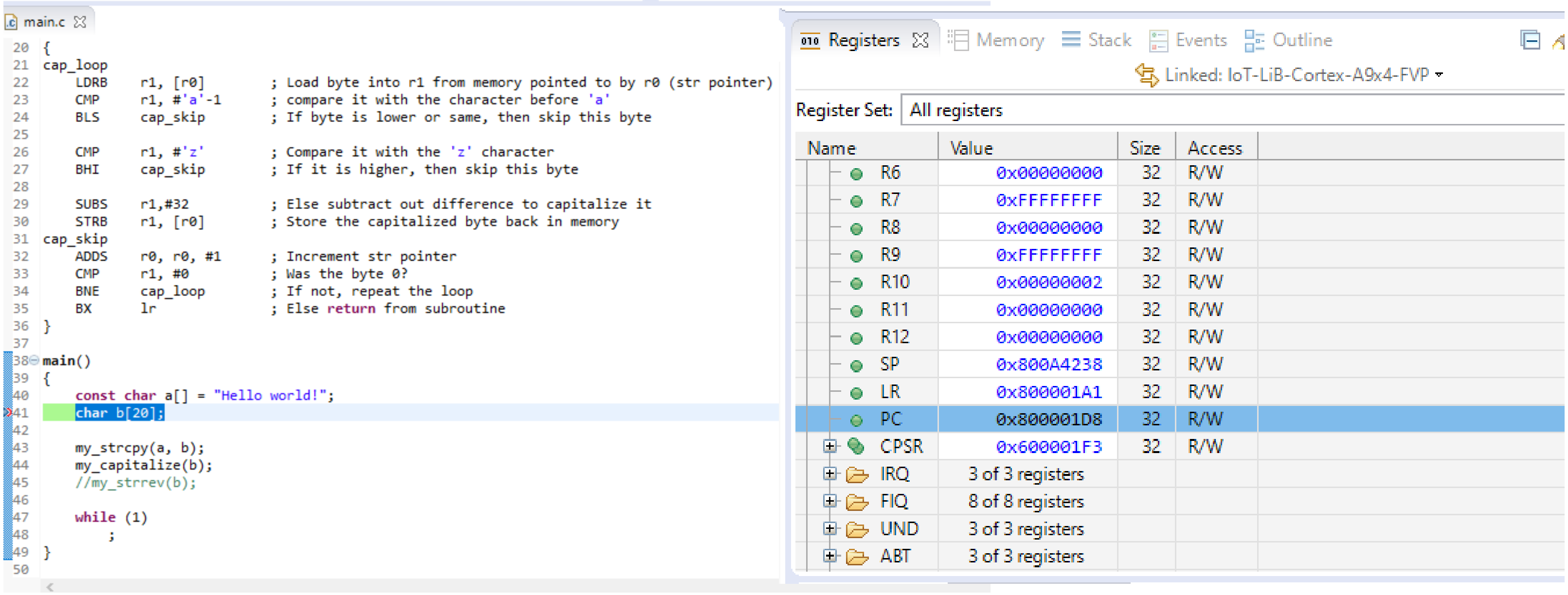
Baron Eiley

EE 277 Embedded SOC Design

1. Run the program until the opening brace in the main function is highlighted. Open the Registers window (Window->Show View->Registers). What are the values of the stack pointer (SP), LR, and the PC? (Insert screenshot of your editor.) **(5 points)**
   I tried setting the breakpoint to the first opening brace on the main function the program wouldn't let me set it there, however, it let me set it on line 41, "char b[20];"

   The value of the stack pointer (SP) register at the point is 0x800A4238. The value of the Link register (LR) is 0x800001A1. The value of the program counter (PC) 0x800001D8**.**



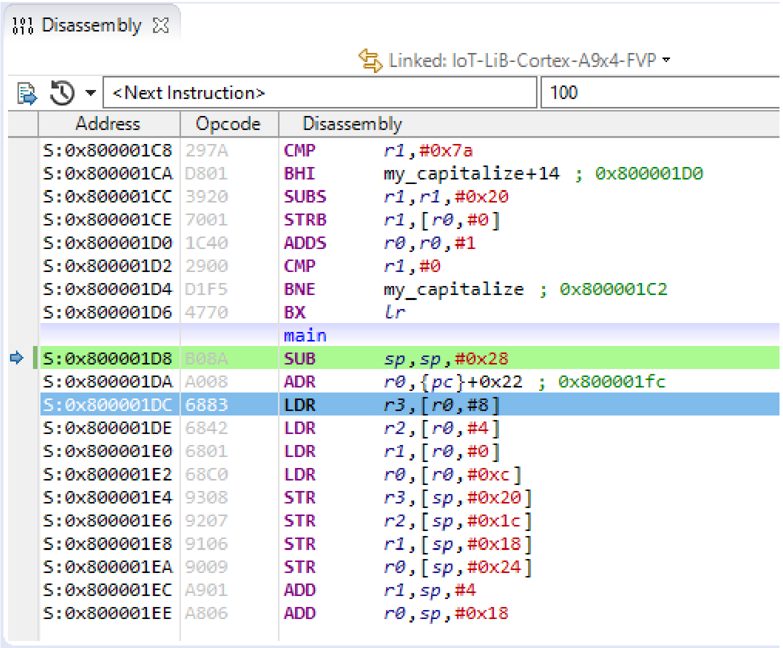
```
20 {
21 cap_loop
22       LDRB     r1, [r0]        ; Load byte into r1 from memory pointed to by r0 (str pointer)
23       CMP      r1, #'a'-1      ; compare it with the character before 'a'
24       BLS      cap_skip        ; If byte is lower or same, then skip this byte
25
26       CMP      r1, #'z'        ; Compare it with the 'z' character
27       BHI      cap_skip        ; If it is higher, then skip this byte
28
29       SUBS     r1,#32          ; Else subtract out difference to capitalize it
30       STRB     r1, [r0]        ; Store the capitalized byte back in memory
31 cap_skip
32       ADDS     r0, r0, #1      ; Increment str pointer
33       CMP      r1, #0          ; Was the byte 0?
34       BNE      cap_loop        ; If not, repeat the loop
35       BX       lr              ; Else return from subroutine
36 }
37
38 main()
39 {
40       const char a[] = "Hello world!";
41       char b[20];
42
43       my_strcpy(a, b);
44       my_capitalize(b);
45       //my_strrev(b);
46
47       while (1)
48           ;
49 }
50
```

