

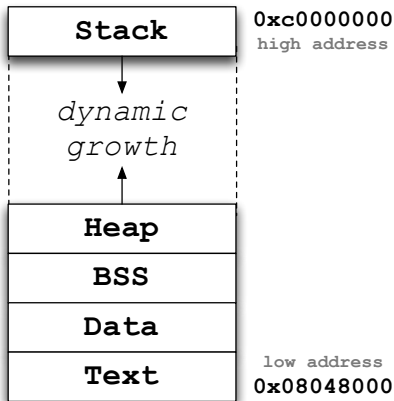
# Process Layout and Function Calls

CS 161 – Spring 2016

January 25, 2016

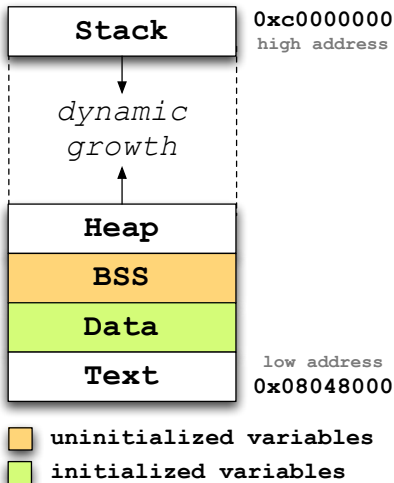
# Process Layout in Memory

- ▶ **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).



- ▶ **Limited Number of Registers**
  - ▶ IA-32 has 6 (%eax, %edx, %ecx, %ebx, %esi, %edi)
    - ▶ This means lots of stack operations!
- ▶ **Operand Directions**
  - ▶ 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!**

## Registers

Use	IA32	Notes
Program Counter	%eip	Can not be referenced directly
Stack Pointer	%esp	
Frame Pointer	%ebp	
Return Value (32 bit)	%eax	%eax not used solely for RV

## Register Terminology

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

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

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

# 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;
}
```

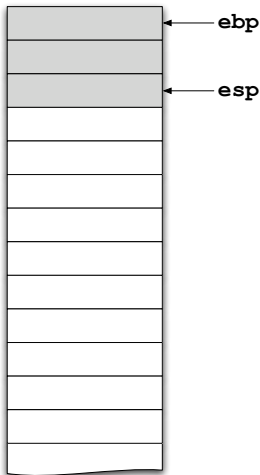
# Function Calls in Assembler

```
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
```



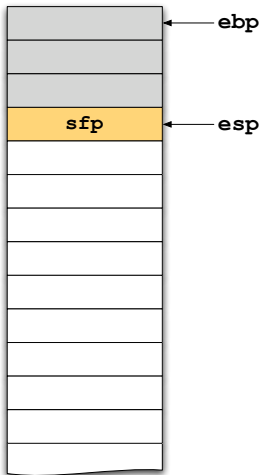
# Function Calls in Assembler

```
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
```





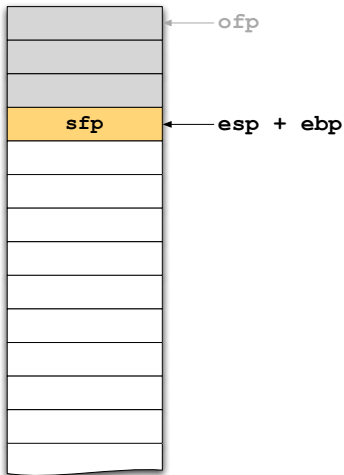
## Function Calls in Assembler

```
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
```

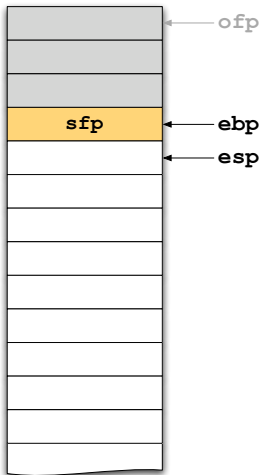


## Function Calls in Assembler

```
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
```



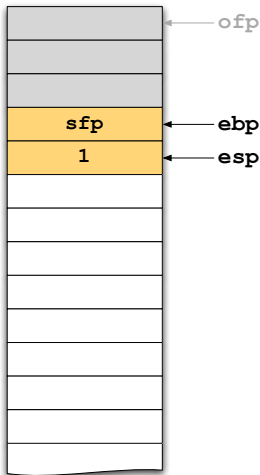
# Function Calls in Assembler

```
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
```

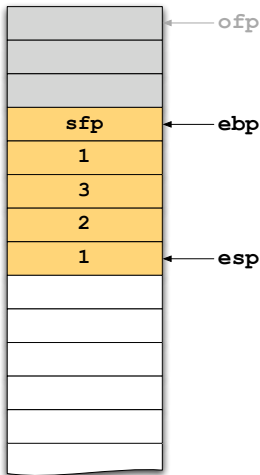


# Function Calls in Assembler

