



# CSE3030 어셈블리 프로그래밍 (Assembly Programming)

## Assembly Programming in VS2017

Dept. of CS&E  
Sogang University



# Program Installation

## ◆ Visual Studio 2017

- ◆ 만일 자신의 컴퓨터에 VS Community 2017이 설치되어 있지 않다면 이를 download한 후 설치한다 (VS2019도 사용 가능).
- ◆ <https://visualstudio.microsoft.com/ko/downloads/>에서 다운 받을 수 있다.
- ◆ 설치 후 VS2017을 실행하면 Microsoft 계정으로 사용허가를 받아야 한다 (계정이 없으면 먼저 가입하고 VS2017을 실행하면 편리하다)

## ◆ Link Libraries and example programs for VS2017

- ◆ <http://kipirvine.com/asm/examples/index.htm> 에서 7th Edition용을 다운받아 C:\Irvine에 설치한다 (다운받은 파일을 더블클릭 후 지시에 따른다).
- ◆ 현재 이 site는 막혀 있으므로 배포한 Irvine\_7th\_Edition.msi를 더블 클릭.

## ◆ 작업 폴더 만들기(1)

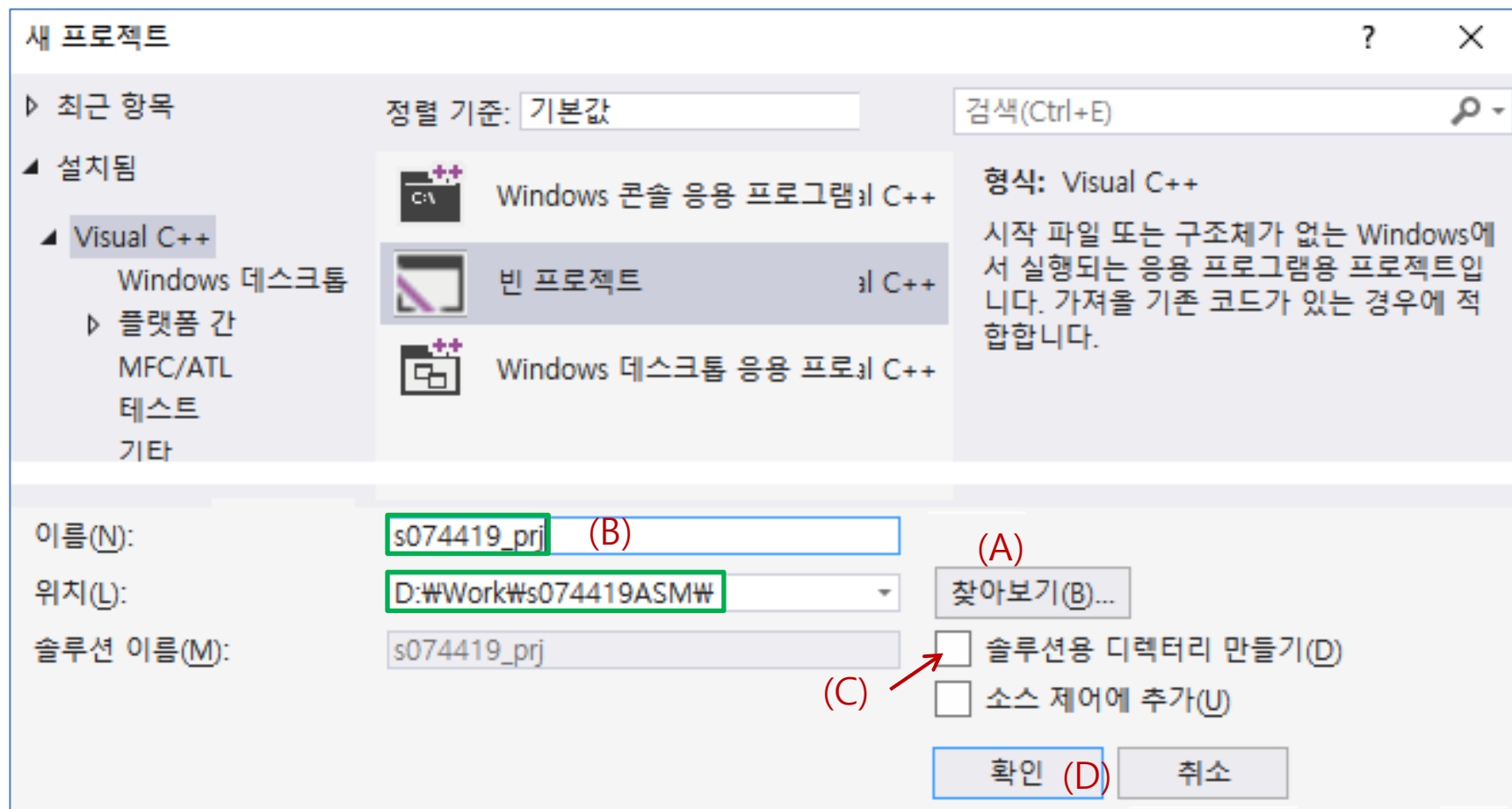
- ◆ 적당한 drive를 선택하여, snnnnnnnASM이라는 폴더를 만든다.
- ◆ snnnnnnnASM안에 이름이 snnnnnnn\_src (소스 파일 저장용), snnnnnnn\_exe (실행 파일 저장용)인 두 개의 폴더를 만든다.

(1) nnnnnnn은 자신의 학번 뒤 6자리.

# 어셈블리 프로그램용 프로젝트 만들기

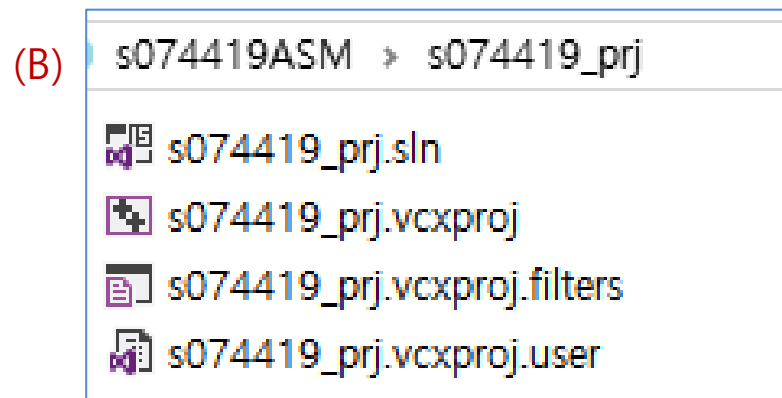
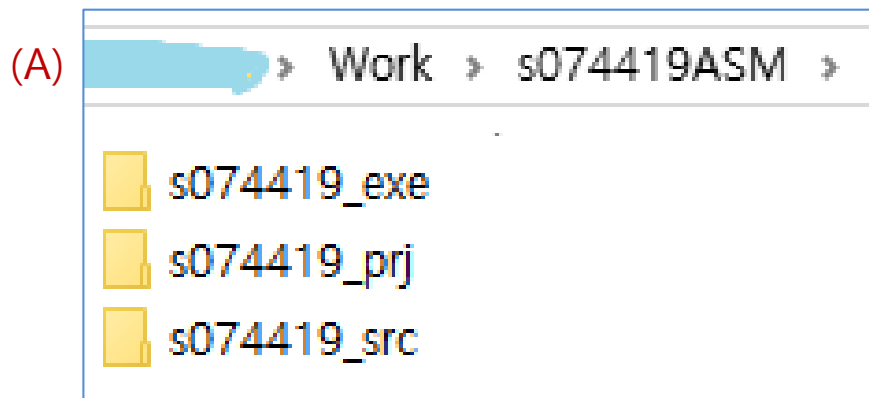
## ◆ 프로젝트 생성

- ◆ 파일 → 새로 만들기 → 프로젝트 → Visual C++ → 빈 프로젝트
- ◆ 폴더 선택(A) → 프로젝트 이름 입력(B) → 체크 해제(C) → 확인(D)



## ◆ 생성한 폴더들

- ◆ 폴더 `snnnnnnnASM`<sup>(A)</sup>
- ◆ 프로젝트 폴더 `snnnnnnn_prj`의 내용<sup>(B)</sup>



- ◆ 이 세 폴더는 실습 후/실습실에서 작업 후 반드시 자신의 USB(또는 Cloud, email 등)에 백업 받고 → 공용 장소 컴퓨터이면 세 폴더 삭제 → 휴지통 비우기를 수행하여야 한다<sup>(1)</sup>.
- ◆ 다시 세팅하려면 시간이 걸리기 때문에 `snnnnnnn_prj`는 삭제 하지 않으며, 모든 프로그램 작성을 이 폴더에서 수행한다.

(1) 자신의 USB 메모리에 이 세 폴더를 생성하여 두는 것도 나쁘지 않다(다만, 이 역시 백업을 철저히 하여야 한다).



