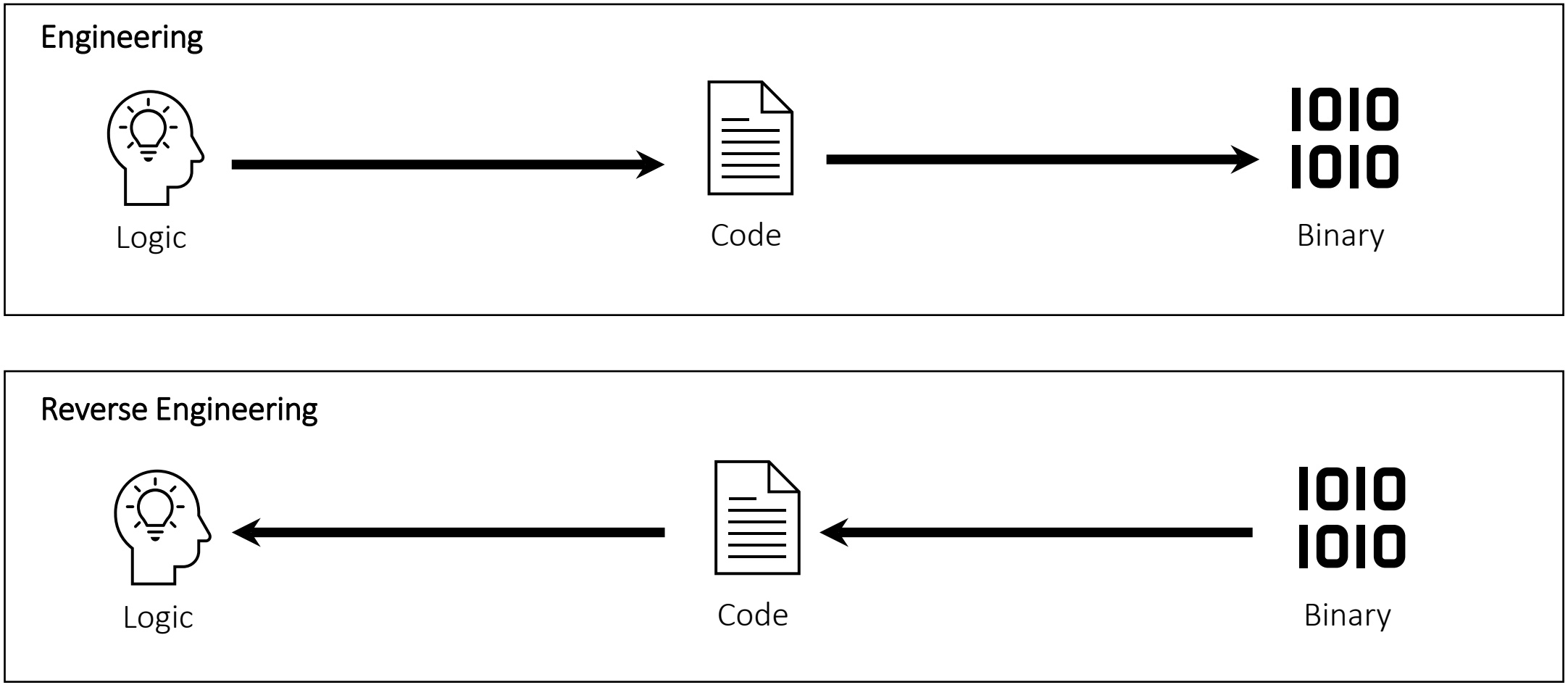
Reverse Engineering

Insu Yun

Today’s lecture

- Understand what reverse engineering is
- Understand static analysis and dynamic analysis
- Understand x86 assembly
- Understand how to use decompiler

What is reverse engineering?



Reverse engineering is challenging due to loss of information

```c
void decode_string(char* str, int len) {
  // XOR string with a given length
  for (int i = 0; i < len; i++) {
    *str ^= 42;
    str++;
  }
}
```

No variable or function names

```c
void FUN_100000e20(byte *param_1,ulong param_2)

{
  int local_1c;
  byte *local_10;

  local_1c = 0;
  local_10 = param_1;
  while ((ulong)(long)local_1c < param_2) {
    *local_10 = *local_10 ^ 0x2a;
    local_10 = local_10 + 1;
    local_1c = local_1c + 1;
  }
  return;
}
```

No comment

Ambiguous representation

Two types of binary analysis: Static analysis vs Dynamic analysis

- Static analysis: Read and understand binary
  - + Give deep understanding
    (e.g., FUN_100000e20 == decode_string?
    + Can find an input to make ”Hello World”)
  - - Time consuming
  - - Error-prone
- Dynamic analysis: Run and infer from results
  - + Understand logic without expensive analysis
    (e.g., FUN_ 100000e20(cryptic, 12) gives us “Hello World”,
    => FUN_ 100000e20 == decode_string?)
  - - Shallow understanding

```
void FUN_100000e20(byte *param_1,ulong param_2)

{
  int local_1c;
  byte *local_10;

  local_1c = 0;
  local_10 = param_1;
  while ((ulong)(long)local_1c < param_2) {
    *local_10 = *local_10 ^ 0x2a;
    local_10 = local_10 + 1;
    local_1c = local_1c + 1;
  }
  return;
}
```

x86 assembly

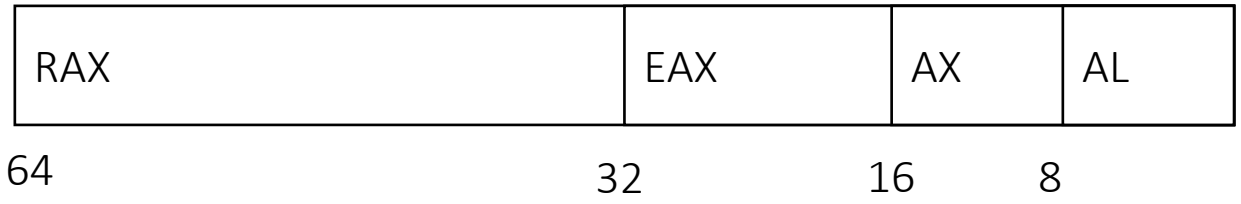
```c
int add(int a, int b) {
  return a + b;
}

int main() {
  int a = 3;
  int b = 7;
  add(a, b);
}
```

```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:     mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```

x86 assembly (Registers)

- Small and fast in-CPU memory that can store variables

- General purpose registers
  - 64bit: rax, rbx, rcx, rdx, rdi, rsi, r8-r15, etc. (8byte)
  - 32bit: eax, ebx, ecx, edx, edi, esi (4byte)



x86 assembly (Special registers)

- Program counter
  - 64bit: rip
  - 32bit: eip

- Stack management
  - 64bit: rsp, rbp
  - 32bit: ebp, esp

x86 assembly (Syntax)

- [operator] [operand1], [operand2], etc.

- Two types of x86 assembly syntax styles exist
  - Intel syntax: [operator] [destination], [source]
  - AT&T syntax: [operator] [source], [destination]

  - We will use Intel syntax in this course

x86 assembly (Examples)

- `mov eax, 1` – Store the value 1 into the eax register
- `mov eax, ebx` – Move the value of ebx to eax, (i.e., eax = ebx)
- `add eax, 1` – Add 1 to eax, (i.e., eax = eax + 1)
- `imul eax, ebx` – Multiply ebx and eax and store the result into eax

- Others: `sub, xor, or, and, shl, ashr, lshr`, …

x86 assembly (More examples)

- `lea eax, DWORD PTR [ebp – 4]`
  - Load effective address; Store ebp-4 to eax
- `mov eax, DWORD PTR [ebp – 4]`
  - Move the value stored at address ebp-4 to eax
- `mov DWORD PTR [ebp – 4], eax`
  - Store the value of eax to the address of ebp-4

x86 assembly (More examples)

- `mov eax, DWORD PTR [ebp – 4]`
  - Move 4 bytes stored at address ebp-4 to eax
- `mov ax,  WORD PTR [ebp – 4]`
  - Move 2 bytes stored at address ebp-4 to ax
- `mov al,  BYTE PTR [ebp – 4]`
  - Move 1 byte stored at address ebp-4 to al

x86 assembly (Stack)

- `push src`

```
    ==  sub esp, 4
        mov [esp], src
```

- `pop dst`

```
    == mov $dst, [esp]
       add esp, 4
```

- `leave`

```
      == mov esp, ebp
         pop ebp
```

x86 assembly (Control flow)

- `cmp  DWORD PTR [ebp – 4], 0x1234`
  - Compare the value of the address ebp-4 to 0x1234 and set EFLAGS
  - i.e., if (i == 0x1234)
- Conditional jump
  - `je  0x8048442 <main+94>` ← Equal or zero
  - `jne  0x8048442 <main+94>` ← Not equal or non-zero
  - `jle  0x8048442 <main+94>` ← Less or equal, signed
  - `jbe  0x8048442 <main+94>` ← Below or equal, unsigned
  - `jge  0x8048442 <main+94>` ← Greater or equal, signed
  - `jae  0x8048442 <main+94>` ← Above or equal, unsigned

x86 assembly (Control flow)

- Unconditional jump
  - `jmp 0x8048442 <main+94>`
- Function call
  - `call 0x8048426 <add>`
  - Push a next instruction address (i.e., return address) to stack and jump
  - `ret == pop eip`

Endianness

- Byte order to store multi-byte data
  - Big Endian: Most significant byte -> Lowest address
  - Little Endian: Most significant byte -> Highest address

- e.g., mov DWORD PTR [eax], 0x41424344

0x08048003	0x44
0x08048002	0x43
0x08048001	0x42
0x08048000	0x41

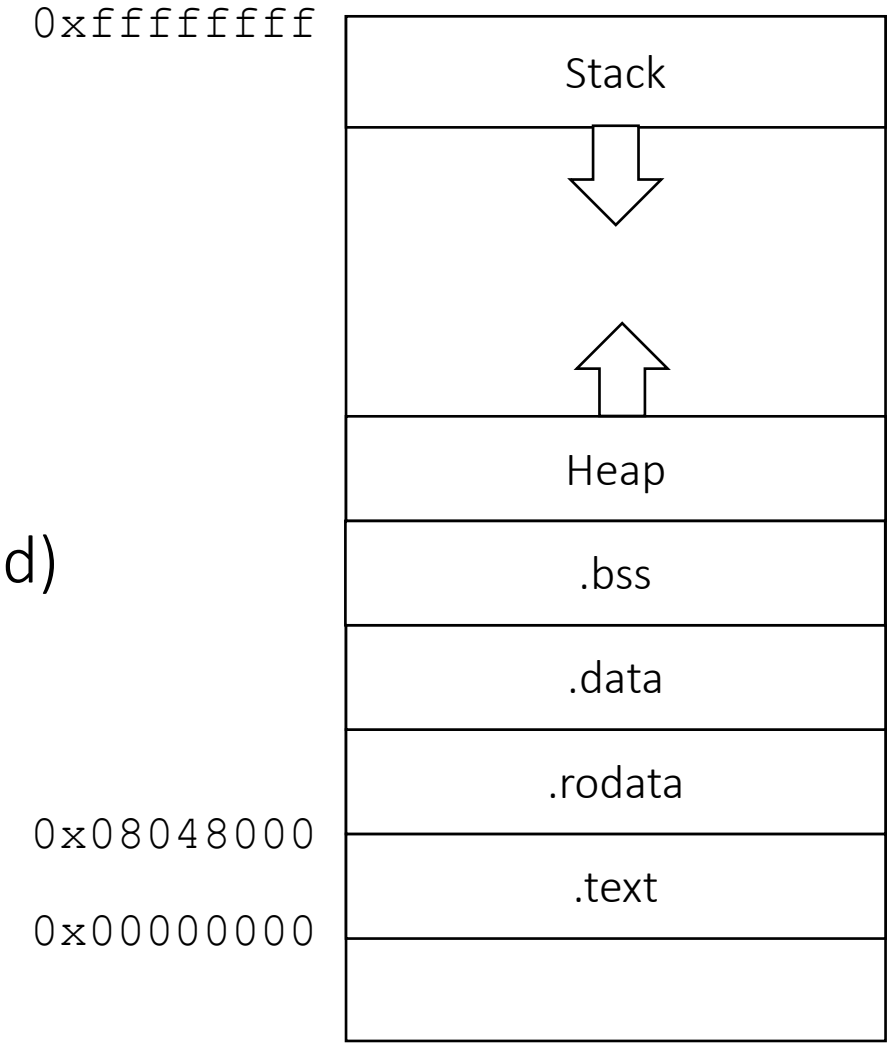
Big Endian

0x08048003	0x41
0x08048002	0x42
0x08048001	0x43
0x08048000	0x44

Little Endian (x86)

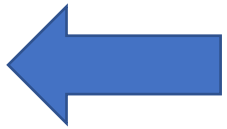
Memory Layout

- Stack
  - Local variables, call contexts, …
  - Up to 8MB in Linux by default
- Heap
  - Dynamically allocated data
- .bss: R/W global variables (not initialized)
- .data: R/W global variables
- .rodata
  - Read-only data
  - e.g., “Hello World”
- .text: Read-only code



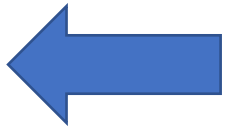
Call stack

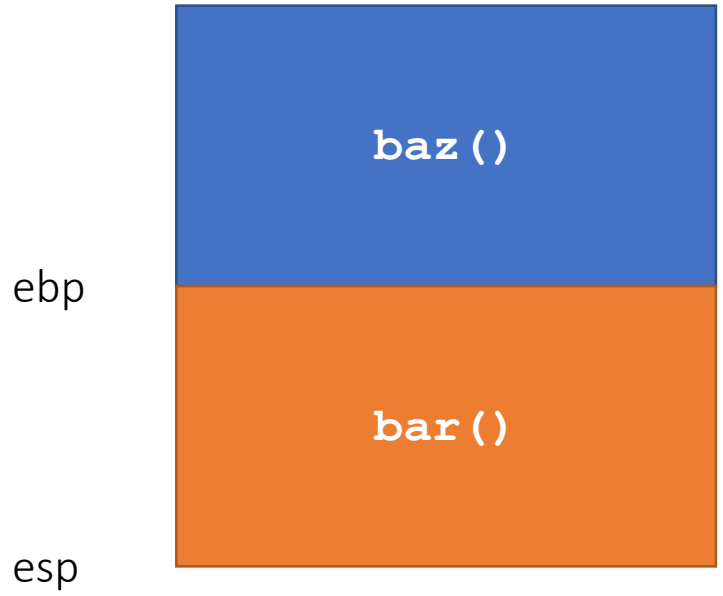
```
void baz() {
  bar();
  ...
}

