

More x86 Code

Computer Systems Sections 3.3-3.6

See Also:

http://www.cs.binghamton.edu/~tbartens/CS220_Spring_2016/examples/xmp_x86

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

```
movl    -4(%ebp), %eax
leave
ret
```

[illegible]

While Loops

```
int myfunc(int a,int b) {  
    int c;  
    while(a<10) {  
        c+=b;  
        a--;  
    }  
    return c;  
}
```

gcc -m32 -O0 -S prog.c

```
int myfunc(int a,int b) {
    int c;
    while(a<10) {
        c+=b;
        a--;
    }
    return c;
}
```

myfunc:

```
    pushl    %ebp
    movl     %esp, %ebp
    subl     $16, %esp
    jmp      .L2
.L3:
    movl     12(%ebp), %eax
    addl     %eax, -4(%ebp)
    subl     $1, 8(%ebp)
.L2:
    cmpl     $9, 8(%ebp)
    jle      .L3
    movl     -4(%ebp), %eax
    leave
    ret
```

```
int myfunc(int a,int b) {
    int c;
    while(a<10) {
        c+=b;
        a--;
    }
    return c;
}
```

myfunc:

```
pushl    %ebp
movl     %esp, %ebp
subl     $16, %esp
jmp      .L2

.L3:
movl     12(%ebp), %eax
addl     %eax, -4(%ebp)
subl     $1, 8(%ebp)

.L2:
cmpl     $9, 8(%ebp)
jle      .L3
movl     -4(%ebp), %eax
leave
ret
```

[illegible]

Diagram illustrating the mapping between C code and assembly code for a function named `myfunc`.

C Code:

```
int myfunc(int a, int b) {  
    int c;  
    while(a < 10) {  
        c += b;  
        a--;  
    }  
    return c;  
}
```

Assembly Code:

```
myfunc:  
    pushl    %ebp  
    movl     %esp, %ebp  
    subl     $16, %esp  
    jmp      .L2  
  
    .L3:  
    movl     12(%ebp), %eax  
    addl     %eax, -4(%ebp)  
    subl     $1, 8(%ebp)  
  
    .L2:  
    cmpl     $9, 8(%ebp)  
    jle      .L3  
  
    movl     -4(%ebp), %eax  
    leave  
    ret
```

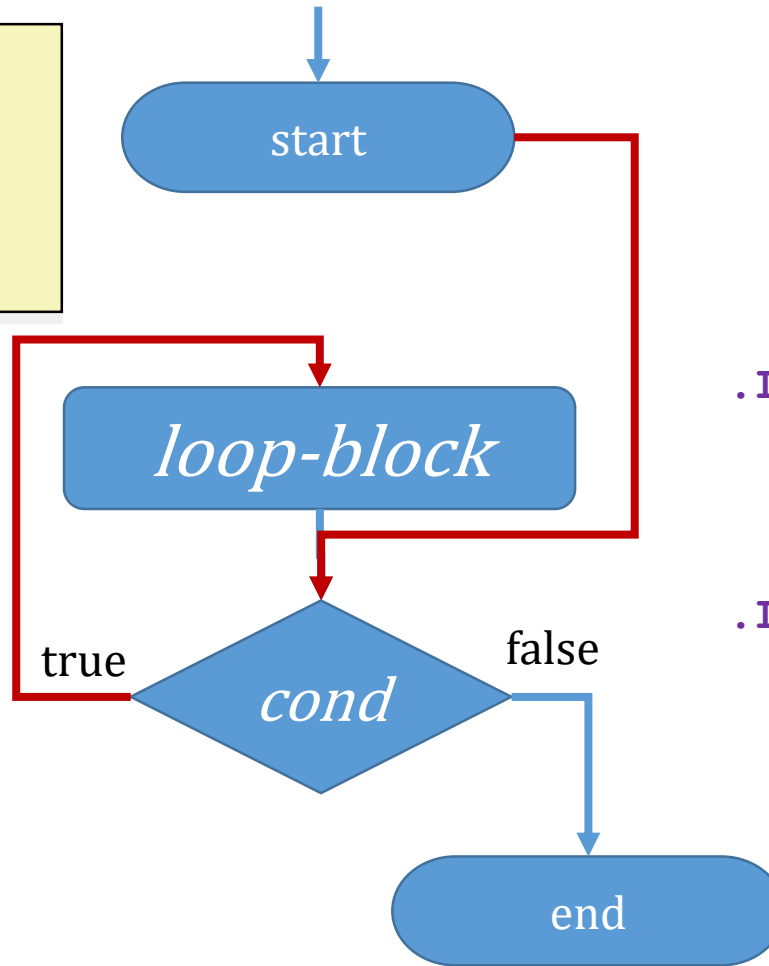
Mapping:

