COMP201

Computer Systems & Programming

Lecture #23 – More Control Flow

Aykut Erdem // Koç University // Fall 2020
Good news, everyone!

• Midterm exam will be held on December 5, 2020 (Saturday) at 11:45am
• Check out the midterm guide for more information.
Recap

• Control Flow Mechanics
  • Condition Codes
  • Assembly Instructions

• If statements
Plan for Today

- If statements (cont’d.)
- Loops
- Other Instructions That Depend On Condition Codes

Disclaimer: Slides for this lecture were borrowed from
—Nick Troccoli's Stanford CS107 class
Lecture Plan

• If statements (cont’d.)
• Loops
• Other Instructions That Depend On Condition Codes
Practice: Fill In The Blank

If-Else In C
if (arg > 3) {
    ret = 10;
} else {
    ret = 0;
}
ret++;

If-Else In Assembly pseudocode
Test
Jump to else-body if test fails
If-body
Jump to past else-body
Else-body
Past else body
Practice: Fill In The Blank

If-Else In C

if ( __________ ) {
    __________;
} else {
    __________;
}

_______;

If-Else In Assembly pseudocode

Test
Jump to else-body if test fails
If-body
Jump to past else-body
Else-body
Past else body
Practice: Fill In The Blank

If-Else In C

```c
if ( arg > 3 ) {
    ret = 10;
} else {
    ret = 0;
}
ret++;```

If-Else In Assembly pseudocode

```
400552 <+0>:  cmp    $0x3,%edi
400555 <+3>:  jle    0x40055e <if_else+12>
400557 <+5>:  mov    $0xa,%eax
40055c <+10>: jmp    0x40055e <if_else+12>
40055e <+12>: mov    $0x0,%eax
400563 <+17>: add    $0x1,%eax
```

Test

Jump to else-body if test **fails**

If-body

Jump to past else-body

Else-body

Past else body
Lecture Plan

• If statements (cont’d.)
• Loops
  • While loops
  • For loops
• Other Instructions That Depend On Condition Codes
Loops and Control Flow

void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}

Set %eax (i) to 0.
Loops and Control Flow

```c
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

Jump to another instruction.
Loops and Control Flow

```c
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

Compare `%eax` (i) to 0x63 (99) by calculating `%eax - 0x63`. This is 0 - 99 = -99, so it sets the Sign Flag to 1.
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}

jle means “jump if less than or equal”. This jumps if %eax <= 0x63. The flags indicate this is true, so we jump.
Loops and Control Flow

```c
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

Add 1 to \%eax (i).
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}

Compare %eax (i) to 0x63 (99) by calculating %eax – 0x63. This is 1 – 99 = -98, so it sets the Sign Flag to 1.
Loops and Control Flow

```c
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

```
mov $0x0,%eax
jmp 0x40057a <loop+10>
add $0x1,%eax
cmp $0x63,%eax
jle 0x400577 <loop+7>
repz retq
```

**jle** means “jump if less than or equal”. This jumps if %eax <= 0x63. The flags indicate this is true, so we jump.
Loops and Control Flow

```c
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

We continue in this pattern until we do not make this conditional jump. When will that be?
Loops and Control Flow

```c
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

We will stop looping when this comparison says that `%eax - 0x63 > 0`!
Loops and Control Flow

void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}

Then, we return from the function.
Common While Loop Construction

C
while (test) {
    body
}

Assembly
Jump to test
Body
Test
Jump to body if success

From Previous Slide:

```
0x00000000000400570 <+0>:    mov   $0x0,%eax
0x00000000000400575 <+5>:    jmp   0x40057a <loop+10>
0x00000000000400577 <+7>:    add   $0x1,%eax
0x0000000000040057a <+10>:    cmp   $0x63,%eax
0x0000000000040057d <+13>:    jle    0x400577 <loop+7>
0x0000000000040057f <+15>:    repz   retq
```
Lecture Plan

• Loops
  • While loops
  • For loops

• Other Instructions That Depend On Condition Codes
Common While Loop Construction

C For loop
for (init; test; update) {
    body
}

Assembly pseudocode
Init
Jump to test
Body
Update
Test
Jump to body if success

C Equivalent While Loop
init
while(test) {
    body
    update
}
```c
int sum_array(int arr[], int nelems) {
    int sum = 0;
    for (int i = 0; i < nelems; i++) {
        sum += arr[i];
    }
    return sum;
}
```

1. Which register is C code's `sum`?
2. Which register is C code's `i`?
3. Which assembly instruction is C code's `sum += arr[i]`?
4. What are the `cmp` and `jl` instructions doing? (`jl`: jump less; signed `<`)
Lecture Plan

• Loops
• Other Instructions That Depend On Condition Codes
Condition Code-Dependent Instructions

There are three common instruction types that use condition codes:

• **jmp** instructions conditionally jump to a different next instruction
• **set** instructions conditionally set a byte to 0 or 1
• new versions of **mov** instructions conditionally move data
set: Read condition codes

**set** instructions conditionally set a byte to 0 or 1.

- Reads current state of flags
- Destination is a single-byte register (e.g., `%al`) or single-byte memory location
- Does not perturb other bytes of register
- Typically followed by `movzbl` to zero those bytes

```c
int small(int x) {
    return x < 16;
}
```

```assembly
    cmp $0xf,%edi
    setle %al
    movzbl %al, %eax
    retq
```
## set: Read condition codes

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Synonym</th>
<th>Set Condition (1 if true, 0 if false)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sete D</td>
<td>setz</td>
<td>Equal / zero</td>
</tr>
<tr>
<td>setne D</td>
<td>setnz</td>
<td>Not equal / not zero</td>
</tr>
<tr>
<td>sets D</td>
<td>setns</td>
<td>Negative</td>
</tr>
<tr>
<td>setns D</td>
<td></td>
<td>Nonnegative</td>
</tr>
<tr>
<td>setg D</td>
<td>setnle</td>
<td>Greater (signed &gt;)</td>
</tr>
<tr>
<td>setge D</td>
<td>setnl</td>
<td>Greater or equal (signed &gt;=)</td>
</tr>
<tr>
<td>setl D</td>
<td>setnge</td>
<td>Less (signed &lt;)</td>
</tr>
<tr>
<td>setle D</td>
<td>setng</td>
<td>Less or equal (signed &lt;=)</td>
</tr>
<tr>
<td>seta D</td>
<td>setnbe</td>
<td>Above (unsigned &gt;)</td>
</tr>
<tr>
<td>setae D</td>
<td>setnb</td>
<td>Above or equal (unsigned &gt;=)</td>
</tr>
<tr>
<td>setb D</td>
<td>setnae</td>
<td>Below (unsigned &lt;)</td>
</tr>
<tr>
<td>setbe D</td>
<td>setna</td>
<td>Below or equal (unsigned &lt;=)</td>
</tr>
</tbody>
</table>
**cmov: Conditional move**

`cmovx src, dst` conditionally moves data in `src` to data in `dst`.
- Mov `src` to `dst` if condition `x` holds; no change otherwise
- `src` is memory address/register, `dst` is register
- May be more efficient than branch (i.e., jump)
- Often seen with C ternary operator: `result = test ? then : else;`

```c
int max(int x, int y) {
    return x > y ? x : y;
}
```

```assembly
cmp   %edi,%esi
mov   %edi, %eax
```

```
cmovge %esi, %eax
retq
```
Ternary Operator

The ternary operator is a shorthand for using if/else to evaluate to a value.