## ◆ 테스트 용 어셈블리 프로그램 작성(1)

- ◆ 파일 → 새로 만들기 → 파일 → 일반 → 텍스트 파일 → 열기 → 파일 편집(아래 보인 프로그램을 입력한다).
- ◆ 파일 → 다른 이름으로 저장 → 파일 형식(\*.txt)을 모든 파일(\*.\*)로 변경 → 폴더 \s074419ASM\s074419\_src에 01\_AddSum.asm으로 저장.

```
TITLE Add and Subtract
INCLUDE Irvine32.inc

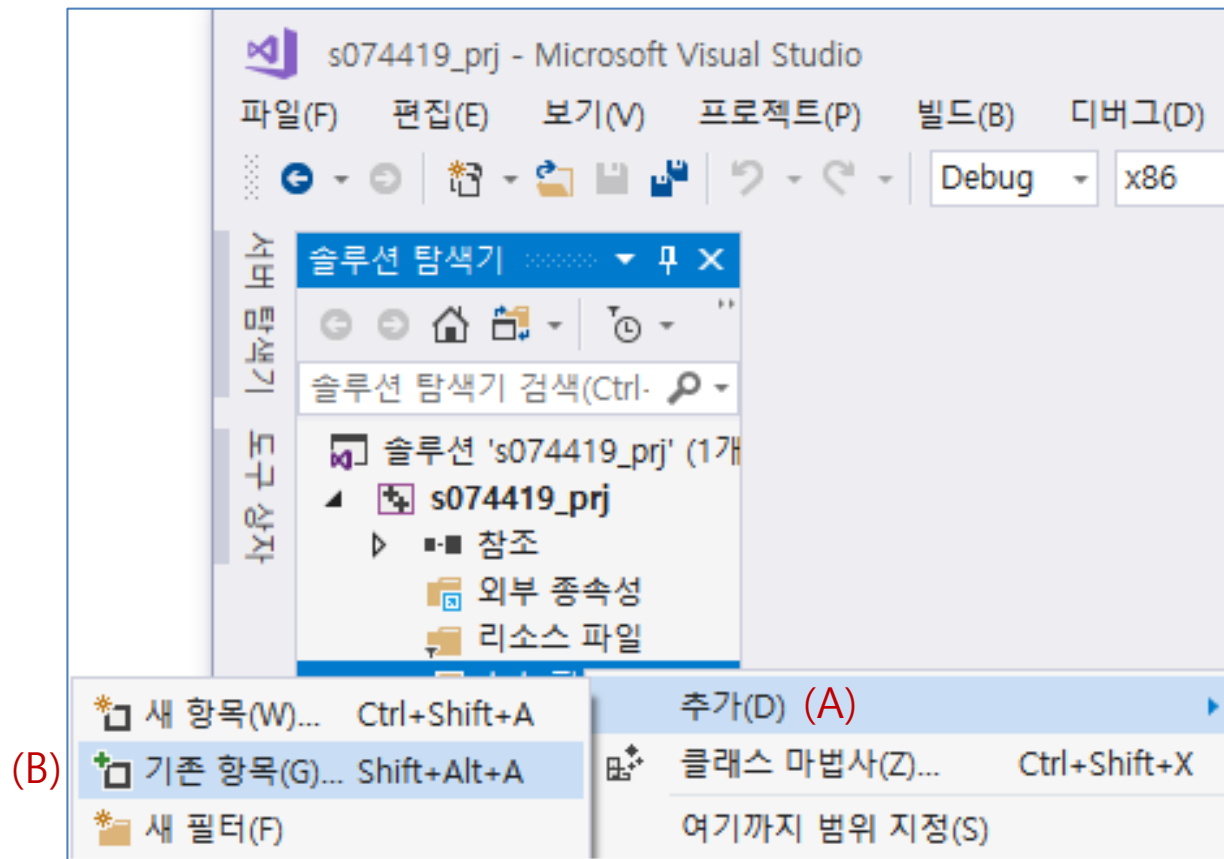
.code
main PROC
    mov  eax, 1000h    ; eax <- 10000h
    add  eax, 4000h    ; eax <- 50000h
    sub  eax, 2000h    ; eax <- 30000h
    call DumpRegs

    exit
main ENDP
END main
```

(1) 이미 기존 파일이 있는 경우 이 단계를 생략한다.

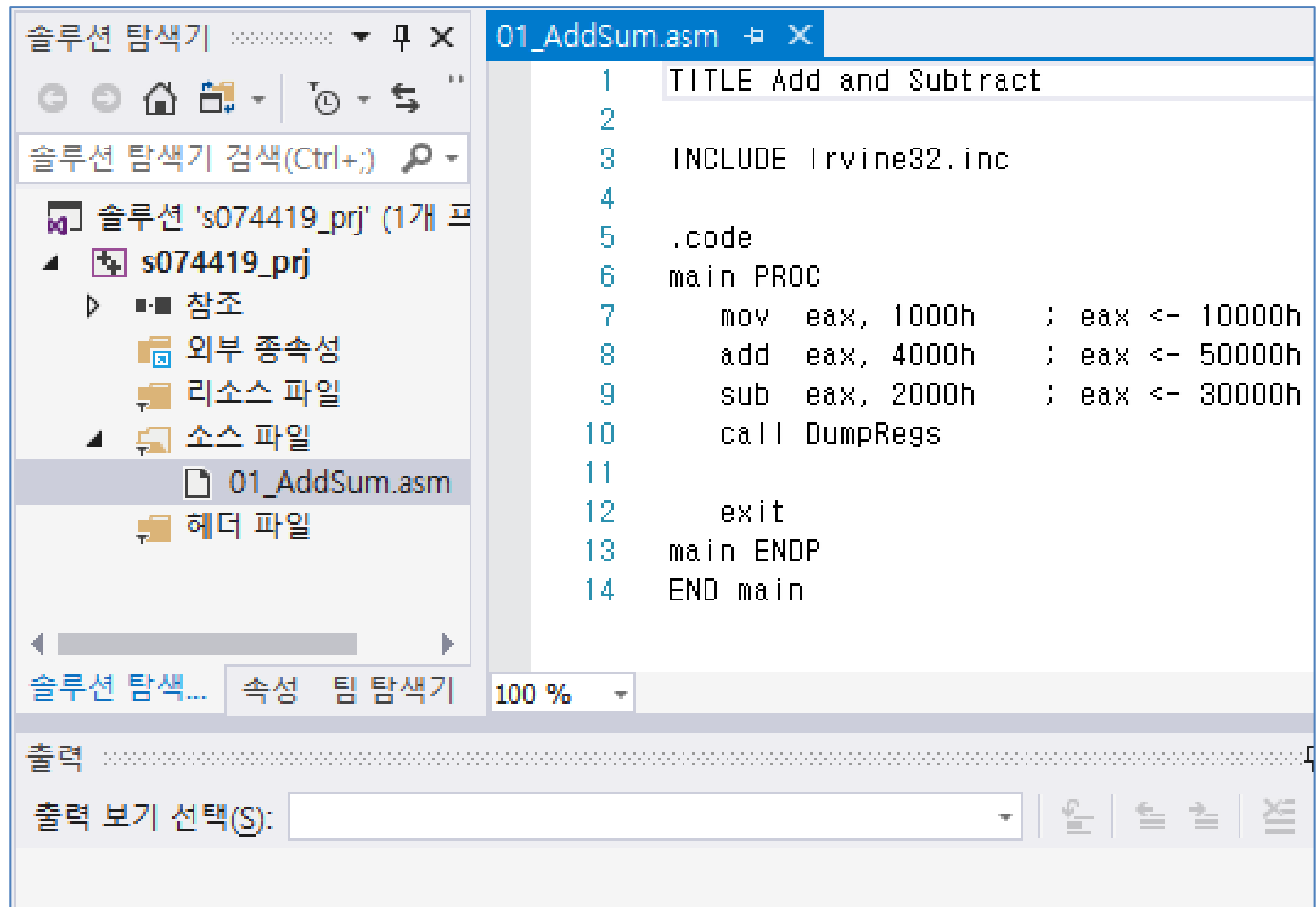
## ◆ 작성한 파일을 프로젝트에 추가

◆ 솔루션 탐색기 → 소스 파일 → 클릭 우버튼 → 추가(A) → 기존 항목(B) → 파일 찾아서 → 작성한 01\_AddSum.asm 파일 추가.



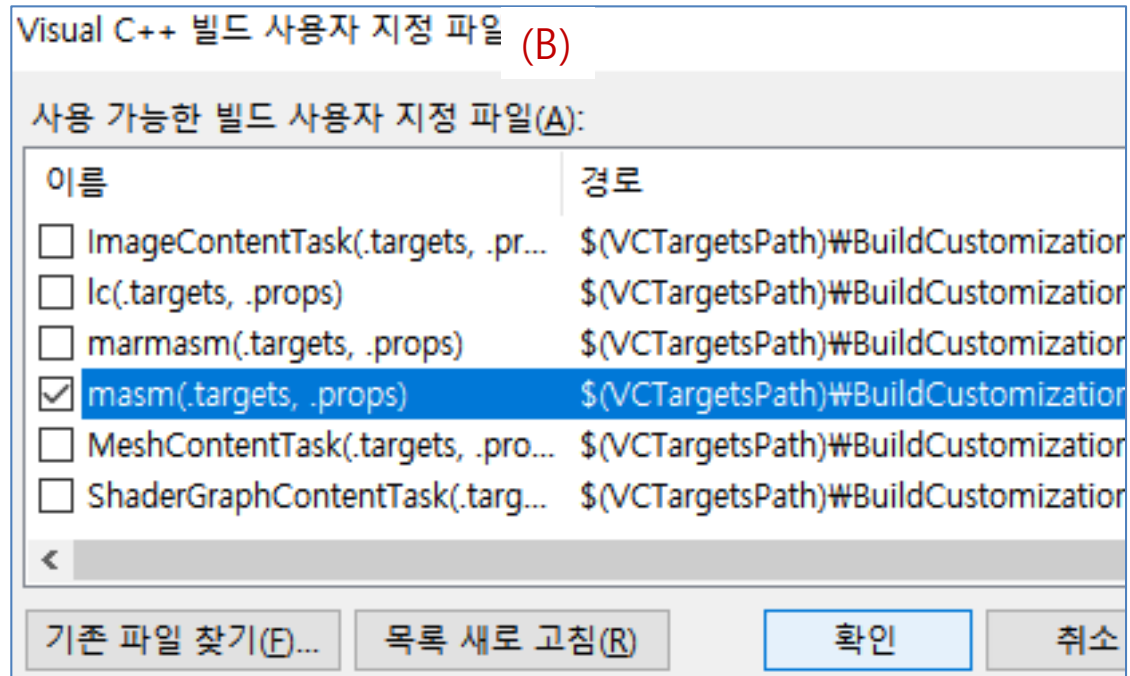
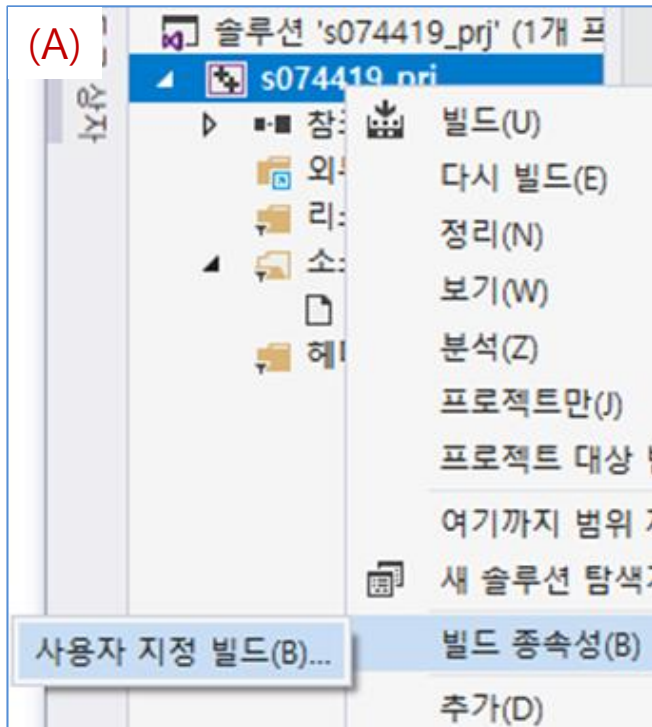
(1) 솔루션 탐색기 창이 안보이면 메뉴 바/보기 → 클릭 솔루션 탐색기 → 위치 조정 (조정 방법은 실습 시간에).

