Lecture 04 – Control Flow II

Stephen Checkoway

CS 343 – Fall 2020

Based on Michael Bailey’s ECE 422

32-bit x86 architecture overview

- 8 general purpose registers eax, ebx, ecx, edx, esi, edi, ebp, esp
  - esp is the stack pointer
  - ebp is the frame pointer (optional)
  - Others are used for integer and pointer operations
  - 16- and 8-bit parts of the registers can be named (ax is least significant 16 bits of eax, al is least sig. 8 bits of eax, etc.)
- Instruction pointer eip holds the address of the next instruction to execute
- eflags register has bits like the zero flag or the carry flag that are set by arithmetic and logical operations, used for conditional control flow

Some x86 instructions (AT&T notation)

- mov src, dest ; Copies src to dest
- Arithmetic and bit operations
  - add src, dest ; computes dest + src, stores in dest
  - sub src, dest ; computes dest – src, stores in dest
  - or, and, xor all work the same way; mul/div use specific registers
- Stack operations
  - push src ; decrements esp by 4, writes src to stack
  - pop dest ; reads top of stack into dest, increments esp by 4

Some x86 instructions (AT&T notation)

- Function calls
  - call foo ; calls the function foo, pushes the address of the next instruction onto the stack
  - leave ; equivalent to movl $ebp, $esp followed by popl $ebp
  - ret ; pops the top of the stack into eip (returns from a function)
- Control flow
  - cmp src2, src1 ; computes src1 – src2 and sets eflags register
  - test src2, src1 ; computes src1 & src2 (bitwise-and) and sets eflags
  - jz label ; jump to label if the zero flag is set
  - jnz label ; jump to label if the zero flag is not set
  - jc label ; jump to label if the carry flag is set
  - jnc label ; jump tot label if the carry flag is not set
  - jmp label ; unconditionally jump to label

Instruction suffixes

- l — (long) 32 bits
- w — (word) 16 bits
- b — (byte) 8 bits

- Examples
  - movw %ax, %dx ; Copies least sig. 16 bits of eax to least sig. 16 bits of edx
  - pushl %edi
  - subl $16, %esp ; Decrements esp by 16
  - cmpl %edx, %eax ; computes eax – edx and sets eflags based on the result

x86 operands

- Constants are prefixed with $
- Registers are prefixed with %
  - movb $8, %bl
- Read/writing to memory has several forms
  - (%eax) ; Refers to the 1, 2, or 4 bytes at address stored in eax
  - -8(%esp) ; Address is %esp – 8
  - 4(%esi, %eax) ; Address is esi + eax + 4
  - 16(%eax, %edx, 4) ; Address is eax + 4*edx + 16

Using memory operands

- Load 4 bytes from ebp + 4 into eax
  - movl 4(%ebp), %eax
- Store 1 byte from dl (least sig. 8-bits of edx) to address edi
  - movb %dl, (%edi)
- Add 4 bytes from address edx to eax and store in eax
  - addl (%edx), %eax
- Xor the constant 0x5555AAAA with 4 bytes at address 8+ebp
  - xorl $0x5555AAAA, 8(%ebp)

What values do eax and edx hold after this?

```
movl      $30, %eax
movl      $10, %edx
subl      %eax, %edx
addl      %eax, %eax
```

A. eax = 40, edx = 10

B. eax = 60, edx = 40

C. eax = 60, edx = -20

D. eax = -40, edx = 10

Function calls on 32-bit x86

- Stack grows down (from high to low addresses)
- Stack consists of 4-byte slots
- esp points to the bottom most “in-use” slot
- ebp “frame pointer” points to the previous ebp on the stack (if used)
- call pushes the return address onto the stack
- Function call arguments can be accessed at a positive offset from ebp
  8(%ebp), 12(%ebp), 16(%ebp), etc.
- Local variables can be accessed at a negative offset from ebp
  -4(%ebp), -8(%ebp), -12(%ebp), etc.

Warning!

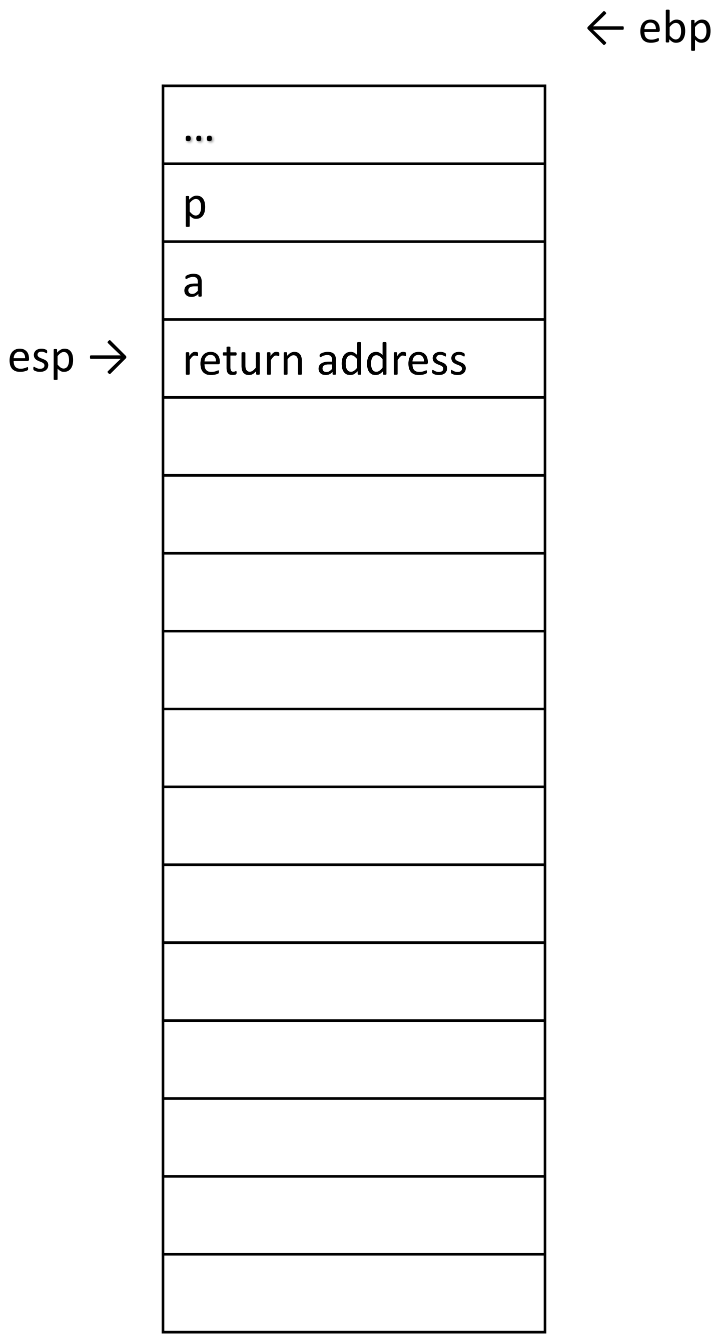
- For most of these slides, the stack is drawn with low addresses on the bottom and high addresses on the top. The stack grows down both numerically and pictorially.