void bar() {
  foo();
  ...
}

void foo() {
  ...
}
```



Call stack

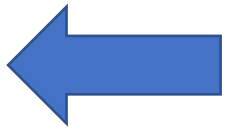
```
void baz() {
  bar();
  ...
}

void bar() {
  foo();
  ...
}

void foo() {
  ...
}
```

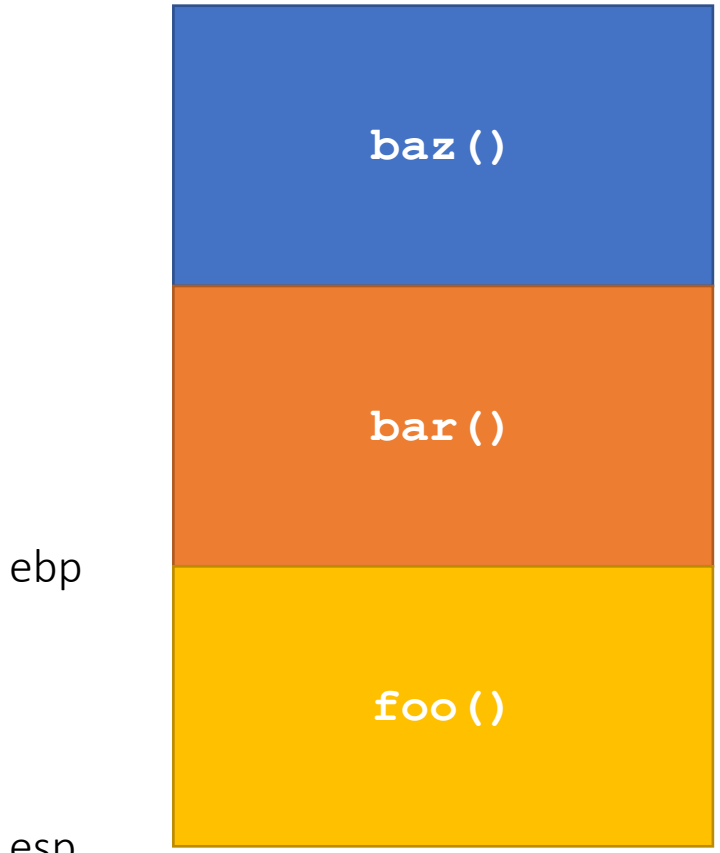




Call stack

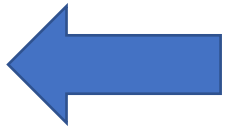
```
void baz() {
  bar();
  ...
}

void bar() {
  foo();
  ...
}

void foo() {
  ...
}
```

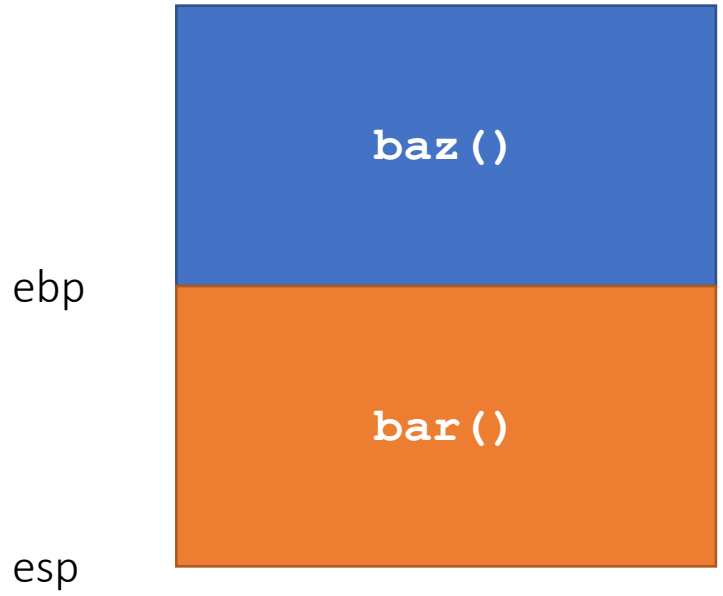




Call stack

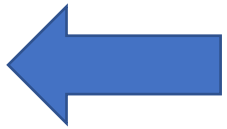
```
void baz() {
  bar();
  ...
}

void bar() {
  foo();
  ...
}

void foo() {
  ...
}
```



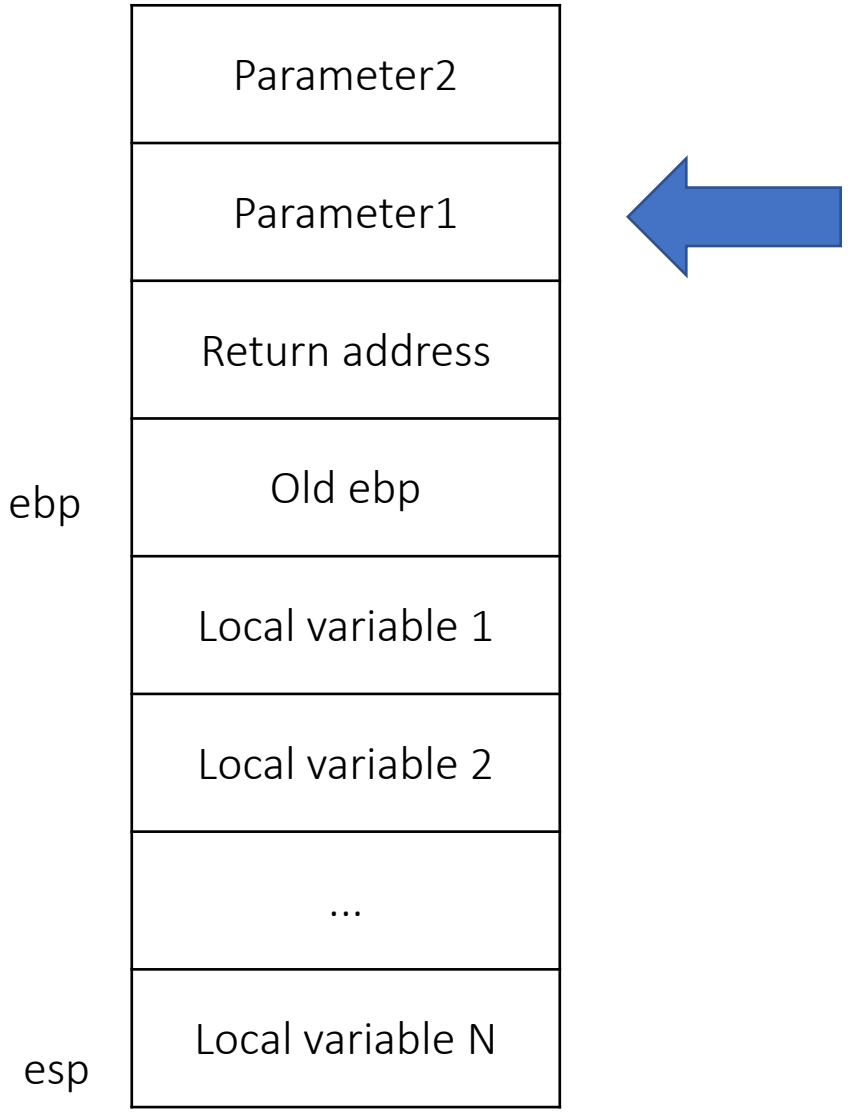


Call stack

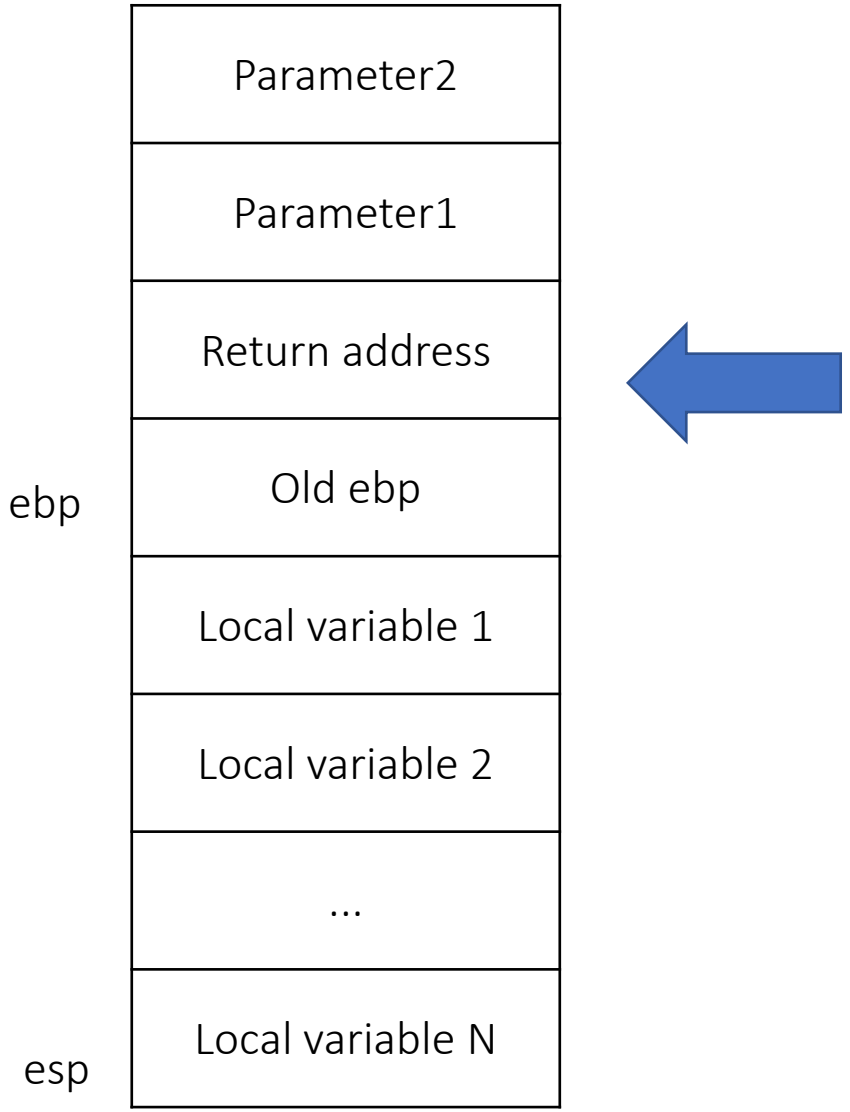
```
void baz() {
  bar();
  ...
}

