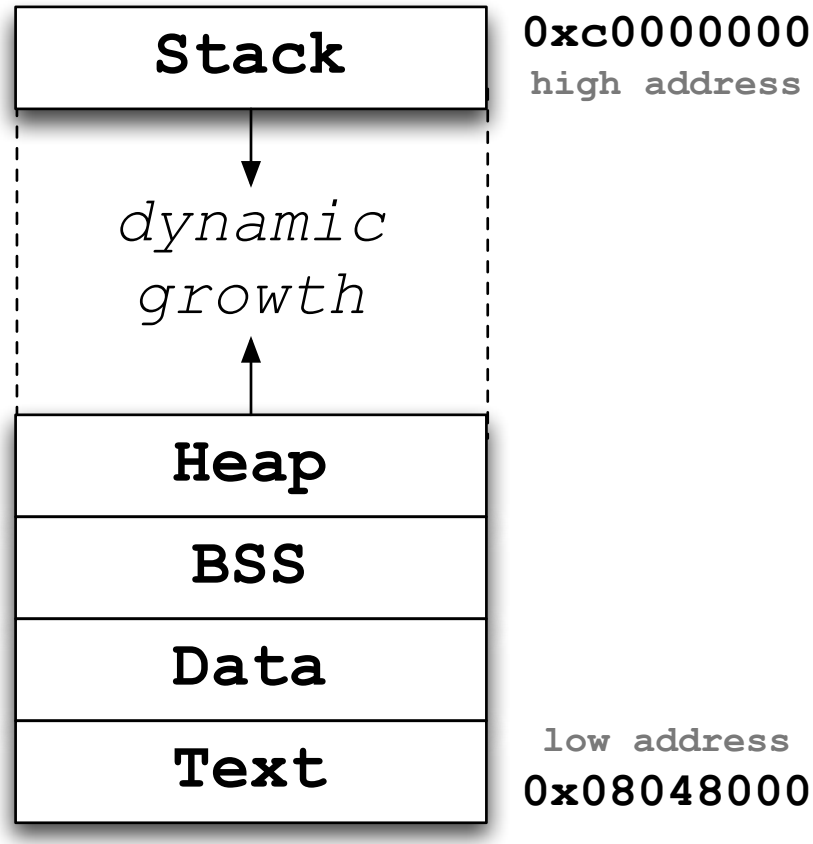
Process Layout and Function Calls

CS 161 – Spring 2017

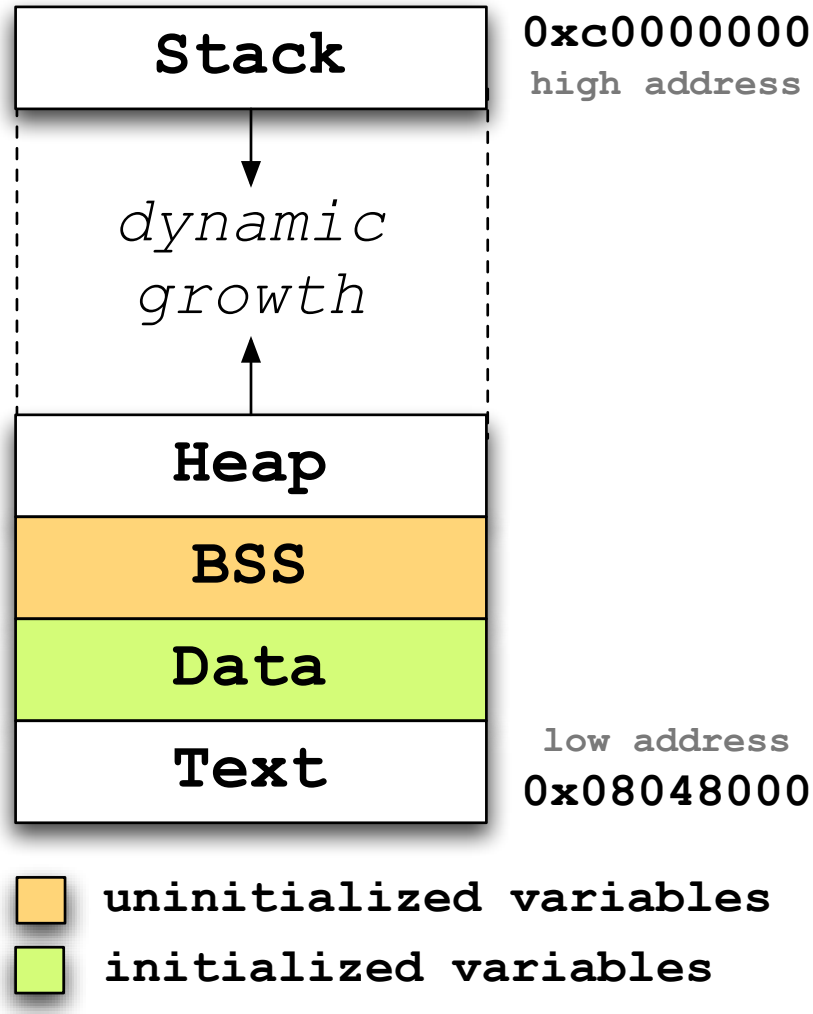
Process Layout in Memory

- **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).



IA-32 Caveats

Key Differences Between AT&T Syntax and Intel Syntax[1]

	AT&T	Intel
Parameter Order	src before dst `movl $4, %eax`	dst before src `mov eax, 5`
Parameter Size	Mnemonics suffixed with a letter indicating size of operands: q for qword, l for long (dword), w for word, and b for byte `addl $4, %esp`	Derived from name of register that is used (e.g. rax, eax, ax, al imply q, l, w, b, respectively) `add esp, 4`
Sigils	Immediate values prefixed with a `$`, registers prefixed with a `%`	Assembler automatically detects type of symbols; i.e., whether they are registers, constants or something else

[1] Adapted from: `https://en.wikipedia.org/wiki/X86_assembly_language#Syntax`

Function Calls

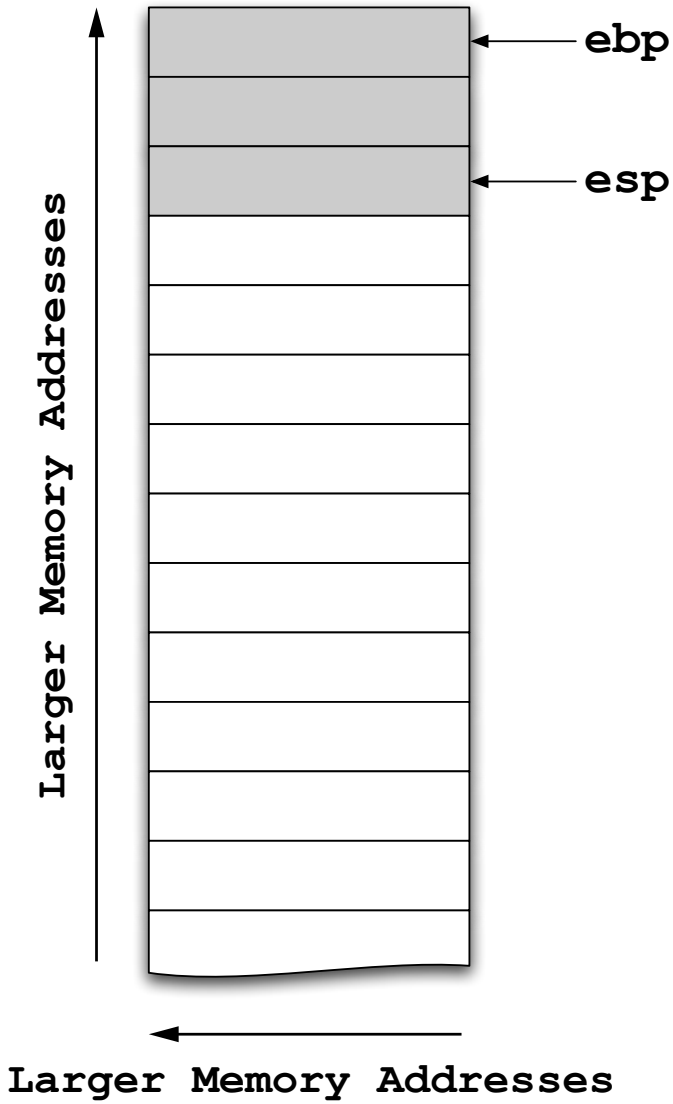
```
void foo(int a, int b, int c)
{
    int bar[2];
    char qux[3];
    bar[0] = 'A';
    qux[0] = 0x42;
}

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;
}
```

