

Multithreading Applications in Win32

(Introduction to Assembly Code and Calling Convention)

Seong Jong Choi
chois@mmlab.net

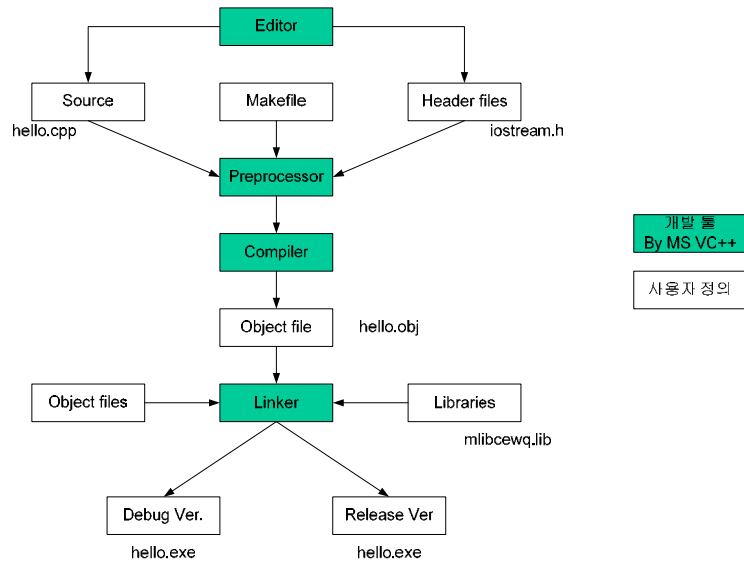
[Multimedia Lab.](#)

Dept. of Electrical and Computer Eng.
University of Seoul
Seoul, Korea

MS VC++

- Integrated Development Environment
 - Language Sensitive Text Editor
 - Preprocessor
 - Compiler
 - Linker
 - Wizard
- Project (.dsp)
 - A collection of all the necessary information to build a binary executables (.exe .dll)
 - Files (source, header)
 - Compile options
 - Link options
- Work Space (.dsw)
 - A collection of projects

Build Process



2005-09-30

Seong Jong Choi

Assembly Language-3

Assembly Code

- Project Setting -> C/C++ -> Category -> Listing files -> Listing file type -> Assembly, Machine Code, and Source
- Then, compile
- You'll see xxx.cod file in the debug directory

