# Process Layout and Function Calls

CS 161 – Spring 2013

Prof. Vern Paxson

TAs: Jethro Beekman, Mobin Javed, Antonio Lupher, Paul Pearce, Matthias Vallentin

January 23, 2013

Zero Day
Ryan Naraine and Daniel Kennedy



# Google pays $14,000 for high-risk Chrome security holes

By Ryan Naraine | January 14, 2011, 9:52am PST

## Summary

*Google has shelled out more than $14,000 in rewards for critical and high-risk vulnerabilities affecting its flagship Chrome web browser.*

## Topics

Google Inc., Team, Adobe PDF, CERT,

Google has shelled out more than $14,000 in rewards for critical and high-risk vulnerabilities affecting its flagship Chrome web browser.

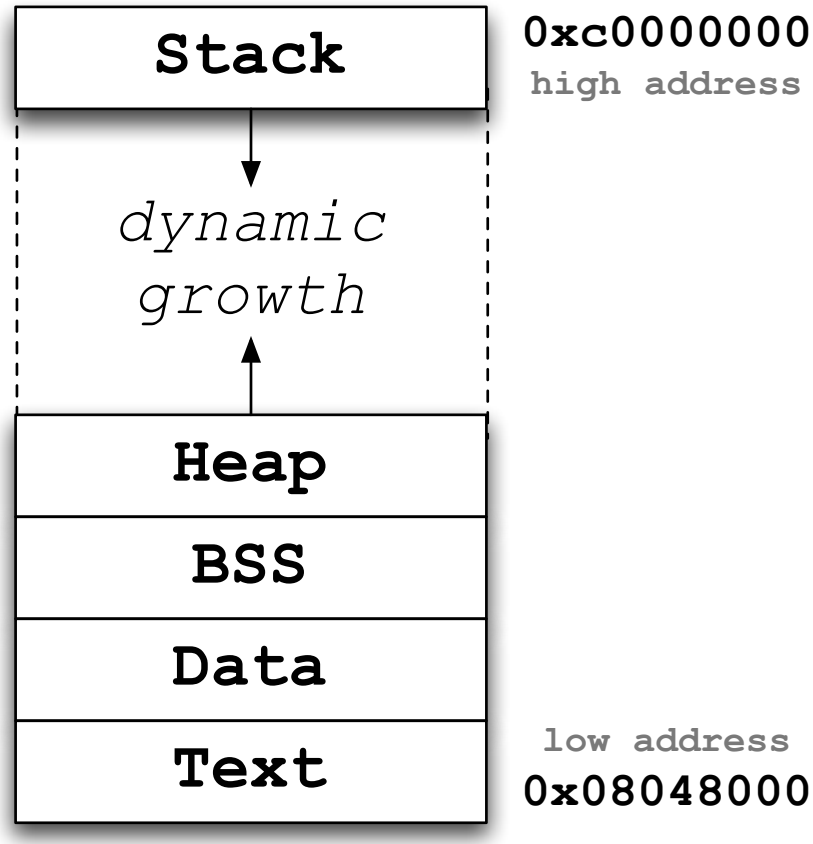
The latest Google Chrome 8.0.552.237, available for all platforms, patches a total of 16 documented vulnerabilties, including one critical bug for which Google paid the first elite $3133.7 award to researcher Sergey Glazunov.

"Critical bugs are harder to come by in Chrome, but Sergey has done it," says Google's Jerome Kersey. "Sergey also collects a $1337 reward and several other rewards at the same time, so congratulations Sergey!," he added.

# Outline

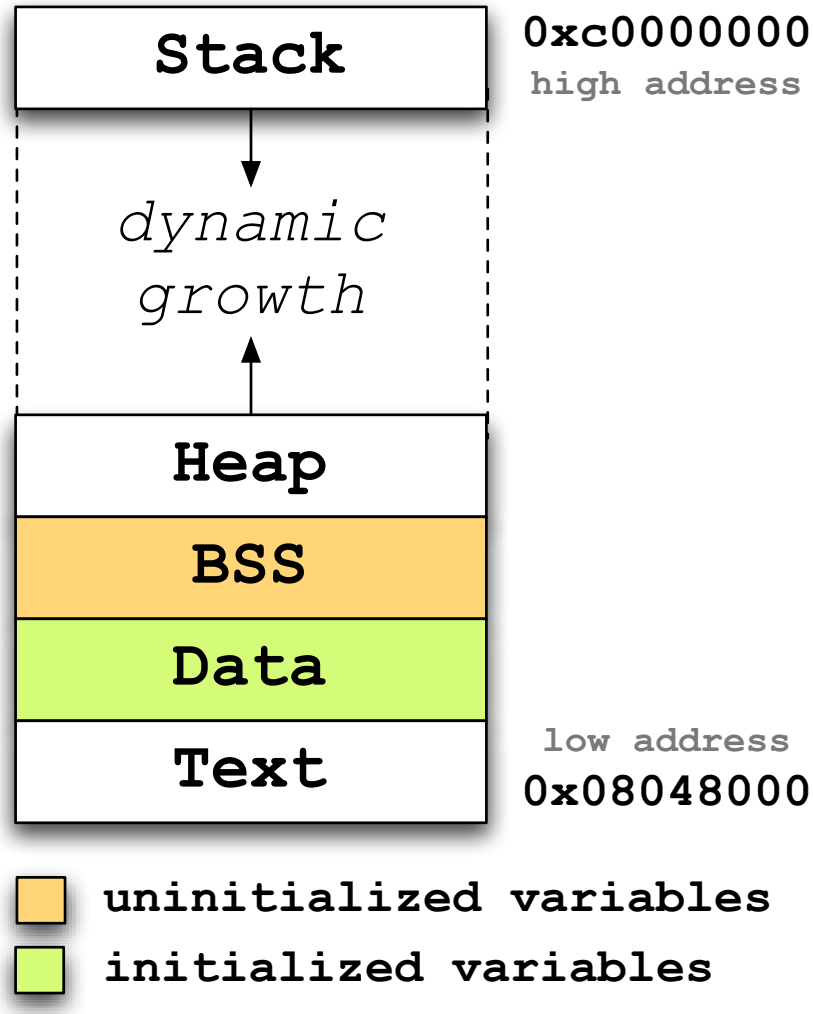
# Process Layout in Memory

- **Stack**
  - grows towards *decreasing* addresses.
  - is initialized at *run-time*.
- **Heap**
  - grow towards *increasing* addresses.
  - is initialized at *run-time*.
- **BSS** section
  - size fixed at *compile-time*.
  - is initialized at *run-time*.
  - was grouped into **Data** in CS61C.
- **Data** section
  - is initialized at *compile-time*.
- **Text** section
  - holds the program instructions (read-only).



# Process Layout in Memory

- **Stack**
  - grows towards *decreasing* addresses.
  - is initialized at *run-time*.
- **Heap**
  - grow towards *increasing* addresses.
  - is initialized at *run-time*.
- **BSS** section
  - size fixed at *compile-time*.
  - is initialized at *run-time*.
  - was grouped into **Data** in CS61C.
- **Data** section
  - is initialized at *compile-time*.
- **Text** section
  - holds the program instructions (read-only).