Register Set: All registers

Name	Value	Size	Access
R6	0x00000000	32	R/W
R7	0xFFFFFFFF	32	R/W
R8	0x00000000	32	R/W
R9	0xFFFFFFFF	32	R/W
R10	0x00000002	32	R/W
R11	0x00000000	32	R/W
R12	0x00000000	32	R/W
SP	0x800A4238	32	R/W
LR	0x800001A1	32	R/W
PC	0x800001D8	32	R/W
CPSR	0x600001F3	32	R/W
IRQ	3 of 3 registers		
FIQ	8 of 8 registers		
UND	3 of 3 registers		
ABT	3 of 3 registers		

\

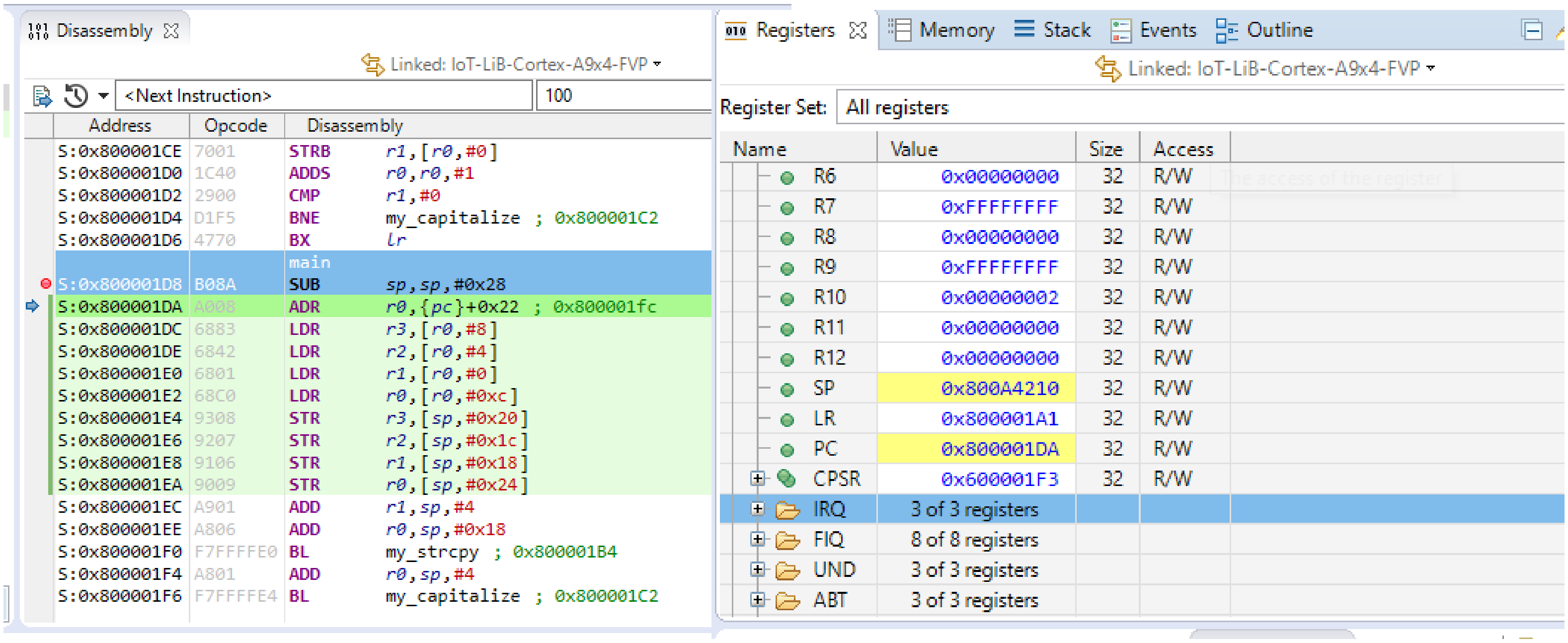
2. Open the Disassembly window (Window->Show View->Disassembly). Which instruction is highlighted, and what is its address? How does this address relate to the value of PC? (Insert screenshot of your disassembly window.) **(5 points)**



```
Address       Opcode  Disassembly
S:0x800001C8  297A    CMP     r1,#0x7a
S:0x800001CA  D801    BHI     my_capitalize+14  ; 0x800001D0
S:0x800001CC  3920    SUBS    r1,r1,#0x20
S:0x800001CE  7001    STRB    r1,[r0,#0]
S:0x800001D0  1C40    ADDS    r0,r0,#1
S:0x800001D2  2900    CMP     r1,#0
S:0x800001D4  D1F5    BNE     my_capitalize  ; 0x800001C2
S:0x800001D6  4770    BX      lr
                      main
S:0x800001D8  B08A    SUB     sp,sp,#0x28
S:0x800001DA  A008    ADR     r0,{pc}+0x22  ; 0x800001fc
S:0x800001DC  6883    LDR     r3,[r0,#8]
S:0x800001DE  6842    LDR     r2,[r0,#4]
S:0x800001E0  6801    LDR     r1,[r0,#0]
S:0x800001E2  68C0    LDR     r0,[r0,#0xc]
S:0x800001E4  9308    STR     r3,[sp,#0x20]
S:0x800001E6  9207    STR     r2,[sp,#0x1c]
S:0x800001E8  9106    STR     r1,[sp,#0x18]
S:0x800001EA  9009    STR     r0,[sp,#0x24]
S:0x800001EC  A901    ADD     r1,sp,#4
S:0x800001EE  A806    ADD     r0,sp,#0x18
```

The instruction that is highlighted is "SUB   sp,sp,#0x28". The address is S:0x800001D8. the address is related to the program counter as the PC always stores the address of the instruction that is currently being performed. This is evident in that the address in the PC is the same as the address in the disassembly.

3. Switch to "Step by Instruction" mode by clicking in the Debug Control window. Step one machine instruction using the F5 key while the Disassembly window is selected. Which two registers have changed (they should be highlighted in the Registers window), and how do they relate to the instruction just executed? (Insert the screenshot.) **(5 points)**


Disassembly
Linked: IoT-LiB-Cortex-A9x4-FVP
<Next Instruction>
100
Registers
Memory
Stack
Events
Outline
Linked: IoT-LiB-Cortex-A9x4-FVP
The access of the register