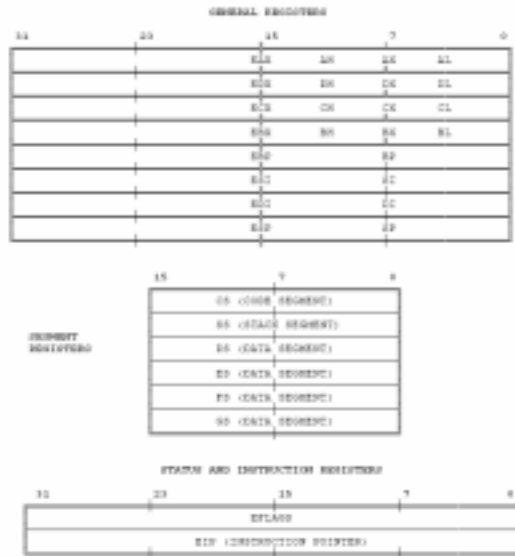
2005-09-30

Seong Jong Choi

Assembly Language-4

Intel 80386 Registers

Figure 2-1. 80386 Applications Register Set



Page 31 of 41

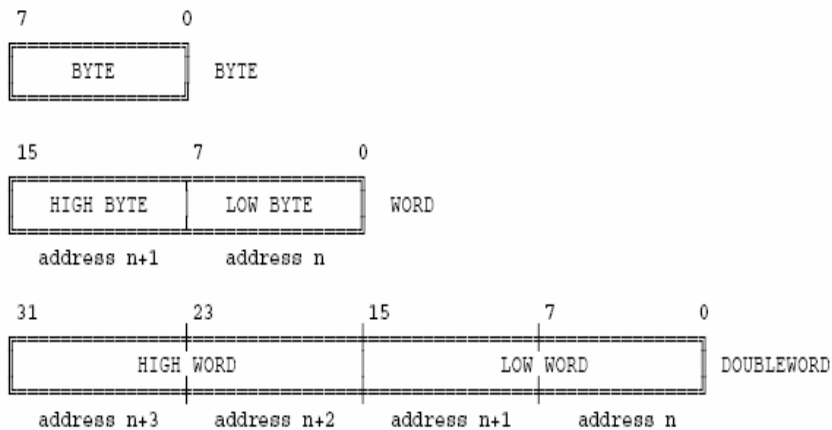
Seong Jong Choi

2005-09-30

Assembly Language-5

Intel Fundamental Data Type

Figure 2-2. Fundamental Data Types



2005-09-30

Seong Jong Choi

Assembly Language-6

Assembly code

- Instruction := operation [operand] [, operand]
- Examples
 - Data movement: mov destaddr, eax
 - Stack operation: pop eax
 - Arithmetic, logic, comparison, etc

Operand: Addressing Mode

- Immediate: Instruction
- Register Direct: Register
- Register indirect: Register
- Memory Direct: Memory
- Memory indirect: Memory
- Index: address +/-

Data Movement Instruction

- **mov** destination, source

```
_a$ = -4  
mov dword ptr _a$[ebp], 0Ah
```

Above two lines are equivalent to:

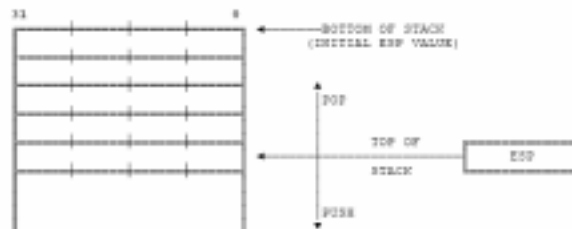
```
mov dword ptr [ebp-4], 0Ah
```

operation Operand: index + register indirect addressing Operand: Immediate addressing

Stack Instructions

- **PUSH**
 - Decrement the stack pointer (ESP)
 - Then, transfer source to the stack indicated by ESP
 - Push eax
- **POP**
 - Transfer data at the current top of stack (ESP)
 - Then, increment ESP
 - Pop eax

Figure 2-7. 9386 Stack



Stack Instructions

Figure 2-1. PUSH

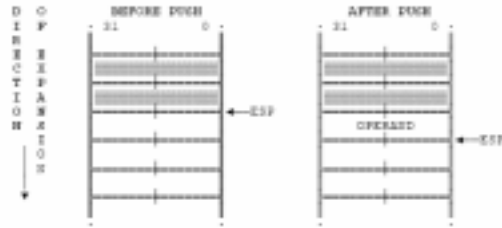
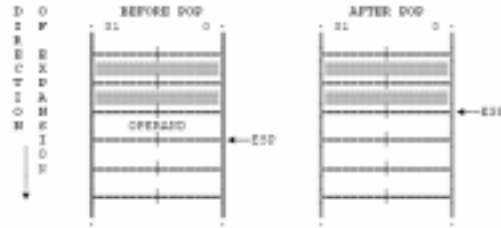


Figure 2-2. POP



Assembly Debugging

- View -> Debug windows
 - Register
 - Memory
 - Disassembly

An Example

```
/* simple.cpp
   demonstrating assembly language code generated by the compiler
*/
#include <windows.h>
int sum(int x, int y);
int WINAPI wsum(int x, int y);

void main() {
    int a, b, c;
    a = 10;
    b = 20;
    c = a + b;
    c = sum(a,b);
    c = wsum(a,b);
}

int sum(int x, int y) {
    int z;
    z = x + y;
    return z;
}

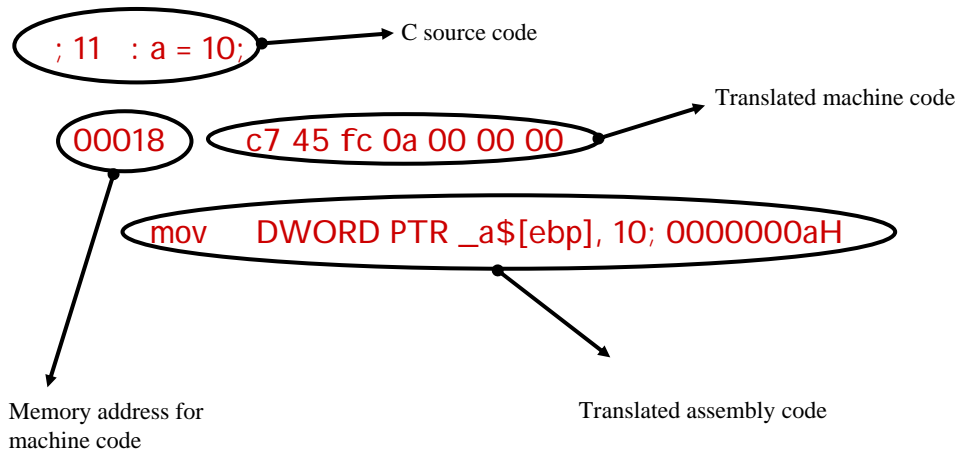
int WINAPI wsum(int x, int y) {
    int z;
    z = x + y;
    return z;
}
```

