Experiment 4: Stack and Subroutine

16.11.2018
Res. Asst. Abdullah Cihan Ak
akab@itu.edu.tr

Denebilir ki, hiçbir şeye muhtaç değiliz, yalnız tek bir şeye çok ihtiyacımız vardır; çalışkan olmak. Emrazı içtimaiyemizi tetkik edersek asıl olarak bundan başka, bundan mühim bir maraz keşfedenleyiz, maraz budur. O halde ilk işimiz bu marazı esaslı surette tedavi etmektir. Milleti çalışkan yapmaktır. Servet ve onun neticei tabiiyesi olan refah ve saadet yalnız ve ancak çalışkanların hakkıdır.[1]

Mustafa Kemal Atatürk, 20 Ocak 1923

1 Introduction

This experiment aims to enhance the practical experience about function calls and usage of the stack. Students are recommended to check Stack operations and also the differences between **CALL** and **JMP** instructions from the documents on Ninova.

2 Part 1

Simple program which consist of several function calls is given below. The given program takes an array of 8-bit integers and changes their signs by using 2's complement method and stores them to a memory location. Run the following program in order to observe and understand the basics of function call mechanism in microcomputer systems.
You should run the given program step by step and fill Table 1 for the first iteration of the *Mainloop* by using Debug Mode of CSS. Also, please add this table to your report with sufficient explanations.
Note: Content of stack should include all the stack, not only the top element.

[1]Atatürk'ün Söylev ve Demeçleri, 2006, 5. Baskı, Cilt II, Sayfa: 61

```
Setup           mov #array , r5
                mov #resultArray , r10

Mainloop        mov.b    @r5,r6
                inc r5
                call #func1
                mov.b r6,0(r10)
                inc r10
                cmp #lastElement , r5
                jlo Mainloop
                jmp finish

func1           xor.b #0FFh, r6
                mov.b r6,r7
                call #func2
                mov.b r7,r6
                ret

func2           inc.b r7
                ret

;Integer array
array           .byte 127, -128,0,55
lastElement

finish          nop
```

Additionally, you should include the following line above *.text* section in your main.asm to define uninitialized memory location where the program stores the output.

```
result          .bss       resultArray ,5
```

As seen in the given program, there are two functions defined such as **func1** and **func2**. The first function uses **R6** to take its input and return the related output. In the same manner, the second function uses **R7**. On the other hand, main program uses **R5** and **R10** in order to store addresses of input and output arrays.

Assume that, there is a collusion in the utilization of registers between he defined functions and the main program. As an example, you could assume that the main program uses **R6** and **R7** to store the address information of the arrays. In the given scenario, any call of **func1** or **func2** will disorder the expected behaviour of the main program since the values of **R6** and **R7** would be overwritten. Thus, a programmer should prevent the loss of any necessary information while using nested functions. In fact, you may have used predefined functions of which you do not know the body (which registers are used in the function) to accomplish some task, how could the loss of information be avoided? Provide a solution to the given problem other than using different registers and include your solution to the report.

Code	PC	R5	R10	R6	R7	SP	Content of the Stack
mov #array, r5							
mov #resultArray, r10							
mov.b @r5,r6							
inc r5							
call #func1							
xor.b #0FFh, r6							
mov.b r6,r7							
call #func2							
inc.b r7							
ret							
mov.b r7,r6							
ret							
mov.b r6,0(r10)							
inc r10							

Tablo 1: Fill in the blanks for the first iteration of the loop

3 Part 2

In this part, you are required to implement subroutines, named *Add*, *Subtract*, *Multiply*, *Divide*.

- Add(a,b)= a+b
- Subtract(a,b)= a-b
- Multiply(a,b)= a*b
- Divide(a,b)= a/b (division between integers, ex: 5/2= 2)

When implementing these functions, you are required to pass the parameters of the method **through the stack** and retrieve the result in the same manner. You may check the *Assembly_language_tutorial* from Ninova for information on how to pass parameters to a subroutine by means of the stack and how to retrieve the result. You may use registers for parameter passing and retrieving the result for deducted points.

4 Part 3

In this part, you are required to implement a subroutine that calculates Combination. As you can guess you should also implement a subroutine that calculates Factorial to calculate Combination.

- Combination(n,k);
- Factorial(n);

Factorial subroutine should be implemented as recursive and in both Factorial and Combination, subroutines implemented in Part 2 should be used for mathematical operations. Parameters and result should pass through the stack.

5 Part 4 (Optional)

This part is optional for implementation but mandatory to discuss in the report. You should realized that division subroutine in Part 2 loses some information because of the integer division. You can not solve this problem since float numbers can not be represented in MSP430 using its features. Propose a solution to solve this problem and implement a division subroutine that can return results for float numbers. Feel free to imagine and try different solutions.