ECSE 324 - Lab 2 Report - Group 46

Jessica Li (260665795)
Claire Liu (260654285)

Discussion on Approaches to Each Section:

stack.s:

The goal of this program is to push 3 numbers and pop all of them on a stack without using the PUSH and POP instructions directly. Instead, the PUSH instruction is achieved by loading the number first to R0, then store the content of R0 to the address 4 bits before the stack pointer (R13). After the number is stored, the the stack pointer is updated correspondingly: it now points to the top of the stack and is 4 bits smaller than its original value. POP is achieved by loading the number on the top of the stack to R0, R1 and R2, then update the stack pointer each time by adding 4 to its original value. This ensures the stack is restored and nothing is left on the stack.

subroutine.s:

This program finds the maximum number in a given array using subroutine.

First of all, we push R0 to R4 and LR on the stack then branch link to subroutine 'findmax'. In the subroutine 'findmax', we first push R0-R4 on the stack to callee-save the registers that 'findmax' will use. In 'findmax', a loop is used to scan the entire list and find the maximum. At the end of the 'findmax' subroutine, store the result on stack, restore the registers by popping them off the stack and branch exit to the main branch. After we return to the main branch, we load the return value of the 'findmax' subroutine (which is the greatest number in the list) from stack, store it in memory, restore the link register LR and restore the stack by popping all the elements.

Fibonacci.s:

The goal of this program is to get the nth Fibonacci number in the Fibonacci series, where the nth component is equal to the sum of (n-1)th and (n-2)th components. After choosing the input n manually from the NUMBERS, the program will enter the subroutine called Fib. At the beginning of every cycle (cycle x) of the recursion, we push R0, R2 and LR, which stands for the current (x-1)th and (x-2)th respectively, onto the stack. By setting the end condition to n<2, we store value 1 to R0 and R2. Then, by popping the previous value from the stack, the link register will enter the next line of the caller from the previous cycle. Then, we add the R0 and R2, and store it to the new R0 and store the previous R0 to the new R2. Finally, after the link register quit the subroutine, we store the final R0

to the specific address in memory, which is contained in R1. By convention, we pop all the value from the stack for restoration.

pure_C.c:

The goal of this program is to find the maximum number in a given list using pure C-programming. We store the first element of the list in variable 'max_val', then a for-loop is constructed. During each iteration, the current element is compared with the value of max_val. If the current element is greater than max_val, then it replaces max_val. This process continues until all elements in the list are checked and max_val is returned in the end.

subroutineC.s:

This assembly program compares two numbers and returns the larger number.

subroutineC.c:

This program identifies the greatest number in a given list using the MAX_2 method constructed in subroutineC.s. This C-program is linked to the assembly program through instruction 'extern int MAX_2(int x, int y)'. A for loop which runs n-1 times (where n is the length of the given list) is used to fetch two neighbouring numbers from the list and the MAX_2 function is used in the loop to find the larger number among the two neighbouring numbers. The greatest number is stored in variable 'e' and returned in the end.

Potential Improvements:

1. For subroutine.s, instead of pushing all registers on the stack (which accesses the memory and is relatively expensive), we can only push R4 and LR on the stack so the code is more efficient.
2. For Fibonacci.s, instead of using recursion two times, like indicated in the lab instructions, we can do one recursion and compute both fib(n-1) and fib(n-2) in that recursive cycle. The instructions told us to compute fib(n-1) first recursively, then compute fib(n-2) recursively, and finally add them together. This is expensive, however fibonacci computation using a one-time recursion algorithm is much more efficient.

Challenges Encountered:

Some of the main challenges we have encountered include understanding the fundamental concept of stack, PUSH and POP.

In the beginning, we had trouble understanding why using the stack is necessary for subroutines. We thought pushing and popping registers do not affect the registers' value because we were trying to understand how the stack works from our past programming experience in Java. Since in Java, the variables are not re-assigned with values when they are been popped. We figured out this problem by realizing that registers' values are updated with the values that were saved on stack when they are popped from the stack. From there, we also gained a deeper understanding for the callee-save convention and the importance of stack restoration.