

การทดลองที่ 7 การสร้างฟังก์ชันในโปรแกรมภาษาแอสเซมบลี

ผู้อ่านควรจะต้องอ่านเนื้อหาของบทที่ 4 และ ทำการทดลองที่ 5 และการทดลองที่ 6 มาก่อน โดยการทดลองนี้จะเสริมความเข้าใจของผู้อ่านให้เพิ่มมากขึ้น ดังนั้น การทดลองมีวัตถุประสงค์เหล่านี้

- เพื่อพัฒนาโปรแกรมภาษาแอสเซมบลีร่วมกับตัวแปรใน Data Segment
- เพื่อพัฒนาโปรแกรมแอสเซมบลีโดยใช้ตัวแปรอะเรียใน Data Segment
- เพื่อพัฒนาโปรแกรมภาษาแอสเซมบลีร่วมกับฟังก์ชันสำเร็จรูป

G.1 การใช้งานตัวแปรใน Data Segment

ตัวแปรต่างๆ ที่ประกาศโดยใช้ชื่อ เลเบล ต้องการพื้นที่ในหน่วยความจำสำหรับจัดเก็บค่าของมัน ตัวแปร มีสองชนิด คือ

- ตัวแปรชนิดโกลบอล (Global Variable) พื้นที่สำหรับเปิดให้ตัวแปรเหล่านี้ เรียกว่า **ดาตาเซ็กเมนต์** (Data Segment) ซึ่งผู้เขียนได้กล่าวไปแล้วในบทที่ 4 และ
- ตัวแปรชนิดโลคอล (Local Variable) อาศัยพื้นที่ภายใน**สแต็คเซ็กเมนต์** (Stack Segment) ในการจัดเก็บค่าชั่วคราว เนื่องจากฟังก์ชันคือโปรแกรมย่อยที่ฟังก์ชัน main() เป็นผู้เรียกใช้ และเมื่อทำงานเสร็จสิ้น ฟังก์ชันใดๆ จะต้องรีเทิร์นกลับมาหาฟังก์ชัน main() ในที่สุด ดังนั้น ตัวแปรชนิดโลคอล จึงไม่จำเป็นต้องใช้พื้นที่ในดาตาเซ็กเมนต์

G.1.1 การโหลดค่าตัวแปรจากหน่วยความจำ

1. ย้ายไดเรกทอรีไปยัง `$ cd /home/pi/Assembly`
2. สร้างไดเรกทอรี Lab7 ภายใต้ `$ cd /home/pi/Assembly`

3. ย้ายไคเรคทอรีเข้าไปใน Lab7
4. ตรวจสอบว่าไคเรคทอรีปัจจุบันโดยใช้คำสั่ง pwd
5. สร้างไฟล์ Lab7_1.s ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประโยคคอมเมนต์ที่ได้ 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[R1]
    LDR R2, addr_thirty     @ R2 <- address_thirty
    LDR R2, [R2]            @ R2 <- Mem[R2]
    ADD R0, R1, R2
end:
    MOV R7, #1
    SWI 0

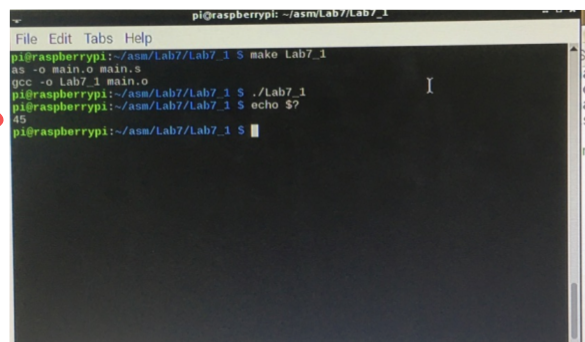
addr_fifteen: .word fifteen
addr_thirty:  .word thirty
```

6. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ ./Lab7_1
$ echo $?
```

