Intro to Operating Systems

CS461 / ECE422 – UIUC Spring 2017

By Zhengping Wang

Outline

x86 Assembly Instructions

x86 32-bit ISA

Registers

Stack

Stack Frame

Assembly Instructions (AT&T Syntax)

Opcode-Source-Destination
mov $0x15,%ebx

Address calculation:
displacement(base reg, offset reg, multiplier)

mov 8(%ebp),%eax M[EBP+8] to eax

mov 12(,%edx,4),%eax M[EDX*4+12] to eax

Example

```
mov   $11,%eax

mov   $12,%ebx

mov   $8,%ecx

add   %ecx,%ebx

sub   %ecx,%eax
```

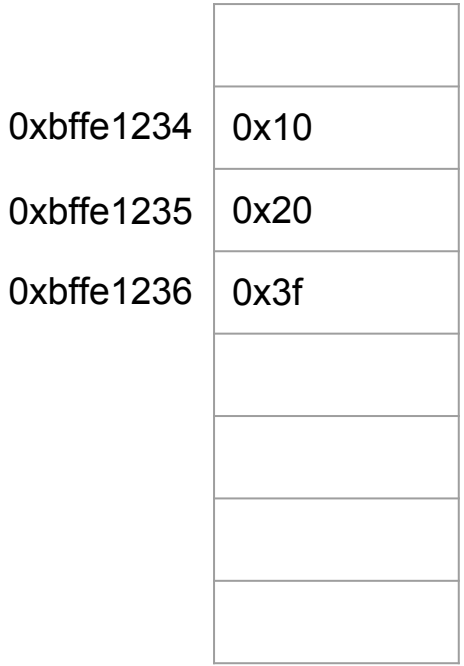
Assembly Instructions (AT&T Syntax)

push, pop, jmp, call, mov, lea, xor, cmp, dec, inc, int, leave, ret, and a lot more!

```
cmp    $0xffffff83,%eax
jne    <label>

call   foo

mov    0x8(%ebp),%eax
mov    $0x15,%ebx

lea    -0x10(%ebp),%eax

xor    %ecx,%ecx
```

32-bit x86 ISA

- 1 byte = 8 bits
- char -> 1 byte
- integer -> 4 bytes
- word -> 2 bytes (in gdb, word -> 4 bytes)
- long -> 4 bytes
- Memory address -> 4 bytes
- Pointer -> ?
- Registers -> 4 bytes
- Each memory location -> 1 byte

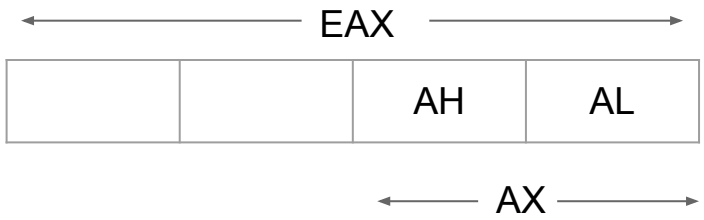


Registers (4 Bytes)

General Purpose: **EAX, EBX, ECX, EDX**, EDI, ESI

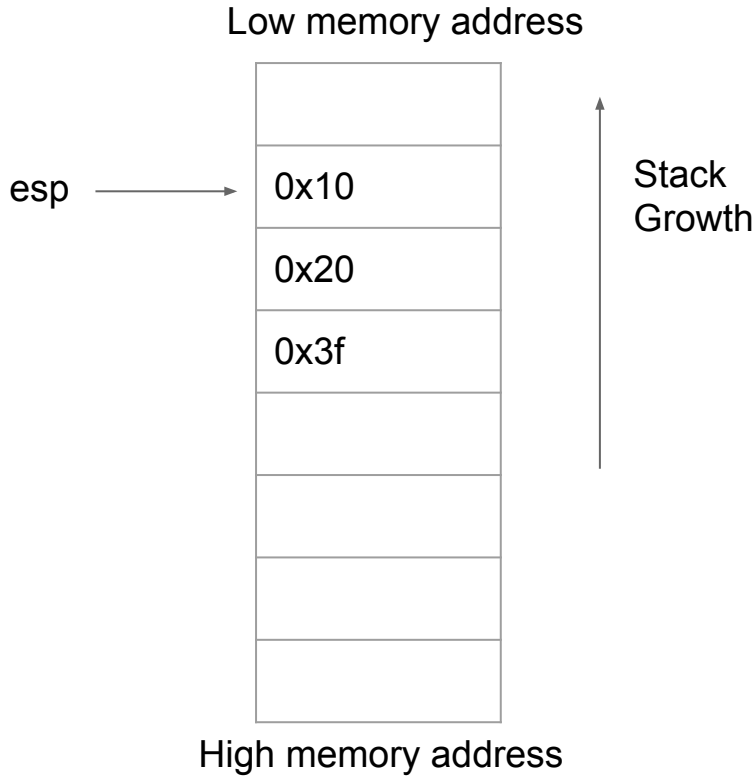
Special:

- EIP: Instruction pointer
- ESP: Stack pointer
- EBP: Base pointer



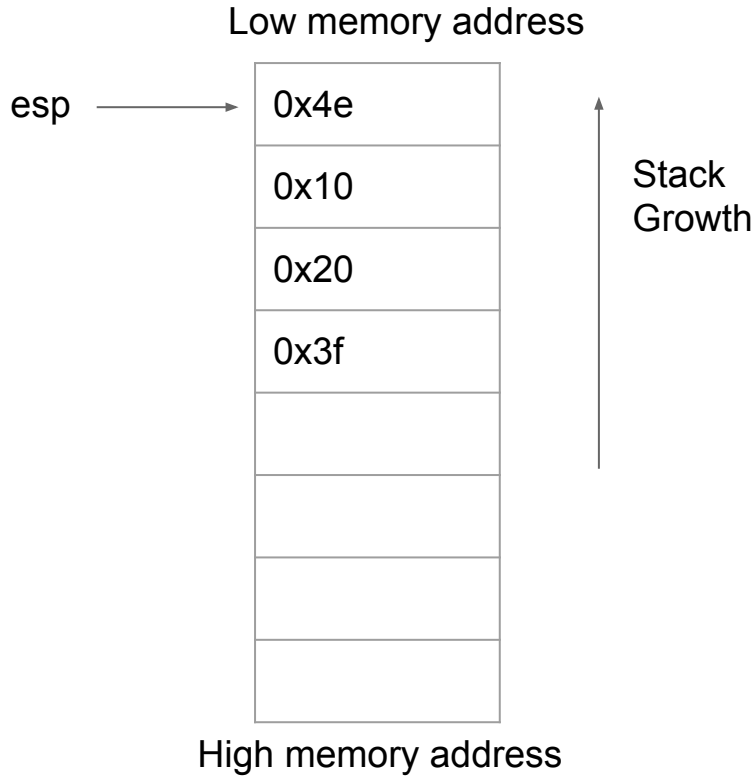
Stack

ESP(stack pointer) points to top of stack



Stack

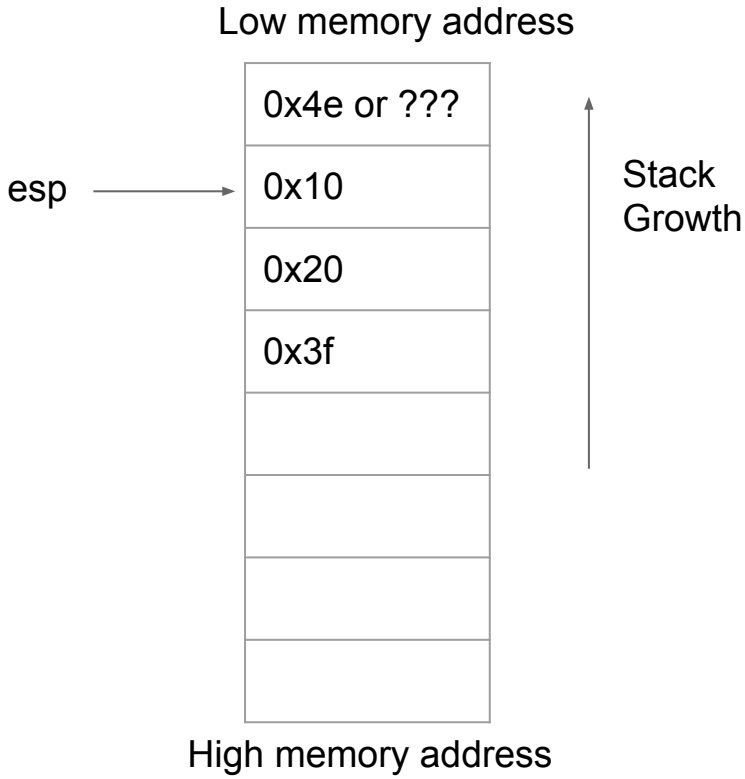
push $0x4e



Stack

push $0x4e

pop %eax (eax contains 0x4e)

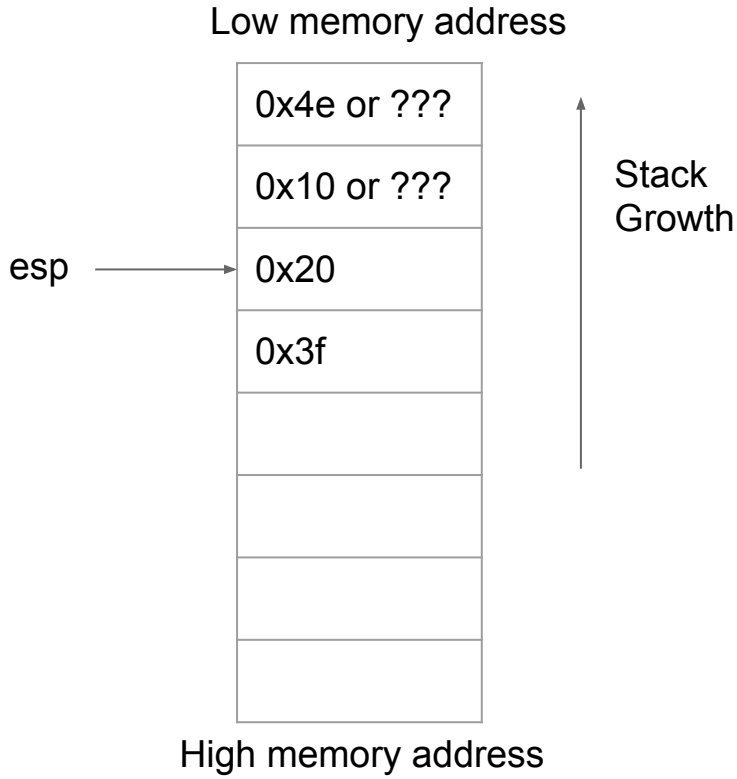


Stack

push $0x4e

pop %eax (eax contains 0x4e)

pop %ebx (ebx contains 0x10)



