Visual Studio를 이용한 어셈블리어 학습 part 2

유영천

https://megayuchi.com

tw: @dgtman

X64 어셈블리어 사용

주요 변경점

- 각 레지스터의 32Bits -> 64Bits 확장
- 스택 기본 단위 64Bits(8Bytes)
- 64Bits 주소 지정
- R8 – R15 레지스터 추가
- XMM8 – XMM15 레지스터 추가
- 기본 Calling convention – fastcall(x86와 x64의 fastcall은 다름)

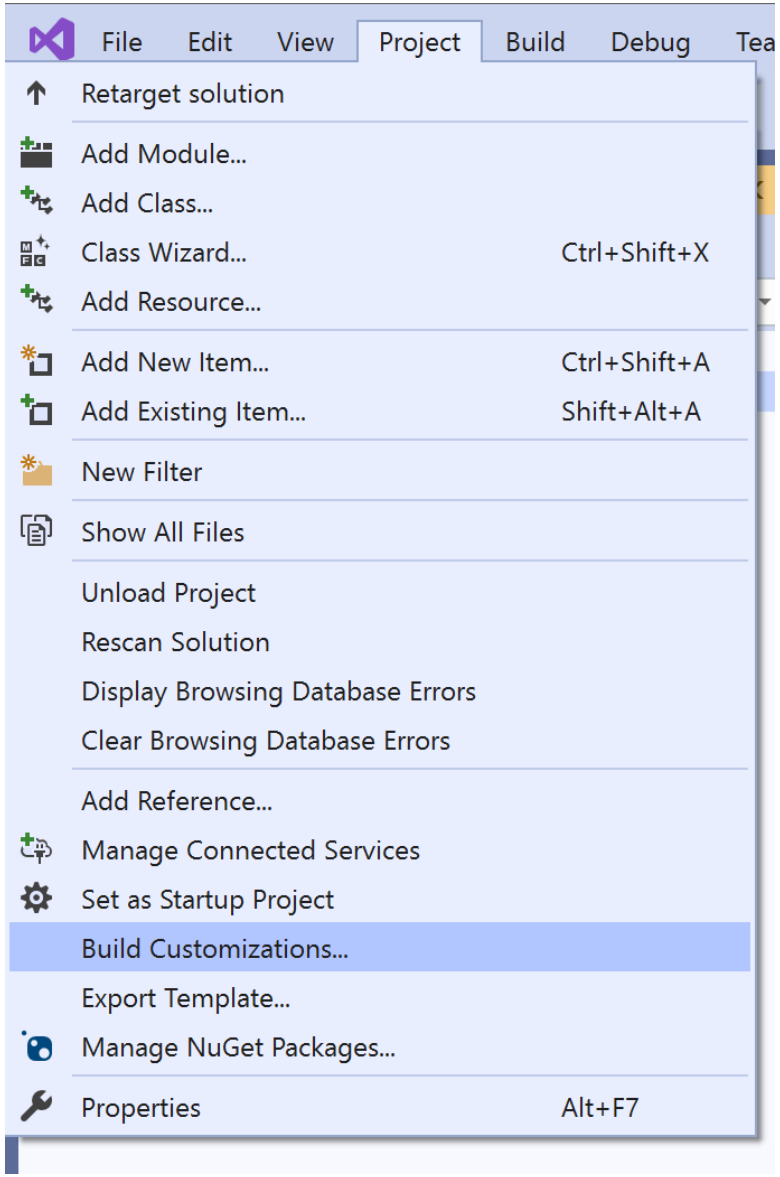
Visual Studio 에서 x64어셈블리어 사용하기

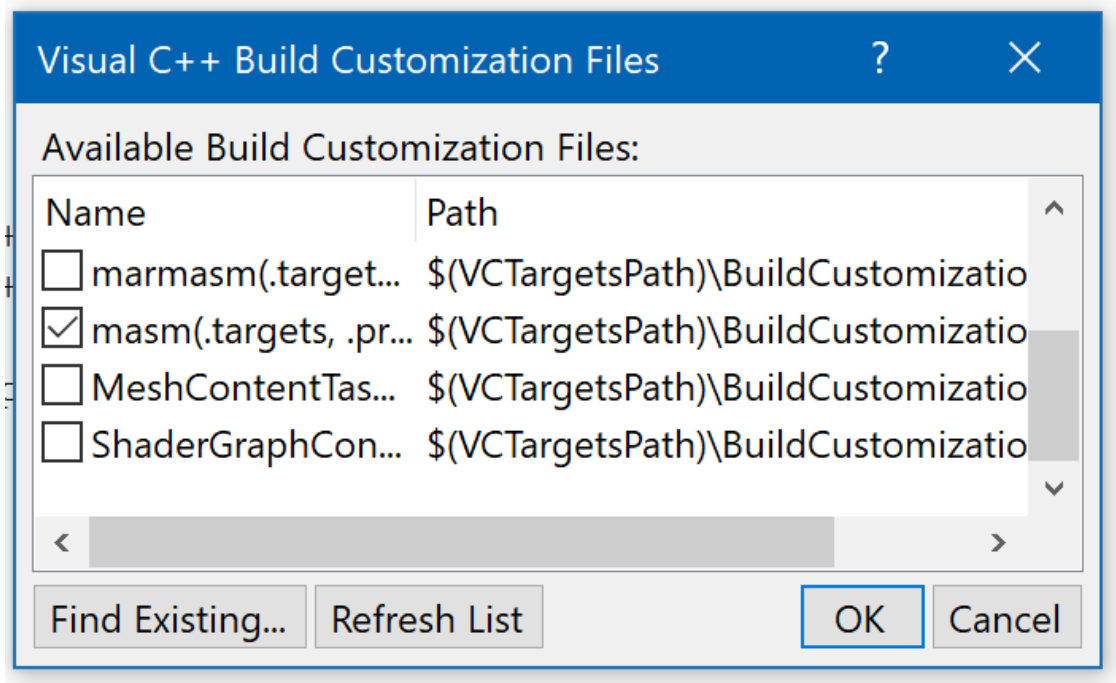
- 슬프게도 x64모드로는 inline 어셈블리어가 지원되지 않는다.
- MASM의 64bits 버전인 ml64.exe를 혼용한다.
- 약간의 수고만 들이면 쉽게 Visual Studio IDE에 통합된다. -> 디버깅 가능!!!

Visual Studio 에서 x64어셈블리어 사용하기

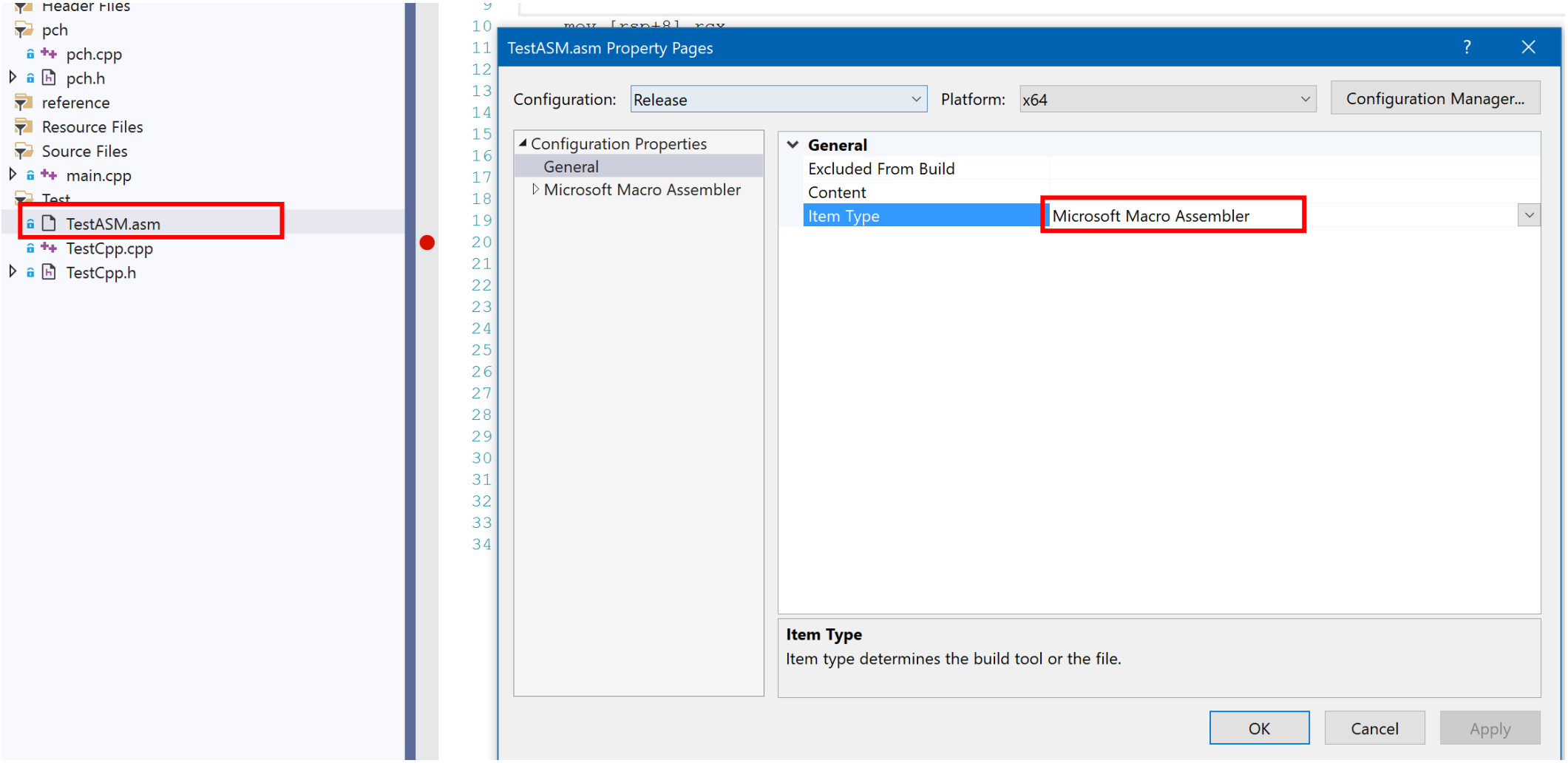
- Project -> Build Customization -> masm항목에 체크
- .asm파일을 Drag & Drop, 또는 new 로 .asm파일 추가.
- .asm파일의 property -> Item Type -> Microsoft Macro Assembler
- 필요한 경우 listing file 설정

Project -> Build Customization -> masm항목에 체크





.asm파일을 Drag & Drop, 또는 new 로 .asm파일 추가.
.asm파일의 property -> Item Type -> Microsoft Macro Assembler

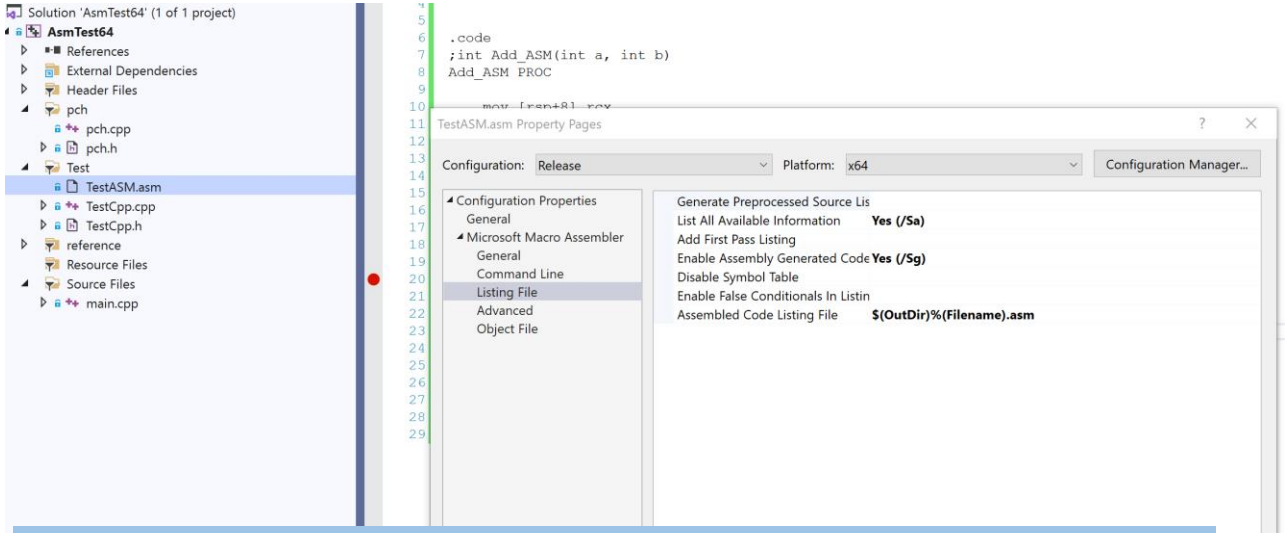


Listing File

TestASM.asm | fscanf_stub.h | fscanf_stub_x64.inc | fscanf_stub_x64.asm | Object Browser | Source Control Explorer

```
00000000  48/ 89 4C 24          mov [rsp+8],rcx
       08
00000005  48/ 89 54 24          mov [rsp+16],rdx
       10
0000000A  48/ 03 CA             add rcx,rdx
0000000D  48/ 8B C1             mov rax,rcx
00000010  C3                ret
00000011            Add_ASM ENDP

                ; int Add_ASM_RBP(int a, int b)
00000011            Add_ASM_RBP PROC    a:QWORD, b:QWORD
                    LOCAL c:QWORD
00000011  55            *       push    rbp
00000012  48/ 8B EC         *       mov     rbp, rsp
00000015  48/ 83 C4 F8      *       add     rsp, 0FFFFFFF8h
00000019  48/ 89 4D 10          mov a,rcx   ;mov [rbp+16],rcx
0000001D  48/ 89 4D 18          mov b,rcx   ;mov [rbp+24],rdx
00000021  48/ 03 CA             add rcx,rdx
00000024  48/ 89 4D F8          mov c, rcx  ; mov [rbp-8],rcx
00000028  48/ 8B 45 F8          mov rax,c
                    ret
0000002C  C9            *       leave
0000002D  C3            *       ret     00000h
0000002E            Add_ASM_RBP ENDP
                end

Microsoft (R) Macro Assembler (x64) Version 14.29.30037.0   06/13/21 11:53:39
TestASM.asm                         Symbols 2 - 1



Procedures, parameters, and locals:

                N a m e                 Type     Value    Attr

Add_ASM_RBP  . . . . . . . . . .    P    00000011 _TEXT Length= 0000001D Public
  a  . . . . . . . . . . . . . .    QWord    rbp + 00000010
  b  . . . . . . . . . . . . . .    QWord    rbp + 00000018
  c  . . . . . . . . . . . . . .    QWord    rbp - 00000008
Add_ASM  . . . . . . . . . . . .    P    00000000 _TEXT Length= 00000011 Public
```

- 완전한 실제 asm코드로 보여줌.
- 가상화(?)된 변수들의 실제 주소지정 코드를 확인 할 수 있다.
- Property-> Macro Assembler -> Listing File



- List All Available Information : Yes
- Enable Assembly Generated Code Listing : Yes
- Assembled Code Listing File : $(OutDir)%(Filename).asm

Visual Studio에서의 x64어셈블리어 코딩

- 함수 단위로 작성.
- Disassembly Window와 .asm코드 에디터 양쪽 모두에서 브레이크 포인트를 찍어가며 디버깅 가능.
- Cpp코드에서의 호출을 위해 .asm의 함수를 선언할때는 extern "C" 로 선언할것.
- x86/x64/ARM64를 함께 지원하는 프로젝트일 경우
  - 각 아키텍처별 코드를 각각의 .asm또는 .cpp로 작성.
  - 아키텍처별로 다른 아키텍처의 .asm/.cpp 파일의 property에서 excluded from build : yes로 설정

.asm파일 작성

```
.CODE

Add_ASM PROC

mov [rsp+8],rcx
mov [rsp+16],rdx
add rcx,rdx
mov rax,rcx
ret

Add_ASM ENDP

END
```

Add_ASM()함수의 시작

Add_ASM()함수의 끝

.asm의 함수를 호출하기 위한 선언

```
extern "C"
{
    INT64 Add_ASM(INT64 a, INT64 b);
    INT64 Add_ASM_RBP(INT64 a, INT64 b);
}
```

masm으로 생성된 함수명을 C++코드에서 호출하기 위해서는 해당 함수명이 C타입의 Name Decoration을 사용해야한다.

x64 레지스터와 스택

Registers

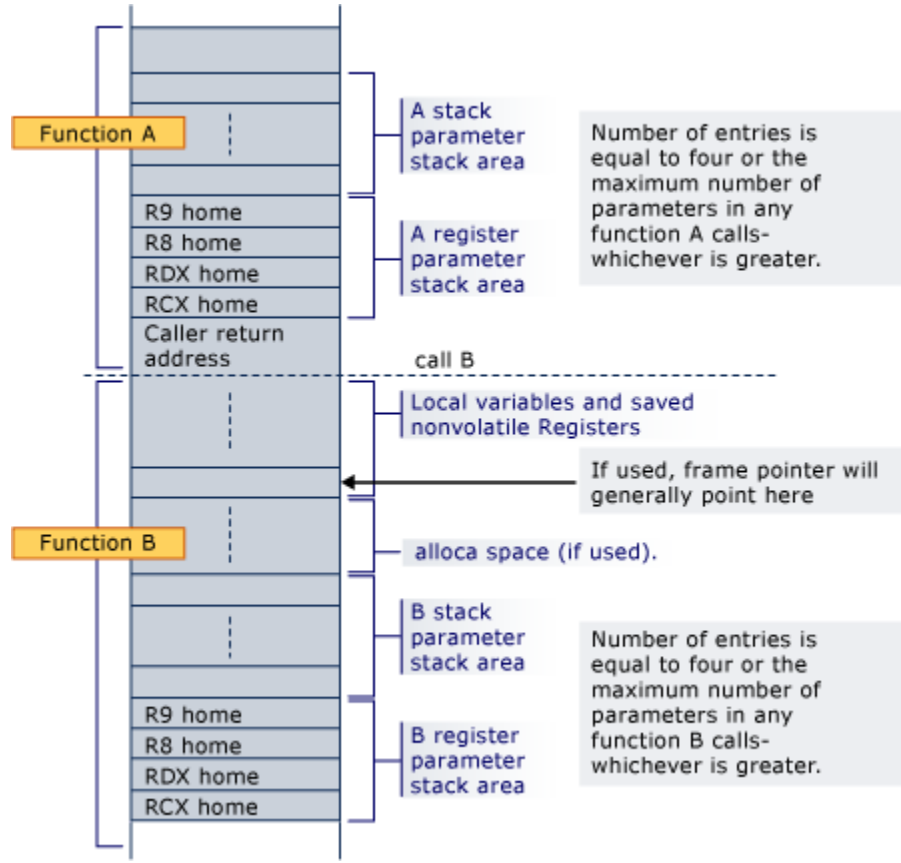
Register	Status	Use
RAX	Volatile	Return value register
RCX	Volatile	First integer argument
RDX	Volatile	Second integer argument
R8	Volatile	Third integer argument
R9	Volatile	Fourth integer argument
R10:R11	Volatile	Must be preserved as needed by caller; used in syscall/sysret instructions
R12:R15	Nonvolatile	Must be preserved by callee
RDI	Nonvolatile	Must be preserved by callee
RSI	Nonvolatile	Must be preserved by callee
RBX	Nonvolatile	Must be preserved by callee
RBP	Nonvolatile	May be used as a frame pointer; must be preserved by callee
RSP	Nonvolatile	Stack pointer
XMM0, YMM0	Volatile	First FP argument; first vector-type argument when **__vectorcall** is used
XMM1, YMM1	Volatile	Second FP argument; second vector-type argument when **__vectorcall** is used
XMM2, YMM2	Volatile	Third FP argument; third vector-type argument when **__vectorcall** is used
XMM3, YMM3	Volatile	Fourth FP argument; fourth vector-type argument when **__vectorcall** is used
XMM4, YMM4	Volatile	Must be preserved as needed by caller; fifth vector-type argument when **__vectorcall** is used
XMM5, YMM5	Volatile	Must be preserved as needed by caller; sixth vector-type argument when **__vectorcall** is used
XMM6:XMM15, YMM6:YMM15	Nonvolatile (XMM), Volatile (upper half of YMM)	Must be preserved by callee. YMM registers must be preserved as needed by caller.

