Computer Architecture

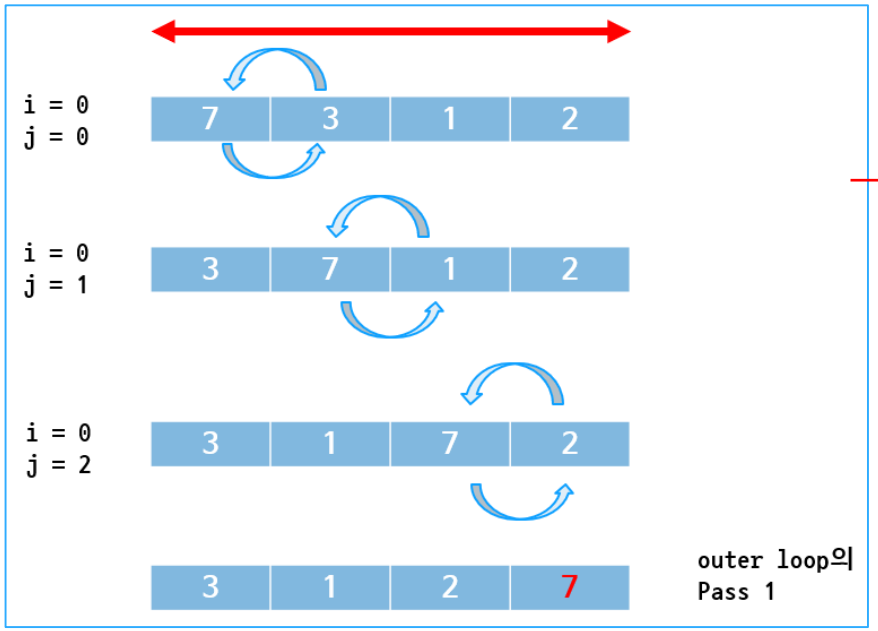
Network Computing Lab.

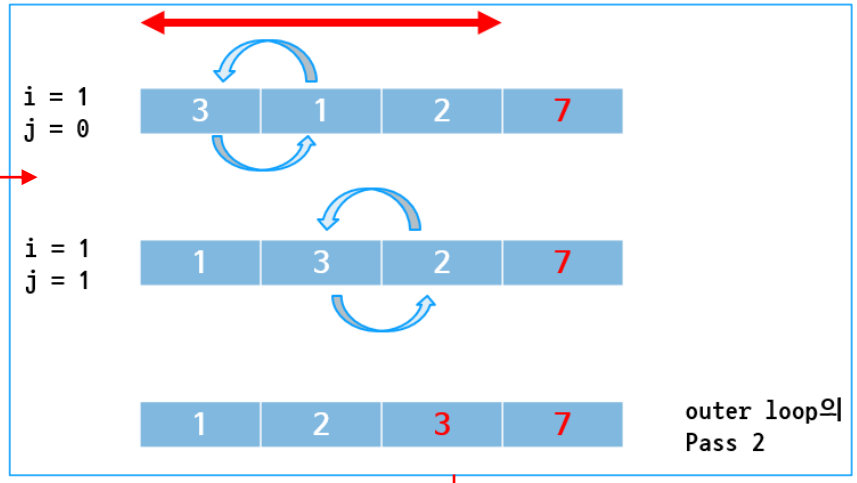
Arm assembly Lab 3

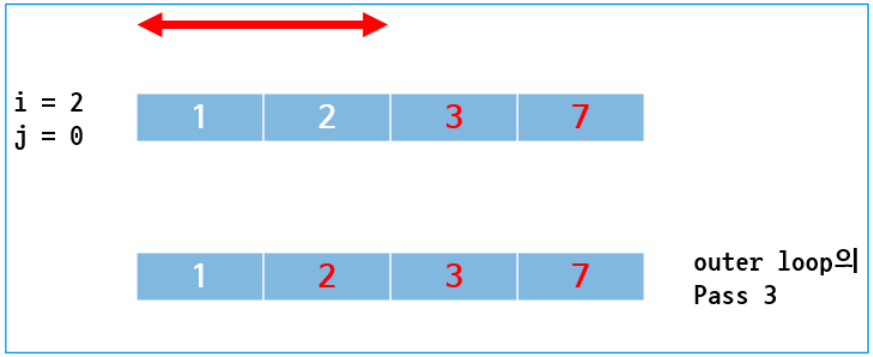
Contents

- **Lab3 : Debugger를 이용한 Bubble Sorting 프로그램 (교재내 코드) 동작 이해**
  - 3-1 : Outer loop 수행 끝날 때마다 배열 값의 변동 추적
  - 3-2 : Inner loop 수행시 배열 값의 변동 추적
  - 3-3 : swap 함수 call 전후 레지스터 저장 및 복구 루틴을 삭제후, swap 함수 call 전후 i,j 값 추적
  - 3-4 : **bubble sorting(다른 방법) 어셈블리 코드 작성 및 실행하기**
  - HW7: 위 실습 결과를 캡쳐/설명 추가하여 보고서로 제출하세요 .