```
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
```

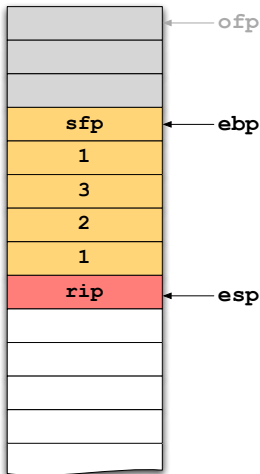


# Function Calls in Assembler

```
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
```

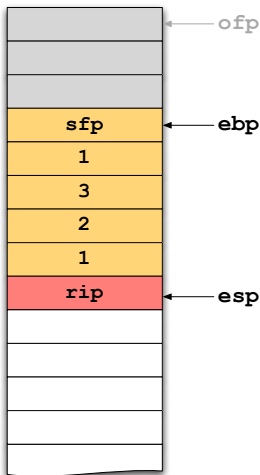


## Function Calls in Assembler

```
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
```

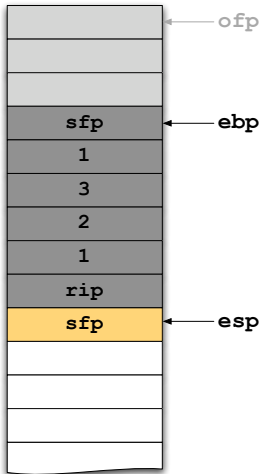


## Function Calls in Assembler

```
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
```

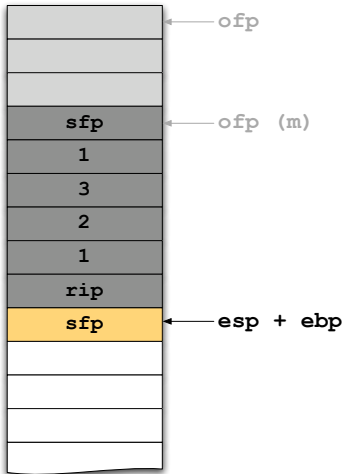


## Function Calls in Assembler

```
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
```



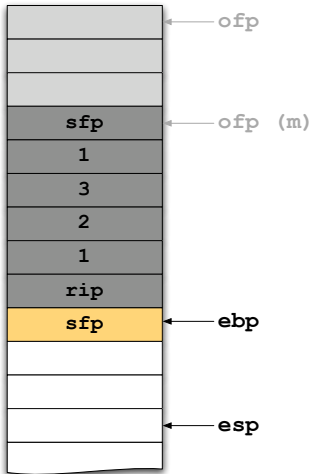


# Function Calls in Assembler

```
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
```

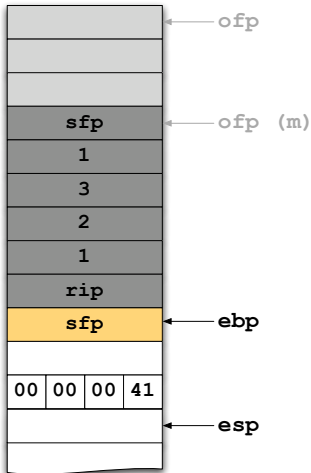


# Function Calls in Assembler

```
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
```

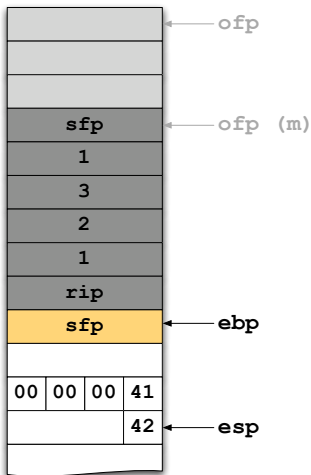


## Function Calls in Assembler

```
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
```

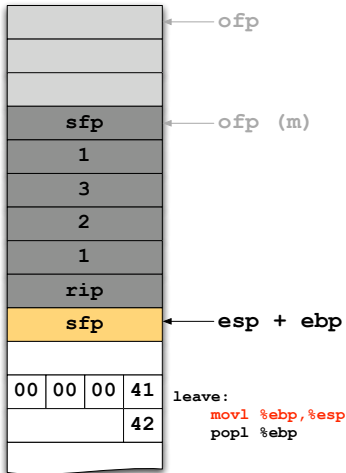


# Function Calls in Assembler

