CS312 : MIPS Assembly Programming
Assignment 9: Recursion and Nested Function Calls

Instructor: Dr. Sukarn Agarwal,
Assistant Professor,
Department of CSE,
IIT (BHU) Varanasi

April 11, 2021

Recursion

As same as the recursive relation exist, for example calculating the factorial of a given number,

```
n! = n * (n-1) * (n-2) ... *2 * 1 = n* (n-1)!
```

The recursive function exist in the programming languages. The recursive functions is a functions that calls itself repeatedly. The recursive function will keep on calling itself, with different parameters, until a terminating condition is met. It's like writing a loop. You can also write a loop to do something over and over again, until some exiting condition is met.

In the MIPS assembly programming, in the case of writing a recursive function, it is the responsibility of the MIPS programmer to save on the stack the contents of all registers relevant to the current invocation of the function before a recursive call is executed. Upon returning from a recursive function call the values saved on the stack must be restored to the relevant registers.

Nested Function Calls As same as recursive function, a similar case happens when you call a function that can call another function.

```
A: ....
   # Put B's args in $a0-$a3
   jal B # $ra = A2
A2: ....


B: ...
   # Put C's args in $a0-$a3
   # erasing B's args !
   jal C
B2: ...
   jr $ra # where does this go?
```

```
C: ...
   jr $ra
```

Consider the above example, that have function `A` that calls `B`, which calls `C`.

- As observe in the above code, the arguments for the call to `C` would be placed in `$a0-$a3`, therefore overwriting the original arguments for `B`.
- Similarly, `jal C` overwrites the return address that was saved in `$ra` by the earlier `jal B`.

Register Spilling These cases incurs due to MIPS machine has a limited number of registers for use by all functions, and it's possible that several functions will require the same register for the different purposes. To handle this, we can save important register from being overwritten by a function and restore them after the function completes.

There are two possible ways to save important registers across function calls:

- The caller saves the important register as it knows which registers are important to it.
- The callee knows exactly which register it will use and accordingly it overwrites it.

The caller save the register This can be done by the caller to save the registers that it needs before the function calls. The saved register restore at the later point of time, when the control return back from the callee. However, the problem with this method is sometimes the caller does not know which registers are important for their execution. As a result, it may end up with saving large number of registers.

```
A: li $s0, 2
   li $s1, 3
   li $t0, 1
   li $t1, 2

   # Code pertaining to save the register $s0, $s1, $t0, $t1

   jal B

   # Code pertaining to restore the register $s0, $s1, $t0, $t1

   add $v0, $t0, $t1
   add $v1, $s0, $s1
   jr $ra
```

In the above example, function `A` saves the register `$s0, $s1, $t0 and $t1` before jump to the procedure `B`. However, it may be possible that the procedure `B` may not use these registers.

The callee save the register Another alternative ways is if callee save the value of register before the callee statement starts or before the register is being overwritten. The saved register is restored at the later point of time, when the callee function finishes their execution.

```
B: # Save registers
# $t0 $t1 $s0 $s2

li $t0, 2
li $t1, 7
li $s0, 1
li $s2, 8
...

# Restore registers
# $t0 $t1 $s0 $s2

jr$ra
```

For example, in the above code, function `B` uses register `$t0, $t1, $s0, $s2`. Hence, before using them, the callee procedure save the original values first. Thereafter, restore them before returning. However, as same as the case with the caller, the callee does not know what registers are important to the caller. As a result, it may again end up with saving more number of register.

The caller and callee work together To overcome the scenario that leads to saving more number of registers. MIPS machines uses conventions again to split the registers spilling chores.

- The caller is responsible for handling it **caller saved registers** by saving and restoring them. In other words, the callee may now freely modifying the following set of register, under the assumption that caller already saved them before jumps to callee.

  `$t0-$t9 $a0-$a3 $v0-$v1`

- The callee is responsible for handling it **callee saved register** by saving and restoring them. In particular, the caller now assume that these following set of registers are not altered by the called. Note that the register `$ra` is handled here carefully; as it is saved by a callee who may be caller at some point of time.

  `$s0-$s7 $ra`

Hence, with register spilling, be careful when working with nested functions, which can act as both caller and callee.

Problem 1: Write a MIPS assembly recursive function to compute N factorial. The value of N is supplied by the user on the console.

Problem 2: Write a MIPS assembly recursive function to find the determinant of a 3 x 3 matrix (it can be integer or floating point array). The address of the array M and the size N are passed to the function on the stack, and the result R is returned on the stack:

Problem 3: Generate the equivalent MIPS assembly code for the following C++ code:

```
int add(int num1, int num2)
{
return (num1 + num2);
}

int main()
{
int num1 = 12, num2 = 34, num3 = 67, num4 = 12;

cout << add(add(add(num1, num2), num3), num4);

return 0;
}
```

Problem 4: Write an efficient MIPS assembly language function that will scan through a string of characters with the objective of locating where all the lower case vowels appear in the string, as well as counting how many total lower case vowels appeared in a string. Vowels are the letters a, e, i, o, u. The address of the string is passed to the function on the stack, and the number of vowels found is returned on the stack. A null character terminates the string. Within this function, you need to call other function that Justifies the relative position within the string where each vowel was found. Notice this will be a nested function call (calling of a function inside the function). Here is an example string:

```
The quick brown fox.

For the above example string the output of your program would be

A Vowel was Found at Relative Position :                3
A Vowel was Found at Relative Position :                6
A Vowel was Found at Relative Position :                7
A Vowel was Found at Relative Position :               13
A Vowel was Found at Relative Position :               18
```

Note: Submit all of your source code and the final screenshots of the console and register panels (both integer and floating-point) to the google classroom portal by the end of the day of 20 April 2021 (Indian Standard Time). Further,

any copy case between the assignments results in zero marks. In case of any doubt(s) regarding the assignment, you can contact TA: Nirbhay and Deepika.