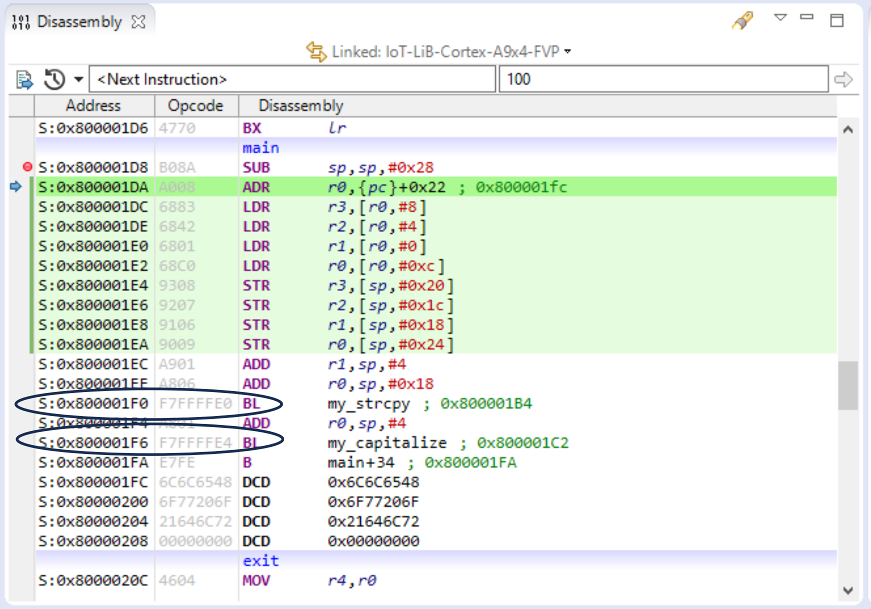
```
Address       Opcode    Disassembly
S:0x800001CE  7001      STRB    r1,[r0,#0]
S:0x800001D0  1C40      ADDS    r0,r0,#1
S:0x800001D2  2900      CMP     r1,#0
S:0x800001D4  D1F5      BNE     my_capitalize  ; 0x800001C2
S:0x800001D6  4770      BX      lr
                        main
S:0x800001D8  B08A      SUB     sp,sp,#0x28
S:0x800001DA  A008      ADR     r0,{pc}+0x22  ; 0x800001fc
S:0x800001DC  6883      LDR     r3,[r0,#8]
S:0x800001DE  6842      LDR     r2,[r0,#4]
S:0x800001E0  6801      LDR     r1,[r0,#0]
S:0x800001E2  68C0      LDR     r0,[r0,#0xc]
S:0x800001E4  9308      STR     r3,[sp,#0x20]
S:0x800001E6  9207      STR     r2,[sp,#0x1c]
S:0x800001E8  9106      STR     r1,[sp,#0x18]
S:0x800001EA  9009      STR     r0,[sp,#0x24]
S:0x800001EC  A901      ADD     r1,sp,#4
S:0x800001EE  A806      ADD     r0,sp,#0x18
S:0x800001F0  F7FFFFE0  BL      my_strcpy  ; 0x800001B4
S:0x800001F4  A801      ADD     r0,sp,#4
S:0x800001F6  F7FFFFE4  BL      my_capitalize  ; 0x800001C2
```

Register Set: All registers

Name	Value	Size	Access
R6	0x00000000	32	R/W
R7	0xFFFFFFFF	32	R/W
R8	0x00000000	32	R/W
R9	0xFFFFFFFF	32	R/W
R10	0x00000002	32	R/W
R11	0x00000000	32	R/W
R12	0x00000000	32	R/W
SP	0x800A4210	32	R/W
LR	0x800001A1	32	R/W
PC	0x800001DA	32	R/W
CPSR	0x600001F3	32	R/W
IRQ	3 of 3 registers		
FIQ	8 of 8 registers		
UND	3 of 3 registers		
ABT	3 of 3 registers		

The two registers that changed were the the SP and the PC. Knowing that the stack pointer acts as a place holder for the location of the top of the stack, it makes sense, the new value of the SP is 0x800A4210 as the previous execution of the instruction subtracted 0x28 from the previous value of 0x800A4238. Secondly, the PC changed to 0x800001DA as that is the address of the currently place in the routine when we jumped to the next instruction.

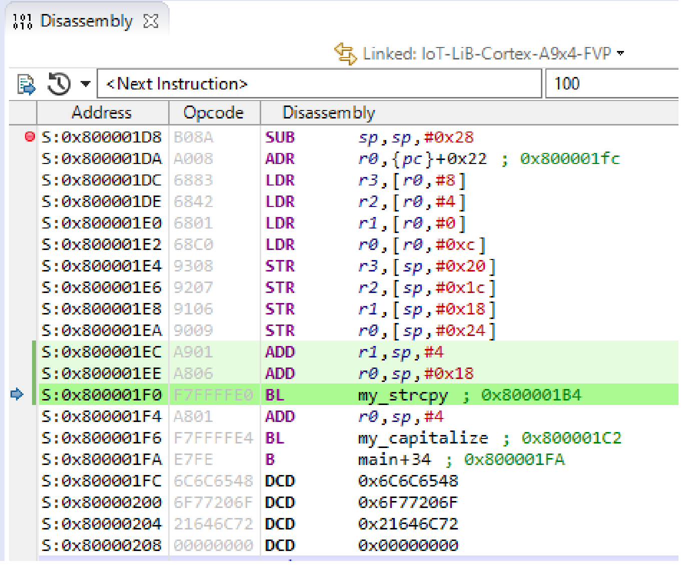
4. Look at the instructions in the Disassembly window. Do you see any instructions that are four bytes long? If so, what are the instructions? (Insert screenshot.) **(5 points)**



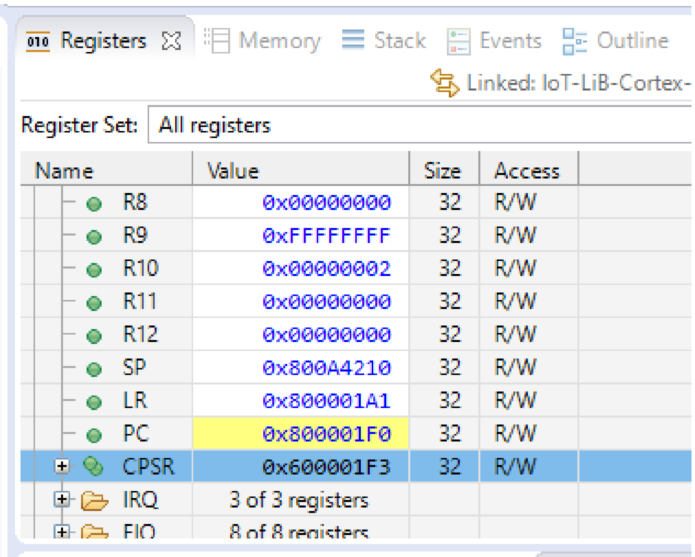
```
Address       Opcode    Disassembly
S:0x800001D6  4770      BX      lr
                        main
S:0x800001D8  B08A      SUB     sp,sp,#0x28
S:0x800001DA  A008      ADR     r0,{pc}+0x22  ; 0x800001fc
S:0x800001DC  6883      LDR     r3,[r0,#8]
S:0x800001DE  6842      LDR     r2,[r0,#4]
S:0x800001E0  6801      LDR     r1,[r0,#0]
S:0x800001E2  68C0      LDR     r0,[r0,#0xc]
S:0x800001E4  9308      STR     r3,[sp,#0x20]
S:0x800001E6  9207      STR     r2,[sp,#0x1c]
S:0x800001E8  9106      STR     r1,[sp,#0x18]
S:0x800001EA  9009      STR     r0,[sp,#0x24]
S:0x800001EC  A901      ADD     r1,sp,#4
S:0x800001EE  A806      ADD     r0,sp,#0x18
S:0x800001F0  F7FFFFE0  BL      my_strcpy  ; 0x800001B4
S:0x800001F4  A801      ADD     r0,sp,#4
S:0x800001F6  F7FFFFE4  BL      my_capitalize  ; 0x800001C2
S:0x800001FA  E7FE      B       main+34  ; 0x800001FA
S:0x800001FC  6C6C6548  DCD     0x6C6C6548
S:0x80000200  6F77206F  DCD     0x6F77206F
S:0x80000204  21646C72  DCD     0x21646C72
S:0x80000208  00000000  DCD     0x00000000
                        exit
S:0x8000020C  4604      MOV     r4,r0
```

