

ภาคผนวก G

การทดลองที่ 7 การเรียกใช้และสร้างฟังก์ชันในโปรแกรมภาษาแอสเซมบลี

ผู้อ่านควรจะต้องทำความเข้าใจเนื้อหาของบทที่ 4 หัวข้อ 4.8 และ ทำการทดลองที่ 5 และการทดลองที่ 6 ในภาคผนวกก่อนหน้า โดยการทดลองนี้จะเสริมความเข้าใจของผู้อ่านให้เพิ่มมากขึ้น ตามวัตถุประสงค์เหล่านี้

- เพื่อพัฒนาโปรแกรมภาษาแอสเซมบลีเรียกใช้งานตัวแปรเดี่ยวหรือตัวแปรสเกลาร์ (Scalar)
- เพื่อพัฒนาโปรแกรมแอสเซมบลีเรียกใช้งานตัวแปรชุดหรืออาร์เรย์ (Array)
- เพื่อเรียกใช้ฟังก์ชันจากไลบรารีพื้นฐานด้วยโปรแกรมภาษาแอสเซมบลี ในหัวข้อที่ 4.8
- เพื่อสร้างฟังก์ชันเสริมในโปรแกรมภาษาแอสเซมบลี

G.1 การใช้งานตัวแปรในดาต้าเซ็กเมนต์

ตัวแปรต่างๆ ที่ประกาศโดยใช้ชื่อ **เลเบล** ต้องการพื้นที่ในหน่วยความจำสำหรับจัดเก็บค่าตามที่ได้สรุปในตารางที่ 2.1 ตัวแปรมีสองชนิดแบ่งตามพื้นที่ในการจัดเก็บค่า คือ

- ตัวแปรชนิด**โกลบอล** (Global Variable) อาศัยพื้นที่สำหรับเก็บค่าของตัวแปรเหล่านี้ เรียกว่า **ดาต้าเซ็กเมนต์** (Data Segment) ซึ่งผู้เขียนได้กล่าวไปแล้วในบทที่ 4 และ
- ตัวแปรชนิด**โลคอล** (Local Variable) อาศัยพื้นที่ภายใน**สแต็กเซ็กเมนต์** (Stack Segment) สำหรับจัดเก็บค่าชั่วคราว เนื่องจากฟังก์ชันคือชุดคำสั่งย่อยที่ฟังก์ชัน main() ในภาษา C หรือ main: ในภาษาแอสเซมบลีเป็นผู้เรียกใช้ และเมื่อทำงานเสร็จสิ้น ฟังก์ชันนั้นจะต้องรีเทิร์นกลับมาหาฟังก์ชัน main() หรือ main: ในที่สุด ดังนั้น ตัวแปรชนิดโลคอลจึงใช้พื้นที่จัดเก็บค่าในสแต็กเฟรมภายในสแต็กเซ็กเมนต์แทน เพราะสแต็กเฟรมจะมีการจองพื้นที่ (PUSH) และคืนพื้นที่ (POP) ในรูปแบบ Last In First Out ตามที่อธิบายในหัวข้อที่ 3.3.3 ทำให้ไม่จำเป็นต้องใช้พื้นที่ในบริเวณดาต้าเซ็กเมนต์ ผู้อ่านสามารถทำความเข้าใจหัวข้อนี้เพิ่มเติมในการทดลองที่ 8 ภาคผนวก H

G.1.1 การโหลดค่าตัวแปรเดี่ยวจากหน่วยความจำมาพักในรีจิสเตอร์

1. ย้ายไคเรกทอรีไปยัง `/home/pi/asm` โดยใช้คำสั่ง `$ cd /home/pi/asm`
2. สร้างไคเรกทอรี `Lab7` โดยใช้คำสั่ง `$ mkdir Lab7`
3. ย้ายไคเรกทอรีเข้าไปใน `Lab7`
4. ตรวจสอบว่าไคเรกทอรีปัจจุบันโดยใช้คำสั่ง `pwd`
5. สร้างไฟล์ `Lab7_1.s` ตามซอร์สโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประโยคคอมเมนต์ได้ เมื่อทำความเข้าใจแต่ละคำสั่งแล้ว

```
.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:    R0 = 15 + 30
```

```
BX LR
```

```
addr_fifteen: .word fifteen
```

```
addr_thirty: .word thirty
```

