Advanced Computer Organization and Architecture with Lab

Lab Assignment 6 – Subroutines

Submitted by – Sanjana Jammi (CS18M522)

Screenshots:

Part 1 - Find the position of an element in an unsorted array

Algorithm used - Linear search through all array elements. Complexity is O(n)

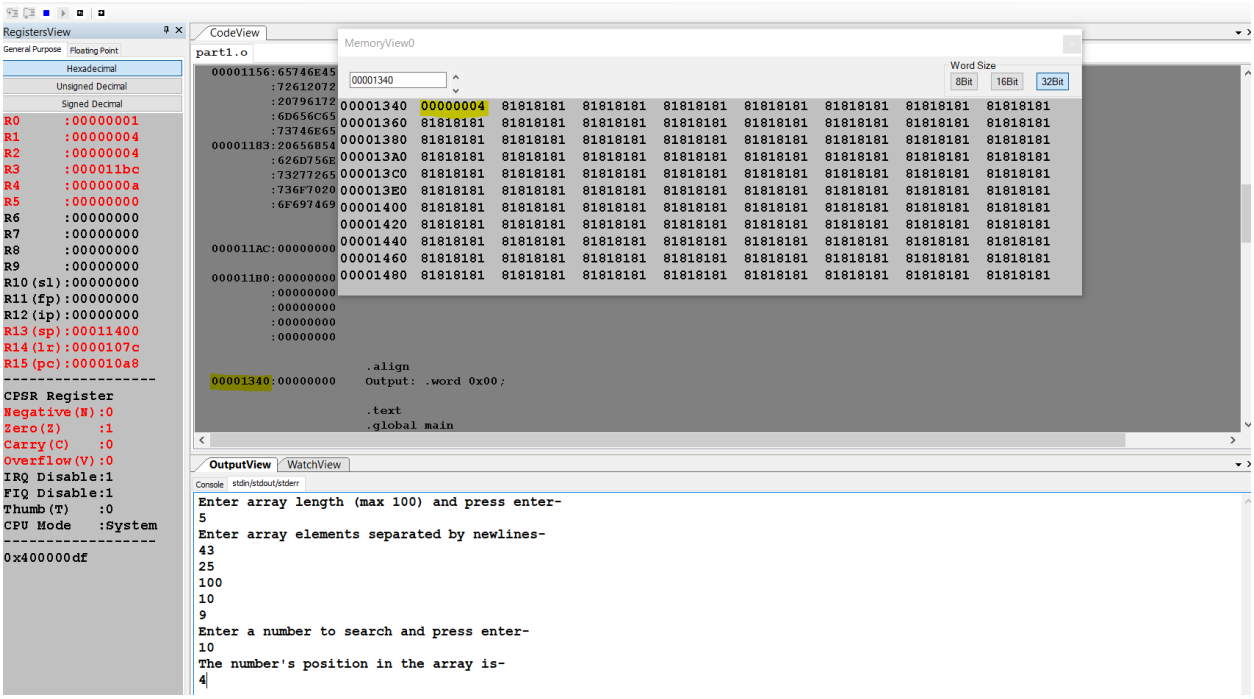
Inputs- Length of array and array elements (unsorted)

Output - Position of element in array if found (-1 otherwise)

Test inputs and outputs:

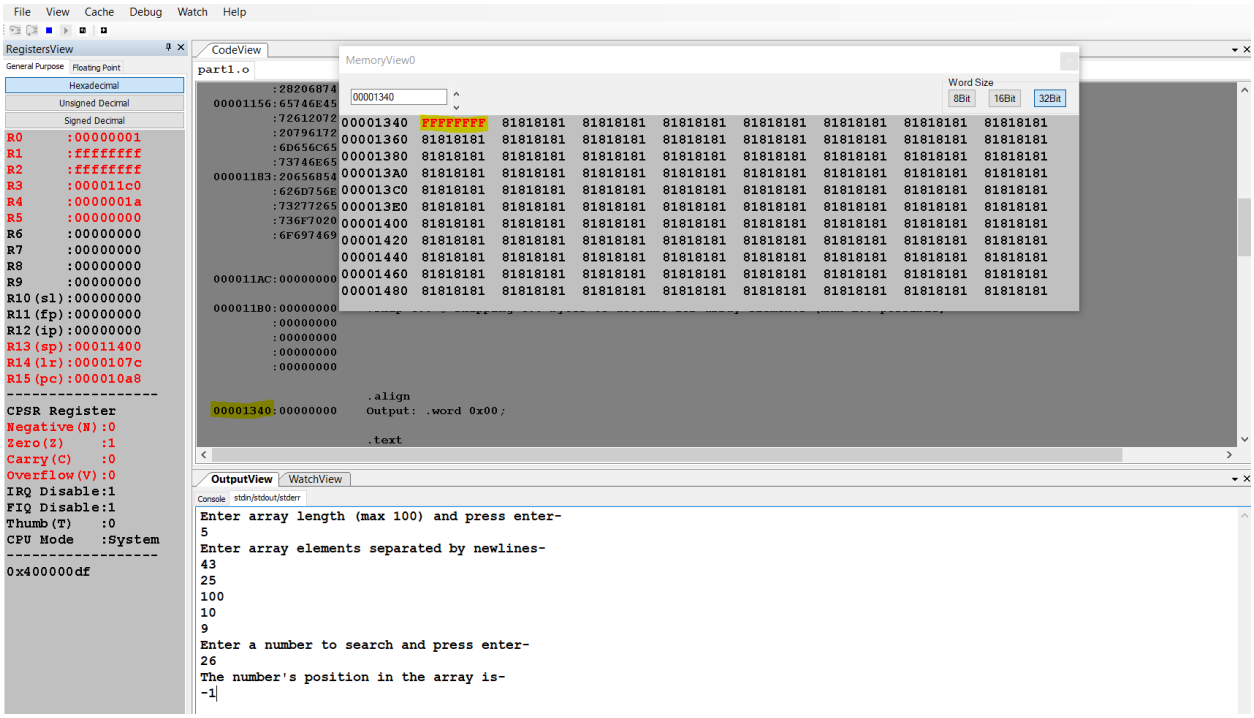
1. **Inputs**:
   Array length – 5
   Array elements – 43, 25, 100, 10, 9
   Number to be searched – 10

   Output:
   Position - 4



2. **Inputs**:
   Array length – 5
   Array elements – 43, 25, 100, 10, 9
   Number to be searched – 26

   Output:
   Position = -1



Part 2 - Find the position of an element in a sorted array using minimum number of memory accesses.

Algorithm used - Binary search algorithm to effectively find the element in a sorted array. Complexity is O(log(n))

