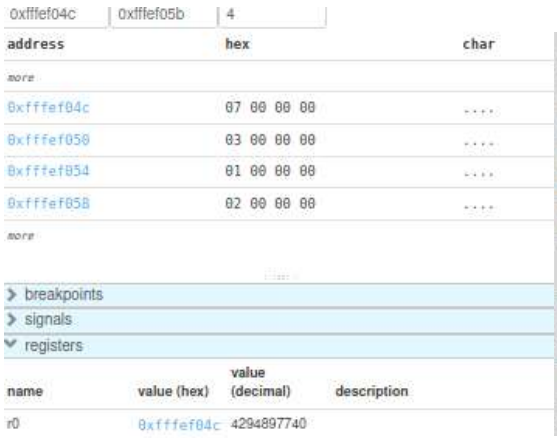
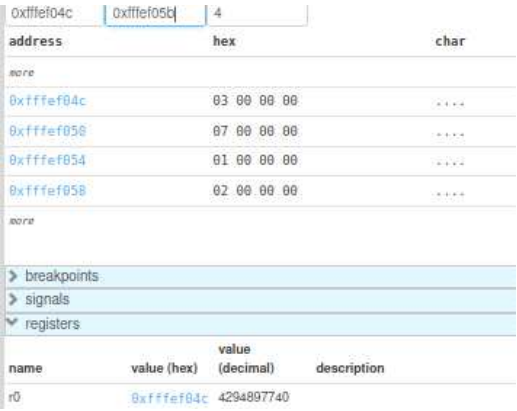
[숙제 7] lab3 실습 내용
제출할 자료:

(1) 실습 A 관련(LAB3 P23-24)
Lab3 p23-24를 참고하여, bubblesort를 수행하면서 outer loop 수행이 끝날 때마다 배열 값의 변동을 확인하세요. (14번 줄, 43번 줄에 breakpoint를 설정하고 p23에 있는 그림처럼 수행되는지 확인하고 그 과정을 캡처하여 보고서에 첨부하세요.)

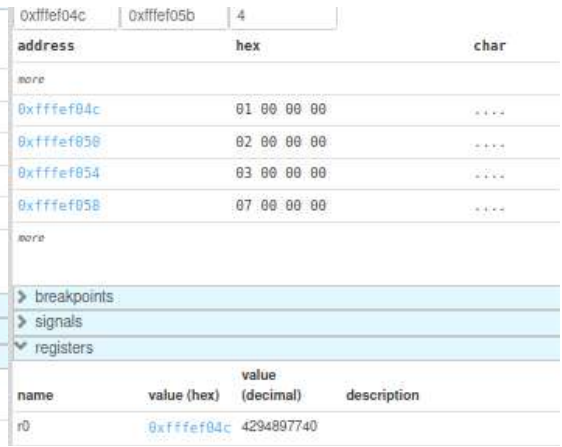


i=0, arr={7,3,1,2}



i=1, arr={3,7,1,2}

i=3, arr={1,3,7,2}



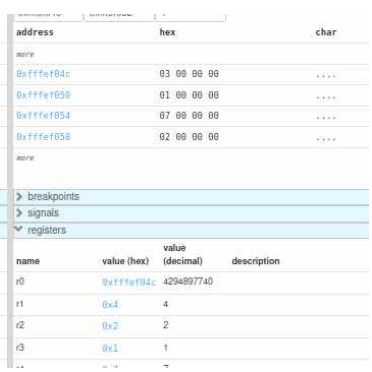
i=4, arr={1,2,3,7}

(2) 실습 B 관련(LAB3 P25-27)

bubblesort를 수행하면서 i가 2일 때, inner loop를 수행하면서 배열 값의 변동을 확인하세요. (14번 줄, 43번 줄에 breakpoint를 설정하고 r2가 2일 때까지 계속 수행합니다. r2가 2가 되고 outer loop(for1tst)를 수행하기 전(43번 줄), 35번 줄, 36번 줄, 38번 줄에 breakpoint를 설정하고 현재 arr에 저장된 값을 확인합니다. 35번 줄에서의 r3, arr에 저장된 값과 36번 줄에서의 r3, arr에 저장된 값을 비교합니다. 43번 줄에서 멈추면 r2, arr에 저장된 값을 p4에 있는 pass i 결과와 비교합니다. 위 과정에 대한 캡처화면을 설명과 함께 보고서에 첨부하세요.)