บันทึกผลและอธิบายผลที่เกิดขึ้น

Ans →



```
pi@raspberrypi: ~/asm/Lab7/Lab7_1
File Edit Tabs Help
pi@raspberrypi:~/asm/Lab7/Lab7_1$ make Lab7_1
as -o main.o main.s
gcc -o Lab7_1 main.o
pi@raspberrypi:~/asm/Lab7/Lab7_1$ ./Lab7_1
15
pi@raspberrypi:~/asm/Lab7/Lab7_1$ echo $?
15
pi@raspberrypi:~/asm/Lab7/Lab7_1$
```

7. สร้างไฟล์ **Lab7_2.s** ตามโค้ดต่อไปนี้จากไฟล์ **Lab7_1.s** ผู้อ่านสามารถข้ามประโยคคอมเมนต์ที่ได้เมื่อทำความเข้าใจแต่ละคำสั่งแล้ว

```
.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[R1] <- R3
    LDR R2, addr_thirty  @ R2 <- address_thirty
    MOV R3, #30          @ R3 <- 30
    STR R3, [R2]         @ Mem[R2] <- R2

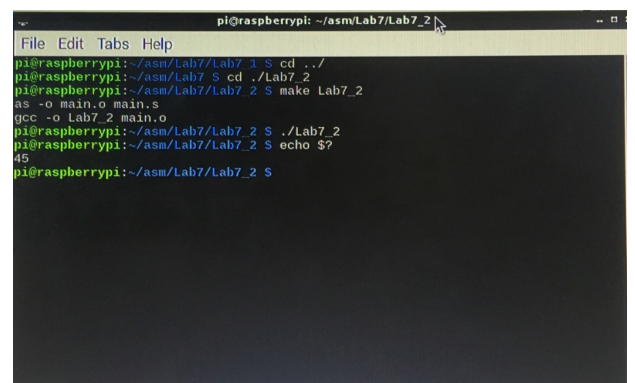
    LDR R1, addr_fifteen @ Load address
    LDR R1, [R1]         @ R1 <- Mem[R1]
    LDR R2, addr_thirty  @ Load address
    LDR R2, [R2]         @ R2 <- Mem[R2]
    ADD R0, R1, R2

end:
    MOV R7, #1
    SWI 0

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

8. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ ./Lab7_2
$ echo $?
```



```
pi@raspberrypi: ~/asm/Lab7/Lab7_2
File Edit Tabs Help
pi@raspberrypi:~/asm/Lab7/Lab7_1 $ cd ../
pi@raspberrypi:~/asm/Lab7 $ cd ./Lab7_2
pi@raspberrypi:~/asm/Lab7/Lab7_2 $ make Lab7_2
as -o main.o main.s
gcc -o Lab7_2 main.o
pi@raspberrypi:~/asm/Lab7/Lab7_2 $ ./Lab7_2
45
pi@raspberrypi:~/asm/Lab7/Lab7_2 $ echo $?
45
pi@raspberrypi:~/asm/Lab7/Lab7_2 $
```

บันทึกผลและอธิบายผลที่เกิดขึ้นเพื่อเปรียบเทียบกับข้อที่แล้ว

G.1.2 การใช้งานตัวแปรอะเรย์ชนิดต่างๆ ใน Data Segment

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

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

เป็นการประกาศและตั้งค่าตัวแปรชนิดอะเรย์ของเวิร์ด ซึ่งต้องการพื้นที่ 4 ไบต์ต่อข้อมูลแต่ละค่า ซึ่งจะตรงกับประโยคต่อไปในภาษา C

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

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

```
.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:
    MOV R7, #1
    SWI 0
```

2. รันโปรแกรม บันทึกและอธิบายผลลัพธ์

ผลลัพธ์ คอ 3 : เมื่อเราโหลด address ของ list
 เราได้ตัวแรกใน list มา
 แล้วเมื่อเราบวก address 4 หรือ 4 bytes
 เราได้ของตัวถัดไปของ list

