Visual Studio를 이용한 어셈블리어 학습 part 1

유영천

https://megayuchi.com

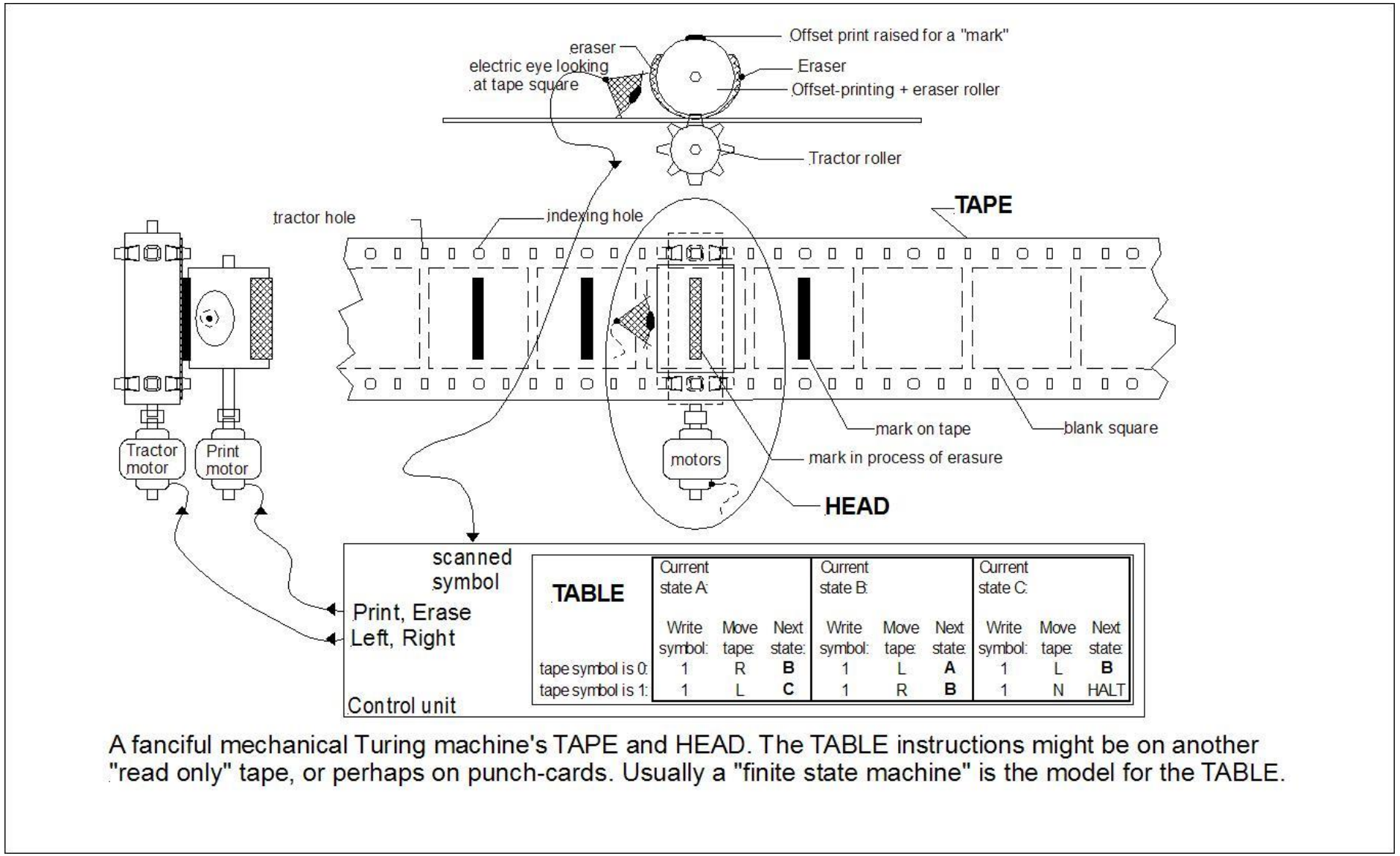
tw: @dgtman

목표

- 직접 코딩할 일은 사실 많지 않다.
- 코드의 작동원리를 알아보자.

CPU기본

Turing machine



A fanciful mechanical Turing machine's TAPE and HEAD. The TABLE instructions might be on another "read only" tape, or perhaps on punch-cards. Usually a "finite state machine" is the model for the TABLE.

https://en.wikipedia.org/wiki/Turing_machine_gallery#/media/File:Turing_machine_1.JPG

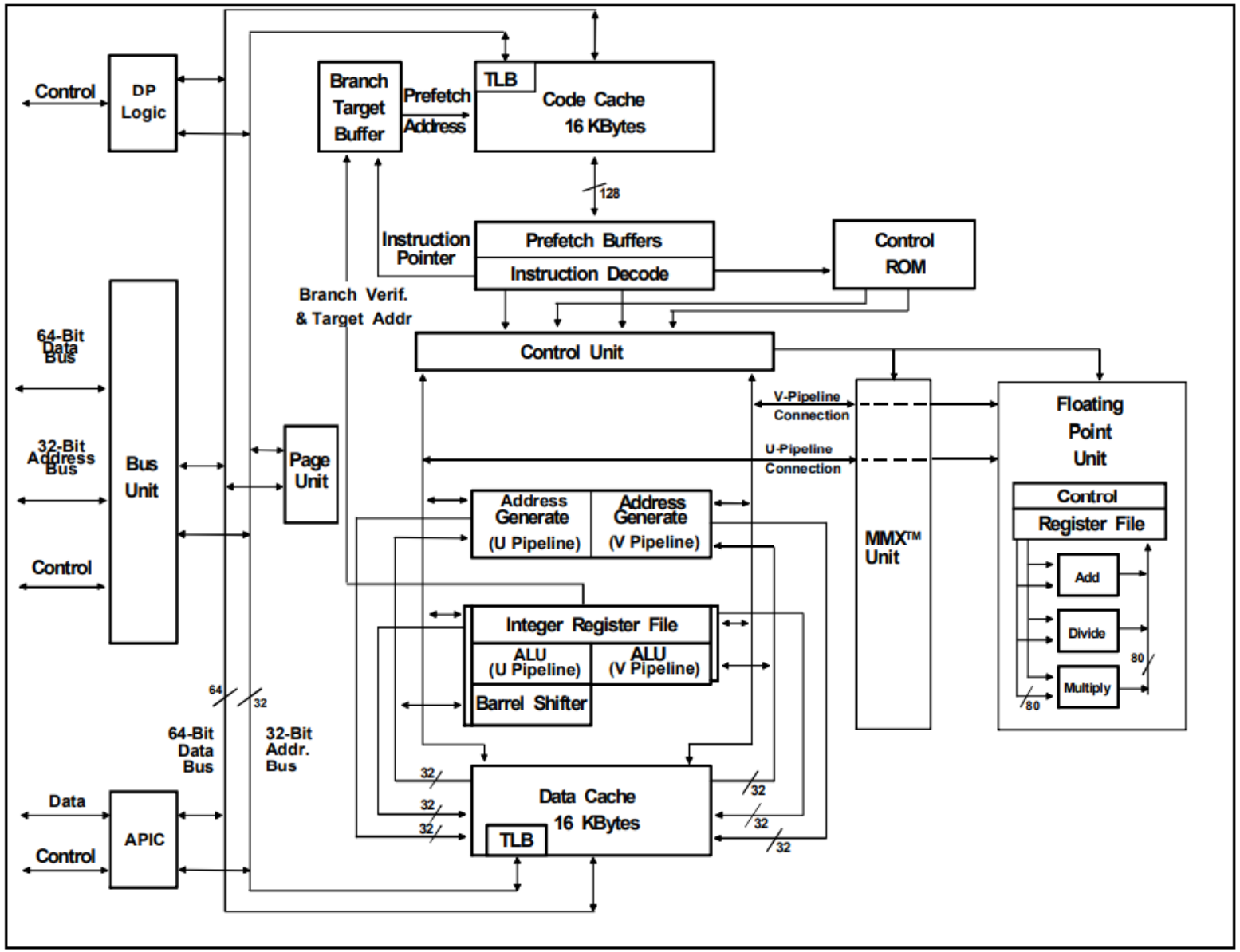


Figure 1. Pentium® Processor with MMX™ Technology Block Diagram

기계어

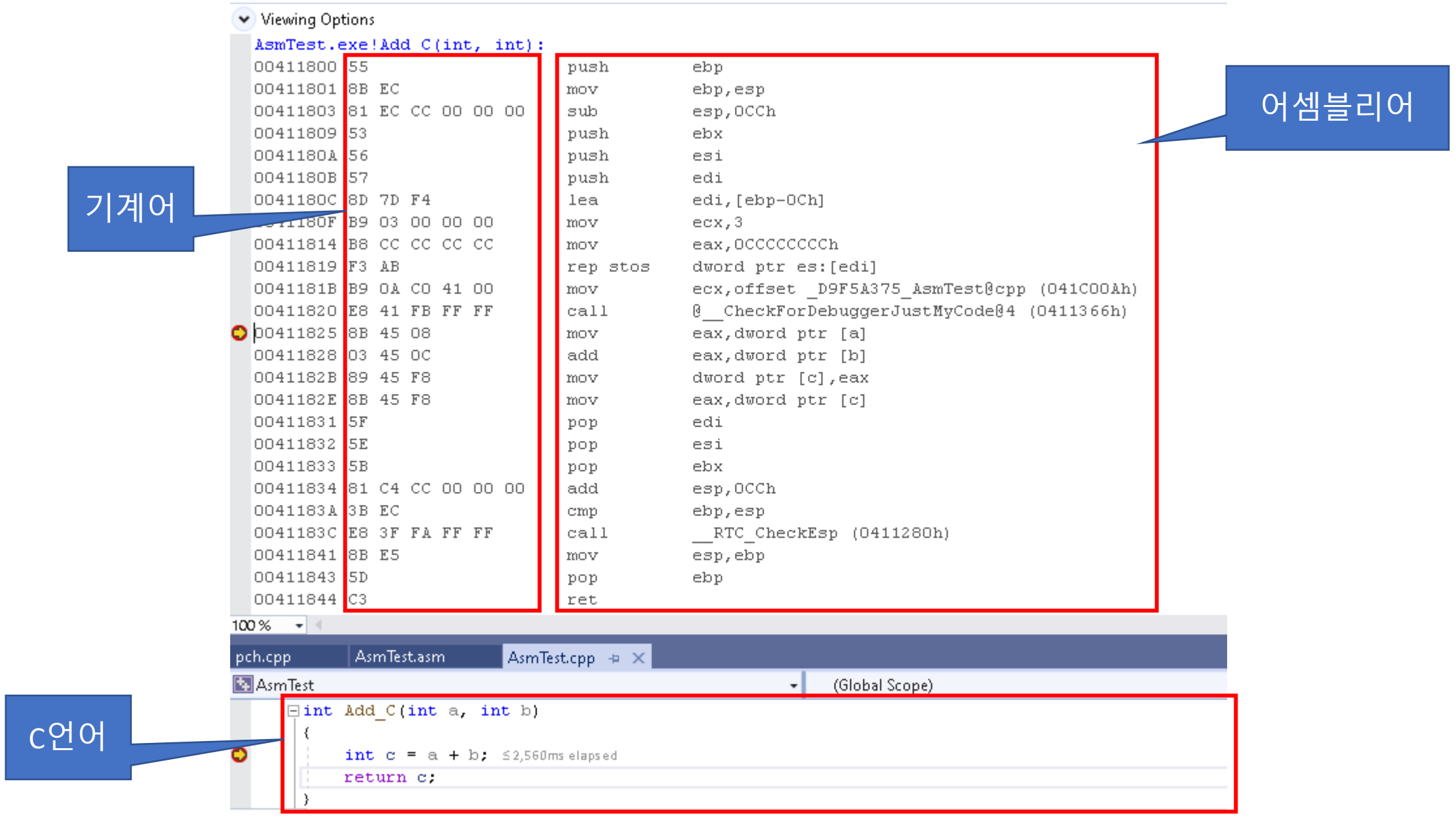
- 명령어와 데이터의 스트림
- 기계어와 어셈블리어는 1:1 대응

어셈블리어

- 기계어와 1:1 대응
- 비교적 사람이 보기 편한 문자로 구성
- 기계어와 어셈블리어의 성능 차이는 없다.

고급언어(C/C++)

- 고급언어 -> 어셈블리어
- 고급언어와 어셈블리어는 1:1 대응되지 않는다.
- C는 꽤~ 1:1대응에 가깝다.
- C++로 작성하는 경우도 스타일에 따라 꽤 1:1대응에 가깝다.
- C or C스타일의 C++코드는 플랫폼 독립적인 어셈블리어라고도 한다.
- 어셈블리어의 성능/저수준 제어 + 이식성이 필요할때 C로 작성한다.


Viewing Options
AsmTest.exe!Add_C(int, int):
00411800 55
00411801 8B EC
00411803 81 EC CC 00 00 00
00411809 53
0041180A 56
0041180B 57
0041180C 8D 7D F4
0041180F B9 03 00 00 00
00411814 B8 CC CC CC CC
00411819 F3 AB
0041181B B9 0A C0 41 00
00411820 E8 41 FB FF FF
00411825 8B 45 08
00411828 03 45 0C
0041182B 89 45 F8
0041182E 8B 45 F8
00411831 5F
00411832 5E
00411833 5B
00411834 81 C4 CC 00 00 00
0041183A 3B EC
0041183C E8 3F FA FF FF
00411841 8B E5
00411843 5D
00411844 C3
push ebp
mov ebp,esp
sub esp,0CCh
push ebx
push esi
push edi
lea edi,[ebp-0Ch]
mov ecx,3
mov eax,0CCCCCCCCh
rep stos dword ptr es:[edi]
mov ecx,offset _D9F5A375_AsmTest@cpp (041C00Ah)
call @__CheckForDebuggerJustMyCode@4 (0411366h)
mov eax,dword ptr [a]
add eax,dword ptr [b]
mov dword ptr [c],eax
mov eax,dword ptr [c]
pop edi
pop esi
pop ebx
add esp,0CCh
cmp ebp,esp
call __RTC_CheckEsp (0411280h)
mov esp,ebp
pop ebp
ret
100 %
pch.cpp
AsmTest.asm
AsmTest.cpp
AsmTest
(Global Scope)
int Add_C(int a, int b)
{
int c = a + b; ≤2,560ms elapsed
return c;
}
기계어
어셈블리어
C언어

어셈블리어 기본

Register

컴퓨터의 프로세서 내에서 자료를 보관하는 아주 빠른 기억 장소이다.