Lab 3 : bubble sort(방법1)





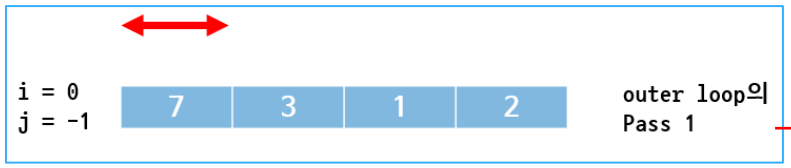



```
void sort (int v[], int n)
{
  int i, j;
  for (i = 0; i < n-1; i ++)

    for (j = 0; j < n-1-i; j ++)

      if (v[j] > v[j + 1]) swap(v,j);
}
```

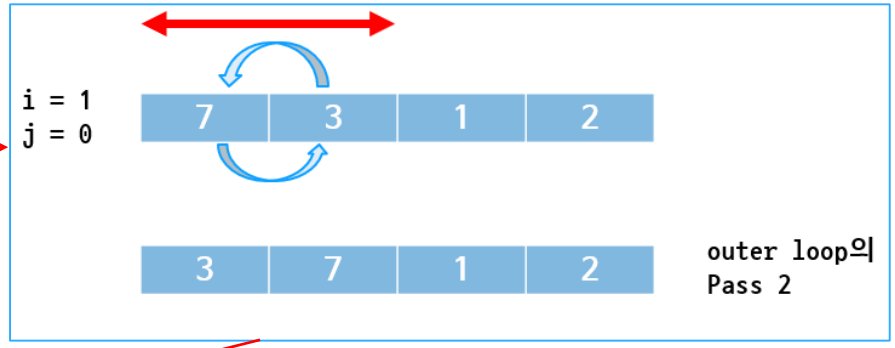
Lab 3 : bubble sort (방법2-교재내용)

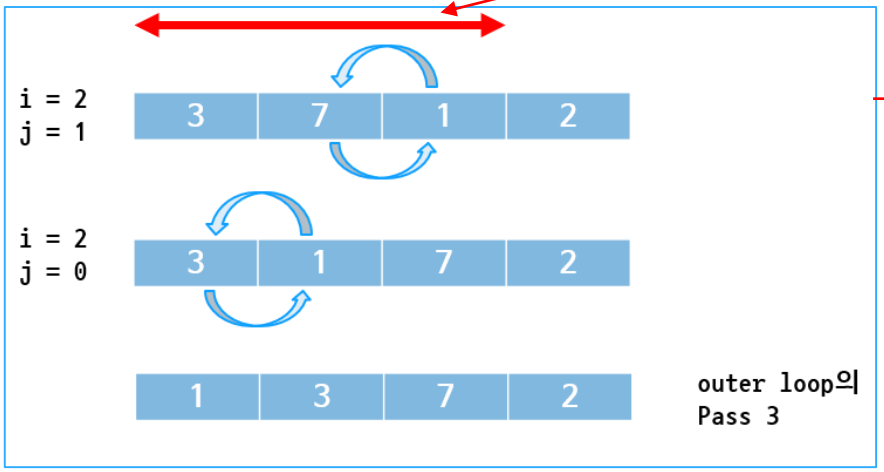


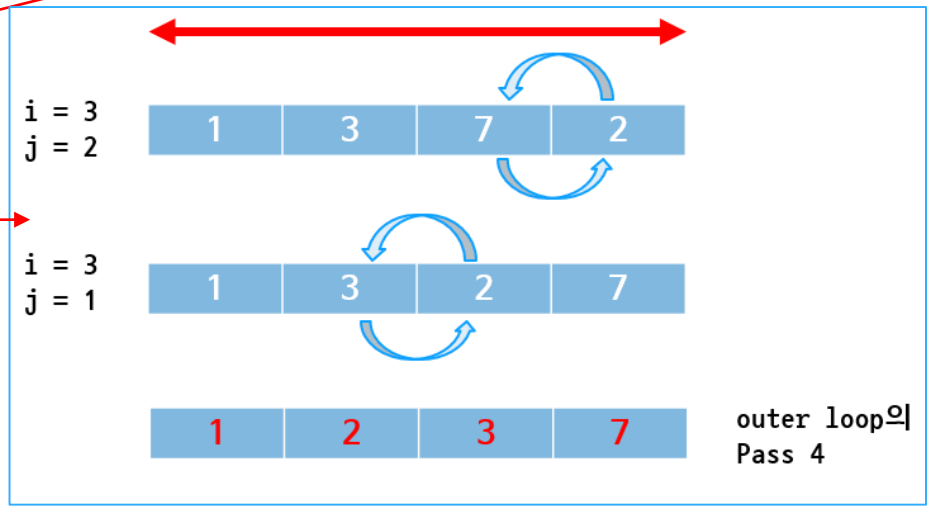

```
void sort (int v[], int n)
{
  int i, j;
  for (i = 0; i < n; i ++)

    for (j = i - 1; j >= 0 ; j --)

      if (v[j] > v[j + 1]) swap(v,j);
}
```







Register allocation and saving registers for *sort* [강의노트 Ch2-Part3-P29]

Network Computing Lab.

Register allocation

v	**RN 0**	**; 1st argument address of v**
n	**RN 1**	**; 2nd argument index n**
i	**RN 2**	**; local variable i**
j	**RN 3**	**; local variable j**
vjAddr	**RN 12**	**; to hold address of v[j]**
vj	**RN 4**	**; to hold a copy of v[j]**
vj1	**RN 5**	**; to hold a copy of v[j+1]**
vcopy	**RN 6**	**; to hold a copy of v**
ncopy	**RN 7**	**; to hold a copy of n**

Saving registers

```
sort:   SUB   sp,sp,#20          ; make room on stack for 5 registers
        STR   lr, [sp, #16]      ; save lr on stack
        STR   ncopy, [sp, #12]   ; save ncopy on stack
        STR   vcopy,  [sp, #8]   ; save vcopy on stack
        STR   j,  [sp, #4]       ; save j on stack
        STR   i,  [sp, #0]       ; save i on stack
```

Procedure body – *sort*

[강의노트 Ch2-Part3-P30-방법2]

```
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);
}
```

