Philip Nevins
ECE 371 Microprocessor
Design Project 1

Design Log

Entry Date: 11/11/2022

For the last week, I have been thinking about good approaches to this project. Today, on the 11th, I had an idea come to mind on how to make this happen efficiently for Part 1.

Idea/Algorithm: Use a loop to add all the values stored in memory. We can use the same method from multo in our book. Then we learned this week, that there is no division command for the BBB, so we will need to shift the bits in the register that needs dividing to the right. Every time you shift right, you divide by 2. To divide by 16, which is given to us from the project directions, you will need to shift right 4 times. (2^4 = 16). Then we save this final value in memory.

This will be the mainline for the scope of the entire project

Entry Date: 11/12/2022

I have been debating ideas for Part 2. This is my first iteration which has ended up being my final algorithm.

Idea/algorithm:
We will branch to this function from the start of the program
Push to stack, registers and return addr (required by design specifications)
We will need to convert the temperature from F to C. We know that
C = (F – 32) * 5/9
Since we do not have a division command, we will need to use the recommended method from the class manual, where you subtract the bottom number from the top number and count how many times you had to do this until you cannot do it anymore.
So the idea would be to subtract 32 from F, then multiple by 5. Now we use a loop to subtract 9 from this value until we cannot subtract 9 anymore. Each time through this, we will need to compare the current value to 5, and if its greater, we add the carry to the current value using adc.
To count the times we did this, inside the loop, after the subtraction, we will increment a counter.
Pop stack and restore registers (required by design specifications)
Then branch back, and continue with Part 1s algorithm. This is stated as a requirement in the design specifications, to branch to a function and then return to the mainline.

Entry Date: 11/13/2022

Part 1 went as planned, worked within a 1 decimal value error for multiple test value sets. Need to figure out why there is an error in the final value.

Part 2 was giving me issues. I cannot figure out how to get the values out of the stack. I was able to allocate a stack and get the converted values written to it, but then I could not get it passed back to the mainline, I keep getting a prefetch error. I will make another attempt at this tomorrow (11/14/2022). I also ran into issues with the total temp and average displaying inside the Celsius memory array. I will come back to this while I am working on the stack.

Entry Date 11/14/2022

I have had a continued issue with getting values out of the stack. I have been able to declare a stack, do the F->C conversions and write the values into the allocated stack location. But I continue to get an error trying to get out of the stack. I keep getting a prefetch error, so I removed the stack all together to get the "meat" of the function working and then come back to the stack

I was able to figure out why the average and total values were displaying inside the Celsius array (after removing the stack). The pointer for the average and total temps was set to a value inside the Celsius array. I could not figure out why this was happening, so I adjusted the pointers to the correct addresses before entering the loops to add and average, so they would display after the Celsius array. I found this by setting break points so I could step through the add and average function and was able to calculate the required pointer adjustment to save the total and average after the Celsius array

I will come back to this 11/15/2022 in the evening before this project is due. I was able to get the project fully functional without the stack, by just allocating memory arrays for the F and C temps and it produces the correct conversions and averages though, and I was able to get them displayed in a nice, orderly fashion.

Entry Date 11/15/2022

We have been given an extension until Friday, 11/18 to finish the project. I have gotten advice from the TA which should enable the stack to work properly. He suggested to make sure I am using the .align 2. I was not using this in my first stack iteration.

After reviewing the books example on the stack from pg 187, I believe I know my issue. I was not calculating the stack address / stack map correctly, so when it was returning to the mainline, it would throw a load value error. I have formulated a plan of attack, that will be implemented on 11/16. I will re-add my stack map that should be calculated properly and adjust the code on my .s file with the stack in it.

I believe the issue is in the LDMFD command, when you pop the pushed values off the stack and back into the registers when leaving the "work" functions, the R13 ## offset change is what is wrong from my understanding. I will be working on this on 11/16 and will be requesting the help of the TA once his office hours start, if I can make progress farther than my last attempt with the stack.

Entry Date 11/16/2022

Today, I got some help from the TA and figured out my issue with the stack. I didn't need to adjust the R13 pointer at all. I was going off the example in the book and I was misconstruing things. The example in the book, the registers that got restored had data that was needed inside the function, which is different than what we are doing here. I also realized that I was using R13 as my register for storing the average temperature. So I was attempting to offset this, when I should of just not used R13 to store the average temp! Oops! Once I removed these commands, changed the average temp location, and just used the STMFD / LDMFD commands, it worked!

Entry Date 11/17/2022

Today, I cleaned up the comments inside each program, double checked everything and made some final optimizations. The final run of the program will happen on Friday, where I will get the required screenshots of the memory browser and registers to make sure I can prove this program works as intended.

Entry Date 11/18/2022

Today, I ran into a completely new issue. Every time I went to load the microprocessor with my program, it would crash. I had to make a whole new project, .s file and copy over my part1 and part2 code. Doing this fixed the issue.

I also addressed the rounding error during the final run of the program. Everything was off by +1. After doing some calculations by hand, I noticed that you need to do the ADC after the loop is finished wherever you are rounding. If you have the ADC inside the loop, you will round up extra times, even if you only "right shift" a 1 out one time, it will add an extra, so moving the ADC outside of the loop fixed this issue on both Part 1 and Part 2. Also, since we did this, to get the correct value stored, we had to add another store command after the ADC outside of the loop. Normally when doing division by hand, you don't round until the division is complete, so this makes sense.

This makes sense due to my hand calculations. A good example to verify this is how you do the rounding, is to take the decimal number 15 into account. In binary, it is 1111. To divide by 16, you need to shift the bits right 4 times. This should yield 0001 (after rounding). Shifting right and adding the carry EACH TIME (like what was previously done), would yield a decimal value of 4, because with 1111, each time you shift right, you set the carry flag, and adding the carry EACH TIME would yield 4. This is incorrect. So, logically, you would shift right 4 times, then if the carry is set after all shifts are complete, you add the carry. We can also see this is correct with the number 16 (10000 in binary). If you shift 10000 to the right 4 times to divide by 16, you will get 00001, you don't have to add carry because the carry flag is not set during any of the shifts since only 0s are being shifted out, which is what we expect. This shows that you need to put the ADC outside of the loop, so if you do have a carry, you add the carry after all of the shifting has been done.

Pseudocode [11/12/2022 after log entry]

Part 1

```
Set counter values
Load pointers to memory locations of temp arrays
Load counter values

Repeat
    Add temperatures one by one
    Save in memory array
Until all temps are added

Repeat
    Shift bits right once (since there is no division function, this is divide by 2)
    If carry is set, add carry (rounding)
    Store average in memory array

Until divide by 16 (4 times) is complete
```

Part 2

```
Set counter values and rounding factor
Load pointers to memory locations of temp arrays, stack
Load counter values

Branch to Function to calculate Celsius temps
```

(start code from part 1)

```
Repeat
    Add temperatures one by one
    Save in memory array
Until all temps are added

Repeat
    Shift bits right once (since there is no division function, this is divide by 2)
    If carry is set, add carry (rounding)
    Store average in memory array

Until divide by 16 (4 times) is complete
```

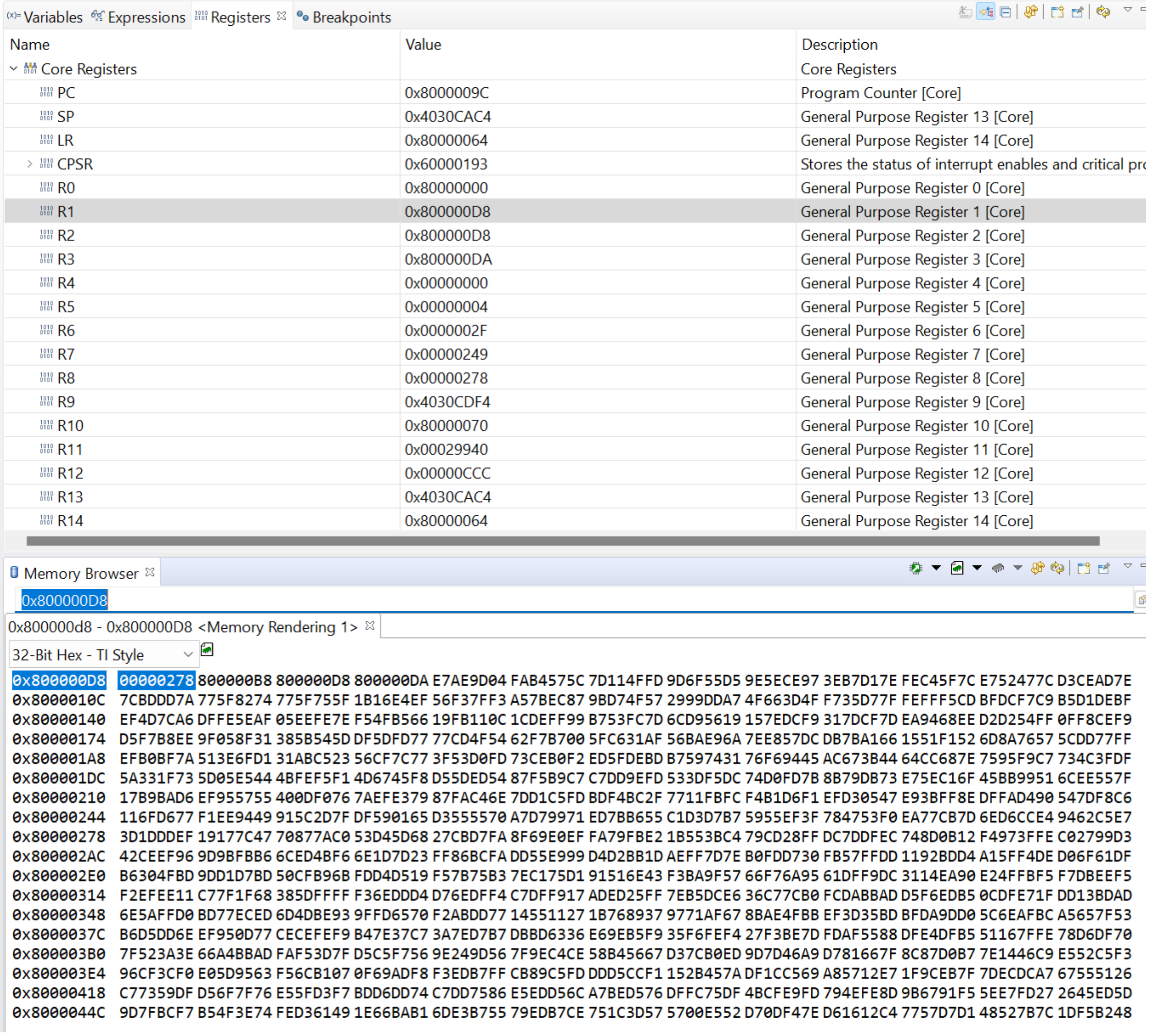
(end code from part 1)

```
FUNCTION calculate Celsius temps:
Push registers used in function + return addr to the stack
```

```
Repeat1
    Subtract 32 from F_temp value
    Multiply by 5
    Repeat2
        Subtract 9 from total
        Increment division counter
        Compare new total to 5
        Add carry if set to total (rounding)
    Until negative flag is set
        Repeat3
            Store value in memory array
            Decrement counter
            Branch to Repeat1 Until zero flag set

Pop the stack to restore all register values
Go back to mainline (part 1 code starts here)
```