Exercise

9 -> EAX

0 -> EBX

ECX -> EDX

M[ECX] -> EBX

M[EDX+4] -> EAX

Opcode-Source-Destination

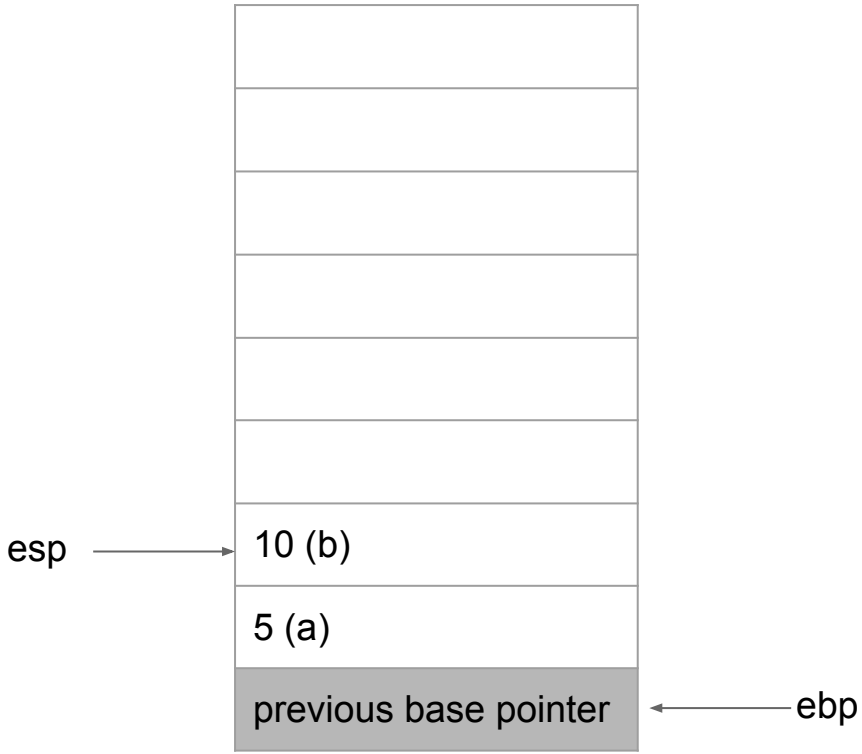
Address calculation:
displacement(base reg, offset reg, multiplier)

Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*

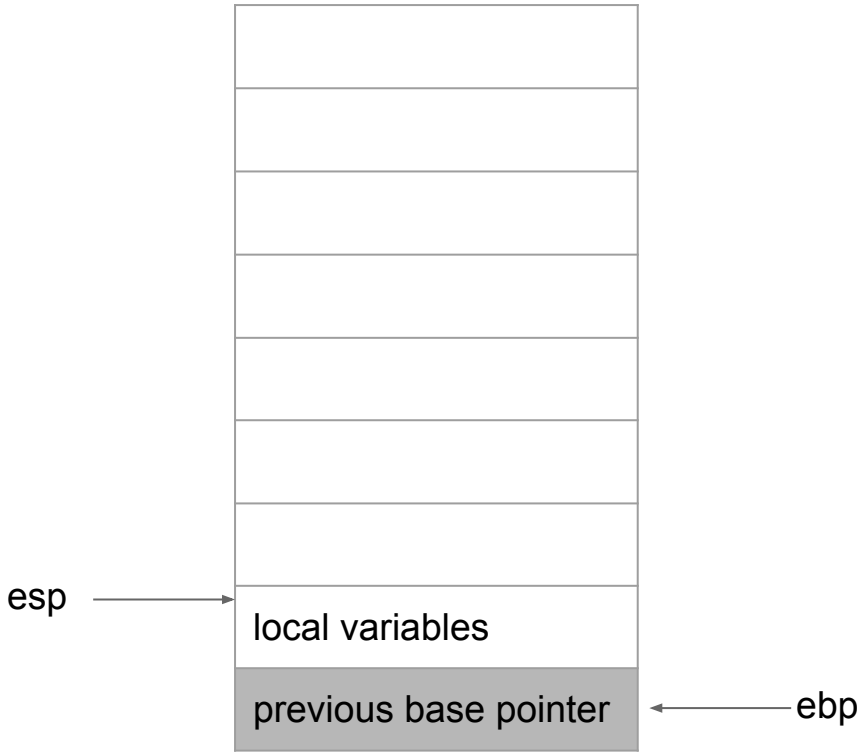
ex:
int a = 5; (push $5)
int b = 10; (push $10)



Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*

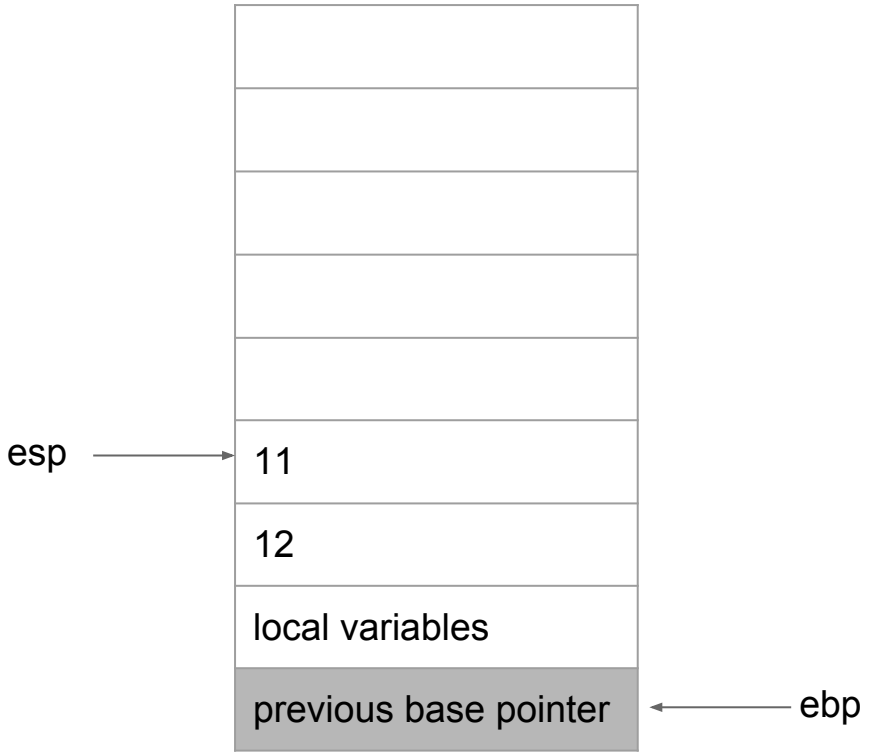


Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*

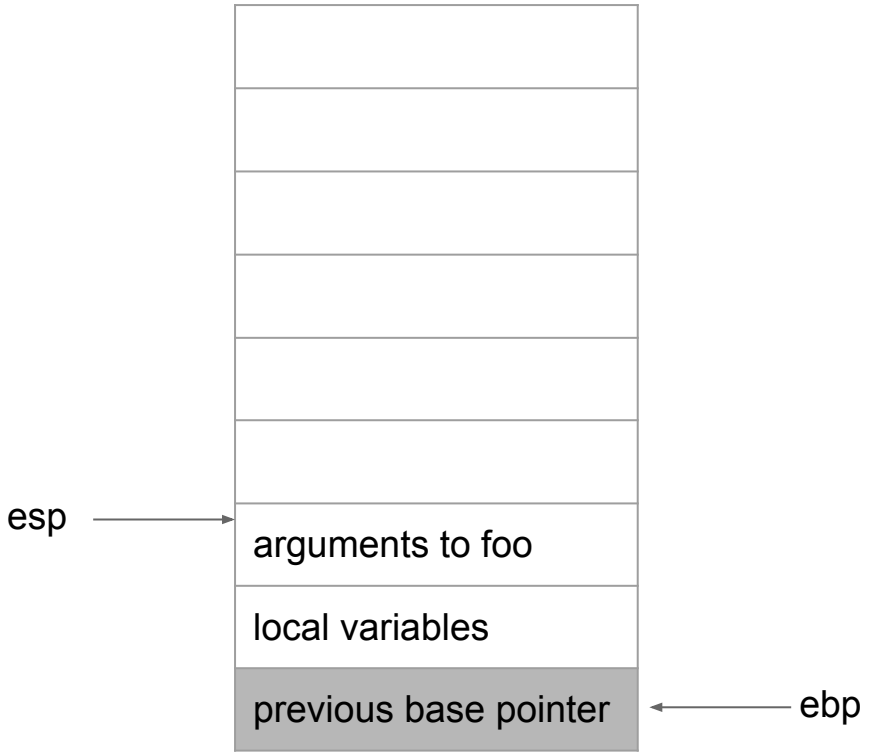
ex:
foo takes two integers,
in main: foo(11,12); (push $12, push $11)



Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*



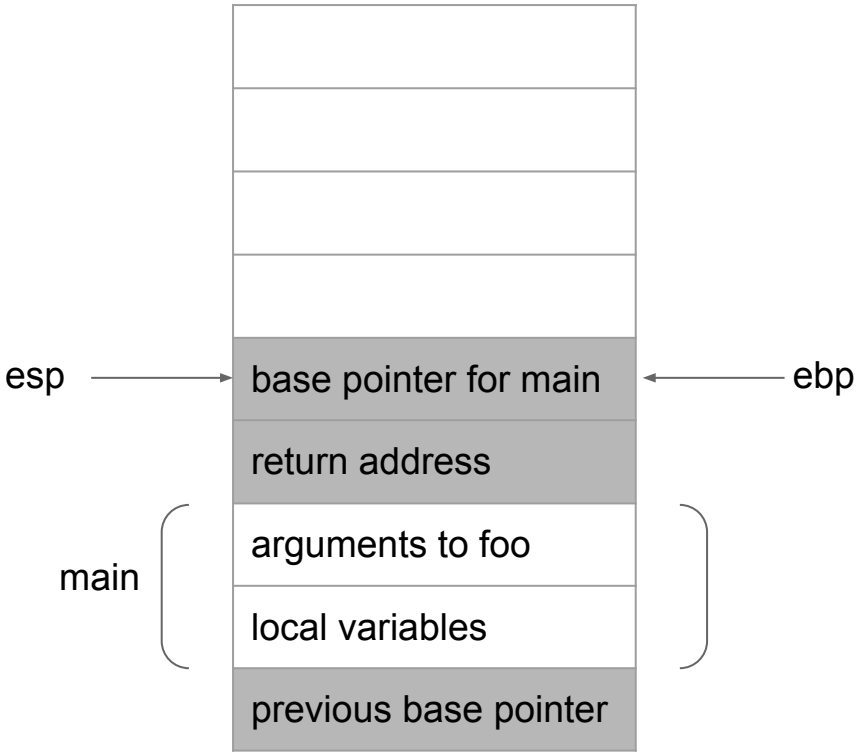
Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*
3. Set up stack frame for *foo*

assembly:

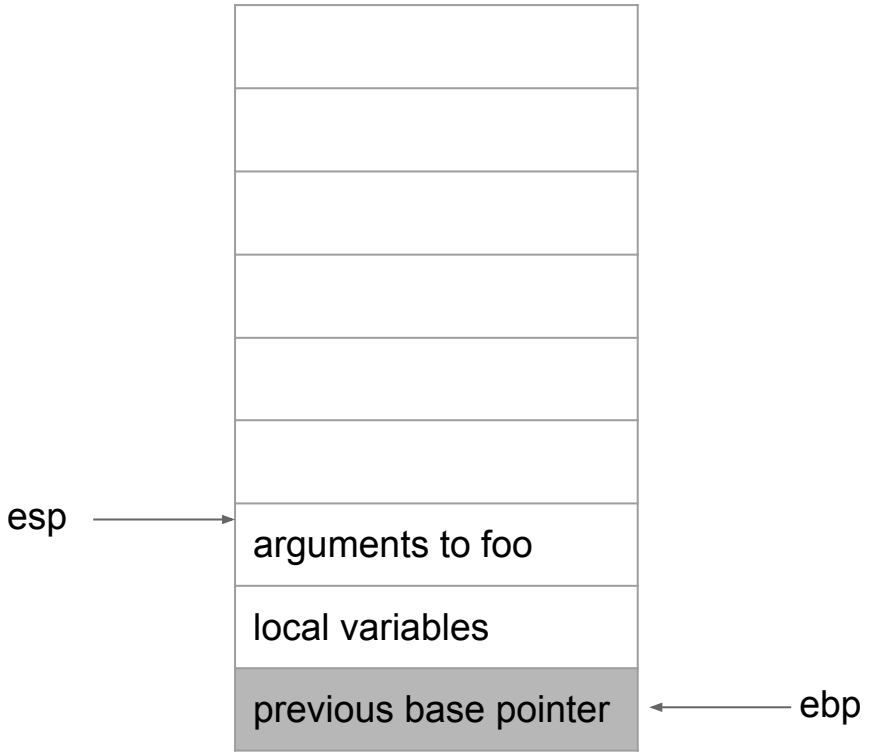
```
call foo
...
foo:
push %ebp
mov %esp,%ebp
...
```



Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*
3. Set up stack frame for *foo*



Stack Frame

example: *main* calls *foo*

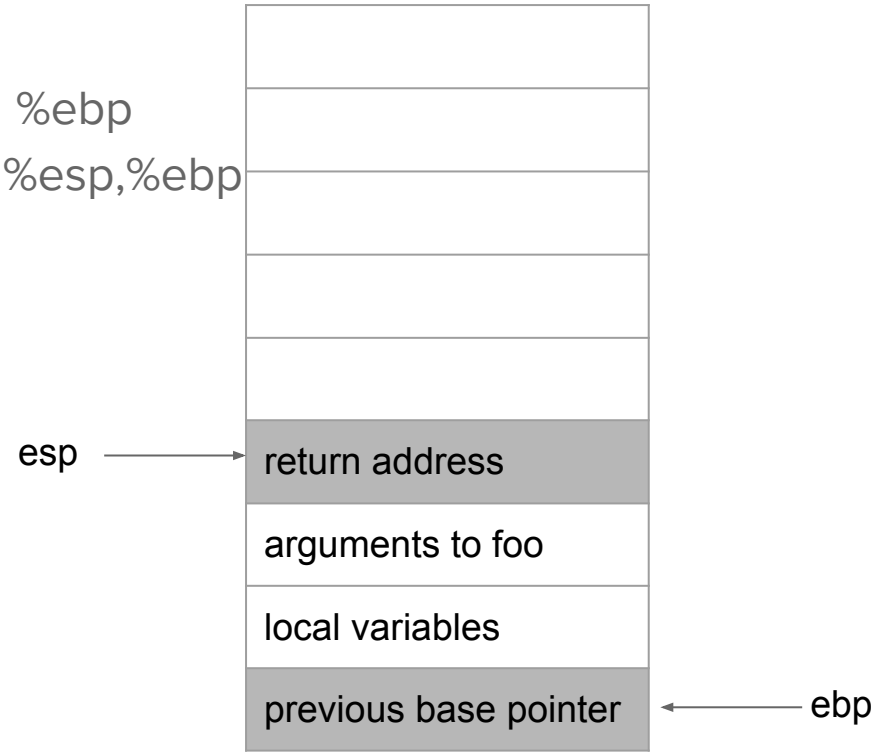
1. Do stuff in *main*
2. Set up arguments to call *foo*
3. Set up stack frame for *foo*

assembly: call foo

(push %eip;

address of foo's first instruction -> eip)

assembly:
call foo
...
foo:
push %ebp
mov %esp,%ebp
...



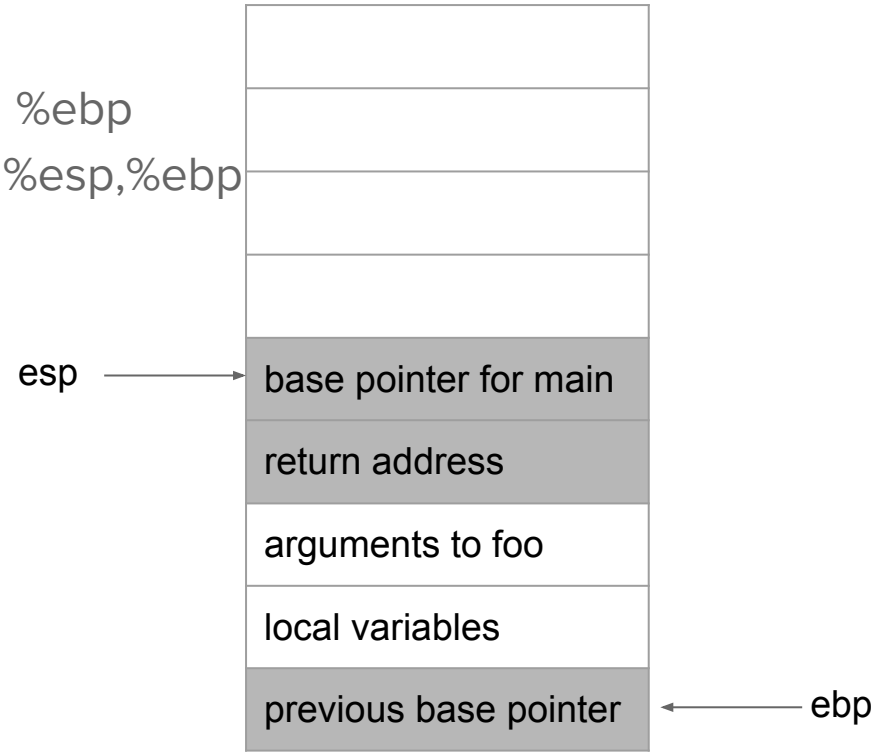
Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*
3. Set up stack frame for *foo*

assembly: push %ebp

assembly:
call foo
...
foo:
push %ebp
mov %esp,%ebp
...