2005-09-30

Seong Jong Choi

Assembly Language-13

Simple.cod File

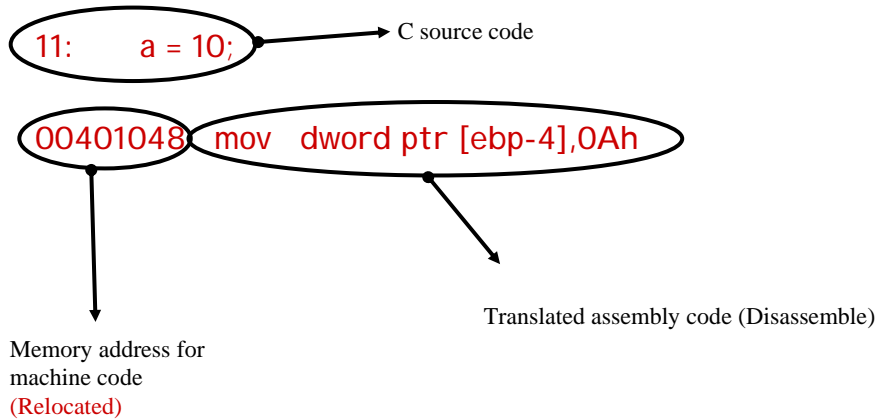


2005-09-30

Seong Jong Choi

Assembly Language-14

Disassembly Window



2005-09-30

Seong Jong Choi

Assembly Language-15

Function Call

Caller

```
: 14 : c = sum(a,b);  
  
mov ecx, DWORD PTR _b$[ebp]  
push ecx  
mov edx, DWORD PTR _a$[ebp]  
push edx  
call ?sum@YAHHH@Z : sum  
add esp, 8  
mov DWORD PTR _c$[ebp], eax
```

Callee

```
: COMDAT ?sum@YAHHH@Z  
_TEXT SEGMENT  
_x$ = 8  
_y$ = 12  
_z$ = -4  
?sum@YAHHH@Z PROC NEAR : sum, COMDAT  
  
: 19 : int sum(int x, int y) {  
  
push ebp  
mov ebp, esp  
sub esp, 68 ; 00000044H  
push ebx  
push esi  
push edi  
lea edi, DWORD PTR [ebp-68]  
mov ecx, 17 ; 00000011H  
mov eax, -858993460 ; cccccccH  
rep stosd  
  
: 20 : int z;  
: 21 : z = x + y;  
  
mov eax, DWORD PTR _x$[ebp]  
add eax, DWORD PTR _y$[ebp]  
mov DWORD PTR _z$[ebp], eax  
  
: 22 : return z;  
  
mov eax, DWORD PTR _z$[ebp]  
  
: 23 : }  
  
pop edi  
pop esi  
pop ebx  
mov esp, ebp  
pop ebp  
ret 0  
?sum@YAHHH@Z ENDP : sum  
_TEXT ENDS
```

2005-09-30

Seong Jong Choi

Assembly Language-16

Before Function Call

- Assume:
 - $esp = n + 4$
 - $ebp = bbpp$
 - $edi = ddii$
 - $esi = ssii$
 - $ebx = bbxx$
- The above registers are used in the callee.