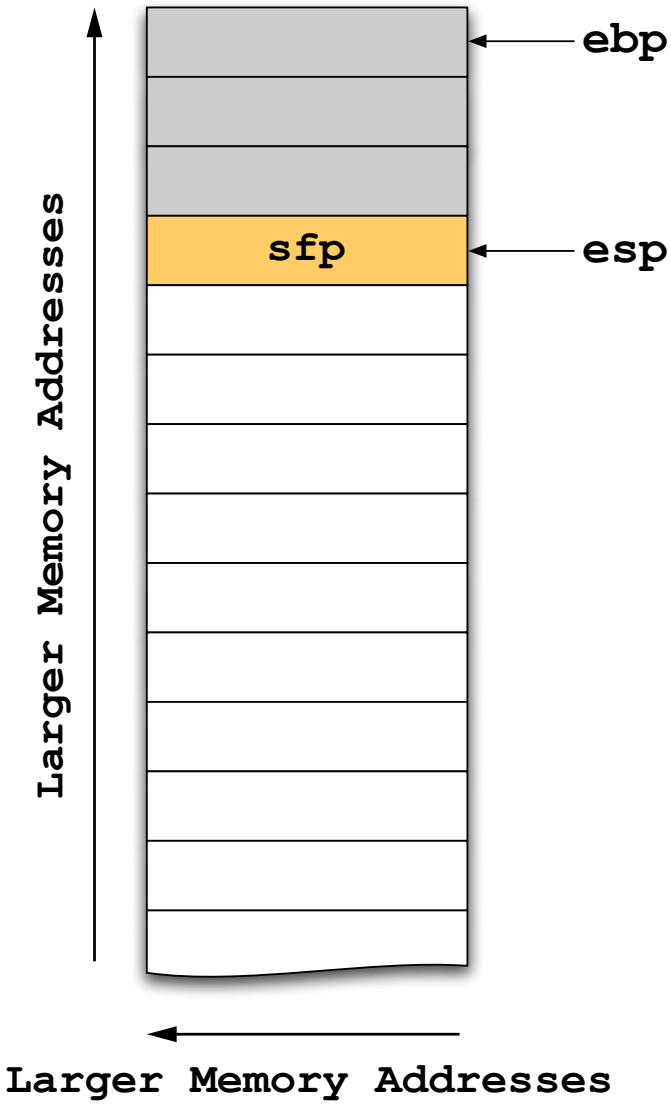
```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



Function Calls in Assembler

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

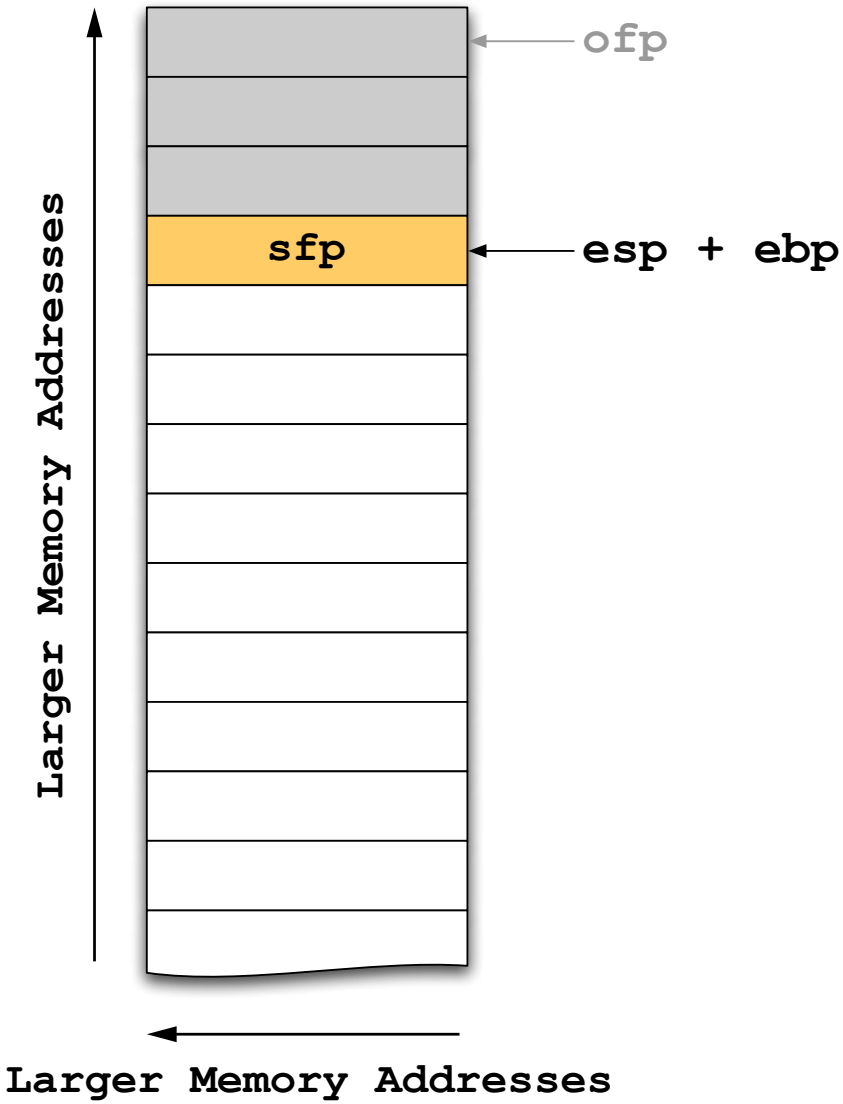
```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