6. สร้าง makefile ภายในไคเรกทอรี `Lab7` และกรอกคำสั่งดังนี้

```
Lab7_1:
```

```
gcc -o Lab7_1 Lab7_1.s
```

7. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_1
```

```
$ ./Lab7_1
```

```
t63010487@Pi432b:~ $ cd /home
t63010487@Pi432b:/home $ cd /t63010487
-bash: cd: /t63010487: No such file or directory
t63010487@Pi432b:/home $ cd /home/t63010487/asm
t63010487@Pi432b:~/asm $ mkdir Lab7
t63010487@Pi432b:~/asm $ ls -la
total 16
drwxr-xr-x 4 t63010487 t63010487 4096 Feb 14 14:21 .
drwxr-xr-x 5 t63010487 t63010487 4096 Feb 14 13:19 ..
drwxr-xr-x 2 t63010487 t63010487 4096 Feb 14 14:09 Lab6
drwxr-xr-x 2 t63010487 t63010487 4096 Feb 14 14:21 Lab7
t63010487@Pi432b:~/asm $ cd Lab7
t63010487@Pi432b:~/asm/Lab7 $ pwd
/home/t63010487/asm/Lab7
t63010487@Pi432b:~/asm/Lab7 $ nano Lab7_1.s
t63010487@Pi432b:~/asm/Lab7 $ nano makefile
t63010487@Pi432b:~/asm/Lab7 $ make Lab7_1
gcc -o Lab7_1 Lab7_1.s
t63010487@Pi432b:~/asm/Lab7 $ ./Lab7_1
t63010487@Pi432b:~/asm/Lab7 $ echo $?
```

8. สร้างไฟล์ **Lab7_2.s** ตามโค้ดต่อไปนี้จากไฟล์ **Lab7_1.s** ผู้อ่านสามารถข้ามประโยคคอมเมนต์ได้ เมื่อทำความเข้าใจแต่ละคำสั่งแล้ว

```
.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 pointer
```

9. เพิ่มประโยคต่อไปนี้ใน makefile ให้รองรับ Lab7_2

```
Lab7_2:
    gcc -o Lab7_2 Lab7_2.s
```

10. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_2
$ ./Lab7_2
```

บันทึกผลและอธิบายผลที่เกิดขึ้นเพื่อเปรียบเทียบกับข้อที่แล้ว

ได้ผลลัพธ์เท่ากับ คือ 45 โดย Code จะเก็บค่า assigned ไปมา โดย R1 เก็บค่า 15 R2 เก็บ 30
 & R1 ยังไม่เก็บค่า 15 จาก R3 ก่อนเปลี่ยนเป็น 30 และ R2 จะไปเก็บค่ามา

G.1.2 การใช้งานตัวแปรชุดหรืออาร์เรย์ ชนิด word

ภาษาแอสเซมบลีจะกำหนดชนิดตามหลังชื่อตัวแปร เช่น `.word`, `.hword`, และ `.byte` ใช้กำหนดขนาดของตัวแปรนั้นๆ ขนาด 32, 16 และ 8 บิตตามลำดับ ยกตัวอย่าง คือ:

```
numbers:    .word 1,2,3,4
```

เป็นการประกาศและตั้งค่าตัวแปรชนิดอาร์เรย์ของ word ซึ่งต้องการพื้นที่ 4 ไบต์ต่อข้อมูลหนึ่งตำแหน่ง ซึ่งจะตรงกับประโยคต่อไปนี้ในภาษา C

```
int numbers={1,2,3,4}
```

1. สร้างไฟล์ **Lab7_3.s** ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประโยคคอมเมนต์ได้ เมื่อทำความเข้าใจแต่ละคำสั่งแล้ว

ข้อมูลประเภท Array

```
.data
primes:
    .word 2
    .word 3
    .word 5
    .word 7

    .text
.global main
main:
    LDR R3, =primes    @ Load the address for the data in R3
    LDR R0, [R3, #4]   @ Get the next item in the list
end:
    BX LR
```

R3 = [2,3,5,7]

เปลี่ยนเป็น 8 จ. (อีกตัว)

