ECE 255 Introduction To Computer Architecture

Lab 1: Using The Eclipse Integrated Development Environment

October 7 2025

Dexter Komen, V01088431

James Kriston, V00993475

Objectives

An introduction to the ARM architecture and assembly language program procedures. The objectives of this lab are:

- Run and debug programs using the Eclipse Integrated Development Environment
- Use single step execution and breakpoints to analyze programs
- Link and assemble ARM architecture assembly programs
- Load programs onto the ARM development board using Eclipse
- Monitor memory locations and registers to see the response of a program

Introduction

The programs that we assembled and ran for this experiment were obtained from the ECE 255 course website, and then imported into eclipse as pre-existing projects. There were three separate projects that we built, debugged, and ran for this experiment: lab1a, lab1as, and lab1bs. These programs were fairly simple, the first one just adding the contents of two registers (integer addition) and storing them into the contents of a third register. The second program was similar, but initialized variables x, y, and sum, each with a size of four bytes. The contents of x and y were loaded into registers, the sum of x and y was computed, and stored in another register. The third program we analyzed had the same principle, but with arrays. A loop was employed to add every element of the x array with the y array, and be stored into a sum array.

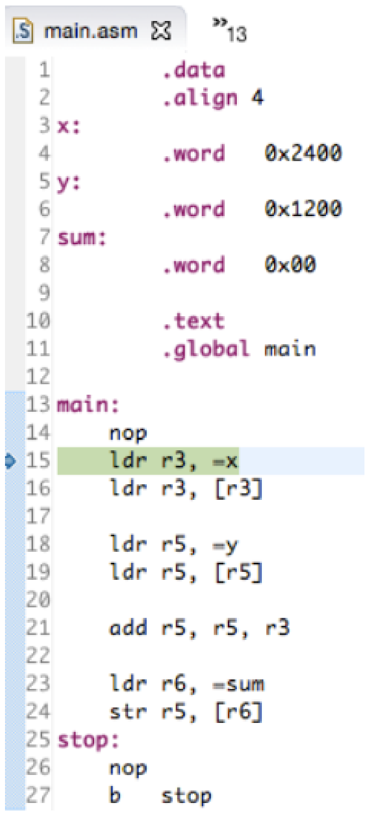
Discussion

When we executed the first program, register 4 was storing the value 5, and register 5 was storing the value 4. After execution, the contents of register 0 contained the value 9 (the sum of 4 and 5). We changed the operation from add to subtract, and this changed the contents of register 0 after execution of the program. Register 0 contained the integer value 1 (5 - 4, or R4 - R5). This is what we were anticipating before making this change to the program, and we could see this result in the contents of the registers.

```
        . text
        . global  main
main:
        nop
        mov     r4, #5
        mov     r5, #4
        add     r0, r4, r5
stop:
        nop
        b       stop
```

Figure 1: Program 1

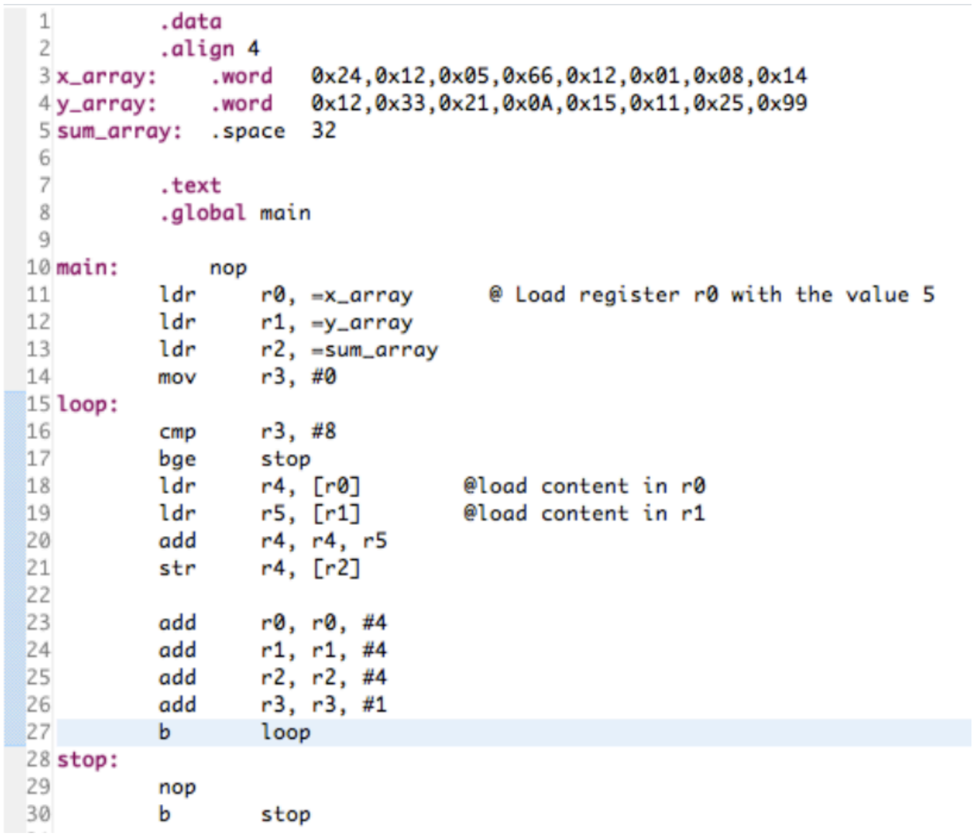
The second program as mentioned added two integer values and stored them in a register, as shown in Figure 2. However, because we were loading a register with the contents of a variable, initialized at a specific memory location, we were able to view those locations in the memory and analyze the content while executing the program step by step. For example, the sum variable was at memory location 0x00, so we were able to see the contents of that location change from 0 to 0x3600 (the sum of x (0x2400) and y(0x1200)). Also, we observed the instructions being transferred into the memory. This program was simple in nature, but the experiment shows clearly how the registers and memory of the ARM architecture respond to a program.



```
        .data
        .align 4
x:
        .word   0x2400
y:
        .word   0x1200
sum:
        .word   0x00

        .text
        .global main

main:
    nop
    ldr r3, =x
    ldr r3, [r3]

    ldr r5, =y
    ldr r5, [r5]

    add r5, r5, r3

    ldr r6, =sum
    str r5, [r6]
stop:
    nop
    b   stop
```

