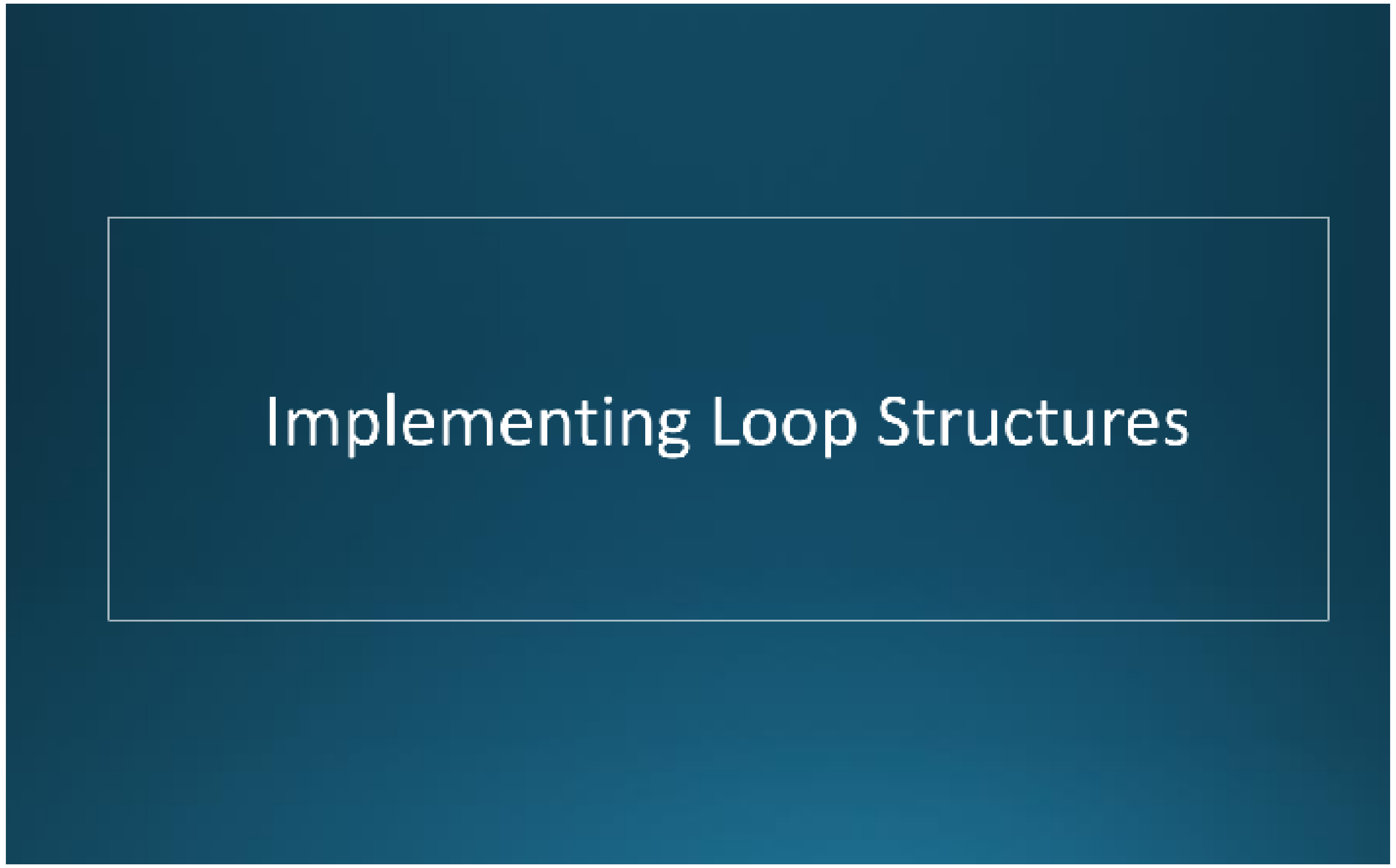
Topics

- Unconditional Jumps
- Conditional jump and compare instructions
- Implementing Loop Structures
- FOR Loops
- Arrays



Implementing Loop Structures

sum of first 5 numbers 1,2,3,4,5

while loop:

```
int i=1,sum=0;
while(i <= 5)
{
    sum=sum+i;
    i=i+1;
}
```

for loop:

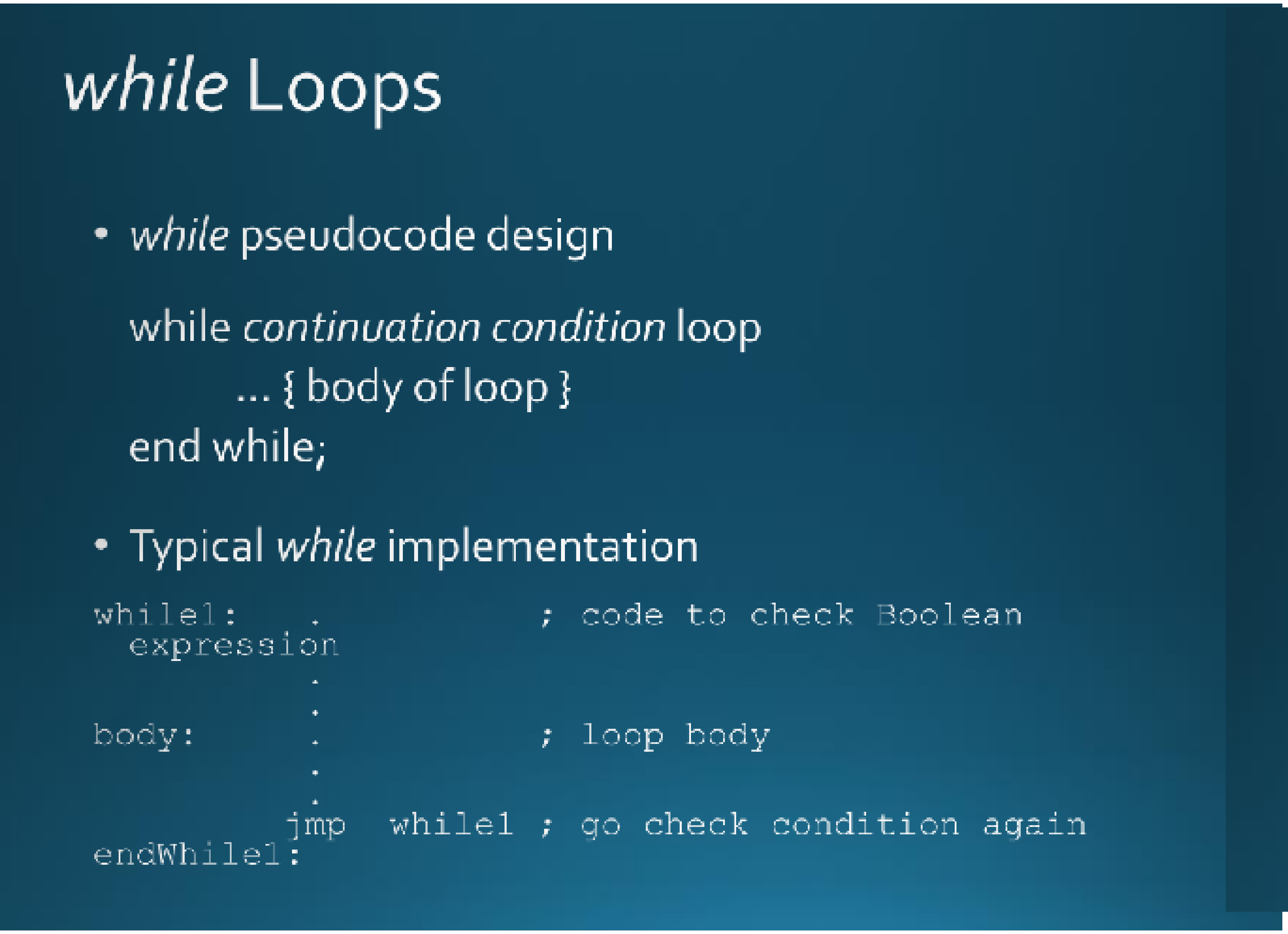
```
int i=1,sum=0;
for(i=1;i<=5;i++)
{
     sum=sum+i;
}
```

```
int i=5,sum=0;
for(i=5; i>0; i--)
{

    sum=sum+i;
 }
```

repeat until loop:

```
int i=1,sum=0;
repeat
{
sum=sum+i;
i=i+1;
}until(i<=5)
```



