

[Assembly Code 생성해보기]

```
#include<stdio.h>

int sum = 0;

void func1(int i, int j){
    int product;

    sum = i + j;
    product = i * j;
}

void main(){

    func1(3, 4);

}
```

먼저, 간단한 코드로 이루어진 **sample.c** 파일을 생성한다. 그 다음, 어셈블리어로 바꿔준 **sample.s** 파일도 생성한다.

```
[11/01/21]seed@VM:~/8week2$ ls
myid                               sample.c    sample.s
```

어셈블리 코드를 확인해보기 위해서, **cat sample.s** 를 통해서 보도록 한다.

```
[11/01/21]seed@VM:~/8week2$ cat sample.s
        .file     "sample.c"
        .globl   sum
        .bss
        .align   4
        .type    sum, @object
        .size    sum, 4
sum:
        .zero    4
        .text
        .globl   func1
        .type    func1, @function
func1:
.LFB0:
        .cfi_startproc
        pushl    %ebp
        .cfi_def_cfa_offset 8
        .cfi_offset 5, -8
        movl     %esp, %ebp
        .cfi_def_cfa_register 5
```

```

    subl    $16, %esp
    movl    8(%ebp), %edx
    movl    12(%ebp), %eax
    addl    %edx, %eax
    movl    %eax, sum
    movl    8(%ebp), %eax
    imull   12(%ebp), %eax
    movl    %eax, -4(%ebp)
    nop
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
.LFE0:
    .size    func1, .-func1
    .globl   main
    .type    main, @function
main:
.LFB1:
    .cfi_startproc

```

```

    pushl    %ebp
    .cfi_def_cfa_offset 8
    .cfi_offset 5, -8
    movl    %esp, %ebp
    .cfi_def_cfa_register 5
    pushl    $4
    pushl    $3
    call     func1
    addl     $8, %esp
    nop
    leave
    .cfi_restore 5
    .cfi_def_cfa 4, 4
    ret
    .cfi_endproc
.LFE1:
    .size    main, .-main
    .ident   "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5
    .note.gnu.build_id, \"20160609\"
    .section .note.GNU-stack,\"\",@progbits

```

어셈블리 코드는 정말 어려운 것 같다. 그래도 가장 중요한 부분은

pushl %ebp와 **movl %esp, %ebp**이다.

수업시간에 교수님께서 말씀하신 바로는, esp와 ebp는 cpu 레지스터의 일종인데, esp는 stack의 가장 첫번째 주소(맨 위에 위치해 있는 주소)를 의미하고, ebp는 현재 실행되고 있는 함수의 대표 위치 주소이다. 사실상, ebp의 주소를 기준으로 func1에 존재하는 지역변수인 i, j, sum의 위치를 알 수 있다. (위의 경우에는 ebp를 기준으로 **i = +8, j = +12, product = -4**)

[Debugging 해보기]

```

[11/01/21]seed@VM:~/8week2$ gcc -g -o sample_dbg sample
.C
[11/01/21]seed@VM:~/8week2$ gdb sample_dbg

```

먼저, 디버깅을 하기 위해서 gcc -g 명령어를 통해 디버깅용 명령을 실행시켜준다. (sample_dbg)

그 다음, gdb 명령을 통해 리눅스에서 프로그램을 단계별로 실행할 수 있도록 걸어주는 디버거를 실행시켜준다.

```
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from sample_dbg...done.
gdb-peda$
```

그러면, 다음과 같이 gdb-peda\$ 라고 명령 프롬프트가 변하게 되면서 디버깅을 실행할 수 있는 상태로 변환된 것을 확인해볼 수 있다.

```
gdb-peda$ b func1
Breakpoint 1 at 0x80483e1: file sample.c, line 9.
gdb-peda$
```

그 다음, 중단 지점을 선택하기 위해서 b func1로 중단점을 걸어준다.

```

[-----stack-----]
0000| 0xbfffed48 --> 0xbfffee0c --> 0xbffff019 ("XDG_VT
NR=7")
0004| 0xbfffed4c --> 0x8048431 (<__libc_csu_init+33>: )
0008| 0xbfffed50 --> 0xb7f1c3dc --> 0xb7f1d1e0 --> 0x0
0012| 0xbfffed54 --> 0x80481ec --> 0x1a
0016| 0xbfffed58 --> 0xbfffed68 --> 0x0
0020| 0xbfffed5c --> 0x8048407 (<main+12>:      add
esp,0x8)
0024| 0xbfffed60 --> 0x3
0028| 0xbfffed64 --> 0x4
[-----]
Legend: code, data, rodata, value

Breakpoint 1, func1 (i=0x3, j=0x4) at sample.c:9
9      sum = i + j;
gdb-peda$

```

```

gdb-peda$ info frame
Stack level 0, frame at 0xbfffed60:
  eip = 0x80483e1 in func1 (sample.c:9);
  saved eip = 0x8048407
  called by frame at 0xbfffed70
  source language c.
  Arglist at 0xbfffed58, args: i=0x3, j=0x4
  Locals at 0xbfffed58, Previous frame's sp is 0xbfffed60
  Saved registers:
    ebp at 0xbfffed58, eip at 0xbfffed5c
gdb-peda$

```

```

gdb-peda$ p $esp
$1 = (void *) 0xbfffed48
gdb-peda$ p $ebp
$2 = (void *) 0xbfffed58
gdb-peda$

```

```

gdb-peda$ p $ebp - $esp
$3 = 0x10
gdb-peda$

```

위와 같은 디버깅 과정으로, 레지스터 ebp와 esp의 실제 위치를 알 수 있고, ebp와 esp의 위치 차이도 알 수 있다.

