CS 354
Machine Organization and Programming
Lecture 19

Michael Doescher
Summer 2020

Function Calls

Accessing Memory

Registers
%eax: 0x100
%ecx:

Assembly
movl %eax, %ecx

```
Memory

0x100 : 42 :
0x104 : 35 :
0x108 : 82 :
0x10C : 16 :
```

Accessing Memory

Registers
%eax: 0x100
%ecx: 0x100

Assembly
movl %eax, %ecx

```
Memory

0x100 : 42 :
0x104 : 35 :
0x108 : 82 :
0x10C : 16 :
```

Accessing Memory

Registers
%eax: 0x100
%ecx: 0x100

Assembly
movl %eax, %ecx
movl (%eax), %ecx

```
Memory

0x100 : 42 :
0x104 : 35 :
0x108 : 82 :
0x10C : 16 :
```

Accessing Memory

Registers
%eax: 0x100
%ecx: 42

Assembly
movl %eax, %ecx
movl (%eax), %ecx

```
Memory

0x100 : 42 :
0x104 : 35 :
0x108 : 82 :
0x10C : 16 :
```

Accessing Memory

Registers
%eax: 0x100
%ecx: 42

Assembly
movl %eax, %ecx
movl (%eax), %ecx
movl 4(%eax), %ecx

```
Memory

0x100 : 42 :
0x104 : 35 :
0x108 : 82 :
0x10C : 16 :
```

Accessing Memory

Registers
%eax: 0x100
%ecx: 35

Assembly
movl %eax, %ecx
movl (%eax), %ecx
movl 4(%eax), %ecx

```
Memory

0x100 : 42 :
0x104 : 35 :
0x108 : 82 :
0x10C : 16 :
```

Control Flow : CS:APP 3.6-7

- Sequential
- Conditional
- Iteration
- Functions

What Features do we need in Assembly / Hardware to implement control flow?

Control Flow : CS:APP 3.6-7

- Sequential
- Conditional
- Iteration
- Functions

What Features do we need in Assembly / Hardware to implement control flow?

- cmpl a, b (computes b-a)
- testl a, b (computes b & a)
- Condition Code Register: %eflags
- Jump statements

Control Flow : CS:APP 3.6-7

- Sequential
- Conditional
- Iteration
- Functions

What Features do we need in Assembly / Hardware to implement control flow?

- cmpl a, b (computes b-a)
- testl a, b (computes b & a)
- Condition Code Register
- Jump statements
- Stack Pointer, Base Pointer
- Stack
- call, leave, return, push, pop

Functions: CSAPP 3.7

```
 1 int sum(int x, int y) {
 2     int total;
 3     total = x + y;
 4     return total;
 5 }
 6
 7 int main() {
 8     int s = sum(1,2);
 9     return 0;
10 }
```

Functions: CSAPP 3.7

```
 1 int sum(int x, int y) {
 2     int total;
 3     total = x + y;
 4     return total;
 5 }
 6
 7 int main() {
 8     int s = sum(1,2);
 9     return 0;
10 }
```

```
main()  -> caller
sum()   -> callee
```

Functions: CSAPP 3.7

```
 1 int sum(int x, int y) {
 2     int total;
 3     total = x + y;
 4     return total;
 5 }
 6
 7 int main() {
 8     int s = sum(1,2);
 9     return 0;
10 }
```

1. How to call a function

Functions: CSAPP 3.7

```
 1 int sum(int x, int y) {
 2     int total;
 3     total = x + y;
 4     return total;
 5 }
 6
 7 int main() {
 8     int s = sum(1,2);
 9     return 0;
10 }
```

1. How to call a function
2. How to return control back

Functions: CSAPP 3.7

```
 1 int sum(int x, int y) {
 2     int total;
 3     total = x + y;
 4     return total;
 5 }
 6
 7 int main() {
 8     int s = sum(1,2);
 9     return 0;
10 }
```

1. How to call a function
2. How to return control back
3. How to return a value

Functions: CSAPP 3.7

```
 1 int sum(int x, int y) {
 2     int total;
 3     total = x + y;
 4     return total;
 5 }
 6
 7 int main() {
 8     int s = sum(1,2);
 9     return 0;
10 }
```

1. How to call a function
2. How to return control back
3. How to return a value
4. How to pass parameters

Functions: CSAPP 3.7

```
 1 int sum(int x, int y) {
 2     int total;
 3     total = x + y;
 4     return total;
 5 }
 6
 7 int main() {
 8     int s = sum(1,2);
 9     return 0;
10 }
```

1. How to call a function
2. How to return control back
3. How to return a value
4. How to pass parameters
5. How to allocate space for local variables

Functions: CSAPP 3.7

```
 1 int sum(int x, int y) {
 2     int total;
 3     total = x + y;
 4     return total;
 5 }
 6
 7 int main() {
 8     int s = sum(1,2);
 9     return 0;
10 }
```

1. How to call a function
2. How to return control back
3. How to return a value
4. How to pass parameters
5. How to allocate space for local variables
6. How to access parameters and local variables

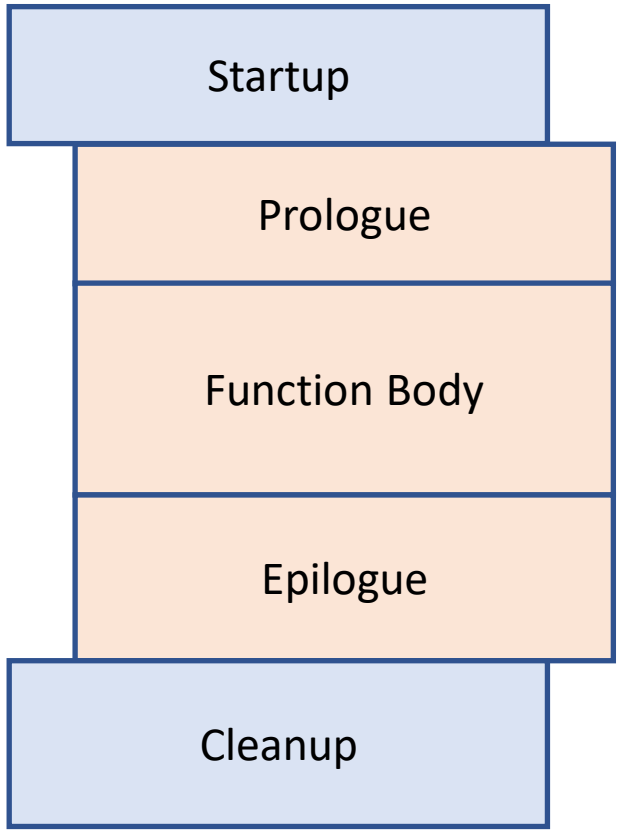
Functions: CSAPP 3.7

```
 1 int sum(int x, int y) {
 2     int total;
 3     total = x + y;
 4     return total;
 5 }
 6
 7 int main() {
 8     int s = sum(1,2);
 9     return 0;
10 }
```

1. How to call a function
2. How to return control back
3. How to return a value
4. How to pass parameters
5. How to allocate space for local variables
6. How to access parameters and local variables
7. How to deal with registers

Calling Conventions

```
 1 int sum(int x, int y) {
 2     int total;
 3     total = x + y;
 4     return total;
 5 }
 6
 7 int main() {
 8     int s = sum(1,2);
 9     return 0;
10 }
```



How to call / return

```
1 void func() {
2     return;
3 }
4
5 int main() {
6     func();
7     return 0;
8 }
```

```
main:
0x100 | instr 1 | %eip
0x104 | instr 2
0x108 | call func
0x10C | instr 3

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret



0xFA0 |
0xFA4 |
0xFA8 | stack | <-%esp
```

How to call / return

```
1 void func() {
2     return;
3 }
4
5 int main() {
6     func();
7     return 0;
8 }
```

Registers

%eax:

%eip: 0x100

%esp: 0xFA8

%ebp

```
main:
0x100 | instr 1 | %eip
0x104 | instr 2
0x108 | call func
0x10C | instr 3

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret



0xFA0 |
0xFA4 |
0xFA8 | stack | <-%esp
```

How to call / return

```
1 void func() {
2     return;
3 }
4
5 int main() {
6     func();
7     return 0;
8 }
```

Call
instruction pointer %eip holds the address of the next instruction

Registers
%eax:

%eip: 0x100
%esp: 0xFA8
%ebp

```
main:
0x100 | instr 1 | %eip
0x104 | instr 2
0x108 | call func
0x10C | instr 3

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret



0xFA0 |
0xFA4 |
0xFA8 | stack | <-%esp
```

How to call / return

```
1 void func() {
2     return;
3 }
4
5 int main() {
6     func();
7     return 0;
8 }
```

Call
instruction pointer %eip holds the address of the next instruction

Registers
%eax:

%eip: 0x10C
%esp: 0xFA8
%ebp:

```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3 | %eip

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret



0xFA0 |
0xFA4 |
0xFA8 | stack | <-%esp
```

How to call / return

```
1 void func() {
2     return;
3 }
4
5 int main() {
6     func();
7     return 0;
8 }
```

Call

1. Push return address
   pushl %eip

Registers

%eax:

%eip: 0x10C
%esp: 0xFA8
%ebp:

```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3 | %eip

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret



0xFA0 |
0xFA4 |
0xFA8 | stack | <-%esp
```

How to call / return

```
1 void func() {
2     return;
3 }
4
5 int main() {
6     func();
7     return 0;
8 }
```

Call

1. Push return address
   pushl %eip
   or
   subl $4, %esp
   movl %eip, %esp

Registers

%eax:

%eip: 0x10C
%esp: 0xFA4
%ebp:

```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3 | %eip

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret



0xFA0 |
0xFA4 |         | <-%esp
0xFA8 | stack
```

How to call / return

```
1 void func() {
2     return;
3 }
4
5 int main() {
6     func();
7     return 0;
8 }
```

Call

1. Push return address
   pushl %eip
   or
   subl $4, %esp
   movl %eip, %esp

Registers

%eax:

%eip: 0x10C
%esp: 0xFA4
%ebp:

```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3 | %eip

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret



0xFA0 |
0xFA4 | 0x10C  | <-%esp
0xFA8 | stack
```

How to call / return

```
1 void func() {
2     return;
3 }
4
5 int main() {
6     func();
7     return 0;
8 }
```

Registers

```
%eax:

%eip: 0x204
%esp: 0xFA4
%ebp:
```

Call

1. Push return address
   pushl %eip
   or
   subl $4, %esp
   movl %eip, %esp
2. Transfer control
   jmp <target>

```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3

func:
0x204 | instr 41 | %eip
0x208 | instr 42
0x20C | ret



0xFA0 |
0xFA4 | 0x10C   | <-%esp
0xFA8 | stack
```

How to call / return

```
1 void func() {
2     return;
3 }
4
5 int main() {
6     func();
7     return 0;
8 }
```

Registers

```
%eax:

%eip: 0x210
%esp: 0xFA4
%ebp:
```

Call

1. Push return address
   pushl %eip
   or
   subl $4, %esp
   movl %eip, %esp
2. Transfer control
   jmp <target>

```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret
0x210 | instr 44 | %eip


0xFA0 |
0xFA4 | 0x10C   | <-%esp
0xFA8 | stack
```

How to call / return

```
1 void func() {
2     return;
3 }
4
5 int main() {
6     func();
7     return 0;
8 }
```

Registers

```
%eax:

%eip: 0x210
%esp: 0xFA4
%ebp:
```

Call

1. Push return address
   - pushl %eip
   - or
   - subl $4, %esp
   - movl %eip, %esp
2. Transfer control
   - jmp <target>

Ret

3. Return control
   - ret

```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret
0x210 | instr 44 | %eip


0xFA0 |
0xFA4 | 0x10C   | <-%esp
0xFA8 | stack
```

How to call / return

```
1 void func() {
2     return;
3 }
4
5 int main() {
6     func();
7     return 0;
8 }
```

Registers

```
%eax:

%eip: 0x210
%esp: 0xFA8
%ebp:
```

Call

1. Push return address
   - pushl %eip
   - or
   - subl $4, %esp
   - movl %eip, %esp
2. Transfer control
   - jmp <target>