- ◆ 파일 등록 후 이를 더블 클릭하여 파일을 편집 창에 연다.
- ◆ 아래와 같은 모양이 될 것이다.



# 프로젝트 세팅

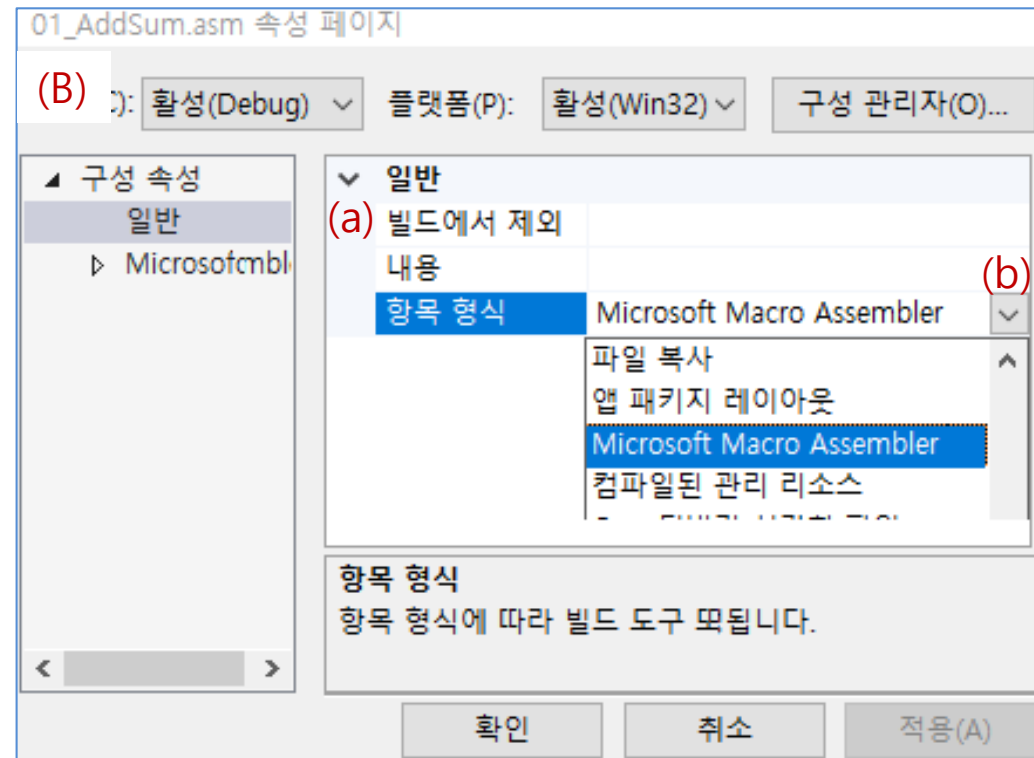
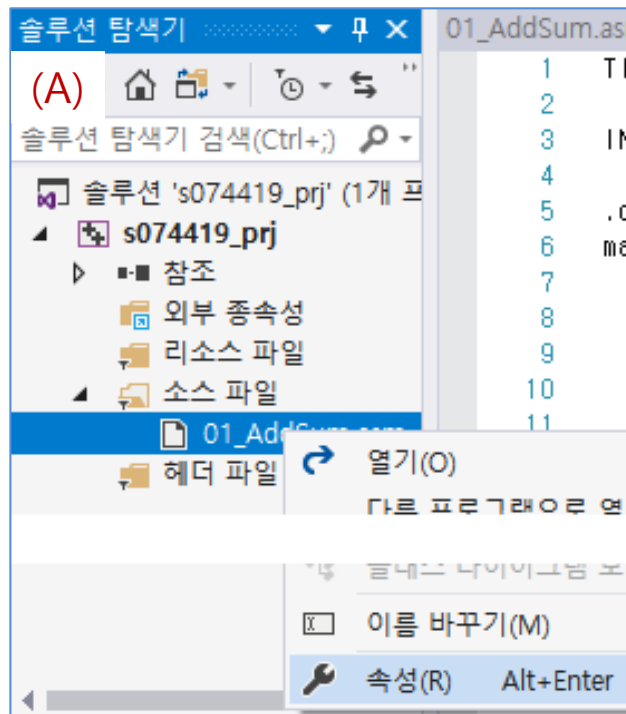
- ◆ 어셈블리 프로그래밍을 위하여 일련의 세팅이 필요하다.
- ◆ 타겟 설정
  - ◆ 솔루션 탐색기/프로젝트(\_prj) → 클릭 우 버튼 → 빌드 종속성 선택 → 클릭 사용자 지정 빌드(A).
  - ◆ VC++ 빌드 사용자 지정 파일 창 → 체크 `masm(tar...)` → 확인(B).





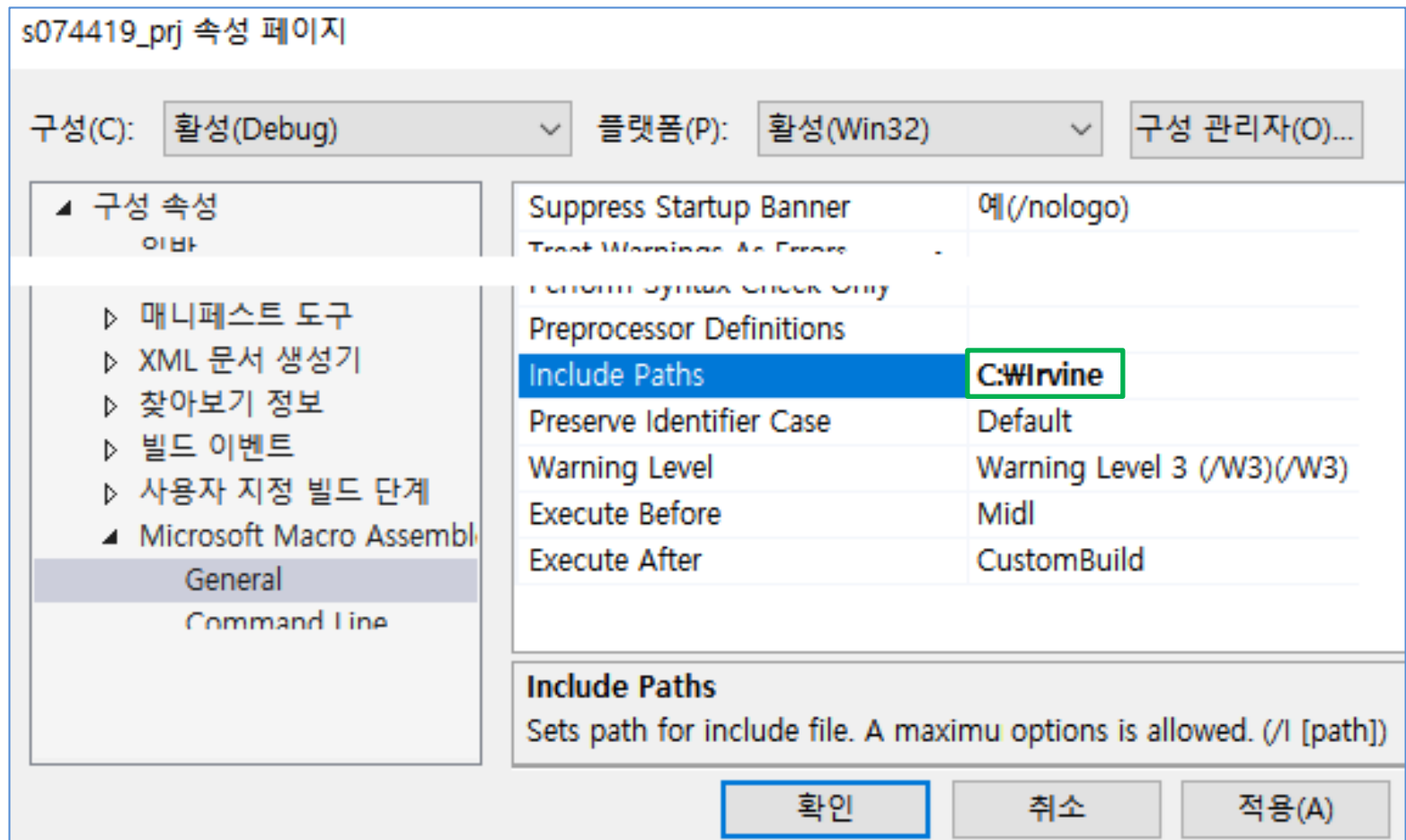
## ◆ Source 파일 속성 설정

- ◆ 추가한 .asm 파일 선택 → 클릭 우 버튼 → 속성 선택(A).
- ◆ .asm 속성 페이지 → 구성 속성/일반 → 일반 펼치기(a) → 클릭 항목 형식 → 우측 클릭하여 펼치기(b) → Microsoft Macro Assembler 선택(B).



