Machine-Level Programming: Loops

CMPU 224 – Computer Organization
Jason Waterman
Looping in C

• There are three looping constructs in C
  • while, do-while, and for loops
• Turn loops into goto construct to implement

```c
while(test) {
    body;
}
do {
    body;
} while(test);
for (init; test; update;){
    body;
}
```
Looping: “do while” example

• Count number of 1’s in argument $x$

• Use conditional branch to either continue looping or to exit loop

**Goto Version**

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

**C Goto Version**

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

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Synonym</th>
<th>Jump condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>jmp Label</td>
<td></td>
<td>1</td>
<td>Direct jump</td>
</tr>
<tr>
<td>je Label</td>
<td>jz</td>
<td>ZF</td>
<td>Equal / zero</td>
</tr>
<tr>
<td>jne Label</td>
<td>jnz</td>
<td>~ZF</td>
<td>Not Equal / not zero</td>
</tr>
<tr>
<td>js Label</td>
<td></td>
<td>SF</td>
<td>Negative</td>
</tr>
<tr>
<td>jns Label</td>
<td></td>
<td>~SF</td>
<td>Nonnegative</td>
</tr>
<tr>
<td>jg Label</td>
<td>jnle</td>
<td>~(SF ^ OF) &amp; ~ZF</td>
<td>Greater (signed &gt; )</td>
</tr>
<tr>
<td>jge Label</td>
<td>jnl</td>
<td>~(SF ^ OF)</td>
<td>Greater or equal (signed &gt;=)</td>
</tr>
<tr>
<td>jl Label</td>
<td>jnge</td>
<td>SF ^ OF</td>
<td>Less (signed &lt;)</td>
</tr>
<tr>
<td>jle Label</td>
<td>jng</td>
<td>(SF ^ OF)</td>
<td>Less or equal (signed &lt;=)</td>
</tr>
<tr>
<td>ja Label</td>
<td>jnbe</td>
<td>~CF &amp; ~ZF</td>
<td>Above (unsigned &gt;)</td>
</tr>
<tr>
<td>jae Label</td>
<td>jnb</td>
<td>~CF</td>
<td>Above or equal (unsigned &gt;=)</td>
</tr>
<tr>
<td>jb Label</td>
<td>jnae</td>
<td>CF</td>
<td>Below (unsigned &lt;)</td>
</tr>
<tr>
<td>jbe Label</td>
<td>jna</td>
<td>CF</td>
<td>Less or equal (unsigned &lt;=)</td>
</tr>
</tbody>
</table>
“Do-While” Loop Compilation

Goto Version

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Use(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rdi</td>
<td>Argument x</td>
</tr>
<tr>
<td>%rax</td>
<td>result</td>
</tr>
</tbody>
</table>

Code:

```assembly
movl $0, %eax  # result = 0
loop:
    movq %rdi, %rdx
    andl $1, %edx  # t = x & 0x1
    addq %rdx, %rax # result += t
    shrq $1, %rdi  # x >>= 1
    jne loop      # if (x) goto loop
ret
```
General “Do-While” Translation

C Code

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

Goto Version

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

• Body:

```c
{
   Statement_1;
   Statement_2;
   ...
   Statement_n;
}
```
General “While” Translation #1

• “Jump-to-test” translation
• A “do-while” loop with a jump to the first test
• 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

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

Jump to Test Version

```c
long count_goto_jtt(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 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

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

Do-While Version

```c
long count_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;
}
```

• Initial conditional guards the entrance to the loop
While Loop Example – O1

Assembly

count_while:
    testq %rdi, %rdi
    je .L4
    movl $0, %eax
.L3:
    movq %rdi, %rdx
    andl $1, %edx
    addq %rdx, %rax
    shrq %rdi
    jne .L3
    ret
.L4:
    movl $0, %eax
    ret

Do-While Version

long count_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;
}

• Initial conditional guards the entrance to the loop
While Loop Example -Og

Assembly

count_while:
  movl $0, %eax
.L2:
  testq %rdi, %rdi
  je .L4
  movq %rdi, %rdx
  andl $1, %edx
  addq %rdx, %rax
  shrq %rdi
  jmp .L2
.L4:
  ret

C Code

long count_goto(unsigned long x) {
  long result = 0;
  loop:
    if (!x) goto done;
    result += x & 0x1;
    x >>= 1;
    goto loop;
  done:
    return result;
}
While Loop Summary

• Convert to a do-while loop that can be implemented with goto statements

• Jump-to-test
  • Jump directly to the test of the do-while loop

• Do-while conversion
  • Guard the entrance to the do-while loop with a !test
  • If the while condition is not true (!test) jump to the end of the do-while loop
"For" Loop Form

General Form

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

```
define WSIZE 8*sizeof(long)
long count_for(unsigned long x){
  size_t i;
  unsigned bit;
  long result = 0;
  for (i = 0; i < WSIZE; i++){
    bit = (x >> i) & 0x1;
    result += bit;
  }
  return result;
}
```
"For" Loop ➔ While Loop

For Version

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

Body

While Version

```
Init;
while (Test) {
    Body
    Update;
}
```
For-While Conversion

```c
long count_for_while(unsigned long x) {
    size_t i;
    unsigned bit
    long result = 0;
    i = 0;
    while (i < WSIZE) {
        bit = (x >> i) & 0x1;
        result += bit;
        i++;
    }
    return result;
}
```
“For” Loop Do-While Conversion

C Code

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

• Initial test can be optimized away

Goto Version

```c
long count_for_goto_dw(unsigned long x) {
    size_t i;
    unsigned bit;
    long result = 0;
    i = 0;
    if (!((i < WSIZE))
        goto done;
loop:
{
    bit = (x >> i) & 0x1;
    result += bit;
}
i++;
if (i < WSIZE)
    goto loop;
done:
    return result;
}
```
“For” Loop Do-While Conversion

Assembly

```
count_for:
    movl $0, %edx
    movl $0, %ecx
.L7:
    movq %rdi, %rax
    shrq %cl, %rax
    andl $1, %eax
    addq %rax, %rdx
    addq $1, %rcx
    cmpq $64, %rcx
    jne .L7
    movq %rdx, %rax
    ret
```

- Initial test can be optimized away

Goto Version

```
long count_for_goto_dw(unsigned long x) {
    size_t i;
    unsigned bit;
    long result = 0;
    i = 0;  // Init
    if (!(i < WSIZE))
        goto done;
    loop:
    {
        bit = (x >> i) & 0x1;  // Test
        result += bit;
    }
    i++;  // Update
    if (i < WSIZE)
        goto loop;
    done:
    return result;
}
```
For Loop Summary

• Convert to a while loop of the following form:
  
  Init;
  While(test)
      Body;
      Update;
  
• Convert while loop to a do-while

• Use goto statements to implement the do-while loop
Summary

• C Control
  • if-then-else
  • do-while
  • while, for
  • switch

• Assembler Control
  • Conditional jump
  • Conditional move
  • Indirect jump (via jump tables)
  • Compiler generates code sequence to implement more complex control

• Standard Techniques
  • Loops converted to do-while or jump-to-test form
  • Large switch statements use jump tables
  • Sparse switch statements may use decision trees (if-elseif-elseif-else)