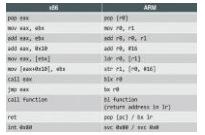
리버싱으로 보는 c언어

키워드들

- C언어 컴파일 과정
- 기본적인 코드흐름

목차

인사말



제작자 Dhhd

- 실습도구는 VISUAL STUDIO 2019
- 리버싱을 더공부하고 싶은분들에게

이건 VS2019로 만들었지 다른 컴파일러나 언어

Dev c++/gcc 등도 그 안에 설정 컴파일러 그리고 VB/C#/DELPHI 등 언어에 따라 달라질 수 있습니다

고유의 언어마다 고유의 맛이 느껴지므로 이걸 기준으로 C는 표준입니다. 한번 분석해보세요

개발자에게

C언어는 이렇게 작동합니다

한번 즐겨보세요

열심히 만들었으니 한번보시면 꽤 재미있을 것입니다

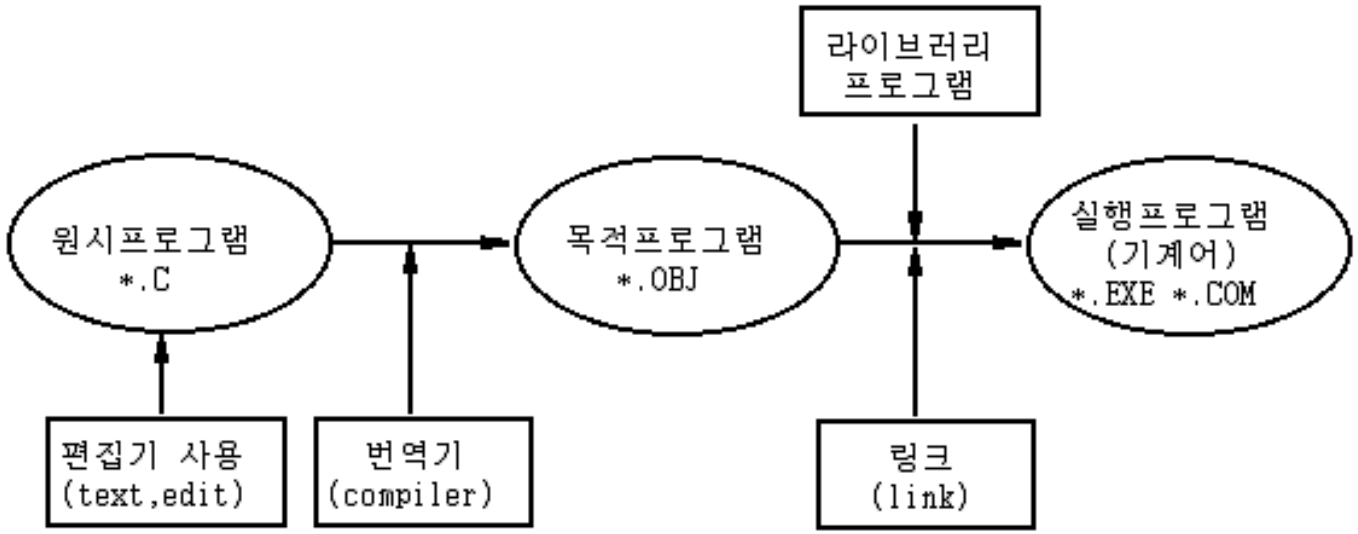
모든 예제는 https://github.com/pec2016/REVERSING-C 에서 다운로드 할수있습니다

제작자 Dhhd

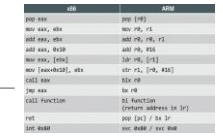
C언어 컴파일 과정

제작자 Dhhd

- C언어 컴파일 과정



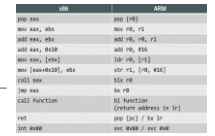
데이터단위



제작자 Dhhd

- BIT
- 데이터의 가장 작은 단위를 말하며, 0 or 1만 표현 가능하다.
- BYTE
- 8bit가 모여서 1 byte를 구성한다. 최대 255까지 표현 가능하고
- WORD
- 2byte를 합쳐서 word라고 부른다. Word는 최대 65535까지 표현 가능하다.
- DOUBLE WORD
- 2word를 합쳐서 double word라고 부르며 최대 0xffffffff까지 표현 가능하다.
- KILO BYTE
- 1024 byte를 Kbyte라고 부른다.
- MEGA BYTE
- 1024Kbyte를 Mbyte라고 부른다.

데이터단위

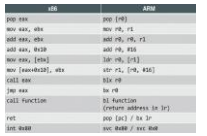


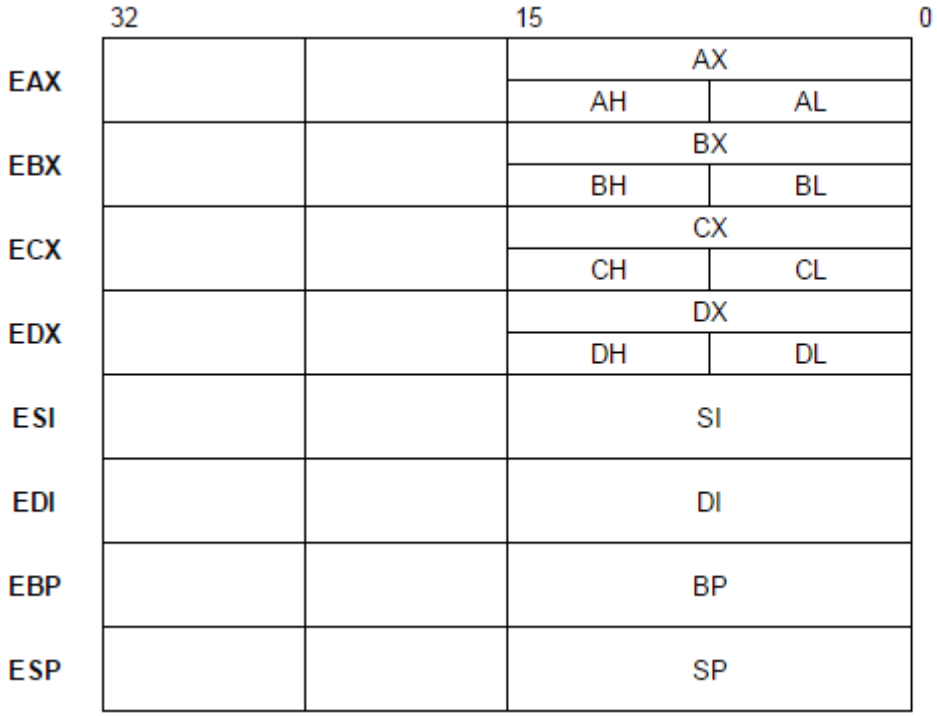
제작자 Dhhd

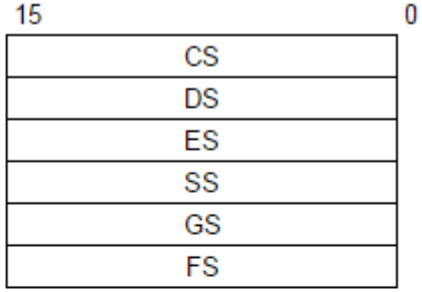
- BIT
- 데이터의 가장 작은 단위를 말하며, 0 or 1만 표현 가능하다.
- BYTE
- Byte 최대 0xFF
- WORD
- 2byte 최대 0xFFFF
- DWORD(DOUBLE WORD)
- 4byte 최대 0xFFFFFFFF
- QWORD
- 8byte 최대 0xFFFFFFFFFFFFFFFF

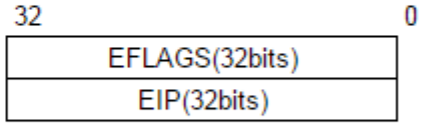
범용레지스터

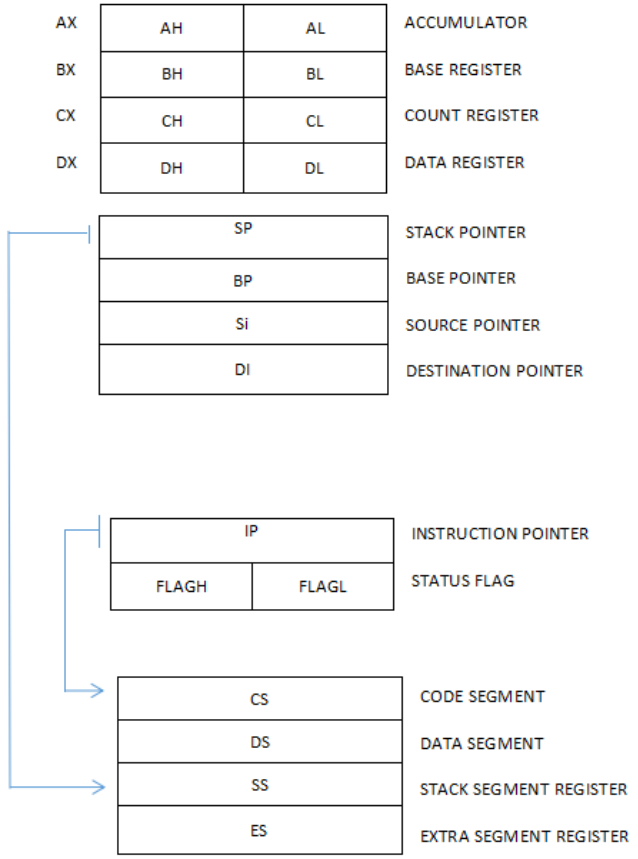
제작자 Dhhd



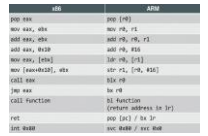








EIP

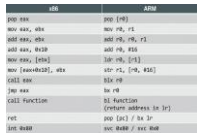


- **EIP : Instruction Pointer**

EIP는 CPU가 처리할 명령어의 주소를 나타내는 레지스터

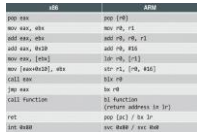
CPU는 EIP에 저장된 메모리 주소의 명령어를 하나 처리 후 자동으로 그 명령어 길이만큼 EIP를 증가한다

EIP 는 소프트웨어의 의해 조작할 수 없고 CALL, JMP, RET 와 같은 control-transfer 명령에 의해서만 영향을 받는다.



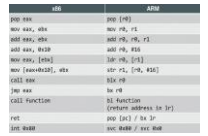
- EAX (Accumulator Register)

산술 연산을 할 때 사용되거나 함수의 리턴 값을 전달한다



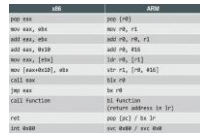
- **EBX (Base Register)**

메모리 주소 지정용 DS 세그먼트를 가르킨다



- ECX (Count Register)

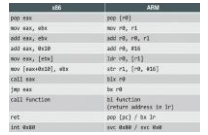
반복문에서 카운트로 사용 한다



- **EDX (Data Register)**

EAX의 확장 개념 곱하기나 나누기때 추가적 데이터 및 범용 저장용

범용 레지스터 용도



제작자 Dhhd

- **EBP (Base Pointer Register)**

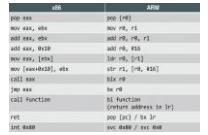
```
    char cmd[4] = { 'c','m','d','\x0' };
00BB1762  mov         byte ptr [cmd],63h
00BB1766  mov         byte ptr [ebp-0Bh],6Dh
00BB176A  mov         byte ptr [ebp-0Ah],64h
00BB176E  mov         byte ptr [ebp-9],0
```

- 스택 프레임의 처음(기준) 지점의 주소를 저장.
- main() 함수가 실행되면 가장 처음 부분의 주소
- main() 함수가 있고 function() 함수가 실행되면 function()의 처음 주소.

- **ESP (Stack Pointer Register)**

- Stack의 끝 지점 주소 가리킨다

범용 레지스터 용도



제작자 Dhhd

- **ESI (Source Index Register)**

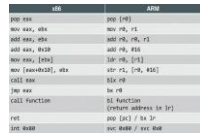
• 데이터를 연산할 때 데이터 주소 가리킨다

- **EDI (Destination Index Register)**

• 데이터를 연산할 때 **목적지의 주소** 가리킨다

• **주의 용도가 꼭맞지는 않다**

세그먼트 레지스터

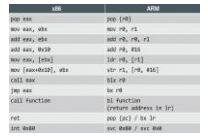


제작자 Dhhd

메모리의 한 영역에 대한 주소지정을 제공한다

- **CS** 레지스터 **:** 코드 세그먼트의 시작 주소
- **DS** 레지스터 : 데이터 세그먼트 시작 주소
- **SS** 레지스터 : 메모리 상에 스택의 구현용
- **ES** 레지스터 : 문자열 처리 명령시 목적지 저장 여분 세그먼트
- **FS, GS** 레지스터 :여분의 세그먼트 특수한 용도로 사용가능

상태 플래그



제작자 Dhhd

☐ **CF(Carry Flag)**

부호 없는 연산 결과가 용량보다 클 때 세트(1)된다.

☐ **ZF(Zero Flag)**

연산 결과가 0일 때 세트(1)된다. 연산 결과가 0이 아닐 때 해제(0)된다.

☐ **OF(Overflow Flag)**

부호 있는 연산 결과가 용량보다 클 때 세트(1)된다.

☐ **SF(Sign Flag)**

연산 결과가 음수가 되었을 때 세트(1)된다. 연산 결과가 양수가 되었을 때 해제(0)된다.

☐ **PF(Parity Flag)**

PF => 연산결과에서 1이 짝수개이면 1

☐ **AF(Auximiliary carry Flag)**

AF => 10진수 연산시 보정(자리올림, 빌림 등)이 필요하면 1

CF => Carry Flag

OF => Overflow Flag

SF => Sign Flag

ZF => Zero Flag

AF => Auximiliary carry Flag

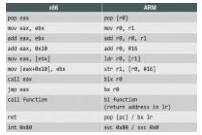
PF => Parity Flag

DF => Direction Flag

IF => Interrupt Flag

TF => Trap Flag

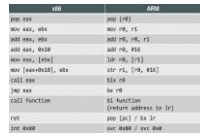
어셈블리 명령어의 구성



제작자 Dhhd

```
   L1:       mov           %eax        ,       %ebx          ;comment
+-----+ +----------+ +-------------+ +-------------+ +----------+
|Label| |작동 코드 | |제 1 오퍼랜드| |제 2 오퍼랜드| |설명문    |
+-----+ +----------+ +------+------+ +-------+-----+ +----------+
                            |               ↑
                            └---------------┘
                          조작의 방향
```

주소 지정 방식(Addressing Mode)



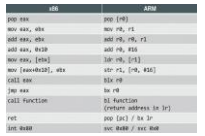
제작자 Dhhd

1. 즉시 지정방식(immediate addressing)

- mov eax, 0x1 : eax에 (16진수)1을 값을 넣는(할당) 방식이다.
- 이렇게 메모리(기억장치)의 주소의 내용을 꺼내지 않고 직접 값을 대응시키는 방식을 즉 시지정방식이라고 한다.

```
B8 01000000 | mov eax,1
```

주소 지정 방식(Addressing Mode)



제작자 Dhhd

2. 레지스터 지정방식(register addressing)

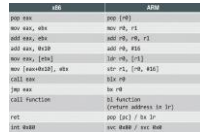
- mov esp, ebp : 레지스터 ebp에 레지스터 esp의 값을 넣는다.(할당 개념)

나중에 알 수 있겠지만, 위의 명령은 스택포인터를 베이스 포인터에 넣는 명령으로 함수가 시작될때 ebp의 값(일종의 시작기준점)을 정하는 명령이다.

