Example x86 Code

Computer Systems Sections 3.3 - 3.6

Simple C Function

```
int myfunc(int a,int b) {
    int c;
    c = a + 3;
    c = c + b;
    return c;
}
```

gcc -m32 -O0 -S prog.c

prog.c

```
int myfunc(int a,int b) {
        int c;
        c = a + 3;
        c = c + b;
        return c;
}
```

prog.s

```
myfunc:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    8(%ebp), %eax
        addl    $3, %eax
        movl    %eax, -4(%ebp)
        movl    12(%ebp), %eax
        addl    %eax, -4(%ebp)
        movl    -4(%ebp), %eax
        leave
        ret
```

Function Invocation Details (Lecture 10)

[OS(caller) invokes Main (callee)]
[Main (callee) startup)]

```
a=3;
b=4;
```

[Main (caller) invokes myfn(callee)]
x=a, y=b+2;
[myfn (callee) startup]

```
x*x + y*y
```

[myfn (callee) return to main]
[main (caller) return from myfn]

```
c = [return value from myfn]
```

[main (callee) return to OS]

- Evaluate arguments
- Copy argument values to parameters
- Preserve Caller's state
- Save return address

- Preserve caller's state
- Create/initialize local variables
- Establish argument accessibility

myfunc entrance

- Preserve caller's state
- Create/initialize local variables
- Establish argument accessibility

```
int myfunc(int a,int b) {
        int c;
        c = a + 3;
        c = c + b;
        return c;
}
```

```
myfunc:
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
movl    8(%ebp), %eax
addl    $3, %eax
movl    %eax, -4(%ebp)
movl    12(%ebp), %eax
addl    %eax, -4(%ebp)
movl    -4(%ebp), %eax
leave
ret
```

Memory		
	xFFFF FFF0	
b	xFFFF FFEC	x0000 0004
a	xFFFF FFE8	x0000 0003
	xFFFF FFE4	
	xFFFF FFE0	
c	xFFFF FFDC	xdead beef
	xFFFF FFD8	
	...	

Reg	Value
ebp	xFFFF FFE0
eax	x0000 0000

myfunc statement 1

```
int myfunc(int a,int b) {
        int c;
        c = a + 3;
        c = c + b;
        return c;
}
```

```
myfunc:
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
movl    8(%ebp), %eax
addl    $3, %eax
movl    %eax, -4(%ebp)
movl    12(%ebp), %eax
addl    %eax, -4(%ebp)
movl    -4(%ebp), %eax
leave
ret
```

Memory		
	xFFFF FFF0	
b	xFFFF FFEC	x0000 0004
a	xFFFF FFE8	x0000 0003
	xFFFF FFE4	
	xFFFF FFE0	
c	xFFFF FFDC	xdead beef
	xFFFF FFD8	
	...	

Reg	Value
ebp	xFFFF FFE0
eax	x0000 0003

myfunc statement 1

```
int myfunc(int a,int b) {
        int c;
        c = a + 3;
        c = c + b;
        return c;
}
```

```
myfunc:
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
movl    8(%ebp), %eax
addl    $3, %eax
movl    %eax, -4(%ebp)
movl    12(%ebp), %eax
addl    %eax, -4(%ebp)
movl    -4(%ebp), %eax
leave
ret
```

Memory		
	xFFFF FFF0	
b	xFFFF FFEC	x0000 0004
a	xFFFF FFE8	x0000 0003
	xFFFF FFE4	
	xFFFF FFE0	
c	xFFFF FFDC	xdead beef
	xFFFF FFD8	
	...	

Reg	Value
ebp	xFFFF FFE0
eax	x0000 0006

myfunc statement 1

```
int myfunc(int a,int b) {
        int c;
        c = a + 3;
        c = c + b;
        return c;
}
```

```
myfunc:
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
movl    8(%ebp), %eax
addl    $3, %eax
movl    %eax, -4(%ebp)
movl    12(%ebp), %eax
addl    %eax, -4(%ebp)
movl    -4(%ebp), %eax
leave
ret
```

Memory		
	xFFFF FFF0	
b	xFFFF FFEC	x0000 0004
a	xFFFF FFE8	x0000 0003
	xFFFF FFE4	
	xFFFF FFE0	
c	xFFFF FFDC	x0000 0006
	xFFFF FFD8	
	...	