void bar() {
  foo();
  ...
}

void foo() {
  ...
}
```



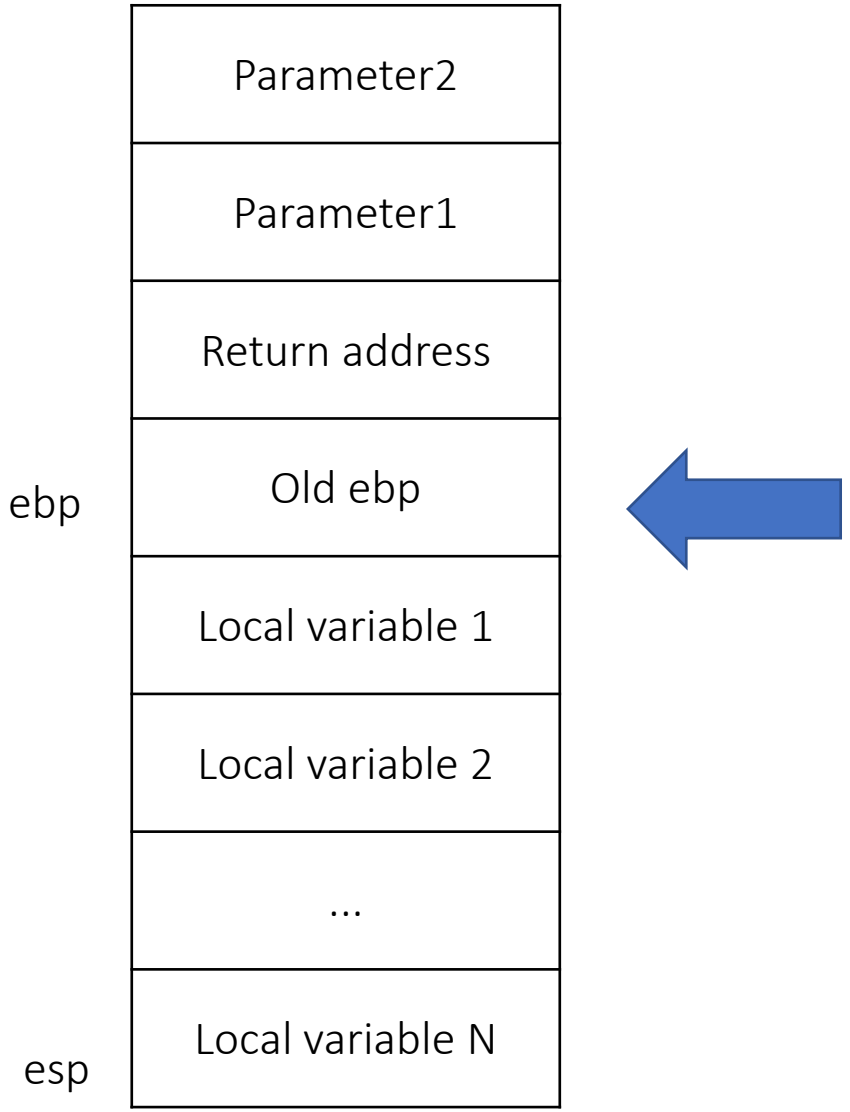
Stack frame


Parameter2
Parameter1
Return address
ebp
Old ebp
Local variable 1
Local variable 2
...
esp
Local variable N

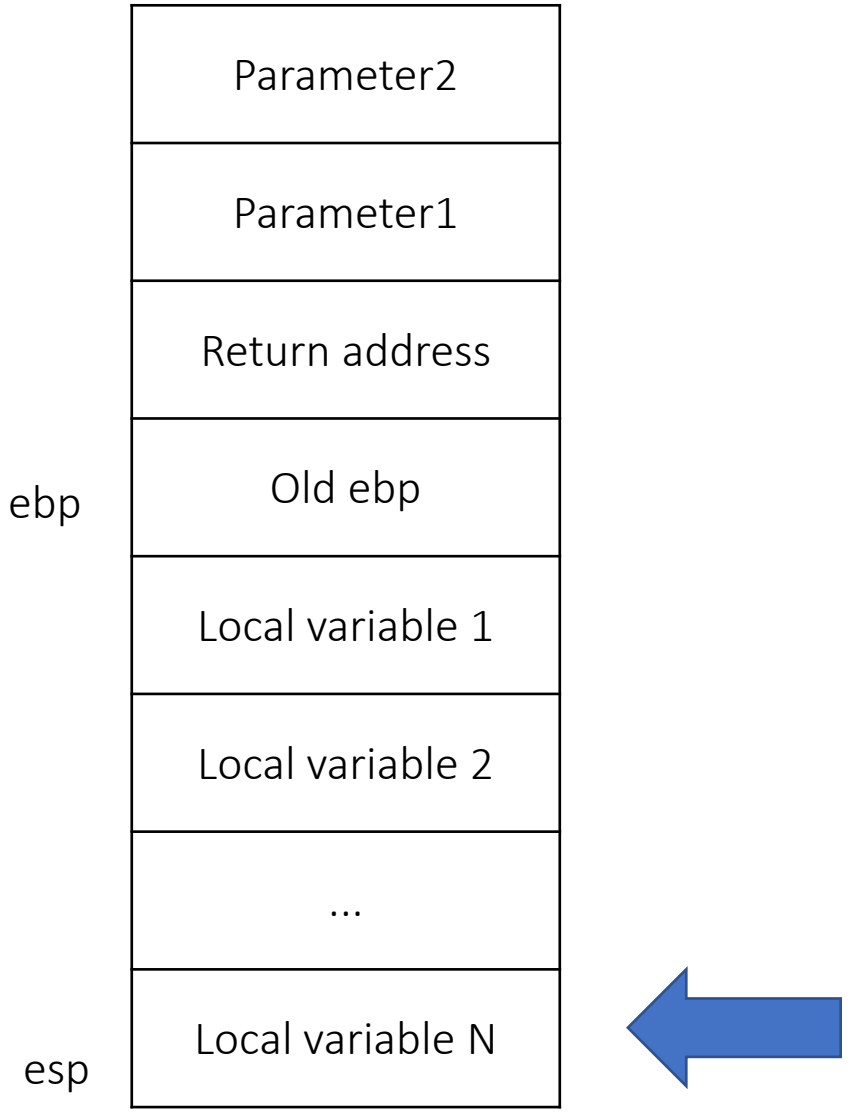
Stack frame


Parameter2
Parameter1
Return address
ebp
Old ebp
Local variable 1
Local variable 2
...
esp
Local variable N

Stack frame


Parameter2
Parameter1
Return address
ebp
Old ebp
Local variable 1
Local variable 2
...
esp
Local variable N

Stack frame


Parameter2
Parameter1
Return address
ebp
Old ebp
Local variable 1
Local variable 2
...
esp
Local variable N

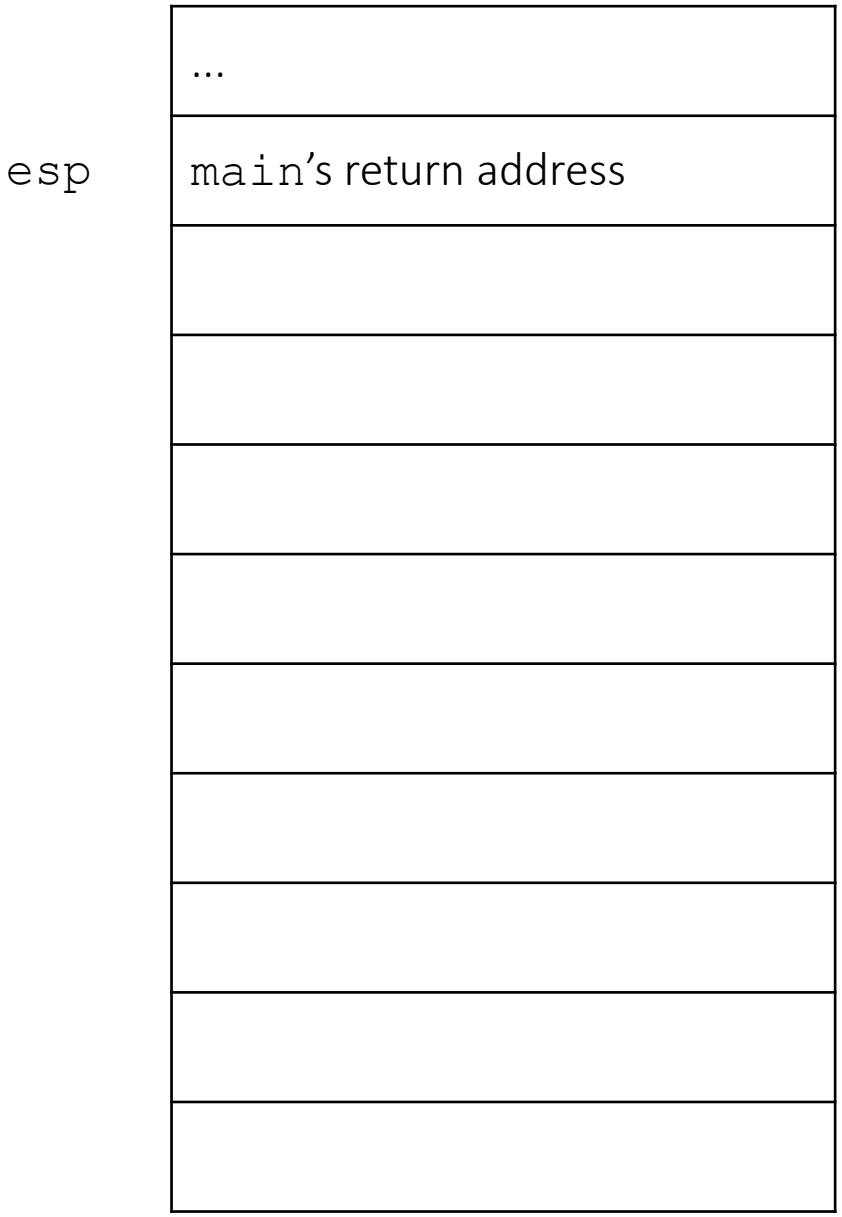
x86 Calling convention (GCC)

- Function arguments
  - Save to stack
- Return value
  - eax
- Register preservation
  - Callee-saved: ebp, edi, esi, ebx
  - Caller-saved: others

x86 assembly

```c
int add(int a, int b) {
  return a + b;
}

int main() {
  int a = 3;
  int b = 7;
  add(a, b);
}
```

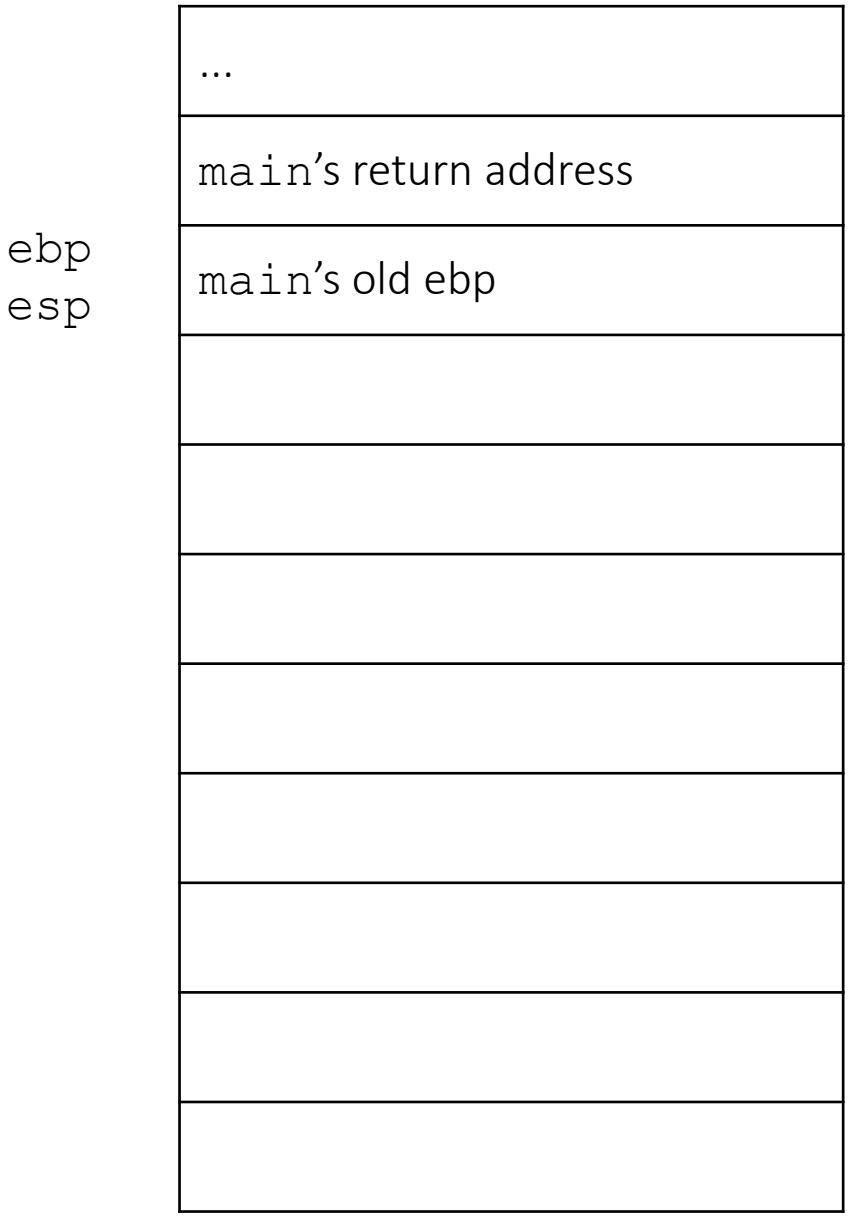
```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:     mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```



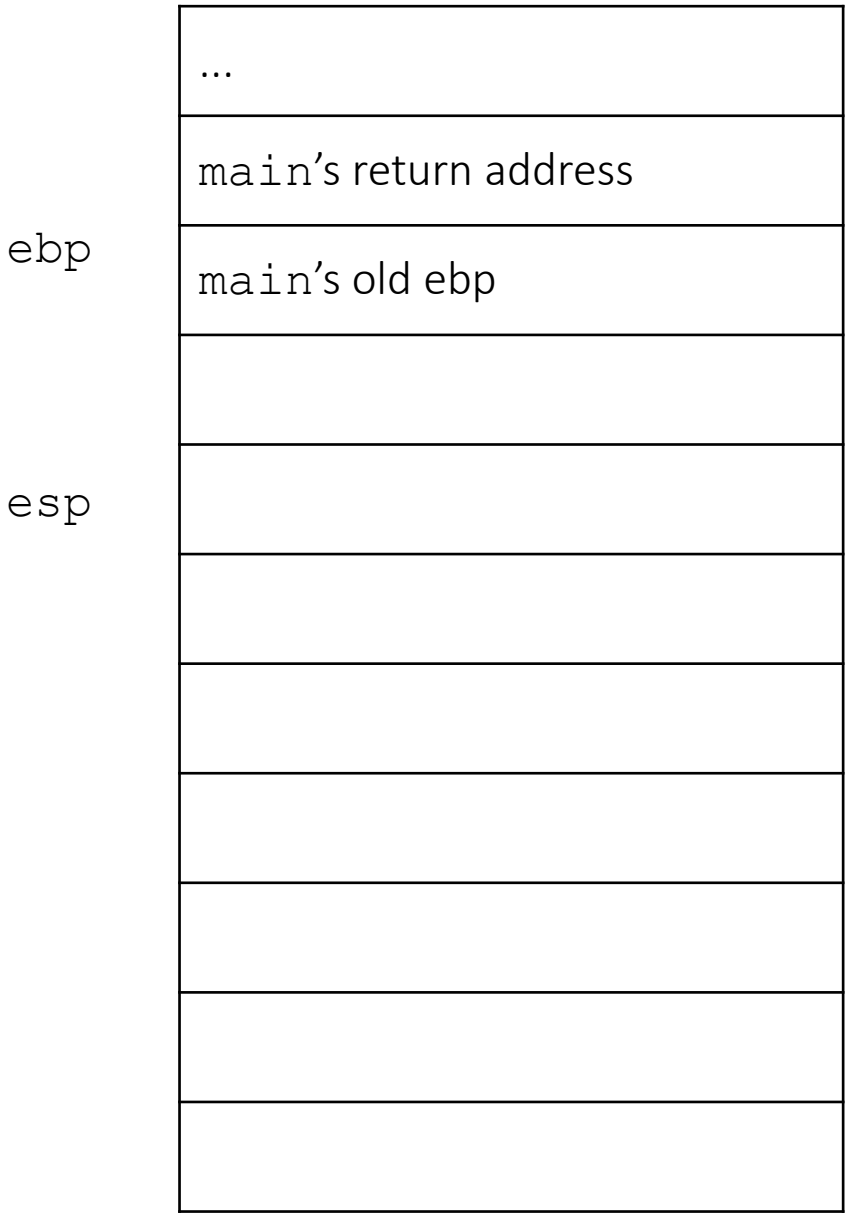

```
; add
0x08048426 <+0>:        push   ebp
0x08048427 <+1>:        mov    ebp,esp
0x08048429 <+3>:        mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:        mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:        add    eax,edx
0x08048431 <+11>:       pop    ebp
0x08048432 <+12>:       ret

; main
0x08048403 <+0>:        push   ebp
0x08048404 <+1>:        mov    ebp,esp
0x08048406 <+3>:        sub    esp,0x8
0x08048409 <+6>:        mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:       mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:       push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:       push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:       call   0x80483f6 <add>
0x08048422 <+31>:       add    esp,0x8
0x08048425 <+34>:       mov    eax,0x0
0x0804842a <+39>:       leave
0x0804842b <+40>:       ret
```