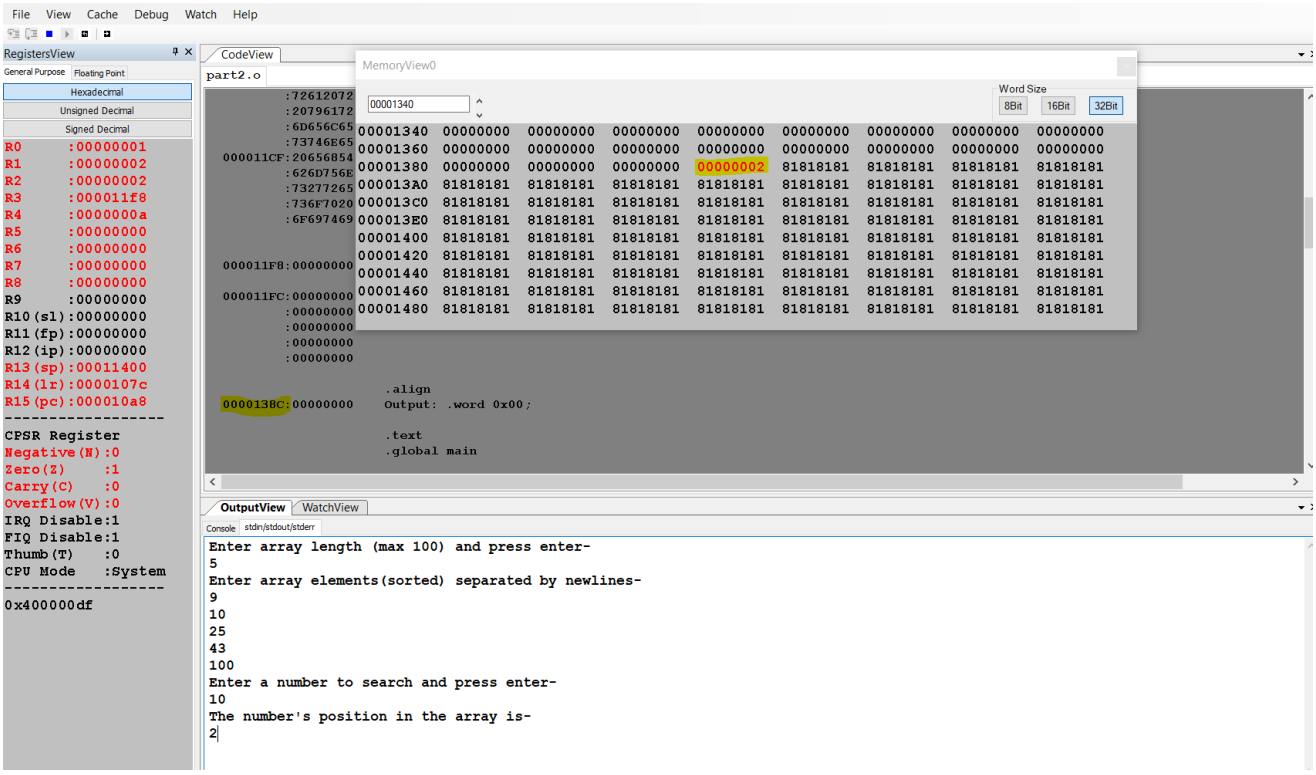
Input- Length of array and array elements (sorted)

Output - Position of element in array if found (-1 otherwise)