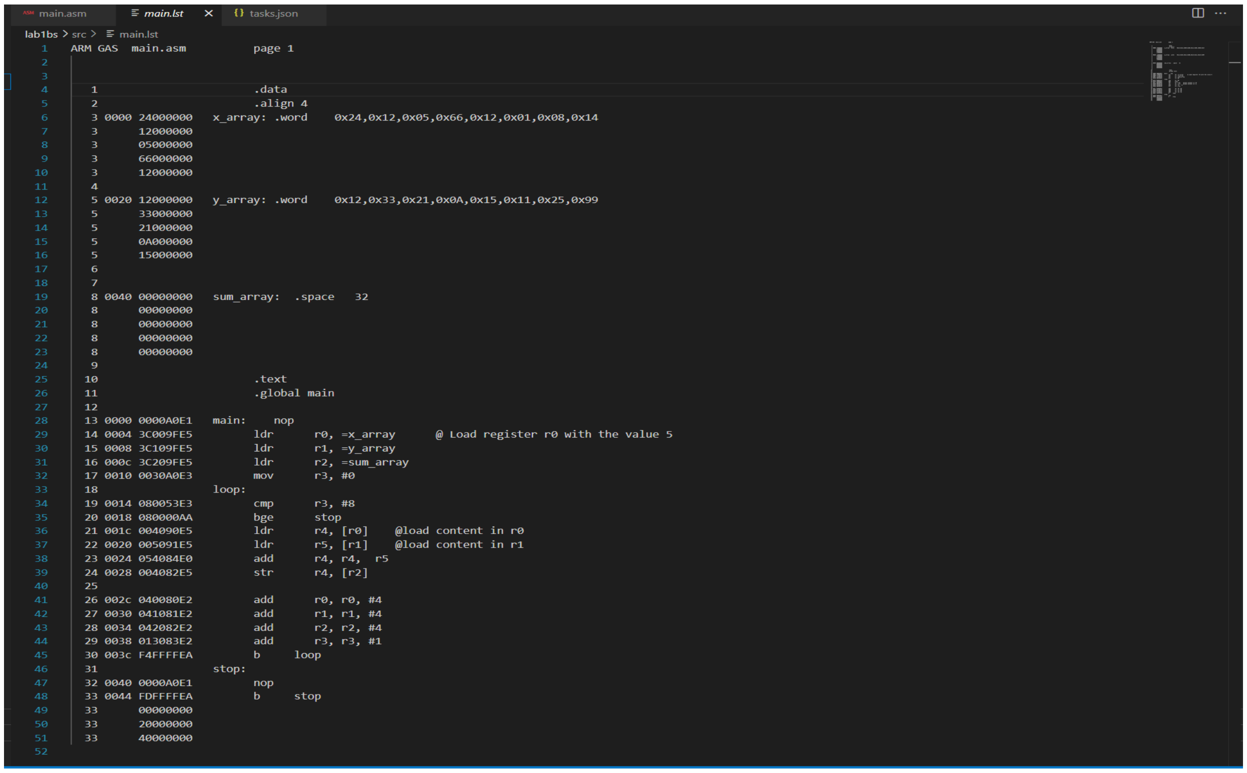
Figure 2: Program 2

The third and final program (as mentioned) initialized three arrays (x_array, y_array, and sum_array). The assembly code for this program is shown in Figure 3. The x and y arrays had 8 values each, and were defined as a word, so each element in the arrays had four bytes of storage. The sum array was not initialized, but space was accounted for in the memory. First this program would store the memory location of the beginning of x_array into register 0, the memory location of y_array into register 1, and sum_array into register 2. These instructions are illustrated in Figure 4. Also, the integer value 0 was moved into register 3. Next in the program, a loop was employed to iterate addition through both arrays. First we check if the contents of register 3 are greater than or equal to the integer value 8, and if it is, we break out of the loop. Register 3 holds our loop index for this program, we add the value 1 each time the loop is complete to keep track. The number 8 is used as comparison because each array has 8 elements, so we only want to execute this loop 8 times. Next this program will load the contents from the memory addresses stored in R0 and R1 into R4 and R5 respectively. Note that R0 and R1 currently contain the memory addresses of the first element in x_array and y_array. So now we have the first element of each array stored in two registers, we add them together, and store this value at memory location held in register 2 (the start of the sum array). Next this program iterates the values of R0, R1, and R2 by four (advance by 4 bytes due to 32bit word), so that they now contain the memory addresses of the next element in each array. Our loop index is iterated by 1, and the process repeats 8 times. After execution of this program, the sum array is now full of values that are the sum of x and y, as shown in Figure 5. Program listing is shown in Figure 4.



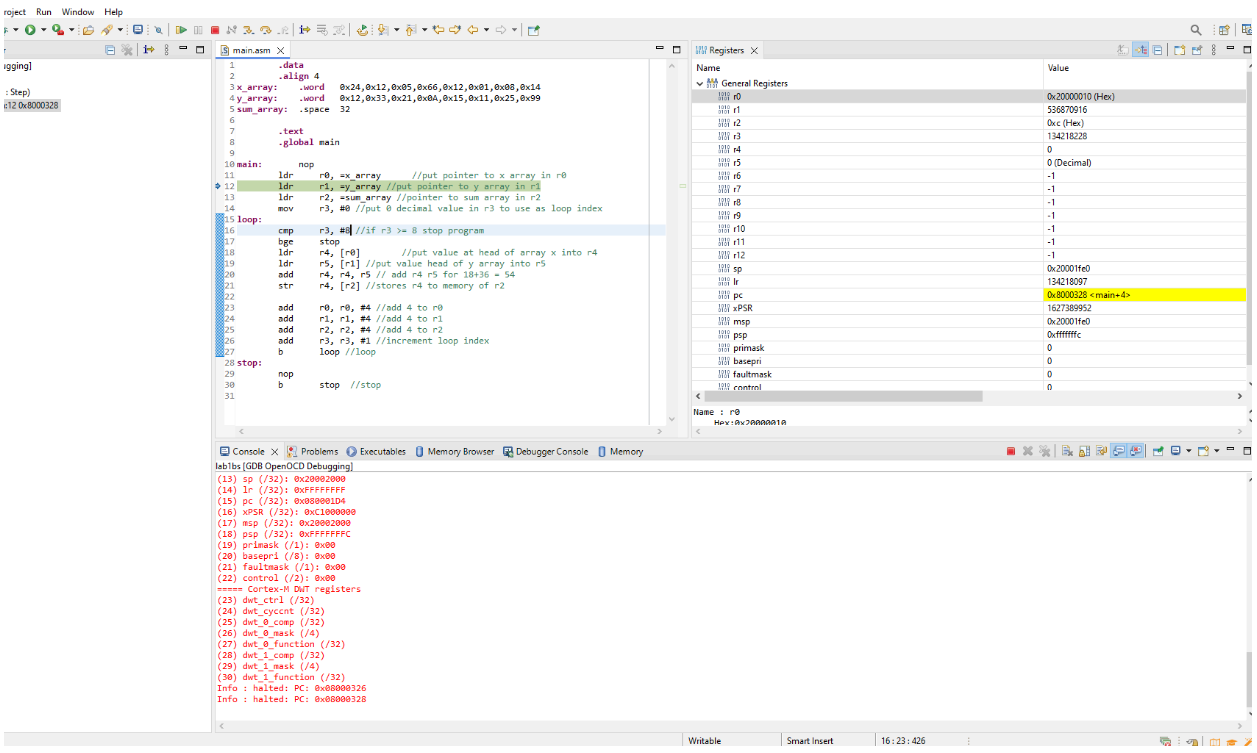
```
        .data
        .align 4
x_array:    .word   0x24,0x12,0x05,0x66,0x12,0x01,0x08,0x14
y_array:    .word   0x12,0x33,0x21,0x0A,0x15,0x11,0x25,0x99
sum_array:  .space  32

        .text
        .global main

main:       nop
        ldr     r0, =x_array      @ Load register r0 with the value 5
        ldr     r1, =y_array
        ldr     r2, =sum_array
        mov     r3, #0
loop:
        cmp     r3, #8
        bge     stop
        ldr     r4, [r0]        @load content in r0
        ldr     r5, [r1]        @load content in r1
        add     r4, r4, r5
        str     r4, [r2]

        add     r0, r0, #4
        add     r1, r1, #4
        add     r2, r2, #4
        add     r3, r3, #1
        b       loop
stop:
        nop
        b       stop
```

