

Sistemas Hardware-Software

Aula 7 – Revisão de condicionais e Loops

2020 – Engenharia

[Igor Montagner](#), Fábio Ayres

Instruções de comparação

Permitem preencher os códigos de condição sem modificar os registradores:

- Instrução **cmp A, B**
 - Compara valores A e B
 - Funciona como **sub A, B** sem gravar resultado no destino

Flag set?	Significado
CF	Carry-out em $B - A$
ZF	$B == A$
SF	$(B - A) < 0$ (quando interpretado como signed)
OF	Overflow de complemento-de-2: $(A > 0 \ \&\& \ B < 0 \ \&\& \ (B - A) < 0) \ $ $(A < 0 \ \&\& \ B > 0 \ \&\& \ (B - A) > 0)$

Instruções de comparação

- Instrução **test A, B**
 - Testa o resultado de **A & B**
 - Funciona como **and A, B** sem gravar resultado no destino
 - Útil para checar um dos valores, usando o outro como máscara
 - Normalmente usado com A e B sendo o mesmo registrador, ou seja: **test %rdi, %rdi**

Flag set?	Significado
ZF	$A \& B == 0$
SF	$A \& B < 0$ (quando interpretado como signed)

Acessando os códigos de condição

Instrução	Condição	Descrição
sete	ZF	E qual /Zero
setne	~ZF	N ot E qual / Not Zero
sets	SF	(signed) N egativo
setns	~SF	(signed) N ão-negativo
setl	(SF^OF)	(signed) L ess than
setle	(SF^OF) ZF	(signed) L ess than or E qual
setge	~(SF^OF)	(signed) G reater than or E qual
setg	~(SF^OF) & ~ZF	(signed) G reater than
setb	CF	(unsigned) B elow
seta	~CF & ~ZF	(unsigned) A bove

Desvios (ou saltos) condicionais

Instrução	Condição	Descrição
jmp	1	Incondicional
je	ZF	E qual /Zero
jne	~ZF	N ot E qual / Not Zero
js	SF	(signed) N egativo
jns	~SF	(signed) N ão-negativo
j1	(SF^OF)	(signed) L ess than
jle	(SF^OF) ZF	(signed) L ess than or E qual
jge	~(SF^OF)	(signed) G reater than or E qual
jg	~(SF^OF) & ~ZF	(signed) G reater than
jb	CF	(unsigned) B elow
ja	~CF & ~ZF	(unsigned) A bove

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!**



Atividade prática

Loops (20 minutos)

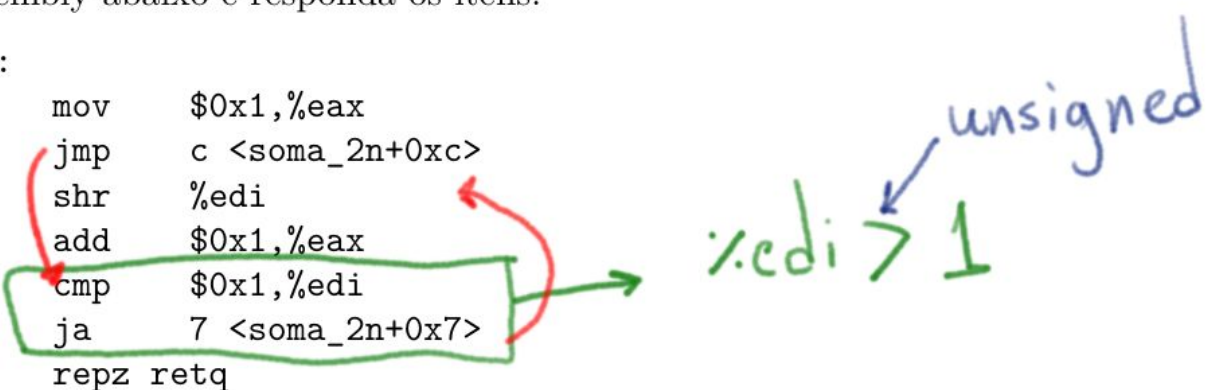
1. Identificar saltos condicionais em ciclos
2. Reconstruir um loop a partir de um programa com if-goto.

Exercício 1 – setas e comparação

Exercício 1: veja o código assembly abaixo e responda os itens.

0000000000000000 <soma_2n>:

```
0:  b8 01 00 00 00      mov     $0x1,%eax
5:  eb 05               jmp     c <soma_2n+0xc>
7:  d1 ef               shr     %edi
9:  83 c0 01            add     $0x1,%eax
c:  83 ff 01            cmp     $0x1,%edi
f:  77 f6               ja      7 <soma_2n+0x7>
11: f3 c3              repz retq
```



unsigned
%edi > 1

1. Localize no código acima as instruções de saltos (`jmp` ou condicionais `j*`). Desenhe setas indicando para qual linha do código elas pulam.