Function Calls in Assembler

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

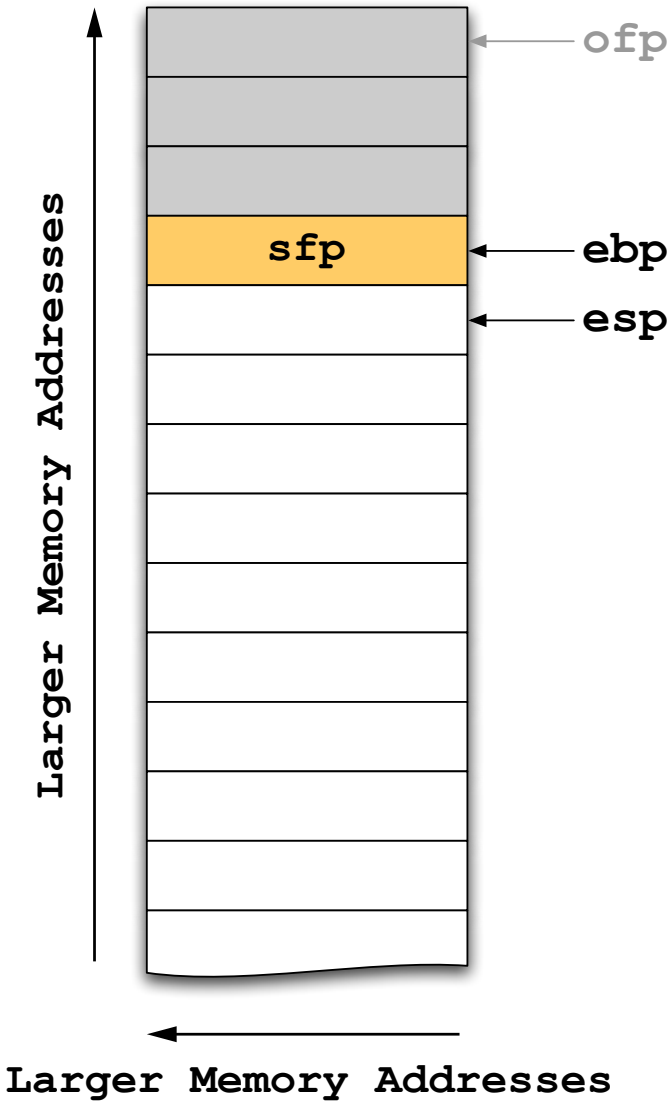
```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



Function Calls in Assembler

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

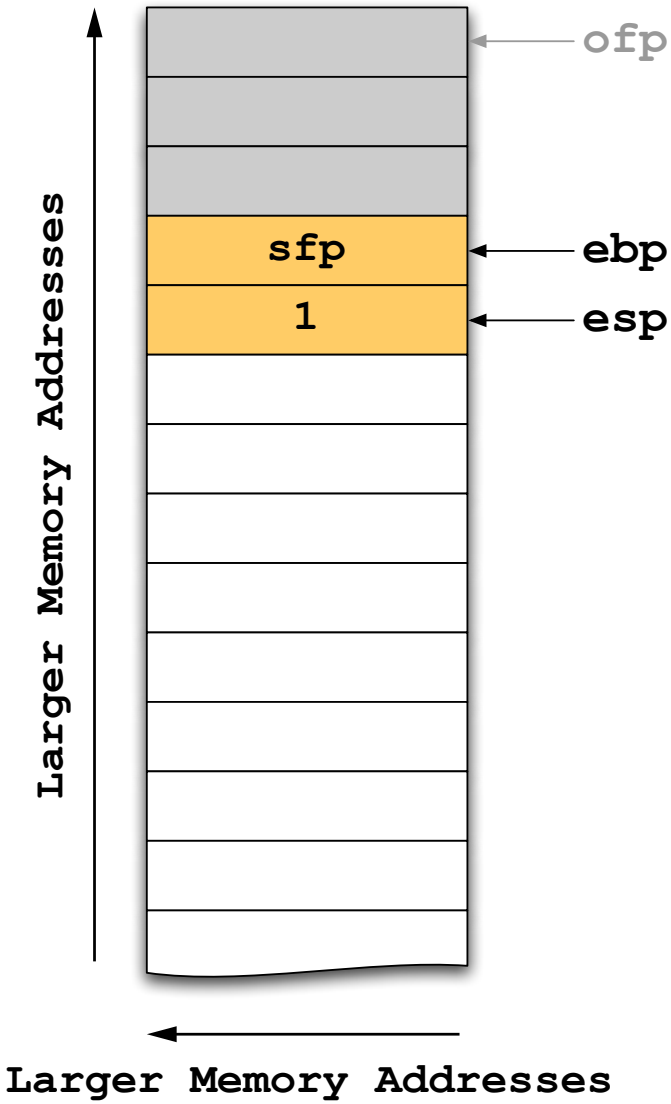
```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



Function Calls in Assembler

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

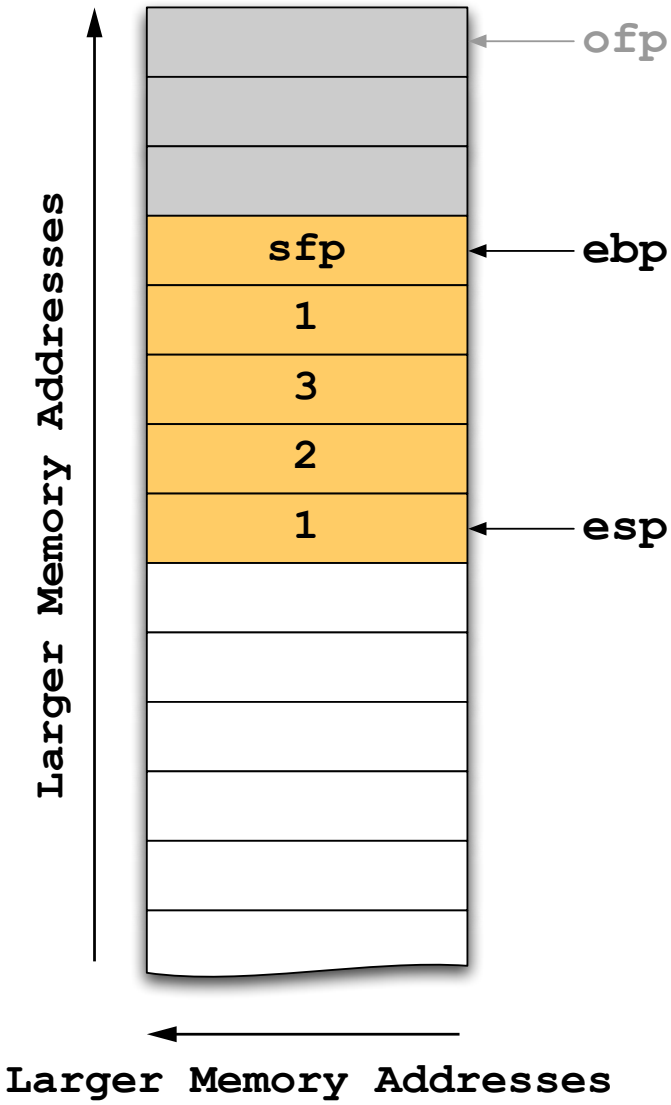
```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



Function Calls in Assembler

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

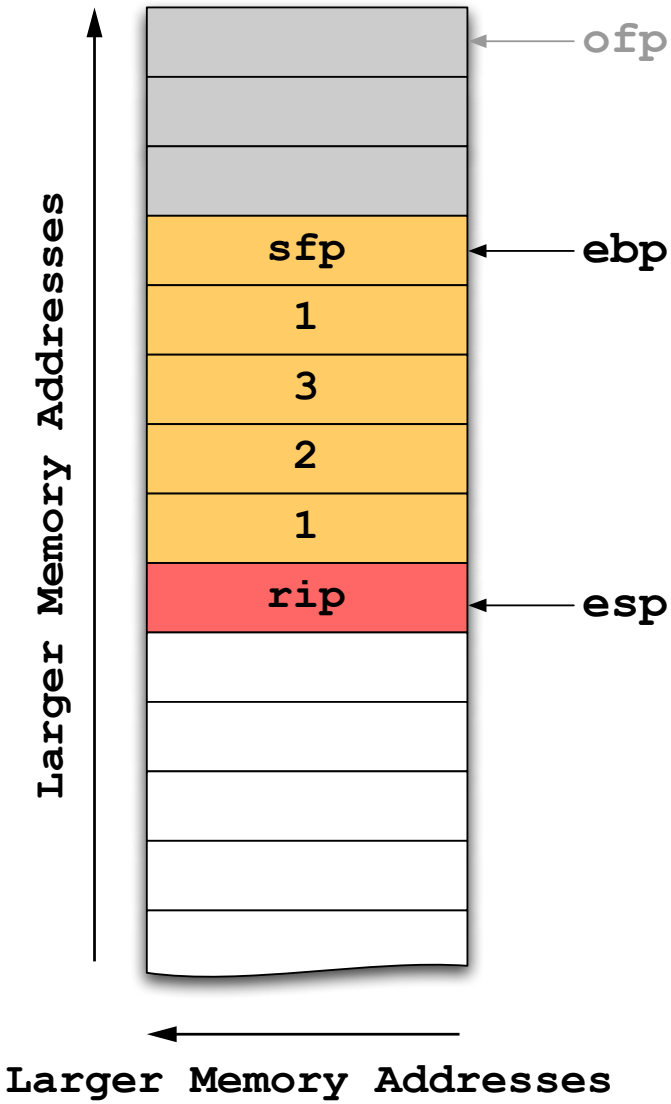
```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