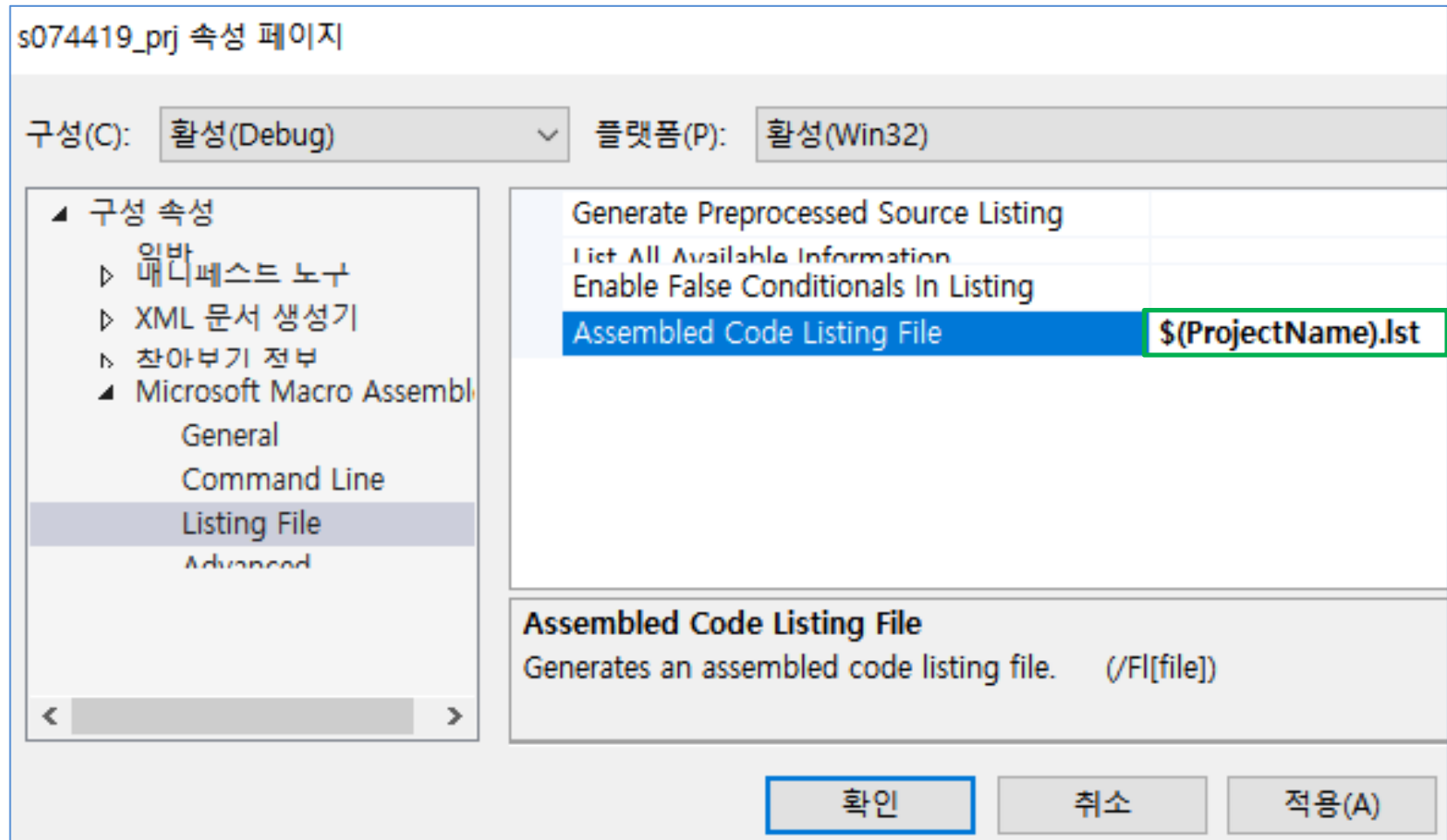
## ◆ 컴파일러, 링커 속성 설정

- ◆ 다시, 솔루션 탐색기/프로젝트(\_prj) → 클릭 우 버튼 → 속성 선택.
- ◆ 구성 속성 → Microsoft Macro Assembler → General → Include Paths → library를 설치한 경로 입력(주로 C:\Irvine) → 클릭 적용.



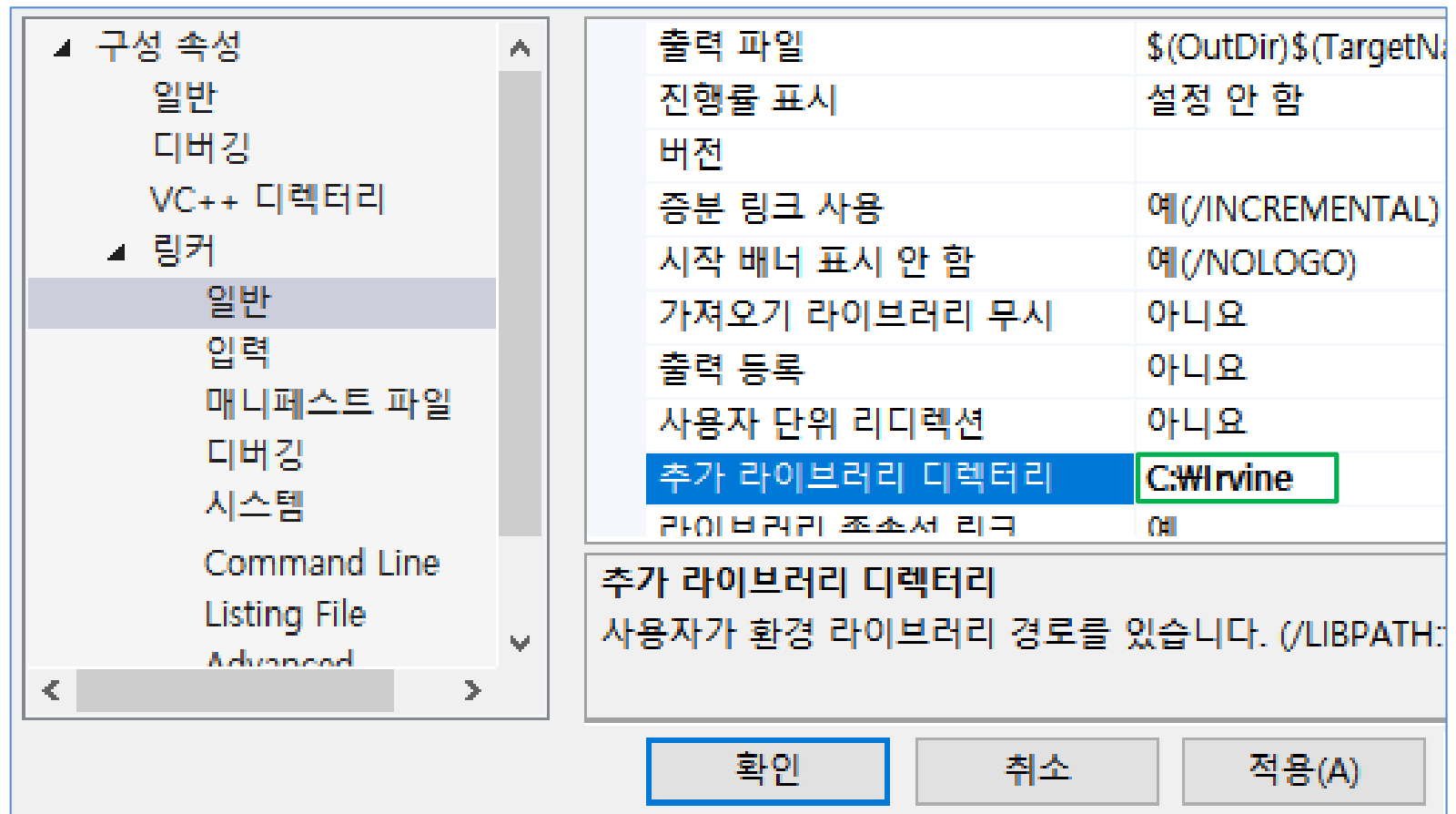


- ◆ 계속해서, 구성 속성 → Microsoft Macro Assembler → Listing File → Assembled Code Listing File 항목에  $\$(ProjectName).lst$  입력 → 적용.



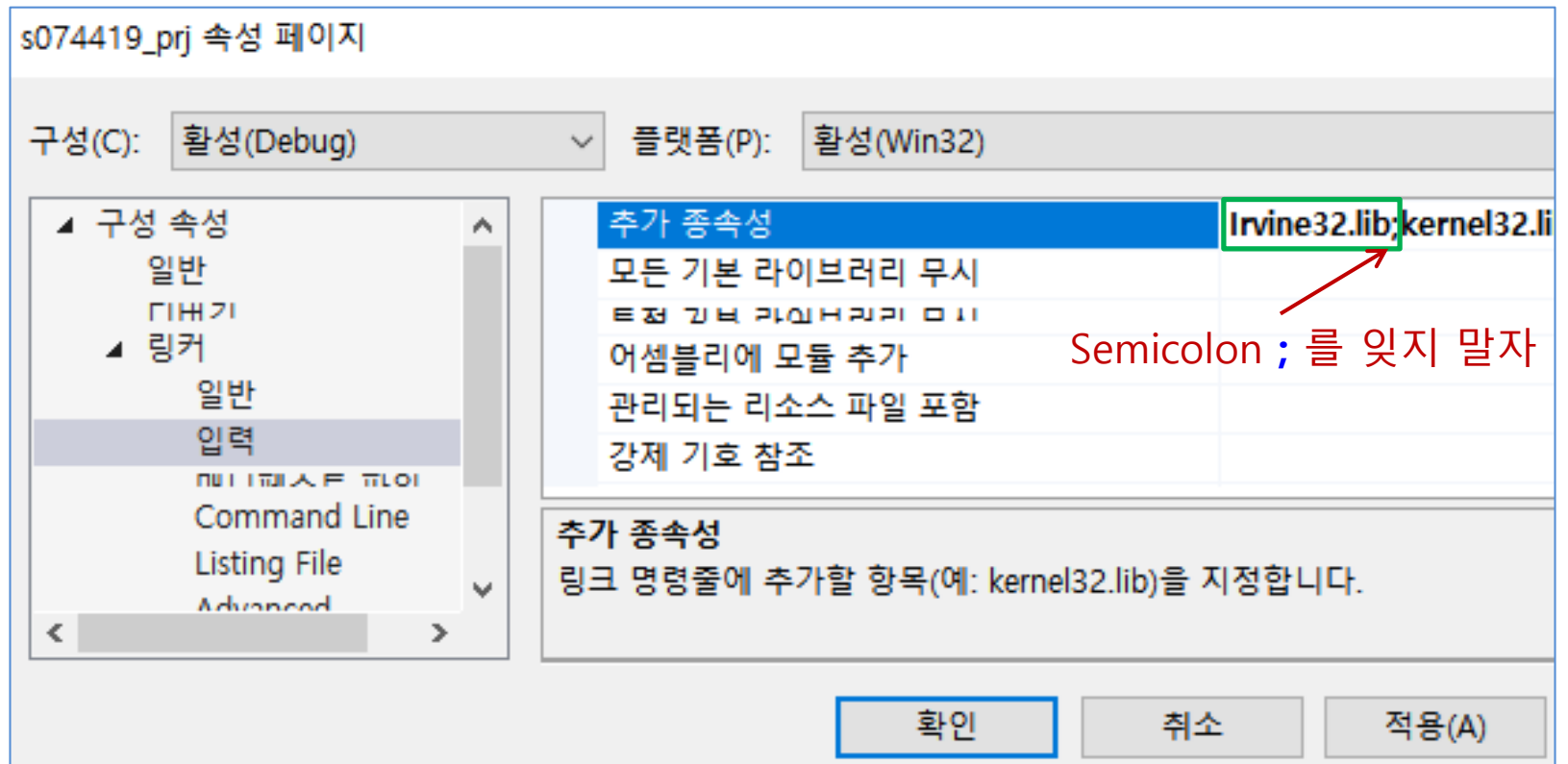


- ◆ 구성 속성 → 링커 → 일반 → 추가 라이브러리 디렉토리에 library 경로 입력 (주로 C:\Irvine) → 적용.





- ◆ 구성 속성 → 링커 → 입력 → 추가 종속성 맨 앞에 **Irvine32.lib**; 추가 → 적용.



