นายธนพล วงศ์อาษา 62010356

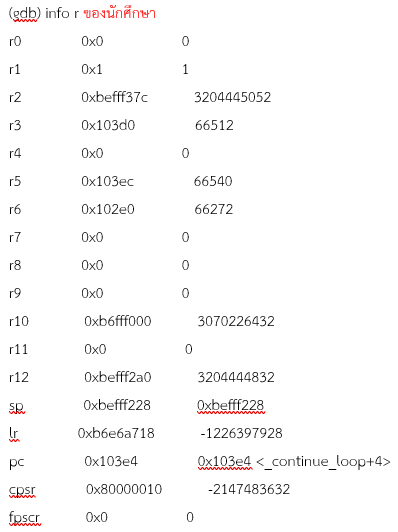
นายสุทธิราช ภูโท 62010966

**การทดลองที่ 8 การพัฒนาโปรแกรมภาษาแอสเซมบลีขั้นสูง**

**Debugger GDB**

9. โปรดสังเกตว่า (gdb) ปรากฏขึ้นแสดงว่าโปรแกรมหยุดที่เบรกพอยท์แล้ว พิมพ์คำสั่ง **(gdb) info r** เพื่อ

แสดงค่าภายในรีจิสเตอร์ต่างๆ ทั้งหมด และบันทึกค่าของรีจิสเตอร์เหล่านี้**r0, r1, r9, sp, pc, cpsr**

หลังรันโปรแกรม

(gdb) info r

r0 0x0 0

r1 0x1 1

r2 0x7effefec 2130702316

r3 0x10408 66568

r4 0x10428 66600

r5 0x0 0

r6 0x102e0 66272

r7 0x0 0

r8 0x0 0

r9 0x0 0

r10 0x76fff000 1996484608

r11 0x0 0

r12 0x7effef10 2130702096

sp 0x7effee90 0x7effee90

lr 0x76e7a678 1994892920

pc 0x1041c 0x1041c <\_continue\_loop+4>

cpsr 0x80000010 -2147483632

ค่าภายในรีจิสเตอร์ต่าง ๆ ได้แก่

**ตอบ** R0=0, R1=1, R9=0, SP=0xbefff228, PC=0x103e4, CPSR=-2147483632

จงตอบคำถามต่อไปนี้ประกอบความเข้าใจ

* อธิบายรายงานบนหน้าจอว่าคอลัมน์แต่ละคอลัมน์มีความหมายอย่างไร และแตกต่างกับหน้าจอ

ของผู้อ่านอย่างไร

**ตอบ** คอลัมน์แรกคือรายการ Register ทั้งหมด คอลัมน์ที่ 2 คือ ค่าที่เก็บใน Register แสดงเป็นเลขฐาน 16 คอลัมน์ที่ 3 คือค่าที่เก็บภายใน Register อาจแสดงเป็นเลขฐาน 10 หรือฐาน 16 จุดที่แตกต่างคือของโปรแกรมที่เขียนขึ้นมาใหม่มี fpscr register เพิ่มขึ้นมาด้วยและค่าที่เก็บภายใน Register R2, R3, R4, R5, R6, R10, R12, SP, LR, CPSR ไม่เท่ากัน

* เหตุใดเลขในคอลัมน์ขวาสุดจึงมีค่าติดลบ หมายเหตุ ศึกษาเรื่องเลขจำนวนเต็มฐานสองชนิดมี

เครื่องหมาย แบบ 2-Complement ในหัวข้อที่2.2.2

**ตอบ** เป็นการแสดงผลเลขฐาน 10 ที่แปลงมาจากเลขฐาน 16 ในคอลัมน์ที่ 2 โดยแสดงด้วยวิธี 2’s-complement

14. คำสั่ง **x/ [count] [format] [address]** แสดงค่าใน หน่วยความจำ ณ ตำแหน่ง address เป็นต้นไป

เป็น จำนวน /count ตาม format ที่ต้องการ ยกตัวอย่างเช่น **x/10i main** คือ แสดงค่าในหน่วยความ

จำ ณ ตำแหน่งเลเบล main จำนวน 10 ค่าตามรูปแบบ instruction ดังตัวอย่างต่อไปนี้

(gdb) x/10i main

0x10408 <main>: mov r0, #0

0x1040c <main+4>: mov r1, #1

0x10410 <main+8>: b 0x10418 <\_continue\_loop>

0x10414 <\_ >: add r0, r0, r1

0x10418 <\_continue\_loop>: cmp r0, #9

=> 0x1041c <\_continue\_loop+4>: ble 0x10414 <\_ >

0x10420 <end>: mov r7, #1

0x10424 <end+4>: svc 0x00000000

0x10428 <\_\_libc\_csu\_init>: push {r4, r5, r6, r7, r8, r9, r10, lr}

0x1042c <\_\_libc\_csu\_init+4>: mov r7, r0

จงตอบคำถามต่อไปนี้

(gdb) x/10i main ของนักศึกษา

0x103d0 <main>: mov r0, #0

0x103d4 <main+4>: mov r1, #1

0x103d8 <main+8>: b 0x103e0 <\_continue\_loop>

0x103dc <\_loop>: add r0, r0, r1

0x103e0 <\_continue\_loop>: cmp r0, #9

=> 0x103e4 <\_continue\_loop+4>: ble 0x103dc <\_loop>

0x103e8 <end>: bx lr

0x103ec <\_\_libc\_csu\_init>: push {r4, r5, r6, r7, r8, r9, r10, lr}

0x103f0 <\_\_libc\_csu\_init+4>: mov r7, r0

0x103f4 <\_\_libc\_csu\_init+8>:

