Insper

07 - Loops `while` e `for`

Sistemas Hardware-Software - 2019/1

Igor Montagner

Parte 1 - revisão

Exercício 1:

```
Dump of assembler code for function ex1:
   0x066a <+0>:     lea    (%rsi,%rdi,2),%eax
   0x066d <+3>:     lea    (%rdx,%rdx,2),%edx
   0x0670 <+6>:     add    %edx,%eax
   0x0672 <+8>:     lea    (%rcx,%rcx,2),%ecx
   0x0675 <+11>:    lea    (%rcx,%rcx,1),%edx
   0x0678 <+14>:    add    %edx,%eax
   0x067a <+16>:    retq
```

1. Quantos argumentos a função recebe? Quais seus tipos? Ela retorna algo? O quê?

2. Traduza o programa acima linha a linha.

3. Escreva uma versão legível do programa acima.

Exercício 2:

```
Dump of assembler code for function ex2:
   0x06da <+0>:     cmp    %esi,%edi
   0x06dc <+2>:     jg     0x6e6 <ex2+12>
   0x06de <+4>:     mov    %esi,(%rdx)
   0x06e0 <+6>:     mov    $0x0,%eax
   0x06e5 <+11>:    retq
   0x06e6 <+12>:    mov    %edi,(%rdx)
   0x06e8 <+14>:    mov    $0x1,%eax
   0x06ed <+19>:    retq
```

1. Quantos argumentos a função recebe? Quais seus tipos? Ela retorna algo? O quê?

2. A função acima faz uma comparação. Qual e entre quais variáveis?

3. As instruções nas linhas `ex2+4` e `ex2+12` fazem acessos a memória. Qual o tipo da variável destino?

4. Faça uma tradução da função acima usando somente `if-goto`

5. Faça uma versão legível da função acima.

Parte 2 - loops `while` e `for`

Exercício 3: veja o código assembly abaixo(arquivo `ex1.o`) e responda os itens.

```
Dump of assembler code for function soma_2n:
   0x0000 <+0>:     mov    $0x1,%eax
   0x0005 <+5>:     cmp    $0x1,%edi
   0x0008 <+8>:     jbe    0x11 <soma_2n+17>
   0x000a <+10>:    shr    %edi
   0x000c <+12>:    add    $0x1,%eax
   0x000f <+15>:    jmp    0x5 <soma_2n+5>
   0x0011 <+17>:    retq
```

1. Localize no código acima as instruções de saltos (`jmp` ou condicionais `j*`). Desenhe setas indicando para qual linha do código elas pulam.
2. Analise o fluxo de saltos do seu código. Existe um loop? Entre quais linhas?

3. Comece fazendo uma versão *C* usando somente `if-goto`

4. Transforme a construção que você fez acima em um código usando `while`.

Vamos agora exercitar o que vimos na aula expositiva.

Exercício 4: Leia o código assembly abaixo e responda.

```
Dump of assembler code for function soma_n:
   0x066a <+0>:     mov    $0x0,%edx
   0x066f <+5>:     mov    $0x0,%eax
   0x0674 <+10>:    jmp    0x67f <soma_n+21>
   0x0676 <+12>:    movslq %edx,%rcx
   0x0679 <+15>:    add    %rcx,%rax
   0x067c <+18>:    add    $0x1,%edx
   0x067f <+21>:    cmp    %edi,%edx
   0x0681 <+23>:    jl     0x676 <soma_n+12>
   0x0683 <+25>:    repz retq
```

1. Desenhe as flechas indicando o destino de cada instrução de pulo (`jmp` ou `j*`).
2. Escreva abaixo o cabeçalho da função `soma_n`. **Dica**: procure por registradores que são lidos *antes* de serem escritos.

3. Faça a tradução do código acima para *C* usando somente `if-goto`

4. Converta o código acima para uma versão legível em *C*.

Exercício 5: Considerando o arquivo *ex5*, responda as perguntas abaixo.

```
Dump of assembler code for function ex5:
   0x1139 <+0>:     mov    $0x0,%ecx
   0x113e <+5>:     mov    $0x0,%r8d
   0x1144 <+11>:    jmp    0x114a <ex4+17>
   0x1146 <+13>:    add    $0x1,%rcx
   0x114a <+17>:    cmp    %rdi,%rcx
   0x114d <+20>:    jge    0x1161 <ex4+40>
   0x114f <+22>:    mov    %rcx,%rax
   0x1152 <+25>:    cqto
   0x1154 <+27>:    idiv   %rsi
   0x1157 <+30>:    test   %rdx,%rdx
   0x115a <+33>:    jne    0x1146 <ex4+13>
   0x115c <+35>:    add    %rcx,%r8
   0x115f <+38>:    jmp    0x1146 <ex4+13>
   0x1161 <+40>:    mov    %r8,%rax
   0x1164 <+43>:    retq
```

1. Quantos argumentos a função acima recebe? Quais seus tipos? **Dica**: não se esqueça de buscar por registradores que são lidos antes de serem escritos.

2. A função retorna algum valor? Se sim, qual seu tipo?

3. A função acima combina loops e condicionais. Desenhe setas para onde as instruções de `jmp` apontam.
4. Com base no exercício anterior, entre quais linhas o loop ocorre? E a condicional?

5. O loop acima tem uma variável contadora. Ela está em qual registrador? Qual seu tipo?

6. Revise o funcionamento da instrução `idiv`. Em qual registrador é armazenado o resultado da divisão? E o resto?

7. Qual a condição testada na condicional?

8. Escreva uma versão do código acima usando somente `if-goto`.

9. Escreva uma versão legível do código acima

/