파라미터 전달

Parameter type	fifth and higher	fourth	third	second	leftmost
floating-point	stack	XMM3	XMM2	XMM1	XMM0
integer	stack	R9	R8	RDX	RCX
Aggregates (8, 16, 32, or 64 bits) and **__m64**	stack	R9	R8	RDX	RCX
Other aggregates, as pointers	stack	R9	R8	RDX	RCX
__m128, as a pointer	stack	R9	R8	RDX	RCX

스택 프레임

- 베이스 포인터 레지스터(RBP) 사용이 calling convention의 일부가 아님.
- VC++의 디버그 빌드(C/C++), ml64에서 local지시어를 이용한 변수명 사용시 베이스 포인터 레지스터(RBP) 사용.
- 다른 함수를 호출할 경우 stack pointer(RSP레지스터의 값)주소가 16 Bytes align 되도록 할것!!!!
  - 함수 호출 전 rsp레지스터의 값을 16으로 나눠서 나머지가 0인지 확인
  - 직접 작성한 asm함수 안에서 로컬변수 할당 사이즈는 16 bytes의 배수로 맞춘다.



[x64 stack usage | Microsoft Docs]

함수호출

- Caller
  - 파라미터를 push하는 대신 4개까지는 rcx, rdx, r8, r9 레지스터로 전달.
  - 함수 호출시 home arg area 확보 32bytes -> sub rsp,32
  - 4개 초과분에 대해서는 rsp레지스터를 직접 조정(sub rsp,xxx)한 후 [rsp+xxxx] 메모리에 카피

- Callee
  - 진입시 rcx,rdx,r8,r9레지스터를 arg home area에 카피 (디버거에서 파라미터 내용 확인할거 아니면 필요없음)
  - 로컬변수 사용영역을 확보
    - 8bytes 변수 4개 사용시 8x4 = 32 bytes - sub rsp, 32 (16 Bytes Align에 주의!)

x64 calling convention, C/C++에서 C/C++함수 호출

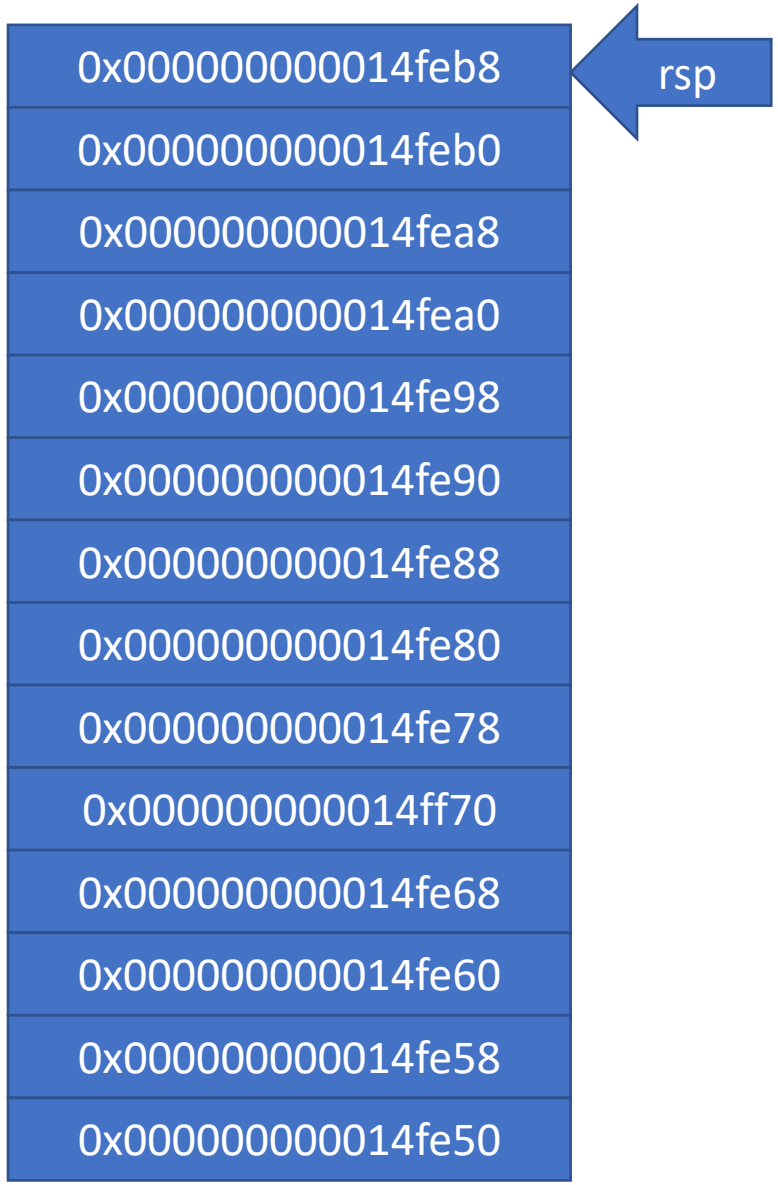
```
void Test()
{
    INT64 c = Add_C(1, 2);
}

INT64 Add_C(INT64 a, INT64 b)
{
    INT64 c = a + b;
    return c;
}
```

```
AsmTest64.exe!Test(void):
 sub            rsp,38h
 mov            edx,2
 mov            ecx,1
 call           Add_C (0140001050h)
 mov            qword ptr [c],rax
 add            rsp,38h
 ret
```

```
AsmTest64.exe!Add_C(__int64, __int64):
 mov            qword ptr [rsp+10h],rdx
 mov            qword ptr [rsp+8],rcx
 sub            rsp,18h
 mov            rax,qword ptr [b]
 mov            rcx,qword ptr [a]
 add            rcx,rax
 mov            rax,rcx
 mov            qword ptr [rsp],rax
 mov            rax,qword ptr [rsp]
 add            rsp,18h
 ret
```

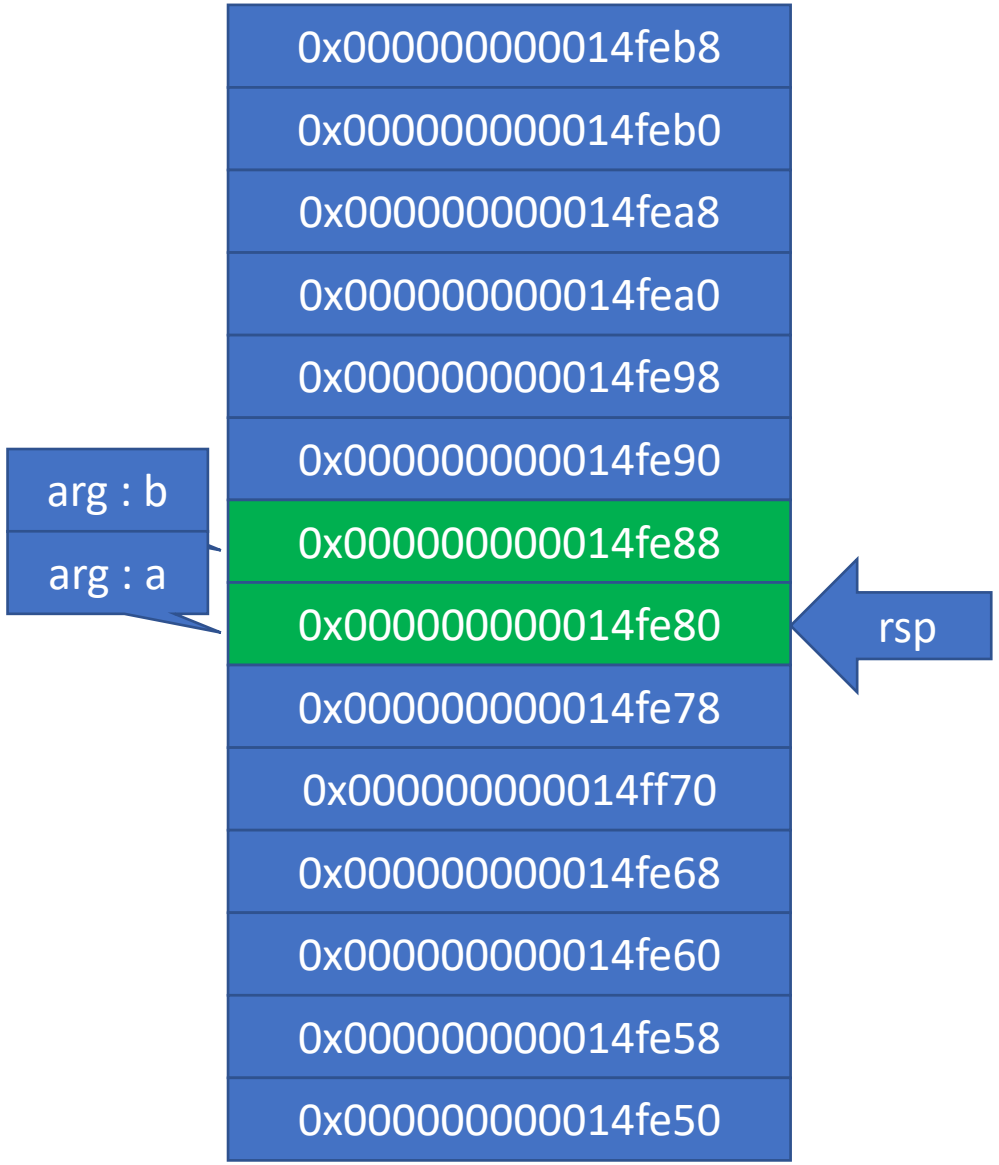
```
AsmTest64.exe!Test(void):
```

```
AsmTest64.exe!Add_C(__int64, __int64):
```



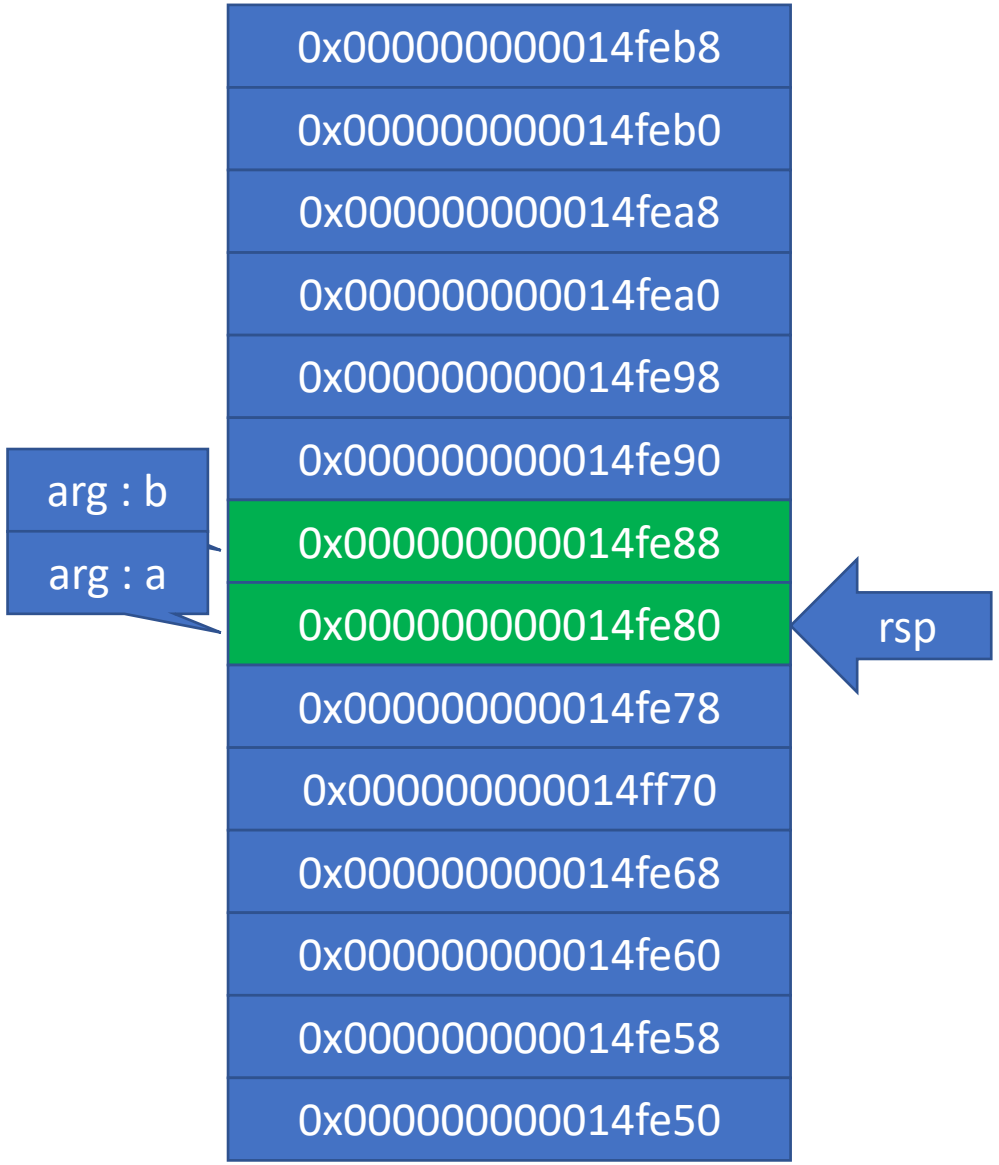
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
```

```
AsmTest64.exe!Add_C(__int64, __int64):
```



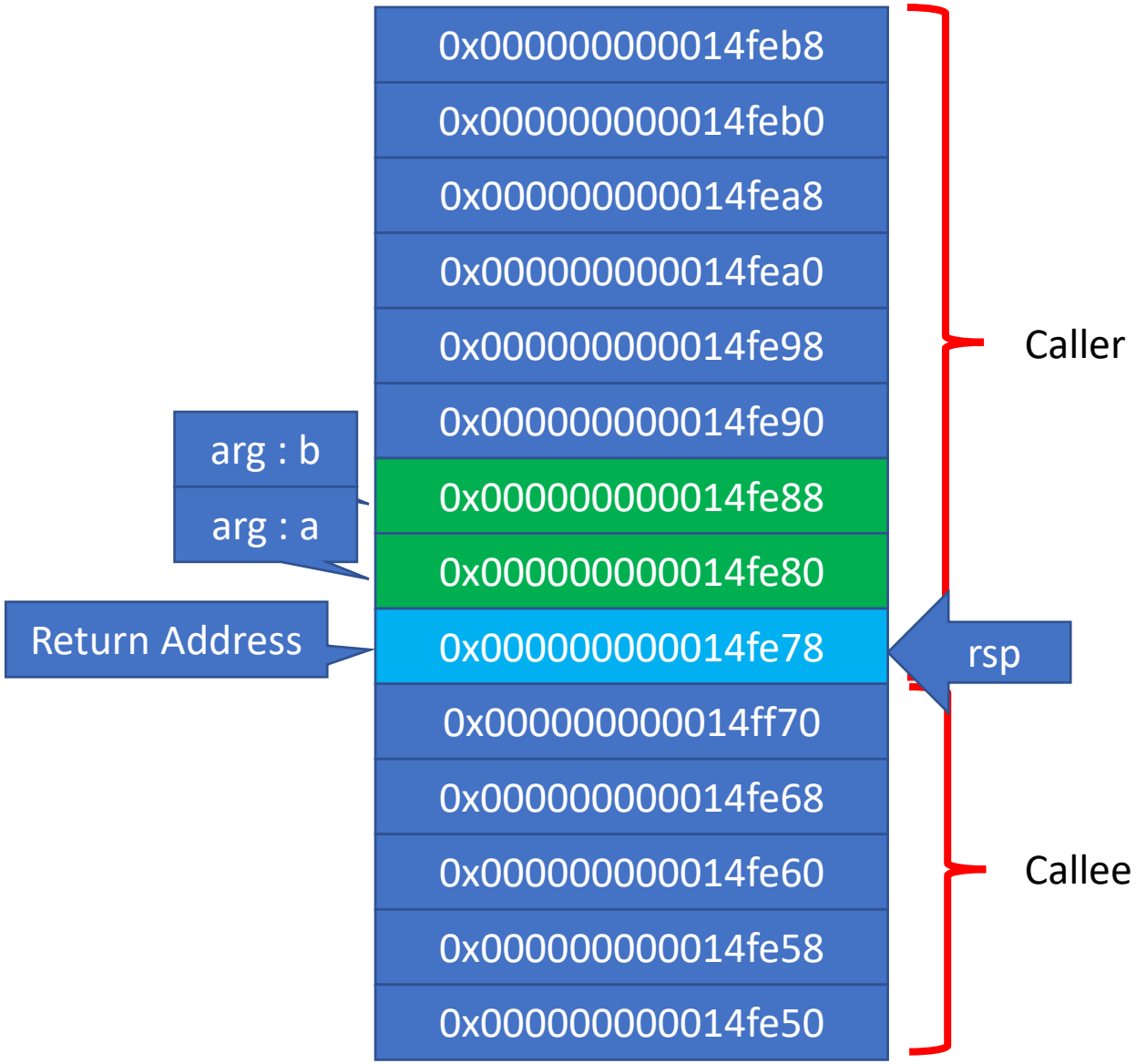
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
```

```
AsmTest64.exe!Add_C(__int64, __int64):
```