ldr r6, [pc, #72] ; 0x10444 <\_\_libc\_csu\_init+88>

• เติมตัวอักษรที่เว้นว่างไว้จากหน้าจอของผู้อ่านในเครื่องหมาย <\_ > สองตำแหน่ง

**ตอบ** ทั้งสองตำแหน่งคือ <\_loop>

• อธิบายว่า หมายเลขที่มาแทนที่<\_ > ได้อย่างไร

**ตอบ** <\_ > คือ label ที่เขียนไว้ข้างหน้าของโปรแกรม เลขที่สามารถแทน label ได้คือตำแหน่งใน Virtual Memory ซึ่งจากโค้ดที่เขียนขึ้นมาใหม่จะได้ว่า <\_loop> อยู่ที่แอดเดรส 0x103dc

• โปรดสังเกตและอธิบายว่าเครื่องหมายลูกศร => ด้านซ้ายสุดหน้าบรรทัดคำสั่ง หมายถึงอะไร

**ตอบ** จุดที่โปรแกรมรันและมาหยุดอยู่ในปัจจุบัน

12. เริ่มต้นการทดลองโดยพิมพ์คำสั่งต่อไปนี้เพื่อหาว่า เลเบล \_loop ตรงกับหน่วยความจำตำแหน่งใด

(gdb) disassemble \_loop

บันทึกผลที่ได้โดย หมายเลขซ้ายสุด คือ แอดเดรสในหน่วยความจำ ที่คำสั่งนั้นบรรจุอยู่ หมายเลข

ตำแหน่งถัดมา คือ จำนวนไบท์นับจากจุดเริ่มต้นของชื่อเลเบลนั้น แล้วตรวจสอบว่าเลเบล ฟังค์ชัน main

อยู่ห่างจากตำแหน่งเริ่มต้นของโปแกรมกี่ไบท์

Dump of assembler code for function \_loop:

0x00010414 <+0>: add r0, r0, r1

End of assembler dump.

**ตอบ** 12 bytes

17. **i**[nfo] b[reak] เพื่อแสดงรายการเบรกพอยท์ทั้งหมดที่ตั้งไว้ก่อนหน้า

(gdb)i b

Num Type Disp Enb Address What

1 breakpoint keep y 0x0001041c Lab8\\_1.s:\_

breakpoint already hit \_ times

ผู้อ่านจะต้องทำความเข้าใจรายงานที่ได้บนหน้าจอ โดยเฉพาะคอลัมน์Address และ What โดยเติมตัว

อักษรลงในช่องว่าง \_ ทั้งสองช่อง

**ตอบ** \_ ดังกล่าวคือ 11 และ 3 ตามลำดับ

(gdb) i b ของนักศึกษา

Num Type Disp Enb Address What

1 breakpoint keep y 0x000103e4 Lab8\_1.s:11

breakpoint already hit 3 times

**การใช้งานสแต็คพอยท์เตอร์ (Stack Pointer)**

1. สร้างไฟล์**Lab8\_2.s** ตามโค้ดต่อไปนี้ผู้อ่านสามารถข้ามประโยคคอมเม้นท์ได้เมื่อทำความเข้าใจแต่ละ

คำสั่งแล้ว

.global main

main:

MOV R1, #1

MOV R2, #2

@ Push (store) R1 onto stack, then subtract SP by 4 bytes

@ The ! (Write-Back symbol) updates the register SP

STR R1, [sp, #-4]!

STR R2, [sp, #-4]!

@ Pop (load) the value and add 4 to SP

LDR R0, [sp], #+4

LDR R0, [sp], #+4

end:

BX LR

2. รันโปรแกรม บันทีกและอธิบายผลลัพธ์

**ตอบ** โปรแกรมจะเริ่มทำงานโดยการนำค่า 1, 2 ไปเก็บไว้ใน R1, R2 ตามลำดับ จากนั้นนำค่า จาก register R1, R2 ไปเก็บไว้ใน stack segment ทีลำตัวและตำแหน่งจะลดลงครั้งละ 4 bytes โดยใช้คำสั่ง STR ในการเก็บค่าของ R1, R2 ตามลำดับ และโปรแกรมจะทำงานต่อโดยนำค่าจาก stack segment ออกมาทีละตัว และตำแหน่งของ stack segment จะเพิ่มทีละ 4 bytes ด้วยคำสั่ง LDR โดยค่าที่นำออกมาจะเก็บไว้ใน R0 และค่าที่ได้คือค่าของ R2, R1 หรือ 2, 1 ตามลำดับ

3. สร้างไฟล์**Lab8\_3.s** ตามโค้ดต่อไปนี้ผู้อ่านสามารถข้ามประโยคคอมเม้นท์ได้เมื่อทำความเข้าใจแต่ละ

คำสั่งแล้ว

.global main

main:

MOV R1, #0

MOV R2, #1

MOV R4, #2

MOV R5, #3

@ SP is subtracted by 8 bytes to save R4 and R5, respectively.