Ret

3. Return control
   - ret
   - or popl %eip

```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret
0x201 | instr 44 | %eip


0xFA0 |
0xFA4 | 0x10C
0xFA8 | stack  | <-%esp
```

How to call / return

```
1 void func() {
2     return;
3 }
4
5 int main() {
6     func();
7     return 0;
8 }
```

Registers

```
%eax:

%eip: 0x10C
%esp: 0xFA8
%ebp:
```

Call

1. Push return address
   - pushl %eip
   - or
   - subl $4, %esp
   - movl %eip, %esp
2. Transfer control
   - jmp <target>

Ret

3. Return control
   - ret
   - or popl %eip

```
main:
0x100 | instr 1
0x104 | instr 2
0x108 | call func
0x10C | instr 3   | %eip

func:
0x204 | instr 41
0x208 | instr 42
0x20C | ret
0x201 | instr 44


0xFA0 |
0xFA4 | 0x10C
0xFA8 | stack   | <-%esp
```

How to call / return

Call

1. Push return address
   pushl %eip
2. Transfer control
   jmp <target>

Ret

3. Return control
   ret
   or popl %eip

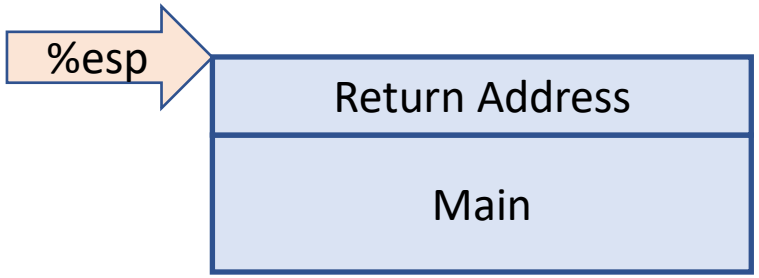
How to call / return

Call

1. Push return address
   pushl %eip
2. Transfer control
   jmp <target>

Ret

3. Return control
   ret
   or popl %eip



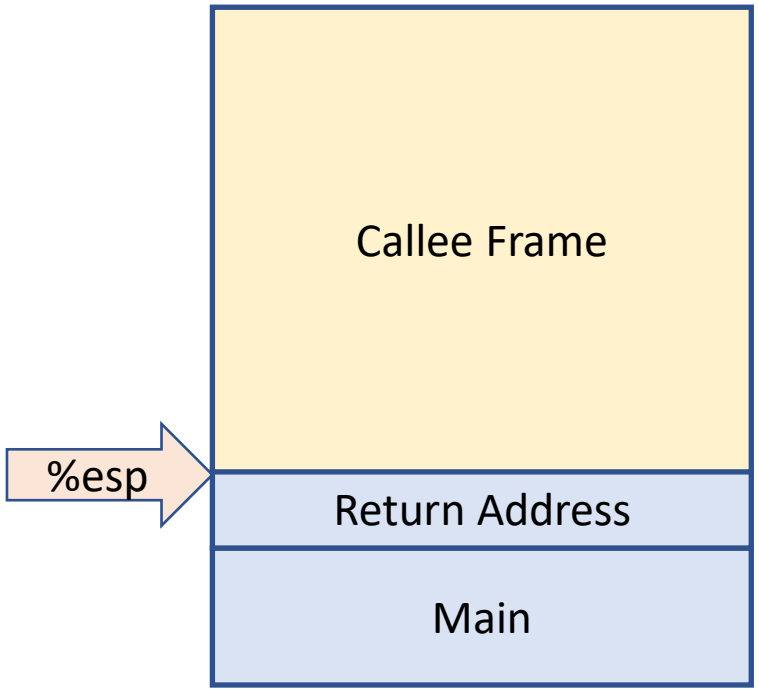
How to call / return

Call

1. Push return address
   pushl %eip
2. Transfer control
   jmp <target>

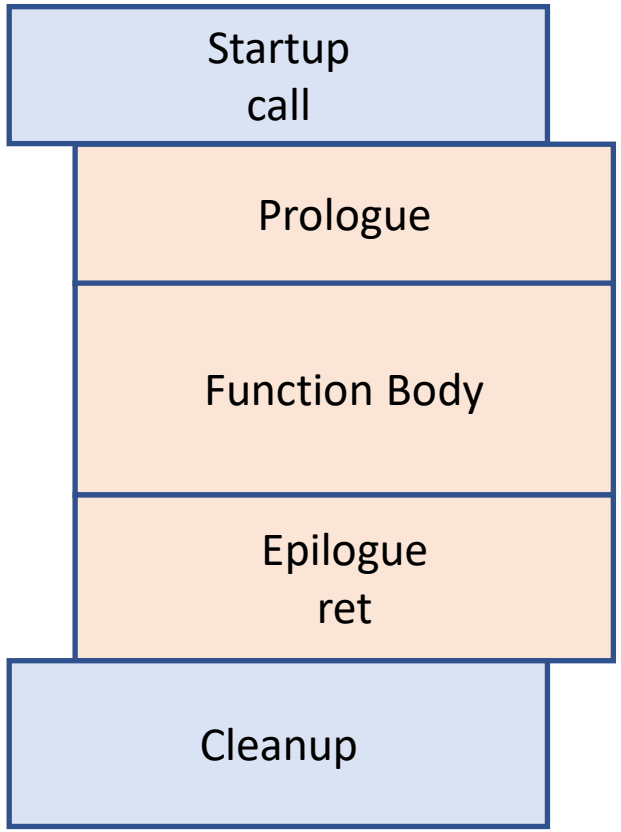
Ret

3. Return control
   ret
   or popl %eip



Calling Conventions

```
 1 int sum(int x, int y) {
 2     int total;
 3     total = x = y;
 4     return total;
 5 }
 6
 7 int main() {
 8     int s = sum(1,2);
 9     return 0;
10 }
```



Return Values

```
 1 int func() {
 2     return 42;
 3 }
 4
 5 int main() {
 6     int answer = 1;
 7     answer = func();
 8     return 0;
 9 }
10
```

1. How to call a function
2. How to return control back
3. **How to return a value**
4. How to pass parameters
5. How to allocate space for local variables
6. How to access parameters and local variables
7. How to deal with registers

Return Values – Method 1: Stack

```c
int func() {
    return 42;
}

int main() {
    int answer = 1;
    answer = func();
    return 0;
}
```

```
func:
movl $42, 4(%esp)
   # prep return value
ret
   # pop return address

main:
movl $1, %eax
subl $4, %esp
   #reserve space
call func
   #pushes return addr
movl (%esp), %eax
```

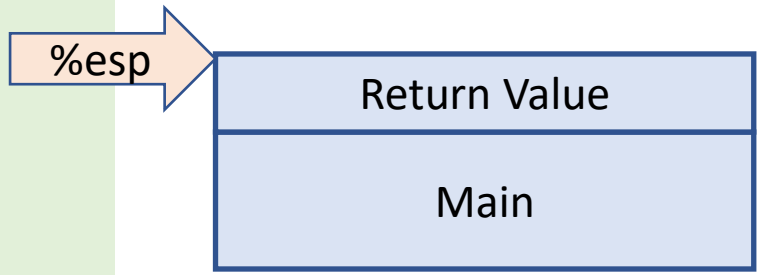
Return Values – Method 1: Stack

```c
int func() {
    return 42;
}

int main() {
    int answer = 1;
    answer = func();
    return 0;
}
```

```
func:
movl $42, 4(%esp)
   # prep return value
ret
   # pop return address

main:
movl $1, %eax
subl $4, %esp
   #reserve space
call func
   #pushes return addr
movl (%esp), %eax
```



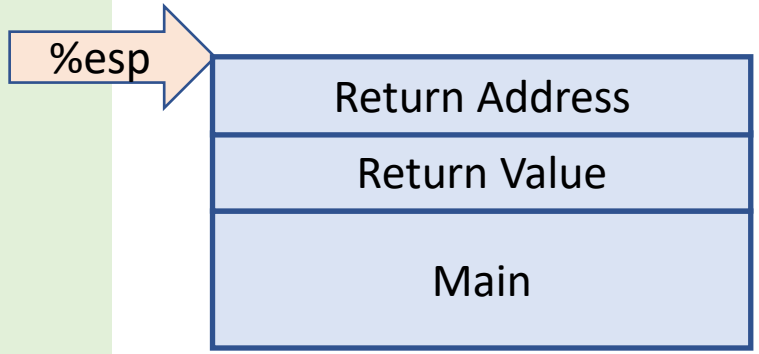
Return Values – Method 1: Stack

```c
int func() {
    return 42;
}

int main() {
    int answer = 1;
    answer = func();
    return 0;
}
```

```
func:
movl $42, 4(%esp)
   # prep return value
ret
   # pop return address

main:
movl $1, %eax
subl $4, %esp
   #reserve space
call func
   #pushes return addr
movl (%esp), %eax
```

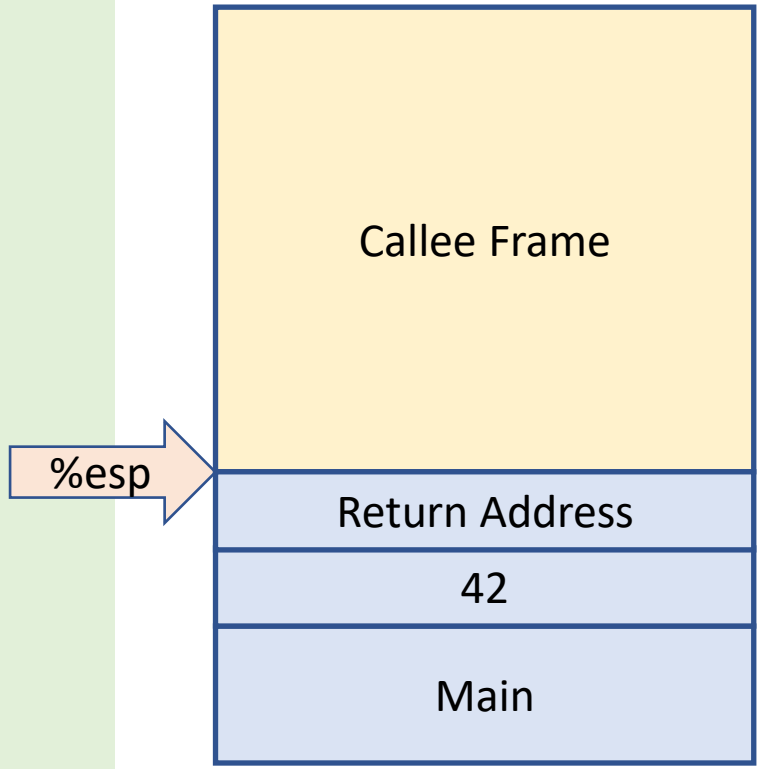


Return Values – Method 1: Stack

```
 1 int func() {
 2     return 42;
 3 }
 4
 5 int main() {
 6     int answer = 1;
 7     answer = func();
 8     return 0;
 9 }
10
```

```
func:
movl $42, 4(%esp)
   # prep return value
ret
   # pop return address

main:
movl $1, %eax
subl $4, %esp
   #reserve space
call func
   #pushes return addr
movl (%esp), %eax
```



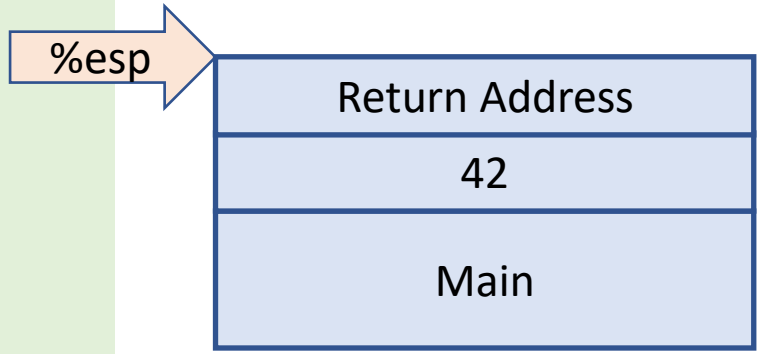
Return Values – Method 1: Stack

```c
int func() {
    return 42;
}

int main() {
    int answer = 1;
    answer = func();
    return 0;
}
```

```
func:
movl $42, 4(%esp)
   # prep return value
ret
   # pop return address

main:
movl $1, %eax
subl $4, %esp
   #reserve space
call func
   #pushes return addr
movl (%esp), %eax
```