- `int myfunc(int a, int b) {` maps to the initial setup instructions: `pushl %ebp`, `movl %esp, %ebp`, `subl $16, %esp`, and `jmp .L2`.
- `c += b;` maps to the loop body instructions: `movl 12(%ebp), %eax`, `addl %eax, -4(%ebp)`, and `subl $1, 8(%ebp)`.
- `a--;` maps to the loop body instruction: `subl $1, 8(%ebp)`.
- `} return c;` maps to the loop exit instructions: `cmpl $9, 8(%ebp)`, `jle .L3`, `movl -4(%ebp), %eax`, `leave`, and `ret`.

[illegible]

Translating C to x86: while

```
while ( cond ) {  
    loop-block  
}
```



...
jmp L6

.L5:
... ; loop block

.L6:
cmpl ...; condition
jxx **.L5**

...

For Loops

```
int myfunc(int a,int b) {  
    int c;  
    for(c=0;c<a;c++) {  
        b=b+a;  
    }  
    return b;  
}
```


gcc -m32 -O0 -S prog.c

	myfunc:	
int myfunc(int a,int b) {		pushl %ebp
int c;		movl %esp, %ebp
for(c=0;c<a;c++) {		subl \$16, %esp
b=b+a;		movl \$0, -4(%ebp)
}		jmp .L2
return b;	.L3:	movl 8(%ebp), %eax
}		addl %eax, 12(%ebp)
		addl \$1, -4(%ebp)
	.L2:	movl -4(%ebp), %eax
		cmpl 8(%ebp), %eax
		jl .L3
		movl 12(%ebp), %eax
		leave
		ret

```
int myfunc(int a, int b) {
    int c;
    for(c=0; c<a; c++) {
        b=b+a;
    }
    return b;
}
```