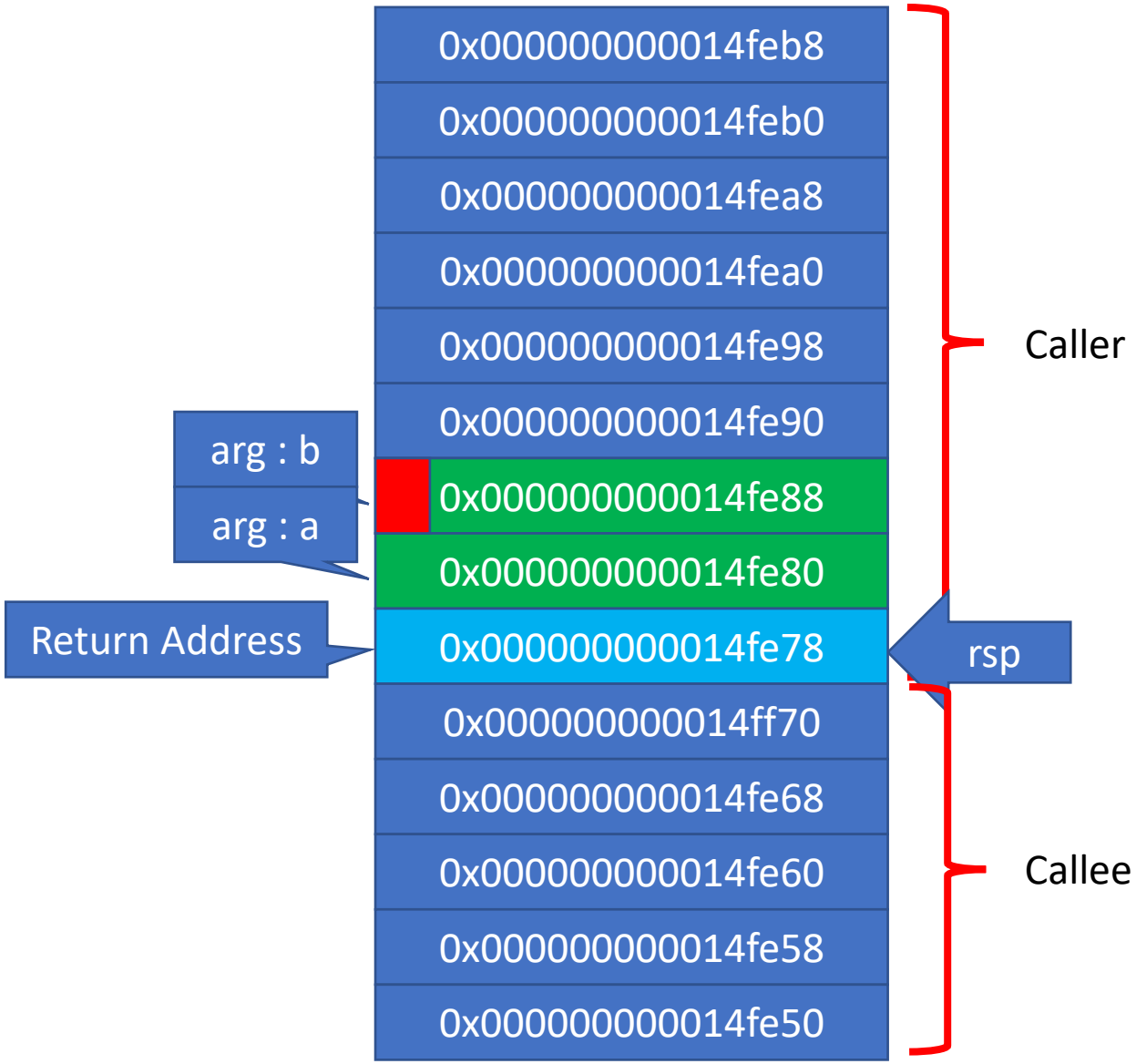
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_C
```

```
AsmTest64.exe!Add_C(__int64, __int64):
```



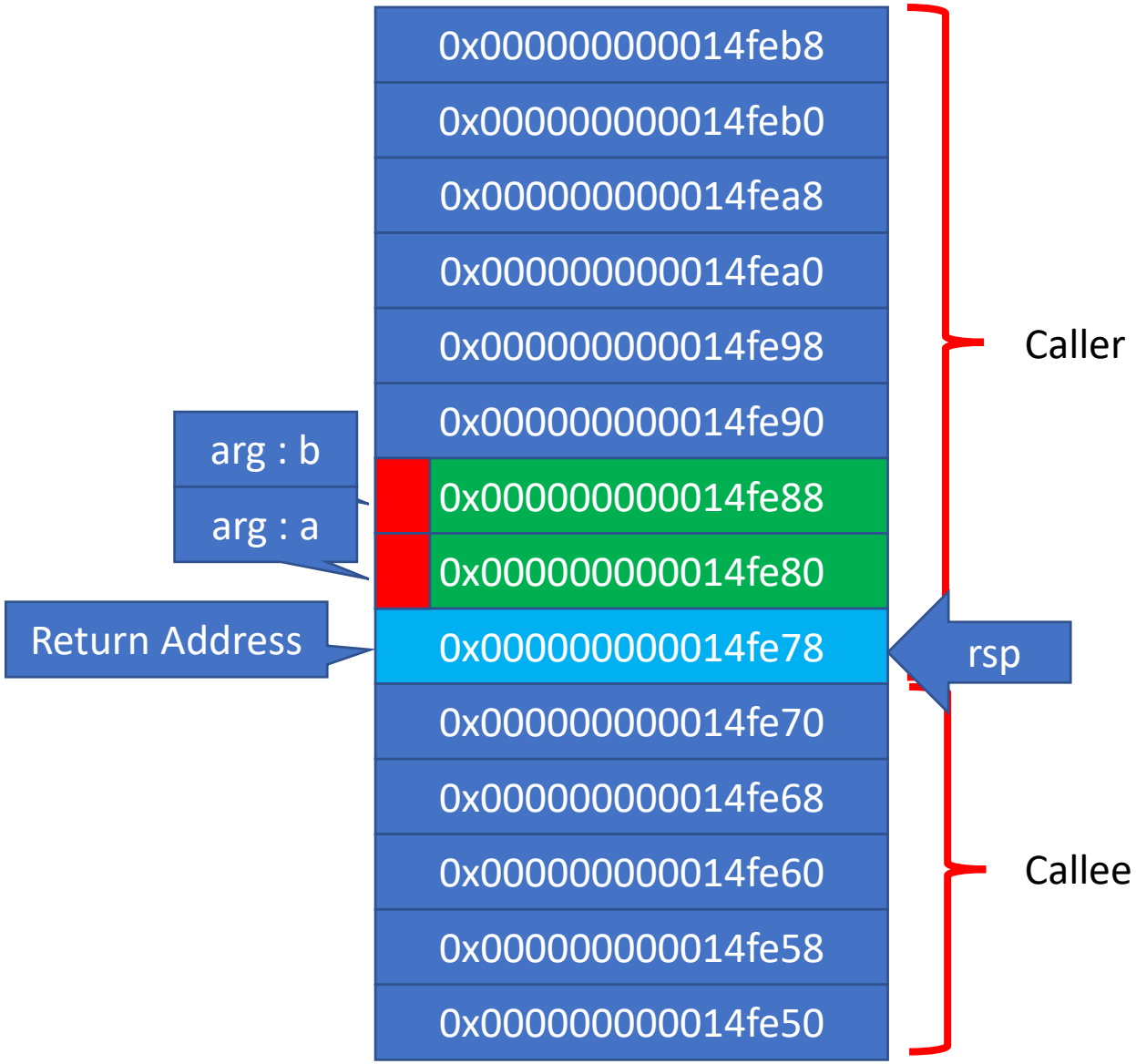
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_C
```

```
AsmTest64.exe!Add_C(__int64, __int64):
 mov          qword ptr [rsp+10h],rdx
```



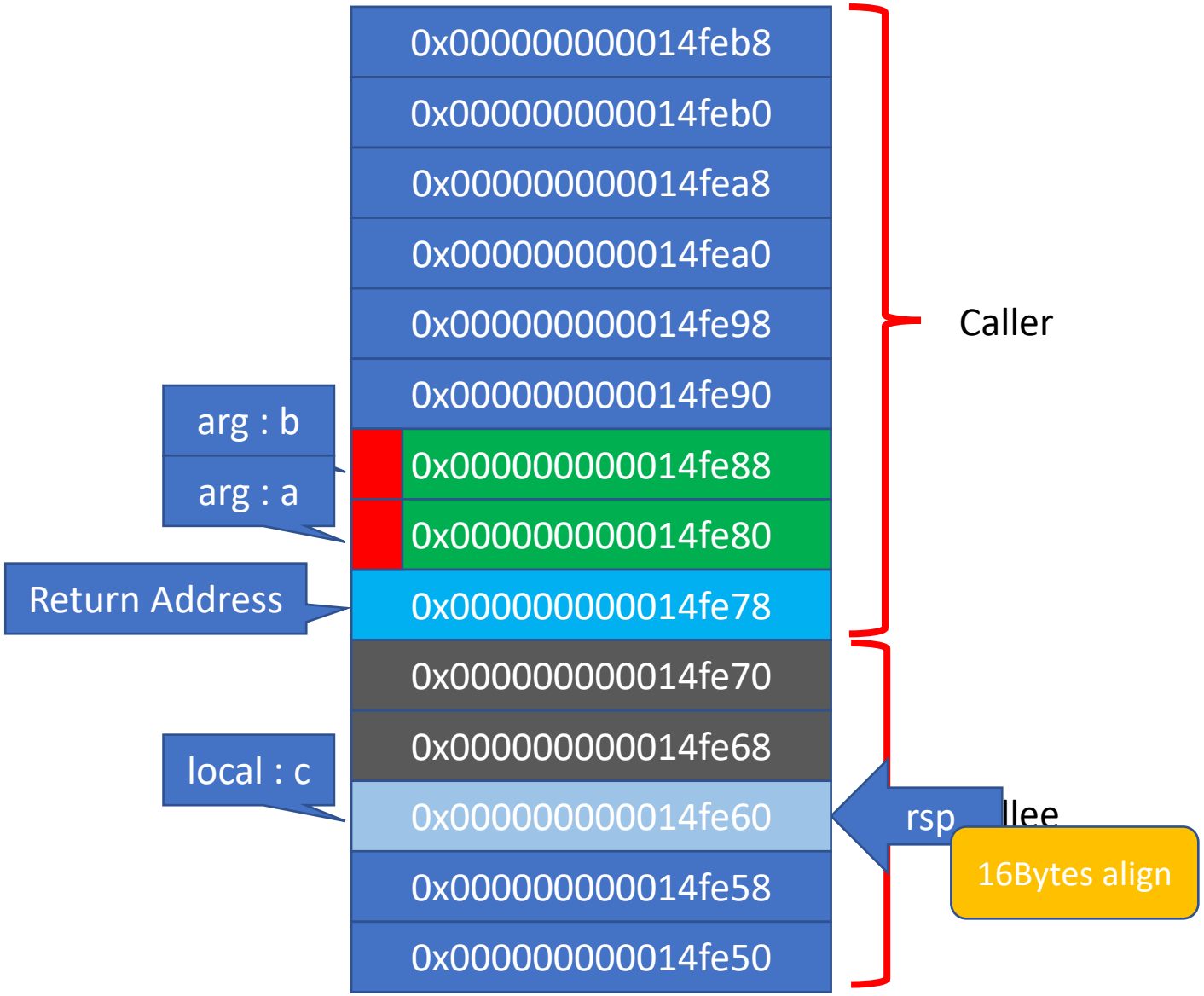
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_C
```

```
AsmTest64.exe!Add_C(__int64, __int64):
 mov          qword ptr [rsp+10h],rdx
 mov          qword ptr [rsp+8],rcx
```



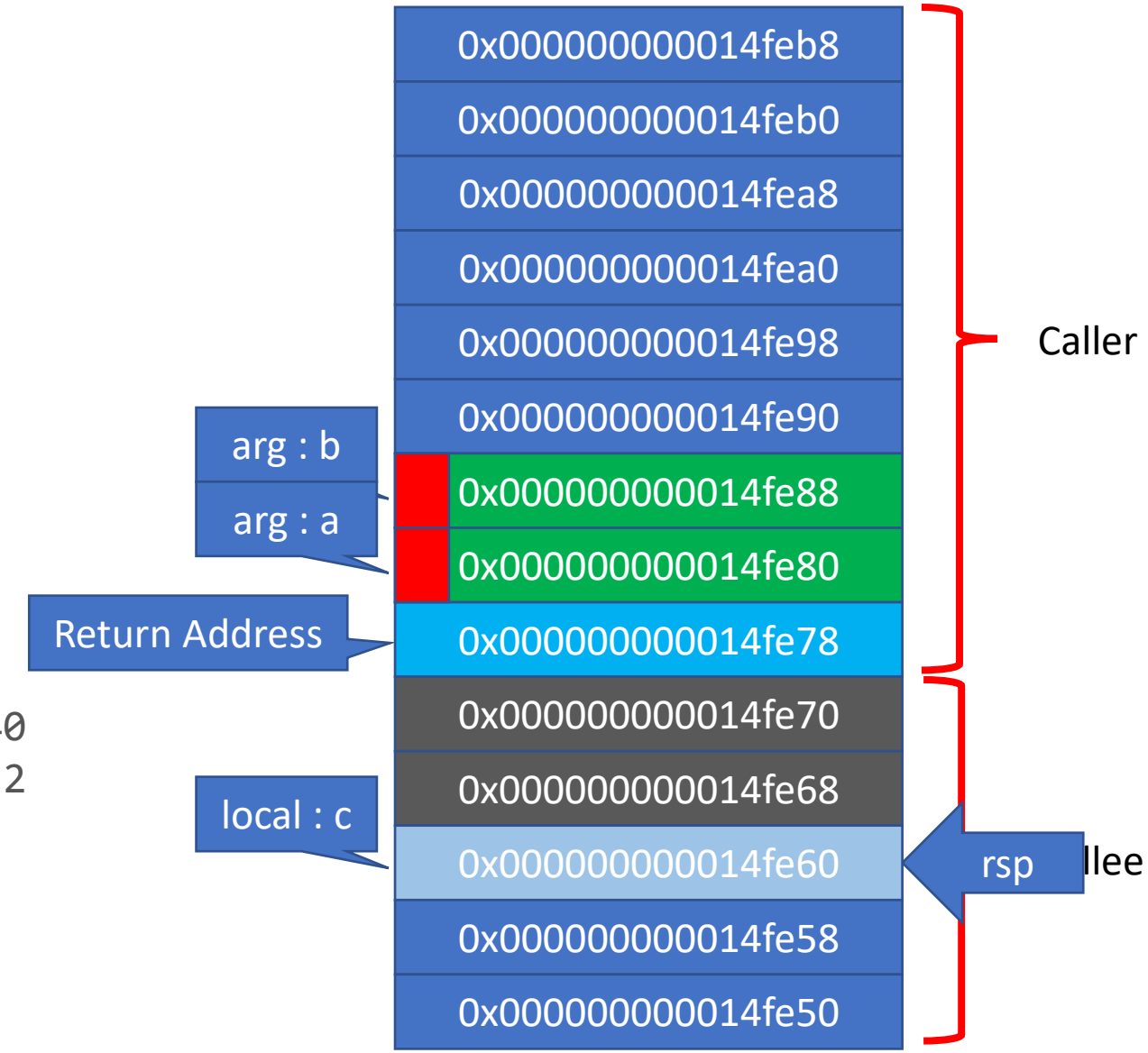
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_C
```

```
AsmTest64.exe!Add_C(__int64, __int64):
 mov          qword ptr [rsp+10h],rdx
 mov          qword ptr [rsp+8],rcx
 sub          rsp,18h
```



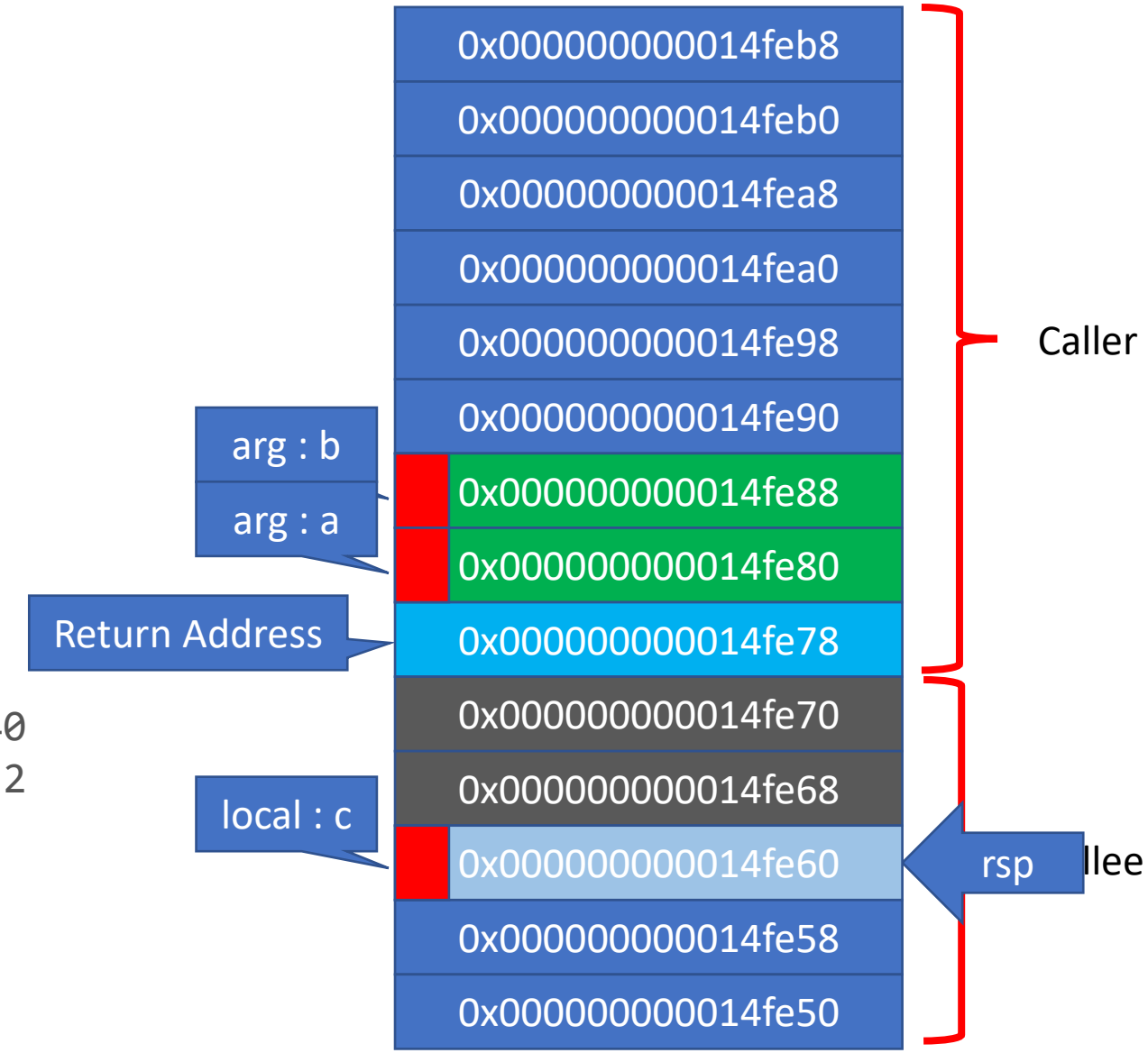
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_C
```