Move parameters		MOV	vcopy, v	; copy parameter v into vcopy (save r0)
		MOV	ncopy, n	; copy parameter n into ncopy (save r1)
Outer loop		MOV	i, #0	; i = 0
	for1tst:	CMP	i, n	; if i ≥ n
		BGE	exit1	; go to exit1 if i ≥ n
Inner loop		SUB	j, i, #1	; j = i - 1
	for2tst:	CMP	j, #0	; if j < 0
		BLT	exit2	; go to exit2 if j < 0
		ADD	vjAddr, v, j, LSL #2	; reg vjAddr = v + (j * 4)
		LDR	vj, [vjAddr,#0]	; reg vj = v[j]
		LDR	vj1, [vjAddr,#4]	; reg vj1 = v[j + 1]
		CMP	vj, vj1	; if vj ≤ vj1
		BLE	exit2	; go to exit2 if vj ≤ vj1
Pass parameters and call		MOV	r0, vcopy	; first swap parameter is v
		MOV	r1, j	; second swap parameter is j
		BL	swap	; swap code shown in Figure 2.23
Inner loop		SUB	j, j, #1	; j -= 1
		B	for2tst	; branch to test of inner loop
Outer loop	exit2:	ADD	i, i, #1	; i += 1
		B	for1tst	; branch to test of outer loop

소스코드1: Main 함수

Network Computing Lab.

- Source code

```c
#include <stdio.h>

extern void bubblesort(int arr[], int index);


int main()
{
        int arrOri[4] = {7,3,1,2};
        int i;

        printf("Array before sorting :");
        for(i = 0; i < sizeof(arr)/sizeof(int); i++)
                printf(" %d ",arr[i]);
        printf("\n");

        bubblesort(arr, 10);

        printf("Array after sorting :");
        for(i = 0; i < sizeof(arr)/sizeof(int); i++)
                printf(" %d ",arr[i]);
        printf("\n");

        return 0;
}
```

소스코드2 : 어셈블리 bubble sorting

Network Computing Lab.

- Source code

```
 1 .text
 2 .global bubblesort
 3 .global swap
 4 .type bubblesort, STT_FUNC
 5
 6 bubblesort:
 7                 sub     sp, sp, #20
 8                 str     lr, [sp, #16]
 9                 str     r7, [sp, #12]
10                 str     r6, [sp, #8]
11                 str     r3, [sp, #4]
12                 str     r2, [sp, #0]
13
14                 mov     r6, r0
15                 mov     r7, r1
16                 mov     r2, #0
17
18 for1tst:
19                 cmp     r2, r1
20                 bge     exit1
21                 sub     r3, r2, #1
22
23 for2tst:
24                 cmp     r3, #0
25                 blt     exit2
26                 add     r12, r0, r3, LSL #2
27                 ldr     r4, [r12, #0]
28                 ldr     r5, [r12, #4]
29                 cmp     r4, r5
30                 ble     exit2
31
```

```
32                 stmdb   sp!,{r0,r1,r2,r3,r12}
33                 mov     r0, r6
34                 mov     r1, r3
35                 bl      swap
36                 ldmia   sp!,{r0,r1,r2,r3,r12}
37
38                 sub     r3, r3, #1
39                 b       for2tst
40
41 exit2:
42                 add     r2, r2, #1
43                 b       for1tst
44
45 exit1:
46                 ldr     r2, [sp, #0]
47                 ldr     r3, [sp, #4]
48                 ldr     r6, [sp, #8]
49                 ldr     r7, [sp, #12]
50                 ldr     lr, [sp, #16]
51                 add     sp, sp, #20
52
53                 mov     pc, lr
54
55 .end
```

소스코드3 : 어셈블리 swap 코드

Network Computing Lab.

- **Source code 분석**

```
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);
}
```

```
.text
.global swap
.type swap, STT_FUNC

swap:
        add     r12, r0, r1, lsl #2

        ldr     r2, [r12, #0]
        ldr     r3, [r12, #4]

        str     r3, [r12, #0]
        str     r2, [r12, #4]

        mov     pc, lr

.end
```

Swap은 말그대로 배열의 두 원소의 값을 서로의 값으로 바꿉니다.

Lab 3 : 소스코드분석(1/13)

Network Computing Lab.

- Source code (bubblesort.s)

```
 1 .text
 2 .global bubblesort
 3 .global swap
 4 .type bubblesort, STT_FUNC
 5
 6 bubblesort:
 7                 sub     sp, sp, #20
 8                 str     lr, [sp, #16]
 9                 str     r7, [sp, #12]
10                 str     r6, [sp, #8]
11                 str     r3, [sp, #4]
12                 str     r2, [sp, #0]
13
14                 mov     r6, r0
15                 mov     r7, r1
16                 mov     r2, #0
17
18 for1tst:
19                 cmp     r2, r1
20                 bge     exit1
21                 sub     r3, r2, #1
22
23 for2tst:
24                 cmp     r3, #0
25                 blt     exit2
26                 add     r12, r0, r3, LSL #2
27                 ldr     r4, [r12, #0]
28                 ldr     r5, [r12, #4]
29                 cmp     r4, r5
30                 ble     exit2
31
32                 stmdb   sp!,{r0,r1,r2,r3,r12}
33                 mov     r0, r6
34                 mov     r1, r3
35                 bl      swap
36                 ldmia   sp!,{r0,r1,r2,r3,r12}
37
38                 sub     r3, r3, #1
39                 b       for2tst
40
41 exit2:
42                 add     r2, r2, #1
43                 b       for1tst
44
45 exit1:
46                 ldr     r2, [sp, #0]
47                 ldr     r3, [sp, #4]
48                 ldr     r6, [sp, #8]
49                 ldr     r7, [sp, #12]
50                 ldr     lr, [sp, #16]
51                 add     sp, sp, #20
52
53                 mov     pc, lr
54
55 .end
```

Lab 3 : 소스코드분석(2/13)

Network Computing Lab.