```
    condition ? expressionIfTrue : expressionIfFalse
```

```
int x;
if (argc > 1) {
    x = 50;
} else {
    x = 0;
}

// equivalent to
int x = argc > 1 ? 50 : 0;
```
**cmov: Conditional move**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Synonym</th>
<th>Move Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmove S,R</td>
<td>cmovz</td>
<td>Equal / zero (ZF = 1)</td>
</tr>
<tr>
<td>cmovne S,R</td>
<td>cmovnznz</td>
<td>Not equal / not zero (ZF = 0)</td>
</tr>
<tr>
<td>cmovs S,R</td>
<td>cmovnz</td>
<td>Negative (SF = 1)</td>
</tr>
<tr>
<td>cmovns S,R</td>
<td>cmovns</td>
<td>Nonnegative (SF = 0)</td>
</tr>
<tr>
<td>cmovg S,R</td>
<td>cmovnle</td>
<td>Greater (signed &gt;) (SF = 0 and SF = OF)</td>
</tr>
<tr>
<td>cmovge S,R</td>
<td>cmovnle</td>
<td>Greater or equal (signed &gt;=) (SF = OF)</td>
</tr>
<tr>
<td>cmovl S,R</td>
<td>cmovnge</td>
<td>Less (signed &lt;) (SF != OF)</td>
</tr>
<tr>
<td>cmovle S,R</td>
<td>cmovnle</td>
<td>Less or equal (signed &lt;=) (ZF = 1 or SF! = OF)</td>
</tr>
<tr>
<td>cmova S,R</td>
<td>cmovnbe</td>
<td>Above (unsigned &gt;) (CF = 0 and ZF = 0)</td>
</tr>
<tr>
<td>cmovae S,R</td>
<td>cmovnbe</td>
<td>Above or equal (unsigned &gt;=) (CF = 0)</td>
</tr>
<tr>
<td>cmovb S,R</td>
<td>cmovnae</td>
<td>Below (unsigned &lt;) (CF = 1)</td>
</tr>
<tr>
<td>cmovbe S,R</td>
<td>cmovnae</td>
<td>Below or equal (unsigned &lt;=) (CF = 1 or ZF = 1)</td>
</tr>
</tbody>
</table>
Practice: Conditional Move

```c
int signed_division(int x) {
    return x / 4;
}
```

signed_division:

```
leal 3(%rdi), %eax  # Put x + 3 into %eax (add appropriate bias, 2^2-1)
testl %edi, %edi     # To see whether x is negative, zero, or positive
cmovns %edi, %eax   # If x is positive, put x into %eax
sarl $2, %eax       # Divide %eax by 4
ret                 # (See Sec. 2.3.7)
```
Extra Practice
Practice: Fill In The Blank

Note: .L2/.L3 are "labels" that make jumps easier to read.

C Code

long loop(long a, long b) {
    long result = _______
    while (__________) {
        result = ____________;
        a = ____________;
    }
    return result;
}

What does this assembly code translate to?

// a in %rdi, b in %rsi
loop:
    movl $1, %eax
    jmp .L2
.L3
    leaq (%rdi,%rsi), %rdx
    imulq %rdx, %rax
    addq $1, %rdi
.L2
    cmpq %rsi, %rdi
    jl .L3
    rep; ret

Common while loop construction:
Jump to test
Body
Test
Jump to body if success
Practice: Fill In The Blank

Note: .L2/.L3 are “labels” that make jumps easier to read.

C Code

```c
long loop(long a, long b) {
    long result = ______; // 1
    while (_________) { // a < b
        result = _____________; // result*(a+b)
        a = _________; // a + 1
    }
    return result;
}
```

What does this assembly code translate to?

```asm
// a in %rdi, b in %rsi
loop:
    movl $1, %eax
    jmp .L2
.L3
    leaq (%rdi,%rsi), %rdx
    imulq %rdx, %rax
    addq $1, %rdi
.L2
    cmpq %rsi, %rdi
    jl .L3
rep; ret
```

Common while loop construction:

- Jump to test
- Body
- Test
- Jump to body if success
Practice: “Escape Room”

escapeRoom:
  leal (%rdi,%rdi), %eax
  cmpl $5, %eax
  jg .L3
  cmpl $1, %edi
  jne .L4
  movl $1, %eax
  ret
.L3:
  movl $1, %eax
  ret
.L4:
  movl $0, %eax
  ret

What must be passed to the escapeRoom function such that it returns true (1) and not false (0)?
Practice: “Escape Room”

escapeRoom:
  leal (%rdi,%rdi), %eax
  cmpl $5, %eax
  jg .L3
  cmpl $1, %edi
  jne .L4
  movl $1, %eax
  ret
.L3:
  movl $1, %eax
  ret
.L4:
  movl $0, %eax
  ret

What must be passed to the escapeRoom function such that it returns true (1) and not false (0)?

First param > 2 or == 1.
Recap

• Assembly Execution and %rip
• Control Flow Mechanics
  • Condition Codes
  • Assembly Instructions
• If statements
• Loops
  • While loops
  • For loops
• Other Instructions That Depend On Condition Codes

Next time: Function calls in assembly