```
AsmTest64.exe!Add_C(__int64, __int64):
 mov          qword ptr [rsp+10h],rdx
 mov          qword ptr [rsp+8],rcx
 sub          rsp,18h
 mov          rax,qword ptr [b]  ; rsp+40
 mov          rcx,qword ptr [a]  ; rsp+32
 add          rcx,rax
 mov          rax,rcx
```



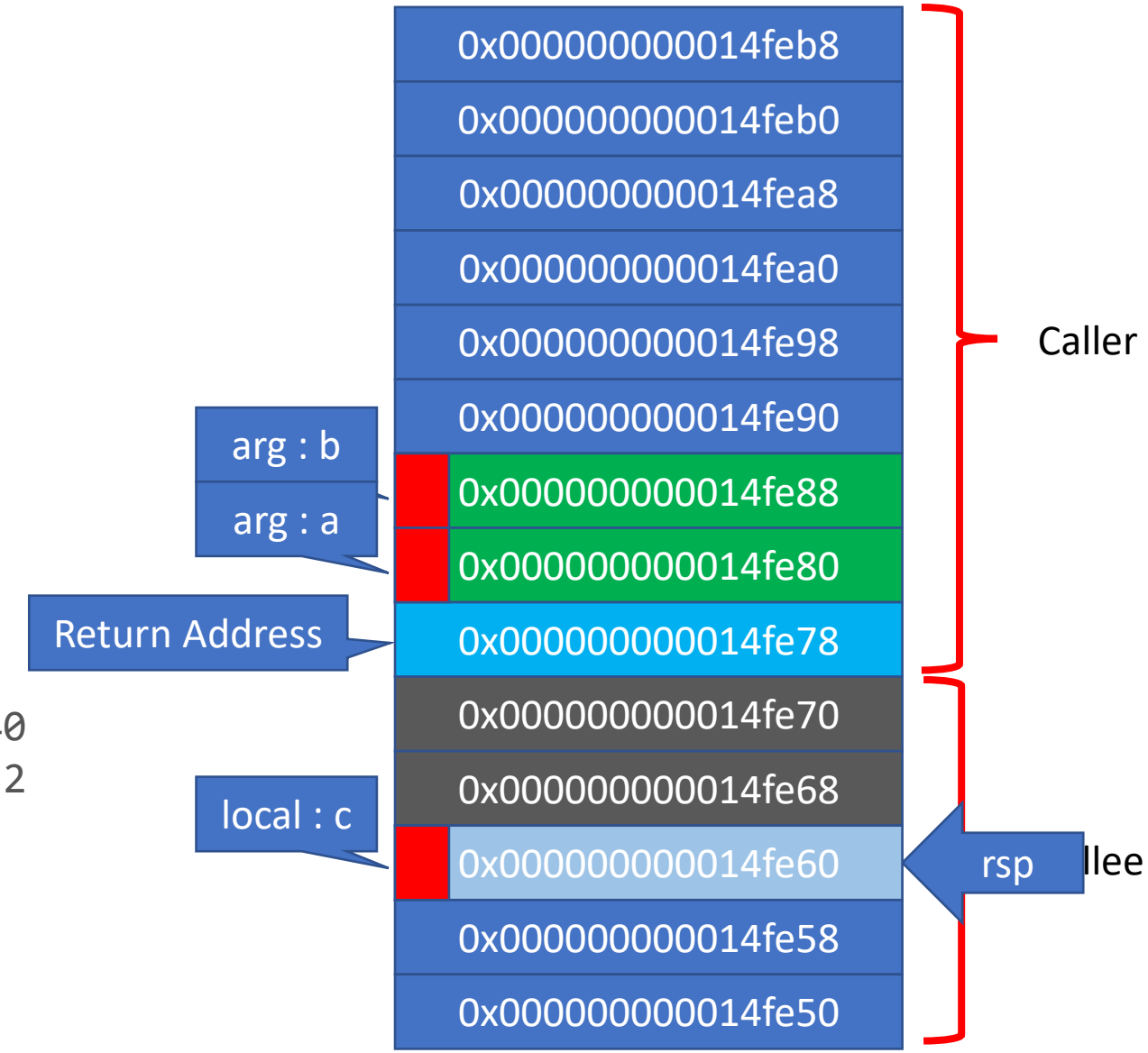
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_C
```

```
AsmTest64.exe!Add_C(__int64, __int64):
 mov          qword ptr [rsp+10h],rdx
 mov          qword ptr [rsp+8],rcx
 sub          rsp,18h
 mov          rax,qword ptr [b]  ; rsp+40
 mov          rcx,qword ptr [a]  ; rsp+32
 add          rcx,rax
 mov          rax,rcx
 mov          qword ptr [rsp],rax
```



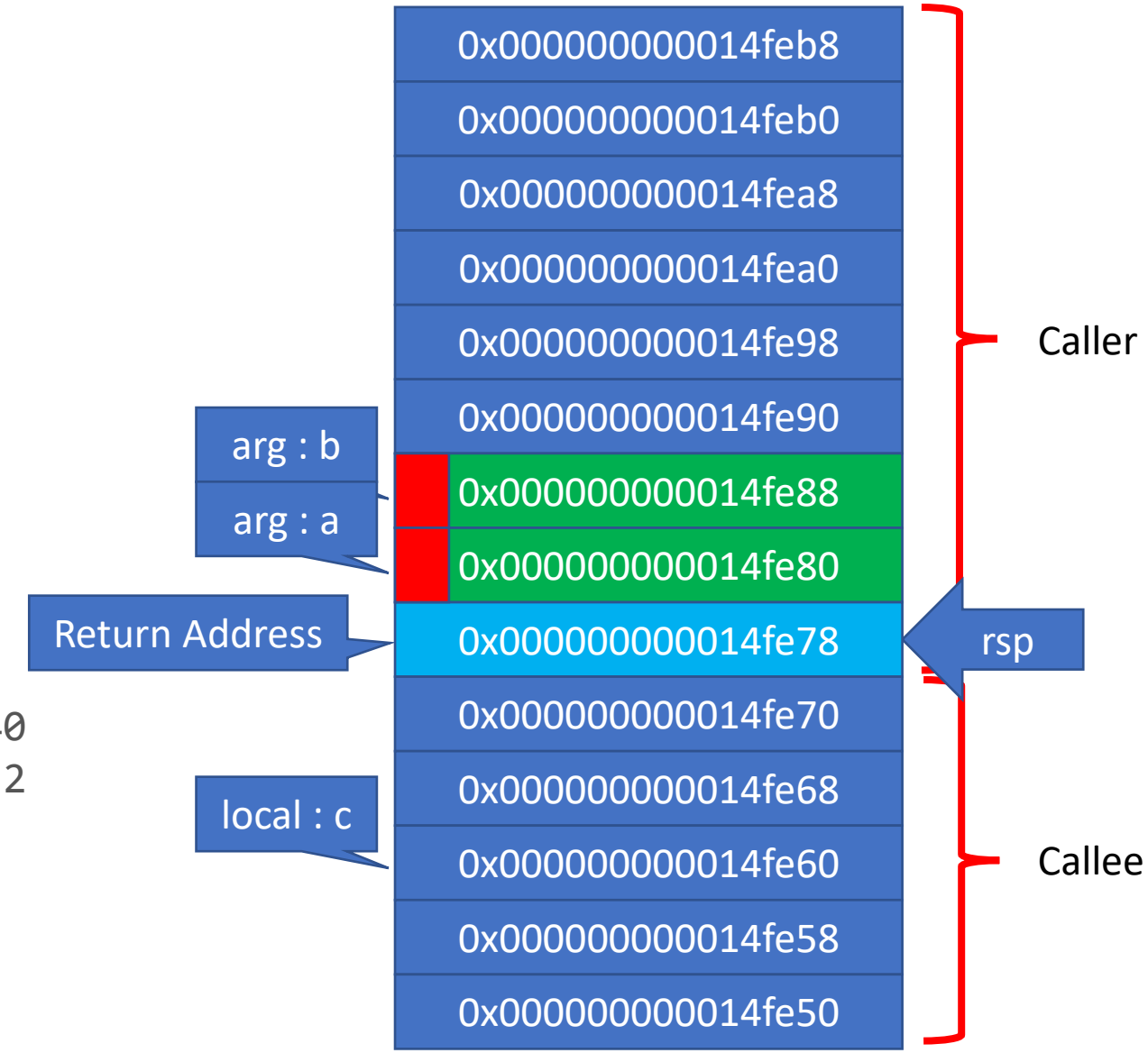
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_C
```

```
AsmTest64.exe!Add_C(__int64, __int64):
 mov          qword ptr [rsp+10h],rdx
 mov          qword ptr [rsp+8],rcx
 sub          rsp,18h
 mov          rax,qword ptr [b]  ; rsp+40
 mov          rcx,qword ptr [a]  ; rsp+32
 add          rcx,rax
 mov          rax,rcx
 mov          qword ptr [rsp],rax
 mov          rax,qword ptr [rsp]
```



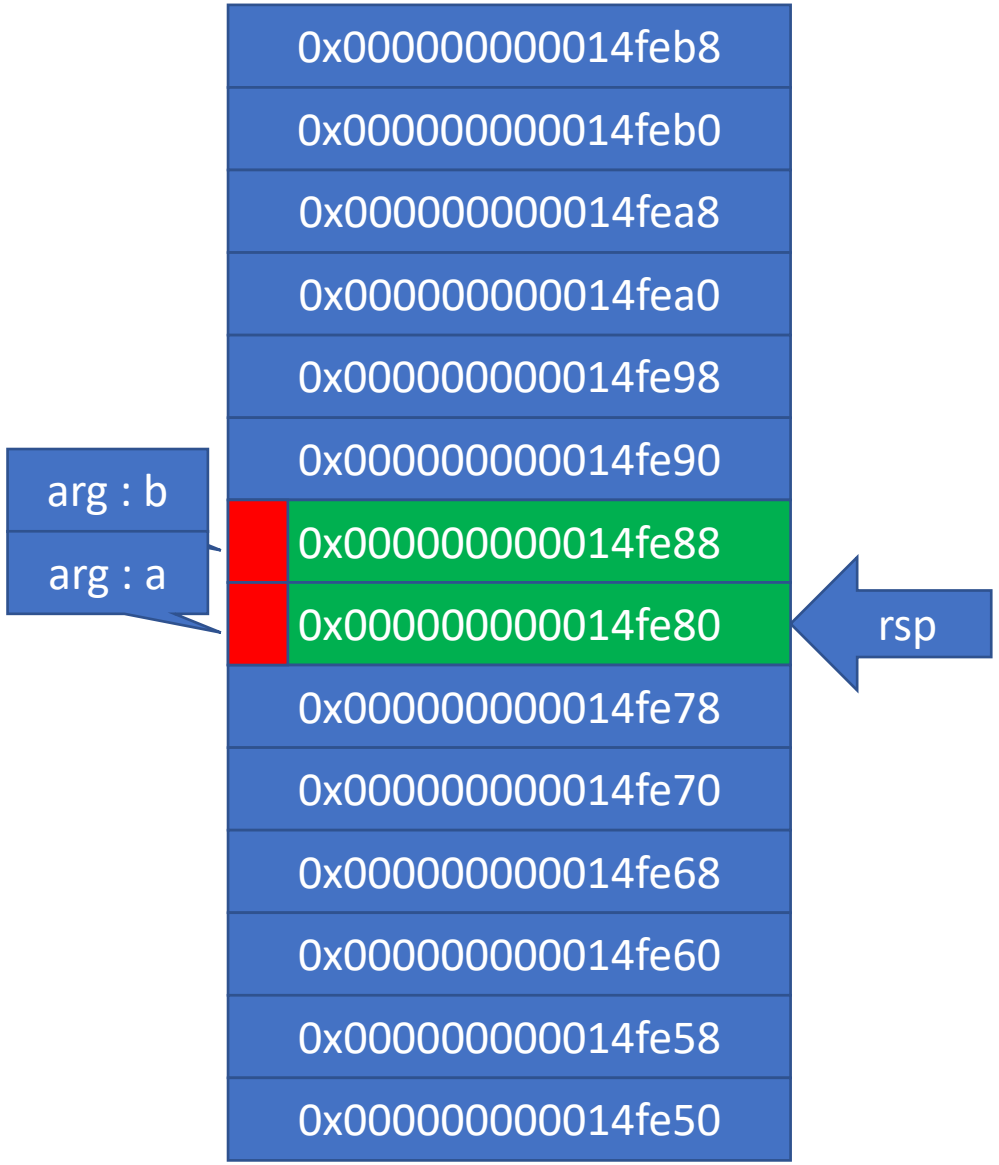
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_C
```

```
AsmTest64.exe!Add_C(__int64, __int64):
 mov          qword ptr [rsp+10h],rdx
 mov          qword ptr [rsp+8],rcx
 sub          rsp,18h
 mov          rax,qword ptr [b]  ; rsp+40
 mov          rcx,qword ptr [a]  ; rsp+32
 add          rcx,rax
 mov          rax,rcx
 mov          qword ptr [rsp],rax
 mov          rax,qword ptr [rsp]
 add          rsp,18h
```



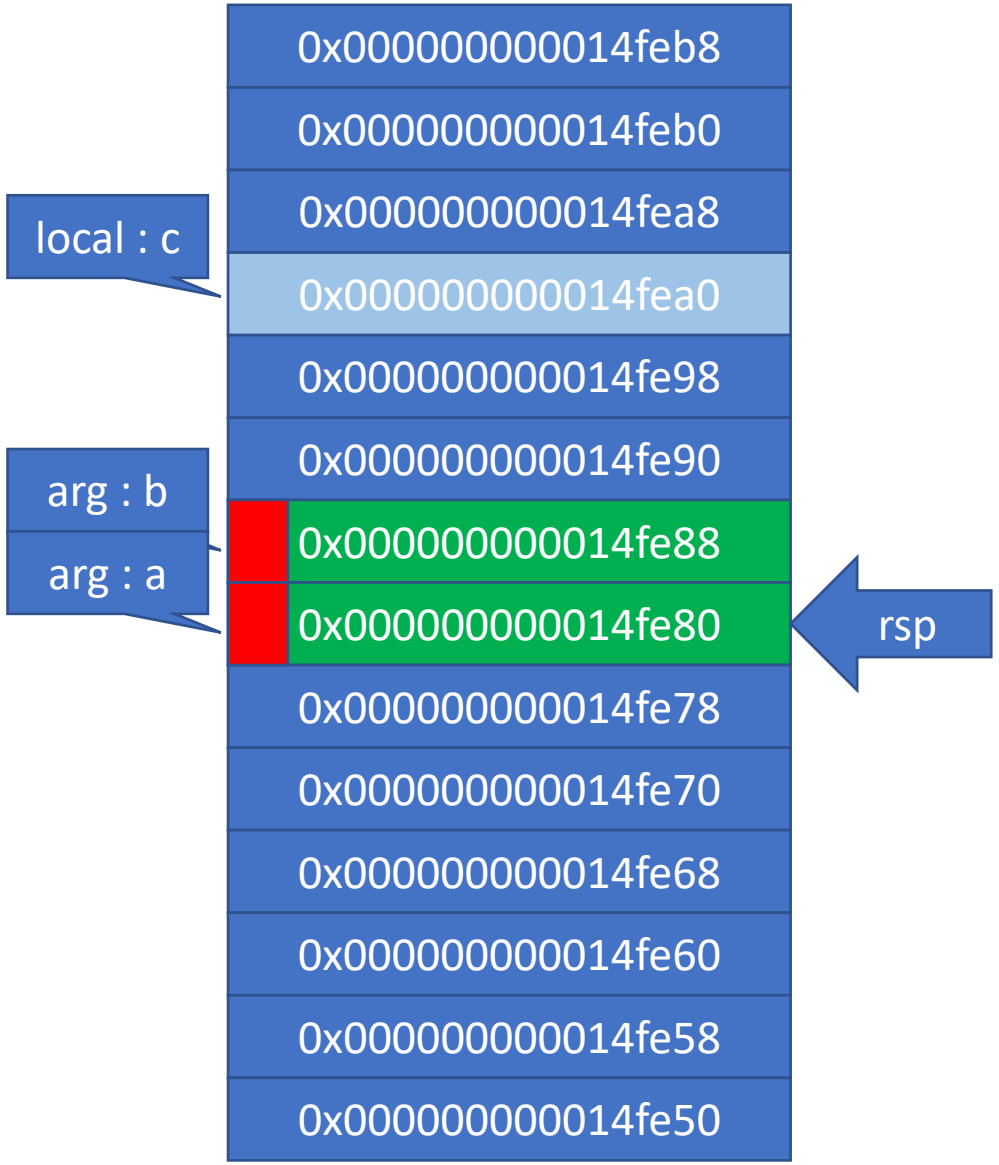
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_C
```

