

ภาคผนวก G

for Simulator.

การทดลองที่ 7 การเรียกใช้และสร้างฟังค์ชันในโปรแกรมภาษาแอกซ์เมบลี

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

- เพื่อพัฒนาโปรแกรมภาษาแอกซ์เมบลีร่วมกับตัวแปรเดี่ยว **Variable**.
- เพื่อพัฒนาโปรแกรมแอกซ์เมบลีโดยใช้ตัวแปรอะเรย์ **Array**.
- เพื่อฟังค์ชันจากไลบรารีพื้นฐานทางโปรแกรมภาษาแอกซ์เมบลี **library หนึ่ง**.
- เพื่อสร้างฟังค์ชันเสริมในโปรแกรมภาษาแอกซ์เมบลี **function**.

G.1 การใช้งานตัวแปรเดี่ยว **ชนิดโกลบอล** ในหน่วยความจำ

ตัวแปรต่างๆ ที่ประกาศโดยใช้ชื่อ **เลเบล** ต้องการพื้นที่ในหน่วยความจำสำหรับจัดเก็บค่าตามที่ได้สรุปในตารางที่ 2.1 **ตัวแปรมีสองชนิดแบ่งตามพื้นที่ในการจัดเก็บค่า** คือ

- ตัวแปรตัวแปรชนิดโกลบอล (Global Variable) พื้นที่สำหรับเก็บค่าของตัวแปรเหล่านี้ เรียกว่า **ดาตา เช็คเมนท์** (Data Segment) ซึ่งผู้เขียนได้กล่าวไว้แล้วในบทที่ 4 และ **#**
- ตัวแปรชนิดโอลโคอล (Local Variable) อาศัยพื้นที่ภายใน **สแต็คเช็คเมนท์** (Stack Segment) ในการจัดเก็บค่าชั่วคราว เนื่องจากฟังค์ชันคือโปรแกรมย่อยที่ฟังค์ชัน **main()** เป็นผู้เรียกใช้ และเมื่อทำงานเสร็จ สิ้น ฟังค์ชันใดๆ จะต้องรีเทิร์นกลับมาหาฟังค์ชัน **main()** ในที่สุด ดังนั้น ตัวแปรชนิดโอลโคอลจึงใช้พื้นที่จัดเก็บค่าใน **สแต็คเฟรม** ใน **สแต็คเช็คเมนท์**แทน เพราะสแต็คเฟรมจะมีการจ่องพื้นที่ **และคืนพื้นที่ในรูปแบบ Last In First Out** ตามที่อธิบายในหัวข้อที่ 3.2.3 และ 3.2.3 ทำให้มีจำเป็นต้องใช้พื้นที่ในบริเวณดาตาเช็คเมนท์ ผู้อ่านสามารถทำความเข้าใจหัวข้อนี้เพิ่มเติมในการทดลองที่ 8 ภาคผนวก H

Stack ,

bara segment



G.1.1 การโหลดค่าตัวแปรจากหน่วยความจำมาพักในรีจิสเตอร์

1. ย้ายไดเรคทอรีไปยัง \$ cd /home/pi/Assembly
2. สร้างไดเรคทอรี Lab7 ภายใน \$ cd /home/pi/Assembly
3. ย้ายไดเรคทอรีเข้าไปใน Lab7
4. ตรวจสอบว่าไดเรคทอรีปัจจุบันโดยใช้คำสั่ง pwd
5. สร้างไฟล์ Lab7_1.s ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประযุคคอมเมนท์ได้ เมื่อทำการเข้าใจแต่ละคำสั่งแล้ว

```

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

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

```

```

.text
.global main

main:
    LDR R1, addr_fifteen
    LDR R1, [R1]
    LDR R2, addr_thirty
    LDR R2, [R2]
    ADD R0, R1, R2

```

end:

BX LR

addr_fifteen: .word fifteen
addr_thirty: .word thirty

Reset to continue editing code		
		Address
1	fifteen	DCD 15
2	thirty	DCD 30
3		
4	main	
5	LDR	R1, =fifteen
6	LDR	R1, [R1]
7	LDR	R2, =thirty
8	LDR	R2, [R2]
9		
10	ADD	R0, R1, R2
11		

6. สร้าง makefile ภายในไดเรคทอรี Lab7 และกรอกคำสั่งดังนี้

Lab7_1:

gcc -o Lab7_1 Lab7_1.s

Compile

and link

executable
 $0x000F + 0x001E = 0x002D$
 $15 + 30 = 45$

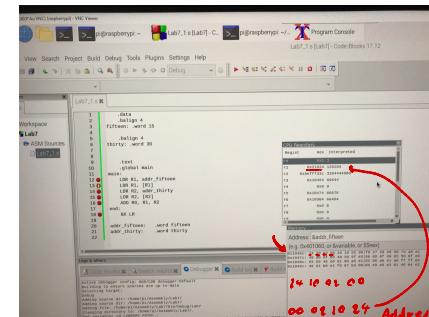
7. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

G.1. การใช้งานตัวแปรเดียวนิดโกลบอลในหน่วยความจำ

273

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

บันทึกผลและอธิบายผลที่เกิดขึ้น ข้อมูลเพิ่มเติมเกี่ยวกับคำสั่ง SWI (Software Interrupt)



var1
var2

8. สร้างไฟล์ **Lab7_2.s** ตามโค้ดต่อไปนี้จากไฟล์ **Lab7_1.s** ผู้อ่านสามารถข้ามประযุคคอมเม้นท์ได้ เมื่อทำการเข้าใจแต่ละคำสั่งแล้ว

```
Byte Alias
.data
.balign 4
fifteen: .word 0
.balign 4
thirty: .word 0

.text
.global main

main:
    LDR R1, addr_fifteen
    MOV R3, #15
    STR R3, [R1]
    LDR R2, addr_thirty
    MOV R3, #30
    STR R3, [R2]

    LDR R1, addr_fifteen
    LDR R1, [R1]
    LDR R2, addr_thirty
    LDR R2, [R2]
    ADD R0, R1, R2

end:
    BX LR
```

@ Request 4 bytes of space
@ fifteen = 0
@ Request 4 bytes of space
@ thirty = 0

Emulation Complete

R0	0x2D
R1	0xF
R2	0x1E
R3	0x1E
R4	0x0
R5	0x0
R6	0x0
R7	0x0
R8	0x0
R9	0x0

NEX.

1. **addr_fifteen** คือค่าที่อยู่ในหน่วยความจำที่กำหนดให้กับตัวแปร fifteen.
2. **addr_thirty** คือค่าที่อยู่ในหน่วยความจำที่กำหนดให้กับตัวแปร thirty.
3. **addr_fifteen** และ **addr_thirty** เป็นค่าที่ได้จากการคำนวณของคำสั่ง LDR R1, [R1] และ LDR R2, [R2] ตามที่ระบุไว้ใน section .data.

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

9. เพิ่มต่อไปนี้ประยุคใน makefile ให้รองรับ Lab7_2

Lab7_2:
gcc -o Lab7_2 Lab7_2.s

output executable file.

10. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_2  
$ ./Lab7_2  
$ echo $? vi
```

บันทึกผลและอธิบายผลที่เกิดขึ้นเพื่อเปรียบเทียบกับข้อที่แล้ว

Joining two memory blocks.
Two blocks of data from registers to memory
Memory merging into N
Registers.

G.1.2 การใช้งานตัวแปรอะเรย์

q̄vumí 27 v. 17 v.

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

numbers: .word 1,2,3,4

เป็นการประกาศและตั้งค่าตัวแปรชนิดอักษรของ Word ซึ่งต้องการพื้นที่ 4 ใบที่ต่อข้อมูลแต่ละค่า ซึ่งจะตรงกับประโยคต่อไปนี้ในภาษา C

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

- สร้างไฟล์ **Lab7_3.s** ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประโยชน์คอมเม้นท์ได เมื่อทำความเข้าใจแต่ละคำสั่งแล้ว

.data

- 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

4 Bute

upon item σ_{array}

Minimum Byte

B7TC	97111446
0	0
4	1
8	2
12	3
16	4
20	5
.	.

2. เพิ่มประโยชน์ต่อไปนี้ใน makefile ให้รองรับ Lab7_3

Lab7_3:

gcc -o Lab7_3 Lab7_3.s

executable

3. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_3
$ ./Lab7_3
$ echo $?
```

