< 과제 7 >

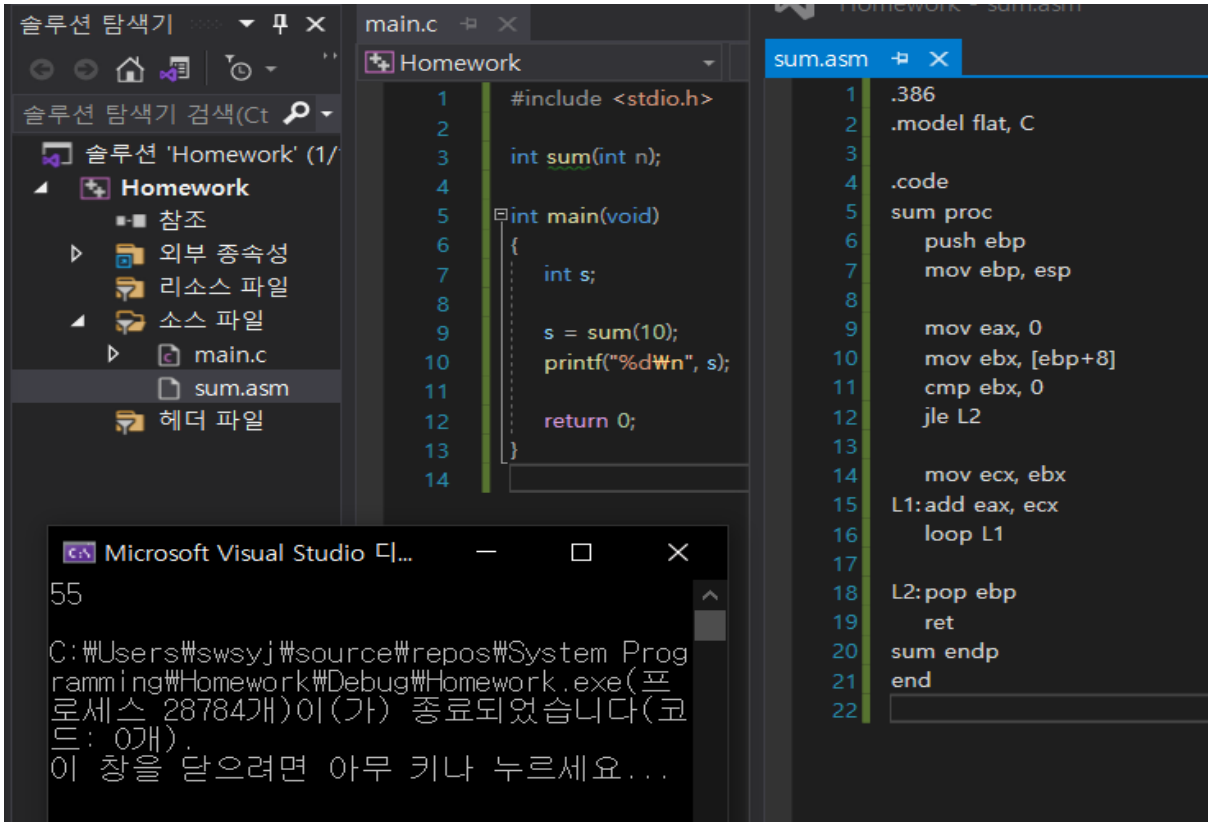
2017253041_홍성우

1.

-sum.asm 분석: ".model flat, C"이므로 sum proc, sum endp로 쓴 것이다. [ebp+8]은 n을 뜻하고 n과 0을 비교하여 작거나 같으면 L2로 jump하여 pop ebp한 후 ret하여 끝내고, n이 0보다 크면 ecx로 옮겨서 L1을 수행한다. L1은 ecx가 1씩 감소하며 0이 될때까지 n까지의 합을 구하는 것이다.