2. เพิ่มประโยคต่อไปนี้ใน makefile ให้รองรับ Lab7_3

```
Lab7_3:
    gcc -o Lab7_3 Lab7_3.s
```

3. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_3
$ ./Lab7_3
```

```
t63010487@Pi432b:~/asm/Lab7 $ nano Lab7_3.s
t63010487@Pi432b:~/asm/Lab7 $ make Lab7_3
gcc -o Lab7_3 Lab7_3.s
t63010487@Pi432b:~/asm/Lab7 $ ./Lab7_3
t63010487@Pi432b:~/asm/Lab7 $ echo $?
3
```

G.1.3 การใช้งานตัวแปรอาร์เรย์ชนิด byte

คำสั่ง **LDRB** ทำงานคล้ายกับคำสั่ง **LDR** แต่เป็นการอ่านค่าของตัวแปรอาร์เรย์ชนิด byte

1. สร้างไฟล์ **Lab7_4.s** ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประโยคคอมเมนต์ได้ เมื่อทำความเข้าใจแต่ละคำสั่งแล้ว

```
.data
numbers:    .byte 1, 2, 3, 4, 5

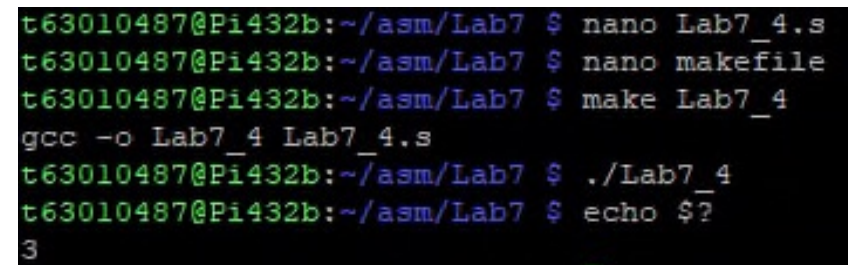
.text
.global main
main:
    LDR R3, =numbers    @ Get address
    LDRB R0, [R3, #2]    @ Get next two bytes
end:
    BX LR
```

2. เพิ่มประโยคต่อไปนี้ใน makefile ให้รองรับ Lab7_4

```
Lab7_4:
    gcc -o Lab7_4 Lab7_4.s
```

3. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_4
$ ./Lab7_4
```



```
t63010487@Pi432b:~/asm/Lab7 $ nano Lab7_4.s
t63010487@Pi432b:~/asm/Lab7 $ nano makefile
t63010487@Pi432b:~/asm/Lab7 $ make Lab7_4
gcc -o Lab7_4 Lab7_4.s
t63010487@Pi432b:~/asm/Lab7 $ ./Lab7_4
t63010487@Pi432b:~/asm/Lab7 $ echo $?
3
```

G.1.4 การเรียกใช้ฟังก์ชันและตัวแปรชนิดประโยครหัส ASCII

ฟังก์ชันสำเร็จรูปที่เข้าใจง่ายและใช้สำหรับเรียนรู้การพัฒนาโปรแกรมภาษา C เบื้องต้น คือ ฟังก์ชัน printf ซึ่งถูกกำหนดอยู่ในไฟล์เฮดเดอร์ stdio.h ตามตัวอย่างซอร์สโค้ด ในรูปที่ 3.9 และการทดลองที่ 5 ภาคผนวก E ในการทดลองต่อไปนี้ ผู้อ่านจะสังเกตเห็นว่าการเรียกใช้ฟังก์ชัน printf ในภาษาแอสเซมบลี โดยอาศัยตัวแปรชนิดประโยค (String) ในรูปที่ 2.11 โดยใช้คำสำคัญ (Key Word) เหล่านี้ คือ .ascii และ .asciz ตัวแปรชนิด asciz จะมีตัวอักษรพิเศษ เรียกว่า อักขรน์ลล์ NULL หรือ /0 ปิดท้ายประโยคเสมอ และอักขร NULL จะมีรหัส ASCII เท่ากับ 00₁₆ ตามตารางรหัสแอสกี ในรูปที่ 2.12