G.1.3 การใช้งานตัวแปรของเรียชnid Byte

คำสั่ง **LDRB** ทำงานคล้ายกับคำสั่ง **LDR** แต่เป็นการอ่านค่าของตัวแปรของเรียชnid byte

1. สร้างไฟล์ **Lab7_4.s** ตามโค้ดต่อไปนี้ ผู้อ่านสามารถข้ามประযุคคอมเม้นท์ได้ 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. เพิ่มต่อไปนี้ประโยคใน makefile ให้รองรับ **Lab7_4**

DCB ถือว่า 1Byte หนึ่ง.
LDRB ถือว่า 1Byte หนึ่ง.
(นั่นคือสอง Byte)



Lab7_4:

gcc -o Lab7_4 Lab7_4.s

executable file

3. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_4
$ ./Lab7_4
$ echo $?
```

ก็จะเห็นผลลัพธ์

G.2 การเรียกใช้ฟังค์ชันและตัวแปรชนิดประโยค

ฟังค์ชันสำหรับที่เข้าใจง่ายและใช้สำหรับเรียนรู้การพัฒนาโปรแกรมภาษา C เป็นต้น คือ ฟังค์ชัน printf ซึ่งถูกกำหนดอยู่ในไฟล์ヘดเดอร์ stdio.h ตามตัวอย่างซอฟต์แวร์สโคดู ในรูปที่ 3.16 และการทดลองที่ 5 ภาคผนวก E ในการทดลองต่อไปนี้ ผู้อ่านจะสังเกตเห็นว่าการเรียกใช้ฟังค์ชัน printf ในภาษาแอสเซมบลี โดยอาศัยตัวแปรชนิดประโยค (String) โดยใช้คำสำคัญ (Key Word) เหล่านี้ คือ .ascii และ .asciz ตัวแปรชนิด asciz จะมีตัวอักษรพิเศษ เรียกว่า อักขระ NULL หรือ /0 ปิดท้ายประโยคเสมอ และอักขระ NULL จะมีรหัส ASCII เท่ากับ 00₁₆ ตามตารางรหัส ASCII ในรูปที่ 2.12

- กรอกคำสั่งต่อไปนี้ลงในไฟล์ชื่อ Lab7_5.s และทำความเข้าใจประโยคคอมเม้นท์แต่ละบรรทัด

Label →

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

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

```
.balign 4
pattern: .asciz "%d"
```

```
.balign 4
number: .word 0
```

backup LR

```
.balign 4
lr_bu: .word 0
```

, comment,

```
.text @ Text segment begins here
```

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

```
.global main
```

```
.func main
```

library file.

printf, scanf,

push.

```
main:
```

@ Backup the value inside Link Register

```
{ LDR R1, addr_lr_bu      = lr_bu.
    STR lr, [R1]      @ Mem[addr_lr_bu] <- LR
```

} save LR

push LR stack

pop LR

@ Load and print question

```
LDR R0, addr_question
```

```
BL printf
```

เบื้องต้น LR ให้ยังนี้.

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

@ Print the message with number

LDR R0, addr_message
 LDR R1, addr_number
 LDR R1, [R1]
 BL printf

printf("%d", number)

@ Load the value of lr_bu to LR

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

Return.

@ 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. เพิ่มประโยชน์ใน makefile ให้รองรับ Lab7_5

Lab7_5:

gcc -o Lab7_5 Lab7_5.s

3. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_5
$ ./Lab7_5
%% echo $?
```