Exercício 1 – versão if-goto

Exercício 1: veja o código assembly abaixo e responda os itens.

0000000000000000 <soma_2n>:

```
0:  b8 01 00 00 00    mov     $0x1,%eax
5:  eb 05             jmp     c <soma_2n+0xc>
7:  d1 ef             shr     %edi → unsigned int a;
9:  83 c0 01          add     $0x1,%eax
c:  83 ff 01          cmp     $0x1,%edi
f:  77 f6             ja      7 <soma_2n+0x7>
11: f3 c3             repz retq
```

```
int r=1;
goto Lc;
L7:
a=a>>1;
r+=1;
Lc: if(a>1) goto L7;
return r;
```

1. Localize no código acima as instruções de saltos (jmp ou condicionais j*). Desenhe setas indicando para qual linha do código elas pulam.

Exercício 1 - versão final

Exercício 1: veja o código assembly abaixo e responda os itens.

0000000000000000 <soma_2n>:

```
0:  b8 01 00 00 00    mov     $0x1,%eax
5:  eb 05             jmp     c <soma_2n+0xc>
7:  d1 ef             shr     %edi
9:  83 c0 01          add     $0x1,%eax
c:  83 ff 01          cmp     $0x1,%edi
f:  77 f6             ja      7 <soma_2n+0x7>
11: f3 c3             repz retq
```

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

1. Localize no código acima as instruções de saltos (`jmp` ou condicionais `j*`). Desenhe setas indicando para qual linha do código elas pulam.

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
```

while, versão 2

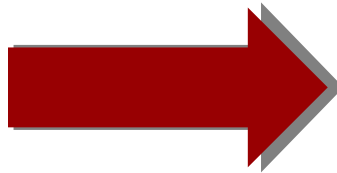
While version

```
while (Test)  
  Body
```



Do-While Version

```
if (!Test)  
  goto done;  
do  
  Body  
  while(Test);  
done:
```



Goto Version

```
if (!Test)  
  goto done;  
loop:  
  Body  
  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_do_while(long n) {  
    long sum = 0;  
  
    if (n <= 0)  
        goto done;  
  
    do {  
        sum += n;  
        n--;  
    } while (n > 0);  
  
done:  
    sum *= sum;  
    return sum;  
}
```


while

```
long foo_while_do_while(long n) {  
    long sum = 0;  
  
    if (n <= 0)  
        goto done;  
  
    do {  
        sum += n;  
        n--;  
    } while (n > 0);  
  
done:  
    sum *= sum;  
    return sum;  
}
```

```
long foo_while_goto_2(long n) {  
    long sum = 0;  
  
    if (n <= 0)  
        goto done;  
  
loop:  
    sum += n;  
    n--;  
  
    if (n > 0)  
        goto loop;  
  
done:  
    sum *= sum;  
    return sum;  
}
```

while

```
long foo_while_goto_2(long n) {  
    long sum = 0;
```

```
    if (n <= 0)  
        goto done;
```

```
loop:  
    sum += n;  
    n--;
```

```
    if (n > 0)  
        goto loop;
```

```
done:  
    sum *= sum;  
    return sum;  
}
```

```
00000000000000007e <foo_while_goto_2>:
```

```
7e:    test    %rdi,%rdi  
81:    jle     96 <foo_while_goto_2+0x18>  
83:    mov     $0x0,%eax  
88:    add     %rdi,%rax  
8b:    sub     $0x1,%rdi  
8f:    test    %rdi,%rdi  
92:    jg      88 <foo_while_goto_2+0xa>  
94:    jmp     9b <foo_while_goto_2+0x1d>  
96:    mov     $0x0,%eax  
9b:    imul    %rax,%rax  
9f:    retq
```

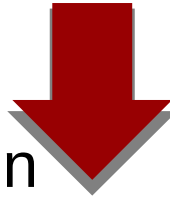
while

```
long foo_while_goto_2(long n) { 0000000000000007e <foo_while_goto_2>:
    long sum;
    if (n <= 0)                7e:    test    %rdi,%rdi
        goto almostdone;      81:    jle     96 <foo_while_goto_2+0x18>
    sum = 0;                   83:    mov     $0x0,%eax
loop:  sum += n;               88:    add     %rdi,%rax
    n--;                       8b:    sub     $0x1,%rdi
    if (n > 0)                 8f:    test    %rdi,%rdi
        goto loop;            92:    jg      88 <foo_while_goto_2+0xa>
    goto done;                 94:    jmp     9b <foo_while_goto_2+0x1d>
almostdone: sum = 0;           96:    mov     $0x0,%eax
done:  sum *= sum;             9b:    imul    %rax,%rax
    return sum;               9f:    retq
}
```

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