Function Calls in Assembler

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

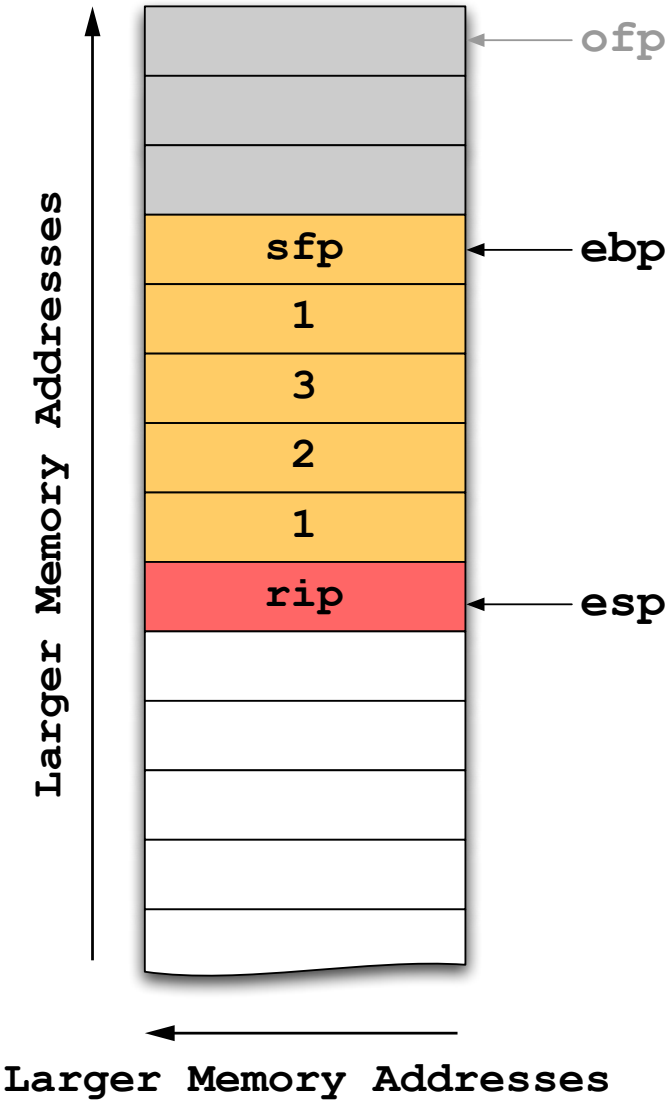
```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



Function Calls in Assembler

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

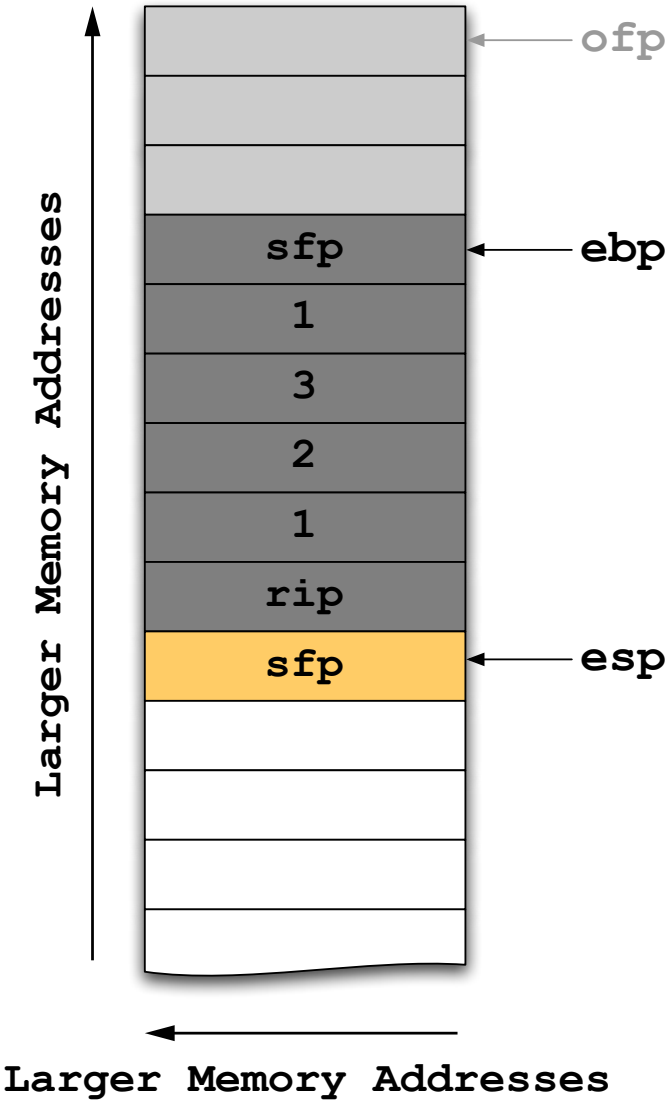
```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



Function Calls in Assembler

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

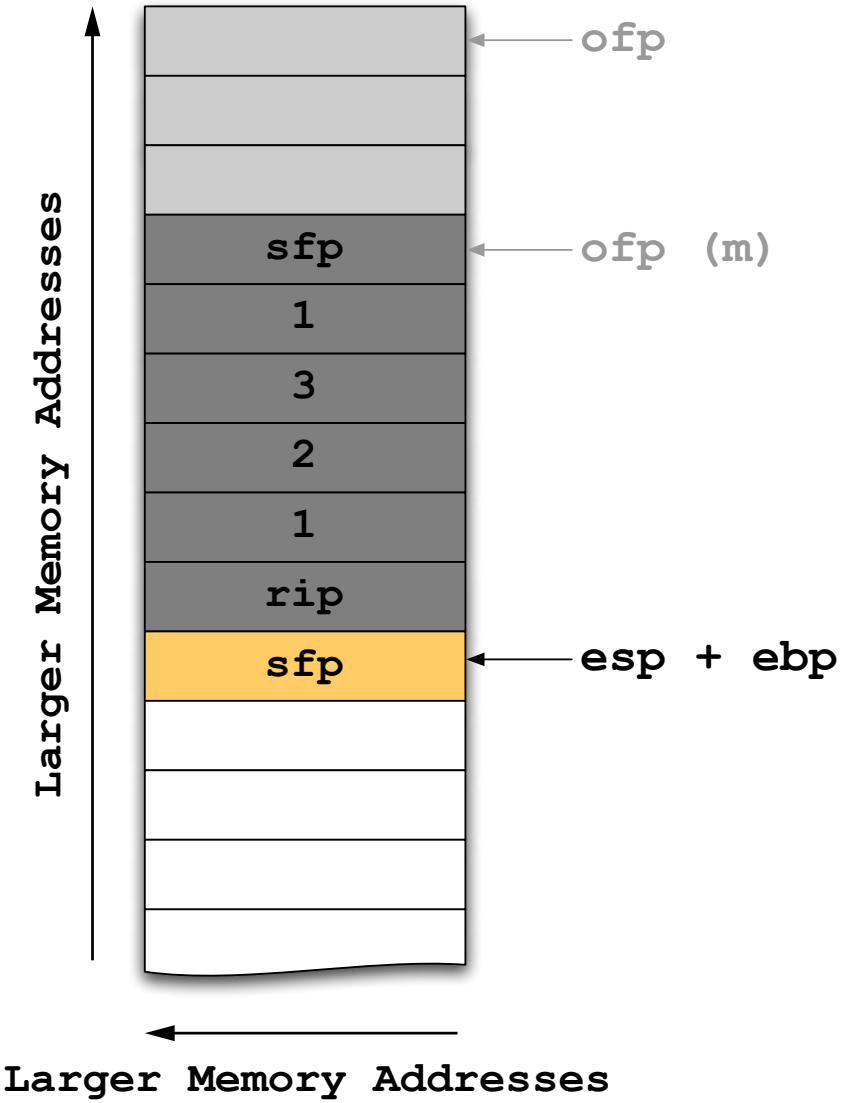
```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



Function Calls in Assembler

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

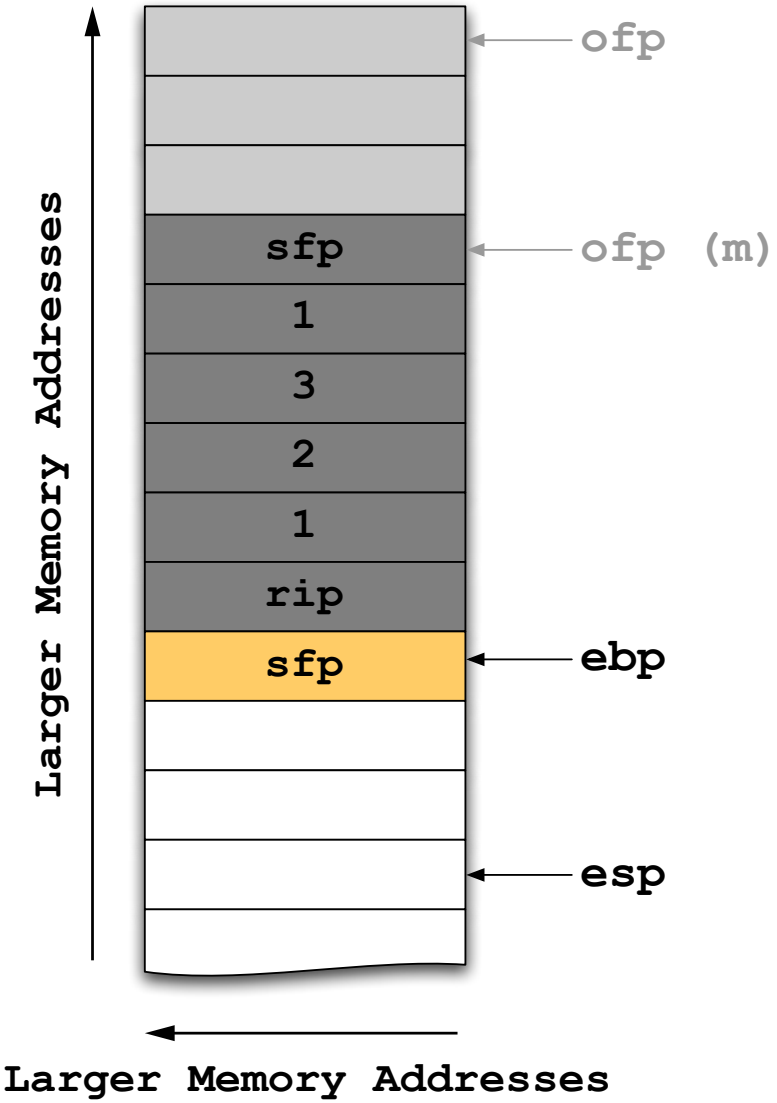
```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



