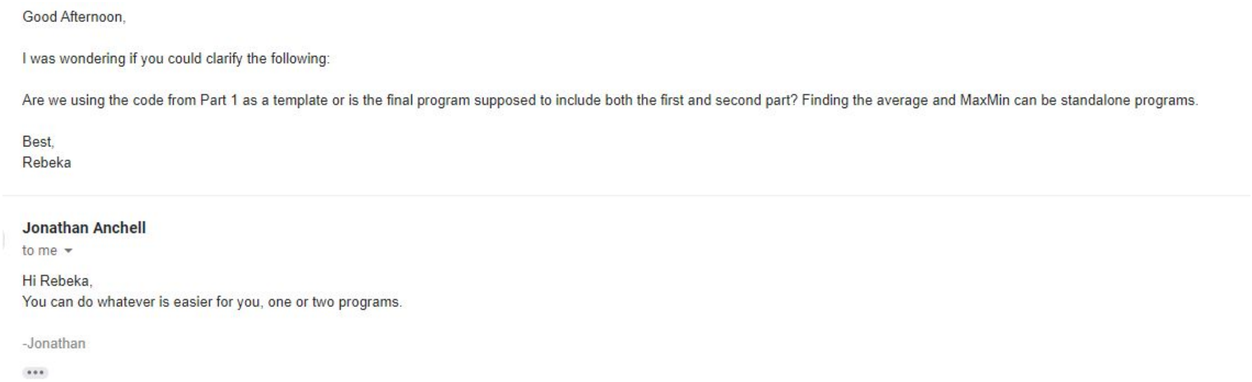
February 11, 2020

Problem

-measure temperature every 1.5 hours in a 24 hour period.
-analog to digital converter produces the 8 bit binary values (to represent temperature in Fahrenheit)
-temperature range: 0-140 degrees
-A/D values (16 temperatures): stored in an array of bytes named Fahrenheit_Temps

NOTE: The professor gave me the go ahead to create two separate standalone programs.



Good Afternoon,

I was wondering if you could clarify the following:

Are we using the code from Part 1 as a template or is the final program supposed to include both the first and second part? Finding the average and MaxMin can be standalone programs.

Best,
Rebeka

Jonathan Anchell
to me

Hi Rebeka,
You can do whatever is easier for you, one or two programs.

-Jonathan

To do: perform calculations on the 16 temperature values in the array

Task 1: calculate the rounded average of 16 temperature values and put the result in a named memory location

Task 2: when the average is working correctly, develop and call a procedure that determines the minimum and maximum temperatures for the day and returns these to the mainline to be put in named memory locations

Part 1:

Brainstorm session for task 1-

Develop the algorithm for the rounded average of 16 temperature values and put the result in a named memory location.

- Separate the two operations of addition and division
- Needs a way to extract those temperature values and place them in an array of temp values (Fahrenheit_Temps). Note: page 154 shows the way to find the average

Algorithm based on programming structures: Draft 1

- Read all 16 8 bit binary values from the temperature sensor **(not a necessary step)**
- Place all the 16 8 bit binary values into the array Fahrenheit_Temps **(repeat until structure)**
- Loop through the array based on the number of values within the array **(16)**
- Add all of the values within the array Fahrenheit_Temps together. Store the addition in a temporary variable
- Divide the temp variable by the number of values within the array to get the average
- Put the result **(average)** in a named memory location

Actual Variables needed:

- Fahrenheit_Temps ---> should hold 16 8 bit values
- Average ---> should be a 1 byte array
- Number of times looping through the array is 16 times and each time we are adding an element. (i+1)

Modified Algorithm:

- Initialize pointers to the Fahrenheit_Temps and Average arrays
- Initialize the element counter to 16
- REPEAT
    - Load byte at element i and add it to the next byte at element i + 1
    - Store all of the bytes added to a temporary register (this register is updated with each addition)

  - Divide sum by 16 to get the average, set the carry flag
  - Round the average: if C = 1, then add 1 to the result in the temporary register
  - Store the rounded average in the average array
- Until all 16 elements are done

<u>**Challenge:**</u> how to add array elements together in assembly. One solution could be to break the array in half or have 2 additional arrays that are then added together. **OR just stick to the above algorithm**

February 12, 2020

Multo Program (page 115)

It has a pointer to an array and it uses MOV for a loop counter. It has a store command and uses a label to branch. Also increments the pointer to the next value using add. It decrements the counter but increments the points to the values in the array and breaks out of the loop using NOP

Potential instructions and Labels: LDRB, MOVS, SUBS, LSR, ADC, ADDS, LDR, LSR
LSR = shift right register… 2^n = 2 ^ 4 = 16. Shift 4 bits to the right for the division part of algorithm

Values using initially: 0, 140, 57, 28, 100, 103, 33, 45, 88, 62, 73, 29, 120, 97, 13, 59

1047/16 = 65.4375

Answer: 65

The code

```
.txt
.global _start
.Equ NUM, 16

               LDR R1, = Fahrenheit_Temps @comment
               LDR R2, = Average          @comment
               MOV R3, #NUM
NEXT:          LDRB R6, [R1], #1
               ADDS R7, R6                @i+1?
               MOVS R7, R7, LSR #4        @divide?
               ADC R7,R7, #0
               STRB R5, [R2], #1
               SUBS R3, R3, #1
               BNE NEXT
               NOP

.data
Fahrenheit_Temps: .byte 0b10001100, 0b00000000, 0b00111001....
Average: .byte 0b00000000
.end
```

Issues I already see

I may have put the average in the wrong spot. I want to divide after and I don’t think it should be within the loop so...

MOVS R7, R7, LSR #4 **should be outside of the loop**
ADC R7,R7, #0 **should be outside of the loop**

Store and load just one data point as the average. Maybe the current way it is ok but just don’t leave it in the loop

STRB R5, [R2], #1 **should be outside of the loop**

SUBS R3, R3, #1 **ok where it is**
BNE NEXT **ok where it is**

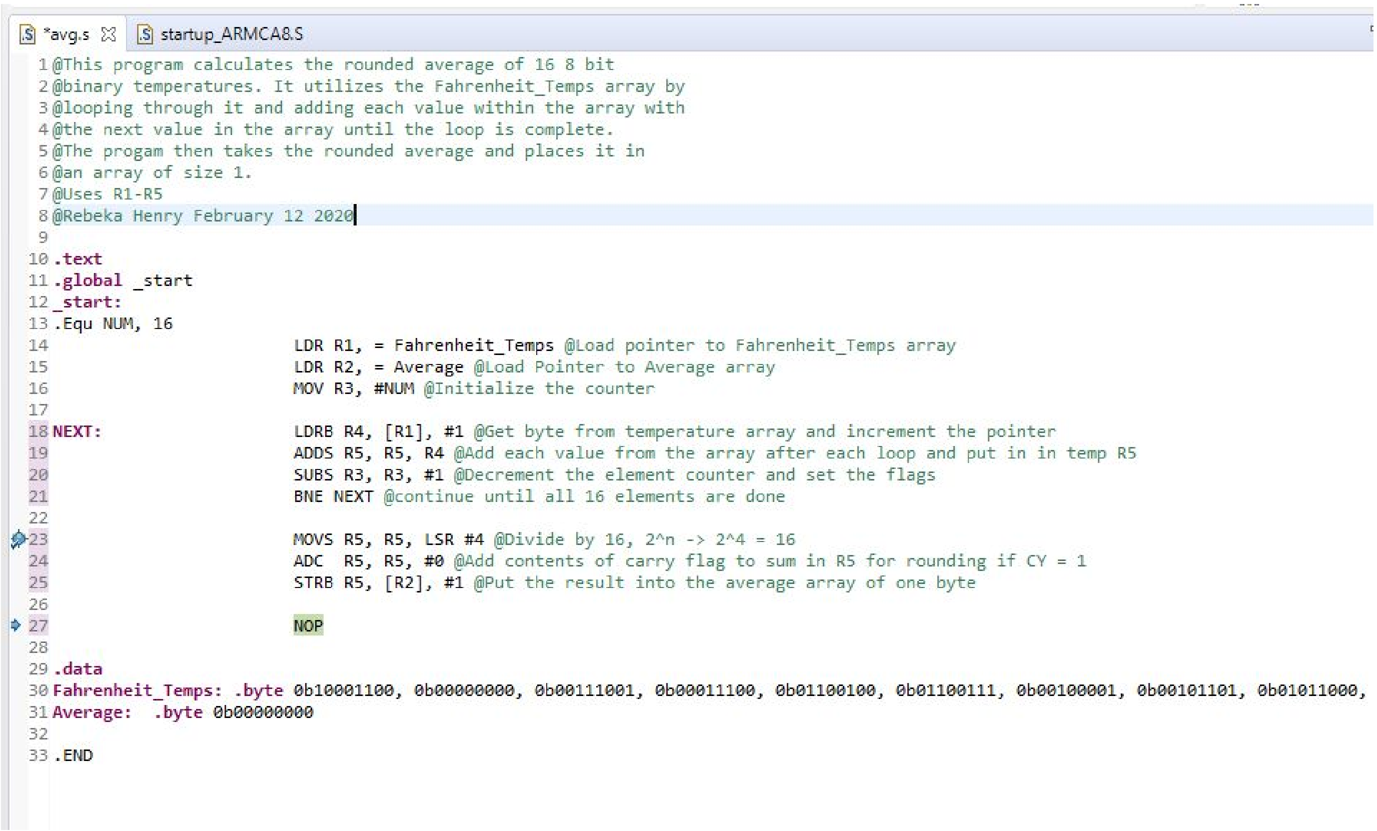
Also, change the registers used to: R1-R4

Modified Code

```
.txt
.global _start
_start:
.Equ NUM, 16

               LDR R1, = Fahrenheit_Temps @load ptr
               LDR R2, = Average          @load ptr
               MOV R3, #NUM               @init counter

NEXT:          LDRB R4, [R1], #1    @get byte from and increment
ptr
               ADDS R5, R5, R4          @add val after each loop
               SUBS R3, R3, #1          @decrement element cnter
               BNE NEXT                 @continue for all 16 elm

               MOVS R5, R5, LSR #4 @divide by 16, 2^n->2^4 = 16
               ADC R5, R5, #0 @add contents of crry flg to sum
               STRB R5, [R2], #1   @put result into average arr
of one byte

               NOP

.data
Fahrenheit_Temps: .byte - - - - - - - 
Average: .byte ---
```