Part 1 Screen Shots (Code is in Appendix 1 at the end)



Name	Value	Description
Core Registers		Core Registers
PC	0x8000009C	Program Counter [Core]
SP	0x4030CAC4	General Purpose Register 13 [Core]
LR	0x80000064	General Purpose Register 14 [Core]
CPSR	0x60000193	Stores the status of interrupt enables and critical pr
R0	0x80000000	General Purpose Register 0 [Core]
R1	0x800000D8	General Purpose Register 1 [Core]
R2	0x800000D8	General Purpose Register 2 [Core]
R3	0x800000DA	General Purpose Register 3 [Core]
R4	0x00000000	General Purpose Register 4 [Core]
R5	0x00000004	General Purpose Register 5 [Core]
R6	0x0000002F	General Purpose Register 6 [Core]
R7	0x00000249	General Purpose Register 7 [Core]
R8	0x00000278	General Purpose Register 8 [Core]
R9	0x4030CDF4	General Purpose Register 9 [Core]
R10	0x80000070	General Purpose Register 10 [Core]
R11	0x00029940	General Purpose Register 11 [Core]
R12	0x00000CCC	General Purpose Register 12 [Core]
R13	0x4030CAC4	General Purpose Register 13 [Core]
R14	0x80000064	General Purpose Register 14 [Core]

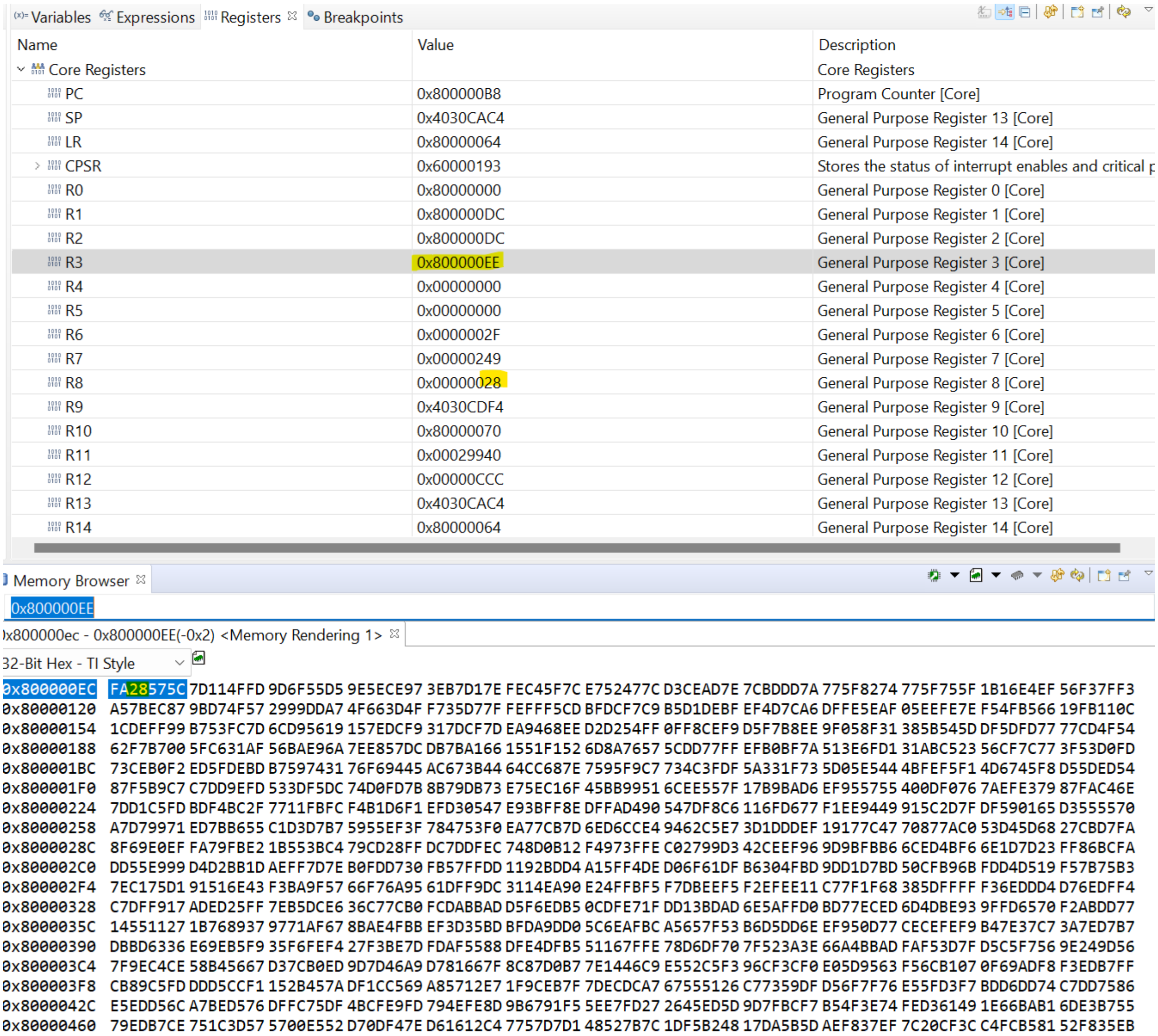
0x800000D8

0x800000d8 - 0x800000D8 <Memory Rendering 1>

32-Bit Hex - TI Style

```
0x800000D8  00000278 800000B8 800000D8 800000DA E7AE9D04 FAB4575C 7D114FFD 9D6F55D5 9E5ECE97 3EB7D17E FEC45F7C E752477C D3CEAD7E
0x8000010C  7CBDDD7A 775F8274 775F755F 1B16E4EF 56F37FF3 A57BEC87 9BD74F57 2999DDA7 4F663D4F F735D77F FEFFF5CD BFDCF7C9 B5D1DEBF
0x80000140  EF4D7CA6 DFFE5EAF 05EEFE7E F54FB566 19FB110C 1CDEFF99 B753FC7D 6CD95619 157EDCF9 317DCF7D EA9468EE D2D254FF 0FF8CEF9
0x80000174  D5F7B8EE 9F058F31 385B545D DF5DFD77 77CD4F54 62F7B700 5FC631AF 56BAE96A 7EE857DC DB7BA166 1551F152 6D8A7657 5CDD77FF
0x800001A8  EFB0BF7A 513E6FD1 31ABC523 56CF7C77 3F53D0FD 73CEB0F2 ED5FDEBD B7597431 76F69445 AC673B44 64CC687E 7595F9C7 734C3FDF
0x800001DC  5A331F73 5D05E544 4BFEF5F1 4D6745F8 D55DED54 87F5B9C7 C7DD9EFD 533DF5DC 74D0FD7B 8B79DB73 E75EC16F 45BB9951 6CEE557F
0x80000210  17B9BAD6 EF955755 400DF076 7AEFE379 87FAC46E 7DD1C5FD BDF4BC2F 7711FBFC F4B1D6F1 EFD30547 E93BFF8E DFFAD490 547DF8C6
0x80000244  116FD677 F1EE9449 915C2D7F DF590165 D3555570 A7D79971 ED7BB655 C1D3D7B7 5955EF3F 784753F0 EA77CB7D 6ED6CCE4 9462C5E7
0x80000278  3D1DDDEF 19177C47 70877AC0 53D45D68 27CBD7FA 8F69E0EF FA79FBE2 1B553BC4 79CD28FF DC7DDFEC 748D0B12 F4973FFE C02799D3
0x800002AC  42CEEF96 9D9BFBB6 6CED4BF6 6E1D7D23 FF86BCFA DD55E999 D4D2BB1D AEFF7D7E B0FDD730 FB57FFDD 1192BDD4 A15FF4DE D06F61DF
0x800002E0  B6304FBD 9DD1D7BD 50CFB96B FDD4D519 F57B75B3 7EC175D1 91516E43 F3BA9F57 66F76A95 61DFF9DC 3114EA90 E24FFBF5 F7DBEEF5
0x80000314  F2EFEE11 C77F1F68 385DFFFF F36EDDD4 D76EDFF4 C7DFF917 ADED25FF 7EB5DCE6 36C77CB0 FCDABBAD D5F6EDB5 0CDFE71F DD13BDAD
0x80000348  6E5AFFD0 BD77ECED 6D4DBE93 9FFD6570 F2ABDD77 14551127 1B768937 9771AF67 8BAE4FBB EF3D35BD BFDA9DD0 5C6EAFBC A5657F53
0x8000037C  B6D5DD6E EF950D77 CECEFEF9 B47E37C7 3A7ED7B7 DBBD6336 E69EB5F9 35F6FEF4 27F3BE7D FDAF5588 DFE4DFB5 51167FFE 78D6DF70
0x800003B0  7F523A3E 66A4BBAD FAF53D7F D5C5F756 9E249D56 7F9EC4CE 58B45667 D37CB0ED 9D7D46A9 D781667F 8C87D0B7 7E1446C9 E552C5F3
0x800003E4  96CF3CF0 E05D9563 F56CB107 0F69ADF8 F3EDB7FF CB89C5FD DDD5CCF1 152B457A DF1CC569 A85712E7 1F9CEB7F 7DECDCA7 67555126
0x80000418  C77359DF D56F7F76 E55FD3F7 BDD6DD74 C7DD7586 E5EDD56C A7BED576 DFFC75DF 4BCFE9FD 794EFE8D 9B6791F5 5EE7FD27 2645ED5D
0x8000044C  9D7FBCF7 B54F3E74 FED36149 1E66BAB1 6DE3B755 79EDB7CE 751C3D57 5700E552 D70DF47E D61612C4 7757D7D1 48527B7C 1DF5B248
```