일반적으로 현재 계산을 수행중인 값을 저장하는 데 사용된다.

대부분의 현대 프로세서는 메인 메모리에서 레지스터로 데이터를 옮겨와 데이터를 처리한 후 그 내용을 다시 레지스터에서 메인 메모리로 저장하는 로드-스토어 설계를 사용하고 있다.

프로세서 레지스터 - 위키백과, 우리 모두의 백과사전 (wikipedia.org)

X86의 주요 레지스터(user모드 기준)

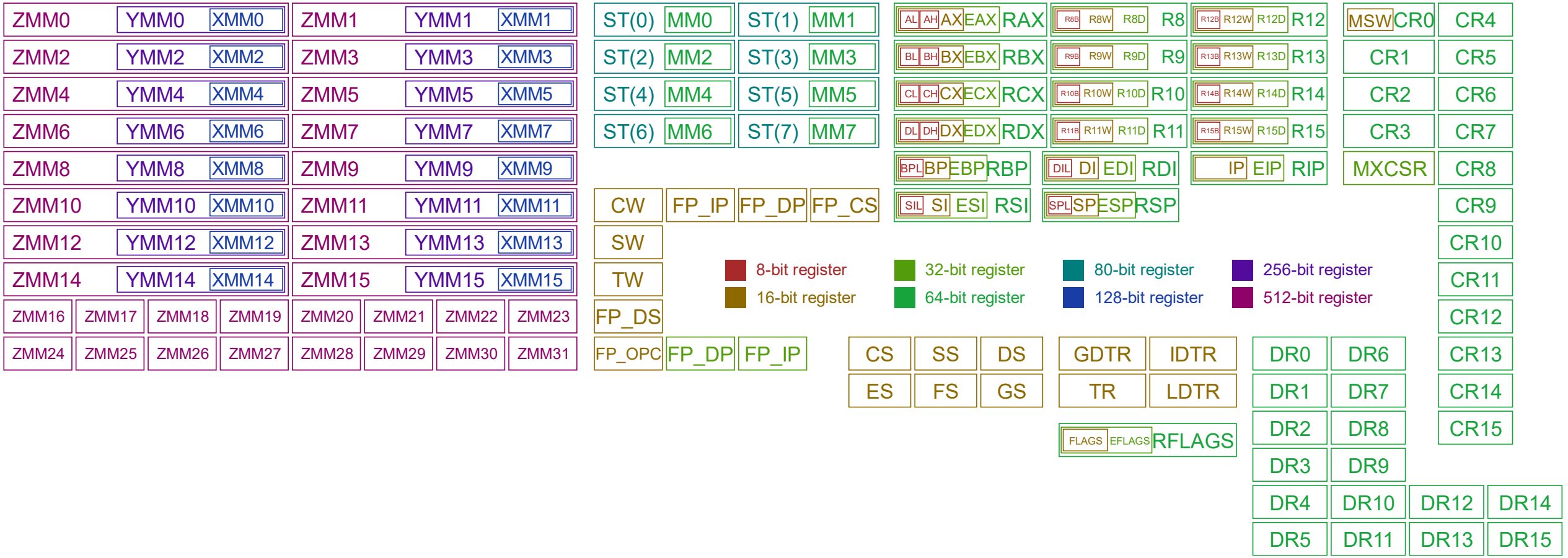
- 범용 레지스터
  - EAX,EBX,ECX,EDX
- 주소지정 레지스터
  - ESI, EDI
- Flags register – eflags
- 프로그램 카운터 – EIP
- 스택 포인터 – ESP
- 스택 프레임 베이스 포인터 - EBP

x86(x64) registers

Register	Accumulator	Counter	Data	Base	Stack Pointer	Stack Base Pointer	Source	Destination
64-bit	RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI
32-bit	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI
16-bit	AX	CX	DX	BX	SP	BP	SI	DI
8-bit	AH, AL	CH, CL	DH, DL	BH, BL	SPL	BPL	SIL	DIL

https://en.wikibooks.org/wiki/X86_Assembly/X86_Architecture

x86(x64) registers



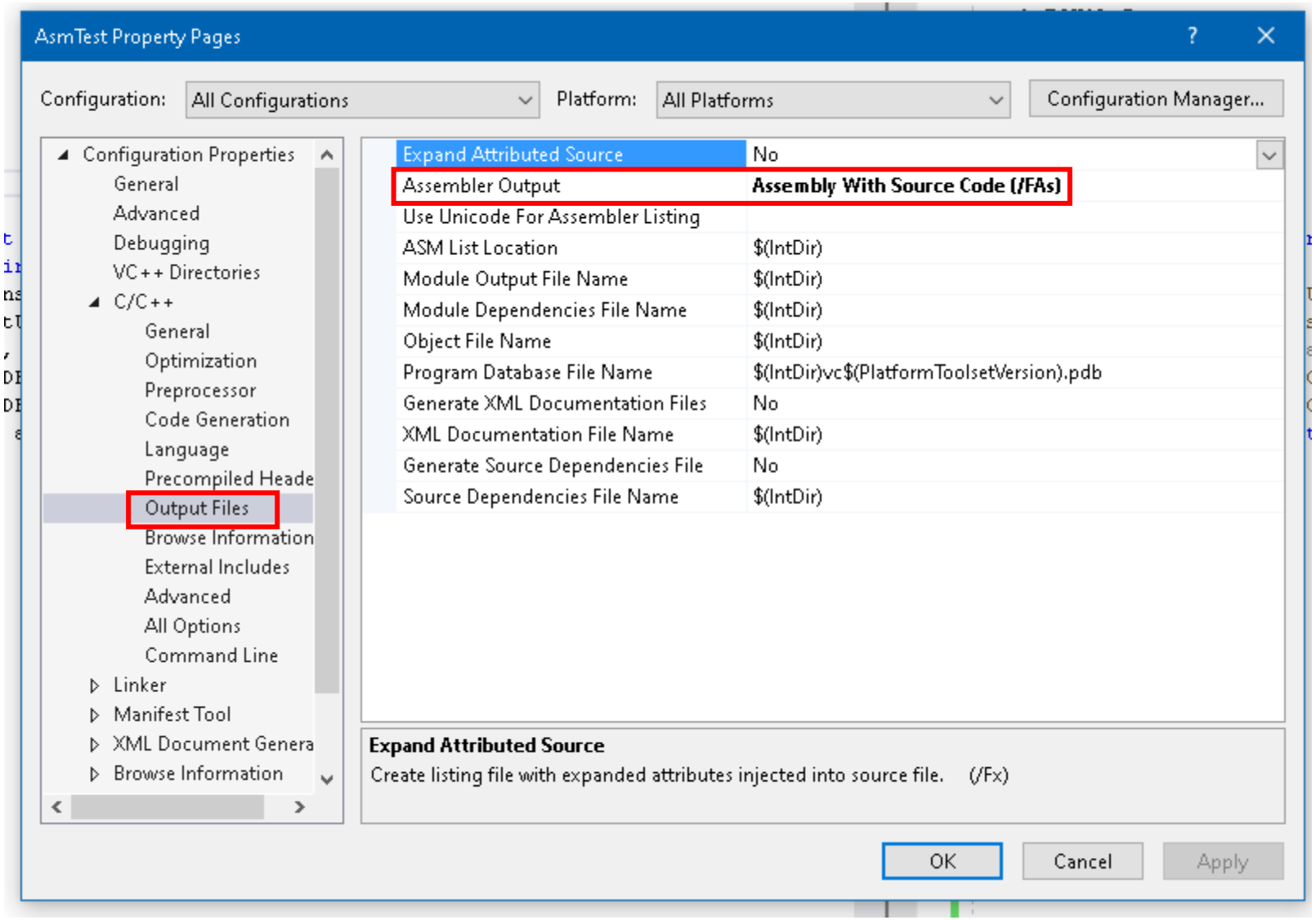
https://en.wikipedia.org/wiki/X86#/media/File:Table_of_x86_Registers_svg.svg

C코드의 디스어셈블리 분석

Listing File 활용

```
int main()
{
    __asm
    {
        mov eax,INT_MAX
        mov edx,INT_MIN
        sub eax,edx
        nop
    }
    int a = -1;
    int b = 0;
    int c = 0;
    c = Add_C(a, b);
    //c = Add_CDECL(a, b);
    //c = Add_NakedCall_CDECL(a, b);
    //c = Add_CDECL_L(a, b);
    c = Add_NakedCall_CDECL_LOCAL_VAR(a, b);
```

```
193  _main   PROC                          ; COMDAT
194  ; File C:\DEV\DAIKON_ROOT\BreadBoard\AsmTest\AsmTest.cpp
195  ; Line 30
196      push    ebp
197      mov ebp, esp
198      sub esp, 32                 ; 00000020H
199  ; Line 33
200      mov eax, 2147483647             ; 7fffffffH
201  ; Line 34
202      mov edx, -2147483648            ; 80000000H
203  ; Line 35
204      sub eax, edx
205  ; Line 36
206      npad    1
207  ; Line 45
208      push    0
209      push    -1
210      mov DWORD PTR _a$[ebp], -1
211      mov DWORD PTR _b$[ebp], 0
212      call    ?Add_NakedCall_CDECL_LOCAL_VAR@@YAHHH@Z ; Add_NakedCall_CDECL_LOCAL_VAR
213  ; Line 51
214      push    OFFSET ??_C@_1BK@KAFNHOOO@?$AAS?$AAi?$AAg?$AAn?$AAe?$AAd?$AA?5?$AAT?$AAe?$AAs?$AAt?$AA?6@
215      mov DWORD PTR _wchInequality$[ebp], OFFSET ??_C@_15PEJIGKFD@?$AA?$DN?$AA?$DN@
216      mov DWORD PTR _wchInequality$[ebp+4], OFFSET ??_C@_13MOEPKPHB@?$AA?$DO@
217      mov DWORD PTR _wchInequality$[ebp+8], OFFSET ??_C@_13GEEGGHPK@?$AA?$DM@
218      mov DWORD PTR _wchInequality$[ebp+12], OFFSET ??_C@_13MOEPKPHB@?$AA?$DO@
219      mov DWORD PTR _wchInequality$[ebp+16], OFFSET ??_C@_13GEEGGHPK@?$AA?$DM@
220      call    _wprintf_s
221      add esp, 12                 ; 0000000cH
```


AsmTest Property Pages
Configuration: All Configurations
Platform: All Platforms
Configuration Manager...
Configuration Properties
General
Advanced
Debugging
VC++ Directories
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Heade
Output Files
Browse Information
External Includes
Advanced
All Options
Command Line
Linker
Manifest Tool
XML Document Genera
Browse Information
Expand Attributed Source
No
Assembler Output
Assembly With Source Code (/FAs)
Use Unicode For Assembler Listing
ASM List Location
$(IntDir)
Module Output File Name
$(IntDir)
Module Dependencies File Name
$(IntDir)
Object File Name
$(IntDir)
Program Database File Name
$(IntDir)vc$(PlatformToolsetVersion).pdb
Generate XML Documentation Files
No
XML Documentation File Name
$(IntDir)
Generate Source Dependencies File
No
Source Dependencies File Name
$(IntDir)
Expand Attributed Source
Create listing file with expanded attributes injected into source file. (/Fx)
OK
Cancel
Apply

Visual Studio에서 Inline 어셈블리 코드 작성

- C/C++ 함수 안에서 __asm {}블록 안에 코딩
- 훌륭한 VS 디버거의 도움으로 무척 쉽게 작성 가능.
- C/C++함수로 전달받은 파라미터, 로컬 변수 그대로 asm코드에서 사용 가능.
- naked call이 아니라면(기본상태) 레지스터 보호 필요 없다.
- X64는 사용불가
  - .asm파일로 작성하고 VS로 드래그앤 드롭하면 ml64로 어셈블 가능.