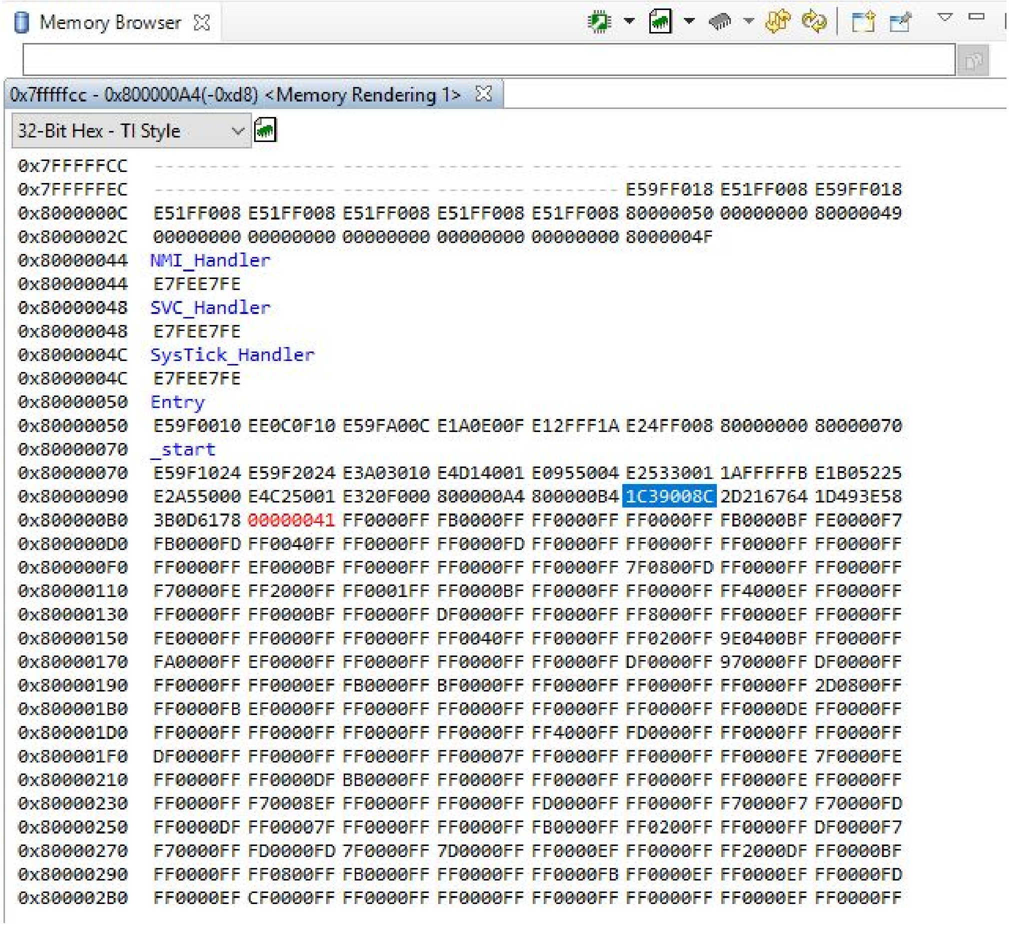
February 14, 2020

Test 1 Code 1:



```
@This program calculates the rounded average of 16 8 bit
@binary temperatures. It utilizes the Fahrenheit_Temps array by
@looping through it and adding each value within the array with
@the next value in the array until the loop is complete.
@The progam then takes the rounded average and places it in
@an array of size 1.
@Uses R1-R5
@Rebeka Henry February 12 2020

.text
.global _start
_start:
.Equ NUM, 16
                        LDR R1, = Fahrenheit_Temps @Load pointer to Fahrenheit_Temps array
                        LDR R2, = Average @Load Pointer to Average array
                        MOV R3, #NUM @Initialize the counter

NEXT:                   LDRB R4, [R1], #1 @Get byte from temperature array and increment the pointer
                        ADDS R5, R5, R4 @Add each value from the array after each loop and put in in temp R5
                        SUBS R3, R3, #1 @Decrement the element counter and set the flags
                        BNE NEXT @continue until all 16 elements are done

                        MOVS R5, R5, LSR #4 @Divide by 16, 2^n -> 2^4 = 16
                        ADC  R5, R5, #0 @Add contents of carry flag to sum in R5 for rounding if CY = 1
                        STRB R5, [R2], #1 @Put the result into the average array of one byte

                        NOP

.data
Fahrenheit_Temps: .byte 0b10001100, 0b00000000, 0b00111001, 0b00011100, 0b01100100, 0b01100111, 0b00100001, 0b00101101, 0b01011000,
Average:  .byte 0b00000000

.END
```

Test 1 Memory Browser 1:



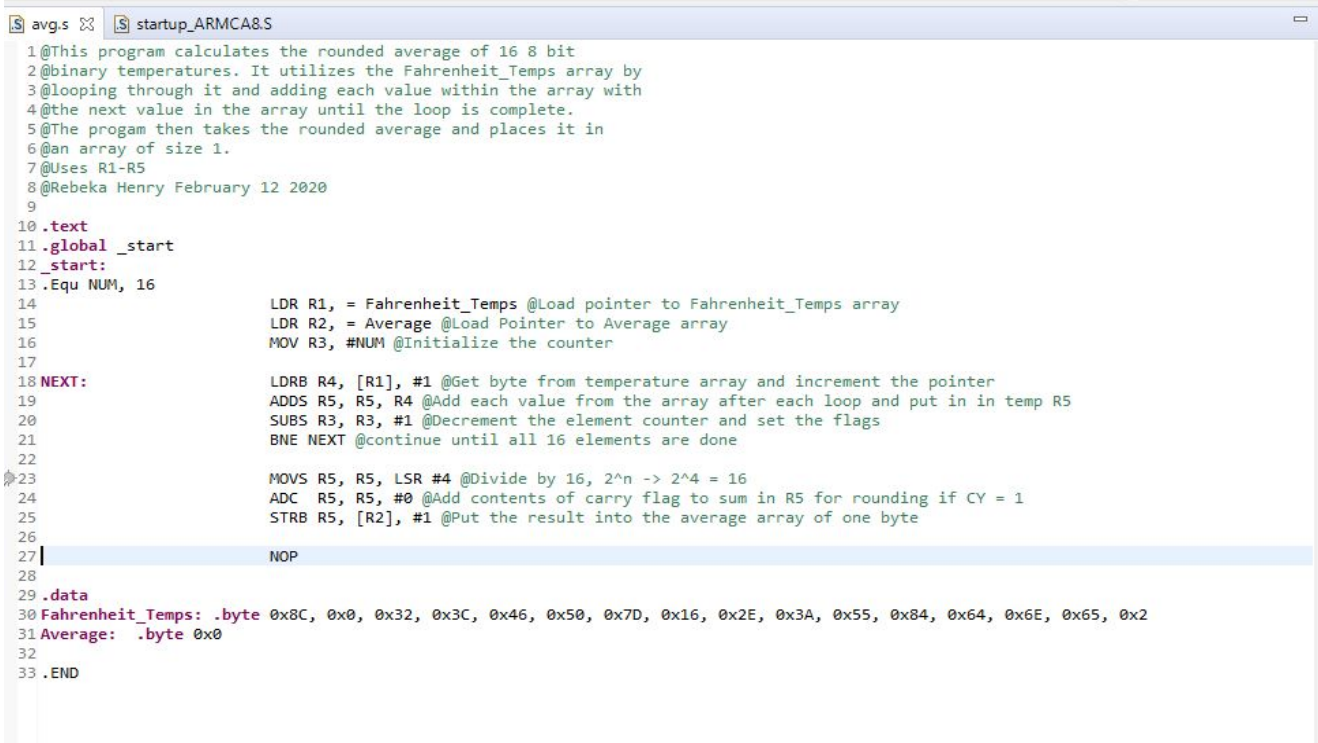
```
0x7FFFFFCC   -------- -------- -------- -------- -------- -------- -------- --------
0x7FFFFFEC   -------- -------- -------- -------- -------- E59FF018 E51FF008 E59FF018
0x8000000C   E51FF008 E51FF008 E51FF008 E51FF008 E51FF008 80000050 00000000 80000049
0x8000002C   00000000 00000000 00000000 00000000 00000000 8000004F
0x80000044  NMI_Handler
0x80000044   E7FEE7FE
0x80000048  SVC_Handler
0x80000048   E7FEE7FE
0x8000004C  SysTick_Handler
0x8000004C   E7FEE7FE
0x80000050  Entry
0x80000050   E59F0010 EE0C0F10 E59FA00C E1A0E00F E12FFF1A E24FF008 80000000 80000070
0x80000070  _start
0x80000070   E59F1024 E59F2024 E3A03010 E4D14001 E0955004 E2533001 1AFFFFFB E1B05225
0x80000090   E2A55000 E4C25001 E320F000 800000A4 800000B4 1C39008C 2D216764 1D493E58
0x800000B0   3B0D6178 00000041 FF0000FF FB0000FF FF0000FF FF0000FF FB0000BF FE0000F7
0x800000D0   FB0000FD FF0040FF FF0000FF FF0000FD FF0000FF FF0000FF FF0000FF FF0000FF
0x800000F0   FF0000FF EF0000BF FF0000FF FF0000FF FF0000FF 7F0800FD FF0000FF FF0000FF
0x80000110   F70000FE FF2000FF FF0001FF FF0000BF FF0000FF FF0000FF FF4000EF FF0000FF
0x80000130   FF0000FF FF0000BF FF0000FF DF0000FF FF0000FF FF8000FF FF0000EF FF0000FF
0x80000150   FE0000FF FF0000FF FF0000FF FF0040FF FF0000FF FF0200FF 9E0400BF FF0000FF
0x80000170   FA0000FF EF0000FF FF0000FF FF0000FF FF0000FF DF0000FF 970000FF DF0000FF
0x80000190   FF0000FF FF0000EF FB0000FF BF0000FF FF0000FF FF0000FF FF0000FF 2D0800FF
0x800001B0   FF0000FB EF0000FF FF0000FF FF0000FF FF0000FF FF0000FF FF0000DE FF0000FF
0x800001D0   FF0000FF FF0000FF FF0000FF FF0000FF FF4000FF FD0000FF FF0000FF FF0000FF
0x800001F0   DF0000FF FF0000FF FF0000FF FF00007F FF0000FF FF0000FF FF0000FE 7F0000FE
0x80000210   FF0000FF FF0000DF BB0000FF FF0000FF FF0000FF FF0000FF FF0000FE FF0000FF
0x80000230   FF0000FF F70008EF FF0000FF FF0000FF FD0000FF FF0000FF F70000F7 F70000FD
0x80000250   FF0000DF FF00007F FF0000FF FF0000FF FB0000FF FF0200FF FF0000FF DF0000F7
0x80000270   F70000FF FD0000FD 7F0000FF 7D0000FF FF0000EF FF0000FF FF2000DF FF0000BF
0x80000290   FF0000FF FF0800FF FB0000FF FF0000FF FF0000FB FF0000EF FF0000EF FF0000FD
0x800002B0   FF0000EF CF0000FF FF0000FF FF0000FF FF0000FF FF0000FF FF0000EF FF0000FF
```

Value 00000041 = 65 in decimal
The algorithm worked for a case where there wasn't any rounding of the temperature. Test 1 succeeded

Test 2 Code 2

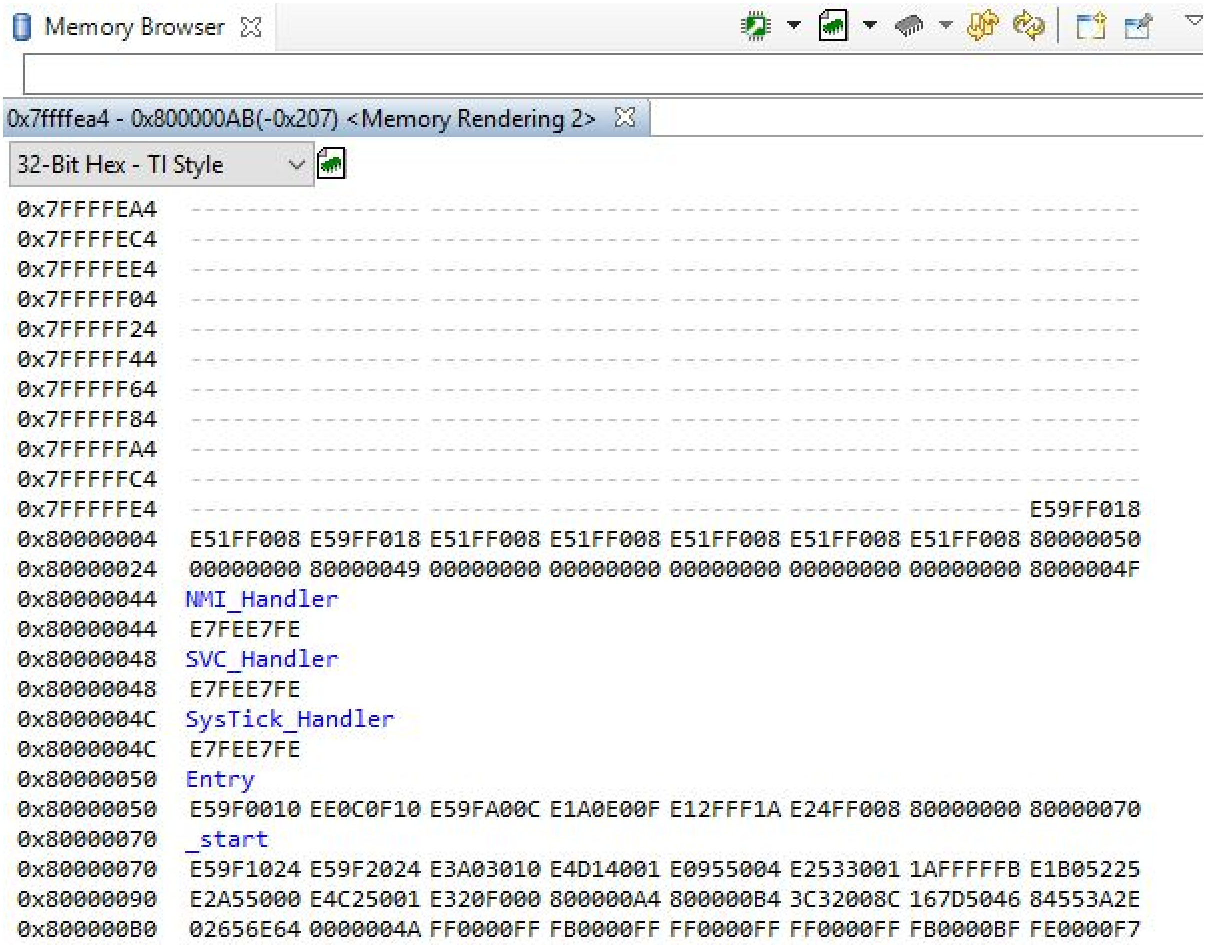
-Instead of binary, the values stored are in hex for readability
-Decimal values supplied: 0, 140, 50, 60, 70, 80, 125, 22, 46, 58, 85, 132, 100, 110, 101, 2

