

Linguagens de Montagem

Sub-Rotinas Funções e Procedimentos Aula 10

Edmar André Bellorini

Ano letivo 2021

Introdução

- Sub-rotinas
 - Também chamadas de sub-programas
 - São funções ou procedimentos
 - Objetivo é **auxiliar** a execução do programa principal
 - São chamadas pelo programa principal para executar tarefas específicas ou rotineiras
 - Protocolo de chamada e retorno diferem entre arquiteturas
 - x86 → uso da pilha (não abordado neste documento)
 - x64 → uso de registradores e pilha
 - Variáveis de escopo local são armazenadas na pilha

Instruções de Chamada e Retorno

■ CALL

- Empilha o próximo endereço de execução
- Altera fluxo de execução para endereço alvo

```
CALL label
```

- Executado por sub-programa **chamador** (*caller*)

■ RET

- Desempilha topo da pilha
- Altera fluxo de execução para endereço desempilhado

```
RET
```

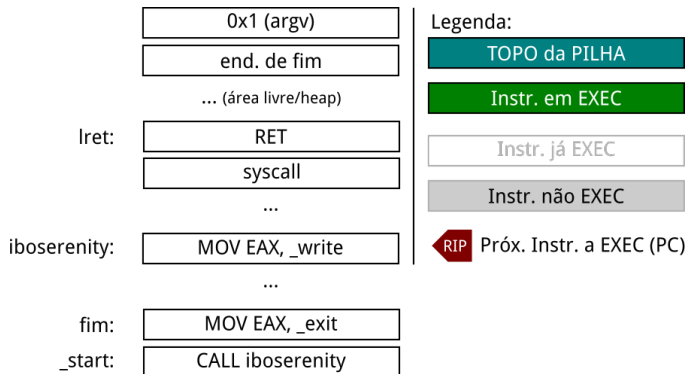
- Endereço no topo da pilha **deve** ser valor empilhado por CALL
- Executado por sub-programa **chamado** (*callee*)

Exemplo a10e01.asm - Chamada de procedimento

```
16  ...  
17  _start:  
18      ; PUSH fim  
19      call iboserenity  
20      ...
```

- GDB
 - b _start, fim, iboserenity
 - verificar *RSP* e conteúdo de topo da pilha
- Acompanhar nos próximos slides

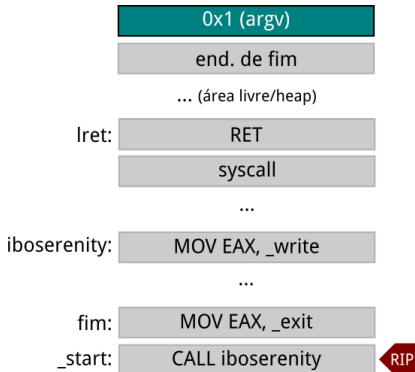
Memória para a10e01.asm - parte 0 - definições



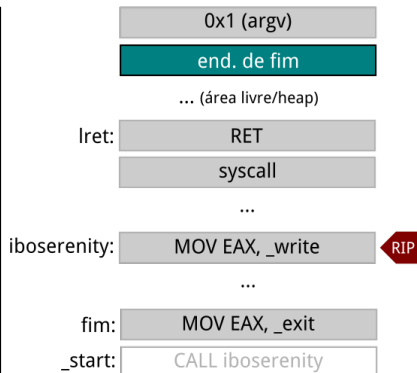
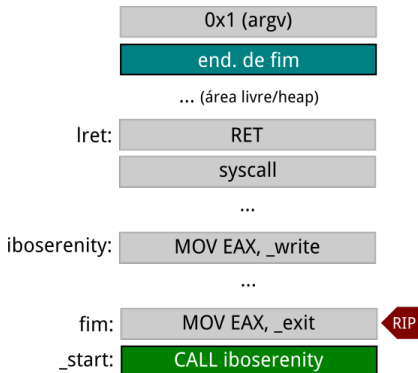
■ GDB