Function call example

```
int foo(int a, char *p) {
        int b = atoi(p);
        return a + b;
}
```

eip →

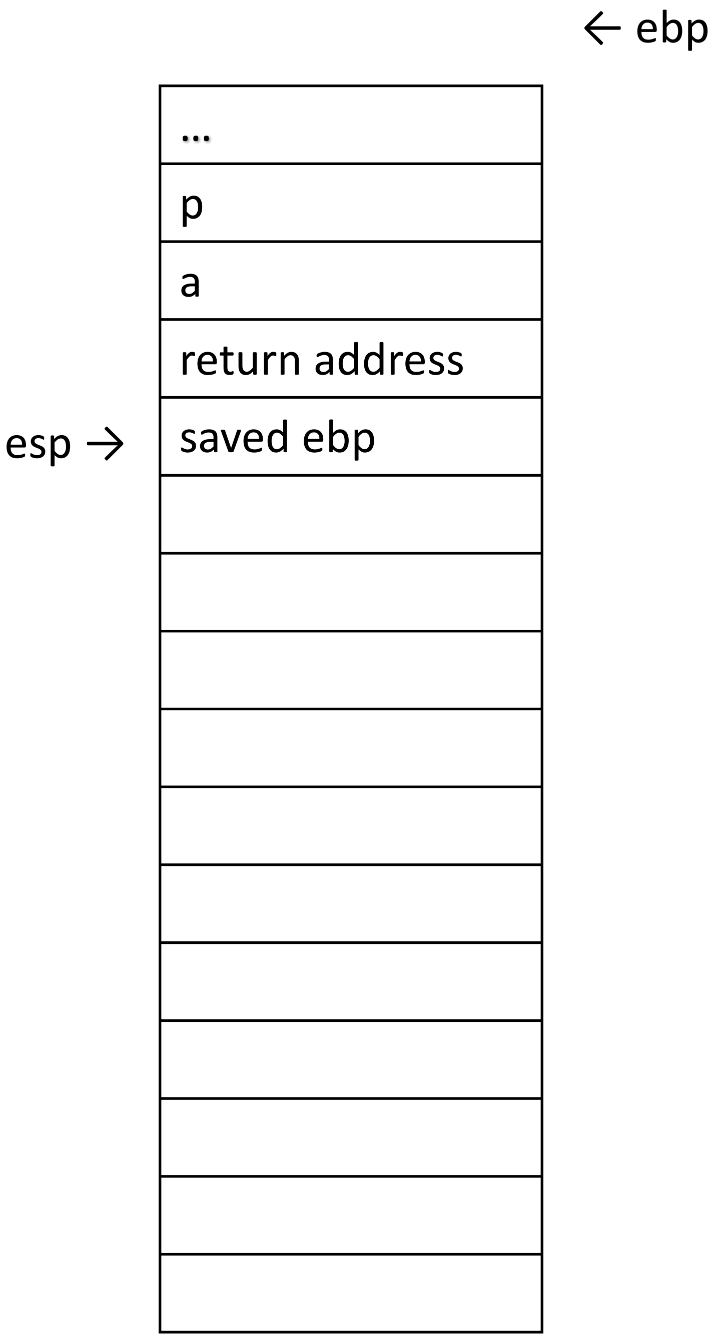
```
foo:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        movl    12(%ebp), %eax
        movl    %eax, (%esp)
        call    atoi
        movl    %eax, -12(%ebp)
        movl    -12(%ebp), %eax
        movl    8(%ebp), %edx
        addl    %edx, %eax
        leave
        ret
```



Function call example

```
int foo(int a, char *p) {
        int b = atoi(p);
        return a + b;
}
```

```
      foo:
              pushl   %ebp
eip →         movl    %esp, %ebp
              subl    $40, %esp
              movl    12(%ebp), %eax
              movl    %eax, (%esp)
              call    atoi
              movl    %eax, -12(%ebp)
              movl    -12(%ebp), %eax
              movl    8(%ebp), %edx
              addl    %edx, %eax
              leave
              ret
```