Test inputs and outputs:

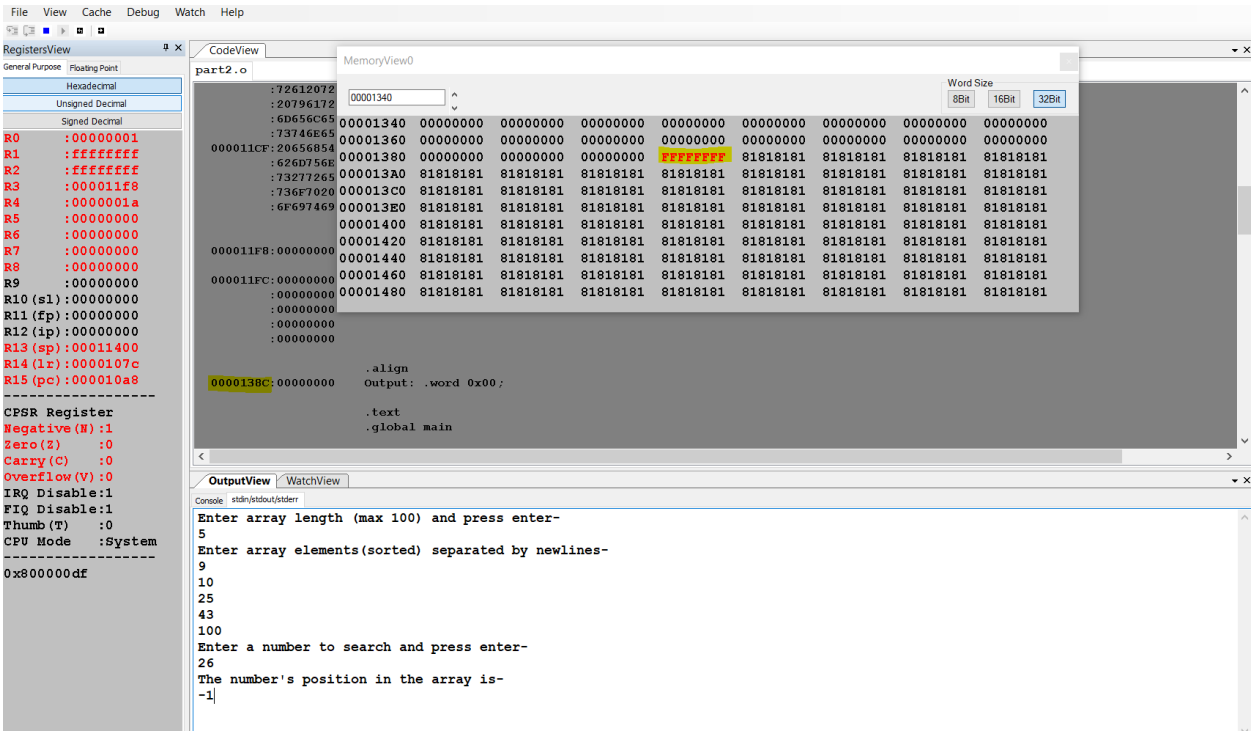
1. **Inputs**:
   Array length – 5
   Array elements – 9, 10, 25, 43, 100
   Number to be searched – 10

