



# Machine-Level Programming II: Control - loops

These slides adapted from materials provided by the textbook

# Machine-Level Programming II: Control

- Control: Condition codes
- Conditional branches
- Loops
- Switch Statements

# “Do-While” Loop Example

C Code

```
long  
pcount_do(unsigned long x)  
{  
    long result = 0;  
    do {  
        result += x & 0x1;  
        x >>= 1;  
    } while (x);  
    return result;  
}
```

Goto Version

```
long  
pcount_goto(unsigned long x)  
{  
    long result = 0;  
    loop:  
        result += x & 0x1;  
        x >>= 1;  
        if(x) goto loop;  
    return result;  
}
```

- Count number of 1's in argument **x** (“popcount”)
- Use conditional branch to either continue looping or to exit loop

# “Do-While” Loop Compilation

Goto Version

```
long pcount_goto(unsigned long x)
{
    long result = 0;
loop:
    result += x & 0x1;
    x >>= 1;
    if(x) goto loop;
    return result;
}
```

Register	Use(s)
%rdi	Argument x
%rax	result

```
        movl    $0, %eax      # result = 0
.L2:                           # loop:
        movq    %rdi, %rdx
        andl    $1, %edx      # t = x & 0x1
        addq    %rdx, %rax    # result += t
        shrq    %rdi          # x >>= 1
        jne     .L2          # if (x) goto loop
        rep; ret
```

# 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>;  
}

# General “While” Translation #1

- “Jump-to-middle” translation
- Used with -Og

While version

```
while (Test)
    Body
```



Goto Version

```
goto test;
loop:
    Body
test:
    if (Test)
        goto loop;
done:
```

# While Loop Example #1

C Code

```
long  
pcount_while(unsigned long x)  
{  
    long result = 0;  
    while (x) {  
        result += x & 0x1;  
        x >>= 1;  
    }  
    return result;  
}
```

Jump to Middle

```
long  
pcount_goto_jtm(unsigned long x)  
{  
    long result = 0;  
    goto test;  
loop:  
    result += x & 0x1;  
    x >>= 1;  
test:  
    if(x) goto loop;  
    return result;  
}
```

- Compare to do-while version of function
- Initial goto starts loop at test

# General “While” Translation #2

While version

```
while (Test)  
    Body
```

- “Do-while” conversion
- Used with -O1

Do-While Version

```
if (!Test)  
    goto done;  
do  
    Body  
    while (Test);  
done:
```

Goto Version

```
if (!Test)  
    goto done;  
loop:  
    Body  
    if (Test)  
        goto loop;  
done:
```

# While Loop Example #2

C Code

```
Long  
pcount_while(unsigned long x)  
{  
    long result = 0;  
    while (x) {  
        result += x & 0x1;  
        x >>= 1;  
    }  
    return result;  
}
```

Do-While Version

```
Long  
pcount_goto_dw(unsigned long x)  
{  
    long result = 0;  
    if (!x) goto done;  
loop:  
    result += x & 0x1;  
    x >>= 1;  
    if(x) goto loop;  
done:  
    return result;  
}
```

- Compare to do-while version of function
- Initial conditional guards entrance to loop

# “For” Loop Form

General Form

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

```
#define WSIZE 8*sizeof(int)
long
pcount_for(unsigned long x)
{
    size_t i;
    long result = 0;
    for (i = 0; i < WSIZE; i++)
    {
        unsigned bit =
            (x >> i) & 0x1;
        result += bit;
    }
    return result;
}
```

Init

```
i = 0
```

Test

```
i < WSIZE
```

Update

```
i++
```

Body

```
{
    unsigned bit =
        (x >> i) & 0x1;
    result += bit;
}
```

# “For” Loop → While Loop

For Version

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



While Version

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

# For-While Conversion

Init

```
i = 0
```

Test

```
i < WSIZE
```

Update

```
i++
```

Body

```
{  
    unsigned bit =  
        (x >> i) & 0x1;  
    result += bit;  
}
```

```
Long  
pcount_for_while(unsigned long x)  
{  
    size_t i;  
    long result = 0;  
    i = 0;  
    while (i < WSIZE)  
    {  
        unsigned bit =  
            (x >> i) & 0x1;  
        result += bit;  
        i++;  
    }  
    return result;  
}
```

