실습: ARM 프로세서 실습(2)

※ 다음 실습도 온라인 시뮬레이터(https://cpulator.01xz.net/)에서 Architecture를 ARMv7으로, System을 ARMv7 generic으로 선택하여 수행하시오.

1. load/store 명령어를 사용하는 다음 프로그램을 각각 작성하여 single step(F2)으로 실행하여 동작을 확인하시오. (주석을 참고하여 결과용 레지스터값, 플래그 값 확인, 참조한 메모리 주소, 필요한 경우 명령어 동작 설명)

(1)

```
.global _start
_start:
        mov r0, #data1          // r0, data1의 주소 확인
        ldr r1, [r0]            // r1
        ldr r2, [r0, #4]        // r2, r0, 메모리 주소
        ldr r3, [r0], #4        // r3, r0, 메모리 주소
        ldr r4, [r0,#4]!        // r4, r0, 메모리 주소

        ldr r0, =#data1         // r0
        mov r1, #12
        ldr r5, [r0, r1]        // r5, 메모리 주소
        mov r1, #3
        ldr r6, [r0, r1, lsl #2]  //  r6, 메모리 주소
stop:
        b stop
data1:  .word 1, 2, 3, 4, 5, 6, 7, 8
```

(2)

```
.global _start
_start:
        mov r0, #src            // r8, src 주소 확인
        mov r1, #dst            // r9, dst 주소
        ldmia r0, {r2-r9}       // r0, r2-r9
        stmia r1!,{r2-r9}       // r1, dst 메모리 블록 내용

        mov sp, #tos            // sp, tos 주소 확인
        stmfd sp!, {r2-r9}      // sp, 스택메모리
        ldmfd sp!, {r2-r9}      // sp, r5-r9

        mov sp, #tos
        push {r2-r4}            // sp, 스택메모리
        pop {r10-r12}           // sp, r10-r12
stop:
        b stop

src:    .word 0x1111, 0x2222, 0x3333, 0x4444, 0x5555, 0x6666, 0x7777, 0x8888
dst:    .space 20
stack:  .space 256
tos:    // top of stack
```

(3) 다음 프로그램을 r0와 r1값을 (a), (b), (c), (d)와 같은 네 가지 경우로 작성하여 플래그와 r2 결과를 확인하시오. 이를 통하여 플래그와 크기 비교와의 관계를 확인하시오.

```
.global _start
_start:                         // (a)   (b)         (c)   (d)
        ldr r0, =#0x80          // 0x80  0x70000000  0x10  0x10
        ldr r1, =#0x10          // 0x10  0x90000000  0x80  0x10
        cmp r0, r1              // N, V, Z ?
        movgt r2, #1
        movlt r2, #2
        moveq r2, #3            // r2 ?
stop:
        b stop
```

2. 최대공약수를 계산하는 다음 예제 프로그램의 동작을 분석하고 수행하시오.

```
.text
.global _start

_start: MOV     R0, #54         /* Register R0 is the first number. */
        MOV     R1, #24         /* Register R1 is the second number. */

GCD:    CMP     R0, R1          /* Set the condition codes. */
        SUBGT   R0, R0, R1      /* If (R0 > R1) then R0 = R0 - R1. */
        SUBLT   R1, R1, R0      /* If (R0 < R1) then R1 = R1 - R0. */
        BNE     GCD             /* If (R0 != R1) then loop again. */

STOP:   B       STOP            /* If (R0 == R1) then stop. */

.end
```

3. 다음 예제 프로그램을 각각 실행시키고, single step 기능 등을 이용하여 동작을 확인하고 서브루틴 호출 기능에 대해서 동작을 설명하시오. (호출 전후 동작을 잘 관찰하시오.)

(1)

```
// Find the sum of numbers from 0 to N

/* Equivalent C code function would be:
int FINDSUM(int N)
{
        int sum = 0;

        while (N > 0) {
                sum = sum + N;
                N = N - 1;
        }
        return sum;
}
*/

.text
.global _start
_start:
        LDR     R0, N
```

```
        BL      FINDSUM
END:    B       END

// FINDSUM subroutine: calculates the sum of numbers from 1 to N
//      Parameters: R0 = N
//      Returns: R0 = sum

FINDSUM:
        MOVS    R1, R0          // R1 : N
        MOV     R0, #0          // R0 : sum
SUM_LOOP:
        BLE     END_LOOP        // if (N <=0) goto end_loop
        ADD     R0, R0, R1      // sum = sum + N
        SUBS    R1, #1          // N = N - 1
        B       SUM_LOOP
END_LOOP:                       // R0 : sum
        MOV     PC, LR          // return

N:      .word   5
.end
```

(2)

```
/* Find the sum of numbers from 0 to N - Recursive function  */
/* Equivalent C code function would be:
        int FINDSUM(int N)
        {
                if (N <= 0) return 0;
                return N + FINDSUM(N-1);
        }
*/
.text
.global _start
_start:
        LDR     R0, N
        BL      FINDSUM
END:    B       END

FINDSUM:
        PUSH    {R4, LR}        // save state (R4, LR)
        MOVS    R4, R0          // save N in R4, and check for 0
        BLE     RETURN          // if N == 0, just return N
RECURSE:
        SUB     R0, R4, #1      // set R0 = N-1
        BL      FINDSUM
        ADD     R0, R4, R0      // R0 = N + FINDSUM(N-1)
RETURN:
        POP     {R4, PC}        // the return value is in R0

N:      .word   5

.end
```