This screen shot is after the addition function is finished running. We expect a 0x278, because our test values are 32 – 47. When these 16 numbers are added, we get 632 decimal, which is 0x278. It is stored in [R1], which is highlighted in blue.



Name	Value	Description
Core Registers		Core Registers
PC	0x800000B8	Program Counter [Core]
SP	0x4030CAC4	General Purpose Register 13 [Core]
LR	0x80000064	General Purpose Register 14 [Core]
CPSR	0x60000193	Stores the status of interrupt enables and critical p
R0	0x80000000	General Purpose Register 0 [Core]
R1	0x800000DC	General Purpose Register 1 [Core]
R2	0x800000DC	General Purpose Register 2 [Core]
R3	0x800000EE	General Purpose Register 3 [Core]
R4	0x00000000	General Purpose Register 4 [Core]
R5	0x00000000	General Purpose Register 5 [Core]
R6	0x0000002F	General Purpose Register 6 [Core]
R7	0x00000249	General Purpose Register 7 [Core]
R8	0x00000028	General Purpose Register 8 [Core]
R9	0x4030CDF4	General Purpose Register 9 [Core]
R10	0x80000070	General Purpose Register 10 [Core]
R11	0x00029940	General Purpose Register 11 [Core]
R12	0x00000CCC	General Purpose Register 12 [Core]
R13	0x4030CAC4	General Purpose Register 13 [Core]
R14	0x80000064	General Purpose Register 14 [Core]

0x800000EE

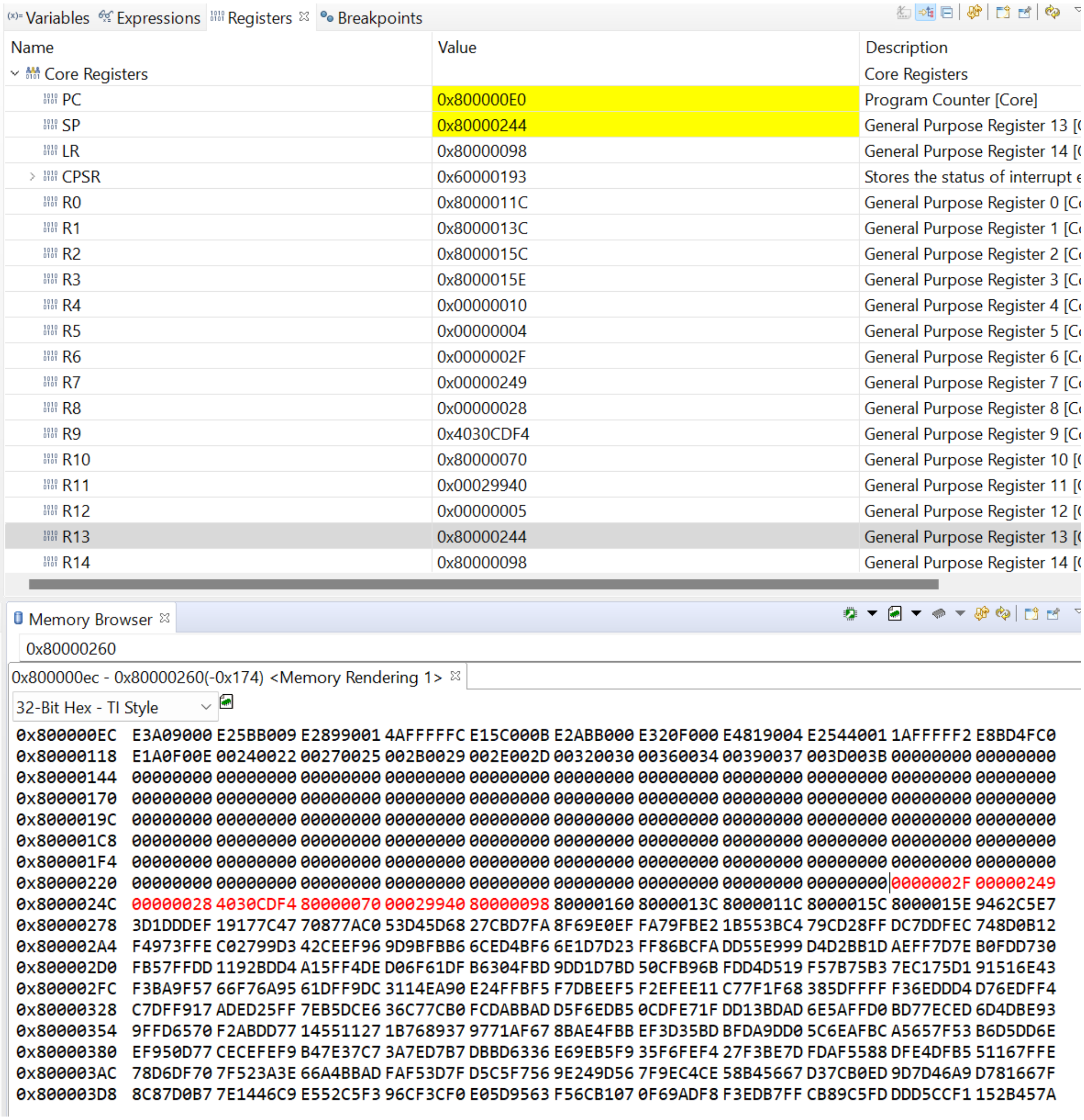
x800000ec - 0x800000EE(-0x2) <Memory Rendering 1>

32-Bit Hex - TI Style

```
0x800000EC  FA28575C 7D114FFD 9D6F55D5 9E5ECE97 3EB7D17E FEC45F7C E752477C D3CEAD7E 7CBDDD7A 775F8274 775F755F 1B16E4EF 56F37FF3
0x80000120  A57BEC87 9BD74F57 2999DDA7 4F663D4F F735D77F FEFFF5CD BFDCF7C9 B5D1DEBF EF4D7CA6 DFFE5EAF 05EEFE7E F54FB566 19FB110C
0x80000154  1CDEFF99 B753FC7D 6CD95619 157EDCF9 317DCF7D EA9468EE D2D254FF 0FF8CEF9 D5F7B8EE 9F058F31 385B545D DF5DFD77 77CD4F54
0x80000188  62F7B700 5FC631AF 56BAE96A 7EE857DC DB7BA166 1551F152 6D8A7657 5CDD77FF EFB0BF7A 513E6FD1 31ABC523 56CF7C77 3F53D0FD
0x800001BC  73CEB0F2 ED5FDEBD B7597431 76F69445 AC673B44 64CC687E 7595F9C7 734C3FDF 5A331F73 5D05E544 4BFEF5F1 4D6745F8 D55DED54
0x800001F0  87F5B9C7 C7DD9EFD 533DF5DC 74D0FD7B 8B79DB73 E75EC16F 45BB9951 6CEE557F 17B9BAD6 EF955755 400DF076 7AEFE379 87FAC46E
0x80000224  7DD1C5FD BDF4BC2F 7711FBFC F4B1D6F1 EFD30547 E93BFF8E DFFAD490 547DF8C6 116FD677 F1EE9449 915C2D7F DF590165 D3555570
0x80000258  A7D79971 ED7BB655 C1D3D7B7 5955EF3F 784753F0 EA77CB7D 6ED6CCE4 9462C5E7 3D1DDDEF 19177C47 70877AC0 53D45D68 27CBD7FA
0x8000028C  8F69E0EF FA79FBE2 1B553BC4 79CD28FF DC7DDFEC 748D0B12 F4973FFE C02799D3 42CEEF96 9D9BFBB6 6CED4BF6 6E1D7D23 FF86BCFA
0x800002C0  DD55E999 D4D2BB1D AEFF7D7E B0FDD730 FB57FFDD 1192BDD4 A15FF4DE D06F61DF B6304FBD 9DD1D7BD 50CFB96B FDD4D519 F57B75B3
0x800002F4  7EC175D1 91516E43 F3BA9F57 66F76A95 61DFF9DC 3114EA90 E24FFBF5 F7DBEEF5 F2EFEE11 C77F1F68 385DFFFF F36EDDD4 D76EDFF4
0x80000328  C7DFF917 ADED25FF 7EB5DCE6 36C77CB0 FCDABBAD D5F6EDB5 0CDFE71F DD13BDAD 6E5AFFD0 BD77ECED 6D4DBE93 9FFD6570 F2ABDD77
0x8000035C  14551127 1B768937 9771AF67 8BAE4FBB EF3D35BD BFDA9DD0 5C6EAFBC A5657F53 B6D5DD6E EF950D77 CECEFEF9 B47E37C7 3A7ED7B7
0x80000390  DBBD6336 E69EB5F9 35F6FEF4 27F3BE7D FDAF5588 DFE4DFB5 51167FFE 78D6DF70 7F523A3E 66A4BBAD FAF53D7F D5C5F756 9E249D56
0x800003C4  7F9EC4CE 58B45667 D37CB0ED 9D7D46A9 D781667F 8C87D0B7 7E1446C9 E552C5F3 96CF3CF0 E05D9563 F56CB107 0F69ADF8 F3EDB7FF
0x800003F8  CB89C5FD DDD5CCF1 152B457A DF1CC569 A85712E7 1F9CEB7F 7DECDCA7 67555126 C77359DF D56F7F76 E55FD3F7 BDD6DD74 C7DD7586
0x8000042C  E5EDD56C A7BED576 DFFC75DF 4BCFE9FD 794EFE8D 9B6791F5 5EE7FD27 2645ED5D 9D7FBCF7 B54F3E74 FED36149 1E66BAB1 6DE3B755
0x80000460  79EDB7CE 751C3D57 5700E552 D70DF47E D61612C4 7757D7D1 48527B7C 1DF5B248 17DA5B5D AEF837EF 7C20CF3C C4FCB581 52F835EB
```