G.1.3 การใช้งานตัวแปรอรรถาณ Byte

คำสั่ง **LDRB** ทำงานคล้ายกับคำสั่ง **LDR** แต่เป็นการอ่านค่าของตัวแปรอรรถาณ Byte

1. ป้อนคำสั่งต่อไปนี้

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

.text
.global main
main:
    LDR R3, =numbers      @ Get address
    LDRB R0, [R3, #2]     @ Get next byte
end:
    MOV R7, #1
    SWI 0
```

[1, 2, 3, 4, 5]

2. รันโปรแกรม บันทึกและอธิบายผลลัพธ์ = 3 : โหนด addr อธิบายได้ 1
เมื่อเก็บค่า addr ไป 2 bytes เก็บได้ 3

G.2 การเรียกใช้ฟังก์ชันสำเร็จรูปและตัวแปรชนิดประโยค

ฟังก์ชันสำเร็จรูปที่เข้าใจง่ายและใช้สำหรับเรียนรู้การพัฒนาโปรแกรมภาษา C เบื้องต้น คือ ฟังก์ชัน `printf` ซึ่งถูกกำหนดอยู่ในไฟล์เฮดเดอร์ `stdio.h` ตามตัวอย่างซอร์สโค้ด ในรูปที่ 3.16 และการทดลองที่ 5 ภาคผนวก E ในการทดลองต่อไปนี้ ผู้อ่านจะสังเกตเห็นว่าการเรียกใช้ฟังก์ชัน `printf` ในภาษาแอสเซมบลี โดยอาศัยตัวแปรชนิดประโยค โดยใช้คำสำคัญ (Key Word) เหล่านี้ คือ `.ascii` และ `.asciz` ตัวแปรชนิด `asciz` จะมีตัวอักษรพิเศษ เรียกว่า อักขร NULL ปิดท้ายประโยคเสมอ และอักขร NULL จะมีรหัส ASCII เท่ากับ `0016` ตามตารางรหัส ASCII ในรูปที่ 1.12

1. กรอกคำสั่งต่อไปนี้ลงในไฟล์ชื่อ **Lab7_4.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

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


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

    @ Load and print question
    LDR R0, addr_question
    BL printf

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

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

    @ Restore the saved value to link register
    LDR lr, addr_lr_bu
```



```
LDR lr, [lr]      @ LR <- Mem[addr_lr_bu]
BX lr             @ Return to main function
```

```
@ Define addresses
```

```
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
```

- สร้าง makefile ภายในไดเรกทอรี Lab7 และกรอกคำสั่งดังนี้

```
Lab7_4:
```

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

- เรียกใช้ make โปรดสังเกตความแตกต่างที่แสดงผลและใน makefile ที่ผ่านมา

ไฟล์ file .o

G.3 การสร้างฟังก์ชันด้วยภาษาแอสเซมบลี

หัวข้อที่ 4.8 อธิบายโฟลว์การทำงานของฟังก์ชัน โดยอาศัย การใช้งานรีจิสเตอร์ R0 - R12 ดังนี้

- รีจิสเตอร์ R0, R1, R2, และ R3 การส่งผ่านพารามิเตอร์ผ่านทางรีจิสเตอร์ R0 ถึง R3 ตามลำดับ หากฟังก์ชันบางตัวต้องการพารามิเตอร์จำนวนมากกว่า 4 ค่า โปรแกรมเมอร์สามารถส่งผ่านทางสแต็คโดยคำสั่ง PUSH หรือคำสั่งที่ใกล้เคียง
- รีจิสเตอร์ R0 สำหรับรีเทิร์นหรือส่งค่ากลับไปหาฟังก์ชันที่เรียกใช้มัน
- R4 - R12 สำหรับการใช้งานทั่วไป การใช้งานรีจิสเตอร์เหล่านี้ ควรตั้งค่าเริ่มต้นก่อนแล้วจึงสามารถนำค่าไปคำนวณต่อได้
- รีจิสเตอร์เฉพาะหน้าที่ ได้แก่ Stack Pointer (SP หรือ R13) Link Register (LR หรือ R14) และ Program Counter (PC หรือ R15) โปรแกรมเมอร์จะต้องบันทึกค่าของรีจิสเตอร์เหล่านี้เก็บไว้ โดยเฉพาะรีจิสเตอร์ LR ก่อนเรียกใช้ฟังก์ชันใดๆ และคืนค่า (Restore) ที่บันทึกเก็บไว้กลับไปให้รีจิสเตอร์ LR ก่อนจะรีเทิร์นกลับ

