

Temporary Object

# Assembly code

lea(load effective address)

---

```
__asm
{
    lea eax, [num1]
    mov ecx, dword ptr[eax]
    mov dword ptr[buf], ecx
}
```

num1의 address를 eax에 저장

# Assembly code

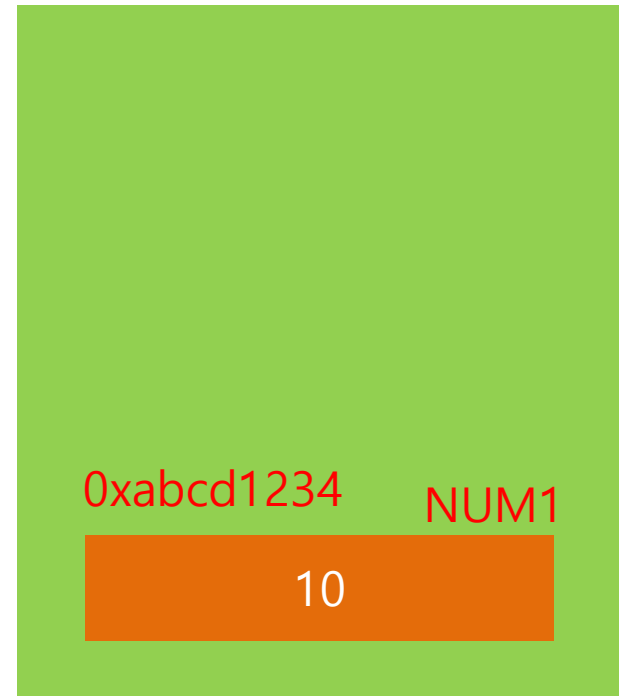
lea(load effective address)

---

```
__asm  
{  
    lea eax, [num1]  
    mov ecx, dword ptr[eax]  
    mov dword ptr[buf], ecx  
}
```

NUM1의 주소값을 저장

STACK FRAME



# Assembly code

lea(load effective address)

```
__asm
```

```
{
```

```
    lea eax, [num1]
```

```
    mov ecx, dword ptr[eax]
```

```
    mov dword ptr[buf], ecx
```

```
}
```

EAX값을 주소로 해당 주소의 값을  
ECX에 복사

STACK FRAME

REG

EAX

abcd1234

ECX

10

0xabcd1234

NUM1

10

# 임시 객체

Point 클래스

---

```
class Point
{
public:
    Point(int x = 0, int y = 0)
    {
        this->x = x;
        this->y = y;
    }
private:
    int x, y;
```

# 임시 객체

func() 함수

---

```
Point func()  
{  
    Point p1(1, 2);  
    //임시객체도 스택에 할당된다.  
    Point(3, 4); 임시 객체  
  
    int num = 10;  
  
    //리턴 시 임시 객체도 스택에  
    //eax에는 임시 객체의 address가 저장됨.  
    return p1; 리턴 시 임시 객체 생성  
}
```

# 임시 객체

Assembly code

---

```
28:      //eax에는 임시 객체의 address가 저장됨.  
29:      return p1;  
010E1900  mov     eax,dword ptr [ebp+8]    ≤ 1ms elapsed  
010E1903  mov     ecx,dword ptr [p1]  
010E1906  mov     dword ptr [eax],ecx  
010E1908  mov     edx,dword ptr [ebp-8]  
010E190B  mov     dword ptr [eax+4],edx  
010E190E  mov     eax,dword ptr [ebp+8]  
30:  }
```

임시 객체를 저장할 주소를 **eax**에 저장

# 임시 객체

Assembly code

```
28:      //eax에는 임시 객체의 address가 저장됨.  
29:      return p1;  
010E1900  mov     eax,dword ptr [ebp+8]      ≤1ms elapsed  
010E1903  mov     ecx,dword ptr [p1]  
010E1906  mov     dword ptr [eax],ecx  
010E1908  mov     edx,dword ptr [ebp-8]  
010E190B  mov     dword ptr [eax+4],edx  
010E190E  mov     eax,dword ptr [ebp+8]  
30:  }
```

p1의 x 값을 ecx에 복사



# 임시 객체

Assembly code

```
28:      //eax에는 임시 객체의 address가 저장됨.  
29:      return p1;  
010E1900  mov     eax,dword ptr [ebp+8]    ≤1ms elapsed  
010E1903  mov     ecx,dword ptr [p1]  
010E1906  mov     dword ptr [eax],ecx  
010E1908  mov     edx,dword ptr [ebp-8]  
010E190B  mov     dword ptr [eax+4],edx  
010E190E  mov     eax,dword ptr [ebp+8]  
30:  }
```

ecx의 값(p1의 x)을  
eax가 가리키는 주소(임시 객체의 x)에 복사

# 임시 객체

Assembly code

```
28:      //eax에는 임시 객체의 address가 저장됨.  
29:      return p1;  
010E1900  mov     eax,dword ptr [ebp+8]      ≤1ms elapsed  
010E1903  mov     ecx,dword ptr [p1]  
010E1906  mov     dword ptr [eax],ecx  
010E1908  mov     edx,dword ptr [ebp-8]  
010E190B  mov     dword ptr [eax+4],edx  
010E190E  mov     eax,dword ptr [ebp+8]  
30:  }
```

p1의 y 값을 edx에 복사

# 임시 객체

Assembly code

```
28:      //eax에는 임시 객체의 address가 저장됨.  
29:      return p1;  
010E1900  mov     eax,dword ptr [ebp+8]      ≤1ms elapsed  
010E1903  mov     ecx,dword ptr [p1]  
010E1906  mov     dword ptr [eax],ecx  
010E1908  mov     edx,dword ptr [ebp-8]  
010E190B  mov     dword ptr [eax+4],edx  
010E190E  mov     eax,dword ptr [ebp+8]  
30:  }
```

edx의 값(p1의 y)을  
eax+4가 가리키는 주소(임시 객체의 y)에 복사

# RVO

## Return Value Optimization

---

```
#if RVO  
    Point p1 = func();  
#endif
```

func()은 임시 객체를 만들어 반환하고  
p1이란 객체를 만든 후

임시 객체의 값을 p1 객체에 대입한 후  
임시 객체는 소멸할 것 같지만.....

