MARMARA UNIVERSITY

FACULTY OF ENGINEERING

CSE 3038 – Computer Organization

Submitted to: Prof. Haluk Topçuoğlu

Due Date: 03.04.2024

	Dept	Student Id	Name Surname
1	CSE	150120027	Mert Muslu
2	CSE	150120051	Erkut Dönmez
3	CSE	150121539	Gülsüm Ece Günay

Question 1:

For the initial inquiry, we request coefficients necessary for computing the function outlined in the homework assignment. Additionally, we prompt the user to provide the first two values of the function, denoted as f(0) and f(1). Based on these inputs, our function structure is established, and the user must then specify the desired element to retrieve.

Once the user inputs are received, we store the coefficients a and b into $t0 and $t1, respectively. Simultaneously, we capture the values of f(0) and f(1), storing them in $t2 and $t3. Moreover, the user's designated element (n) is stored in $t4. Subsequently, we verify whether the value of n is less than or equal to 1. If this condition is met, the program redirects to a section labeled "invalid_input." Here, the user is prompted again to input the coefficients, the initial function values, and the desired element (n). This loop continues until an appropriate value for n is entered, ensuring valid inputs.

Once a suitable value for n is provided, the program progresses to the subsequent segment known as "sequence_calculation." In this phase, the value of $t4 (n) is transferred to $t7 for further computations, serving as a control variable for subsequent operations. Following this, the program enters the sequence_loop, executing the formula a * f(x-1) + b * f(x-2) - 2. The resulting value is stored in $t3, replacing the previous value of f(1), which is now stored in $t2. This exchange prepares the register for the next iteration of the loop, optimizing resource usage.

Sample run:

```
Please enter the coefficients: 2
1
Please enter the first two numbers of the sequence: 1
2
Enter the number you want to calculate (it must be greater than 1): 4
Output: 6
-- program is finished running --
```

```
Please enter the coefficients: 6
1
Please enter the first two numbers of the sequence: 0
1
Enter the number you want to calculate (it must be greater than 1): 5
Output: 140
-- program is finished running --
```

```
Please enter the coefficients: 6
1
Please enter the first two numbers of the sequence: 0
1
Enter the number you want to calculate (it must be greater than 1): 6
Output: 861
-- program is finished running --
```

Question 2

Names of the parts and their responsibilities are has examined in details:

.data:

~An array named “array” is declared for 10 integers of space.

~eleNum: prompts the user to enter the number of elements in array.

~mess: prompts the user to enter numbers to the array.

~output: string to display output on the screen

~newline: to print a newline

~hata: error message.

main:

~Programs starting point.

~Jumps to the part we called as “read”

read:

~Prompts the user to enter the number of elements in the array.

~Reads the input.

~Stores the size of the array in both $t1 and $k0.

~Initialized the needed variables.

~Jumps to the readLoop

readLoop:

~Loops to read and store integers as many as user wished.

firstEleAdress:

~Initialized the base address of our array into the $t0

~Jumps to the GCD

GCD:

~Calculates the greatest common divisor of two numbers.

~Uses euclid’s algorithm to find gcd.

~Swaps if necessary to ensure the first number is greater than second.

~Computes GCD recursively.

~If the numbers are coprime(GCD is 1), updates the second number in the array.

~Reorders the array to place zeros at the end.

ReArray:

~Rearranges the array by moving zeros to the end.

Sample run:

```
 Enter the number of elements in the array: 6
 Enter 10 numbers to be stored in the array. 6
 Enter 10 numbers to be stored in the array. 4
 Enter 10 numbers to be stored in the array. 2
 Enter 10 numbers to be stored in the array. 3
 Enter 10 numbers to be stored in the array. 7
 Enter 10 numbers to be stored in the array. 13
12 - 7 - 13 -
-- program is finished running --
```

```
 Enter the number of elements in the array: 7
 Enter 10 numbers to be stored in the array. 25
 Enter 10 numbers to be stored in the array. 2
 Enter 10 numbers to be stored in the array. 3
 Enter 10 numbers to be stored in the array. 9
 Enter 10 numbers to be stored in the array. 6
 Enter 10 numbers to be stored in the array. 4
 Enter 10 numbers to be stored in the array. 5
25 - 36 - 5 -
-- program is finished running --
```

```
 Enter the number of elements in the array: 5
 Enter 10 numbers to be stored in the array. 3
 Enter 10 numbers to be stored in the array. 5
 Enter 10 numbers to be stored in the array. 15
 Enter 10 numbers to be stored in the array. 4
 Enter 10 numbers to be stored in the array. 7
15 - 4 - 7 -
-- program is finished running --
```

Question 3:

To solve the question 3 we create a mixed C-pseudocode and implement that logic to mips code here is the logic that we implemented for question 3:

```
// Pseudocode for Question 3
int z = (n/pow(2,a))-1
for(int a= 1; a <= shuffle_number;a++){
    int c = 0;
    for(int i = 0; i <=z;i++){
        swap $t0 + i with $t0 + n/pow(2,a) + i
    }
    if(a > 1 && c != 0){
        i = n/pow(2,a-1) - n/pow(2,a) + 1;
        z = n / pow(2,a-1);
        c--;
        goto inner for loop;
    }
}
```

Pseudocode:

Initialization: Initialize variable z as (n / 2^a) - 1, where n is the length of the string and is initially decided by the string tha the user has entered. Start a loop that iterates from a = 1 to shuffle_number, where shuffle_number represents the number of times the shuffling process should occur.

Outer Loop: Within the outer loop, set a variable c to 0. Start an inner loop that iterates from i = 0 to z. Inside this loop, perform a swapping operation between characters located at the positions $t0 + i and $t0 + n / 2^a + i. This swapping effectively shuffles the string.

Check and Update: After the inner loop, check if a is greater than 1 and if c is not equal to 0. If both conditions are met, calculate a new value for i as n / 2^(a-1) - n / 2^a + 1. Update z as n / 2^(a-1). Decrement c by 1. Jump back to the inner loop to continue shuffling.

Repeat: Repeat the process until the outer loop finishes executing shuffle_number times.

This pseudocode describes a shuffling algorithm where each iteration of the outer loop performs a shuffling operation based on a specific calculation of z. After each shuffle, it checks a condition to determine whether to modify the shuffling behavior for subsequent iterations.

Sample run:

```
Enter a string to shuffle: computer
shuffle: 1
output: utercomp

-- program is finished running (dropped off bottom) --

Enter a string to shuffle: computer
shuffle: 2
output: erutmpco
-- program is finished running (dropped off bottom) --

Enter a string to shuffle: computer
shuffle: 3
output: retupmoc
-- program is finished running (dropped off bottom) --
```