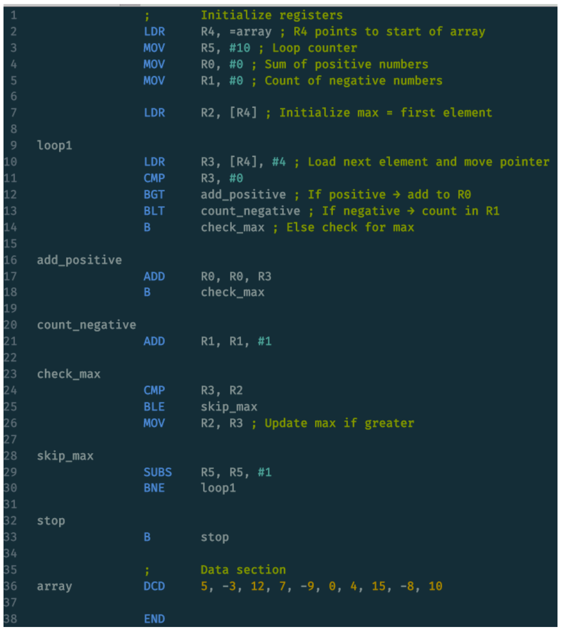
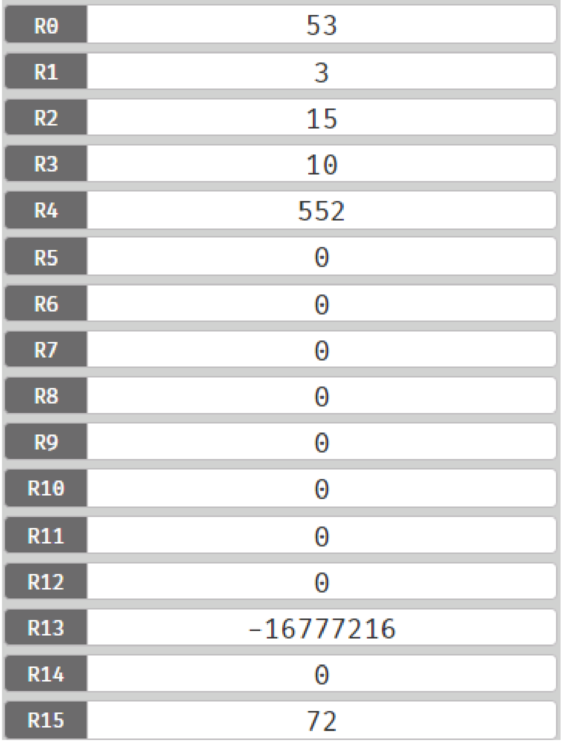
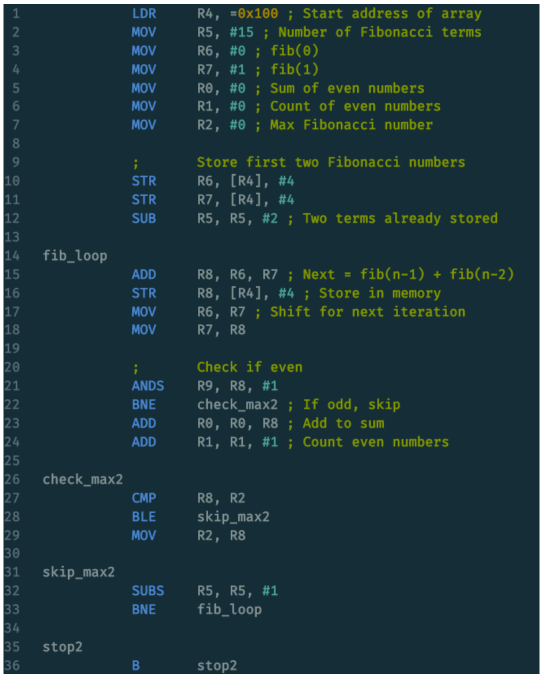
- Problem 1
    - 

```
            ;       Initialize registers
            LDR     R4, =array ; R4 points to start of array
            MOV     R5, #10 ; Loop counter
            MOV     R0, #0 ; Sum of positive numbers
            MOV     R1, #0 ; Count of negative numbers

            LDR     R2, [R4] ; Initialize max = first element

loop1
            LDR     R3, [R4], #4 ; Load next element and move pointer
            CMP     R3, #0
            BGT     add_positive ; If positive → add to R0
            BLT     count_negative ; If negative → count in R1
            B       check_max ; Else check for max

add_positive
            ADD     R0, R0, R3
            B       check_max

count_negative
            ADD     R1, R1, #1

check_max
            CMP     R3, R2
            BLE     skip_max
            MOV     R2, R3 ; Update max if greater

skip_max
            SUBS    R5, R5, #1
            BNE     loop1

stop
            B       stop

            ;       Data section
array       DCD     5, -3, 12, 7, -9, 0, 4, 15, -8, 10

            END
```