- 1. กรอกรหัสต่อไปนี้ลงในไฟล์ใหม่ชื่อ Lab7_5.s และทำความเข้าใจประโยคคอมเมนต์แต่ละบรรทัด

```
.data
.balign 4
question: .asciz "What is your favorite number?"

.balign 4
message: .asciz "%d is a great number \n"

.balign 4
pattern: .asciz "%d"

.balign 4
number: .word 0

.balign 4
lr_bu: .word 0

.text    @ Text segment begins here

@ Used by the compiler to tell libc where main is located
.global main
.func main

main:
    @ Backup the value inside Link Register
    LDR R1, addr_lr_bu
    STR lr, [R1]      @ Mem[addr_lr_bu] <- LR

    @ Load and print question
    LDR R0, addr_question
    BL printf
```

```

@ Define pattern to scanf and where to store number
LDR R0, addr_pattern
LDR R1, addr_number
BL scanf

```

```

@ Print the message with number
LDR R0, addr_message
LDR R1, addr_number
LDR R1, [R1]
BL printf

```

```

@ Load the value of lr_bu to LR
LDR lr, addr_lr_bu
LDR lr, [lr]      @ LR <- Mem[addr_lr_bu]
BX lr

```

```

@ Define addresses of variables
addr_question: .word question
addr_message:  .word message
addr_pattern:  .word pattern
addr_number:   .word number
addr_lr_bu:    .word lr_bu

```

```

@ Declare printf and scanf functions to be linked with
.global printf
.global scanf

```

2. เพิ่มประโยคใน makefile ให้รองรับ Lab7_5

```

Lab7_5:
    gcc -o Lab7_5 Lab7_5.s

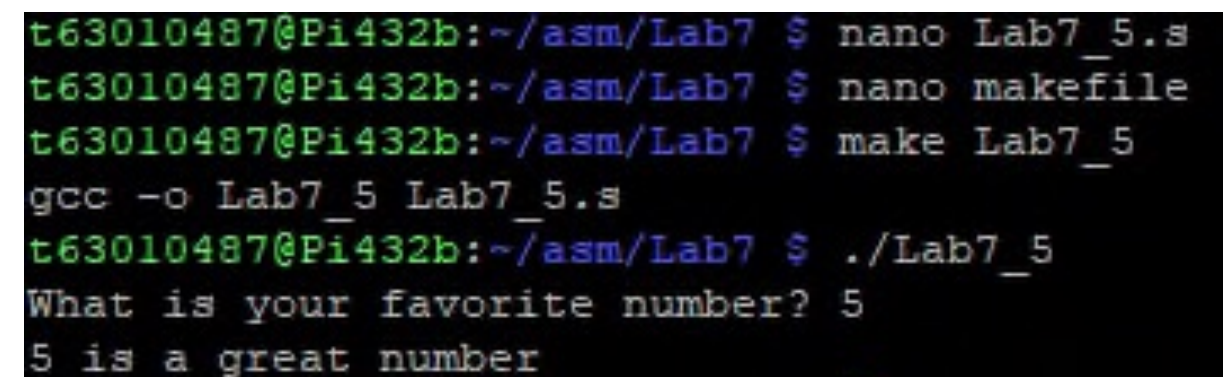
```

3. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```

$ make Lab7_5
$ ./Lab7_5

```



```

t63010487@Pi432b:~/asm/Lab7 $ nano Lab7_5.s
t63010487@Pi432b:~/asm/Lab7 $ nano makefile
t63010487@Pi432b:~/asm/Lab7 $ make Lab7_5
gcc -o Lab7_5 Lab7_5.s
t63010487@Pi432b:~/asm/Lab7 $ ./Lab7_5
What is your favorite number? 5
5 is a great number

```


G.2 การสร้างฟังก์ชันเสริมด้วยภาษาแอสเซมบลี

หัวข้อที่ 4.8 อธิบายโฟลว์การทำงานของฟังก์ชัน โดยใช้งานรีจิสเตอร์ R0 - R12 ดังนี้

- รีจิสเตอร์ R0, R1, R2, และ R3 การส่งผ่านพารามิเตอร์ผ่านทางรีจิสเตอร์ R0 ถึง R3 ตามลำดับไปยังฟังก์ชันที่ถูกเรียก (Callee Function) ฟังก์ชันบางตัวต้องการจำนวนพารามิเตอร์มากกว่า 4 ค่า โปรแกรมเมอร์สามารถส่งพารามิเตอร์ผ่านทางสแต็กโดยคำสั่ง PUSH หรือคำสั่งที่ใกล้เคียง
- รีจิสเตอร์ R0 สำหรับรีเทิร์นหรือส่งค่ากลับไปหาฟังก์ชันผู้เรียก (Caller Function)
- R4 - R12 สำหรับการใช้งานทั่วไป การใช้งานรีจิสเตอร์เหล่านี้ ควรตั้งค่าเริ่มต้นก่อนแล้วจึงสามารถนำค่าไปคำนวณต่อได้
- รีจิสเตอร์เฉพาะ ได้แก่ Stack Pointer (SP หรือ R13) Link Register (LR หรือ R14) และ Program Counter (PC หรือ R15) โปรแกรมเมอร์จะต้องเก็บค่าของรีจิสเตอร์เหล่านี้เก็บไว้ (Back up) ในสแต็ก โดยเฉพาะรีจิสเตอร์ LR ก่อนเรียกใช้ฟังก์ชัน LR ตามที่อธิบายในหัวข้อที่ 4.8.2

ผู้อ่านสามารถสำเนาซอร์สโค้ดในการทดลองที่แล้วมาปรับแก้เป็นการทดลองนี้ได้

1. ปรับแก้ Lab7_5.s ที่มีให้เป็นไฟล์ใหม่ชื่อ Lab7_6.s ดังต่อไปนี้

```
.data
@ Define all the strings and variables
.balign 4
get_num_1: .asciz "Number 1 :\n"

.balign 4
get_num_2: .asciz "Number 2 :\n"

@ printf and scanf use %d in decimal numbers
.balign 4
pattern: .asciz "%d"

@ Declare and initialize variables: num_1 and num_2
.balign 4
num_1: .word 0

.balign 4
num_2: .word 0

@ Output message pattern
.balign 4
output: .asciz "Resulf of %d + %d = %d\n"
```



```
@ Variables to backup link register
.balign 4
lr_bu: .word 0

.balign 4
lr_bu_2: .word 0

.text
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

@ main function
.global main

main:
    @ Store (back up) Link Register
    LDR R1, addr_lr_bu
    STR lr, [R1]      @ Mem[addr_lr_bu] <- LR

    @ Print Number 1 :
    LDR R0, addr_get_num_1
    BL printf

    @ Get num_1 from user via keyboard
    LDR R0, addr_pattern
```

```
LDR R1, addr_num_1
BL scanf

@ Print Number 2 :
LDR R0, addr_get_num_2
BL printf

@ Get num_2 from user via keyboard
LDR R0, addr_pattern
LDR R1, addr_num_2
BL scanf

@ Pass values of num_1 and num_2 to sum_func
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

@ Copy returned value from sum_func to R3
MOV R3, R0    @ to printf

@ Print the output message, num_1, num_2 and result
LDR R0, addr_output
LDR R1, addr_num_1
LDR R1, [R1]
LDR R2, addr_num_2
LDR R2, [R2]
BL printf

@ Restore Link Register to return
LDR lr, addr_lr_bu
LDR lr, [lr]    @ LR <- Mem[addr_lr_bu]
BX lr

@ Define pointer variables
addr_get_num_1: .word get_num_1
addr_get_num_2: .word get_num_2
addr_pattern:   .word pattern
addr_num_1:     .word num_1
```

```
addr_num_2:    .word num_2
addr_output:   .word output
addr_lr_bu:    .word lr_bu
```

```
@ Declare printf and scanf functions to be linked with
.global printf
.global scanf
```

2. เพิ่มประโยคใน makefile ให้รองรับ Lab7_6 ดังนี้

```
Lab7_6:
    gcc -o Lab7_6 Lab7_6.s
```

3. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_6
$ ./Lab7_6
```

4. ระบุซอร์สโค้ดใน Lab7_6.s ว่าตรงกับประโยคภาษา C ต่อไปนี้

```
int num1, num2
```