This screen shot is after the division function is finished running. 632 / 16 = 40 (decimal) which is 0x28. The highlighted yellow shows [R3], then the value we calculated is saved in R8, and that was stored in memory at [R3] blue highlighted part. (Blue block starts at 0x8….EC, but our spot we saved R8 at is 0x8….EE, which is the yellow highlighted part inside the blue block)

Part 2 Screen Shots (Code is in Appendix 2 at the end)



Name	Value	Description
Core Registers		Core Registers
PC	0x800000E0	Program Counter [Core]
SP	0x80000244	General Purpose Register 13 [(
LR	0x80000098	General Purpose Register 14 [(
CPSR	0x60000193	Stores the status of interrupt e
R0	0x8000011C	General Purpose Register 0 [Co
R1	0x8000013C	General Purpose Register 1 [Co
R2	0x8000015C	General Purpose Register 2 [Co
R3	0x8000015E	General Purpose Register 3 [Co
R4	0x00000010	General Purpose Register 4 [Co
R5	0x00000004	General Purpose Register 5 [Co
R6	0x0000002F	General Purpose Register 6 [Co
R7	0x00000249	General Purpose Register 7 [Co
R8	0x00000028	General Purpose Register 8 [Co
R9	0x4030CDF4	General Purpose Register 9 [Co
R10	0x80000070	General Purpose Register 10 [(
R11	0x00029940	General Purpose Register 11 [(
R12	0x00000005	General Purpose Register 12 [(
R13	0x80000244	General Purpose Register 13 [(
R14	0x80000098	General Purpose Register 14 [(

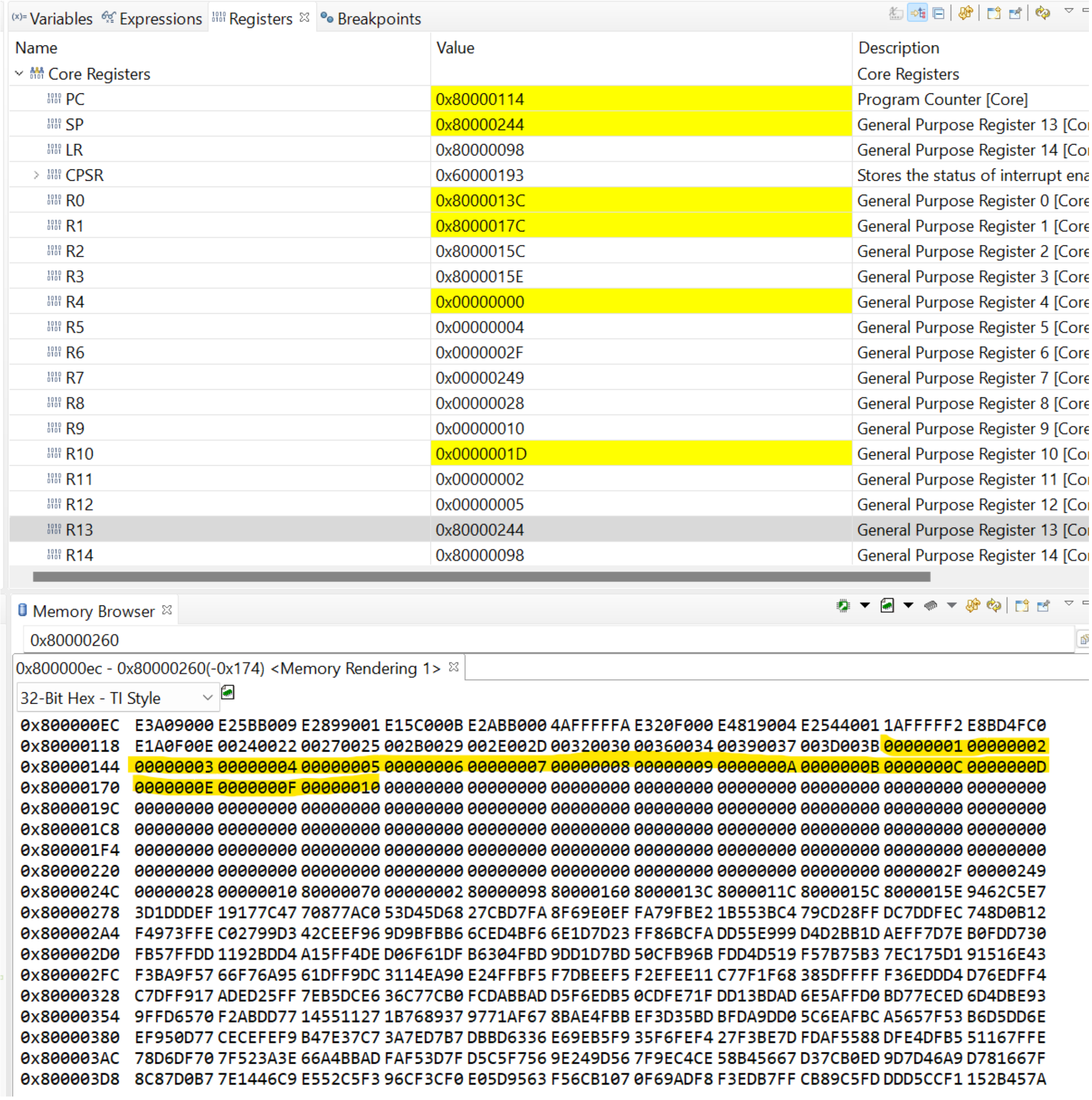
0x80000260

0x800000ec - 0x80000260(-0x174) <Memory Rendering 1>

32-Bit Hex - TI Style

```
0x800000EC  E3A09000 E25BB009 E2899001 4AFFFFFC E15C000B E2ABB000 E320F000 E4819004 E2544001 1AFFFFF2 E8BD4FC0
0x80000118  E1A0F00E 00240022 00270025 002B0029 002E002D 00320030 00360034 00390037 003D003B 00000000 00000000
0x80000144  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000170  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x8000019C  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001C8  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001F4  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000220  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000002F 00000249
0x8000024C  00000028 4030CDF4 80000070 00029940 80000098 80000160 8000013C 8000011C 8000015C 8000015E 9462C5E7
0x80000278  3D1DDDEF 19177C47 70877AC0 53D45D68 27CBD7FA 8F69E0EF FA79FBE2 1B553BC4 79CD28FF DC7DDFEC 748D0B12
0x800002A4  F4973FFE C02799D3 42CEEF96 9D9BFBB6 6CED4BF6 6E1D7D23 FF86BCFA DD55E999 D4D2BB1D AEFF7D7E B0FDD730
0x800002D0  FB57FFDD 1192BDD4 A15FF4DE D06F61DF B6304FBD 9DD1D7BD 50CFB96B FDD4D519 F57B75B3 7EC175D1 91516E43
0x800002FC  F3BA9F57 66F76A95 61DFF9DC 3114EA90 E24FFBF5 F7DBEEF5 F2EFEE11 C77F1F68 385DFFFF F36EDDD4 D76EDFF4
0x80000328  C7DFF917 ADED25FF 7EB5DCE6 36C77CB0 FCDABBAD D5F6EDB5 0CDFE71F DD13BDAD 6E5AFFD0 BD77ECED 6D4DBE93
0x80000354  9FFD6570 F2ABDD77 14551127 1B768937 9771AF67 8BAE4FBB EF3D35BD BFDA9DD0 5C6EAFBC A5657F53 B6D5DD6E
0x80000380  EF950D77 CECEFEF9 B47E37C7 3A7ED7B7 DBBD6336 E69EB5F9 35F6FEF4 27F3BE7D FDAF5588 DFE4DFB5 51167FFE
0x800003AC  78D6DF70 7F523A3E 66A4BBAD FAF53D7F D5C5F756 9E249D56 7F9EC4CE 58B45667 D37CB0ED 9D7D46A9 D781667F
0x800003D8  8C87D0B7 7E1446C9 E552C5F3 96CF3CF0 E05D9563 F56CB107 0F69ADF8 F3EDB7FF CB89C5FD DDD5CCF1 152B457A
```