Reg	Value
ebp	xFFFF FFE0
eax	x0000 0006

myfunc statement 2

```
int myfunc(int a,int b) {
        int c;
        c = a + 3;
        c = c + b;
        return c;
}
```

```
myfunc:
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
movl    8(%ebp), %eax
addl    $3, %eax
movl    %eax, -4(%ebp)
movl    12(%ebp), %eax
addl    %eax, -4(%ebp)
movl    -4(%ebp), %eax
leave
ret
```

Memory		
	xFFFF FFF0	
b	xFFFF FFEC	x0000 0004
a	xFFFF FFE8	x0000 0003
	xFFFF FFE4	
	xFFFF FFE0	
c	xFFFF FFDC	x0000 0006
	xFFFF FFD8	
	...	

Reg	Value
ebp	xFFFF FFE0
eax	x0000 0004

myfunc statement 2

```
int myfunc(int a,int b) {
        int c;
        c = a + 3;
        c = c + b;
        return c;
}
```

```
myfunc:
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
movl    8(%ebp), %eax
addl    $3, %eax
movl    %eax, -4(%ebp)
movl    12(%ebp), %eax
addl    %eax, -4(%ebp)
movl    -4(%ebp), %eax
leave
ret
```

Memory		
	xFFFF FFF0	
b	xFFFF FFEC	x0000 0004
a	xFFFF FFE8	x0000 0003
	xFFFF FFE4	
	xFFFF FFE0	
c	xFFFF FFDC	x0000 000A
	xFFFF FFD8	
	...	

Reg	Value
ebp	xFFFF FFE0
eax	x0000 0004

Function Return Details

[OS(caller) invokes Main (callee)]
[Main (callee) startup)]
a=3;
b=4;
[Main (caller) invokes myfn(cal
x=a, y=b+2;
[myfn (callee) startup]
x*x + y*y
[myfn (callee) return to main]
[main (caller) return from myfn]
c = [return value from myfn]
[main (callee) return to OS]

- Evaluate return expression
- Save return value
- Free local variables
- Restore caller's state
- Branch to return address

- Restore caller's state
- Free space for arguments/return address
- Use returned value

myfunc return

```
int myfunc(int a,int b) {
        int c;
        c = a + 3;
        c = c + b;
        return c;
}
```

myfunc:

```
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
movl    8(%ebp), %eax
addl    $3, %eax
movl    %eax, -4(%ebp)
movl    12(%ebp), %eax
addl    %eax, -4(%ebp)
movl    -4(%ebp), %eax
leave
ret
```

- Evaluate return expression
- Save return value
- Free local variables
- Restore caller's state
- Branch to return address

Memory		
	xFFFF FFF0	
b	xFFFF FFEC	x0000 0004
a	xFFFF FFE8	x0000 0003
	xFFFF FFE4	
	xFFFF FFE0	
c	xFFFF FFDC	x0000 000A
	xFFFF FFD8	
	...	

Reg	Value
ebp	xFFFF FFE0
eax	x0000 000A

If/Then/Else

```
int myfunc(int a,int b) {
        int c;
        if (a>b) c = a;
        else c = b;
        return c;
}
```

```
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    8(%ebp), %eax
        cmpl    12(%ebp), %eax
        jle     .L2
        movl    8(%ebp), %eax
        movl    %eax, -4(%ebp)
        jmp     .L3
.L2:
        movl    12(%ebp), %eax
        movl    %eax, -4(%ebp)
.L3:
        movl    -4(%ebp), %eax
        leave
        ret
```

If/Then/Else

```
int myfunc(int a,int b) {
        int c;
        if (a>b)
                c = a;
        else
                c = b;
        return c;
}
```