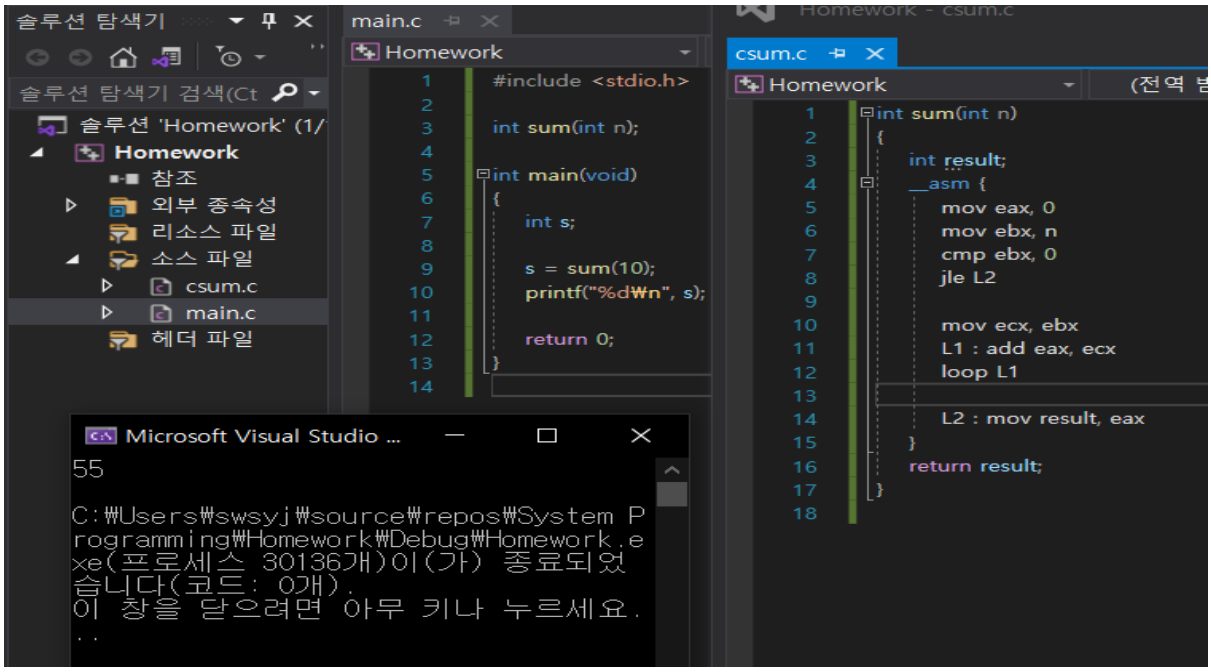
```
54  _TEXT SEGMENT
55  _sum$ = -8                      ; size = 4
56  _i$ = -4                        ; size = 4
57  _n$ = 8                         ; size = 4
58  _sum PROC
59  ; File C:₩Users₩swsyj₩source₩repos₩System Programming₩lab3₩sum.c
60  ; Line 16
61      push  ebp
62      mov   ebp, esp
63      sub   esp, 8
64  ; Line 18
65      mov   DWORD PTR _sum$[ebp], 0
66  ; Line 19
67      mov   DWORD PTR _i$[ebp], 1
68      jmp   SHORT $LN4@sum
69  $LN2@sum:
70      mov   eax, DWORD PTR _i$[ebp]
71      add   eax, 1
72      mov   DWORD PTR _i$[ebp], eax
73  $LN4@sum:
74      mov   ecx, DWORD PTR _i$[ebp]
75      cmp   ecx, DWORD PTR _n$[ebp]
76      jg  SHORT $LN3@sum
77  ; Line 20
78      mov   edx, DWORD PTR _sum$[ebp]
79      add   edx, DWORD PTR _i$[ebp]
80      mov   DWORD PTR _sum$[ebp], edx
81      jmp   SHORT $LN2@sum
82  $LN3@sum:
83  ; Line 21
84      mov   eax, DWORD PTR _sum$[ebp]
85  ; Line 22
86      mov   esp, ebp
87      pop   ebp
88      ret 0
89  _sum ENDP
90  _TEXT ENDS
```

-컴파일러가 생성한 어셈블리 프로그램 분석: 63번줄은 i, sum의 공간을 만든다. 65번줄은 sum에 0을 넣는 것이고 67번줄은 i에 1을 넣는 것이다. 그리고는 LN4로 jmp하는데 LN4는 i와 n을 비교하여 i가 n보다 클 때 LN3로 jump하여 sum을 eax에 옮기고 프로시져를 종료하여 return하고, i가 n보다 작거나 같으면 sum을 edx로 옮겨 i를 더하고, 더하고 나온 값을 다시 sum에 집어넣고 LN2로 jump하는 것이다. LN2는 i를 eax로 옮겨 1을 더하고, 더하고 나온 값을 다시 i에 집어넣는 것이다. 이 과정을 반복해서 ret 0을 실행하게 되면 프로시져는 종료된다.



-링커와 masm 빌드 규칙 설정을 해준뒤 C코드와 asm코드를 짜주고 빌드 후 실행시키면 55라는 결과값으로 올바르게 실행된다.

2.



-1번에서 main.c를 유지시키고 sum.asm만 지운뒤, inline asm코드를 짜서 실행시키면 55라는 결과값으로 올바르게 실행된다.