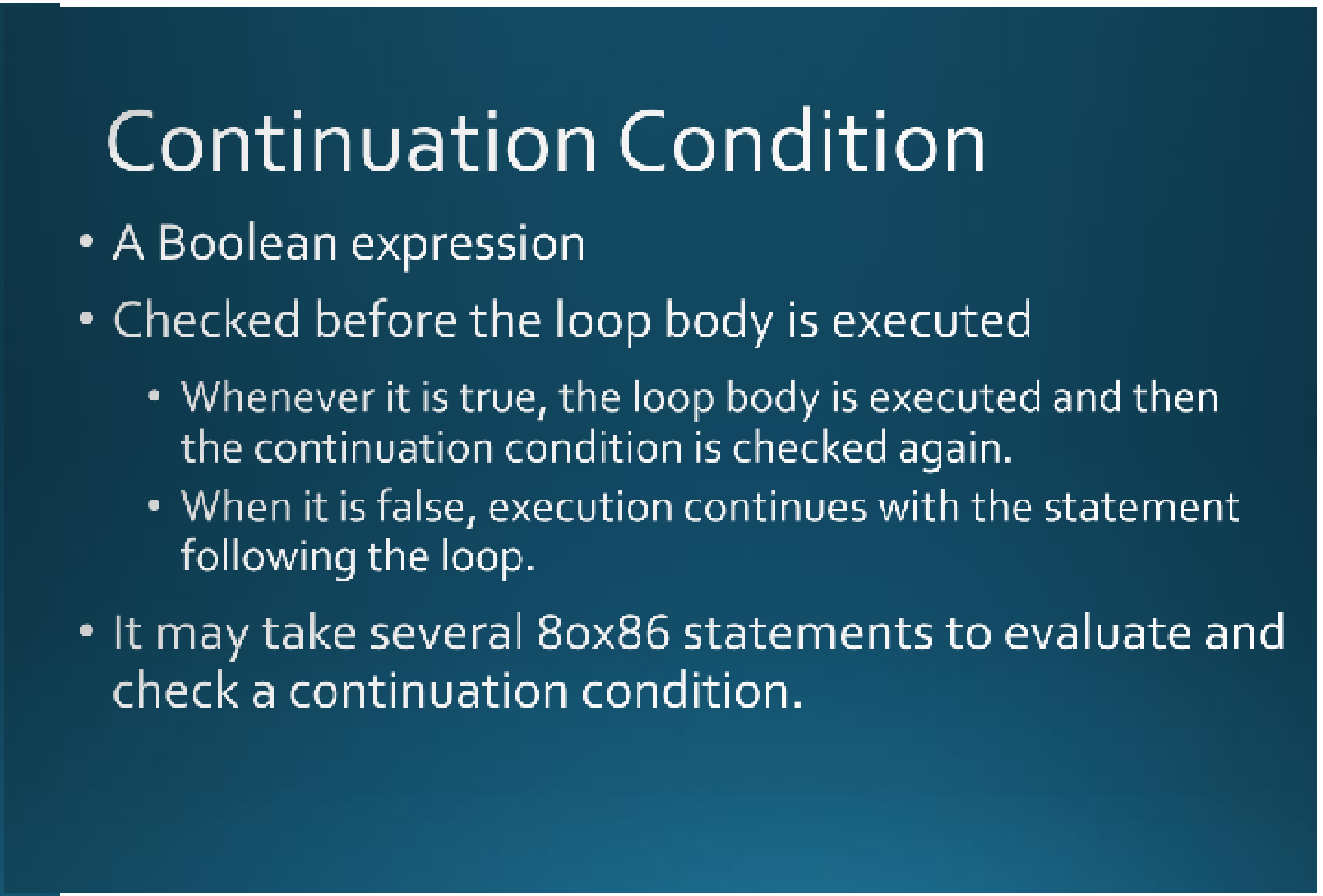
while Loops

- *while* pseudocode design

```
while continuation condition loop
        … { body of loop }
end while;
```

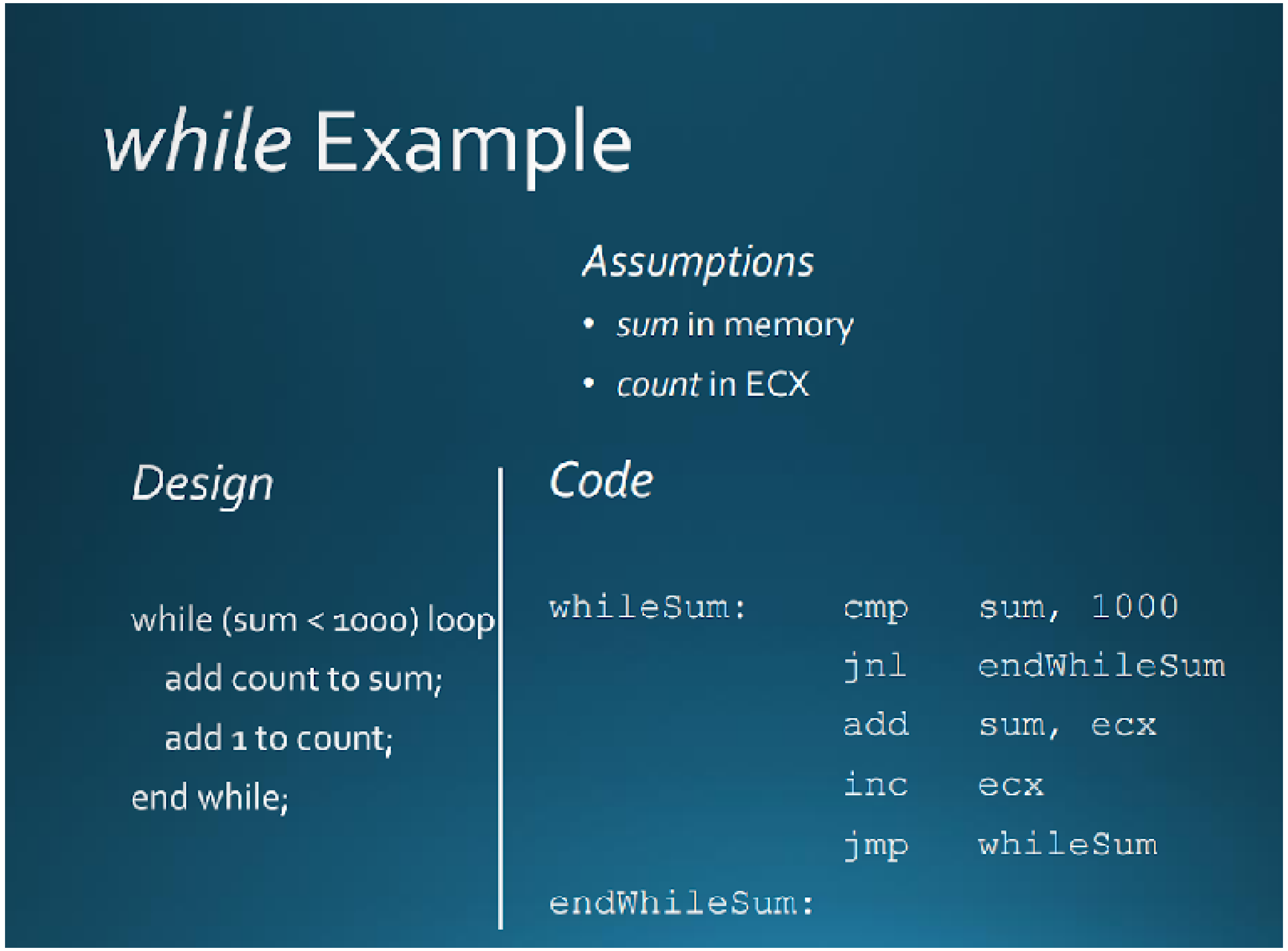
- Typical *while* implementation

```
while1:     .           ; code to check Boolean
  expression
            .
            .
body:       .           ; loop body
            .
            .
          jmp   while1  ; go check condition again
endWhile1:
```