# “For” Loop Do-While Conversion

C Code

Goto Version

```
long  
pcount_for(unsigned long x)  
{  
    size_t i;  
    long result = 0;  
    for (i = 0; i < WSIZE; i++)  
    {  
        unsigned bit =  
            (x >> i) & 0x1;  
        result += bit;  
    }  
    return result;  
}
```

```
Long  
pcount_for_godw(unsigned long x)  
{  
    size_t i;  
    long result = 0;  
    i = 0;  
    if (!(i < WSIZE))  
        goto done;  
loop:  
{  
    unsigned bit =  
        (x >> i) & 0x1;  
    result += bit;  
}  
    i++;  
    if (i < WSIZE)  
        goto loop;  
done:  
    return result;  
}
```

Annotations:

- Init*: A light blue box containing the code `i = 0;`
- ! Test*: A light blue box containing the code `if (!(i < WSIZE))` and `goto done;`
- Body*: A light blue box containing the code `unsigned bit = (x >> i) & 0x1;` and `result += bit;`
- Update*: A light blue box containing the code `i++;`
- Test*: A light blue box containing the code `if (i < WSIZE)` and `goto loop;`

- Initial test can be optimized away

# Reverse Engineering Assembly

```
.text:00400ea9
.text:00400ea9
.text:00400ea9 55
.text:00400eaa 53
.text:00400eab 4883ec28
.text:00400eaf 64488b042528000000
.text:00400eb8 4889442418
.text:00400eb8 31c0
.text:00400ebf 4889e6
.text:00400ec2 e806070000
.text:00400ec7 833c2401
.text:00400ecb 7405
.text:00400ecd e8c5060000
.text:00400ed2
.text:00400ed2 4889e3
.text:00400ed5 488d6c2414
.text:00400eda
.text:00400eda 8b03
.text:00400edc 01c0
.text:00400ede 394304
.text:00400ee1 7405
.text:00400ee3 e8af060000
.text:00400ee8
.text:00400ee8 4883c304
.text:00400eec 4839eb
.text:00400eef 75e9
.text:00400ef1 488b442418
.text:00400ef6 644833042528000000
.text:00400eff 7405
.text:00400f01 e8fafbffff
.text:00400f06
.text:00400f06 4883c428
.text:00400f0a 5b
.text:00400f0b 5d
.text:00400f0c c3
+---.00100fa0
```

*unknown phase\_2 (unknown)*

```
push %rbp
push %rbx
sub $0x28,%rsp
mov %fs:0x28,%rax
mov %rax,0x18(%rsp)
xor %eax,%eax
mov %rsp,%rsi
callq read_six_numbers
cmpl $0x1,(%rsp)
je loc_00400ed2
callq explode_bomb

loc_00400ed2:
mov %rsp,%rbx
lea 0x14(%rsp),%rbp

loc_00400eda:
mov (%rbx),%eax
add %eax,%eax
cmp %eax,0x4(%rbx)
je loc_00400ee8
callq explode_bomb

loc_00400ee8:
add $0x4,%rbx
cmp %rbp,%rbx
jne loc_00400eda
mov 0x18(%rsp),%rax
xor %fs:0x28,%rax
je loc_00400f06
callq func_00400b00

loc_00400f06:
add $0x28,%rsp
pop %rbx
pop %rbp
retq
```

**When translating assembly to C, it's useful to draw arrows to “spot the if/loop”**

**Forward branches are if/then/else**

**Backwards are loops**

# Even easier with <https://onlinedisassembler.com>