Figure 3: Program 3



```
ARM GAS  main.asm          page 1


   1                        .data
   2                        .align 4
   3 0000 24000000   x_array: .word    0x24,0x12,0x05,0x66,0x12,0x01,0x08,0x14
   3      12000000
   3      05000000
   3      66000000
   3      12000000
   4
   5 0020 12000000   y_array: .word    0x12,0x33,0x21,0x0A,0x15,0x11,0x25,0x99
   5      33000000
   5      21000000
   5      0A000000
   5      15000000
   6
   7
   8 0040 00000000   sum_array:  .space   32
   8      00000000
   8      00000000
   8      00000000
   8      00000000
   9
  10                        .text
  11                        .global main
  12
  13 0000 0000A0E1   main:    nop
  14 0004 3C009FE5          ldr      r0, =x_array      @ Load register r0 with the value 5
  15 0008 3C109FE5          ldr      r1, =y_array
  16 000c 3C209FE5          ldr      r2, =sum_array
  17 0010 0030A0E3          mov      r3, #0
  18                 loop:
  19 0014 080053E3          cmp      r3, #8
  20 0018 080000AA          bge      stop
  21 001c 004090E5          ldr      r4, [r0]    @load content in r0
  22 0020 005091E5          ldr      r5, [r1]    @load content in r1
  23 0024 054084E0          add      r4, r4,  r5
  24 0028 004082E5          str      r4, [r2]
  25
  26 002c 040080E2          add      r0, r0, #4
  27 0030 041081E2          add      r1, r1, #4
  28 0034 042082E2          add      r2, r2, #4
  29 0038 013083E2          add      r3, r3, #1
  30 003c F4FFFFEA          b     loop
  31                 stop:
  32 0040 0000A0E1          nop
  33 0044 FDFFFFEA          b     stop
  33      00000000
  33      20000000
  33      40000000
```