@ The ! (Write-Back symbol) updates SP.

STMDB SP!, {R4, R5}

@ Pop (load) the values and increment SP after that

LDMIA SP!, {R1, R2}

ADD R0, R1, #0

ADD R0, R0, R2

end:

BX LR

4. รันโปรแกรม บันทีกและอธิบายผลลัพธ์

**ตอบ** โปรแกรมจะทำงานโดยการที่นำค่า 0, 1, 2, 3 ไปเก็บไว้ใน R1, R2, R4, R5 ตามลำดับ จากนั้นจะนำค่าที่เก็บไว้ที่ R4, R5 รวมกันเป็นข้อมูลขนาด 8 bytes โดยค่าของ R4 จะอยู่ข้างหน้า R5 จะอยู่ข้างหลัง และลดตำแหน่งของ stack segment ลง 8 bytes ด้วยคำสั่ง STMDB แล้วจะนำข้อมูลออกมาจาก stack segment ด้วยคำสั่ง LDMIA ข้อมูลที่ได้มาจะเป็นค่าที่อยู่ข้างหน้าก่อน ไปจนหลังค่าที่อยู่ท้ายสุด ดังคำสั่งด้านบน R1 จะเก็บค่า R4 ส่วน R2 จะเก็บค่า R5 แล้วนำค่าทีได้ไปบวกกันแล้วเก็บไว้ใน R0

**การพัฒนาโปรแกรมภาษาแอสเซมบลีร่วมกับภาษา C**

1. เปิดโปรแกรม CodeBlocks

2. สร้างโปรเจ็คท์Lab8\_4 ภายใต้ไดเรคทอรี/home/pi/Assembly/Lab8

3. สร้างไฟล์ชื่อ add\_s.s และป้อนคำสั่งต่อไปนี้

.global add\_s

add\_s:

ADD R0, R0, R1

BX LR

4. เพิ่มไฟล์add\_s.s ในโปรเจ็คท์Lab8\_4 ที่สร้างไว้ก่อนหน้า

5. สร้างไฟล์ชื่อ main.c และป้อนคำสั่งต่อไปนี้

#include <stdio.h>

int main(){

int a = 16;

int b = 4;

int i = add\_s(a, b);

printf(”%d + %d = %d \n”, a, b, i);

return 0;

}

6. ทำการ Build และแก้ไขหากมีข้อผิดพลาดจนสำเร็จ

7. Run และสังเกตการเปลี่ยนแปลง

8. อธิบายว่าเหตุใดการทำงานจึงถูกต้อง ฟังค์ชัน add\_s รับข้อมูลทางรีจิสเตอร์ตัวไหนบ้างและรีเทิร์นค่าที่

คำนวณเสร็จแล้วทางรีจิสเตอร์อะไร

**ตอบ** ฟังค์ชัน add\_s รับข้อมูลผ่านทางรีจิสเตอร์ R0 และ R1 และทำการรีเทิร์นค่ากลับทางรีจิสเตอร์ R0

**กิจกรรมท้ายการทดลอง**

5. จงนําโปรแกรมภาษาแอสเซมบลีสําหรับคํานวณค่า mod ในการทดลองที่ 7 มาเรียกใช้ผ่านโปรแกรม

ภาษา C

**ตอบ**

**mod\_func.s**

.global mod\_func

mod\_func:

mod\_loop:

CMP R0,R1

BLT end\_mod

SUB R0, R0, R1

B mod\_loop

end\_mod:

BX lr

**main.c**

#include <stdio.h>

int main(){

int a,b;

printf("Positive modulus\n");

printf("Enter number 1: ");

scanf("%d", &a);

printf("Enter number 2: ");

scanf("%d", &b);

int i=mod\_func(a,b);

printf("%d %% %d = %d",a,b,i);

return 0;

}

6. จงนําโปรแกรมภาษาแอสเซมบลีสําหรับคํานวณค่า GCD ในการทดลองที่ 7 มาเรียกใช้ผ่านโปรแกรม

ภาษา C

**ตอบ**

**Gcd\_func.s**

.global gcd\_func

gcd\_func:

mov r4, #0

gcd\_loop:

cmp r1, r4

beq end\_gcd

mod\_loop:

cmp r0, r1

blt end\_mod

sub r0, r0, r1

b mod\_loop

end\_mod:

mov r2, r0

mov r0, r1

mov r1, r2

b gcd\_loop

end\_gcd:

bx lr

**main.c**

#include <stdio.h>

int main(){

int a, b;

printf("GCD : Greatest Common Divisor\n");

printf("Enter number 1: ");

scanf("%d", &a);

printf("Enter number 2: ");

scanf("%d", &b);

int gcd = gcd\_func(a, b);

printf("GCD of %d and %d is %d.\n", a, b, gcd);

return 0;

}