2005-09-30

Seong Jong Choi

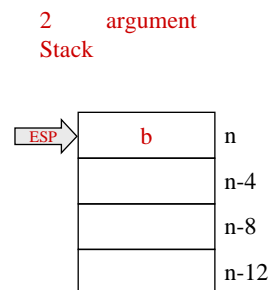
Assembly Language-17

Function Call - Caller

```
; 14 :      c = sum(a,b); //esp = n+4
```

```
mov     ecx, DWORD PTR _b$[ebp]  
push   ecx
```

```
mov     edx, DWORD PTR _a$[ebp]  
push   edx  
call   ?sum@@YAHHH@Z ; sum  
add    esp, 8  
mov    DWORD PTR _c$[ebp], eax
```



2005-09-30

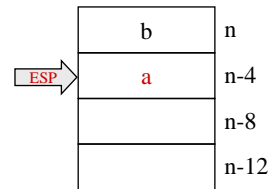
Seong Jong Choi

Assembly Language-18

Function Call - Caller

```
; 14 :      c = sum(a,b);  
  
mov     ecx, DWORD PTR _b$[ebp]  
push   ecx  
mov     edx, DWORD PTR _a$[ebp]  
push   edx  
call   ?sum@@YAH@Z ; sum  
add    esp, 8  
mov    DWORD PTR _c$[ebp], eax
```

1 argument
Stack



2005-09-30

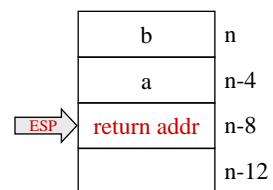
Seong Jong Choi

Assembly Language-19

Function Call - Caller

```
; 14 :      c = sum(a,b);  
  
mov     ecx, DWORD PTR _b$[ebp]  
push   ecx  
mov     edx, DWORD PTR _a$[ebp]  
push   edx  
call   ?sum@@YAH@Z ; sum  
return addr: add esp, 8  
mov    DWORD PTR _c$[ebp], eax
```

Return address
Stack



2005-09-30

Seong Jong Choi

Assembly Language-20

Function Call - Callee

```
; 19 : int sum(int x, int y) {
```

```
push    ebp; [ebp] = bbpp
```

```
movebp, esp
```

```
sub esp, 68 ;00000044H
```

```
push    ebx
```

```
push    esi
```

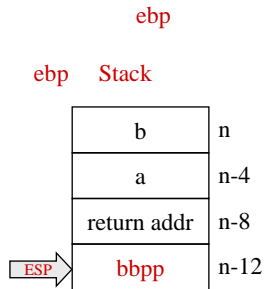
```
push    edi
```

```
lea edi, DWORD PTR [ebp-68]
```

```
mov ecx, 17 ; 00000011H
```

```
mov eax, -858993460; cccccccH
```

```
rep stosd
```



2005-09-30

Seong Jong Choi

Assembly Language-21

Function Call - Callee

```
; 19 : int sum(int x, int y) {
```

```
push    ebp; [ebp] = bbpp
```

```
move    bp, esp
```

```
sub esp, 68 ;00000044H
```

```
push    ebx
```

```
push    esi
```

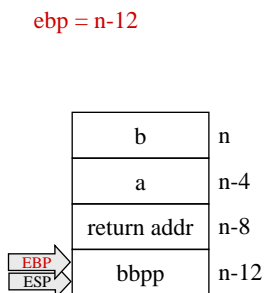
```
push    edi
```

```
lea edi, DWORD PTR [ebp-68]
```

```
mov ecx, 17 ; 00000011H
```

```
mov eax, -858993460; cccccccH
```

```
rep stosd
```



2005-09-30

Seong Jong Choi

Assembly Language-22

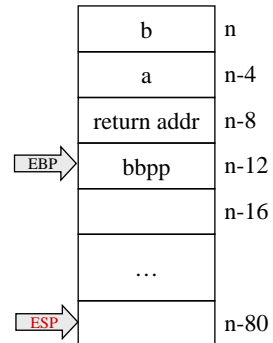
Function Call - Callee

```
; 19 : int sum(int x, int y) {
```

esp = n-80

```

push    ebp; [ebp] = bbpp
move    bp, esp
sub     esp, 68 ;00000044H
push    ebx
push    esi
push    edi
lea     edi, DWORD PTR [ebp-68]
mov     ecx, 17 ; 00000011H
mov     eax, -858993460; cccccccH
rep stosd
    
```