Function Calls in Assembler

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

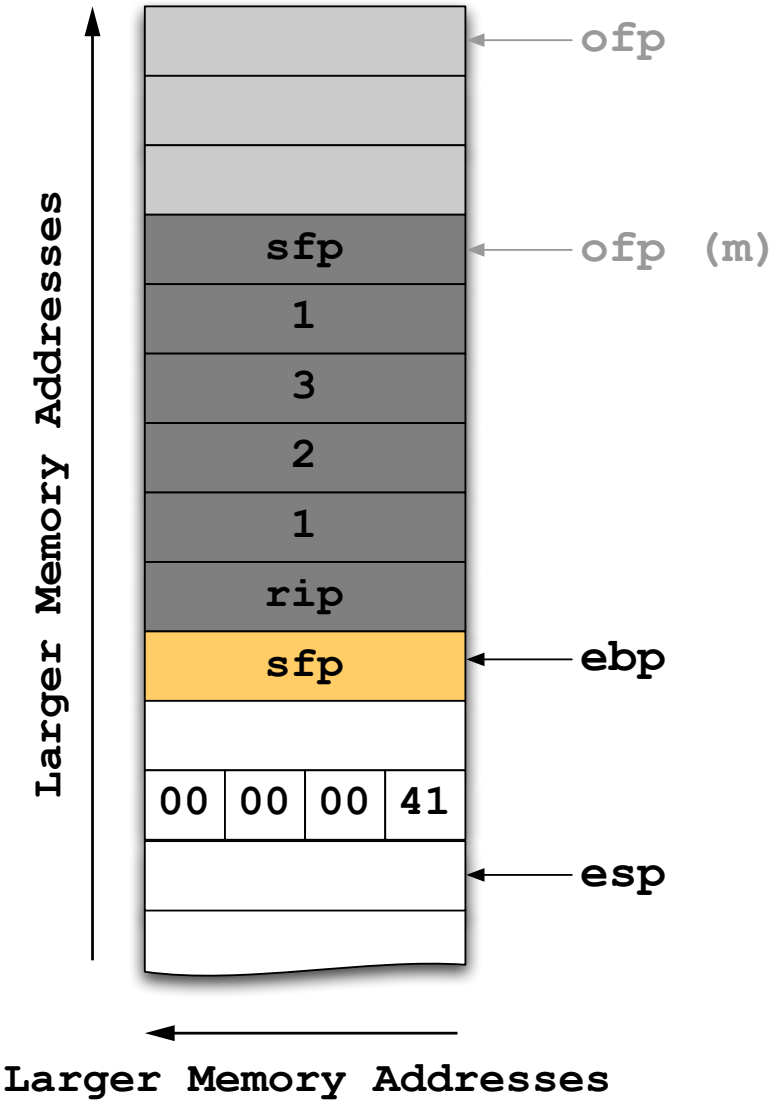
```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



Function Calls in Assembler

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

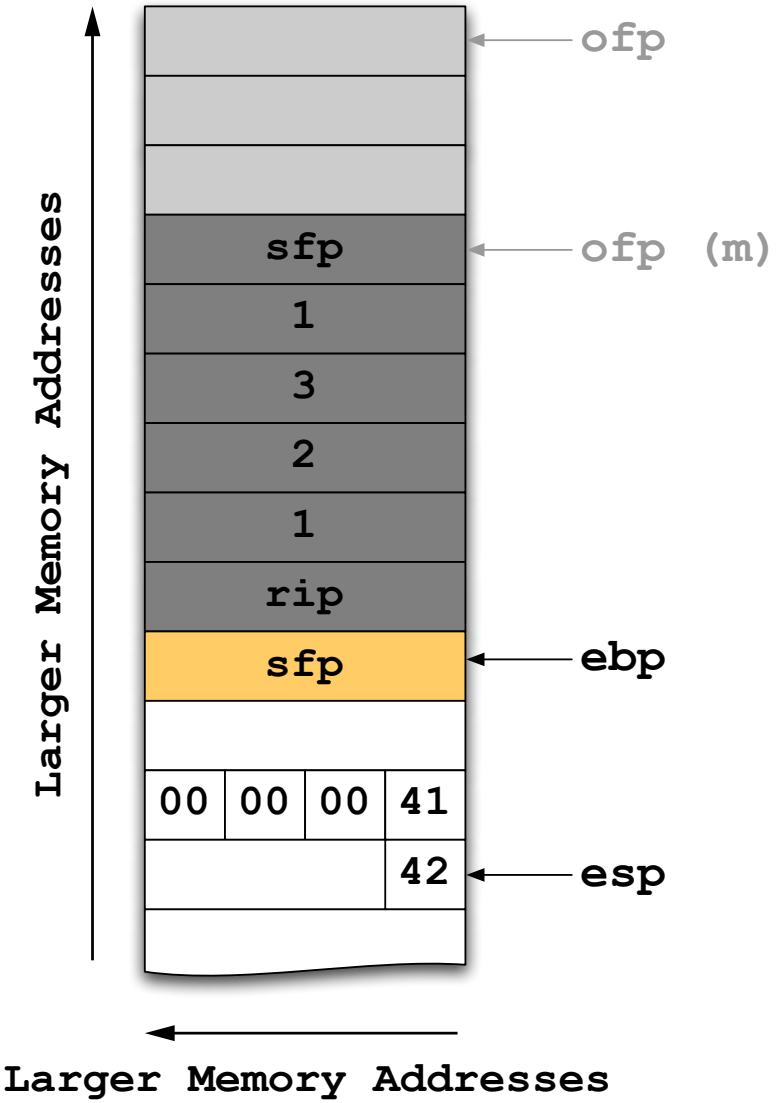
```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