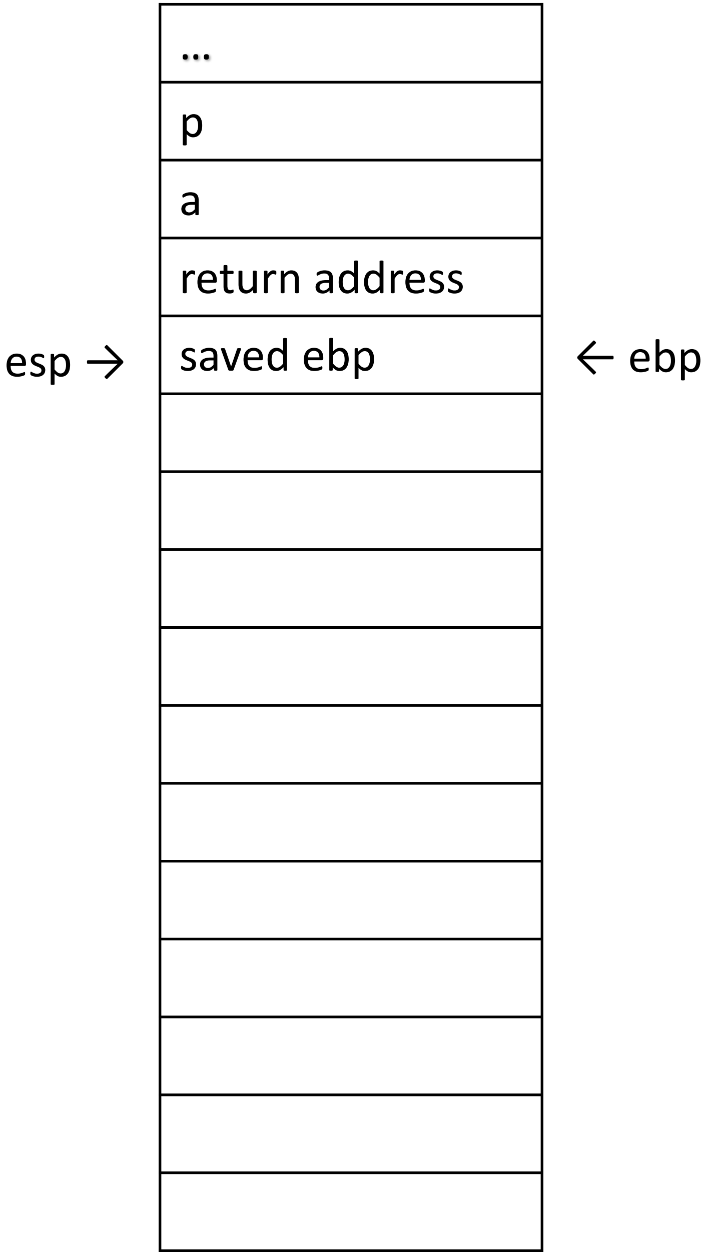


Function call example

```c
int foo(int a, char *p) {
        int b = atoi(p);
        return a + b;
}
```

eip →

```
foo:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        movl    12(%ebp), %eax
        movl    %eax, (%esp)
        call    atoi
        movl    %eax, -12(%ebp)
        movl    -12(%ebp), %eax
        movl    8(%ebp), %edx
        addl    %edx, %eax
        leave
        ret
```



…

p

a

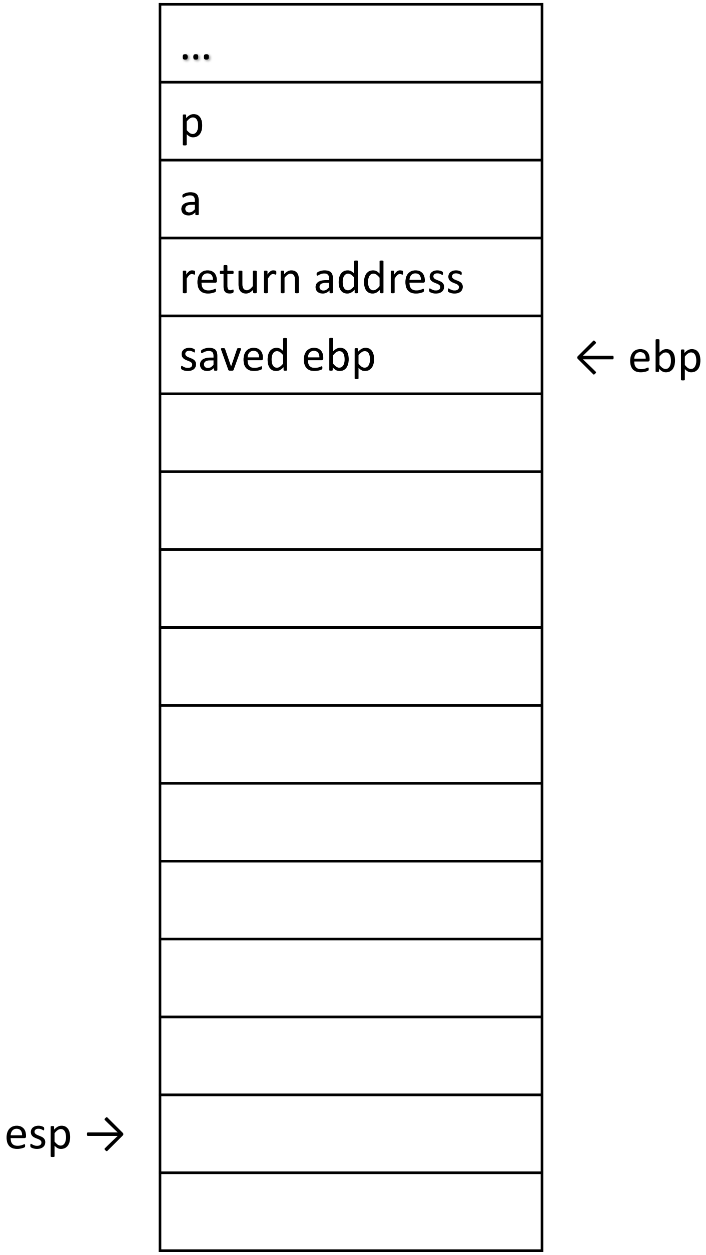
return address

esp → saved ebp ← ebp

Function call example

```
int foo(int a, char *p) {
        int b = atoi(p);
        return a + b;
}
```

```
      foo:
              pushl   %ebp
              movl    %esp, %ebp
              subl    $40, %esp
eip →         movl    12(%ebp), %eax
              movl    %eax, (%esp)
              call    atoi
              movl    %eax, -12(%ebp)
              movl    -12(%ebp), %eax
              movl    8(%ebp), %edx
              addl    %edx, %eax
              leave
              ret
```