데이터 전송

- 메모리 -> 레지스터
- 레지스터 -> 메모리
- 메모리 -> 메모리

데이터 전송

- mov
  - 같은 사이즈의 레지스터와 메모리간 카피
- movzx
  - 1 byte -> 2, 2 bytes -> 4 bytes 로 카피하되 빈영역을 0으로 채움
- movsx
  - 1 byte -> 2, 2 bytes -> 4 bytes 로 카피하되 빈영역을 0으로 채우고 부호를 유지함
- movs
  - 레지스터 ESI = src, 레지스터 EDI = dest일때 메모리 -> 메모리로 카피
  - movsb / movsw / movsd / movsq 각각 사이즈별로 사용

```
__declspec(align(16)) char szSrc[64] = "ABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEFGHABCDEF.";
__declspec(align(16)) char szDest[64] = {};
__asm
{
    lea esi, dword ptr[szSrc]
    lea edi, dword ptr[szDest]

    ; copy 1byte, szDest[0] = szSrc[0]
    mov al, byte ptr[esi]
    mov byte ptr[edi], al

    ; copy 2bytes, szDest[0 - 1] = szSrc[0 - 1]
    mov ax, word ptr[esi]
    mov word ptr[edi], ax

    ; copy 4bytes, szDest[0 - 3] = szSrc[0 - 3]
    mov eax, dword ptr[esi]
    mov dword ptr[edi], eax

    ; copy 4bytes * 16 , szDest[0 - 63] = szSrc[0 - 63]
    mov ecx, 64
    shr ecx,2
    rep movsd
    nop
}
```

산술연산

- add
- sub
- inc
- dec
- mul / imul
- div / idiv

산술연산

```
int __declspec(naked) __cdecl Mul_ASM(int a, int b)
{
    __asm
    {
        push ebp
        mov ebp,esp

        mov eax,dword ptr[a]
        mov ecx,dword ptr[b]

        imul ecx

        mov esp,ebp
        pop ebp

        ret
    }
}
```

곱셈

```
int __declspec(naked) __cdecl Div_ASM(int a, int b)
{
    __asm
    {
        push ebp
        mov ebp,esp

        mov eax,dword ptr[a]
        mov ecx,dword ptr[b]
        idiv ecx

        mov esp,ebp
        pop ebp

        ret
    }
}
```

나눗셈

비교분기

- cmp 명령 사용 후 flags register의 내용에 따라 조건 분기
- cmp -> destination의 값을 변경하지 않는 sub와 같다.

```
__asm
{
    mov eax, dword ptr[a]
    mov edx, dword ptr[b];
    cmp eax, edx
    je lb_a_equal_b
    ja lb_a_above_b
    jb lb_a_below_b
    int 3

lb_a_equal_b:
    mov dword ptr[r], A_EQUAL_B
    jmp lb_exit

lb_a_above_b :
    mov dword ptr[r], A_ABOVE_B
    jmp lb_exit

lb_a_below_b :
    mov dword ptr[r], A_BELOW_B
    jmp lb_exit

lb_exit :
    nop

}
```

flags register – eflags/rflags

- 연산 결과를 저장하거나 연산의 일부 옵션을 지정 가능한 레지스터
- User모드에서 사용 가능한 항목은 몇 개 안됨.

Visual Studio의 registers 윈도우

Flag	Set value
Overflow (OF)	OV = 1
Direction	UP = 1
Interrupt	EI = 1
Sign (SF)	PL = 1
Zero (ZF)	ZR = 1
Auxiliary carry (AF)	AC = 1
Parity (PF)	PE = 1
Carry (CF)	CY = 1

비교 분기 코드

```
CMP_RESULT CmpTestSigned(int a, int b)
{
    CMP_RESULT r = A_EQUAL_B;
    // signed - (a > b) -> jg (ZF = 0 and SF = OF)
    // signed - (a < b) -> jl (SF <> OF)
    // signed - (a == b) -> je (ZF = 1)
    __asm
    {
        mov eax, dword ptr[a]
        mov edx, dword ptr[b];
        cmp eax, edx
        je lb_a_equal_b
        jg lb_a_greator_b
        jl lb_a_less_b
        int 3

    lb_a_equal_b:
        mov dword ptr[r], A_EQUAL_B
        jmp lb_exit

    lb_a_greator_b :
        mov dword ptr[r], A_GREATOR_B
        jmp lb_exit

    lb_a_less_b :
        mov dword ptr[r], A_LESS_B
        jmp lb_exit

    lb_exit :
        nop
    }
    return r;

}
```

함수호출

- call addr
- call [variable/register]

함수호출

- 직접 호출

정적 링크된 함수들

```
__asm
{
    lea eax, dword ptr[src]
    lea edx, dword ptr[dest]
    push 64;
    push eax
    push edx

    call memcpy
    add esp,12
}
```

- 간접 호출

함수 포인터/DLL함수등(win32 API)

```
__asm
{
    mov eax, dword ptr[wchCaption]
    mov edx, dword ptr[wchText]
    push MB_OK
    push eax
    push edx
    push 0
    call dword ptr[MessageBox]
}
```

비트연산

- and
- or
- Test
  - Flags레지스터만 갱신, dest를 변경하지 않음.
- shl
- shr

Calling convention

- cdecl
  - C표준 calling convention.
  - 인자는 뒤에서부터 push
  - 인자를 넣느라 변경한 sp레지스터는 caller쪽에서 복구
- stdcall
  - Pascal 기본 calling convention, win32에서 기본 calling convection
  - 인자는 뒤에서부터 push
  - 인자를 넣느라 변경한 sp레지스터는 callee쪽에서 복구
- thiscall
  - cdecl과 기본 같다. this포인터를 cx레지스터로 전달.
- fastcall
  - 인자를 전달할때 cx,dx레지스터를 우선 사용
- fastcall(x64)
  - x64에선 기본 calling convention
  - rcx,rdx, r8, r9 레지스터를 우선사용

Stack frame

- 완전히 똑같지는 않지만 C에서의 함수 블록 {}에 해당하는 스택 영역
- 함수 진입시 스택 메모리를 확보
- 함수에서 나갈때 스택 메모리를 해제

```
int __declspec(naked) __cdecl Add_NakedCall_CDECL(int a, int b)
{
    __asm
    {
        push ebp
        mov ebp, esp

        mov eax,dword ptr[a]
        mov edx,dword ptr[b]
        add eax,edx

        mov esp,ebp
        pop ebp
        ret
    }
}
```

Prologue

epilogue

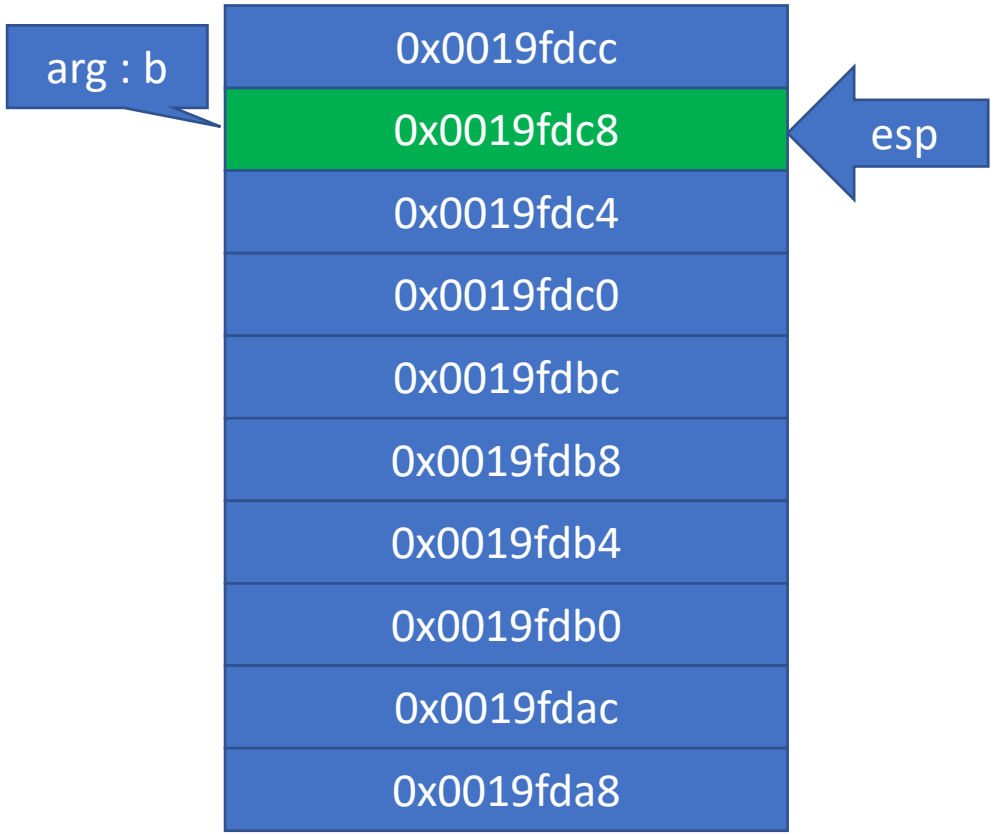
```
int __declspec(naked) __cdecl Add_NakedCall_CDECL(int a, int b)
{
    __asm
    {
        push ebp
        mov ebp, esp

        mov eax,dword ptr[a]
        mov edx,dword ptr[b]
        add eax,edx

        mov esp,ebp
        pop ebp
        ret
    }
}
```

Stack frame

```
mov         eax,dword ptr [b]
push        eax
mov         ecx,dword ptr [a]
push        ecx
call        Add_NakedCall_CDECL
add         esp,8
```

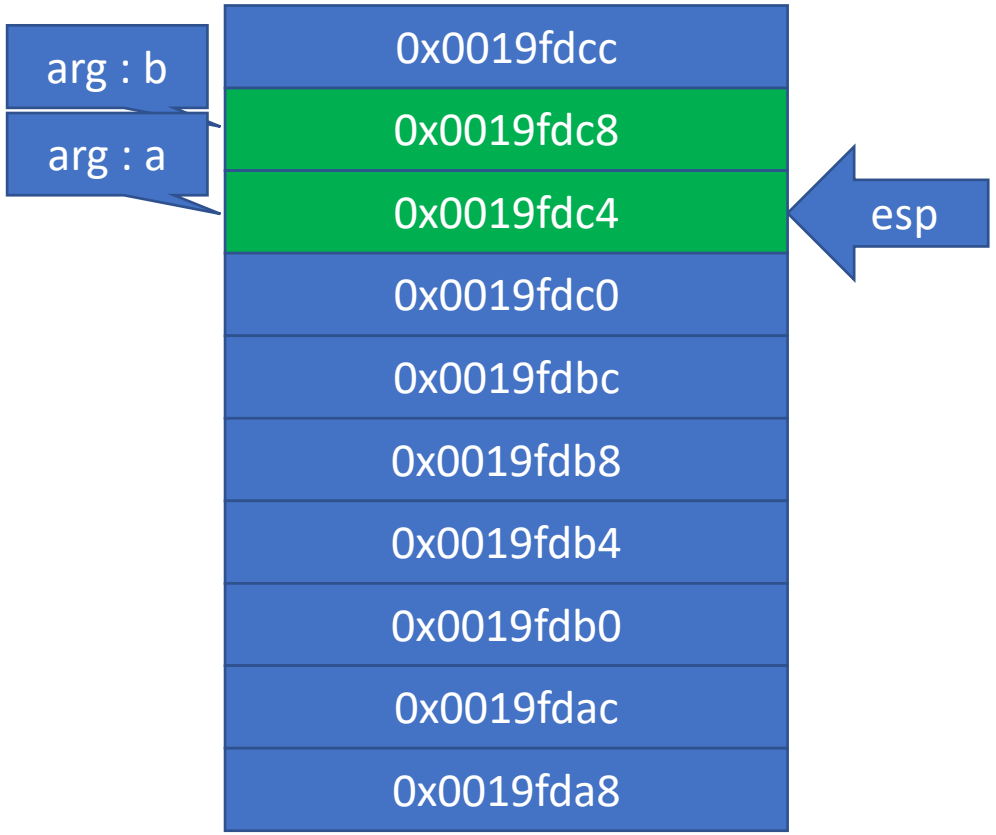
Stack frame

```
mov         eax,dword ptr [b]
push        eax
```



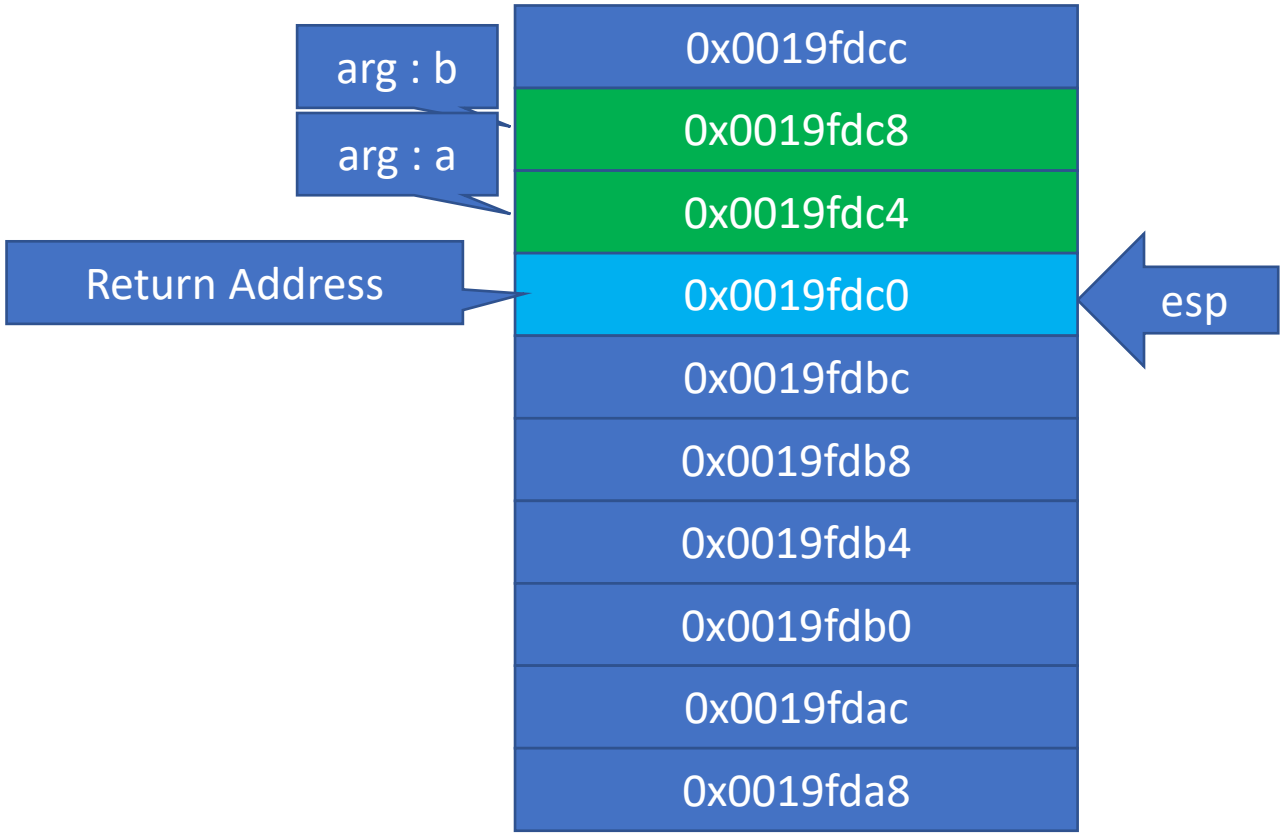
Stack frame

```
mov         eax,dword ptr [b]
push        eax
mov         ecx,dword ptr [a]
push        ecx
```