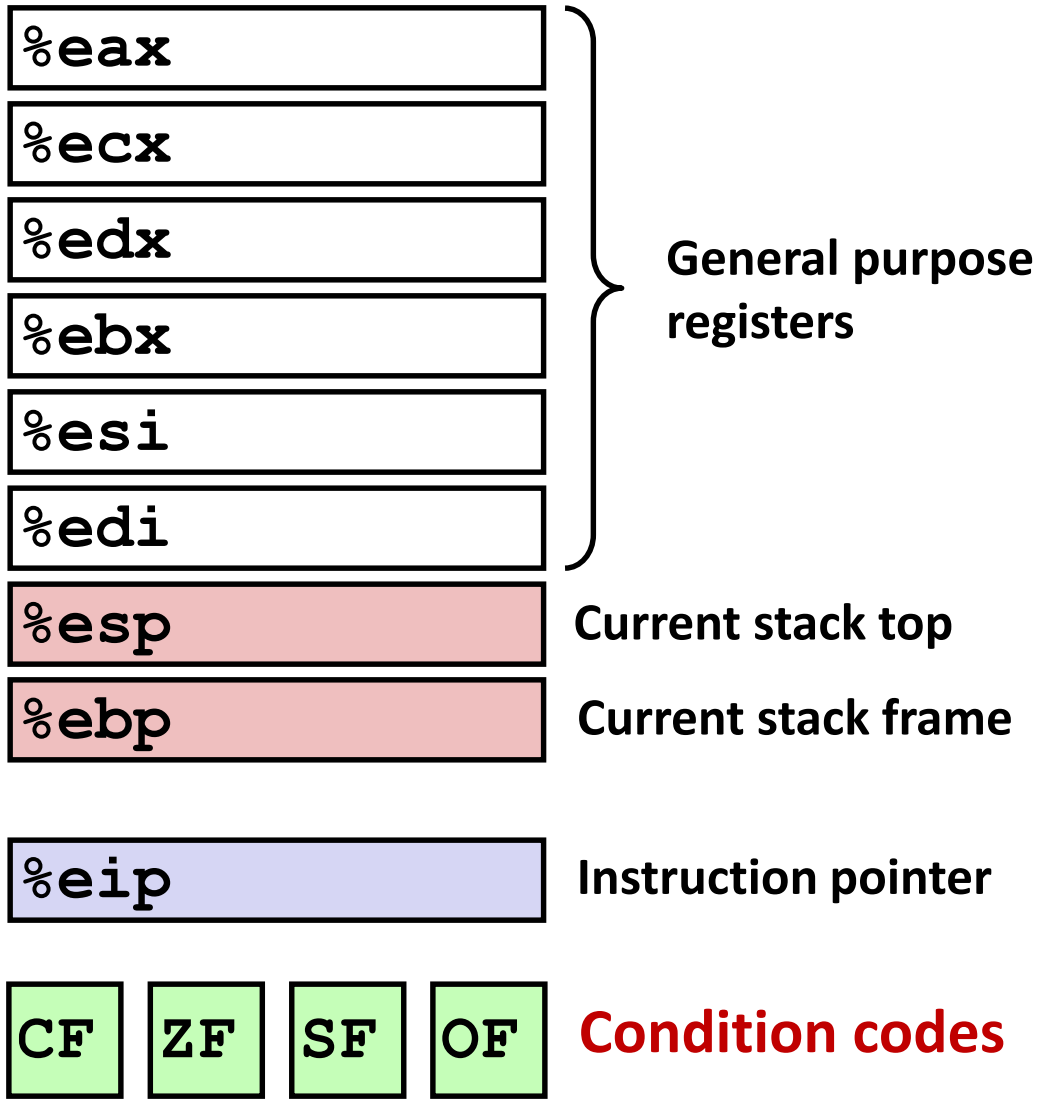
```
pushl   %ebp
movl    %esp, %ebp
subl    $16, %esp
movl    8(%ebp), %eax
cmpl    12(%ebp), %eax
jle     .L2
movl    8(%ebp), %eax
movl    %eax, -4(%ebp)
jmp     .L3
.L2:
movl    12(%ebp), %eax
movl    %eax, -4(%ebp)
.L3:
movl    -4(%ebp), %eax
leave
ret
```

