Lab Guide 8B

In this Lab, you will write ARM assembly program, and test it using browser-based ARM simulator.

Here is the link for the simulator: https://cpulator.01xz.net/

Step-1: Choose "ARMv7" from Architecture menu

Step-2: Choose "ARMv7 DE1-SoC" from System menu

You will be directed to this link: https://cpulator.01xz.net/?sys=arm-de1soc

Exercise 1: Generate N even numbers

Write a program that generates the first N even numbers and stores them in an array. N is a user-defined positive integer. Use a loop to iterate through the even numbers and store them in the array.

Hints: For reserving the space in memory for the array named as "result", you can use following directive at the end of your code:

```
. . . . .
.data
result:  .skip 40       @ Reserve space for 10 integers (4 bytes each)
```

.skip – is the special directive to reserve given amount of bytes in the memory.

You can load the address of the array "result" into any register using following:

```
LDR R6, =result    @ Load the address of the result array
```

=result - reads the address of the array variable "result"

Exercise 2: Calculate the sum of array elements

Write a program that calculates the sum of elements in a user-defined array of integers and stores the result to the some register. The array and its size should be user-defined. Use a loop to iterate through the array and accumulate the sum.

Hint: you can create an array of integers (4 bytes each) as follows:

```
.data
array:  .word 1,2,3,4,5
```

Exercise 3: Find the minimum and maximum elements in an array

Write a function that finds the minimum and maximum elements in a user-defined array of integers and stores the results to the registers R0 – minimum, R1-maximum. The function should take a pointer to the beginning of the array and its size as arguments. Use a loop to iterate through the array elements and update the minimum and maximum values accordingly. After function return, store the results to another registers before exiting: R2-minimum, R3-maximum.

Hint: you should use “PUSH” and “POP” instructions inside function to temporarily store used registers and retrieve them after the function call. If your function uses R4,R5,R6 registers, here is how you can save and retrieve original values:

```
@ function starts…
PUSH {R4,R5,R6}
. . . @ function body . . .
POP {R4,R5,R6}
BX LR  @return to the caller
```