Experiment 4

Sevim Eftal Akşehirli, 150190028
Hacer Yeter Akıncı, 150200006
Büşra Özdemir, 150200036
Aslı Yel, 150200054

***Abstract*—This experiment is about understanding how functions and stacks work on microcomputers, focusing on the MSP430. run a program step by step to see how functions are used, how memory is managed, and how information is transferred between different parts of the program. A table was filled out showing which parts of the program used certain portions of the computer's memory.**

***Index Terms*—CALL, JMP, Stack, Memory, Recursion, Recursive Subroutine,**

I. Introduction

Knowing how functions work and using the stack is important when working with microcontrollers. This experiment is all about giving hands-on experience in these areas. It's key for us to understand how the stack works, how CALL and JMP instructions are different, and how they affect how programs run on the MSP430 microcontroller.

II. Materials and Methods

A. Part 1

Part 1 of the task requires filling in a table with values corresponding to the initial iteration of a program or process. It involves recording specific data, likely variables, or memory states, at the beginning or after the first cycle of execution. This helps capture the initial state of the system and how values change or progress during the program's initial run.

```
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    dec.b  r6
         mov.b  r6, r7
         call   #func2
         mov.b  r7, r6
         ret

func2    xor.b  #0FFh, r7
         ret

;Integer array,
array .byte 1, 0, 127, 55
lastElement

finish   nop
```

Setup: It sets memory addresses for data storage. It puts the address of an array called "array" into register R5 and the address of another array named "resultArray" into register R10. These lines help locate where the data is stored in the computer's memory, readying it for processing or storage during the program's run.

Mainloop: This code operates on an array. It retrieves data from the array at a location indicated by R5, calls "func1", and then saves the updated data in resultArray at R10 index. This process continues until it reaches "lastElement", marking the end of the loop. It then jumps to the "finish" branch to end the program.

func1: This function decreases a value in register R6 and stores in R7, sends it to another "func2,". It assigns R7 to R6. Then return from the call.

func2: The function called, there's a bitwise XOR operation performed on the value stored in register R7 and the value 0xFF. The operation flips all the bits in R7. Then return from the call. If we examine func1 and func2 together. These two functions calculate the component of the value decreased by one. In the main loop, performs this operation for all elements in array and writes the resultArray.

B. Part 2

this task involves creating three subroutines: multiply, div (division), and power. Each subroutine is designed to take two word-sized parameters, perform its operation, and return the result. The power subroutine relies on the multiply subroutine for its computation. To manage parameters and results effectively, the subroutines use the stack. Registers are saved at the start of each subroutine by pushing them onto the stack and then restored by popping them before the subroutine ends. This ensures the integrity of register values throughout the subroutine execution.

```
Main     mov    #9d, R4
         push   R4
         mov    #3d, R5
         push   R5
```

```
        call   #Division
        ;call  #Multiply
        mov    #2d, R10
        mov    #3d, R11
        push   R10
        push   R11
        call   #Power

Multiply  pop    R13; return address
        pop    R5
        pop    R4
        push     R13
        mov    #0d, R6
L2        cmp      #0d, R5
        jeq      Return
        sub    #1d, R5
        add      R4, R6
        jmp    L2

Power     pop    R13
        pop      R8
        pop    R7
        push   R13
        push   #1d
mul       cmp    #0, R8
        jeq    Return
        push   R7
        sub    #1d, R8
        call   #Multiply
        jmp    mul

Division  pop    R13; return address
        pop    R5
        pop    R4
        push     R13
        mov    #0d, R6
L1        cmp      R5, R4
        jlo      Return
        sub    R5, R4
        add      #1d, R6
        jmp    L1


Return      ret
```

Main: The code starts by initializing the R4 with 9 and R5 with 3. Then, it pushes these register onto the stack. Then it can be called either Division or Multiply. After that, it pushes the initialized registers R10 and R11 for power operation.

Multiply: The return address is stored in R13. The reason is to not lose it because of popping other variables from stack. It then enters the L2 loop and adds R4 to the R6 result register R5 times.

Power: It begins by saving the return address and collecting parameters from the memory. Then, it puts the return address back where it was taken from. Also, it adds the number 1 to the stack, start value of result.

mul: It checks if R8 ,the exponent, is zero. If it is, it stops the loop. If not, it saves a value from R7, reduces the number in R8 by 1, calls "Multiply," and repeats the loop. This continues until R8 becomes zero.

Division: It gathers the return address and the numbers needed for division from the memory stack. It then saves the return address and sets up a register to store the result of the division operation.

L1: It keeps subtracting the value in R5 from R4 until R5 becomes less than R4. During each turn, it counts and adds one to R6. This way, we find the R4/R5 and stores in R6.

C. Part 3

Since we could not finish Part 2 properly, we did not start implementing Part 3.

III. Results

Our experiment with the MSP430 microcontroller provided us with valuable insights into how function calls, memory management and stack utilization operate. The table we created in Part 1 highlighted specific memory locations involved during the program's start, showing us how data interacted within different parts of the program. In Part 2, we implemented subroutines for multiplication, division, and power calculations. These subroutines used the stack to handle parameters and results, demonstrating various computational methods like repeated addition and subtraction to perform arithmetic tasks. Additionally, we observed how the power subroutine relied on the multiply subroutine, showing us how functions can work together in this microcontroller environment. Overall, our findings emphasized the crucial role of executing functions, managing memory, and handling the stack in guiding program flow and managing resources in microcomputer systems.

IV. Discussion and Summary

In our MSP430 microcontroller experiment, we had a problem in Part 3, as things didn't go as we expected. Still, we learned a lot. In Part 1, we made a table that showed how different parts of the program used the computer's memory at the beginning. Part 2 was about making small programs for math like multiplication and division. We found out that these programs used the computer's memory in different ways and sometimes worked together. For example, one program used another to do some math. This experiment showed us how computers manage information and do different tasks. Even though we had a problem, we learned a lot about how computers work with memory and do math.

Additionally, the problems we experienced in the stack show us that it is important to store the return address correctly. We

have observed that pushing variables onto the stack can create problems when returning from the call.