Function Calls in Assembler

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

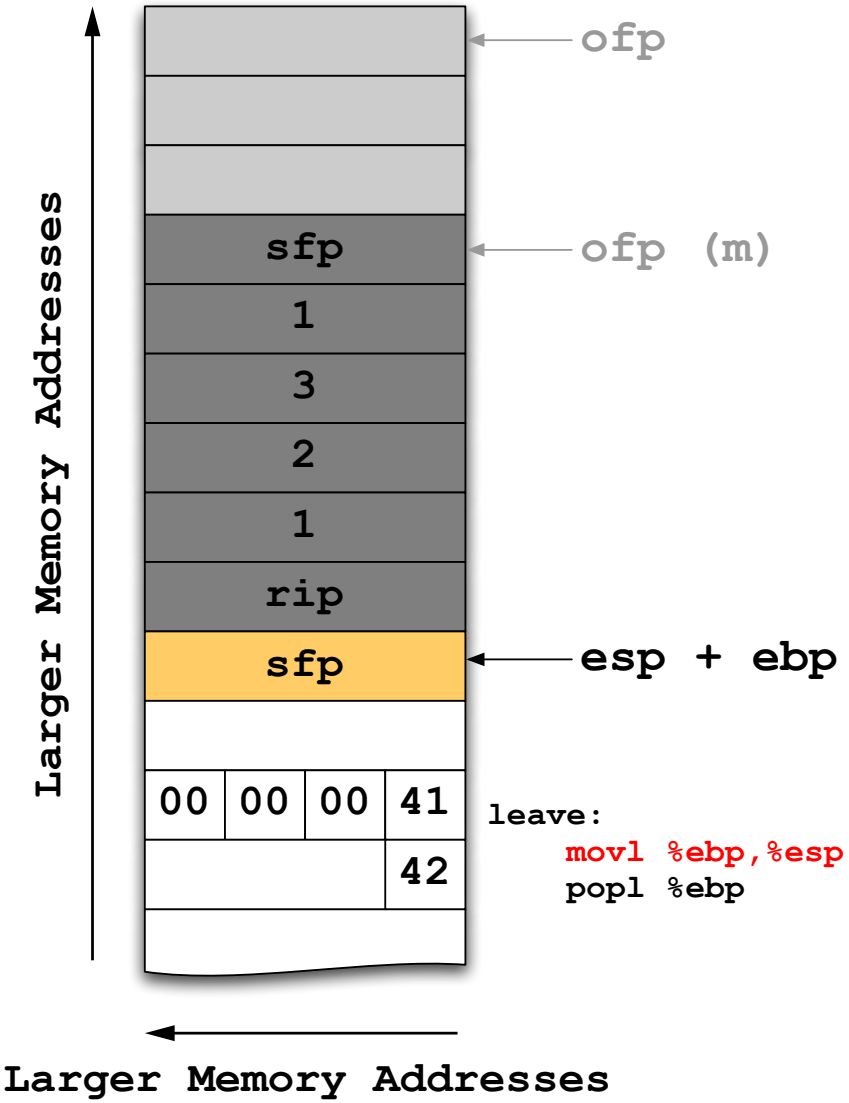
```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



Function Calls in Assembler

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

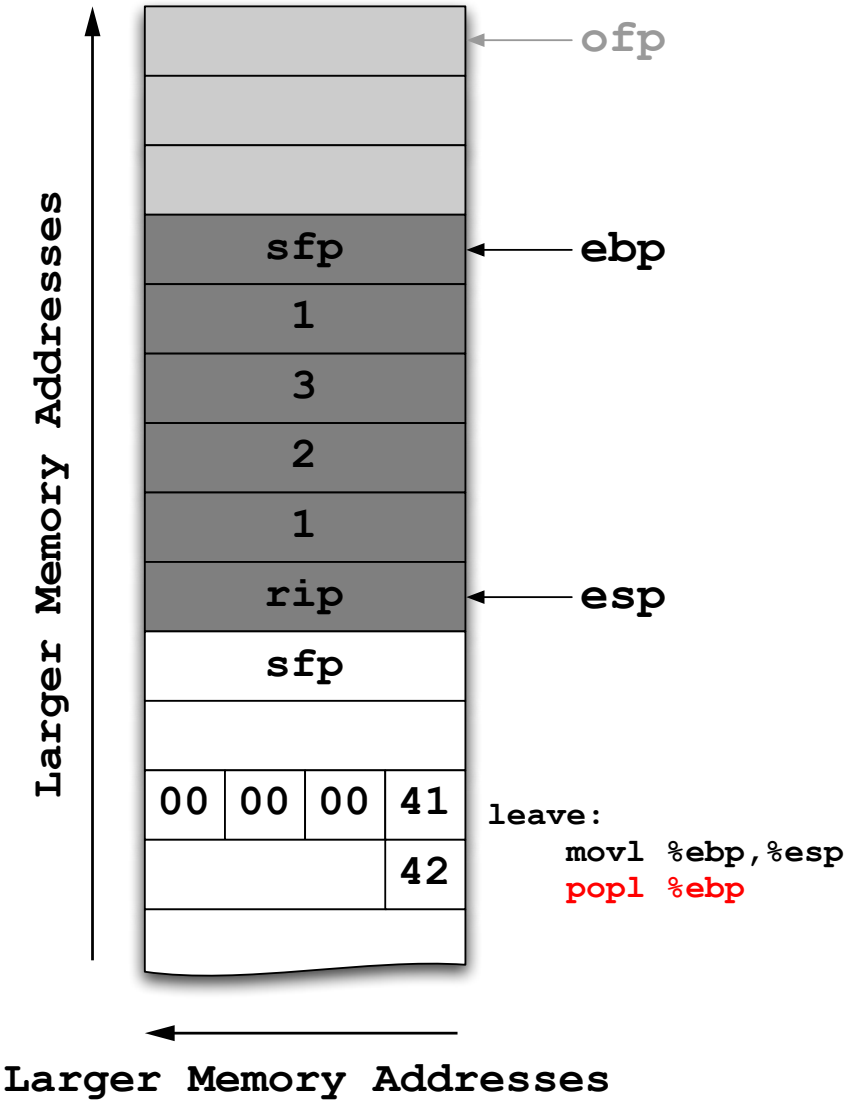
```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