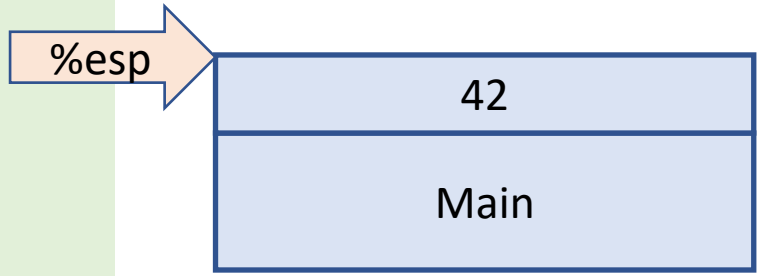
Return Values – Method 1: Stack

```c
int func() {
    return 42;
}

int main() {
    int answer = 1;
    answer = func();
    return 0;
}
```

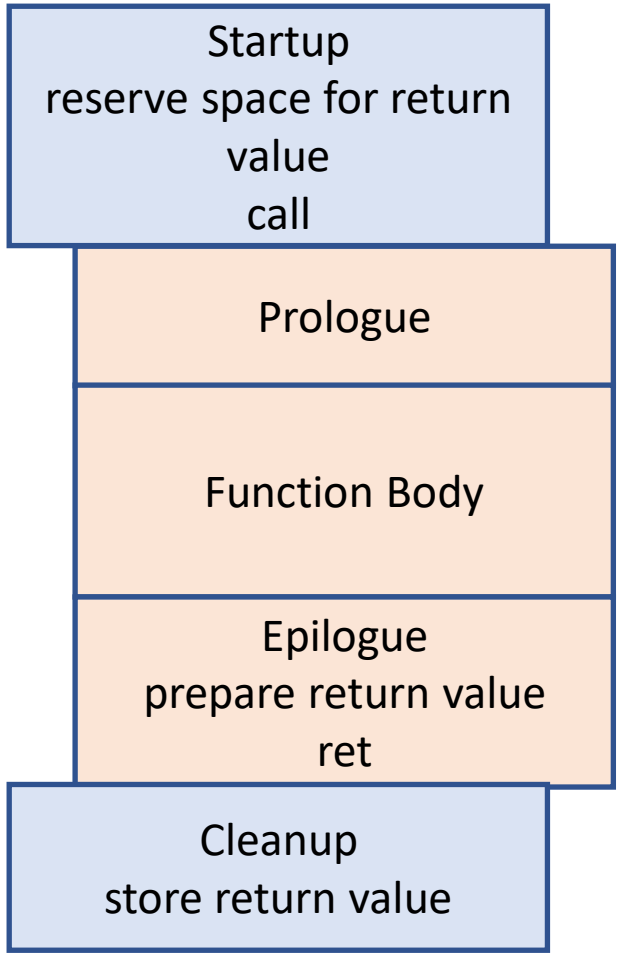
```
func:
movl $42, 4(%esp)
   # prep return value
ret
   # pop return address

main:
movl $1, %eax
subl $4, %esp
   #reserve space
call func
   #pushes return addr
movl (%esp), %eax
```



Calling Conventions

```
 1 int func() {
 2     return 42;
 3 }
 4
 5 int main() {
 6     int answer = 1;
 7     answer = func();
 8     return 0;
 9 }
10
```



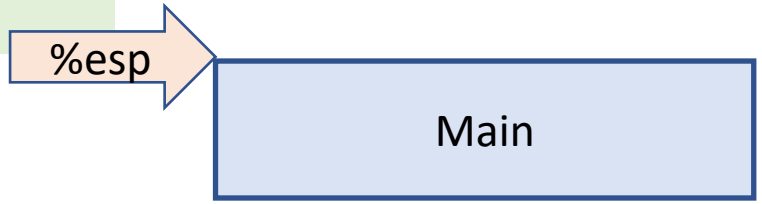
Return Values – Method 2: %eax

```c
int func() {
    return 42;
}

int main() {
    int answer = 1;
    answer = func();
    return 0;
}
```

```
func:
movl $42, %eax
ret
   # pop return address

main:
movl $1, %ebx
call func
movl %eax, %ebx
```

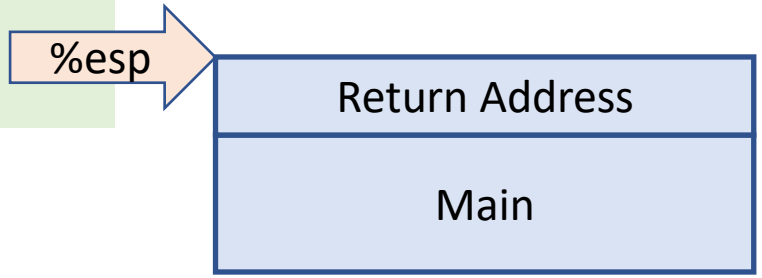


Return Values – Method 2: %eax

```
 1 int func() {
 2     return 42;
 3 }
 4
 5 int main() {
 6     int answer = 1;
 7     answer = func();
 8     return 0;
 9 }
10
```

```
func:
movl $42, %eax
ret
   # pop return address

main:
movl $1, %ebx
call func
movl %eax, %ebx
```



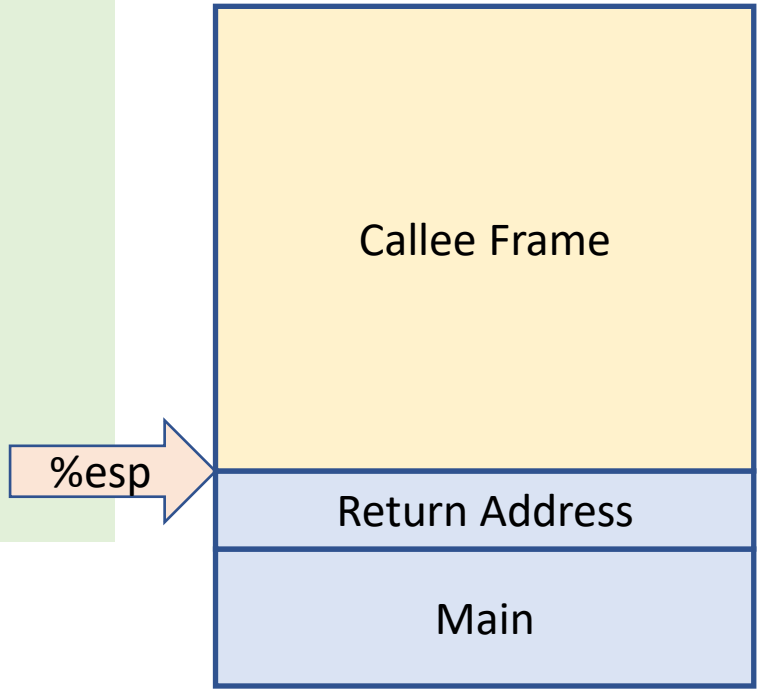
Return Values – Method 2: %eax

```c
int func() {
    return 42;
}

int main() {
    int answer = 1;
    answer = func();
    return 0;
}
```

```
func:
movl $42, %eax
ret
   # pop return address

main:
movl $1, %ebx
call func
movl %eax, %ebx
```

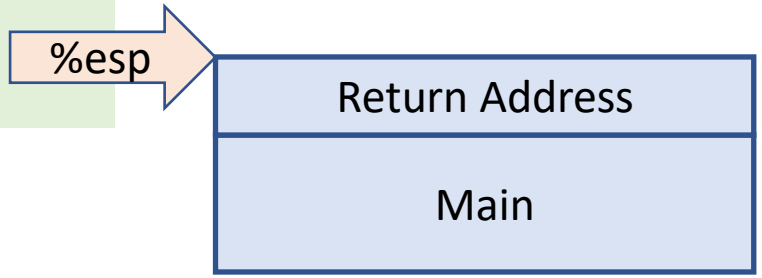


Return Values – Method 2: %eax

```
 1 int func() {
 2     return 42;
 3 }
 4
 5 int main() {
 6     int answer = 1;
 7     answer = func();
 8     return 0;
 9 }
10
```

```
func:
movl $42, %eax
ret
   # pop return address

main:
movl $1, %ebx
call func
movl %eax, %ebx
```



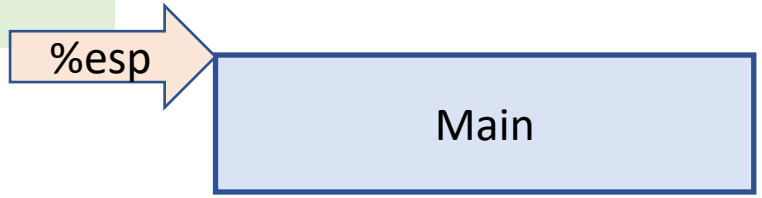
Return Values – Method 2: %eax

```c
int func() {
    return 42;
}

int main() {
    int answer = 1;
    answer = func();
    return 0;
}
```

```
func:
movl $42, %eax
ret
   # pop return address

main:
movl $1, %ebx
call func
movl %eax, %ebx
```



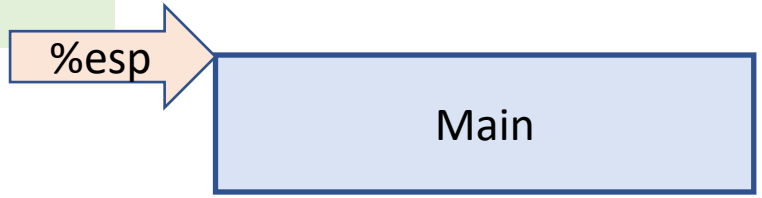
Return Values – Method 2: %eax

```c
int func() {
    return 42;
}

int main() {
    int answer = 1;
    answer = func();
    return 0;
}
```

```
func:
movl $42, %eax
ret
   # pop return address

main:
movl $1, %ebx
call func
movl %eax, %ebx
```



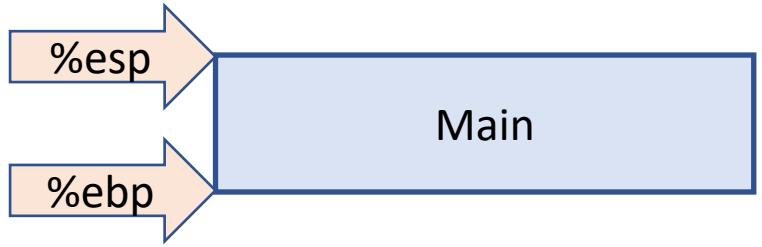
Base Pointer

```
1 int func(int x, int y) {
2     return x + y;
3 }
4
5 int main() {
6     int answer = 1;
7     answer = func(2,3);
8     return 0;
9 }
```

1. How to call a function
2. How to return control back
3. How to return a value
4. **How to pass parameters**
5. **How to allocate space for local variables**
6. **How to access parameters and local variables**
7. How to deal with registers

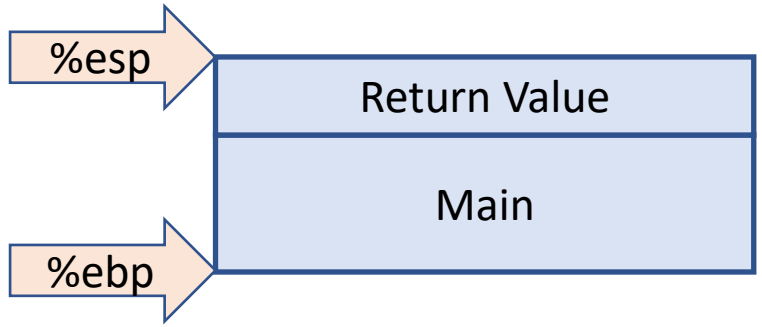
Base Pointer

```
1 int func(int x, int y) {
2     return x + y;
3 }
4
5 int main() {
6     int answer = 1;
7     answer = func(2,3);
8     return 0;
9 }
```



