CS 354 Machine Organization and Programming

Lecture 19

Michael Doescher Summer 2020 **Function Calls**

Registers

%eax: 0x100

%ecx:

Assembly movl %eax, %ecx

```
Memory

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

Registers

%eax: 0x100

%ecx: 0x100

Assembly movl %eax, %ecx

```
Memory
```

```
0x100 : 42 : 0x104 : 35 :
```

0x108 : 82 :

Registers

%eax: 0x100

%ecx: 0x100

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

```
Memory
```

0x100 : 42 : 0x104 : 35 :

0x108 : 82 :

Registers

%eax: 0x100

%ecx: 42

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

```
Memory
```

```
0x100 : 42 :
```

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 :
```

Registers

%eax: 0x100

%ecx: 35

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

```
Memory

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

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

```
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 int sum(int x, int y) {
2 int total;
3 total = x + y;
 return total;
6
7 int main() {
 int s = sum(1,2);
  return 0;
```

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

```
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

```
int sum(int x, int y) {
     int total;
 total = x + y;
  return total;
6
 int main()
     int s = sum(1,2);
    return 0;
```

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

```
int sum(int x, int y) {
     int total;
 total = x + y;
  return total;
6
 int main()
     int s = sum(1,2);
    return 0;
```

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

```
int sum(int x, int y) {
     int total;
     total = x + y;
     return total;
6
 int main()
     int s = sum(1,2);
     return 0;
```

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

```
int sum(int x, int y) {
     int total;
     total = x + y;
     return total;
6
 int main()
     int s = sum(1,2);
     return 0;
```

- 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

```
int sum(int x, int y) {
     int total;
      total = x + y;
      return total;
6
 int main()
     int s = sum(1,2);
      return 0;
```

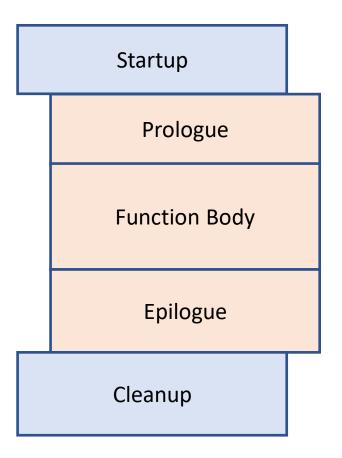
- 1. How to call a function
- 2. How to return control back
- 3. How to return a value
- 4. How to pass parameters
- 5. How to allocate space for local variables
- 6. How to access parameters and local variables

```
int sum(int x, int y)
      int total;
      total = x + y;
      return total;
6
 int main()
     int s = sum(1,2);
      return 0;
```

- 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

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



```
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</pre>
```

```
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</pre>
```

```
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</pre>
```

```
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</pre>
```

```
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</pre>
```

```
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

```
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
```

```
Registers
```

%eax:

%eip: 0x10C

%esp: 0xFA4

%ebp:

```
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

```
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
```

```
Registers
```

%eax:

%eip: 0x10C

%esp: 0xFA4

%ebp:

```
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

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
```

```
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

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
```

```
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

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
```

```
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

Transfer control jmp <target>

Ret

```
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
0 \times FA0
0xFA4 | 0x10C
0xFA8 | stack
                  <-%esp
```