```
AsmTest64.exe!Add_C(__int64, __int64):
 mov          qword ptr [rsp+10h],rdx
 mov          qword ptr [rsp+8],rcx
 sub          rsp,18h
 mov          rax,qword ptr [b]  ; rsp+40
 mov          rcx,qword ptr [a]  ; rsp+32
 add          rcx,rax
 mov          rax,rcx
 mov          qword ptr [rsp],rax
 mov          rax,qword ptr [rsp]
 add          rsp,18h
 ret
```



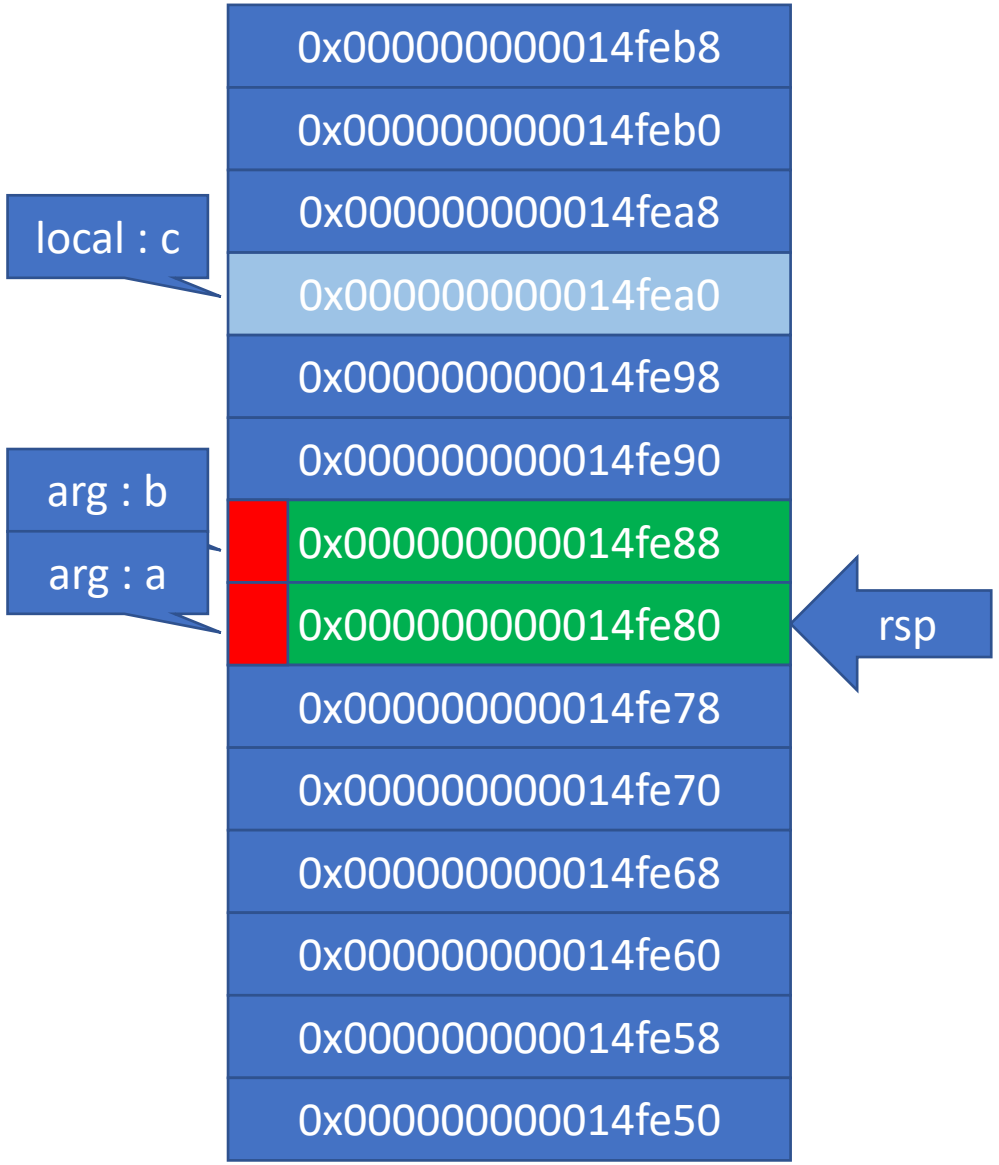
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_C
 mov          qword ptr [c],rax
```

```
AsmTest64.exe!Add_C(__int64, __int64):
 mov          qword ptr [rsp+10h],rdx
 mov          qword ptr [rsp+8],rcx
 sub          rsp,18h
 mov          rax,qword ptr [b]  ; rsp+40
 mov          rcx,qword ptr [a]  ; rsp+32
 add          rcx,rax
 mov          rax,rcx
 mov          qword ptr [rsp],rax
 mov          rax,qword ptr [rsp]
 add          rsp,18h
 ret
```

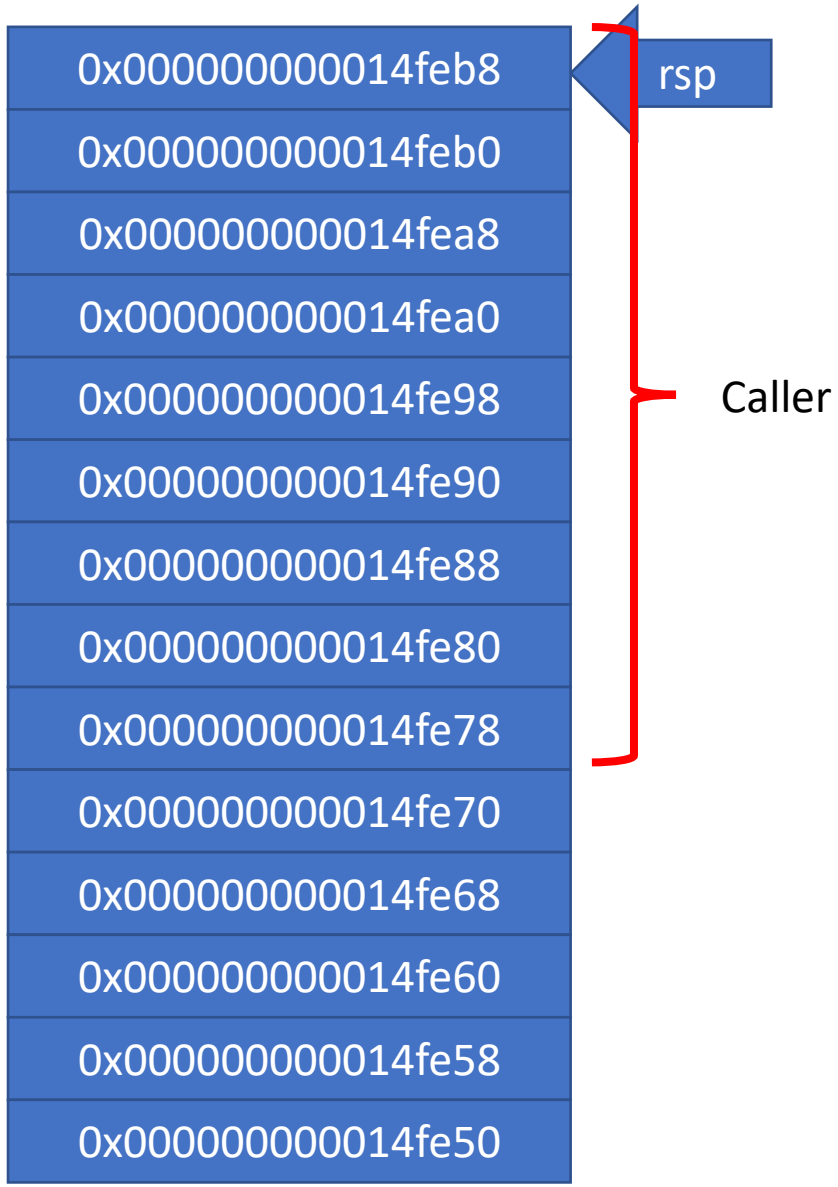


```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_C
 mov          qword ptr [c],rax
```

```
AsmTest64.exe!Add_C(__int64, __int64):
 mov          qword ptr [rsp+10h],rdx
 mov          qword ptr [rsp+8],rcx
 sub          rsp,18h
 mov          rax,qword ptr [b]  ; rsp+40
 mov          rcx,qword ptr [a]  ; rsp+32
 add          rcx,rax
 mov          rax,rcx
 mov          qword ptr [rsp],rax
 mov          rax,qword ptr [rsp]
 add          rsp,18h
 ret
```



```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_C
 mov          qword ptr [c],rax
 add          rsp,38h
 ret

AsmTest64.exe!Add_C(__int64, __int64):
 mov          qword ptr [rsp+10h],rdx
 mov          qword ptr [rsp+8],rcx
 sub          rsp,18h
 mov          rax,qword ptr [b]  ; rsp+40
 mov          rcx,qword ptr [a]  ; rsp+32
 add          rcx,rax
 mov          rax,rcx
 mov          qword ptr [rsp],rax
 mov          rax,qword ptr [rsp]
 add          rsp,18h
 ret
```



X64어셈블리어 prologue, epilogue

- x86함수와 비교
- 베이스 포인터 사용이 기본은 아니다.
- 함수 내부에서 편리하게 사용하고 싶으면 베이스 포인터를 사용한다.

MASM에서의 Stack Frame 자동생성

asm code in MASM

```
Add_ASM_RBP PROC a:QWORD, b:QWORD

    LOCAL c:QWORD
    mov a,rcx
    mov b,rdx
    add rcx,rdx
    mov c, rcx
    mov rax,c
    ret

Add_ASM_RBP ENDP
```

assembled code

```
push        rbp
mov         rbp,rsp
add         rsp,0FFFFFFFFFFFFFFF8h
mov         qword ptr [rbp+16],rcx
mov         qword ptr [rbp+24],rdx
add         rcx,rdx
mov         qword ptr [rbp-8],rcx
mov         rax,qword ptr [rbp-8]
leave
ret
```

x64 calling convention, C/C++에서 asm함수 호출

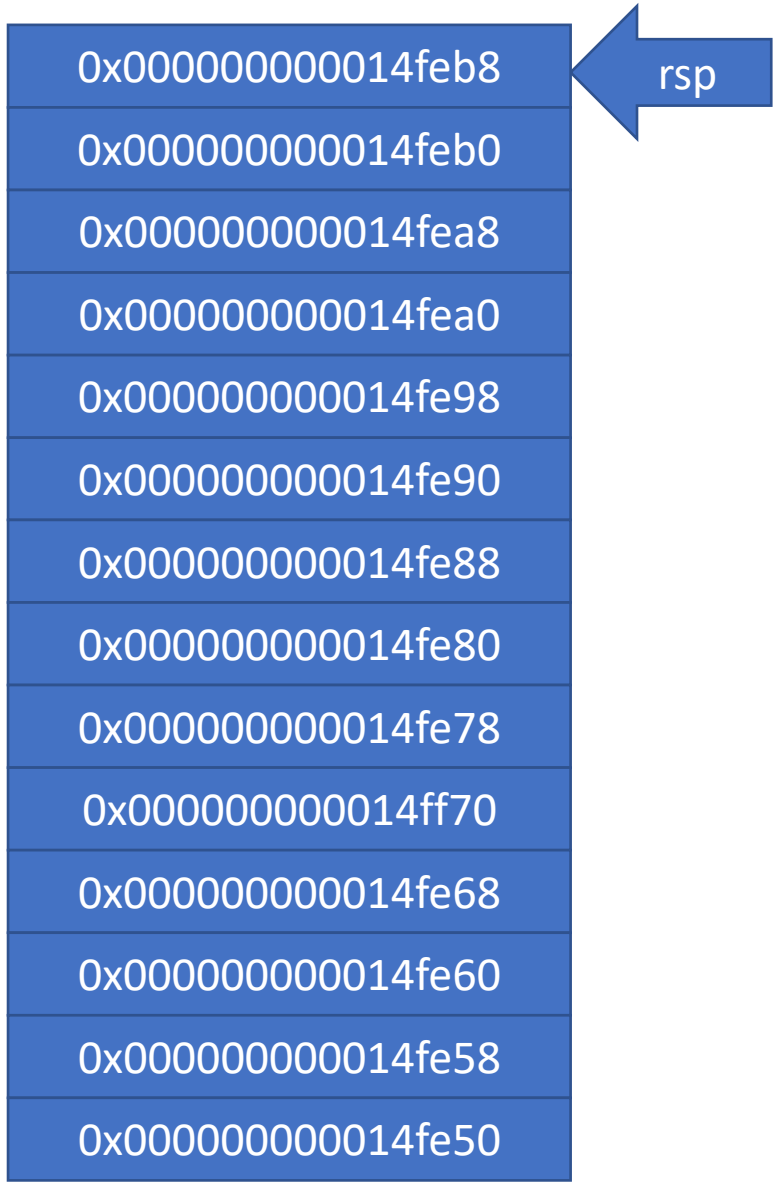
```
void Test()
{
    INT64 c = Add_ASM_RBP(1, 2);
}

Add_ASM_RBP PROC a:QWORD, b:QWORD
    LOCAL c:QWORD
    LOCAL d:QWORD
    LOCAL e:QWORD
    LOCAL f:QWORD

    mov a,rcx
    mov b,rdx

    add rcx,rdx
    mov c, rcx
    mov rax,c
    ret
Add_ASM_RBP ENDP
```

```
AsmTest64.exe!Test(void):
 sub        rsp,38h
 mov        edx,2
 mov        ecx,1
 call       Add_ASM_RBP
 mov        qword ptr [c],rax
 add        rsp,38h
 ret
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push       rbp
 mov        rbp,rsp
 add        rsp,0FFFFFFFFFFFFFFE0h
 mov        qword ptr [a],rcx
 mov        qword ptr [b],rdx
 add        rcx,rdx
 mov        qword ptr [c],rcx
 mov        rax,qword ptr [c]
 leave
 ret
```

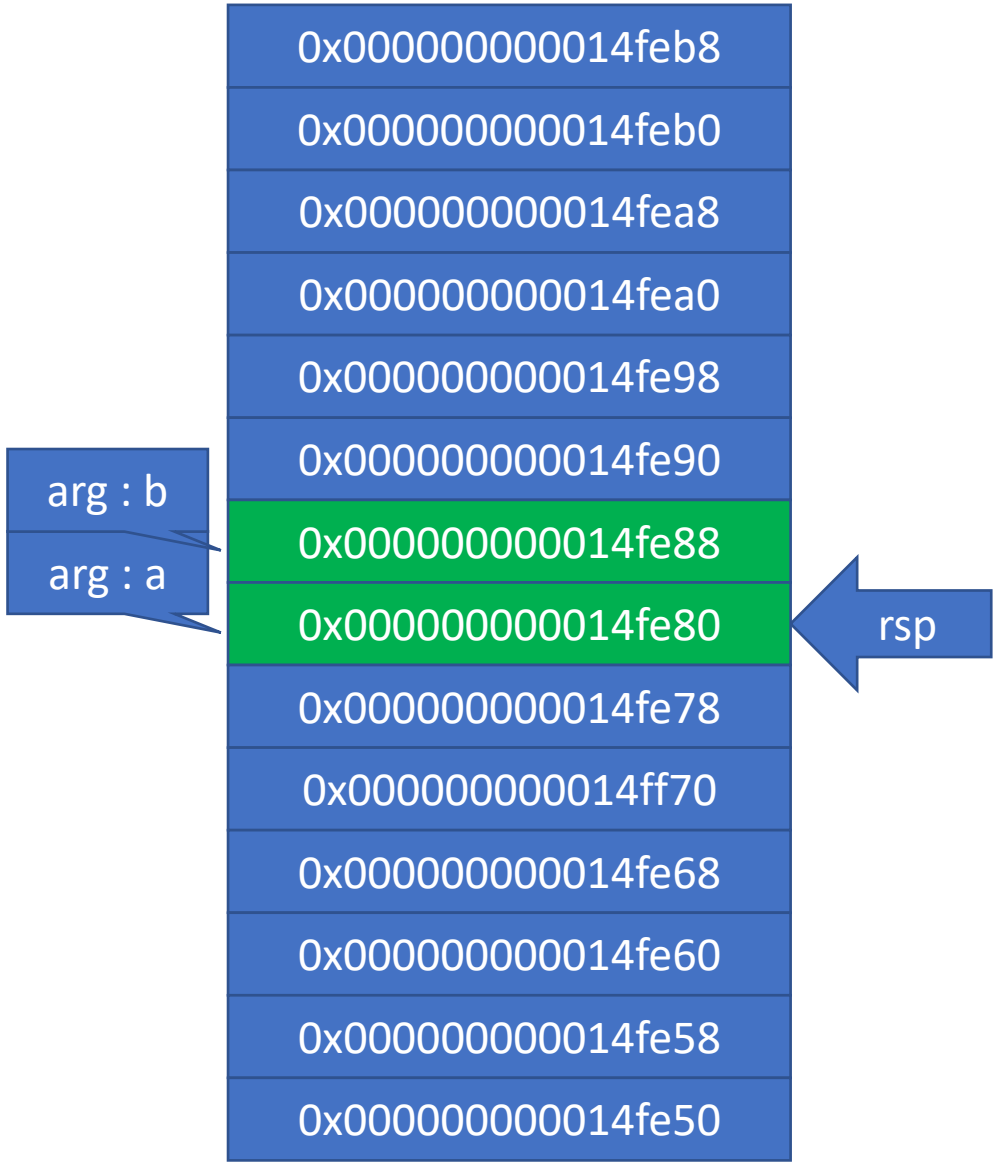
AsmTest64.exe!Test(void):

AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):



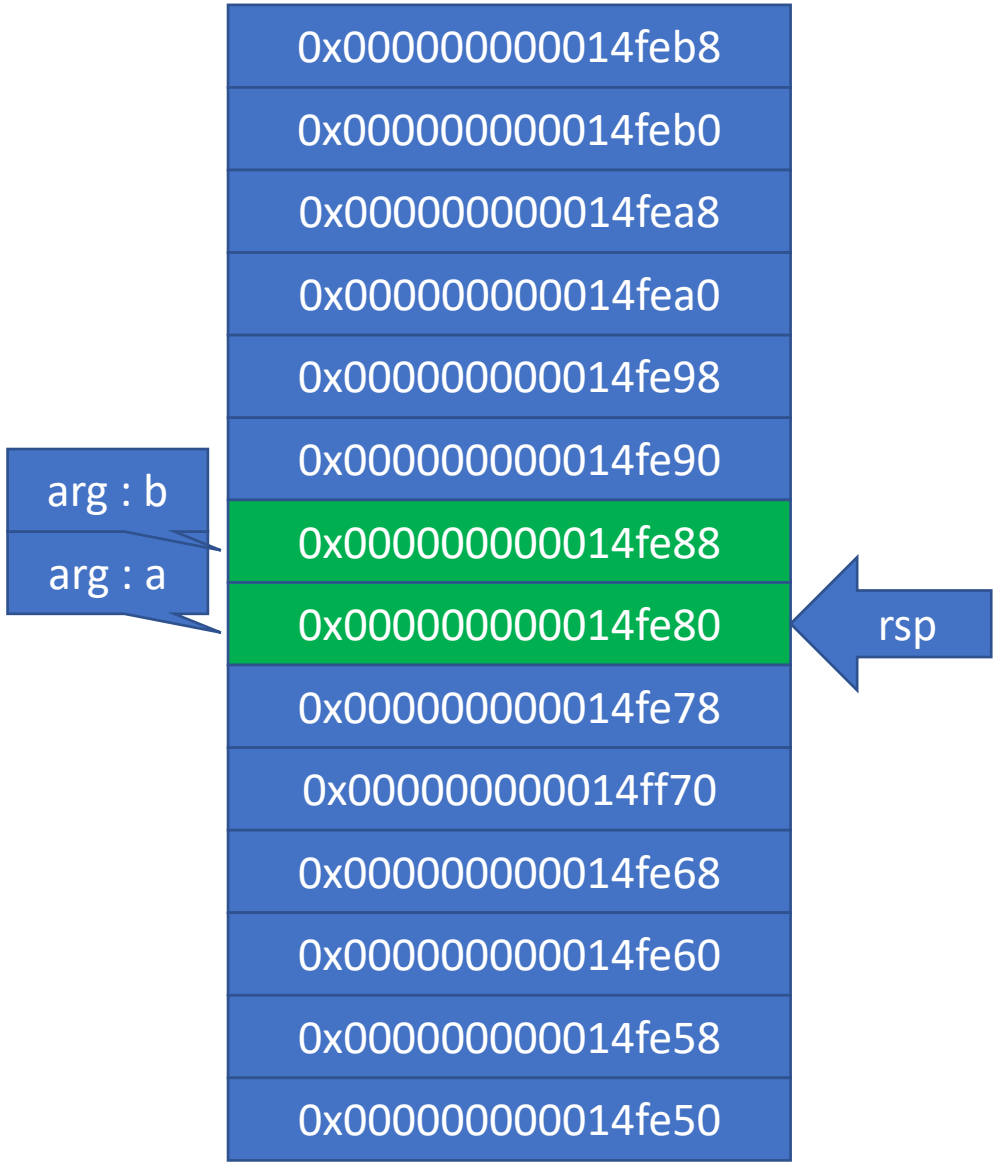
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
```



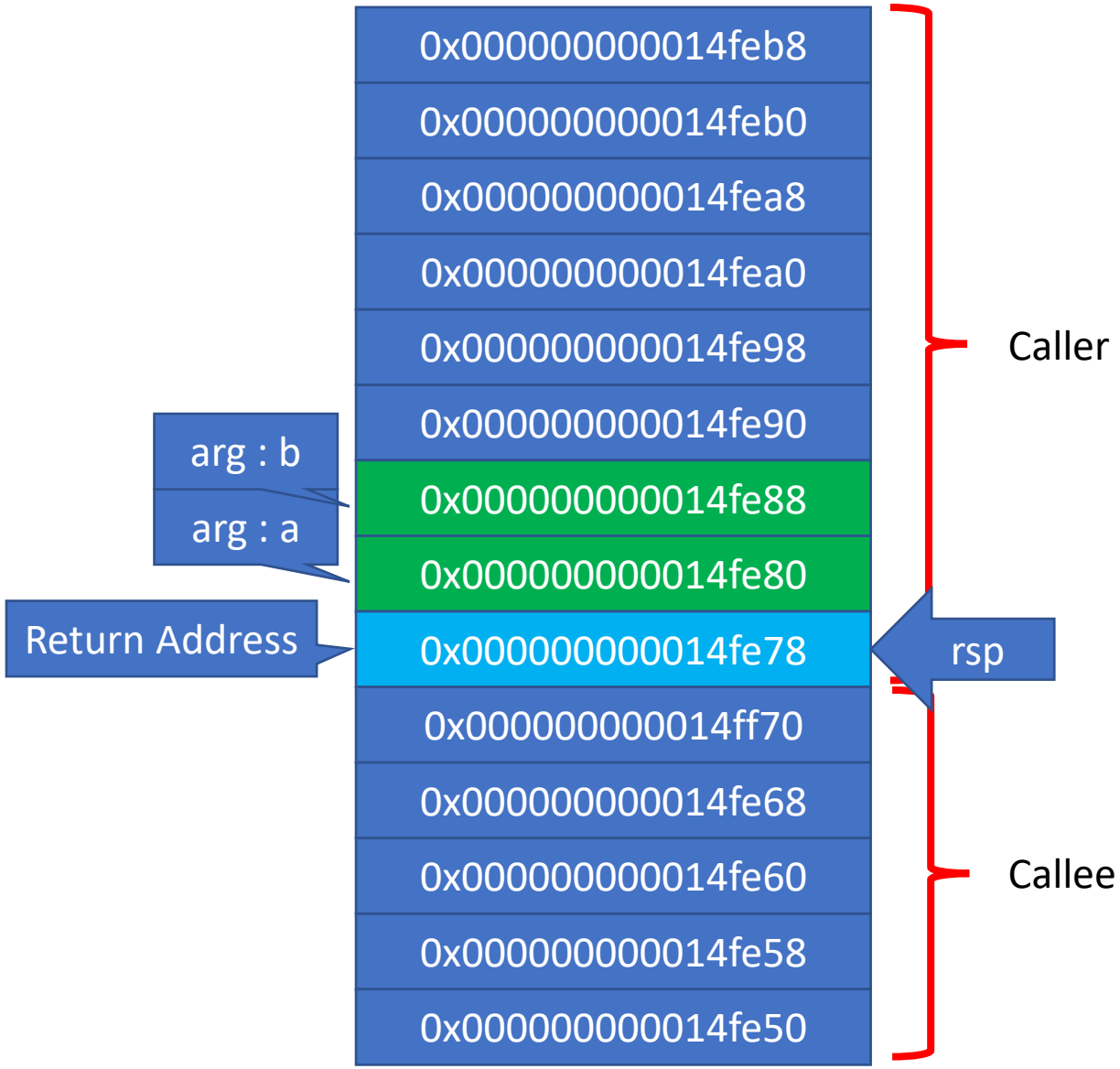
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
```