1181/4 = 73.8125
Rounded: 74



```
@This program calculates the rounded average of 16 8 bit
@binary temperatures. It utilizes the Fahrenheit_Temps array by
@looping through it and adding each value within the array with
@the next value in the array until the loop is complete.
@The progam then takes the rounded average and places it in
@an array of size 1.
@Uses R1-R5
@Rebeka Henry February 12 2020

.text
.global _start
_start:
.Equ NUM, 16
                        LDR R1, = Fahrenheit_Temps @Load pointer to Fahrenheit_Temps array
                        LDR R2, = Average @Load Pointer to Average array
                        MOV R3, #NUM @Initialize the counter

NEXT:                   LDRB R4, [R1], #1 @Get byte from temperature array and increment the pointer
                        ADDS R5, R5, R4 @Add each value from the array after each loop and put in in temp R5
                        SUBS R3, R3, #1 @Decrement the element counter and set the flags
                        BNE NEXT @continue until all 16 elements are done

                        MOVS R5, R5, LSR #4 @Divide by 16, 2^n -> 2^4 = 16
                        ADC  R5, R5, #0 @Add contents of carry flag to sum in R5 for rounding if CY = 1
                        STRB R5, [R2], #1 @Put the result into the average array of one byte

                        NOP

.data
Fahrenheit_Temps: .byte 0x8C, 0x0, 0x32, 0x3C, 0x46, 0x50, 0x7D, 0x16, 0x2E, 0x3A, 0x55, 0x84, 0x64, 0x6E, 0x65, 0x2
Average:  .byte 0x0

.END
```

Test 2 Memory Browser 2



```
0x7ffffea4 - 0x800000AB(-0x207) <Memory Rendering 2>
32-Bit Hex - TI Style
0x7FFFFEA4   -------- -------- -------- -------- -------- -------- -------- --------
0x7FFFFEC4   -------- -------- -------- -------- -------- -------- -------- --------
0x7FFFFEE4   -------- -------- -------- -------- -------- -------- -------- --------
0x7FFFFF04   -------- -------- -------- -------- -------- -------- -------- --------
0x7FFFFF24   -------- -------- -------- -------- -------- -------- -------- --------
0x7FFFFF44   -------- -------- -------- -------- -------- -------- -------- --------
0x7FFFFF64   -------- -------- -------- -------- -------- -------- -------- --------
0x7FFFFF84   -------- -------- -------- -------- -------- -------- -------- --------
0x7FFFFFA4   -------- -------- -------- -------- -------- -------- -------- --------
0x7FFFFFC4   -------- -------- -------- -------- -------- -------- -------- --------
0x7FFFFFE4   -------- -------- -------- -------- -------- -------- -------- E59FF018
0x80000004   E51FF008 E59FF018 E51FF008 E51FF008 E51FF008 E51FF008 E51FF008 80000050
0x80000024   00000000 80000049 00000000 00000000 00000000 00000000 00000000 8000004F
0x80000044  NMI_Handler
0x80000044   E7FEE7FE
0x80000048  SVC_Handler
0x80000048   E7FEE7FE
0x8000004C  SysTick_Handler
0x8000004C   E7FEE7FE
0x80000050  Entry
0x80000050   E59F0010 EE0C0F10 E59FA00C E1A0E00F E12FFF1A E24FF008 80000000 80000070
0x80000070  _start
0x80000070   E59F1024 E59F2024 E3A03010 E4D14001 E0955004 E2533001 1AFFFFFB E1B05225
0x80000090   E2A55000 E4C25001 E320F000 800000A4 800000B4 3C32008C 167D5046 84553A2E
0x800000B0   02656E64 0000004A FF0000FF FB0000FF FF0000FF FF0000FF FB0000BF FE0000F7
```

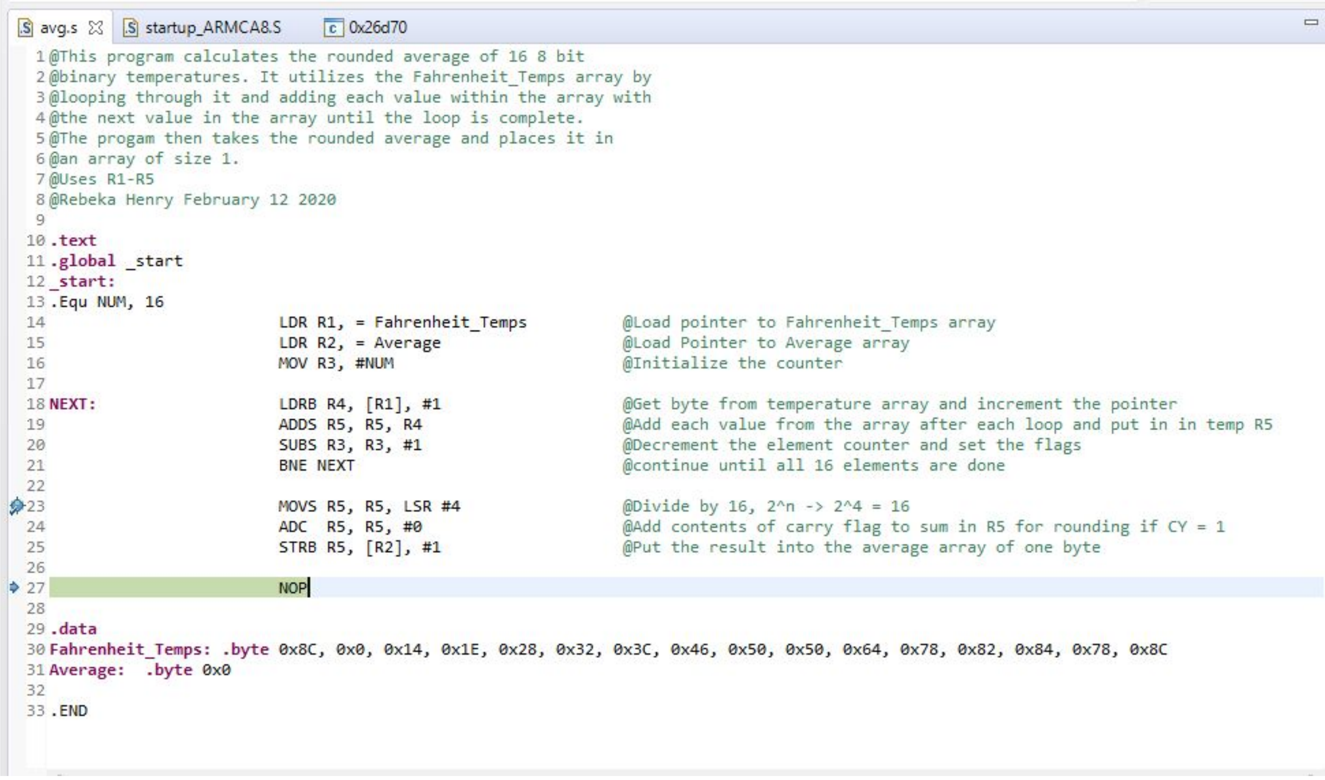
Value 0000004A = 74 in decimal

The algorithm worked for a case where there was a rounding of the temperature worked. Test 2 succeeded

Test 3 Code 3

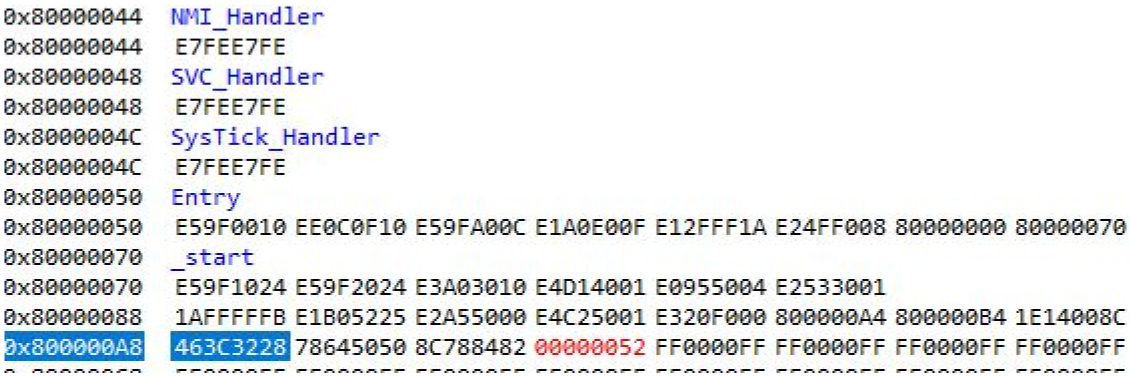
-Decimal values supplied: 0, 140, 20, 30, 40, 50, 60, 70, 80, 80, 100, 120, 130, 132, 120, 140

1312/16
Answer: 82



```
@This program calculates the rounded average of 16 8 bit
@binary temperatures. It utilizes the Fahrenheit_Temps array by
@looping through it and adding each value within the array with
@the next value in the array until the loop is complete.
@The progam then takes the rounded average and places it in
@an array of size 1.
@Uses R1-R5
@Rebeka Henry February 12 2020

.text
.global _start
_start:
.Equ NUM, 16
                        LDR R1, = Fahrenheit_Temps          @Load pointer to Fahrenheit_Temps array
                        LDR R2, = Average                   @Load Pointer to Average array
                        MOV R3, #NUM                        @Initialize the counter

NEXT:                   LDRB R4, [R1], #1                   @Get byte from temperature array and increment the pointer
                        ADDS R5, R5, R4                     @Add each value from the array after each loop and put in in temp R5
                        SUBS R3, R3, #1                     @Decrement the element counter and set the flags
                        BNE NEXT                            @continue until all 16 elements are done

                        MOVS R5, R5, LSR #4                 @Divide by 16, 2^n -> 2^4 = 16
                        ADC  R5, R5, #0                     @Add contents of carry flag to sum in R5 for rounding if CY = 1
                        STRB R5, [R2], #1                   @Put the result into the average array of one byte

                        NOP

.data
Fahrenheit_Temps: .byte 0x8C, 0x0, 0x14, 0x1E, 0x28, 0x32, 0x3C, 0x46, 0x50, 0x50, 0x64, 0x78, 0x82, 0x84, 0x78, 0x8C
Average:  .byte 0x0

.END
```

