Chapter 13:
고급언어 인터페이스

일반적 규약

- 고급언어에서 어셈블리 언어 프로시저 호출 시의 고려사항
  - 같은 작명 규약(naming convention)
    - 변수와 프로시저 이름 작성에 대한 규칙
  - 같은 메모리 모델 – flat, small 등
  - 같은 호출 규약
- 프로시저 호출 시의 고려 사항
  - 보존해야 할 register 명시
  - 인수 전달 방법
    - registers, stack, shared memory
  - 인수 전달 순서
  - 인수 전달 방식
    - pass by value, pass by reference
  - 프로시저 호출 후의 stack pointer 복원 방법
  - 함수 값 반환 방법

고급 언어와 외부 식별자

- 외부 식별자
  - module의 external object에 사용되는 이름
  - 고급 언어 프로그램의 변수, 프로시저 이름이 변형되어 사용됨
  - 어셈블리 언어 프로그램의 프로시저 이름도 언어 지정자에 따라서 약간 변형되어 사용됨
- 고급언어의 외부 식별자

		어셈블리 코드
(ex) C언어:	ArraySum( )	➔ _ArraySum
C++언어:	Inc(int n)	➔ ?MySub@@YAHH@Z
	Inc(double n)	➔ ?MySub@@YANN@Z

함수 overloading 지원을 위한 name decoration

C/C++ code에 대한 Assembly code 생성

- ASM code 생성을 위한 Visual C++ command-line option
  - /FA assembly only listing
  - /FAc assembly with machine code
  - /FAs assembly with source code
  - /FAcs assembly with machine code and source code

```
C> cl /FA prog.c
C> cl /FA prog.cpp
```

  - 컴파일을 수행할 때에 prog.asm파일을 함께 생성
- 생성된 Assembly code를 수정하여 최적화를 할 수 있음

언어 지정자

- 언어지정자
  - . model flat, C — C언어와 링크할 때 사용
  - . model flat, stdcall — Windows 함수를 호출할 때 사용
- C 지정자
  - 호출하는 측에서 스택 인수를 정리함
    - add esp, 8
  - export되는 프로시저 이름 앞에 _ 를 붙임
    - AddTwo → _AddTwo
- stdcall 지정자
  - 호출되는 프로시저에서 스택 인수를 정리함
    - ret 8
  - export되는 프로시저 이름에 다음과 같이 스택 인수 바이트 수를 표시
    - AddTwo → _AddTwo**@8**

C 언어 함수에 대한 Assembly Code

```
#include <stdio.h>
int addtwo(int a, int b)

int main(void)
{
    int res;

    res = addtwo(5, 6);
    printf("%d\n", res);
    return 0;
}

int addtwo(int a, int b)
{
    int s;

    s = a+b;
    return s;
}
```

⇩

```
PUBLIC  _addtwo
PUBLIC  _main
EXTRN   _printf:PROC
```

```
Line 8
        push    6
        push    5
        call    _addtwo
        add     esp, 8
        mov     DWORD PTR _res$[ebp], eax
-----------------------------------------

_s$ = -4
_a$ = 8
_b$ = 12
_addtwo PROC
; Line 14
        push    ebp
        mov     ebp, esp
        push    ecx
; Line 16
        mov     eax, DWORD PTR _a$[ebp]
        add     eax, DWORD PTR _b$[ebp]
        mov     DWORD PTR _s$[ebp], eax
; Line 17
        mov     eax, DWORD PTR _s$[ebp]
; Line 18
        mov     esp, ebp
        pop     ebp
        ret     0
_addtwo ENDP
```


stdcall 지정 C 언어 함수에 대한 Assembly Code

```
#include <stdio.h>
int __stdcall addtwo(int a, int b);

int main(void)
{
    int res;

    res = addtwo(5, 6);
    printf("%d\n", res);
    return 0;
}

int __stdcall addtwo(int a, int b)
{
    int s;

    s = a+b;
    return s;
}
```

⇩

```
PUBLIC  _addtwo@8
PUBLIC  _main
EXTRN   _printf:PROC
```

```
; Line 8
        push    6
        push    5
        call    _addtwo@8
        mov     DWORD PTR _res$[ebp], eax
-----------------------------------------

_s$ = -4
_a$ = 8
_b$ = 12
_addtwo@8 PROC
; Line 14
        push    ebp
        mov     ebp, esp
        push    ecx
; Line 17
        mov     eax, DWORD PTR _a$[ebp]
        add     eax, DWORD PTR _b$[ebp]
        mov     DWORD PTR _s$[ebp], eax
; Line 18
        mov     eax, DWORD PTR _s$[ebp]
; Line 19
        mov     esp, ebp
        pop     ebp
        ret     8
_addtwo@8 ENDP
```


C언어와 링크하기

- 기본 구조
  - C언어 프로그램 모듈
    - C code에서 어셈블리언어 프로지저 호출
      result = func(...)
  - 어셈블리 언어 프로그램 모듈
    - 어셈블리 언어 프로시저 정의

```
.386
.model flat, C
func proc


func  endp
end
```

또는

```
.386
.model flat
_func proc


_func  endp
end
```

예제

- C언어 프로그램 : cmain.c

```
#include <stdio.h>

int sum(int n);

int main(void)
{
        int s;

        s = sum(10);          // 1부터 10까지의 합
        printf("%d\n", s);

        return 0;
}
```