j=1, before swap, L35

j=1, after swap, L36



j=1, L43

r2와 r3의 값은 각각 c코드에서의 인덱스 i, j를 의미한다. L36에서만 r2 값이 다르게 나오는데 swap을 call 할 때, {r0,r1,r2,r3,r12} 레지스터를 이용해 값을 변경시키게 되고 swap 전에 스택에 저장해둔 값을 ldmia sp!,{r0,r1,r2,r3,r12} 명령어를 통해 가져오기 때문에 L43 에서는 이전값이 복구된다. swap에서 ldr r2, [r12, #0] 명령어를 이용해 r2에 7을 저장하기 때문에 L36에서 r2는 7이 된다.

j=0, before swap, L35

j=0, after swap, L36

j=0, L43

마찬가지로 swap을 call 할 때, {r0,r1,r2,r3,r12} 레지스터를 이용해 값을 변경시키게 되고 swap 전에 스택에 저장해둔 값을 ldmia sp!,{r0,r1,r2,r3,r12} 명령어를 통해 가져오기 때문에 L43 에서는 이전값이 복구된다. swap에서 ldr r2, [r12, #0] 명령어를 이용해 r2에 3을 저장하기 때문에 L36에서 r2는 3이 된다.

(3) 실습 C 관련(LAB3 P28-29)

swap 함수 수행 전후 레지스터 저장 및 복구 루틴을 삭제한 후, swap 함수 수행 전후 i, j 값을 확인하세요. (bubblesort 어셈블리 코드 32번 줄, 36번 줄을 주석 처리하고 재실행합니다. 15번 줄, 43번 줄에 breakpoint를 설정하고 arr에 저장된 값들을 확인합니다. 43번 줄에서 멈추면 현재 r2값을 확인하고, 만약 1이면 35번 줄, 38번 줄에 breakpoint를 설정합니다. 35번 줄과 38번 줄에서 r3값과 arr에 저장된 값을 각각 확인하고 비교합니다. 35번 줄에서의 r2, r3값과 38번 줄에서의 r2, r3값이 동일한지 확인하고 동일하지 않다면 이유를 분석한다. 위 과정에 대한 캡처화면을 설명과 함께 보고서에 첨부하세요.)

L35

L38

r2와 r3의 값은 각각 c코드에서의 인덱스 i, j를 의미하는데 문제에서는 r2에 저장된 값이 1일 경우의 35번 줄에서의 r2, r3값과 38번 줄에서의 r2, r3값을 비교한다. 두 코드는 동일한 inner Loop에 선언되어 있어 r2, r3의 값이 변경되지 않아야 한다. 하지만 multiple store/multiple load 코드(L32, L36)를 삭제함으로서 r2, r3를 사용하는 swap을 수행 할 때 저장한 값을 그대로 사용하게 되어 의도하지 않은 결과로 이어 질 수 있다.

multiple store/multiple load 코드는 2번에서 설명

(4) 실습 D 관련(LAB3 P30)

p8에 있는 어셈블리 코드는 bubble sorting 방법 2로 구현되었습니다. 이를 bubble sorting 방법 1(p3)로 동작하도록 수정합니다. (만일 정상적으로 동작하지 않는다면 p3에 있는 예제에서 outer loop의 pass i에 해당된 결과가 나왔는지 단계별로 확인해보세요. 실습 A, B에서 설정했던 breakpoint를 활용해 디버깅해보세요).

정상적으로 동작한다면 line by line으로 설명된 어셈블리 코드와 수행 결과를 캡처하여 보고서에 첨부하세요. 화면캡쳐, 설명내용, 실습 D 소스코드를 압축하여 하나의 file(파일이름명:HW7-학번-이름)로 스마트캠퍼스에 제출하세요.

방법 2 :

```
void sort (int v[], int n){
        int i, j;
        for (i = 0; i < n; i++)
                for (j = i - 1 ; j >= 0 ; j --)
                        if(v[j] > v[j + 1]) swap(v,j);
}
```

방법 1 :