This screen shot is after we branch (BL) to the Calc Celsius Temp function and we have pushed the registers we will be using inside this function to the stack, which is in red. It starts with R6 (0x00…2F) and ends with R11 (0x00029940) and has R14 (0x80….98)



Name	Value	Description
Core Registers		Core Registers
PC	0x80000114	Program Counter [Core]
SP	0x80000244	General Purpose Register 13 [Co
LR	0x80000098	General Purpose Register 14 [Co
CPSR	0x60000193	Stores the status of interrupt ena
R0	0x8000013C	General Purpose Register 0 [Core
R1	0x8000017C	General Purpose Register 1 [Core
R2	0x8000015C	General Purpose Register 2 [Core
R3	0x8000015E	General Purpose Register 3 [Core
R4	0x00000000	General Purpose Register 4 [Core
R5	0x00000004	General Purpose Register 5 [Core
R6	0x0000002F	General Purpose Register 6 [Core
R7	0x00000249	General Purpose Register 7 [Core
R8	0x00000028	General Purpose Register 8 [Core
R9	0x00000010	General Purpose Register 9 [Core
R10	0x0000001D	General Purpose Register 10 [Co
R11	0x00000002	General Purpose Register 11 [Co
R12	0x00000005	General Purpose Register 12 [Co
R13	0x80000244	General Purpose Register 13 [Co
R14	0x80000098	General Purpose Register 14 [Co

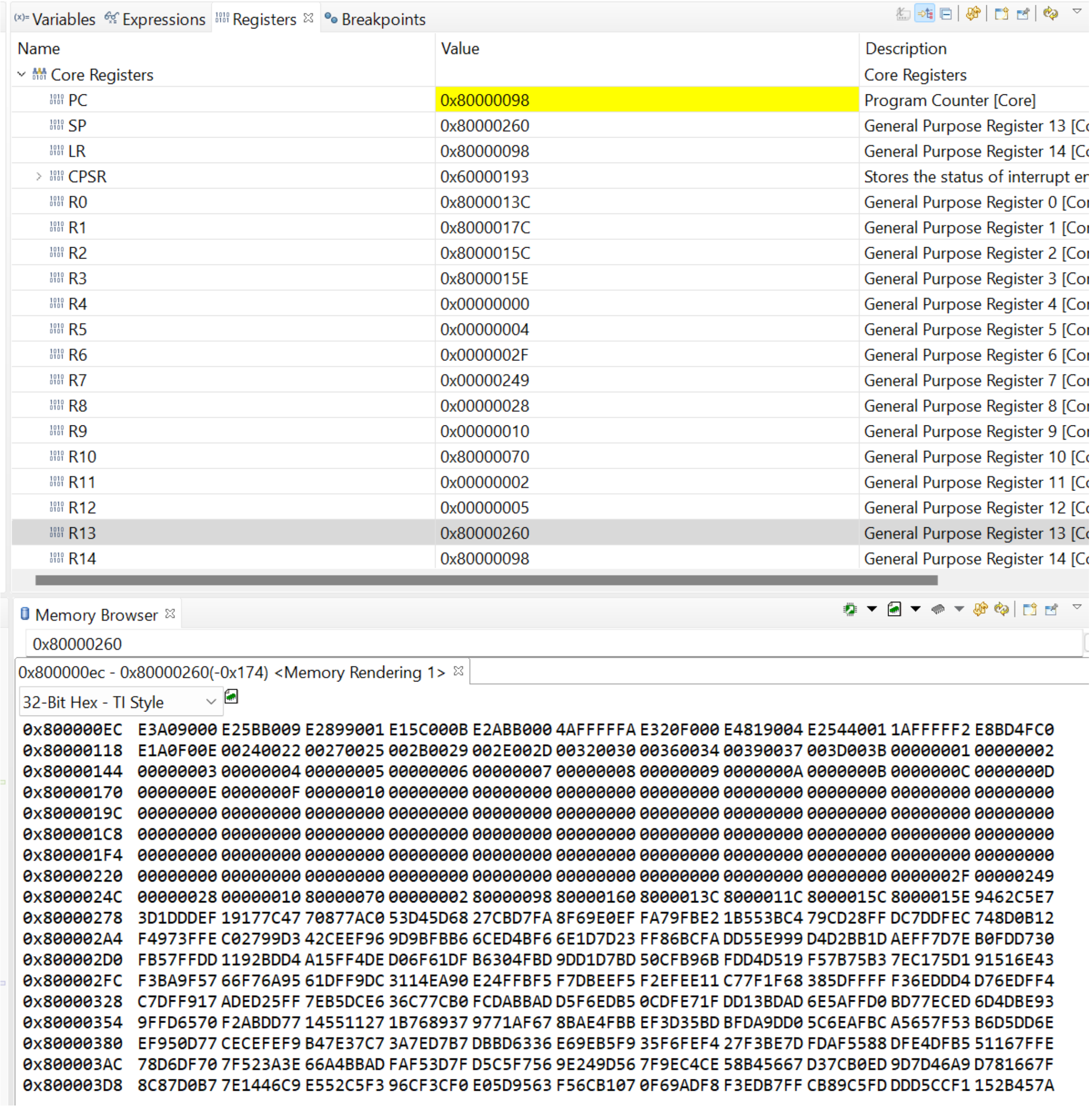
0x80000260

0x800000ec - 0x80000260(-0x174) <Memory Rendering 1>

32-Bit Hex - TI Style

```
0x800000EC  E3A09000 E25BB009 E2899001 E15C000B E2ABB000 4AFFFFFA E320F000 E4819004 E2544001 1AFFFFF2 E8BD4FC0
0x80000118  E1A0F00E 00240022 00270025 002B0029 002E002D 00320030 00360034 00390037 003D003B 00000001 00000002
0x80000144  00000003 00000004 00000005 00000006 00000007 00000008 00000009 0000000A 0000000B 0000000C 0000000D
0x80000170  0000000E 0000000F 00000010 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x8000019C  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001C8  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001F4  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000220  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000002F 00000249
0x8000024C  00000028 00000010 80000070 00000002 80000098 80000160 8000013C 8000011C 8000015C 8000015E 9462C5E7
0x80000278  3D1DDDEF 19177C47 70877AC0 53D45D68 27CBD7FA 8F69E0EF FA79FBE2 1B553BC4 79CD28FF DC7DDFEC 748D0B12
0x800002A4  F4973FFE C02799D3 42CEEF96 9D9BFBB6 6CED4BF6 6E1D7D23 FF86BCFA DD55E999 D4D2BB1D AEFF7D7E B0FDD730
0x800002D0  FB57FFDD 1192BDD4 A15FF4DE D06F61DF B6304FBD 9DD1D7BD 50CFB96B FDD4D519 F57B75B3 7EC175D1 91516E43
0x800002FC  F3BA9F57 66F76A95 61DFF9DC 3114EA90 E24FFBF5 F7DBEEF5 F2EFEE11 C77F1F68 385DFFFF F36EDDD4 D76EDFF4
0x80000328  C7DFF917 ADED25FF 7EB5DCE6 36C77CB0 FCDABBAD D5F6EDB5 0CDFE71F DD13BDAD 6E5AFFD0 BD77ECED 6D4DBE93
0x80000354  9FFD6570 F2ABDD77 14551127 1B768937 9771AF67 8BAE4FBB EF3D35BD BFDA9DD0 5C6EAFBC A5657F53 B6D5DD6E
0x80000380  EF950D77 CECEFEF9 B47E37C7 3A7ED7B7 DBBD6336 E69EB5F9 35F6FEF4 27F3BE7D FDAF5588 DFE4DFB5 51167FFE
0x800003AC  78D6DF70 7F523A3E 66A4BBAD FAF53D7F D5C5F756 9E249D56 7F9EC4CE 58B45667 D37CB0ED 9D7D46A9 D781667F
0x800003D8  8C87D0B7 7E1446C9 E552C5F3 96CF3CF0 E05D9563 F56CB107 0F69ADF8 F3EDB7FF CB89C5FD DDD5CCF1 152B457A
```

This screenshot is after the Calc Celsius Temp function is finished running. We have our 16 expected values saved in the right spot, which is highlighted in yellow (1 – 16)



Name	Value	Description
Core Registers		Core Registers
PC	0x80000098	Program Counter [Core]
SP	0x80000260	General Purpose Register 13 [Co
LR	0x80000098	General Purpose Register 14 [Co
CPSR	0x60000193	Stores the status of interrupt er
R0	0x8000013C	General Purpose Register 0 [Cor
R1	0x8000017C	General Purpose Register 1 [Cor
R2	0x8000015C	General Purpose Register 2 [Cor
R3	0x8000015E	General Purpose Register 3 [Cor
R4	0x00000000	General Purpose Register 4 [Cor
R5	0x00000004	General Purpose Register 5 [Cor
R6	0x0000002F	General Purpose Register 6 [Cor
R7	0x00000249	General Purpose Register 7 [Cor
R8	0x00000028	General Purpose Register 8 [Cor
R9	0x00000010	General Purpose Register 9 [Cor
R10	0x80000070	General Purpose Register 10 [Co
R11	0x00000002	General Purpose Register 11 [Co
R12	0x00000005	General Purpose Register 12 [Co
R13	0x80000260	General Purpose Register 13 [Co
R14	0x80000098	General Purpose Register 14 [Co