# RVO

## Return Value Optimization

---

```
#if RVO  
    Point p1 = func();  
#endif
```

RVO, 리턴 값 최적화를 통해  
임시 객체를 생성하지 않고  
스택에 이미 만들어져 있는 p1 객체에  
값을 바로 대입한다.  
(리턴 시 임시 객체를 생성하지 않음)

# RVO

## Return Value Optimization

```
29:      return p1;
013F1900  mov     eax,dword ptr [ebp+8]
013F1903  mov     ecx,dword ptr [p1]    ; 1ms elapsed
013F1906  mov     dword ptr [eax],ecx
013F1908  mov     edx,dword ptr [ebp+8]
```

임시 객체 반환 주소가

Image Watch    Memory 1

Address: 0x0117FD74	Columns:
0x0117FD74	cc cc cc cc cc cc cc cc cc cc cc 94 fd 17 01 8e 20
0x0117FD86	3f 01 01 00 00 00 08 8c 4b 01 20 d7 4b 01 ec fd 17 01
0x0117FD98	f0 1e 3f 01 fe 6d c2 0e 46 10 3f 01 46 10 3f 01 00 f0
0x0117FDAA	e1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

p1 객체  
(함수의 리턴 값을 받는 객체)  
의 주소 값과

Memory 2    Registers

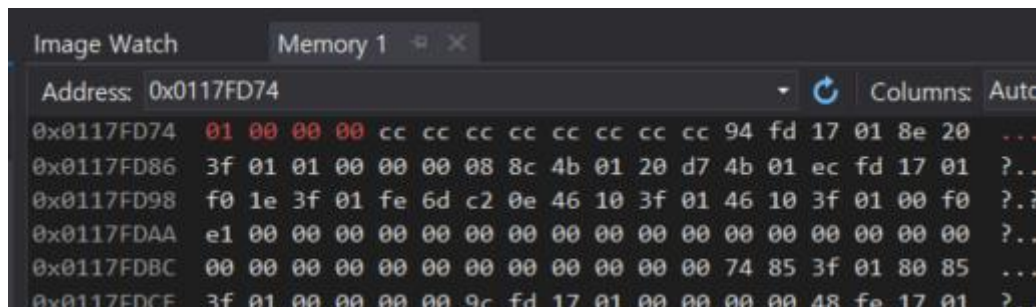
EAX = 0117FD74	EBX = 00E1F000	ECX = 00000004	EDX = 013F8580
ESI = 013F1046	EDI = 0117FC98	EIP = 013F1903	ESP = 0117FBA0
EBP = 0117FC98	EFL = 00000210		

0x0117fc8c = 00000001

같다

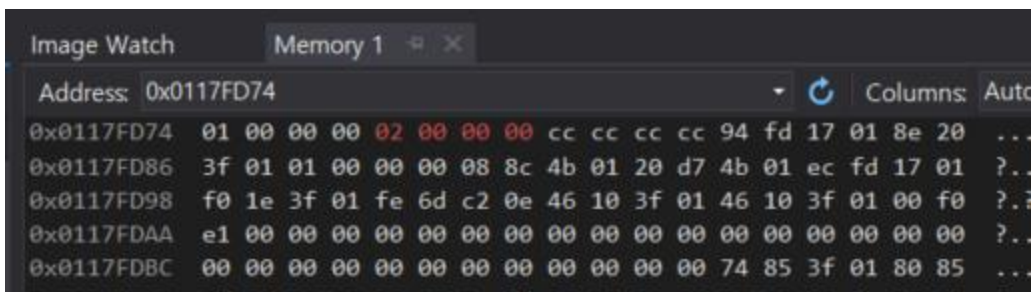
# RVO

## Return Value Optimization



Address	0x0117FD74	Columns	Auto
0x0117FD74	01 00 00 00 cc cc cc cc cc cc cc cc 94 fd 17 01 8e 20		...
0x0117FD86	3f 01 01 00 00 00 08 8c 4b 01 20 d7 4b 01 ec fd 17 01		?
0x0117FD98	f0 1e 3f 01 fe 6d c2 0e 46 10 3f 01 46 10 3f 01 00 f0		?
0x0117FDAA	e1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		?
0x0117FDBC	00 00 00 00 00 00 00 00 00 00 00 00 74 85 3f 01 80 85		...
0x0117FDCF	3f 01 00 00 00 00 9c fd 17 01 00 00 00 00 48 fe 17 01		?

x 값 복사



Address	0x0117FD74	Columns	Auto
0x0117FD74	01 00 00 00 02 00 00 00 cc cc cc cc 94 fd 17 01 8e 20		...
0x0117FD86	3f 01 01 00 00 00 08 8c 4b 01 20 d7 4b 01 ec fd 17 01		?
0x0117FD98	f0 1e 3f 01 fe 6d c2 0e 46 10 3f 01 46 10 3f 01 00 f0		?
0x0117FDAA	e1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		?
0x0117FDBC	00 00 00 00 00 00 00 00 00 00 00 00 74 85 3f 01 80 85		...

y 값 복사

컴파일러가 성능의 최적화를 위해  
임시 객체를 만들지 않고 p1 객체를 생성하면서 바로 대입을 진행한다.  
이를 RVO라고 한다.