G21_Lab2_Report

Dong Han, Yutian Jing

The approaches for Lab 2 mainly focus on using stacks and subroutines to write programs, and writing a program in C that calls a subroutine program written in ARM assembly.

1. Subroutines

A subroutine program usually uses stack, either to save registers when there are only limited numbers of registers, or to store the state of the caller code when calling subroutines.

1.1 The stack

The stack operations include push, pop and peak. In ARM assembly language, there exists PUSH and POP operations that enables programmers to operate stack directly. Despite of using PUSH and POP operations, pushing and popping can also be implemented in ARM assembly by using STR and LDR operations. The R13 register is considered as SP, the stack pointer, which is a register who stores the memory address of the top of the stack.

Running PUSH R1 would have the same result as running STR R1, [SP, #-4]! since storing R1 to the SP-4 memory location is the same with PUSH R1. After storing, the SP value is also changed due to the use of !.

Similarly, LDR R1, [SP, #4] will have the same outcome with POP R1, as R1 is loaded out and the SP value has gone down by 4 after the operation.

The test program includes the initialization of the R1 value and the two instructions after. Running the two instructions by steps, the changing value of SP and R1 can be seen on the register panel.

1.2 The subroutine calling convention

This part requires the “Find the max of an array” program to be written in a subroutine manner. At the start of the code, 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. After that, the value of LR is pushed onto the stack by the STR LR, [SP, #-4]! instruction, and then a BL subroutine. The LR value now is the line after the BL instruction.

In the subroutine, it is similar with the original finding max program which used a loop to find max.

First there is SUBS R2, R2, #1 and BXEQ LR, using R2 as the loop counter, then move the pointer to next number and compare with the first number, if larger then store it to R0, if smaller then branch back to the start of the subroutine. Once the counter reached zero, BXEQ LR let the code branch back to the caller code, and the caller code just ends. Finally, the value of the max stored in R0 can be viewed on the register panel of the Intel FPGA Monitor Program.

1.3 Fibonacci calculation using recursive subroutine calls

The basic idea of the part is to use the recursive method to find the fib(n).

First, let R1 hold the value of Nth fib number and move 0 to R0 which initialize the result.

Push LR,R1,R0 on the stack which push parameters to the subroutine and link register.

Then, call subroutine 'FIB'.

In the subroutine 'FIB', first, we push R0 to R2 on the stack. These are registers we are going to use in the subroutine. Compare N(R1) with 2. If R1 is smaller than 2, return 1 in the result. Otherwise, find fib(n-1) and fib(n-2)

First, for fib(n-1), we subtract R1 by 1 and than push R0 R1 and LR on the stack (notice that R1's value is updated(N-1)). Then, recursive call FIB itself until R1 reach 1 in which case, goes to 'SMALL'. In 'SMALL', R0 become 1 and goes to branch to 'DONE'.

In DONE, store the value of updated R0 and replace the original R0([sp, #12]).

Then, pop R0 to R2 out of the stack and return back to the link register which is under fib(n-1) part. Load the value of R0 to R2.R2 hold the value of fib(n-1).

After this, we are going to find the value of fib(n-2). First, subtract 1 from R1. The reason not subtract by 2 is R1 is already updated on the stack. Thus R2 = (N-1)-1. Similarly as before, push R0 R1 and LR on the stack than call FIB itself. This time, it goes to 'SMALL' since R1 is smaller than 2. Same thing as before, move R0 to 1 and replace the previous R0 value on the stack. And return to the link register. However, this time, it doesn't return to fib(n-1) but the LR under fib(n-2). From that, pop LR, R1,and R0 out of the stack. Then, add R2 and R0 value to R0. And store new R0 value to the previous R0 on the stack. Then, it return back to LR under fib(n-1). Then pop R0, R1 and LR. And store the new R0 into R2. R2 is updated. Then, goes to fib(n-2) and call FIB itself again. The same routine keeps happening until all registers on the stack from FIB are popped out. Finally, from 'DONE' return back to 'start'. Pop out the R0, R1 and LR which are pushed at the very beginning. Then store R0 value in the result. Actually the actual adding part is happening while popping out the register and keep updating the value in those registers.

The challenge is to use subroutine to find the recursive method of fib(n). At first, we use the loop method which is finding fib(n) forward. However, the recursive method requires the subroutine to call itself recursively which means to find fib(n) backward. This is the most difficult part. We believe that it is cruitial to know how the stack and subroutine work in order to complete this part.

2. C Programming

2.1 Pure C

The logic to find the maximum of an array using C is also using loops. A for loop like the following:

```
for (i = 0; i<5; i++) {
        if(a[i+1] > max_val) {
        max_val = a[i+1];
        }
}
```

showed the logic to iterate through an array “a” with 5 elements to update the max_val.

In the for loop, if max_val is smaller than a[i+1], update the value of max_val to be the value of a[i+1].

2.2 Calling an assembly subroutine from C

Instead of using if statements, the assembly subroutine is called in the for loop to update the current maximum.

```
for (i = 0; i<5; i++) {
        max_val = MAX_2(max_val,a[i]);
}
```