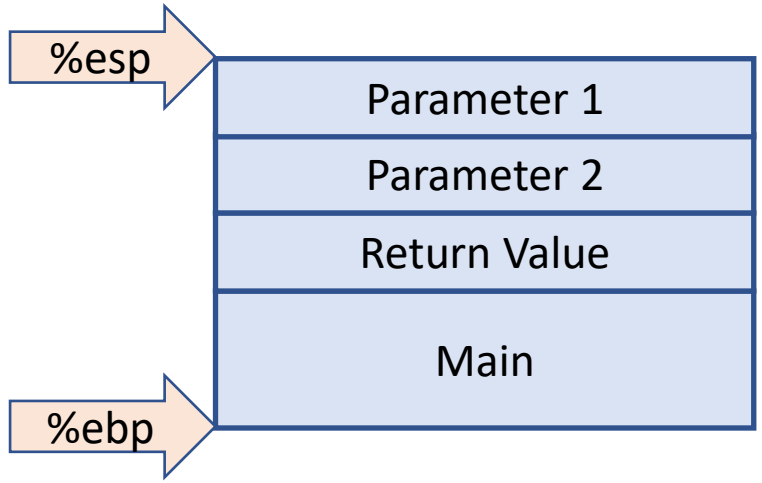
Base Pointer

```
1 int func(int x, int y) {
2     return x + y;
3 }
4
5 int main() {
6     int answer = 1;
7     answer = func(2,3);
8     return 0;
9 }
```



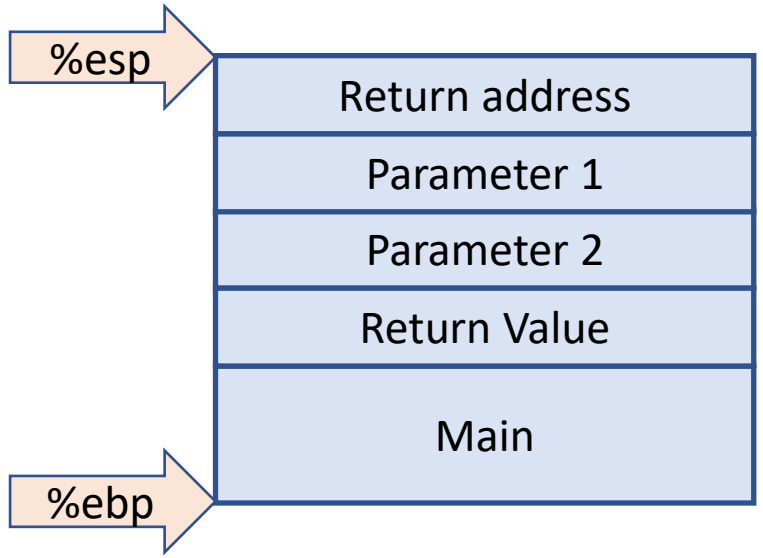
Base Pointer

```
1 int func(int x, int y) {
2     return x + y;
3 }
4
5 int main() {
6     int answer = 1;
7     answer = func(2,3);
8     return 0;
9 }
```



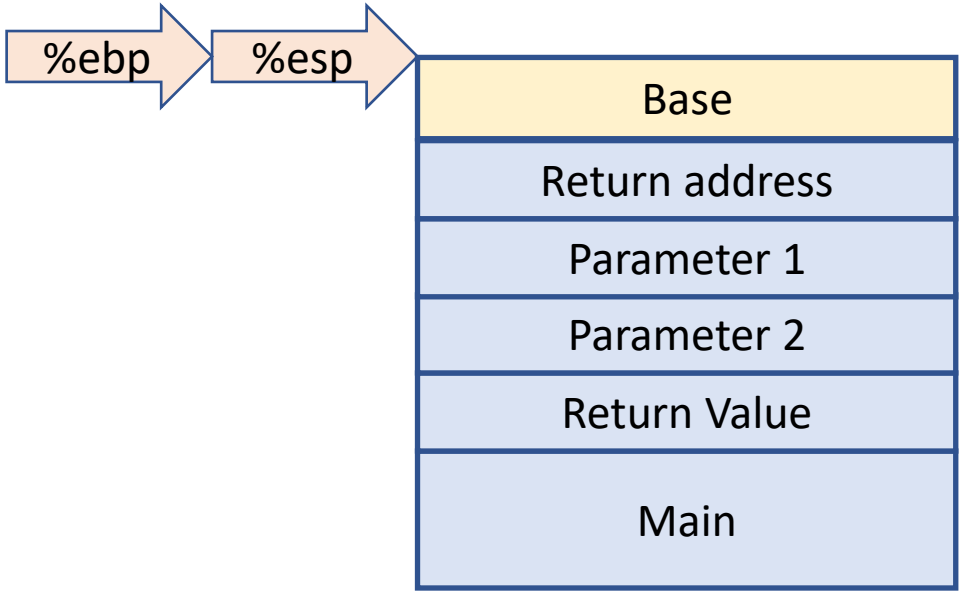
Base Pointer

```
1 int func(int x, int y) {
2     return x + y;
3 }
4
5 int main() {
6     int answer = 1;
7     answer = func(2,3);
8     return 0;
9 }
```



Base Pointer

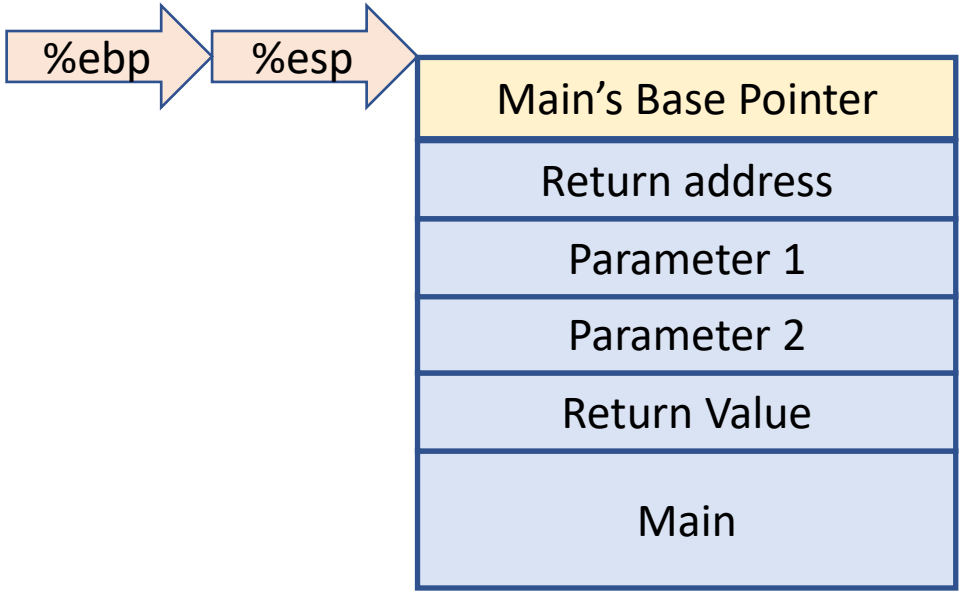
```
1 int func(int x, int y) {
2     return x + y;
3 }
4
5 int main() {
6     int answer = 1;
7     answer = func(2,3);
8     return 0;
9 }
```




```
push %ebp
movl %esp, ebp
```

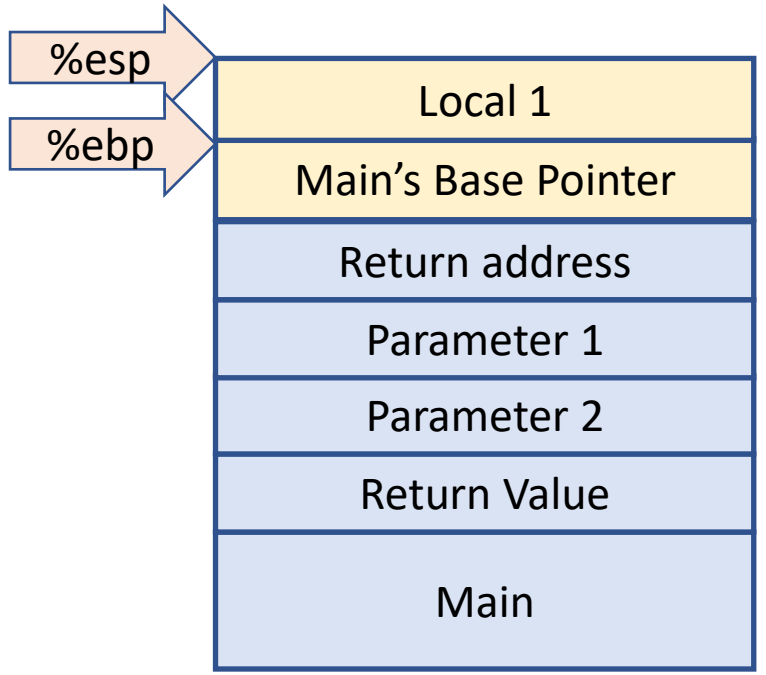
Base Pointer

```
1 int func(int x, int y) {
2     return x + y;
3 }
4
5 int main() {
6     int answer = 1;
7     answer = func(2,3);
8     return 0;
9 }
```



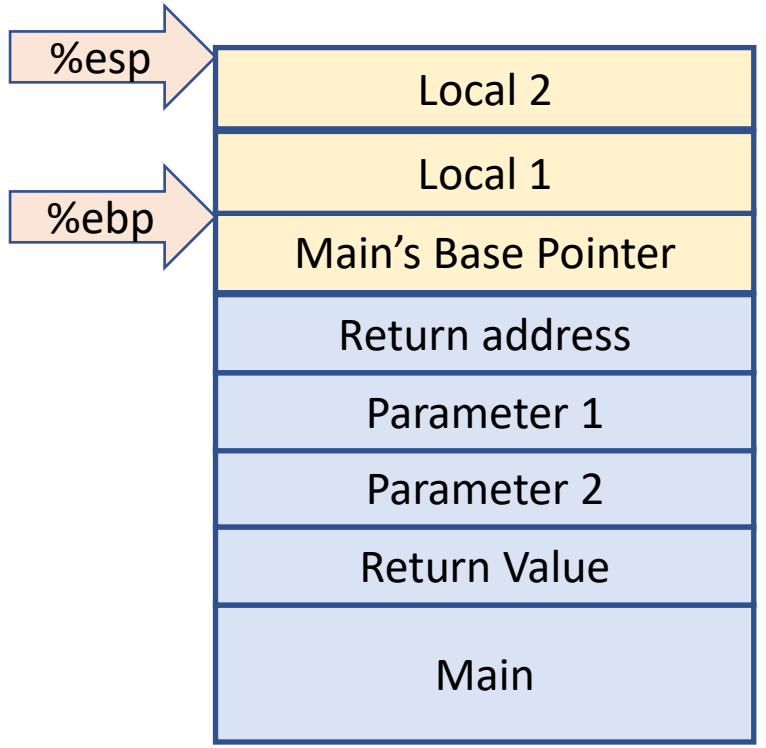
Base Pointer

```
1 int func(int x, int y) {
2     return x + y;
3 }
4
5 int main() {
6     int answer = 1;
7     answer = func(2,3);
8     return 0;
9 }
```