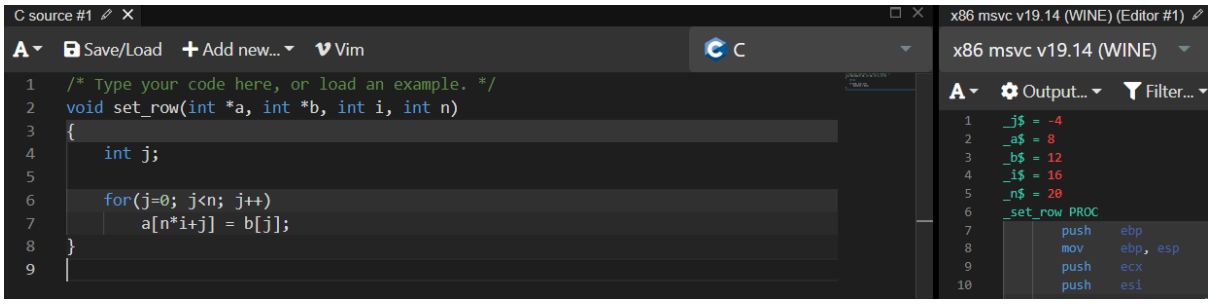
```
int sum(int n)
{
    int result;
    __asm {
        mov eax, 0
        mov ebx, n
        cmp ebx, 0
        jle L2

        mov ecx, ebx
        L1 : add eax, ecx
        loop L1

        L2 : mov result, eax
    }
    return result;
}
```

```
.386
.model flat, C

.code
sum proc
    push ebp
    mov ebp, esp

    mov eax, 0
    mov ebx, [ebp+8]
    cmp ebx, 0
    jle L2

    mov ecx, ebx
L1:add eax, ecx
    loop L1

L2:pop ebp
    ret
sum endp
end
```

-왼쪽이 inline asm코드이고 오른쪽이 sum.asm코드이다. 한눈에 보이듯 inline asm코드가 훨씬 짜기 편하고 간단하다. 또한 call, ret이 필요없으니 훨씬 효율적이다.

3.

(1)



(2)

-최적화하지 않은 것:

```
_j$ = -4                                                ; size = 4
_a$ = 8                                                 ; size = 4
_b$ = 12                                                ; size = 4
_i$ = 16                                                ; size = 4
_n$ = 20                                                ; size = 4
_set_row PROC
        push    ebp
        mov     ebp, esp
        push    ecx
        push    esi
        mov     DWORD PTR _j$[ebp], 0
        jmp     SHORT $LN4@set_row
$LN2@set_row:
        mov     eax, DWORD PTR _j$[ebp]
        add     eax, 1
        mov     DWORD PTR _j$[ebp], eax
$LN4@set_row:
        mov     ecx, DWORD PTR _j$[ebp]
        cmp     ecx, DWORD PTR _n$[ebp]
        jge     SHORT $LN1@set_row
        mov     edx, DWORD PTR _n$[ebp]
        imul    edx, DWORD PTR _i$[ebp]
        add     edx, DWORD PTR _j$[ebp]
        mov     eax, DWORD PTR _a$[ebp]
        mov     ecx, DWORD PTR _j$[ebp]
        mov     esi, DWORD PTR _b$[ebp]
        mov     ecx, DWORD PTR [esi+ecx*4]
        mov     DWORD PTR [eax+edx*4], ecx
        jmp     SHORT $LN2@set_row
$LN1@set_row:
        pop     esi
        mov     esp, ebp
        pop     ebp
        ret     0
_set_row ENDP
```

-최적화한 것:

```
_a$ = 8                                                 ; size = 4
_b$ = 12                                                ; size = 4
_i$ = 16                                                ; size = 4
_n$ = 20                                                ; size = 4
_set_row PROC                                           ; COMDAT
        push    ebp
        mov     ebp, esp
        xor     edx, edx
        cmp     DWORD PTR _n$[ebp], edx
        jle     SHORT $LN3@set_row
        mov     ecx, DWORD PTR _i$[ebp]
        imul    ecx, DWORD PTR _n$[ebp]
        mov     eax, DWORD PTR _a$[ebp]
        push    edi
        mov     edi, DWORD PTR _b$[ebp]
        lea     ecx, DWORD PTR [eax+ecx*4]
$LL4@set_row:
        mov     eax, DWORD PTR [edi+edx*4]
        inc     edx
        mov     DWORD PTR [ecx], eax
        lea     ecx, DWORD PTR [ecx+4]
        cmp     edx, DWORD PTR _n$[ebp]
        jl      SHORT $LL4@set_row
        pop     edi
$LN3@set_row:
        pop     ebp
        ret     0
_set_row ENDP
```

-set_row 함수에서 n*i는 for문과 무관하기 때문에 최적화를 하기 전에는 imul연산이 for문 안에서 실행되었다면, 최적화를 한 후에는 imul연산이 for문 밖에서 실행되어 컴파일된 결과의 asm코드의 길이도 짧아졌다.