2005-09-30

Seong Jong Choi

Assembly Language-23

Function Call - Callee

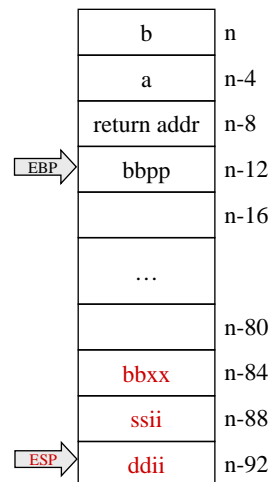
```
; 19 : int sum(int x, int y) {
```

Register

Stack

```

push    ebp; [ebp] = bbpp
move    bp, esp
sub     esp, 68 ;00000044H
push    ebx
push    esi
push    edi
lea     edi, DWORD PTR [ebp-68]
mov     ecx, 17 ; 00000011H
mov     eax, -858993460; cccccccH
rep stosd
    
```



2005-09-30

Seong Jong Choi

Assembly Language-24

Function Call - Callee

```
; 19 : int sum(int x, int y) {
```

```

push    ebp; [ebp] = bbpp
move    bp, esp
sub     esp, 68 ; 00000044H
push    ebx
push    esi
push    edi

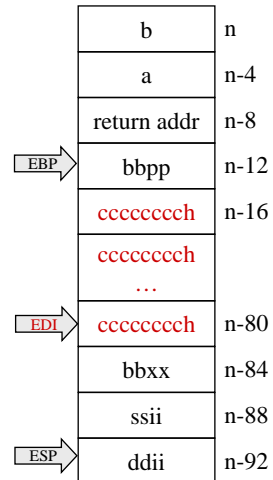
```

```

lea    edi, DWORD PTR [ebp-68]
mov    ecx, 17 ; 00000011H
mov    eax, -858993460; cccccccH
rep stosd

```

Stack
cch
4 x 17 = 68



2005-09-30

Seong Jong Choi

Assembly Language-25

Function Call - Callee

```
_x$ = 8
```

```
_y$ = 12
```

```
_z$ = -4
```

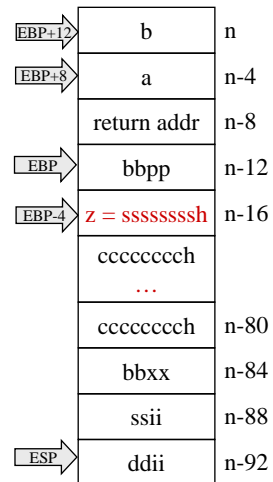
```
; 21 :      z = x + y;
```

```

mov    eax, DWORD PTR _x$[ebp]
add    eax, DWORD PTR _y$[ebp]
mov    DWORD PTR _z$[ebp], eax

```

a b
[ebp-4]



2005-09-30

Seong Jong Choi

Assembly Language-26

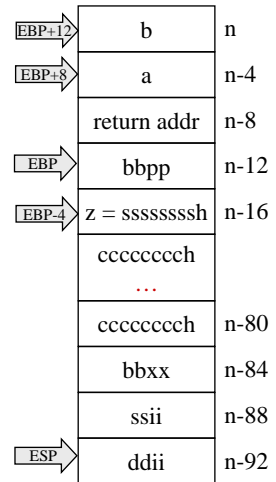
Function Call - Callee

```
; 22 : return z;
```

```
mov    eax, DWORD PTR _z$[ebp]
```

Return eax

eax = sssssssh



2005-09-30

Seong Jong Choi

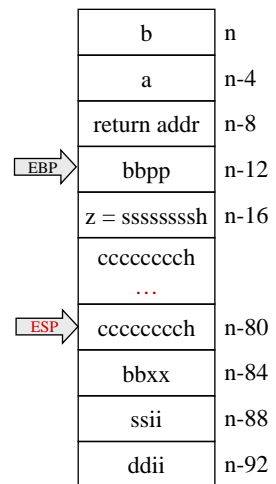
Assembly Language-27

Function Call - Callee

```
; 23 : }
```

```
pop    edi
pop    esi
pop    ebx
mov    esp, ebp
pop    ebp
ret    0
```

Register



2005-09-30

Seong Jong Choi

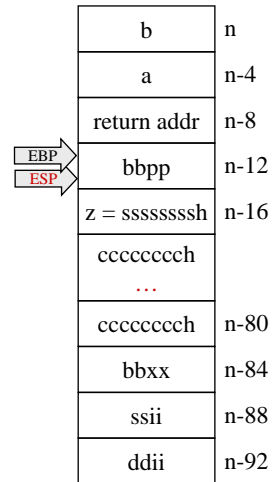
Assembly Language-28

Function Call - Callee

```
; 23 :}
```

esp 68

```
pop    edi
pop    esi
pop    ebx
mov    esp, ebp
pop    ebp
ret    0
```