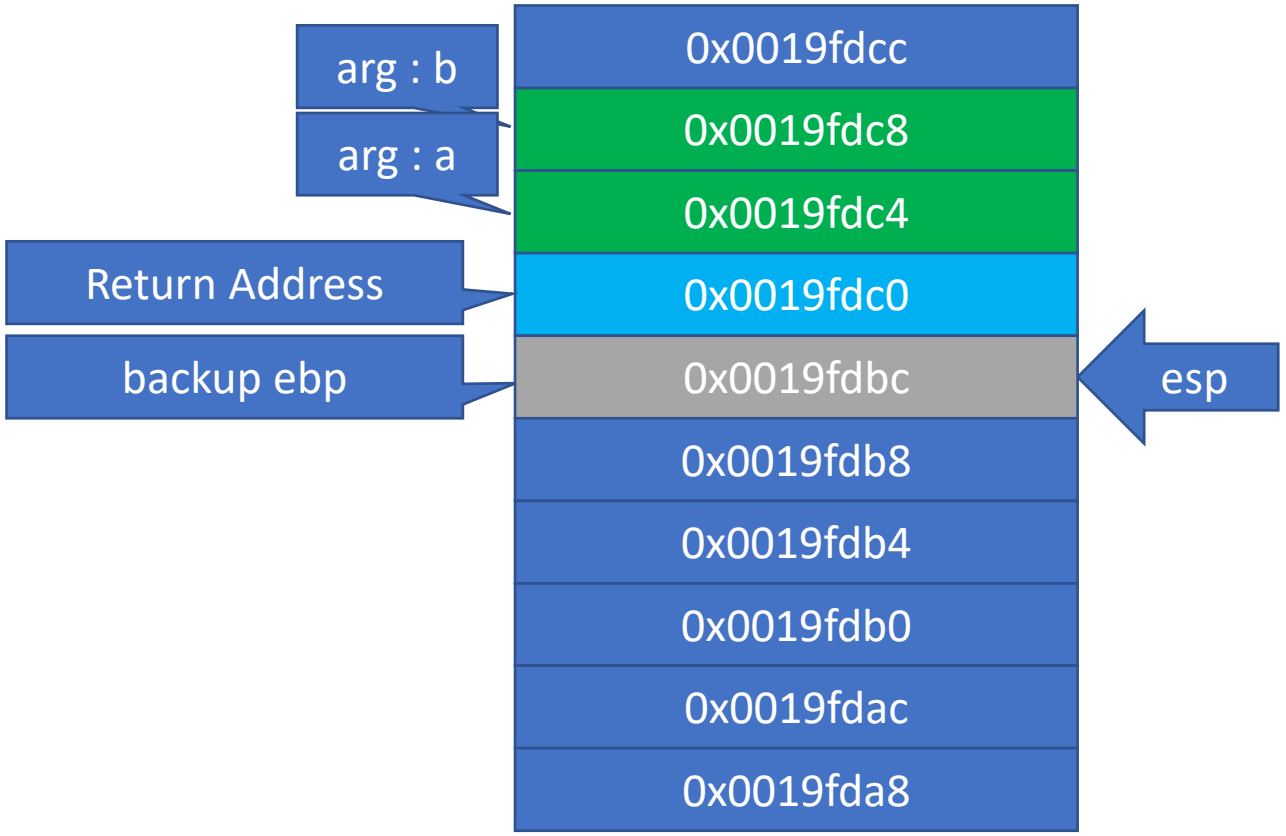
Stack frame

```
mov           eax,dword ptr [b]
push          eax
mov           ecx,dword ptr [a]
push          ecx
call          Add_NakedCall_CDECL

Add_NakedCall_CDECL(int, int):
```



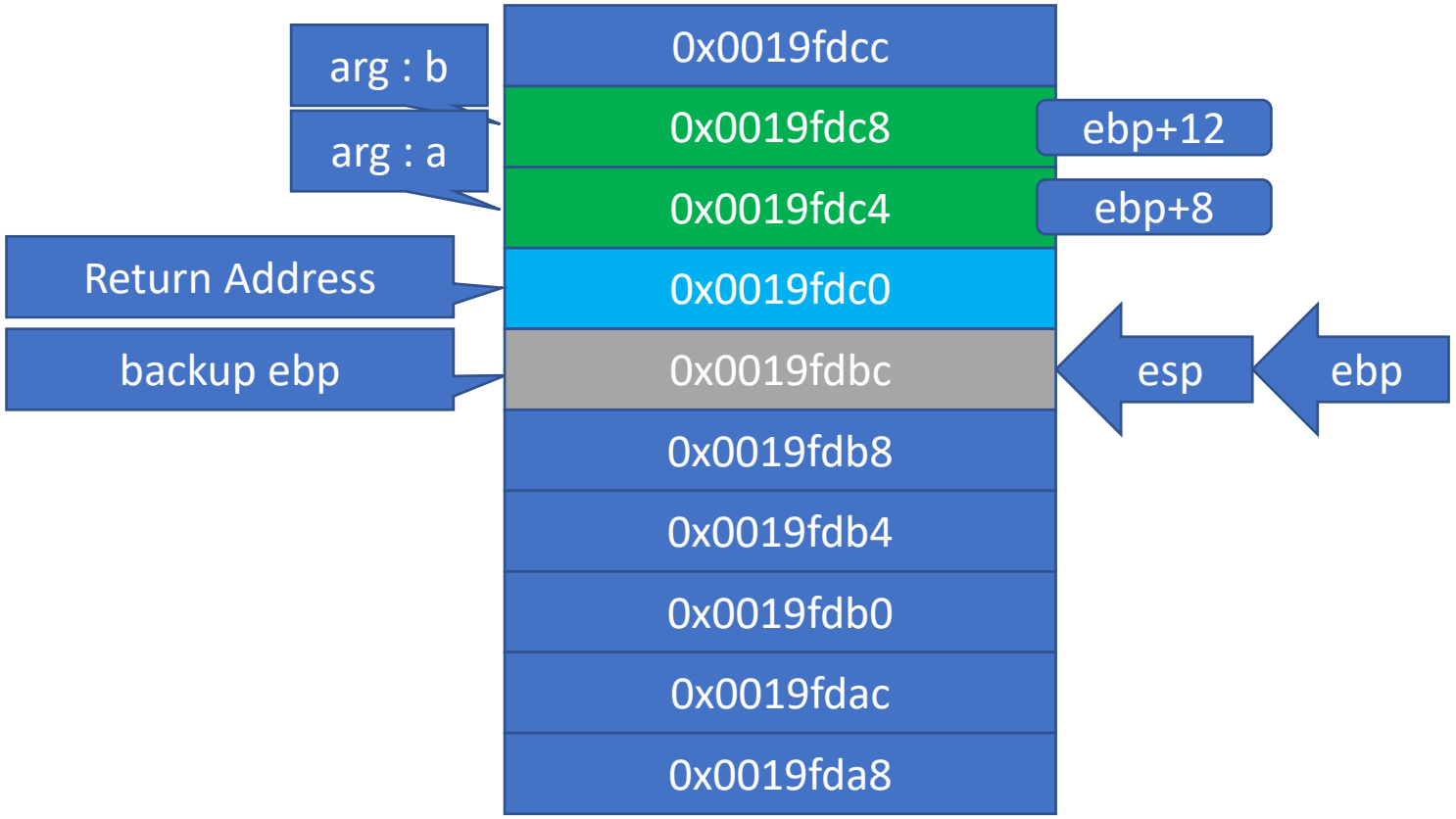
Stack frame

```
mov           eax,dword ptr [b]
push          eax
mov           ecx,dword ptr [a]
push          ecx
call          Add_NakedCall_CDECL

Add_NakedCall_CDECL(int, int):
push          ebp
```



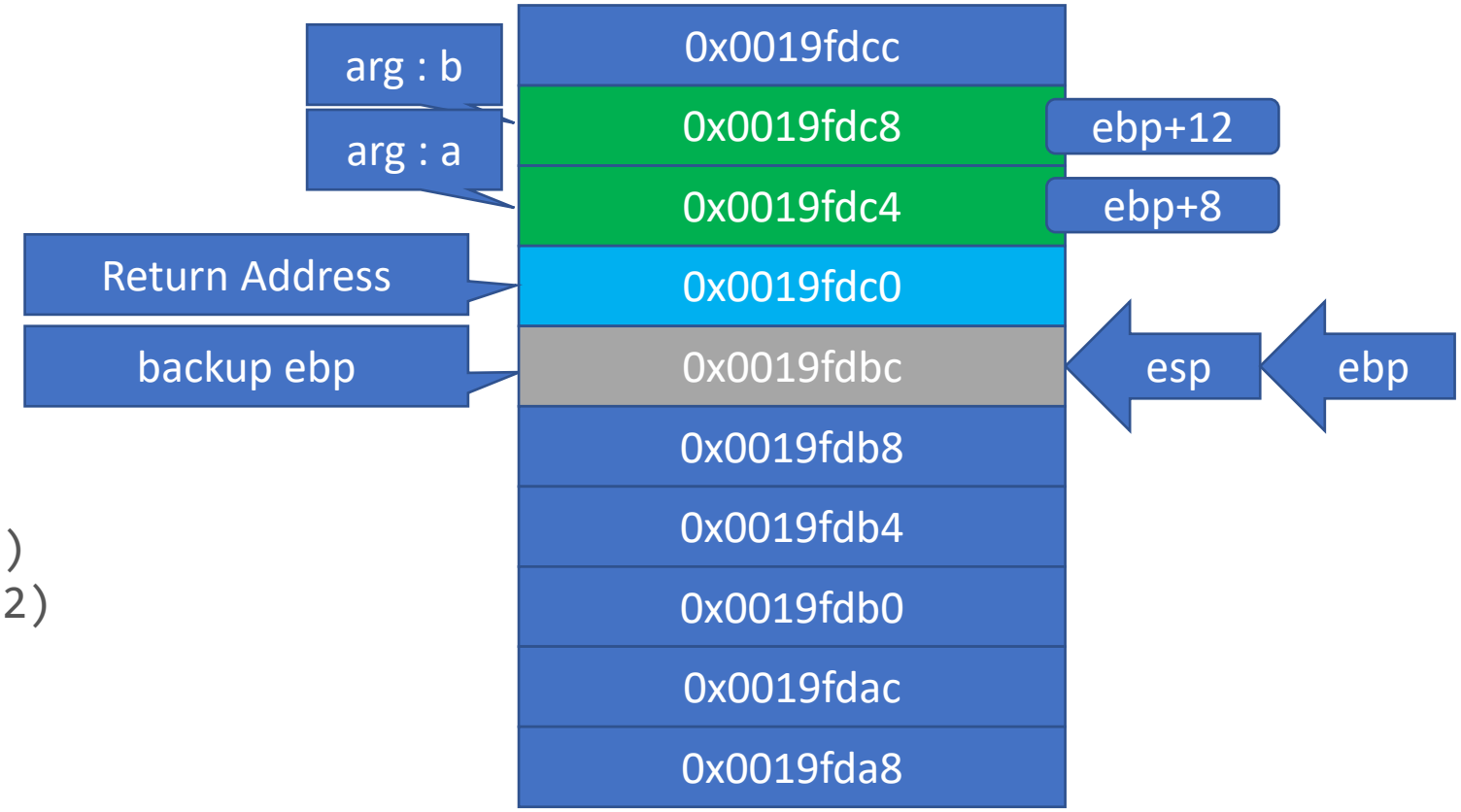
Stack frame

```
mov           eax,dword ptr [b]
push          eax
mov           ecx,dword ptr [a]
push          ecx
call          Add_NakedCall_CDECL

Add_NakedCall_CDECL(int, int):
push          ebp
mov           ebp,esp
```