Continuation Condition

- A Boolean expression
- Checked before the loop body is executed
  - Whenever it is true, the loop body is executed and then the continuation condition is checked again.
  - When it is false, execution continues with the statement following the loop.
- It may take several 80x86 statements to evaluate and check a continuation condition.



while Example

Assumptions

- *sum* in memory
- *count* in ECX

Design

```
while (sum < 1000) loop
   add count to sum;
   add 1 to count;
end while;
```

Code

```
whileSum:     cmp     sum, 1000
              jnl     endWhileSum
              add     sum, ecx
              inc     ecx
              jmp     whileSum
endWhileSum:
```

for Loops

- Counter-controlled loop
- *for* pseudocode design

```
for index := initialValue to finalValue loop
   ... { body of loop }
end for;
```

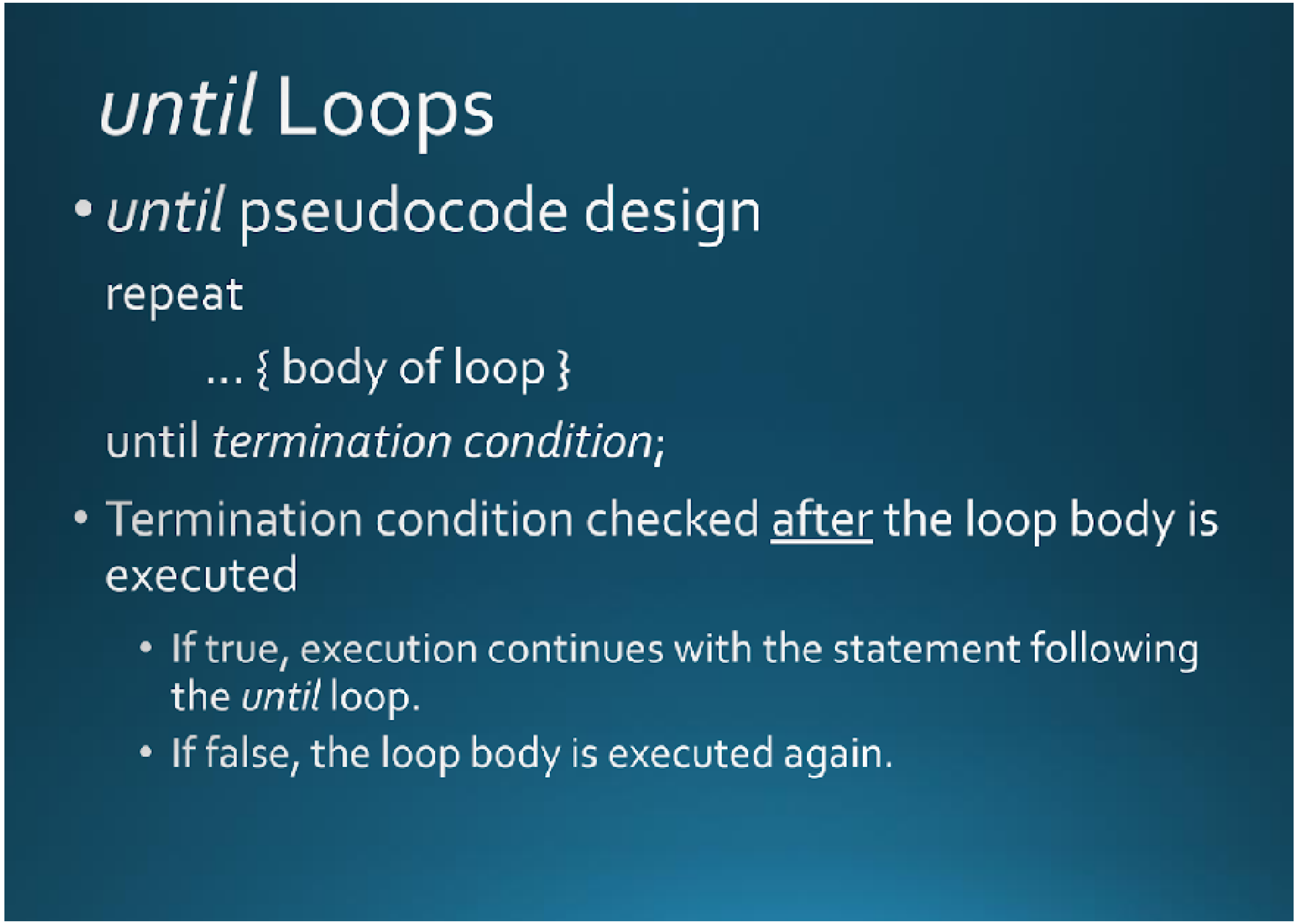
- Loop body executed once for each value of the loop index in the given range