Test 3 Memory Browser 3



```
0x80000044  NMI_Handler
0x80000044  E7FEE7FE
0x80000048  SVC_Handler
0x80000048  E7FEE7FE
0x8000004C  SysTick_Handler
0x8000004C  E7FEE7FE
0x80000050  Entry
0x80000050  E59F0010 EE0C0F10 E59FA00C E1A0E00F E12FFF1A E24FF008 80000000 80000070
0x80000070  _start
0x80000070  E59F1024 E59F2024 E3A03010 E4D14001 E0955004 E2533001
0x80000088  1AFFFFFB E1B05225 E2A55000 E4C25001 E320F000 800000A4 800000B4 1E14008C
0x800000A8  463C3228 78645050 8C788482 00000052 FF0000FF FF0000FF FF0000FF FF0000FF
```

Value 00000052 = 82 in decimal

The algorithm worked for a case where the temperature was even and there were no remainders in the division. Test 3 succeeded

February 16, 2020

Part 2

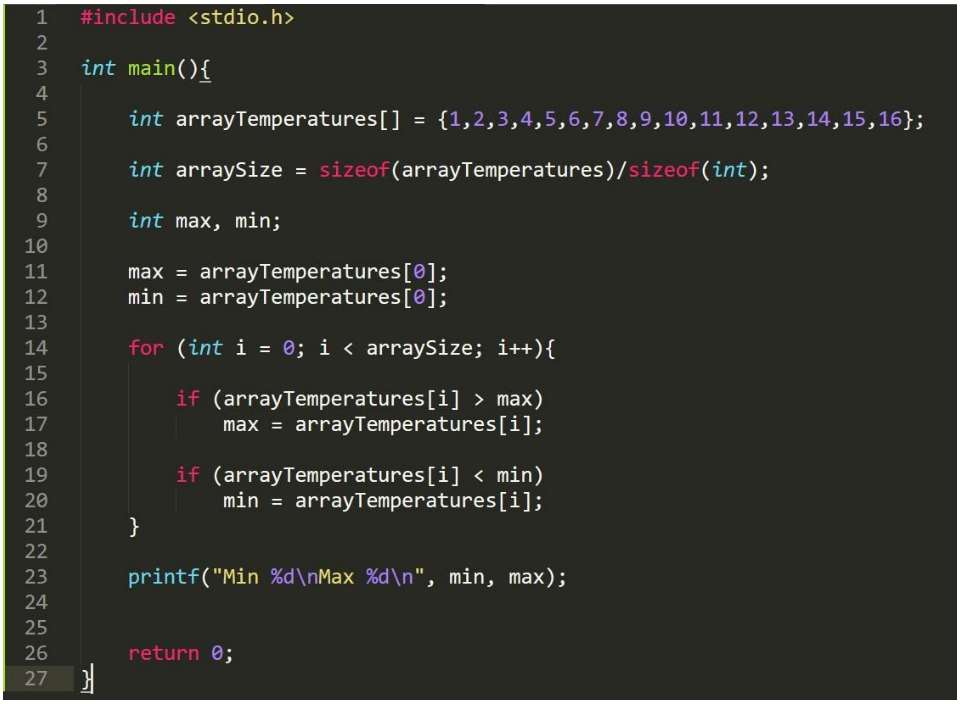
Task 2- write an algorithm for the procedure that will determine the maximum and minimum temperatures

Initial thoughts: have 2 temporary registers that will be used to store the maximum and minimum temperatures… I think that the procedure should either loop through the array twice or one

Algorithm for procedure: Draft 1

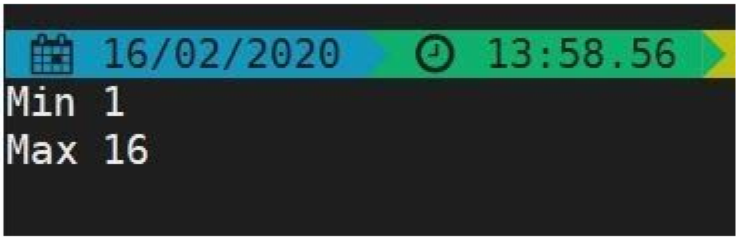
```
Temp_min = 0
Temp_max = 0

Loop through the array for min and max value
     Set temp_min = first index of Fahrenheit_Temps
     If Fahrenheit_Temps at index i is less than temp_min:
          Set temp_min to the value at that index
     If Fahrenheit_Temps at index i is greater than temp_max
          Set temp_max to the value at that index
Save the temp_min and temp_max values to named memory locations
```

Testing Algorithm in C for visualization Purposes



```c
#include <stdio.h>

int main(){

    int arrayTemperatures[] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};

    int arraySize = sizeof(arrayTemperatures)/sizeof(int);

    int max, min;

    max = arrayTemperatures[0];
    min = arrayTemperatures[0];

    for (int i = 0; i < arraySize; i++){

        if (arrayTemperatures[i] > max)
            max = arrayTemperatures[i];

        if (arrayTemperatures[i] < min)
            min = arrayTemperatures[i];
    }

    printf("Min %d\nMax %d\n", min, max);


    return 0;
}
```

Result of C Program



```
16/02/2020  13:58.56
Min 1
Max 16
```

Further Notes...

I can do something similar to the first program where I have 2 arrays of 1 byte but they are Temp_min and Temp_max

I will be calling a procedure that does the requested task… I should include a way to save the values by checking the condition. Page 110 shows signed less than and signed greater than. Could be useful… BNE also useful….

Modified Algorithm

Assuming that the stack has been initialized as well as the desired registers

```
Call procedure MINMAX
W/in the procedure
     REPEAT
          Load byte at element 0
          Set it to temp register for min and max
          Call the CMP instruction to check if less than
               If register value is less than,jump to an instruction that sets that value at element i to the min register temp value, overwriting the value that was originally there
     Keep doing this until there is no value that is smaller than what is in the temp register for min

     When done with this, store the temporary register value to TEMP_MIN with array of one element (this should happen outside of the loop)

          Call the CMP instruction to check if greater than
               If register value is greater than, jump to an instruction that sets the value at element i to the max register temp value, overwriting the value that was already there
     Keep doing this until there is no value that greater than what is in the temp register for max

     When done with this store the temporary register value to TEMP_MAX with array of one element (this should happen outside of the loop)

Restore the registers and then return (done using the stack)
```

The code

```
.text
.global _start
_start:
.Equ NUM, 16

               LDR R13, = STACK
               Add R13, R13, #0x100
               LDR R0, = Fahrenheit_Temps
               LDR R1, = Min
               LDR R2, = Max
               MOV R3, #NUM
               BL  MINMAX

MINMAX:        STMFD R13!, {R6-R8, R14}
               ADD R7, R7, #0x8C
               ADD R8, R8, #0x8C
NEXT:          LDRB R6, [R0], #1
               CMP R6, R7
               BLT  LESS_THAN

LESS_THAN:     MOV R7, R6

               CMP R6, R8
               BGT  GREATER_THAN

GREATER_THAN:  MOV R8, R6

               SUBS R3, R3, #1
               BNE NEXT

               STRB R1, [R7], #1
               STRB R2, [R8], #1

               LDMFD R13!, {R6-R8,PC}


.data
```

```
Fahrenheit_Temps: .byte 0x8C, 0x0, 0x14, 0x1E, 0x28, 0x32, 0x3C,
0x46, 0x50, 0x50, 0x64, 0x78, 0x82, 0x84, 0x78, 0x8C
Min: .byte 0x0
Max: .byte 0x0
STACK: .rept 256
        .byte 0x00
        .endr
.end

.END
```

Issues I see

Although it was built successfully, I did not properly save the correct values at R7 and R8. I want it to be the first index of the Fahrenheit_Temps array and in cases where we do not know what the value at that index is going to be. The fix that I can try is this:

Loop through the array but only once and store the desired values into the temporary registers before I begin the second loop that does the maxmin. NOTE: this is all happening within the MAXMIN procedure since it uses the registers for the stack

```
.Equ NUM2, 1

          MOV  R4,  #NUM2  @intialize  the  counter  for  temporary
     registers

NEXT1:         LDRB R6, [R0], #1
               MOV R7, R6
               MOV R8, R6
               SUBS R4, R4, #1
               BNE NEXT1
```

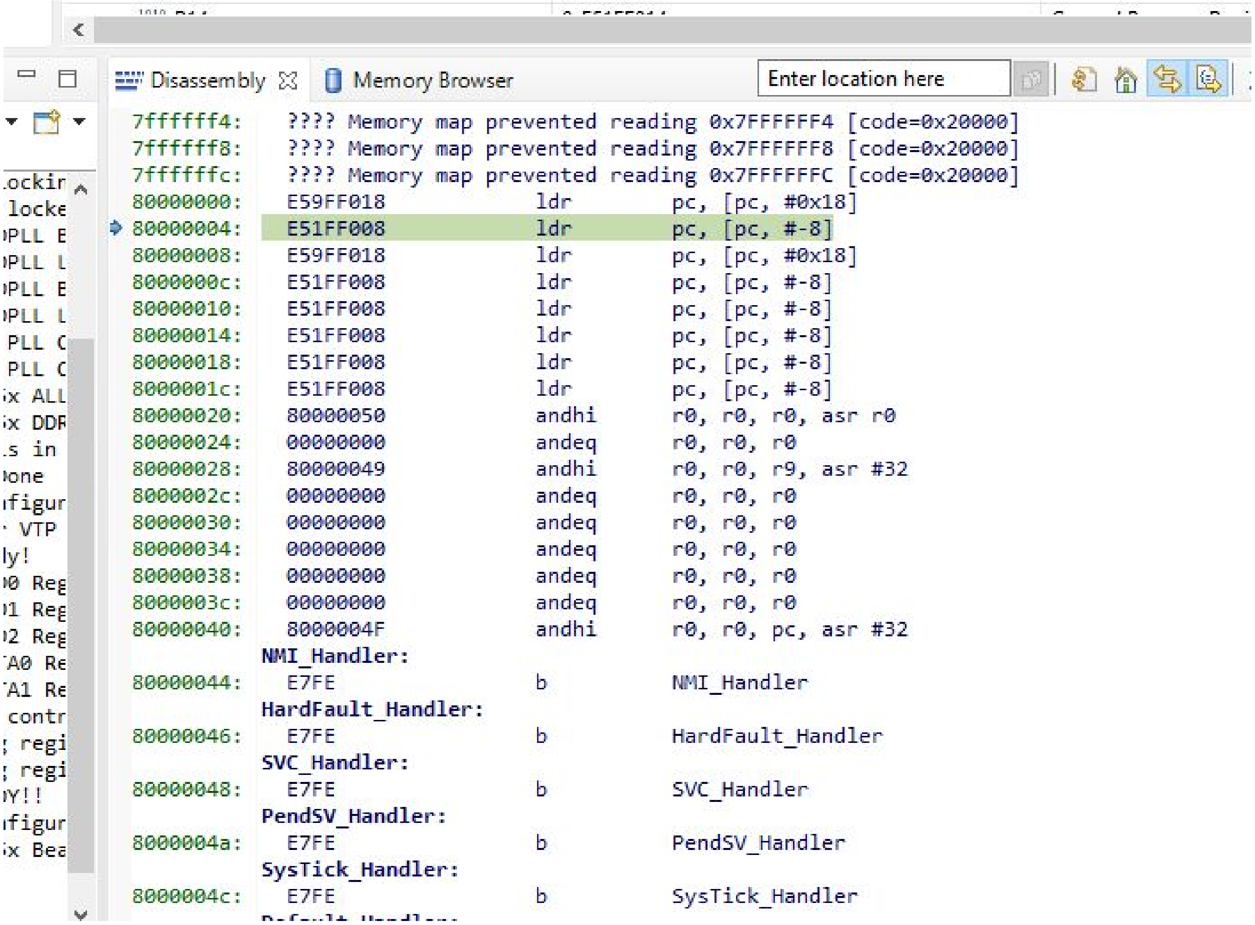
Modified Code

```
.text
.global _start
_start:
.Equ NUM, 16
.Equ NUM2, 1

               LDR R13, = STACK
               Add R13, R13, #0x100
               LDR R0, = Fahrenheit_Temps
               LDR R1, = Min
               LDR R2, = Max
               MOV R3, #NUM
               MOV R4, #NUM2
               BL  MINMAX

MINMAX:        STMFD R13!, {R6-R8, R14}

NEXT1:         LDRB R6, [R0], #1
               MOV R7, R6
               MOV R8, R6
               SUBS R4, R4, #1
               BNE NEXT1

NEXT:          LDRB R6, [R0], #1

               CMP R6, R7
               BLT  LESS_THAN

LESS_THAN:     MOV R7, R6

               CMP R6, R8
               BGT  GREATER_THAN

GREATER_THAN:       MOV R8, R6

                    SUBS R3, R3, #1
                    BNE NEXT
                    STRB R1, [R7], #1
                    STRB R2, [R8], #1

                    LDMFD R13!, {R6-R8,PC}



.data
Fahrenheit_Temps: .byte 0x8C, 0x0, 0x14, 0x1E, 0x28, 0x32, 0x3C,
0x46, 0x50, 0x50, 0x64, 0x78, 0x82, 0x84, 0x78, 0x8C
Min: .byte 0x0
Max: .byte 0x0
STACK: .rept 256
        .byte 0x00
        .endr

.END
```

