Department of Computer Engineering

BLG 351E
Microcomputer Laboratory
Experiment Report

Experiment No : 4

Experiment Date : 11/14/16

Group Number : M - 13

Group Members :

ID	Name	Surname
150130032	Baran	Kaya
40130051	Halil İbrahim	Onuş
150130002	Ahmet Seha	Çelenk

Laboratory Assistant : Gökhan Seçinti

1 INTRODUCTION

In this experiment, we have shown the difference between JMP and CALL. We have used CALL with stack pointer. We have worked on modulo finding algorithms.

2 EXPERIMENT

2.1 PART 1 - BASICS OF A SUBROUTINE CALL

In the first part we have used the given code. This program is used to check whether we can use CALL functions and stack pointers or not. This code was converting numbers using 2's complement method. The code is shown below:

```
result      .bss resultArray ,5
            .text                           ; Assemble into program memory.
            .retain                         ; Override ELF conditional linking
                                            ; and retain current section.
            .retainrefs                     ; And retain any sections that have
                                            ; references to current section.

;-------------------------------------------------------------------------
RESET       mov.w   #__STACK_END,SP         ; Initialize stackpointer
StopWDT     mov.w   #WDTPW|WDTHOLD,&WDTCTL  ; Stop watchdog timer


;-------------------------------------------------------------------------
; Main loop here
;-------------------------------------------------------------------------
Setup    mov #array , r5 ;use r5 as the pointer
         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, we have used array definition to store the results on the right memory locations.

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

In this part we have filled this table to check how the program is going. This table is filled through the data flowing while the program is running.

Code	PC	R5	R10	R6	R7	SP	content
mov #array, r5	C00E	C038	DFFF	FFE4	BEFB	0400	0000
mov #resultArray, r10	C012	C038	0200	FFE4	BEFB	0400	FFFF
mov.b @r5,r6	C014	C038	0200	007F	BEFB	0400	FFFF
inc r5	C016	C039	0200	007F	BEFB	0400	FFFF
call #func1	C028	C039	0200	007F	BEFB	03FE	C01A
xor.b #0FFh, r6	C02A	C039	0200	0080	BEFB	03FE	C01A
mov.b r6,r7	C02C	C039	0200	0080	0080	03FE	C01A
call #func2	C034	C039	0200	0080	0080	03FC	C030
inc.b r7	C036	C039	0200	0080	0081	03FC	C030
ret	C030	C039	0200	0080	0081	03FE	C01A
mov.b r7,r6	C032	C039	0200	0081	0081	03FE	C01A
ret	C01A	C039	0200	0081	0081	0400	FFFF
mov.b r6,0(r10)	C01E	C039	0200	0081	0081	0400	FFFF
inc r10	C020	C039	0201	0081	0081	0400	FFFF

2.2 EXPERIMENT - PART 2

In this part we are expected to prepare a code to calculate modulo of some numbers divided by others. For this part we have used Modulo Algorithm pseudo code given below:

```
1: procedure MODULO(a, b)
2:    while (a ≥ b) do
3:    a- = b;
4:    end while
5:    return a;
6: end procedure
```

To obtain this code in assembly, we wrote down the following code:

```
27 Setup     mov #12, r6 ;use r5 as the pointer
28           mov #5 , r5
29           push r5
30           push r6
31
32 Mainloop pop r6
33           pop r5
34           cmp r5,r6
35           jge fCall
36           jmp finish
37
38 fCall     push r5
39           push r6
40           call #funcMod
41           push r5
42           push r6
43           jmp  Mainloop
44
45 funcMod   sub r5, r6
46           ret
47
48
49 finish nop
```

In this code, first we use `r6` and `r5` to hold the given numbers. This example ran 12/5 division. This division ends up with 2 on hand.

Pushing these 2 numbers into stack gives us the ability of saving space and using stacks.

We continue with `Mainloop`. In this loop, we pop these 2 numbers and compare them if we divided them enough times. If we jumping to `finish` means we have reached the limit, we have made the division and we got the lowest number as modulo. Otherwise, if we have not reached this yet and if we jump to `fCall` we push the popped numbers, we divide then using `funcMod` and we go back to `Mainloop` to check again.

This loops are ran till we reach a module smaller than the value on `r6`.

2.3 EXPERIMENT - PART 3

Due to lack of time, we could not run this part correctly. We were expected to run modulo algorithm in iterative way. But we could not complete,

3 CONCLUSION

In this experiment, we were expected to use stack pointer as musch as possible. In order to increase performance, we have used function calls and returnings rather than jumps. This experiment was hard in some ways likely the pointers in C programming. But somehow we have solved the problems except the last part.

Again, the board belong to Table 13 was broken and we have used another one in this experiment.