for Implementation

- Convert *for* loop to equivalent *while* loop

```
index := initialValue;
while index ≤ finalValue loop
   ... { body of loop }
   add 1 to index;
end while;
```

- Implement *while* loop in 80x86 code
- Section 5.4 gives another implementation.

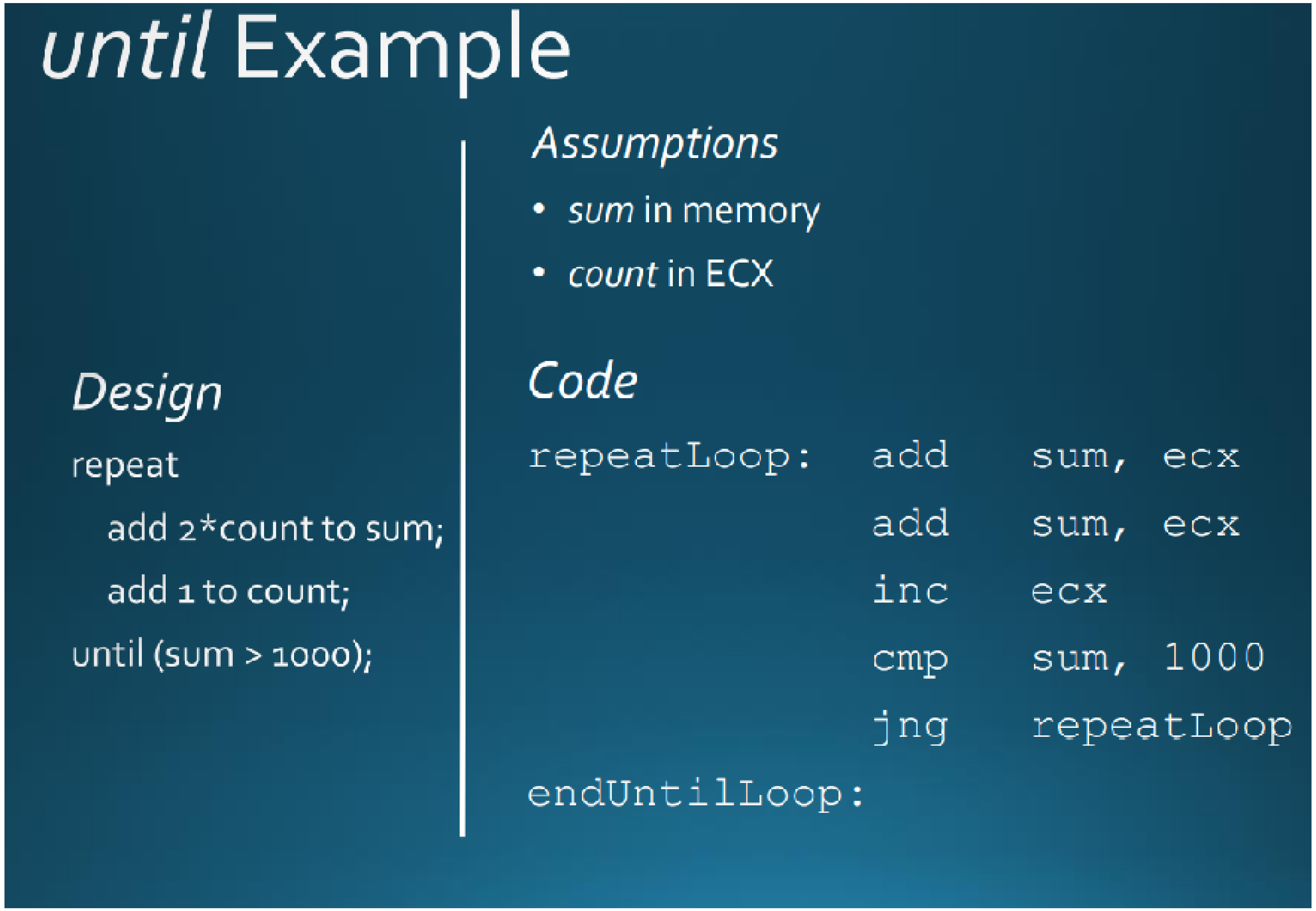


until Loops

- *until* pseudocode design

```
repeat
        ... { body of loop }
until termination condition;
```

- Termination condition checked <u>after</u> the loop body is executed
  - If true, execution continues with the statement following the *until* loop.
  - If false, the loop body is executed again.



