## ข้อที่ 8 while: .data CMP R1,R2 @cmp A,B .balign 4 BLT endW @if A<B to endW get\_A: .asciz "A:" SUB R1,R1,R2 @A=A-B .balign 4 B while get\_B: .asciz "B:" endW: .balign 4 CMP R6,#0 pattern: .asciz "%d" BEQ pos @if A = 0 to pos (A is positive) .balign 4 MOV R3,R1 output: .asciz "%d mod %d = %d\n" MOV R4,#-1 MUL R3,R3,R4 @R3=R3\*(-1),result .balign 4 B endRe A: .word 0 pos: .balign 4 MOV R3,R1 @result B: .word 0 endRe: .balign 4 result: .word 0 @ Load Link Register from back up 2 LDR Ir, addr\_Ir\_bu\_2 @ Variables to backup link register LDR lr, [lr] @ LR <- Mem[addr\_lr\_bu\_2] .balign 4 Ir\_bu: .word 0 BX Ir .balign 4 Ir\_bu\_2: .word 0 @ address of Link Register back up 2 addr\_lr\_bu\_2: .word lr\_bu\_2 .text mod\_func: .global main @ Save (Store) Link Register to Ir\_bu\_2 main: LDR R7, addr\_lr\_bu\_2 STR lr,[R7] @ Mem[addr\_lr\_bu\_2] <- LR @ Store (back up) Link Register LDR R1, addr Ir bu MOV R6,#0 @ $A \ge 0$ , R6 = 0 STR Ir, [R1] @ Mem[addr\_Ir\_bu] <- LR CMP R1,#0 @cmp A,0 BGE else1 @if A>=0 to else1 @ Print A: MOV R6,#1 @ A<0, R6=1 LDR RO, addr\_get\_A MOV R5,#-1 BL printf MUL R1,R1,R5 @ A=A\*(-1) else1: @ Get A from user via keyboard CMP R2,#0 @cmp B,0 LDR RO, addr pattern BGE end @if B >= 0 to end LDR R1, addr A MOV R5,#-1 BL scanf MUL R2,R2,R5 end: @ Print B: LDR RO, addr\_get\_B

## **BL** printf

@ Get B from user via keyboard LDR R0, addr\_pattern LDR R1, addr\_B BL scanf

LDR R1, addr\_A LDR R1, [R1] LDR R2, addr\_B LDR R2, [R2] BL mod\_func

LDR R0,=output LDR R1, addr\_A LDR R1, [R1] LDR R2, addr\_B 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

addr\_get\_A: .word get\_A addr\_get\_B: .word get\_B addr\_pattern: .word pattern addr\_output: .word output addr\_A: .word A addr\_B: .word B

addr\_lr\_bu: .word lr\_bu

.global printf .global scanf

```
ข้อที่ 9
                                                      .text
                                                      gcd func:
  .data
                                                         @ Save (Store) Link Register to Ir_bu_2
  .balign 4
                                                        LDR R7, addr_lr_bu_2
get_A: .asciz "A:"
                                                        STR lr,[R7] @ Mem[addr_lr_bu_2] <- LR
  .balign 4
get_B: .asciz "B:"
                                                          MOV R6,#0 @ A>=0 , R6 = 0
  .balign 4
pattern: .asciz "%d"
                                                          CMP R1,#0 @cmp A,0
  .balign 4
                                                          BGE else1 @if A>=0 to else1
output: .asciz "GCD of %d and %d = %d\n"
                                                          MOV R6,#1 @ A<0, R6=1
  .balign 4
                                                          MOV R5,#-1
A: .word 0
                                                          MUL R1,R1,R5 @ A=A*(-1)
  .balign 4
                                                        else1:
B: .word 0
                                                          CMP R2,#0 @cmp B,0
  .balign 4
                                                          BGE end @if B >= 0 to end
result: .word 0
                                                          MOV R5,#-1
                                                          MUL R2, R2, R5
@ Variables to backup link register
                                                        end:
.balign 4
Ir_bu: .word 0
                                                        gcd:
                                                          CMP R1,R2 @cmp A,B
.balign 4
                                                          BEQ endGcd @if A=B to endGcd
Ir_bu_2: .word 0
                                                          CMP R1,R2 @cmp A,B
                                                          BLE elseGcd @if A<=B to elseGcd
                                                          SUB R1,R1,R2 @A=A-B if A>B
                                                          b gcd
                                                        elseGcd:
                                                          SUB R2,R2,R1 @B=B-A if B>A
                                                          b gcd
                                                        endGcd:
                                                          MOV R3,R1
                                                         @ Load Link Register from back up 2
                                                        LDR lr, addr_lr_bu_2
```

LDR lr, [lr] @ LR <- Mem[addr\_lr\_bu\_2] BX Ir @ address of Link Register back up 2 addr\_lr\_bu\_2: .word lr\_bu\_2 .global main main: @ Store (back up) Link Register LDR R1, addr\_lr\_bu STR Ir, [R1] @ Mem[addr Ir bu] <- LR @ Print A: LDR R0, addr\_get\_A BL printf @ Get A from user via keyboard LDR R0, addr\_pattern LDR R1, addr\_A BL scanf @ Print B: LDR R0, addr\_get\_B BL printf @ Get B from user via keyboard LDR R0, addr\_pattern LDR R1, addr\_B BL scanf LDR R1, addr A LDR R1, [R1] LDR R2, addr\_B LDR R2, [R2]

BL gcd\_func

@print output
LDR R0,=output
LDR R1, addr\_A
LDR R1, [R1]
LDR R2, addr\_B
LDR R2, [R2]
BL printf
@ Restore Link Register to return
LDR Ir, addr\_Ir\_bu
LDR Ir, [Ir] @ LR <- Mem[addr Ir bu]</pre>

addr\_get\_A: .word get\_A
addr\_get\_B: .word get\_B
addr\_pattern: .word pattern
addr\_output: .word output
addr\_A: .word A
addr\_B: .word B
addr\_lr\_bu: .word lr\_bu

global printf. global scanf