…

p

a

return address

saved ebp ← ebp

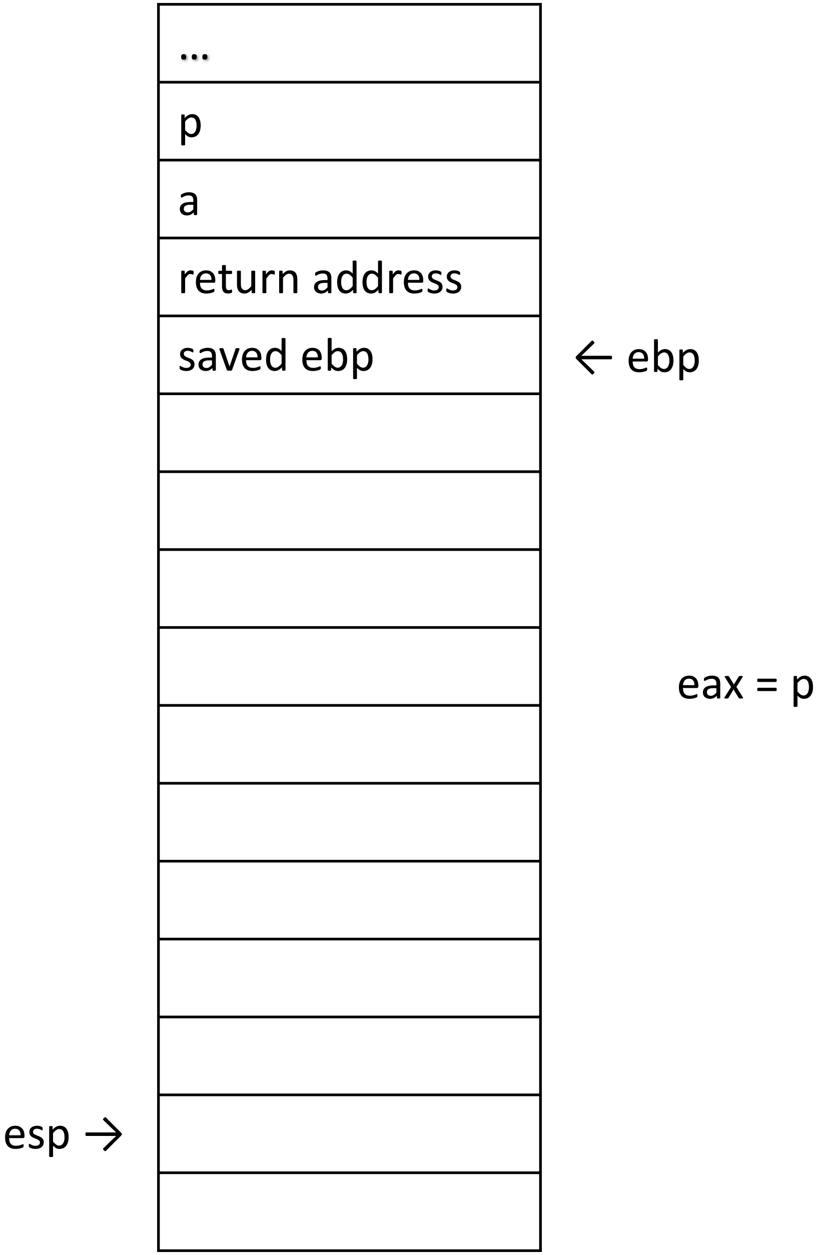
esp →

Function call example

```c
int foo(int a, char *p) {
        int b = atoi(p);
        return a + b;
}
```

```asm
foo:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        movl    12(%ebp), %eax
        movl    %eax, (%esp)
        call    atoi
        movl    %eax, -12(%ebp)
        movl    -12(%ebp), %eax
        movl    8(%ebp), %edx
        addl    %edx, %eax
        leave
        ret
```