# Outline

# MIPS → IA-32: Differences

- **RISC vs CISC**
  - IA-32 has many more instructions
  - IA-32 instructions are variable length
  - IA-32 instructions can have implicit arguments and side effects
- **Limited Number of Registers**
  - MIPS has 18 general purpose registers ($s0-$s7, $t0-$t9)
  - IA-32 has 6 (%eax, %edx, %ecx, %ebx, %esi, %edi)
    - This means lots of stack operations!
- **Operand Directions**
  - MIPS: mov dst src
  - IA-32: mov src dst
- **Memory operations**
  - Very common to see push/pop/mov in IA-32
    - We'll see more of this later
- **The list goes on!**

# MIPS → IA-32

## Registers

| Use | MIPS | IA32 | Notes |
|---|---|---|---|
| Program Counter | PC | %eip | Can not be referenced directly |
| Stack Pointer | $sp | %esp | |
| Frame Pointer | $fp | %ebp | |
| Return Address | $ra | - | RA kept on stack in IA-32 |
| Return Value (32 bit) | $v0 | %eax | %eax not used solely for RV |
| Argument Registers | $a0-$a3 | - | Passed on stack in IA-32 |
| Zero | $0 | - | Use immediate value on IA-32 |

## Register Terminology

**SFP** **saved frame pointer**: saved %ebp on the stack

**OFP** **old frame pointer**: old %ebp from the previous stack frame

**RIP** **return instruction pointer**: return address on the stack

# Outline

# Function Calls

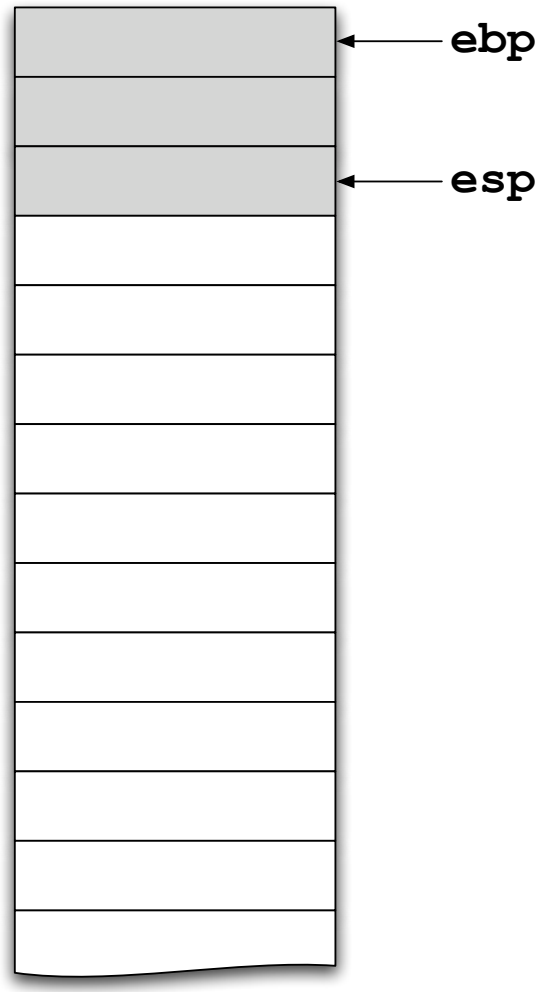
```
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}

int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

# Function Calls in Assembler

```
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

---

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```

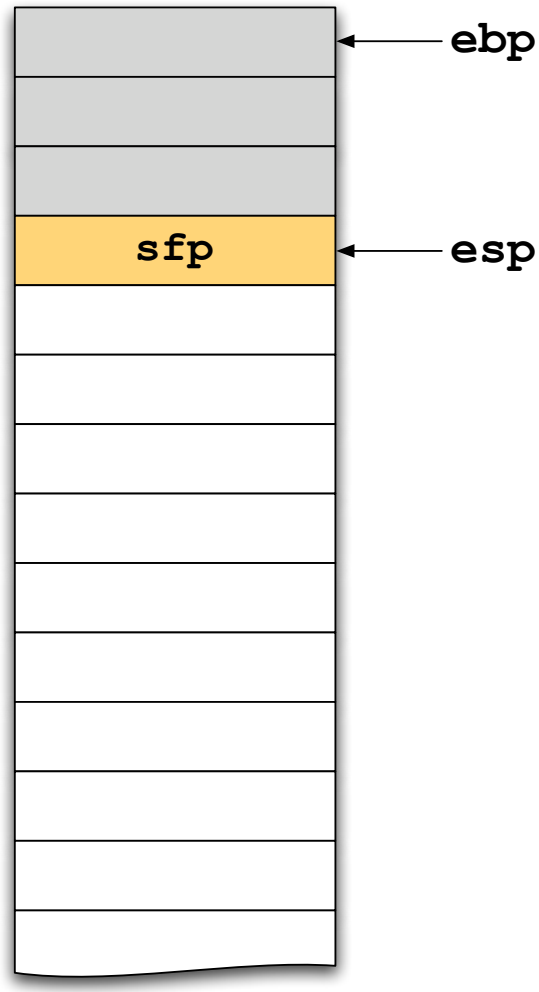


# Function Calls in Assembler

```
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

---

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```

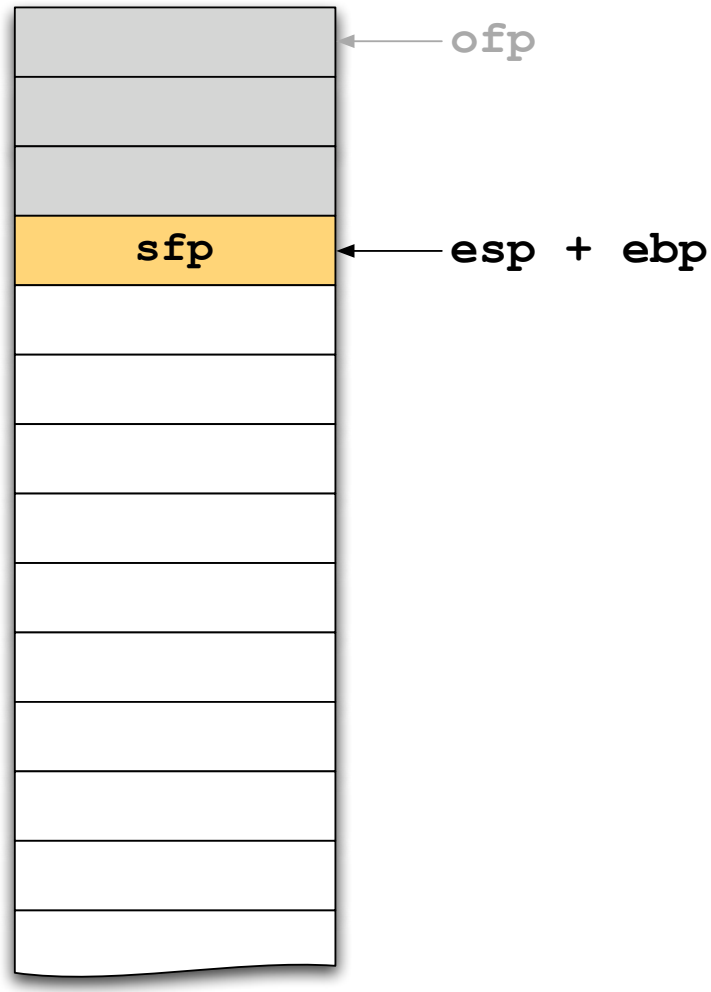


# Function Calls in Assembler