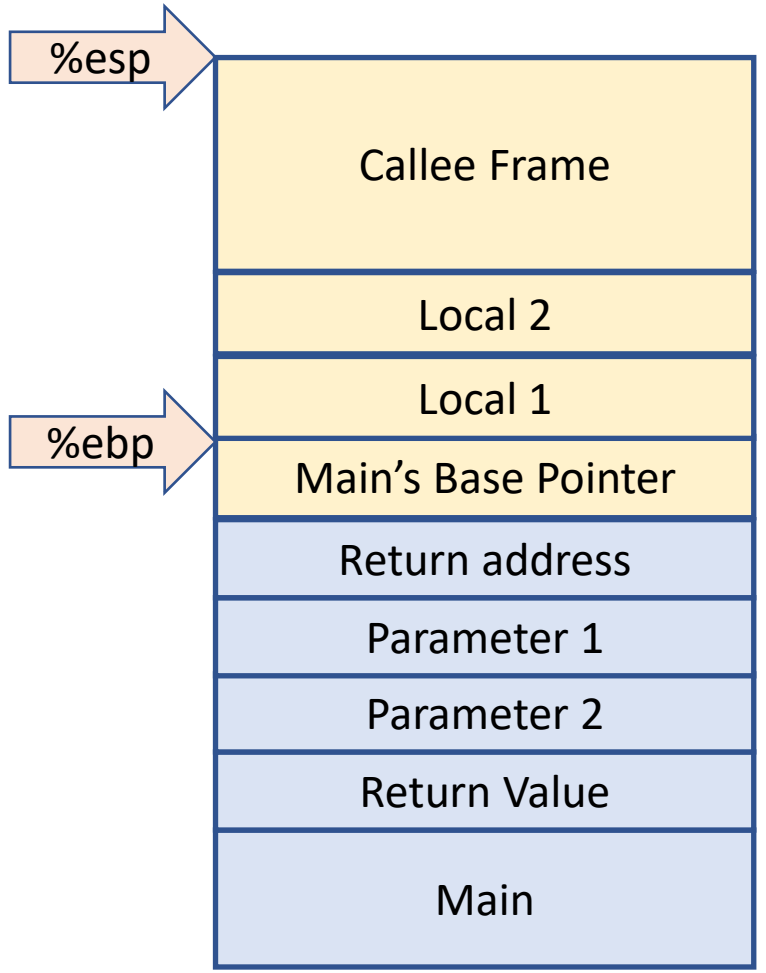
Base Pointer

```
1 int func(int x, int y) {
2     return x + y;
3 }
4
5 int main() {
6     int answer = 1;
7     answer = func(2,3);
8     return 0;
9 }
```



Base Pointer

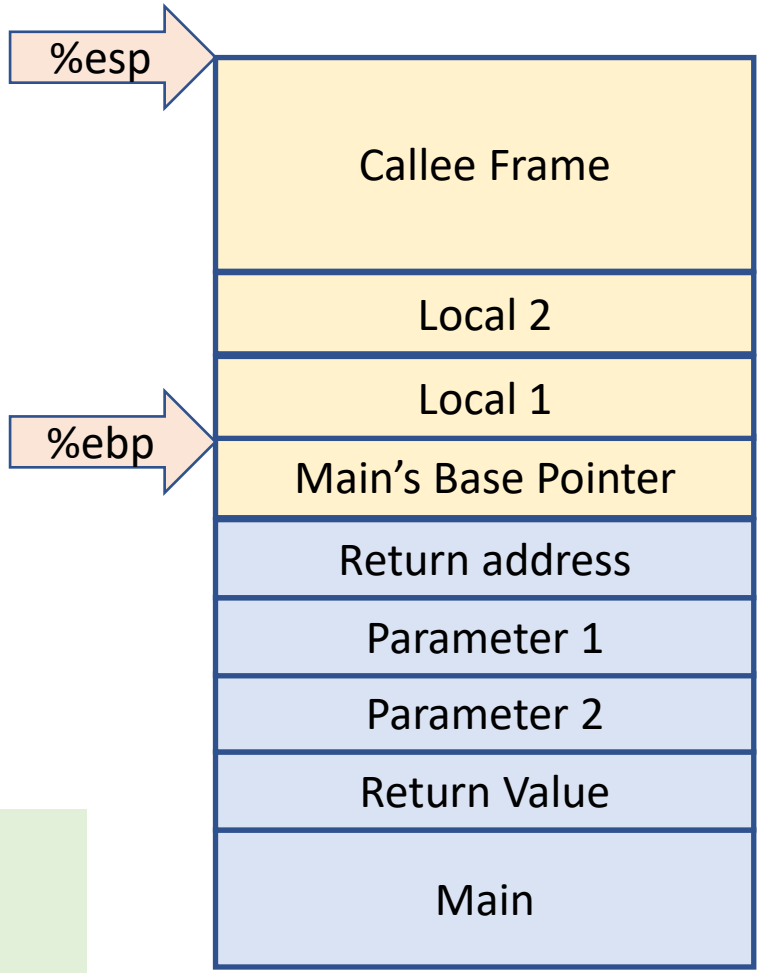
```
1 int func(int x, int y) {
2     return x + y;
3 }
4
5 int main() {
6     int answer = 1;
7     answer = func(2,3);
8     return 0;
9 }
```



Base Pointer

```
1 int func(int x, int y) {
2     return x + y;
3 }
4
5 int main() {
6     int answer = 1;
7     answer = func(2,3);
8     return 0;
9 }
```

```
leave
    movl %ebp, %esp
    popl %ebp
ret
```



Parameter Passing

```
1 int func(int x, int y) {
2     return x + y;
3 }
4
5 int main() {
6     int answer = 1;
7     answer = func(2,3);
8     return 0;
9 }
```

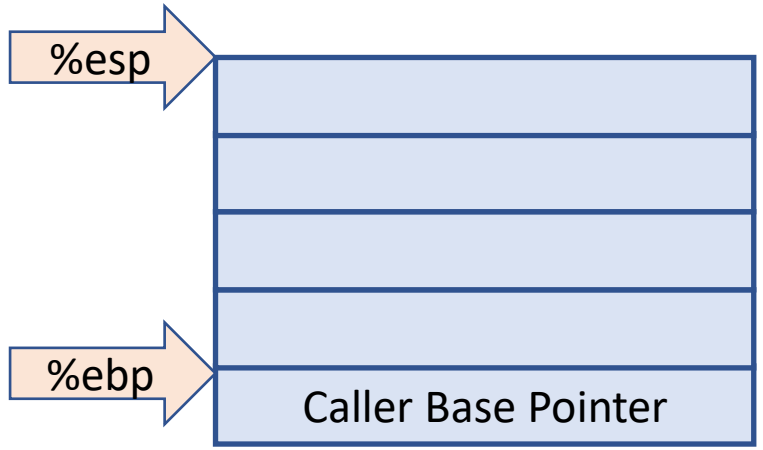
1. How to call a function
2. How to return control back
3. How to return a value
4. **How to pass parameters**
5. How to allocate space for local variables
6. How to access parameters and local variables
7. How to deal with registers

Parameter Passing

```
32      pushl   %ebp
33      .cfi_def_cfa_offset 8
34      .cfi_offset 5, -8
35      movl    %esp, %ebp
36      .cfi_def_cfa_register 5
37      subl    $16, %esp
38      call    __x86.get_pc_thunk.ax
39      addl    $_GLOBAL_OFFSET_TABLE_, %eax
40      movl    $1, -4(%ebp)
41      pushl   $3
42      pushl   $2
43      call    func
44      addl    $8, %esp
45      movl    %eax, -4(%ebp)
46      movl    $0, %eax
47      leave
```

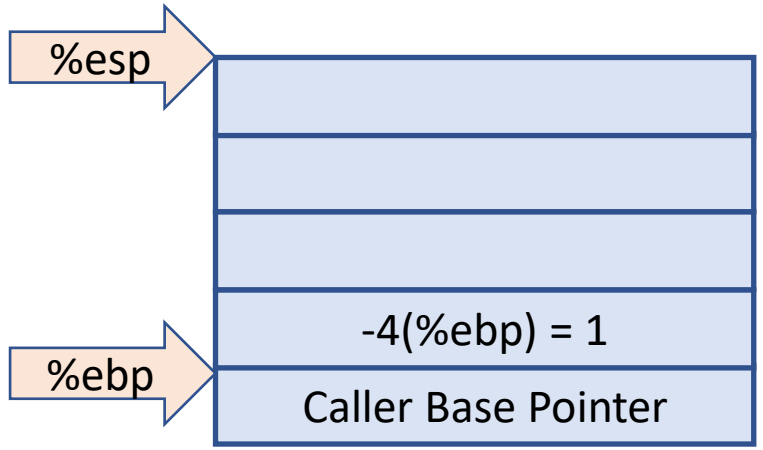
Parameter Passing

```
32      pushl   %ebp
33      .cfi_def_cfa_offset 8
34      .cfi_offset 5, -8
35      movl    %esp, %ebp
36      .cfi_def_cfa_register 5
37      subl    $16, %esp
38      call    __x86.get_pc_thunk.ax
39      addl    $_GLOBAL_OFFSET_TABLE_, %eax
40      movl    $1, -4(%ebp)
41      pushl   $3
42      pushl   $2
43      call    func
44      addl    $8, %esp
45      movl    %eax, -4(%ebp)
46      movl    $0, %eax
47      leave
```



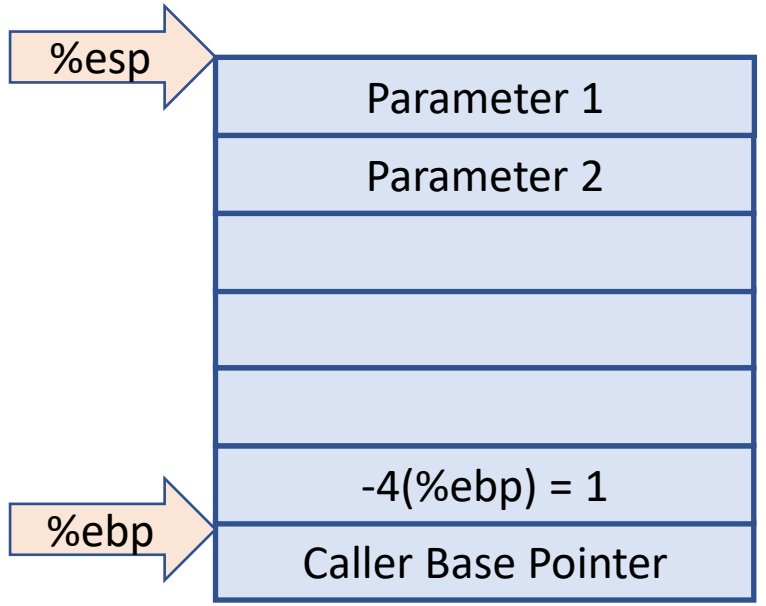
Parameter Passing

```
32      pushl   %ebp
33      .cfi_def_cfa_offset 8
34      .cfi_offset 5, -8
35      movl    %esp, %ebp
36      .cfi_def_cfa_register 5
37      subl    $16, %esp
38      call    __x86.get_pc_thunk.ax
39      addl    $_GLOBAL_OFFSET_TABLE_, %eax
40      movl    $1, -4(%ebp)
41      pushl   $3
42      pushl   $2
43      call    func
44      addl    $8, %esp
45      movl    %eax, -4(%ebp)
46      movl    $0, %eax
47      leave
```



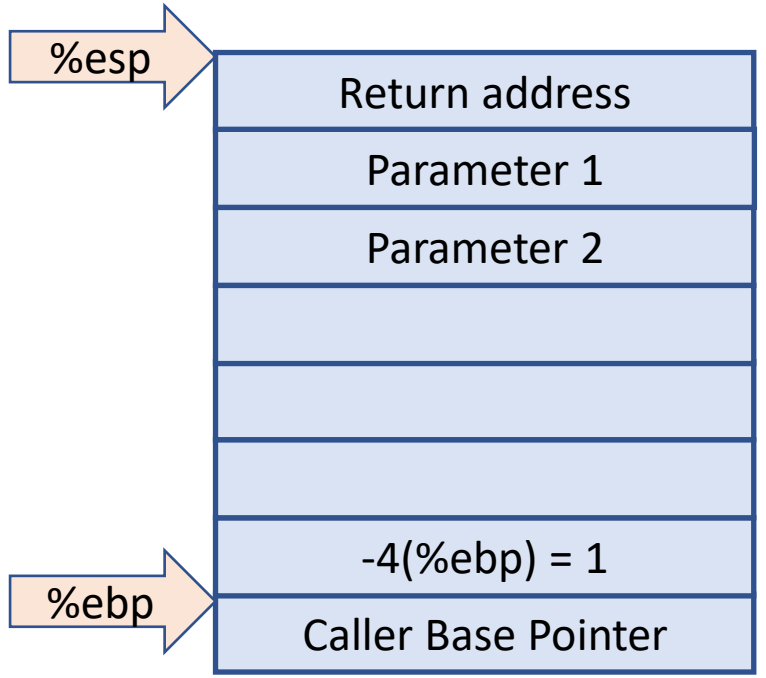
Parameter Passing

```
32      pushl   %ebp
33      .cfi_def_cfa_offset 8
34      .cfi_offset 5, -8
35      movl    %esp, %ebp
36      .cfi_def_cfa_register 5
37      subl    $16, %esp
38      call    __x86.get_pc_thunk.ax
39      addl    $_GLOBAL_OFFSET_TABLE_, %eax
40      movl    $1, -4(%ebp)
41      pushl   $3
42      pushl   $2
43      call    func
44      addl    $8, %esp
45      movl    %eax, -4(%ebp)
46      movl    $0, %eax
47      leave
```