eip →



…

p

a

return address

saved ebp

← ebp

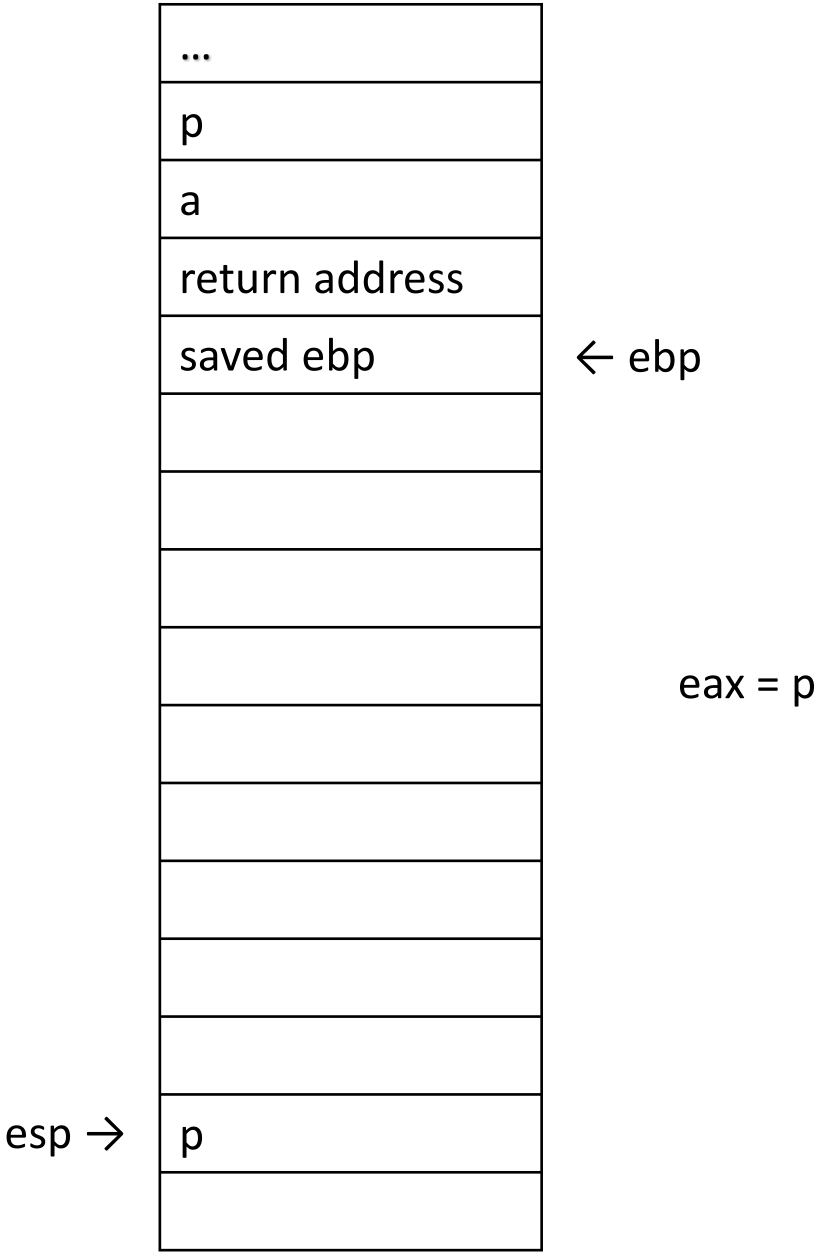
eax = p

esp →

Function call example

```c
int foo(int a, char *p) {
        int b = atoi(p);
        return a + b;
}
```

```
      foo:
              pushl   %ebp
              movl    %esp, %ebp
              subl    $40, %esp
              movl    12(%ebp), %eax
              movl    %eax, (%esp)
eip →         call    atoi
              movl    %eax, -12(%ebp)
              movl    -12(%ebp), %eax
              movl    8(%ebp), %edx
              addl    %edx, %eax
              leave
              ret
```

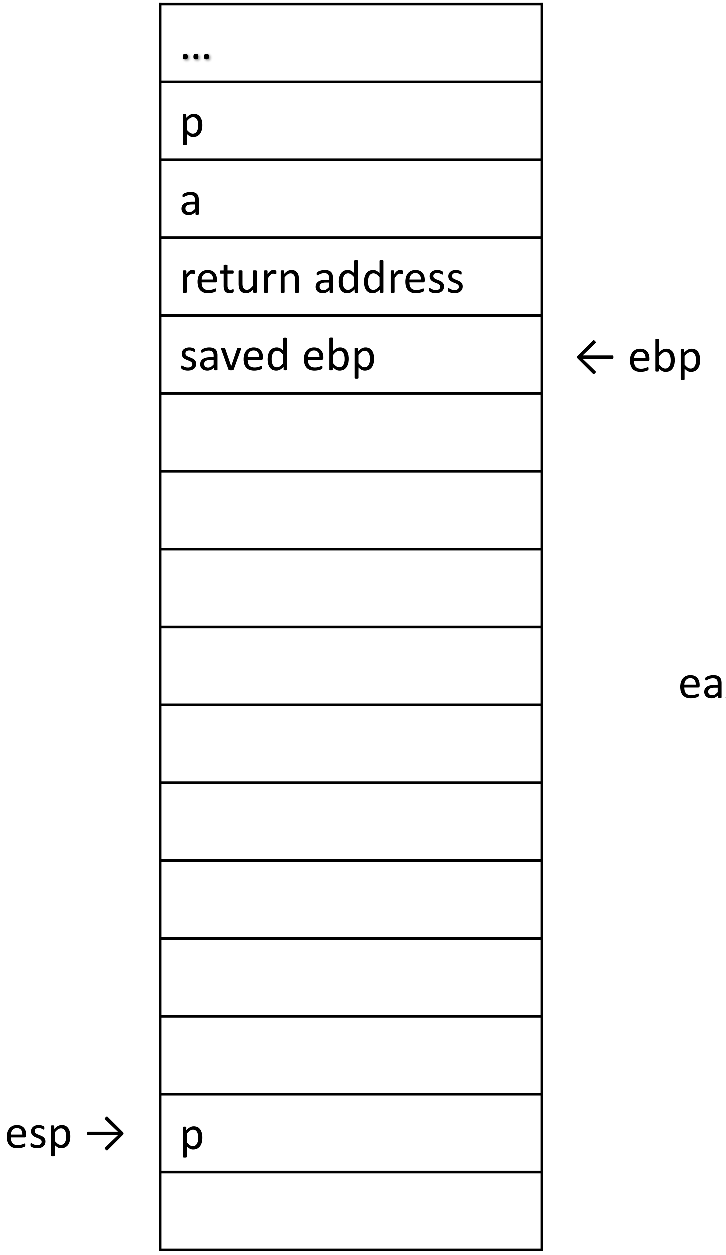


Function call example

```c
int foo(int a, char *p) {
        int b = atoi(p);
        return a + b;
}
```

```
foo:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        movl    12(%ebp), %eax
        movl    %eax, (%esp)
        call    atoi
        movl    %eax, -12(%ebp)
        movl    -12(%ebp), %eax
        movl    8(%ebp), %edx
        addl    %edx, %eax
        leave
        ret
```

eip → (line 8: movl %eax, -12(%ebp))



