컴퓨터구조 PA1

Code Write-up

2019-11563
자유전공학부
김지원

[Digit Sum Problem]

Operands are saved in a0 and a1. In the ‘digitsum’ part of the code, first I moved one operand from a0 to a3 since a0 is going to be used to store the total sum (the result). After the move, a0 is initialized to 0 and also, create an additional copy of the total sum at t0. Also, initialize t1 to 10 since we’re going to use it when calculating the digits in the later part of the code.

In the lhs_loop, we check if a1 is zero, which means the calculation for lhs is done and we need to branch to the rhs_loop. If not, we proceed to calculate the digit sum of lhs. First do lhs%10 to calculate the digit and add this to t0(total sum). After, do ‘lhs = lhs/10’ and repeat the lhs_loop.

In the rhs_loop, the procedures are exactly same as the lhs_loop. Except at the beginning we compare the rhs operand with 0 and if it is 0, we proceed to ‘done’.

After calculating digit sum for both lhs and rhs, we move to ‘done’ part of the code and copy t0 (total sum) to a0 register and return.

[Fibonacci Problem]

In the first ‘fibonacci’ part of the code, we first adjust the stack pointer to store ‘n(stored in a0)’, ‘t1(later in the code this will store the result of fibonacci value we need to store temporarily)’, and ra(return address).
And we’re going to initialize t0 to 2 and load ‘n’ value from the stack to a1. Compare a1 and t0 which is basically comparing if ‘n>2’, if so, we proceed to the ‘not_base_case’ code. If not, it means n<=2 and it is the base case, we return 1 on a0, restore the ra and restore the stack pointer up 12, and then return to the caller.

In the ‘not_base_case’, it first calculates Fibonacci (n-1) and then, Fibonacci (n-2) sequentially. First, we load ‘n’ from the stack to a0. And decrement 1 from a0 (n-1). After, we call the Fibonacci again for recursive call. Since we use ‘jal’ instruction, after processing all the ‘fibonacci’ part, it will return to the next line which will save the result of the calculated Fibonacci (n-1) onto the stack (4(sp)).
In the Fibonacci (n-2) part, we first calculate n-2, and then do the recursive call for Fibonacci (n-2) just like we did before. After, we load the value of Fibonacci (n-1) to a1 register and calculate

Fibonacci(n-1) + Fibonacci(n-1) and save this value at a0. We restore the return address and stack pointer and return to the caller.

[Tree Sum Problem]

First, we initialize necessary variables. t0 will hold the total sum value, t2 will hold the head (at first initialize as a1 since it's the beginning of the queue), and t1 will hold the cur value which will be a1+12 (since we're going to store three values for each tree node. Then, we will first copy the root node onto the queue using t3, t4, t5.

In the 'loop', we first check if head != cur. If not, we load the node at head from the queue. We use a1 register to store the value of the node, and sum it to t0. After, we load the left child pointer to a0 register. If this value is 0 (means the node is null), we branch to 'skip_left'. If not, we proceed to copy the value, left pointer, right pointer of this tree node onto the queue using t3, t4, t5. After copying the values, we need to increment the cur(t1) value by 12 since we've stored 3 values.

In the 'skip_left' part of the code, we proceed to look at the right child. We copy the pointer of right child to a0 and check if it's null. If not, we do the same procedure as before, copying all the necessary information onto the queue using t3, t4, t5 and then incrementing the cur(t1) by 12.

After, the code proceeds to 'skip_right' part. In this part since we've looked at both the left and right child, we move the head(t2) by 12 and then repeat the loop to look at the next node in the queue. After looking at all the nodes in the queue (head == cur) we proceed to 'done' part where it moves the final result from t0 to a0 and returns.