- ◆ 구성 속성 → 링커 → 디버깅 → 디버그 정보 생성 → 우측 모서리(A) 클릭 → 디버그 정보 생성(/DEBUG) 선택 설정 → 적용.

s074419\_prj 속성 페이지

구성(C): **활성(Debug)** 플랫폼(P): **활성(Win32)**

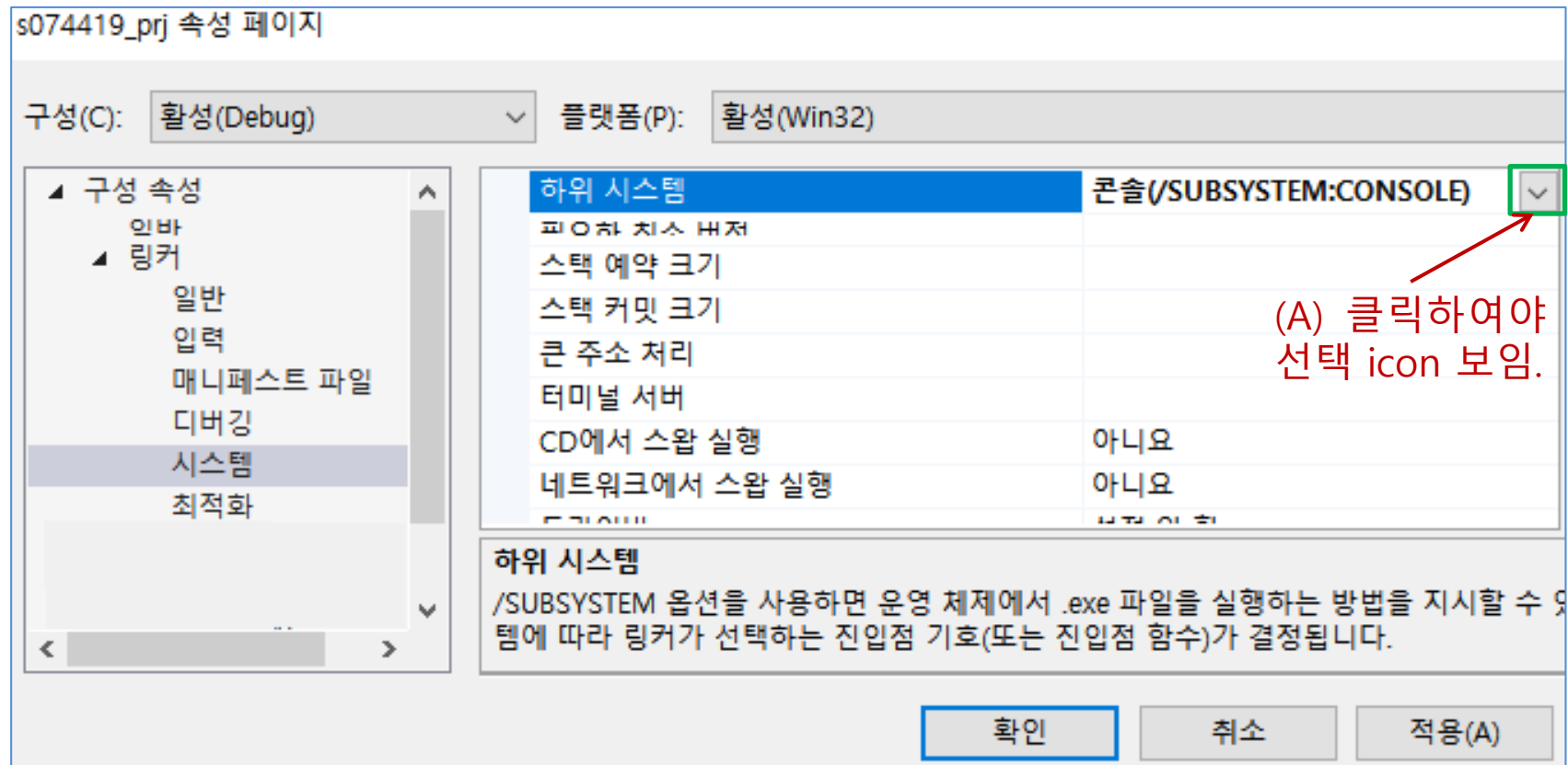
구성 속성	디버그 정보 생성	디버그 정보 생성(/DEBUG)
일반	프로그램 데이터베이스 파일 생성	\$(OutDir)\$(TargetName).pdb
디버깅	전용 기호 제거	
VC++ 디렉터리	맵 파일 생성	아니요
링커	맵 파일 이름	
일반	맵 내보내기	아니요
입력	디버깅 가능한 어셈블리	
매니페스트 파일		
디버깅		
시스템		
Command Line		
Listing File		
Advanced		

**디버그 정보 생성**  
이 옵션을 사용하면 .exe 파일 또는 DLL 디버깅 정보를 만들 수 있습니다.

(A) 클릭하여야 선택 icon 보임.

확인 취소 적용(A)

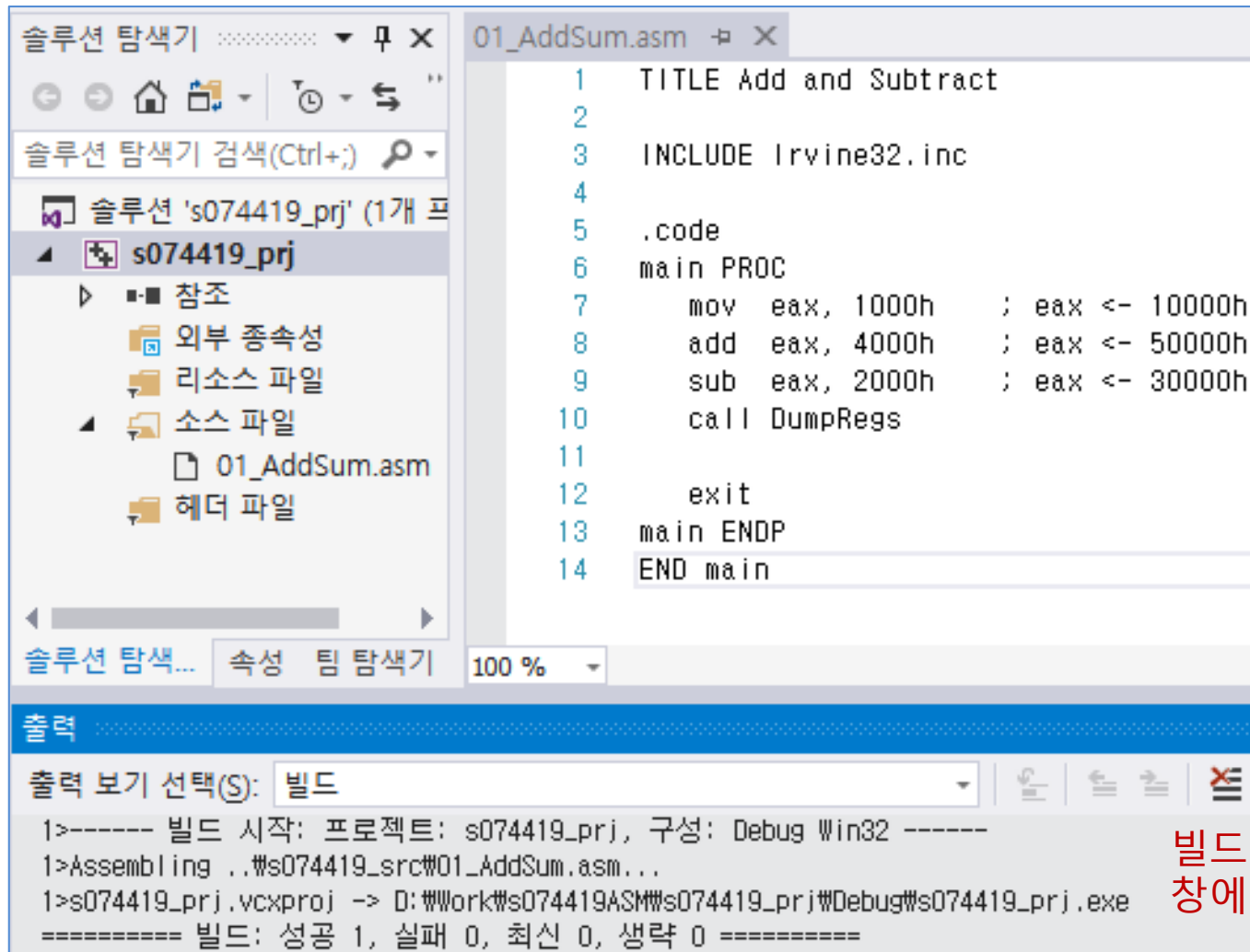
- ◆ 구성 속성 → 링커 → 시스템 → 하위 시스템 → 우측 모서리(A) 클릭 → 콘솔(/SUBSYSTEM:CONSOLE) 선택 설정 → 적용 → 확인.



# Build and Run the Program

## ◆ Build

◆ 메뉴 바 → Build Solution(Ctrl+Shift+B)



빌드 결과가 출력  
창에 나타난다





## ◆ Run

- ◆ 메뉴 바 → 디버그 → 디버그 하지 않고 시작(Ctrl + F5)
- ◆ 아래 보인 디버그 콘솔 창이 생기고, 출력 결과가 보인다.
- ◆ 검토를 마치면 아무 키나 누른다 → 창이 닫힌다.
- ◆ Visual Studio에서는 계속 작업할 수 있다.

Microsoft Visual Studio 디버그 콘솔

```
EAX=00003000  EBX=006E7000  ECX=0121100A  EDX=0121100A  
ESI=0121100A  EDI=0121100A  EBP=0095FAEC  ESP=0095FADC  
EIP=01213674  EFL=00000206  CF=0  SF=0  ZF=0  OF=0  AF=0  PF=1
```

D:\Work\ss074419ASM\ss074419\_prj\Debug\ss074419\_prj.exe(9324 프로세스)  
로되었습니다.  
이 창을 닫으려면 아무 키나 누르세요.



## ◆ 명령 프롬프트에서 실행

- ◆ 폴더 `\Work\snnnnnnnASM\snnnnnnn_prj\Debug`에 파일 `snnnnnnn_prj.exe`가 생성되는데 이는 명령 프롬프트에서 실행할 수 있다<sup>(1)</sup>.
- ◆ 이 파일을 직접 실행해도 되지만, 폴더 `snnnnnnn_exe`에 복사 후 실행하자.
- ◆ `snnnnnnn_prj.exe`를 `snnnnnnn_exe` 폴더로 복사(`01_AddSub.exe`로 이름변경).
- ◆ 명령 프롬프트에서 드라이브 선택(`D: ↵`)
- ◆ `cd D:\Work\snnnnnnnASM\snnnnnnn_exe ↵`
- ◆ `01_AddSub.exe ↵`

```
명령 프롬프트
Microsoft Windows [Version 10.0.17134.472]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\csrim>D:

D:\>cd D:\Work\074419ASM\074419_exe

D:\Work\074419ASM\074419_exe>01_AddSub

EAX=00003000  EBX=00D30000  ECX=00FD100A  EDX=00FD100A
ESI=00FD100A  EDI=00FD100A  EBP=00EFF810  ESP=00EFF800
EIP=00FD367A  EFL=00000206  CF=0   SF=0   ZF=0   OF=0   AF=0   PF=1

D:\Work\074419ASM\074419_exe>
```

(1) nnnnnn은 자신의 학번 뒤 6자리.