until Example

Assumptions

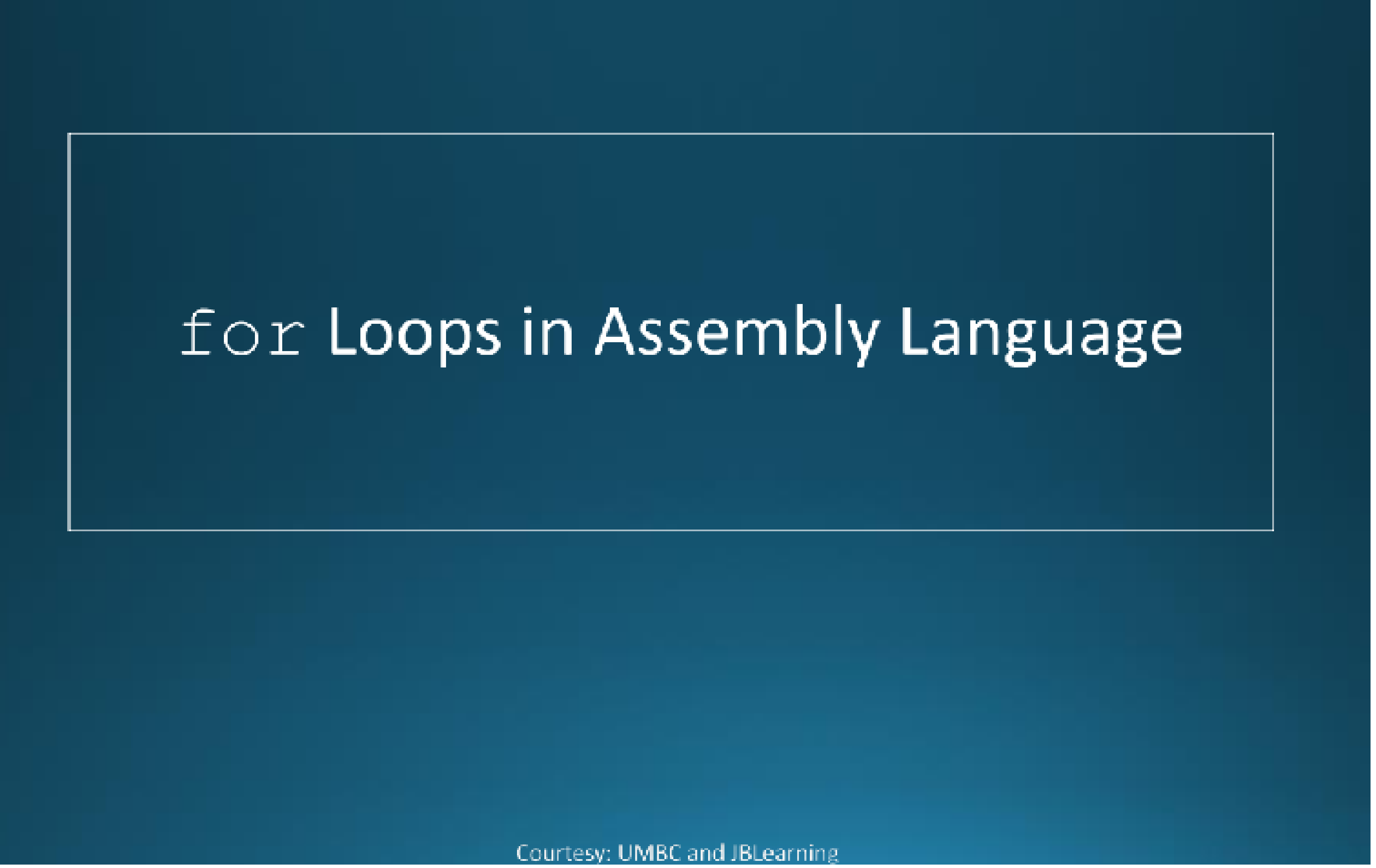
- *sum* in memory
- *count* in ECX

Design

```
repeat
   add 2*count to sum;
   add 1 to count;
until (sum > 1000);
```

Code

```
repeatLoop:   add    sum, ecx
              add    sum, ecx
              inc    ecx
              cmp    sum, 1000
              jng    repeatLoop
endUntilLoop:
```



`for` Loops in Assembly Language

Courtesy: UMBC and JBLearning

we can implement for loops in assembly in two ways
1> is to convert into the while loop format and use it