- Source code

```
bubblesort:
        sub     sp, sp, #20
        str     lr, [sp, #16]
        str     r7, [sp, #12]
        str     r6, [sp, #8]
        str     r3, [sp, #4]
        str     r2, [sp, #0]
```

자, bubblesort 함수의 시작입니다.
본격적으로 sorting하기 전, Local 변수를 사용하기 위해 stack에 레지스터들을 저장하는 작업을 하고 있네요.

이는 bubblesort 이전의 흐름을 유지하기 위함인거죠.

Lab 3 : 소스코드분석(3/13)

Network Computing Lab.

- Source code (bubblesort.s)

```
 1 .text
 2 .global bubblesort
 3 .global swap
 4 .type bubblesort, STT_FUNC
 5
 6 bubblesort:
 7                 sub     sp, sp, #20
 8                 str     lr, [sp, #16]
 9                 str     r7, [sp, #12]
10                 str     r6, [sp, #8]
11                 str     r3, [sp, #4]
12                 str     r2, [sp, #0]
13
14                 mov     r6, r0
15                 mov     r7, r1
16                 mov     r2, #0
17
18 for1tst:
19                 cmp     r2, r1
20                 bge     exit1
21                 sub     r3, r2, #1
22
23 for2tst:
24                 cmp     r3, #0
25                 blt     exit2
26                 add     r12, r0, r3, LSL #2
27                 ldr     r4, [r12, #0]
28                 ldr     r5, [r12, #4]
29                 cmp     r4, r5
30                 ble     exit2
31
32                 stmdb   sp!,{r0,r1,r2,r3,r12}
33                 mov     r0, r6
34                 mov     r1, r3
35                 bl      swap
36                 ldmia   sp!,{r0,r1,r2,r3,r12}
37
38                 sub     r3, r3, #1
39                 b       for2tst
40
41 exit2:
42                 add     r2, r2, #1
43                 b       for1tst
44
45 exit1:
46                 ldr     r2, [sp, #0]
47                 ldr     r3, [sp, #4]
48                 ldr     r6, [sp, #8]
49                 ldr     r7, [sp, #12]
50                 ldr     lr, [sp, #16]
51                 add     sp, sp, #20
52
53                 mov     pc, lr
54
55 .end
```

Lab 3 : 소스코드분석(4/13)

Network Computing Lab.

Source code 분석

```
                mov     r6, r0
                mov     r7, r1
                mov     r2, #0
for1tst:
                cmp     r2, r1
                bge     exit1
```

호출 규약에 의해 r0에 배열의 시작 주소, r1에 인덱스 값을 가지고 왔었죠.
그리고 r2는 iterator i가 되겠네요.

초반에 i(r2)는 0일테고, R1은 4이니 exit1로 분기하지 않겠죠?

```
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);
}
```

Lab 3 : 소스코드분석(5/14)

- Source code (bubblesort.s)

```
 1 .text
 2 .global bubblesort
 3 .global swap
 4 .type bubblesort, STT_FUNC
 5
 6 bubblesort:
 7                 sub     sp, sp, #20
 8                 str     lr, [sp, #16]
 9                 str     r7, [sp, #12]
10                 str     r6, [sp, #8]
11                 str     r3, [sp, #4]
12                 str     r2, [sp, #0]
13
14                 mov     r6, r0
15                 mov     r7, r1
16                 mov     r2, #0
17
18 for1tst:
19                 cmp     r2, r1
20                 bge     exit1
21                 sub     r3, r2, #1
22
23 for2tst:
24                 cmp     r3, #0
25                 blt     exit2
26                 add     r12, r0, r3, LSL #2
27                 ldr     r4, [r12, #0]
28                 ldr     r5, [r12, #4]
29                 cmp     r4, r5
30                 ble     exit2
31
32                 stmdb   sp!,{r0,r1,r2,r3,r12}
33                 mov     r0, r6
34                 mov     r1, r3
35                 bl      swap
36                 ldmia   sp!,{r0,r1,r2,r3,r12}
37
38                 sub     r3, r3, #1
39                 b       for2tst
40
41 exit2:
42                 add     r2, r2, #1
43                 b       for1tst
44
45 exit1:
46                 ldr     r2, [sp, #0]
47                 ldr     r3, [sp, #4]
48                 ldr     r6, [sp, #8]
49                 ldr     r7, [sp, #12]
50                 ldr     lr, [sp, #16]
51                 add     sp, sp, #20
52
53                 mov     pc, lr
54
55 .end
```

Lab 3 : 소스코드분석(6/13)

Network Computing Lab.

Source code 분석

```
                sub     r3, r2, #1
for2tst:
                cmp     r3, #0
                blt     exit2
                add     r12, r0, r3, LSL #2
                ldr     r4, [r12, #0]
                ldr     r5, [r12, #4]
                cmp     r4, r5
                ble     exit2
```

두번째 for loop로 들어왔어요.
똑같이 iterator j의 값을 설정하고
V[j] > v[j+1]의 값을 비교하는 동작을 하고 있습니다.

Bubble sorting이 인접한 원소를 비교해서 정렬하는 방식이라고 했었죠.

```
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);
}
```

Lab 3 : 소스코드분석(7/13)

Network Computing Lab.

