Jordan Christian
CS 320 POCO
MIPS Project

Design of MIPS Fibonacci Program

How It Works:

First off, the Fibonacci sequence takes the sum of the two previous numbers and this number is the next number in the sequence. The traditional Fibonacci sums all of these values. For example, if I enter 5, then the sequence would be 0, 1, 1, 2, 3, 5. The Fibonacci number for 5 is the sum of all of these which is 12. For this project, instead of printing the sum, we are printing the actual sequence that corresponds to the number entered by the user.

My MIPS assembly code is based upon the C code below which uses a For Loop to print the sequence of values in the Fibonacci sequence given the n^{th} value. For example, if the user inputs the value 5 for n, then the output will be 0, 1, 1, 2, 3, 5. (MIPS code will output the same thing) For every Fibonacci sequence, 0 will always be the first value printed no matter the input of n. In order to handle this in both the C and MIPS code, I printed the value of 0 before the loop begins.

For the MIPS code, I start by declaring a stack and saving the return address and frame pointer to the stack. I then initialize term1 and term2 and save those to the stack as well. I use li function to display a message and take the input from the user which will be ‘n’. Before I start the loop section, I use the add intermediate to increase the value of ‘n’ by 1. This will help for the comparison part of the for loop. Without increment, the loop only goes until n-1, but now I guarantee n will be included when the slt instruction is executed. To make it a little bit clearer:

-User Input for ‘n’ = 5, when n=5 the slt instruction will produce 0 for $t1 since $s4 will no longer be less $s2. Therefore, the loop will terminate and the 5th value will not be printed in the output. By adding 1 to ‘n’ before the loop, this allows the loop to reach the true input value and thus the correct sequence will be printed.

Next, I loaded the values of term1 and term2 and add those to form the sum. This value is then saved to the stack under $s3. The next two instructions, *term1=term2* & *term2=sum,* are executed in a similar fashion in MIPS. The address of term 1 is overridden and saved with the value from term2 and similarly the value of term2 is overridden and saved with the value that was in sum. Then the value of term2, which is now the sum, is printed at the beginning of the next loop. This swapping of values continues as the loop progresses. Once the value of ‘i’ is equivalent to the value of ‘n’, the ‘slt’ instruction will produce 0 and this will then cause the ‘beq’ instruction to produce true and jump to the instruction labeled ‘DONE’. This means that the loop has finished and the ‘DONE’ section simply terminates the program.

One restriction my program has is handling inputs that are not numbers. Currently, if a user inputs a letter, the system only prints a 0 and that’s because it is told to print that in an instruction before the loop begins. Technically, if the input is anything other than a number then it jumps straight to the ‘DONE’ section and terminates the program. I am not sure of a better way to handle this or how to raise errors through MIPS code.

Design of Project:

My code uses 5 saved temporaries $s0, $s1, $s2, and $s3 and $s4. These variables are declared at the beginning of the function and therefore require I use saved registers. I used $s4 for 'I' and incrementing the For-Loop counter ($t1 was used once to save the result of the slt instruction). Register $v0 is used only when a string or digit needs to be printed. Once syscall is called, this register is free to be used again elsewhere. Register $a0 is used for arguments. When needing to print a string declared at the beginning, the string is passed to this register so it can then be printed.

C Program:

```c
#include <stdio.h>
int main() {
    int i;
    int n;
    int term1 = 0;
    int term2 = 1;
    int sum;

    printf("Enter the number of terms: ");
    scanf("%d", &n);

    printf("%d", term1);
    printf("\n");

    for (i = 1; i <= n; ++i)
    {
        printf("%d  ", term2);
        printf("\n");
        sum = term1 + term2;
        term1 = term2;
        term2 = sum;
    }
    return 0;
}
```

MIPS Register	Corresponding C variable
$s0	int term1;
$s1	int term2;
$s2	int n;
$s3	int sum;
$s4	int i;