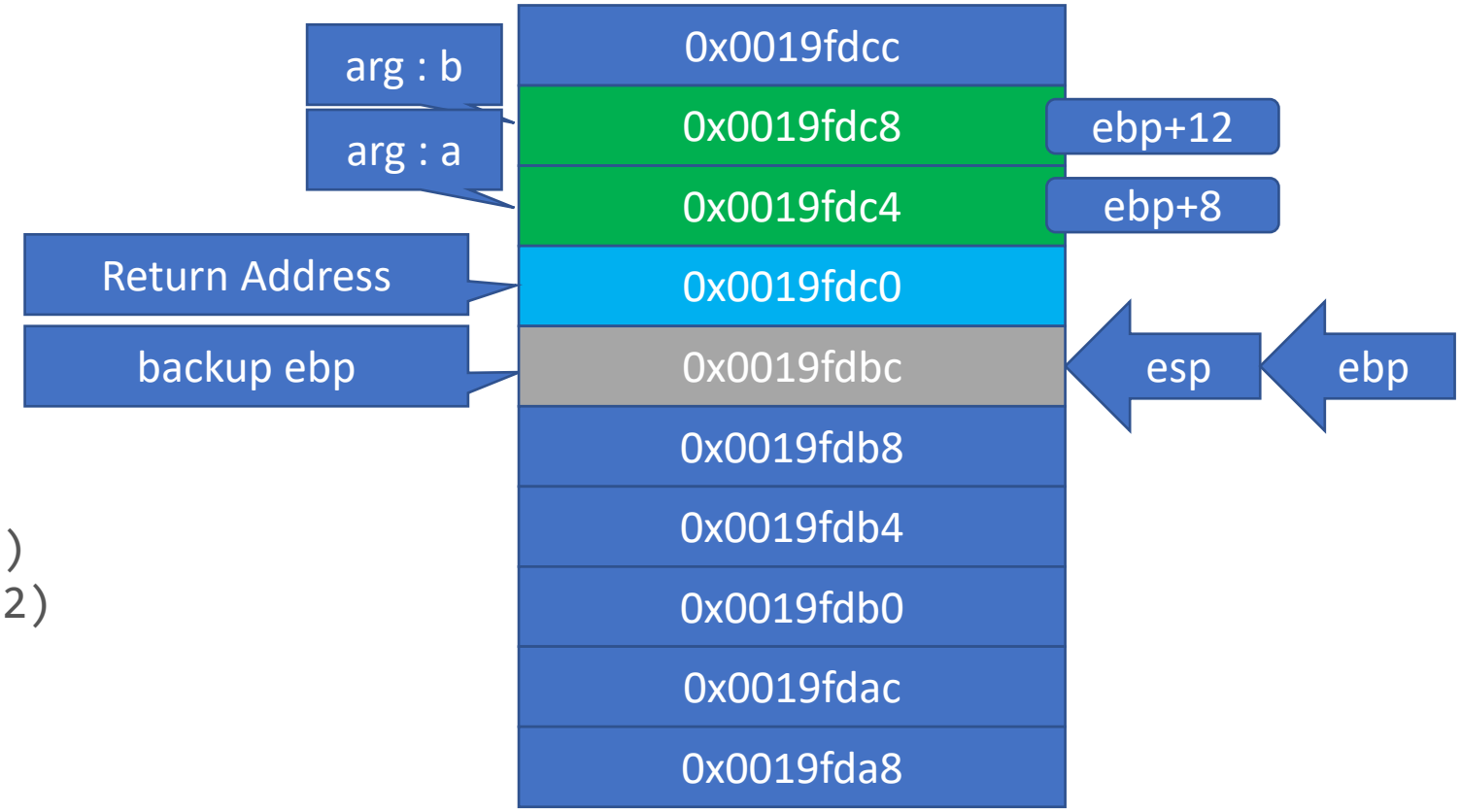
Stack frame

```
mov           eax,dword ptr [b]
push          eax
mov           ecx,dword ptr [a]
push          ecx
call          Add_NakedCall_CDECL

Add_NakedCall_CDECL(int, int):
push          ebp
mov           ebp,esp
mov           eax,dword ptr [a] (ebp+8)
mov           edx,dword ptr [b] (ebp+12)
add           eax,edx
```



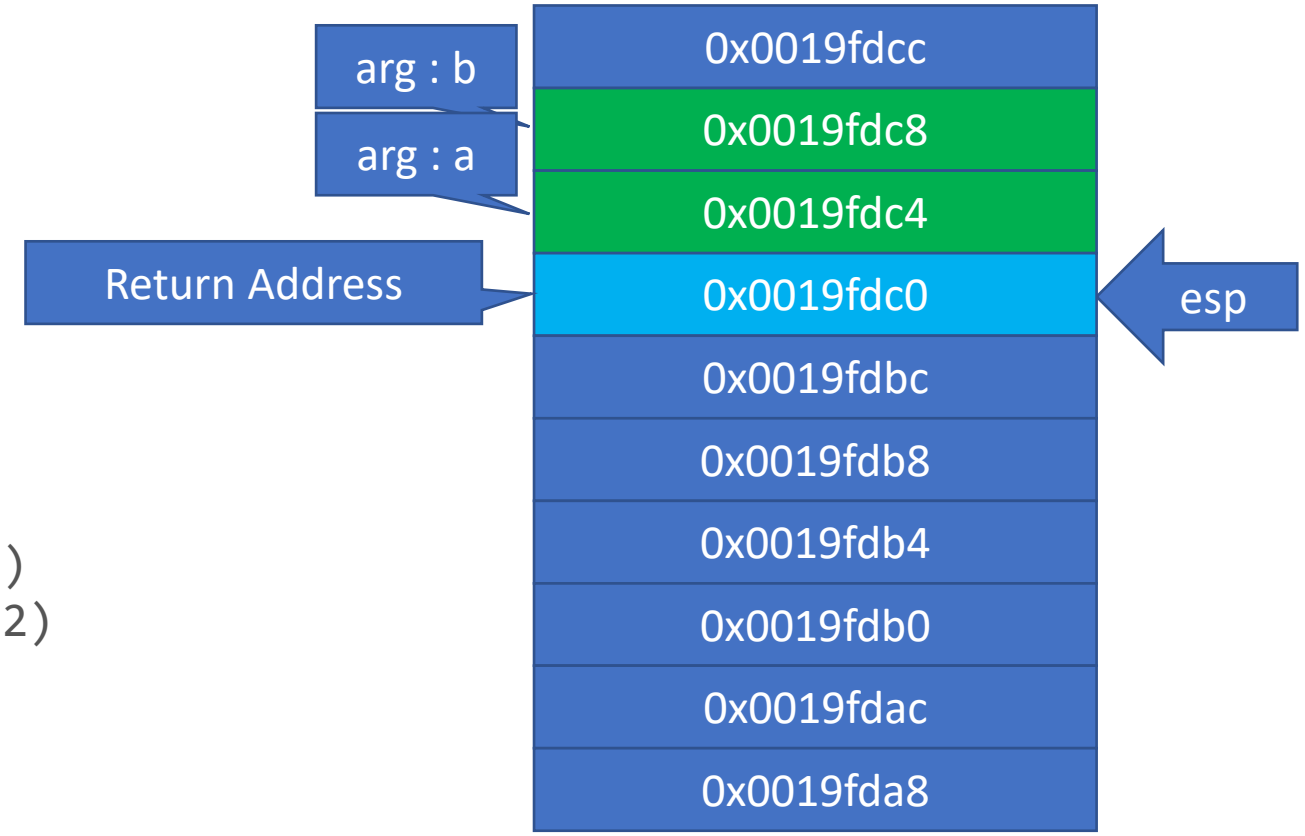
Stack frame

```
mov           eax,dword ptr [b]
push          eax
mov           ecx,dword ptr [a]
push          ecx
call          Add_NakedCall_CDECL

Add_NakedCall_CDECL(int, int):
push          ebp
mov           ebp,esp
mov           eax,dword ptr [a] (ebp+8)
mov           edx,dword ptr [b] (ebp+12)
add           eax,edx
mov           esp,ebp
```



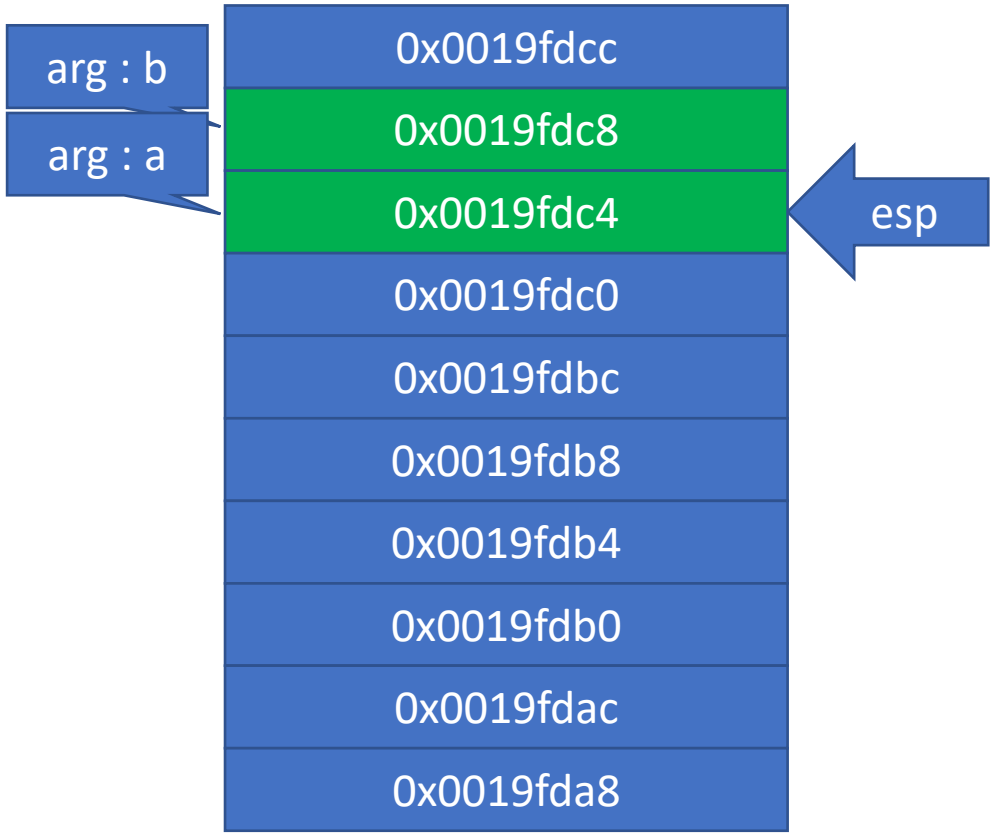
Stack frame

```
mov           eax,dword ptr [b]
push          eax
mov           ecx,dword ptr [a]
push          ecx
call          Add_NakedCall_CDECL

Add_NakedCall_CDECL(int, int):
push          ebp
mov           ebp,esp
mov           eax,dword ptr [a] (ebp+8)
mov           edx,dword ptr [b] (ebp+12)
add           eax,edx
mov           esp,ebp
pop           ebp
```



Stack frame

```
mov         eax,dword ptr [b]
push        eax
mov         ecx,dword ptr [a]
push        ecx
call        Add_NakedCall_CDECL

Add_NakedCall_CDECL(int, int):
push        ebp
mov         ebp,esp
mov         eax,dword ptr [a] (ebp+8)
mov         edx,dword ptr [b] (ebp+12)
add         eax,edx
mov         esp,ebp
pop         ebp
ret
```



Stack frame

```
mov         eax,dword ptr [b]
push        eax
mov         ecx,dword ptr [a]
push        ecx
call        Add_NakedCall_CDECL
add         esp,8
```

Stack frame

```
int __declspec(naked) __cdecl Add_NakedCall_CDECL_LOCAL_VAR(int a, int b)
{
#define ARRAY_COUNT 4
	int temp[ARRAY_COUNT];

	__asm
	{
		push ebp
		mov ebp, esp

		push edi

		; local variable
		sub esp, __LOCAL_SIZE

		; clear local variable
		; &temp[0] -> (ebp-20) , &temp[1] -> (ebp-16), &temp[2] -> (ebp-12), &temp[3] -> (ebp-8)
		mov ecx, ARRAY_COUNT
		xor eax, eax

		lea edi, dword ptr[temp]//
		rep stosd

		; a + b
		mov eax,dword ptr[a]
		mov edx,dword ptr[b]
		add eax,edx

		pop edi

		mov esp,ebp
		pop ebp
		ret
	}
}
```

Local 변수가 있을 경우

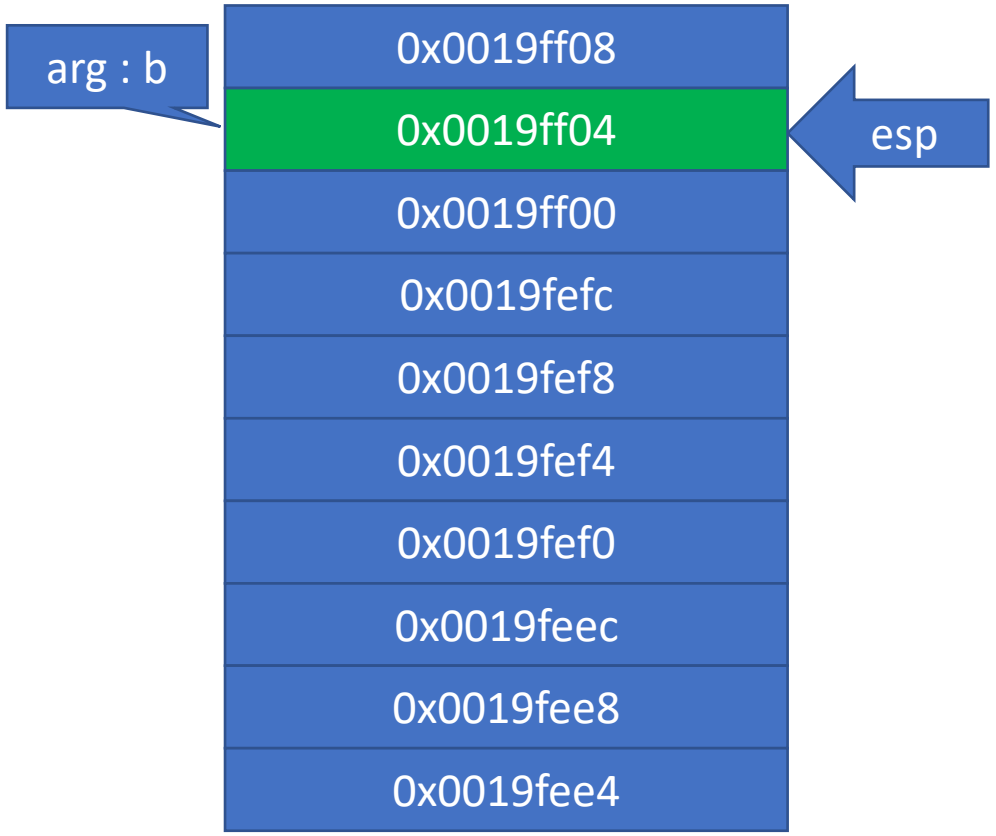
Release 모드 기준!!!