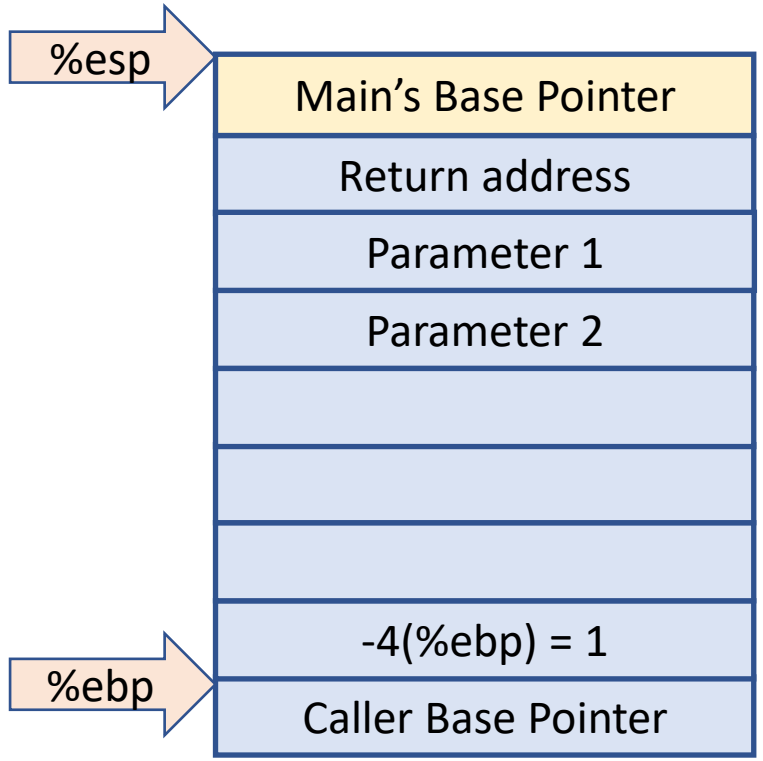
Parameter Passing

```
32      pushl   %ebp
33      .cfi_def_cfa_offset 8
34      .cfi_offset 5, -8
35      movl    %esp, %ebp
36      .cfi_def_cfa_register 5
37      subl    $16, %esp
38      call    __x86.get_pc_thunk.ax
39      addl    $_GLOBAL_OFFSET_TABLE_, %eax
40      movl    $1, -4(%ebp)
41      pushl   $3
42      pushl   $2
43      call    func
44      addl    $8, %esp
45      movl    %eax, -4(%ebp)
46      movl    $0, %eax
47      leave
```



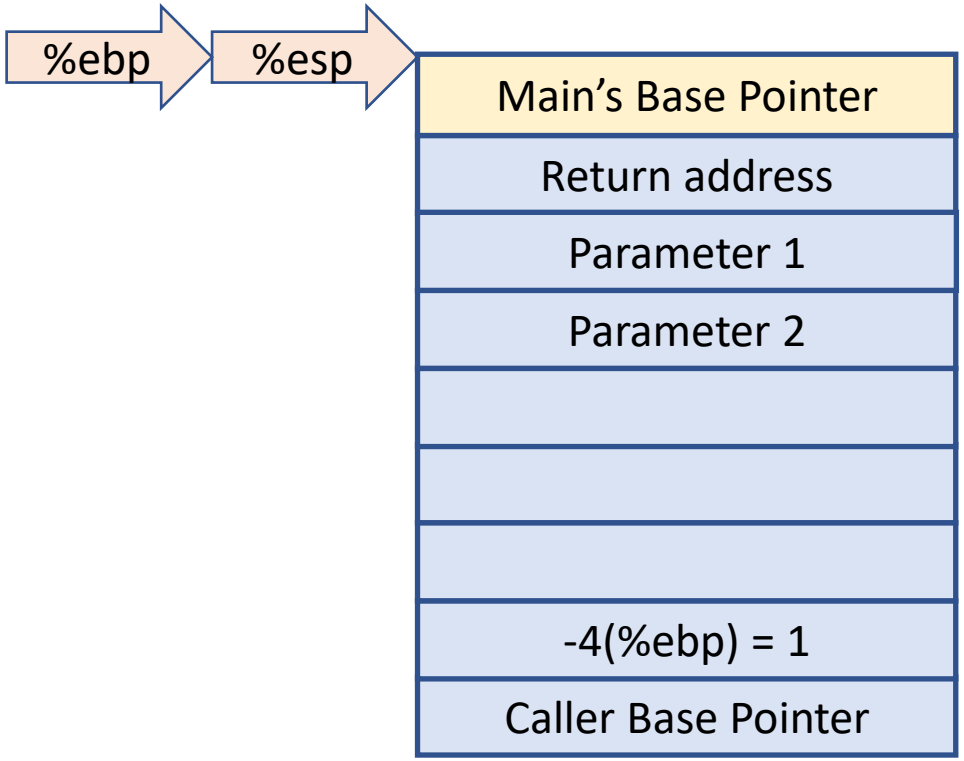
Parameter Passing

```
 5 func:
 6 .LFB0:
 7     .cfi_startproc
 8     endbr32
 9     pushl   %ebp
10     .cfi_def_cfa_offset 8
11     .cfi_offset 5, -8
12     movl    %esp, %ebp
13     .cfi_def_cfa_register 5
14     call    __x86.get_pc_thunk.ax
15     addl    $_GLOBAL_OFFSET_TABLE_, %eax
16     movl    8(%ebp), %edx
17     movl    12(%ebp), %eax
18     addl    %edx, %eax
19     popl    %ebp
20     .cfi_restore 5
21     .cfi_def_cfa 4, 4
22     ret
```



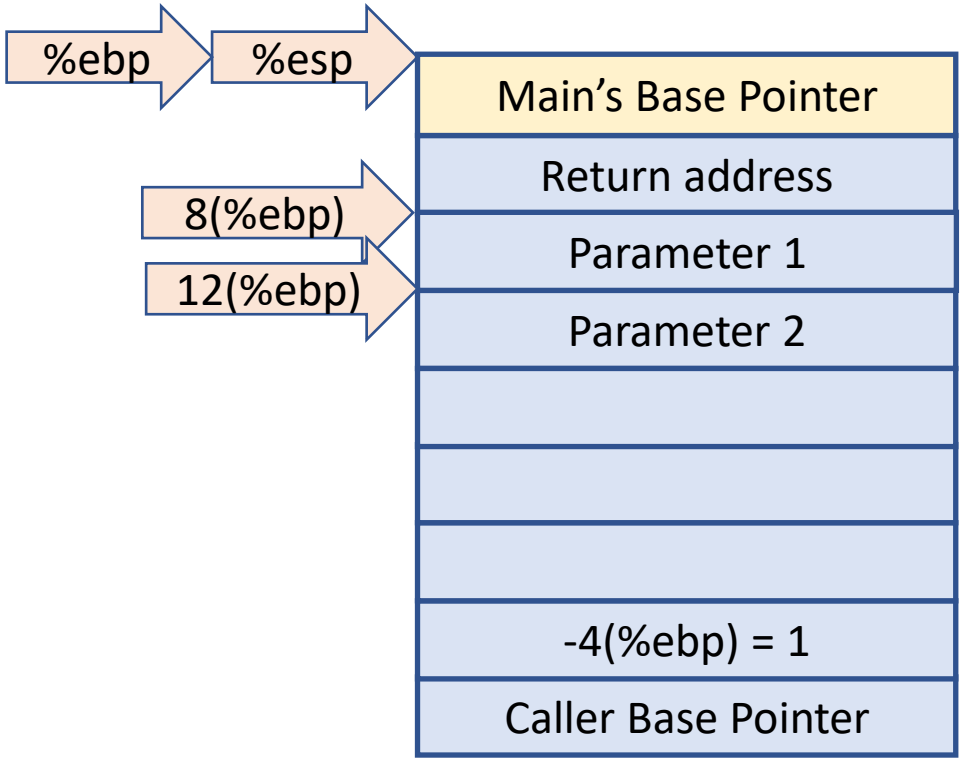
Parameter Passing

```
 5 func:
 6 .LFB0:
 7     .cfi_startproc
 8     endbr32
 9     pushl   %ebp
10     .cfi_def_cfa_offset 8
11     .cfi_offset 5, -8
12     movl    %esp, %ebp
13     .cfi_def_cfa_register 5
14     call    __x86.get_pc_thunk.ax
15     addl    $_GLOBAL_OFFSET_TABLE_, %eax
16     movl    8(%ebp), %edx
17     movl    12(%ebp), %eax
18     addl    %edx, %eax
19     popl    %ebp
20     .cfi_restore 5
21     .cfi_def_cfa 4, 4
22     ret
```



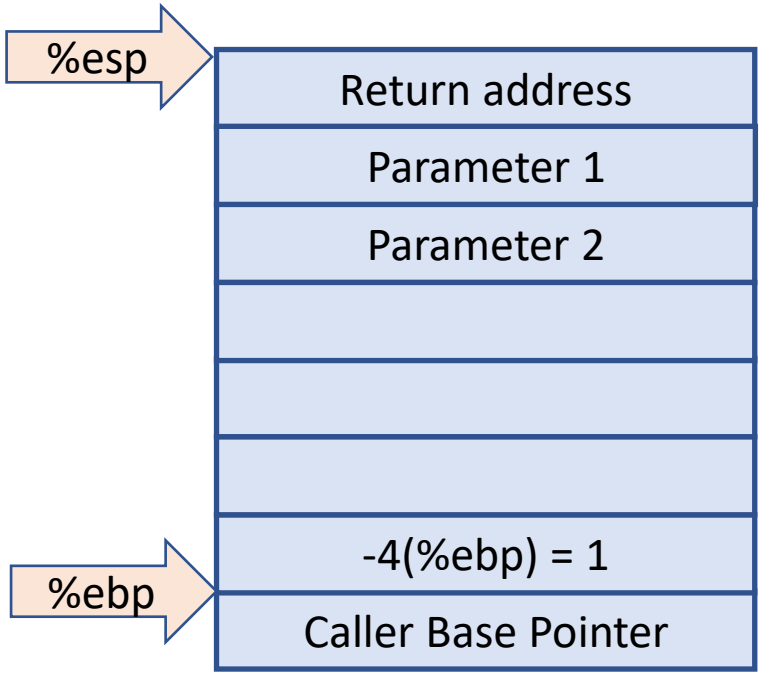
Parameter Passing

```
 5 func:
 6 .LFB0:
 7     .cfi_startproc
 8     endbr32
 9     pushl   %ebp
10     .cfi_def_cfa_offset 8
11     .cfi_offset 5, -8
12     movl    %esp, %ebp
13     .cfi_def_cfa_register 5
14     call    __x86.get_pc_thunk.ax
15     addl    $_GLOBAL_OFFSET_TABLE_, %eax
16     movl    8(%ebp), %edx
17     movl    12(%ebp), %eax
18     addl    %edx, %eax
19     popl    %ebp
20     .cfi_restore 5
21     .cfi_def_cfa 4, 4
22     ret
```