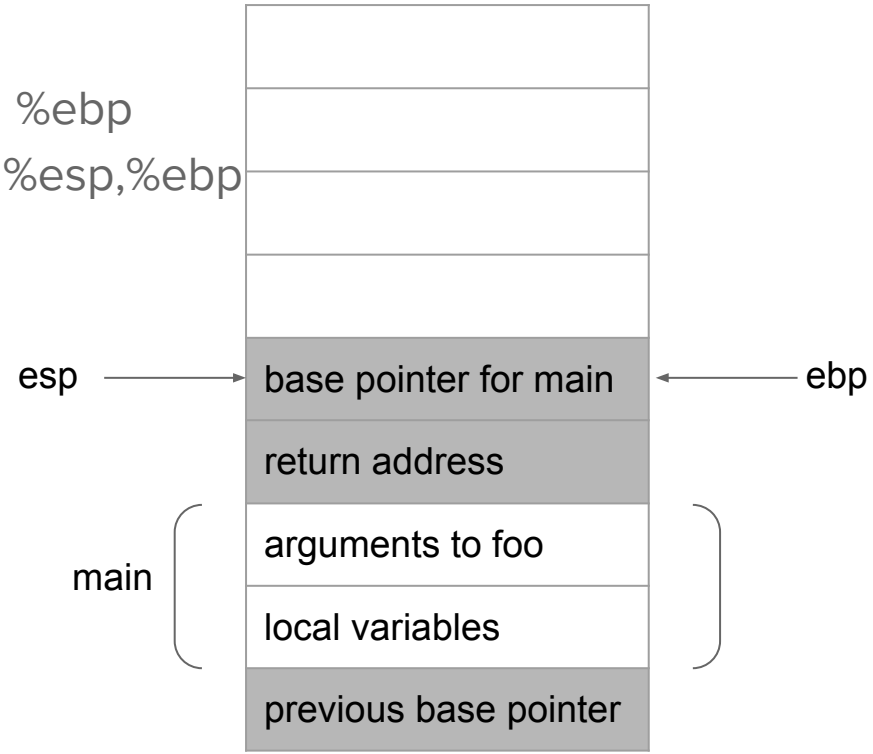
Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*
3. Set up stack frame for *foo*

assembly: mov %esp,%ebp

assembly:
call foo
...
foo:
push %ebp
mov %esp,%ebp
...



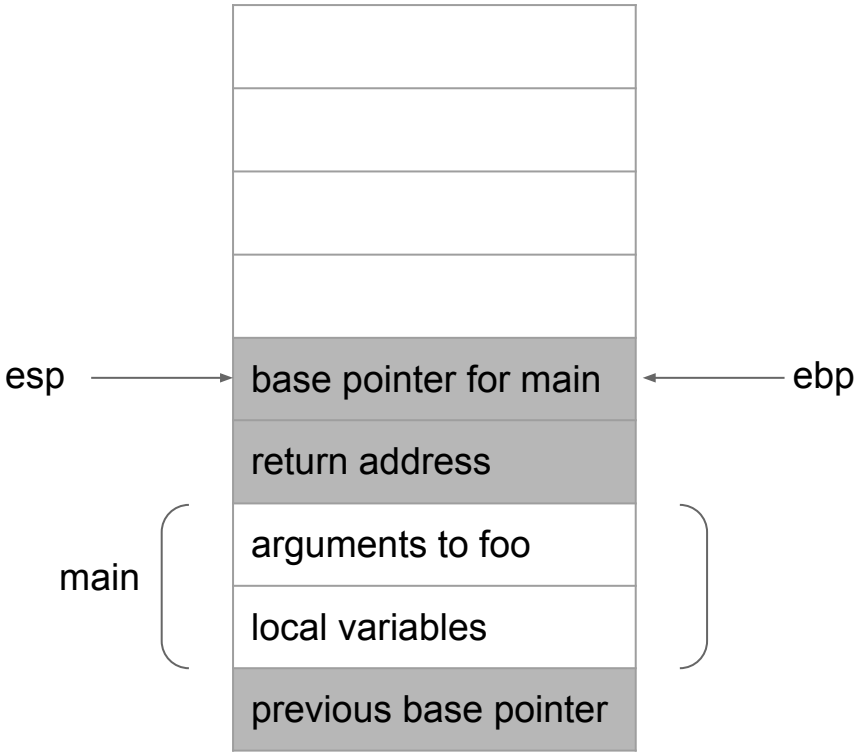
Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*
3. Set up stack frame for *foo*

assembly:

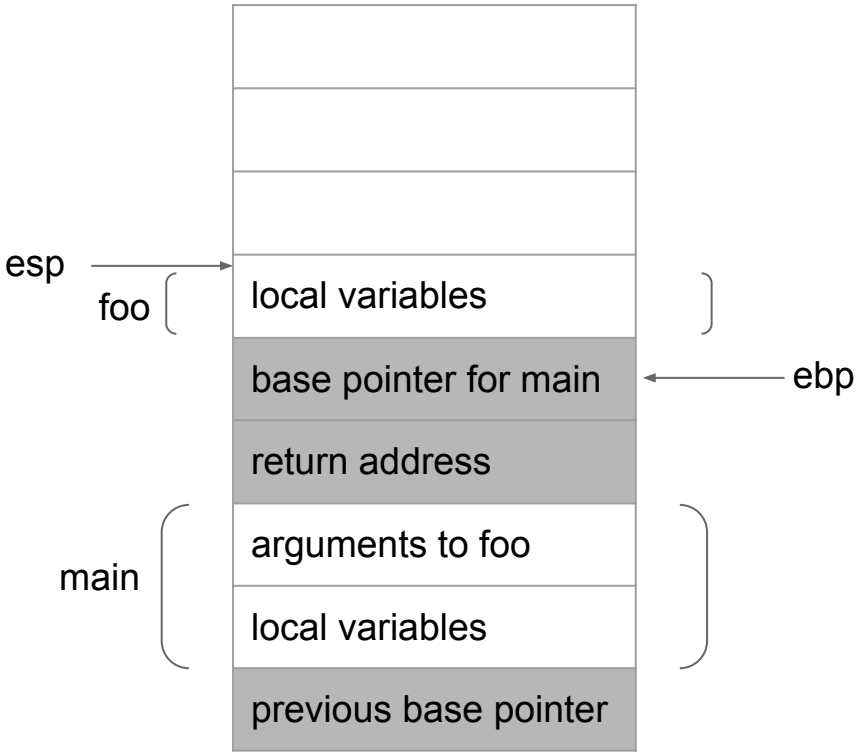
```
call foo
...
foo:
push %ebp
mov %esp,%ebp
...
```



Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*
3. Set up stack frame for *foo*
4. Do stuff in *foo*