ผู้อ่านสามารถสำเนาซอร์สโค้ดในการทดลองที่แล้วมาปรับแก้เป็นการทดลองนี้ได้

1. ปรับแก้ Lab7_4.s ที่มีให้เป็น Lab7_5.s ดังต่อไปนี้

```
.data
@ Define all the strings and variables
.balign 4
get_val_1: .asciz "Number 1 :\n"

.balign 4
get_val_2: .asciz "Number 2 :\n"

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

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

.balign 4
num_2: .word 0

@ Output format
.balign 4
output: .asciz "%d + %d = %d\n"

@ Back up the link register
.balign 4
lr_bu: .word 0

.balign 4
lr_bu_2: .word 0

.text
sum_vals:
    @ Save Link Register
    LDR R2, addr_lr_bu_2
    STR lr, [R2]    @ Mem[R2] <- LR
```



```

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

    @ Restore Link Register
    LDR lr, addr_lr_bu_2
    LDR lr, [lr]      @ LR <- Mem[addr_lr_bu2]

    BX lr

    @ variable to back up Link Register
    addr_lr_bu_2: .word lr_bu_2

    @ Tell LIBC where main is
    .global main

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

    @ Print out message to get 1st value
    LDR R0, addr_get_val_1
    BL printf

    @ Get num1 from user via keyboard
    LDR R0, addr_pattern
    LDR R1, addr_num_1
    BL scanf

    LDR R0, addr_get_val_2
    BL printf

    @ Get num2 from user via keyboard
    LDR R0, addr_pattern
    LDR R1, addr_num_2

```

```
BL scanf

@ Pass by values entered to sum_vals
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_vals

@ Keep returned value from sum_vals in R3
MOV R3, R0

@ Pass the values to display
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
LDR lr, addr_lr_bu
LDR lr, [lr]    @ LR <- Mem[addr_lr_bu]
BX lr

@ Define pointer variables
addr_get_val_1: .word get_val_1
addr_get_val_2: .word get_val_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. ปรับแก้ makefile เพื่อแปลและรันโปรแกรม Lab7_6 แล้วสังเกตผลลัพธ์ที่ได้

3. ระบุข้อผิดพลาดใน Lab7_5.s ว่าตรงกับประโยคภาษา C ต่อไปนี้

```
int num1, num2
```

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

4. ระบุข้อผิดพลาดใน Lab7_5.s ว่าตรงกับประโยคภาษา C ต่อไปนี้

```
sum = num1 + num2
```

```
ADD R0, R0, R1
```

- อันนี้ต่างใน 7.1
เลข 2 ตัว เมื่อใส่แล้วมันบวก
ผลบวกของเลข 2 ตัวนั้น

G.4 กิจกรรมท้ายการทดลอง

1. 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

2. จงพัฒนาโปรแกรมด้วยภาษา Assembly เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณด้วยคำสั่งภาษาแอสเซมบลี และแสดงผลลัพธ์ ตามตารางในข้อ 1
3. จงพัฒนาโปรแกรมด้วยภาษา C เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณหาค่า หรม หรือ หาร่วมมาก และแสดงผลลัพธ์ ตามตารางต่อไปนี้

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

4. จงพัฒนาโปรแกรมด้วยภาษา Assembly เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณหาค่า หรม หรือ หาร่วมมาก ด้วยคำสั่งภาษาแอสเซมบลีและแสดงผลลัพธ์ ตามตารางในข้อ 3

ပုံစံ ၁