Stack frame

```
mov         eax,dword ptr [b]
push        eax
mov         ecx,dword ptr [a]
push        ecx
call        Add_NakedCall_CDECL_LOCAL_VAR
add         esp,8
```

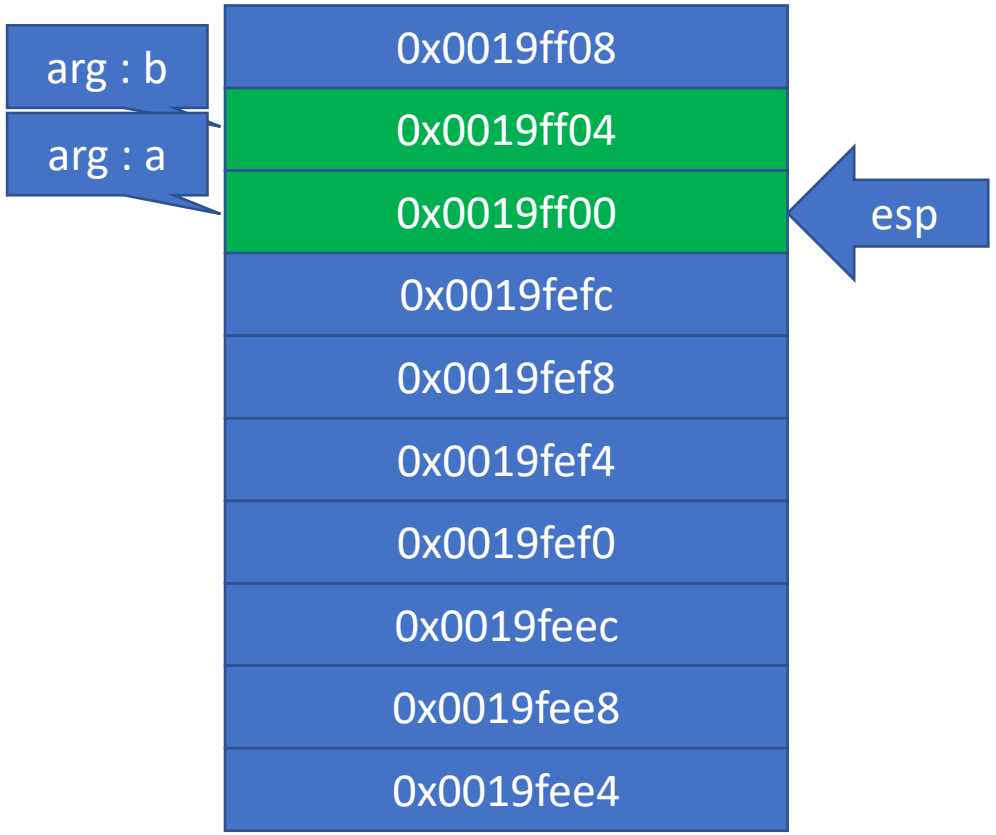
Stack frame

```
mov         eax,dword ptr [b]
push        eax
```



Stack frame

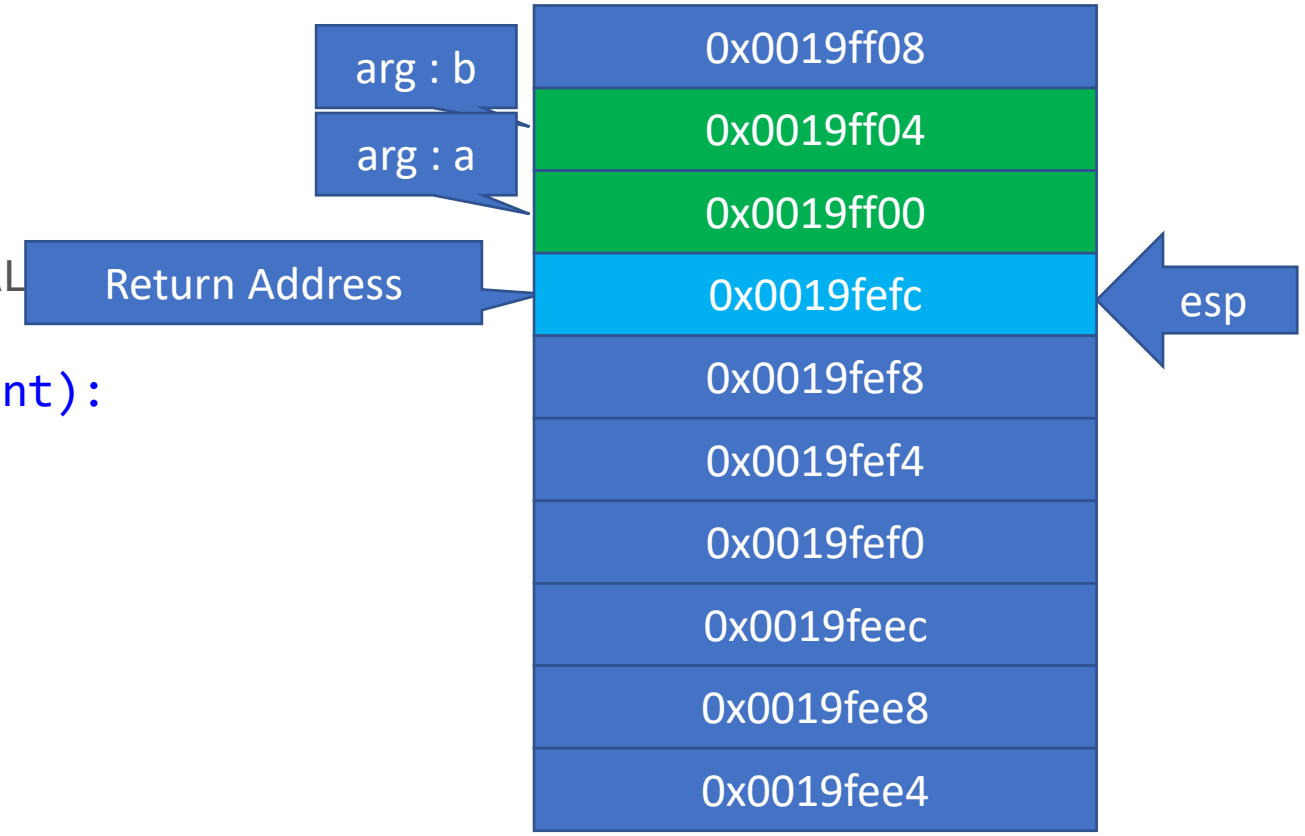
```
mov         eax,dword ptr [b]
push        eax
mov         ecx,dword ptr [a]
push        ecx
```



Stack frame

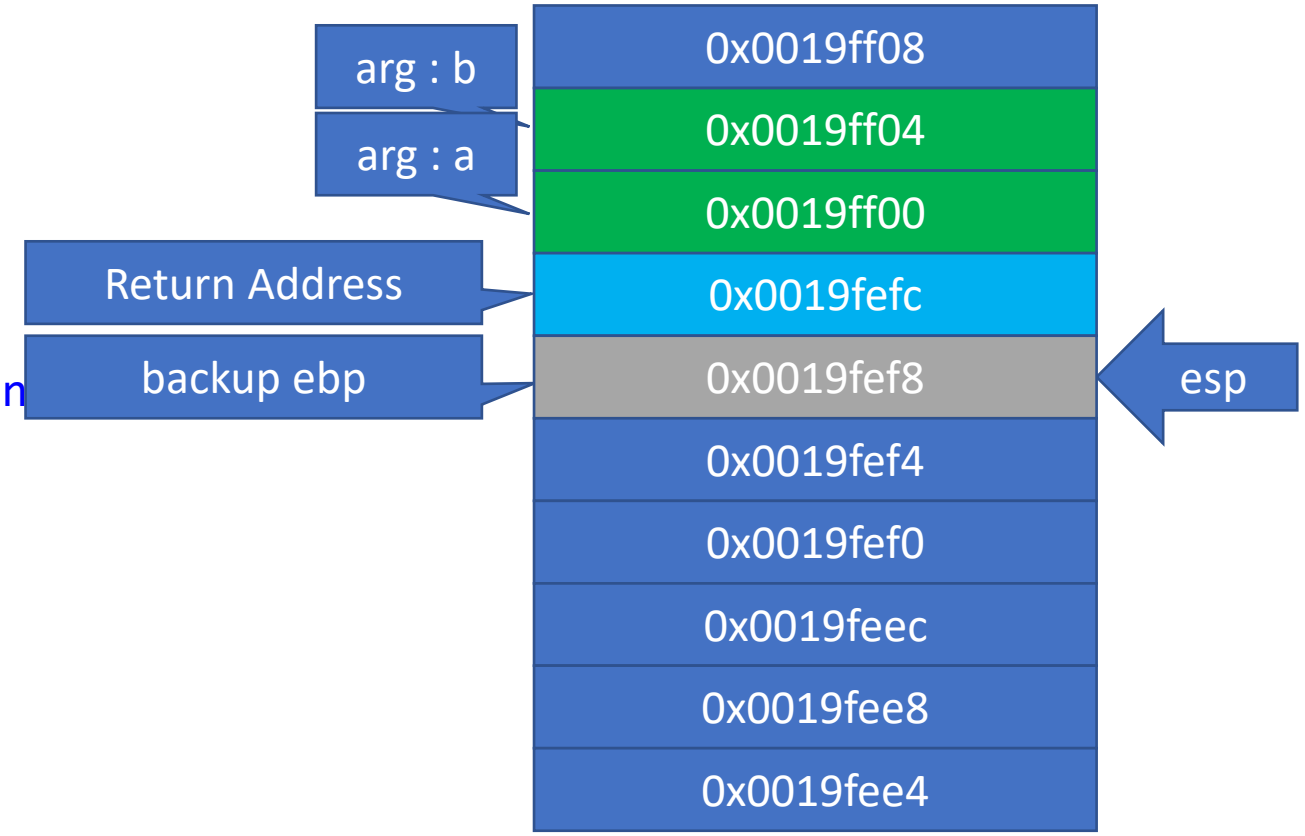
```
mov           eax,dword ptr [b]
push          eax
mov           ecx,dword ptr [a]
push          ecx
call          Add_NakedCall_CDECL_LOCAL
```

```
Add_NakedCall_CDECL_LOCAL_VAR(int, int):
```



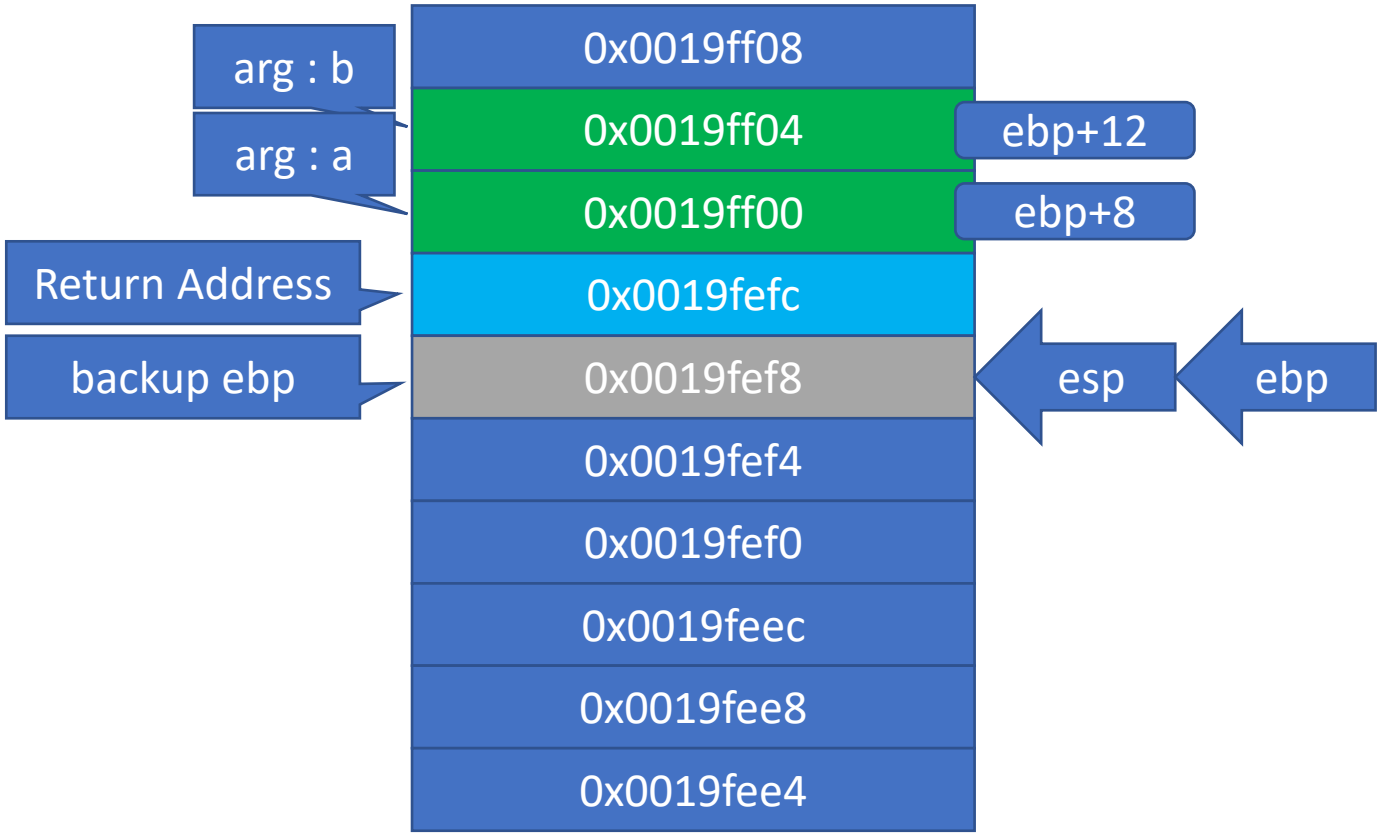
Stack frame

```
mov           eax,dword ptr [b]
push          eax
mov           ecx,dword ptr [a]
push          ecx
call          Add_NakedCall_CDECL

Add_NakedCall_CDECL_LOCAL_VAR(int, in
push          ebp
```