Figure 4: Program Listing



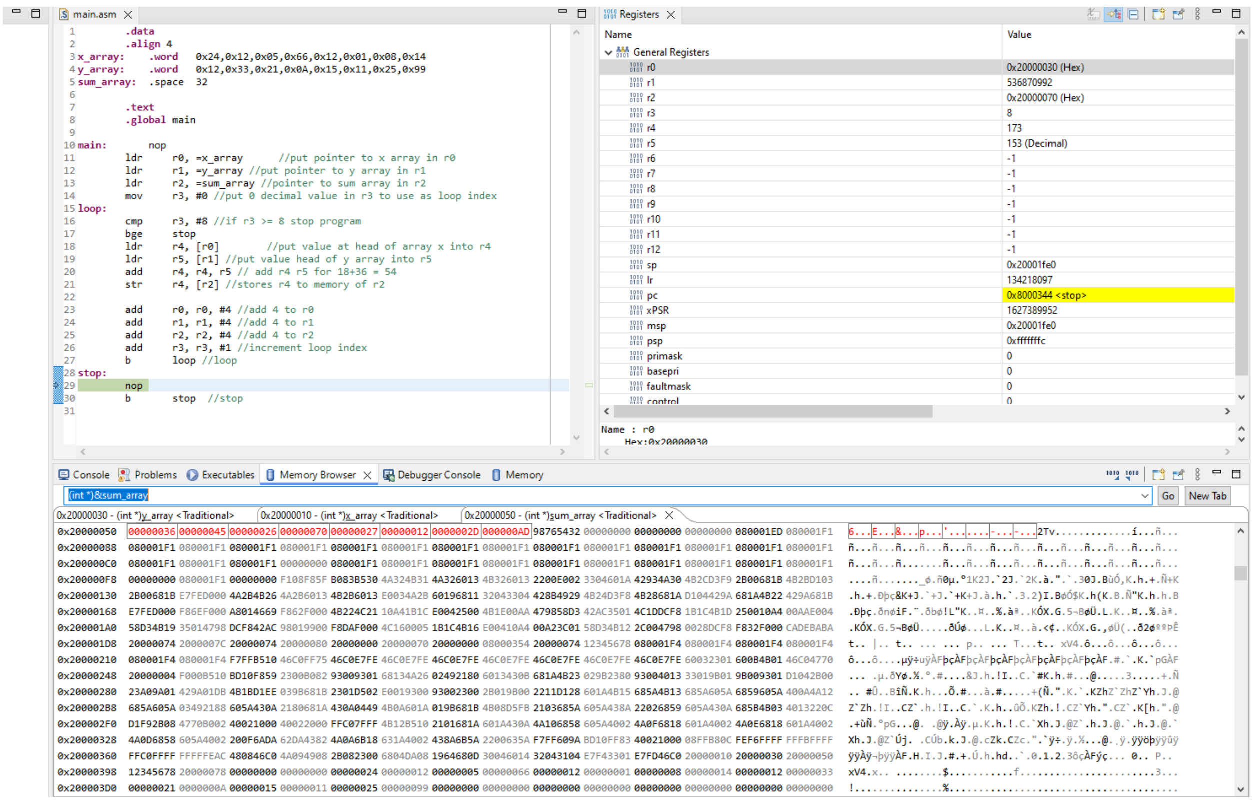
```
        .data
        .align 4
x_array:    .word   0x24,0x12,0x05,0x66,0x12,0x01,0x08,0x14
y_array:    .word   0x12,0x33,0x21,0x0A,0x15,0x11,0x25,0x99
sum_array:  .space  32

        .text
        .global main

main:       nop
        ldr     r0, =x_array      //put pointer to x array in r0
        ldr     r1, =y_array //put pointer to y array in r1
        ldr     r2, =sum_array //pointer to sum array in r2
        mov     r3, #0 //put 0 decimal value in r3 to use as loop index
loop:
        cmp     r3, #8 //if r3 >= 8 stop program
        bge     stop
        ldr     r4, [r0]        //put value at head of array x into r4
        ldr     r5, [r1] //put value head of y array into r5
        add     r4, r4, r5 // add r4 r5 for 18+36 = 54
        str     r4, [r2] //stores r4 to memory of r2

        add     r0, r0, #4 //add 4 to r0
        add     r1, r1, #4 //add 4 to r1
        add     r2, r2, #4 //add 4 to r2
        add     r3, r3, #1 //increment loop index
        b       loop //loop
stop:
        nop
        b       stop  //stop
```

Name	Value
General Registers	
r0	0x20000010 (Hex)
r1	536870916
r2	0xc (Hex)
r3	134218228
r4	0
r5	0 (Decimal)
r6	-1
r7	-1
r8	-1
r9	-1
r10	-1
r11	-1
r12	-1
sp	0x20001fe0
lr	134218097
pc	0x8000328 <main+4>
xPSR	1627389952
msp	0x20001fe0
psp	0xfffffffc
primask	0
basepri	0
faultmask	0
control	0

```
Name : r0
    Hex:0x20000010
```

```
lab1bs [GDB OpenOCD Debugging]
(13) sp (/32): 0x20002000
(14) lr (/32): 0xFFFFFFFF
(15) pc (/32): 0x080001D4
(16) xPSR (/32): 0xC1000000
(17) msp (/32): 0x20002000
(18) psp (/32): 0xFFFFFFFC
(19) primask (/1): 0x00
(20) basepri (/8): 0x00
(21) faultmask (/1): 0x00
(22) control (/2): 0x00
===== Cortex-M DWT registers
(23) dwt_ctrl (/32)
(24) dwt_cyccnt (/32)
(25) dwt_0_comp (/32)
(26) dwt_0_mask (/4)
(27) dwt_0_function (/32)
(28) dwt_1_comp (/32)
(29) dwt_1_mask (/4)
(30) dwt_1_function (/32)
Info : halted: PC: 0x08000326
Info : halted: PC: 0x08000328
```