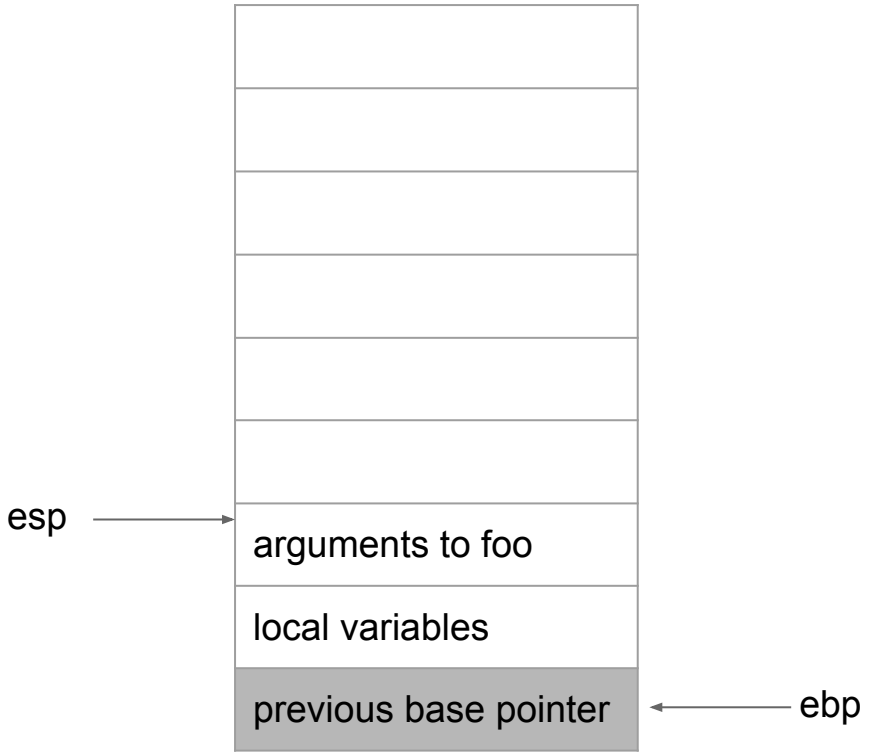


Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*
3. Set up stack frame for *foo*
4. Do stuff in *foo*
5. Return to *main*

assembly:
leave
ret

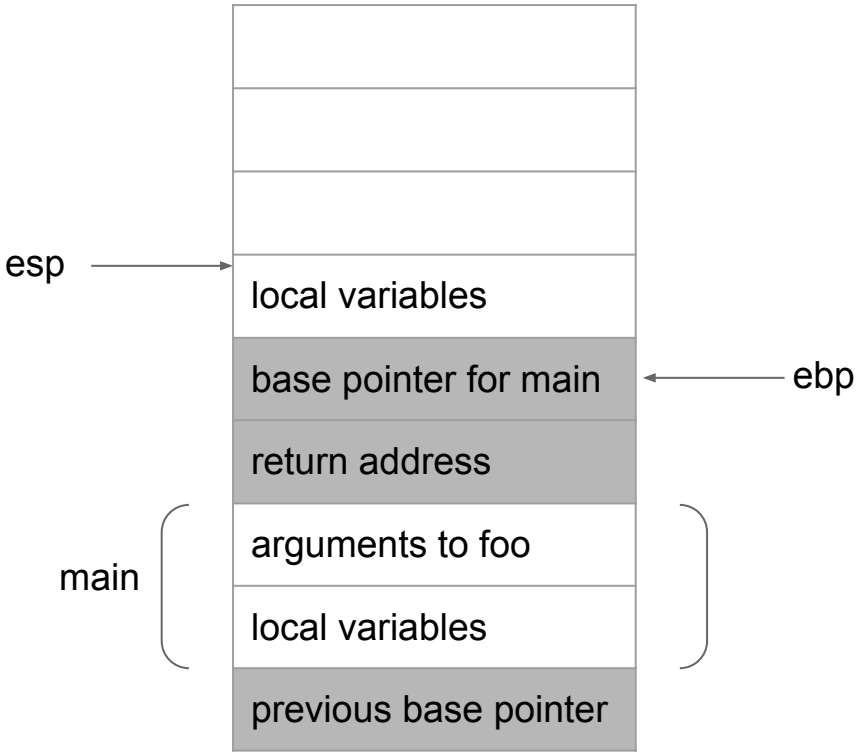


Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*
3. Set up stack frame for *foo*
4. Do stuff in *foo*
5. Return to *main*

assembly:
leave
ret



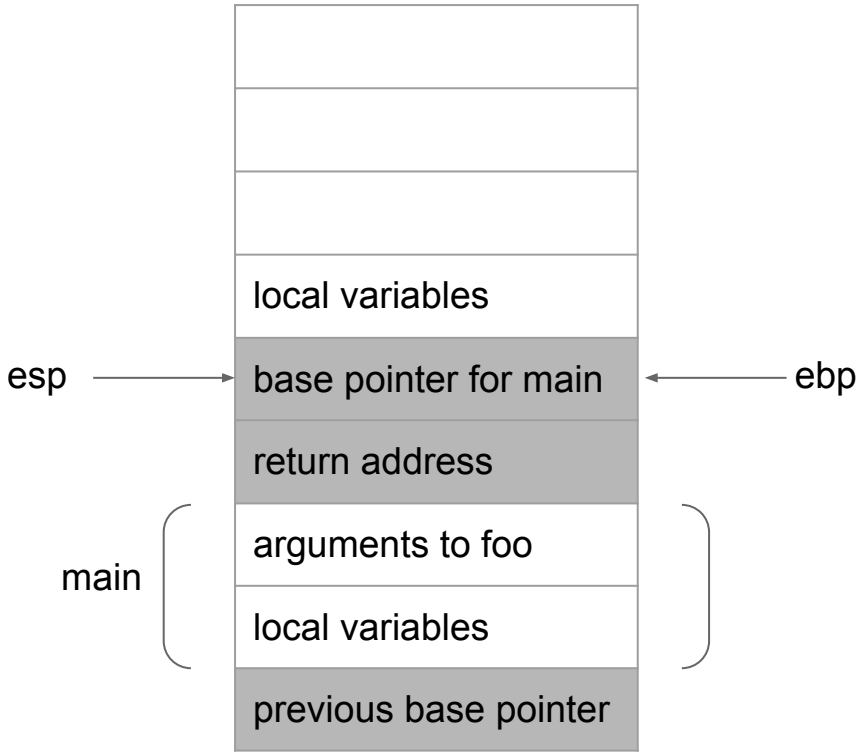
Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*
3. Set up stack frame for *foo*
4. Do stuff in *foo*
5. Return to *main*

assembly: leave

(**mov %ebp, %esp;**
  pop %ebp)