Looking at the disassembly window, specifically, the “Opcode” column, we can see that two instructions my_strcpy and my_capitilize are 32 bits long. We know this because it is hex value where each hex character is 4 bits long, 8 characters equates to 32 bits.

5. Continue execution (using F5) until reaching the BL my_strcpy instruction. What are the values of the SP, PC, and LR? (Insert screenshot.) **(5 points)**



```
Address       Opcode    Disassembly
S:0x800001D8  B08A      SUB     sp,sp,#0x28
S:0x800001DA  A008      ADR     r0,{pc}+0x22  ; 0x800001fc
S:0x800001DC  6883      LDR     r3,[r0,#8]
S:0x800001DE  6842      LDR     r2,[r0,#4]
S:0x800001E0  6801      LDR     r1,[r0,#0]
S:0x800001E2  68C0      LDR     r0,[r0,#0xc]
S:0x800001E4  9308      STR     r3,[sp,#0x20]
S:0x800001E6  9207      STR     r2,[sp,#0x1c]
S:0x800001E8  9106      STR     r1,[sp,#0x18]
S:0x800001EA  9009      STR     r0,[sp,#0x24]
S:0x800001EC  A901      ADD     r1,sp,#4
S:0x800001EE  A806      ADD     r0,sp,#0x18
S:0x800001F0  F7FFFFE0  BL      my_strcpy  ; 0x800001B4
S:0x800001F4  A801      ADD     r0,sp,#4
S:0x800001F6  F7FFFFE4  BL      my_capitalize  ; 0x800001C2
S:0x800001FA  E7FE      B       main+34  ; 0x800001FA
S:0x800001FC  6C6C6548  DCD     0x6C6C6548
S:0x80000200  6F77206F  DCD     0x6F77206F
S:0x80000204  21646C72  DCD     0x21646C72
S:0x80000208  00000000  DCD     0x00000000
```



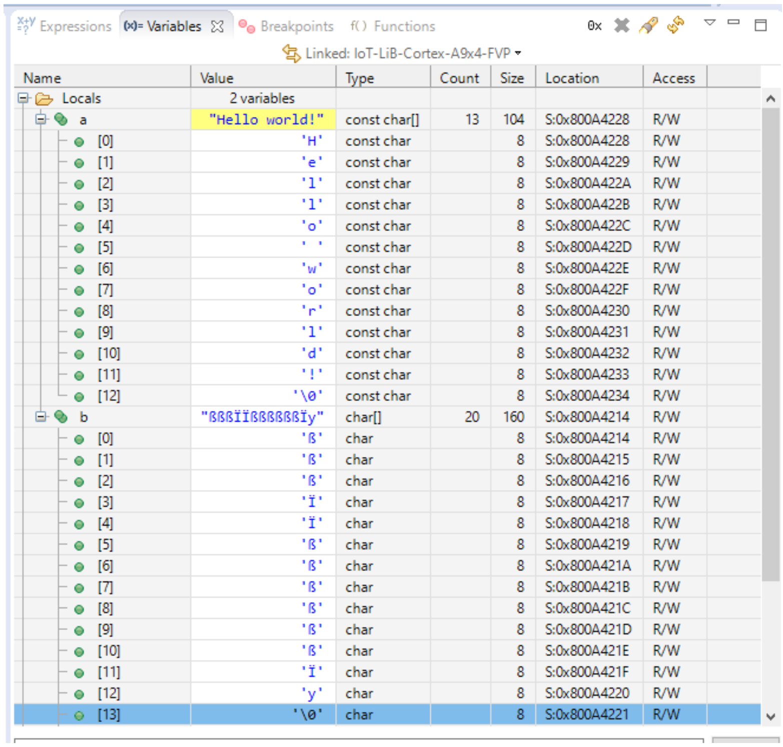
Register Set: All registers

Name	Value	Size	Access
R8	0x00000000	32	R/W
R9	0xFFFFFFFF	32	R/W
R10	0x00000002	32	R/W
R11	0x00000000	32	R/W
R12	0x00000000	32	R/W
SP	0x800A4210	32	R/W
LR	0x800001A1	32	R/W
PC	0x800001F0	32	R/W
CPSR	0x600001F3	32	R/W
IRQ	3 of 3 registers		
FIQ	8 of 8 registers		

Once the debug program reached BL my_strcpy the value of the SP is 0x800A4210, the value of the PC is 0x800001F0, and the value of the LR is 0x800001A1.

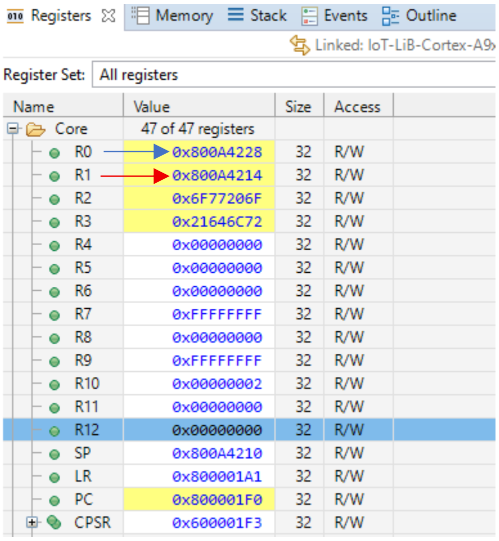
6. Watch the Variables window (Window->Show View-> Variables) to analyze the variables "a" and "b" (Insert screenshot). What is the value of "a"? What is the value of "b"? **(5 points)**

The value of "a" is the character string we asserted as "Hello World" Secondly, the value of "b" is variables is metadata as we have not assigned any particular value to that variable in the code "**char** b[20];"