February 17, 2020

New challenges:

Debugging the program, there was an issue with the program counter at address E51FF008



```
7ffffff4:   ???? Memory map prevented reading 0x7FFFFFF4 [code=0x20000]
7ffffff8:   ???? Memory map prevented reading 0x7FFFFFF8 [code=0x20000]
7ffffffc:   ???? Memory map prevented reading 0x7FFFFFFC [code=0x20000]
80000000:   E59FF018          ldr       pc, [pc, #0x18]
80000004:   E51FF008          ldr       pc, [pc, #-8]
80000008:   E59FF018          ldr       pc, [pc, #0x18]
8000000c:   E51FF008          ldr       pc, [pc, #-8]
80000010:   E51FF008          ldr       pc, [pc, #-8]
80000014:   E51FF008          ldr       pc, [pc, #-8]
80000018:   E51FF008          ldr       pc, [pc, #-8]
8000001c:   E51FF008          ldr       pc, [pc, #-8]
80000020:   80000050          andhi     r0, r0, r0, asr r0
80000024:   00000000          andeq     r0, r0, r0
80000028:   80000049          andhi     r0, r0, r9, asr #32
8000002c:   00000000          andeq     r0, r0, r0
80000030:   00000000          andeq     r0, r0, r0
80000034:   00000000          andeq     r0, r0, r0
80000038:   00000000          andeq     r0, r0, r0
8000003c:   00000000          andeq     r0, r0, r0
80000040:   8000004F          andhi     r0, r0, pc, asr #32
          NMI_Handler:
80000044:   E7FE              b         NMI_Handler
          HardFault_Handler:
80000046:   E7FE              b         HardFault_Handler
          SVC_Handler:
80000048:   E7FE              b         SVC_Handler
          PendSV_Handler:
8000004a:   E7FE              b         PendSV_Handler
          SysTick_Handler:
8000004c:   E7FE              b         SysTick_Handler
```

Stepping into the program, at NEXT1, there is an error because that is where I am no longer able to continue stepping. Or it could be before for when the stack is being set up

I am thinking that perhaps I did not allocate enough space on the stack so I will try to increase the size of the stack and see if it did anything

UPDATE 1: it did not change anything to increase the size of the stack. The same error with stepping has occurred.

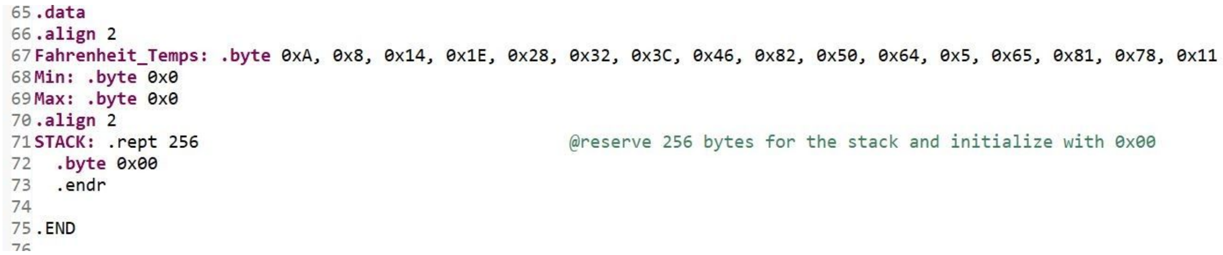
I am starting to think that maybe I should be using a different register for the different loops because it may be impacting the use in the other loop.

UPDATE 2: It is not the case. The message is actually an illegal instruction

```
LDR   pc, [pc,#-8]       @ 0x10 Data Abort
```

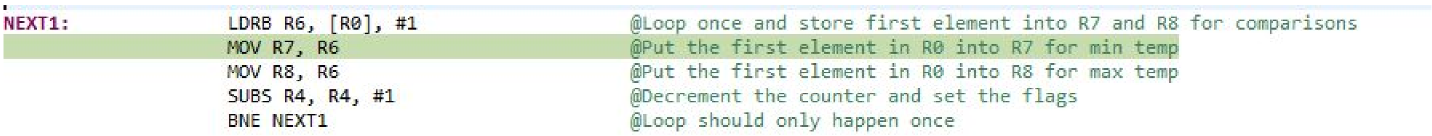
So I am likely loading or storing at an illegal address. The stack has been set up properly using page 181 but it appears that I may be doing something in the wrong way

I added .align 2 per the instructions provided by the TA and Professor. It worked! I can now step through the entire program. I fixed an illegal instruction as well.



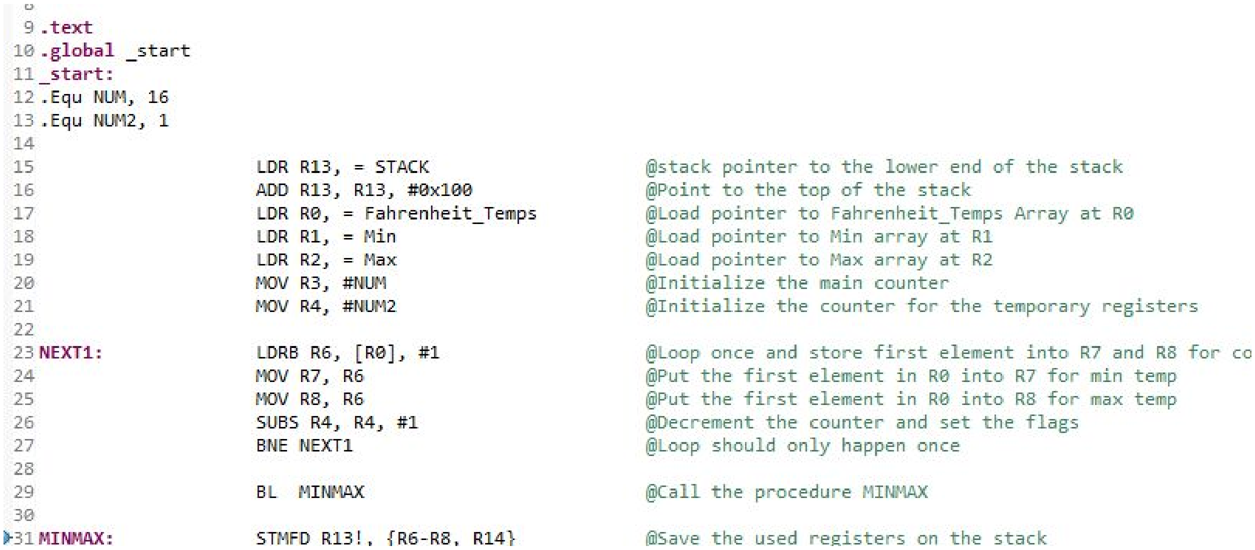
```
.data
.align 2
Fahrenheit_Temps: .byte 0xA, 0x8, 0x14, 0x1E, 0x28, 0x32, 0x3C, 0x46, 0x82, 0x50, 0x64, 0x5, 0x65, 0x81, 0x78, 0x11
Min: .byte 0x0
Max: .byte 0x0
.align 2
STACK: .rept 256                                @reserve 256 bytes for the stack and initialize with 0x00
  .byte 0x00
  .endr

.END
```

It appears that I have an infinite loop at NEXT1



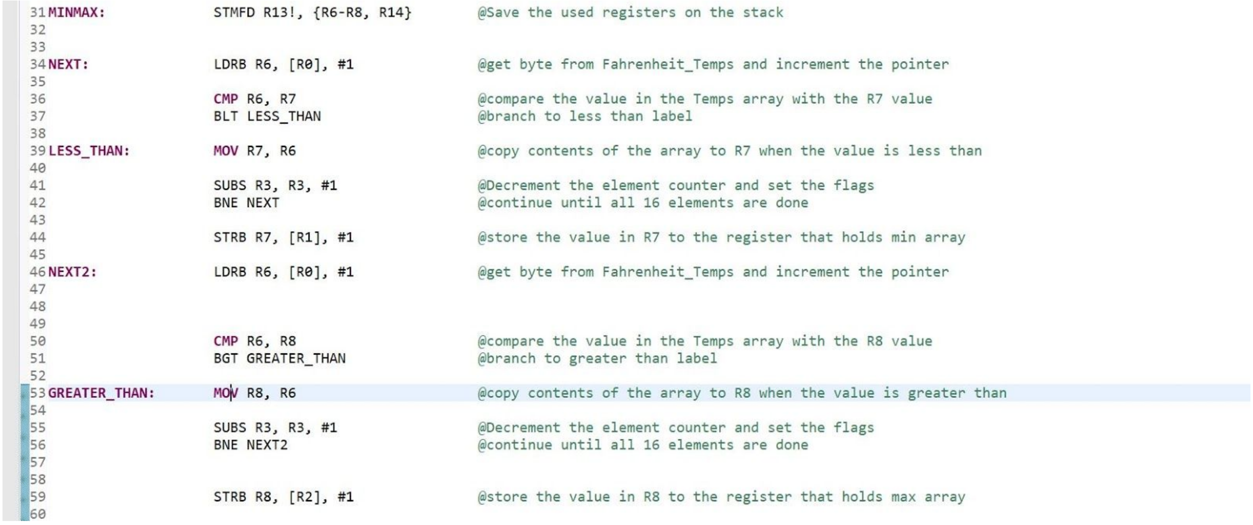
```
NEXT1:           LDRB R6, [R0], #1             @Loop once and store first element into R7 and R8 for comparisons
                 MOV R7, R6                    @Put the first element in R0 into R7 for min temp
                 MOV R8, R6                    @Put the first element in R0 into R8 for max temp
                 SUBS R4, R4, #1               @Decrement the counter and set the flags
                 BNE NEXT1                     @Loop should only happen once
```

I fixed it by having the loop only run once in the code and at the top before going to the MINMAX procedure