break em _start, fim, iboserenity, lret

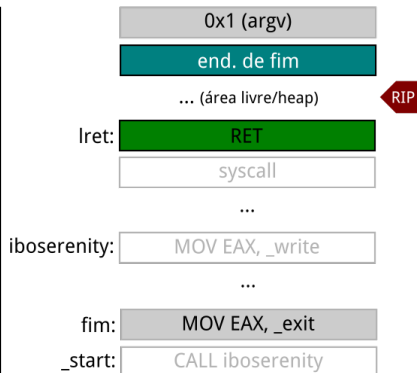
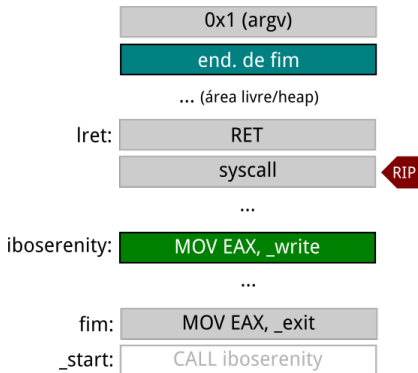
Memória para a10e01.asm - parte 1



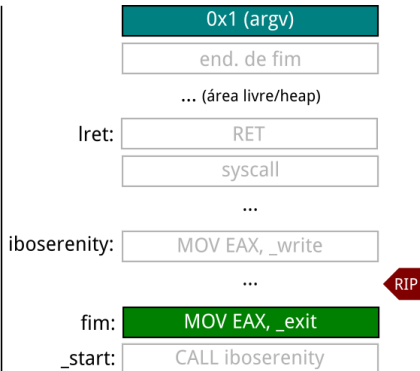
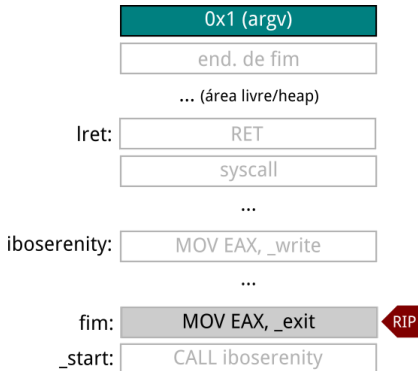
Memória para a10e01.asm - parte 2



Memória para a10e01.asm - parte 3



Memória para a10e01.asm - parte 4



Importante

- TOPO da PILHA deve ser gerenciado
 - Se existirem PUSHs e POPs **internos** no sub-programa, **antes** de executar RET, o TOPO da PILHA **deve** conter o endereço de retorno.
- Como *buggar* um programa?

- No exemplo a10e01.asm, insira o código abaixo na linha 31

```
32     push rax
```

Assim:

```
30     ...  
31     syscall  
32  
33     push rax           ; precedendo comentario ; POP PC  
34     ret
```

- *segmentation fault?*

Protocolo x86 para chamadas de sub-programas

- **IMPORTANTE:** chamada x86 não é mais estudada na disciplina
 - Seção alterada para ANEXO.
 - Arquivo de exemplo A10e02.asm

Protocolo x64 para chamadas de sub-programas

■ Convenção de chamadas

■ Chamador (*caller*)

■ Antes da instrução CALL

- 1 Salvar *caller-saved registers* (se necessário)
RAX, RCX, RDX, R8, R9, R10 e R11
- 2 Passagem de parâmetros para sub-programa via REGISTRADORES e PILHA
RDI^{arg1}, RSI^{arg2}, RDX^{arg3}, RCX^{arg4}, R8^{arg5} e R9^{arg6}
Demais argumentos são passados via PILHA (*like x86*)
- 3 **Sempre** usar instrução CALL

■ Após instrução CALL

- 4 Remover parâmetros da PILHA (se existirem)
- 5 Recuperar os valores dos *caller-saved registers*

Protocolo x64 para chamadas de sub-programas

■ Convenção de chamadas

■ Chamado (*callee*)

■ Antes de executar corpo de sub-programa

- ① Criar *stack-frame*
- ② Alocar espaço na PILHA para variáveis locais
- ③ *calle-saved registers* (se necessário)
RBX, RDI, RSI, RBP, RSP, R12, R13, R14 e R15

