**Exercise 1:**

**Input Location and Adding Input to Executor:**

- The location of the input is specified in the **“.data”** section which is globally declared using **“global INPUT”.**

.data 0

.global INPUT

    12

A screenshot of a number

Description automatically generated**“INPUT”** is initialized to 12, meaning that it will compute the first 12 Fibonacci numbers. The location of this input value is implicitly determined by its declaration as a global variable. It is accessible from any part of the program.

As we can see here, the input location is at line data 0 R0. It is because of this line:

**Output Location and Interpreting Output After Execution:**

.data 32

.global OUTPUT

    0 1

- This section initializes the memory space for storing the Fibonacci numbers. Since the first two Fibonacci numbers are 0 and 1, these values are stored in memory. The location of this output array is also implicitly determined by its declaration as a global variable. Like the input, it is accessible from any part of the program.

A screenshot of a computer code

Description automatically generated

- The location of the output is specified in the **“.data 32”** section which is globally declared using **“.global OUTPUT”.**

A screenshot of a number game

Description automatically generated**-** The output will start in data 32 R0 and R1 with values 0 and 1. I will set R2 and R3 to 0 and 1 to start the loop. The loop will work to calculate R4 from R2 and R3, and then, change R2 to the new R3 and R3 to R4 to calculate the next loop until it reaches the Fibonacci number 12.

**Explanation of Input Usage and Output Production:**

- The program loads the input value from the specified memory location into register R0 initially.

- It then checks if the input is 0 or 1. If so, it directly exits because the Fibonacci sequence for these inputs is predetermined (0 and 1 respectively).

- If the input is greater than 1, the program computes the Fibonacci sequence iteratively.

- The computed Fibonacci sequence is stored starting from the memory location specified by OUTPUT.

A screenshot of a computer program

Description automatically generated- The program iterates INPUT times to compute the Fibonacci sequence, storing each computed value in consecutive memory locations starting from OUTPUT.

**Exercise 2:**

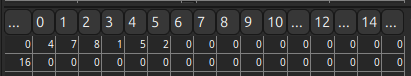
**Input Location and Adding Input to Executor:**

- The location of the input is specified in the **“.data”** section which is globally declared using **“global INPUT”.**

.data 0

.global ARR

    4 7 8 1 5 2

- The input will appear in the first line data 0:

**Output Location and Interpreting Output After Execution:**

- The output of this program can be interpreted as the sorted version of the input array ARR.

- The program implements a sorting algorithm that sorts the elements of ARR in ascending order.

- After execution, you can interpret the output by reading the values of ARR. The values will be sorted in ascending order.

A screenshot of a number

Description automatically generated

**Explanation of Input Usage and Output Production:**

- The output of this program can be interpreted as the sorted version of the input array ARR.

- The program implements a sorting algorithm that sorts the elements of ARR in ascending order.

- After execution, you can interpret the output by reading the values of ARR. The values will be sorted in ascending order.

- The program uses a sorting algorithm to sort the elements of the input array ARR.

- It iterates through the elements of ARR using nested loops.

- In each iteration, it compares adjacent elements of the array and swaps them if they are in the wrong order.

- This process continues until the array is fully sorted.

- After execution, the sorted array is stored in the same memory locations originally occupied by ARR.

🡪 Therefore, to interpret the output, you would simply read the values of ARR after the program has executed, and these values would be sorted in ascending order.

A screenshot of a computer program

Description automatically generated