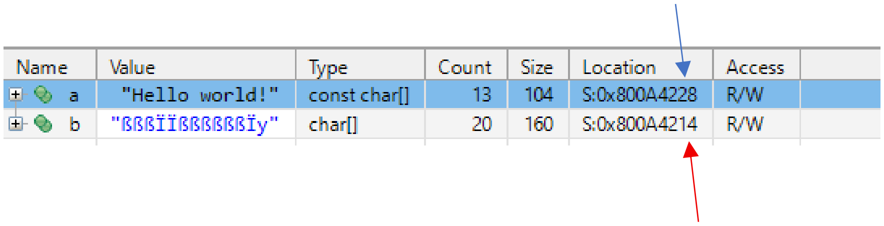
Name	Value	Type	Count	Size	Location	Access
Locals	2 variables					
a	"Hello world!"	const char[]	13	104	S:0x800A4228	R/W
[0]	'H'	const char		8	S:0x800A4228	R/W
[1]	'e'	const char		8	S:0x800A4229	R/W
[2]	'l'	const char		8	S:0x800A422A	R/W
[3]	'l'	const char		8	S:0x800A422B	R/W
[4]	'o'	const char		8	S:0x800A422C	R/W
[5]	' '	const char		8	S:0x800A422D	R/W
[6]	'w'	const char		8	S:0x800A422E	R/W
[7]	'o'	const char		8	S:0x800A422F	R/W
[8]	'r'	const char		8	S:0x800A4230	R/W
[9]	'l'	const char		8	S:0x800A4231	R/W
[10]	'd'	const char		8	S:0x800A4232	R/W
[11]	'!'	const char		8	S:0x800A4233	R/W
[12]	'\0'	const char		8	S:0x800A4234	R/W
b	"ßßßÏÏßßßßßßÏy"	char[]	20	160	S:0x800A4214	R/W
[0]	'ß'	char		8	S:0x800A4214	R/W
[1]	'ß'	char		8	S:0x800A4215	R/W
[2]	'ß'	char		8	S:0x800A4216	R/W
[3]	'Ï'	char		8	S:0x800A4217	R/W
[4]	'Ï'	char		8	S:0x800A4218	R/W
[5]	'ß'	char		8	S:0x800A4219	R/W
[6]	'ß'	char		8	S:0x800A421A	R/W
[7]	'ß'	char		8	S:0x800A421B	R/W
[8]	'ß'	char		8	S:0x800A421C	R/W
[9]	'ß'	char		8	S:0x800A421D	R/W
[10]	'ß'	char		8	S:0x800A421E	R/W
[11]	'Ï'	char		8	S:0x800A421F	R/W
[12]	'y'	char		8	S:0x800A4220	R/W
[13]	'\0'	char		8	S:0x800A4221	R/W

7. Which registers hold the arguments to my_strcpy, and what are their contents? (Insert screenshot.) **(5 points)**



Register Set: All registers

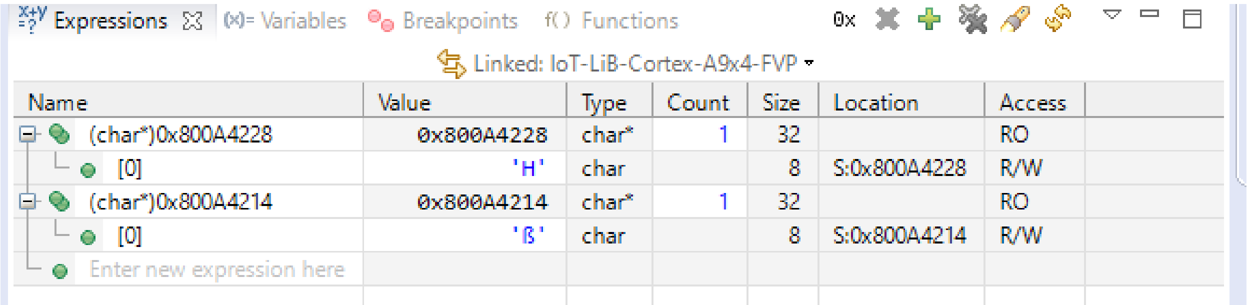
Name	Value	Size	Access
Core	47 of 47 registers		
R0	0x800A4228	32	R/W
R1	0x800A4214	32	R/W
R2	0x6F77206F	32	R/W
R3	0x21646C72	32	R/W
R4	0x00000000	32	R/W
R5	0x00000000	32	R/W
R6	0x00000000	32	R/W
R7	0xFFFFFFFF	32	R/W
R8	0x00000000	32	R/W
R9	0xFFFFFFFF	32	R/W
R10	0x00000002	32	R/W
R11	0x00000000	32	R/W
R12	0x00000000	32	R/W
SP	0x800A4210	32	R/W
LR	0x800001A1	32	R/W
PC	0x800001F0	32	R/W
CPSR	0x600001F3	32	R/W



Name	Value	Type	Count	Size	Location	Access
a	"Hello world!"	const char[]	13	104	S:0x800A4228	R/W
b	"ßßßÏÏßßßßßßÏy"	char[]	20	160	S:0x800A4214	R/W

The registers that hold the arguments to the arguments are R0 (source: 0x800A4228) and R1 (destination: 0x800A4214) We can confirm the address within the stack with that loaded into R0 and R1.

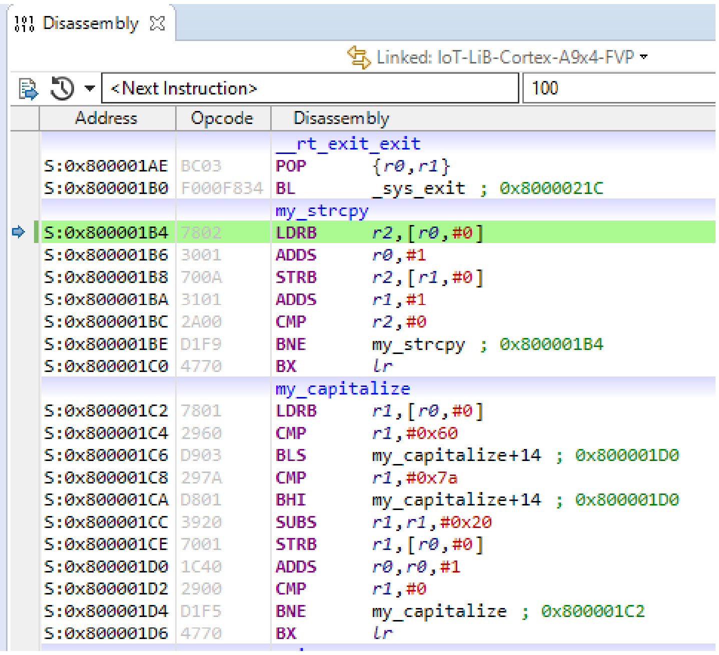
8. Use the Expressions window to watch the values in the address held in R0 and R1. Do the values match variables “a” and “b”? (Insert screenshot.) **(5 points)**