■ Depois da execução do corpo do sub-programa

- ④ Deixar resultado/retorno do sub-programa em RAX
- ⑤ Recuperar os *calle-saved registers*
- ⑥ Desalocar todas as variáveis locais
- ⑦ Garantir endereço de retorno no topo da PILHA
- ⑧ **Sempre** retornar com instrução RET

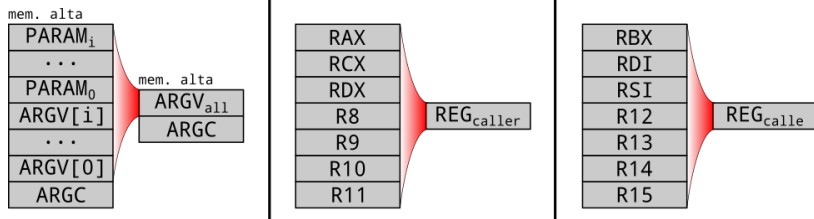
Huge example - a10e03.asm

■ Será estudado passo-a-passo

```
50      ; passo 2 - Antes da chamada CALL
51      ; Passagem de parametros
52      mov rdi, strTeste1
53
54      ; passo 3 - chamada CALL
55      call strLength
56
57      ; retorno do sub-programa em EAX
58      mov [str1L], eax ; inteiro de 32bit
59      ...
```

a10e03.asm - Antes do passo-a-passo (a)

■ Abstração compactada



a10e03.asm - Antes do passo-a-passo (b)

Estado inicial da pilha

mem. alta

ARGV _{all}
ARGC
REG _{caller}
ret ADDR
old RBP
deslocador
REG _{callee}

legenda:



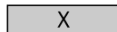
RSP

End. apontado por RSP



RBP

End. apontado por RBP



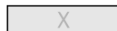
Alocado/usável



Topo da Pilha



Desalocado/lixo



Desconsiderar

Parâmetros em registradores:



Conteúdo

Passo-a-passo com exemplo a10e03.asm - parte 01

■ Convenção de chamadas

■ Chamador (caller)

■ Antes da instrução CALL

- 1 Salvar *caller-saved registers* (se necessário)
RAX, RCX, RDX, R8-11

```
39      ...  
40      ; passo 1 - Antes da chamada CALL  
41      ; salvar registradores RAX, RCX, RDX, R8-11  
42      push rax  
43      push rcx  
44      push rdx  
45      ...
```

Passo-a-passo com exemplo a10e03.asm - PILHA

mem. alta

ARGV _{all}
ARGC
REG _{caller}
ret ADDR
old RBP
deslocador
REG _{calle}

RSP

RBP

Parâmetros em registradores:

REG

Conteúdo

Passo-a-passo com exemplo a10e03.asm - parte 02

■ Convenção de chamadas

■ Chamador (caller)

■ Antes da instrução CALL

- ② Passagem de parâmetros para sub-programa via REG+PILHA
RDI^{arg1}, RSI^{arg2}, RDX^{arg3}, RCX^{arg4}, R8^{arg5} e R9^{arg6}
PILHA (*like* x86)

```
50      ...  
51      ; passo 2 - Antes da chamada CALL  
52      ; Parametro em RDI  
53      mov rdi, strTeste1  
54      ...
```

Passo-a-passo com exemplo a10e03.asm - PILHA

mem. alta

ARGV _{all}
ARGC
REG _{caller}
ret ADDR
old RBP
deslocador
REG _{calle}

RSP

RET

Parâmetros em registradores:

1º

RDI

*strTeste1

Passo-a-passo com exemplo a10e03.asm - parte 03

- Convenção de chamadas
 - **Chamador** (caller)
 - **Antes** da instrução CALL
 - ③ **Sempre** usar instrução CALL

```
54     ...  
55     ; passo 3 - chamada CALL  
56     call strlen  
57     ...
```