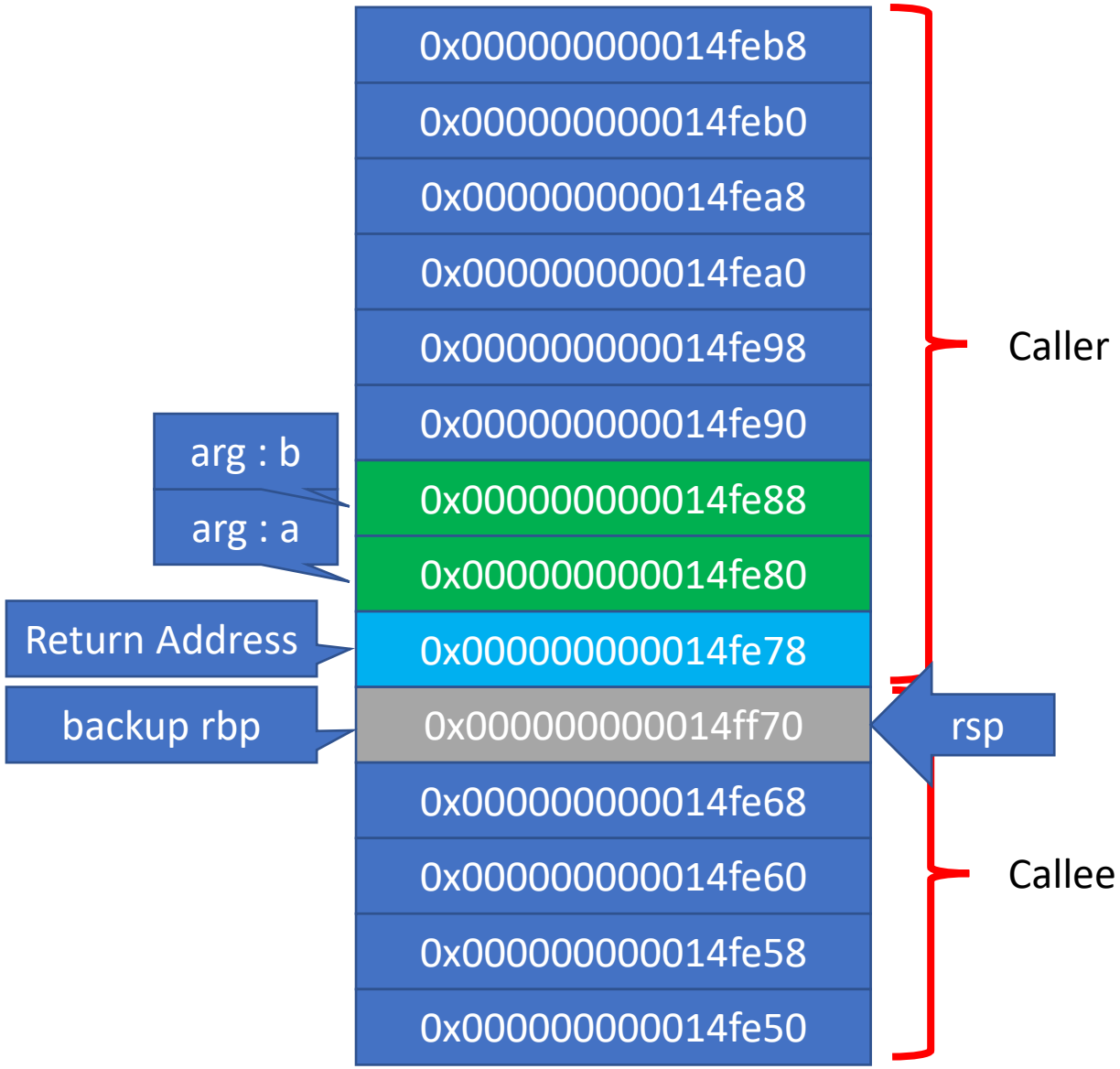
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
```



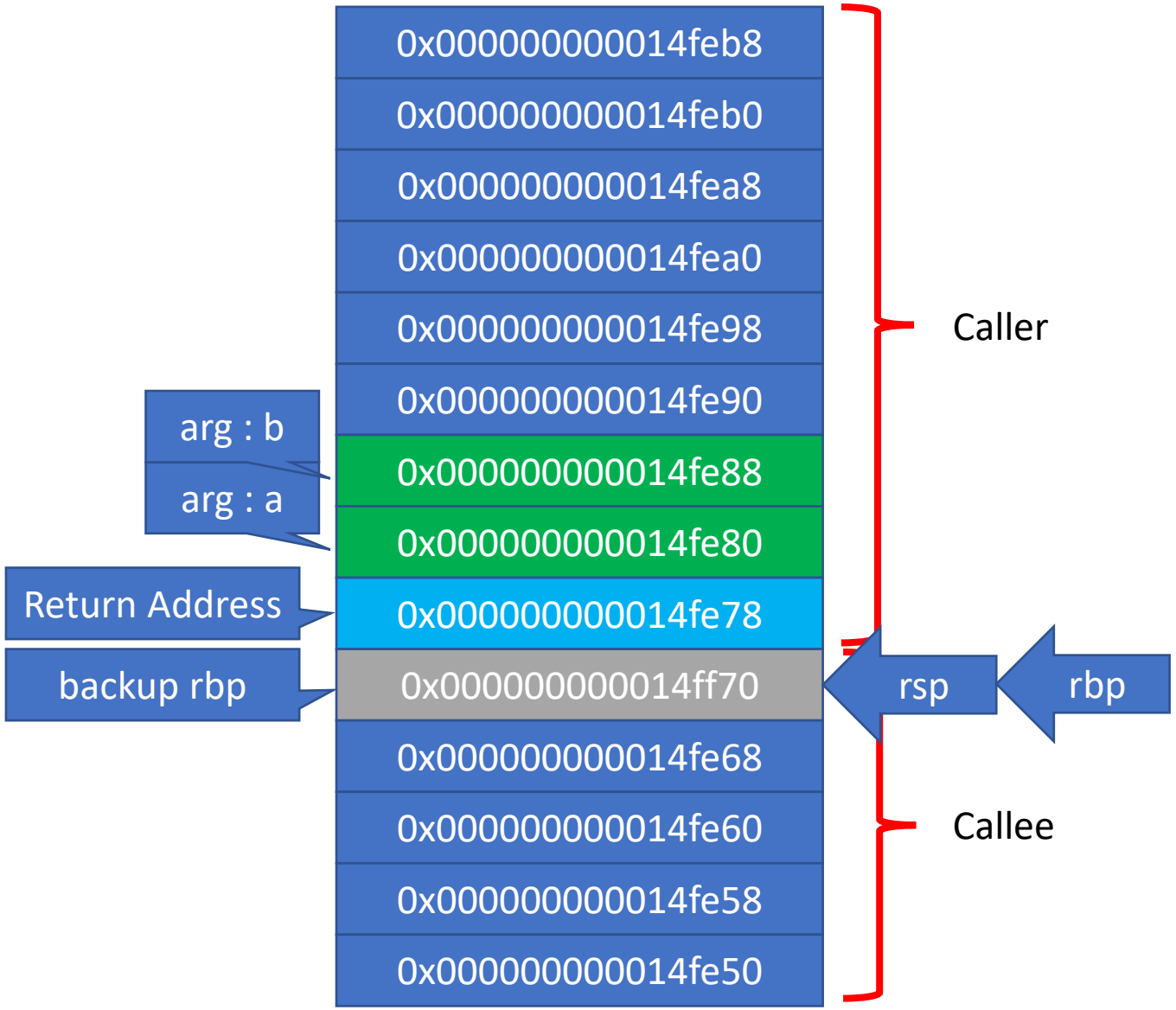
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push         rbp
```



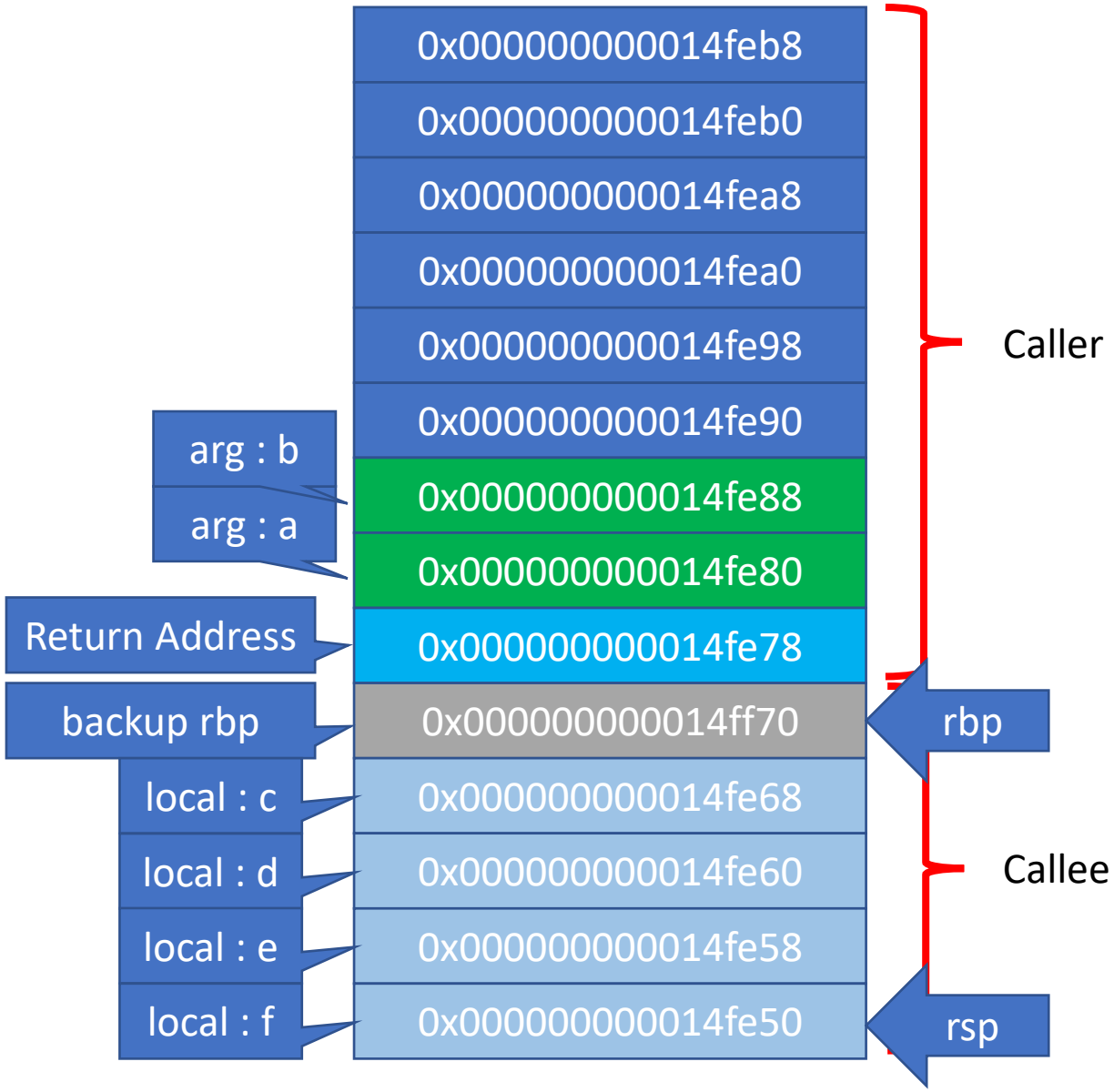
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push         rbp
 mov          rbp,rsp
```



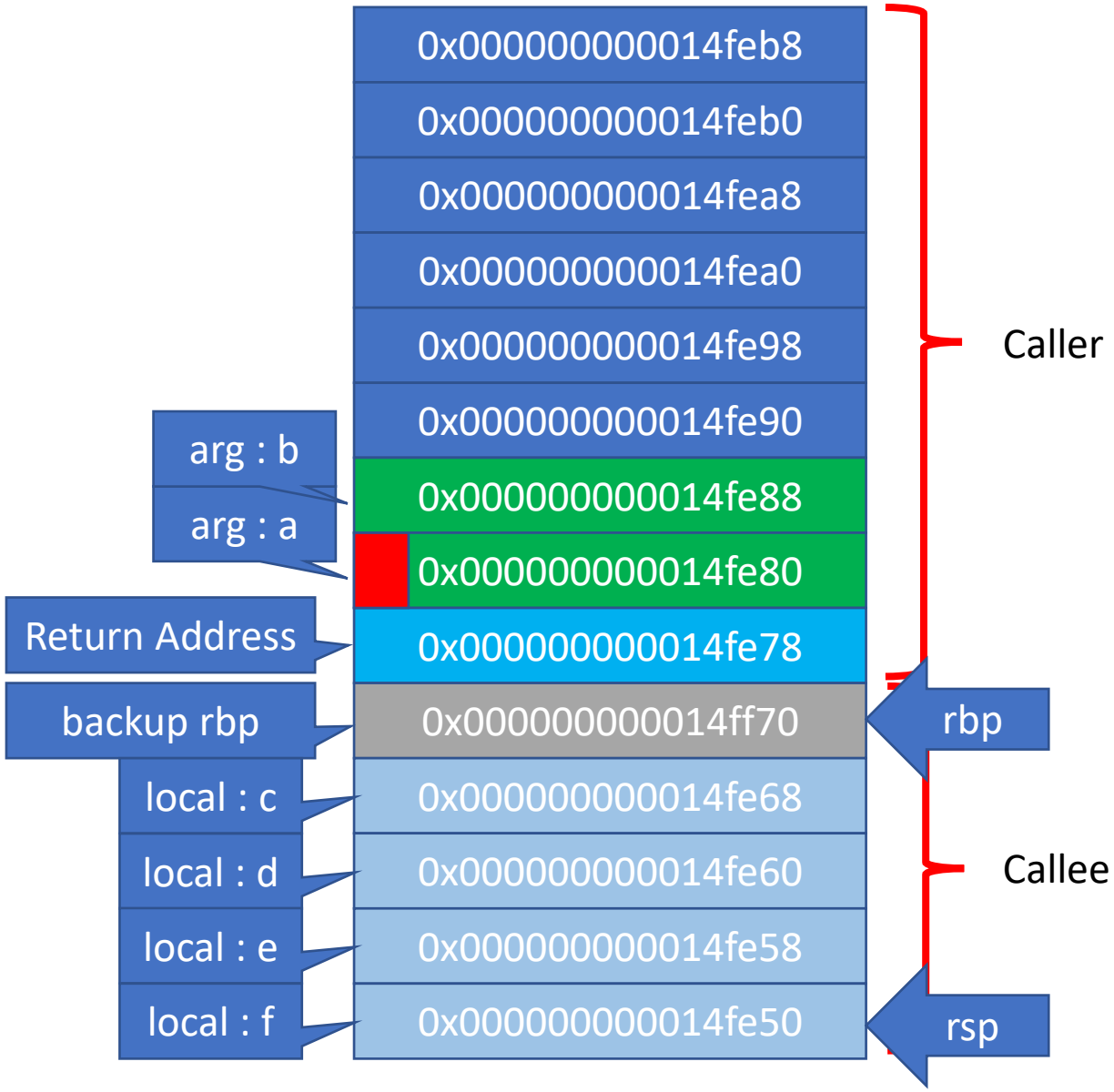
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push         rbp
 mov          rbp,rsp
 add          rsp,0FFFFFFFFFFFFFFE0h
```