- Source code (bubblesort.s)

```
 1 .text
 2 .global bubblesort
 3 .global swap
 4 .type bubblesort, STT_FUNC
 5
 6 bubblesort:
 7                 sub     sp, sp, #20
 8                 str     lr, [sp, #16]
 9                 str     r7, [sp, #12]
10                 str     r6, [sp, #8]
11                 str     r3, [sp, #4]
12                 str     r2, [sp, #0]
13
14                 mov     r6, r0
15                 mov     r7, r1
16                 mov     r2, #0
17
18 for1tst:
19                 cmp     r2, r1
20                 bge     exit1
21                 sub     r3, r2, #1
22
23 for2tst:
24                 cmp     r3, #0
25                 blt     exit2
26                 add     r12, r0, r3, LSL #2
27                 ldr     r4, [r12, #0]
28                 ldr     r5, [r12, #4]
29                 cmp     r4, r5
30                 ble     exit2
31
```

```
32                 stmdb   sp!,{r0,r1,r2,r3,r12}
33                 mov     r0, r6
34                 mov     r1, r3
35                 bl      swap
36                 ldmia   sp!,{r0,r1,r2,r3,r12}
37
38                 sub     r3, r3, #1
39                 b       for2tst
40
41 exit2:
42                 add     r2, r2, #1
43                 b       for1tst
44
45 exit1:
46                 ldr     r2, [sp, #0]
47                 ldr     r3, [sp, #4]
48                 ldr     r6, [sp, #8]
49                 ldr     r7, [sp, #12]
50                 ldr     lr, [sp, #16]
51                 add     sp, sp, #20
52
53                 mov     pc, lr
54
55 .end
```

Lab 3 : 소스코드분석(8/13)

Network Computing Lab.

- **Source code 분석**

```
stmdb   sp!,{r0,r1,r2,r3,r12}
mov     r0, r6
mov     r1, r3
bl      swap
ldmia   sp!,{r0,r1,r2,r3,r12}
```

Bubble sort의 swap 조건에 부합하면 앞, 뒤 두 원소를 swap하는 함수로 분기하게 됩니다. STM(store multiple), LDM(load multiple을 사용하여 분기 전 stack에 저장해놓고 swap으로 분기합니다.
R0에는 배열의 base 주소, R1에는 swap하는 원소의 인덱스 값을 파라미터로 전달합니다.

```
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);
}
```

Lab 3 : 소스코드분석(9/13)

Network Computing Lab.

- **Source code 분석**

```
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);
}
```

swap.s

```
.text
.global swap
.type swap, STT_FUNC

swap:
        add     r12, r0, r1, lsl #2

        ldr     r2, [r12, #0]
        ldr     r3, [r12, #4]

        str     r3, [r12, #0]
        str     r2, [r12, #4]

        mov     pc, lr

.end
```

swap은 말그대로 배열의 두 원소의 값을 서로의 값으로 바꿉니다.

Lab 3 : 소스코드분석(10/13)

Network Computing Lab.

- Source code (bubblesort.s)

```
 1 .text
 2 .global bubblesort
 3 .global swap
 4 .type bubblesort, STT_FUNC
 5
 6 bubblesort:
 7                 sub     sp, sp, #20
 8                 str     lr, [sp, #16]
 9                 str     r7, [sp, #12]
10                 str     r6, [sp, #8]
11                 str     r3, [sp, #4]
12                 str     r2, [sp, #0]
13
14                 mov     r6, r0
15                 mov     r7, r1
16                 mov     r2, #0
17
18 for1tst:
19                 cmp     r2, r1
20                 bge     exit1
21                 sub     r3, r2, #1
22
23 for2tst:
24                 cmp     r3, #0
25                 blt     exit2
26                 add     r12, r0, r3, LSL #2
27                 ldr     r4, [r12, #0]
28                 ldr     r5, [r12, #4]
29                 cmp     r4, r5
30                 ble     exit2
31
```

```
32                 stmdb   sp!,{r0,r1,r2,r3,r12}
33                 mov     r0, r6
34                 mov     r1, r3
35                 bl      swap
36                 ldmia   sp!,{r0,r1,r2,r3,r12}
37
38                 sub     r3, r3, #1
39                 b       for2tst
40
41 exit2:
42                 add     r2, r2, #1
43                 b       for1tst
44
45 exit1:
46                 ldr     r2, [sp, #0]
47                 ldr     r3, [sp, #4]
48                 ldr     r6, [sp, #8]
49                 ldr     r7, [sp, #12]
50                 ldr     lr, [sp, #16]
51                 add     sp, sp, #20
52
53                 mov     pc, lr
54
55 .end
```

Lab 3 : 소스코드분석(11/13)

Network Computing Lab.

- **Source code 분석**

```
                sub     r3, r3, #1
                b       for2tst

exit2:
                add     r2, r2, #1
                b       for1tst
```

이 부분은 그냥 iterator가 증가 혹은 감소하는 부분입니다.

```
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);
}
```

Lab 3 : 소스코드분석(12/13)

Network Computing Lab.