eip

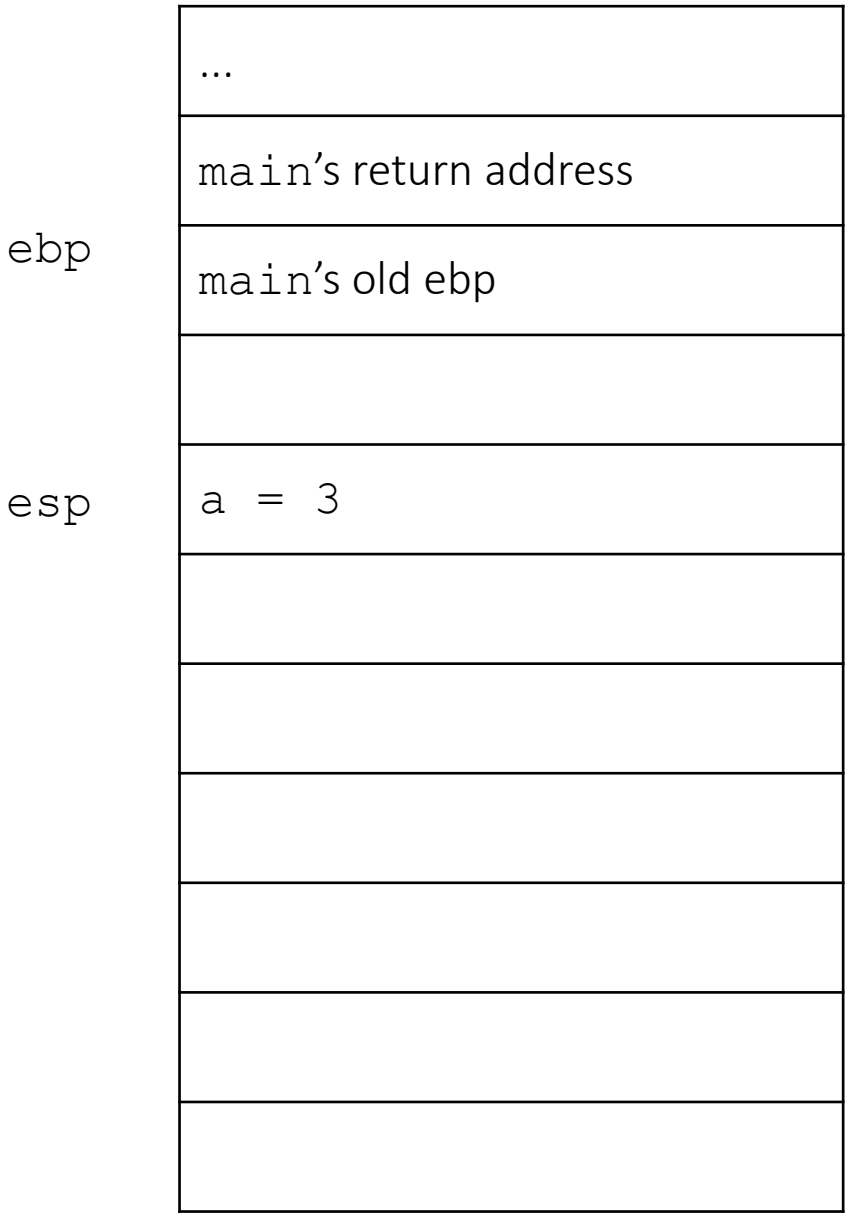

```
; add
0x08048426 <+0>:        push   ebp
0x08048427 <+1>:        mov    ebp,esp
0x08048429 <+3>:        mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:        mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:        add    eax,edx
0x08048431 <+11>:       pop    ebp
0x08048432 <+12>:       ret

; main
0x08048403 <+0>:        push   ebp
0x08048404 <+1>:        mov    ebp,esp
0x08048406 <+3>:        sub    esp,0x8
0x08048409 <+6>:        mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:       mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:       push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:       push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:       call   0x80483f6 <add>
0x08048422 <+31>:       add    esp,0x8
0x08048425 <+34>:       mov    eax,0x0
0x0804842a <+39>:       leave
0x0804842b <+40>:       ret
```



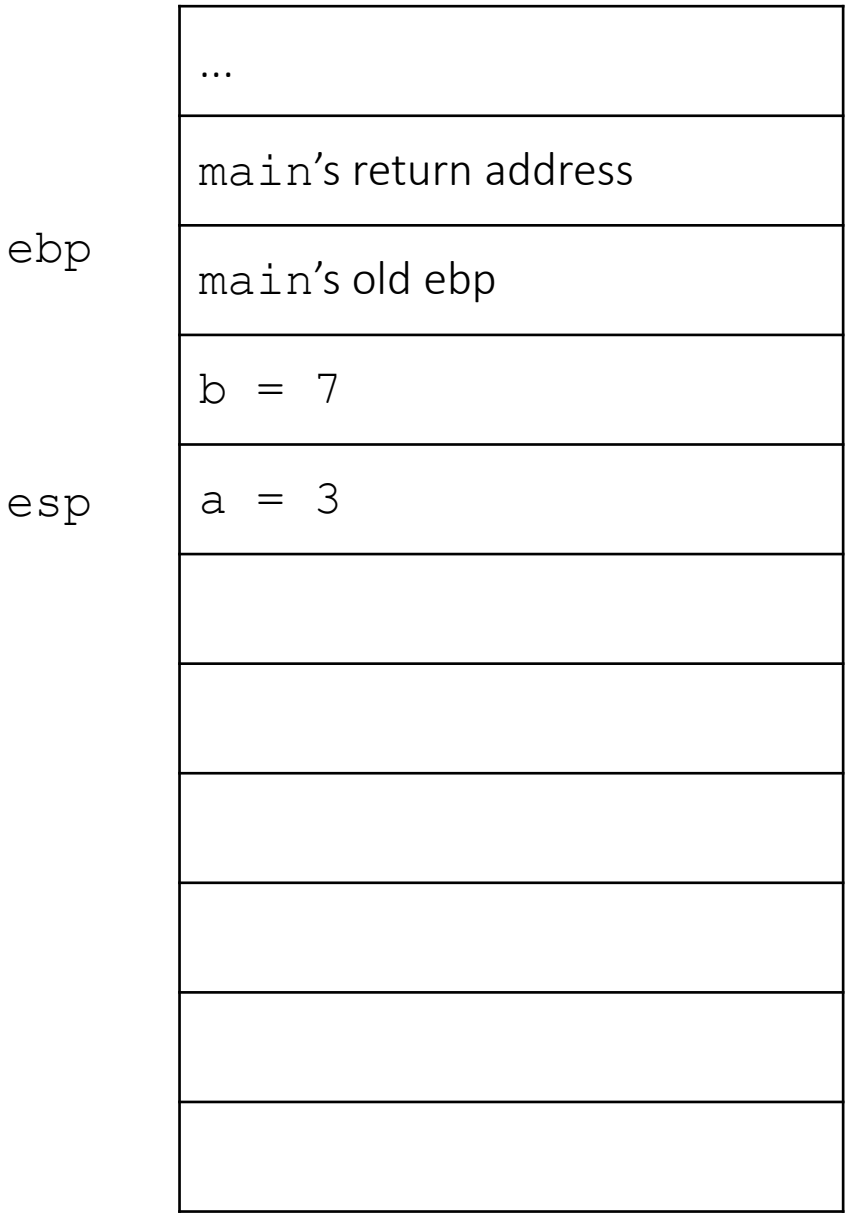

```
; add
0x08048426 <+0>:        push   ebp
0x08048427 <+1>:        mov    ebp,esp
0x08048429 <+3>:        mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:        mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:        add    eax,edx
0x08048431 <+11>:       pop    ebp
0x08048432 <+12>:       ret

; main
0x08048403 <+0>:        push   ebp
0x08048404 <+1>:        mov    ebp,esp
0x08048406 <+3>:        sub    esp,0x8
0x08048409 <+6>:        mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:       mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:       push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:       push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:       call   0x80483f6 <add>
0x08048422 <+31>:       add    esp,0x8
0x08048425 <+34>:       mov    eax,0x0
0x0804842a <+39>:       leave
0x0804842b <+40>:       ret
```



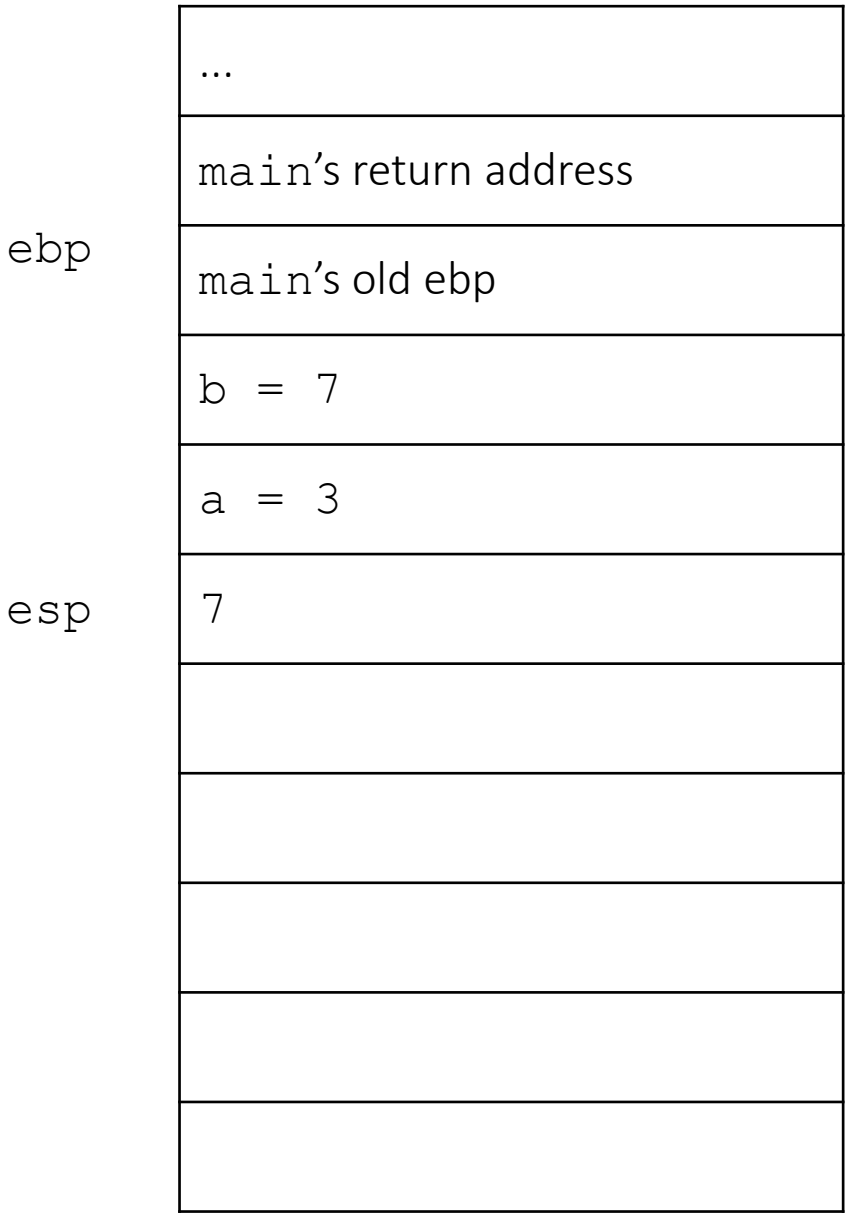

```
; add
0x08048426 <+0>:        push   ebp
0x08048427 <+1>:        mov    ebp,esp
0x08048429 <+3>:        mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:        mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:        add    eax,edx
0x08048431 <+11>:       pop    ebp
0x08048432 <+12>:       ret

; main
0x08048403 <+0>:        push   ebp
0x08048404 <+1>:        mov    ebp,esp
0x08048406 <+3>:        sub    esp,0x8
0x08048409 <+6>:        mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:       mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:       push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:       push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:       call   0x80483f6 <add>
0x08048422 <+31>:       add    esp,0x8
0x08048425 <+34>:       mov    eax,0x0
0x0804842a <+39>:       leave
0x0804842b <+40>:       ret
```



