Lab 1

Submit Assignment

Due Feb 1 by 12:30pm **Points** 100 **Submitting** a file upload **File Types** s
Available Jan 16 at 1:30pm - Feb 1 at 12:30pm 16 days

Purpose

You will be writing a CISC (Intel x64) assembly program to understand how programming languages instruct the actual CPU. Granted, assembly is one step removed from machine language, but it is essentially human-readable machine language.

This lab will give you experience with the syntax and logic associated with low-level languages. Furthermore, this lab will give you experience with stack frames, recursive functions, and tail recursion.

Structure

Use the following structures for the functions below:

Stack Structure

```
struct Stack {
     int *data;
};
```

The Stack structure contains a single integer pointer. When the stack is initially created, the data pointer will point to the very bottom of the stack.

Node Structure

```
struct Node {
      int value;
      struct Node *left;
      struct Node *right;
};
```

The Node structure contains a value and a pointer to the left and right nodes of a binary search tree.

Assignment

You will be writing x64 assembly in the Intel syntax using the GNU Assembler (GAS). Your code must assemble on the Hydra or Tesla lab machines. **Any code that does not assemble will not be graded.**

You will need to write the following seven (7) functions to perform the given operations:

Fibonacci numbers

```
long FibRecur(short n);
```

FibRecur will return the nth number in the Fibonacci sequence starting with Fib(0) = 1 and Fib(1) = 1. You must use recursive stack frames to solve this function. If the parameter n is negative, return Fib(0).

```
long FibTail(short n);
```

FibTail will return the nth number in the Fibonacci sequence starting with Fib(0) = 1 and Fib(1) = 1. You must use tail recursion to solve this function. You may **NOT** use iteration for this function. If the parameter n is negative, return Fib(0).

Reminder: The fibonacci sequence is Fib(n) = Fib(n-1) + Fib(n-2)

Stack functions

```
void Push(Stack *s, int value);
```

Push will push the parameter *value* onto the stack given by the parameter *s*. Remember that the actual data is stored as a pointer in the Stack structure. You must properly move the stack pointer in this function. **Note:** You will not be checking for boundary cases.

```
int Pop(Stack *s);
```

Pop will pop the topmost value located on the stack *s* and return its value. You must properly move the stack pointer in this function. **Note:** You will not be checking for boundary cases.

Tree functions

```
Node *Insert(Node *root, int value);
```

Insert will insert the value onto the binary search tree given by the parameter root. Insert must use stack frames and recursion. You must call malloc to create memory for a new node.

```
Node *Search(Node *root, int search_value);
```

Search will search for the value given by *search_value* in the tree given by the parameter *root*. This function will return a pointer to the node with the given value, or NULL if no such node was found in the tree. Search must use stack frames and recursion.

```
void PrintTree(Node *root);
```

PrintTree will print each node's value in the tree given by the parameter *root*. This function must use stack frames and recursion.

Plagiarism

Please remember that all labs are an individual effort. Please review the **plagiarism policy on the course syllabus**.

Submission

Submit your lab as a .S file on Canvas.