Figure 5: Register Contents



```
main.asm

        .data
        .align 4
x_array:    .word   0x24,0x12,0x05,0x66,0x12,0x01,0x08,0x14
y_array:    .word   0x12,0x33,0x21,0x0A,0x15,0x11,0x25,0x99
sum_array:  .space  32

        .text
        .global main

main:       nop
        ldr     r0, =x_array      //put pointer to x array in r0
        ldr     r1, =y_array //put pointer to y array in r1
        ldr     r2, =sum_array //pointer to sum array in r2
        mov     r3, #0 //put 0 decimal value in r3 to use as loop index
loop:
        cmp     r3, #8 //if r3 >= 8 stop program
        bge     stop
        ldr     r4, [r0]        //put value at head of array x into r4
        ldr     r5, [r1] //put value head of y array into r5
        add     r4, r4, r5 // add r4 r5 for 18+36 = 54
        str     r4, [r2] //stores r4 to memory of r2

        add     r0, r0, #4 //add 4 to r0
        add     r1, r1, #4 //add 4 to r1
        add     r2, r2, #4 //add 4 to r2
        add     r3, r3, #1 //increment loop index
        b       loop //loop
stop:
        nop
        b       stop  //stop
```

Registers

Name	Value
General Registers	
r0	0x20000030 (Hex)
r1	536870992
r2	0x20000070 (Hex)
r3	8
r4	173
r5	153 (Decimal)
r6	-1
r7	-1
r8	-1
r9	-1
r10	-1
r11	-1
r12	-1
sp	0x20001fe0
lr	134218097
pc	0x8000344 <stop>
xPSR	1627389952
msp	0x20001fe0
psp	0xfffffffc
primask	0
basepri	0
faultmask	0
control	0