- Source code (bubblesort.s)

```
 1 .text
 2 .global bubblesort
 3 .global swap
 4 .type bubblesort, STT_FUNC
 5
 6 bubblesort:
 7                 sub     sp, sp, #20
 8                 str     lr, [sp, #16]
 9                 str     r7, [sp, #12]
10                 str     r6, [sp, #8]
11                 str     r3, [sp, #4]
12                 str     r2, [sp, #0]
13
14                 mov     r6, r0
15                 mov     r7, r1
16                 mov     r2, #0
17
18 for1tst:
19                 cmp     r2, r1
20                 bge     exit1
21                 sub     r3, r2, #1
22
23 for2tst:
24                 cmp     r3, #0
25                 blt     exit2
26                 add     r12, r0, r3, LSL #2
27                 ldr     r4, [r12, #0]
28                 ldr     r5, [r12, #4]
29                 cmp     r4, r5
30                 ble     exit2
31
32                 stmdb   sp!,{r0,r1,r2,r3,r12}
33                 mov     r0, r6
34                 mov     r1, r3
35                 bl      swap
36                 ldmia   sp!,{r0,r1,r2,r3,r12}
37
38                 sub     r3, r3, #1
39                 b       for2tst
40
41 exit2:
42                 add     r2, r2, #1
43                 b       for1tst
44
45 exit1:
46                 ldr     r2, [sp, #0]
47                 ldr     r3, [sp, #4]
48                 ldr     r6, [sp, #8]
49                 ldr     r7, [sp, #12]
50                 ldr     lr, [sp, #16]
51                 add     sp, sp, #20
52
53                 mov     pc, lr
54
55 .end
```

Lab 3 : 소스코드분석(13/13)

Network Computing Lab.

- Source code 분석

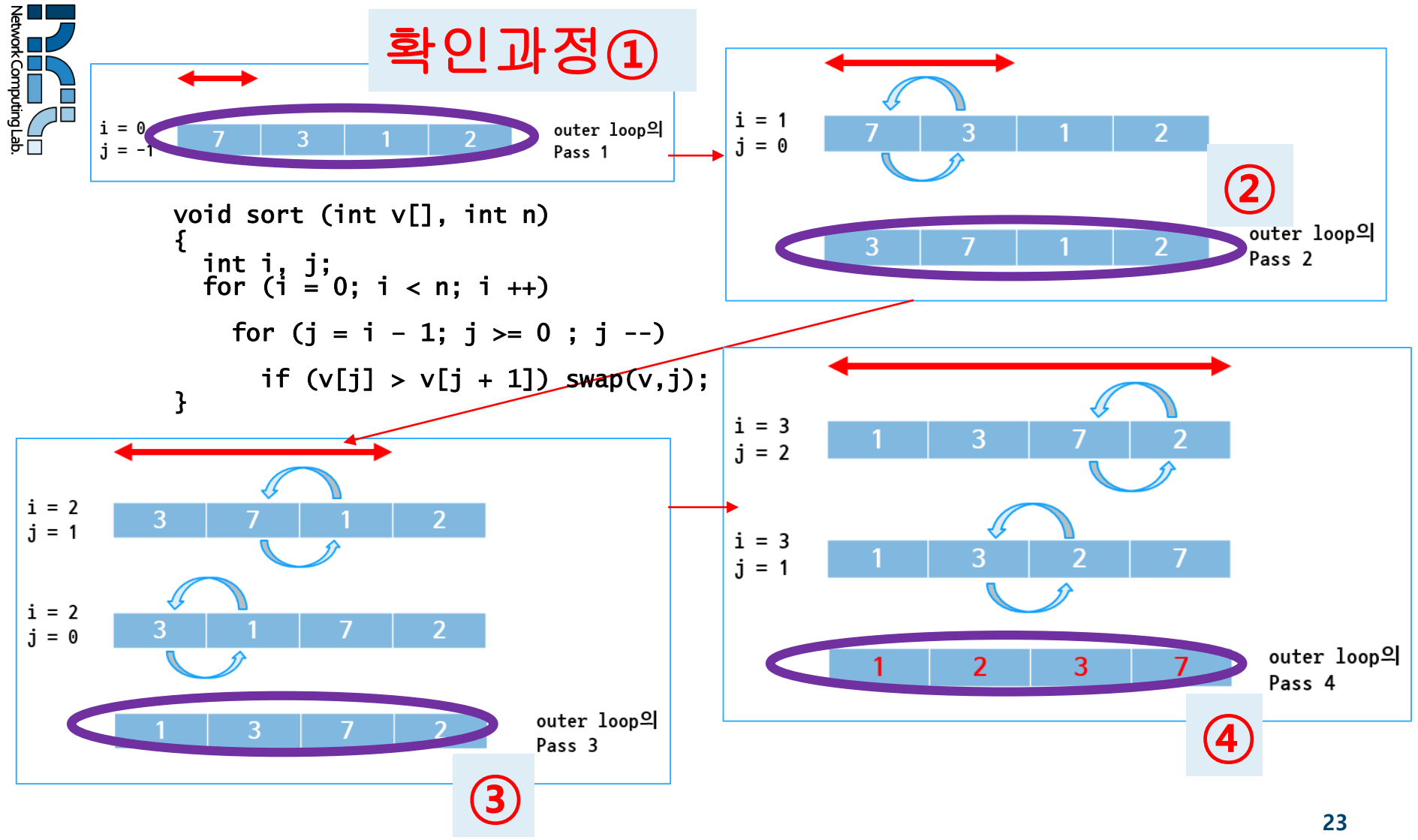
```
exit1:
                ldr     r2, [sp, #0]
                ldr     r3, [sp, #4]
                ldr     r6, [sp, #8]
                ldr     r7, [sp, #12]
                ldr     lr, [sp, #16]
                add     sp, sp, #20

                mov     pc, lr

.end
```

Bubble sort 함수 초반, 저장해 놓았던 register 값들을 다시 복원하는 작업입니다.
Sorting을 완료했으니
다시 main 함수로 돌아갑니다.