- 레지스터에서 직접 레지스터로 값을 대응시키는 방식을 레지스터 지정방식이라고 한다.
- 속도는 빠르지만 레지스터의 크기(32비트)로 인해 크기가 제한된다.

```
8BE5    mov esp,ebp
```

주소 지정 방식(Addressing Mode)



제작자 Dhhd

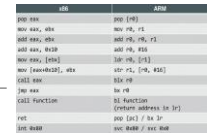
3. 직접 주소 지정방식(directly addressing)

- mov eax, 0x80482f2 : 주소 0x80482f2에 있는 값을 eax에 할당한다.

- 가장 일반적인 주소지정방식이며, 메모리의 주소를 직접 지정해서 바로 찾아오는 방식이 다. 즉 eax레지스터에 0x80482f2주소의 내용을 로드(load)한다는 의미이다.

```
A1 F2820408          mov eax,dword ptr ds:[80482F2]
```

주소 지정 방식(Addressing Mode)



4) 변위를 갖는 베이스 인덱스 주소 지정

MOV AL, [BX+SI+0x20]

MOV AL, [BP+SI+0x20]

5. 베이스 상대 주소 지정 방식

- mov eax, [esi+ 0x4]

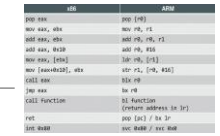
esi 레지스터에서 4(byte)를 더한 주소의 값을 eax레지스터에 할당한다.

통 레지스터의 크기가 4byte이기 때문에 레지스터 다음 주소를 의미한다.

문자열 열산이나 메모리 블록 전송등에 나오는 방식이다.

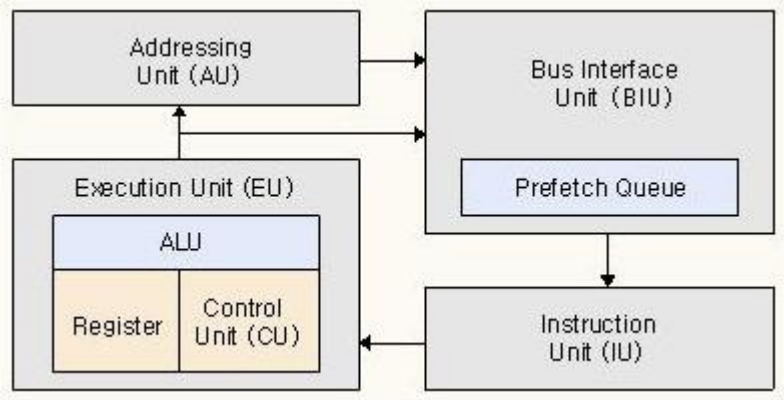
```
67:8A40 14      mov al,byte ptr ds:[bx+si+14]
8B46 04         mov eax,dword ptr ds:[esi+4]
```

어셈블리어 와 니모닉 언어



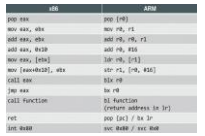
제작자 Dhhd

- 어셈블리어란 컴퓨터(엄밀히는 CPU)가 이해할 수 있는 기계어 명령들을 사람들이 보다 쉽게 이해할 수 있도록
- 간단한 니모닉 언어(mnemonic symbol, 연상문자)로 나타낸 것. 이 어셈블리어들은 기계어와 1:1로 매칭된다.
- (기계어 한 바이트당 하나의 니모닉 언어를 가지나 자주 쓰는 기계어의 경우 둘 이상의 바이트가 매칭되기도 한다.)



기계어	어원	니모닉 언어(어셈블리어)
B8 01000000	Move eax 1	mov eax,1
E9	JUMP MUST	jmp
38C0	compare eax,eax	cmp
C3	Return	ret

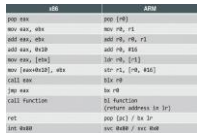
어셈블리어



제작자 Dhhd

명령어	예제	설명	분류
push	push %eax	eax의 값을 스택에 저장.	스택 조작
pop	pop %eax	스택 가장 상위에 있는 값을 꺼내서 eax에 저장	스택 조작
mov	mov %eax, %ebx	메모리나 레지스터의 값을 옮길때 사용	데이터 이동
lea	leal(%esi), %ecx	%esi의 주소값을 %ecx에 옮긴다.	주소 이동
inc	inc %eax	%eax의 값을 1 증가시킨다.	데이터 조작
dec	dec %eax	%eax의 값을 1 감소시킨다.	데이터 조작
add	add %eax, %ebx	레지스터나 메모리의 값을 덧셈할 때 쓰인다.	논리, 연산
sub	sub $0x8, %esp	레지스터나 메모리의 값을 뺄셈할 때 쓰인다.	논리, 연산
call	call proc	프로시져를 호출한다.	프로시져
ret	ret	호출했던 바로 다음 지점으로 이동.	프로시져
cmp	cmp %eax, %ebx	레지스터와 레지스터값을 비교	비교
jmp	jmp proc	특정한 곳으로 분기	분기
int	int $0x80	OS에 할당된 인터럽트 영역을 system call	인터럽트
nop	nop	아무 동작도 하지 않는다.(No Operation)	

어셈블리어 cmp



제작자 Dhhd

1. cmp (compare)

레지스터나 메모리의 값을 변경하지 않는다

플래그 레지스터에만 영향을 준다 (flag register - 상태 레지스터)

-> SF(부호), CF(올림수), ZF(0,두개의 비교값이 같을 때)

두 피연산자의 비교 (CMP)

cmp eax, ebx

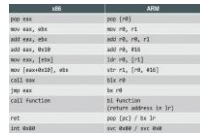
=> **eax-ebx**

=> 결과가 0인 경우: 같은 경우 => SF:0, CF:0, ZF : 1

=> 결과가 음수인 경우 : eax < ebx => SF:1 , CF:0, ZF: 0

=> 결과가 양수인 경우 : eax > ebx => SF:0, ZF:0 , CF:0

분기조건문



제작자 Dhhd

JA Jump Above

JG Jump Grater

JGE Jump Grater Equal <-> **JNL** Jump Not Less

JB Jump Below

JL Jump Less

JLE Jump Less Equal <-> **JNG** JUMP Not Grater

JC Jump Carry (CF=1)

JS Jump Sign (SF=1)

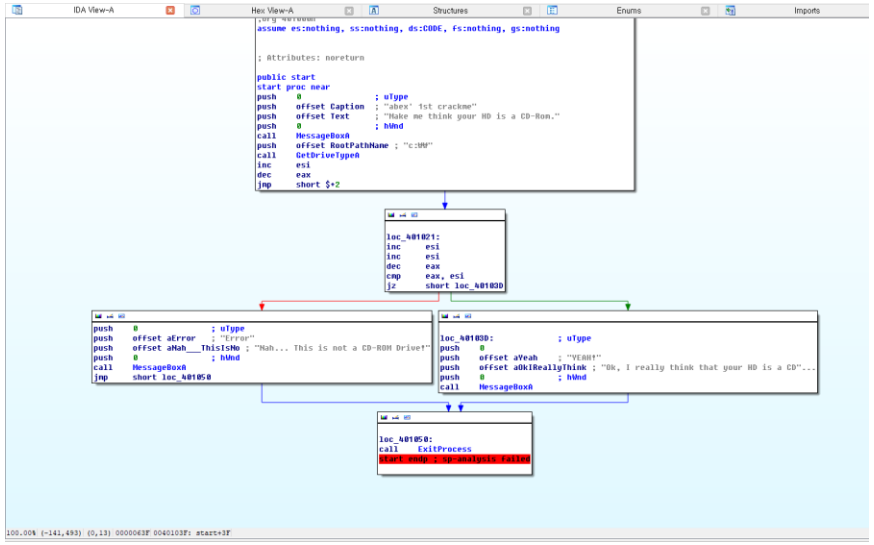
JZ Jump Zero (ZF=1)

명령어1	명령어2	설명	부등호 조건	Flag 조건
JA	JNBE	크면 분기 (부호 없이 비교[unsigned])	OP1>OP2면 분기	CF=0 & ZF=0
JAE	JNB	크거나 같으면 분기 (부호 없이 비교[unsigned])	OP1≥OP2면 분기	CF=0 | ZF=1
JB	JNAE	작으면 분기 (부호 없이 비교[unsigned])	OP1<OP2면 분기	CF=1
JBE	JNA	작거나 같으면 분기 (부호 없이 비교[unsigned])	OP1≤OP2면 분기	CF=1 | ZF=1
JG	JNLE	크면 분기 (부호 있는 비교[signed])	OP1>OP2면 분기	ZF=0&SF ==OF
JGE	JNL	크거나 같으면 분기 (부호 있는 비교[signed])	OP1≥OP2면 분기	SF==OF
JL	JNGE	작으면 분기 (부호 있는 비교[signed])	OP1<OP2면 분기	SF!=OF
JLE	JNG	작거나 같으면 분기 (부호 있는 비교[signed])	OP1≤OP2면 분기	ZF==1|SF!=OF
JE		같으면 분기	OP1=OP2면 분기	ZF==1
JNE		같지 않으면 분기	OP1≠OP2면 분기	ZF==0
JC		Carry Flag가 Set되면 분기		CF==1
JNC		Carry Flag가 해제되어 있으면 분기		CF==0
JO		Overflow가 Set되어 있으면 분기		OF==1
JNO		Overflow가 해제되어 있으면 분기		OF==0
JS		Sign Flag가 Set되어 있으면(음수이면) 분기		SF==1
JNS		Sign Flag가 해제되어 있으면 (양수이면) 분기		SF==0
JZ		Zero Flag가 Set되어 있으면 분기		ZF==1
JNZ		Zero Flag가 해제되어 있으면 분기		ZF==0
JP		Parity Flag가 Set되어 있으면 분기		PF==1
JNP		Parity Flag가 해제되어 있으면 분기		PF==0

제작자 Dhhd

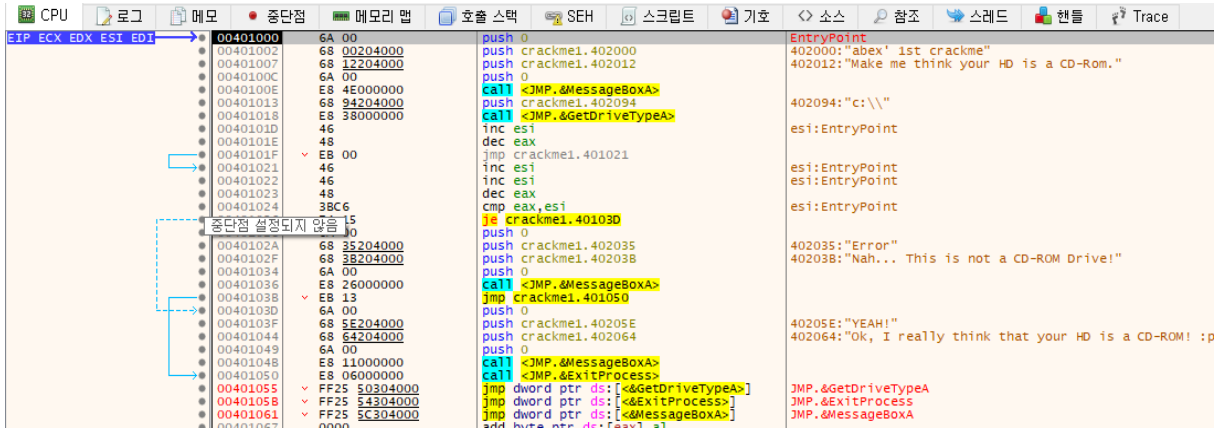
정적 분석

파일 실행을 안한다
어셈이나 코드흐름등을 보고 분석한다
도구 : IDA,기드라

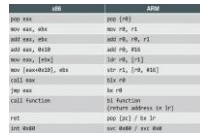


동적 분석

파일 실행을 한다
동적으로 실행하면서 분석한다
도구 : x64DBG



• 함수호출구약



cdecl
```
test(1, 2, 3, 4)
sub esp, 16
push 4
push 3
push 2
push 1
call test
add esp, 16
```

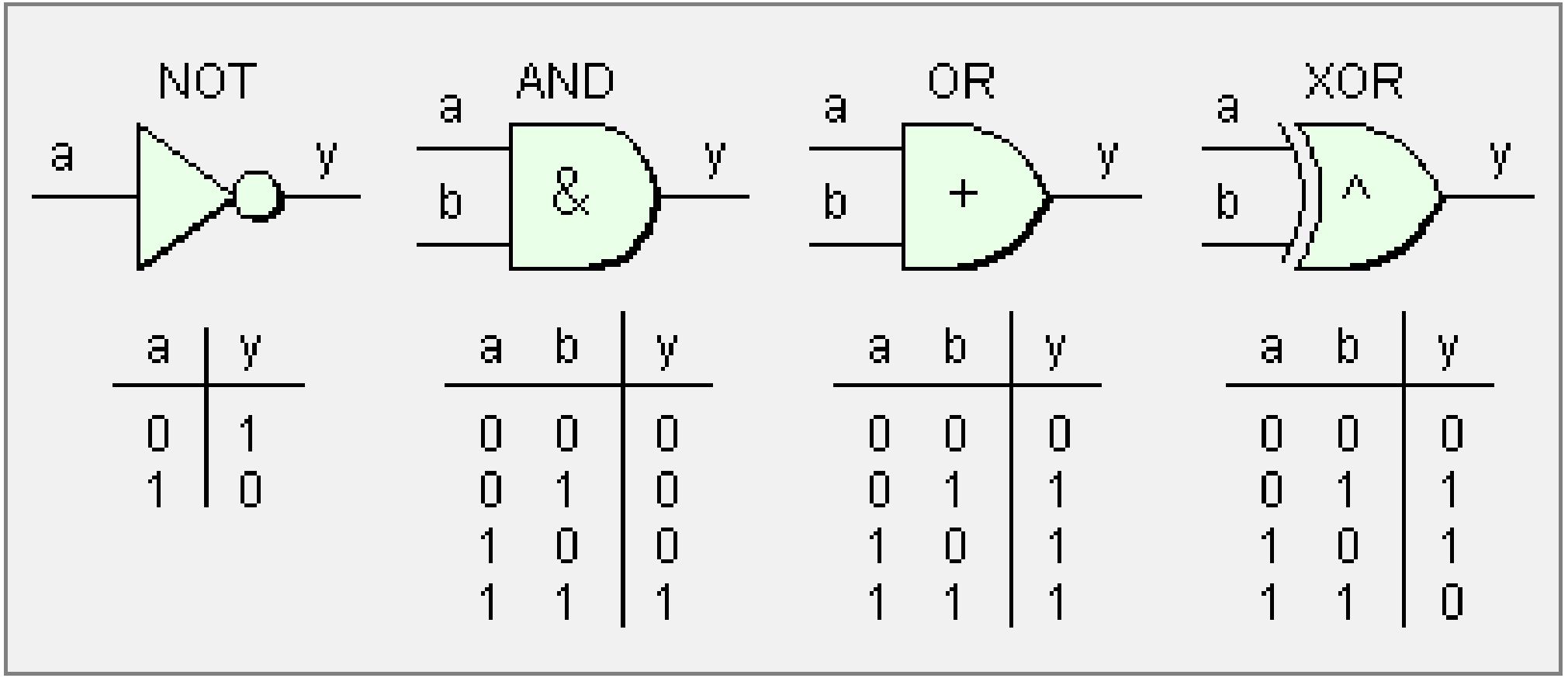
stdcall
```
test(1, 2, 3, 4)
sub esp, 16
push 4
push 3
push 2
push 1
call test
```

fastcall <파라미터 : CPU 레지스터 2개 이용. (ECX, EDX)>
```
test(1, 2, 3, 4)
sub esp, 16
push 4
push 3
mov edx, 2
mov ecx, 1
call test
```

stdcall과 fastcall에서는 test함수 내부에서 add esp, 16 수행한다

비트연산

제작자 Dhhd



a	y
0	1
1	0

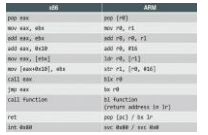
a	b	y
0	0	0
0	1	0
1	0	0
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	1