Passo-a-passo com exemplo a10e03.asm - PILHA

mem. alta

ARGV _{all}
ARGC
REG _{caller}
ret ADDR
old RBP
deslocador
REG _{calle}

RSP

Parâmetros em registradores:

1º

RDI

*strTeste1

Passo-a-passo com exemplo a10e03.asm - parte 04

■ Convenção de chamadas

■ Chamado (*callee*)

■ Antes de executar corpo de sub-programa

① Criar *stack-frame*

```
84      ...  
85      ; passo 1 - Antes do corpo do sub-programa  
86      ; criar stack-frame  
87      push rbp  
88      mov rbp, rsp  
89      ...
```

Passo-a-passo com exemplo a10e03.asm - PILHA

mem. alta

ARGV _{all}
ARGC
REG _{caller}
ret ADDR
old RBP
deslocador
REG _{calle}



Parâmetros em registradores:

1º

RDI

*strTeste1

Passo-a-passo com exemplo a10e03.asm - parte 05

■ Convenção de chamadas

■ Chamado (*callee*)

■ Antes de executar corpo de sub-programa

② Alocar espaço na PILHA para variáveis locais

```
88      ...  
89      ; passo 2 - Antes do corpo do sub-programa  
90      ; criar espaco para variaveis locais  
91      sub rsp, 4 ; 4 bytes para variavel inteira local  
92      ; [esp-4] eh o deslocador ate encontrar '0'  
93      ...
```

Passo-a-passo com exemplo a10e03.asm - PILHA

mem. alta

ARGV _{all}
ARGC
REG _{caller}
ret ADDR
old RBP
deslocador
REG _{calle}



Parâmetros em registradores:

1º

RDI

*strTeste1

Passo-a-passo com exemplo a10e03.asm - parte 06

- Convenção de chamadas
 - Chamado (*callee*)
 - Antes de executar corpo de sub-programa
 - ③ *calle-saved registers* (se necessário)
RBX, RDI, RSI, R12-15

```
94      ...
95      ; passo 3 - Antes do corpo do sub-programa
96      ; salvar registradores RBX, RDI, RSI, R12-15
97      push rbx
98      push rdi
99      push rsi
100     ...
```

Passo-a-passo com exemplo a10e03.asm - PILHA

mem. alta

ARGV _{all}
ARGC
REG _{caller}
ret ADDR
old RBP
deslocador
REG _{calle}

RSP

RBP

[RBP-4]

Parâmetros em registradores:

1º

RDI

*strTeste1

Passo-a-passo com exemplo a10e03.asm - parte 07

■ Convenção de chamadas

■ Chamado (*callee*)

- sub-programa
- parâmetro `*char[]` → RDI
- variável local → `[ebp-4]`

```

103  ...
104      ; sub-programa - inicio -----
105      ; RDI = char *c[] - parametro 1
106      mov dword [rbp-4], 0
107      laco:
108          mov edx, [rbp-4] ; deslocador de 32 bits
109          mov al, [rdi+rdx] ; char[deslocador]
110          cmp al, 0        ; char eh zero?
111          je finaliza
112          inc edx           ; desloc++
113          mov [rbp-4], edx  ; guarda deslocador
114          jmp laco
115  ...

```

Passo-a-passo com exemplo a10e03.asm - parte 08

■ Convenção de chamadas

■ Chamado (*callee*)

- **depois** da execução do corpo de sub-programa

④ Deixar resultado/retorno do sub-programa em EAX

```
101     ...  
102     ; passo 4 - depois do corpo do sub-programa  
103     ; copiar resultado para RAX  
104     mov eax, [rbp-4] ; 'zero-fill' 32H + 32L <- [rbp-4]  
105     ...
```

Passo-a-passo com exemplo a10e03.asm - PILHA

mem. alta

ARGV _{all}
ARGC
REG _{caller}
ret ADDR
old RBP
deslocador
REG _{calle}

RSP

RBP
[RBP-4]

Parâmetros em registradores:

1º

RDI

*strTeste1

RAX

resposta

Passo-a-passo com exemplo a10e03.asm - parte 09

■ Convenção de chamadas

■ Chamado (*callee*)

