

# Compiling Loops

C/Java code:

```
while ( sum != 0 ) {  
    <loop body>  
}
```

Machine code:

```
loopTop:    cmpl    $0, %eax  
            je      loopDone  
            <loop body code>  
            jmp     loopTop  
loopDone:
```

- **How to compile other loops should be straightforward**
  - The only slightly tricky part is to be sure where the conditional branch occurs: top or bottom of the loop
- **How would `for(i=0; i<100; i++)` be implemented?**

# “Do-While” Loop Example

## C Code

```
int fact_do(int x)
{
    int result = 1;
    do {
        result *= x;
        x = x-1;
    } while (x > 1);
    return result;
}
```

## Goto Version

```
int fact_goto(int x)
{
    int result = 1;
loop:
    result *= x;
    x = x-1;
    if (x > 1) goto loop;
    return result;
}
```

- Use backward branch to continue looping
- Only take branch when “while” condition holds

# “Do-While” Loop Compilation

## Goto Version

```
int
fact_goto(int x)
{
    int result = 1;

loop:
    result *= x;
    x = x-1;
    if (x > 1)
        goto loop;

    return result;
}
```

## Assembly

```
fact_goto:
    pushl %ebp                # Setup
    movl %esp,%ebp           # Setup
    movl $1,%eax              # eax = 1
    movl 8(%ebp),%edx          # edx = x

.L11:
    imull %edx,%eax           # result *= x
    decl %edx                  # x--
    cmpl $1,%edx               # Compare x : 1
    jg .L11                   # if > goto loop

    movl %ebp,%esp            # Finish
    popl %ebp                 # Finish
    ret                       # Finish
```

### Registers:

%edx	x
%eax	result

# General “Do-While” Translation

## C Code

```
do  
    Body  
while (Test) ;
```

## Goto Version

```
loop:  
    Body  
    if (Test)  
        goto loop
```

■ *Body*: {  
    *Statement*<sub>1</sub>;  
    *Statement*<sub>2</sub>;  
    ...  
    *Statement*<sub>*n*</sub>;  
}

■ *Test* returns integer  
    = 0 interpreted as false  
    ≠ 0 interpreted as true

# “While” Loop Translation

## C Code

```
int fact_while(int x)
{
    int result = 1;
    while (x > 1) {
        result *= x;
        x = x-1;
    };
    return result;
}
```

## Goto Version

```
int fact_while_goto(int x)
{
    int result = 1;
    goto middle;
loop:
    result *= x;
    x = x-1;
middle:
    if (x > 1)
        goto loop;
    return result;
}
```

- Used by GCC for both IA32 & x86-64
- First iteration jumps over body computation within loop straight to test

# “While” Loop Example

```
int fact_while(int x)
{
    int result = 1;
    while (x > 1) {
        result *= x;
        x--;
    };
    return result;
}
```

```
# x in %edx, result in %eax
    jmp     .L34          # goto Middle
.L35:                # Loop:
    imull   %edx, %eax    # result *= x
    decl    %edx          # x--
.L34:                # Middle:
    cmpl    $1, %edx      # x:1
    jg      .L35          # if >, goto
                        # Loop
```

# “For” Loop Example: Square-and-Multiply

```

/* Compute x raised to nonnegative power p */
int ipwr_for(int x, unsigned int p)
{
    int result;
    for (result = 1; p != 0; p = p>>1) {
        if (p & 0x1)
            result *= x;
        x = x*x;
    }
    return result;
}

```

## ■ Algorithm

- Exploit bit representation:  $p = p_0 + 2p_1 + 2^2p_2 + \dots + 2^{n-1}p_{n-1}$
- Gives:  $x^p = z_0 \cdot z_1^2 \cdot (z_2^2)^2 \cdot \dots \cdot \underbrace{(\dots((z_{n-1}^2)^2)\dots)^2}_{n-1 \text{ times}}$ 
  - $z_i = 1$  when  $p_i = 0$
  - $z_i = x$  when  $p_i = 1$
- Complexity  $O(\log p)$

### Example

$$\begin{aligned}
 3^{10} &= 3^2 * 3^8 \\
 &= 3^2 * ((3^2)^2)^2
 \end{aligned}$$

# ipwr Computation

```

/* Compute x raised to nonnegative power p */
int ipwr_for(int x, unsigned int p)
{
    int result;
    for (result = 1; p != 0; p = p>>1) {
        if (p & 0x1)
            result *= x;
        x = x*x;
    }
    return result;
}

```

before iteration	result	x=3	p=10
1	1	3	10=1010 <sub>2</sub>
2	1	9	5= 101 <sub>2</sub>
3	9	81	2= 10 <sub>2</sub>
4	9	6561	1= 1 <sub>2</sub>
5	59049	43046721	0 <sub>2</sub>



# “For” Loop Example

```
int result;  
for (result = 1; p != 0; p = p>>1)  
{  
    if (p & 0x1)  
        result *= x;  
    x = x*x;  
}
```

## General Form

```
for (Init; Test; Update)  
    Body
```

*Init*

```
result = 1
```

*Test*

```
p != 0
```

*Update*

```
p = p >> 1
```

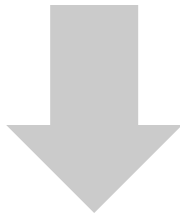
*Body*

```
{  
    if (p & 0x1)  
        result *= x;  
    x = x*x;  
}
```

# “For” → “While”

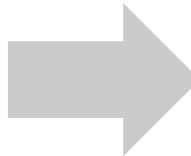
## For Version

```
for (Init; Test; Update )  
    Body
```



## While Version

```
Init ;  
while (Test) {  
    Body  
    Update ;  
}
```



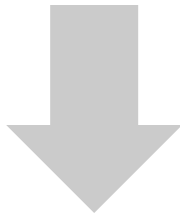
## Goto Version

```
Init ;  
    goto middle ;  
loop:  
    Body  
    Update ;  
middle:  
    if (Test)  
        goto loop ;  
done:
```

# For-Loop: Compilation

For Version

```
for (Init; Test; Update )
    Body
```



Goto Version

```
Init;
goto middle;
loop:
    Body
    Update ;
middle:
    if (Test)
        goto loop;
done:
```

```
for (result = 1; p != 0; p = p>>1)
{
    if (p & 0x1)
        result *= x;
    x = x*x;
}
```



```
    result = 1;
goto middle;
loop:
    if (p & 0x1)
        result *= x;
    x = x*x;
    p = p >> 1;
middle:
    if (p != 0)
        goto loop;
done:
```