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

HELLO
WORLD

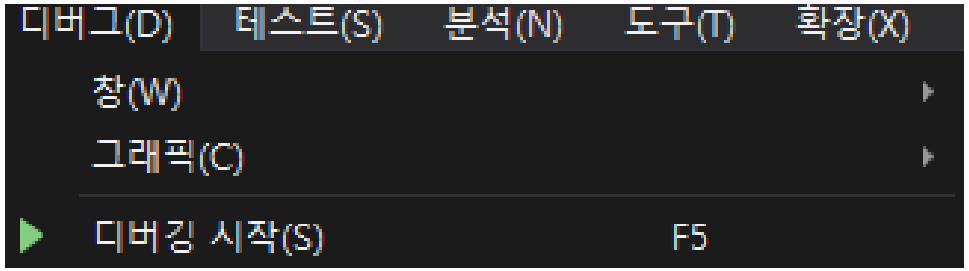
Debug?



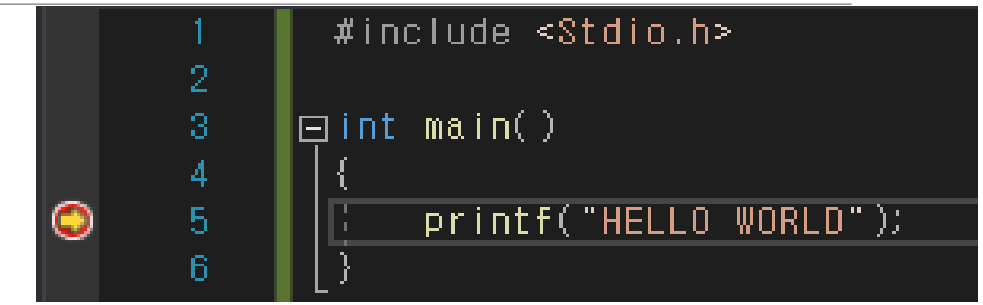
- VISUAL STUDIO 2019

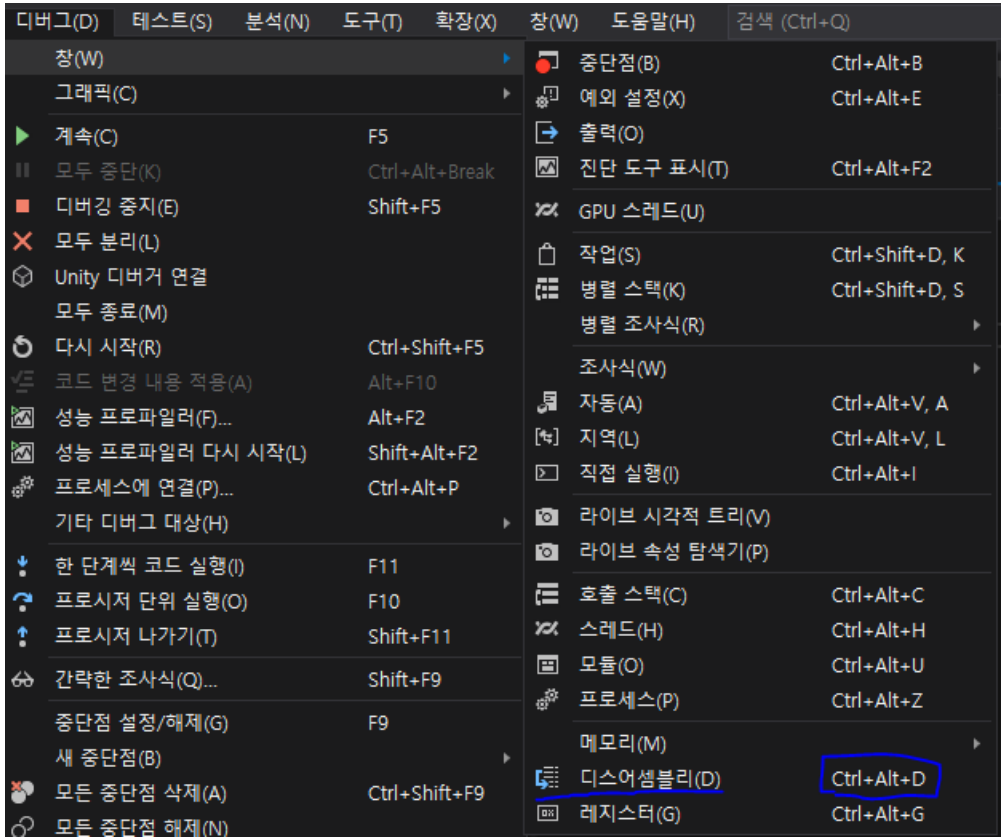
```
1  #include <Stdio.h>
2
3  int main()
4  {
5      printf("HELLO WORLD");
6  }
```

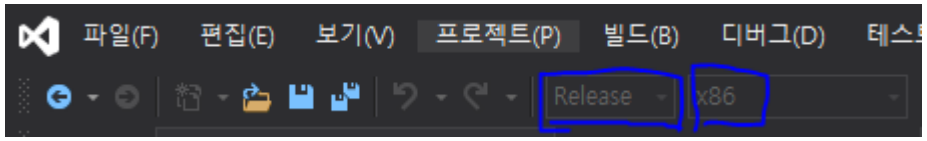
- F9로 빨간점(BP[BREAKPOINT])을 생성



- F5로 디버깅 시작
- 디스어셈블리는 Ctrl+Alt+D







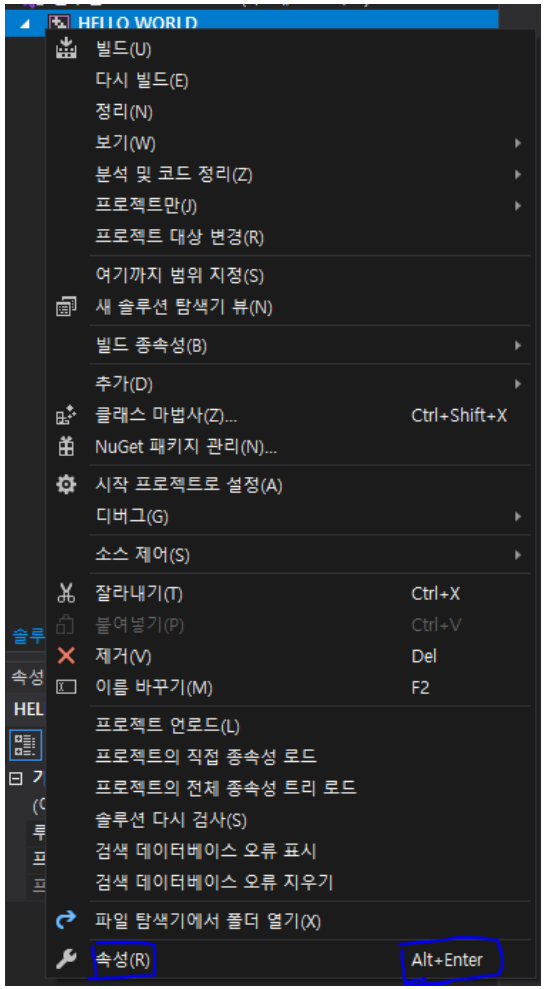
- Release

```
    printf("HELLO WORLD");
007C1040  push        offset string "HELLO WORLD" (07C2100h)
007C1045  call        printf (07C1010h)
007C104A  add         esp,4
}
007C104D  xor         eax,eax
007C104F  ret
```

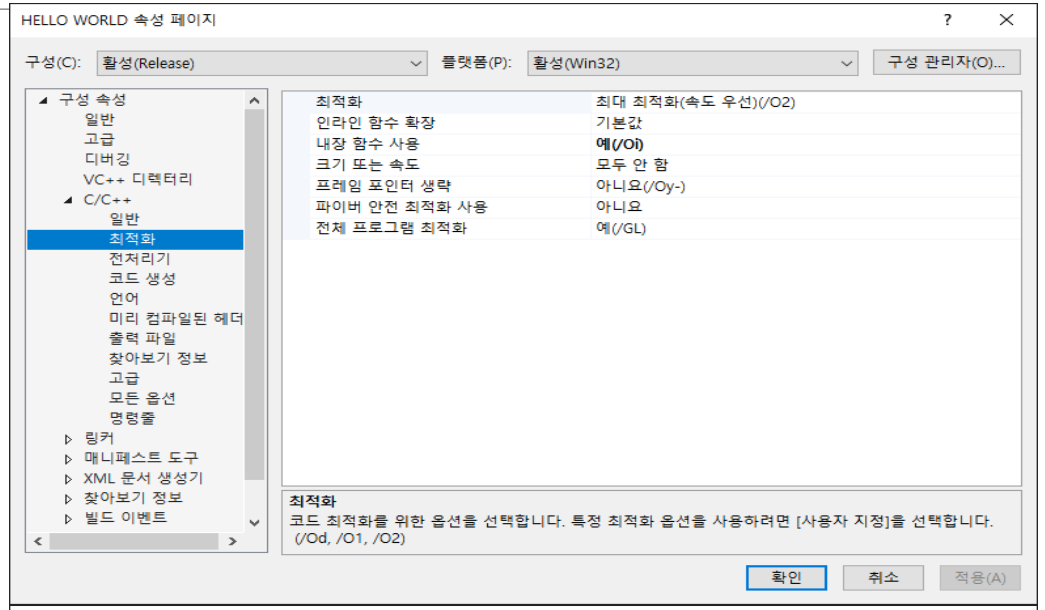
- Debug

```
#include <Stdio.h>

int main()
{
00371810  push        ebp
00371811  mov         ebp,esp
00371813  sub         esp,0C0h
00371819  push        ebx
0037181A  push        esi
0037181B  push        edi
0037181C  lea         edi,[ebp-0C0h]
00371822  mov         ecx,30h
00371827  mov         eax,0CCCCCCCCh
0037182C  rep stos    dword ptr es:[edi]
0037182E  mov         ecx,offset _9F9313A7_소스@c (037C003h)
00371833  call        @__CheckForDebuggerJustMyCode@4 (0371217h)
    printf("HELLO WORLD");
00371838  push        offset string "HELLO WORLD" (0377B30h)
0037183D  call        _printf (0371046h)
00371842  add         esp,4
}
00371845  xor         eax,eax
00371847  pop         edi
00371848  pop         esi
00371849  pop         ebx
0037184A  add         esp,0C0h
00371850  cmp         ebp,esp
00371852  call        __RTC_CheckEsp (0371221h)
00371857  mov         esp,ebp
00371859  pop         ebp
0037185A  ret
```