```
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

---

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```

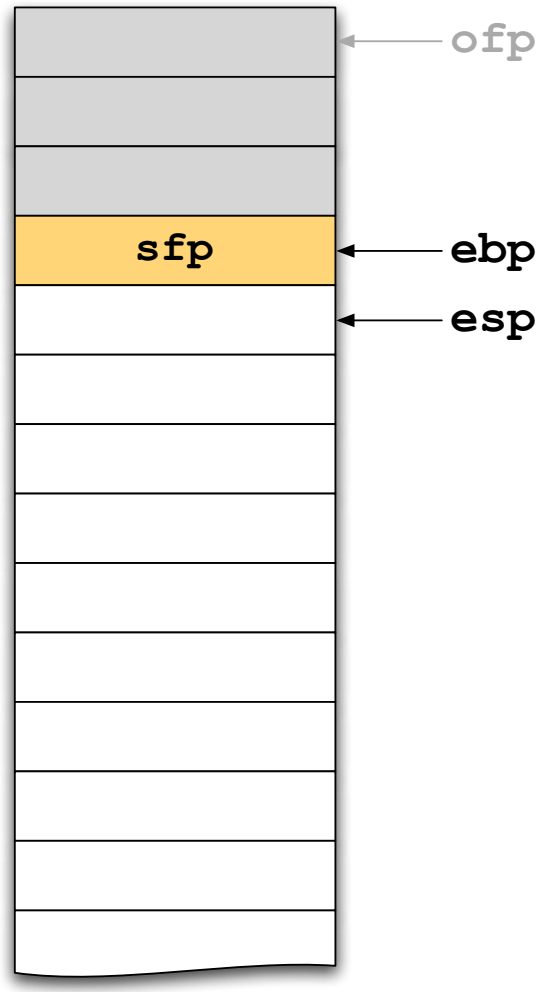


# Function Calls in Assembler

```
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

---

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```

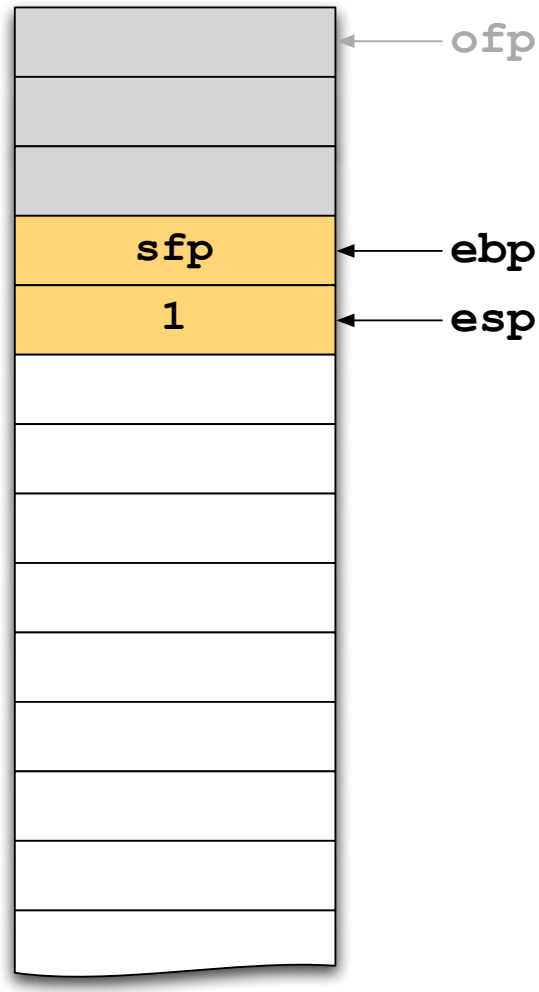


# Function Calls in Assembler

```
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

---

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```

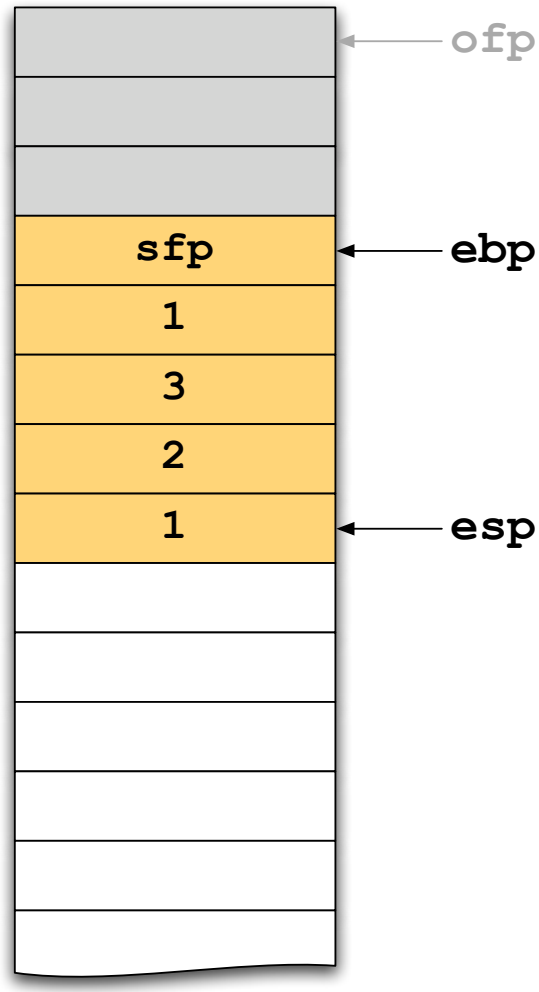


# Function Calls in Assembler

```
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

---

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```

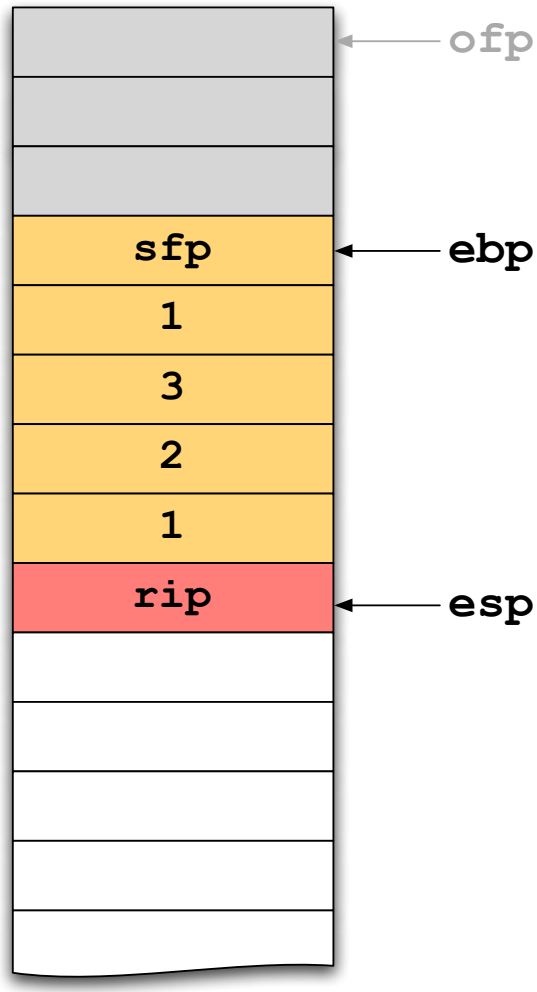


# Function Calls in Assembler