# Debugging

## ◆ 디버깅 모드에서의 주요 단축키

F5 – Go(다음 breakpoint에서 정지)  
F9 – Insert/Remove Breakpoint  
F10 – Step Over,  
Ctrl+F5 – Execute Program

Shift+F5 – 디버그모드 종료

F11 – Step In

## ◆ 디버깅 모드 시작

◆ Breakpoint를 원하는 위치에 잡는다 → F5

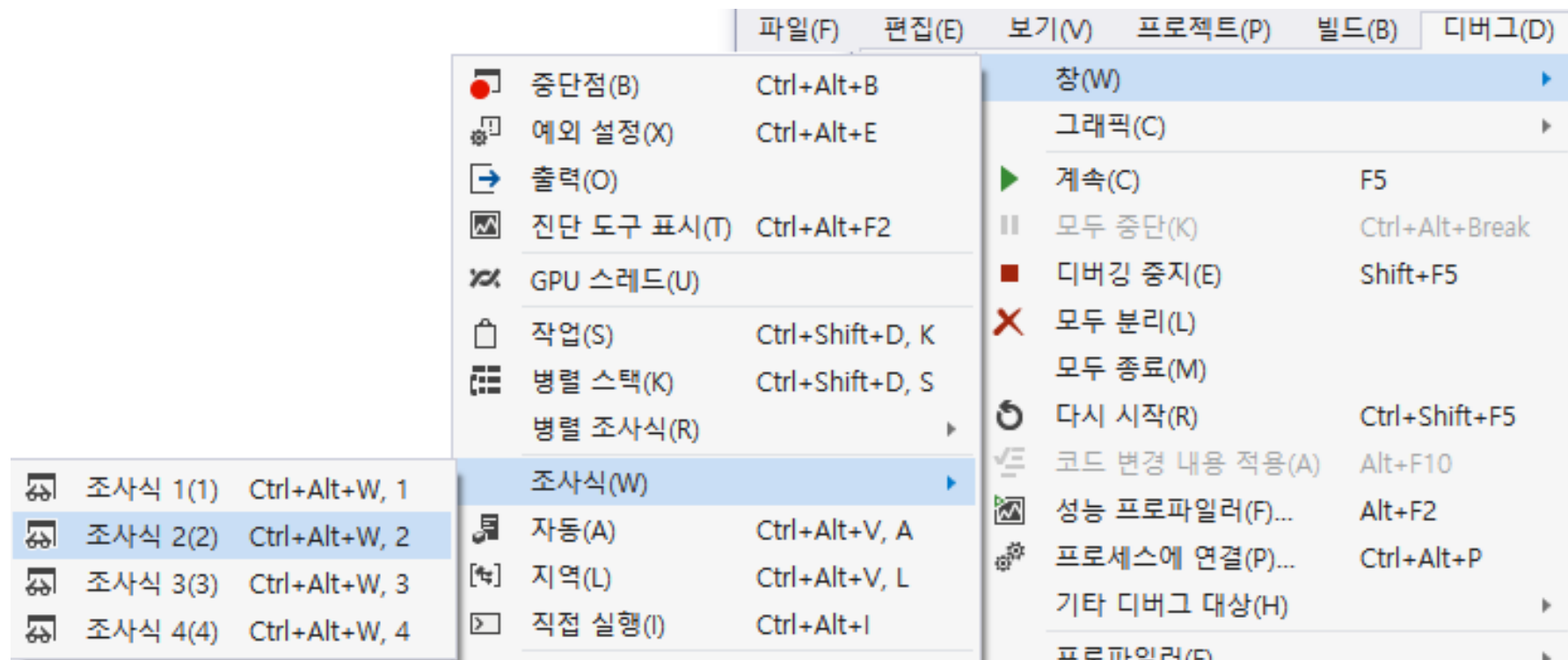
01\_AddSum.asm

```
1  TITLE Add and Subtract
2
3  INCLUDE Irvine32.inc
4
5  .code
6  main PROC
7      mov  eax, 1000h
8      add  eax, 4000h
9      sub  eax, 2000h
10     call DumpRegs
11
12     exit
13 main ENDP
14 END main
```

빨강 동그라미:  
break 잡은 위치

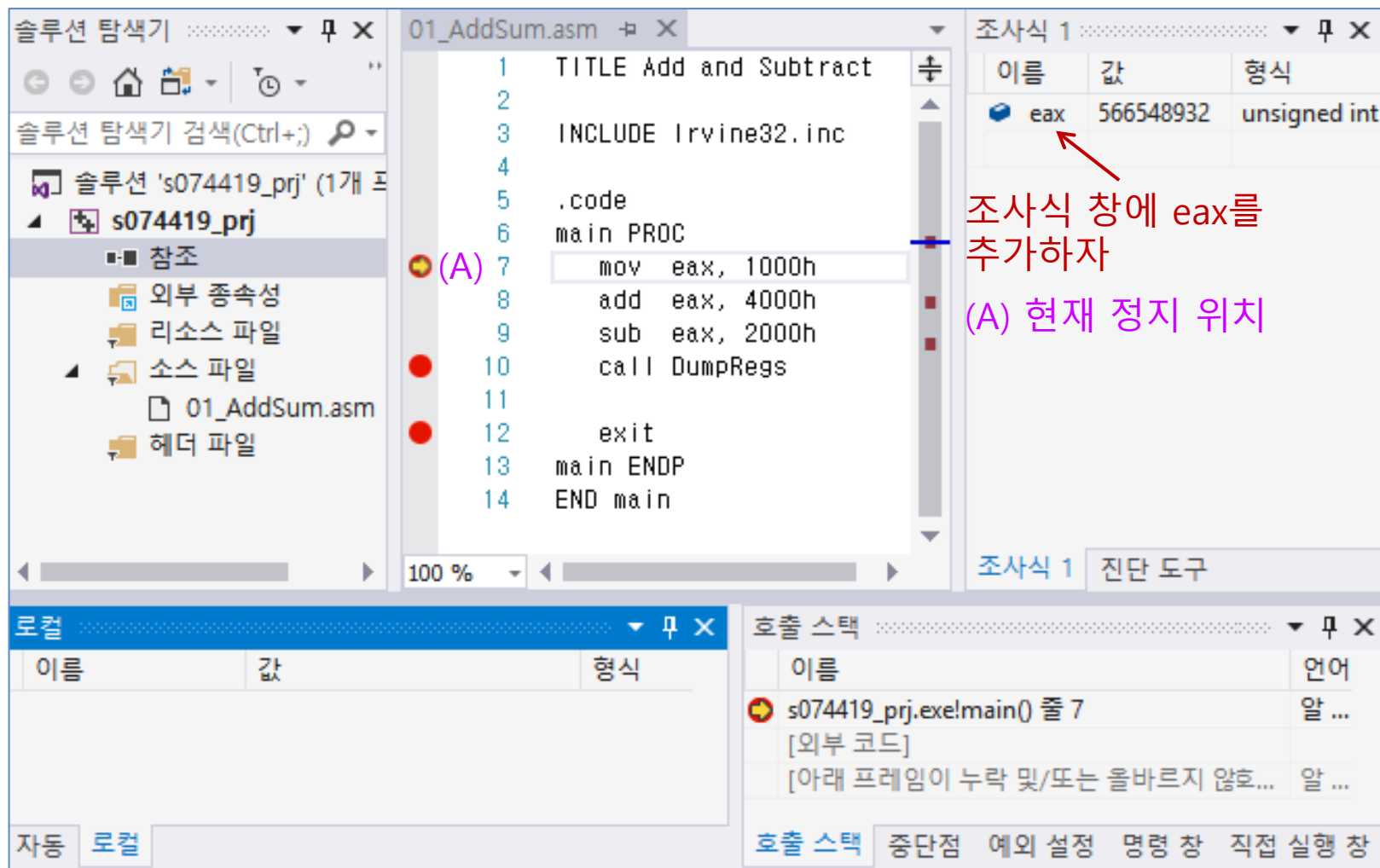
## ◆ 창 추가

- ◆ 조사식, 메모리 등의 창을 디버깅 모드에서 자주 사용한다.
- ◆ 조사식 창 추가: 메뉴 바/디버그 → 창 → 조사식 → 조사식 선택 → 클릭.



## ◆ IDE 레이아웃

◆ 아래 그림과 같이 창을 배치해 보자<sup>(1)</sup>.



(1) 창 배치하는 방법에 익숙하지 않으면 실습시간 강의를 참조하자(글/그림으로 설명하기 쉽지 않다).



## ◆ 디버깅 모드에서의 실행

◆ F10을 누르면 아래와 같이 한 스텝 실행한다.