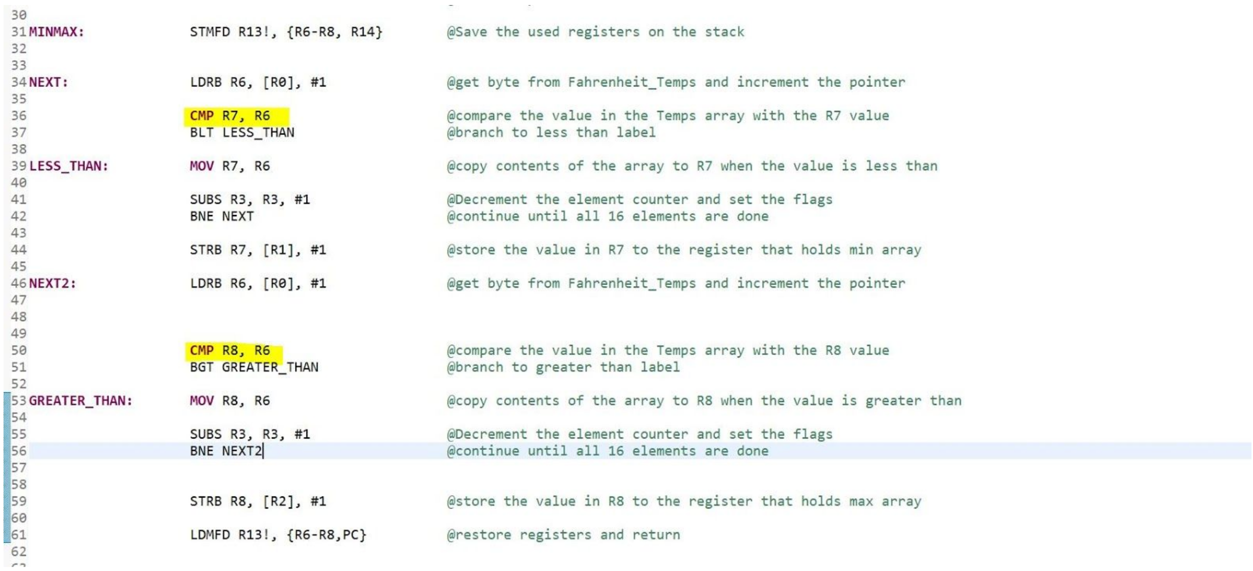
```
.text
.global _start
_start:
.Equ NUM, 16
.Equ NUM2, 1

                        LDR R13, = STACK                    @stack pointer to the lower end of the stack
                        ADD R13, R13, #0x100                @Point to the top of the stack
                        LDR R0, = Fahrenheit_Temps          @Load pointer to Fahrenheit_Temps Array at R0
                        LDR R1, = Min                       @Load pointer to Min array at R1
                        LDR R2, = Max                       @Load pointer to Max array at R2
                        MOV R3, #NUM                        @Initialize the main counter
                        MOV R4, #NUM2                       @Initialize the counter for the temporary registers

NEXT1:                  LDRB R6, [R0], #1                   @Loop once and store first element into R7 and R8 for co
                        MOV R7, R6                          @Put the first element in R0 into R7 for min temp
                        MOV R8, R6                          @Put the first element in R0 into R8 for max temp
                        SUBS R4, R4, #1                     @Decrement the counter and set the flags
                        BNE NEXT1                           @Loop should only happen once

                        BL  MINMAX                          @Call the procedure MINMAX

MINMAX:                 STMFD R13!, {R6-R8, R14}            @Save the used registers on the stack
```

Another change to make is to separately loop for the min and max. This is for debug purposes



```
MINMAX:             STMFD R13!, {R6-R8, R14}        @Save the used registers on the stack


NEXT:               LDRB R6, [R0], #1               @get byte from Fahrenheit_Temps and increment the pointer

                    CMP R6, R7                      @compare the value in the Temps array with the R7 value
                    BLT LESS_THAN                   @branch to less than label

LESS_THAN:          MOV R7, R6                      @copy contents of the array to R7 when the value is less than

                    SUBS R3, R3, #1                 @Decrement the element counter and set the flags
                    BNE NEXT                        @continue until all 16 elements are done

                    STRB R7, [R1], #1               @store the value in R7 to the register that holds min array

NEXT2:              LDRB R6, [R0], #1               @get byte from Fahrenheit_Temps and increment the pointer



                    CMP R6, R8                      @compare the value in the Temps array with the R8 value
                    BGT GREATER_THAN                @branch to greater than label

GREATER_THAN:       MOV R8, R6                      @copy contents of the array to R8 when the value is greater than

                    SUBS R3, R3, #1                 @Decrement the element counter and set the flags
                    BNE NEXT2                       @continue until all 16 elements are done


                    STRB R8, [R2], #1               @store the value in R8 to the register that holds max array

```

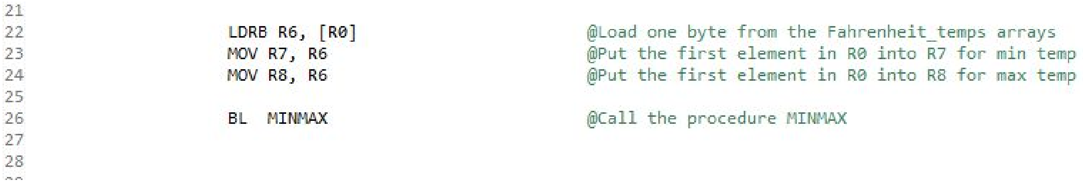
I may also change how the CMP instruction works so that it fits the code in C. I should be comparing the value in the register that holds the array with the value in the temporary register. That is the proper way to update. But testing it will tell me if it works or not



```

MINMAX:             STMFD R13!, {R6-R8, R14}        @Save the used registers on the stack


NEXT:               LDRB R6, [R0], #1               @get byte from Fahrenheit_Temps and increment the pointer

                    CMP R7, R6                      @compare the value in the Temps array with the R7 value
                    BLT LESS_THAN                   @branch to less than label

LESS_THAN:          MOV R7, R6                      @copy contents of the array to R7 when the value is less than

                    SUBS R3, R3, #1                 @Decrement the element counter and set the flags
                    BNE NEXT                        @continue until all 16 elements are done

                    STRB R7, [R1], #1               @store the value in R7 to the register that holds min array

NEXT2:              LDRB R6, [R0], #1               @get byte from Fahrenheit_Temps and increment the pointer



                    CMP R8, R6                      @compare the value in the Temps array with the R8 value
                    BGT GREATER_THAN                @branch to greater than label

GREATER_THAN:       MOV R8, R6                      @copy contents of the array to R8 when the value is greater than

                    SUBS R3, R3, #1                 @Decrement the element counter and set the flags
                    BNE NEXT2                       @continue until all 16 elements are done


                    STRB R8, [R2], #1               @store the value in R8 to the register that holds max array

                    LDMFD R13!, {R6-R8,PC}          @restore registers and return

```

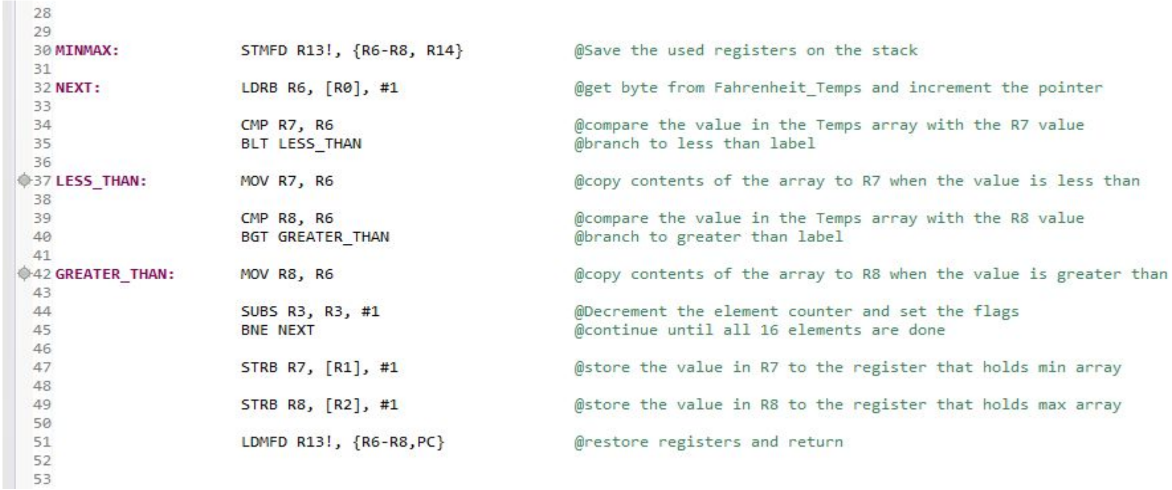
February 18, 2020

There were a few issues discussed today with Tyler. The first issue has to do with getting the first index from the array and putting them in the temp registers R6 and R7. The suggested solution is to only load one byte and to not increment to the next value. Essentially, remove the loop and just load one byte. The code from before may have been getting in garbage data over and over again



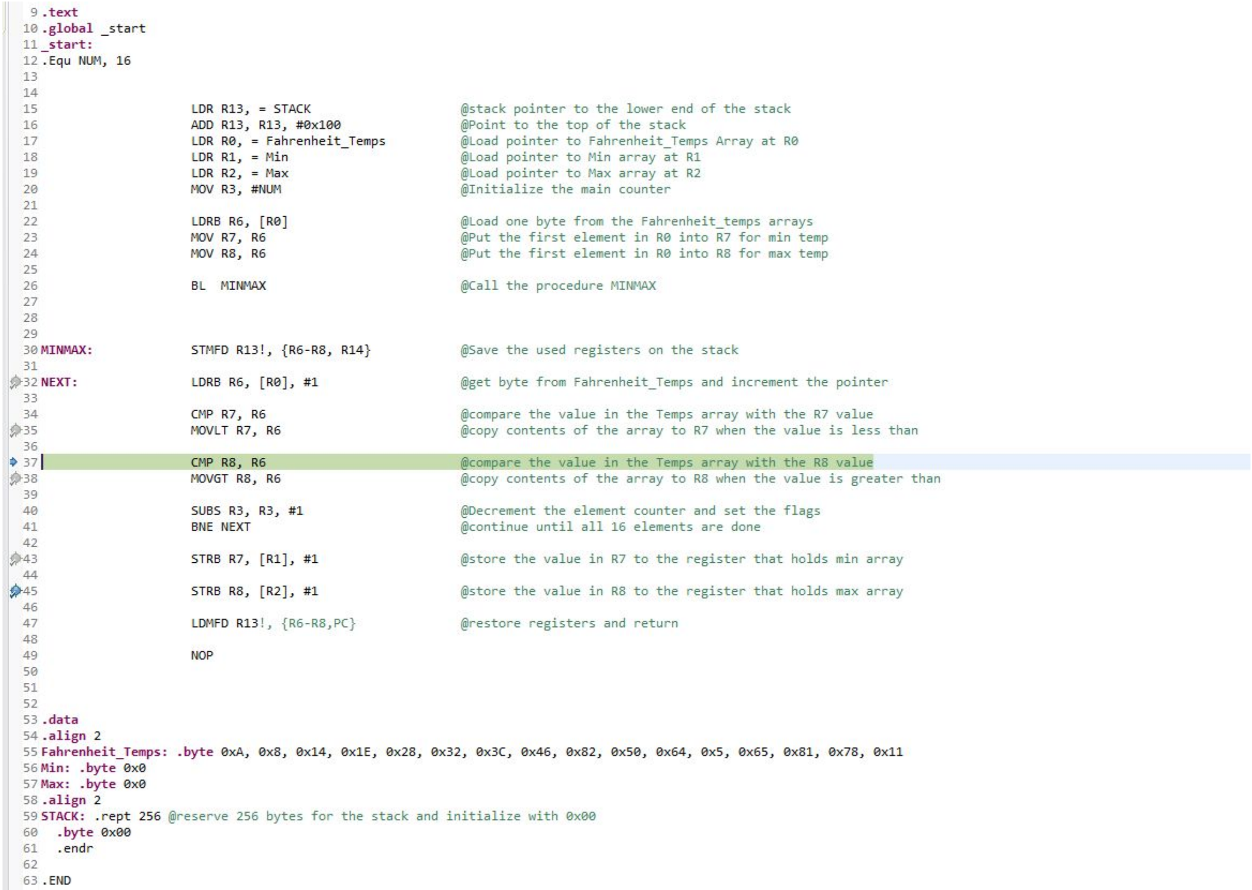
```
                LDRB R6, [R0]                   @Load one byte from the Fahrenheit_temps arrays
                MOV R7, R6                      @Put the first element in R0 into R7 for min temp
                MOV R8, R6                      @Put the first element in R0 into R8 for max temp

                BL  MINMAX                      @Call the procedure MINMAX
```