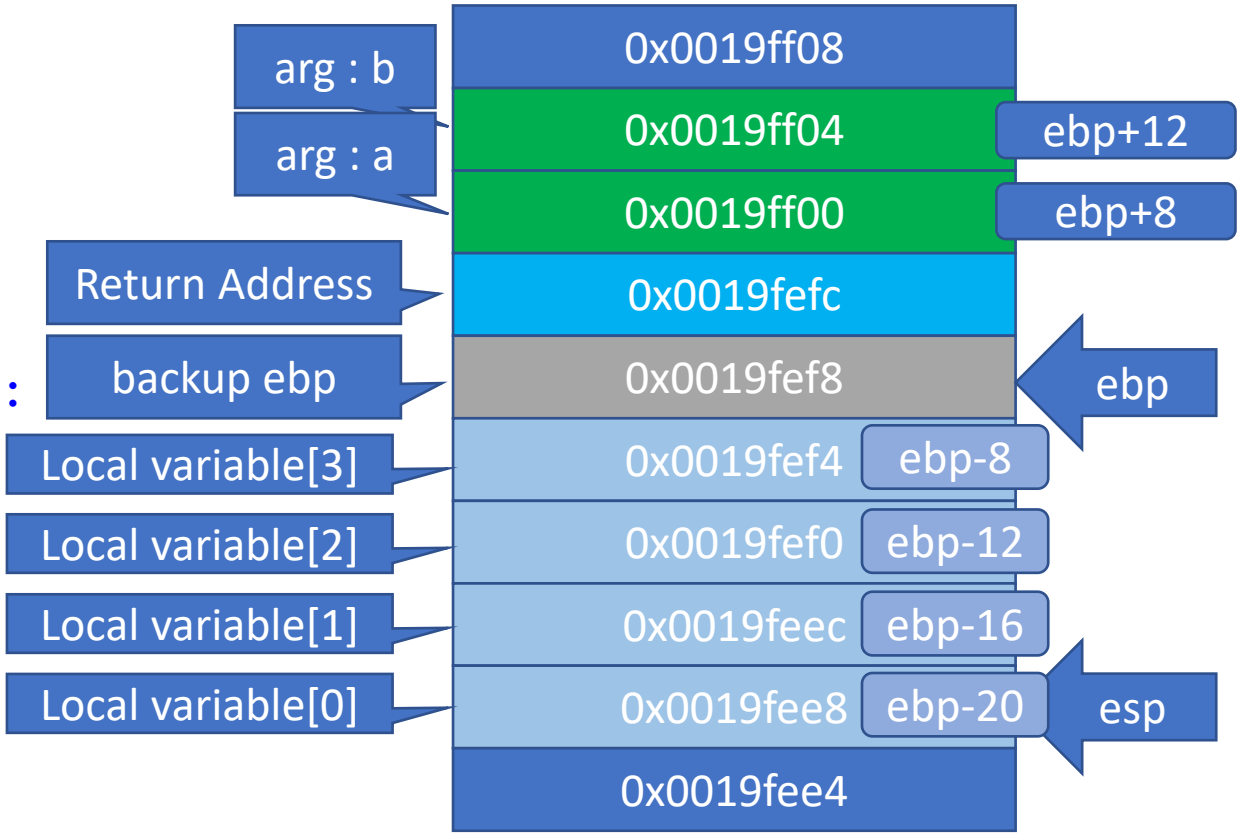
Stack frame

```
mov           eax,dword ptr [b]
push          eax
mov           ecx,dword ptr [a]
push          ecx
call          Add_NakedCall_CDECL

Add_NakedCall_CDECL_LOCAL_VAR(int, int):
push          ebp
mov           ebp,esp
```



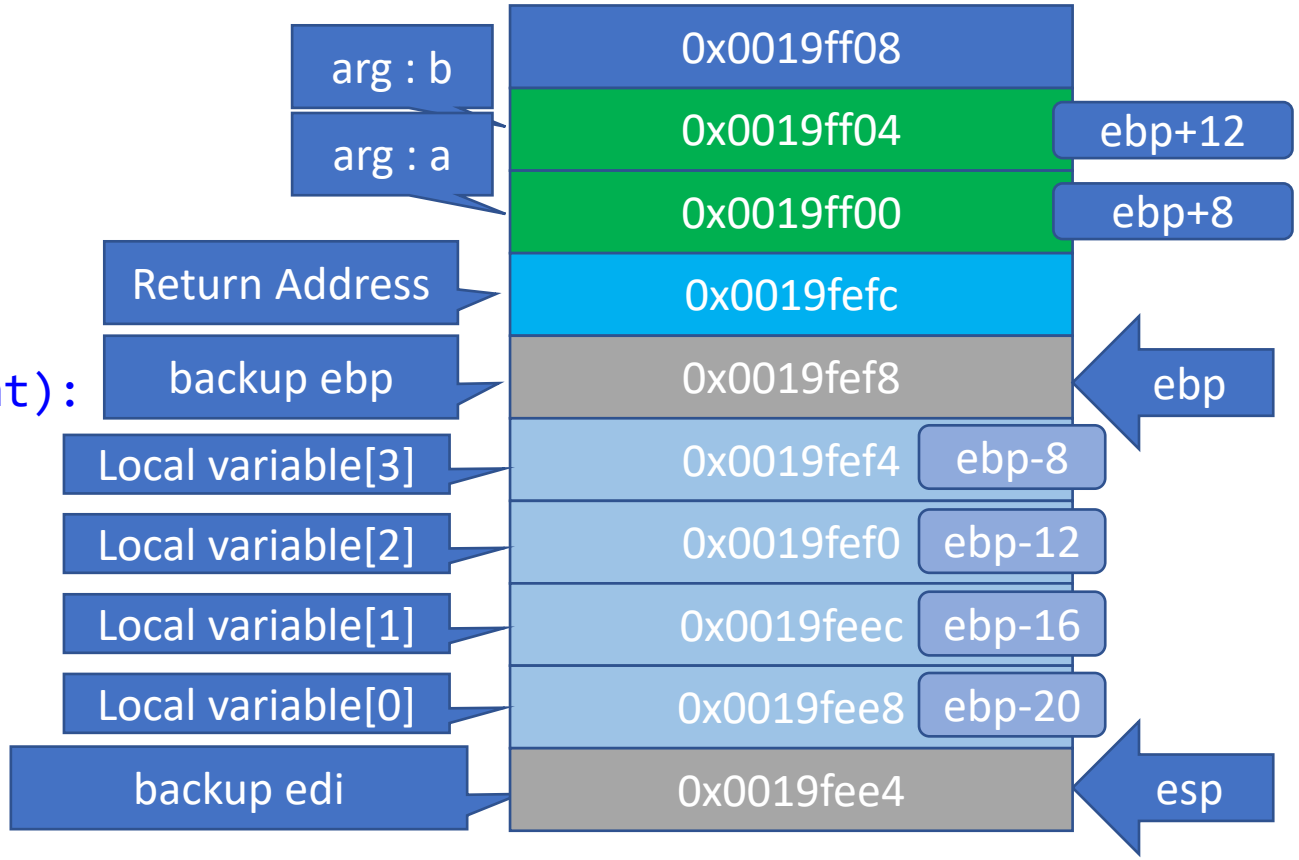
Stack frame

```
mov           eax,dword ptr [b]
push          eax
mov           ecx,dword ptr [a]
push          ecx
call          Add_NakedCall_CDECL

Add_NakedCall_CDECL_LOCAL_VAR(int, int):
push          ebp
mov           ebp,esp
sub           esp,10h
```



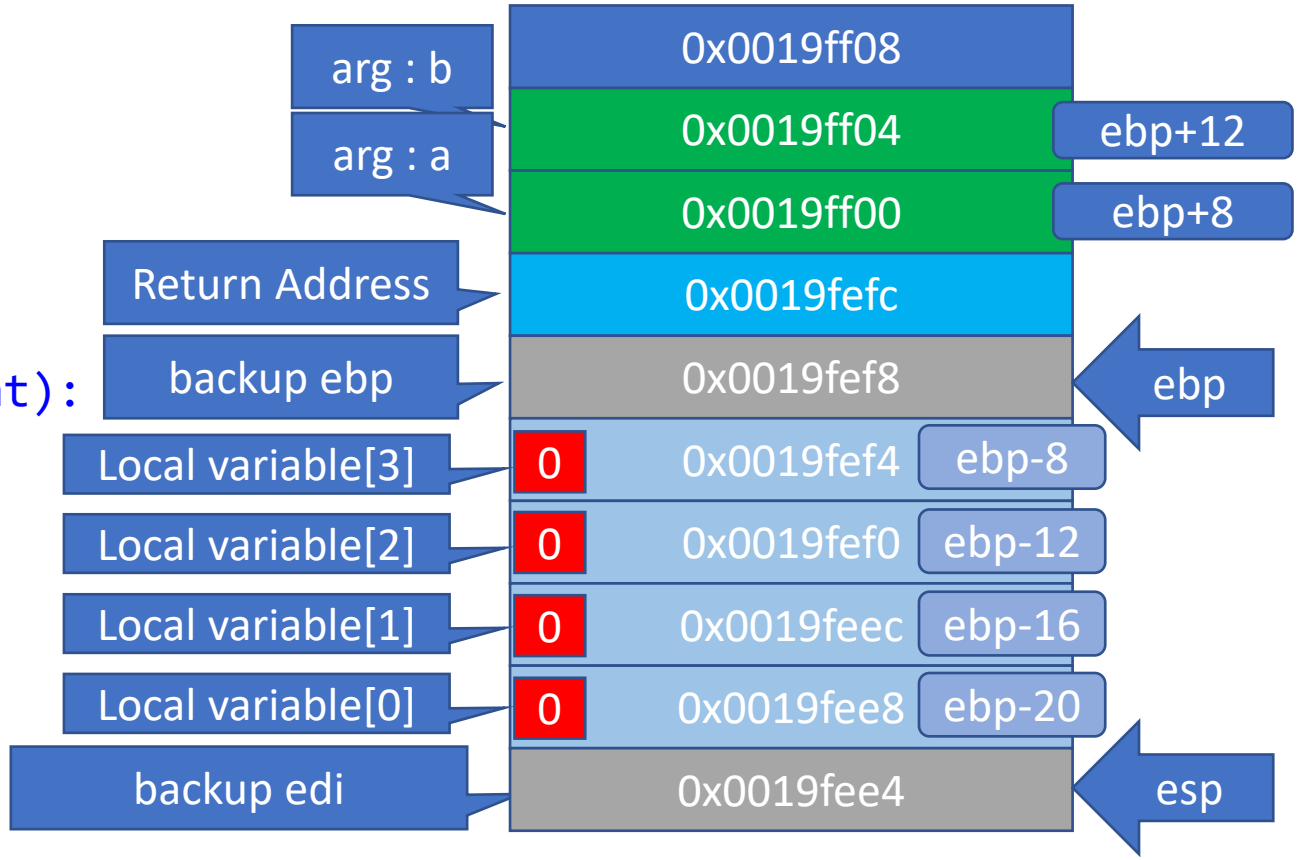
Stack frame

```
mov           eax,dword ptr [b]
push          eax
mov           ecx,dword ptr [a]
push          ecx
call          Add_NakedCall_CDECL

Add_NakedCall_CDECL_LOCAL_VAR(int, int):
push          ebp
mov           ebp,esp
sub           esp,10h
push          edi
```



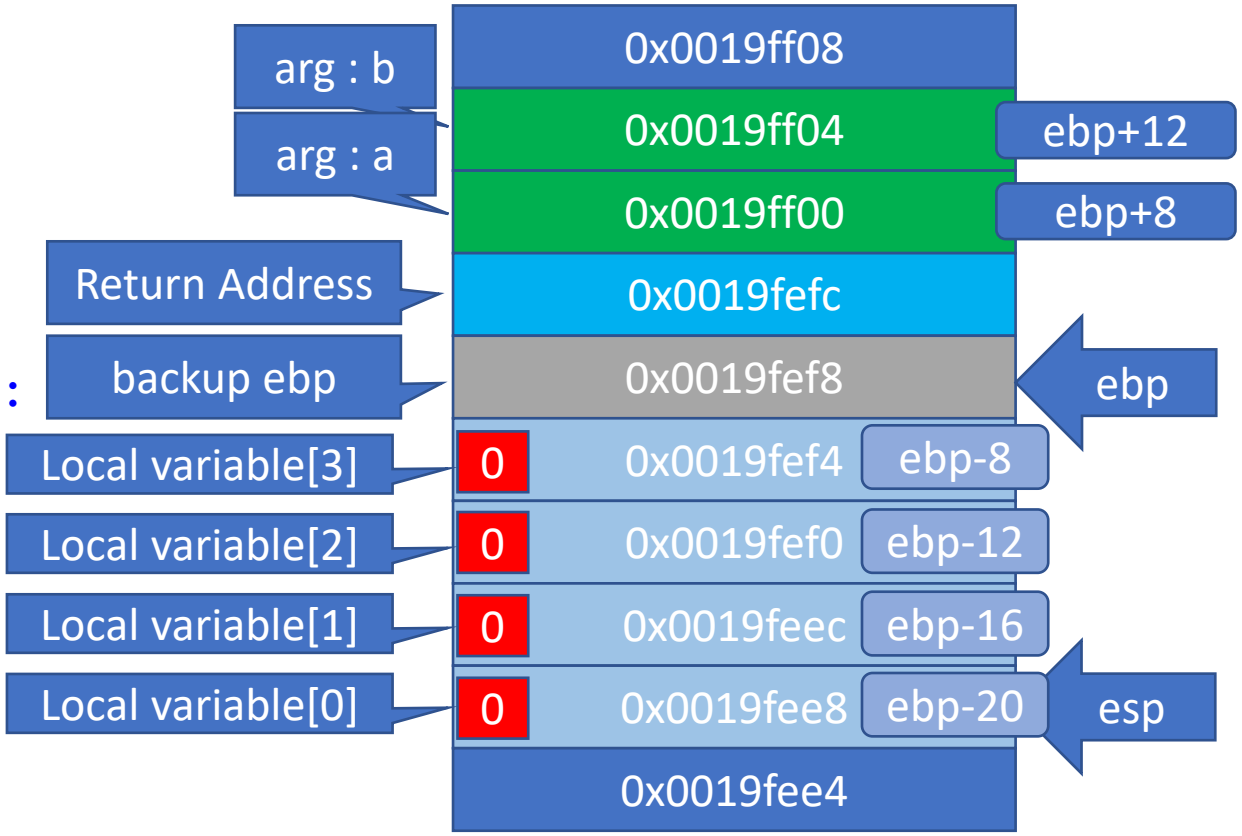
Stack frame

```
mov           eax,dword ptr [b]
push          eax
mov           ecx,dword ptr [a]
push          ecx
call          Add_NakedCall_CDECL

Add_NakedCall_CDECL_LOCAL_VAR(int, int):
push          ebp
mov           ebp,esp
sub           esp,10h
push          edi
mov           ecx,4
xor           eax,eax
lea           edi,[temp]
rep stos      dword ptr es:[edi]
mov           eax,dword ptr [a]
mov           edx,dword ptr [b]
add           eax,edx
```



