Insper

Sistemas Hardware-Software

Aula 8 – Variáveis na pilha

2022 – Engenharia

Maciel C Vidal
Igor Montagner
Fábio Ayres

Aulas 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

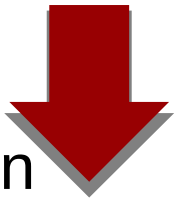
```
long foo_while_goto_1(long n) {
  long sum = 0;

  goto test;

loop:
  sum += n;
  n--;

test:
  if (n > 0)
    goto loop;

  sum *= sum;
  return sum;
}
```

```
0000000000000044 <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 (Init; Test; Update )
        Body
```



While Version

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

for

while

```
000000000000002c <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

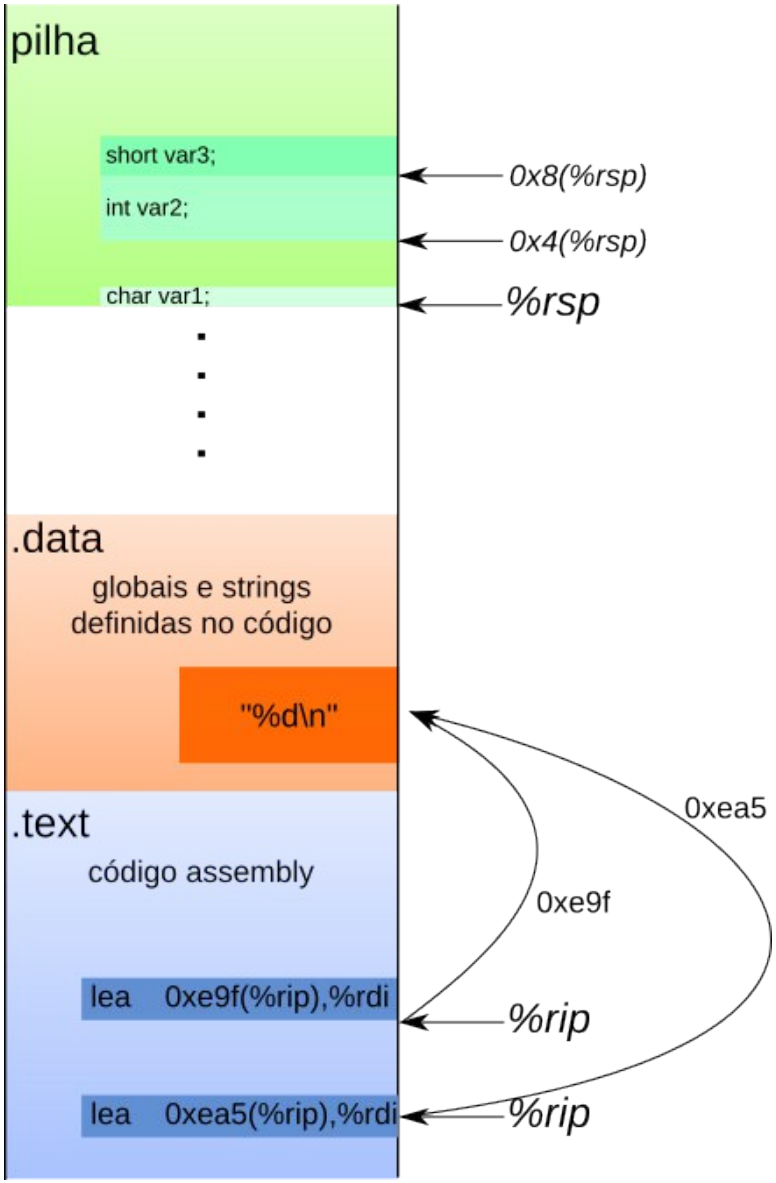
```
00000000000000a0 <foo_for>:
  a0:    mov    $0x0,%eax
  a5:    jmp    ae <foo_for+0xe>
  a7:    add    %rdi,%rax
  aa:    sub    $0x1,%rdi
  ae:    test   %rdi,%rdi
  b1:    jg     a7 <foo_for+0x7>
  b3:    imul   %rax,%rax
  b7:    retq
```

Variáveis locais

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

```
$0x4(%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 fina

Insper

Criando variáveis locais

```
sub $0x10, %rsp

. . .
mov 0x2, 0x4(%rsp)
. . .
mov 0x4(%rsp), %rdx
. . .


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

Criando variáveis locais

```
sub $0x8, %rsp

. . .
mov %rdx, 0x8(%rsp)
. . .
mov 0x8(%rsp), %rdx
. . .