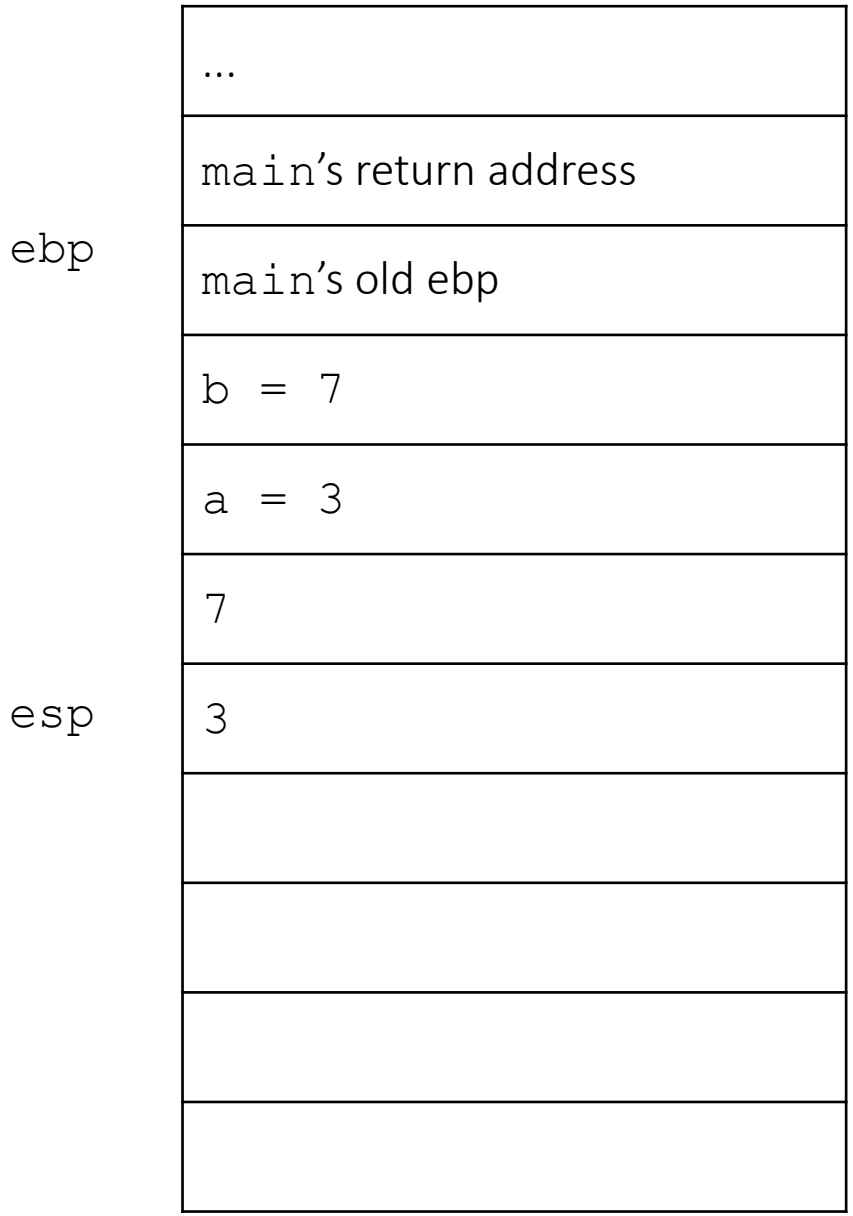

```
; add
0x08048426 <+0>:        push   ebp
0x08048427 <+1>:        mov    ebp,esp
0x08048429 <+3>:        mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:        mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:        add    eax,edx
0x08048431 <+11>:       pop    ebp
0x08048432 <+12>:       ret

; main
0x08048403 <+0>:        push   ebp
0x08048404 <+1>:        mov    ebp,esp
0x08048406 <+3>:        sub    esp,0x8
0x08048409 <+6>:        mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:       mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:       push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:       push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:       call   0x80483f6 <add>
0x08048422 <+31>:       add    esp,0x8
0x08048425 <+34>:       mov    eax,0x0
0x0804842a <+39>:       leave
0x0804842b <+40>:       ret
```



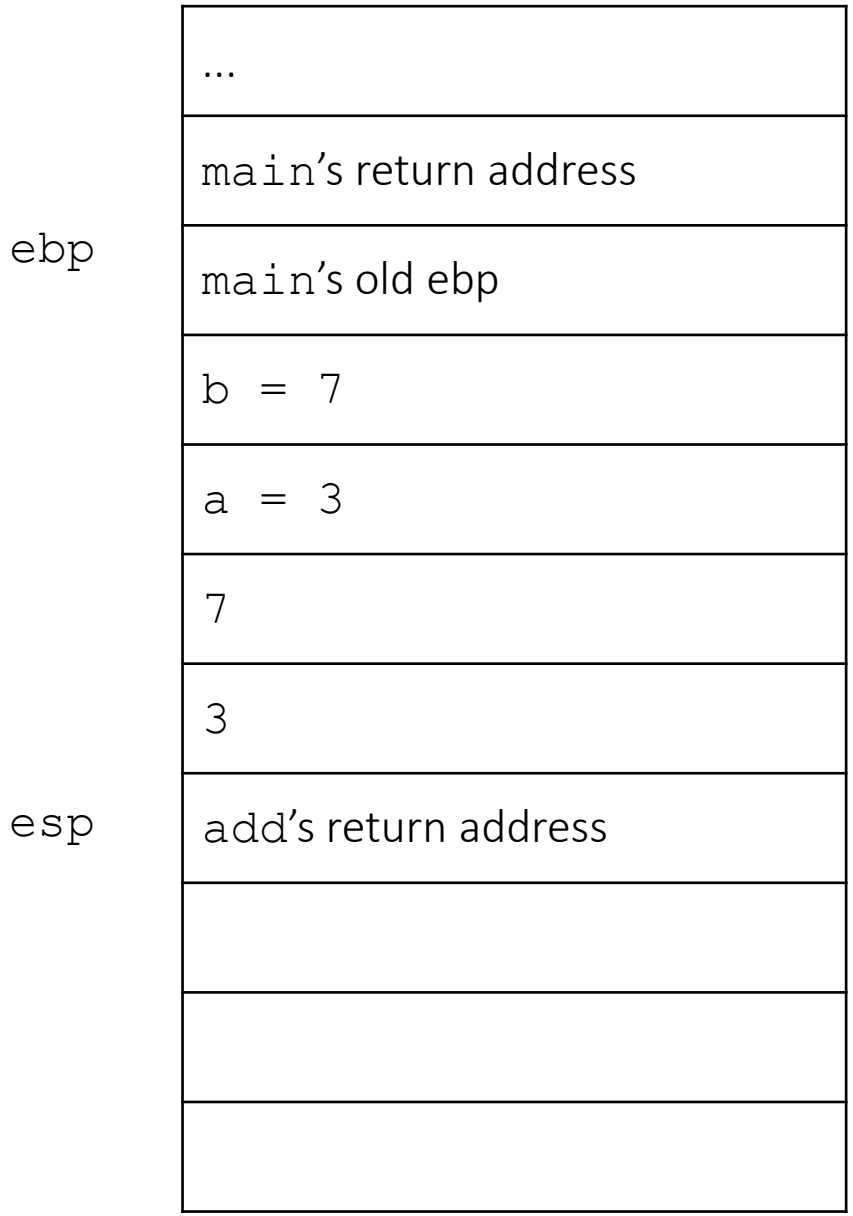

```
; add
0x08048426 <+0>:        push   ebp
0x08048427 <+1>:        mov    ebp,esp
0x08048429 <+3>:        mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:        mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:        add    eax,edx
0x08048431 <+11>:       pop    ebp
0x08048432 <+12>:       ret

; main
0x08048403 <+0>:        push   ebp
0x08048404 <+1>:        mov    ebp,esp
0x08048406 <+3>:        sub    esp,0x8
0x08048409 <+6>:        mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:       mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:       push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:       push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:       call   0x80483f6 <add>
0x08048422 <+31>:       add    esp,0x8
0x08048425 <+34>:       mov    eax,0x0
0x0804842a <+39>:       leave
0x0804842b <+40>:       ret
```



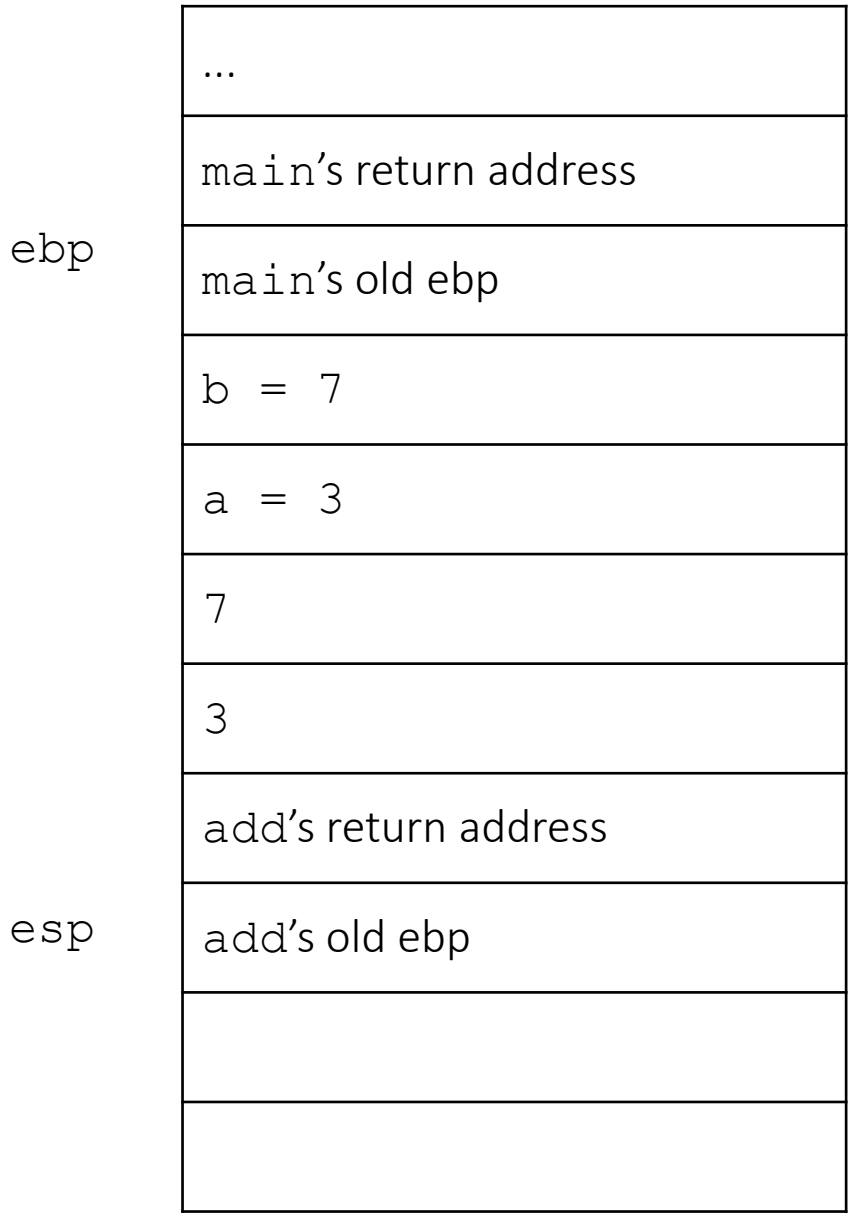

```
; add
0x08048426 <+0>:        push   ebp
0x08048427 <+1>:        mov    ebp,esp
0x08048429 <+3>:        mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:        mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:        add    eax,edx
0x08048431 <+11>:       pop    ebp
0x08048432 <+12>:       ret

; main
0x08048403 <+0>:        push   ebp
0x08048404 <+1>:        mov    ebp,esp
0x08048406 <+3>:        sub    esp,0x8
0x08048409 <+6>:        mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:       mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:       push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:       push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:       call   0x80483f6 <add>
0x08048422 <+31>:       add    esp,0x8
0x08048425 <+34>:       mov    eax,0x0
0x0804842a <+39>:       leave
0x0804842b <+40>:       ret
```



