R5,#0x0 @initialize R5 buffer

80000084: E3A05000 mov r5, #0

19 MOV R6,#0x0 @initialize R6 buffer

80000088: E3A06000 mov r6, #0

20 LDR R7,=ROUNDED\_AVERAGE

8000008c: E59F70EC ldr r7, [pc, #0xec]

21 MOV R8,#0x0

80000090: E3A08000 mov r8, #0

22 MOV R9,#0x0

80000094: E3A09000 mov r9, #0

23 LDR R13,=STACK

80000098: E59FD0E4 ldr sp, [pc, #0xe4]

24 ADD R13,R13,#0x100

8000009c: E28DDC01 add sp, sp, #0x100

26 BL STEP1

800000a0: EB000002 bl #0x800000b0

27 NOP

800000a4: E320F000 nop

28 BL STEP2

800000a8: EB000020 bl #0x80000130

29 NOP

800000ac: E320F000 nop

33 STEP1: STMFD R13!, {R0-R9,R14}

800000b0: E92D43FF push {r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, lr}

35 LDR R2,=CORRECTION\_FACTOR @Reset the pointer for CORRECTION\_FACTOR

800000b4: E59F20C0 ldr r2, [pc, #0xc0]

36 LDR R4,[R0] @Load half word into R0 into R4

800000b8: E5904000 ldr r4, [r0]

37 CMP R4,#20 @if R4<=20 N and Z is set 1

800000bc: E3540014 cmp r4, #0x14

38 BLE CORRECTION @Branch if less or equal

800000c0: DA00000E ble #0x80000100

39 ADD R2,R2,#4 @increment pointer 1 R2=1

800000c4: E2822004 add r2, r2, #4

40 CMP R4,#39 @if R4<=39 N and Z is set 1

800000c8: E3540027 cmp r4, #0x27

41 BLE CORRECTION @Load half word R0 into R4 (F rough)

800000cc: DA00000B ble #0x80000100

42 ADD R2,R2,#4 @increment pointer 1 R2=2

800000d0: E2822004 add r2, r2, #4

43 CMP R4,#59 @if R4<=59 N and Z is set 1

800000d4: E354003B cmp r4, #0x3b

44 BLE CORRECTION @Load half word R0 into R4 (F rough)

800000d8: DA000008 ble #0x80000100

45 ADD R2,R2,#4 @increment pointer 1 R2=3

800000dc: E2822004 add r2, r2, #4

46 CMP R4,#79 @if R4<=79 N and Z is set 1

800000e0: E354004F cmp r4, #0x4f

47 BLE CORRECTION @Load half word R0 into R4 (F rough)

800000e4: DA000005 ble #0x80000100

48 ADD R2,R2,#4 @increment pointer 1 R2=4

800000e8: E2822004 add r2, r2, #4

49 CMP R4,#99 @if R4<=99 N and Z is set 1

800000ec: E3540063 cmp r4, #0x63

50 BLE CORRECTION @Load half word R0 into R4 (F rough)

800000f0: DA000002 ble #0x80000100

51 ADD R2,R2,#4 @increment pointer 1 R2=5

800000f4: E2822004 add r2, r2, #4

52 CMP R4,#120 @if R4<=120 N and Z is set 1

800000f8: E3540078 cmp r4, #0x78

53 BLE CORRECTION

800000fc: DAFFFFFF ble #0x80000100

56 MOV R5,#0x0 @clear R5 buffer

80000100: E3A05000 mov r5, #0

57 MOV R6,#0x0 @clear R6 buffer

80000104: E3A06000 mov r6, #0

58 LDR R5,[R1] @ R5 (F True) get value from R1

80000108: E5915000 ldr r5, [r1]

59 LDR R6,[R2] @ R6 (Correction factor)get value from R2

8000010c: E5926000 ldr r6, [r2]

60 ADD R5,R4,R6 @Add the correction into rough value

80000110: E0845006 add r5, r4, r6

61 STR R5,[R1] @store R5 into R1 array

80000114: E5815000 str r5, [r1]

62 ADD R0,R0,#4 @increment R0 pointer

80000118: E2800004 add r0, r0, #4

63 ADD R1,R1,#4 @increment R1 pointer

8000011c: E2811004 add r1, r1, #4

64 SUBS R3,R3,#1 @decrement R3 Counter

80000120: E2533001 subs r3, r3, #1

65 BNE CONVERSION

80000124: 1AFFFFE2 bne #0x800000b4

66 LDMFD R13!,{R0-R9,R14} @Pop the stack

80000128: E8BD43FF pop {r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, lr}

67 MOV PC,R14

8000012c: E1A0F00E mov pc, lr

71 STEP2: STMFD R13!, {R0-R9,R14}

80000130: E92D43FF push {r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, lr}

73 LDR R5,[R1] @Load half word R1 into R5

80000134: E5915000 ldr r5, [r1]

74 ADD R6,R6,R5 @Adding sum = sum + F true

80000138: E0866005 add r6, r6, r5

75 ADD R1,R1,#4 @increment F true pointer

8000013c: E2811004 add r1, r1, #4

76 SUBS R3,R3,#1 @decrement the counter

80000140: E2533001 subs r3, r3, #1

80000144: 1AFFFFFA bne #0x80000134

80000148: E5978000 ldr r8, [r7]