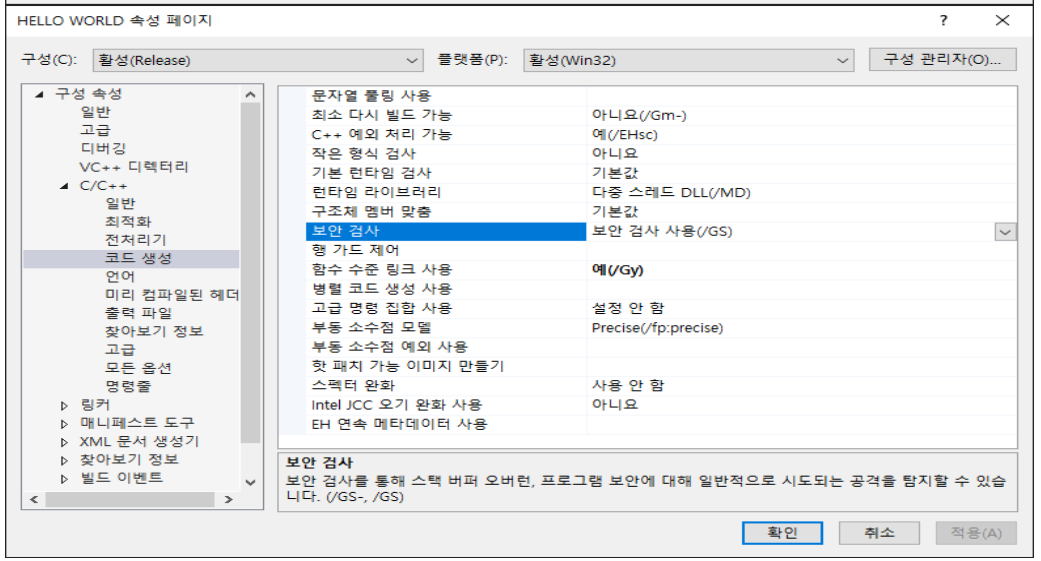
- 속성 Alt+Enter

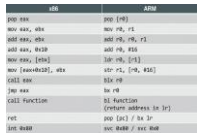


최적화

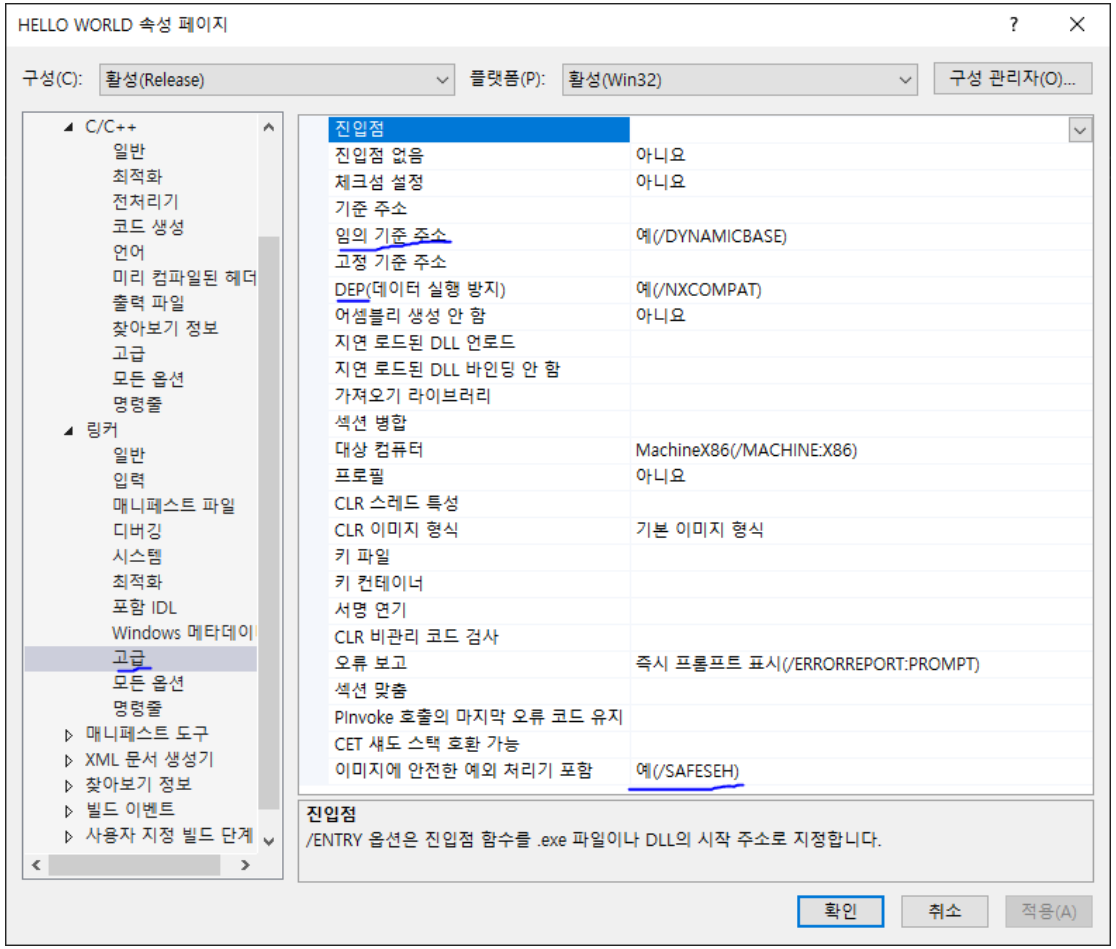


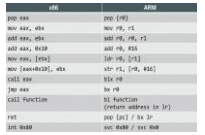
GS





- ASLR/DEP/SAFESEH





- 보안기능들 다 사용안함 설정후 디스어셈블리 그전 사진

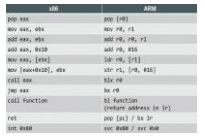
```
#include <Stdio.h>

int main( )
{
00401080  push        ebp
00401081  mov         ebp,esp
    printf("HELLO WORLD");
00401083  push        403020h
00401088  call        printf (0401040h)
0040108D  add         esp,4
}
00401090  xor         eax,eax
00401092  pop         ebp
00401093  ret
```

```
    printf("HELLO WORLD");
007C1040  push        offset string "HELLO WORLD" (07C2100h)
007C1045  call        printf (07C1010h)
007C104A  add         esp,4
}
007C104D  xor         eax,eax
007C104F  ret
```

- 디버깅 총정리
- F9 BP/F10 step over/F11 step in/shift + F11 함수나가기

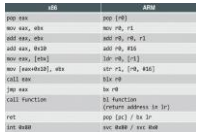
변수

int? char?



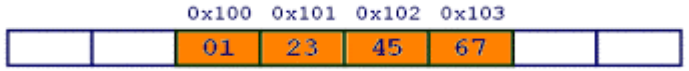
- 변수의 특징

C Basic Data Types	32-bit CPU		64-bit CPU	
	Size (bytes)	Range	Size (bytes)	Range
char	1	-128 to 127	1	-128 to 127
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	8	- 9,223,372,036,854,775,808- 9,223,372,036,854,775,807
long long	8	9,223,372,036,854,775,808- 9,223,372,036,854,775,807	8	9,223,372,036,854,775,808- 9,223,372,036,854,775,807
float	4	3.4E +/- 38	4	3.4E +/- 38
double	8	1.7E +/- 308	8	1.7E +/- 308



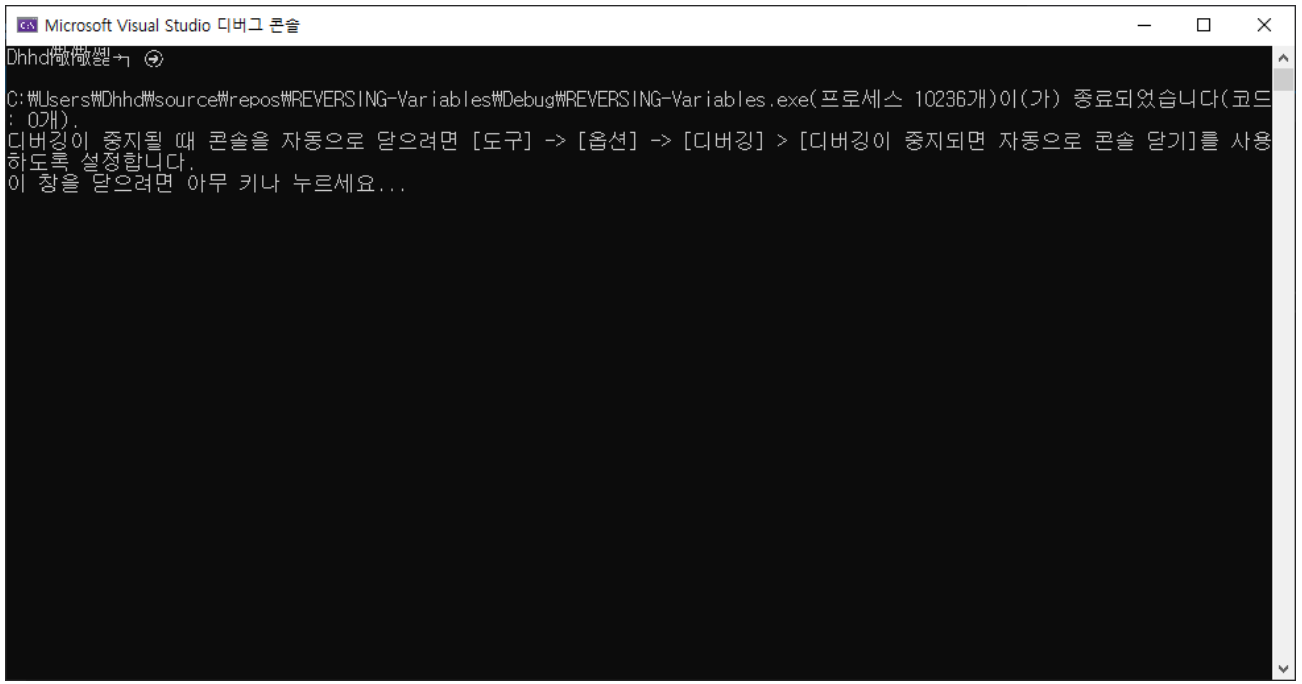
- 소스코드

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int stringDhhd = 0x64686844;
6      puts(&stringDhhd);
7  }
8
```



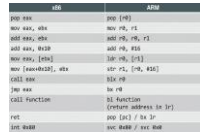
Big Endian

Little Endian

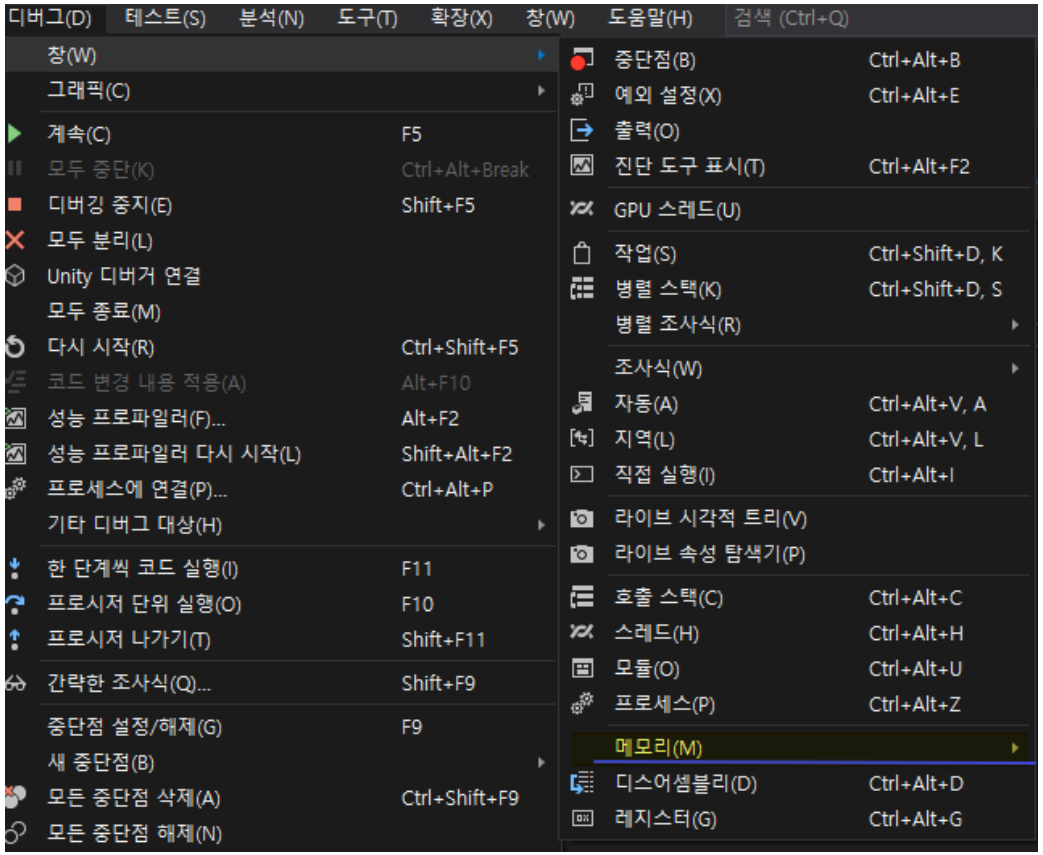


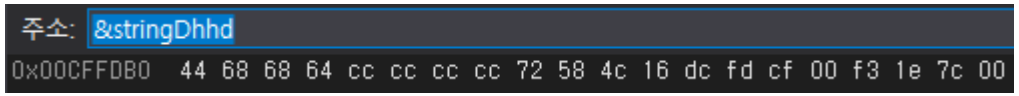
제어 문자 | 공백 문자 | 구두점 | 숫자 | 알파벳

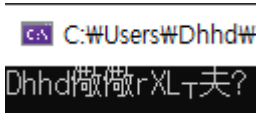
10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자
0	0x00	NUL	32	0x20	SP	64	0x40	@	96	0x60	`
1	0x01	SOH	33	0x21	!	65	0x41	A	97	0x61	a
2	0x02	STX	34	0x22	"	66	0x42	B	98	0x62	b
3	0x03	ETX	35	0x23	#	67	0x43	C	99	0x63	c
4	0x04	EOT	36	0x24	$	68	0x44	D	100	0x64	d
5	0x05	ENQ	37	0x25	%	69	0x45	E	101	0x65	e
6	0x06	ACK	38	0x26	&	70	0x46	F	102	0x66	f
7	0x07	BEL	39	0x27	'	71	0x47	G	103	0x67	g
8	0x08	BS	40	0x28	(	72	0x48	H	104	0x68	h
9	0x09	HT	41	0x29	)	73	0x49	I	105	0x69	i
10	0x0A	LF	42	0x2A	*	74	0x4A	J	106	0x6A	j
11	0x0B	VT	43	0x2B	+	75	0x4B	K	107	0x6B	k
12	0x0C	FF	44	0x2C	,	76	0x4C	L	108	0x6C	l
13	0x0D	CR	45	0x2D	-	77	0x4D	M	109	0x6D	m
14	0x0E	SO	46	0x2E	.	78	0x4E	N	110	0x6E	n
15	0x0F	SI	47	0x2F	/	79	0x4F	O	111	0x6F	o
16	0x10	DLE	48	0x30	0	80	0x50	P	112	0x70	p
17	0x11	DC1	49	0x31	1	81	0x51	Q	113	0x71	q
18	0x12	DC2	50	0x32	2	82	0x52	R	114	0x72	r
19	0x13	DC3	51	0x33	3	83	0x53	S	115	0x73	s
20	0x14	DC4	52	0x34	4	84	0x54	T	116	0x74	t
21	0x15	NAK	53	0x35	5	85	0x55	U	117	0x75	u
22	0x16	SYN	54	0x36	6	86	0x56	V	118	0x76	v
23	0x17	ETB	55	0x37	7	87	0x57	W	119	0x77	w
24	0x18	CAN	56	0x38	8	88	0x58	X	120	0x78	x
25	0x19	EM	57	0x39	9	89	0x59	Y	121	0x79	y
26	0x1A	SUB	58	0x3A	:	90	0x5A	Z	122	0x7A	z
27	0x1B	ESC	59	0x3B	;	91	0x5B	[	123	0x7B	{
28	0x1C	FS	60	0x3C	<	92	0x5C	₩	124	0x7C	|
29	0x1D	GS	61	0x3D	=	93	0x5D	]	125	0x7D	}
30	0x1E	RS	62	0x3E	>	94	0x5E	^	126	0x7E	~
31	0x1F	US	63	0x3F	?	95	0x5F	_	127	0x7F	DEL



- 메모리 Ctrl+Alt+M 조금뒤에 1







- 이렇게 바꾸었을때 출력결과는

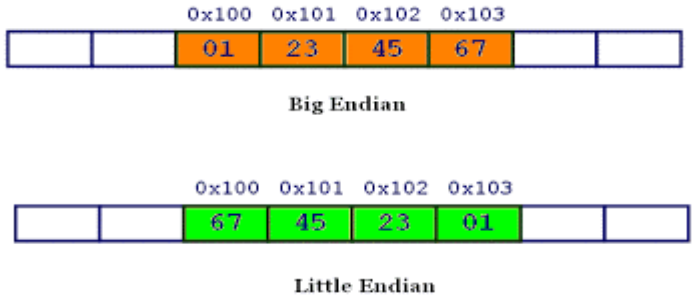
```c
#include <stdio.h>