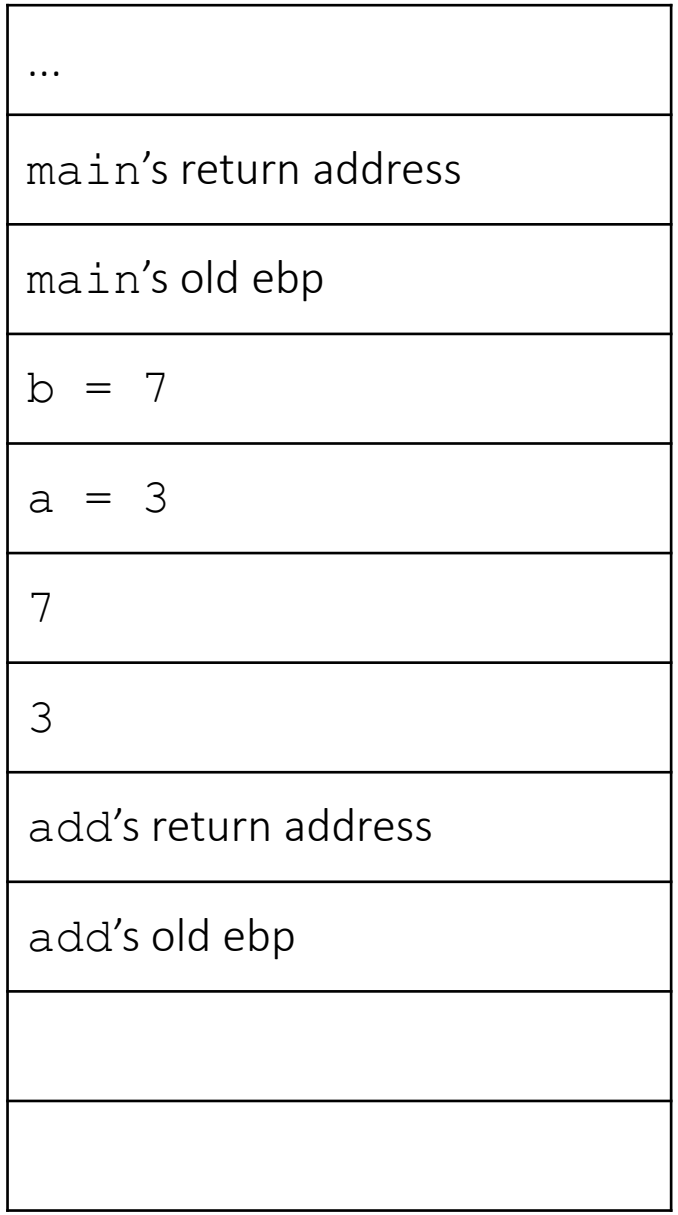

```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:     mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```



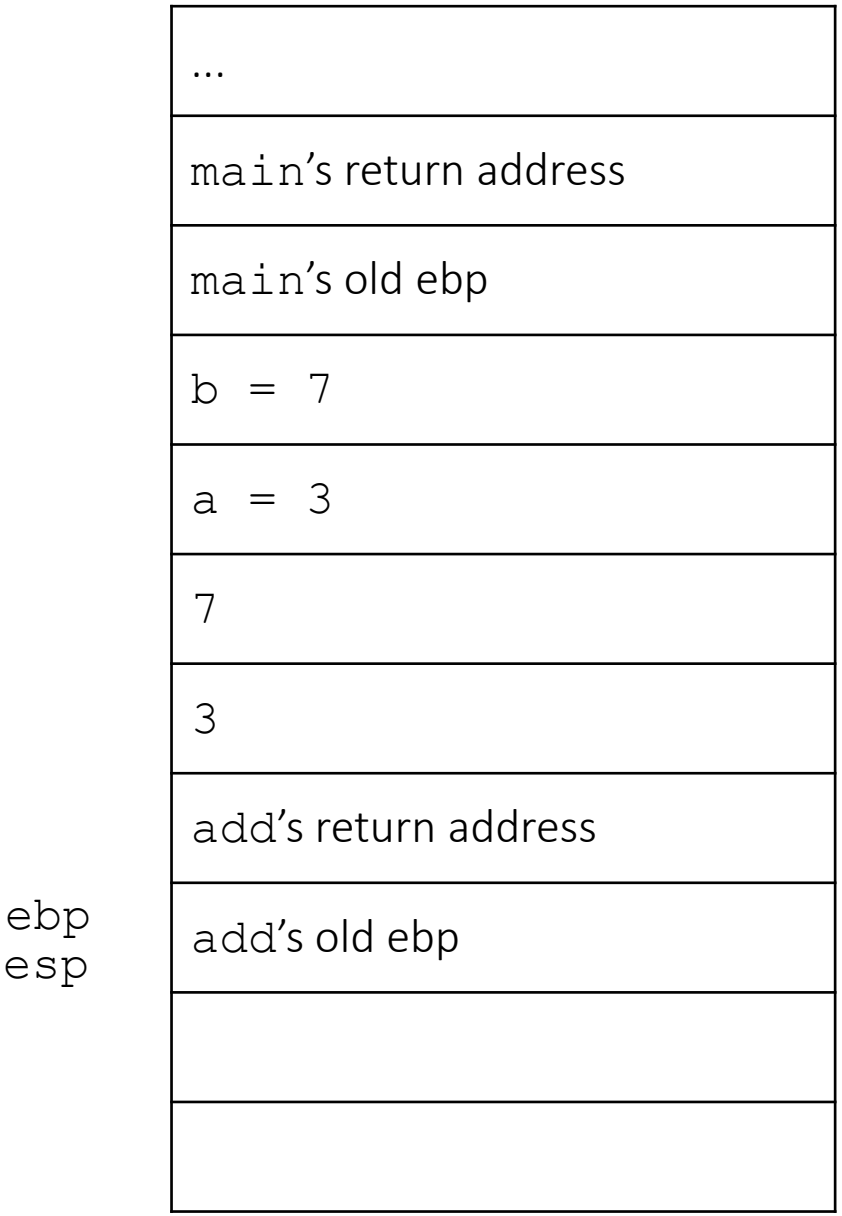

```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:     mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```




```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:     mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```

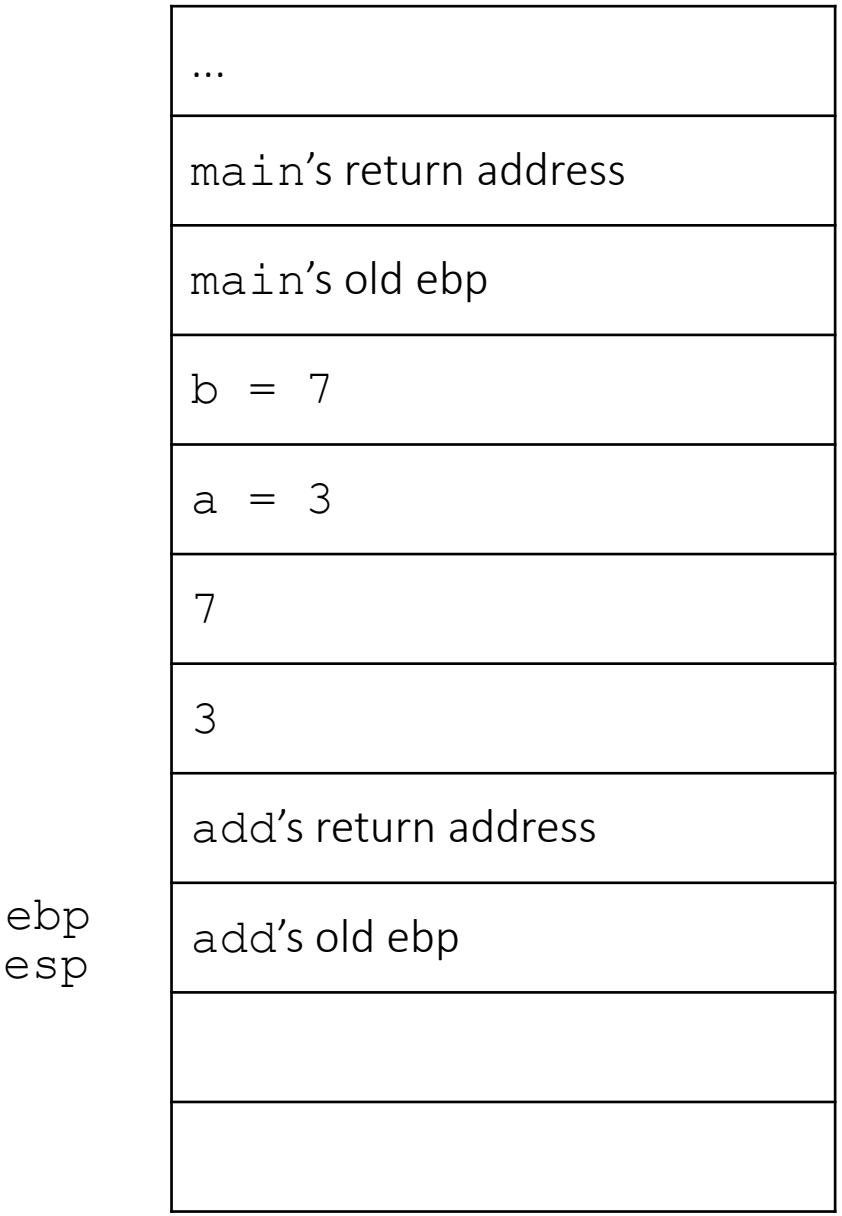



```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:     mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```



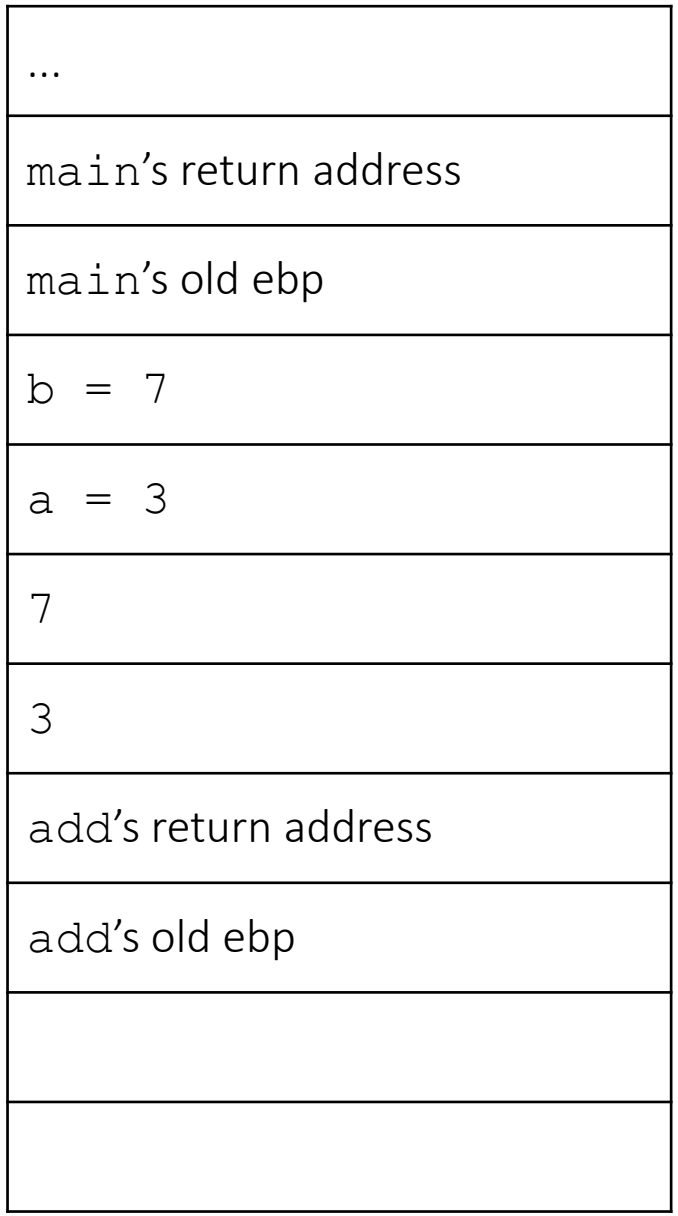

```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:     mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```

edx = 3



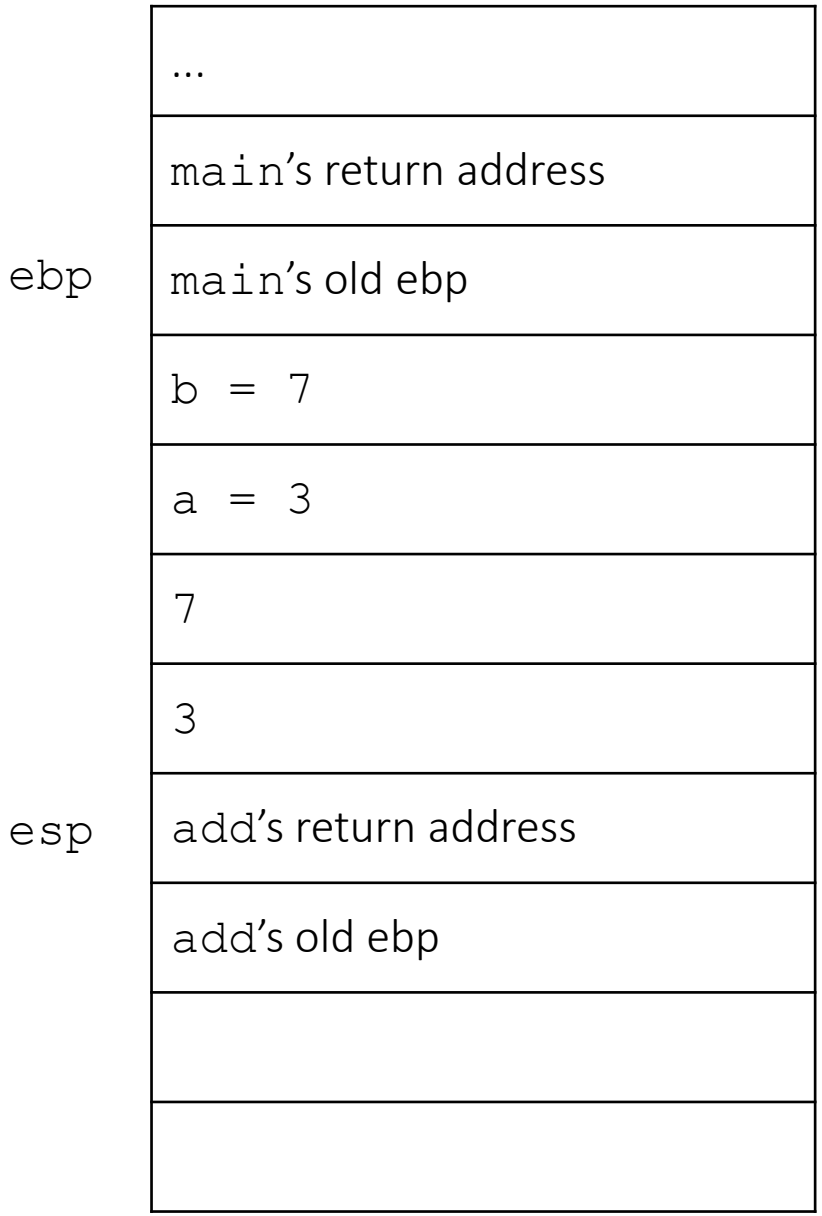

