

Sistemas Hardware-Software

Aula 07 – Loops

2021 – Engenharia

Igor Montagner
Maciel Calebe Vidal

O par de comandos **if-goto**

O par de comandos if-goto é equivalente às instruções `cmp/test` seguidas de um `jump` condicional

```
[ cmp 0x4, %rdi  
  jle label  
  }(bloco 1)  
  label:  
  ...
```

```
if (a <= 4) {  
    goto label;  
}  
(bloco1)  
label:  
... .
```

Vamos chamar código **C** que use somente `if-goto` de **gotoC!**



voltaremos 16h15

Qualquer dúvida, marquem @professores
Atividade prática

Loops (20 minutos)

1. Identificar saltos condicionais em ciclos
2. Reconstruir um loop a partir de um programa com if-goto.

Exercício 1 – setas e comparação

Dump of assembler code for function soma_2n:

```
0x066a <+0>:      mov     $0x1,%eax
0x066f <+5>:      jmp     0x676 <soma_2n+12>
0x0671 <+7>:      shr     %edi
0x0673 <+9>:      add     $0x1,%eax
0x0676 <+12>:     cmp     $0x1,%edi
0x0679 <+15>:     ja      0x671 <soma_2n+7>
0x067b <+17>:     repz   retq
```

The diagram illustrates the control flow of the assembly code. A green arrow points from the `jmp` instruction at address `0x066f` to the `cmp` instruction at address `0x0676`. A blue arrow points from the `ja` instruction at address `0x0679` to the `shr` instruction at address `0x0671`, representing a loop back.

Exercício 1 – versão if-goto

uint a = 19
16 8 4 2 1
1 0 0 1 1
a >> 1 0 1 0 0 1 \Rightarrow 9

-16 + 3 = -13
-16 + 8 + 1 = -7

Dump of assembler code for function soma_2n:

```
0x066a <+0>:    mov     $0x1,%eax
0x066f <+5>:    jmp     0x676 <soma_2n+12>
0x0671 <+7>:    shr     %edi
0x0673 <+9>:    add     $0x1,%eax
0x0676 <+12>:   {cmp     $0x1,%edi
0x0679 <+15>:   }ja     0x671 <soma_2n+7>
0x067b <+17>:   repz    retq
```

int res = 1;

goto VERIFICA;

FAZ-ALHO:
a = a >> 1;
res += 1;

VERIFICA:

if (a > 1) {
 goto FAZ-ALHO;
}
RETURN res;

Exercício 1 – versão C

Dump of assembler code for function soma_2n:

```
0x066a <+0>:    mov     $0x1,%eax
0x066f <+5>:    jmp     0x676 <soma_2n+12>
0x0671 <+7>:    shr     %edi
0x0673 <+9>:    add     $0x1,%eax
0x0676 <+12>:   cmp     $0x1,%edi
0x0679 <+15>:   ja      0x671 <soma_2n+7>
0x067b <+17>:   repz    retq
```

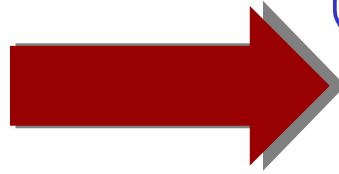

```
int res = 1;
goto VERIFICA;
FAZ-ALHO:
    a = a >> 1;
    res += 1;
VERIFICA:
    if (a > 1) {
        goto FAZ-ALHO;
    }
    return res;
```

```
int SOMA-2n(UNSIGNED int a){
    int RES = 1;
    WHILE(a > 1) {
        a = a / 2;
        res ++;
    }
    RETURN RES;
}
```

while

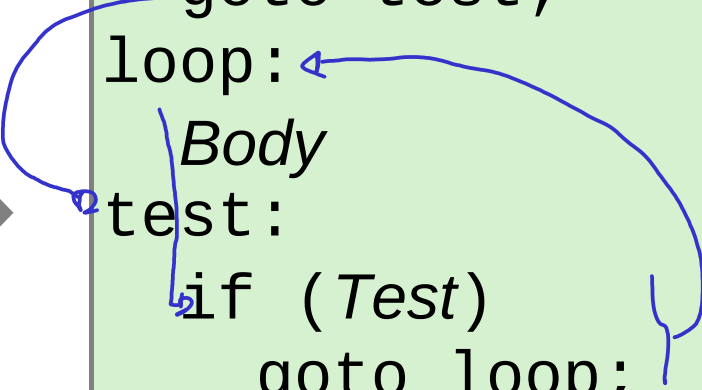
While version

```
while (Test)  
    Body
```