```
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

Transfer control jmp <target>

Ret

```
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
```

Call

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

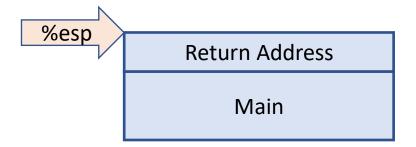
Ret



Call

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

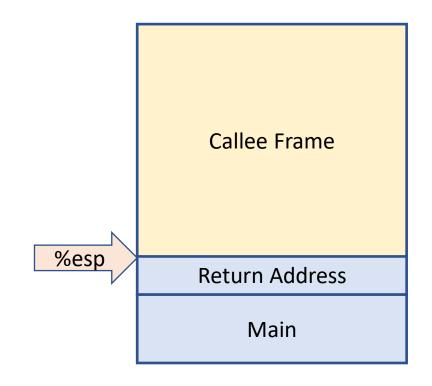
Ret



Call

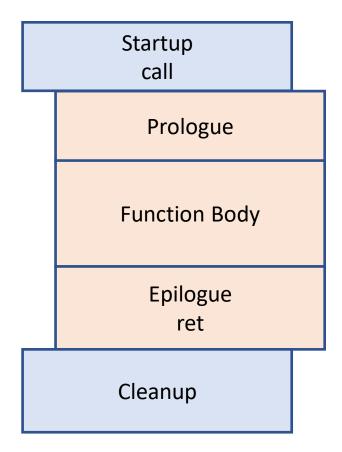
- 1. Push return address pushl %eip
- 2. Transfer control jmp <target>

Ret



Calling Conventions

```
1 int sum(int x, int y) {
2 int total;
     total = x = y;
  return total;
6
7 int main() {
 int s = sum(1,2);
  return 0;
```



Return Values

```
1 int func()
      return 42;
 int main()
      int answer = 1;
      answer = func();
      return 0;
```

- 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

```
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
                             %esp
  #reserve space
                                            Main
call func
  #pushes return addr
movl (%esp), %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, 4(%esp)
  # prep return value
ret
  # pop return address
main:
movl $1, %eax
subl $4, %esp
                              %esp
                                          Return Value
  #reserve space
                                             Main
call func
  #pushes return addr
movl (%esp), %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, 4(%esp)
  # prep return value
ret
  # pop return address
main:
                              %esp
movl $1, %eax
                                          Return Address
subl $4, %esp
                                          Return Value
  #reserve space
                                             Main
call func
  #pushes return addr
movl (%esp), %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, 4(%esp)
  # prep return value
ret
                                          Callee Frame
  # pop return address
main:
                              %esp
movl $1, %eax
                                          Return Address
subl $4, %esp
                                              42
  #reserve space
                                             Main
call func
  #pushes return addr
movl (%esp), %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, 4(%esp)
  # prep return value
ret
  # pop return address
main:
                              %esp
movl $1, %eax
                                         Return Address
subl $4, %esp
                                              42
  #reserve space
                                             Main
call func
  #pushes return addr
movl (%esp), %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, 4(%esp)
  # prep return value
ret
  # pop return address
main:
movl $1, %eax
                             %esp
subl $4, %esp
                                             42
  #reserve space
                                            Main
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
```

Startup reserve space for return value call Prologue **Function Body** Epilogue prepare return value ret Cleanup store return value

```
func:
                            movl $42, %eax
1 int func()
      return 42;
                            ret
                              # pop return address
 int main()
                            main:
      int answer = 1;
                            movl $1, %ebx
      answer = func();
      return 0;
                            call func
                            movl %eax, %ebx
                                                       %esp
```

sp

Return Address

```
func:
                              movl $42, %eax
1 int func()
      return 42;
                             ret
                                # pop return address
5 int main()
                                                                     Callee Frame
                              main:
      int answer = 1;
                              movl $1, %ebx
       answer = func();
      return 0;
                              call func
                                                          %esp
                              movl %eax, %ebx
                                                                    Return Address
                                                                        Main
```

Return Address

```
func:
                            movl $42, %eax
1 int func()
      return 42;
                            ret
                              # pop return address
 int main()
                            main:
      int answer = 1;
                            movl $1, %ebx
      answer = func();
      return 0;
                            call func
                            movl %eax, %ebx
                                                       %esp
```

```
func:
                            movl $42, %eax
1 int func()
      return 42;
                            ret
                              # pop return address
 int main()
                            main:
      int answer = 1;
                            movl $1, %ebx
      answer = func();
      return 0;
                            call func
                            movl %eax, %ebx
                                                       %esp
```