int main()
{
    int stringDhhd = 0x00434241;
    puts(&stringDhhd);
}
```

메모리 1

주소: 0x00AFF714

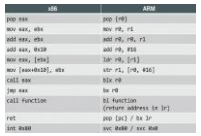
0x00AFF714 41 42 43 00

- 힌트들

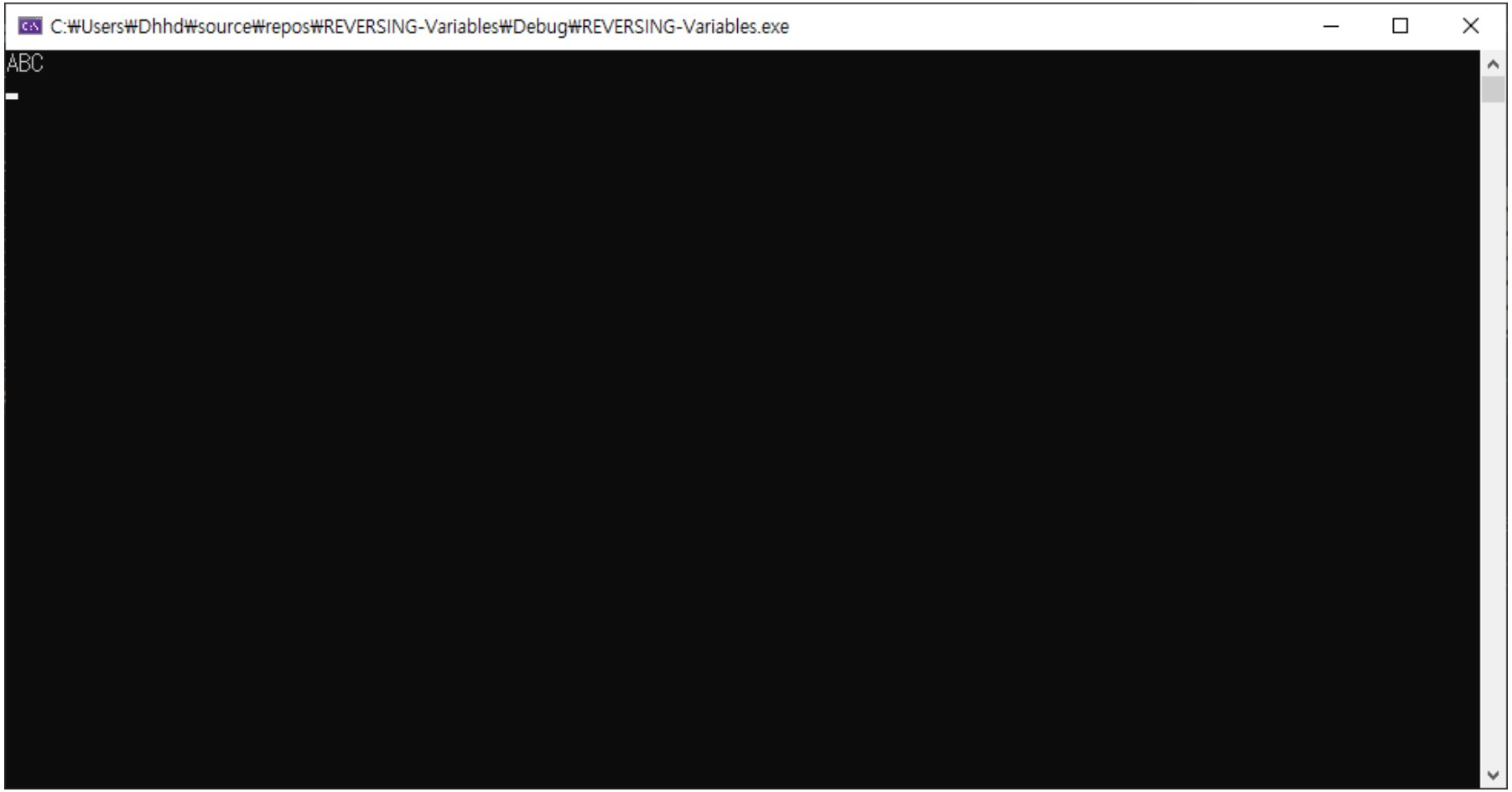


제어 문자 | 공백 문자 | 구두점 | 숫자 | 알파벳

10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자
0	0x00	NUL	32	0x20	SP	64	0x40	@	96	0x60	`
1	0x01	SOH	33	0x21	!	65	0x41	A	97	0x61	a
2	0x02	STX	34	0x22	"	66	0x42	B	98	0x62	b
3	0x03	ETX	35	0x23	#	67	0x43	C	99	0x63	c
4	0x04	EOT	36	0x24	$	68	0x44	D	100	0x64	d
5	0x05	ENQ	37	0x25	%	69	0x45	E	101	0x65	e
6	0x06	ACK	38	0x26	&	70	0x46	F	102	0x66	f
7	0x07	BEL	39	0x27	'	71	0x47	G	103	0x67	g
8	0x08	BS	40	0x28	(	72	0x48	H	104	0x68	h
9	0x09	HT	41	0x29	)	73	0x49	I	105	0x69	i
10	0x0A	LF	42	0x2A	*	74	0x4A	J	106	0x6A	j
11	0x0B	VT	43	0x2B	+	75	0x4B	K	107	0x6B	k
12	0x0C	FF	44	0x2C	,	76	0x4C	L	108	0x6C	l
13	0x0D	CR	45	0x2D	-	77	0x4D	M	109	0x6D	m
14	0x0E	SO	46	0x2E	.	78	0x4E	N	110	0x6E	n
15	0x0F	SI	47	0x2F	/	79	0x4F	O	111	0x6F	o
16	0x10	DLE	48	0x30	0	80	0x50	P	112	0x70	p
17	0x11	DC1	49	0x31	1	81	0x51	Q	113	0x71	q
18	0x12	DC2	50	0x32	2	82	0x52	R	114	0x72	r
19	0x13	DC3	51	0x33	3	83	0x53	S	115	0x73	s
20	0x14	DC4	52	0x34	4	84	0x54	T	116	0x74	t
21	0x15	NAK	53	0x35	5	85	0x55	U	117	0x75	u
22	0x16	SYN	54	0x36	6	86	0x56	V	118	0x76	v
23	0x17	ETB	55	0x37	7	87	0x57	W	119	0x77	w
24	0x18	CAN	56	0x38	8	88	0x58	X	120	0x78	x
25	0x19	EM	57	0x39	9	89	0x59	Y	121	0x79	y
26	0x1A	SUB	58	0x3A	:	90	0x5A	Z	122	0x7A	z
27	0x1B	ESC	59	0x3B	;	91	0x5B	[	123	0x7B	{
28	0x1C	FS	60	0x3C	<	92	0x5C	₩	124	0x7C	|
29	0x1D	GS	61	0x3D	=	93	0x5D	]	125	0x7D	}
30	0x1E	RS	62	0x3E	>	94	0x5E	^	126	0x7E	~
31	0x1F	US	63	0x3F	?	95	0x5F	_	127	0x7F	DEL

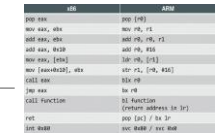


- 정답


C:₩Users₩Dhhd₩source₩repos₩REVERSING-Variables₩Debug₩REVERSING-Variables.exe
ABC

연산

+-/*



01

02

03

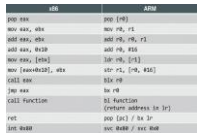
04

05

- +-/*의 흐름

```
1   #include <stdio.h>
2
3   int main()
4   {
5       int a = 1;
6       int b = 3;
7       printf("%d ₩n", a + b);
8       printf("%d ₩n", a - b);
9       printf("%d ₩n", a / b);
10      printf("%d ₩n", a * b);
11  }
```

```
    printf("%d ₩n", a + b);
00414106  mov         eax,dword ptr [a]
00414109  add         eax,dword ptr [b]
0041410C  push        eax
0041410D  push        offset string "%d" (0417BCCh)
00414112  call        _printf (0411375h)
00414117  add         esp,8
    printf("%d ₩n", a - b);
0041411A  mov         eax,dword ptr [a]
0041411D  sub         eax,dword ptr [b]
00414120  push        eax
00414121  push        offset string "%d" (0417BCCh)
00414126  call        _printf (0411375h)
0041412B  add         esp,8
    printf("%d ₩n", a / b);
0041412E  mov         eax,dword ptr [a]
00414131  cdq
00414132  idiv        eax,dword ptr [b]
00414135  push        eax
00414136  push        offset string "%d" (0417BCCh)
0041413B  call        _printf (0411375h)
00414140  add         esp,8
    printf("%d ₩n", a * b);
00414143  mov         eax,dword ptr [a]
00414146  imul        eax,dword ptr [b]
0041414A  push        eax
0041414B  push        offset string "%d" (0417BCCh)
00414150  call        _printf (0411375h)
00414155  add         esp,8
```



01

02

03

04

05

- 비트연산(기본)

```
12      printf("%d ₩n", a >> 2);
13      printf("%d ₩n", a << b);
14      printf("%d ₩n", a & b);
15      printf("%d ₩n", a ^ 2);
16      printf("%d ₩n", a | 0xFF);
```

AND		
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

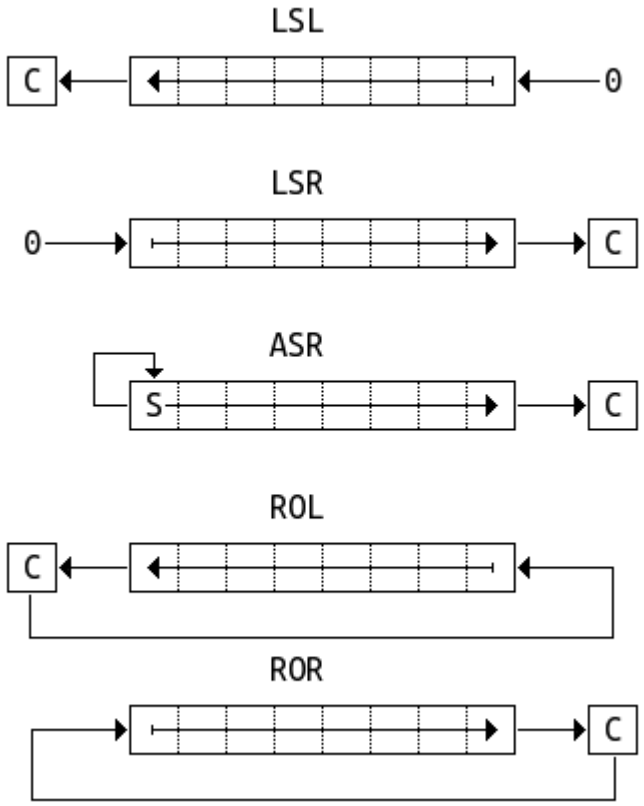
OR		
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

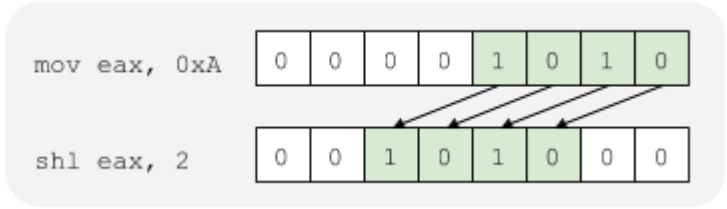
XOR		
A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

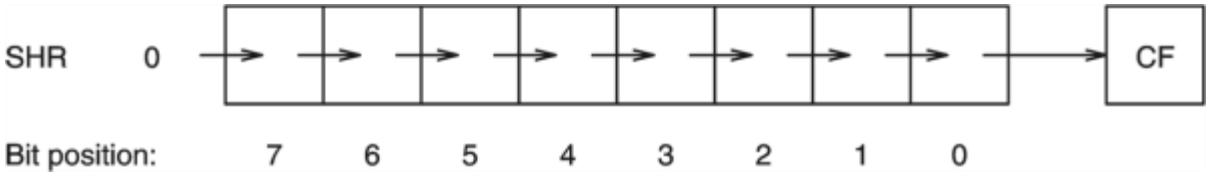
exclusive-OR

```
    printf("%d ₩n", a >> 2);
00414E58  mov         eax,dword ptr [a]
00414E5B  sar         eax,2
00414E5E  push        eax
00414E5F  push        offset string "%d" (0417BCCh)
00414E64  call        _printf (0411375h)
00414E69  add         esp,8
    printf("%d ₩n", a << b);
00414E6C  mov         eax,dword ptr [a]
00414E6F  mov         ecx,dword ptr [b]
00414E72  shl         eax,cl
00414E74  push        eax
00414E75  push        offset string "%d" (0417BCCh)
00414E7A  call        _printf (0411375h)
00414E7F  add         esp,8
    printf("%d ₩n", a & b);
00414E82  mov         eax,dword ptr [a]
00414E85  and         eax,dword ptr [b]
00414E88  push        eax
00414E89  push        offset string "%d" (0417BCCh)
00414E8E  call        _printf (0411375h)
00414E93  add         esp,8
    printf("%d ₩n", a ^ 2);
00414E96  mov         eax,dword ptr [a]
00414E99  xor         eax,2
00414E9C  push        eax
00414E9D  push        offset string "%d" (0417BCCh)
00414EA2  call        _printf (0411375h)
00414EA7  add         esp,8
    printf("%d ₩n", a | 0xFF);
00414EAA  mov         eax,dword ptr [a]
00414EAD  or          eax,0FFh
00414EB2  push        eax
00414EB3  push        offset string "%d" (0417BCCh)
00414EB8  call        _printf (0411375h)
00414EBD  add         esp,8
```

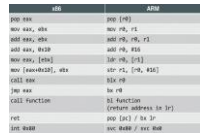
- 시프트연산







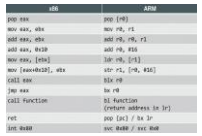