```
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

---

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```

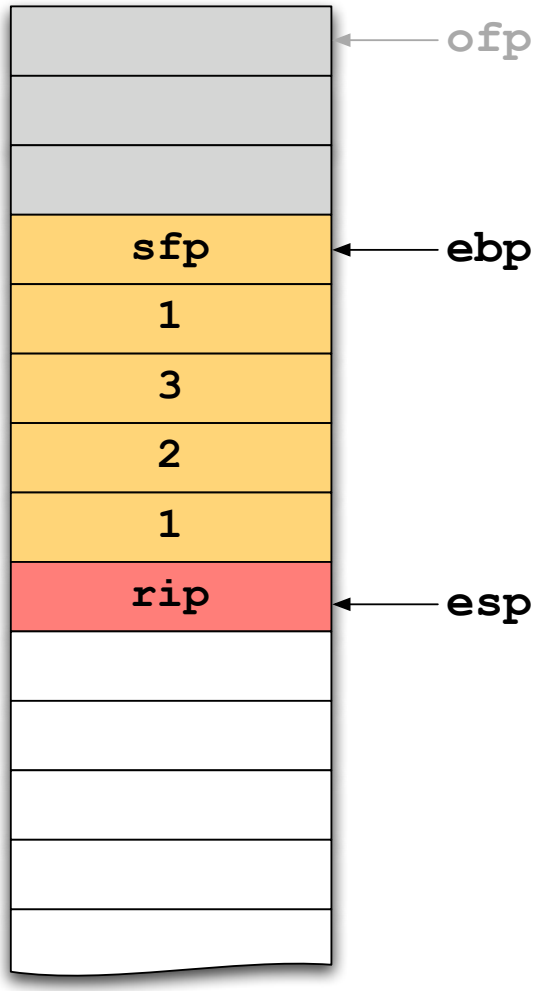


# Function Calls in Assembler

```
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}
```

---

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```

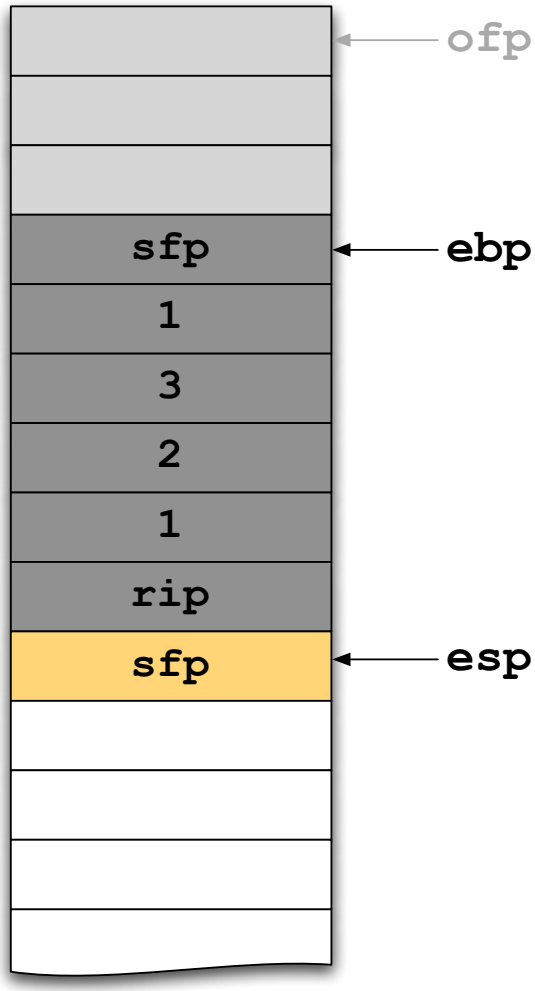


# Function Calls in Assembler

```
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}
```

---

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```

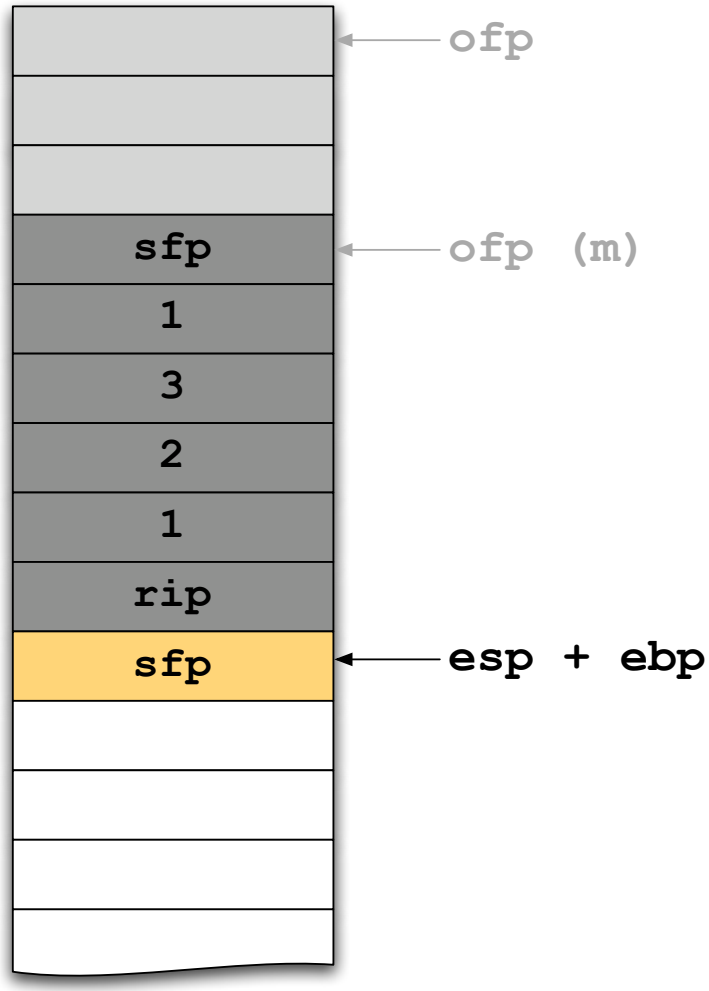


# Function Calls in Assembler

```