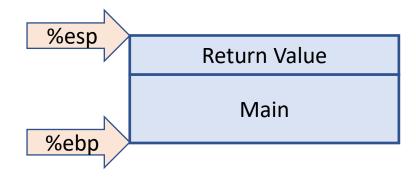
```
1 int func(int x, int y) {
2    return x + y;
3 }
4
5 int main() {
6    int answer = 1;
7    answer = func(2,3);
8    return 0;
9 }
```

- 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

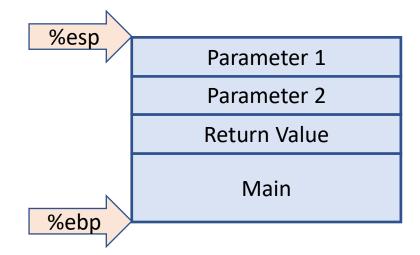
```
1 int func(int x, int y) {
2    return x + y;
3 }
4
5 int main() {
6    int answer = 1;
7    answer = func(2,3);
8    return 0;
9 }
```



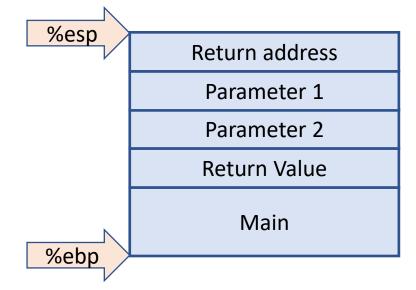
```
1 int func(int x, int y) {
2    return x + y;
3 }
4
5 int main() {
6    int answer = 1;
7    answer = func(2,3);
8    return 0;
9 }
```



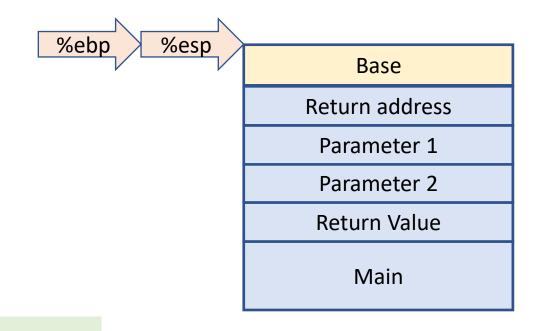
```
1 int func(int x, int y) {
2    return x + y;
3 }
4
5 int main() {
6    int answer = 1;
7    answer = func(2,3);
8    return 0;
9 }
```



```
1 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 }
```

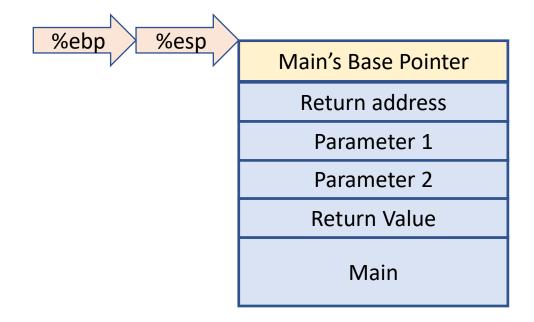


```
1 int func(int x, int y) {
2    return x + y;
3 }
4
5 int main() {
6    int answer = 1;
7    answer = func(2,3);
8    return 0;
9 }
```

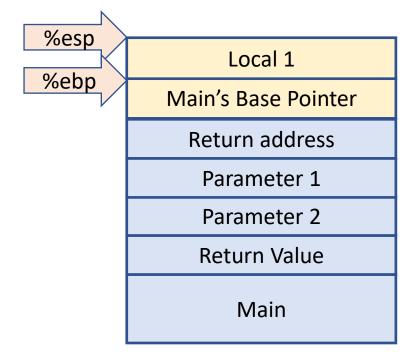


```
push %ebp
movl %esp, ebp
```

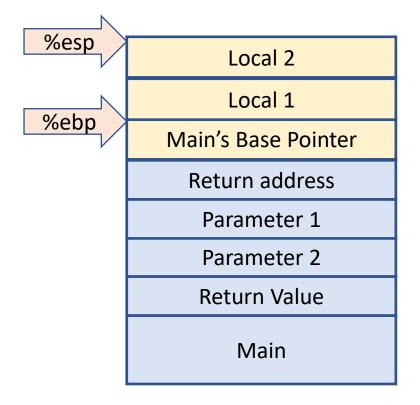
```
1 int func(int x, int y) {
2    return x + y;
3 }
4
5 int main() {
6    int answer = 1;
7    answer = func(2,3);
8    return 0;
9 }
```