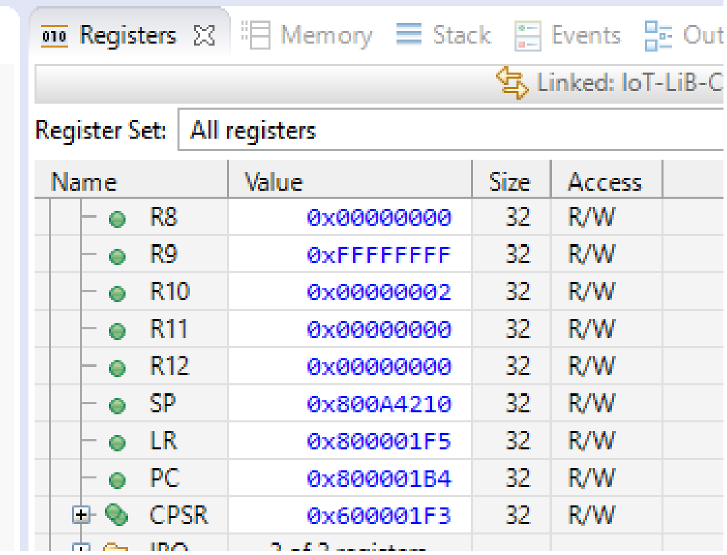
Name	Value	Type	Count	Size	Location	Access
(char*)0x800A4228	0x800A4228	char*	1	32		RO
[0]	'H'	char		8	S:0x800A4228	R/W
(char*)0x800A4214	0x800A4214	char*	1	32		RO
[0]	'ß'	char		8	S:0x800A4214	R/W
Enter new expression here						

Yes, they match.

9. Execute the BL instruction. What are the values of the SP, PC, and LR? What has changed and why? Does the PC value agree with what is shown in the Disassembly window? (Insert screenshot.) **(5 points)**



```
Address        Opcode    Disassembly
                         __rt_exit_exit
S:0x800001AE   BC03      POP       {r0,r1}
S:0x800001B0   F000F834  BL        _sys_exit ; 0x8000021C
                         my_strcpy
S:0x800001B4   7802      LDRB      r2,[r0,#0]
S:0x800001B6   3001      ADDS      r0,#1
S:0x800001B8   700A      STRB      r2,[r1,#0]
S:0x800001BA   3101      ADDS      r1,#1
S:0x800001BC   2A00      CMP       r2,#0
S:0x800001BE   D1F9      BNE       my_strcpy ; 0x800001B4
S:0x800001C0   4770      BX        lr
                         my_capitalize
S:0x800001C2   7801      LDRB      r1,[r0,#0]
S:0x800001C4   2960      CMP       r1,#0x60
S:0x800001C6   D903      BLS       my_capitalize+14 ; 0x800001D0
S:0x800001C8   297A      CMP       r1,#0x7a
S:0x800001CA   D801      BHI       my_capitalize+14 ; 0x800001D0
S:0x800001CC   3920      SUBS      r1,r1,#0x20
S:0x800001CE   7001      STRB      r1,[r0,#0]
S:0x800001D0   1C40      ADDS      r0,r0,#1
S:0x800001D2   2900      CMP       r1,#0
S:0x800001D4   D1F5      BNE       my_capitalize ; 0x800001C2
S:0x800001D6   4770      BX        lr
```

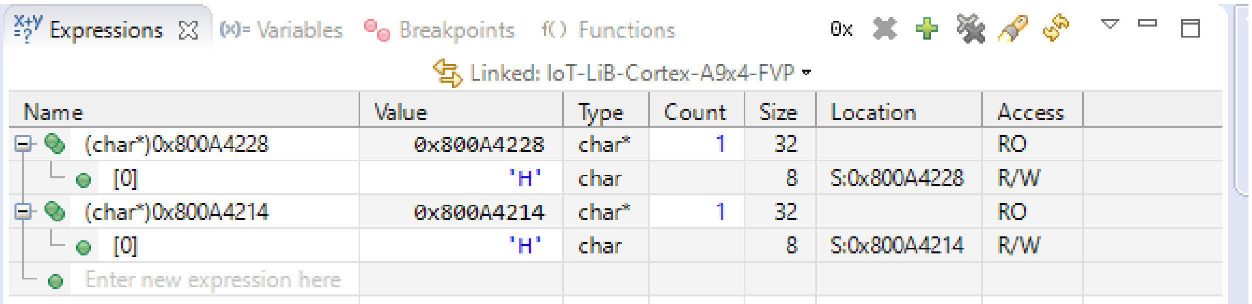


Register Set: All registers

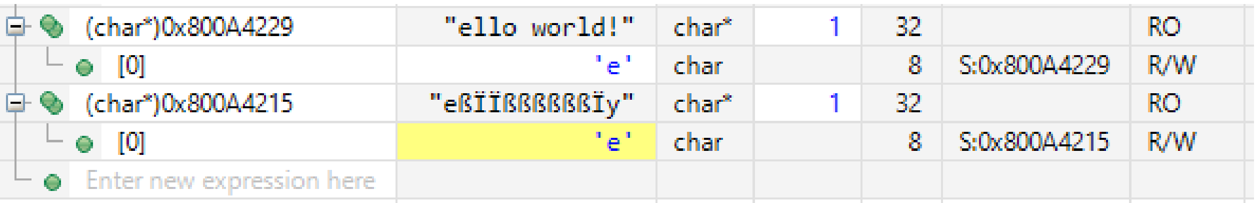
Name	Value	Size	Access
R8	0x00000000	32	R/W
R9	0xFFFFFFFF	32	R/W
R10	0x00000002	32	R/W
R11	0x00000000	32	R/W
R12	0x00000000	32	R/W
SP	0x800A4210	32	R/W
LR	0x800001F5	32	R/W
PC	0x800001B4	32	R/W
CPSR	0x600001F3	32	R/W

After the execution of the BL, The value of the SP is 0x800A4210, the value of the PC is 0x800001B4, and the value of the LR is 0x800001F5. Yes. The PC, agrees with the address in the disassembly window.