…

p

a

return address

saved ebp ← ebp

esp → p

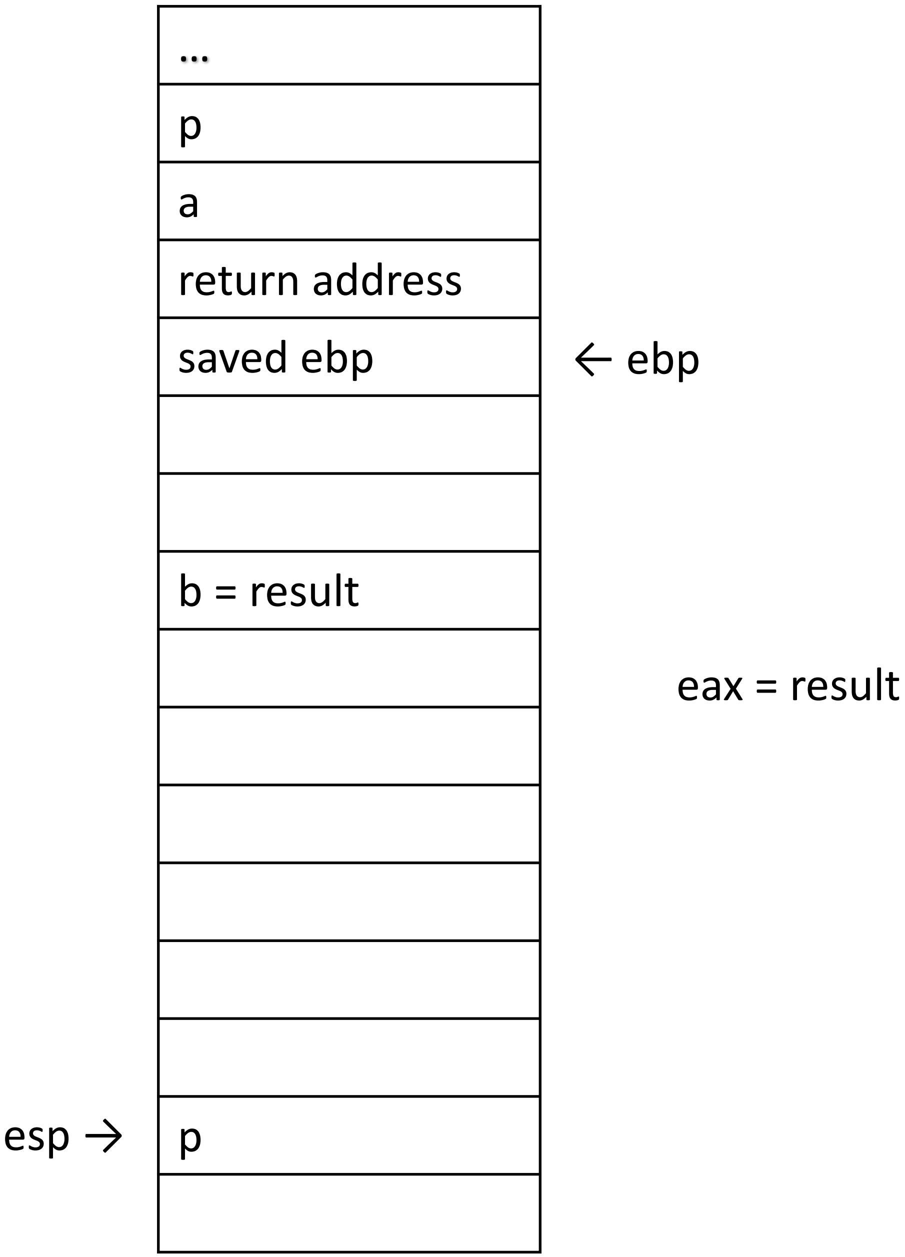
eax = result

Function call example

```c
int foo(int a, char *p) {
        int b = atoi(p);
        return a + b;
}
```

```
foo:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        movl    12(%ebp), %eax
        movl    %eax, (%esp)
        call    atoi
        movl    %eax, -12(%ebp)
        movl    -12(%ebp), %eax
        movl    8(%ebp), %edx
        addl    %edx, %eax
        leave
        ret
```

eip →



...

p

a

return address

saved ebp ← ebp

b = result

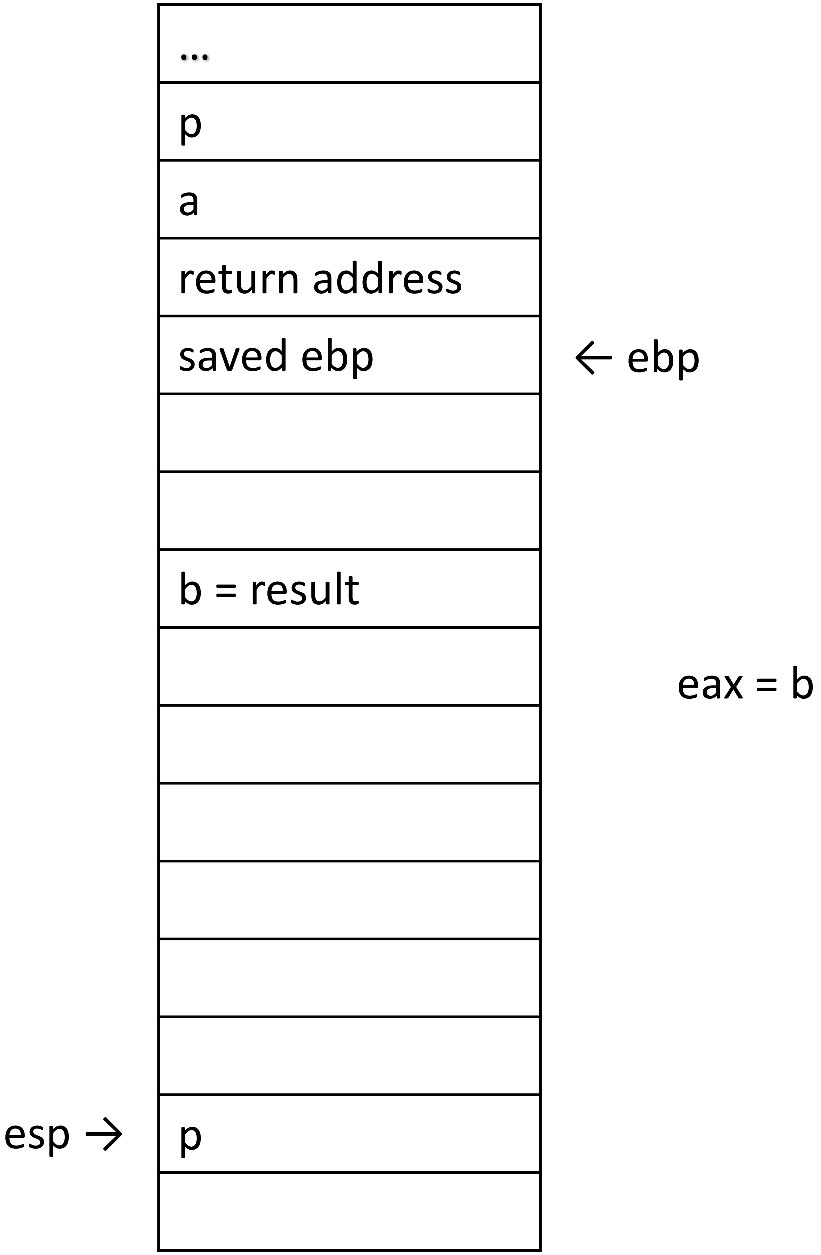
eax = result

esp → p

Function call example

```c
int foo(int a, char *p) {
        int b = atoi(p);
        return a + b;
}
```

```
foo:
              pushl   %ebp
              movl    %esp, %ebp
              subl    $40, %esp
              movl    12(%ebp), %eax
              movl    %eax, (%esp)
              call    atoi
              movl    %eax, -12(%ebp)
              movl    -12(%ebp), %eax
eip →         movl    8(%ebp), %edx
              addl    %edx, %eax
              leave
              ret
```