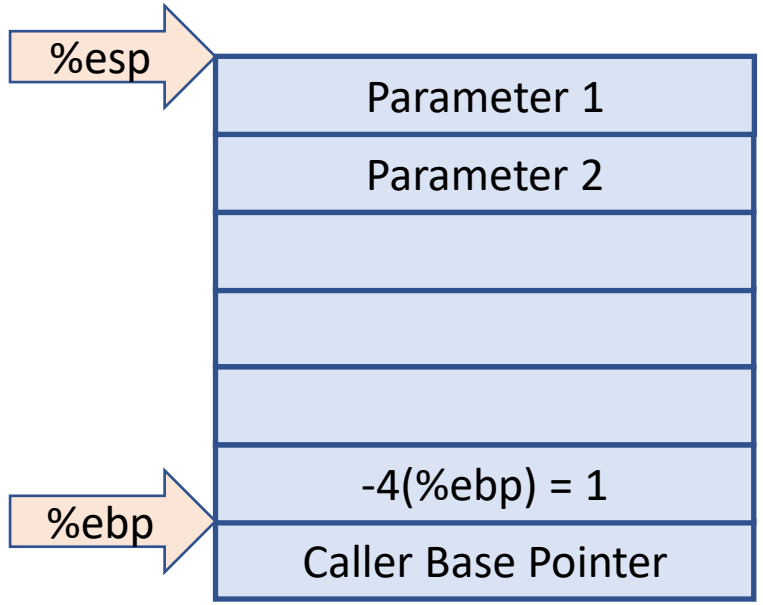
Parameter Passing

```
 5 func:
 6 .LFB0:
 7     .cfi_startproc
 8     endbr32
 9     pushl   %ebp
10     .cfi_def_cfa_offset 8
11     .cfi_offset 5, -8
12     movl    %esp, %ebp
13     .cfi_def_cfa_register 5
14     call    __x86.get_pc_thunk.ax
15     addl    $_GLOBAL_OFFSET_TABLE_, %eax
16     movl    8(%ebp), %edx
17     movl    12(%ebp), %eax
18     addl    %edx, %eax
19     popl    %ebp
20     .cfi_restore 5
21     .cfi_def_cfa 4, 4
22     ret
```



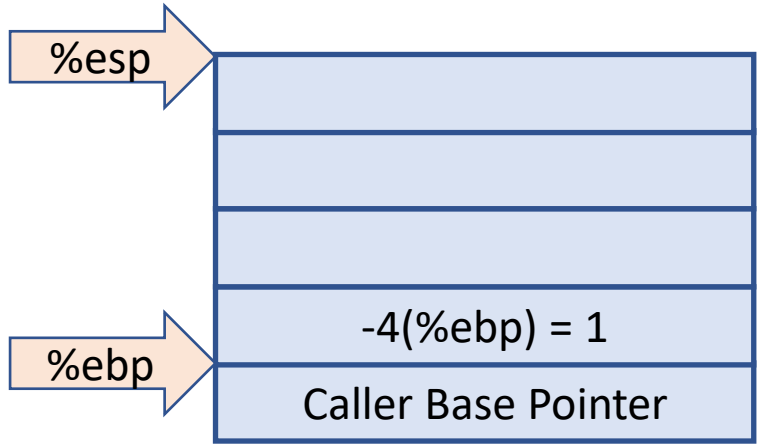
Parameter Passing

```
32      pushl   %ebp
33      .cfi_def_cfa_offset 8
34      .cfi_offset 5, -8
35      movl    %esp, %ebp
36      .cfi_def_cfa_register 5
37      subl    $16, %esp
38      call    __x86.get_pc_thunk.ax
39      addl    $_GLOBAL_OFFSET_TABLE_, %eax
40      movl    $1, -4(%ebp)
41      pushl   $3
42      pushl   $2
43      call    func
44      addl    $8, %esp
45      movl    %eax, -4(%ebp)
46      movl    $0, %eax
47      leave
```



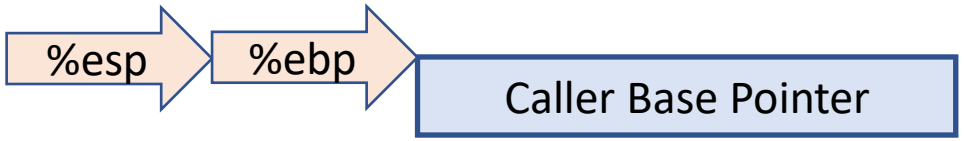
Parameter Passing

```
32      pushl   %ebp
33      .cfi_def_cfa_offset 8
34      .cfi_offset 5, -8
35      movl    %esp, %ebp
36      .cfi_def_cfa_register 5
37      subl    $16, %esp
38      call    __x86.get_pc_thunk.ax
39      addl    $_GLOBAL_OFFSET_TABLE_, %eax
40      movl    $1, -4(%ebp)
41      pushl   $3
42      pushl   $2
43      call    func
44      addl    $8, %esp
45      movl    %eax, -4(%ebp)
46      movl    $0, %eax
47      leave
```



Parameter Passing

```
32      pushl   %ebp
33      .cfi_def_cfa_offset 8
34      .cfi_offset 5, -8
35      movl    %esp, %ebp
36      .cfi_def_cfa_register 5
37      subl    $16, %esp
38      call    __x86.get_pc_thunk.ax
39      addl    $_GLOBAL_OFFSET_TABLE_, %eax
40      movl    $1, -4(%ebp)
41      pushl   $3
42      pushl   $2
43      call    func
44      addl    $8, %esp
45      movl    %eax, -4(%ebp)
46      movl    $0, %eax
47      leave
```



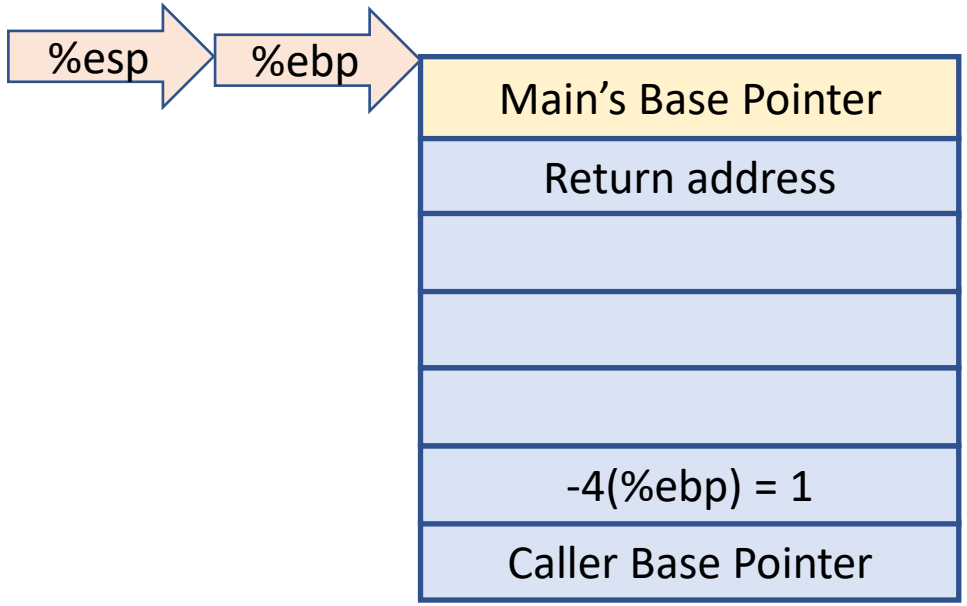
Local Variables

```
 1 int func() {
 2     int a = 1;
 3     int b = 2;
 4     int c = 3;
 5     int d = 4;
 6     return a + b + c + d;
 7 }
 8
 9 int main() {
10     int answer = 1;
11     answer = func();
12     return 0;
13 }
```

1. How to call a function
2. How to return control back
3. How to return a value
4. How to pass parameters
5. **How to allocate space for local variables**
6. How to access parameters and local variables
7. How to deal with registers

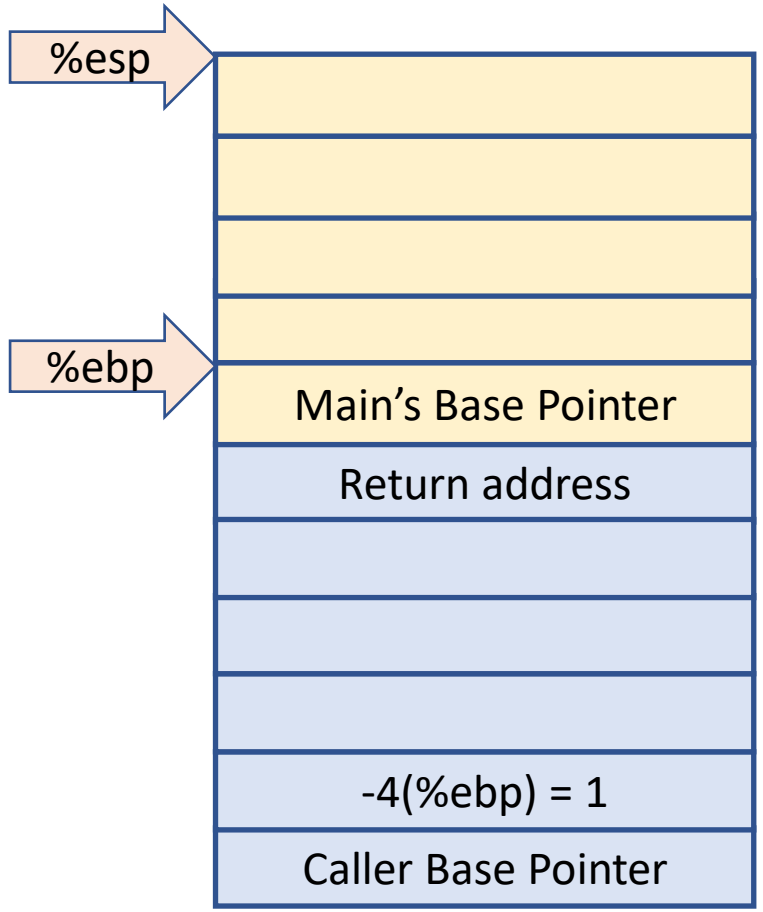
Local Variables

```
 5 func:
 6     pushl   %ebp
 7     movl    %esp, %ebp
 8     subl    $16, %esp
 9     movl    $1, -16(%ebp)
10     movl    $2, -12(%ebp)
11     movl    $3, -8(%ebp)
12     movl    $4, -4(%ebp)
13     movl    -16(%ebp), %edx
14     movl    -12(%ebp), %eax
15     addl    %eax, %edx
16     movl    -8(%ebp), %eax
17     addl    %eax, %edx
18     movl    -4(%ebp), %eax
19     addl    %edx, %eax
20     leave
21     ret
```