4. คำสั่ง echo \$? มีไว้เพื่ออะไร \rightarrow non-void return default by R0

|| กรณี Return ไม่ได้ LR (link register)

ห้อง,

G.3 การสร้างฟังค์ชันเสริมด้วยภาษาแอสเซมบลี

หัวข้อที่ 4.8 อธิบายโดยวิธีการทำงานของฟังค์ชัน โดยอาศัย การใช้งานรีจิสเตอร์ R0 - R12 ดังนี้

- รีจิสเตอร์ R0, R1, R2 และ R3 การส่งผ่านพารามิเตอร์ผ่านทางรีจิสเตอร์ R0 ถึง R3 ตามลำดับ ไปยังฟังค์ชันที่ถูกเรียก (Callee Function) ฟังค์ชันบางตัวต้องการจำนวนพารามิเตอร์มากกว่า 4 ค่า โปรแกรมเมอร์สามารถส่งพารามิเตอร์ผ่านทางสแต็คโดยคำสั่ง PUSH หรือคำสั่งที่ใกล้เคียง
- รีจิสเตอร์ R0 สำหรับรีเทิร์นหรือส่งค่ากลับไปหาฟังค์ชันผู้เรียก (Caller Function)
- R4 - R12 สำหรับการใช้งานทั่วไป การใช้งานรีจิสเตอร์เหล่านี้ ควรตั้งค่าเริ่มต้นก่อนแล้วจึงสามารถนำค่าไปคำนวณต่อได้
- รีจิสเตอร์เฉพาะหน้าที่ ได้แก่ Stack Pointer (SP หรือ R13) Link Register (LR หรือ R14) และ Program Counter (PC หรือ R15) โปรแกรมเมอร์จะต้องบันทึกค่าของรีจิสเตอร์เหล่านี้เก็บไว้ (Backup) โดยเฉพาะรีจิสเตอร์ LR ก่อนเรียกใช้ฟังค์ชันเดียว และคืนค่า (Restore) ที่บันทึกเก็บไว้กลับไปให้รีจิสเตอร์ LR ก่อนจะรีเทิร์นกลับ บันทึก

ผู้อ่านสามารถสำเนาขอร์สโค้ดในการทดลองที่แล้วมาปรับแก้เป็นการทดลองนี้ได้

- ปรับแก้ Lab7_5.s ที่มีให้เป็น 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 "Result 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

```

SCREEN.

(^ number))

```
LDR R0, addr_pattern
LDR R1, addr_num_1
BL scanf
```

KB "%d", &num.

@ Print Number 2 :

```
LDR R0, addr_get_num_2
BL printf
```

SCREEN
(number 2)

@ Get num_2 from user via keyboard

```
LDR R0, addr_pattern
LDR R1, addr_num_2
BL scanf
```

KB "%d", num

@ Pass values of num_1 and num_2 to add

```
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. เพิ่มประโยชน์ใน makefile ให้รองรับ Lab7_6

Lab7_6:

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

3. ทำการ make และรันโปรแกรมโดยใช้คำสั่ง

```
$ make Lab7_6
$ ./Lab7_6
%$ echo $?
```

4. ระบุชอร์สโค้ดใน Lab7_6.s ว่าตรงกับประโยชน์ภาษา C ต่อไปนี้

```
int num1, num2
```

วิเคราะห์โค้ด

```

.balign 4
pattern: .asciz "%d"
.balign 4
num_1: .word 0
.balign 4
num_2: .word 0
.balign 4
output: .asciz "Result of %d + %d = %d\n"