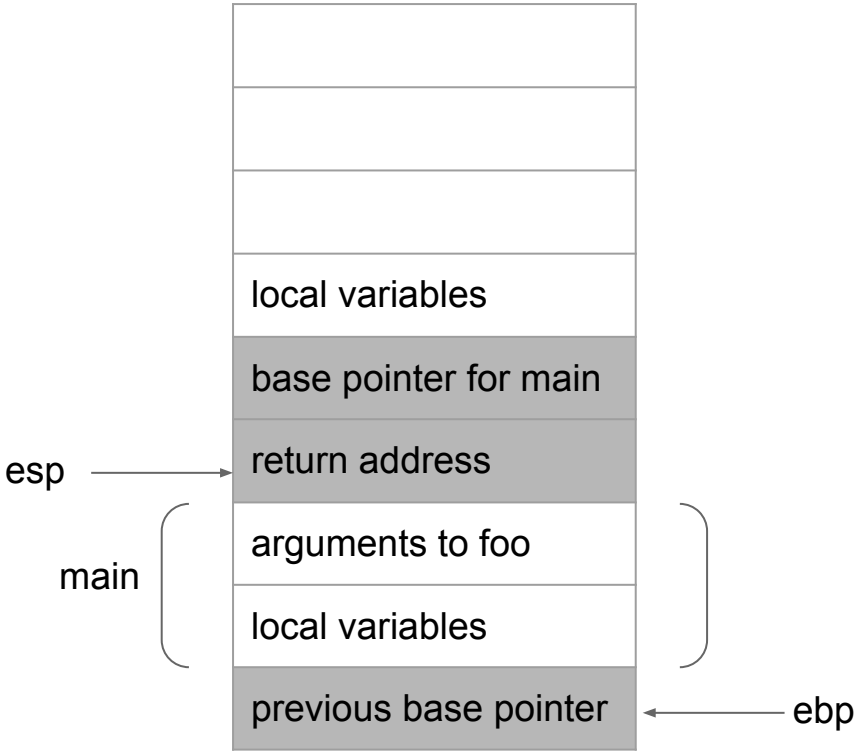
Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*
3. Set up stack frame for *foo*
4. Do stuff in *foo*
5. Return to *main*

assembly: leave

(mov %ebp, %esp;
pop %ebp)



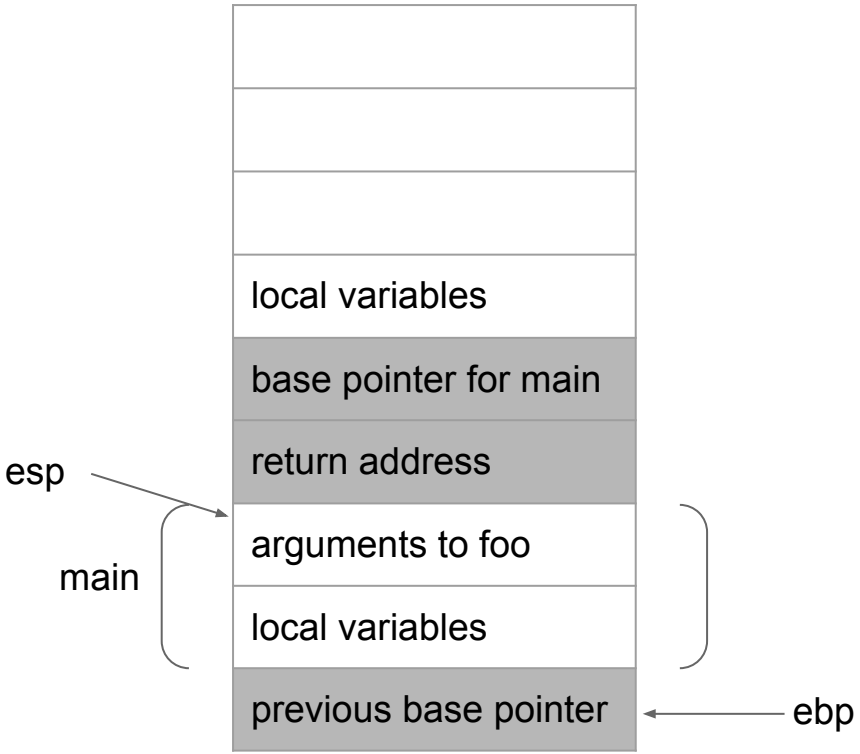
Stack Frame

example: *main* calls *foo*

1. Do stuff in *main*
2. Set up arguments to call *foo*
3. Set up stack frame for *foo*
4. Do stuff in *foo*
5. Return to *main*

assembly: ret

(pop %eip)



Exercise - Translate to x86 Assembly

```
int main()
{
        int a = 3;
        addnumbers(2,6);
}

int addnumbers(int x, int y)
{
        int b = 1;
        b = x+y;

        return b;
}
```

Possible Solution

```
main:
push %ebp              //setting up stack frame
mov %esp,%ebp
push $3                //int a = 3;
push $6                //addnumbers(2,6);
push $2
call addnumbers
leave
ret
```

```
addnumbers:
push %ebp              //setting up stack frame
mov %esp,%ebp
push $1                //int b = 1;
mov 8(%ebp),%eax //b = x+y;
add 12(%ebp),%eax
mov %eax,(%esp)
leave
ret
```

Next Week

MP1 Checkpoint 1