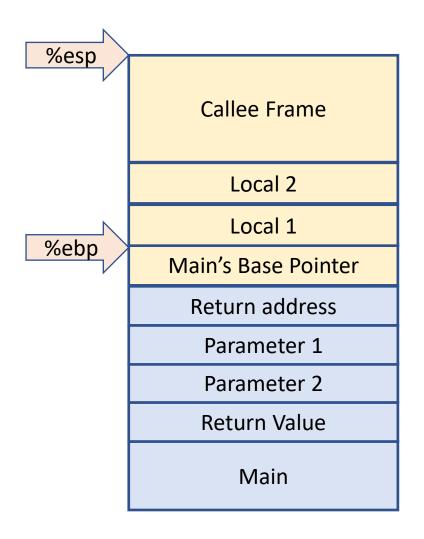
```
1 int func(int x, int y) {
2    return x + y;
3 }
4
5 int main() {
6    int answer = 1;
7    answer = func(2,3);
8    return 0;
9 }
```



```
1 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 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 int func(int x, int y) {
  return x + y;
5 int main() {
      int answer = 1;
      answer = func(2,3);
      return 0;
                   leave
                    movl %ebp, %esp
                    popl %ebp
                  ret
```

%esp Callee Frame Local 2 Local 1 %ebp Main's Base Pointer Return address Parameter 1 Parameter 2 Return Value Main

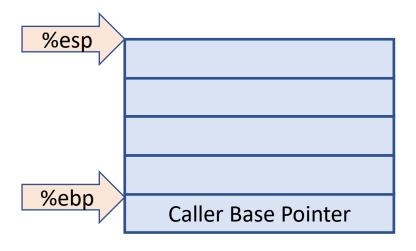
```
1 int func(int x, int y) {
2    return x + y;
3 }
4
5 int main() {
6    int answer = 1;
7    answer = func(2,3);
8    return 0;
9 }
```

- 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

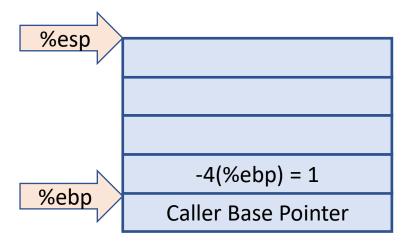
```
32
               %ebp
       pushl
33
       .cfi def cfa offset 8
34
       .cfi offset 5, -8
35
               %esp, %ebp
       movl
36
       .cfi def cfa register 5
37
             $16, %esp
       subl
38
                 x86.get pc thunk.ax
       call
       addl $ GLOBAL OFFSET TABLE , %eax
39
40
       movl $1, -4(\$ebp)
41
               $3
       pushl
42
               $2
       pushl
43
       call
               func
44
               $8, %esp
       addl
45
               %eax, -4(%ebp)
      movl
46
               $0, %eax
       movl
       leave
```



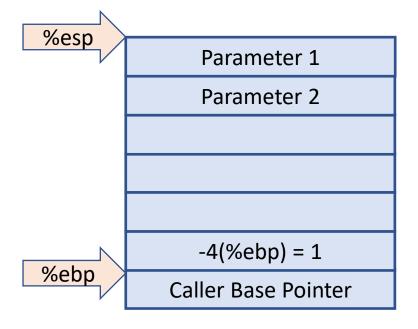
```
32
              %ebp
      pushl
33
       .cfi def cfa offset 8
34
       .cfi offset 5, -8
35
      movl %esp, %ebp
36
       .cfi def cfa register 5
37
      subl
            $16, %esp
38
                x86.get pc thunk.ax
      call
39
      addl $ GLOBAL OFFSET TABLE , %eax
40
      movl $1, -4(\$ebp)
41
             $3
      pushl
42
              $2
      pushl
43
      call
              func
44
              $8, %esp
      addl
45
              %eax, -4(%ebp)
      movl
46
              $0, %eax
      movl
      leave
```