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

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
3. Está no seu repositório de atividade

Correção dos exercícios 2 e 3

Ex2

```
Dump of assembler code for function func1:
   0x05fe <+0>:     sub    $0x10,%rsp
   0x0602 <+4>:     movl   $0xa,0xc(%rsp)
   0x060a <+12>:    movl   $0xb,0x8(%rsp)
   0x0612 <+20>:    lea    0xc(%rsp),%rdi
   0x0617 <+25>:    callq  0x5fa <func2>
   0x061c <+30>:    addl   $0x1,0x8(%rsp)
   0x0621 <+35>:    lea    0x8(%rsp),%rdi
   0x0626 <+40>:    callq  0x5fa <func2>
   0x062b <+45>:    add    $0x10,%rsp
   0x062f <+49>:    retq
```

Ex2

Variáveis auxiliares:
int *p1, *p2;

```
Dump of assembler code for function func1:
   0x05fe <+0>:     sub    $0x10,%rsp
   0x0602 <+4>:     movl   $0xa,0xc(%rsp)
   0x060a <+12>:    movl   $0xb,0x8(%rsp)
   0x0612 <+20>:    lea    0xc(%rsp),%rdi
   0x0617 <+25>:    callq  0x5fa <func2>
   0x061c <+30>:    addl   $0x1,0x8(%rsp)
   0x0621 <+35>:    lea    0x8(%rsp),%rdi
   0x0626 <+40>:    callq  0x5fa <func2>
   0x062b <+45>:    add    $0x10,%rsp
   0x062f <+49>:    retq
```

→ int a = 10;
→ int b = 11;
→ p1 = &a;
→ func2(p1);
→ b++;
→ p2 = &b;
→ func2(p2);

Ex3

```
Dump of assembler code for function main:
   0x1149 <+0>:     sub    $0x18,%rsp
   0x114d <+4>:     lea    0xc(%rsp),%rsi
   0x1152 <+9>:     lea    0xeab(%rip),%rdi        # 0x2004
   0x1159 <+16>:    mov    $0x0,%eax
   0x115e <+21>:    callq  0x1040 <__isoc99_scanf@plt>
   0x1163 <+26>:    cmpl   $0x0,0xc(%rsp)
   0x1168 <+31>:    js     0x1180 <main+55>
   0x116a <+33>:    lea    0xe9f(%rip),%rdi        # 0x2010
   0x1171 <+40>:    callq  0x1030 <puts@plt>
   0x1176 <+45>:    mov    $0x0,%eax
   0x117b <+50>:    add    $0x18,%rsp
   0x117f <+54>:    retq
   0x1180 <+55>:    lea    0xe80(%rip),%rdi        # 0x2007
   0x1187 <+62>:    callq  0x1030 <puts@plt>
   0x118c <+67>:    jmp    0x1176 <main+45>
```

Ex3

```
Dump of assembler code for function main:
   0x1149 <+0>:     sub    $0x18,%rsp
   0x114d <+4>:     lea    0xc(%rsp),%rsi
   0x1152 <+9>:     lea    0xeab(%rip),%rdi        # 0x2004
   0x1159 <+16>:    mov    $0x0,%eax
   0x115e <+21>:    callq  0x1040 <__isoc99_scanf@plt>
   0x1163 <+26>:    cmpl   $0x0,0xc(%rsp)
   0x1168 <+31>:    js     0x1180 <main+55>
   0x116a <+33>:    lea    0xe9f(%rip),%rdi        # 0x2010
   0x1171 <+40>:    callq  0x1030 <puts@plt>
   0x1176 <+45>:    mov    $0x0,%eax
   0x117b <+50>:    add    $0x18,%rsp
   0x117f <+54>:    retq
   0x1180 <+55>:    lea    0xe80(%rip),%rdi        # 0x2007
   0x1187 <+62>:    callq  0x1030 <puts@plt>
   0x118c <+67>:    jmp    0x1176 <main+45>
```

```
int n;
scanf("%d", &n);

if (n<0) {
    goto negativo;
}

printf("Positivo\n");

retorno:
    return 0;

negativo:
    printf("Negativo\n");
    goto retorno;
```

Insper

www.insper.edu.br