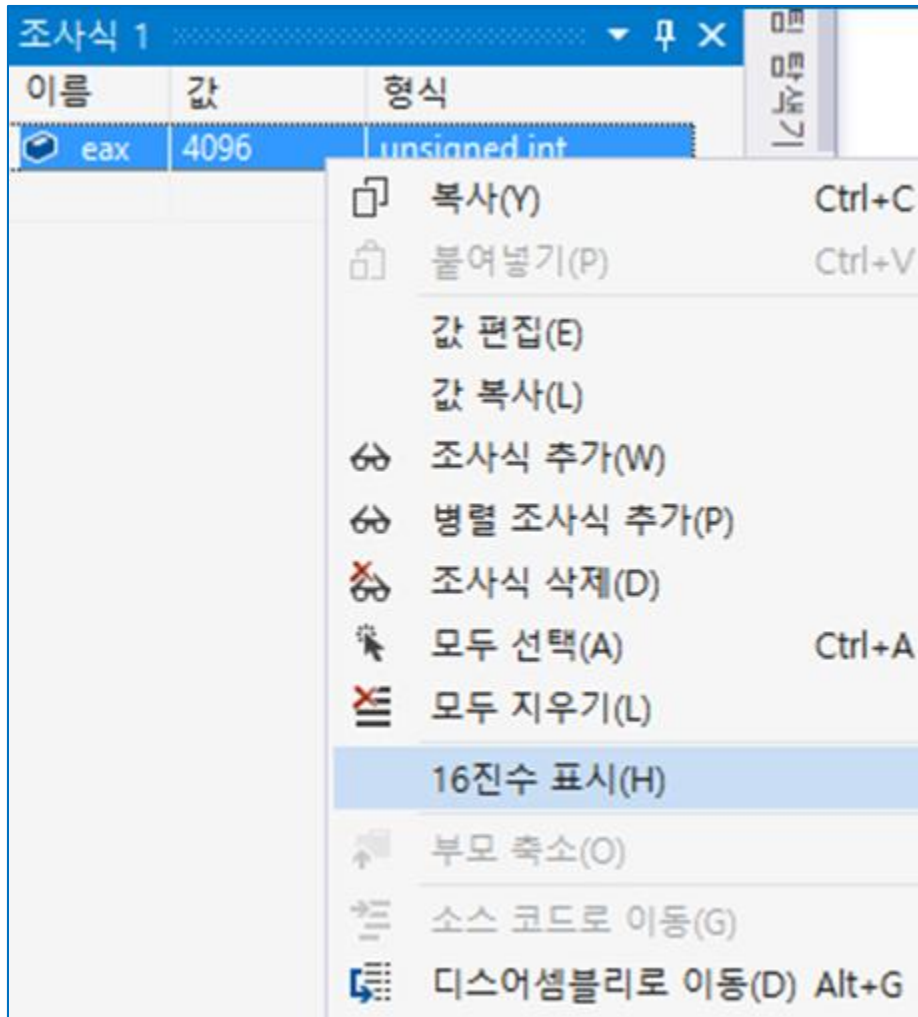
The screenshot shows a debugger window with two panes. The left pane displays assembly code for a file named '01\_AddSum.asm'. The code includes a title, an include directive, and a main procedure with several instructions. The right pane shows a register window titled '조사식 1' (Watch Expression 1) with a table of register values.

이름	값	형식
eax	4096	unsigned int

The assembly code in the left pane is as follows:

```
1  TITLE Add and Subtract
2
3  INCLUDE Irvine32.inc
4
5  .code
6  main PROC
7      mov  eax, 1000h    ; eax <- 10000h
8      add  eax, 4000h    ; eax <- 50000h
9      sub  eax, 2000h    ; eax <- 30000h
10     call DumpRegs
11
12     exit
13 main ENDP
14 END main
```

- ◆ 다음과 같이 클릭하여 조사식 창 결과를 16 진수로 바꿀 수 있다.
  - ◆ 조사식 레지스터/변수 우클릭 → 클릭 16진수 표시



조사식 1		
이름	값	형식
eax	0x00001000	unsigned int

- F10을 클릭하여 계속 진행하자.
- 프로그램 실행이 완료되면 자동으로 편집 모드로 바뀐다.
- 만일 중간에 디버그 모드를 종료하고자 한다면 Shift+F5.



## ◆ 변수(variable) 추가

- ◆ 현재 프로그램을 다음과 같이 수정하자

```
TITLE Add and Subtract

INCLUDE Irvine32.inc

.data
var1  DWORD 1000h ; 32 bit variable var1
var2  DWORD 4000h
sum   DWORD 0

.code
main PROC
    mov  eax, var1      ; eax <- 1000h
    add  eax, var2      ; eax <- 5000h
    sub  eax, 2000h     ; eax <- 3000h
    mov  sum, eax       ; sum <- eax
    call DumpRegs

    exit
main ENDP
END main
```





- ◆ Breakpoint는 다음과 같이 설정한다.

```
01_AddSum.asm*  X
1  TITLE Add and Subtract
2
3  INCLUDE Irvine32.inc
4  .data
5  var1    DWORD 1000h
6  var2    DWORD 4000h
7  sum     DWORD 0
8
9  .code
10 main PROC
11     mov  eax, var1      ; eax <- 1000h
12     add  eax, var2      ; eax <- 5000h
13     sub  eax, 2000h     ; eax <- 3000h
14     mov  sum, eax       ; sum <- eax
15     call DumpRegs
16
17     exit
18 main ENDP
19 END main
```



## ◆ Debugging

- ◆ 편집 모드로 가서 F5를 누르면 자동으로 빌드되고 디버깅 모드로 진입하여 첫번째 breakpoint에서 멈춘다.

```

9      .code
10     main PROC
11         mov     eax, var1
12         add     eax, var2
13         sub     eax, 2000h
14         mov     sum, eax
15         call    DumpRegs
  
```

- ◆ 이때 아래 그림과 같이 변수 그리고 변수들의 포인터를 입력하자.
- ◆ 그러면 변수 값과 이들의 주소를 볼 수 있다.

조사식 1			
이름	값	형식	
eax	0x92eca423	unsigned int	
var1	0x00001000	unsigned long	
var2	0x00004000	unsigned long	
sum	0x00000000	unsigned long	
&var1	0x01346000 {s074419_prj.exe!unsigned long var1} {0x00001000}	unsigned long *	
&var2	0x01346004 {s074419_prj.exe!unsigned long var2} {0x00004000}	unsigned long *	
&sum	0x01346008 {s074419_prj.exe!unsigned long sum} {0x00000000}	unsigned long *	








- ◆ F5을 누르면 실행 계속 후 Line 15에서 멈추고 직전에 변화된 값을 볼 수 있다.

```

10  main PROC
11      mov  eax, var1    ; eax <- 10000h
12      add  eax, var2    ; eax <- 50000h
13      sub  eax, 2000h   ; eax <- 30000h
14      mov  sum, eax     ; sum <- eax
15      call DumpRegs    경과 시간 1ms 이하

```

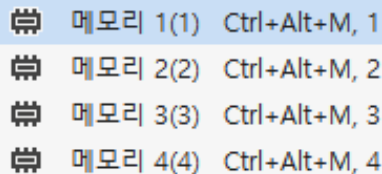
#### 조사식 1

이름	값	형식
 eax	0x00003000	unsigned int
 var1	0x00001000	unsigned long
 var2	0x00004000	unsigned long
 sum	0x00003000	unsigned long
▶  &var1	0x01346000 {s074419_prj.exe!unsigned long var1} {0x00001000}	unsigned long *
▶  &var2	0x01346004 {s074419_prj.exe!unsigned long var2} {0x00004000}	unsigned long *
▶  &sum	0x01346008 {s074419_prj.exe!unsigned long sum} {0x00003000}	unsigned long *

포인터 값은 변동 없음



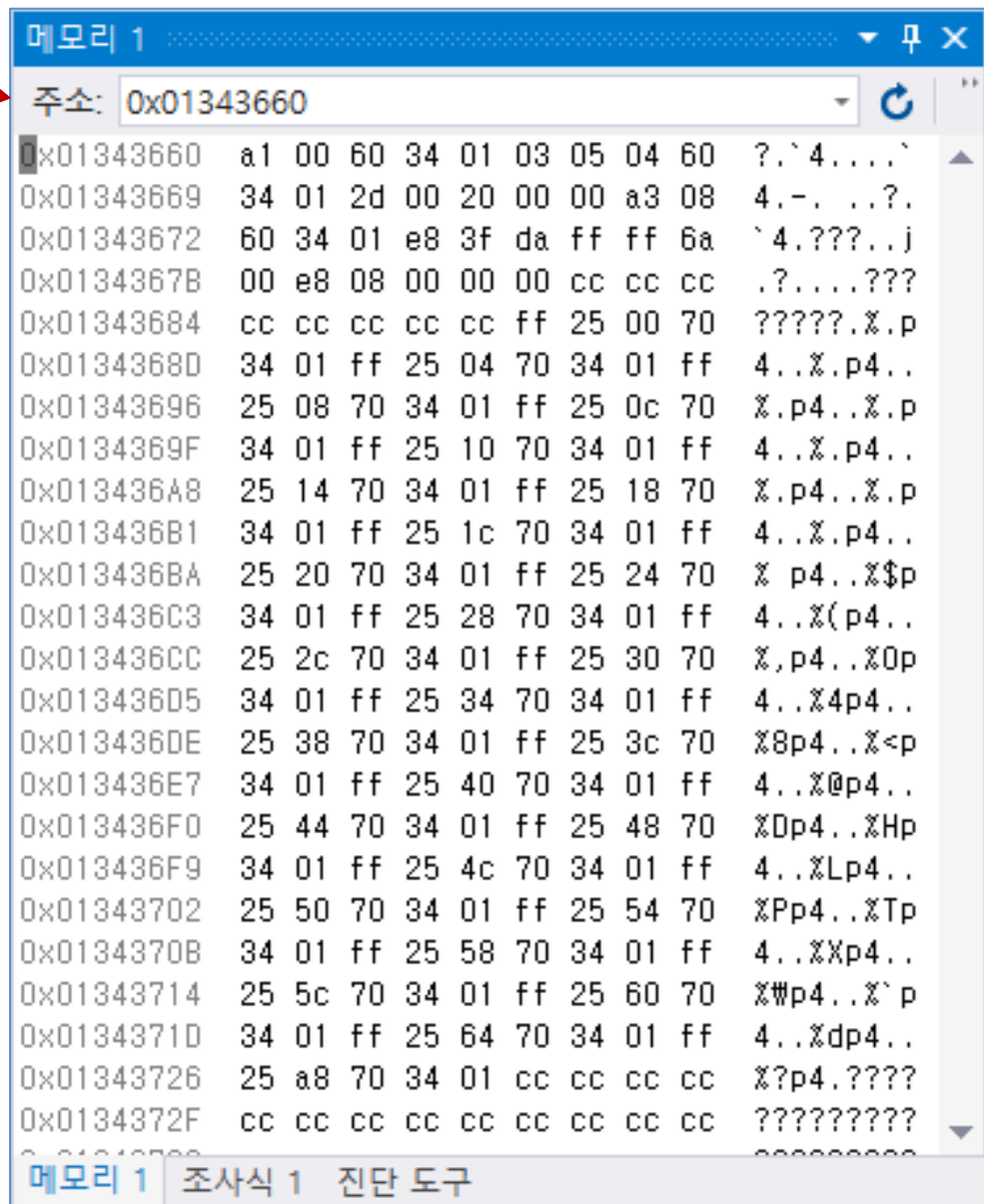
◆ 메뉴 바 → 디버그 → 창 → 메모리 → 메모리 창 선택 → 메모리 창 위치 조정



## ◆ 메모리 창 모습

메모리 내용 보기 위한  
시작 주소를 적는다

초기 위치는 진입점, 즉,  
실행 코드 시작 위치를  
가리킨다.