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}
```

---

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```

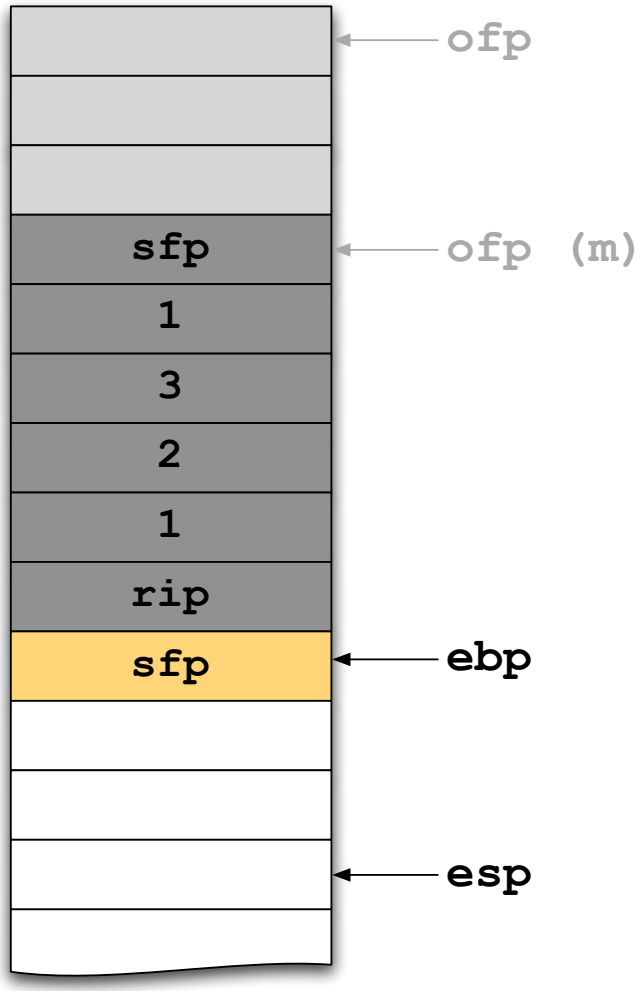


# Function Calls in Assembler

```
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}
```

---

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```

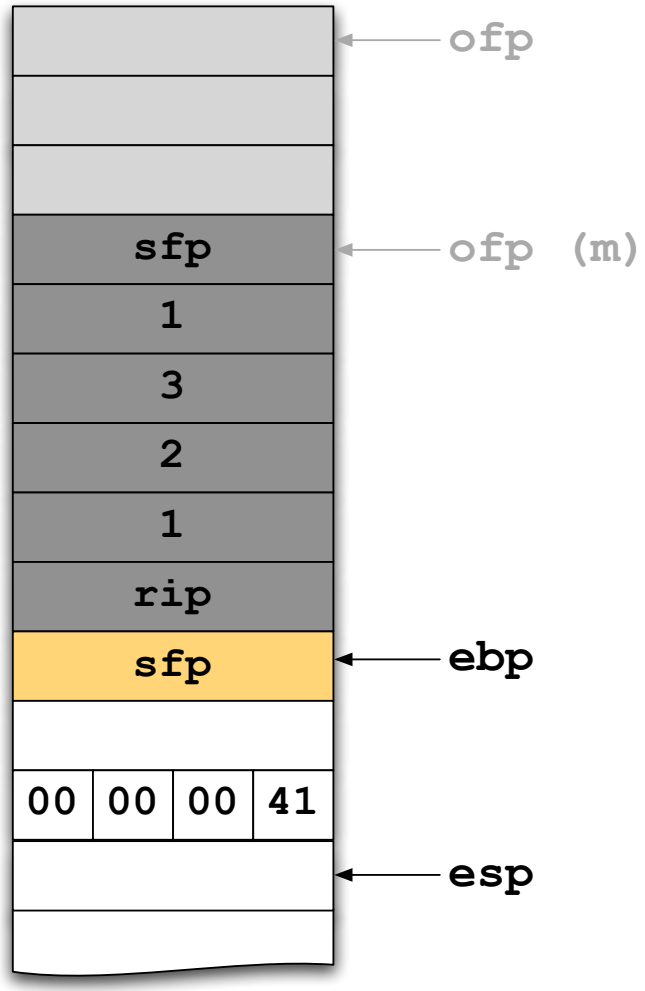


# Function Calls in Assembler

```
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}
```

---

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```

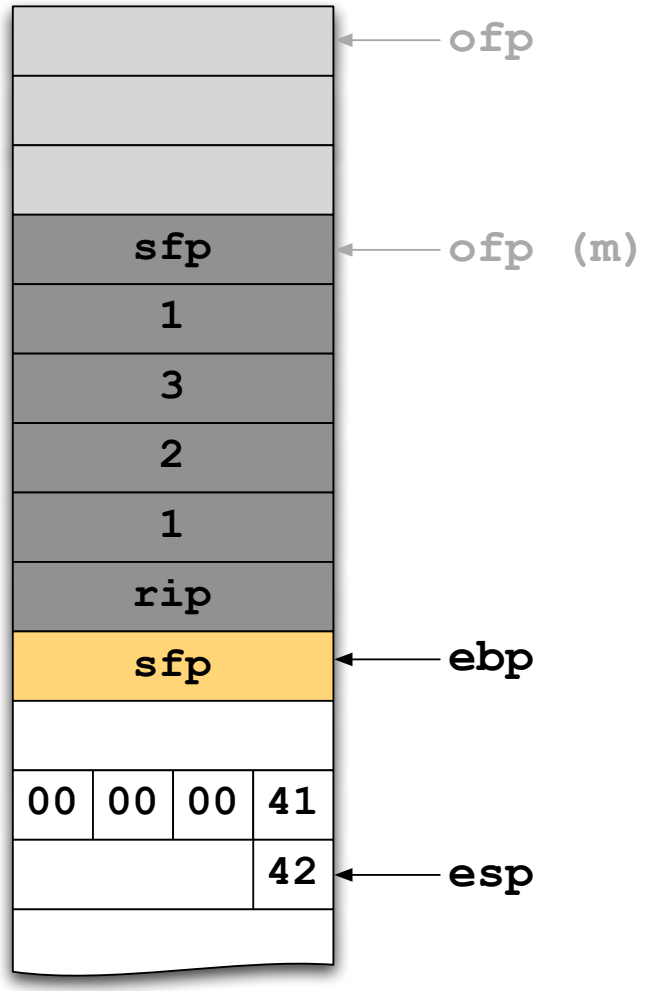


# Function Calls in Assembler

```