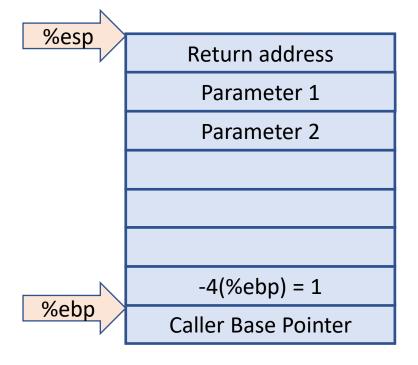
```
32
               %ebp
      pushl
33
       .cfi def cfa offset 8
34
       .cfi offset 5, -8
35
      movl %esp, %ebp
36
       .cfi def cfa register 5
37
            $16, %esp
       subl
                x86.get pc thunk.ax
38
       call
39
      addl $ GLOBAL OFFSET TABLE , %eax
40
      movl $1, -4(\$ebp)
41
              $3
      pushl
42
               $2
      pushl
43
      call
              func
44
              $8, %esp
       addl
45
              %eax, -4(%ebp)
      movl
46
              $0, %eax
      movl
       leave
```



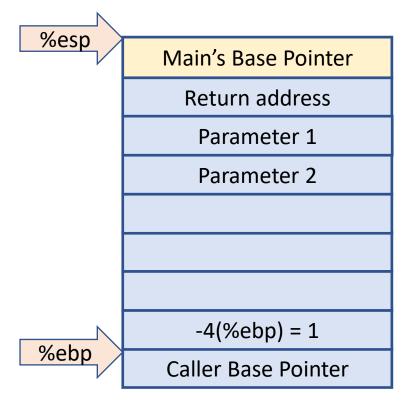
```
32
              %ebp
      pushl
33
       .cfi def cfa offset 8
34
       .cfi offset 5, -8
35
      movl %esp, %ebp
36
       .cfi def cfa register 5
37
           $16, %esp
      subl
38
                x86.get pc thunk.ax
      call
39
      addl $ GLOBAL OFFSET TABLE , %eax
40
      movl $1, -4 (\$ebp)
41
             $3
      pushl
42
              $2
      pushl
43
      call
              func
44
              $8, %esp
      addl
45
              %eax, -4(%ebp)
      movl
46
              $0, %eax
      movl
      leave
```



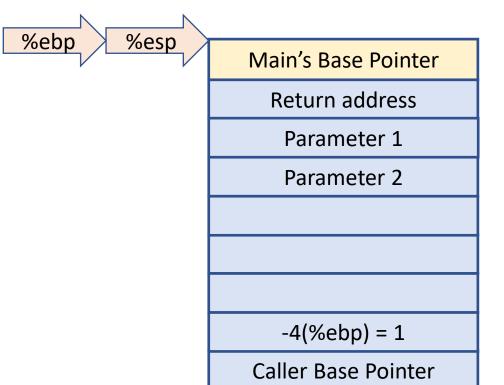
```
32
              %ebp
      pushl
33
       .cfi def cfa offset 8
34
       .cfi offset 5, -8
35
      movl %esp, %ebp
36
       .cfi def cfa register 5
37
      subl $16, %esp
                x86.get pc thunk.ax
38
      call
      addl $ GLOBAL OFFSET TABLE , %eax
39
40
      movl $1, -4 (\$ebp)
41
             $3
      pushl
42
              $2
      pushl
43
      call
              func
44
              $8, %esp
      addl
45
              %eax, -4(%ebp)
      movl
46
              $0, %eax
      movl
      leave
```