메모리 1

주소: 0x01343660

0x01343660	a1	00	60	34	01	03	05	04	60	?.`4....`
0x01343669	34	01	2d	00	20	00	00	a3	08	4.-. .?.
0x01343672	60	34	01	e8	3f	da	ff	ff	6a	`4.???..j
0x0134367B	00	e8	08	00	00	00	cc	cc	cc	.?....???
0x01343684	cc	cc	cc	cc	cc	ff	25	00	70	?????.%.p
0x0134368D	34	01	ff	25	04	70	34	01	ff	4..%.p4..
0x01343696	25	08	70	34	01	ff	25	0c	70	%.p4..%.p
0x0134369F	34	01	ff	25	10	70	34	01	ff	4..%.p4..
0x013436A8	25	14	70	34	01	ff	25	18	70	%.p4..%.p
0x013436B1	34	01	ff	25	1c	70	34	01	ff	4..%.p4..
0x013436BA	25	20	70	34	01	ff	25	24	70	% p4..%\$p
0x013436C3	34	01	ff	25	28	70	34	01	ff	4..%(p4..
0x013436CC	25	2c	70	34	01	ff	25	30	70	%,p4..%0p
0x013436D5	34	01	ff	25	34	70	34	01	ff	4..%4p4..
0x013436DE	25	38	70	34	01	ff	25	3c	70	%8p4..%<p
0x013436E7	34	01	ff	25	40	70	34	01	ff	4..%@p4..
0x013436F0	25	44	70	34	01	ff	25	48	70	%Dp4..%Hp
0x013436F9	34	01	ff	25	4c	70	34	01	ff	4..%Lp4..
0x01343702	25	50	70	34	01	ff	25	54	70	%Pp4..%Tp
0x0134370B	34	01	ff	25	58	70	34	01	ff	4..%Xp4..
0x01343714	25	5c	70	34	01	ff	25	60	70	%#p4..%`p
0x0134371D	34	01	ff	25	64	70	34	01	ff	4..%dp4..
0x01343726	25	a8	70	34	01	cc	cc	cc	cc	%?p4.????
0x0134372F	cc	cc	cc	cc	cc	cc	cc	cc	cc	??????????

메모리 1 조사식 1 진단 도구

- ◆ 시작 주소로 0x01346000을 입력하자. 이 주소는 var1의 포인터 값이다.
- ◆ 다음, F5를 누르면 Line 15에서 멈추는데, Line 14에서 variable sum에 eax 값을 저장 하였었기에, 저장된 값이 빨강 색으로 메모리 창에 보인다.

```

10  main PROC
11      mov  eax, var1    ; eax <- 10000h
12      add  eax, var2    ; eax <- 50000h
13      sub  eax, 2000h   ; eax <- 30000h
14      mov  sum, eax     ; sum <- eax
15      call DumpRegs    경과시간 1ms 이하
16
17      exit

```

메모리 1

주소: 0x01346000

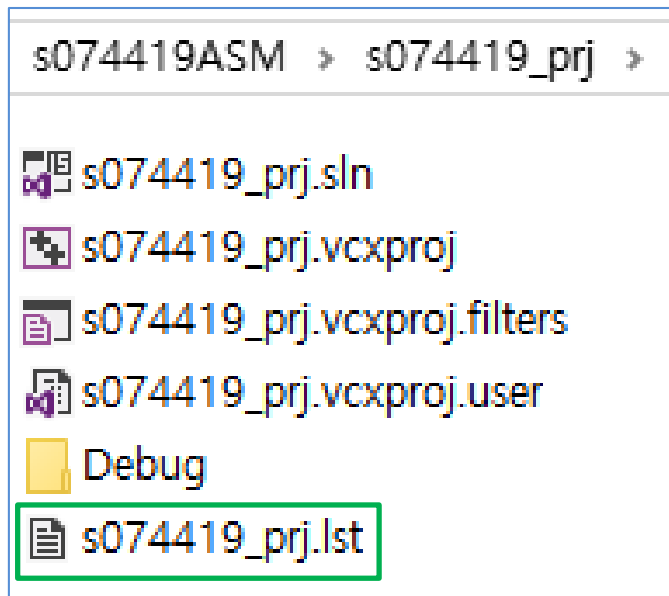
0x01346000	00	10	00	00	00	40	00	00	00	30	00	.....@...0.
0x0134600B	00	00	00	00	00	00	30	31	32	33	34	.....01234
0x01346016	35	36	37	38	39	41	42	43	44	45	46	56789ABCDEF
0x01346021	20	20	20	20	20	20	20	20	20	20	20	}
0x0134602C	20	20	20	20	20	20	20	20	20	20	20	
0x01346037	20	20	20	20	20	20	20	20	20	20	20	
0x01346042	20	20	20	20	20	20	20	20	20	20	20	
0x0134604D	20	20	20	20	20	20	20	20	20	20	20	
0x01346058	20	20	20	20	20	20	20	20	20	20	20	
0x01346063	20	20	20	20	20	20	20	20	20	20	20	
0x0134606E	20	20	20	20	20	20	20	20	20	20	20	
0x01346079	20	20	20	20	20	20	20	20	20	20	20	
0x01346084	20	20	20	20	20	20	20	20	20	20	20	
0x0134608F	20	20	20	20	20	20	20	20	20	20	20	
0x0134609A	20	20	20	20	20	20	20	20	20	20	20	

이 부분은 char로 보인다. Invisible  
인 경우 . (dot)으로 표시된다.

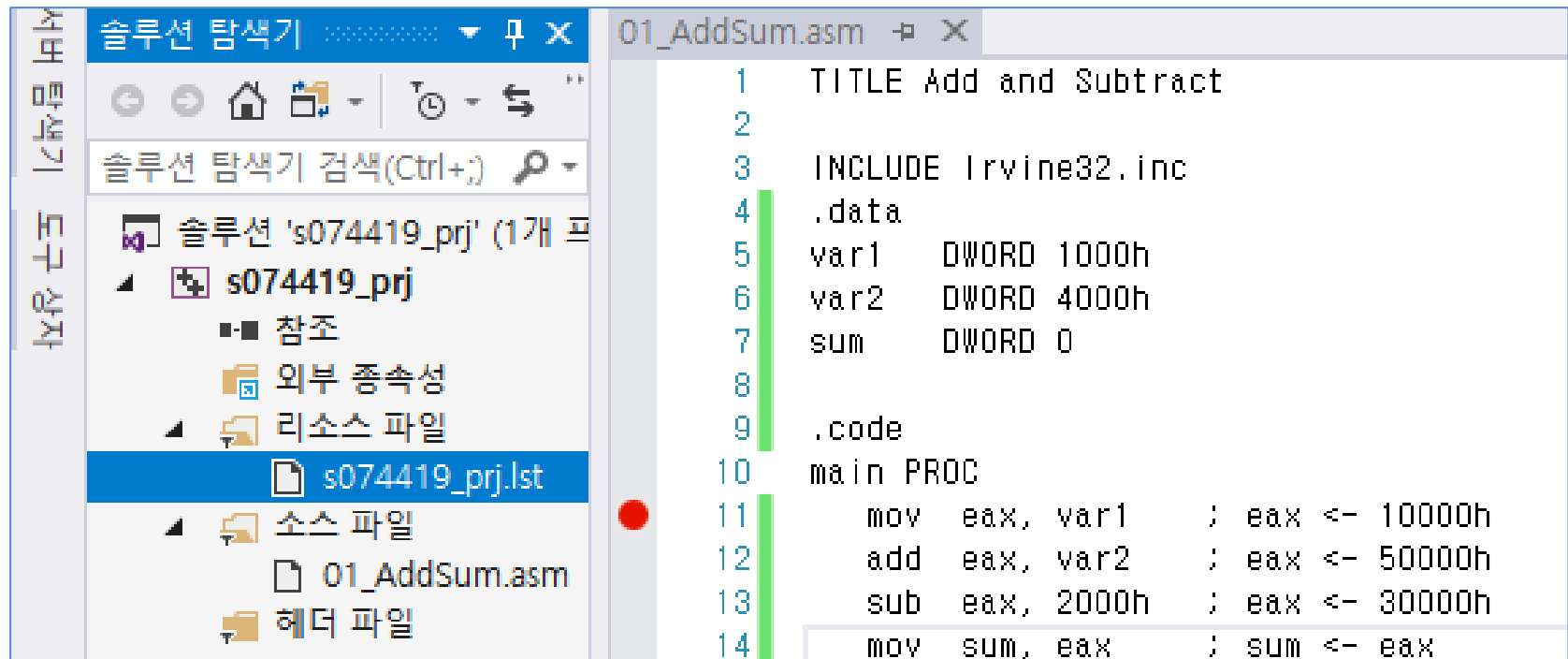


## ◆ 리스트(.list) 파일

- ◆ 리스트 파일은 어셈블리 프로그램에 대한 machine code를 보여준다.
- ◆ 또한, 각종 변수에 대한 주소, 크기, 초기 값 등을 보여준다.
- ◆ 오류가 생겼을 경우, 오류 상태를 파일에 표시한다.
- ◆ 리스트 파일은 다음과 같이 확장자가 **.lst**이다.



- ◆ 리스트 파일도 Visual Studio에 등록하여 편집 창으로 열람할 수 있다.
- ◆ 솔루션 탐색기 → 리소스 파일<sup>(1)</sup> → 마우스 우 클릭 → 추가 → 기존 항목 → 파일 **snnnnnn\_prj.lst**를 찾아 선택
- ◆ 파일을 더블 클릭하여 내용을 살펴 보자.



(1) 다른 항목에 추가해도 무방할 것이다.





## ◆ snnnnnnn\_prj.lst 파일 내용

Microsoft (R) Macro Assembler Version 14.16.27025.1

01/08/19 12:04:10

Page 1 - 1

Add and Subtrac

TITLE Add and Subtract

INCLUDE Irvine32.inc

C ; Include file for Irvine32.lib (Irvine32.inc)

C

C ;OPTION CASEMAP:NONE ; optional: make identifiers case-sensitive

C

C INCLUDE SmallWin.inc ; MS-Windows prototypes, structures, and constants

C .NOLIST

C .LIST

C

C INCLUDE VirtualKeys.inc

C ; VirtualKeys.inc

C .NOLIST

C .LIST

C

C

C .NOLIST

C .LIST

C

00000000

.data

00000000 00001000

var1 DWORD 1000h

00000004 00004000

var2 DWORD 4000h

00000008 00000000

sum DWORD 0

00000000

.code

00000000

main PROC

00000000 A1 00000000 R mov eax, var1 ; eax <- 1000h

00000005 03 05 00000004 R add eax, var2 ; eax <- 5000h

0000000B 2D 00002000 sub eax, 2000h ; eax <- 3000h

00000010 A3 00000008 R mov sum, eax ; sum <- eax

00000015 E8 00000000 E call DumpRegs

exit

00000021

main ENDP