CSO Assignment #2

Instructor: Dr. Avinash Sharma *Due date: 07/03/2020*

The aim of this assignment is to familiarize you with procedure calls, conditional jumps and stacks.

Instructions: Read all the instructions below carefully before you start working on the assignment.

- There are 3 problems for this assignment.
- Writing complete code with successful execution guarantees full marks. Failure on test cases will result in penalisation. Therefore ensure all edge cases are handled.
- The assignments will be manually evaluated for plagiarism. Any and all forms of plagiarism will result in 0 credit for this assignment.
- Write well-organised code using procedures for repeated operations.
- **Comment every line of your code and justify why you write that statement**. Well-commented code will be awarded extra credit.

Submission Format: Strictly adhere to the following submission format. Failure to do so may result in an erroneous evaluation of your assignment.

- The following directory structure is expected,

```
./<roll_number>
    |__ Q1.s
    |__ Q2.s
    |__ Q3.s
    |__ Q4.s
    |__ Q4.pdf
```

- Zip the ./ <roll_number> folder and name the zipped folder as <roll_no_assign2.zip>

Problem 1: More Loops

(20 points)

Given 2 numbers A and B, find the sum of digits of X such that X is the maximum number that satisfies the following conditions:

- A is divisible by X
- B and X are coprime

For example, if A = 30 and B = 8, X will be 15 and the required answer will be 6.

Instructions
- You have to write the appropriate code in x86-64 assembly language.
- A and B will be stored in register *%rbx* and *%rcx* respectively.
- Store the final answer in *%rdx*.

Problem 2: Procedure Calls (35 points)

Given numbers n and k, compute the sum of factorials of all Fibonacci numbers less than or equal to n. Report the final answer mod k. Take 1 and 2 as the first 2 Fibonacci numbers.

Input/Output Format

- You will be given the integer n and k.
  $1 \leq n \leq 100$ and $2 \leq k \leq 100$.

Sample Test Case

- n = 5
  k = 13
  answer = (1! + 2! + 3! + 5!) mod 13
  answer = 12

Instructions

- You have to write the code in x86-64 assembly language
- The number n and k will be stored in register *%rbx* and *%rcx* respectively
- Store the final answer in *%rdx*
- Ensure that overflow doesn't occur by taking appropriate modulo operations at each stage.
- As the problem involves multiple repeated operations, it is suggested to write procedures for the different operations.

Problem 3: Stack (45 points)

There are n students standing in a line. You are given the height for each student. For each student find the height of the nearest shorter person to the left of the student. In case no such student exists , the answer is -1.

Input/Output Format

- You will be given the integer n the size of the input array. $1 \leq n \leq 100$
- The input and output arrays will be stored in the memory with only the base address provided.

Sample Test Case

- n = 6
  input array = [4, 4, 5, 2, 10, 8]
  output array = [-1, -1, 4, -1, 2, 2]

Instructions

- You have to write the code in x86-64 assembly language
- The number n will be stored in register *%rbx.*
- The base address of the input array will be stored in register *%rcx.*
- The base address of the output array will be stored in register *%rdx.*
- Consider the heights to be 8-byte (64-bits) integer. Use move suffixes and store the answers in memory accordingly.

Important Notes

- For testing, store some values in the memory taking the default %rcx and %rdx values.
- It is **mandatory** to use stacks to solve this question. We will be monitoring the stack-usage during the evaluation. Other implementations will be penalised

Problem 4: What's the Question ? (25 points)

We are giving you the code for a famous problem. code link The original output for the code was following , however some bugs were introduced which resulted in erroneous functioning of the code. You are required to do the following.

- Identify what this code does. Briefly explain the functioning of the code and relevant functions. (You can ignore the pt(print) functions).
- Debug the code by adding/changing appropriate commands so as to achieve the desired output. (You can ignore the pt(print) functions)
- You need to submit a pdf report describing the code functioning , relevant functions detail and the bugs found. Also submit the corrected code in a .s file.

Correct Output:

```
3 2 1



3 2

1

3
2
1

3
2 1



2 1
3

1
2
3

1

3 2



3 2 1
```