5. ระบุซอร์สโค้ดใน Lab7_6.s ว่าตรงกับประโยคภาษา C ต่อไปนี้ `sum = num1 + num2`

6. มีการแบ็กอัปค่าของ LR ลงในสแต็กหรือไม่ หากไม่มีแล้วในการทดลองเก็บค่าของ LR ไว้ที่ใด เพราะเหตุใด
ไม่ได้ บันทึกไว้ที่ตำแหน่ง address ของตัวแปร lr_bu & lr_bu-2 เพราะเก็บไว้ใน data segment

7. วิธีการแบ็กอัปค่า LR ในการทดลองสามารถใช้กับฟังก์ชันชนิดรีเคอร์ซีฟ (Recursive) ได้หรือไม่ เพราะเหตุใด
ไม่ได้ เนื่องจาก ตัวแปร lr_bu & lr_bu-2 ที่เก็บ Address ของ lr สะสมค่าขึ้นเรื่อยๆ ไม่สามารถใช้งานแบบ recursive จึงต้องใช้แบบ stack

```
t63010487@Pi432b:~/asm/Lab7 $ nano Lab7_6.s
t63010487@Pi432b:~/asm/Lab7 $ nano makefile
t63010487@Pi432b:~/asm/Lab7 $ make Lab7_6
gcc -o Lab7_6 Lab7_6.s
```

```
t63010487@Pi432b:~/asm/Lab7 $ ./Lab7_6
Number 1 :
5
Number 2 :
8
Result of 5 + 8 = 13
```

```
.balign 4
num_1: .word 0

.balign 4
num_2: .word 0
```

```
sum_func:
    LDR R2, addr_lr_bu_2
    STR lr, [R2]
    ADD R0, R0, R1
    LDR lr, addr_lr_bu_2
    LDR lr, [lr]
    BX lr

addr_lr_bu_2: .word lr_bu_2
```

G.3 กิจกรรมท้ายการทดลอง

- 1. จงเปรียบเทียบการเรียกใช้ฟังก์ชัน printf และ scanf ในภาษา C จากการทดลองที่ 5 ภาคผนวก E กับการทดลองนี้ด้านการส่งพารามิเตอร์
- 2. จงบอกความแตกต่างระหว่างการส่งค่าพารามิเตอร์แบบ Pass by Values และ Pass by Reference
- 3. จงยกตัวอย่างการเรียกใช้ฟังก์ชัน printf ด้วยการส่งค่าพารามิเตอร์แบบ Pass by Values
- 4. จงยกตัวอย่างการเรียกใช้ฟังก์ชัน scanf ด้วยการส่งค่าพารามิเตอร์แบบ Pass by Reference
- 5. จงพัฒนาโปรแกรมด้วยภาษา C เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณและแสดงผลลัพธ์ ตามตารางต่อไปนี้ "A % B = <Result>".

Input	Output
5 2	5 % 2 = 1
18 6	18 % 6 = 0
5 10	5 % 10 = 5
10 5	10 % 5 = 0

- 6. จงเปรียบเทียบฟังก์ชัน scanf และ printf ในการทดลองนี้กับการทดลองที่ 5 ภาคผนวก E
- 7. จงพัฒนาโปรแกรมด้วยภาษา C เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณหาค่า หาร่วมมาก (Greatest Common Divisor) หรือ หรม (GCD) และแสดงผลลัพธ์ตามตัวอย่างในตารางต่อไปนี้

Input	Output
5 2	1
18 6	6
49 42	7
81 18	9

- 8. จงพัฒนาโปรแกรมด้วยภาษา Assembly เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B และแสดงผลลัพธ์ A หรือ B ที่มีค่ามากกว่าด้วยคำสั่งภาษาแอสเซมบลี
- 9. จงพัฒนาโปรแกรมด้วยภาษา Assembly เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B และแสดงผลลัพธ์ค่า A modulus B ซึ่งเท่ากับ ค่าเศษจากการคำนวณ A/B ด้วยคำสั่งภาษาแอสเซมบลี
- 10. จงพัฒนาโปรแกรมด้วยภาษา Assembly เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณหาค่า หาร่วมมาก (Greatest Common Divisor) หรือ หรม (GCD) ด้วยคำสั่งภาษาแอสเซมบลีและแสดงผลลัพธ์ ตามตารางในข้อ 6