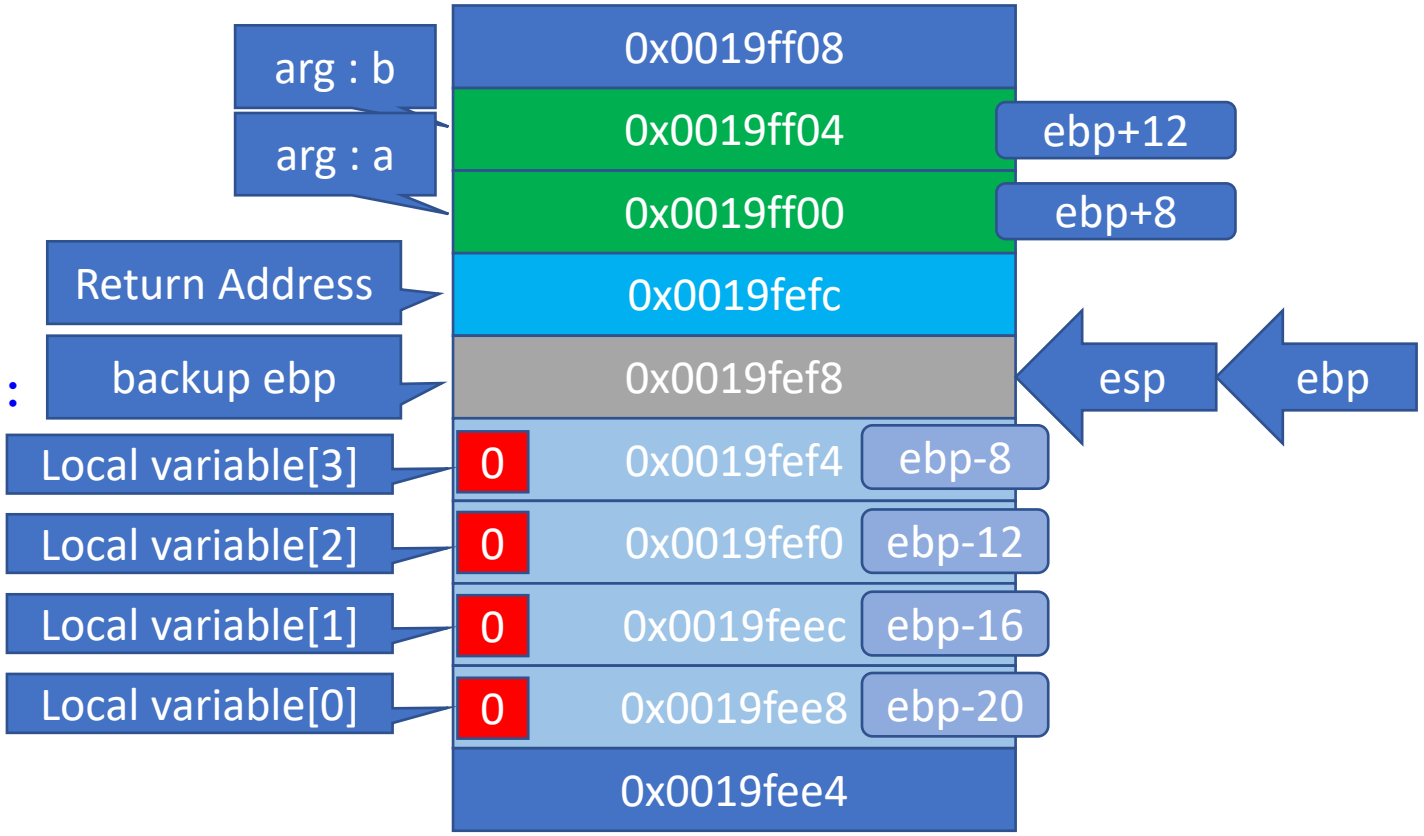
Stack frame

```
mov           eax,dword ptr [b]
push          eax
mov           ecx,dword ptr [a]
push          ecx
call          Add_NakedCall_CDECL

Add_NakedCall_CDECL_LOCAL_VAR(int, int):
push          ebp
mov           ebp,esp
sub           esp,10h
push          edi
mov           ecx,4
xor           eax,eax
lea           edi,[temp]
rep stos      dword ptr es:[edi]
mov           eax,dword ptr [a]
mov           edx,dword ptr [b]
add           eax,edx
pop           edi
```



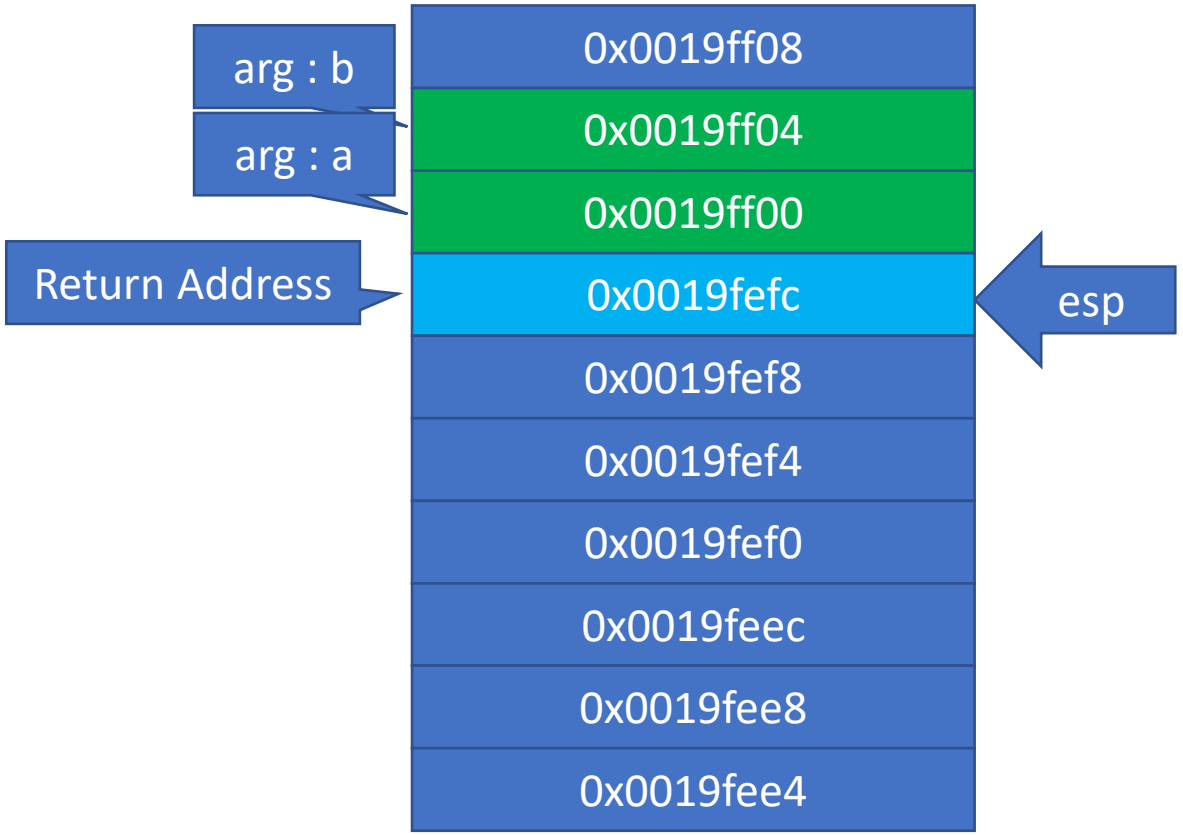
Stack frame

```
mov         eax,dword ptr [b]
push        eax
mov         ecx,dword ptr [a]
push        ecx
call        Add_NakedCall_CDECL

Add_NakedCall_CDECL_LOCAL_VAR(int, int):
push        ebp
mov         ebp,esp
sub         esp,10h
push        edi
mov         ecx,4
xor         eax,eax
lea         edi,[temp]
rep stos    dword ptr es:[edi]
mov         eax,dword ptr [a]
mov         edx,dword ptr [b]
add         eax,edx
pop         edi
mov         esp,ebp
```



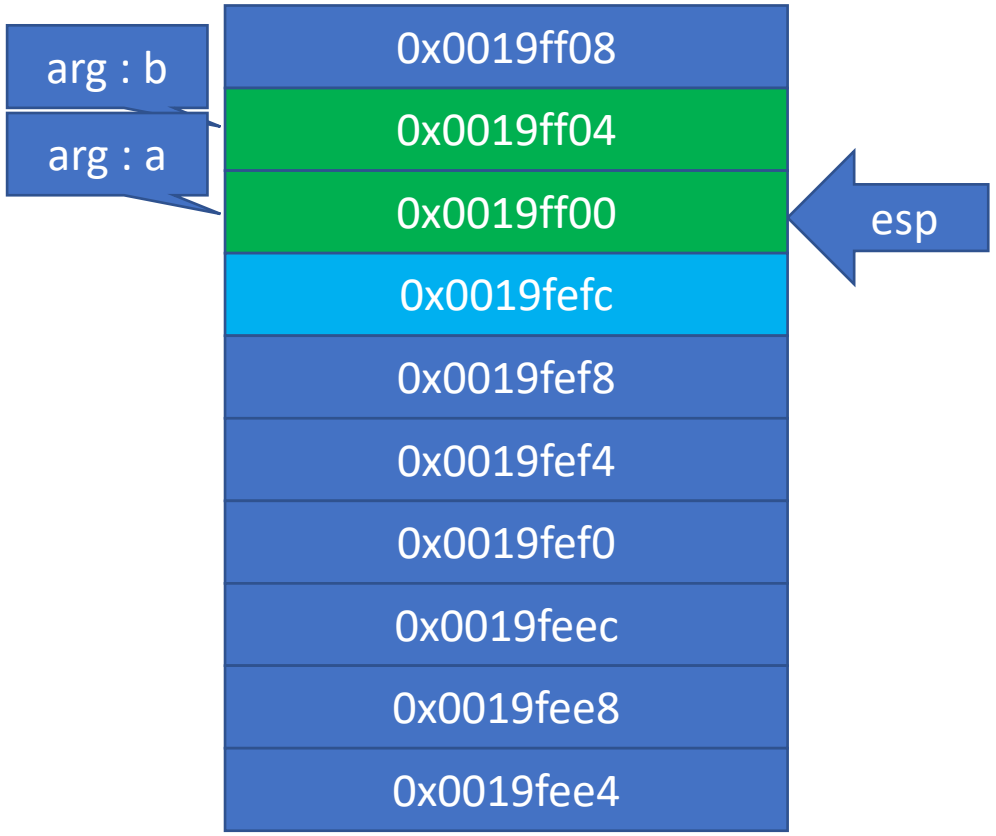
Stack frame

```
mov         eax,dword ptr [b]
push        eax
mov         ecx,dword ptr [a]
push        ecx
call        Add_NakedCall_CDECL

Add_NakedCall_CDECL_LOCAL_VAR(int, int):
push        ebp
mov         ebp,esp
sub         esp,10h
push        edi
mov         ecx,4
xor         eax,eax
lea         edi,[temp]
rep stos    dword ptr es:[edi]
mov         eax,dword ptr [a]
mov         edx,dword ptr [b]
add         eax,edx
pop         edi
mov         esp,ebp
pop         ebp
```



Stack frame

```
mov         eax,dword ptr [b]
push        eax
mov         ecx,dword ptr [a]
push        ecx
call        Add_NakedCall_CDECL

Add_NakedCall_CDECL_LOCAL_VAR(int, int):
push        ebp
mov         ebp,esp
sub         esp,10h
push        edi
mov         ecx,4
xor         eax,eax
lea         edi,[temp]
rep stos    dword ptr es:[edi]
mov         eax,dword ptr [a]
mov         edx,dword ptr [b]
add         eax,edx
pop         edi
mov         esp,ebp
pop         ebp
ret
```



Stack frame

```
mov         eax,dword ptr [b]
push        eax
mov         ecx,dword ptr [a]
push        ecx
call        Add_NakedCall_CDECL_LOCAL_VAR
add         esp,8
```

Stack frame