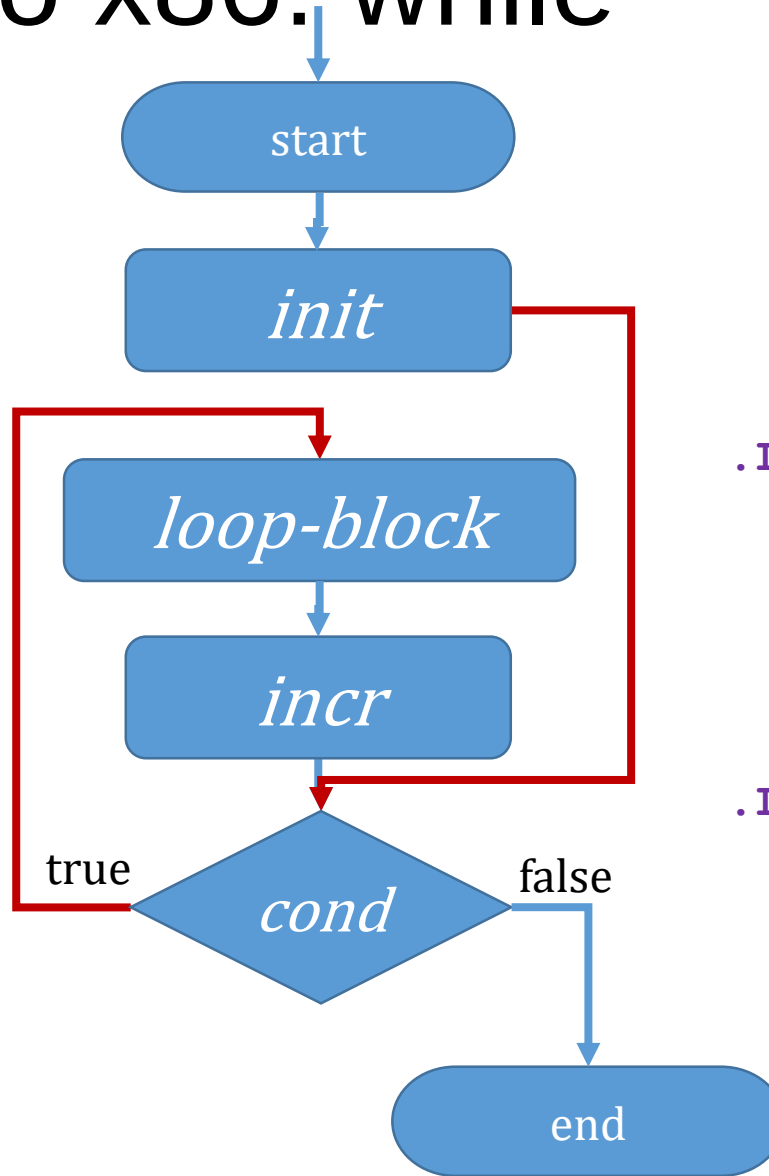
myfunc:

pushl	%ebp
movl	%esp, %ebp
subl	\$16, %esp
movl	\$0, -4(%ebp)
jmp	.L2
.L3:	
movl	8(%ebp), %eax
addl	%eax, 12(%ebp)
addl	\$1, 12(%ebp)
.L2:	
movl	-4(%ebp), %eax
cmpl	8(%ebp), %eax
jl	.L3
movl	12(%ebp), %eax
leave	
ret	

[illegible]

Translating C to x86: while

```
for(init, cond, incr) {  
    loop-block  
}
```



```

...
...      ; init
jmp L6
.L5:
...      ; loop block

...      ; incr

.L6:
cmpl    ...; condition
jxx     .L5
...
    
```