10. Single step through the assembly code watching the “Expressions” window to see the string being copied character by character from a to b. Which register holds the character? **(5 points)**



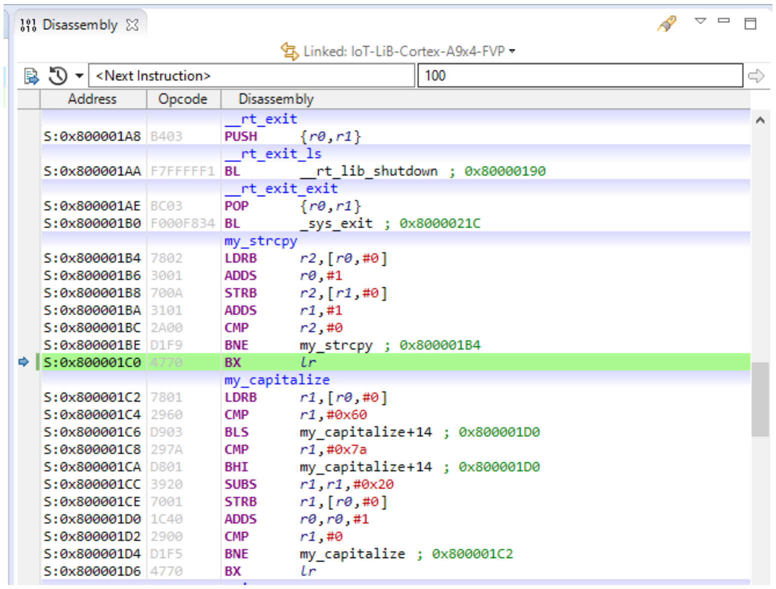
Name	Value	Type	Count	Size	Location	Access
(char*)0x800A4228	0x800A4228	char*	1	32		RO
[0]	'H'	char		8	S:0x800A4228	R/W
(char*)0x800A4214	0x800A4214	char*	1	32		RO
[0]	'H'	char		8	S:0x800A4214	R/W
Enter new expression here						



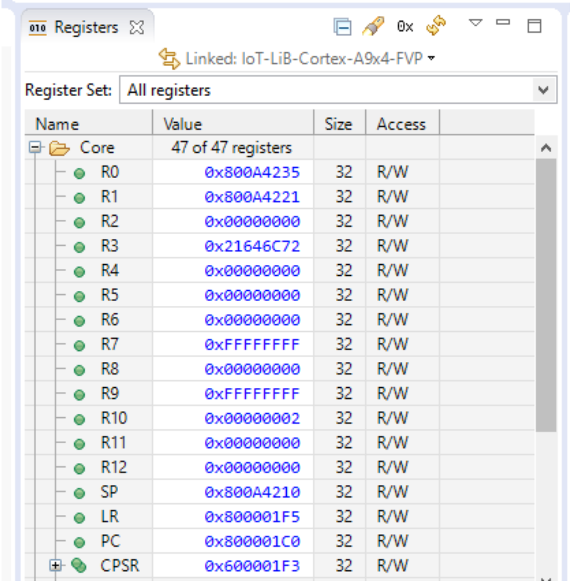
(char*)0x800A4229	"ello world!"	char*	1	32		RO
[0]	'e'	char		8	S:0x800A4229	R/W
(char*)0x800A4215	"eßÏÏßßßßßßÏy"	char*	1	32		RO
[0]	'e'	char		8	S:0x800A4215	R/W
Enter new expression here						

After two iterations, we can see that the letters are being copied correctly from a to b. The register that holds the character is the R2 register.

11. What are the values of the character, the src pointer, the dst pointer, the LR, and the PC when the code reaches the last instruction in the subroutine (BX lr)? (Insert screenshot.) **(5 points)**



```
Address       Opcode    Disassembly
                        __rt_exit
S:0x800001A8  B403      PUSH     {r0,r1}
                        __rt_exit_ls
S:0x800001AA  F7FFFFF1  BL       __rt_lib_shutdown  ; 0x80000190
                        __rt_exit_exit
S:0x800001AE  BC03      POP      {r0,r1}
S:0x800001B0  F000F834  BL       _sys_exit  ; 0x8000021C
                        my_strcpy
S:0x800001B4  7802      LDRB     r2,[r0,#0]
S:0x800001B6  3001      ADDS     r0,#1
S:0x800001B8  700A      STRB     r2,[r1,#0]
S:0x800001BA  3101      ADDS     r1,#1
S:0x800001BC  2A00      CMP      r2,#0
S:0x800001BE  D1F9      BNE      my_strcpy  ; 0x800001B4
S:0x800001C0  4770      BX       lr
                        my_capitalize
S:0x800001C2  7801      LDRB     r1,[r0,#0]
S:0x800001C4  2960      CMP      r1,#0x60
S:0x800001C6  D903      BLS      my_capitalize+14  ; 0x800001D0
S:0x800001C8  297A      CMP      r1,#0x7a
S:0x800001CA  D801      BHI      my_capitalize+14  ; 0x800001D0
S:0x800001CC  3920      SUBS     r1,r1,#0x20
S:0x800001CE  7001      STRB     r1,[r0,#0]
S:0x800001D0  1C40      ADDS     r0,r0,#1
S:0x800001D2  2900      CMP      r1,#0
S:0x800001D4  D1F5      BNE      my_capitalize  ; 0x800001C2
S:0x800001D6  4770      BX       lr
```

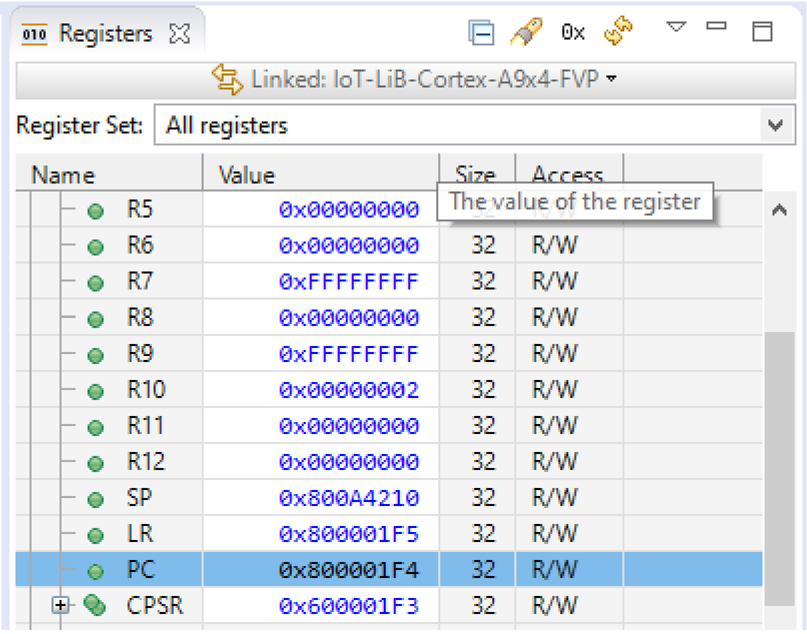


Register Set: All registers

Name	Value	Size	Access
Core	47 of 47 registers		
R0	0x800A4235	32	R/W
R1	0x800A4221	32	R/W
R2	0x00000000	32	R/W
R3	0x21646C72	32	R/W
R4	0x00000000	32	R/W
R5	0x00000000	32	R/W
R6	0x00000000	32	R/W
R7	0xFFFFFFFF	32	R/W
R8	0x00000000	32	R/W
R9	0xFFFFFFFF	32	R/W
R10	0x00000002	32	R/W
R11	0x00000000	32	R/W
R12	0x00000000	32	R/W
SP	0x800A4210	32	R/W
LR	0x800001F5	32	R/W
PC	0x800001C0	32	R/W
CPSR	0x600001F3	32	R/W

The value of the src pointer is 0x8004A225 and the value of the dst pointer 0x8004A221. The value of the LR is 0x800001F5, and the PC is 0x800001C0.