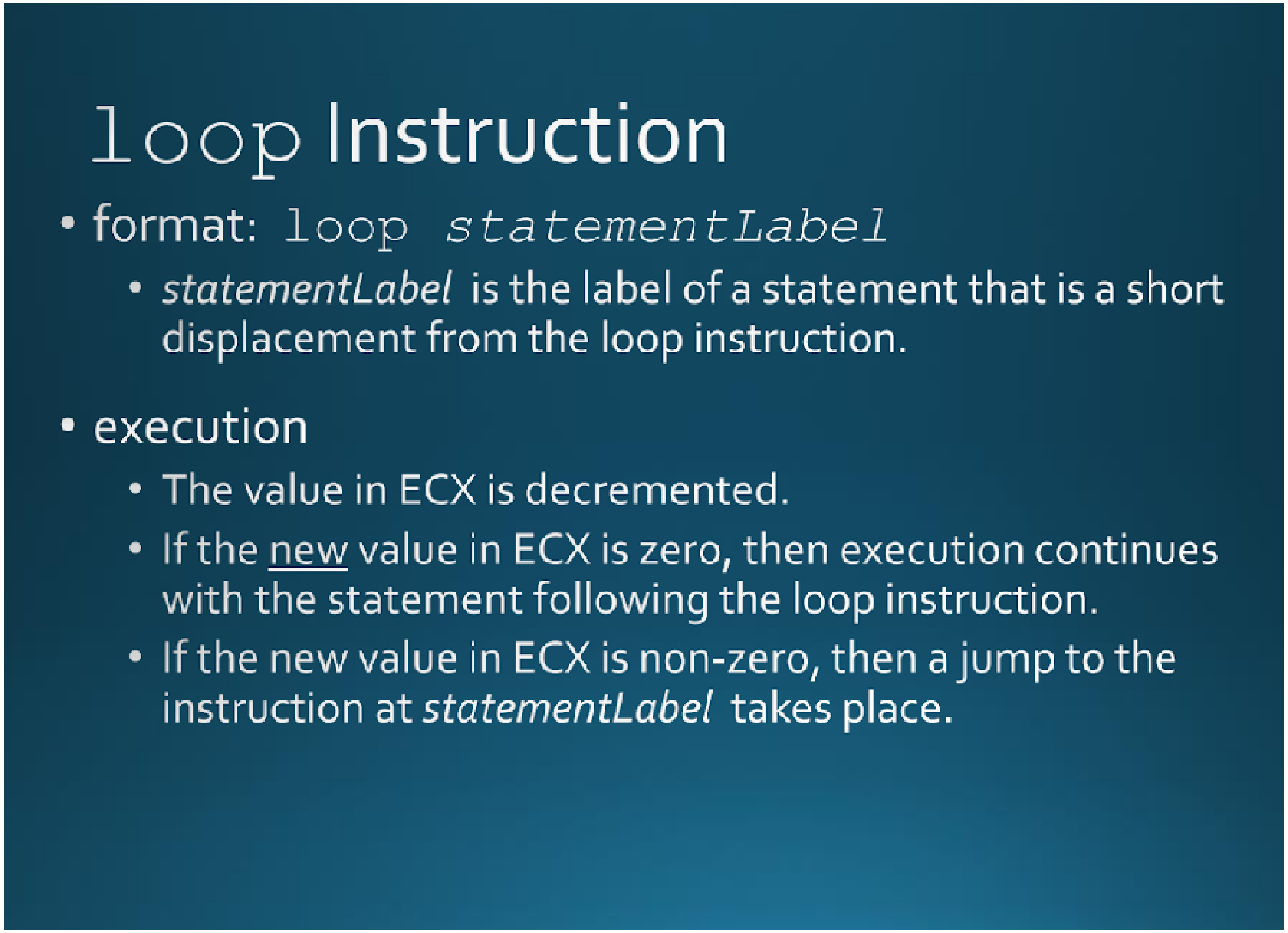
2> by using instruction mnemonic LOOP
and it uses second way to write for loop (from count to 1)

loop mnemonic uses
ECX as count by default

for Loops

- Can be implemented by converting into *while* loops
- 80x86 `loop` instruction designed to implement "backward" counter-controlled loops:

```
for index := count downto 1 loop
        … { body of loop }
end for;
```