```
18      __asm
19      {
20          mov         eax, dword ptr[a]
21          ror         al, 5
22          push        eax
23      }
24      printf("%d");
25  }
```



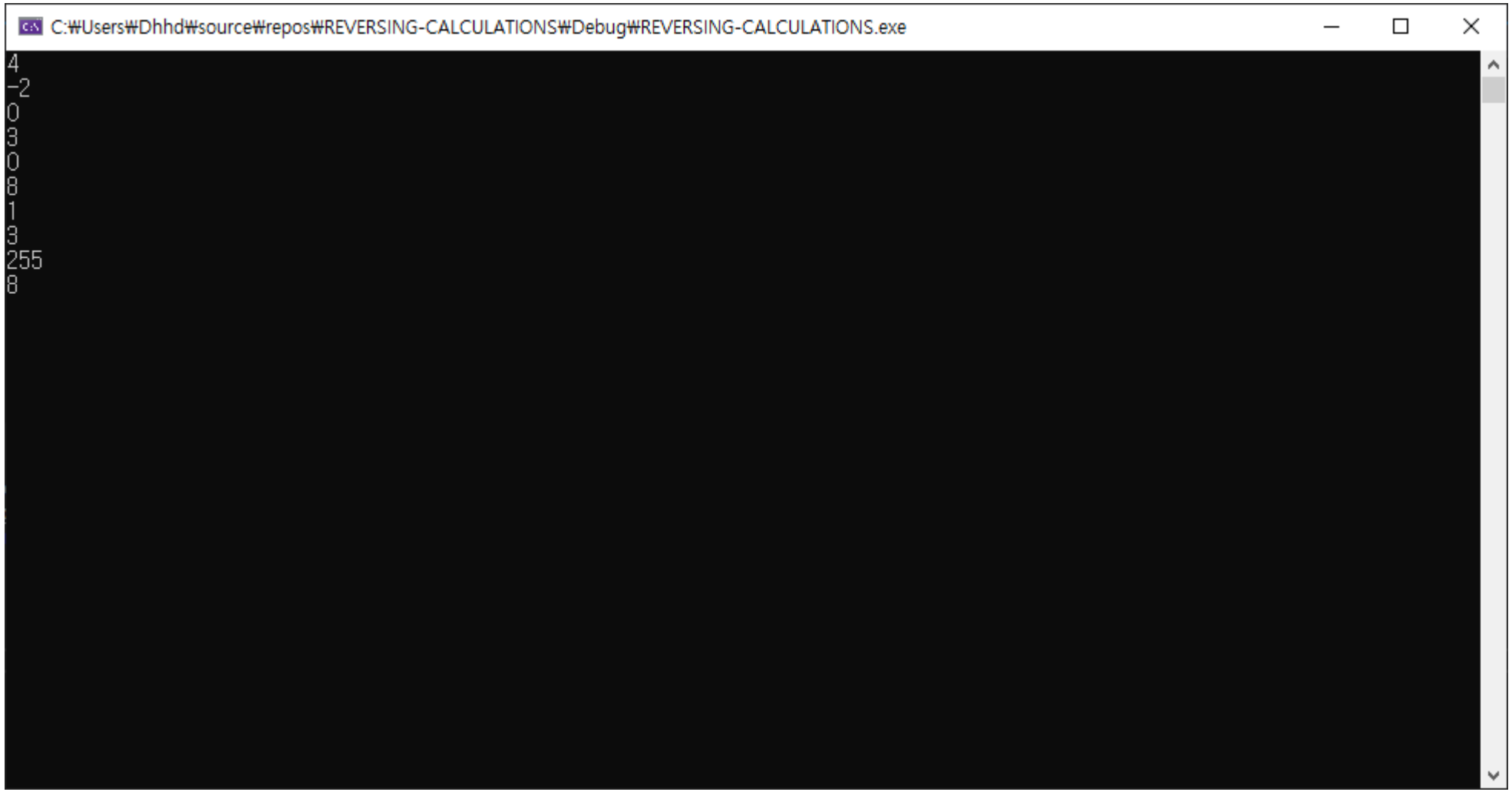
- 총정리 소스코드

```
1   #include <stdio.h>
2
3   int main()
4   {
5       int a = 1;
6       int b = 3;
7       printf("%d \n", a + b);
8       printf("%d \n", a - b);
9       printf("%d \n", a / b);
10      printf("%d \n", a * b);
11
12      printf("%d \n", a >> 2);
13      printf("%d \n", a << b);
14      printf("%d \n", a & b);
15      printf("%d \n", a ^ 2);
16      printf("%d \n", a | 0xFF);
17
18      __asm
19      {
20      mov         eax, dword ptr[a]
21      ror         al, 5
22      push        eax
23      }
24      printf("%d");
25  }
```

퀴즈 출력결과를 맞추시오

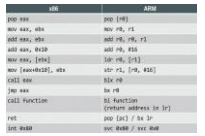


- 정답


C:₩Users₩Dhhd₩source₩repos₩REVERSING-CALCULATIONS₩Debug₩REVERSING-CALCULATIONS.exe
4
-2
0
3
0
8
1
3
255
8

if else
while
for
goto

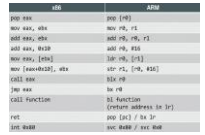
switch



- if else

```c
#include <stdio.h>

int main()
{
    bool a = true;
    int b = false;
    int zero = 0;
    int true1 = 1; //0빼고 나머지 다가능
    if (a) printf("a₩n");
    else if (b) printf("b₩n");
    if (zero) printf("zero₩n");
    if (true1)  printf("true1₩n");
    else printf("else₩n");
}
```

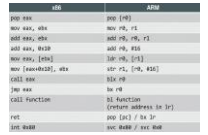
```
int main()
{
00631080   push          ebp
00631081   mov           ebp,esp
00631083   sub           esp,14h
    bool a = true;
00631086   mov           byte ptr [a],1
    int b = false;
0063108A   mov           dword ptr [b],0
    int zero = 0;
00631091   mov           dword ptr [zero],0
    int true1 = 1; //0빼고 나머지 다가능
00631098   mov           dword ptr [true1],1
    if (a) printf("a₩n");
0063109F   movzx         eax,byte ptr [a]
006310A3   test          eax,eax
006310A5   je            main+36h (06310B6h)
006310A7   push          632100h
006310AC   call          printf (0631040h)
006310B1   add           esp,4
006310B4   jmp           main+49h (06310C9h)
    else if (b) printf("b₩n");
006310B6   cmp           dword ptr [b],0
006310BA   je            main+49h (06310C9h)
006310BC   push          632104h
006310C1   call          printf (0631040h)
006310C6   add           esp,4
    if (zero) printf("zero₩n");
006310C9   cmp           dword ptr [zero],0
006310CD   je            main+5Ch (06310DCh)
006310CF   push          632108h
006310D4   call          printf (0631040h)
006310D9   add           esp,4
    if (true1)  printf("true1₩n");
006310DC   cmp           dword ptr [true1],0
006310E0   je            main+71h (06310F1h)
006310E2   push          632110h
006310E7   call          printf (0631040h)
006310EC   add           esp,4
006310EF   jmp           main+7Eh (06310FEh)
    else printf("else₩n");
006310F1   push          632118h
006310F6   call          printf (0631040h)
006310FB   add           esp,4
```



- switch

```
15      switch (a)
16      {
17      case true: printf("true");
18      case false: printf("false");
19                  break;
20      case 10: printf("10");
21      default: printf("default");
22      }
23  }
```

```
    switch (a)
00631OFE  mov         cl,byte ptr [a]
00631101  mov         byte ptr [ebp-8],cl
00631104  cmp         byte ptr [ebp-8],0
00631108  je          main+0A5h (0631125h)
0063110A  cmp         byte ptr [ebp-8],1
0063110E  je          main+98h (0631118h)
00631110  cmp         byte ptr [ebp-8],0Ah
00631114  je          main+0B4h (0631134h)
00631116  jmp         main+0C1h (0631141h)
    {
    case true: printf("true");
00631118  push        632120h
0063111D  call        printf (0631040h)
00631122  add         esp,4
    case false: printf("false");
00631125  push        632128h
0063112A  call        printf (0631040h)
0063112F  add         esp,4
                break;
00631132  jmp         main+0CEh (063114Eh)
    case 10: printf("10");
00631134  push        632130h
00631139  call        printf (0631040h)
0063113E  add         esp,4
    default: printf("default");
00631141  push        632134h
00631146  call        printf (0631040h)
0063114B  add         esp,4
    }
}
```

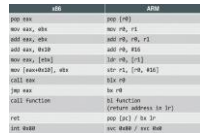


- while for

```
24      while (a)
25      {
26          a = false;
27          printf("a is not true / a != true");
28      }
```

```
30      for (zero = 0; zero != true; zero++)
31      {
32          printf("TRUE IS ONE / TRUE = 1");
33      }
```

```
    while (a)
00DB114E  movzx       edx,byte ptr [a]
00DB1152  test        edx,edx
00DB1154  je          main+0E9h (0DB1169h)
    {
        a = false;
00DB1156  mov         byte ptr [a],0
        printf("a is not true / a != true");
00DB115A  push        0DB213Ch
00DB115F  call        printf (0DB1040h)
00DB1164  add         esp,4
    }
00DB1167  jmp         main+0CEh (0DB114Eh)

    for (zero = 0; zero != true; zero++)
00E61169  mov         dword ptr [zero],0
00E61170  jmp         main+0FBh (0E6117Bh)
00E61172  mov         eax,dword ptr [zero]
00E61175  add         eax,1
00E61178  mov         dword ptr [zero],eax
00E6117B  cmp         dword ptr [zero],1
00E6117F  je          main+110h (0E61190h)
    {
        printf("TRUE IS ONE / TRUE = 1₩n");
00E61181  push        0E6215Ch
00E61186  call        printf (0E61040h)
00E6118B  add         esp,4
    }
00E6118E  jmp         main+0F2h (0E61172h)
```



• goto

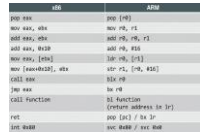
```
35      goto HERE;
36      printf("IT IS NOT WORK FUCTION₩n");
37  HERE:
```

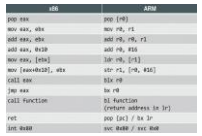
```
    goto HERE;
00E61190  jmp         HERE (0E611A1h)
00E61192  jmp         HERE (0E611A1h)
    printf("IT IS NOT WORK FUCTION₩n");
00E61194  push        0E62174h
00E61199  call        printf (0E61040h)
00E6119E  add         esp,4
HERE:
    return 0;
00E611A1  xor         eax,eax
}
00E611A3  mov         esp,ebp
00E611A5  pop         ebp
00E611A6  ret
```

- 소스코드

결과는?