Memory	
b	x0000 0004
a	x0000 0003
(%ebp)	
c	xdead beef

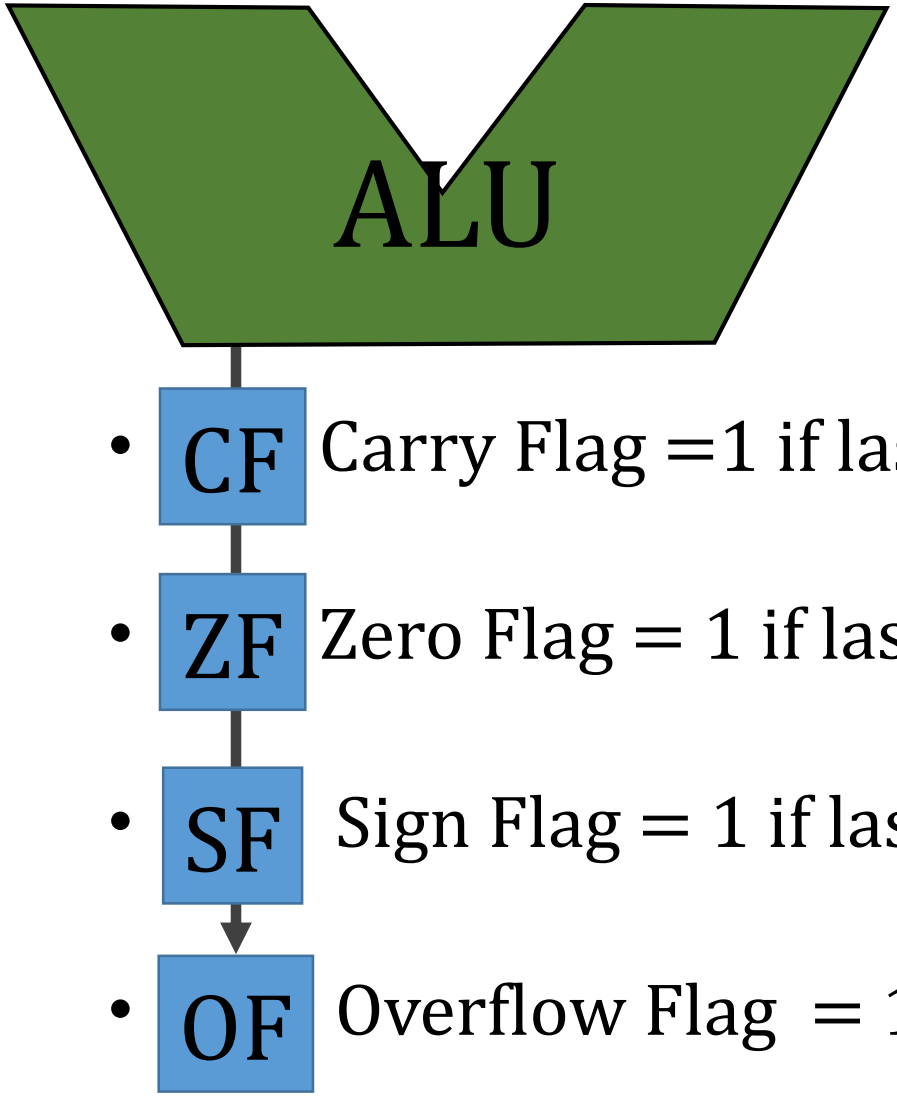
Processor State (IA32)

Information about currently executing program

- Temporary data
  (**%eax**, …)
- Location of runtime stack
  (**%ebp**,**%esp**)
- Location of current code control point
  (**%eip**, …)
- Status of recent tests
  (**CF, ZF, SF, OF**)
- …



Condition Code Registers



- CF Carry Flag =1 if last result MSB overflow (unsigned)
- ZF Zero Flag = 1 if last result was zero
- SF Sign Flag = 1 if last result < 0 (MSB==1)
- OF Overflow Flag = 1 if last result sign is incorrect (++- or --+)