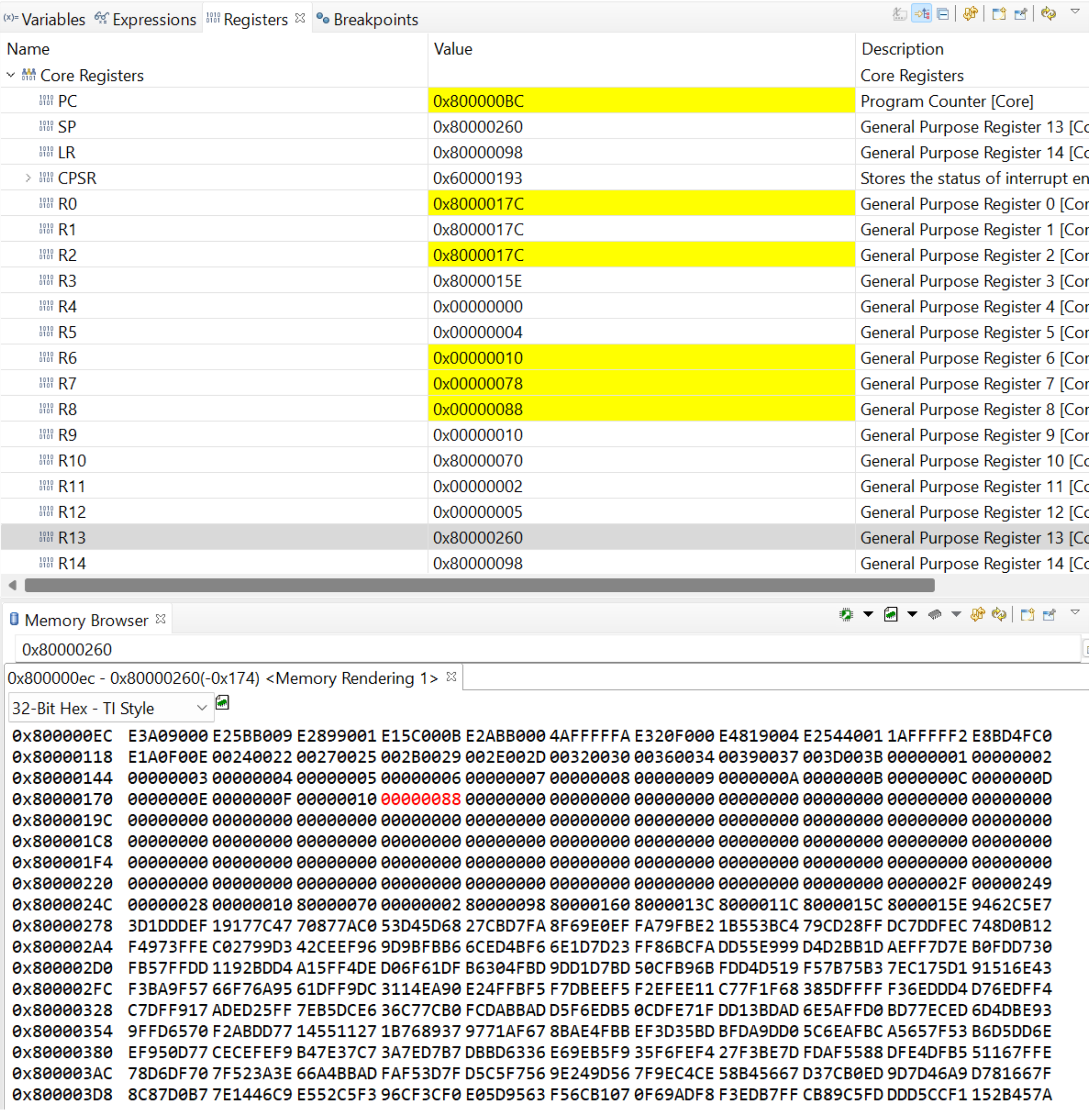
0x80000260

0x800000ec - 0x80000260(-0x174) <Memory Rendering 1>

32-Bit Hex - TI Style

```
0x800000EC  E3A09000 E25BB009 E2899001 E15C000B E2ABB000 4AFFFFFA E320F000 E4819004 E2544001 1AFFFFF2 E8BD4FC0
0x80000118  E1A0F00E 00240022 00270025 002B0029 002E002D 00320030 00360034 00390037 003D003B 00000001 00000002
0x80000144  00000003 00000004 00000005 00000006 00000007 00000008 00000009 0000000A 0000000B 0000000C 0000000D
0x80000170  0000000E 0000000F 00000010 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x8000019C  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001C8  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001F4  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000220  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000002F 00000249
0x8000024C  00000028 00000010 80000070 00000002 80000098 80000160 8000013C 8000011C 8000015C 8000015E 9462C5E7
0x80000278  3D1DDDEF 19177C47 70877AC0 53D45D68 27CBD7FA 8F69E0EF FA79FBE2 1B553BC4 79CD28FF DC7DDFEC 748D0B12
0x800002A4  F4973FFE C02799D3 42CEEF96 9D9BFBB6 6CED4BF6 6E1D7D23 FF86BCFA DD55E999 D4D2BB1D AEFF7D7E B0FDD730
0x800002D0  FB57FFDD 1192BDD4 A15FF4DE D06F61DF B6304FBD 9DD1D7BD 50CFB96B FDD4D519 F57B75B3 7EC175D1 91516E43
0x800002FC  F3BA9F57 66F76A95 61DFF9DC 3114EA90 E24FFBF5 F7DBEEF5 F2EFEE11 C77F1F68 385DFFFF F36EDDD4 D76EDFF4
0x80000328  C7DFF917 ADED25FF 7EB5DCE6 36C77CB0 FCDABBAD D5F6EDB5 0CDFE71F DD13BDAD 6E5AFFD0 BD77ECED 6D4DBE93
0x80000354  9FFD6570 F2ABDD77 14551127 1B768937 9771AF67 8BAE4FBB EF3D35BD BFDA9DD0 5C6EAFBC A5657F53 B6D5DD6E
0x80000380  EF950D77 CECEFEF9 B47E37C7 3A7ED7B7 DBBD6336 E69EB5F9 35F6FEF4 27F3BE7D FDAF5588 DFE4DFB5 51167FFE
0x800003AC  78D6DF70 7F523A3E 66A4BBAD FAF53D7F D5C5F756 9E249D56 7F9EC4CE 58B45667 D37CB0ED 9D7D46A9 D781667F
0x800003D8  8C87D0B7 7E1446C9 E552C5F3 96CF3CF0 E05D9563 F56CB107 0F69ADF8 F3EDB7FF CB89C5FD DDD5CCF1 152B457A
```

This screenshot is after we pop the stack / restore registers, and one instruction after the BL Calc Celsius Temp, to show we are back where we need to be after the BL function.



Name	Value	Description
Core Registers		Core Registers
PC	0x800000BC	Program Counter [Core]
SP	0x80000260	General Purpose Register 13 [Cc
LR	0x80000098	General Purpose Register 14 [Cc
CPSR	0x60000193	Stores the status of interrupt en
R0	0x8000017C	General Purpose Register 0 [Cor
R1	0x8000017C	General Purpose Register 1 [Cor
R2	0x8000017C	General Purpose Register 2 [Cor
R3	0x8000015E	General Purpose Register 3 [Cor
R4	0x00000000	General Purpose Register 4 [Cor
R5	0x00000004	General Purpose Register 5 [Cor
R6	0x00000010	General Purpose Register 6 [Cor
R7	0x00000078	General Purpose Register 7 [Cor
R8	0x00000088	General Purpose Register 8 [Cor
R9	0x00000010	General Purpose Register 9 [Cor
R10	0x80000070	General Purpose Register 10 [Cc
R11	0x00000002	General Purpose Register 11 [Cc
R12	0x00000005	General Purpose Register 12 [Cc
R13	0x80000260	General Purpose Register 13 [Cc
R14	0x80000098	General Purpose Register 14 [Cc