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}
```

---

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```

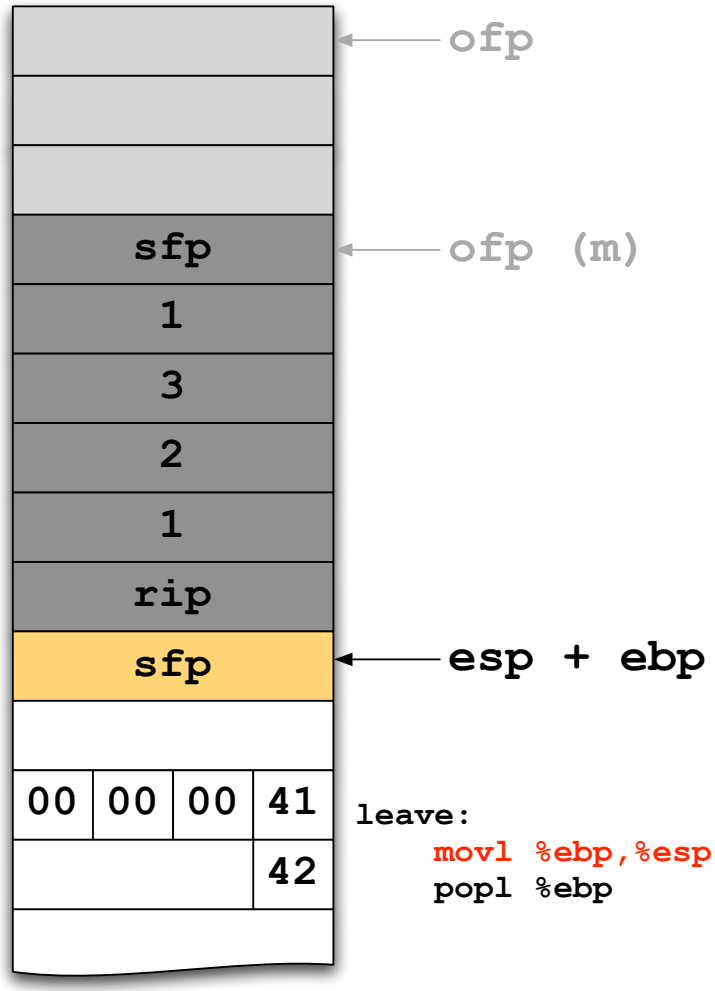


# Function Calls in Assembler

```
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}
```

---

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```

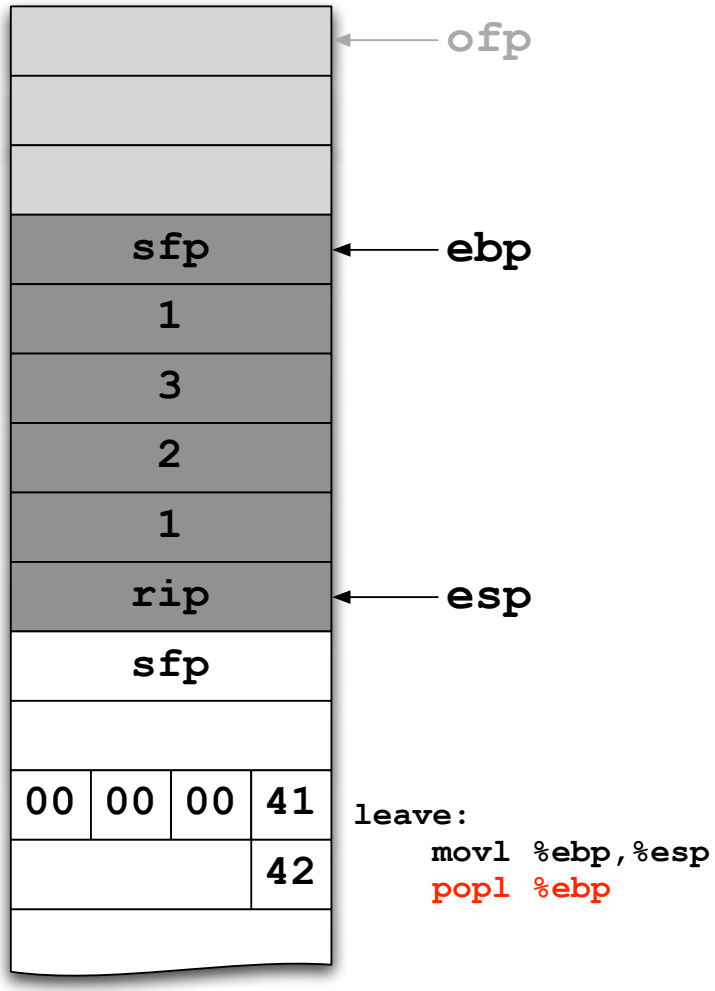


# Function Calls in Assembler

```
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}
```

---

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```

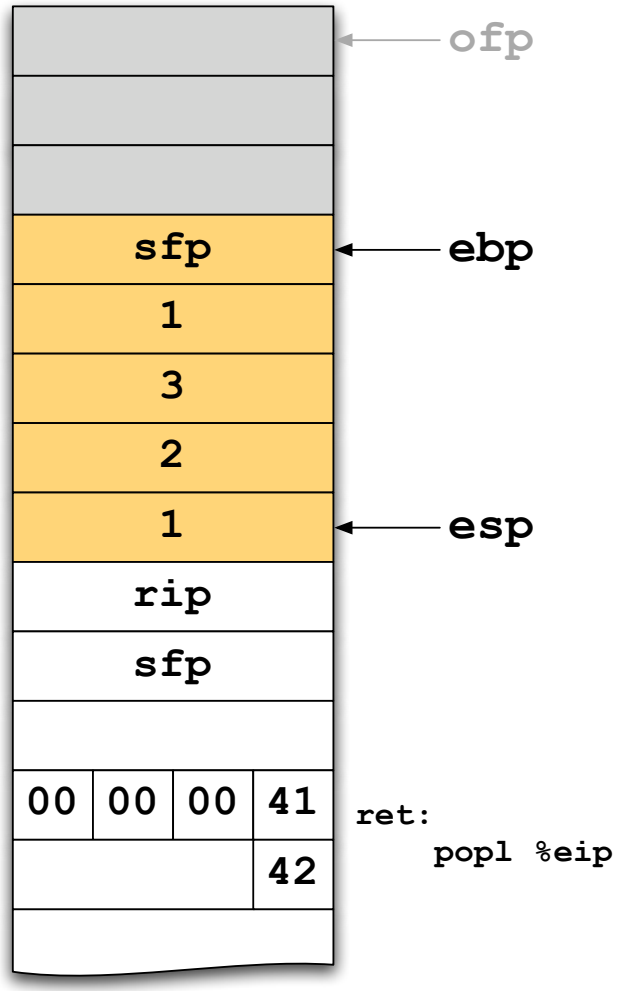


# Function Calls in Assembler

```