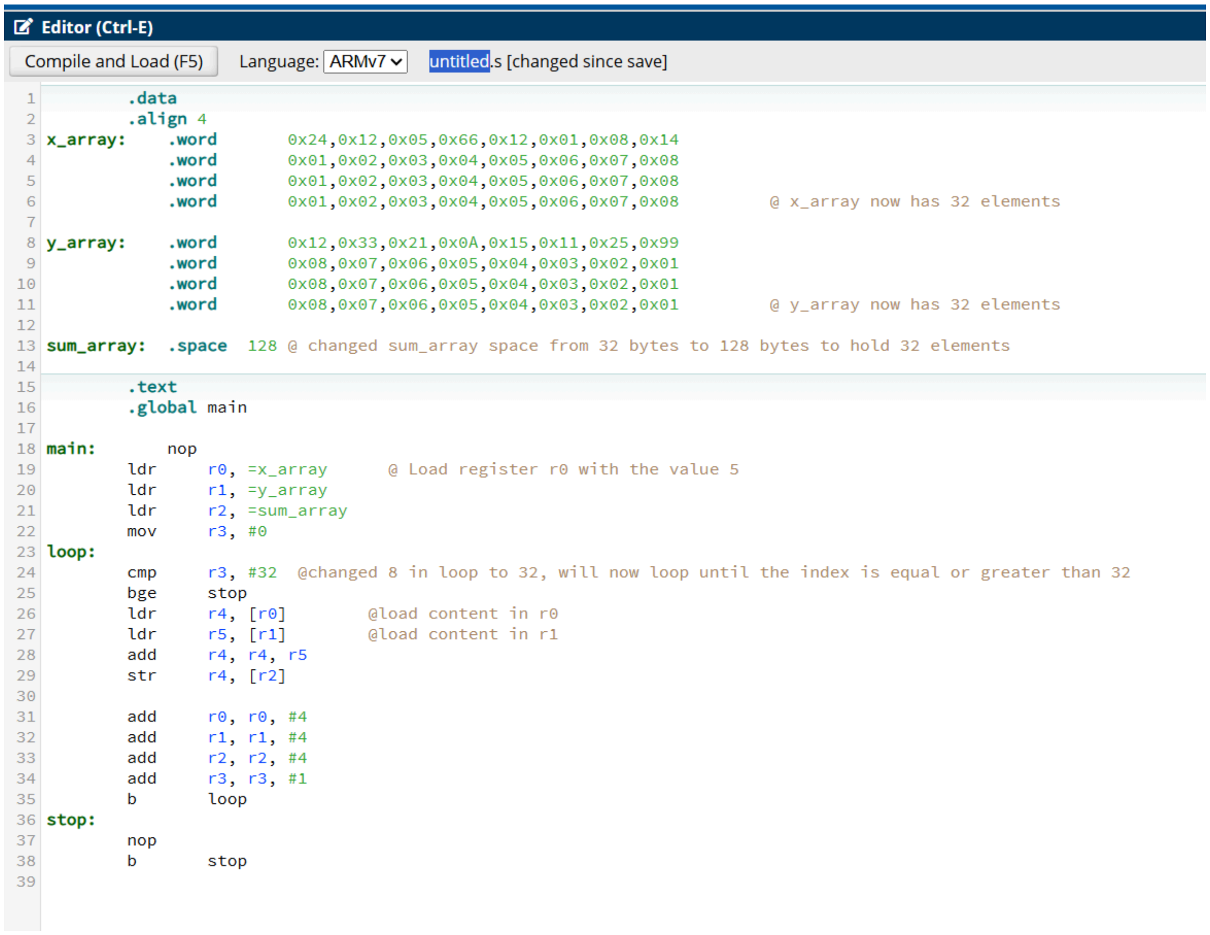
```
Name : r0
    Hex:0x20000030
```

```
(int *)&sum_array

0x20000030 - (int *)y_array <Traditional>   0x20000010 - (int *)x_array <Traditional>   0x20000050 - (int *)sum_array <Traditional>

0x20000050  00000036 00000045 00000026 00000070 00000027 00000012 0000002D 000000AD 98765432 00000000 00000000 00000000 080001ED 080001F1  6...E...&...p...'.......-...-...2Tv.............í...ñ...
0x20000088  080001F1 080001F1 080001F1 080001F1 080001F1 080001F1 080001F1 080001F1 080001F1 080001F1 080001F1 080001F1 080001F1 080001F1  ñ...ñ...ñ...ñ...ñ...ñ...ñ...ñ...ñ...ñ...ñ...ñ...ñ...ñ...
0x200000C0  080001F1 080001F1 080001F1 00000000 080001F1 080001F1 080001F1 080001F1 080001F1 080001F1 080001F1 080001F1 080001F1 080001F1  ñ...ñ...ñ.......ñ...ñ...ñ...ñ...ñ...ñ...ñ...ñ...ñ...ñ...
0x200000F8  00000000 080001F1 00000000 F108F85F B083B530 4A324B31 4A326013 4B326013 2200E002 3304601A 42934A30 4B2CD3F9 2B00681B 4B2BD103  ....ñ......._ø.ñ0µ.°1K2J.`2J.`2K.à.".`.30J.BùÓ,K.h.+.Ñ+K
0x20000130  2B00681B E7FED000 4A2B4B26 4A2B6013 4B2B6013 E0034A2B 60196811 32043304 428B4929 4B24D3F8 4B28681A D104429A 681A4B22 429A681B  .h.+.Ðþç&K+J.`+J.`+K+J.à.h.`.3.2)I.BøÓ$K.h(K.B.Ñ"K.h.h.B
0x20000168  E7FED000 F86EF000 A8014669 F862F000 4B224C21 10A41B1C E0042500 4B1E00AA 479858D3 42AC3501 4C1DDCF8 1B1C4B1D 250010A4 00AAE004  .Ðþç.ðnøiF.¨.ðbø!L"K..¤..%.àª..KÓX.G.5¬BøÜ.L.K..¤..%.àª.
0x200001A0  58D34B19 35014798 DCF842AC 98019900 F8DAF000 4C160005 1B1C4B16 E00410A4 00A23C01 58D34B12 2C004798 0028DCF8 F832F000 CADEBABA  .KÓX.G.5¬BøÜ.....ðÚø...L.K..¤..à.<¢..KÓX.G.,øÜ(..ð2øººÞÊ
0x200001D8  20000074 2000007C 20000074 20000080 20000000 20000070 20000000 08000354 20000074 12345678 080001F4 080001F4 080001F4 080001F4  t.. |.. t.. ... ... p.. ... T...t.. xV4.ô...ô...ô...ô...