Switch Statements

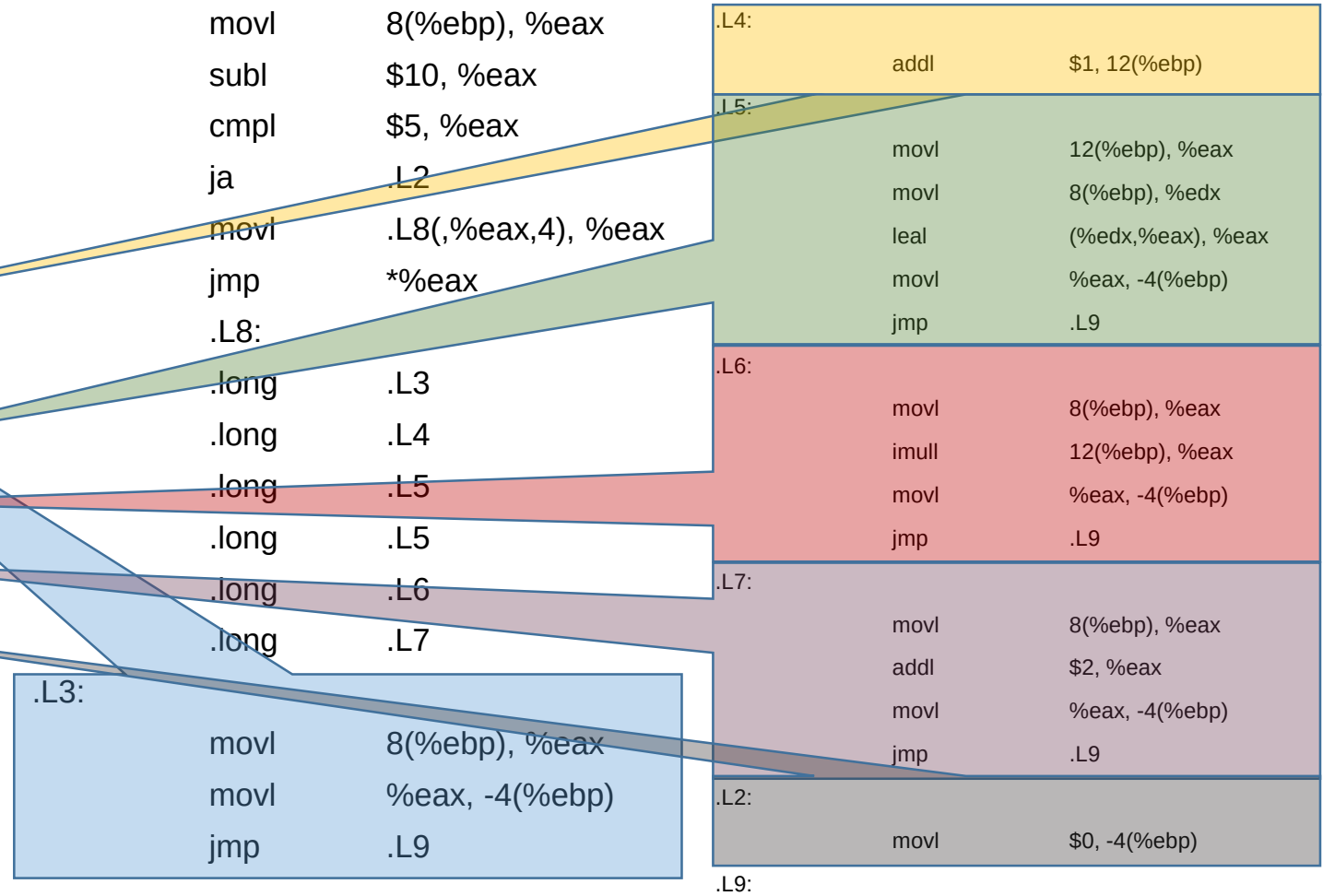
```
int myfunc(int a,int b) {  
    int c;  
    switch(a) {  
        case 10 : c=a; break;  
        case 11: b++;  
        case 12:  
        case 13: c=a+b; break;  
        case 14: c=a*b; break;  
        case 15: c=a+2; break;  
        default: c=0;  
    }  
    return c;  
}
```

gcc -m32 -O0 -S progSwitch.c

		movl	8(%ebp), %eax	.L4:		
		subl	\$10, %eax		addl	\$1, 12(%ebp)
int myfunc(int a,int b) {		cmpl	\$5, %eax	.L5:		
int c;		ja	.L2		movl	12(%ebp), %eax
switch(a) {		movl	.L8(,%eax,4),		movl	8(%ebp), %edx
case 10 : c=a; break;	%eax	jmp	*%eax		leal	(%edx,%eax), %eax
case 11: b++;		.L8:		.L6:	movl	%eax, -4(%ebp)
case 12:		.long	.L3		jmp	.L9
case 13: c=a+b; break;		.long	.L4		movl	8(%ebp), %eax
case 14: c=a*b; break;		.long	.L5		imull	12(%ebp), %eax
case 15: c=a+2; break;		.long	.L5		movl	%eax, -4(%ebp)
default: c=0;		.long	.L6	.L7:	jmp	.L9
}		.long	.L7		movl	8(%ebp), %eax
return c;	.L3:				addl	\$2, %eax
}		movl	8(%ebp), %eax	.L2:	movl	%eax, -4(%ebp)
		movl	%eax, -4(%ebp)		jmp	.L9
		jmp	.L9	.L9:	movl	\$0, -4(%ebp)