12. Execute the BX lr instruction. Now what is the value of the PC? **(5 points)**



Register Set: All registers

Name	Value	Size	Access
R5	0x00000000	32	R/W
R6	0x00000000	32	R/W
R7	0xFFFFFFFF	32	R/W
R8	0x00000000	32	R/W
R9	0xFFFFFFFF	32	R/W
R10	0x00000002	32	R/W
R11	0x00000000	32	R/W
R12	0x00000000	32	R/W
SP	0x800A4210	32	R/W
LR	0x800001F5	32	R/W
PC	0x800001F4	32	R/W
CPSR	0x600001F3	32	R/W

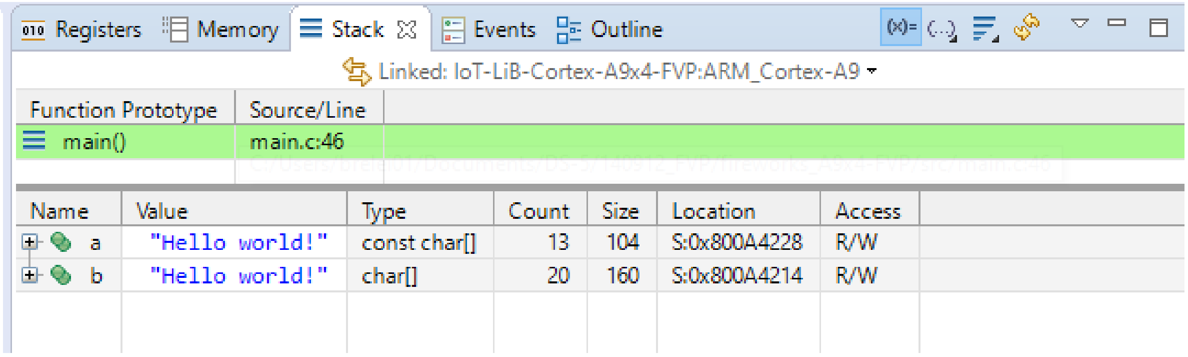
The value of the register

The value of the PC is the 0x800001F4.

13. What is the relationship between the PC value and the previous LR value? Explain. **(5 points)**

After execution, the PC value equals the previous LR value. The LR temporarily held the return address during the subroutine.

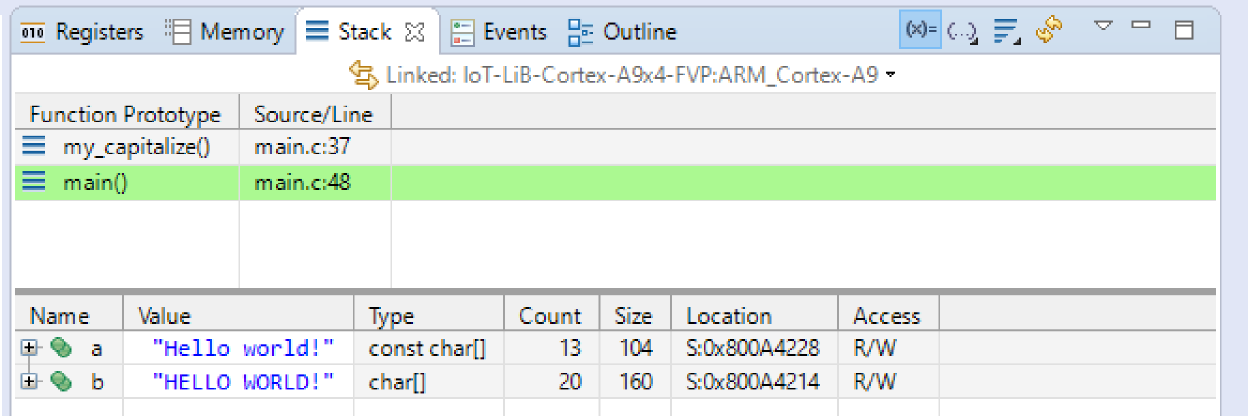
14. Now step through the my_capitalize subroutine and verify it works correctly, converting from "Hello world!" to "HELLO WORLD!". ((Insert final screenshot.) **(5 points)**



Function Prototype	Source/Line
main()	main.c:46

C:/Users/brelea01/Documents/DS-5/140912_FVP/fireworks_A9x4-FVP/src/main.c:46

Name	Value	Type	Count	Size	Location	Access
a	"Hello world!"	const char[]	13	104	S:0x800A4228	R/W
b	"Hello world!"	char[]	20	160	S:0x800A4214	R/W



Function Prototype	Source/Line
my_capitalize()	main.c:37
main()	main.c:48

Name	Value	Type	Count	Size	Location	Access
a	"Hello world!"	const char[]	13	104	S:0x800A4228	R/W
b	"HELLO WORLD!"	char[]	20	160	S:0x800A4214	R/W

15. Please explain your debugging experience in your own words **(10 points)**

The debugging experience provided real-world hands-on experience with the ARM architecture. The lab provided understanding of critical components within register of R0 – R4, Stack Pointer, Link Register, and the Program Counter. As an embedded engineer, it is important to understand how the software integrates with the hardware, and this is evident how the software arguments in "my_strcpy" are saved into the into the hardware stack of R0 and R1. Secondly, as the assembly code looped through the debug, we analyzed the increments of the program counter and understood the function of the PC as it relates to current location of the program execution. Thirdly, we analyzed the function of the stack pointer being the register that points to the top of the stack. In the case of assembly functions, we saw the stack pointer pointing to the top of the stack in memory as we looped through the code storing local variables. Finally, as the assembly jumped in and out of the custom functions, the link register showed the location of where the program needed to return to after the completion of the function. Overall, the lab provided critical step by step debugging practice which is critical when pinpointing where in the code a possible bug might be when troubleshooting. Understanding the registers, their functions and deterministic behaviors are paramount when designing embedded systems in the future.