นายธนพล วงศ์อาษา 62010356
นายสุทธิราช ภูโท 62010966

การทดลองที่ 7 การเรียกใช้และสร้างฟังค์ชันในโปรแกรมภาษาแอสเซมบลี

ส่วนการทดลอง *โค้ดเพิ่มเติมดูได้จากไฟล์ที่แนบมา

1. **การโหลดค่าตัวแปรจากหน่วยความจำมาพักในรีจิสเตอร์**
   a. บันทึกผลและอธิบายผลที่เกิดขึ้น ข้อมูลเพิ่มเติมเกี่ยวกับคำสั่ง SWI (Software Interrupt)

```
    .data
    .balign 4          @ Request 4 bytes of space
fifteen:  .word 15     @ fifteen = 15

    .balign 4          @ Request 4 bytes of space
thirty:  .word 30      @ thirty = 30

    .text
    .global main
main:
    LDR R1, addr_fifteen      @ R1 <- address_fifteen
    LDR R1, [R1]              @ R1 <- Mem[address_fifteen]
    LDR R2, addr_thirty       @ R2 <- address_thirty
    LDR R2, [R2]              @ R2 <- Mem[address_thirty]
    ADD R0, R1, R2
end:
    BX LR

addr_fifteen: .word fifteen
addr_thirty:  .word thirty
```

โปรแกรมนี้จะทำการบวกเลข 15 และ 30 เข้าด้วยกันและนำไปเก็บไว้ใน register R0

b. บันทึกผลและอธิบายผลที่เกิดขึ้นเพื่อเปรียบเทียบกับข้อที่แล้ว

```
    .data
    .balign 4           @ Request 4 bytes of space
fifteen:  .word 0       @ fifteen = 0
    .balign 4           @ Request 4 bytes of space
thirty:  .word 0        @ thirty = 0

    .text
    .global main
main:
    LDR R1, addr_fifteen  @ R1 <- address_fifteen
    MOV R3, #15           @ R3 <- 15
    STR R3, [R1]          @ Mem[address_fifteen] <- R3
    LDR R2, addr_thirty   @ R2 <- address_thirty
    MOV R3, #30           @ R3 <- 30
    STR R3, [R2]          @ Mem[address_thirty] <- R2

    LDR R1, addr_fifteen  @ Load address
    LDR R1, [R1]          @ R1 <- Mem[address_fifteen]
    LDR R2, addr_thirty   @ Load address
    LDR R2, [R2]          @ R2 <- Mem[address_thirty]
    ADD R0, R1, R2
end:
    BX LR

@ Labels for addresses in the data section
addr_fifteen: .word fifteen
addr_thirty:  .word thirty
```

โปรแกรมนี้จะทำการกำหนดค่าเริ่มต้นให้ตัวแปร fifteen และ thirty เป็น 0 ก่อนและจึงนำค่าคงที่ 15,30 ที่กำหนดไว้ใน register ไป store ไว้ที่ตัวแปรดังกล่าว แล้วจึงค่อยทำการคำนวณแบบข้อที่แล้ว จุดที่แตกต่างคือโปรแกรมนี้ค่าเริ่มต้นตัวแปรเริ่มที่ 0 แล้วทำการ store ค่าใหม่เข้าไปอีกทีแต่โปรแกรมแรกตัวแปรแต่ละตัวจะมีค่าเริ่มต้นเลย

2. การเรียกใช้ฟังค์ชันและตัวแปรชนิดประโยค

a. คำสั่ง echo $? มีไว้เพื่ออะไร: แสดงค่าที่เก็บภายใน register R0 ออกมาทาง terminal

3. **การสร้างฟังค์ชันเสริมด้วยภาษาแอสเซมบลี**

a. ระบุซอร์สโค้ดใน Lab7_6.s ว่าตรงกับประโยคภาษา C ต่อไปนี้ **int num1, num2**

```
.balign 4
num_1: .word 0

.balign 4
num_2: .word 0
```

b. ระบุซอร์สโค้ดใน Lab7_6.s ว่าตรงกับประโยคภาษา C ต่อไปนี้ **sum = num1 + num2**

```
LDR R0, addr_num_1
LDR R0, [R0]     @ R0 <- Mem[addr_num_1]
LDR R1, addr_num_2
LDR R1, [R1]     @ R1 <- Mem[addr_num_2]
BL sum_func
```

โดยใช้ฟังก์ชัน sum_func ซึ่งมีเนื้อหาภายในคือ

```
sum_func:
    @ Save (Store) Link Register to lr_bu_2
      LDR R2, addr_lr_bu_2
      STR lr,[R2]     @ Mem[addr_lr_bu_2] <- LR

      @ Sum values in R0 and R1 and return in R0
      ADD R0, R0, R1

      @ Load Link Register from back up 2
      LDR lr, addr_lr_bu_2
      LDR lr, [lr]     @ LR <- Mem[addr_lr_bu_2]

      BX lr

  @ address of Link Register back up 2
  addr_lr_bu_2: .word lr_bu_2
```

c. เหตุใดจึงผู้อ่านจึงไม่ต้องใช้คำสั่ง echo $? แล้ว: มีการใช้คำสั่ง printf แล้วซึ่งจะทำการแสดงผลออกทาง terminal ทำให้ไม่จำเป็นต้องเรียกคำสั่ง echo $? เพื่ออ่านค่าใน register R0 อีก