Case Blocks

```
int myfunc(int a,int b) {
    int c;
    switch(a) {
        case 10 : c=a; break;
        case 11: b++;
        case 12:
        case 13: c=a+b; break;
        case 14: c=a*b; break;
        case 15: c=a+2; break;
        default: c=0;
    }
    return c;
}
```



Switch “Jump Table”

```
int myfunc(int a,int b) {
    int c;
    switch(a) {
        case 10 : c=a; break;
        case 11: b++;
        case 12:
        case 13: c=a+b; break;
        case 14: c=a*b; break;
        case 15: c=a+2; break;
        default: c=0;
    }
    return c;
}
```

	movl	8(%ebp), %eax	.L4:		
	subl	\$10, %eax		addl	\$1, 12(%ebp)
	cmpl	\$5, %eax	.L5:		
	ja	.L2		movl	12(%ebp), %eax
	movl	.L8(,%eax,4),		movl	8(%ebp), %edx
%eax	jmp	*%eax		leal	(%edx,%eax), %eax
				movl	%eax, -4(%ebp)
				jmp	.L9
	.L8:		.L6:		
	.long	.L3		movl	8(%ebp), %eax
	.long	.L4		imull	12(%ebp), %eax
	.long	.L5		movl	%eax, -4(%ebp)
	.long	.L5	.L7:	jmp	.L9
	.long	.L6			
	.long	.L7		movl	8(%ebp), %eax
.L3:				addl	\$2, %eax
	movl	8(%ebp), %eax	.L2:	movl	%eax, -4(%ebp)
	movl	%eax, -4(%ebp)		jmp	.L9
	jmp	.L9	.L9:	movl	\$0, -4(%ebp)

L8 @ x080484d0 – “Jump Table”

Location	Label	Index	“a” value	Value	Points to
x080484d0	.L3	0	10	x080483ae	case 10
x080484d4	.L4	1	11	x080483b6	case 11
x080484d8	.L5	2	12	x080483ba	case 13
x080484dc	.L5	3	13	x080483ba	case 13
x080484e0	.L6	4	14	x080483c8	case 14
x080484e4	.L7	5	15	x080483d4	case 15

Evaluating the Switch

```
int myfunc(int a,int b) {
    int c;
    switch(a) {
        case 10 : c=a; break;
        case 11: b++;
        case 12:
        case 13: c=a+b; break;
        case 14: c=a*b; break;
        case 15: c=a+2; break;
        default: c=0;
    }
    return c;
}
```

movl	8(%ebp), %eax	.L4:		
subl	\$10, %eax		addl	\$1, 12(%ebp)
cmpl	\$5, %eax	.L5:		
ja	.L2		movl	12(%ebp), %eax
movl	.L8(,%eax,4), %eax		movl	8(%ebp), %edx
jmp	*%eax		leal	(%edx,%eax), %eax
			movl	%eax, -4(%ebp)
			jmp	.L9
.L8:				
.long	.L3	.L6:	movl	8(%ebp), %eax
.long	.L4		imull	12(%ebp), %eax
.long	.L5		movl	%eax, -4(%ebp)
.long	.L5		jmp	.L9
.long	.L6	.L7:		
.long	.L7		movl	8(%ebp), %eax
			addl	\$2, %eax
.L3:			movl	%eax, -4(%ebp)
movl	8(%ebp), %eax		jmp	.L9
movl	%eax, -4(%ebp)	.L2:		
jmp	.L9		movl	\$0, -4(%ebp)
		.L9:		

Indirect Jump

`jmp *<reference>`

- `<reference>` : A location/register that contains the target
- May also see table addressing notation: `jmp *.L4(,%eax,4)`
 - Use `%eax` as index into a table starting at `L4`
 - Each entry in the table is 4 bytes wide
 - Use the `%eaxth` entry as the target for the jump