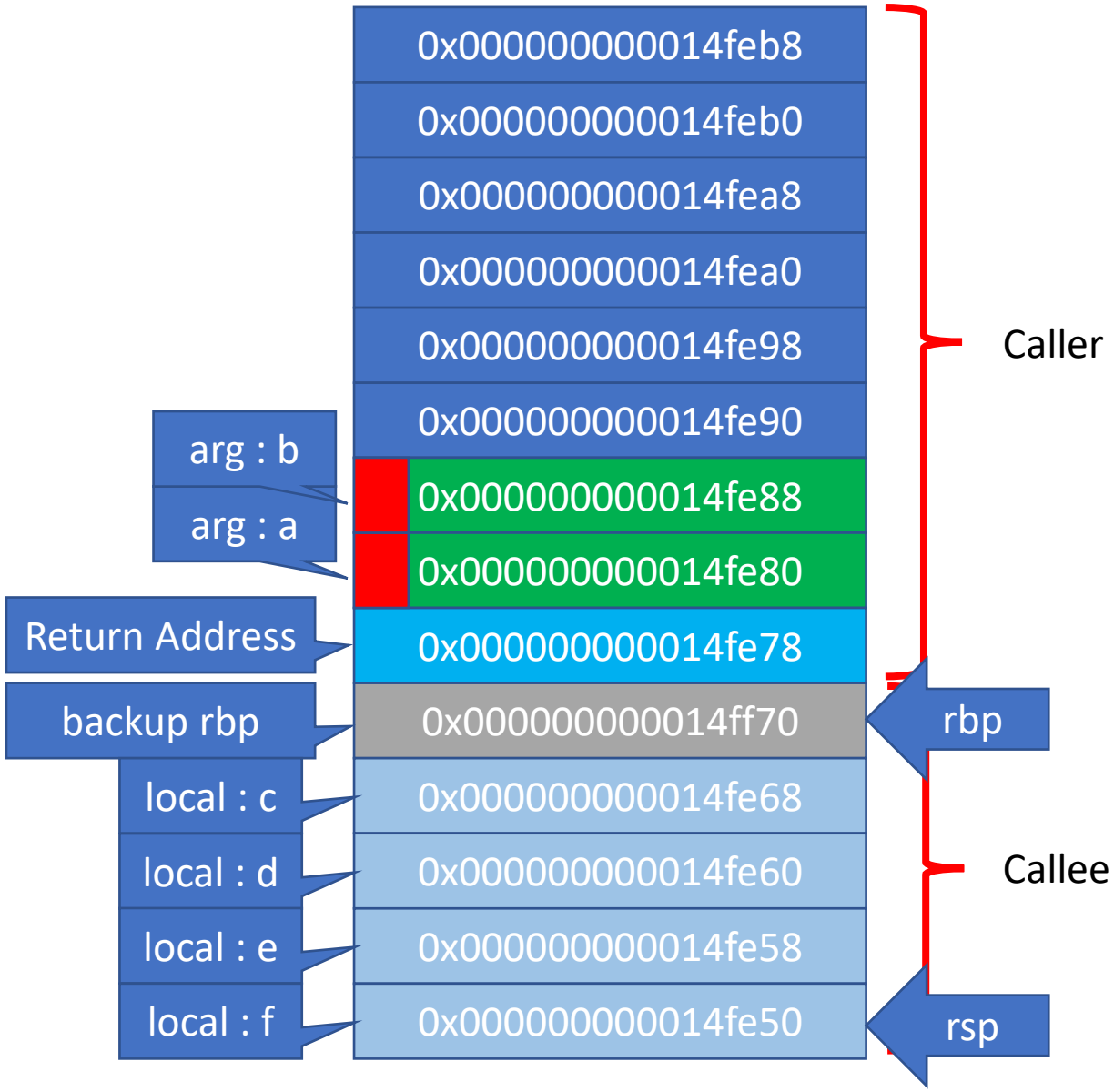
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push         rbp
 mov          rbp,rsp
 add          rsp,0FFFFFFFFFFFFFFE0h
 mov          qword ptr [a],rcx ; rbp+16
```



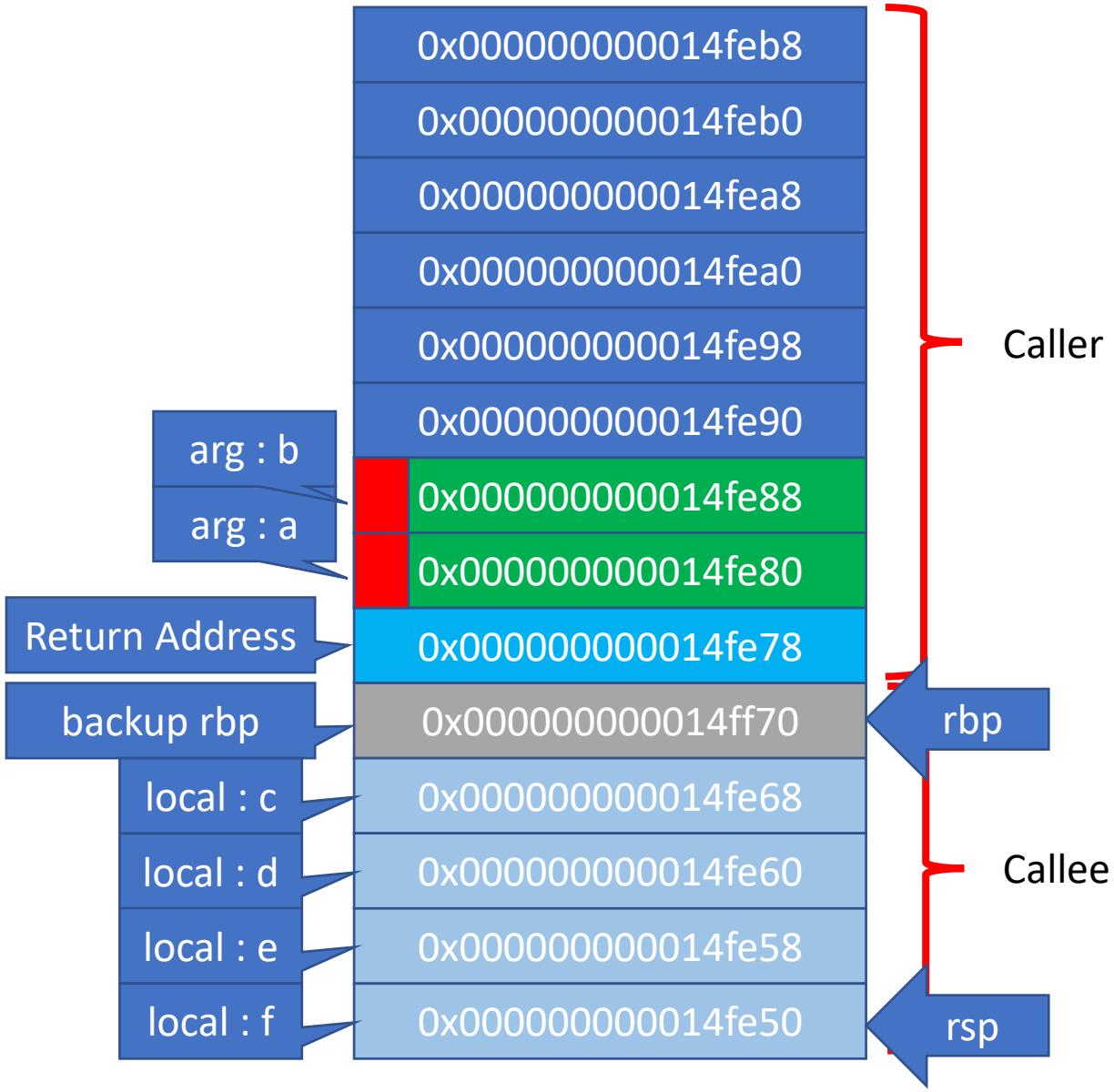
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push         rbp
 mov          rbp,rsp
 add          rsp,0FFFFFFFFFFFFFFE0h
 mov          qword ptr [a],rcx ; rbp+16
 mov          qword ptr [b],rdx ; rbp+24
```



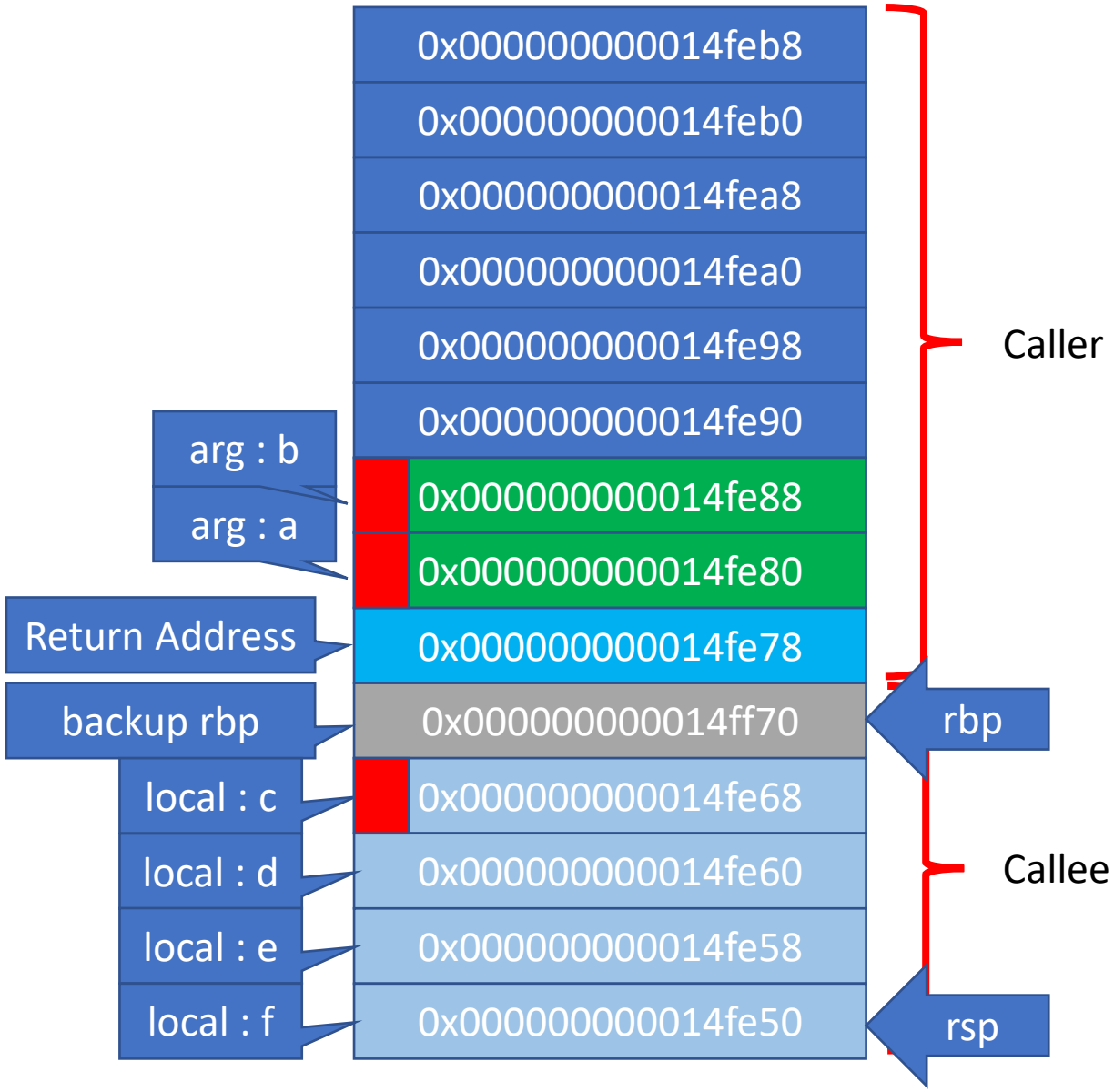
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push         rbp
 mov          rbp,rsp
 add          rsp,0FFFFFFFFFFFFFFE0h
 mov          qword ptr [a],rcx ; rbp+16
 mov          qword ptr [b],rdx ; rbp+24
 add          rcx,rdx
```



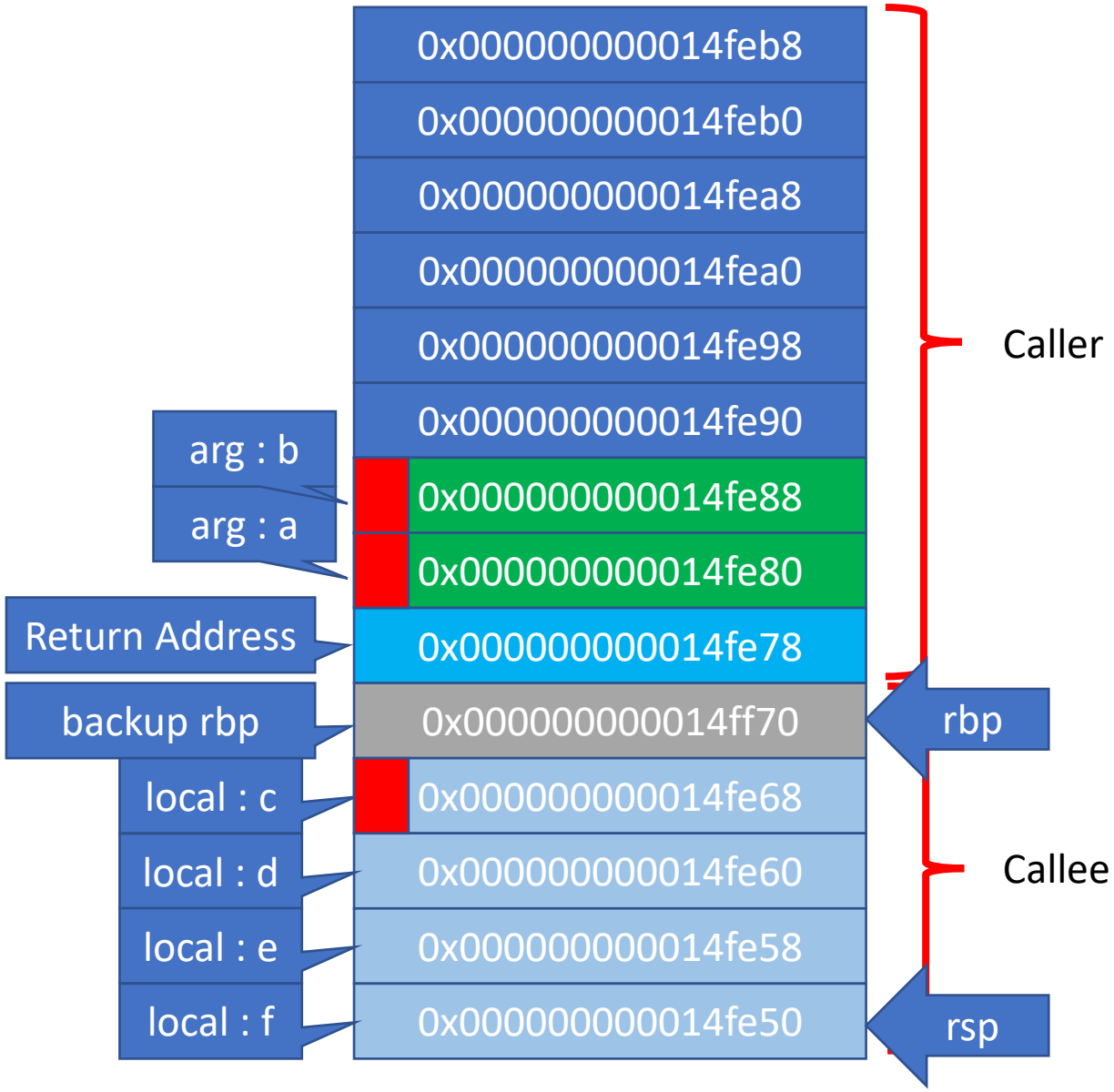
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push         rbp
 mov          rbp,rsp
 add          rsp,0FFFFFFFFFFFFFFE0h
 mov          qword ptr [a],rcx ; rbp+16
 mov          qword ptr [b],rdx ; rbp+24
 add          rcx,rdx
 mov          qword ptr [c],rcx ; rbp-8
```



