

# Lab 6 : 62010694 & 62010718

1. จงปรับแก้คำสั่ง ORR เป็นคำสั่ง AND ในโปรแกรม main.s และตรวจสอบผลการเปลี่ยนแปลงแล้วจึงอธิบาย

R0	0b110	...	...	...
R1	0b10	...	...	...
R2	0b100	...	...	...

```
1      MOV      R0, #0
2      MOV      R1, #2
3      MOV      R2, #4
4      AND      R0, R1, R2
5
```

เป็นการใช้ Bitwise Or (ORR) ระหว่าง R0,R1,R2 มาเป็นการใช้ Bitwise And (AND) ระหว่าง R0,R1,R2 แทน

จะเห็นได้ว่าการนำ 0b010 OR 0b100 จะได้ 0b110 ซึ่งเก็บค่าไว้ใน R0 ซึ่งจะต่างกับการใช้ AND

R0	0b0	...	...	...
R1	0b10	...	...	...
R2	0b100	...	...	...

```
1      MOV      R0, #0
2      MOV      R1, #2
3      MOV      R2, #4
4      ORR      R0, R1, R2
5
```

ซึ่งนำ 0b010 AND 0b100 จะได้ 0b000 ซึ่งจะเก็บไว้ในค่า R0

2. จงปรับแก้โปรแกรมใน main.s เป็นดังนี้ จดบันทึกผลการทดสอบและอธิบาย

```
.global main
main:
    MOV R5, #1
loop:
    CMP R4, #0
    BLE end
else:
    MOV R5, #2
end:
    MOV R0, R5
    BX LR
```

R0	0b1	...	...	...
R1	0b0	...	...	...
R2	0b0	...	...	...
R3	0b0	...	...	...
R4	0b0	...	...	...
R5	0b1	...	...	...

- เริ่มจากการ Assign ค่าของ R5 ให้เป็น 1
- เปรียบเทียบ R4 กับ 0
- ถ้า R4 น้อยกว่าหรือเท่ากับ 0 จะข้ามไปทำงานที่ฟังก์ชัน end
- แต่ถ้าหาก R4 มากกว่า 0 จะลงมาทำงานที่ฟังก์ชัน else ต่อ
- Assign ค่าของ R0 เป็น R5
- Return ค่ากลับไปยัง Terminal

1	main	MOV	R5, #1
2	loop	CMP	R4, #0
3		BLE	lend
4	else	MOV	R5, #2
5	lend	MOV	R0, R5
6			

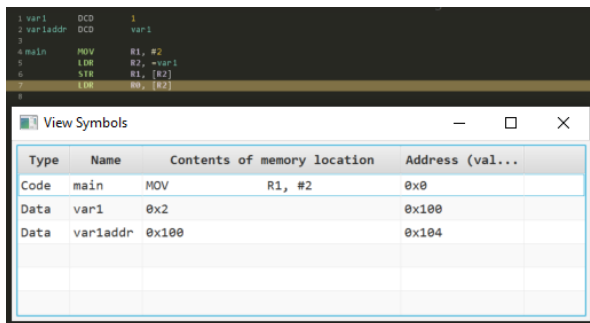
จะเห็นได้ว่าเนื่องจาก R4 เท่ากับ 0 ทำให้เข้าเงื่อนไข ข้ามไปทำงานฟังก์ชัน end และ Assign ค่า R0 ให้เท่ากับ R5 สุดท้าย R0 เลยเท่ากับ R5

3. จงปรับแก้โปรแกรมใน main.s เป็นดังนี้ จดบันทึกผลการทดสอบและอธิบาย

```
.data
.balign 4
var1: .word 1
.text
.global main
main:
```

#### F.5. กิจกรรมท้ายการทดลอง

```
MOV R1, #2
LDR R2, var1addr
STR R1, [R2]
LDR R0, [R2]
BX LR
var1addr: .word var1
```



The screenshot shows a debugger interface. The top pane displays assembly code with line numbers 1 through 8. The bottom pane, titled 'View Symbols', shows a table of symbols.

Type	Name	Contents of memory location	Address (val...)
Code	main	MOV R1, #2	0x0
Data	var1	0x2	0x100
Data	var1addr	0x100	0x104

R0	0x2	...	...	...
R1	0x2	...	...	...
R2	0x100	...	...	...
R3	0x0	...	...	...
R4	0x0	...	...	...
R5	0x0	...	...	...
R6	0x0	...	...	...
R7	0x0	...	...	...
R8	0x0	...	...	...
R9	0x0	...	...	...
R...	0x0	...	...	...
R...	0x0	...	...	...
R...	0x0	...	...	...
R...	0xFF000000	...	...	...
LR	0x0	...	...	...
PC	0x14	...	...	...

- สร้างตัวแปร var1 ให้เท่ากับ 1 คือ 0x2 มี Address เป็น 0x100
- สร้างตัวแปร var1addr ให้ชี้ไปที่ Address ของตัวแปร var1 ซึ่งเป็น 0x100 ตอนนี้เลย var1addr มีค่าเป็น 0x100 และมี Address เป็น 0x104
- Assign ค่า R1 เป็น 2
- Load Register ทำการ Load ค่า Address ของ var1 ซึ่งคือ 0x100 มาใส่ R2

- **Store Register** ทำการ Save ค่า R1 ลงไปที่ Address R2 ซึ่งชี้ไปที่ var1
- **Load Register** ทำการ Load ค่าจาก Address ของ R2 ซึ่งชี้ไปที่ var1 ไปใส่ไว้ใน R0

จะเห็นได้ว่าการกำหนดตัวแปรนั้นจะสามารถชี้ไปที่ Address ของตัวแปรอื่นได้เช่นเดียวกัน ซึ่งทำหน้าที่เหมือน

**Pointer**