```
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
```

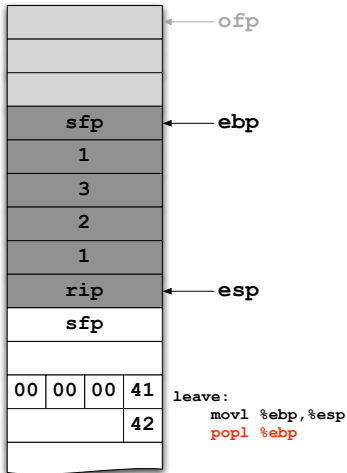


# Function Calls in Assembler

```
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
```

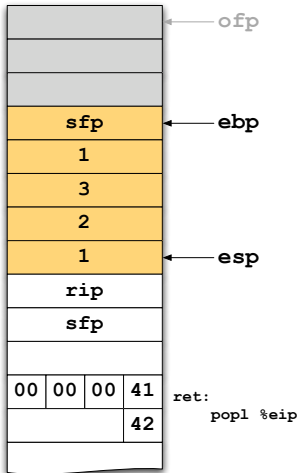


# Function Calls in Assembler

```
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
```

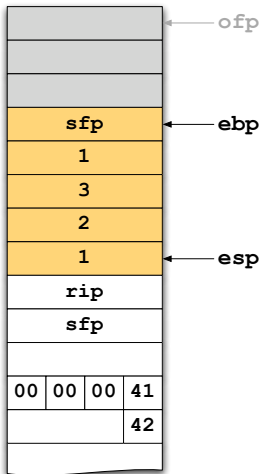


# Function Calls in Assembler

```
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
```

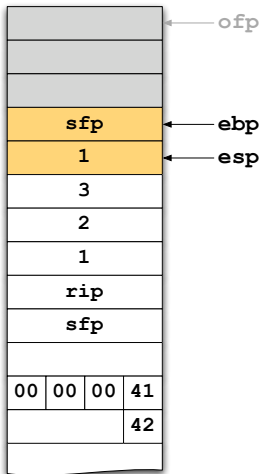


# Function Calls in Assembler

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

---

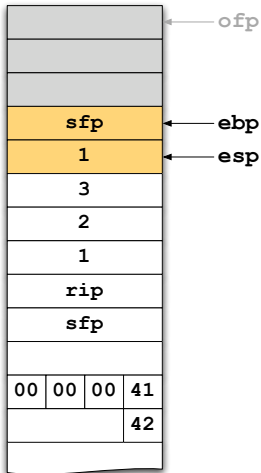
```
main:
    pushl %ebp
    movl  %esp,%ebp
    subl  $4,%esp
    movl  $1,-4(%ebp)
    pushl $3
    pushl $2
    pushl $1
    call  foo
    addl  $12,%esp
    xorl  %eax,%eax
    leave
    ret
```





# 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
```

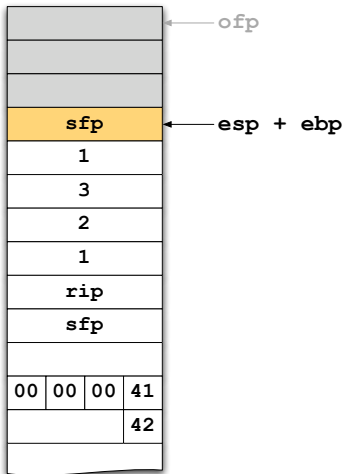


# Function Calls in Assembler

```
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
```

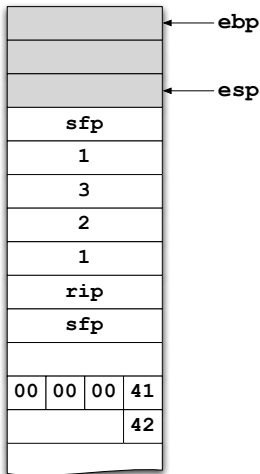


# Function Calls in Assembler

```
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
```



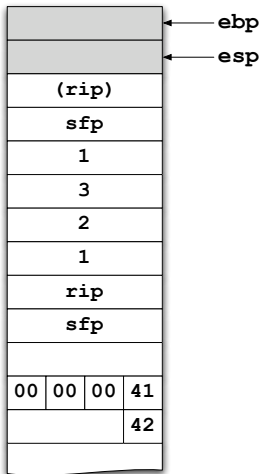
# Function Calls in Assembler

```
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

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

## Addressing Modes

Immediate	<i>Sval</i>	<i>Val</i>
Normal	(R)	Mem[Reg[R]]
		•Register R specifies memory address
		<code>movl (%ecx), %eax</code>
Displacement	D(R)	Mem[Reg[R]+D]
		•Register R specifies start of memory region
		•Constant displacement D specifies offset
		<code>movl 8(%ebp), %edx</code>
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

`%eax``%edx``%ecx``%ebx``%esi``%edi``%esp``%ebp`