Condition Codes (Implicit Setting)

- Implicitly set by arithmetic operations

  Example: **`addl`** *a,b* ; ***`b′=a+b`***

  CF set if carry out from most significant bit (unsigned overflow)

  ZF set if **`b′ == 0`**

  SF set if **`b′ < 0`** (as signed)

  OF set if two's-complement (signed) overflow

  `(a>0 && b>0 && b′<0) || (a<0 && b<0 && b′>=0)`

- Not set by **`lea`** instruction

- Full documentation (IA32) or Wikibooks X86 Control Flow

Condition Codes (Explicit Set: Compare)

Condition code can be set explicitly by the "cmp" instruction…

- **`cmpl/cmpq`** *Src2, Src1*
- **`cmpl b,a`** like computing **`a-b`** without setting destination

- **ZF set** if **`a == b`**
- **SF set** if **`(a-b) < 0`** (as signed)
- **OF set** if two's-complement (signed) overflow

```
(a>0 && b<0 && (a-b)<0) || (a<0 && b>0 && (a-b)>0)
```

- **CF set** if carry out from most significant bit (used for unsigned comparisons)

Condition Code Interpretations

Based on CMP B,A → A-B or SUB B,A → A=A-B

Signed Comparisons

Branch	Means	ZF	SF	OF	CF
je	a== b	1	X	X	X
jne	a!=b	0	X	X	X
jg	a>b	0	0	0	X
		0	1	1	X
jge	a>=b	X	0	0	X
		X	1	1	X
jl	a<b	0	1	0	X
		0	0	1	X
jle	a<=b	X	1	0	X
		X	0	1	X

Unsigned Comparisons

Branch	Means	ZF	SF	OF	CF
je	a==b	1	X	X	X
jne	a!=b	0	X	X	X
ja	a>b	0	X	X	0
jae	a>=b	X	X	X	0
jb	b>a	X	X	X	1
jbe	b>=a	X	X	X	1
		1	X	X	X

Carry Flag Confusion

- Question: How does the carry flag get set when performing unsigned subtraction?
- Official Answer: The carry flag is set if either the most significant bit overflows OR if a borrow is required to the most significant bit to perform the subtraction

							1
	0	…	0	0	0	~~1~~ 0	0
-	0	…	0	0	0	0	1
	1	…	1	1	1	0	1

1	1	…	1	1	1	1	
	1 Ø	…	1 Ø	1 Ø	1 Ø	0	1
-	0	…	0	0	0	1	0
	1	…	1	1	1	0	1

Carry Flag Confusion

- Unofficial Answer: The carry flag is set if either the most significant bit overflows OR if A+(-B) results in negative (assuming extra sign bit)

1	1	1	…	1	1	1		
	0	0	…	0	0	0	1	0
+	1	1	…	1	1	1	1	1
	0	0	…	0	0	0	0	1

	SGN							
	0	0	…	0	0	0	0	1
+	1	1	…	1	1	1	1	0
	1	1	…	1	1	1	0	1

Condition Codes (Explicit Set: Test)