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}
```

---

```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```

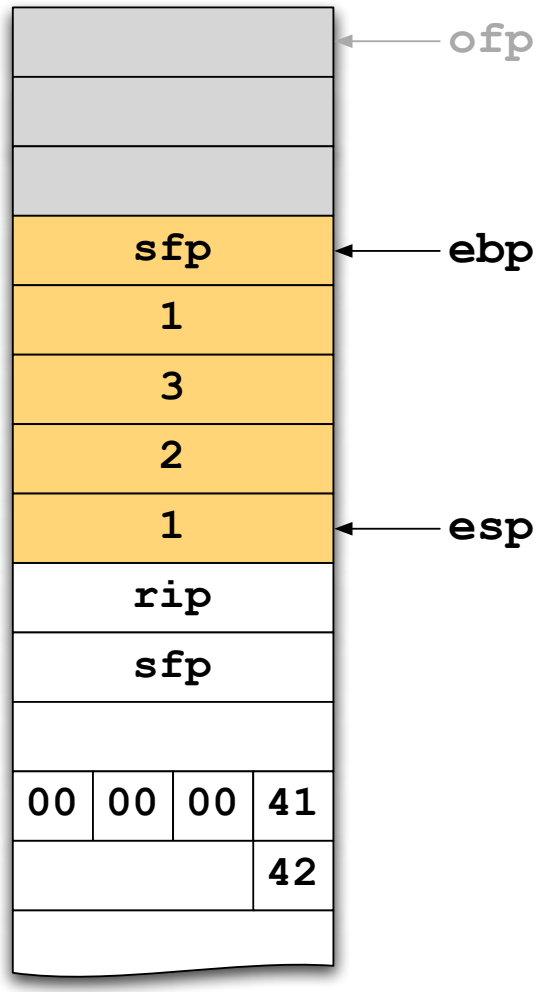


# Function Calls in Assembler

```
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

---

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```

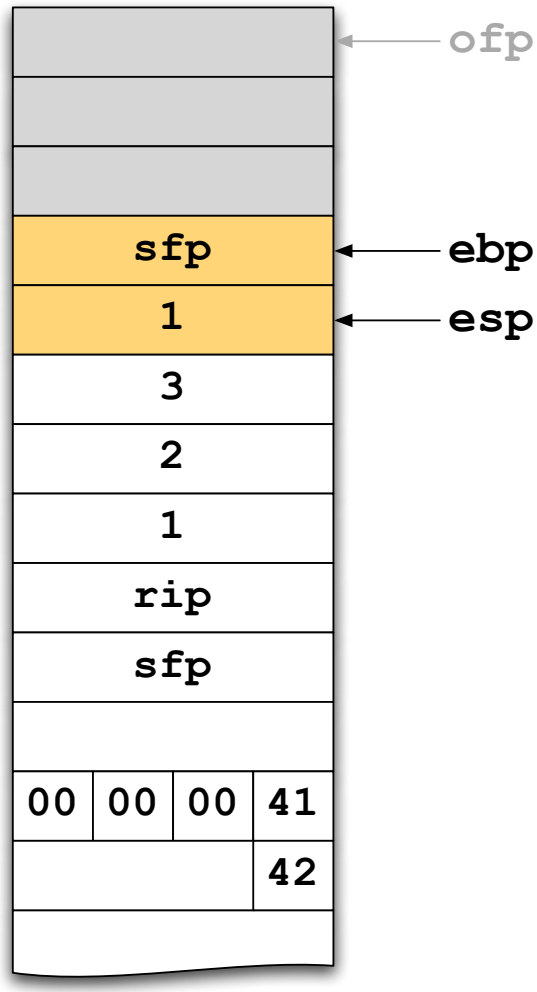


# Function Calls in Assembler

```
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

---

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```

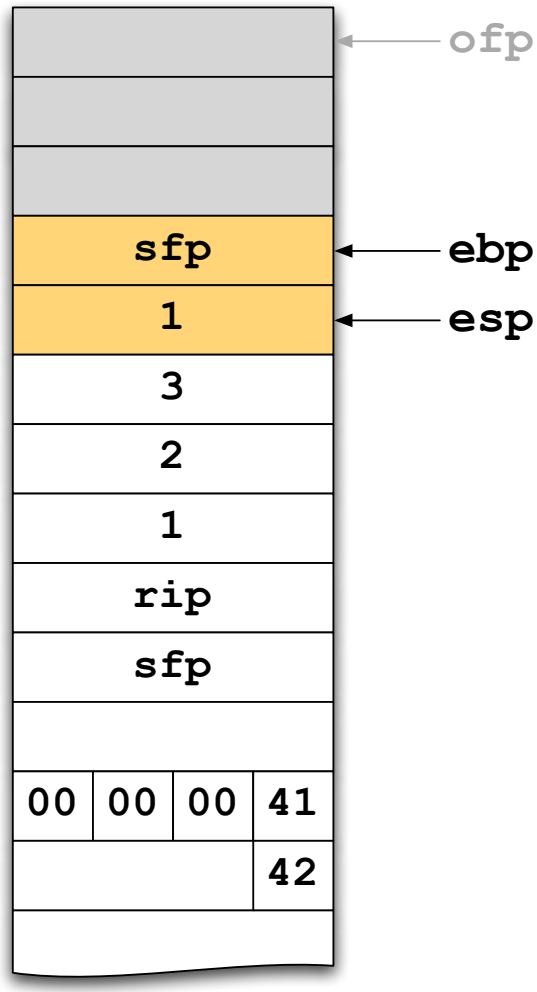


# Function Calls in Assembler