```
#include <stdio.h>

int main()
{
    int a, b;
    scanf("%d %d", &a, &b);
    printf("%d %% %d = %d", a, b, a*b);
    return 0;
}
```

ပုံစံ ၂

```
.data
    .balign 4
    a: .word 0
    .balign 4
    b: .word 0
    .balign 4
    msg: .asciz "Enter number : "
    .balign 4
    output: .asciz "%d %% %d = %d\n"
    .balign 4
    format: .asciz "%d"
    .balign 4
    lr_bu: .word 0

.text
.global main
.func main
main:
    LDR R9, addr_lr_bu
    STR LR, [R9]

    LDR R0, addr_msg
    BL printf

    LDR R0, addr_format
    LDR R1, addr_a
    BL scanf

    LDR R0, addr_msg
    BL printf

    LDR R0, addr_format
    LDR R1, addr_b
    BL scanf

    LDR R1, addr_a
    LDR R1, [R1]
    LDR R2, addr_b
    LDR R2, [R2]
```

```
loop:
    CMP R1, R2
    BLT end_loop
    SUB R1, R1, R2
    B loop

end_loop:
    MOV R3, R1
    LDR R0, addr_output
    LDR R1, addr_a
    LDR R1, [R1]
    BL printf

end:
    LDR LR, addr_lr_bu
    LDR LR, [LR]
    BX LR

addr_a: .word a
addr_b: .word b
addr_lr_bu: .word lr_bu
addr_msg: .word msg
addr_output: .word output
addr_format: .word format

.global printf
.global scanf
```

ပုံစံ 3

```
#include <stdio.h>

int main()
{
    int a, b, gcd;
    scanf("%d %d", &a, &b);

    for (int i=1; i <= a && i <= b; i++)
    {
        if (a%i == 0 && b%i == 0)
            gcd = i;
    }

    printf("%d", gcd);

    return 0;
}
```

ပုံစံ 4

```
.data
    .balign 4
    a: .word 0
    .balign 4
    b: .word 0
    .balign 4
    gcd: .word 0
    .balign 4
    msg: .asciz "Enter number : "
    .balign 4
    format: .asciz "%d"
    .balign 4
    output: .asciz "GCD of %d and %d is %d"
    .balign 4
    lr_bu: .word 0

.text
    modulo: @(R1, R2) return R1
    loop_modulo:
        CMP R1, R2
        BLT end_modulo
        SUB R1, R1, R2
        B loop_modulo

.global main
main:
    LDR R1, addr_lr_bu
    STR LR, [R1]

    LDR R0, addr_msg
    BL printf

    LDR R0, addr_format
    LDR R1, addr_a
    BL scanf

    LDR R0, addr_msg
    BL printf
```

```
    LDR R0, addr_format
    LDR R1, addr_b
    BL scanf

    LDR R1, addr_a
    LDR R1, [R1]
    LDR R2, addr_b
    LDR R2, [R2]
loop:
    CMP R1, #0
    BEQ end
    MOV R3, R1
    MOV R1, R2
    MOV R2, R3
    BL modulo
    end_modulo:
    B loop

end:
    LDR R4, addr_gcd
    STR R2, [R4]

    LDR R0, addr_output
    LDR R1, addr_a
    LDR R1, [R1]
    LDR R2, addr_b
    LDR R2, [R2]
    LDR R3, addr_gcd
    LDR R3, [R3]
    BL printf

    LDR LR, addr_lr_bu
    LDR LR, [LR]
    BX LR

addr_a: .word a
addr_b: .word b
addr_gcd: .word gcd
addr_lr_bu: .word lr_bu
addr_msg: .word msg
addr_format: .word format
addr_output: .word output

.global printf
.global scanf
```