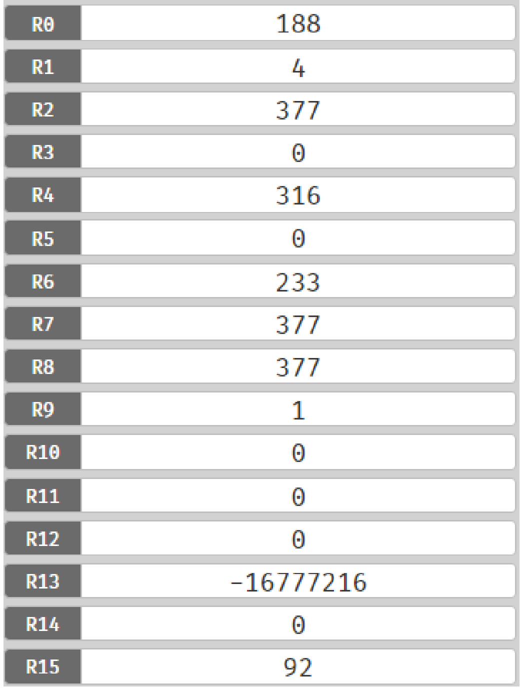
    - 

```
R0     53
R1     3
R2     15
R3     10
R4     552
R5     0
R6     0
R7     0
R8     0
R9     0
R10    0
R11    0
R12    0
R13    -16777216
R14    0
R15    72
```

    - Loop through the array, use the conditional checks BGT and BLT to check if a value is positive or negative, which sends to their respective conditional statement. Check the largest number using CMP, if there is a new largest number, use MOV to change the value of R2. If the value is positive, add it to the value at R0 and send to check_max. If negative, increment R1 by one.
- Problem 2



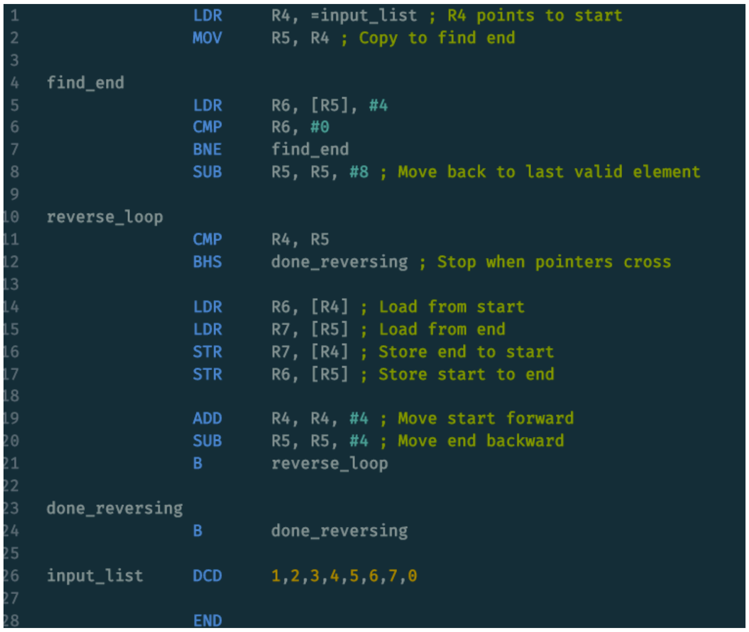
```
            LDR     R4, =0x100 ; Start address of array
            MOV     R5, #15 ; Number of Fibonacci terms
            MOV     R6, #0 ; fib(0)
            MOV     R7, #1 ; fib(1)
            MOV     R0, #0 ; Sum of even numbers
            MOV     R1, #0 ; Count of even numbers
            MOV     R2, #0 ; Max Fibonacci number

            ;       Store first two Fibonacci numbers
            STR     R6, [R4], #4
            STR     R7, [R4], #4
            SUB     R5, R5, #2 ; Two terms already stored

fib_loop
            ADD     R8, R6, R7 ; Next = fib(n-1) + fib(n-2)
            STR     R8, [R4], #4 ; Store in memory
            MOV     R6, R7 ; Shift for next iteration
            MOV     R7, R8

            ;       Check if even
            ANDS    R9, R8, #1
            BNE     check_max2 ; If odd, skip
            ADD     R0, R0, R8 ; Add to sum
            ADD     R1, R1, #1 ; Count even numbers

check_max2
            CMP     R8, R2
            BLE     skip_max2
            MOV     R2, R8

skip_max2
            SUBS    R5, R5, #1
            BNE     fib_loop

stop2
            B       stop2
```