9

```

.data
.align 4
get_num_1: .asciz "Number 1 : "

.align 4
get_num_2: .asciz "Number 2 : "

.align 4
pattern: .asciz "%d"

.align 4
num_1: .word 0

.align 4
num_2: .word 0

.align 4
output: .asciz "Result Modulo of %d , %d = %d\n"

.align 4
lr_bu: .word 0

.align 4
lr_bu_2: .word 0

.text
mod_func:
    LDR R2, addr_lr_bu_2
    STR lr, [R2]
    for:
        CMP R0, R1
        BLT end
        SUB R0, R0, R1
        B for

    end:
    LDR lr, addr_lr_bu_2
    LDR lr, [lr]

    BX lr

addr_lr_bu_2: .word lr_bu_2

.global main

main:
    LDR R1, addr_lr_bu
    STR lr, [R1]

    LDR R0, addr_get_num_1
    BL printf

    LDR R0, addr_pattern
    LDR R1, addr_num_1
    BL scanf

    LDR R0, addr_get_num_2
    BL printf

    LDR R0, addr_pattern
    LDR R1, addr_num_2
    BL scanf

```

```

LDR R0, addr_num_1
LDR R0, [R0]
LDR R1, addr_num_2
LDR R1, [R1]
BL mod_func

MOV R3, R0

LDR R0, addr_output
LDR R1, addr_num_1
LDR R1, [R1]
LDR R2, addr_num_2
LDR R2, [R2]
BL printf

```

```

LDR lr, addr_lr_bu
LDR lr, [lr]
BX lr

```

```

addr_get_num_1: .word get_num_1
addr_get_num_2: .word get_num_2
addr_pattern: .word pattern
addr_num_1: .word num_1
addr_num_2: .word num_2
addr_output: .word output
addr_lr_bu: .word lr_bu

```

```

.global printf
.global scanf

```

```

t63010487@Pi432b:~/asm/Lab7 $ ./test
Number 1 : 7
Number 2 : 2
Result Modulo of 7 , 2 = 1
t63010487@Pi432b:~/asm/Lab7 $ ./test
Number 1 : 15
Number 2 : 4
Result Modulo of 15 , 4 = 3

```

8)

```
t63010034@Pi432b:~/topfee/lab7/lab7_8 $ ./output
Enter A : 50
Enter B : 39
50 is greater
t63010034@Pi432b:~/topfee/lab7/lab7_8 $
```

```
t63010034@Pi432b:~/topfee/lab7/lab7_8 $ cat lab7_8.s
.data
.balign 4
prompt1 : .asciz "Enter A : "
.balign 4
prompt2 : .asciz "Enter B : "
.balign 4
message_result: .asciz "%d is greater\n"
.balign 4
pattern : .asciz "%d"
.balign 4
input_A : .word 0
.balign 4
input_B : .word 0
.balign 4
lr_bu : .word 0
.balign 4
lr_bu_2 : .word 0
.text
compare :
    @ Store LR
    LDR R2, =lr_bu_2
    STR LR, [R2]
    @ Execute
    CMP R0, R1
    BGT a_wins

b_wins:
    MOV R0,R1
    B return

a_wins:
    B return

return:
    @ Load LR
    LDR R2, =lr_bu_2
    LDR LR, [R2]

    @ RETURN
    BX LR

.global main

main:
    @ Store LR
    LDR R2, =lr_bu
    STR LR, [R2]

    @ Print prompt1
    LDR R0,=prompt1
    BL printf

    @ Scan number 1
    LDR R0, =pattern
    LDR R1, =input_A
    BL scanf

    @ Print prompt2
    LDR R0,=prompt2
    BL printf

    @ Scan number 2
    LDR R0, =pattern
    LDR R1, =input_B
    BL scanf

    @ Call function compare
    LDR R0, =input_A
    LDR R0, [R0]
    LDR R1, =input_B
    LDR R1, [R1]
    BL compare
    MOV R1, R0

    @ Print result
    LDR R0, =message_result
    BL printf

end:
    @ Load LR
    LDR R2, =lr_bu
    LDR LR, [R2]
    BX LR

.global printf
.global scanf
t63010034@Pi432b:~/topfee/lab7/lab7_8 $
```