

Sistemas Hardware-Software

Aula 07 – Loops

Engenharia

Fabio Lubacheski

Maciel C. Vidal

Igor Montagner

Fábio Ayres

Relembrando instruções condicionais

Importante: **cmp** é como **sub**, **test** é como **and**
O resultado nunca é armazenado.

		cmp a, b	test a, b
je	"Equal"	$b == a$	$b \& a == 0$
jne	"Not equal"	$b != a$	$b \& a != 0$
js	"Sign" (negative)	$b - a < 0$	$b \& a < 0$
jns	(non-negative)	$b - a \geq 0$	$b \& a \geq 0$
jg	"Greater"	$b > a$	$b \& a > 0$
jge	"Greater or equal"	$b \geq a$	$b \& a \geq 0$
jl	"Less"	$b < a$	$b \& a < 0$
jle	"Less or equal"	$b \leq a$	$b \& a \leq 0$
ja	"Above" (unsigned >)	$b > a$	$b \& a > 0U$
jb	"Below" (unsigned <)	$b < a$	$b \& a < 0U$

```
cmp 5, b
```

```
je:   b == 5
```

```
jne:  b != 5
```

```
jg:   b > 5
```

```
jl:   b < 5
```

```
test a, a
```

```
je:   a == 0
```

```
jne:  a != 0
```

```
jg:   a > 0
```

```
jl:   a < 0
```

O par de comandos **if-goto** \Leftrightarrow **gotoC**

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

Código **Assembly**

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

Código **gotoC**

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

Código na **linguagem C**

```
if (a > 4) {
    (bloco1);
}
. . .
```

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

Exemplo – setas e comparação

Dump of assembler code for function `funcao`:

```
0x000000000000001149 <+0>:      endbr64
0x00000000000000114d <+4>:      mov     %edi,%eax
0x00000000000000114f <+6>:      mov     $0x0,%edx
0x000000000000001154 <+11>:     jmp     0x115b <funcao+18>
0x000000000000001156 <+13>:     add     %edi,%edx
0x000000000000001158 <+15>:     add     $0x1,%eax
0x00000000000000115b <+18>:     cmp     %esi,%eax
0x00000000000000115d <+20>:     jle     0x1156 <funcao+13>
0x00000000000000115f <+22>:     mov     %edx,%eax
0x000000000000001161 <+24>:     ret
```

End of assembler dump.

Exemplo – setas e comparação

Dump of assembler code for function `funcao`:

```
0x00000000000001149 <+0>:      endbr64
0x0000000000000114d <+4>:      mov     %edi,%eax
0x0000000000000114f <+6>:      mov     $0x0,%edx
0x00000000000001154 <+11>:     jmp     0x115b <funcao+18>
0x00000000000001156 <+13>:     add     %edi,%edx ←-----
0x00000000000001158 <+15>:     add     $0x1,%eax
0x0000000000000115b <+18>:     cmp     %esi,%eax
0x0000000000000115d <+20>:     jle     0x1156 <funcao+13> -----
0x0000000000000115f <+22>:     mov     %edx,%eax
0x00000000000001161 <+24>:     ret
```

End of assembler dump.

Exemplo – setas e comparação

Dump of assembler code for function `funcao`:

```
0x0000000000000149 <+0>:      endbr64
0x000000000000014d <+4>:      mov     %edi,%eax
0x000000000000014f <+6>:      mov     $0x0,%edx
0x0000000000000154 <+11>:     - jmp     0x115b <funcao+18> → sempre pula
0x0000000000000156 <+13>:     add     %edi,%edx ←-----
0x0000000000000158 <+15>:     add     $0x1,%eax
0x000000000000015b <+18>:     -> cmp     %esi,%eax -----
0x000000000000015d <+20>:     jle     0x1156 <funcao+13> -
0x000000000000015f <+22>:     mov     %edx,%eax
0x0000000000000161 <+24>:     ret
```

End of assembler dump.