⑤ Recuperar os *calle-saved registers*

```
107     ...  
108     ; passo 5 - depois do corpo do sub-programa  
109     ; recuperar registradores RBX, RDI, RSI, R12-15  
110     pop r15 ; ultimo empilhado  
111     pop r14  
112     pop r13  
113     ...
```


Passo-a-passo com exemplo a10e03.asm - PILHA

mem. alta

ARGV _{all}
ARGC
REG _{caller}
ret ADDR
old RBP
deslocador
REG _{calle}



Parâmetros em registradores:

1º

RDI

*strTeste1

RAX

resposta

Passo-a-passo com exemplo a10e02.asm - parte 10

■ Convenção de chamadas

■ Chamado (*callee*)

⑥ Desalocar todas as variáveis locais

```
134     ...  
135     ; passo 6 - depois do corpo do sub-programa  
136     ; desalocar variaveis locais  
137     mov rsp, rbp  
138     ...
```

Passo-a-passo com exemplo a10e03.asm - PILHA

mem. alta

ARGV _{all}
ARGC
REG _{caller}
ret ADDR
old RBP
deslocador
REG _{calle}



Parâmetros em registradores:

1º

RDI

*strTeste1

RAX

resposta

Passo-a-passo com exemplo a10e02.asm - parte 11

■ Convenção de chamadas

■ Chamado (*callee*)

⑦ Garantir endereço de retorno no topo da PILHA

```
138     ...  
139     ; passo 7 - depois do corpo do sub-programa  
140     ; recuperar rbp antigo, garantir endereço de retorno  
141     pop rbp  
142     ...
```

Passo-a-passo com exemplo a10e03.asm - PILHA

mem. alta

ARGV _{all}
ARGC
REG _{caller}
ret ADDR
old RBP
deslocador
REG _{calle}

← RSP

Parâmetros em registradores:

1º

RDI

*strTeste1

RAX

resposta

Passo-a-passo com exemplo a10e03.asm - parte 12

- Convenção de chamadas
 - Chamado (*callee*)
 - Recuperar os *calle-saved registers*
 - ⑧ Sempre retornar com instrução RET

```
142     ...  
143     ; passo 8 - depois do corpo do sub-programa  
144     ; sempre, sempre, sempre retorne com RET  
145     ret  
146     ...
```

Passo-a-passo com exemplo a10e03.asm - PILHA

mem. alta



← RSP

Parâmetros em registradores:

1º

RDI

*strTeste1

RAX

resposta

Passo-a-passo com exemplo a10e03.asm - parte 13

■ Convenção de chamadas

■ Chamador (caller)

■ Após instrução CALL

④ Remover parâmetros da PILHA

```
61      ...  
62      ; passo 4  
63      ; remover parametros da PILHA  
64      ; add rsp, 8 ; +8 para cada parametro empilhado  
65          ; porem nao existem neste exemplo  
66          ; 90% dos codigos utilizam  
67          ;     ate 6 parametros  
68      ...
```


Passo-a-passo com exemplo a10e03.asm - parte 14

■ Convenção de chamadas

■ Chamador (caller)

■ Após instrução CALL

⑤ recuperar os valores dos *caller-saved registers*

```
66      ...  
67      ; passo 5  
68      ; recuperar registradores RAX, RCX, RDX, R8-11  
69      pop r11 ; ultimo empilhado  
70      pop r10  
71      pop r9  
72      ...
```

Passo-a-passo com exemplo a10e03.asm - PILHA

mem. alta

ARGV _{all}
ARGC
REG _{caller}
ret ADDR
old RBP
deslocador
REG _{calle}

← RSP

Parâmetros em registradores:

1º

RDI

*strTeste1

RAX

resposta

Múltiplos arquivos

- Usado para organizar códigos extensos.
 - Separa o arquivo principal de seus subprogramas
 - Semelhante ao C com suas headers + arquivos

- Protocolo:

- ① Cria-se o código em arquivo separado
- ② Inclui-se o arquivo no local com o comando

```
%include "nomeDoArquivoSeparado.asm"
```