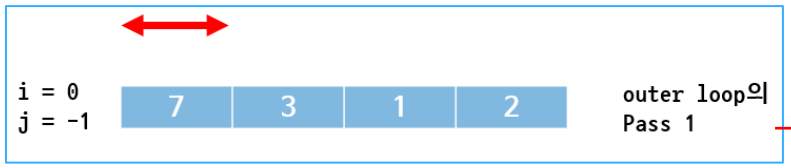
Lab 3-1 : Outer loop(i) 수행 끝날 때마다 배열 값의 변동 추적(1/2)



Lab 3-1 : Outer loop(i) 수행 끝날 때마다 배열 값의 변동 추적(2/2)

- 컴파일 후 gdbgui로 프로그램을 실행시킨다.
- Bubble sort 어셈블리 코드(P8) 14번 줄(mov r6, r0), 43번줄 (b for1st)에 breakpoint를 설정한다.
- 수행을 시작한다.
- 14번줄에서 멈추면 r0에 arr배열의 시작주소가 들어 있다. 메모리윈도우에 가서 **그 메모리주소를 입력하여 arr에 저장된 값들을 확인한다**. **화면 캡쳐한다.** 계속 수행한다.
- 43번줄에서 멈추면 **현재 r2값(변수i에 해당), arr에 저장된 값을 확인한다. P4에 있는 예제[그림]에서 outer loop의 pass i에 해당된 결과가 나왔는지 확인한** 후 **화면 캡쳐한다**. 계속 수행한다.
- arr 저장값 변화: (7,3,1,2) -> (3,7,1,2) ->(1,3,7,2) -> (1,2,3,7)

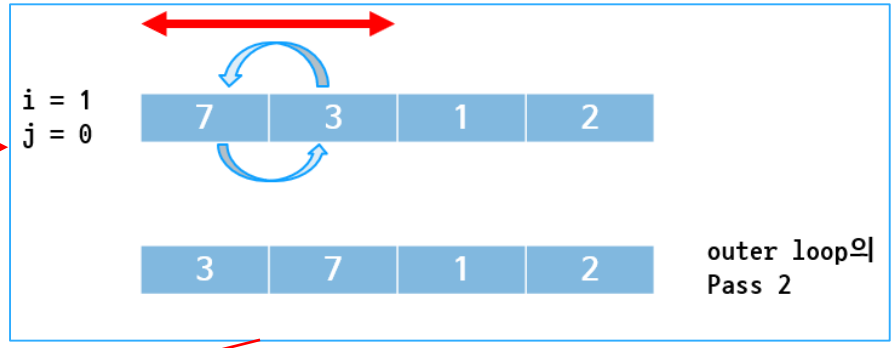
Lab 3-2 : i=2일 때 Inner loop(j) 수행시 배열 값의 변동 추적(1/3)

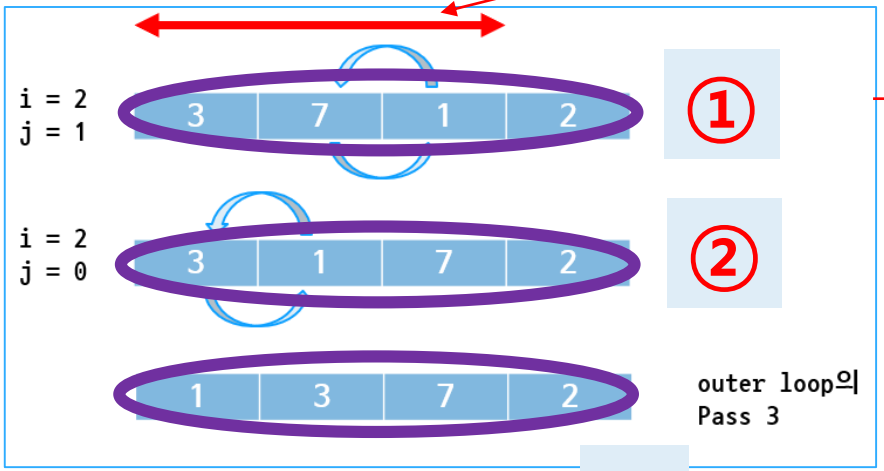


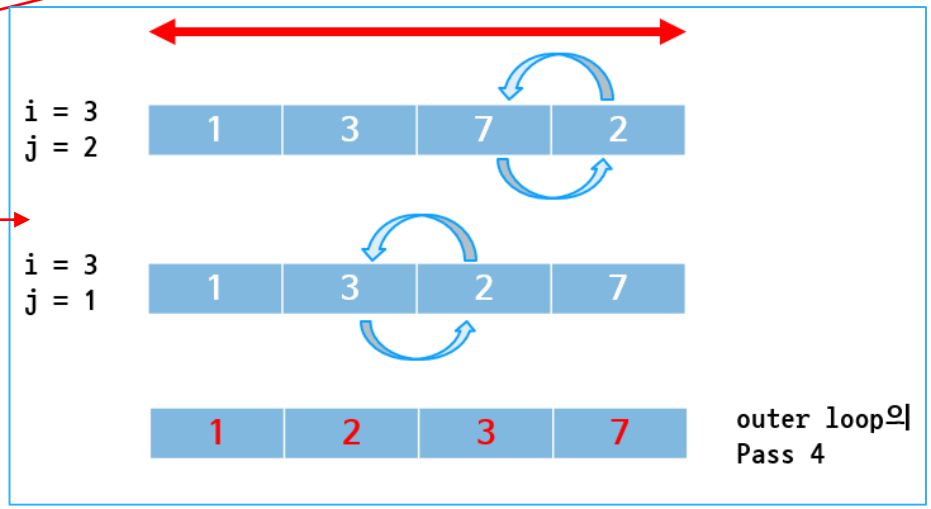

```
void sort (int v[], int n)
{
  int i, j;
  for (i = 0; i < n; i ++)

    for (j = i - 1; j >= 0 ; j --)

      if (v[j] > v[j + 1]) swap(v,j);
}
```