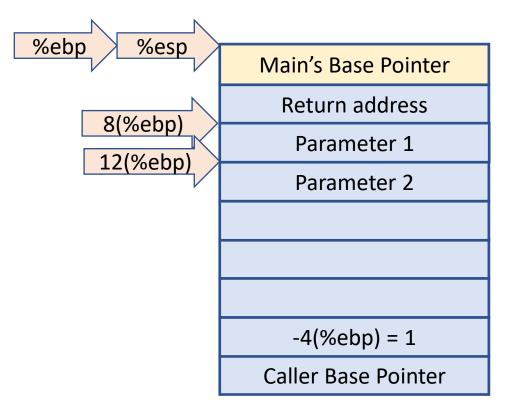
```
func:
   .LFB0:
       .cfi startproc
      endbr32
 8
 9
      pushl %ebp
       .cfi def cfa offset 8
10
       .cfi offset 5, -8
11
12
      movl %esp, %ebp
       .cfi def cfa register 5
13
      call x86.get pc thunk.ax
14
      addl $ GLOBAL OFFSET TABLE , %eax
15
      movl 8(%ebp), %edx
16
17
      movl 12(%ebp), %eax
18
      addl %edx, %eax
19
      popl %ebp
20
       .cfi restore 5
21
       .cfi def cfa 4, 4
       ret
```



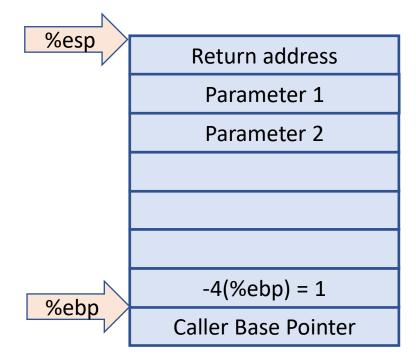
```
func:
   .LFB0:
       .cfi startproc
 8
      endbr32
 9
      pushl %ebp
10
       .cfi def cfa offset 8
       .cfi offset 5, -8
11
12
           %esp, %ebp
      movl
13
       .cfi def cfa register 5
14
       call x86.get pc thunk.ax
      addl $ GLOBAL OFFSET TABLE , %eax
15
      movl 8(%ebp), %edx
16
17
      movl 12(%ebp), %eax
18
      addl %edx, %eax
19
            %ebp
      popl
20
       .cfi restore 5
21
       .cfi def cfa 4, 4
       ret
```



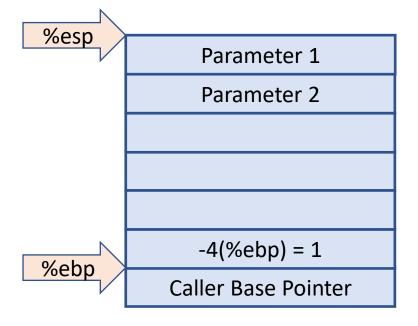
```
func:
   .LFB0:
       .cfi startproc
 8
      endbr32
 9
      pushl %ebp
10
       .cfi def cfa offset 8
       .cfi offset 5, -8
11
      movl %esp, %ebp
12
13
       .cfi def cfa register 5
14
       call x86.get pc thunk.ax
      addl $ GLOBAL OFFSET TABLE , %eax
15
      movl 8(%ebp), %edx
16
17
      movl 12(%ebp), %eax
18
      addl %edx, %eax
19
      popl %ebp
20
       .cfi restore 5
21
       .cfi def cfa 4, 4
       ret
```