Versão if-goto

```
int funcao(int edi, int esi) {
    int edx = 0;
    int eax = edi;
    goto compara;
faz_algo:
    edx += edi;
    eax += 1;
compara:
    if (eax-esi <= 0) {
        goto faz_algo;
    }
    return edx;
}
```

Exemplo – setas e comparação

Dump of assembler code for function `funcao`:

```
0x0000000000000149 <+0>:      endbr64
0x000000000000014d <+4>:      mov     %edi,%eax
0x000000000000014f <+6>:      mov     $0x0,%edx
0x0000000000000154 <+11>:     - jmp     0x115b <funcao+18> → sempre pula
0x0000000000000156 <+13>:     add     %edi,%edx ←-----
0x0000000000000158 <+15>:     add     $0x1,%eax
0x000000000000015b <+18>:     - cmp     %esi,%eax
0x000000000000015d <+20>:     jle     0x1156 <funcao+13> -----
0x000000000000015f <+22>:     mov     %edx,%eax
0x0000000000000161 <+24>:     ret
```

End of assembler dump.

Versão if-goto

```
int funcao(int edi, int esi) {
    int edx = 0;
    int eax = edi;
    goto compara;
faz_algo:
    edx += edi;
    eax += 1;
compara:
    if (eax-esi <= 0) {
        goto faz_algo;
    }
    return edx;
}
```

Versão legível

```
int funcao(int a, int b) {
    int res = 0;
    int i = a;
    while (i <= b) {
        res += a;
        i += 1;
    }
    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_gotoC(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_gotoC(long n) {  
    long sum = 0;
```

```
    goto test;
```

```
loop:
```

```
    sum += n;
```

```
    n--;
```

```
test:
```

```
    if (n > 0)
```

```
        goto loop;
```

```
    sum *= sum;
```

```
    return sum;
```

```
}
```

```
000000000000000044 <foo_while_gotoC>:
```

```
44:    mov    $0x0,%eax
```

```
49:    jmp    52 <foo_while_gotoC+0xe>
```

```
4b:    add    %rdi,%rax
```

```
4e:    sub    $0x1,%rdi
```

```
52:    test   %rdi,%rdi
```

```
55:    jg     4b <foo_while_gotoC+0x7>
```

```
57:    imul   %rax,%rax
```

```
5b:    retq
```

while

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

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

while - Outra variação

While version

```
while (Test)  
    Body
```



Goto Version

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

while - Outra variação

While version

```
while (Test)  
    Body
```



Goto Version

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

Código em C:

```
while ( sum != 0 ) {  
    <loop body>  
}
```

Assembly

```
loop:  testq %rax, %rax  
       je     done  
       <loop body code>  
       jmp    loop  
done:
```


for

For Version

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

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

```
00000000000000002c <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

```
0000000000000000a0 <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 (20 minutos)

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

Resposta atividade prática (exercício 1 a 6)

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
```

Resposta – 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
```

Registrador	tipo	identificador
%edi	int	a
%eax	int	res

Resposta – setas e comparação

Dump of assembler code for function soma_2n:

```
0x066a <+0>:      mov     $0x1,%eax
0x066f <+5>:      jmp     0x676 <soma_2n+12> → sempre pula
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
```

Registrador	tipo	identificador
%edi	int	a
%eax	int	res

Expressão

```
cmp     $0x1,%edi
ja      0x671 <soma_2n+7>
```

$a - 1 > 0$
 $a > 1$

Resposta – versão gotoC

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 soma_2n(unsigned int a) {
    int res = 1;
    goto verifica;

faz_algo:
    a = a >> 1;
    res += 1;

->verifica:
    if (a > 1) {
        goto faz_algo;
    }

    return res;
}
```

Resposta – 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 soma_2n(unsigned int a) {
    int res = 1;
    goto verifica;

faz_algo:
    a = a >> 1;
    res += 1;

verifica:
    if (a > 1) {
        goto faz_algo;
    }

    return res;
}
```

Versão C legível

```
int soma_2n(unsigned int a) {
    int res = 1;
    while (a > 1) {
        a = a/2;
        res++;
    }

    return res;
}
```




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