

Sistemas Hardware-Software

Aula 8 – Variáveis na pilha

2020 – Engenharia

[Igor Montagner](#), Fábio Ayres

Aula passadas

- Operações aritméticas
- Acessos à memória
- Chamadas de funções
- Expressões booleanas e pulos condicionais
- Loops

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, versão 2

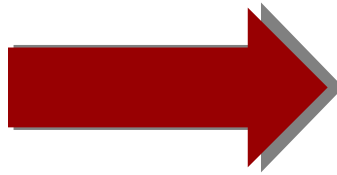
While version

```
while (Test)  
    Body
```



Do-While Version

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



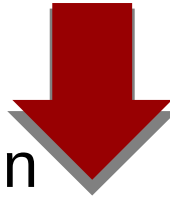
Goto Version

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

for

For Version

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



While Version

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

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

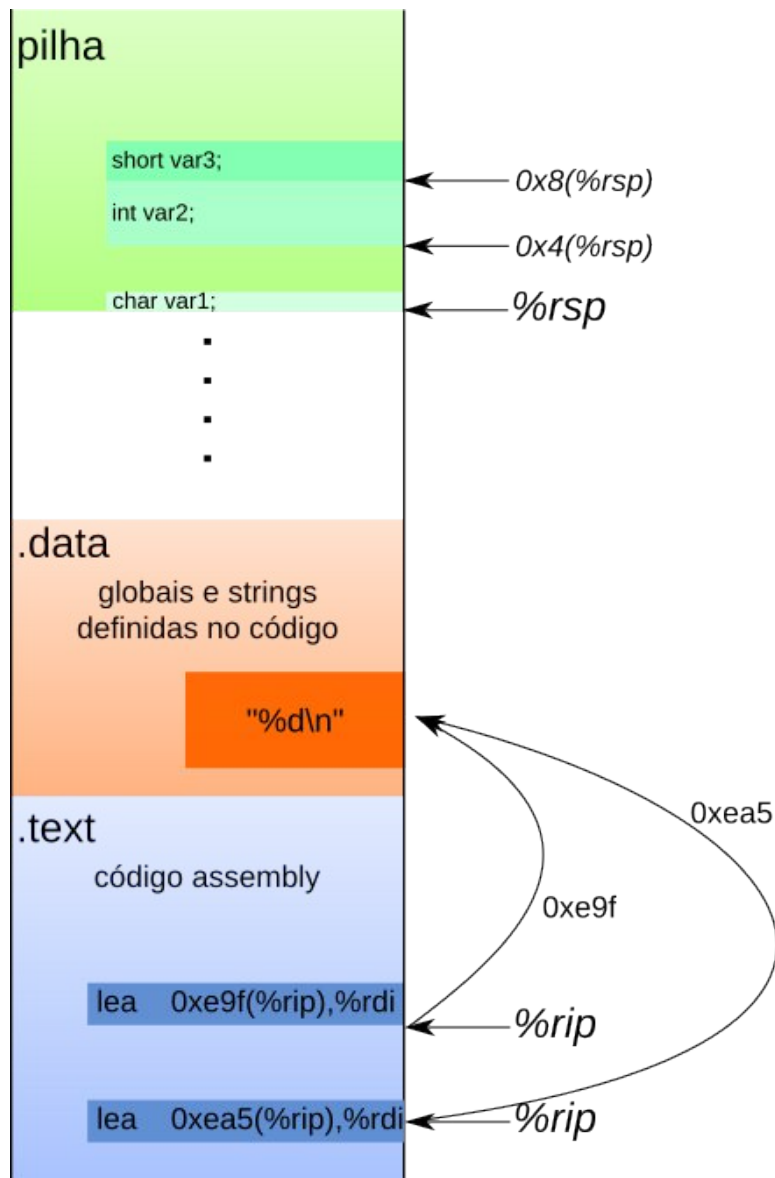
```
0000000000000000a0 <foo_for>:
 a0:  mov    $0x0,%eax
 a5:  jmp    ae <foo_for+0xe>
 a7:  add    %rdi,%rax
 aa:  sub    $0x1,%rdi
 ae:  test   %rdi,%rdi
 b1:  jg     a7 <foo_for+0x7>
 b3:  imul   %rax,%rax
 b7:  retq
```


Variáveis locais

- Na maioria do tempo são colocadas em registradores
- Se não for possível colocamos na pilha (memória)
 - Uso &var requer colocar na pilha.
 - Registrador não tem endereço
- Topo da pilha está armazenado em %rsp
- Sabemos acessar memória de maneira relativa a %rsp

\$0xF(%rsp)

Executável na memória



Variáveis locais

- Armazenadas na pilha
- Acessadas via deslocamentos relativos a `%rsp` (*stack pointer*)
- Colocadas e retiradas de registradores frequentemente

Variáveis globais / strings constantes

- Acessadas usando pulos relativos a `%rip` (*instruction pointer*)
- Como `%rip` muda a cada instrução, o deslocamento usado muda também
- É necessário fazer o cálculo para chegar ao endereço final

Criando variáveis locais

```
sub $0x10, %rsp
```

```
. . .  
mov 0x4(%rsp), %rdx
```

```
. . .  
mov %rdx, 0x4(%rsp)  
. . .
```

```
add $0x10, %rsp
```

- . Subtrair de %rsp equivale a empilhar, somar equivale a desempilhar
- . Não existe suporte para operações memória-memória
- . No fim da função deletamos todas as variáveis locais

Atividade prática

Exercícios de aula

1. Identificar funções que usem variáveis locais
2. Listar todas as variáveis locais de uma função que foram alocadas na pilha

Correção dos exercícios 2 e 3



Atividade prática

Exercícios para entrega

1. Identificar funções que usem variáveis locais
2. Listar todas as variáveis locais de uma função que foram alocadas na pilha

Insper

www.insper.edu.br