2005-09-30

Seong Jong Choi

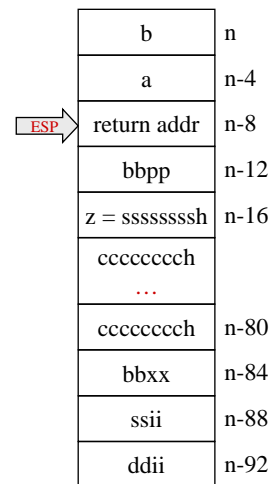
Assembly Language-29

Function Call - Callee

```
; 23 :}
```

ebp
ebp = bbpp

```
pop    edi
pop    esi
pop    ebx
mov    esp, ebp
pop    ebp
ret    0
```



2005-09-30

Seong Jong Choi

Assembly Language-30

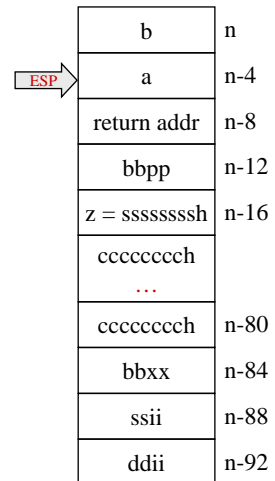
Function Call - Callee

```
; 23 : }
```

```
pop    edi
pop    esi
pop    ebx
mov    esp, ebp
pop    ebp
```

```
ret    0
```

caller 가
eip
eip = return addr



2005-09-30

Seong Jong Choi

Assembly Language-31

Function Call - Caller

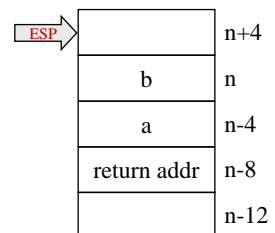
```
; 14 :    c = sum(a,b);
```

```
mov    ecx, DWORD PTR _b$[ebp]
push   ecx
mov    edx, DWORD PTR _a$[ebp]
push   edx
call   ?sum@@YAH@Z ; sum
```

```
return addr: add    esp, 8
```

```
mov    DWORD PTR _c$[ebp], eax
```

esp



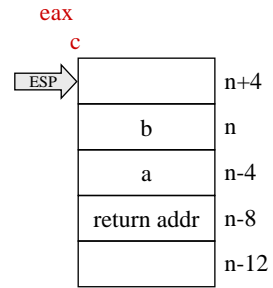
2005-09-30

Seong Jong Choi

Assembly Language-32

Function Call - Caller

```
; 14 :      c = sum(a,b);  
  
      mov    ecx, DWORD PTR _b$[ebp]  
      push  ecx  
      mov    edx, DWORD PTR _a$[ebp]  
      push  edx  
      call  ?sum@@YAH@Z ; sum  
return addr: add    esp, 8  
      mov    DWORD PTR _c$[ebp], eax
```



Function Call: Summary

- Stack is used to transfer function arguments.
- Function arguments are pushed to the stack **in the sequence from right.**
- Stack is used as a storage for callee's local variables.
- Registers are restored after returning from the callee.
- Return value (integer) is stored in eax .
- Caller sets up esp after the return ($add\ esp, 8$)

Return Instruction

- The difference between `sum()` and `wsum()` is that who cleans the stack for input parameters.
- Compare between `sum()` return instruction (`ret 0`) and `wsum()` return instruction (`ret 8`).
- `ret n`
 - `RET` transfers control to a return address located on the stack. The address is usually placed on the stack by a `CALL` instruction, and the return is made to the instruction that follows the `CALL`.
 - The optional numeric parameter to `RET` gives the number of stack bytes to be released after the return address is popped. These items are typically used as input parameters to the procedure called.

Calling Convention

Keyword	Stack cleanup	Parameter passing
<code>__cdecl</code> (C default)	Caller	Pushes parameters on the stack, in reverse order (right to left)
<code>__stdcall</code> (<code>#define WINAPI __stdcall</code>)	Callee	Pushes parameters on the stack, in reverse order (right to left)
<code>__fastcall</code>	Callee	Stored in registers, then pushed on stack
<code>thiscall</code> (not a keyword)	Callee	Pushed on stack; this pointer stored in ECX