- Importante: arquivos devem estar no mesmo diretório, ou usar caminhos relativos ou estáticos no `%include`

Exemplo - a10e03b.asm

- Não será estudado passo-a-passo, pois é semelhante ao exemplo a10e03.asm
- Porém, utiliza o arquivo a10e03bfuncoes/a10e03bstrLengh.asm em anexo

```
75     ....
76 fim:
77     mov rax, _exit
78     mov rdi, 0
79     syscall
80
81     ; %include "a10e03bstrLengh.asm"
82     %include "a10e03bfuncoes/a10e03bstrLengh.asm"
83     mov [str1L], eax ; inteiro de 32bit
84     ...
```

Chamada de funções externas (C)

- Utilização de funções externas

- Declaração

```
extern nomeDaFuncao
```

- Chamada utilizando CALL

```
CALL nomeDaFuncao
```

- Passagem de parâmetro utilizando protocolo da arquitetura

Chamada de funções externas (C)

- Alteração no código .asm

```
section .text  
global _start
```

é alterado para:

```
section .text  
global main
```

- Agora temos a função main
 - que requer a criação do stack-frame

Exemplo a10e05.asm - printf em x64

■ Chamada para *printf* em arquitetura x64

```
29     ...
30     mov rdi, strCtrl ; controle para printf
31     lea rsi, [str1]  ; endereco str1
32     mov edx, [day]   ; conteudo 'day'
33     mov ecx, [year]; condeudo 'year'
34     call printf
35     ...
```

■ Montar:

```
nasm -f elf64 a10e05.asm
```

■ Linkar:

```
gcc -m64 -no-pie a10e05.o -o a10e05.x
```

Exemplo a10e05.asm - printf em x64

- Importante: Parâmetros para Ponto-Flutuante
 - São passados em registradores XMM0 (será visto numa futura aula)
 - Deve ser informado ao *printf* que não serão passados parâmetros P.F.

```
21     ...  
22     ; nao sera passado ponto-flutuante  
23     xor rax,rax ; numero de P.F. eh zero  
24     ...
```


Atividades Práticas

- a10at01: Usando o exemplo **a10e03.asm**, realize a chamada da função

```
int strlen(char *c[])
```

passando como parâmetro *strTeste2* e *strTeste3*

- Escreva o código das chamadas a partir da linha 60
- Guarde o resultado das chamadas em *str2L* e *str3L*

Atividade(s)

- a10at02: Criação de Funções (professor enlouqueceu)
 - Implementar um código que receba uma entrada do usuário e realize a conversão necessária:
maiúscula → minúscula
minúscula → maiúscula
 - Funções prontas (usar *extern*):
 - printf
 - scanf
 - Funções que devem ser criadas
 - `*char[] upperCase(char* c[])`
 - `*char[] lowerCase(char* c[])`
 - Arquitetura x64

Atividade(s)

- a10at02: Criação de Funções - Continuação
 - Comportamento:
 - Solicitar ao usuário para entrar com uma string
Usuário é esperto, maxChars = 25
e somente caracteres alfabéticos
 - Aplicar *upperCase* se string iniciar com minúscula
 - Aplicar *lowerCase* se string iniciar com maiúscula
 - Encerrar programa se string for exit (*non-case-sensitive*)
 - Exemplo:

```
$: ./a10at02.x
Entrada: <texTODOUuario>
Convert: <TEXTODOUSUARIO>
Entrada: <OutR0tEXTo>
Convert: <outrotexto>
Entrada: <ExIt>
2 convert(s), Farewell my friend!
$:
```

Atividade(s)

■ a10at03: Função Fatorial Recursivo

■ Crie um código que:

- Leia do teclado um inteiro
- Calcule o fatorial utilizando recursividade
- Mostre o resultado
- O código não é “Endless Factorial”



■ Exemplo:

```
$ ./a10at03.x
Fatorial de <15>
eh <1.307.674.368.000>
$:
```

- 15 é entrada do usuário
- 1.307.674.368.000 é o resultado

■ A importância do *stack-frame* é observado nesta atividade