...

p

a

return address

saved ebp ← ebp

b = result

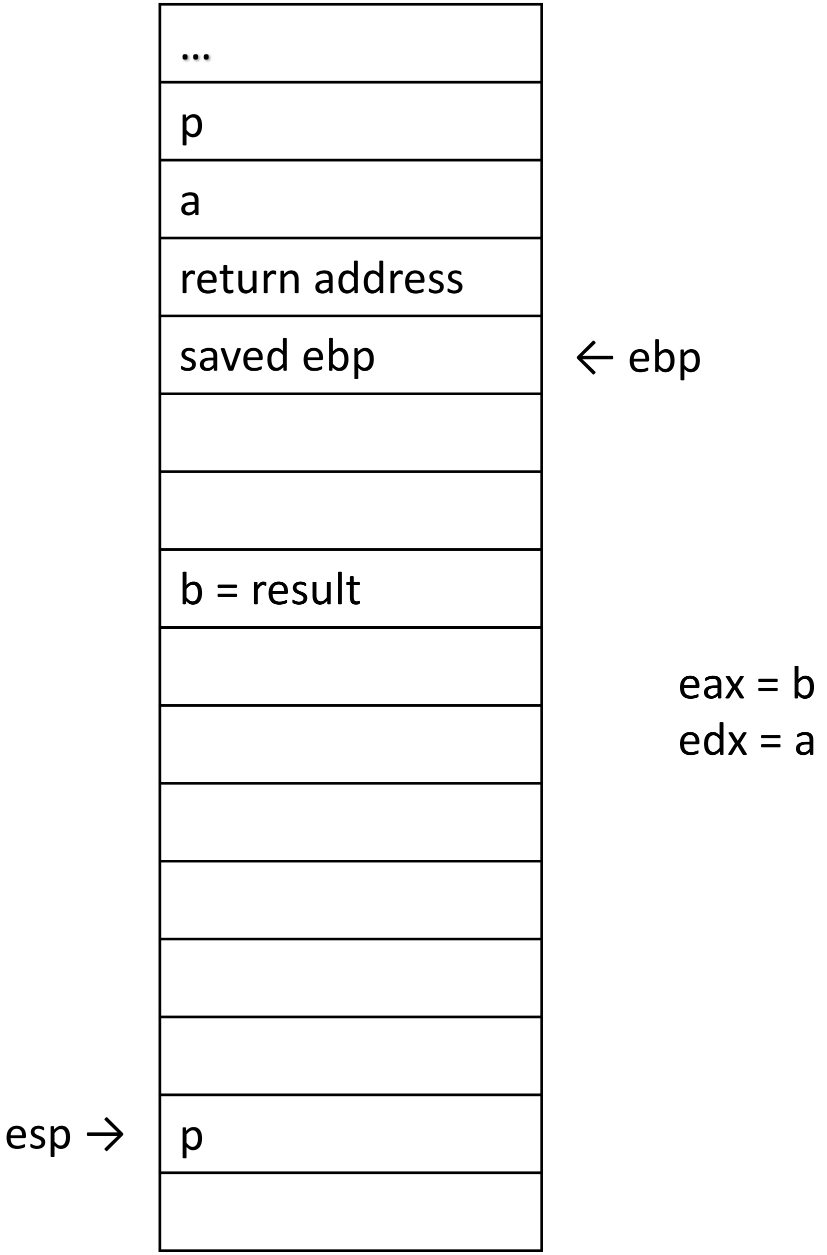
eax = b

esp → p

Function call example

```c
int foo(int a, char *p) {
        int b = atoi(p);
        return a + b;
}
```

```
      foo:
              pushl   %ebp
              movl    %esp, %ebp
              subl    $40, %esp
              movl    12(%ebp), %eax
              movl    %eax, (%esp)
              call    atoi
              movl    %eax, -12(%ebp)
              movl    -12(%ebp), %eax
              movl    8(%ebp), %edx
eip →         addl    %edx, %eax
              leave
              ret
```



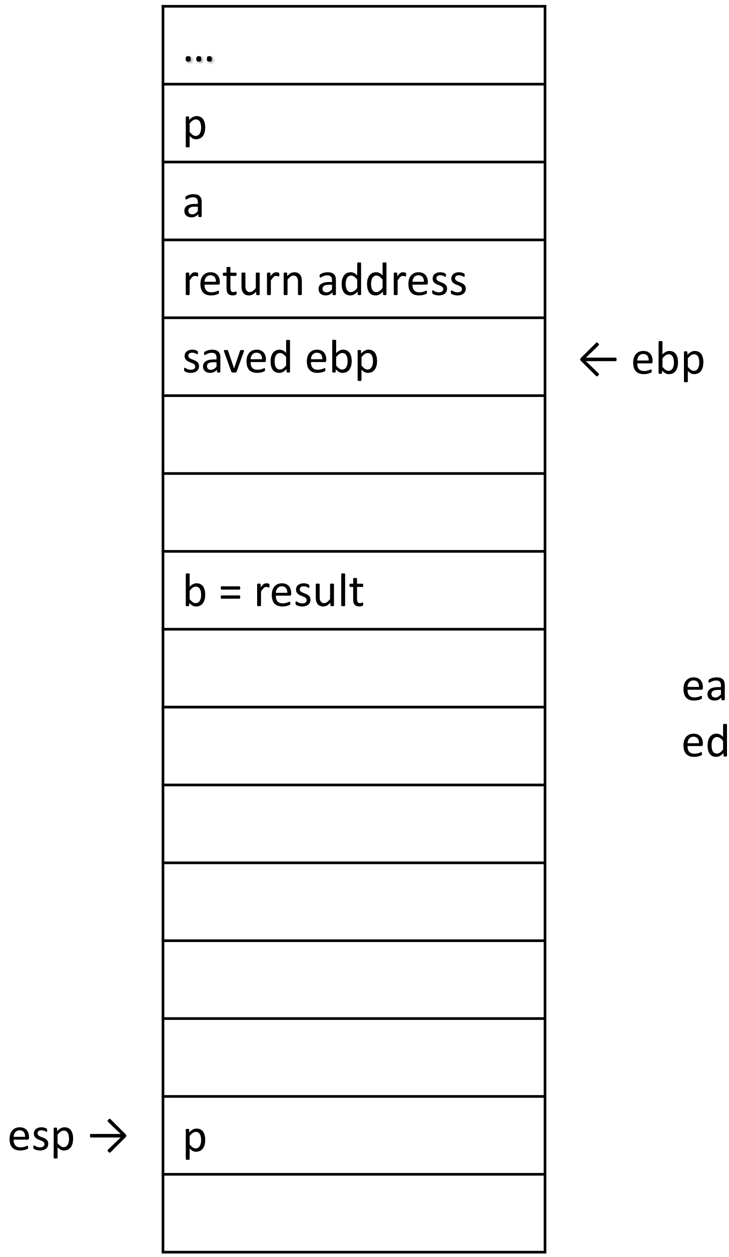
	…	
	p	
	a	
	return address	
	saved ebp	← ebp
	b = result	
esp →	p	