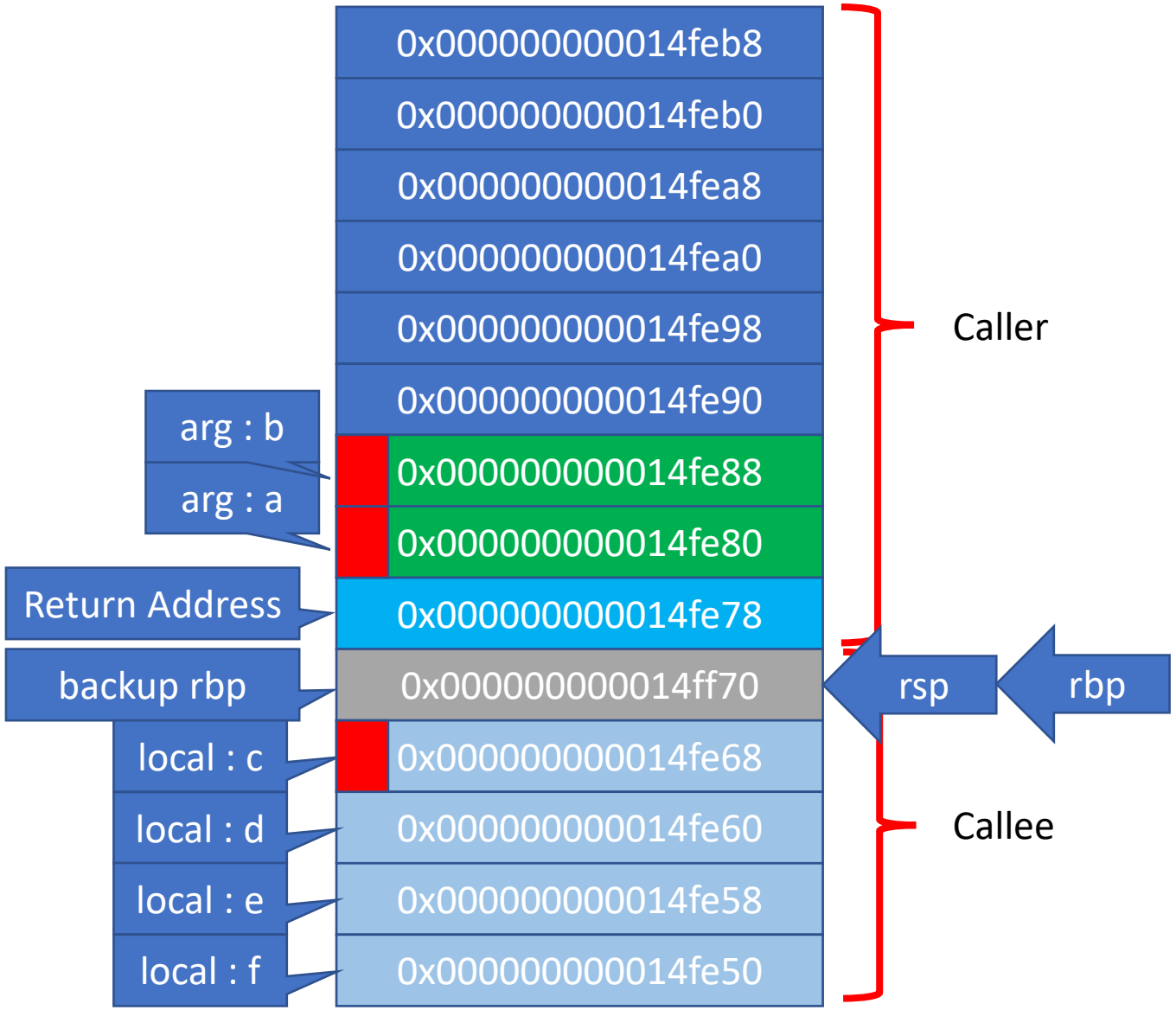
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push         rbp
 mov          rbp,rsp
 add          rsp,0FFFFFFFFFFFFFFE0h
 mov          qword ptr [a],rcx ; rbp+16
 mov          qword ptr [b],rdx ; rbp+24
 add          rcx,rdx
 mov          qword ptr [c],rcx ; rbp-8
 mov          rax,qword ptr [c]
```



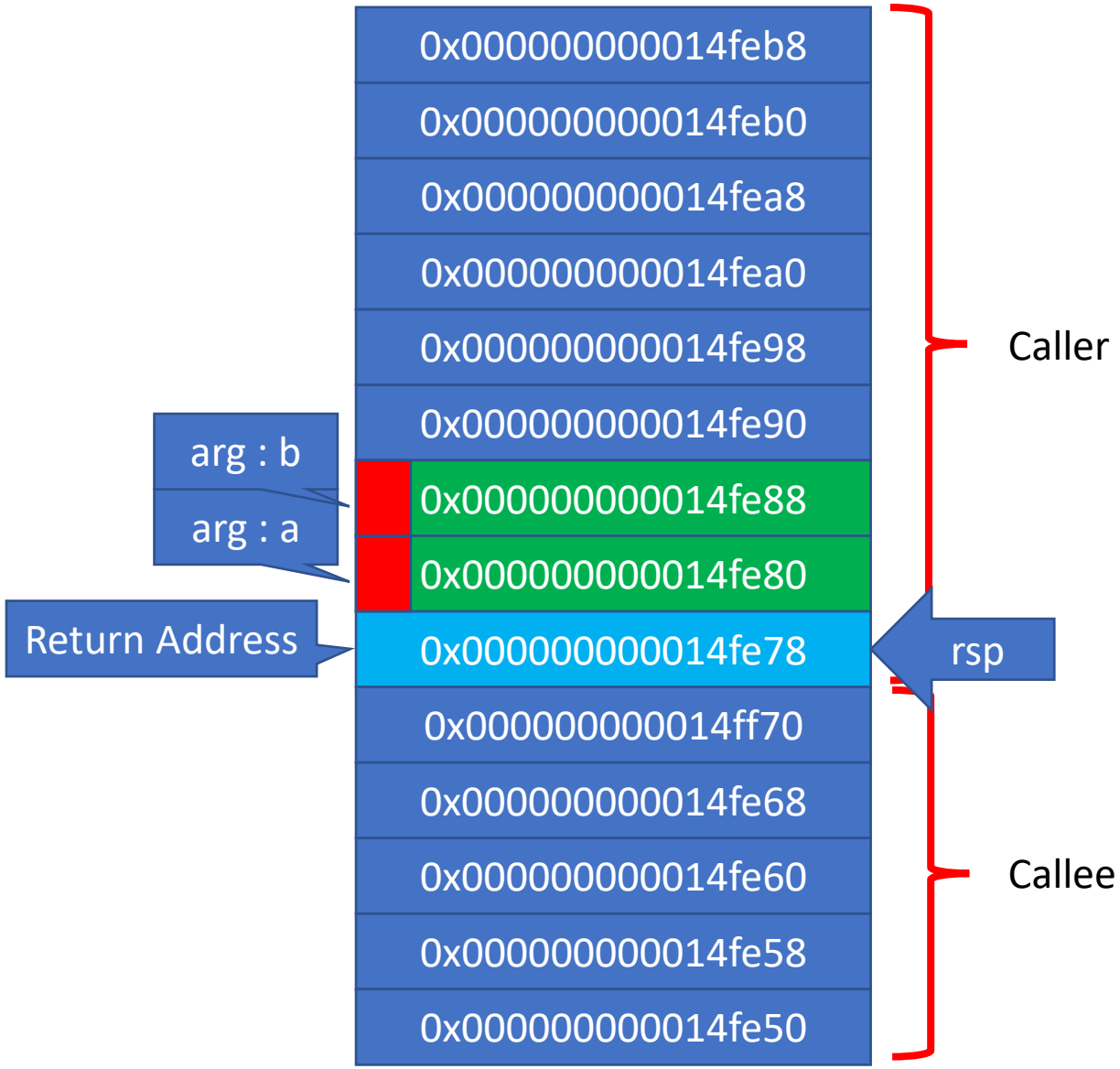
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push         rbp
 mov          rbp,rsp
 add          rsp,0FFFFFFFFFFFFFFE0h
 mov          qword ptr [a],rcx ; rbp+16
 mov          qword ptr [b],rdx ; rbp+24
 add          rcx,rdx
 mov          qword ptr [c],rcx ; rbp-8
 mov          rax,qword ptr [c]
 leave (mov   rsp,rbp)
```



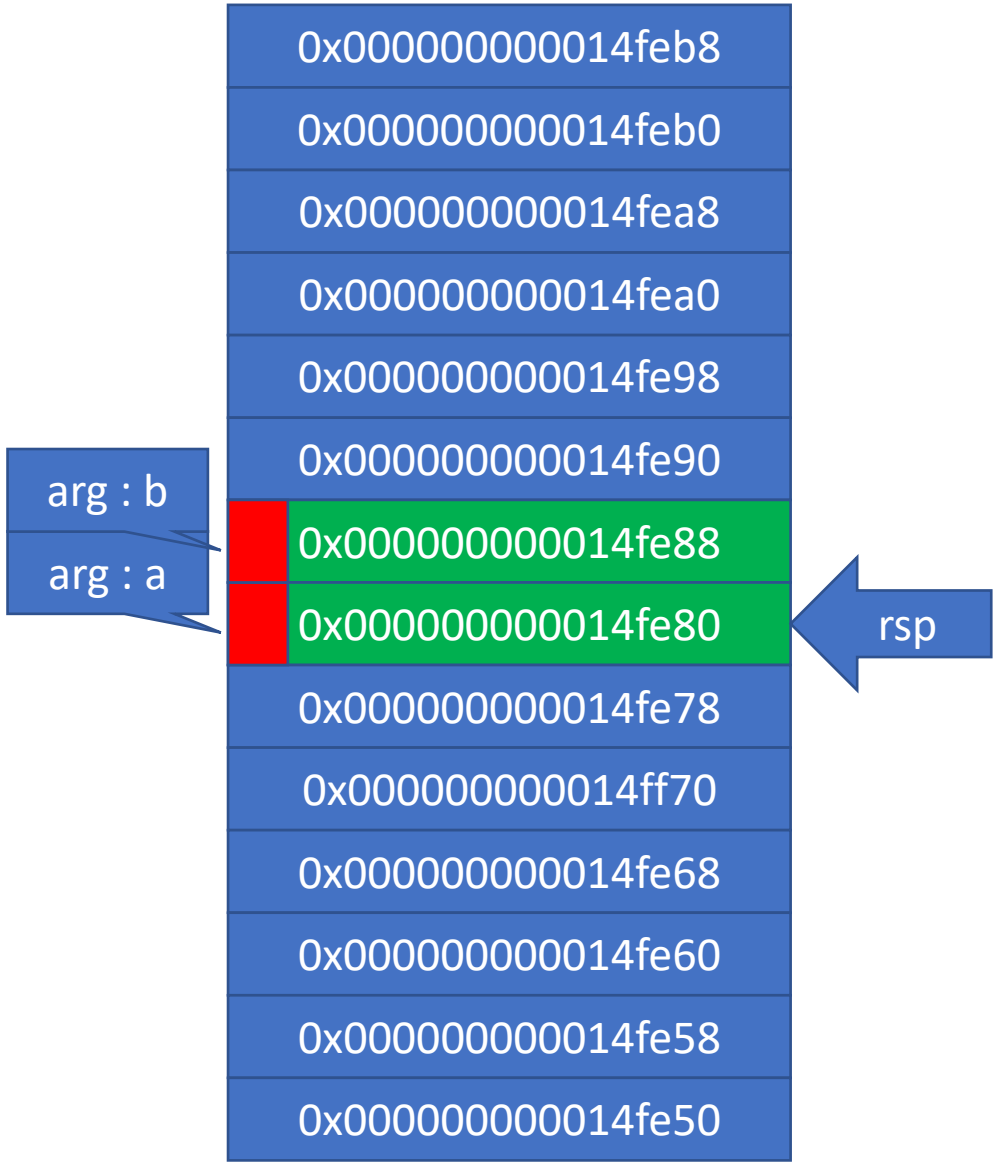
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push         rbp
 mov          rbp,rsp
 add          rsp,0FFFFFFFFFFFFFFE0h
 mov          qword ptr [a],rcx ; rbp+16
 mov          qword ptr [b],rdx ; rbp+24
 add          rcx,rdx
 mov          qword ptr [c],rcx ; rbp-8
 mov          rax,qword ptr [c]
 leave (pop   rbp)
```



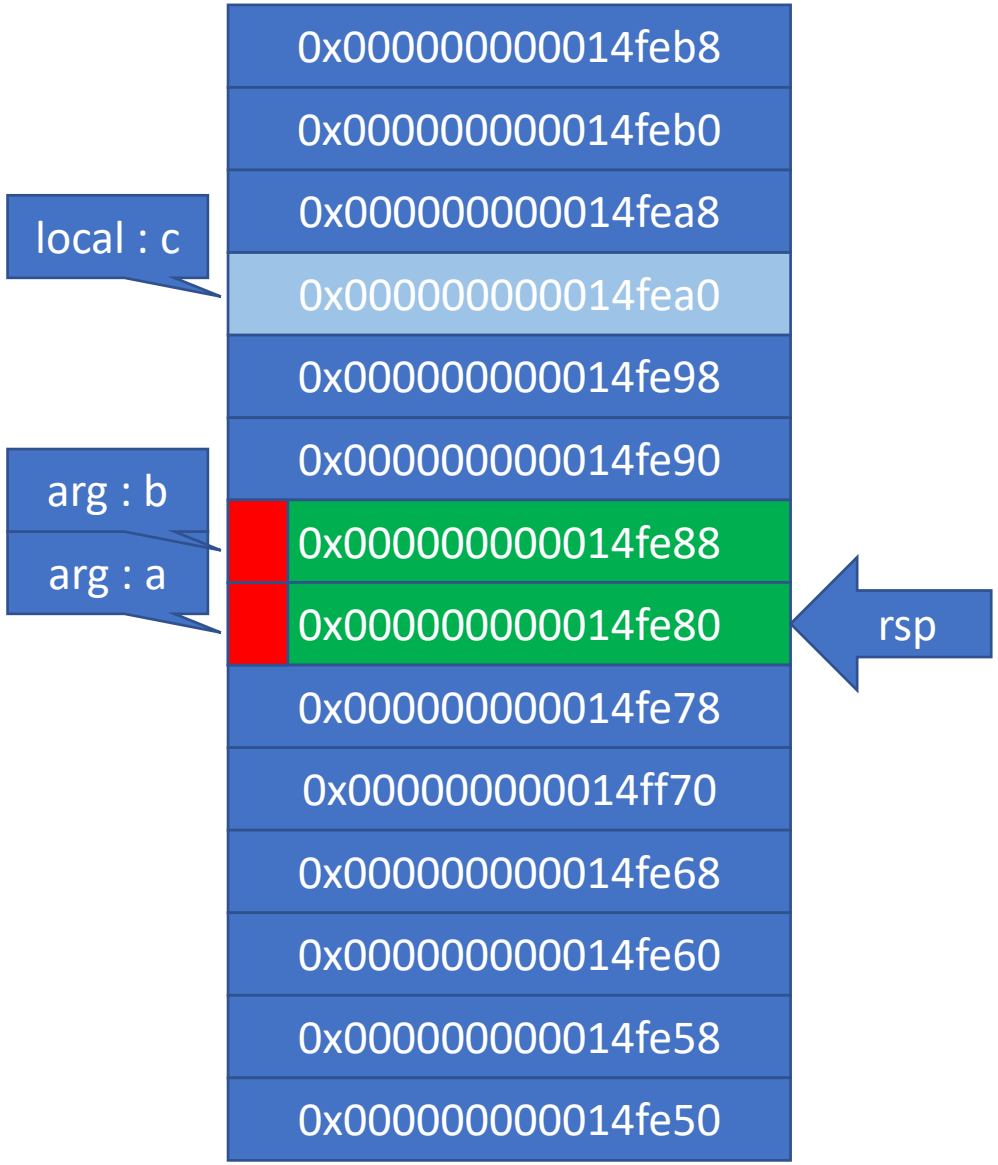
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push         rbp
 mov          rbp,rsp
 add          rsp,0FFFFFFFFFFFFFFE0h
 mov          qword ptr [a],rcx ; rbp+16
 mov          qword ptr [b],rdx ; rbp+24
 add          rcx,rdx
 mov          qword ptr [c],rcx ; rbp-8
 mov          rax,qword ptr [c]
 leave
 ret
```



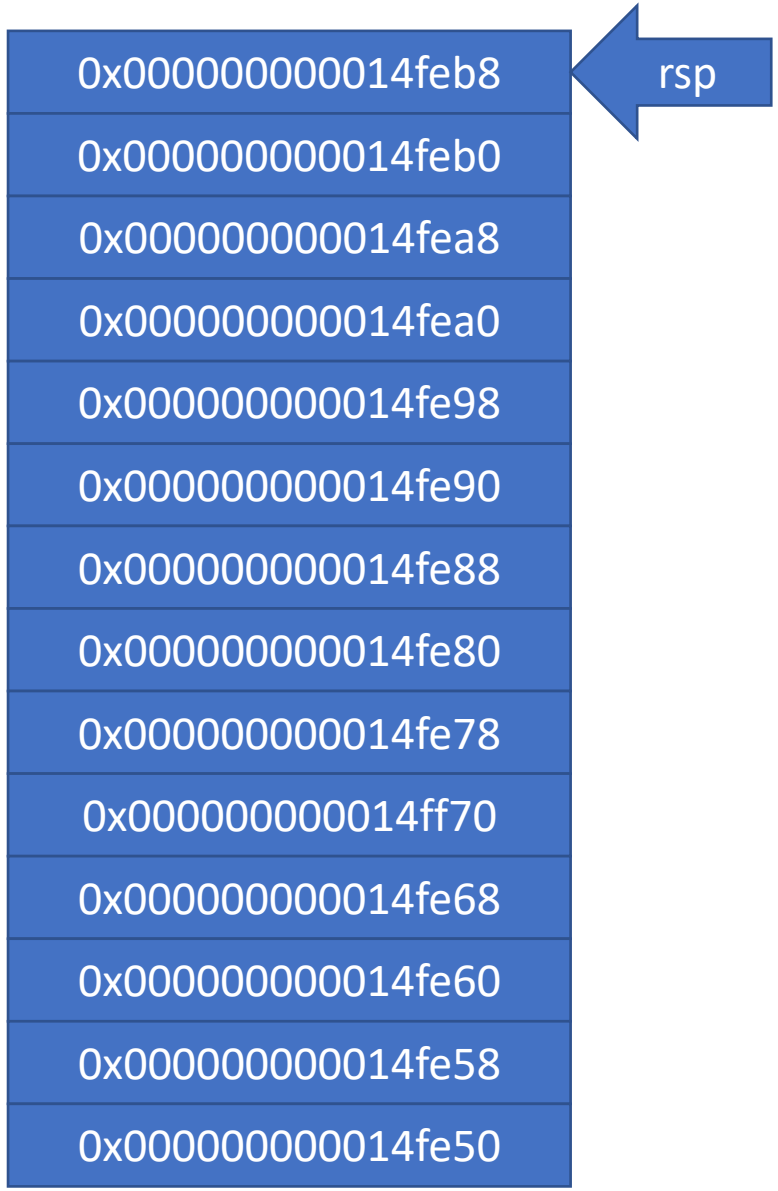
```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
 mov          dword ptr [c],eax
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push         rbp
 mov          rbp,rsp
 add          rsp,0FFFFFFFFFFFFFFE0h
 mov          qword ptr [a],rcx ; rbp+16
 mov          qword ptr [b],rdx ; rbp+24
 add          rcx,rdx
 mov          qword ptr [c],rcx ; rbp-8
 mov          rax,qword ptr [c]
 leave
 ret
```



```
AsmTest64.exe!Test(void):
 sub          rsp,38h
 mov          edx,2
 mov          ecx,1
 call         Add_ASM_RBP
 mov          dword ptr [c],eax
 add          rsp,38h
 ret
```

```
AsmTest64.exe! Add_ASM_RBP(QWORD, QWORD):
 push         rbp
 mov          rbp,rsp
 add          rsp,0FFFFFFFFFFFFFFE0h
 mov          qword ptr [a],rcx ; rbp+16
 mov          qword ptr [b],rdx ; rbp+24
 add          rcx,rdx
 mov          qword ptr [c],rcx ; rbp-8
 mov          rax,qword ptr [c]
 leave
 ret
```



asm에서 다른 함수 호출

1. rcx,rdx,r8,r9에 4개의 파라미터 카피
2. 파라미터 개수 4개 초과일 경우
    1. 추가로 전달할 파라미터 사이즈만큼 RSP레지스터의 값을 빼준다.
    2. RSP의 주소로부터 높은 주소로 진행하며 파라미터를 스택에 카피
3. 파라미터 전달용 스택 메모리(arg home area)를 확보해준다.
    1. arg home area를 위해 sub rsp,32
    2. 호출 전 rsp레지스터의 값이 16의 배수여야 한다.

호출할 함수 선언

- CRT함수나 win32 API함수 호출시 추가적인 선언 필요

```
fscanf      PROTO
strlen      PROTO
memcpy      PROTO
rand        PROTO
MessageBoxA PROTO
```

함수호출 예제

- 16 Byte Align 규칙 위반 테스트

구조체 사용

```
VECTOR4 STRUCT
    x REAL4 ?
    y REAL4 ?
    z REAL4 ?
    w REAL4 ?
VECTOR4 ENDS

VECTOR4_SIZE EQU 16
```

코드 샘플

비교 및 치환

vector x matrix

가변인자 함수 호출

가상함수 호출 분석

Reference

- x64 소프트웨어 규칙 | Microsoft Docs
- Rules and Limitations for Naked Functions | Microsoft Docs
- Introduction to x64 Assembly (intel.com)