0x20000210  080001F4 080001F4 F7FFB510 46C0FF75 46C0E7FE 46C0E7FE 46C0E7FE 46C0E7FE 46C0E7FE 46C0E7FE 46C0E7FE 60032301 600B4B01 46C04770  ô...ô.....µÿ÷uÿÀFþçÀFþçÀFþçÀFþçÀFþçÀFþçÀF.#.`.K.`pGÀF
0x20000248  20000004 F000B510 BD10F859 2300B082 93009301 68134A26 02492180 6013430B 681A4B23 029B2380 93004013 33019B01 9B009301 D1042B00  ... .µ.ðYø.½.°.#....&J.h.!I..C.`#K.h.#...@.....3.....+.Ñ
0x20000280  23A09A01 429A01DB 4B1BD1EE 039B681B 2301D502 E0019300 93002300 2B019B00 2211D128 601A4B15 685A4B13 685A605A 6859605A 400A4A12  .. #Û..BîÑ.K.h...Õ.#...à.#.....+(Ñ.".K.`.KZhZ`ZhZ`Yh.J.@
0x200002B8  685A605A 03492188 605A430A 2180681A 430A0449 4B0A601A 019B681B 4B08D5FB 2103685A 605A438A 22026859 605A430A 685B4B03 4013220C  Z`Zh.!I..CZ`.h.!I..C.`.K.h..ûÕ.KZh.!.CZ`Yh.".CZ`.K[h.".@
0x200002F0  D1F92B08 4770B002 40021000 40022000 FFC07FFF 4B12B510 2101681A 601A430A 4A106858 605A4002 4A0F6818 601A4002 4A0E6818 601A4002  .+ùÑ.°pG...@. .@ÿ.Àÿ.µ.K.h.!.C.`Xh.J.@Z`.h.J.@.`.h.J.@.`
0x20000328  4A0D6858 605A4002 200F6ADA 62DA4382 4A0A6B18 631A4002 438A6B5A 2200635A F7FF609A BD10FF83 40021000 08FFB80C FEF6FFFF FFFBFFFF  Xh.J.@Z`Új. .CÚb.k.J.@.cZk.CZc.".`ÿ÷.ÿ.½...@.,ÿ.ÿÿöþÿÿûÿ
0x20000360  FFC0FFFF FFFFFEAC 480846C0 4A094908 2B082300 6804DA08 1964680D 30046014 32043104 E7F43301 E7FD46C0 20000010 20000030 20000050  ÿÿÀÿ¬þÿÿÀF.H.I.J.#.+.Ú.h.hd..`.0.1.2.3ôçÀFýç... 0.. P..
0x20000398  12345678 20000078 00000000 00000000 00000024 00000012 00000005 00000066 00000012 00000001 00000008 00000014 00000012 00000033  xV4.x.. ........$...........f.......................3...
0x200003D0  00000021 0000000A 00000015 00000011 00000025 00000099 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000  !...............%.......................................
```