   Output:
   Position - 2



2. **Inputs**:
   Array length – 5
   Array elements – 9, 10, 25, 43, 100
   Number to be searched – 26

   Output:
   Position = -1



Part 3 - Print the Nth fibonacci number in the sequence given a number N

Algorithm used ->

- Calculating fiboncacci sequence by recursion using stack
- Fibonacci(n) = Fibonacci(n-1) + Fibonacci(n-2)
- Popping the first two values which are results of the previous iterations, summing them up and pushing the results to the stack.

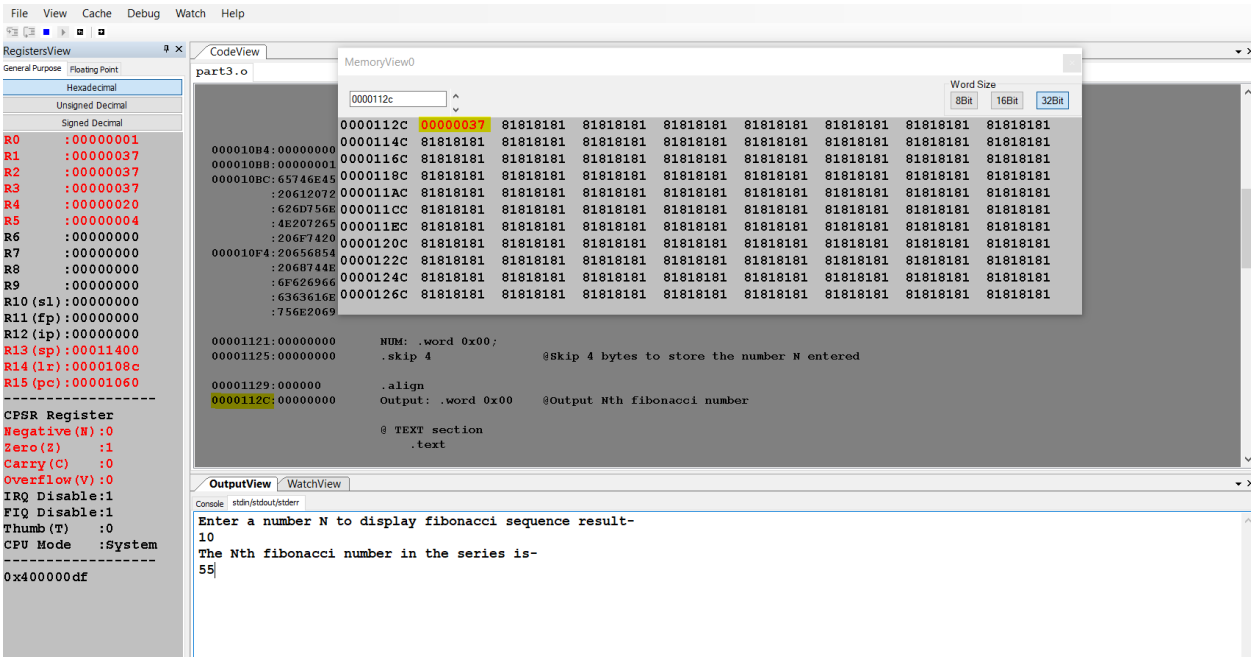
Input - Number N

Output - Nth number in the Fibonacci sequence

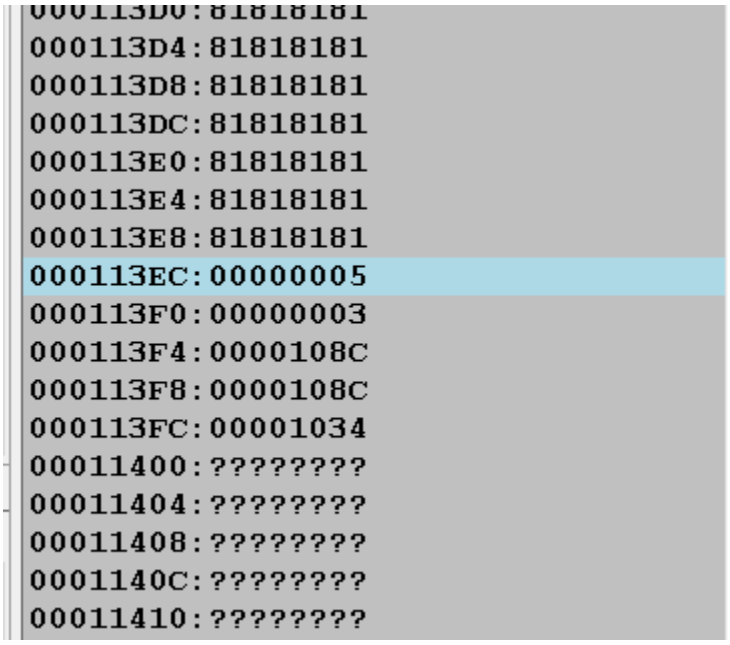
Test inputs and outputs:

Input – 10

Output – 55



Stack during operation ->



```
                        F_CONT:
00001088:EBFFFFF5               BL FIBONACCI;

                        DONE:
0000108C:E49D0004               POP {R0};
00001090:E3A05004               MOV R5, #4;
00001094:E0040594               MUL R4, R4, R5;
00001098:E08DD004               ADD SP, SP, R4;
0000109C:E49DF004               POP {PC};
000010A0:00000004               .end
```

In each recursive call, the result of the previous two iterations are on the stack along with the return addresses.

Here 0x108c is the return address of the subroutine FIBONACCI for every call after the initial one(which returns to the main program).

0x1034 is the return address to the main program. So to obtain this, incremented the SP by the required number of times while returning to main program.

```
000113C0:81818181
000113C4:81818181
000113C8:81818181
000113CC:81818181
000113D0:81818181
000113D4:81818181
000113D8:81818181
000113DC:00000022
000113E0:00000015
000113E4:0000108C
000113E8:0000108C
000113EC:0000108C
000113F0:0000108C
000113F4:0000108C
000113F8:0000108C
000113FC:00001034
00011400:????????
00011404:????????
00011408:????????
```

```
0000102C:E52D1004               PUSH {R1};
00001030:EB00000B               BL FIBONACCI;
00001034:E59F1070               LDR R1, =Output;
00001038:E5810000               STR R0, [R1];
0000103C:E1A02000               MOV R2, R0;
```