The second fix was to ensure that I use only one loop for the max and min operations. This way, I don’t have to `LDR R0, = Fahrenheit_Temps` again. This would have been a necessary fix with the original code that I had.



```
MINMAX:         STMFD R13!, {R6-R8, R14}        @Save the used registers on the stack

NEXT:           LDRB R6, [R0], #1               @get byte from Fahrenheit_Temps and increment the pointer

                CMP R7, R6                      @compare the value in the Temps array with the R7 value
                BLT LESS_THAN                   @branch to less than label

LESS_THAN:      MOV R7, R6                      @copy contents of the array to R7 when the value is less than

                CMP R8, R6                      @compare the value in the Temps array with the R8 value
                BGT GREATER_THAN                @branch to greater than label

GREATER_THAN:   MOV R8, R6                      @copy contents of the array to R8 when the value is greater than

                SUBS R3, R3, #1                 @Decrement the element counter and set the flags
                BNE NEXT                        @continue until all 16 elements are done

                STRB R7, [R1], #1               @store the value in R7 to the register that holds min array

                STRB R8, [R2], #1               @store the value in R8 to the register that holds max array

                LDMFD R13!, {R6-R8,PC}          @restore registers and return
```

The final fix is instead of using the labels `LESS_THAN` and `GREATER_THAN` I am going to use `MOVLT` and `MOVGT` and in this way, the program doesn't run through and incorrectly saves values just because the loop increments.



```
.text
.global _start
_start:
.Equ NUM, 16


                    LDR R13, = STACK                    @stack pointer to the lower end of the stack
                    ADD R13, R13, #0x100                @Point to the top of the stack
                    LDR R0, = Fahrenheit_Temps          @Load pointer to Fahrenheit_Temps Array at R0
                    LDR R1, = Min                       @Load pointer to Min array at R1
                    LDR R2, = Max                       @Load pointer to Max array at R2
                    MOV R3, #NUM                        @Initialize the main counter

                    LDRB R6, [R0]                       @Load one byte from the Fahrenheit_temps arrays
                    MOV R7, R6                          @Put the first element in R0 into R7 for min temp
                    MOV R8, R6                          @Put the first element in R0 into R8 for max temp

                    BL  MINMAX                          @Call the procedure MINMAX



MINMAX:             STMFD R13!, {R6-R8, R14}            @Save the used registers on the stack

NEXT:               LDRB R6, [R0], #1                   @get byte from Fahrenheit_Temps and increment the pointer

                    CMP R7, R6                          @compare the value in the Temps array with the R7 value
                    MOVLT R7, R6                        @copy contents of the array to R7 when the value is less than

                    CMP R8, R6                          @compare the value in the Temps array with the R8 value
                    MOVGT R8, R6                        @copy contents of the array to R8 when the value is greater than

                    SUBS R3, R3, #1                     @Decrement the element counter and set the flags
                    BNE NEXT                            @continue until all 16 elements are done

                    STRB R7, [R1], #1                   @store the value in R7 to the register that holds min array

                    STRB R8, [R2], #1                   @store the value in R8 to the register that holds max array

                    LDMFD R13!, {R6-R8,PC}              @restore registers and return

                    NOP



.data
.align 2
Fahrenheit_Temps: .byte 0xA, 0x8, 0x14, 0x1E, 0x28, 0x32, 0x3C, 0x46, 0x82, 0x50, 0x64, 0x5, 0x65, 0x81, 0x78, 0x11
Min: .byte 0x0
Max: .byte 0x0
.align 2
STACK: .rept 256 @reserve 256 bytes for the stack and initialize with 0x00
  .byte 0x00
  .endr

.END
```

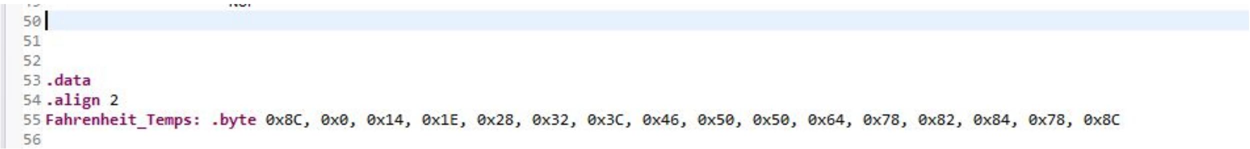
Test Cases Overview

Test case 1 will leave 140 and 0 in their place

Decimal- 140, 0, 20, 30, 40, 50, 60, 70, 80, 80, 100, 120, 130, 132, 120, 140

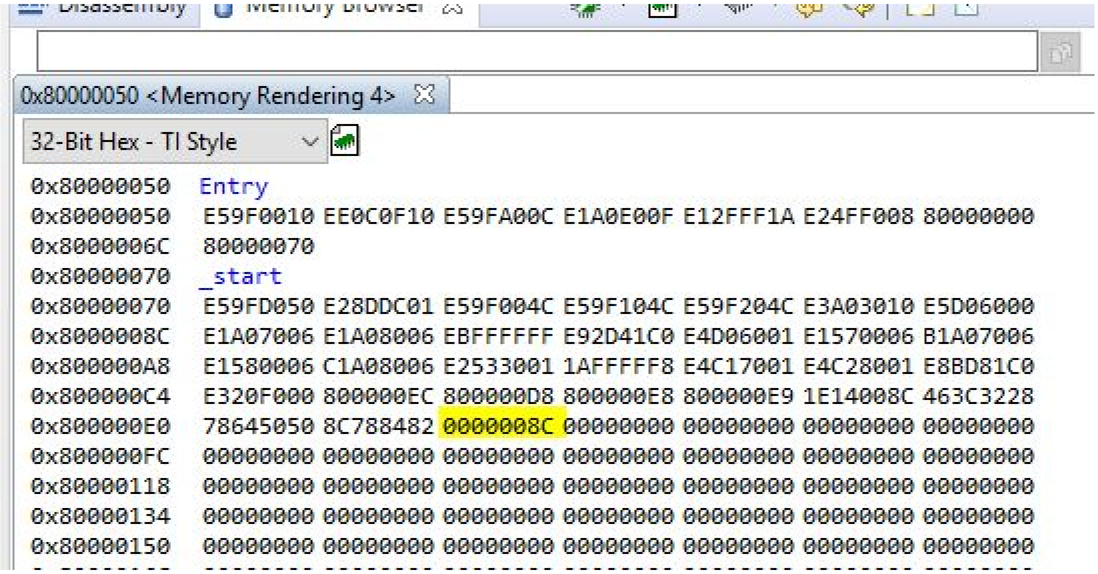
Hex- 8C, 0, 14, 1E, 28, 32, 3C, 46, 50, 50, 64, 78, 82, 84, 78, 8C

Min: 0 → 0x0
Max: 140 → 0x8C



```
.data
.align 2
Fahrenheit_Temps: .byte 0x8C, 0x0, 0x14, 0x1E, 0x28, 0x32, 0x3C, 0x46, 0x50, 0x50, 0x64, 0x78, 0x82, 0x84, 0x78, 0x8C
```

Since it is little Endian, Min is first and then Max



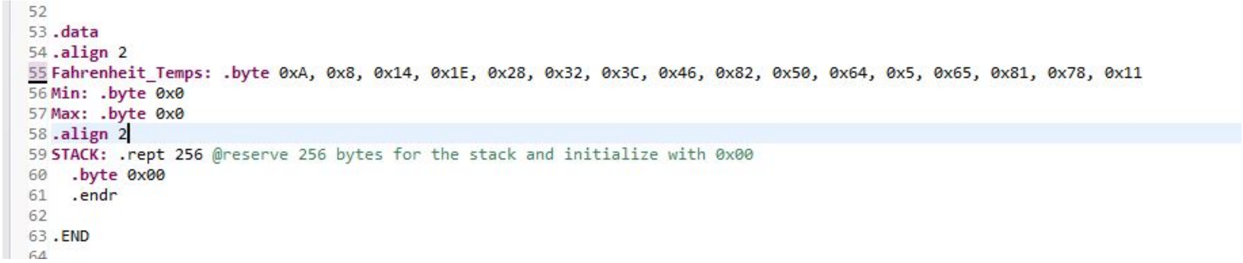
```
0x80000050   Entry
0x80000050   E59F0010 EE0C0F10 E59FA00C E1A0E00F E12FFF1A E24FF008 80000000
0x8000006C   80000070
0x80000070   _start
0x80000070   E59FD050 E28DDC01 E59F004C E59F104C E59F204C E3A03010 E5D06000
0x8000008C   E1A07006 E1A08006 EBFFFFFF E92D41C0 E4D06001 E1570006 B1A07006
0x800000A8   E1580006 C1A08006 E2533001 1AFFFFF8 E4C17001 E4C28001 E8BD81C0
0x800000C4   E320F000 800000EC 800000D8 800000E8 800000E9 1E14008C 463C3228
0x800000E0   78645050 8C788482 0000008C 00000000 00000000 00000000 00000000
0x800000FC   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000118   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000134   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000150   00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Test case 2 will make the max 130 and min 5 and it places them within the array to be located later in program

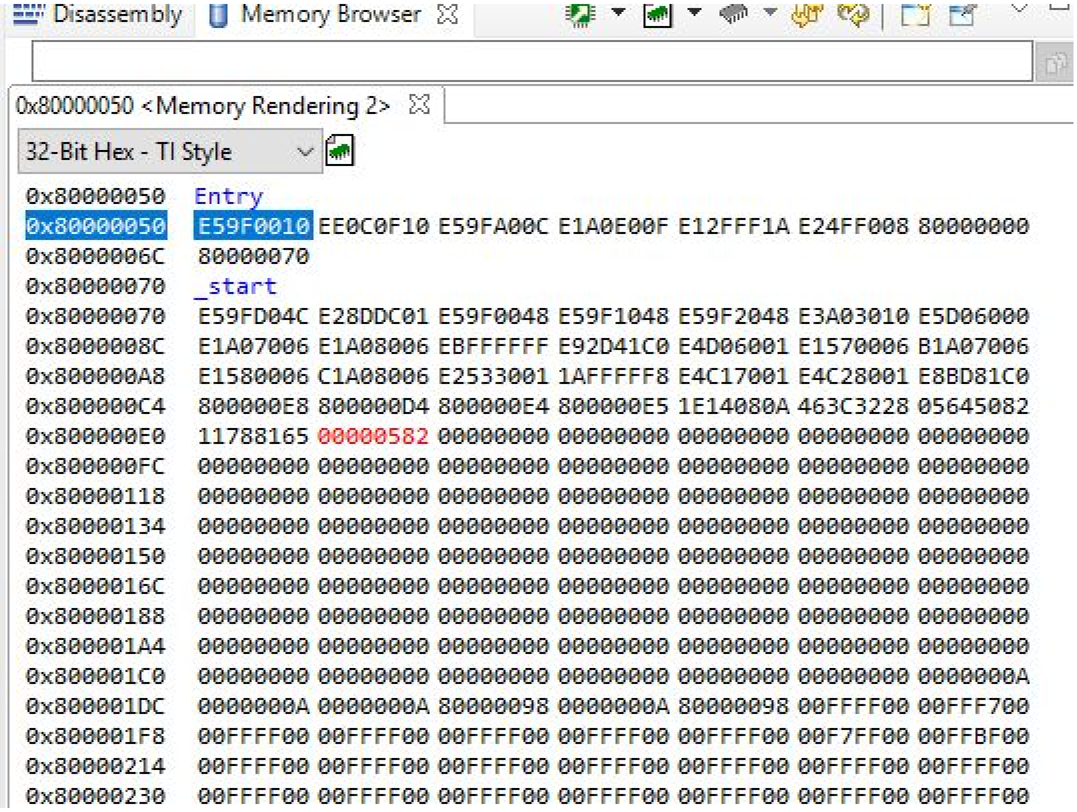
Decimal- 10, 8, 20, 30, 40, 50, 60, 70, 130, 80, 100, 5, 101, 129, 120, 17

Hex- A, 8, 14, 1E, 28, 32, 3C, 46, 82, 50, 64, 5, 65, 81, 78, 11

Min: 5 → 0x5
Max: 130 → 0x82