```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:     mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```

eax = 7

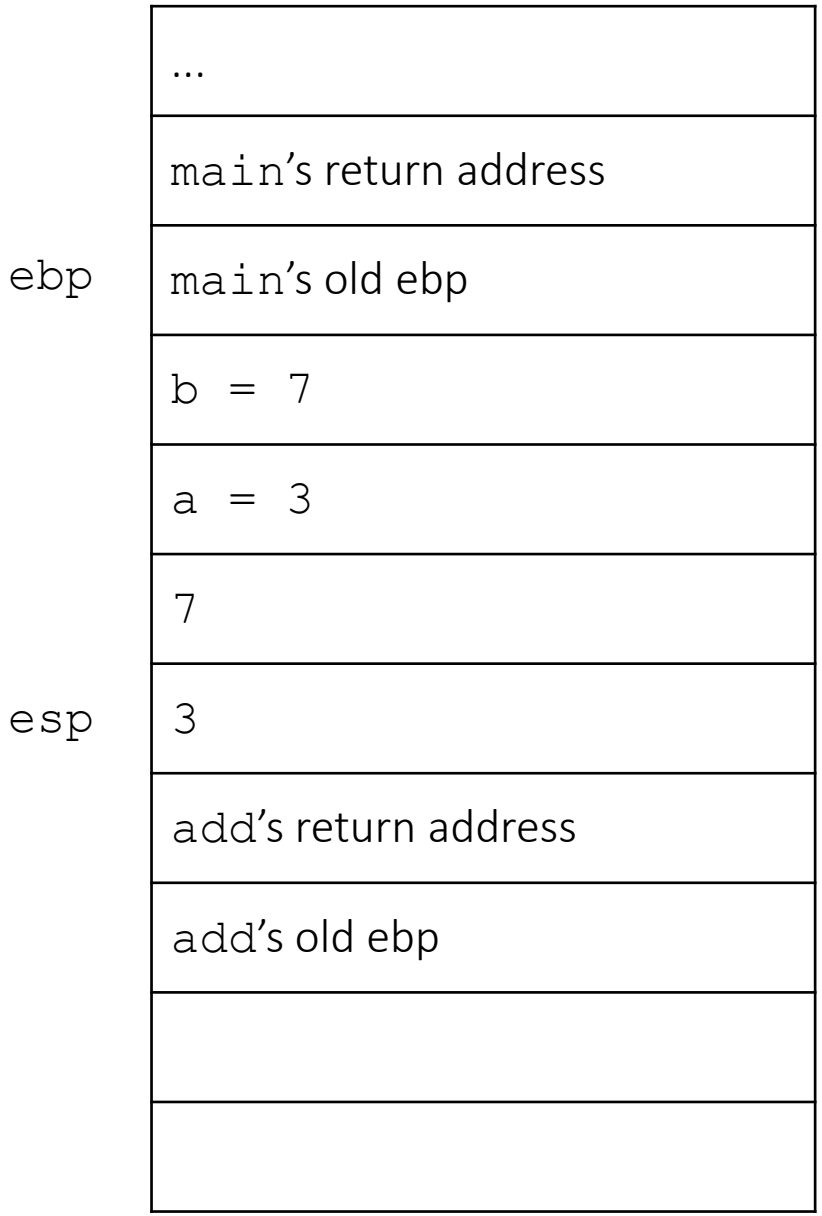



```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD P
0x0804842c <+6>:     mov    eax,DWORD
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```

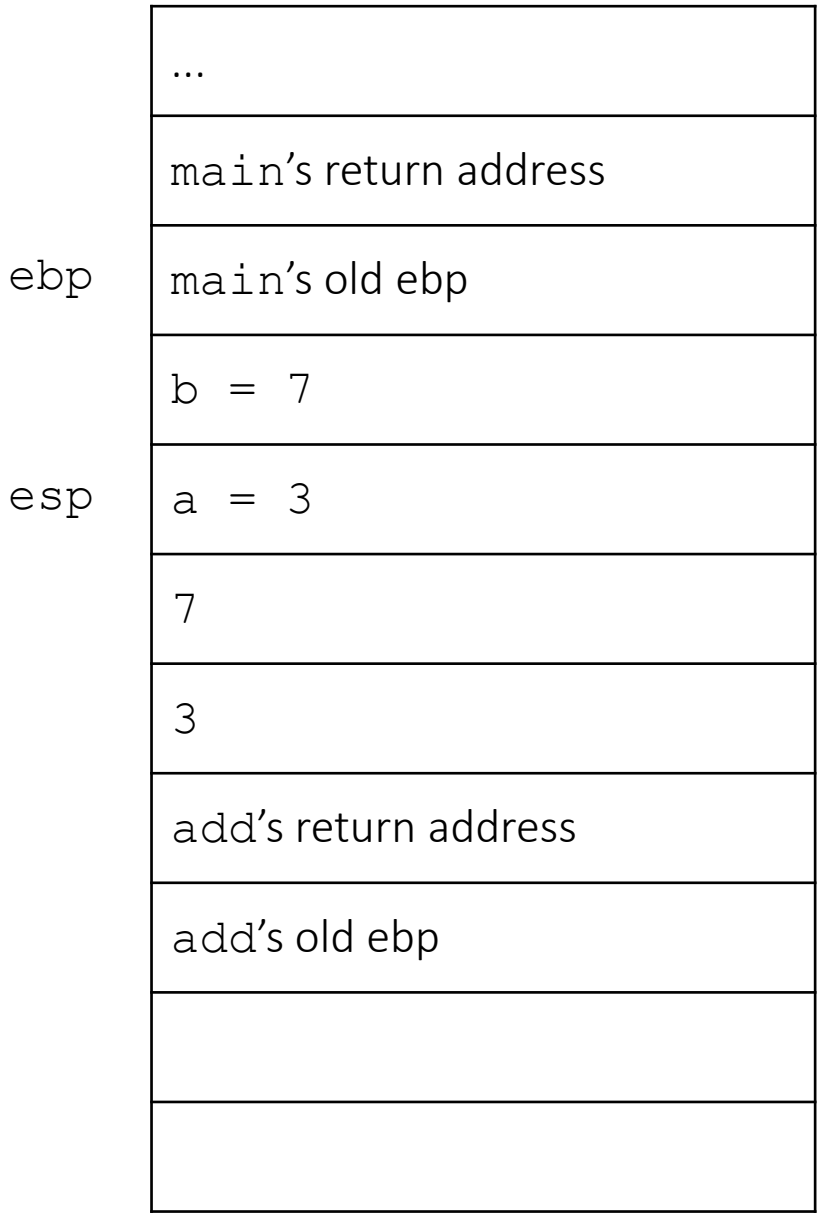
eax
= 7 + 3 = 10



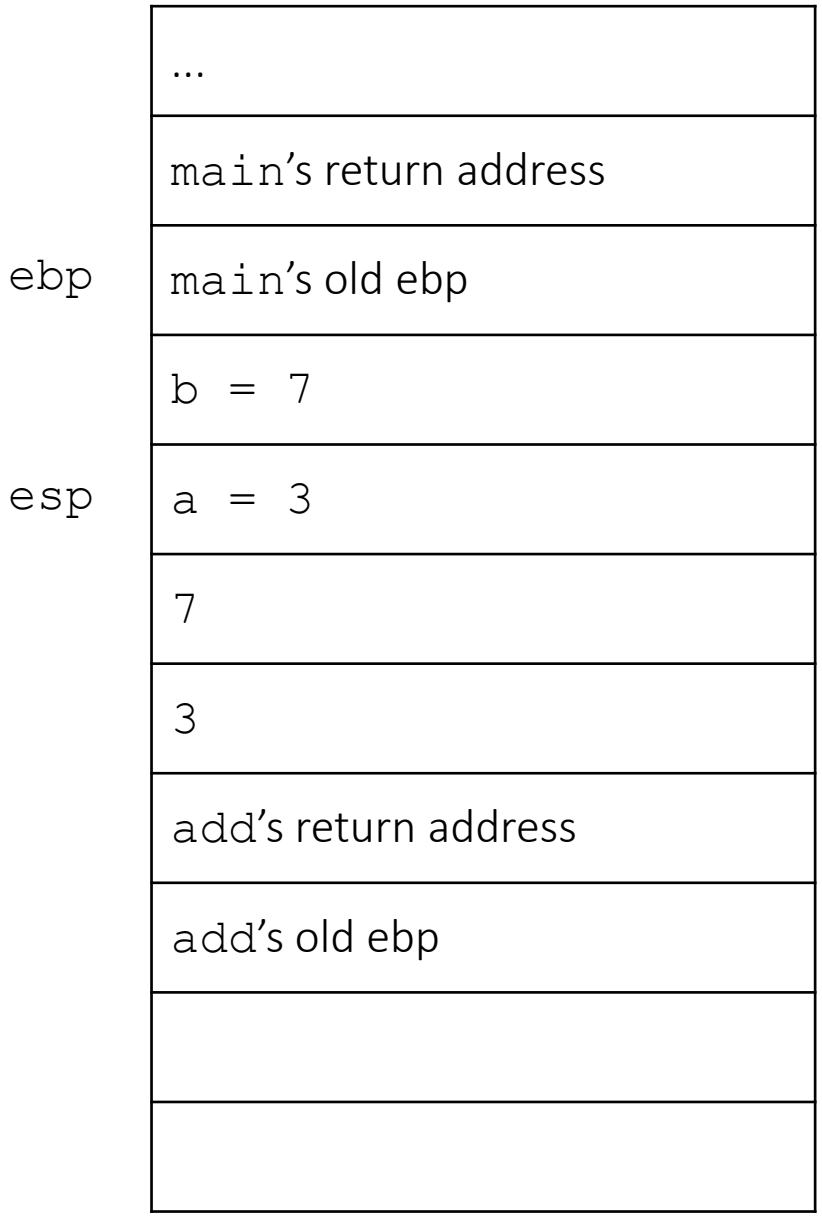

```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:     mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```



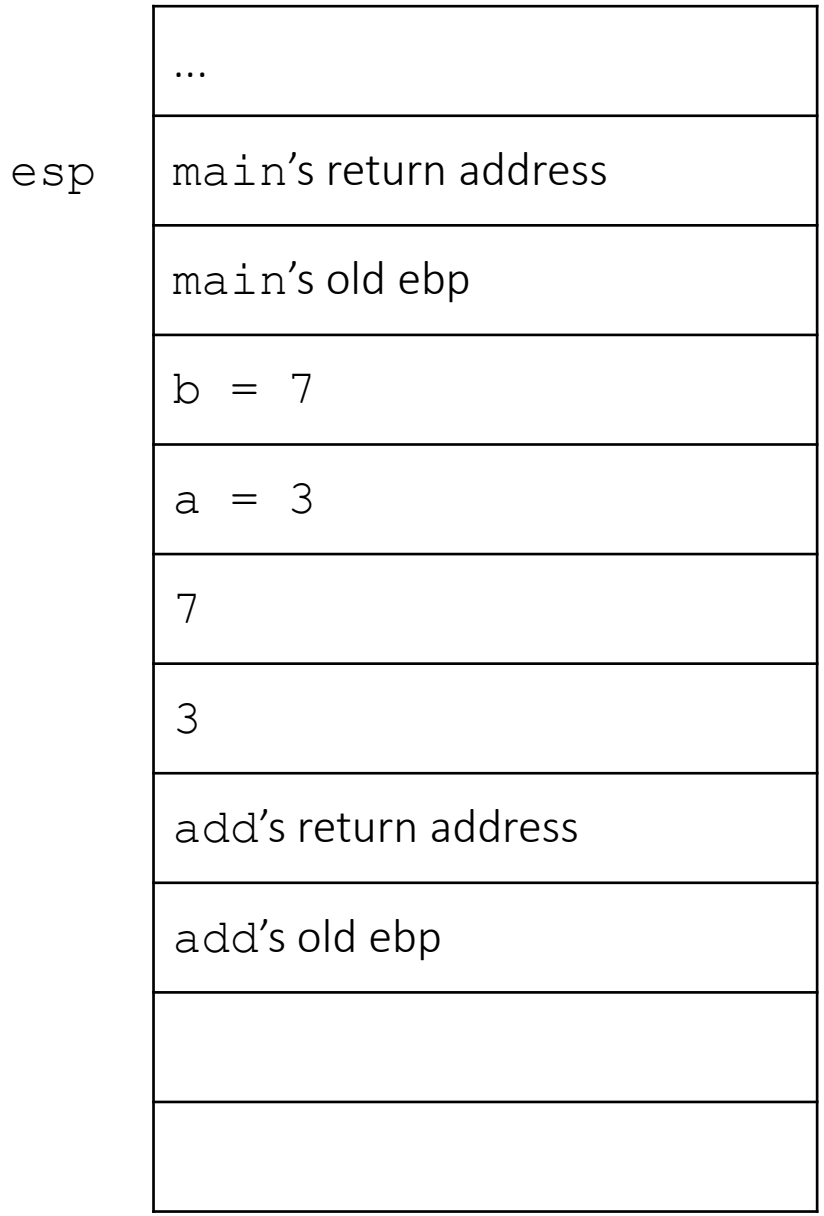

```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:     mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```




```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:     mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```




```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:     mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```




```
; add
0x08048426 <+0>:     push   ebp
0x08048427 <+1>:     mov    ebp,esp
0x08048429 <+3>:     mov    edx,DWORD PTR [ebp+0x8]
0x0804842c <+6>:     mov    eax,DWORD PTR [ebp+0xc]
0x0804842f <+9>:     add    eax,edx
0x08048431 <+11>:    pop    ebp
0x08048432 <+12>:    ret

