CSC 256 - Machine Structures
Project 5

Total Points: 100 Points

Description

For project five, your objective is to convert the given C++ code into MIPS assembly. Please do not modify the C++ code itself. You are only allowed to make modifications to the assembly file. Start writing your code below the sumOfDoubleEvenPlace: and getDigit: labels.

When performing a C++ to MIPS conversion with functions, do so in the following steps:

1 Assign variables to registers. When inspecting the C++ code, any constant values(literals) may need to be assigned to temporary registers. Please check the comments in the .s file to see a list of pre-assigned variables to registers.

2 Initialize variables to registers. (actually put the values into the registers.)

3 Then move onto the rest of the code.

4 For functions, remember for non-leaf functions(functions that call other functions, you must saved the values of certain registers you are using to ensure correct code execution. You will save these values on the stack. Pushing values is done via storeword(sw) and popping values is done via loadword(lw).

5 Remember that $t, $v, $a, and $ra registers are ***assumed not*** to be preserved across function calls.

Before you begin, please make sure you click the link on ilearn to create your GitHub repo. After created please clone this repo with the *git clone repo_url* command.

Expected Output:

```
Expected Value: 23 Value: 23
Expected Value: 21 Value: 21
```

Submission

When you have completed the assignment please commit all work done to your private repository. This can be done with the following commands:

```
git add .
git commit -m "some message"
git push
```

Base MIPS Code

```
.data
  expVal23:     .asciiz  "Expected Value : 23  Your Value : "
  expVal21:     .asciiz  "Expected Value : 21  Your Value : "
  endl:         .asciiz  "\n"

.text

##
# int getDigit(int number);
# List Used Registers Here:
#
##
getDigit:



##
# int sumOfDoubleEvenPlace(int number);
# List Used Registers Here:
# sum --> $s0
# digit --> $s1
#
##
sumOfDoubleEvenPlace:

main:
  li $s0, 89744563  # int test1 = 89744563;
  li $s1, 98756421  # int test2 = 98756421;
  li $s2, 0         # int result1 = 0;
  li $s3, 0         # int result2 = 0;


 # code for first function call

  add $a0, $0, $s0
  jal sumOfDoubleEvenPlace
  add $s2, $0, $v0
```

```
38
39  la   $a0, expVal23
40  addi $v0, $0, 4
41  syscall
42
43  move $a0, $s2
44  addi $v0, $0, 1
45  syscall
46
47  la   $a0, endl
48  addi $v0, $0, 4
49  syscall
50
51   # code for first function call
52
53  add $a0, $0, $s1
54  jal sumOfDoubleEvenPlace
55  add $s3, $0, $v0
56
57  la   $a0, expVal21
58  addi $v0, $0, 4
59  syscall
60
61  move $a0, $s3
62  addi $v0, $0, 1
63  syscall
64
65  la   $a0, endl
66  addi $v0, $0, 4
67  syscall
68
69  li $v0, 10
70  syscall
```

C++ Equivalent

```cpp
#include <bits/stdc++.h>
using namespace std;

int sumOfDoubleEvenPlace(int number);
int getDigit(int number);

int main(void)
{
    int test1 = 89744563;
    int test2 = 98756421;
    int result1 = 0;
    int result2 = 0;

    result1 = sumOfDoubleEvenPlace(test1);
    cout << "Expected Value: 23  Value: " << result1 << endl;

    result2 = sumOfDoubleEvenPlace(test2);
    cout << "Expected Value: 21  Value: " << result2 << endl;

}
/*
* Function returns the sum of the even placed
* digits(after being doubled) starting from the left.
* Note that the algorithm starts counting from 1 not 0. Therefore,
* given the number 1234, 4 is the first digit from the left.
* So the even placed digits are 3 and 1 and the odd place digits are
* 4 and 2 from the left.
*/
int sumOfDoubleEvenPlace(int number) {
    int sum = 0;
    int digit;

    //Remove first odd digit
    number = number / 10;

    while (number > 0) {
        //Grab even placed digit
        digit = (number % 10);
        //Double the digit and pass it to getDigit,
        //Add result to sum
        sum += getDigit(digit*2);
        //Remove current even digit and the next odd digit.
        number = number/100;
    }
    return sum;
}

/* getDigit returns the sum of the digits in
 * a 1 or 2 digit number.
```

```
50  * if number is < 10,
51  * then we return the number.
52  * else we return the sum of the digits in the 2 digit
53  * number.
54  * For example:
55  * 1 would return 1
56  * 11 would return 2
57  * 18 would return 9
58  */
59 int getDigit(int number) {
60     int sum = 0;
61     if (number < 10) {
62         sum = number;
63     } else {
64         sum = number%10 + number/10;
65     }
66     return sum;
67 }
```