```

5. ระบุชอร์สโค้ดใน Lab7_6.s ว่าตรงกับประโยชน์ภาษา C ต่อไปนี้ $sum = num1 + num2$

มองหา sum-func ใน main

ตรวจสอบ return ของ main

6. เหตุใดจึงผู้อ่านจึงไม่ต้องใช้คำสั่ง echo \$? แล้ว

ไม่รับ input printf กับ scanf ไม่ได้

execute จะรับค่าจากหน้าจอ

```

62    LDR R0, addr_num_1
63    LDR R0, [R0]
64    LDR R1, addr_num_2
65    LDR R1, [R1]
66    BL sum_func          ← พิมพ์ผลลัพธ์
67
68    @ RETURN R0
69
70    MOV R3, R0
71
72    LDR R0, addr_output   @string output
73    LDR R1, addr_num_1
74    LDR R1, [R1]
75    LDR R2, addr_num_2
76    LDR R2, [R2]
77    BL printf
78
79    LDR lr, addr_lr_bu
80    LDR lr, [lr]
81    BX lr
82
83    addr_get_num_1: .word get_num_1
84    addr_get_num_2: .word get_num_2
85    addr_pattern: .word pattern
86    addr_num_1: .word num_1

```

```

19    lr_bu: .word 0
20    .balign 4
21    lr_bu_2: .word 0
22
23
24    .text
25    sum_func:
26    LDR R2, addr_lr_bu_2
27    STR lr, [R2]
28
29    ADD R8, R0, R1
30
31    BLDR lr, addr_lr_bu_2
32    BLDR lr, [lr]
33
34    BX lr
35
36    addr_lr_bu_2: .word lr_bu_2
37
38

```

1st 2nd 3rd
R0 R1 R2 R3
IF AVOID?
1st 2nd 3rd
ตรวจสอบ

G.4 กิจกรรมท้ายการทดลอง

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

$$\begin{aligned} -5 \% 2 &= -1 \\ -2 \times 2 + -1 &= -5 \\ -11 \% 3 &= -7 \end{aligned}$$

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

$$\begin{aligned} 5 - 2 &\rightarrow 3 \\ 3 - 2 &\rightarrow 1 \\ 1 - 2 &\rightarrow \text{IF } (x \leq 0) \\ &\quad \text{break} \end{aligned}$$

6. จงพัฒนาโปรแกรมด้วยภาษา C เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B แล้วคำนวณหาค่า หารร่วมมาก (Greatest Common Divisor) หรือ หرم (GCD) และแสดงผลลัพธ์ตามตัวอย่างในตารางต่อไปนี้

$$\begin{aligned} -1 \times 3 + (-7) &= -11 \\ -11 \times -7 &= \\ -11 + 7 &= -4 \\ -4 + 7 &= \end{aligned}$$

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

$$\begin{aligned} -5 &\rightarrow 2 \\ -5 + 2 &= -3 \\ -3 + 2 &= -1 \\ -1 + 2 &= \end{aligned}$$

7. จงพัฒนาโปรแกรมด้วยภาษา Assembly เพื่อรับตัวเลขจำนวน 2 ตัวจากผู้ใช้ผ่านทางคีย์บอร์ด เรียกว่า A และ B และแสดงผลลัพธ์ A หรือ B ที่มีค่ามากกว่าด้วยคำสั่งภาษาแอสเซมบลี

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

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

$$\begin{aligned} \gcd(b/a, a) & \left\{ \begin{array}{l} \gcd(b/a, a) \\ (\frac{b}{a}, a) \end{array} \right. \\ & \left\{ \begin{array}{l} (\frac{b}{a}, a) \\ (\frac{b}{a}, \frac{a}{b}) \end{array} \right. \\ & \left\{ \begin{array}{l} (\frac{b}{a}, \frac{a}{b}) \\ (\frac{b}{a}, b) \end{array} \right. \\ & \left\{ \begin{array}{l} (\frac{b}{a}, b) \\ (\frac{b}{a}, a) \end{array} \right. \\ & \left\{ \begin{array}{l} (\frac{b}{a}, a) \\ (\frac{b}{a}, a) \end{array} \right. \end{aligned}$$

```

gcd(a, b)
if(a == 0)
    return b
return gcd(b, a)

```