```
1   #include <stdio.h>
2
3   int main()
4   {
5       bool a = true;
6       int b = false;
7       int zero = 0;
8       int true1 = 1; //0빼고 나머지 다가능
9       if (a) printf("a₩n");
10      else if (b) printf("b₩n");
11      if (zero) printf("zero₩n");
12      if (true1)  printf("true1₩n");
13      else printf("else₩n");
14
15      printf("₩n");
16
17      switch (a)
18      {
19      case true: printf("true₩n");
20      case false: printf("false₩n");
21          break;
22      case 10: printf("10₩n");
23      default: printf("default₩n");
24      }
25
26      printf("₩n");
27
28      while (a)
29      {
30          a = false;
31          printf("a is not true / a != true₩n");
32      }
33
34      printf("₩n");
35
36      for (zero = 0; zero != true; zero++)
37      {
38          printf("TRUE IS ONE / TRUE = 1₩n");
39      }
40
41      goto HERE;
42      printf("IT IS NOT WORK FUCTION₩n");
43  HERE:
44      return 0;
45  }
```





01

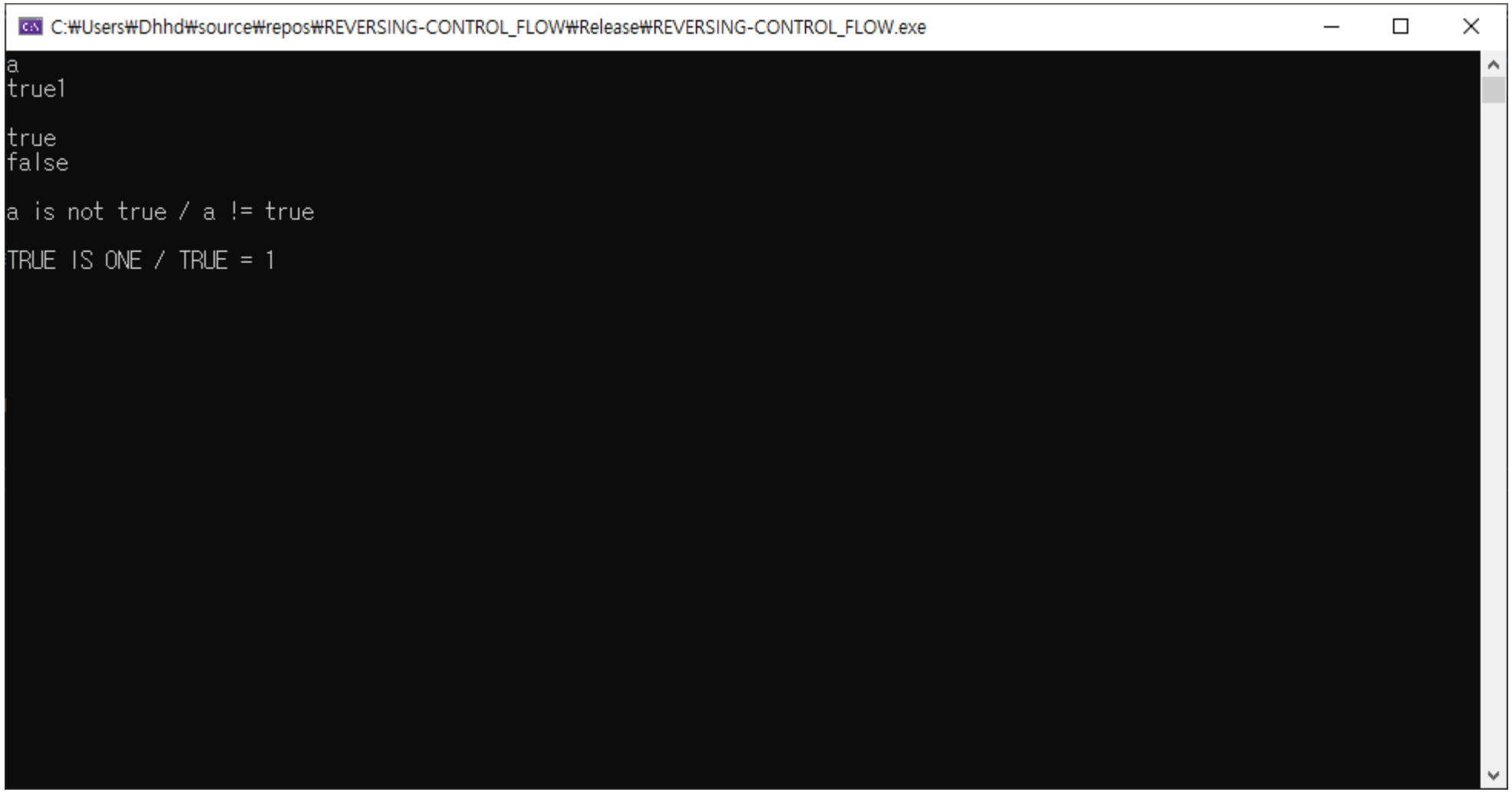
02

03

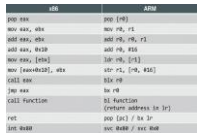
04

05

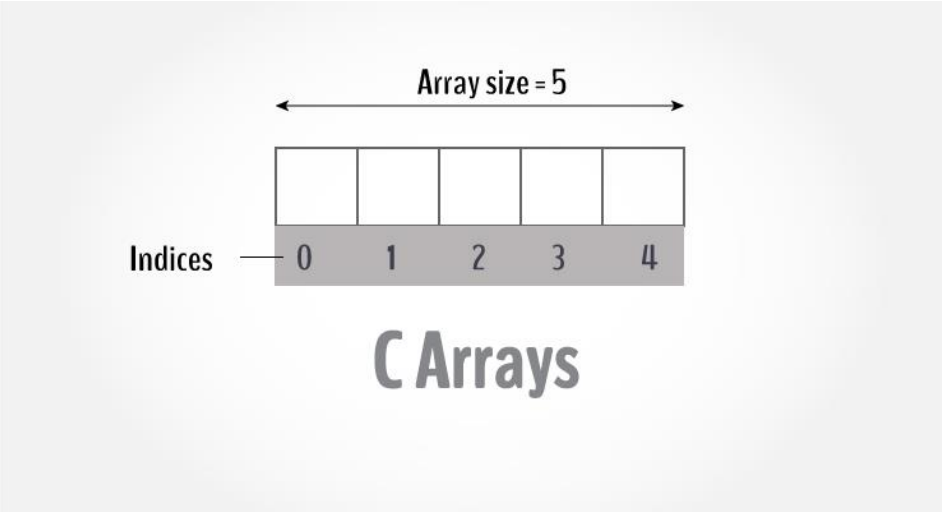
- 결과



배열
포인터

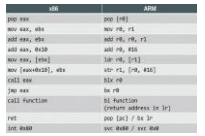


- 배열

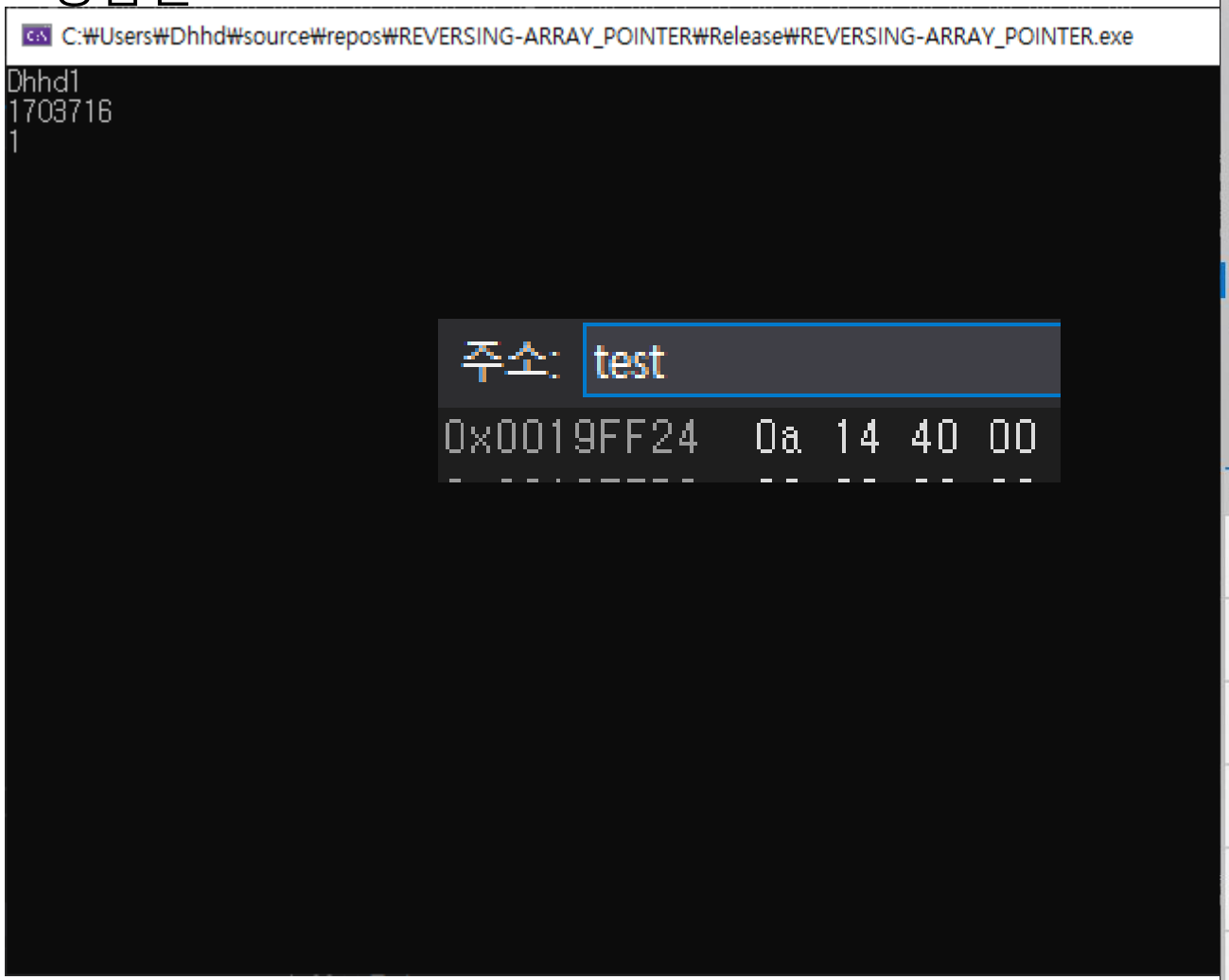


- 퀴즈 출력결과는

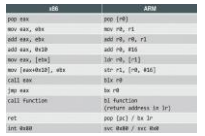
```c
#include <Stdio.h>
#include <Windows.h> // BYTE WORD DWORD

int main()
{
    char Dhhd[4] = "Dhhd";
    short array[2] =
    {
        0x31,
        0x32
    };
    BYTE test[2] =
    {
        10,
        20
    };

    printf("%s₩n", Dhhd);
    printf("%d₩n", test);
    printf("%s₩n", array);
}
```



• 정답은


C:₩Users₩Dhhd₩source₩repos₩REVERSING-ARRAY_POINTER₩Release₩REVERSING-ARRAY_POINTER.exe
Dhhd1
1703716
1
주소: test
0x0019FF24 0a 14 40 00

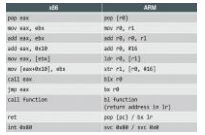

계산기
프로그래머
메모리
메모리에 저장된 항목 없음
1,703,716
HEX 19 FF24
DEC 1,703,716
OCT 6 377 444
BIN 0001 1001 1111 1111 0010 0100
DWORD
MS
비트
비트 시프트
CE
7 8 9
4 5 6
1 2 3
0



- 포인터

```
22      int* p;
23      p = &Dhhd;
24      puts(p);
```

```
Microsoft Visual Studio 디버그 콘솔
Dhhd1
15727732
1
Dhhd1

C:₩Users₩Dhhd₩source₩repos₩REVERSING-ARRAY_POINTER₩Release₩REVERSING-ARRAY_POINTER.exe(프로세스 9032개)이(가) 종료되었습니다(코드: -1개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

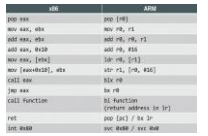


01
02
03
04
05

- 소스코드

```
#include <Stdio.h>
#include <Windows.h> // BYTE WORD DWORD

int main( )
{
00401080  push        ebp
00401081  mov         ebp,esp
00401083  sub         esp,10h
    char Dhhd[4] = "Dhhd";
00401086  mov         eax,dword ptr ds:[0040302Ch]
0040108B  mov         dword ptr [Dhhd],eax
    short array[2] =
0040108E  mov         ecx,31h
00401093  mov         word ptr [array],cx
00401097  mov         edx,32h
0040109C  mov         word ptr [ebp-6],dx
    {
        0x31,
        0x32
    };
    BYTE test[2] =
004010A0  mov         byte ptr [test],0Ah
004010A4  mov         byte ptr [ebp-3],14h
    {
        10,
        20
    };

    printf("%s₩n", Dhhd);
004010A8  lea         eax,[Dhhd]
004010AB  push        eax
004010AC  push        403028h
004010B1  call        printf (0401040h)
004010B6  add         esp,8
    printf("%d₩n", test);
004010B9  lea         ecx,[test]
004010BC  push        ecx
004010BD  push        403024h
004010C2  call        printf (0401040h)
004010C7  add         esp,8
    printf("%s₩n", array);
004010CA  lea         edx,[array]
004010CD  push        edx
004010CE  push        403020h
004010D3  call        printf (0401040h)
004010D8  add         esp,8
```

```
    int* p;
    p = &Dhhd;
004010DB  lea         eax,[Dhhd]
004010DE  mov         dword ptr [p],eax
    puts(p);
004010E1  mov         ecx,dword ptr [p]
004010E4  push        ecx
004010E5  call        dword ptr [__imp__puts (04020B8h)]
004010EB  add         esp,4
}
004010EE  xor         eax,eax
004010F0  mov         esp,ebp
004010F2  pop         ebp
004010F3  ret
```



01

- lea(load effective address)

02

lea eax,[주소] // eax = 주소

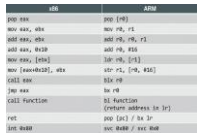
03

mov eax,[주소] // eax = 주소안에 들어있는값

04

05

구조체



01

02

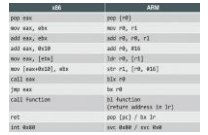
03

04

- 소스코드(최적화(O)/GS(O))

```
1   #include <Stdio.h>
2
3   int main()
4   {
5       typedef struct {
6           int age;
7           char name[20];
8       } person;
9
10      person p1;
11
12      strcpy(p1.name, "Dhhd");
13      p1.age = 17;
14
15      printf("이름: %s₩n", p1.name);
16      printf("나이: %d₩n", p1.age);
```

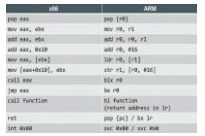
```
#include <Stdio.h>

int main()
{
00E31050  push        ebp
00E31051  mov         ebp,esp
00E31053  and         esp,0FFFFFFF8h
00E31056  sub         esp,1Ch
00E31059  mov         eax,dword ptr [__security_cookie (0E33004h)]
00E3105E  xor         eax,esp
00E31060  mov         dword ptr [esp+18h],eax
    typedef struct {
        int age;
        char name[20];
    } person;

    person p1;

    strcpy(p1.name, "Dhhd");
00E31064  mov         eax,dword ptr [string "Dhhd" (0E32108h)]
00E31069  mov         dword ptr [esp+4],eax
00E3106D  mov         al,byte ptr ds:[00E3210Ch]
00E31072  mov         byte ptr [esp+8],al
    p1.age = 17;

    printf("이름: %s₩n", p1.name);
00E31076  lea         eax,[esp+4]
00E3107A  push        esi
00E3107B  push        eax
00E3107C  push        offset string "₩xc0₩xcc₩xb8₩xa7: %s₩n" (0E32110h)
00E31081  mov         dword ptr [esp+0Ch],11h
00E31089  call        printf (0E31010h)
    printf("나이: %d₩n", p1.age);
00E3108E  push        dword ptr [esp+0Ch]
00E31092  push        offset string "₩xb3₩xaa₩xc0₩xcc: %d₩n" (0E3211Ch)
00E31097  call        printf (0E31010h)
```



- 소스코드2 (최적화(X)/GS(X))

```
18      person* p2 = malloc(sizeof(person));
19
20      strcpy(p2->name, "김인덕");
21      p2->age = 17;
22
23      printf("이름: %s₩n", p2->name);
24      printf("나이: %d₩n", p2->age);
25
26      free(p2);
27  }
```

```
    person* p2 = malloc(sizeof(person));
00E3109C  push        18h
00E3109E  call        dword ptr [__imp__malloc (0E3204Ch)]

    strcpy(p2->name, "김인덕");
00E310A4  mov         ecx,dword ptr [string "₩xb1₩xe8₩xc0₩xce₩xb4₩xf6" (0E32128h)]
00E310AA  mov         esi,eax
00E310AC  lea         edx,[esi+4]
00E310AF  add         esp,14h
00E310B2  mov         dword ptr [edx],ecx
00E310B4  mov         cx,word ptr ds:[0E3212Ch]
00E310BB  mov         word ptr [edx+4],cx
00E310BF  mov         cl,byte ptr ds:[0E3212Eh]
    p2->age = 17;

    printf("이름: %s₩n", p2->name);
00E310C5  push        edx
00E310C6  push        offset string "₩xc0₩xcc₩xb8₩xa7: %s₩n" (0E32110h)
00E310CB  mov         byte ptr [edx+6],cl
00E310CE  mov         dword ptr [esi],11h
00E310D4  call        printf (0E31010h)
    printf("나이: %d₩n", p2->age);
00E310D9  push        dword ptr [esi]
00E310DB  push        offset string "₩xb3₩xaa₩xc0₩xcc: %d₩n" (0E3211Ch)
00E310E0  call        printf (0E31010h)

    free(p2);
00E310E5  push        esi
00E310E6  call        dword ptr [__imp__free (0E32050h)]
}
00E310EC  mov         ecx,dword ptr [esp+30h]
00E310F0  add         esp,14h
00E310F3  xor         eax,eax
00E310F5  pop         esi
00E310F6  xor         ecx,esp
00E310F8  call        __security_check_cookie (0E31101h)
00E310FD  mov         esp,ebp
00E310FF  pop         ebp
00E31100  ret
```



01

02

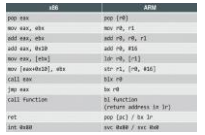
03

04

- 소스코드 (최적화(X)/GS(X))

```
1   #include <Stdio.h>
2
3   int main()
4   {
5       typedef struct {
6           int age;
7           char name[20];
8       } person;
9
10      person p1;
11
12      strcpy(p1.name, "Dhhd");
13      p1.age = 17;
14
15      printf("이름: %s₩n", p1.name);
16      printf("나이: %d₩n", p1.age);
```

```
#include <Stdio.h>

int main()
{
007C1080  push        ebp
007C1081  mov         ebp,esp
007C1083  sub         esp,1Ch
    typedef struct {
        int age;
        char name[20];
    } person;

    person p1;

    strcpy(p1.name, "Dhhd");
007C1086  mov         eax,dword ptr ds:[007C3058h]
007C108B  mov         dword ptr [ebp-18h],eax
007C108E  mov         cl,byte ptr ds:[7C305Ch]
007C1094  mov         byte ptr [ebp-14h],cl
    p1.age = 17;
007C1097  mov         dword ptr [p1],11h

    printf("이름: %s₩n", p1.name);
007C109E  lea         edx,[ebp-18h]
007C10A1  push        edx
007C10A2  push        7C304Ch
007C10A7  call        printf (07C1040h)
007C10AC  add         esp,8
    printf("나이: %d₩n", p1.age);
007C10AF  mov         eax,dword ptr [p1]
007C10B2  push        eax
007C10B3  push        7C3040h
007C10B8  call        printf (07C1040h)
007C10BD  add         esp,8
```