; main
0x08048403 <+0>:     push   ebp
0x08048404 <+1>:     mov    ebp,esp
0x08048406 <+3>:     sub    esp,0x8
0x08048409 <+6>:     mov    DWORD PTR [ebp-0x8],0x3
0x08048410 <+13>:    mov    DWORD PTR [ebp-0x4],0x7
0x08048417 <+20>:    push   DWORD PTR [ebp-0x4]
0x0804841a <+23>:    push   DWORD PTR [ebp-0x8]
0x0804841d <+26>:    call   0x80483f6 <add>
0x08048422 <+31>:    add    esp,0x8
0x08048425 <+34>:    mov    eax,0x0
0x0804842a <+39>:    leave
0x0804842b <+40>:    ret
```

x86 Calling convention (GCC)

- Function arguments
  - Save to stack
- Return value
  - eax
- Register preservation
  - Callee-saved: ebp, edi, esi, ebx
  - Caller-saved: others

x86 (32bit)

- Function arguments
  - rdi, rsi, rdx, rcx, r8, r9
  - Save to stack
- Return value
  - rax
- Register preservation
  - Callee-saved: rbp, rdi, rsi, rbx, r12-r15
  - Caller-saved: others

x86-64

Tool for dynamic analysis: GNU DeBugger (GDB)

- Can read assembly of a program
  - `disassemble main`
- Can read the status of a program such as registers and memory
  - `x/[Length?][Format] [Expression]`
  - `x/wx $eax` (print the value of eax as 32-bit integer)
  - `x/s 0x804858f` (read the string value at the address 0x804858f)
  - `x/wx 0x804858f` (read the integer value at the address 0x804858f)
- Can set a breakpoint
  - `b main` (break if main function is called)
  - `b *main+53` (break before running the instruction at main+53)

GDB cont.

- Execute a program
  - `r`
- Controlling the execution after a break
  - `c` (continue to a next breakpoint)
  - `ni` (run a instruction, do not get into the function)
  - `si` (run an instruction, get into the function)

pwndbg

- A gdb plugin for exploit development
- We highly recommend to use pwndbg (which is install by default) for your further assignment
- Ref: https://github.com/pwndbg/pwndbg

Tool for static analysis: Ghidra

- Ghidra: A decompiler developed by NSA
  - IDA would be better, but too expensive ☹

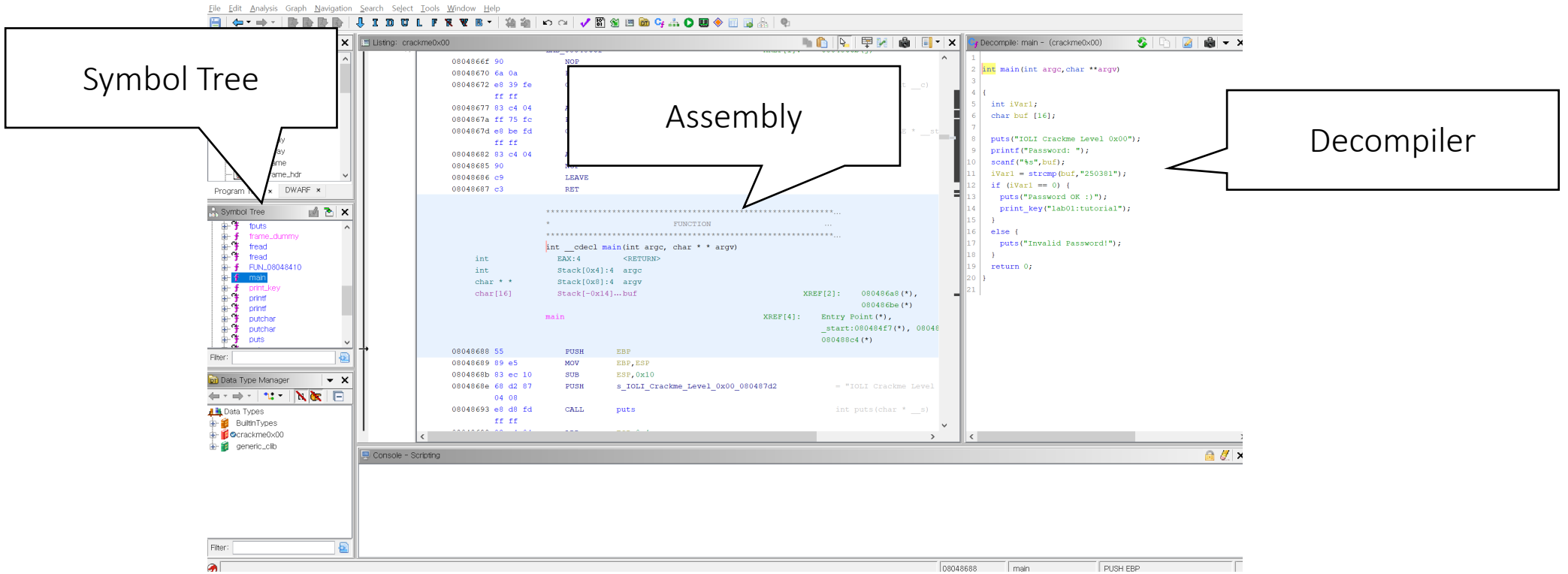
- You can download https://ghidra-sre.org/

How to use ghidra

- Create a new project: File -> New project
- Import a file: File -> Import a file
  - e.g., scp -P9000 YOUR_ID@teemo.kaist.ac.kr:/ee517/lab01/tut01-crackme/crackme0x00 ./
  - NOTE: -P is a capital character unlike ssh
  - Or you can use filezilla for ssh copy

How to use ghidra

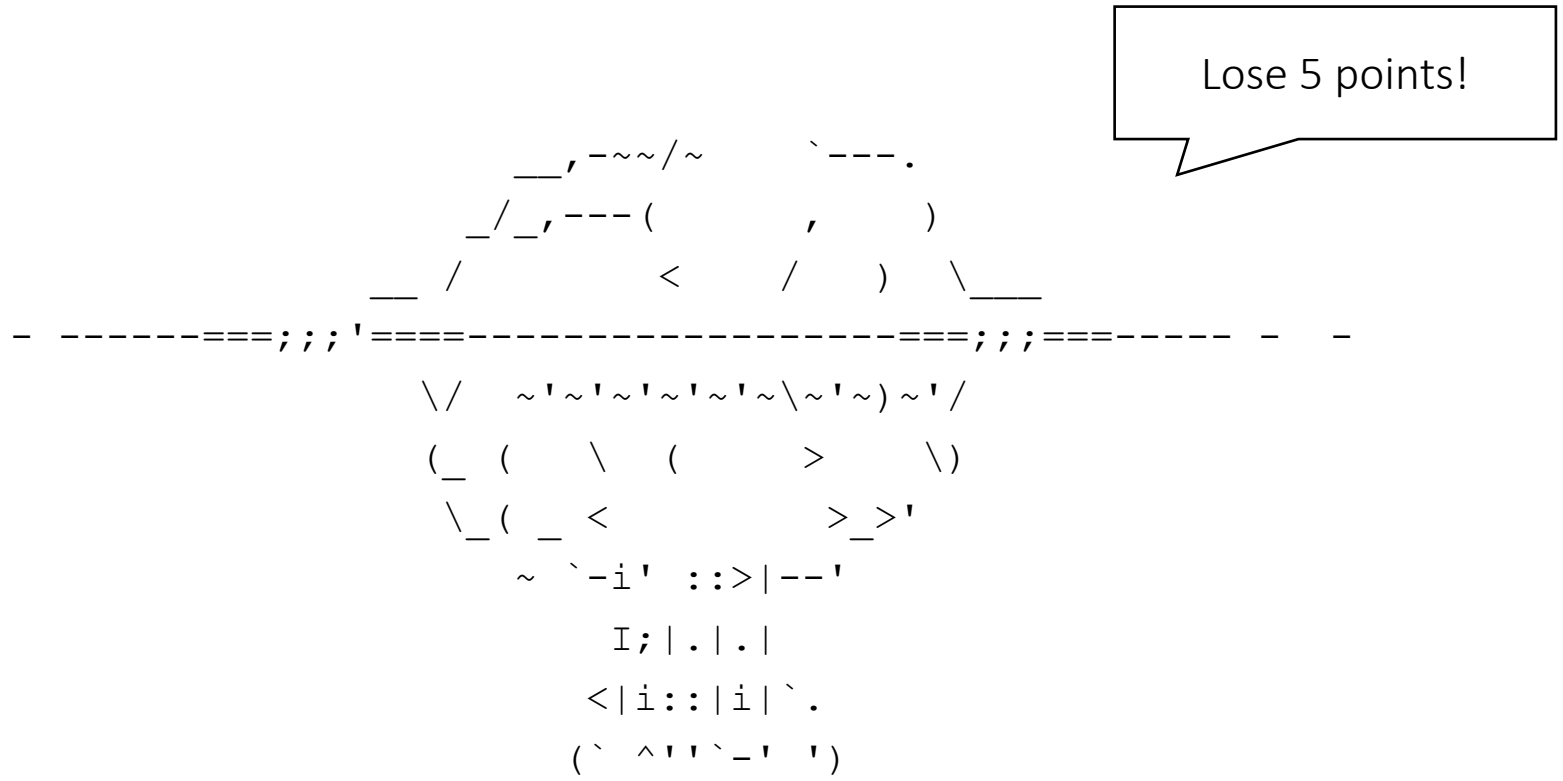
- Symbol tree -> Functions -> main



Decompiler

```
int main(int argc,char **argv)

{
  int iVar1;
  char buf [16];

  puts("IOLI Crackme Level 0x00");
  printf("Password: ");
  scanf("%s",buf);
  iVar1 = strcmp(buf,"250381");
  if (iVar1 == 0) {
    puts("Password OK :)");
    print_key("lab01:tutorial");
  }
  else {
    puts("Invalid Password!");
  }
  return 0;
}
```

Note on bomb



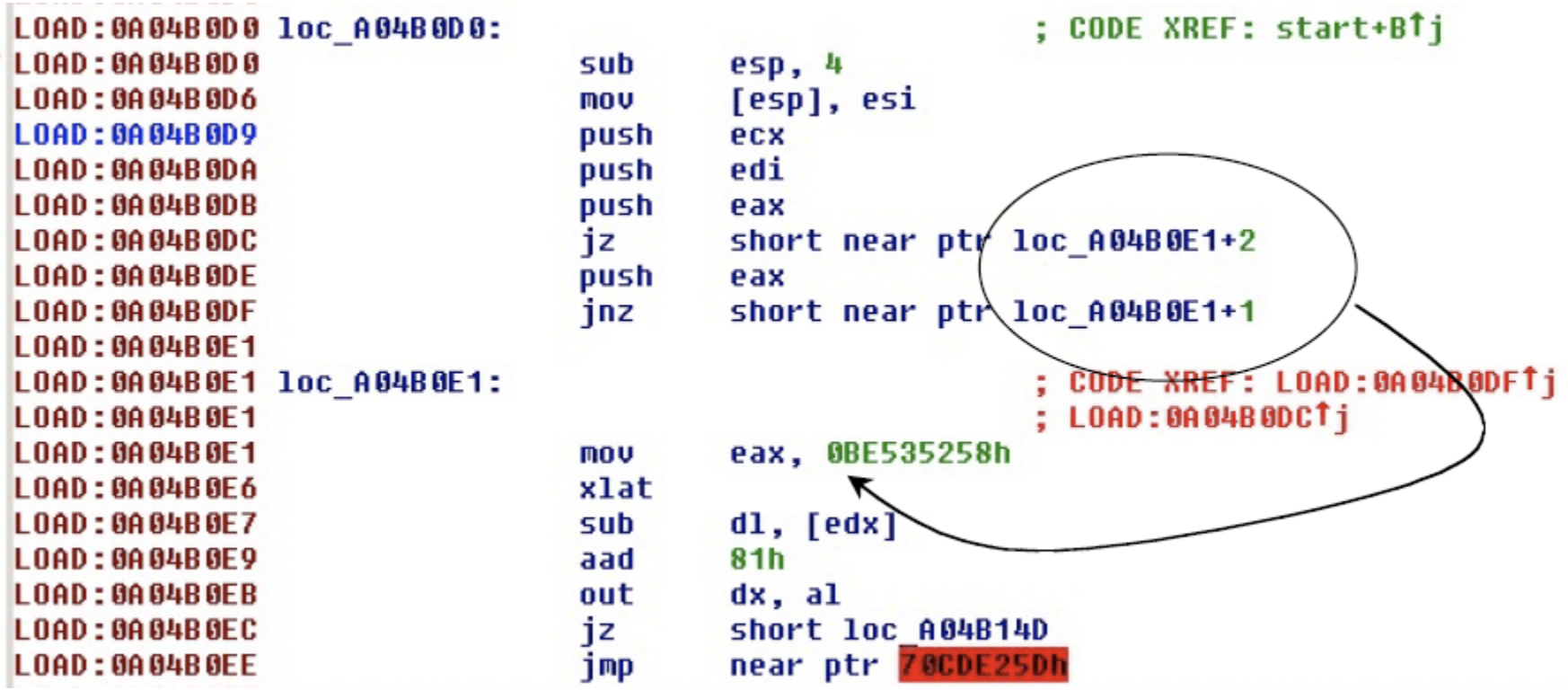
BOOM!! The bomb has blown up.

Tip: Try to read assembly as much as you can!

- Recommend: Solve *first THREE* challenges in bomblab without decompiler

- Reason:
  - In shellcode lab, you need to write assembly code
  - Writing exploit requires understanding of assembly (e.g., ROP)
  - We have tedious binaries in later lab challenges, which make a decompiler fail to analyze

Anti-reversing



https://marcoramilli.com/2011/02/11/disassembly-desynchronization-overview/