ECSE324 COMPUTER ORGANIZATION

Lab2 Report

Group 29

Yi Zhu

260716006

Shaluo Wu

260713923

2018-10-24

Lab2 consists of five parts, three of them mainly focus on stacks and subroutines: the stack program, subroutine calling convention and Fibonacci calculation using recursive subroutine calls. Two focus on programs in C and calling ARM assembly in C: pure C program and calling an assembly subroutine from C. This report briefly introduces the approach used challenged met and improvements for each program.

1. Subroutines

1.1 The stack

The stack is a data structure that can help to store data when there aren't enough registers for a program. Two basic stack operations are PUSH and POP. In ARM assembly language, user can directly call PUSH and POP operations to accomplish stack operations. Besides on, pushing and popping can also be implemented in a traditional way by using STR and LDR operations. To do stack operations in this traditional way, we will use the Stack Pointer (SP), which is a register that stores the memory address of the top item in the stack. In the RISC architecture, R13 register is generally used as the SP.

Since the stack structure follows the Las-in-First-out principle, to PUSH R0, we will put R0 at the top of the stack and then update the SP to point to it. The ARM code will look like STR R0, [SP, #-4]! which means to store the content of R0 in the 4-memory address less than the current address of SP and move SP to point at this new address. To obtain POP we use LDR R0, [SP], #4 ARM code which means to load content of SP in R0 and let SP point to the next register.

In the test program, we initialized three registers R0 R1 and R2 with values 4, 5, -3. Then we implemented PUSH and POP instructions for these three values by steps in the traditional way. The relatively challenging problem in this part of lab is to figure out the difference between using each memory addressing modes such as when to use '!' and where to put '[]' and what is the specific meanings for these symbols in ARM code.

There will still be space for improvement that we can use STMDB and LDMIA instructions to store and load multiple elements into the stack at the same time to increase the efficiency.

1.2 The subroutine calling convention

In this part we are asked to convert the 'Find the max of an array' program from Lab 1 into a program which has a main method and uses a subroutine to call this 'Find Max' method. A subroutine is a reused block of code in a program. It can be called from anywhere in a program and we should be able to pass parameters in it. A subroutine must also be able to return a value.

The general idea of this code is to initialize all registers that will be used in the main method and then call a subroutine named MAX to return the max value of the array and store it in the memory. At the start point of the code, R4 will be used to store the result in the memory, R2 holds the number of elements in the list, R3 points to the first number location, and R0 holds the value of the first number in the array. Then we PUSH R4 and the link register (LR) and call the subroutine MAX. Now the LP will point to the next line after the call of subroutine instruction. In the subroutine, it is basically the same method to find the maximum value in an array. At the start point of the code, R0 holds the first number in the list which will also be used as a result. The first element of the array will be compared with each element after it, if any element is greater than the first element, the first element will be replaced by the greater one. Otherwise, move to next element to compare it with the first one. After the counter reaches zero, R0 holds the maximum value of the array. We branch back to the main method and then store the value into R4. Finally, the value of the max is stored in R4.

The challenge we encountered in this part of lab is how to use stack when we are calling a subroutine. But after we compared the calling method in C language and the use of stack in subroutine, we figured out when and how to use link register and how POP and PUSH work. Then our problems are solved.

Since most part of the code has been done in previous lab, we do think there is further space for improvement.

1.3 Fibonacci calculation using recursive subroutine calls

In this part of the lab, we are asked to create a Fibonacci sequence using subroutine in ARM. The general idea of the code is to using recursion method. We considered the C language provided in the lab manual and then tried to convert it into ARM language. At the start point of the code, we created the main method and initialized R0 to be the input number of the sequence we want to get. Then we call the FIB subroutine. In the FIB subroutine, we first compare the input number with 1, if it is greater than 1, we will go to the REC which is a recursion method to calculate the result. If it is less than or equal to 1, we will set the result to be 1 and branch back to the main method. In the REC, we first set R1 to be the number of the result that we want and use R1 as a counter. We will decrease the counter each time and store the result in R0 and branch back to the main method to check whether it reaches 1. If it does not reach 1 we will load the previous value into R1, and sore it into stack. We also used R2 as an intermediate value for Fib(n-1). Now R2 will be the Fib(n) and after decrease R0 by 2 it will be Fib(n-2). We also will check R0 whether it reaches 1 now. Then unless Fib(n-2) reaches1, we add Fib(n-2) and Fib(n-1) together to get the new Fib(n) value and load it into R0. After the registers in stack have all been pop out in REC, we will branch back to the main method. At this point, R0 will hold the final result of the Fibonacci sequences. Then we will store the final result into R4 in the memory.

The biggest challenge in this part will be thinking about the recursion implementation in ARM assembly and using subroutine. We have considered the example given in tutorials and problem sets and generated the recursion method.

For the improvement, we also came up with another solution of recursion method for the Fibonacci sequences. Instead of dismantling each element to one and using the counter from high to low, we can also use the upgrowing counter and add Fib(n) = Fib(n-1) + Fib(n-2) sequentially.

2. C Programming

2.1 Pure C

The main code of this C language program is given on myCourses. We have been given an array with 5 elements. The logic to find the maximum of an array using C language is the same as we did in ARM language that we used a for loop to compare the first element with all other element in the array. If the first one is greater than next one, then keep it. If the other one is greater, we will replace the first one by the greater one. Otherwise, move to next element for comparison. We wrote the for loop using i as a counter which is initialized to be 0. An integer variable max_val to store the maximum value in the array. At the end of the main class, it will return the max_val as the result.

Since this a very basic program we wrote several times, we met no challenges and there is nothing much to improved.

2.2 Calling an assembly subroutine from C

Instead of implementing a for loop to find maximum value in the C language as we did in 2.1, we will use the EXTERN instruction to call an ARM assembly language project MAX_2 in our C program to find the maximum value in an array. The main class of the C language code and whole part of ARM code MAX_2 have been provided on myCourses. The MAX_2 program has the same working mechanism as all previous finding the greatest value programs.

This program is quite basic, and it is a learning process, we do no think there is further we can improve.

3. Conclusion

In this lab, we got more familiar with implementing subroutines in ARM language and the use of stack while calling a subroutine. We also have a better understanding on how to implement a recursion method in ARM language. We learned that we can also write C language program in Intel FPGA Monitor Program and we can also call a global ARM program from a C language program using EXTERN instruction.