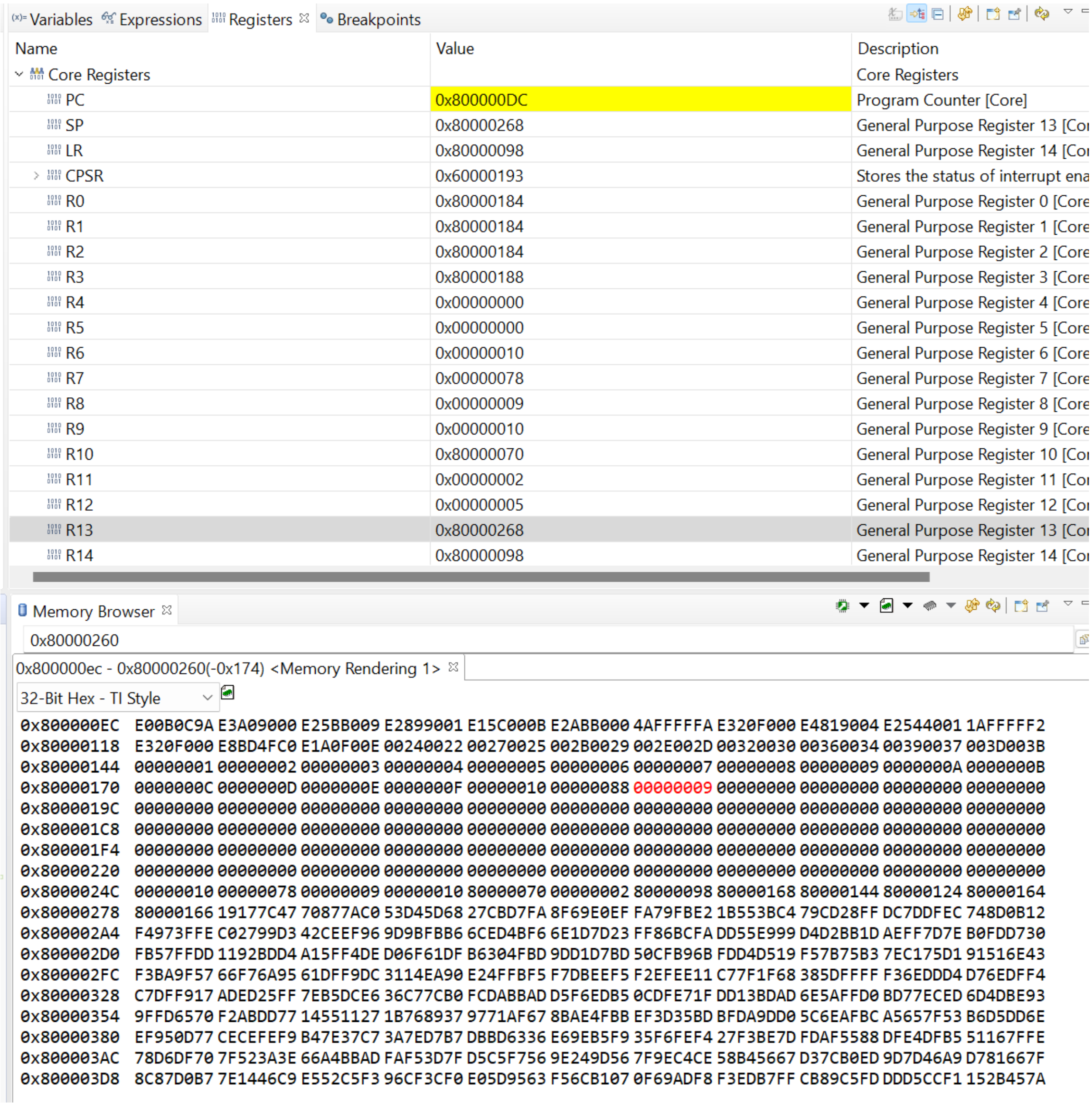
0x80000260

0x800000ec - 0x80000260(-0x174) <Memory Rendering 1>

32-Bit Hex - TI Style

```
0x800000EC  E3A09000 E25BB009 E2899001 E15C000B E2ABB000 4AFFFFFA E320F000 E4819004 E2544001 1AFFFFF2 E8BD4FC0
0x80000118  E1A0F00E 00240022 00270025 002B0029 002E002D 00320030 00360034 00390037 003D003B 00000001 00000002
0x80000144  00000003 00000004 00000005 00000006 00000007 00000008 00000009 0000000A 0000000B 0000000C 0000000D
0x80000170  0000000E 0000000F 00000010 00000088 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x8000019C  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001C8  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001F4  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000220  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 0000002F 00000249
0x8000024C  00000028 00000010 80000070 00000002 80000098 80000160 8000013C 8000011C 8000015C 8000015E 9462C5E7
0x80000278  3D1DDDEF 19177C47 70877AC0 53D45D68 27CBD7FA 8F69E0EF FA79FBE2 1B553BC4 79CD28FF DC7DDFEC 748D0B12
0x800002A4  F4973FFE C02799D3 42CEEF96 9D9BFBB6 6CED4BF6 6E1D7D23 FF86BCFA DD55E999 D4D2BB1D AEFF7D7E B0FDD730
0x800002D0  FB57FFDD 1192BDD4 A15FF4DE D06F61DF B6304FBD 9DD1D7BD 50CFB96B FDD4D519 F57B75B3 7EC175D1 91516E43
0x800002FC  F3BA9F57 66F76A95 61DFF9DC 3114EA90 E24FFBF5 F7DBEEF5 F2EFEE11 C77F1F68 385DFFFF F36EDDD4 D76EDFF4
0x80000328  C7DFF917 ADED25FF 7EB5DCE6 36C77CB0 FCDABBAD D5F6EDB5 0CDFE71F DD13BDAD 6E5AFFD0 BD77ECED 6D4DBE93
0x80000354  9FFD6570 F2ABDD77 14551127 1B768937 9771AF67 8BAE4FBB EF3D35BD BFDA9DD0 5C6EAFBC A5657F53 B6D5DD6E
0x80000380  EF950D77 CECEFEF9 B47E37C7 3A7ED7B7 DBBD6336 E69EB5F9 35F6FEF4 27F3BE7D FDAF5588 DFE4DFB5 51167FFE
0x800003AC  78D6DF70 7F523A3E 66A4BBAD FAF53D7F D5C5F756 9E249D56 7F9EC4CE 58B45667 D37CB0ED 9D7D46A9 D781667F
0x800003D8  8C87D0B7 7E1446C9 E552C5F3 96CF3CF0 E05D9563 F56CB107 0F69ADF8 F3EDB7FF CB89C5FD DDD5CCF1 152B457A
```

This screenshot is after the addition of all temps has been completed and stored in memory (red part)



Name	Value	Description
Core Registers		Core Registers
PC	0x800000DC	Program Counter [Core]
SP	0x80000268	General Purpose Register 13 [Cor
LR	0x80000098	General Purpose Register 14 [Cor
CPSR	0x60000193	Stores the status of interrupt ena
R0	0x80000184	General Purpose Register 0 [Core
R1	0x80000184	General Purpose Register 1 [Core
R2	0x80000184	General Purpose Register 2 [Core
R3	0x80000188	General Purpose Register 3 [Core
R4	0x00000000	General Purpose Register 4 [Core
R5	0x00000000	General Purpose Register 5 [Core
R6	0x00000010	General Purpose Register 6 [Core
R7	0x00000078	General Purpose Register 7 [Core
R8	0x00000009	General Purpose Register 8 [Core
R9	0x00000010	General Purpose Register 9 [Core
R10	0x80000070	General Purpose Register 10 [Cor
R11	0x00000002	General Purpose Register 11 [Cor
R12	0x00000005	General Purpose Register 12 [Cor
R13	0x80000268	General Purpose Register 13 [Cor
R14	0x80000098	General Purpose Register 14 [Cor

0x80000260

0x800000ec - 0x80000260(-0x174) <Memory Rendering 1>

32-Bit Hex - TI Style

```
0x800000EC  E00B0C9A E3A09000 E25BB009 E2899001 E15C000B E2ABB000 4AFFFFFA E320F000 E4819004 E2544001 1AFFFFF2
0x80000118  E320F000 E8BD4FC0 E1A0F00E 00240022 00270025 002B0029 002E002D 00320030 00360034 00390037 003D003B
0x80000144  00000001 00000002 00000003 00000004 00000005 00000006 00000007 00000008 00000009 0000000A 0000000B
0x80000170  0000000C 0000000D 0000000E 0000000F 00000010 00000088 00000009 00000000 00000000 00000000 00000000
0x8000019C  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001C8  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x800001F4  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x80000220  00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0x8000024C  00000010 00000078 00000009 00000010 80000070 00000002 80000098 80000168 80000144 80000124 80000164
0x80000278  80000166 19177C47 70877AC0 53D45D68 27CBD7FA 8F69E0EF FA79FBE2 1B553BC4 79CD28FF DC7DDFEC 748D0B12
0x800002A4  F4973FFE C02799D3 42CEEF96 9D9BFBB6 6CED4BF6 6E1D7D23 FF86BCFA DD55E999 D4D2BB1D AEFF7D7E B0FDD730
0x800002D0  FB57FFDD 1192BDD4 A15FF4DE D06F61DF B6304FBD 9DD1D7BD 50CFB96B FDD4D519 F57B75B3 7EC175D1 91516E43
0x800002FC  F3BA9F57 66F76A95 61DFF9DC 3114EA90 E24FFBF5 F7DBEEF5 F2EFEE11 C77F1F68 385DFFFF F36EDDD4 D76EDFF4
0x80000328  C7DFF917 ADED25FF 7EB5DCE6 36C77CB0 FCDABBAD D5F6EDB5 0CDFE71F DD13BDAD 6E5AFFD0 BD77ECED 6D4DBE93
0x80000354  9FFD6570 F2ABDD77 14551127 1B768937 9771AF67 8BAE4FBB EF3D35BD BFDA9DD0 5C6EAFBC A5657F53 B6D5DD6E
0x80000380  EF950D77 CECEFEF9 B47E37C7 3A7ED7B7 DBBD6336 E69EB5F9 35F6FEF4 27F3BE7D FDAF5588 DFE4DFB5 51167FFE
0x800003AC  78D6DF70 7F523A3E 66A4BBAD FAF53D7F D5C5F756 9E249D56 7F9EC4CE 58B45667 D37CB0ED 9D7D46A9 D781667F
0x800003D8  8C87D0B7 7E1446C9 E552C5F3 96CF3CF0 E05D9563 F56CB107 0F69ADF8 F3EDB7FF CB89C5FD DDD5CCF1 152B457A
```

This screenshot is after the program has finished running. We can see our expected average of 0x9 = 9 decimal (red part). This screenshot also shows all the relevant data. Celsius temps, then total, then average.