8000014c: E206900F and r9, r6, #0xf

80000150: E1A08226 lsr r8, r6, #4

80000154: E3590008 cmp r9, #8

80000158: 5A000003 bpl #0x8000016c

8000015c: E320F000 nop

80000160: E5878000 str r8, [r7]

80000164: E8BD43FF pop {r0, r1, r2, r3, r4, r5, r6, r7, r8, r9, lr}

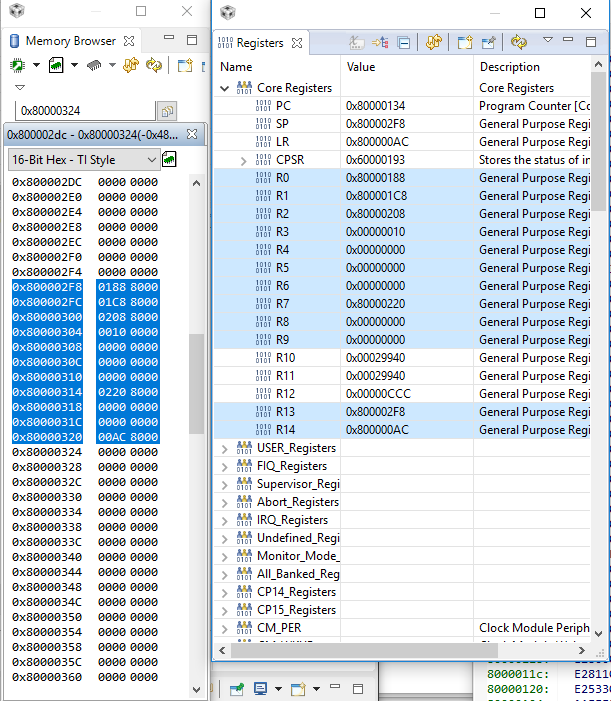
80000168: E1A0F00E mov pc, lr

8000016c: E2888001 add r8, r8, #1

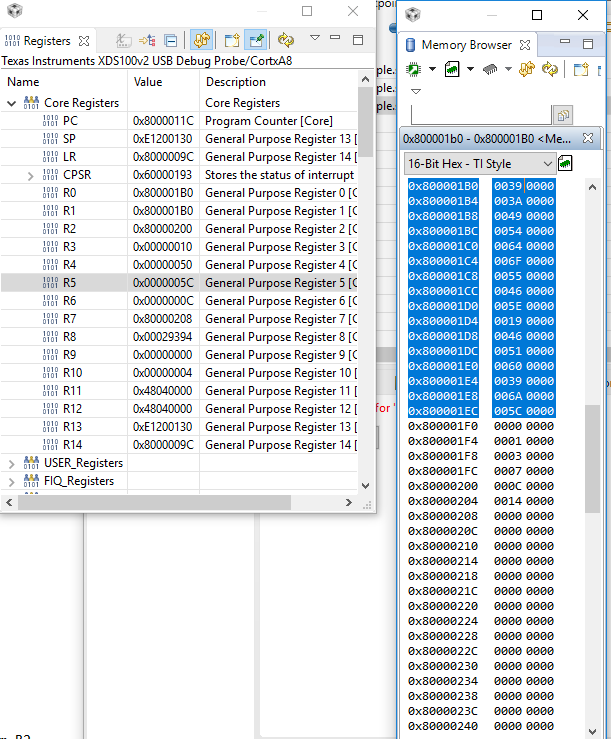
80000170: EAFFFFFA b #0x80000160

**Screen capture of running program**

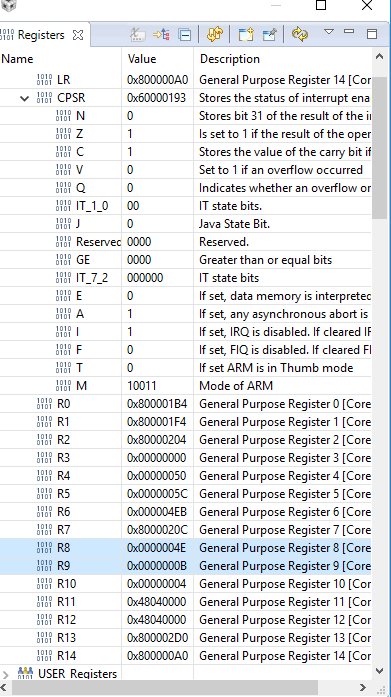
**My stack in the memory dumb**



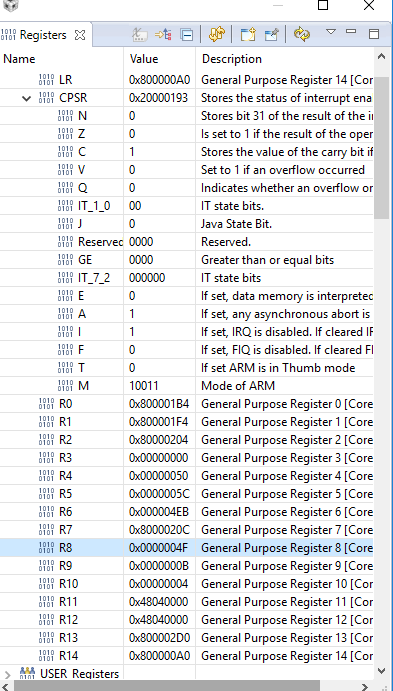
**Fahrenheit True in the memory dumb**



**Offset R9 and not rounded Average R8**



**Rounded Average R8**



**Fahrenheit True being store in R6**