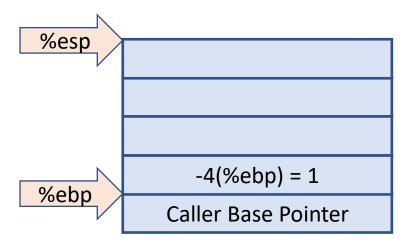
```
func:
   .LFB0:
       .cfi startproc
      endbr32
 8
 9
      pushl %ebp
10
       .cfi def cfa offset 8
       .cfi offset 5, -8
11
      movl %esp, %ebp
12
13
       .cfi def cfa register 5
14
       call x86.get pc thunk.ax
      addl $ GLOBAL OFFSET TABLE , %eax
15
      movl 8(%ebp), %edx
16
      movl 12(%ebp), %eax
17
18
      addl %edx, %eax
19
      popl %ebp
20
       .cfi restore 5
21
       .cfi def cfa 4, 4
       ret
```



```
32
              %ebp
      pushl
33
       .cfi def cfa offset 8
34
       .cfi offset 5, -8
35
      movl %esp, %ebp
36
       .cfi def cfa register 5
37
            $16, %esp
      subl
38
                x86.get pc thunk.ax
      call
39
      addl $ GLOBAL OFFSET TABLE , %eax
40
      movl $1, -4 (\$ebp)
41
             $3
      pushl
42
              $2
      pushl
43
      call
              func
44
              $8, %esp
      addl
45
              %eax, -4(%ebp)
      movl
46
              $0, %eax
      movl
      leave
```



```
32
              %ebp
      pushl
33
       .cfi def cfa offset 8
34
       .cfi offset 5, -8
35
      movl %esp, %ebp
36
       .cfi def cfa register 5
37
            $16, %esp
       subl
                x86.get pc thunk.ax
38
       call
39
      addl $ GLOBAL OFFSET TABLE , %eax
40
      movl $1, -4(%ebp)
41
              $3
      pushl
42
               $2
      pushl
43
      call
              func
44
              $8, %esp
       addl
45
              %eax, -4(%ebp)
      movl
46
              $0, %eax
      movl
       leave
```



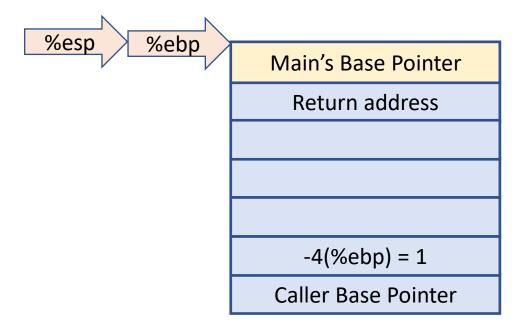
```
32
               %ebp
       pushl
33
       .cfi def cfa offset 8
34
       .cfi offset 5, -8
35
               %esp, %ebp
       movl
36
       .cfi def cfa register 5
37
             $16, %esp
       subl
38
                 x86.get pc thunk.ax
       call
       addl $ GLOBAL OFFSET TABLE , %eax
39
40
       movl $1, -4(\$ebp)
41
               $3
       pushl
42
               $2
       pushl
43
       call
               func
44
               $8, %esp
       addl
45
               %eax, -4(%ebp)
      movl
46
               $0, %eax
       movl
       leave
```



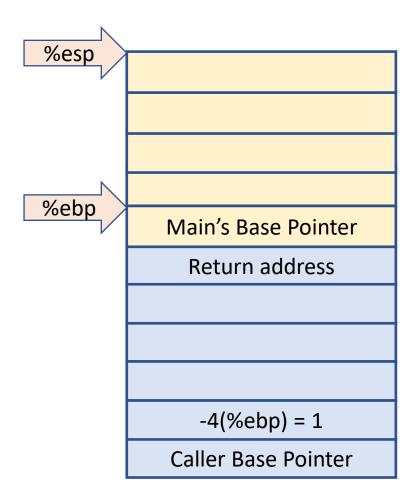
```
int func()
       int a = 1;
       int b = 2;
       int c = 3;
       int d = 4;
       return a + b + c + d;
   int main()
10
       int answer = 1;
       answer = func();
       return 0;
```