Figure 6: Memory Contents

Limitations in the Code

From Figure. 3 and the relevant discussion behind the functionality of the code, it is clear that that version can only operate on 8 elements of an array. Once the loop index reaches 8, the program reaches the stop point and no more operations are performed. In terms of scalability this program with the current code cannot be scaled to do any operations on the arrays beyond 8 elements. Fortunately the changes one would need to make in order to scale this program beyond the 8 elements (perhaps to 32 as proposed in the lab manual) are very small changes, which can be seen in Figure 6. If you had an array with 32 elements you would only need to change the compare expression, the size of the sum array, and of course make sure to give your x and y arrays 32 elements. Simply change the number 8 in the cmp statement to 32, accordingly set the sum array to 32*4 = 128 bytes, and set both x_array and y_array to have 32 elements. These changes are shown below in Figure 7.



```
        .data
        .align 4
x_array:    .word       0x24,0x12,0x05,0x66,0x12,0x01,0x08,0x14
            .word       0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08
            .word       0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08
            .word       0x01,0x02,0x03,0x04,0x05,0x06,0x07,0x08         @ x_array now has 32 elements

y_array:    .word       0x12,0x33,0x21,0x0A,0x15,0x11,0x25,0x99
            .word       0x08,0x07,0x06,0x05,0x04,0x03,0x02,0x01
            .word       0x08,0x07,0x06,0x05,0x04,0x03,0x02,0x01
            .word       0x08,0x07,0x06,0x05,0x04,0x03,0x02,0x01         @ y_array now has 32 elements

sum_array:  .space  128 @ changed sum_array space from 32 bytes to 128 bytes to hold 32 elements

        .text
        .global main

main:       nop
        ldr     r0, =x_array      @ Load register r0 with the value 5
        ldr     r1, =y_array
        ldr     r2, =sum_array
        mov     r3, #0
loop:
        cmp     r3, #32  @changed 8 in loop to 32, will now loop until the index is equal or greater than 32
        bge     stop
        ldr     r4, [r0]        @load content in r0
        ldr     r5, [r1]        @load content in r1
        add     r4, r4, r5
        str     r4, [r2]

        add     r0, r0, #4
        add     r1, r1, #4
        add     r2, r2, #4
        add     r3, r3, #1
        b       loop
stop:
        nop
        b       stop

```

Figure 7: Program 3 Modified for 32 Elements

Conclusion

This lab demonstrated the understanding of cross-assemblers and the usage of the Eclipse IDE debugging and memory monitoring tools. Assembly code was translated by Eclipse IDE into machine code for the STM32F051R8T6 microcontroller. Single step execution and breakpoints were used alongside the memory and registry monitors to analyze the execution and response of the program. By stepping into operations one at a time the program's loading of values from memory into the registry were able to be recorded. Then the addition of these values within the loop was able to be tracked one at a time. Finally the storage of these values into a different location in the memory was seen. These lab exercises displayed the foundational principles behind embedded systems.