คำถามท้ายการทดลอง

8. จงพัฒนาโปรแกรมด้วยภาษา Assembly เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B และแสดงผลลัพธ์ค่า A modulus B ซึ่งเท่ากับ ค่าเศษจากการคำนวณ A/B ด้วยคำสั่งภาษาแอสเซมบลี

<u>ตอบ</u>

```
    .data
    .balign 4
get_num_1:  .asciz "Number 1 :\n"
    .balign 4
get_num_2:  .asciz "Number 2 :\n"

    .balign 4
pattern:    .asciz "%d"

    .balign 4
num_1: .word 0
    .balign 4
num_2: .word 0

    .balign 4
output: .asciz "%d mod %d = %d\n"

    .balign 4
lr_bu: .word 0
    .balign 4
lr_bu_2:    .word 0

    .text
```

```
mod_func:
    LDR R2, =lr_bu_2
    STR lr,[R2]
mod_loop:
    CMP R0,R1
    BLT end_mod
    SUB R0, R0, R1
    BL mod_loop
end_mod:
    LDR lr,=lr_bu_2
    LDR lr,[lr]

    BX lr

    .global main
main:
    LDR R1, =lr_bu
    STR lr,[R1]

    LDR R0,=get_num_1
    BL printf
    LDR R0, =pattern
    LDR R1, =num_1
    BL scanf

    LDR R0, =get_num_2
    BL printf
    LDR R0, =pattern
    LDR R1, =num_2
    BL scanf
```

```
    LDR R0, =num_1
    LDR R0, [R0]
    LDR R1, =num_2
    LDR R1,[R1]
    BL mod_func

    MOV R3, R0

    LDR R0, =output
    LDR R1, =num_1
    LDR R1, [R1]
    LDR R2, =num_2
    LDR R2, [R2]
    BL printf

    LDR lr, =lr_bu
    LDR lr,[lr]
    BX lr

.global printf
.global scanf
```

9. จงพัฒนาโปรแกรมด้วยภาษา Assembly เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณหาค่า หารร่วมมาก (Greatest Common Divisor) หรือ หรม (GCD) ด้วยคำสั่งภาษาแอสเซมบลีและแสดงผลลัพธ์ ตามตารางในข้อ 3

ตอบ

```
        .data
        .balign 4
message: .asciz "GCD : Greatest Common Divisor\n"
```

```
        .balign 4
get_num_1:      .asciz "Number 1 : "
        .balign 4
get_num_2:      .asciz "Number 2 : "

        .balign 4
pattern:  .asciz "%d"
        .balign 4
test:           .asciz "Test %d\n"

        .balign 4
num_1:          .word 0
        .balign 4
num_2:          .word 0

        .balign 4
output:         .asciz "GCD of %d and %d is %d.\n"

        .balign 4
lr_bu:          .word 0
        .balign 4
lr_bu_2:  .word 0
        .balign 4
lr_bu_3:  .word 0

        .text
mod_func:
        ldr r2, =lr_bu_3
        str lr, [r2]
mod_loop:
        cmp r0, r1
        blt end_mod
        sub r0, r0, r1
```

```
        bl mod_loop
end_mod:
        ldr lr, =lr_bu_3
        ldr lr, [lr]

        bx lr

gcd_func:
        ldr r2, =lr_bu_2
        str lr, [r2]
        mov r4, #0
gcd_loop:
        cmp r1, r4
        beq end_gcd
        bl mod_func

        mov r2, r0
        mov r0, r1
        mov r1, r2
        bl gcd_loop
end_gcd:
        ldr lr, =lr_bu_2
        ldr lr, [lr]

        bx lr

        .global main

main:
        ldr r1, =lr_bu
        str lr, [r1]

        ldr r0, =message
```

```
bl printf

ldr r0, =get_num_1
bl printf
ldr r0, =pattern
ldr r1, =num_1
bl scanf

ldr r0, =get_num_2
bl printf
ldr r0, =pattern
ldr r1, =num_2
bl scanf

ldr r0, =num_1
ldr r0, [r0]
ldr r1, =num_2
ldr r1, [r1]
bl gcd_func

mov r3, r0

ldr r0, =output
ldr r1, =num_1
ldr r1, [r1]
ldr r2, =num_2
ldr r2, [r2]
bl printf

ldr lr, =lr_bu
ldr lr, [lr]
bx lr
```

```
.global printf
.global scanf
```