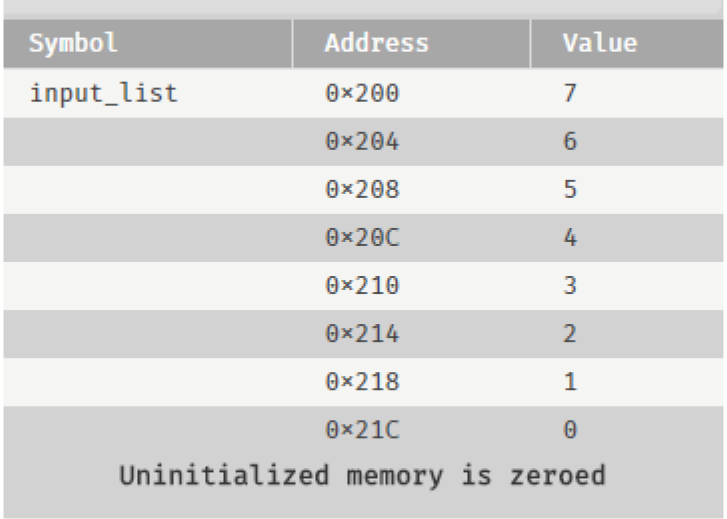
R0	188
R1	4
R2	377
R3	0
R4	316
R5	0
R6	233
R7	377
R8	377
R9	1
R10	0
R11	0
R12	0
R13	-16777216
R14	0
R15	92

- The fibonacci sequence is generated iteratively through the fib_loop sequence, which is stored in 0x100 using STR statements. Check if the value is even through the use of ANDS. If it is, increment R1 by one and add the value to R0. Check if the newest value, stored in R8, is greater than that stored in R2. If it is, set R2 to the value stored in R8.

- Problem 3

- 

```
                LDR     R4, =input_list ; R4 points to start
                MOV     R5, R4 ; Copy to find end

find_end
                LDR     R6, [R5], #4
                CMP     R6, #0
                BNE     find_end
                SUB     R5, R5, #8 ; Move back to last valid element

reverse_loop
                CMP     R4, R5
                BHS     done_reversing ; Stop when pointers cross

                LDR     R6, [R4] ; Load from start
                LDR     R7, [R5] ; Load from end
                STR     R7, [R4] ; Store end to start
                STR     R6, [R5] ; Store start to end

                ADD     R4, R4, #4 ; Move start forward
                SUB     R5, R5, #4 ; Move end backward
                B       reverse_loop

done_reversing
                B       done_reversing

input_list      DCD     1,2,3,4,5,6,7,0

                END
```

- 

Symbol	Address	Value
input_list	0x200	7
	0x204	6
	0x208	5
	0x20C	4
	0x210	3
	0x214	2
	0x218	1
	0x21C	0

Uninitialized memory is zeroed

- Traverse through the array until the value is equal to 0, indicating the end of the array. Then set start and end pointers, R4 and R5 respectively, swapping the values between the two pointers the the value for start is greater than that of end. Each loop sets the address of the address of R4 forward and R5 backward. Once R4 is greater than R5, jump to done_reversing before more shifts occur.