```

.data
.align 2
Fahrenheit_Temps: .byte 0xA, 0x8, 0x14, 0x1E, 0x28, 0x32, 0x3C, 0x46, 0x82, 0x50, 0x64, 0x5, 0x65, 0x81, 0x78, 0x11
Min: .byte 0x0
Max: .byte 0x0
.align 2
STACK: .rept 256 @reserve 256 bytes for the stack and initialize with 0x00
  .byte 0x00
  .endr

.END
```



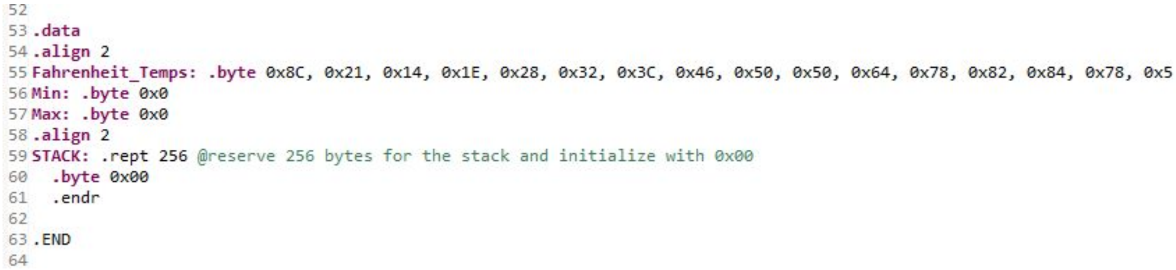
```
0x80000050  Entry
0x80000050  E59F0010 EE0C0F10 E59FA00C E1A0E00F E12FFF1A E24FF008 80000000
0x8000006C  80000070
0x80000070  _start
0x80000070  E59FD04C E28DDC01 E59F0048 E59F1048 E59F2048 E3A03010 E5D06000
0x8000008C  E1A07006 E1A08006 EBFFFFFF E92D41C0 E4D06001 E1570006 B1A07006
0x800000A8  E1580006 C1A08006 E2533001 1AFFFFF8 E4C17001 E4C28001 E8BD81C0
0x800000C4  800000E8 800000D4 800000E4 800000E5 1E14080A 463C3228 05645082
0x800000E0  11788165 00000582 00000000 00000000 00000000 00000000 00000000
0x800000FC  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000118  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000134  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000150  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x8000016C  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000188  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001A4  00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001C0  00000000 00000000 00000000 00000000 00000000 00000000 0000000A
0x800001DC  0000000A 0000000A 80000098 0000000A 80000098 00FFFF00 00FFF700
0x800001F8  00FFFF00 00FFFF00 00FFFF00 00FFFF00 00FFFF00 00F7FF00 00FFBF00
0x80000214  00FFFF00 00FFFF00 00FFFF00 00FFFF00 00FFFF00 00FFFF00 00FFFF00
0x80000230  00FFFF00 00FFFF00 00FFFF00 00FFFF00 00FFFF00 00FFFF00 00FFFF00
```

Test Case 3 will place the max at the beginning and the min at the end

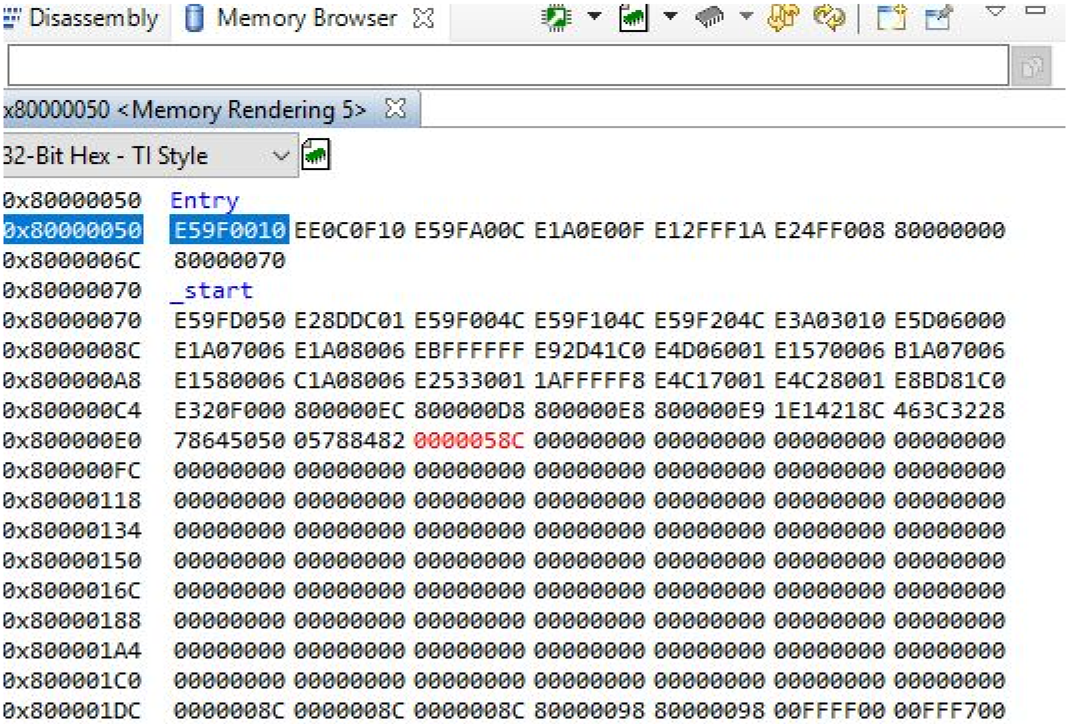
Decimal- 140, 33, 20, 30, 40, 50, 60, 70, 80, 80, 100, 120, 130, 132, 120, 5

Hex- 8C, 21, 14, 1E, 28, 32, 3C, 46, 50, 50, 64, 78, 82, 84, 78, 5

Min: 5 → 0x5
Max: 140 → 0x8C



```

.data
.align 2
Fahrenheit_Temps: .byte 0x8C, 0x21, 0x14, 0x1E, 0x28, 0x32, 0x3C, 0x46, 0x50, 0x50, 0x64, 0x78, 0x82, 0x84, 0x78, 0x5
Min: .byte 0x0
Max: .byte 0x0
.align 2
STACK: .rept 256 @reserve 256 bytes for the stack and initialize with 0x00
  .byte 0x00
  .endr

.END

```



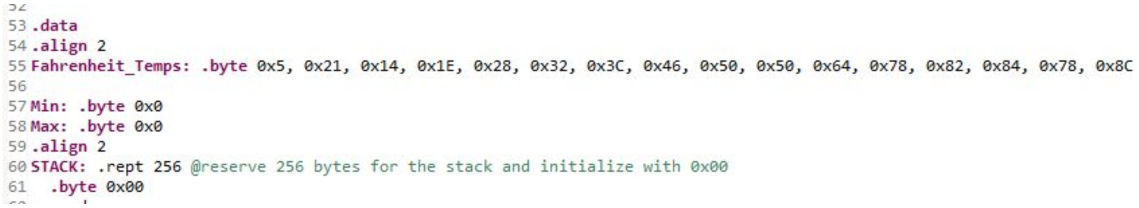
```
0x80000050   Entry
0x80000050   E59F0010 EE0C0F10 E59FA00C E1A0E00F E12FFF1A E24FF008 80000000
0x8000006C   80000070
0x80000070   _start
0x80000070   E59FD050 E28DDC01 E59F004C E59F104C E59F204C E3A03010 E5D06000
0x8000008C   E1A07006 E1A08006 EBFFFFFF E92D41C0 E4D06001 E1570006 B1A07006
0x800000A8   E1580006 C1A08006 E2533001 1AFFFFF8 E4C17001 E4C28001 E8BD81C0
0x800000C4   E320F000 800000EC 800000D8 800000E8 800000E9 1E14218C 463C3228
0x800000E0   78645050 05788482 0000058C 00000000 00000000 00000000 00000000
0x800000FC   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000118   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000134   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000150   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x8000016C   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000188   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001A4   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001C0   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001DC   0000008C 0000008C 0000008C 80000098 80000098 00FFFF00 00FFF700
```

Test Case 4 will place the min at the beginning and the max at the end

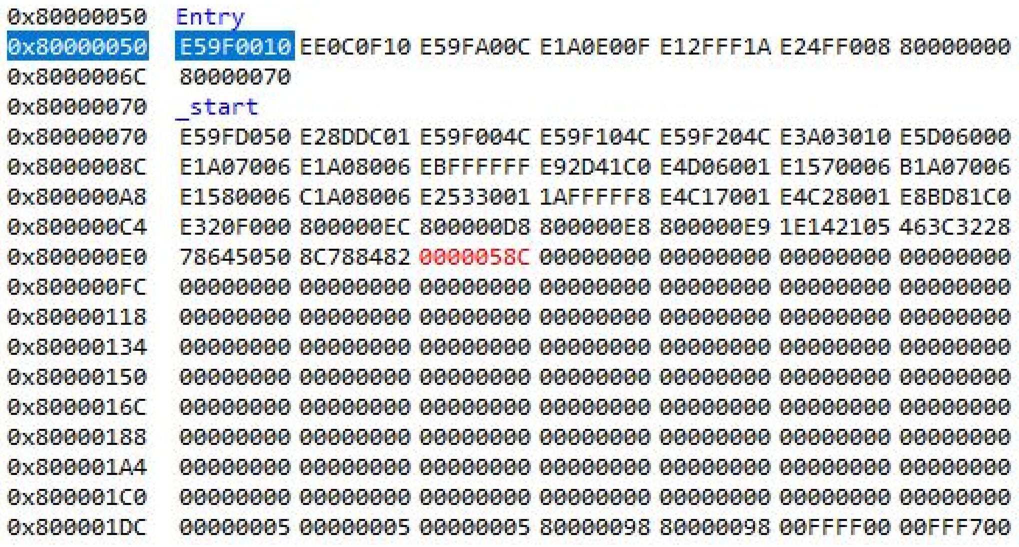
Decimal- 5, 33, 20, 30, 40, 50, 60, 70, 80, 80, 100, 120, 130, 132, 120, 140

Hex- 5, 21, 14, 1E, 28, 32, 3C, 46, 50, 50, 64, 78, 82, 84, 78, 8C

Min: 5 → 0x5
Max: 140 → 0x8C



```
.data
.align 2
Fahrenheit_Temps: .byte 0x5, 0x21, 0x14, 0x1E, 0x28, 0x32, 0x3C, 0x46, 0x50, 0x50, 0x64, 0x78, 0x82, 0x84, 0x78, 0x8C

Min: .byte 0x0
Max: .byte 0x0
.align 2
STACK: .rept 256 @reserve 256 bytes for the stack and initialize with 0x00
  .byte 0x00
```



```
0x80000050   Entry
0x80000050   E59F0010 EE0C0F10 E59FA00C E1A0E00F E12FFF1A E24FF008 80000000
0x8000006C   80000070
0x80000070   _start
0x80000070   E59FD050 E28DDC01 E59F004C E59F104C E59F204C E3A03010 E5D06000
0x8000008C   E1A07006 E1A08006 EBFFFFFF E92D41C0 E4D06001 E1570006 B1A07006
0x800000A8   E1580006 C1A08006 E2533001 1AFFFFF8 E4C17001 E4C28001 E8BD81C0
0x800000C4   E320F000 800000EC 800000D8 800000E8 800000E9 1E142105 463C3228
0x800000E0   78645050 8C788482 0000058C 00000000 00000000 00000000 00000000
0x800000FC   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000118   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000134   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000150   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x8000016C   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000188   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001A4   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001C0   00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001DC   00000005 00000005 00000005 80000098 80000098 00FFFF00 00FFF700
```