```
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

---

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```

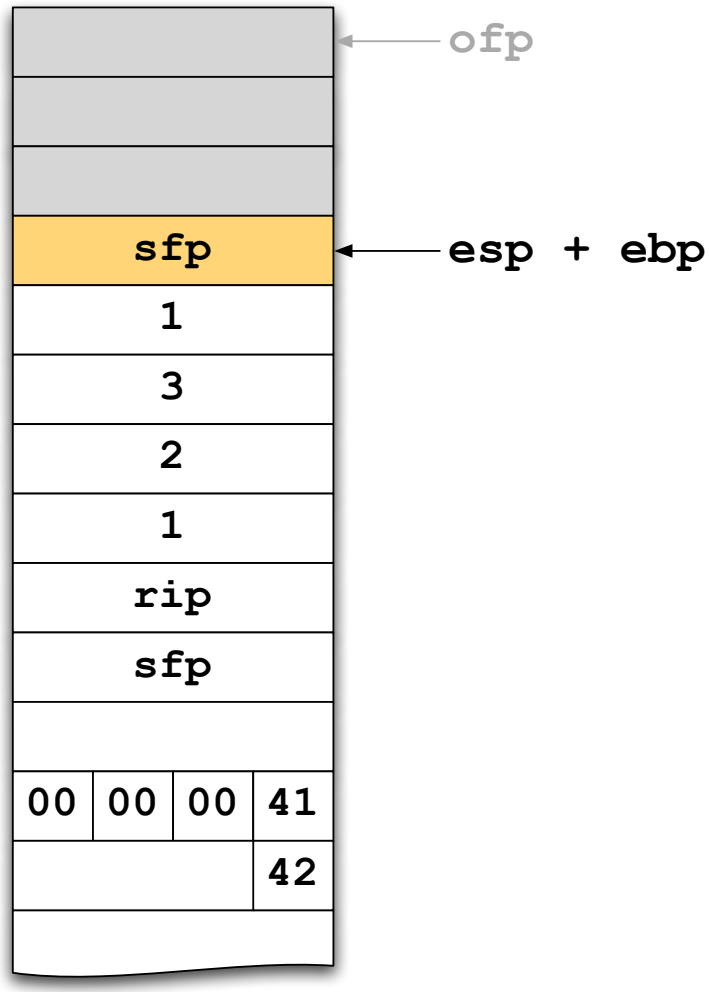


# Function Calls in Assembler

```
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

---

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```

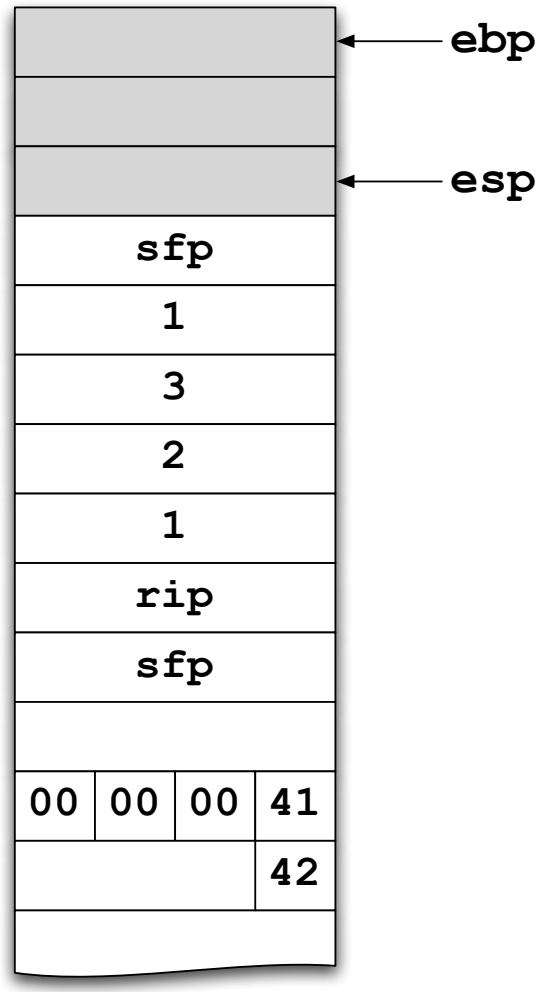


# Function Calls in Assembler

```
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

---

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```

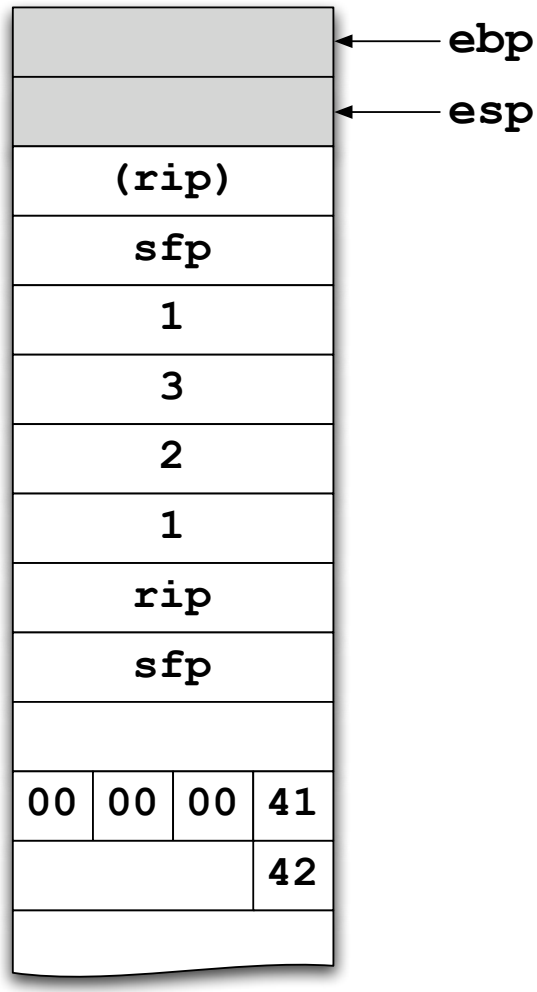


# Function Calls in Assembler

```
int main(void)
{
    int i = 1;
    foo(1, 2, 3);
    return 0;
}
```

---

```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



# IA-32 Reference

## IA32 Instructions

| | | |
|---|---|---|
| `movl` | *Src,Dest* | *Dest = Src* |
| `addl` | *Src,Dest* | *Dest = Dest + Src* |
| `subl` | *Src,Dest* | *Dest = Dest - Src* |
| `imull` | *Src,Dest* | *Dest = Dest * Src* |
| `sall` | *Src,Dest* | *Dest = Dest << Src* |
| `sarl` | *Src,Dest* | *Dest = Dest >> Src* |
| `shrl` | *Src,Dest* | *Dest = Dest >> Src* |
| `xorl` | *Src,Dest* | *Dest = Dest ^ Src* |
| `andl` | *Src,Dest* | *Dest = Dest & Src* |
| `orl` | *Src,Dest* | *Dest = Dest \| Src* |
| `incl` | *Dest* | *Dest = Dest + 1* |
| `decl` | *Dest* | *Dest = Dest - 1* |
| `negl` | *Dest* | *Dest = - Dest* |
| `notl` | *Dest* | *Dest = ~ Dest* |
| `leal` | *Src,Dest* | *Dest* = address of *Src* |
| `cmpl` | *Src2,Src1* | *Sets CCs Src1 – Src2* |
| `testl` | *Src2,Src1* | *Sets CCs Src1 & Src2* |
| `jmp` | *label* | *jump* |
| `je` | *label* | *jump equal* |
| `jne` | *label* | *jump not equal* |
| `js` | *label* | *jump negative* |
| `jns` | *label* | *jump non-negative* |
| `jg` | *label* | *jump greater (signed)* |
| `jge` | *label* | *jump greater or equal (signed)* |
| `jl` | *label* | *jump less (signed)* |
| `jle` | *label* | *jump less or equal (signed)* |
| `ja` | *label* | *jump above (unsigned)* |
| `jb` | *label* | *jump below (unsigned)* |

## Addressing Modes

**Immediate** $*val* *Val*

**Normal** (R) Mem[Reg[R]]

- Register R specifies memory address

```
movl (%ecx),%eax
```

**Displacement** D(R) Mem[Reg[R]+D]

- Register R specifies start of memory region
- Constant displacement D specifies offset

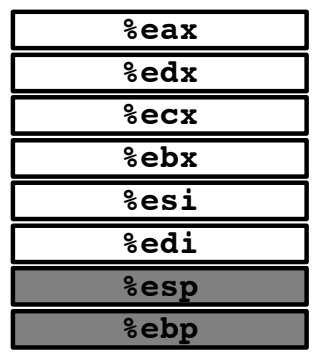
```
movl 8(%ebp),%edx
```

**Indexed** D(Rb,Ri,S) Mem[Reg[Rb]+S*Reg[Ri]+ D]

- D: Constant “displacement” 1, 2, or 4 bytes
- Rb: Base register: Any of 8 integer registers
- Ri: Index register:
- S: Scale: 1, 2, 4, or 8

## Condition Codes

| | |
|---|---|
| `CF` | Carry Flag |
| `ZF` | Zero Flag |
| `SF` | Sign Flag |
| `OF` | Overflow Flag |




Additional references will be posted on Piazza