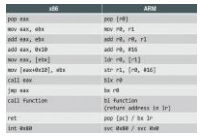
- 소스코드2 (최적화(X)/GS(X))

```
18      person* p2 = malloc(sizeof(person));
19
20      strcpy(p2->name, "김인덕");
21      p2->age = 17;
22
23      printf("이름: %s₩n", p2->name);
24      printf("나이: %d₩n", p2->age);
25
26      free(p2);
27  }
```

```
    person* p2 = malloc(sizeof(person));
007C10C0  push          18h
007C10C2  call          dword ptr [__imp__malloc (07C204Ch)]
007C10C8  add           esp,4
007C10CB  mov           dword ptr [p2],eax

    strcpy(p2->name, "김인덕");
007C10CE  mov           ecx,dword ptr [p2]
007C10D1  add           ecx,4
007C10D4  mov           edx,dword ptr ds:[7C3038h]
007C10DA  mov           dword ptr [ecx],edx
007C10DC  mov           ax,word ptr ds:[007C303Ch]
007C10E2  mov           word ptr [ecx+4],ax
007C10E6  mov           dl,byte ptr ds:[7C303Eh]
007C10EC  mov           byte ptr [ecx+6],dl
    p2->age = 17;
007C10EF  mov           eax,dword ptr [p2]
007C10F2  mov           dword ptr [eax],11h

    printf("이름: %s₩n", p2->name);
007C10F8  mov           ecx,dword ptr [p2]
007C10FB  add           ecx,4
007C10FE  push          ecx
007C10FF  push          7C302Ch
007C1104  call          printf (07C1040h)
007C1109  add           esp,8
    printf("나이: %d₩n", p2->age);
007C110C  mov           edx,dword ptr [p2]
007C110F  mov           eax,dword ptr [edx]
007C1111  push          eax
007C1112  push          7C3020h
007C1117  call          printf (07C1040h)
007C111C  add           esp,8

    free(p2);
007C111F  mov           ecx,dword ptr [p2]
007C1122  push          ecx
007C1123  call          dword ptr [__imp__free (07C2050h)]
007C1129  add           esp,4
}
007C112C  xor           eax,eax
007C112E  mov           esp,ebp
007C1130  pop           ebp
007C1131  ret
```

변수종류

전역
지역
정적
외부
레지스터



01

02

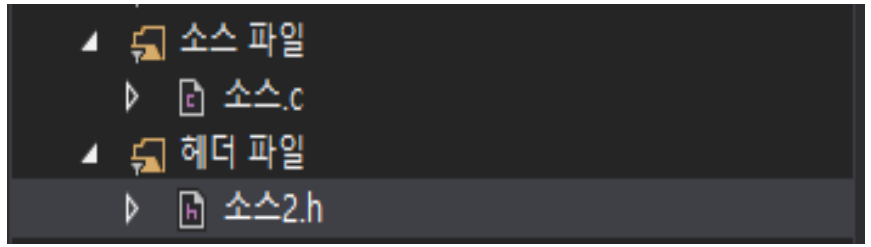
03

04

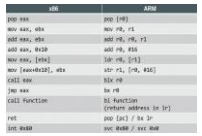
05

- 소스코드

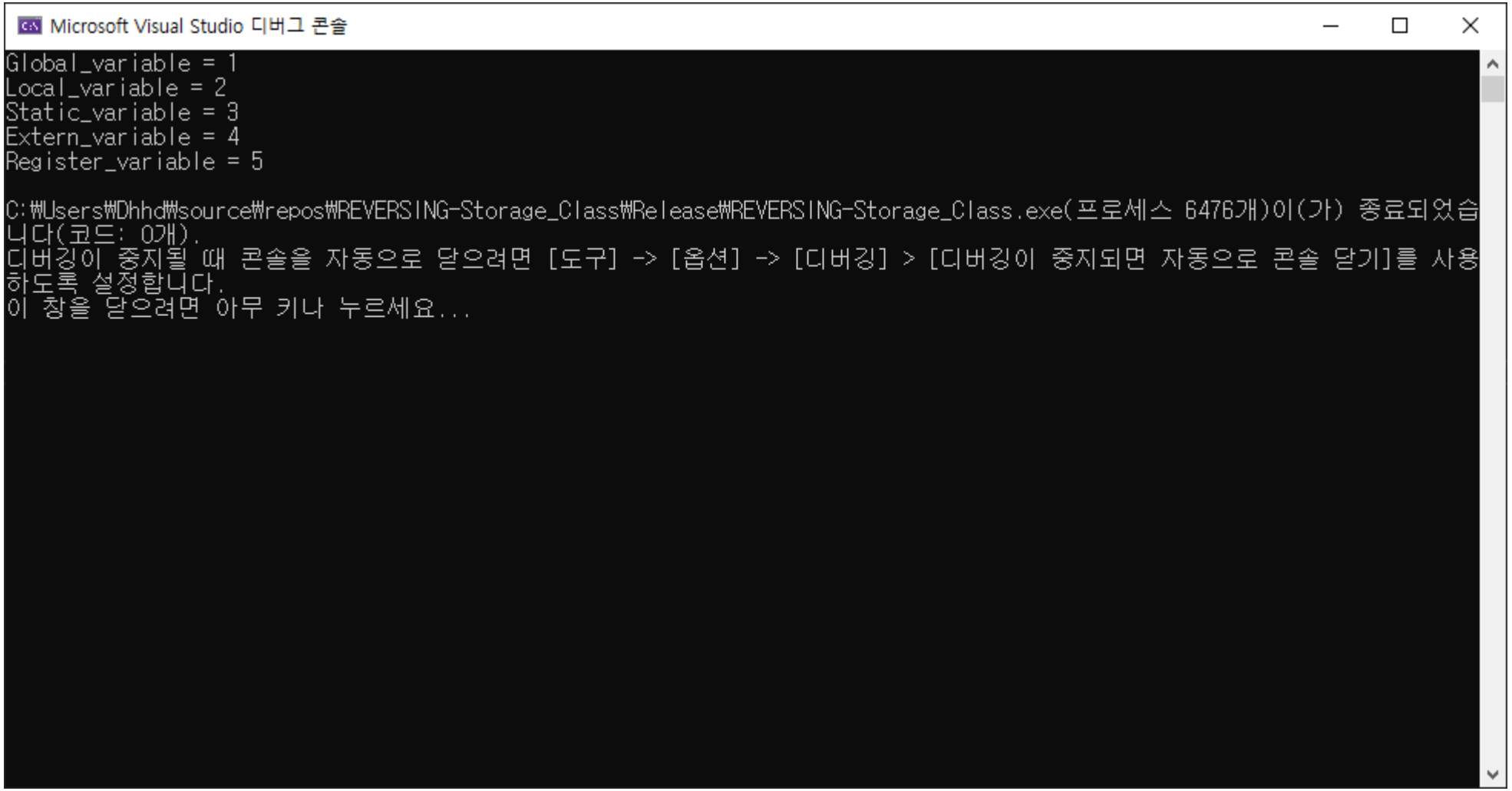
```c
#include <stdio.h>
#include "소스2.h"

int Global_variable = 1; //전역변수 main위에있다

int Extern_variable; //외부변수
int main()
{
    auto int Local_variable = 2; // 뒤에것랑같다 int count; (기본)(지역변수)

    static int Static_variable = 3;//정적 변수


    register int Register_variable = 5; //레지스터 변수

    printf("Global_variable = %d₩n", Global_variable);
    printf("Local_variable = %d₩n", Local_variable);
    printf("Static_variable = %d₩n", Static_variable);
    printf("Extern_variable = %d₩n", Extern_variable);
    printf("Register_variable = %d₩n", Register_variable);
}
```

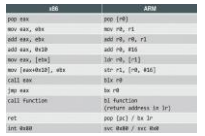



```c
extern int Extern_variable = 4;
```



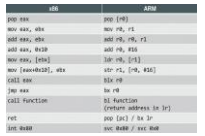
- 출력결과


Microsoft Visual Studio 디버그 콘솔
Global_variable = 1
Local_variable = 2
Static_variable = 3
Extern_variable = 4
Register_variable = 5
C:₩Users₩Dhhd₩source₩repos₩REVERSING-Storage_Class₩Release₩REVERSING-Storage_Class.exe(프로세스 6476개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...



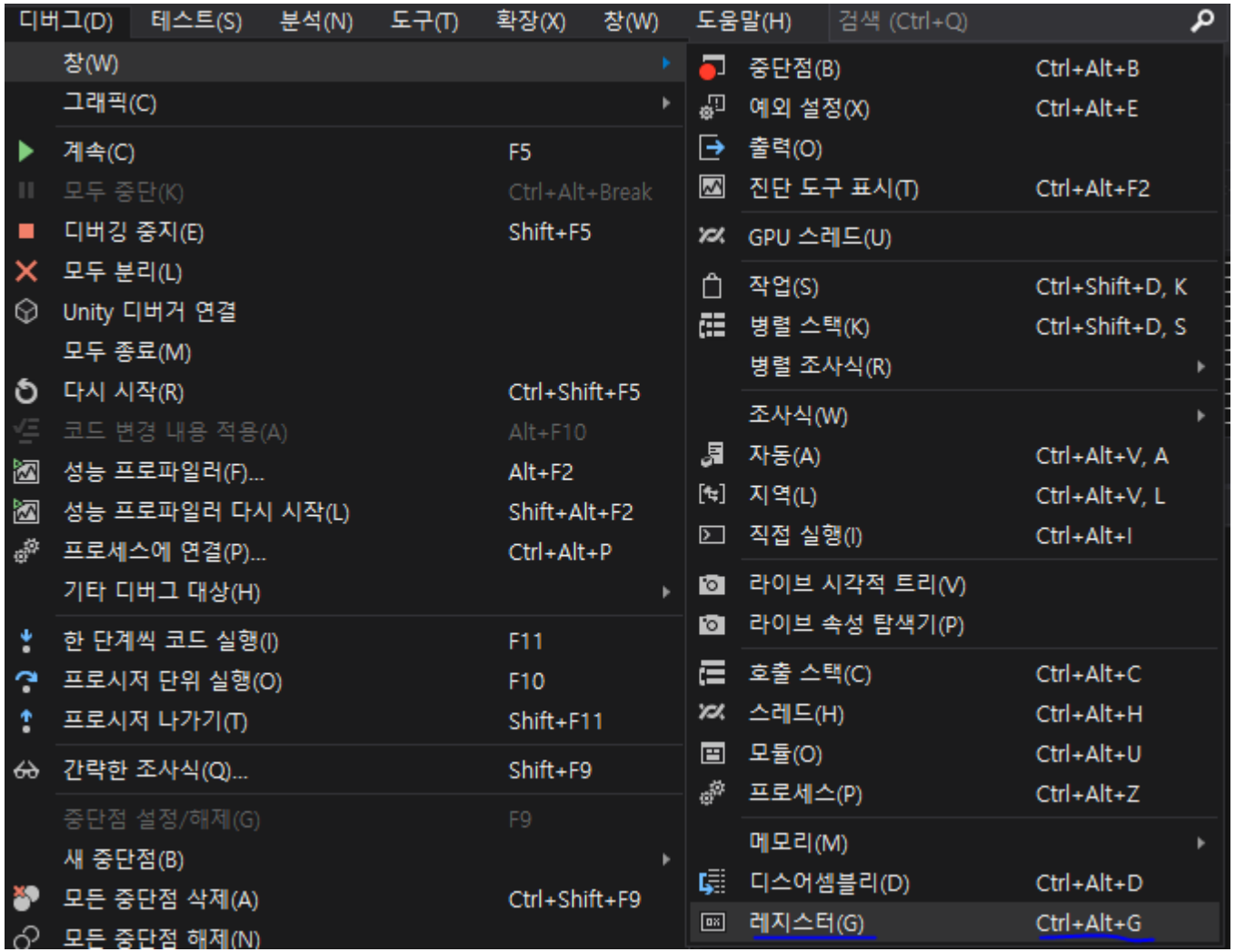
- 디스어셈블리

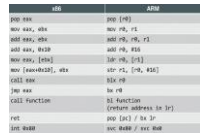
```
#include <stdio.h>
#include "소스2.h"

int Global_variable = 1; //전역변수 main위에있다

int Extern_variable; //외부변수
int main()
{
00401080  push        ebp
00401081  mov         ebp,esp
00401083  sub         esp,8
    auto int Local_variable = 2; // 뒤에것랑같다 int count; (기본)(지역변수)
00401086  mov         dword ptr [Local_variable],2

    static int Static_variable = 3;//정적 변수


    register int Register_variable = 5; //레지스터 변수
0040108D  mov         dword ptr [Register_variable],5

    printf("Global_variable = %d₩n", Global_variable);
00401094  mov         eax,dword ptr [Global_variable (0403018h)]
00401099  push        eax
0040109A  push        403084h
0040109F  call        printf (0401040h)
004010A4  add         esp,8
    printf("Local_variable = %d₩n", Local_variable);
004010A7  mov         ecx,dword ptr [Local_variable]
004010AA  push        ecx
004010AB  push        40306Ch
004010B0  call        printf (0401040h)
004010B5  add         esp,8
    printf("Static_variable = %d₩n", Static_variable);
004010B8  mov         edx,dword ptr ds:[403020h]
004010BE  push        edx
004010BF  push        403054h
004010C4  call        printf (0401040h)
004010C9  add         esp,8
    printf("Extern_variable = %d₩n", Extern_variable);
004010CC  mov         eax,dword ptr [Extern_variable (040301Ch)]
004010D1  push        eax
004010D2  push        40303Ch
004010D7  call        printf (0401040h)
004010DC  add         esp,8
    printf("Register_variable = %d₩n", Register_variable);
004010DF  mov         ecx,dword ptr [Register_variable]
004010E2  push        ecx
004010E3  push        403024h
004010E8  call        printf (0401040h)
004010ED  add         esp,8
}
004010F0  xor         eax,eax
004010F2  mov         esp,ebp
004010F4  pop         ebp
004010F5  ret
```



- 메모리
- 레지스터 Ctrl+Alt+G

```
주소: &Global_variable
0x00403018   01 00 00 00
주소: &Local_variable
0x0019FF24   02 00 00 00
주소: &Static_variable
0x00403020   03 00 00 00
주소: &Extern_variable
0x0040301C   04 00 00 00
주소: &Register_variable
0x0019FF20   05 00 00 00
```





01

02

03

04

05

- 레지스터

```
레지스터
EAX = 75B212F0 EBX = 003FD000 ECX = 00000000 EDX = 00000000 ESI = 008352C8 EDI = 00835AA0 EIP = 004010A4 ESP = 0019FF20 EBP = 0019FF28 EFL = 00000202
```