Function Calls in Assembler

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

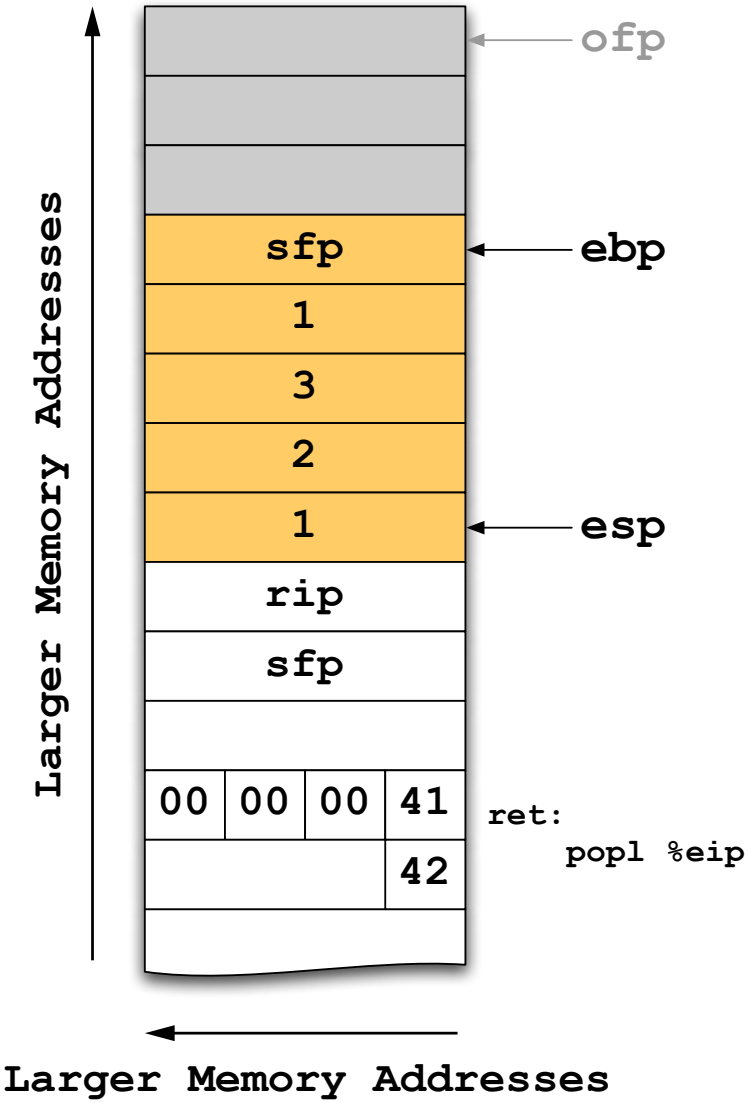
```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



Function Calls in Assembler

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

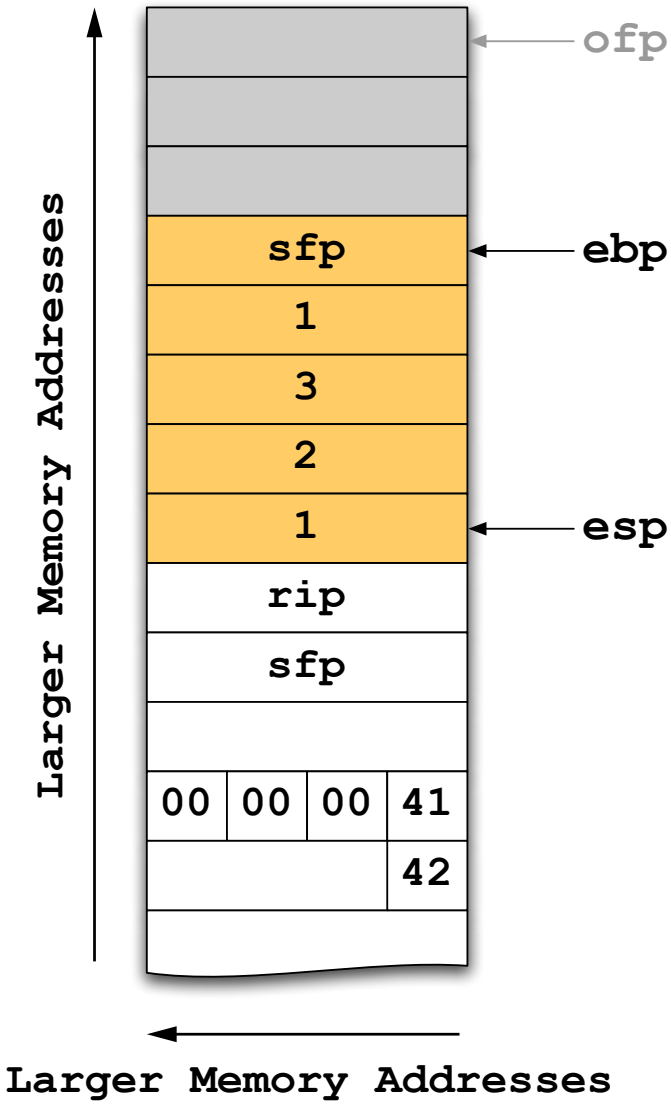
```
foo:
    pushl %ebp
    movl  %esp,%ebp
    subl  $12,%esp
    movl  $65,-8(%ebp)
    movb  $66,-12(%ebp)
    leave
    ret
```



Function Calls in Assembler

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

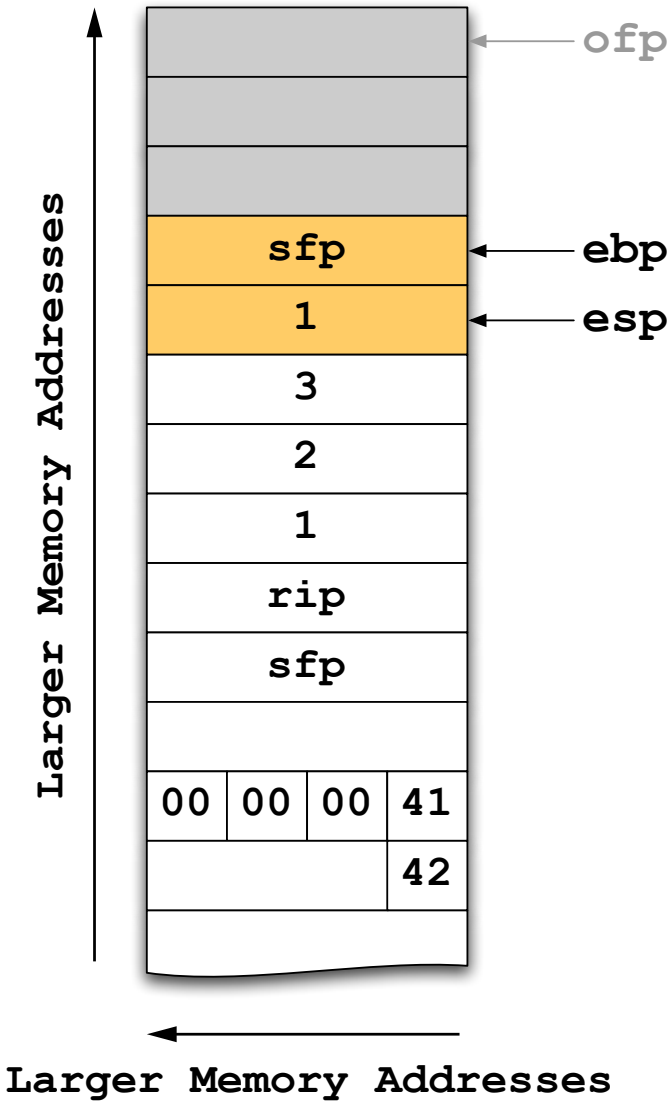
```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



