DAKOTA STATE UNIVERSITY

Chapter 8-Control Structures

Jihene Kaabi & Youssef Harrath

DAKOTA STATE UNIVERSITY

Translating high-level Structures

- We are used to using high-level structures rather than just branches.
- Therefore, it is useful to know how to translate these structures in Assembly, so that we can just use the same patterns as when writing, say, C code.
  - A compiler does such translation for us.
- Let's start with the most common high-level control structure if-then-else.

DAKOTA STATE UNIVERSITY

If-then-else

- A generic if-then-else construct

```
if (condition) then
    then_block
else
    else_block
```

- Translation into x86 assembly

```
    <instructions to set flags (e.g., cmp, …)>
    jxx     else_block      ; select xx so that branches if condition false
    <code for the then block>
    jmp  endif
else_block:
    <code for the else block>
endif:
```

DAKOTA STATE UNIVERSITY

If-then-else: Example

- Consider the following C code, which assigns the larger of value1 and value2 to bigger. All three variables are declared as integers (int data type).

```
if (value1 > value2)
    bigger = value1;
else
    bigger = value2;
```

- Translation into x86 assembly

```
    mov eax, [value1]
    cmp eax, value2
    jle  else_part
then_part:
    mov eax, [value1]
    mov [bigger], eax
    jmp  end_if
else_part:
    mov eax, [value2]
    mov [bigger], eax
end_if:
```

DAKOTA STATE UNIVERSITY

No Else?

- A generic if-then construct

```
if (condition) then
    then_block
```

- Translation into x86 assembly

```
    <instructions to set flags (e.g., cmp, …)>
    jxx     endif           ; select xx so that branches if condition
                            ; false
    <code for the then block>
endif:
```

DAKOTA STATE UNIVERSITY

For Loop

- Consider the following loop:

```
sum = 0
for (i = 0; i <= 10; i++)
    sum += i;
```

- Translation into x86 assembly

```
        mov eax, 0          ; eax is sum
        mov ebx, 0          ; ebx is i
loop_start:
        cmp ebx, 10         ; compare i and 10
        jg  loop_end        ; if (i>10) goto loop_end
        add eax, ebx        ; sum += i
        inc ebx             ; i++
        jmp loop_start      ; goto loop_start
loop_end:
```

DAKOTA STATE UNIVERSITY

The loop Instructions

- The 80x86 provides several instructions designed to implement for–like loops.
- Each of these instructions takes a code label as its single operand.
- **LOOP** Decrements ECX, if ECX ≠ 0, branches to label
- **LOOPE, LOOPZ** Decrement ECX (FLAGS register is not modied), if ECX ≠ 0 and ZF = 1, branches
- **LOOPNE, LOOPNZ** Decrement ECX (FLAGS unchanged), if ECX ≠ 0 and ZF = 0, branches
- The last two instructions are useful in writing loops for applications that require two termination conditions.

DAKOTA STATE UNIVERSITY

The loop Instruction

- Consider the following loop:

```
sum = 0
for (i = 0; i <= 10; i++)
    sum += i;
```

- The x86 loop instruction requires that
  - The loop index be stored in **ecx**
  - The loop index de decremented
  - The loop exits when the loop index is equal to zero.
- Given this, we really have to think of this loop in reverse

```
sum = 0
for (i = 10; i >0; i--)
    sum += i;
```

- This loop is equivalent to the previous one, but now it can be directly translated to assembly using the loop instruction.

DAKOTA STATE UNIVERSITY

The loop Instruction: Example

- Here is out "reversed" loop:

```
sum = 0
for (i = 10;  i >0;  i--)
    sum += i;
```

- Translation into x86 assembly

```
    mov eax, 0          ; eax is sum
    mov ecx, 10         ; ecx is i
loop_start:
    add eax, ecx        ; sum += i
    loop loop_start     ; if i>0 then goto loop_start
```

DAKOTA STATE UNIVERSITY

While Loop

- A generic while loop

```
while (condition) {
    body
}
```

- Is translated as

```
jmp while_cond
while_body:
        <instructions for while loop body>
while_cond:
        <instructions for while_cond>
end_while:
```

While Loop: Example

- Consider the following while loop:

```
sum = 0;  i = 0;
while (i <= 10){
    sum += i;
     i++;
}
```

- Translation into x86 assembly

```
   mov eax, 0                  ; eax is sum
   mov ebx, 0                  ; ebx is i
   jmp while_cond:             ; goto while_cond
while_body:
       add eax, ebx            ; sum += i
       inc ebx                 ; i++
       jmp while_cond          ; goto while_cond
while_cond:
       cmp ebx, 10             ; compare i and 10
       jle  while_body         ; if (i<=10) goto while_body
end_while:
```

DAKOTA STATE UNIVERSITY

Do While Loop

- A generic do while loop

```
do {
    body
} while  (condition)
```

- Is translated as

```
Dowhile_body:
        <instructions for do while loop body>
        jmp while_cond
while_cond:
        <instructions for while_cond>
end_while:
```

do while Loop: Example

- Consider the following do while loop:

```
sum = 0;  i = 0;
do {
      sum += i;
      i++;
   } while (i <= 10)
```

- Translation into x86 assembly

```
   mov eax, 0                  ; eax is sum
   mov ebx, 0                  ; ebx is i
while_body:
      add eax, ebx             ; sum += i
      inc ebx                  ; i++
      jmp while_cond           ; goto while_cond
while_cond:
      cmp ebx, 10              ; compare i and 10
      jle  while_body          ; if (i<=10) goto while_body
end_while:
```

DAKOTA STATE UNIVERSITY

In-class Exercise

Write an assembly program that prompts the user to enter a number n and checks if it is prime.

Pseudo-code:

```
If n <= 1
    return false
For (i=2 ; i < n ; i++)
     if (n % i == 0)
    return false
return true
```

Sample output:

Enter a number: 29

The number you've entered is prime

Enter a number: 55

The number you've entered is not prime