Department of Computer Engineering

BLG 351E
Microcomputer Laboratory
Experiment Report

Experiment No : 4

Experiment Date : 18.11.2016

Group Number : Friday - 3

Group Members :

ID	Name	Surname
150140142	Gamze	Akyol
150130136	Mehmet Barış	Yaman
150130109	Güllü	Katık

Laboratory Assistant : Çağatay Koç

1 INTRODUCTION

In this experiment we examined a simple assembly program to observe the values taken by stack pointers and registers and wrote 2 assembly programs mainly using stacks.

2 EXPERIMENT

2.1 PART 1 – BASICS OF A SUBROUTINE CALL

In this part of the experiment, a given program is run to understand the subroutine call procedure. In each step of the program, given registers' and stack's content is observed. According to this observation, the following table is filled.

When a subroutine is called, content of the stack and stack pointer (SP) is changed, because the return address of this subroutine is pushed to stack. When return to top function, the return address of the subroutine is popped from stack and content of the stack and SP are changed again. Also, program counter (PC) is changed in each step of the program according to number of bits in step.

Code	PC	R5	R10	R6	R7	SP	Content of the stack
mov #array, r5	0xC00E	0xC038	0x0004	0x000C	0x0005	0x0400	0xFFFF
mov #resultArray, r10	0xC012	0xC038	0x0200	0x000C	0x0005	0x0400	0xFFFF
mov.b @r5, r6	0xC014	0xC038	0x0200	0x007F	0x0005	0x0400	0xFFFF
inc r5	0xC016	0xC039	0x0200	0x007F	0x0005	0x0400	0xFFFF
call #func1	0xC01A	0xC039	0x0200	0x0081	0x0081	0x0400	0xFFFF
xor.b #0FFh, r6	0xC02A	0xC039	0x0200	0x0080	0x00C9	0x03FE	0xC01A
mov.b r6, r7	0xC02C	0xC039	0x0200	0x0080	0x0080	0x03FE	0xC01A
call #func2	0xC034	0xC039	0x0200	0x0080	0x0080	0x03FC	0xC030
inc.b r7	0xC036	0xC039	0x0200	0x0080	0x0081	0x03FC	0xC030
ret	0xC030	0xC039	0x0200	0x0080	0x0081	0x03FE	0xC01A
mov.b r7, r6	0xC032	0xC039	0x0200	0x0081	0x0081	0x03FE	0xC01A
ret	0xC01A	0xC039	0x0200	0x0081	0x0081	0x0400	0xFFFF
mov.b r6, 0(r10)	0xC01E	0xC039	0x0200	0x0081	0x0081	0x0400	0xFFFF
inc r10	0xC020	0xC039	0x0201	0x0081	0x0081	0x0400	0xFFFF

2.2 EXPERIMENT – PART 2

In this part of the experiment, modulo function is implemented according to given algorithm.

The program of this part is given below:

```
Setup       mov.b   #11111111b, &P1DIR   ; P1 is set as output to observe
                                          ; the result
Mainloop    push.b  #00001000b          ; Push the a and b numbers
            push.b  #00000101b          ; (parameters) to stack
            call    #mod                ; call the mod subroutine
                                        ; (and push its return address to
                                        ; stack)
            mov.b   R5, &P1OUT          ; R5 is set as the result register
            nop

mod         pop.w   R6                  ; pop the last element of the
                                        ; stack to register R6 (in this
                                        ; step, it is the return address of
                                        ; the mod subroutine)
            pop.b   R10                 ; pop one of the parameters
                                        ; to register R10
            pop.b   R5                  ; pop other parameter to
                                        ; register R5
            push.w  R6                  ; again, push the return address
                                        ; of the subroutine which is in
                                        ; R6 to stack for not losing it
compare     cmp     R5, R10             ; compare the parameters
                                        ; which are in R5 and R10
            jn      subtract            ; if R10 is smaller than R5,
                                        ; jump to subtract
            ret                         ; else, return successfully
subtract    sub.w   R10, R5             ; subtract R10 from R5
            jmp     compare             ; Jump to compare
```

2.3 EXPERIMENT - PART 3

In this part of the experiment we wrote an assembly program that works according to the Euclid Algorithm which is for finding greatest common divisors between two numbers.

```
Setup   mov.b   #11111111b, &P1DIR
        push.b  #20d            ; a in pseudocode
        push.b  #15d            ; b in pseudocode
        call    #euclid         ; euclid subprogram is called

        mov.b   R10, &P1OUT     ; Output will be in R10 after
                                ; euclid subprogram is called
        nop


euclid  pop.w   R7              ; pop the main programs return
                                ; address
loop    call    #mod            ; mod subprogram is called
        clr.b   R14
        cmp     R5, R14         ; is a 0?
        jn      pushing         ; if it is push the resulted values
        push.w  R7              ; Main program address is
                                ; pushed
        ret


mod     pop.w   R6              ; euclid subprogram address is
                                ; held
        pop.b   R10             ; R10 holds b
        pop.b   R5              ; R5 holds a
        push.w  R6              ; euclid program address is
                                ; pushed to return successfully
```

```
compare     cmp     R5, R10     ; a needs to be decreased if it is
                                ; bigger than b
            jn      subtract
            ret

subtract    sub.w   R10, R5     ; subtract b from a
            jmp     compare

pushing     push.b  R10         ; push the new numbers for
                                ; next iteration
            push.b  R5
            jmp     loop        ; new euclid loop is needed
```

3 CONCLUSION

From Part 1, we have learned by observation that the main programs and subprograms addresses are held in Stack pointers and each time we are calling the subprograms, the value of Stack pointer is changed. Stack pointer values are kept on stacks, so each time we push and pop in the program we are simply changing the stack.

From Part 2, the main difficulty was that our program did not return the main, after subprogram is called. After we had worked on this we saw that, we are using “pop.b” and “push.b” function instead of “push.w” and “pop.w”. We know that “.w” functions are worked on 16 bit values instead of 8. Since main and subprograms addresses kept in 16 bits, we should use “.w” functions to hold addresses. We have mainly learned that, we can hold main and subprogram addresses by using functions that have effect on 16 bits of registers.

In Part3, we could not return from subprograms to main programs carefully. We could not write the assembly program correctly in this part. Our error was in the Euclid subprogram, it was not holding the values of registers correctly in each loop. That’s why we had wrong results on that part.