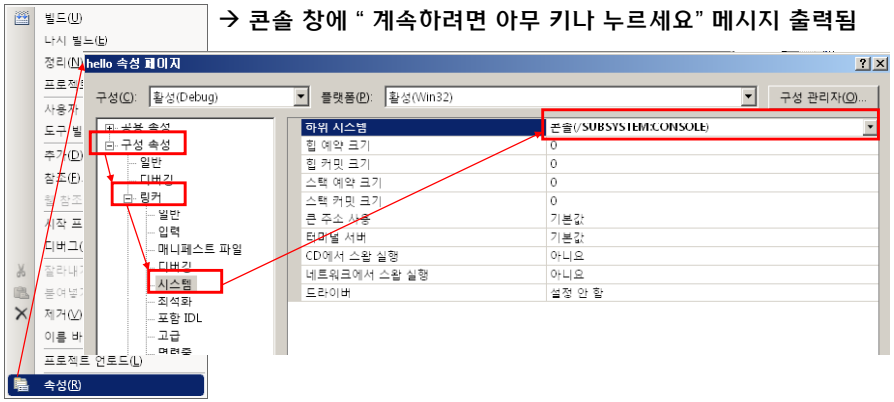

- 어셈블리언어 프로그램: sum.asm

```
.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:     leave
        ret
sum endp
end
```


컴파일/어셈블리를 위한 설정

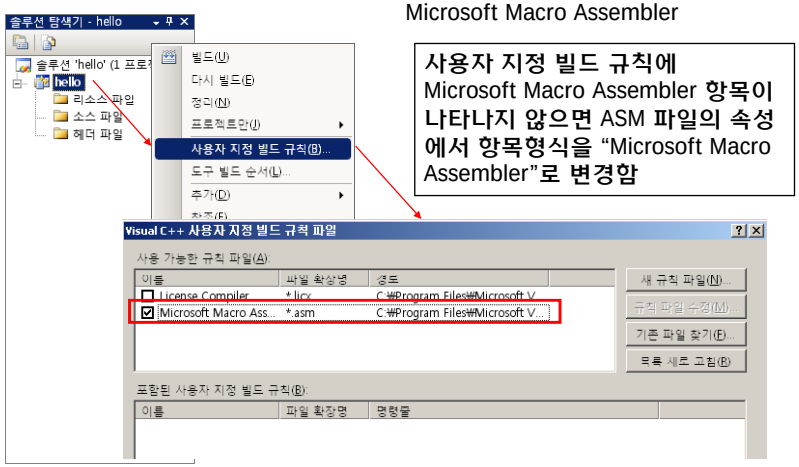
- visual studio project의 속성 변경
  - 구성 속성 > 링커 > 시스템 > 하위 시스템 메뉴
  - 콘솔(/SUBSYSTEM:CONSOLE) 선택

→ 콘솔 창에 " 계속하려면 아무 키나 누르세요" 메시지 출력됨



- visual studio project의 "사용자 지정 빌드 규칙" 추가

Microsoft Macro Assembler



사용자 지정 빌드 규칙에 Microsoft Macro Assembler 항목이 나타나지 않으면 ASM 파일의 속성에서 항목형식을 "Microsoft Macro Assembler"로 변경함

어셈블리 언어 프로그램의 다른 버전

- 어셈블리언어 프로그램: sum.asm

```
.386
.model flat             ; 언어 지정자를 사용하지 않음
.code
_sum proc               ; C언어 호출을 위해서 _를 붙임
        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:     leave
        ret
_sum endp
end
```


C++와 링크하기

- C++언어 프로그램 : cppmain.cpp
  - 어셈블리 언어 프로시저를 C언어 함수로 선언함

```
#include <iostream>
using namespace std;

extern "C" int sum(int n);   // C언어 이름 규약 사용

int main(void)
{
        int s;

        s = sum(10);
        cout << s << endl;
        return 0;
}
```


Inline Assembly Code

- Inline Assembly Code
  - 고급 언어 프로그램에 삽입되는 어셈블리 언어 소스 코드
- 특징
  - external name, memory model, naming convention이 포함되지 않으므로 assembly program이 간단함
  - CALL/RET이 필요 없으므로 코드가 효율적임
  - compiler-specific directives 사용
  - 고급 언어 프로그램의 이식성이 부족함

__asm 디렉티브

- 구문 형식

```
__asm  statement          → single assembly statement

__asm {
  statement-1
  statement-2             → a block of assembly statements
  ...
  statement-n
}
```

- comment

```
mov  esi,buf      ; initialize index register
mov  esi,buf     // initialize index register
mov  esi,buf     /* initialize index register */
```

MS-VC++의 Inline Assembly Code의 특징

- 사용 가능한 것
  - 어셈블리 언어 명령어, 레지스터 이름
  - 함수 매개변수
  - asm 블록 바깥에 선언된 변수, 코드 레이블
  - 어셈블리 또는 C 스타일의 상수 표기 (예) 0A26h, 0xA26
  - PTR 연산자, LENGTH, SIZE, TYPE 연산자, EVEN, ALIGN 디렉티브
- 사용할 수 없는 것
  - 데이터 정의 디렉티브 (예) DB, DWORD ...
  - STRUCT, MACRO 등
  - _TEXT와 같은 세그먼트 이름
    (cf) CS와 같은 세그먼트 레지스터 이름은 사용 가능
- Resister 사용
  - EAX, EBX, ECX, EDX는 보존하지 않고 사용 가능 (caller save)
  - ESI, EDI, EBP를 사용하려면 항상 save, restore해야 함 (callee save)

Inline Code를 포함한 C 프로그램

```
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;
}
```