eax = b
edx = a

Function call example

```
1 int foo(int a, char *p) {
2         int b = atoi(p);
3         return a + b;
4 }
```

```
       1  foo:
       2          pushl   %ebp
       3          movl    %esp, %ebp
       4          subl    $40, %esp
       5          movl    12(%ebp), %eax
       6          movl    %eax, (%esp)
       7          call    atoi
       8          movl    %eax, -12(%ebp)
       9          movl    -12(%ebp), %eax
      10          movl    8(%ebp), %edx
      11          addl    %edx, %eax
eip → 12          leave
      13          ret
```



eax = b + a
edx = a

Function call example

```c
int foo(int a, char *p) {
        int b = atoi(p);
        return a + b;
}
```

```asm
foo:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        movl    12(%ebp), %eax
        movl    %eax, (%esp)
        call    atoi
        movl    %eax, -12(%ebp)
        movl    -12(%ebp), %eax
        movl    8(%ebp), %edx
        addl    %edx, %eax
        leave
        ret
```

eip →

← ebp

	…
	p
	a
esp →	return address
	saved ebp
	b = result
	p

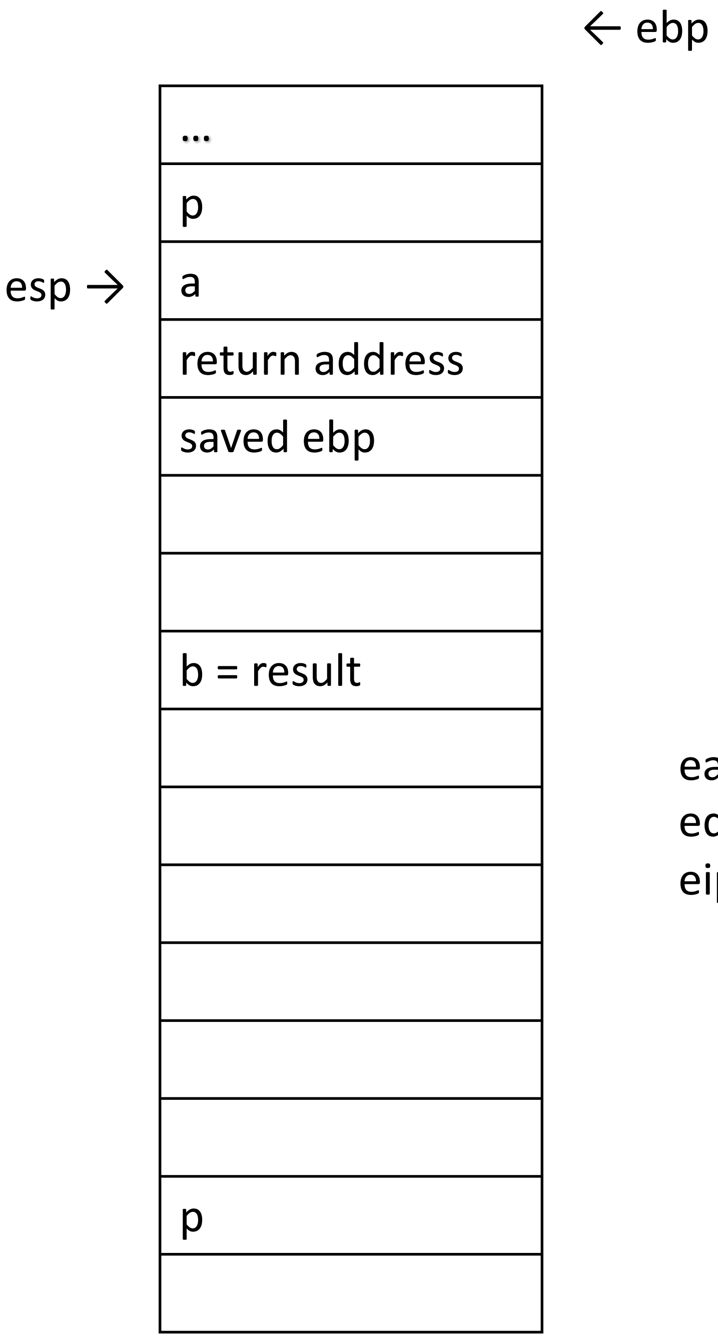
eax = b + a

edx = a

Function call example

```c
int foo(int a, char *p) {
        int b = atoi(p);
        return a + b;
}
```

```asm
foo:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $40, %esp
        movl    12(%ebp), %eax
        movl    %eax, (%esp)
        call    atoi
        movl    %eax, -12(%ebp)
        movl    -12(%ebp), %eax
        movl    8(%ebp), %edx
        addl    %edx, %eax
        leave
        ret
```



eax = b + a
edx = a
eip = ret addr