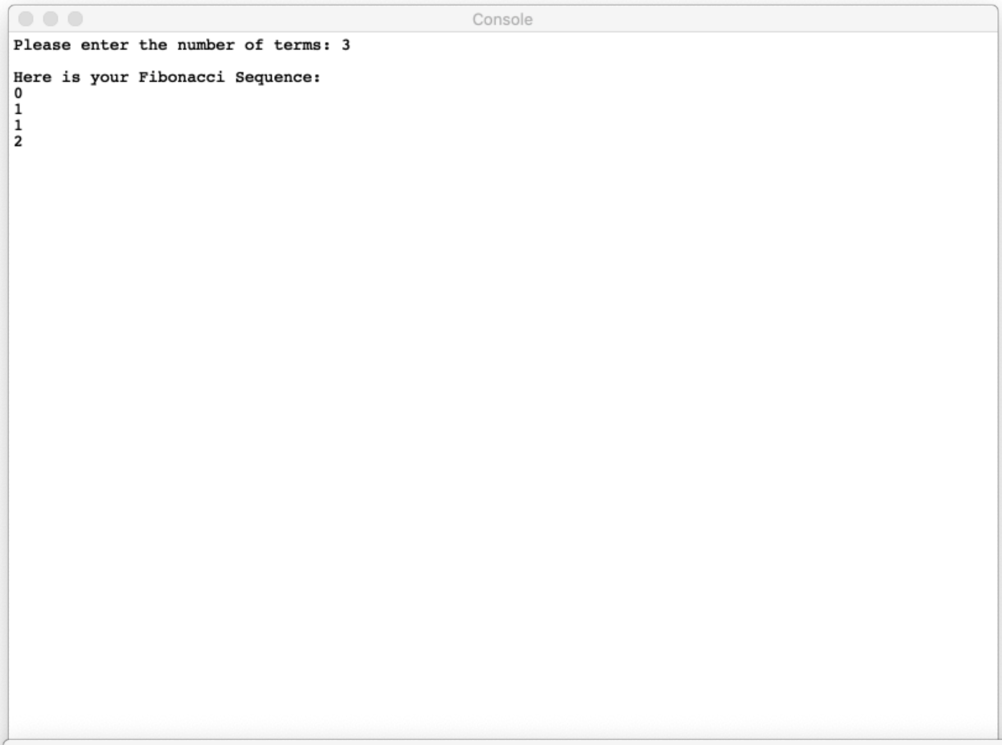
-All of these variables are declared within the C code above, therefore in MIPS they should all be stored in Saved Registers, $s0 → $s4.

Static Memory Used:

Static memory space is used for Global Variables. My code contains 3 strings which are defined as global variables. Those strings are the following:

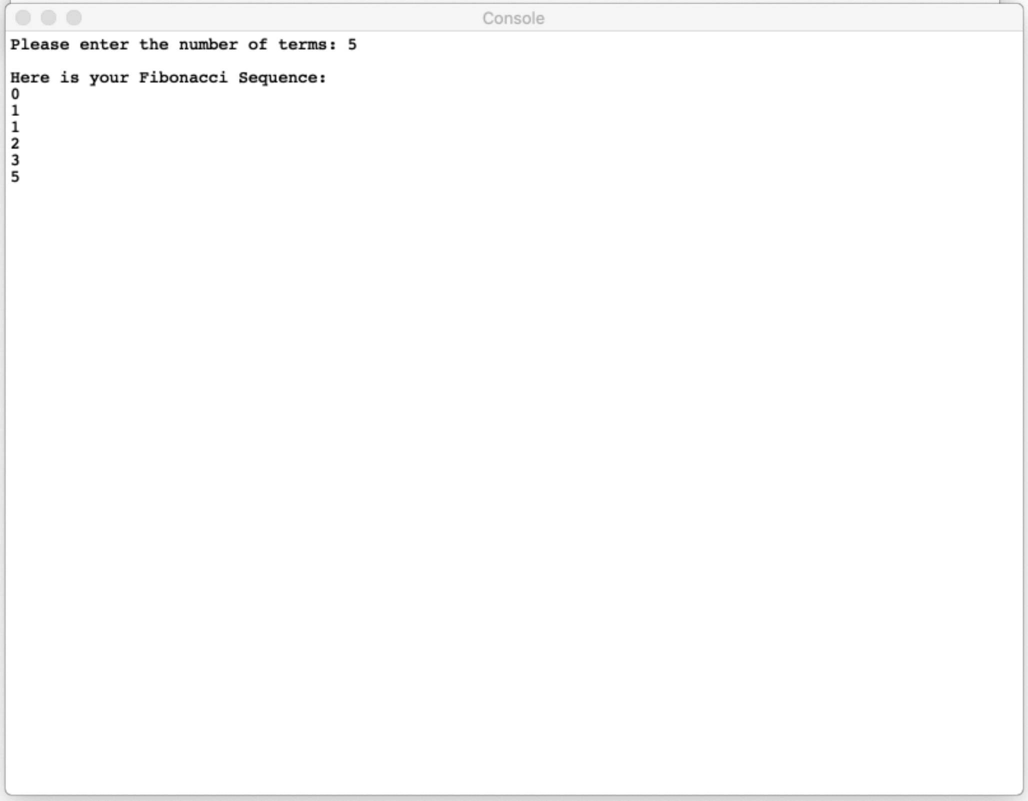
```
greeting: .asciiz "Please enter the number of terms: "
sequence: .asciiz "Here is your Fibonacci Sequence:"
NewLine: .asciiz "\n"
```

Output of MIPS Program:



```
Please enter the number of terms: 3

Here is your Fibonacci Sequence:
0
1
1
2
```



```
Please enter the number of terms: 5

Here is your Fibonacci Sequence:
0
1
1
2
3
5
```