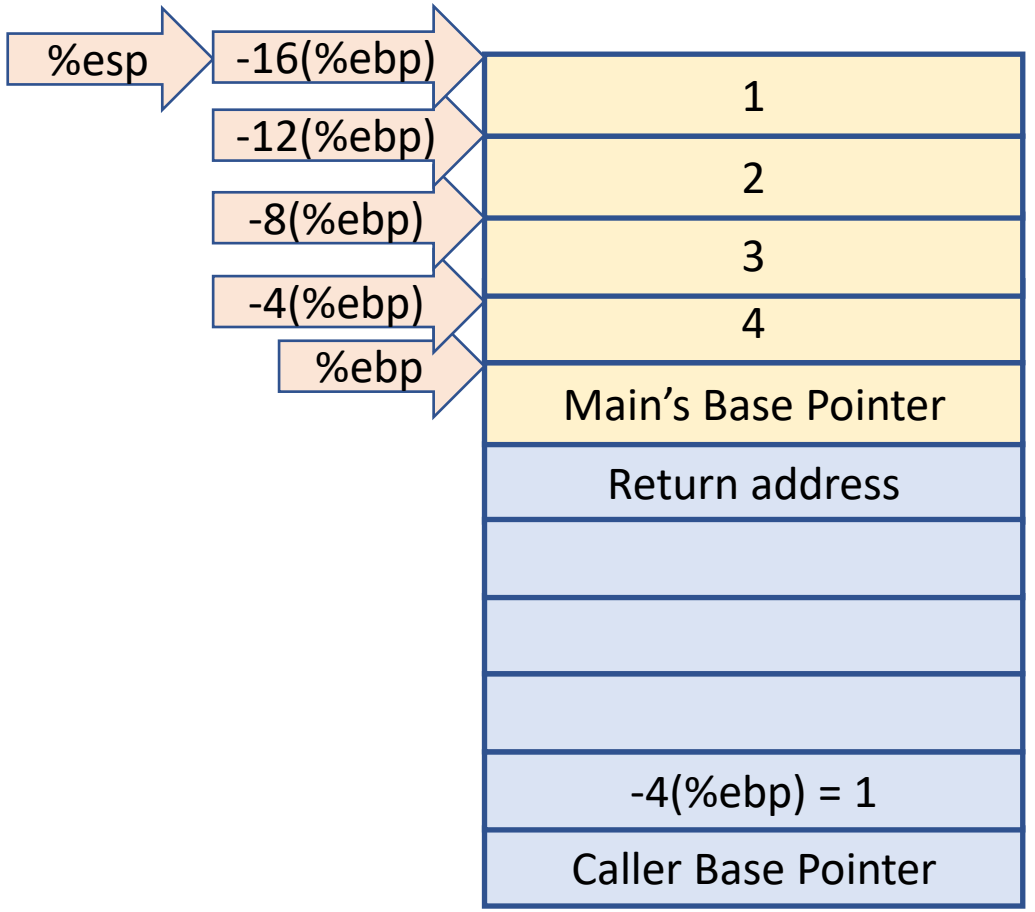
Local Variables

```
 5 func:
 6     pushl   %ebp
 7     movl    %esp, %ebp
 8     subl    $16, %esp
 9     movl    $1, -16(%ebp)
10     movl    $2, -12(%ebp)
11     movl    $3, -8(%ebp)
12     movl    $4, -4(%ebp)
13     movl    -16(%ebp), %edx
14     movl    -12(%ebp), %eax
15     addl    %eax, %edx
16     movl    -8(%ebp), %eax
17     addl    %eax, %edx
18     movl    -4(%ebp), %eax
19     addl    %edx, %eax
20     leave
21     ret
```



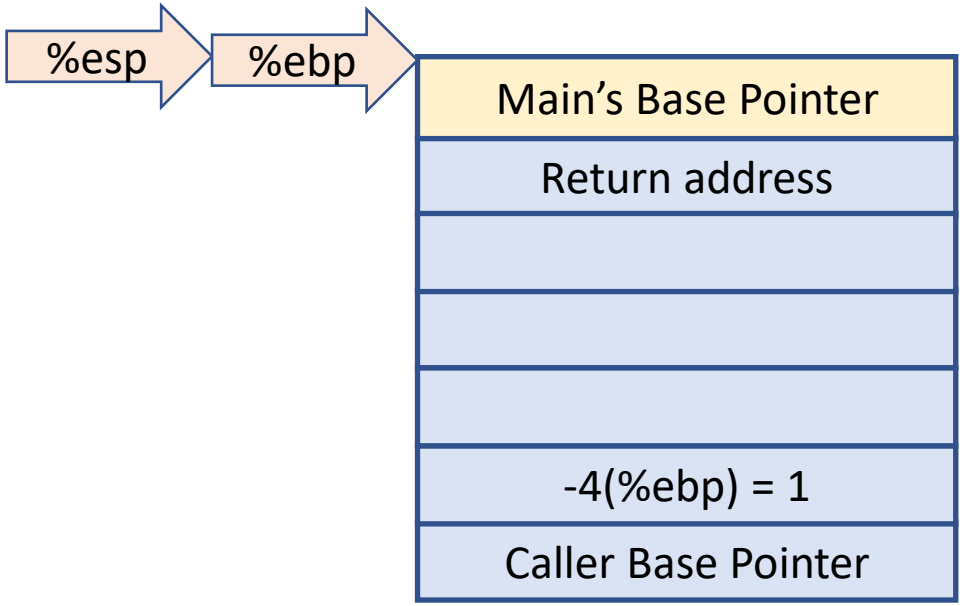
Local Variables

```
 5 func:
 6     pushl   %ebp
 7     movl    %esp, %ebp
 8     subl    $16, %esp
 9     movl    $1, -16(%ebp)
10     movl    $2, -12(%ebp)
11     movl    $3, -8(%ebp)
12     movl    $4, -4(%ebp)
13     movl    -16(%ebp), %edx
14     movl    -12(%ebp), %eax
15     addl    %eax, %edx
16     movl    -8(%ebp), %eax
17     addl    %eax, %edx
18     movl    -4(%ebp), %eax
19     addl    %edx, %eax
20     leave
21     ret
```



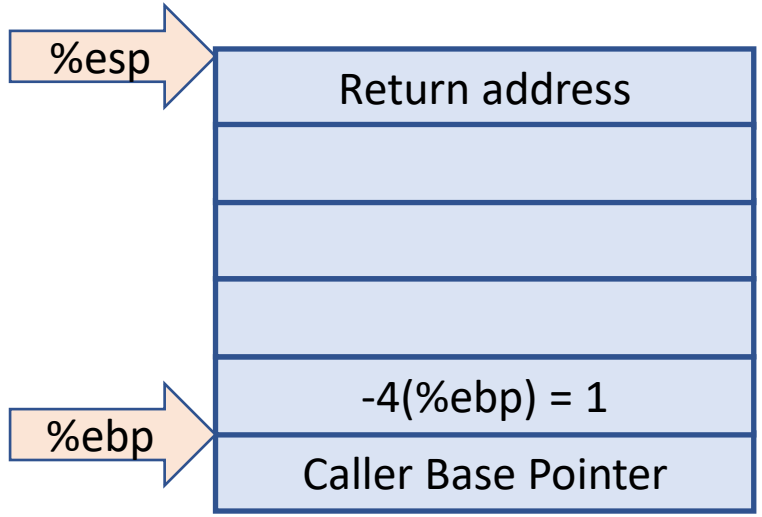
Local Variables

```
 5 func:
 6     pushl   %ebp
 7     movl    %esp, %ebp
 8     subl    $16, %esp
 9     movl    $1, -16(%ebp)
10     movl    $2, -12(%ebp)
11     movl    $3, -8(%ebp)
12     movl    $4, -4(%ebp)
13     movl    -16(%ebp), %edx
14     movl    -12(%ebp), %eax
15     addl    %eax, %edx
16     movl    -8(%ebp), %eax
17     addl    %eax, %edx
18     movl    -4(%ebp), %eax
19     addl    %edx, %eax
20     leave
21     ret
```



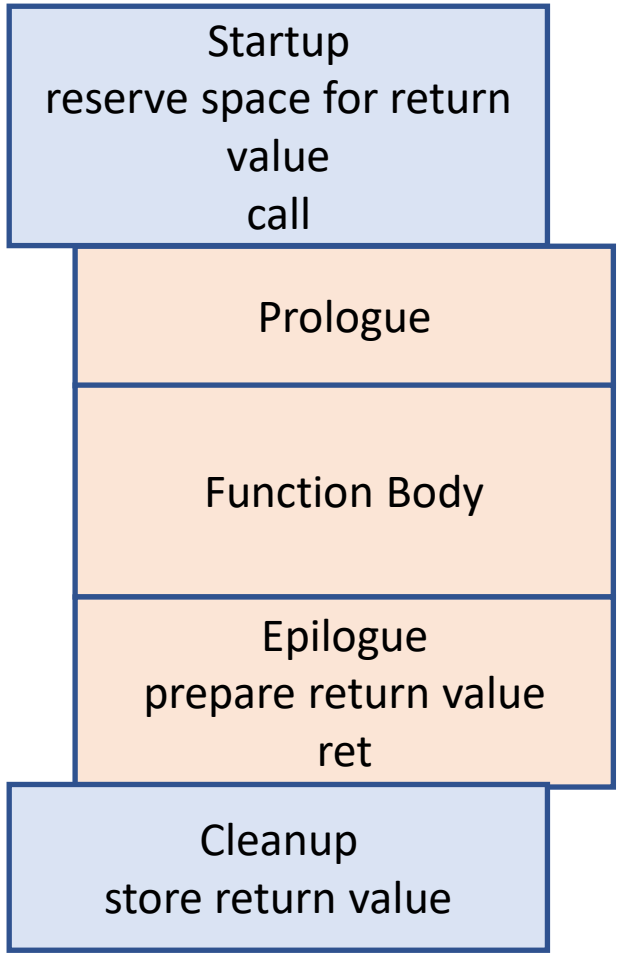
Local Variables

```
 5 func:
 6     pushl   %ebp
 7     movl    %esp, %ebp
 8     subl    $16, %esp
 9     movl    $1, -16(%ebp)
10     movl    $2, -12(%ebp)
11     movl    $3, -8(%ebp)
12     movl    $4, -4(%ebp)
13     movl    -16(%ebp), %edx
14     movl    -12(%ebp), %eax
15     addl    %eax, %edx
16     movl    -8(%ebp), %eax
17     addl    %eax, %edx
18     movl    -4(%ebp), %eax
19     addl    %edx, %eax
20     leave
21     ret
```



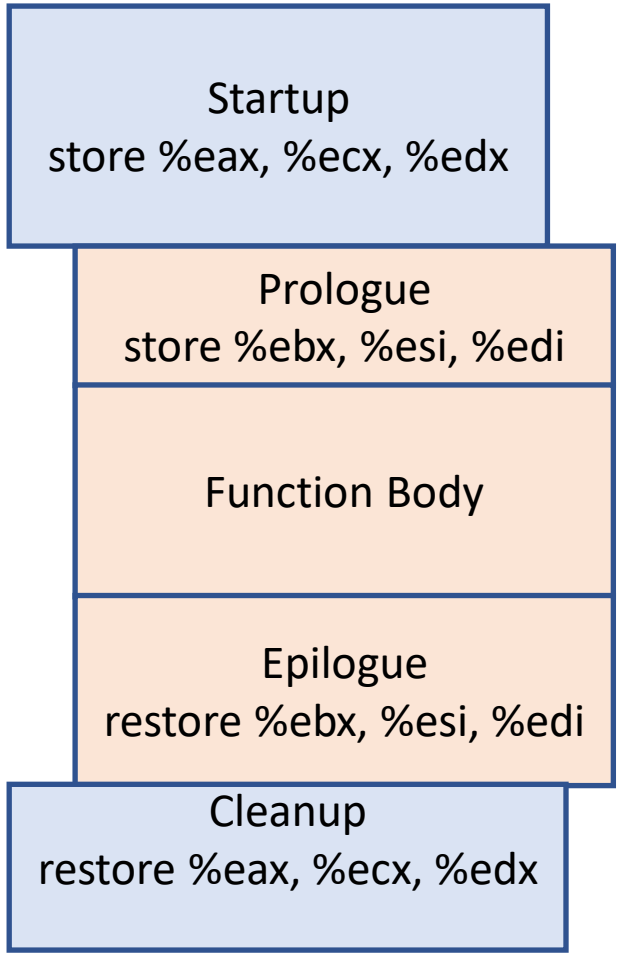
Calling Conventions

```
 5 func:
 6     pushl   %ebp
 7     movl    %esp, %ebp
 8     subl    $16, %esp
 9     movl    $1, -16(%ebp)
10     movl    $2, -12(%ebp)
11     movl    $3, -8(%ebp)
12     movl    $4, -4(%ebp)
13     movl    -16(%ebp), %edx
14     movl    -12(%ebp), %eax
15     addl    %eax, %edx
16     movl    -8(%ebp), %eax
17     addl    %eax, %edx
18     movl    -4(%ebp), %eax
19     addl    %edx, %eax
20     leave
21     ret
```



Registers are Shared

1. How to call a function
2. How to return control back
3. How to return a value
4. How to pass parameters
5. How to allocate space for local variables
6. How to access parameters and local variables
7. **How to deal with registers**



Calling Convention

Startup

- push %eax, %ecx, %edx
- push parameters
- reserve space for return value
  - subl $8, %esp
- push return address
- call

Cleanup

- deal with return value
- deallocate space for parameters
  - addl ($8, %esp)
- pop %eax, %ecx, %edx

Prologue

- push base pointer
- movl %esp, %ebp
- push %ebx, %esi, %edi
- Reserve space for locals
  - subl $16, %esp

Function Body

Epilogue

- prepare return value
- pop %ebx, %esi, %edi
- leave (movl %ebp, %esp; popl %ebp)
- ret