Group 24

Gorantla Thoyajakshi – 21CS10026

Ashwin Prasanth – 21CS30009

Computer Organization and Architecture Lab

Assignment 4, Question 1

- We created a recursive function called **sum_series**. The recursive function works like so:
- **If n=1, sum_series(n) = 1 (ie, n) (base case)**
- **else, sum_series(n) = sum_series(n-1) + n^n**
    - in the **sum_series** function, we checked for base case (n==1). If it is true, we stored 1 (n) in $v0 (return value) then went directly to **exit_rec**, where stack was popped, ra and t0 were restored and return was done to the caller
    - If it is not base case, then we called sum(n-1), then calculated n^n using **cal_npown** function (described below), and then added both and stored to $v0
    - The **input** was taken in **$a0** and **output** was stored in **$v0**
- For calculating n^n, we made a function called **cal_npown**, which calculates **n^n** iteratively. This could also have been done recursively, but recursive calls within recursive calls could potentially result in stack overflow
    - In the **cal_npown** function, we initialized a result to 1, and iteratively multiplied it by n, n times
    - The **input** was taken in **$a1** and **output** was stored in **$v1**
- Also, a sanity check was done in main to ensure that the input was greater than 1. If not, invalid prompt is shown and program is exited.

Assignment 4, Question 2

- We created a recursive function called collatz. The recursive function works like so:
- **If n=1, return (base case)**
- **Else,**
- **If n%2==0 (n is even), collatz(n/2) is called**
- **If n%2==1 (n is odd), collatz(3n+1) is called**

    - In the **collatz** function, we also used a counter variable **$s1** to count the number of steps taken to reach 1. Everytime function is called, the counter is incremented, hence we get number of steps
    - But we decreased 1 from the counter because it is also counting the initial case of when we just got n
    - For base case(n=1), we directly jump to **exit_rec,** where stack was popped, and ra and t0 were restored
    - Else, we checked whether n was even or not, and correspondingly executed either **odd_rec** or **even_rec** branch
    - The **input** was taken in **$a0** and output was stored in **$s1**
- Just like question 1, a sanity check was done in main to ensure that the input was greater than 0. If not, invalid prompt is shown and program is exited.