Appendix 1

```
@ECE 371 Design Project 1, Part 1
@This program will take 16 8-bit temperatures from the array (Fahrenheit_Temps) and average them
@Then store the average in memory (Average_Temp)
@Uses R1-R3 for Fahreinheit_Temps, Total_Temp and Average_Temp pointers
@Uses R4 - R5 for counters
@Uses R6 - R8 to add and average temps
@Phil Nevins, 11/13/2022
@NOTE: With our test values, we expect the answer to be 39.5 -> 40
@NOTE: Final Answer Yields 0x28 -> 40_decimal

.text
.global _start

_start:

.equ AddCounter, 16             @Set counter for adding temps to 16 (# of temps to be added)
.equ DivideCounter, 4       @Set counter for shifting right to divide to 4 (# of shifts to divide by 16)

LDR R1, =Fahrenheit_Temps@Load pointer to Fahrenheit_Temps array
LDR R2, =Total_Temp             @Load pointer to Total_Temp array
LDR R3, =Average_Temp           @Load pointer to Average_Temp array
MOV R4, #AddCounter             @Load R4 with AddCounter
MOV R5, #DivideCounter          @Load R5 with DivideCounter

Add_Temps_Loop:
    LDRH R6, [R1], #2       @Load a Fahrenheit_Temp half word into R6 then increment to next addr in memory
    LDRH R7, [R2]               @Load a Total_Temp half word into R7. No Need to INC since the total will
                                @get overwritten each time, which is what we want
    ADD R8, R6, R7              @Add new temp from R6 and previous total from R7, store in R8
    STR R8, [R2]            @Move new total in R8 into memory pointed to by R2
    SUBS R4, #1                 @Decrement AddCounter for Add_Temps_Loop counter by 1
    BNE Add_Temps_Loop
    NOP
    @At the end of this loop, all temps in Fahrenheit_Temps array
    @will be added together and saved memory at R2 EA

Avg_Temps_Loop:
    LSR R8, #1                  @Logical Shift Right memory value at R3 EA by 1 bit (divide by 2)
    STRB R8, [R3], #4       @Store value from R8 into EA at R3
    SUBS R5, #1                 @Decrement DivideCounter for Avg_Temps_Loop counter by 1
    BNE Avg_Temps_Loop
    ADC R8, R8, #0              @Add one to R8 if there is a carry from shift right
    STRB R8, [R3]               @Store Value + Carry in R3
    NOP
    @At the end of this loop, Average_Temp array
    @will contain average temperature of Fahrenheit_Temps

Fahrenheit_Temps: .HWORD 0x20, 0x21, 0x22, 0x23, 0x24, 0x25, 0x26, 0x27, 0x28, 0x29, 0x2A, 0x2B, 0x2C, 0x2D,
0x2E, 0x2F
                                            @Test Values Array ^^^^

Total_Temp:         .HWORD 0x0      @Total Temp Array
```

```
Average_Temp:         .HWORD 0x0       @Temp Average Array

.END           @End of program
```

Appendix 2

```
@ECE 371 Design Project 1, Part 2
@This program will take 16 8-bit temperatures from the array (Fahrenheit_Temps) and average them
@Then store the average in memory (Average_Temp)
@Uses R1 - R3 for Fahreinheit_Temps, Total_Temp and Average_Temp pointers
@Uses R4 - R5, R12 Counters and Rounding Factor
@Uses R6 - R11 to convert, add and average temps
@Phil Nevins, 11/13/2022
@NOTE: With our test values, we should get 1-16 celsius temps in memory,
@and then 8.5 exact, 9 rounded average
@NOTE: Final Answer Yields 0x09 -> 9_decimal

.text
.global _start

_start:

.equ AddCounter, 16                          @Set counter for adding temps to 16 (# of temps to be added)
.equ DivideCounter, 4                              @Set counter for shifting right to divide by 16
.equ RoundingFactor, 5                @Set rounding factor for divide function

LDR R13, =STACK                            @Load stack pointer into R13
ADD R13, R13, #0x100          @Point to bottom of the stack
LDR R1, =Celsius_Temps                @Load pointer to Celsius_Temps
LDR R0, =Fahrenheit_Temps@Load pointer to Fahrenheit_Temps array
LDR R2, =Total_Temp                   @Load pointer to Total_Temp array
LDR R3, =Average_Temp                 @Load pointer to Average_Temp array
MOV R4, #AddCounter                   @Load R4 with AddCounter
MOV R5, #DivideCounter                @Load R5 with DivideCounter
MOV R12, #RoundingFactor @Load R12 with decimal 5

BL Calculate_Celsius_Temp
MOV R4, #AddCounter                   @Reset AddCounter. Doing this allows use in Calculate_Celsius_Temp and Add /
Avg temps loop (determined during debugging)
MOV R7, #0                            @Clear R7. Keeps loading #9 into it for no reason (determined during debugging)
ADD R2, R2, #32                       @Adjust R2 pointer to right after Celsius array (determined during debugging)

Add_Temps_Loop:
      LDRH R6, [R0], #4               @Load a Celsius_Temp half word into R6 then increment to next addr in memory
      @Determine why we are using R0 instead of R1. R1 should be Celsius_Temp array
      LDRH R7, [R2]                         @Load a Total_Temp half word into R7. No Need to INC since the total will
get overwritten each time
      ADD R8, R6, R7                        @Add new temp from R6 and previous total from R7, store in R8
      STR R8, [R2]                    @Move new total in R8 into memory pointed to by R2
      SUBS R4, #1                           @Decrement AddCounter for Add_Temps_Loop counter by 1
      BNE Add_Temps_Loop                    @Branch if zero flag not set
      NOP
      @At the end of this loop, all temps in Celsius_Temps array will be added together and saved memory at R2 EA

ADD R3, R3, #34                             @Adjust R3 pointer to display Avg Temp after total temp (deteremined during
debugging)
Avg_Temps_Loop:
      LSR R8, #1                            @Logical Shift Right R8 by 1 bit (divide by 2)
      STRB R8, [R3]                         @Store value in R8 into memory at R3 EA
```

```
    SUBS R5, #1                     @Decrement DivideCounter for Avg_Temps_Loop counter by 1
    BNE Avg_Temps_Loop              @Branch if zero flag not set
    ADC R8, R8, #0                  @Add one to R8 for rounding (will be set if LSR shifts a 1 out)
    STRB R8, [R3]                   @Update total if carry added
    B DONE                          @Branch to end
    @At the end of this loop, Average_Temp array should contain average temperature of Fahrenheit_Temps

Calculate_Celsius_Temp: STMFD R13!, {R6 - R11, R14}  @Function Call for F->C conversion 7 registers saved
Loop1:                                                    @Function to convert from F -> C.... C = 5/9 * (F - 32)
    LDRH R10, [R0], #2              @Load a Fahrenheit_Temp half word into R10 then increment to next addr in
memory
    SUB R10, #0x20                  @R10 - 32
    MUL R11, R10, R12               @Multiply R10 by 5, store in R11
    MOV R9, #0x0                    @Set R9 to 0 for subtract 9 counter

        Divide_By_9:            @Since we have no division command, we have to do the subtract 9 method
            SUBS R11, #0x9          @Subtract 9
            ADD R9, R9, #0x1 @Add 1 to the division counter. Example: 18 / 9 = 2, so R9 will be 2 when we
branch out of loop
            CMP R12, R11            @5 < R11, C = 1
            ADC R11, R11, #0 @Round up (+ 1) if carry set
        BMI Divide_By_9             @Branch if negative flag is not set
        NOP

        Add_Values_To_Memory:
            STR R9, [R1], #0x4                  @Store value in R9 in memory at R1 EA, then increment to next
memory address
            SUBS R4, #0x1                       @Decrement AddCounter for Calculate_Celsius_Temp counter by 1
            BNE Loop1                               @Branch if zero flag not set
            NOP
LDMFD R13!, {R6 - R11, R14}                     @Restore registers
MOV PC, LR                                      @Return to mainline

Fahrenheit_Temps: .HWORD 0x22, 0x24, 0x25, 0x27, 0x29, 0x2B, 0x2D, 0x2E, 0x30, 0x32, 0x34, 0x36, 0x37, 0x39,
0x3B, 0x3D @Test Values Array

Celsius_Temps:          .HWORD 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0
@Converted To Celsius Array

Total_Temp:         .HWORD 0x0      @Total Temp Array

Average_Temp:           .HWORD 0x0      @Temp Average Array

.align 2                            @Stack allocation
STACK: .rept 256
        .byte 0x00
        .endr
DONE:
.END                                    @End of program
```

By signing this statement, I affirm that I did not give any help to any other person, did not receive any help from any other person, except TA and Instructor and did not obtain any information from the Internet or other sources.

Signature______Philip A Nevins________________________

Date:_____11/18/2022________ Philip A Nevins (Nov 18, 2022 18:36 PST)

Design Project 1 Full Report Finished (Phil Nevins)

Final Audit Report 2022-11-19

Created:	2022-11-19
By:	philip nevins (p.nevins971@gmail.com)
Status:	Signed
Transaction ID:	CBJCHBCAABAAYk1WlaYyKqqW5ksq-9hsnxAS_381slgV

"Design Project 1 Full Report Finished (Phil Nevins)" History

Document created by philip nevins (p.nevins971@gmail.com)

2022-11-19 - 2:35:07 AM GMT- IP address: 67.170.140.219

Document emailed to pnevins@pdx.edu for signature

2022-11-19 - 2:35:38 AM GMT

Email viewed by pnevins@pdx.edu

2022-11-19 - 2:35:47 AM GMT- IP address: 66.249.84.95

Signer pnevins@pdx.edu entered name at signing as Philip A Nevins

2022-11-19 - 2:36:18 AM GMT- IP address: 67.170.140.219

Document e-signed by Philip A Nevins (pnevins@pdx.edu)

Signature Date: 2022-11-19 - 2:36:20 AM GMT - Time Source: server- IP address: 67.170.140.219

✓ Agreement completed.

2022-11-19 - 2:36:20 AM GMT