Function Calls in Assembler

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

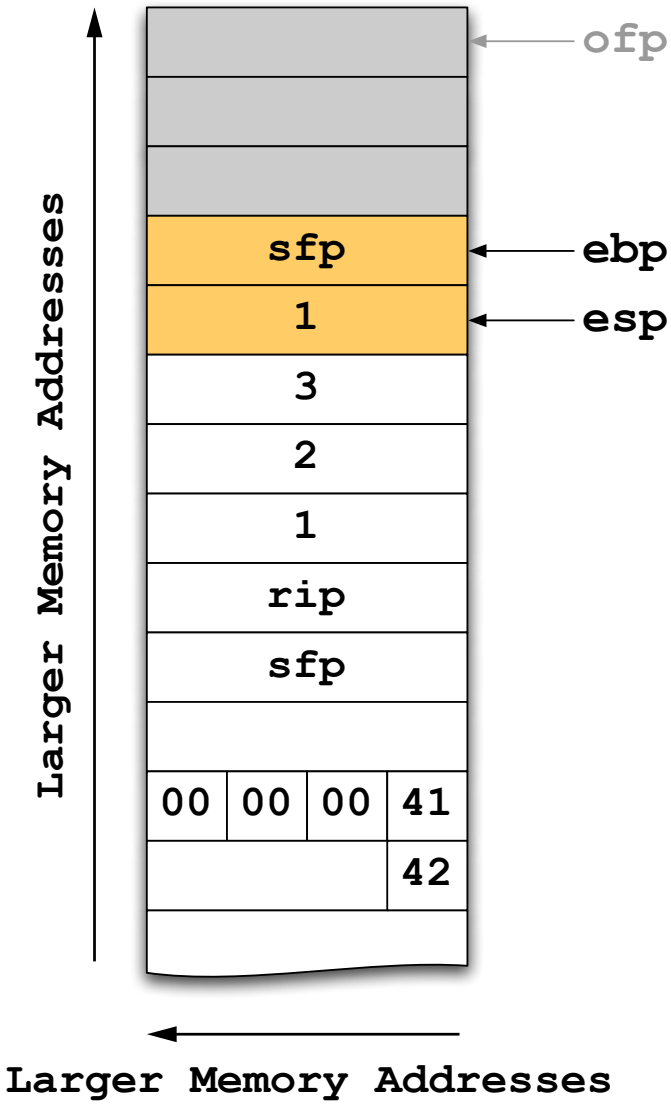
```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



Function Calls in Assembler

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

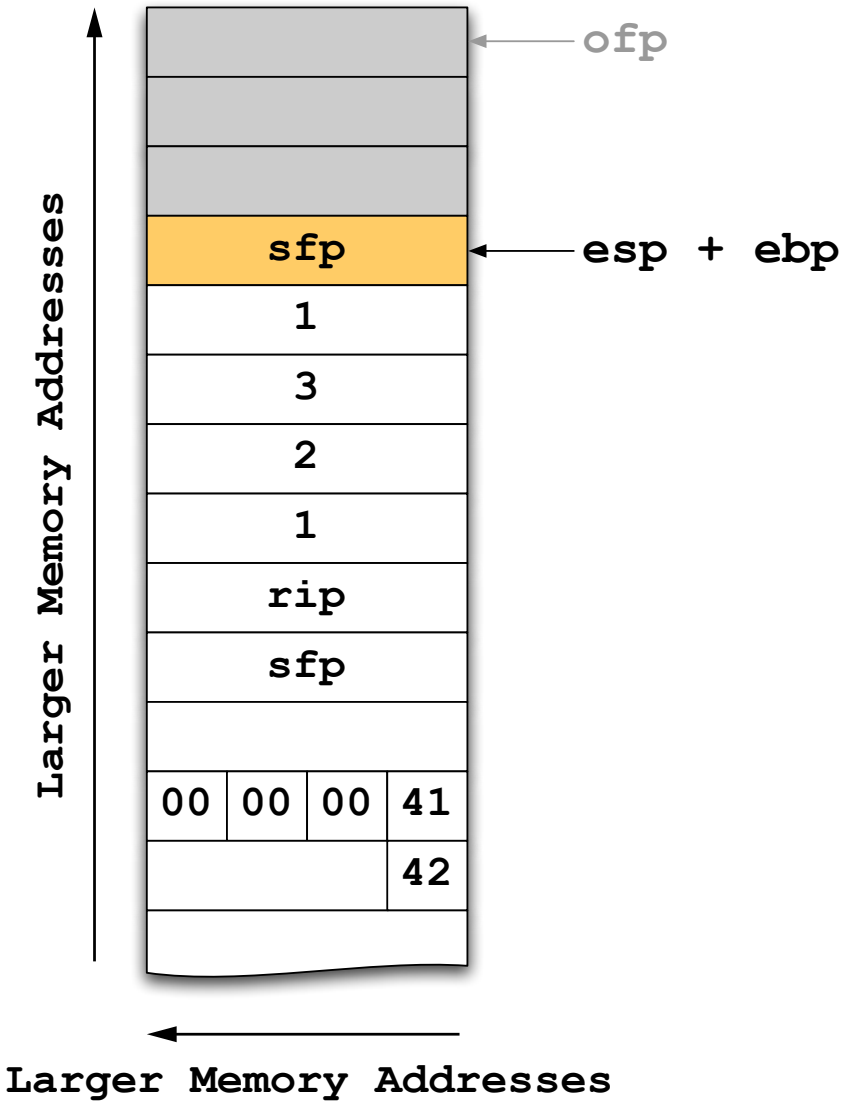
```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



Function Calls in Assembler

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

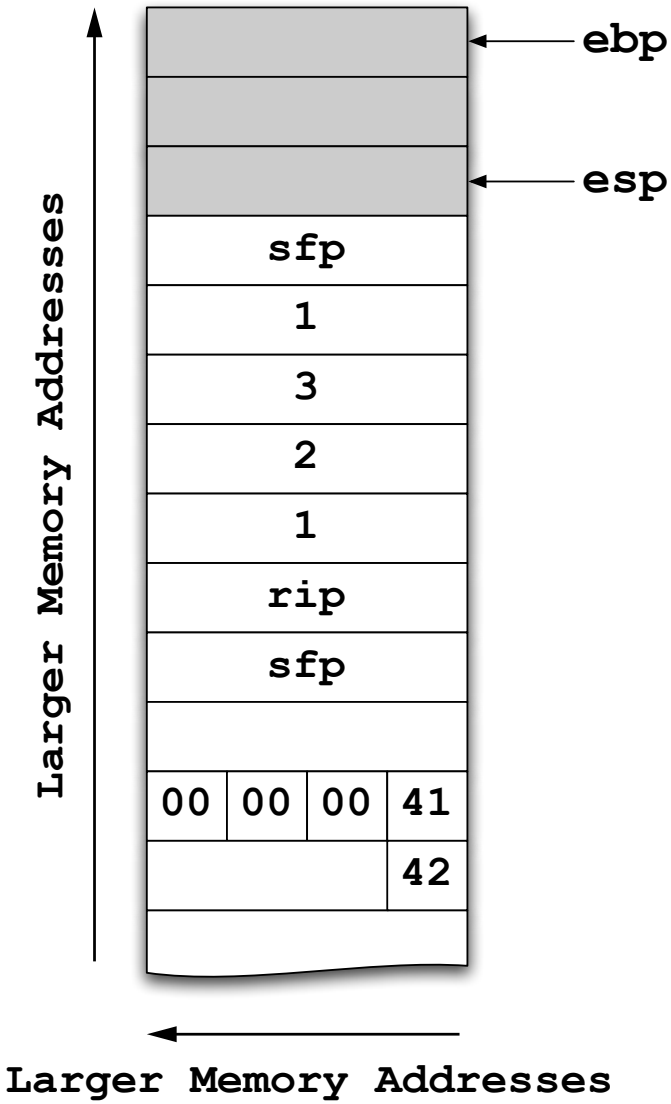
```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



Function Calls in Assembler

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

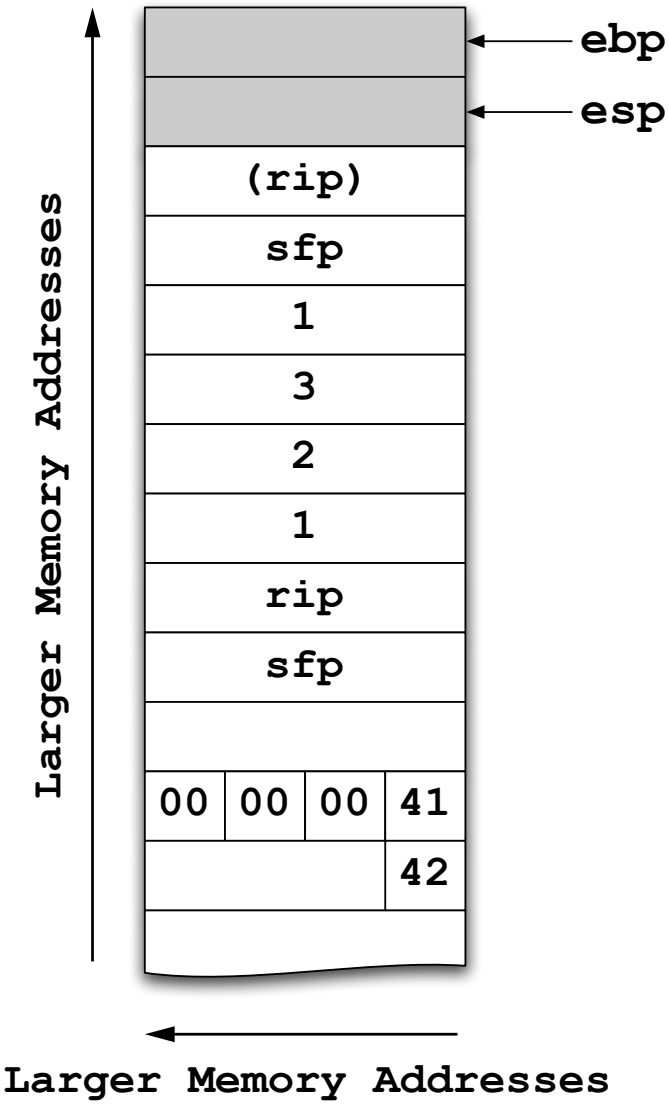
```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```



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
```



MIPS → IA-32 [Reference]

- **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 [Reference]

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

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