Goto Version

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



while

```
long foo_while(long n) {  
    long sum = 0;  
  
    while (n > 0) {  
        sum += n;  
        n--;  
    }  
  
    sum *= sum;  
    return sum;  
}
```

```
long foo_while_goto_1(long n) {  
    long sum = 0;  
  
    goto test;  
  
loop:  
    sum += n;  
    n--;  
  
test:  
    if (n > 0)  
        goto loop;  
  
    sum *= sum;  
    return sum;  
}
```


while

```
long foo_while_goto_1(long n) {  
    long sum = 0;  
  
    goto test;  
  
loop:  
    sum += n;  
    n--;  
  
test:  
    if (n > 0)  
        goto loop;  
  
    sum *= sum;  
    return sum;  
}
```

```
000000000000000044 <foo_while_goto_1>:  
44:    mov    $0x0,%eax  
49:    jmp     52 <foo_while_goto_1+0xe>  
4b:    add     %rdi,%rax  
4e:    sub     $0x1,%rdi  
52:    test    %rdi,%rdi  
55:    jg      4b <foo_while_goto_1+0x7>  
57:    imul    %rax,%rax  
5b:    retq
```

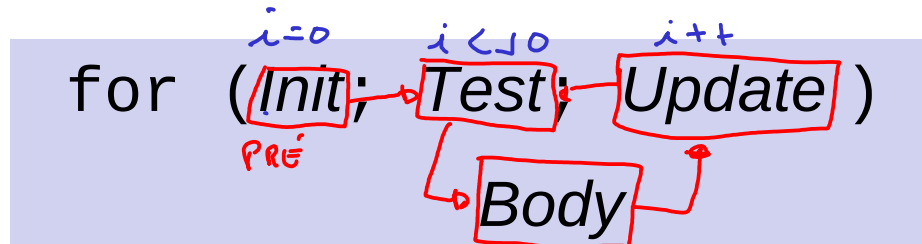
while

```
long foo_while_goto_1(long n){  
    long sum = 0;  
    goto test;  
loop:    sum += n;  
        n--;  
test:    if (n > 0)  
        goto loop;  
    sum *= sum;  
    return sum;  
}
```

```
000000000000000044 <foo_while_goto_1>:  
44:    mov    $0x0,%eax  
49:    jmp     52 <foo_while_goto_1+0xe>  
4b:    add     %rdi,%rax  
4e:    sub     $0x1,%rdi  
52:    test    %rdi,%rdi  
55:    jg      4b <foo_while_goto_1+0x7>  
57:    imul    %rax,%rax  
5b:    retq
```

for

For Version



for

For Version

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



While Version

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

for

```
long foo_for(long n) {  
    long sum;  
  
    for (sum = 0; n > 0; n--) {  
        sum += n;  
    }  
  
    sum *= sum;  
    return sum;  
}
```

```
long foo_while(long n) {  
    long sum = 0;  
  
    while (n > 0) {  
        sum += n;  
        n--;  
    }  
  
    sum *= sum;  
    return sum;  
}
```

for

while

0000000000000002c <foo_while>:

```
2c:  mov $0x0,%eax
31:  jmp 3a <foo_while+0xe>
33:  add %rdi,%rax
36:  sub $0x1,%rdi
3a:  test %rdi,%rdi
3d:  jg 33 <foo_while+0x7>
3f:  imul %rax,%rax
43:  retq
```

for

000000000000000a0 <foo_for>:

```
a0:  mov $0x0,%eax
a5:  jmp ae <foo_for+0xe>
a7:  add %rdi,%rax
aa:  sub $0x1,%rdi
ae:  test %rdi,%rdi
b1:  jg a7 <foo_for+0x7>
b3:  imul %rax,%rax
b7:  retq
```



Atividade prática

Loops (para entrega)

1. Reconstruir um loop a partir de um programa com if-goto.
2. Identificar corretamente estruturas de controle aninhadas (loop + condicional)

Insper

www.insper.edu.br