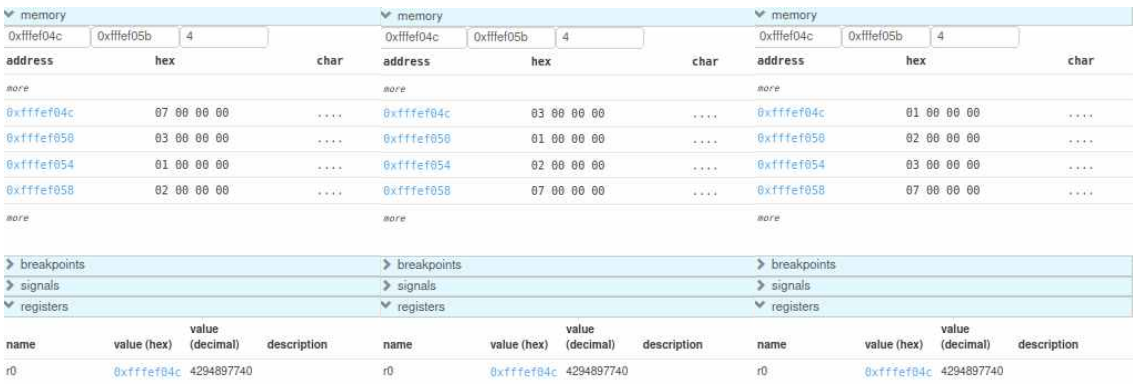
```
void sort (int v[], int n){
        int i, j;
        for (i = 0; i < n; i++)
                for (j = 0 ; j < n ; j ++)
                        if(v[j] > v[j + 1]) swap(v,j);
}
```

=> 방법 2에서 방법 1로 변환해야함

```
bubblesort1:
                sub     sp, sp, #20     // 스택공간을 할당(int값 4개, lr값 저장 위치)
                str     lr, [sp, #16]   // main으로의 return address 저장
                str     r7, [sp, #12]   // v[3]
                str     r6, [sp, #8]    // v[2]
                str     r3, [sp, #4]    // v[1]
                str     r2, [sp, #0]    // v[0]
                mov     r6, r0          // r6에 r0 저장(인자로 받은 배열의 시작주소)
                mov     r7, r1          // r7에 r1 저장(인자로 받은 n값)
                mov     r2, #0          // r2에 #0저장(c코드 : i = 0)

for1tst:(outer loop)
                cmp     r2, r1          // r2와 r1값 비교(c코드 : i와 n)
                bge     exit1           // r2가 r1보다 크거나 같을 경우 exit1 이동
                mov     r3, #0          // r3에 #0 저장
```

```
for2tst: (inner loop)
        cmp     r3, r1              // r3와 r1비교
        bge     exit2               // r3가 r1보다 크거나 같을 경우 exit2 이동
        add     r12, r0, r3, LSL #2 // 배열 시작 주소 r0 + 인덱스 r3 * int형
                                        // 현재 참조할 원소의 주소를 r12에 저장
        ldr     r4, [r12, #0]       // r12 주소의 값을 r4에 저장
        ldr     r5, [r12, #4]       // r12+ #4 주소의 값을 r5에 저장
        cmp     r4, r5              // r4, r5를 비교(인접한 원소 값을 비교)
        ble     exit2               // r4 < r5 일 경우 exit2로 이동
        stmdb   sp!,{r0,r1,r2,r3,r12}       // r0,r1,r2,r3,r12을 스택에 저장
        mov     r0, r6                      // r0에 r6 저장
        mov     r1, r3                      // r1에 r3 저장
        bl      swap                // lr에 복귀할 주소를 저장하고 swap수행
        ldmia sp!,{r0,r1,r2,r3,r12} // r0,r1,r2,r3,r12을 스택에서 불러옴
        add     r3, r3, #1                  // r3 1증가(c코드 : j++)
        b       for2tst                     // inner loop로 이동
exit2:
        add     r2, r2, #1                  // r2 1증가(c코드 : i++)
        b       for1tst                     // outer loop로 이동
exit1:
        ldr     r2, [sp, #0]        // sp+#0을 r2로 불러옴  v[0]
        ldr     r3, [sp, #4]        // sp+#4을 r3로 불러옴  v[1]
        ldr     r6, [sp, #8]        // sp+#8을 r6로 불러옴  v[2]
        ldr     r7, [sp, #12]       // sp+#12을 r7로 불러옴 v[3]
        ldr     lr, [sp, #16]       // sp+#16을 lr로 불러옴
        add     sp, sp, #20         // stack pointer 복구
        mov     pc, lr    // lr값을 pc에 복사(main으로 복귀)
.end
```



i=0, j=0　　i=1, j=0　　i=2, j=0

```
Sungwon@ubuntu:~/lab3$ qemu-arm -g 8080 lab3
Array before sorting : 7  3  1  2
Array after sorting : 1  2  3  7
```

결과화면