- 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

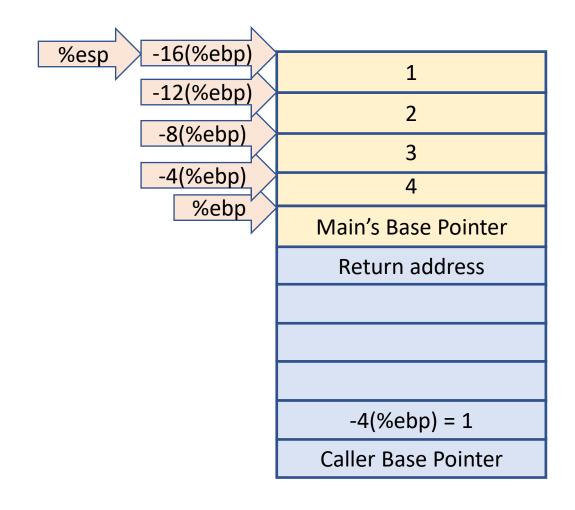
```
func:
 6
       pushl
                %ebp
                %esp, %ebp
       movl
 8
       subl
               $16, %esp
               $1, -16(%ebp)
       movl
10
               $2, -12(%ebp)
       movl
11
               $3, -8(%ebp)
       movl
12
               $4, -4(%ebp)
       movl
13
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



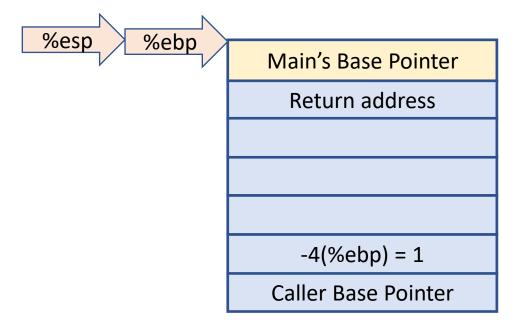
```
func:
               %ebp
 6
       pushl
               %esp, %ebp
       movl
 8
       subl
              $16, %esp
            $1, -16(%ebp)
       movl
10
              $2, -12(%ebp)
      movl
11
              $3, -8(%ebp)
       movl
12
              $4, -4(%ebp)
       movl
13
          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



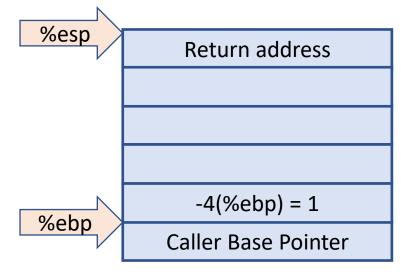
```
func:
              %ebp
 6
      pushl
              %esp, %ebp
      movl
 8
      subl
              $16, %esp
      movl $1, -16(%ebp)
10
      movl $2, -12(%ebp)
11
      movl $3, -8 (\$ebp)
12
      movl $4, -4(\$ebp)
13
      movl -16 (%ebp), %edx
14
      movl -12 (%ebp), %eax
15
      addl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



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

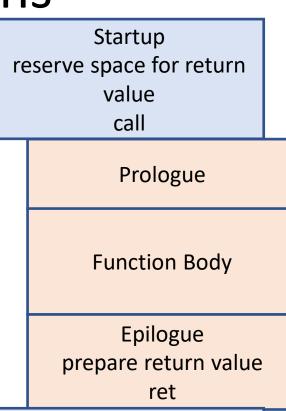


```
func:
               %ebp
 6
       pushl
               %esp, %ebp
       movl
 8
       subl
               $16, %esp
 9
             $1, -16(%ebp)
       movl
10
               $2, -12(%ebp)
       movl
11
            $3, -8(%ebp)
       movl
12
               $4, -4(%ebp)
       movl
13
    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



Calling Conventions

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



Cleanup store return value

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

Startup store %eax, %ecx, %edx

Prologue store %ebx, %esi, %edi

Function Body

Epilogue restore %ebx, %esi, %edi

Cleanup restore %eax, %ecx, %edx

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