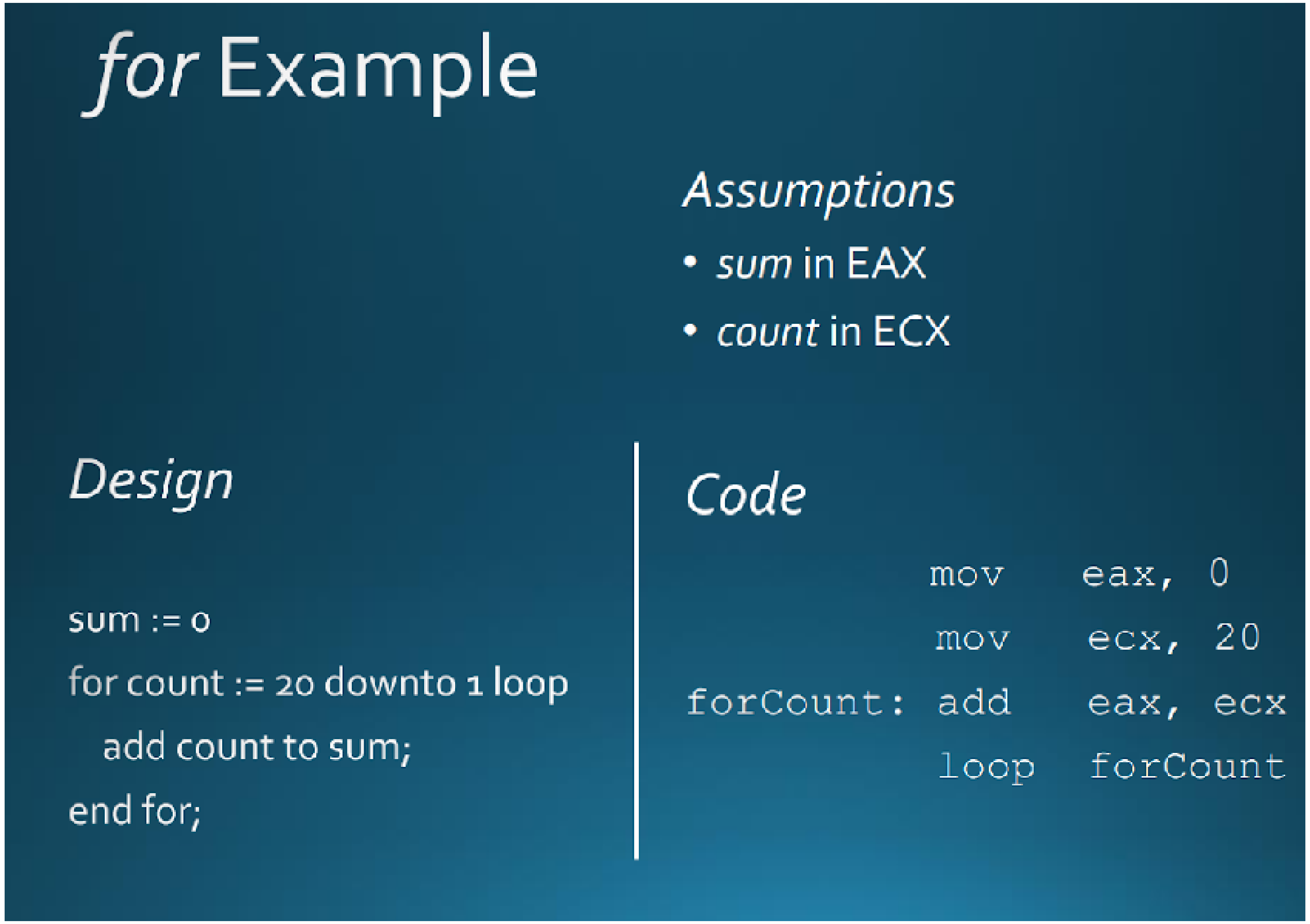


`loop` Instruction

- format: `loop` *`statementLabel`*
  - *statementLabel* is the label of a statement that is a short displacement from the loop instruction.
- execution
  - The value in ECX is decremented.
  - If the new value in ECX is zero, then execution continues with the statement following the loop instruction.
  - If the new value in ECX is non-zero, then a jump to the instruction at *statementLabel* takes place.

if ecx is zero it comes out of loop

if ecx is not zero it will be in the loop



for Example

Assumptions

- *sum* in EAX
- *count* in ECX

Design

```
sum := 0
for count := 20 downto 1 loop
   add count to sum;
end for;
```

Code

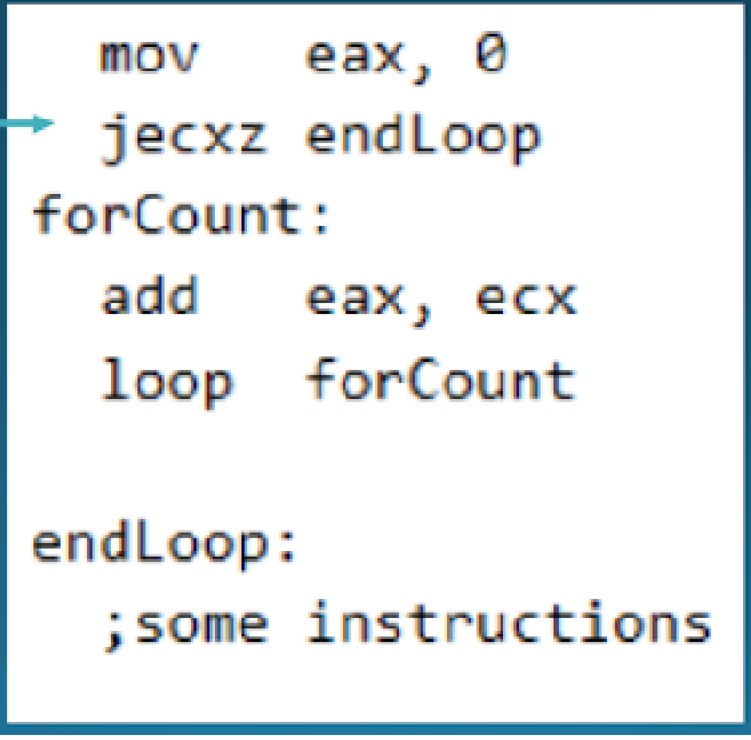
```
           mov    eax, 0
           mov    ecx, 20
forCount:  add    eax, ecx
           loop   forCount
```

iteration	ecx
1	20
2	19
3	18
....	
....	
20	1

ecx--0
it comes out of loop

Cautions

- If ECX is initially 0, then 00000000 will be decremented to FFFFFFFF, then FFFFFFFE, etc., for a total of 4,294,967,296 iterations.
- The `jecxz` ("jump if ECX is zero") instruction can be used to guard a loop implemented with the `loop` instruction.



```
  mov   eax, 0
  jecxz endLoop
forCount:
  add   eax, ecx
  loop  forCount

endLoop:
  ;some instructions
```