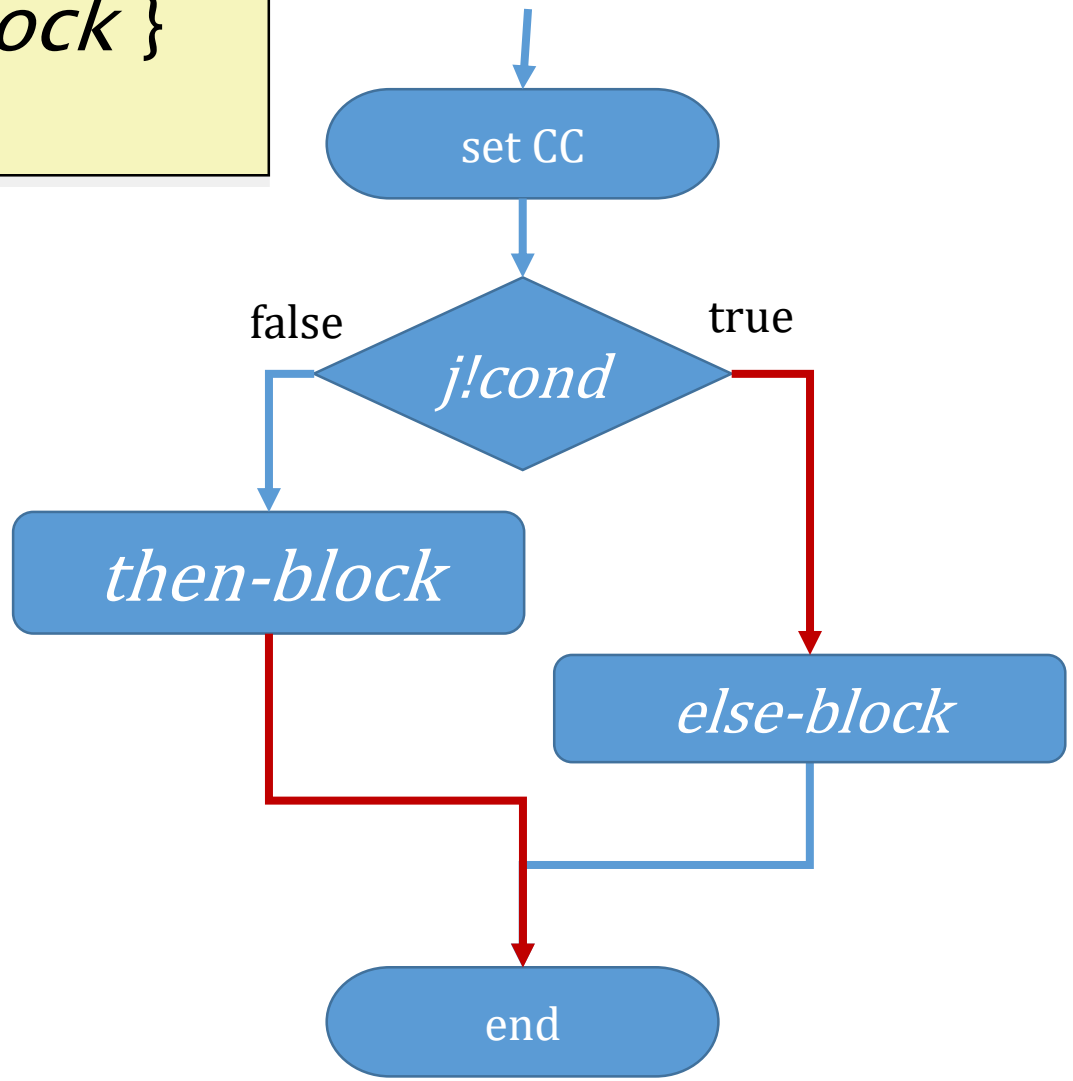
Condition code can be set explicitly by the "test" instruction…

- **`test`** *Src2*, *Src1*
- **`testl a,b`** like computing **`a&b`** without setting destination

- **ZF set** if **`a & b == 0 (a bitwise different from b)`**
- **SF set** if **`(a&b) most significant bit=1 (sign bit)`**
- **OF unset (0)**
- **CF unset (0)**

Translating C to x86: If/then/else

if (*cond*) { *then-block* }
else { *else-block* }




```
    ...
    cmpl    ...

    jxx     .L6

    ...     ; then block
    jmp .L7
.L6:
    ...     ; else block

.L7:
```

If/Then/Else

```
int myfunc(int a,int b) {
        int c;
        if (a>b) c = a;
        else c = b;
        return c;
}
```

```
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    8(%ebp), %eax
        cmpl    12(%ebp), %eax
        jle     .L2
        movl    8(%ebp), %eax
        movl    %eax, -4(%ebp)
        jmp     .L3
.L2:
        movl    12(%ebp), %eax
        movl    %eax, -4(%ebp)
.L3:
        movl    -4(%ebp), %eax
        leave
        ret
```

Memory	
b	x0000 0004
a	x0000 0003
(%ebp)	
c	xdead beef