Lab 3-2 : i=2일 때 Inner loop(j) 수행시 배열 값의 변동 추적(2/3)

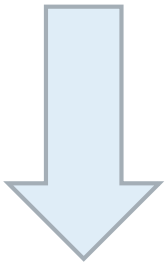
- 프로그램을 다시 실행한다.
- Bubble sort 어셈블리 코드(P8) 14번 줄(mov r6, r0), 43번 줄 (b for1tst)에 breakpoint를 설정한다.
- 수행을 시작한다.
- 14번줄에서 멈추면 r0에 arr 배열의 시작주소가 들어 있다. 메모리윈도우에 가서 **그 메모리주소를 입력하여 arr저장된 값들을 확인** 한 후 **화면 캡쳐한다**. 계속 수행한다.
- 43번줄에서 멈추면 현재 r2값(변수i에 해당) 확인
  - 만일 2가 아니면 계속 수행한다.
  - **만일 2라면 (i=2인 outer loop 수행하는 모드로 들어갈 예정임) 35번줄(bl swap), 36번줄(ldmia sp!, {r0, r1. r2, r3, r12}), 38번 줄 (sub r3, r3, #1, 여기서 r3는 j에 해당)에 breakpoint를 설정 한다.** 현재 arr에 저장된 값을 확인한다. 계속 수행한다.

Lab 3-2 : i=2일 때 Inner loop(j) 수행시 배열 값의 변동 추적(3/3)

- 35번줄에서 멈추면(swap 수행전) **현재 r3값(변수j에 해당), arr에 저장된 값을 확인한다**. **화면 캡쳐한다**. 계속 수행한다. 36번줄에서 멈추면 (swap 수행후) **현재 r3값(변수j에 해당), arr에 저장된 값을 확인한다**. **화면 캡쳐한다**. 35번줄에서 확인한 arr에 저장된 값과 비교한다.
- 43번줄에 서 멈추면 **현재 r2값 (변수i에 해당), arr에 저장된 값을 확인한다. P4에 있는 예제에서 outer loop의 pass i 에 해당된 결과가 나왔는지 확인한** 후 **화면 캡쳐한다.** **수행종료한다.**

Lab 3-3 : swap 함수 call 전후 레지스터 저장 및 복구 루틴을 삭제후, swap 함수 call 전후 i(=r2), j(=r3) 값 추적(1/2)


```
32                  stmdb   sp!,{r0,r1,r2,r3,r12}
33                  mov     r0, r6
34                  mov     r1, r3
35                  bl      swap
36                  ldmia   sp!,{r0,r1,r2,r3,r12}
37
```



- 아래처럼 코드 수정
- 35번 bl swap 수행전과 수행후 r2, r3 값 비교하기

```
33                  mov     r0, r6
34                  mov     r1, r3
35                  bl      swap
```

Lab 3-3 : swap 함수 call 전후 레지스터 저장 및 복구 루틴을 삭제후, swap 함수 call 전후 i(=r2), j(=r3) 값 추적(2/2)

- Bubble sort 어셈블리 코드 32번(stmdb sp!, {r0, r1. r2, r3, r12}), 36번 줄 (ldmia sp!, {r0, r1, r2, r3, r12}) 주석 처리(#) 후 재실행한다.
- 15번 줄(mov r6, r0), 43번줄 (b for1tst)에 breakpoint를 설정한다. 수행한다.
- 15번줄에서 멈추면 r0에 arr 배열의 시작주소가 들어 있다. 메모리윈도우에 가서 메모리주소를 입력하여 arr에 저장된 값들을 확인한다. 계속 수행한다.
- 43번줄에서 멈추면 현재 r2값(변수 i에 해당) 확인
  - 만일 0이 라면 계속 수행한다.
  - 만일 1이라면 (outer loop 수행하는 모드로 들어갈 예정임) 35번줄(bl swap), 38번줄 (sub r3, r3, #1) 에 breakpoint를 설정한다. 현재 arr에 저장된 값을 확인한다. 계속 수행한다.
- 35번줄에서 멈추면(swap 수행전) **현재 r2값(변수i에 해당), 현재 r3값(변수j에 해당), arr에 저장된 값을 확인한다. 화면 캡쳐한다**. 계속 수행한다. 38번줄에서 멈추면 (swap 수행후) **현재 r2값(변수i에 해당), 현재 r3값(변수j에 해당), arrOri 저장된 값을 확인한다. 화면 캡쳐한다**. 35번줄에서 확인한 r2, r3값과 38번줄에서 확인한 r2, r3값이 동일한지 확인한다. 동일하지 않으면 수행종료하고 동일하지 않게 된 이유를 분석한다. 보고서에 그 이유를 설명한다.

Lab 3-4 : bubble sorting(방법 1)로 어셈블리 코드 작성 및 실행하기

- P8에 있는 어셈블리 코드는 bubble sorting (방법 2 – 교재내용)으로 구현된 것이다. 이를 bubble sorting (P3에 있는 방법 1)으로 동작하도록 수정한다.
- 컴파일 후 수행한다.
- **원하는 수행결과가 나오는지 확인한다.**
- 만일 원하는 수행결과가 나오지 않으면 P3에 있는 예제에서 outer loop의 pass i에 해당된 결과가 나왔는지 단계별로 확인한다. (Lab 3-1 Outer loop 수행 끝날 때마다 배열 값의 변동 추적 방법을 이용하여 debugging 한다. 그래도 문제가 있으면 Lab 3-2 Inner loop 수행시 배열 값의 변동 추적 방법을 이용하여 debugging 한다)
- **어셈블리코드 및 수행결과를 화면 캡쳐 한다.**