Linguagem de Montagem

Sub-Rotinas
Funções e Procedimentos
Aula 11

Edmar André Bellorini

Ano letivo 2016

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 $\rightarrow$ uso da pilha
    - x64 $\rightarrow$ 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
```

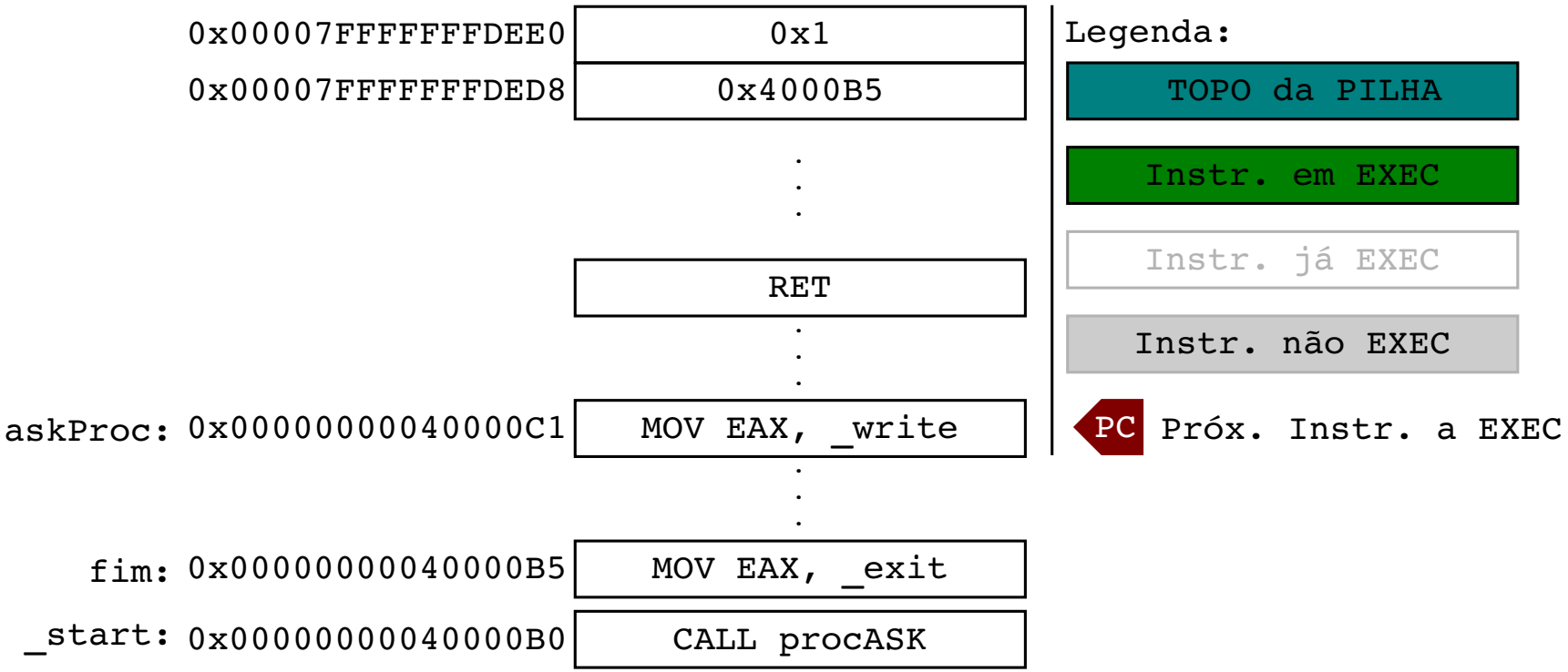
    - Endereço no topo da pilha **deve** ser valor empilhado por CALL
  - Executado por sub-programa **chamado** (*callee*)

Exemplo a11e01.asm - Chamada de procedimento

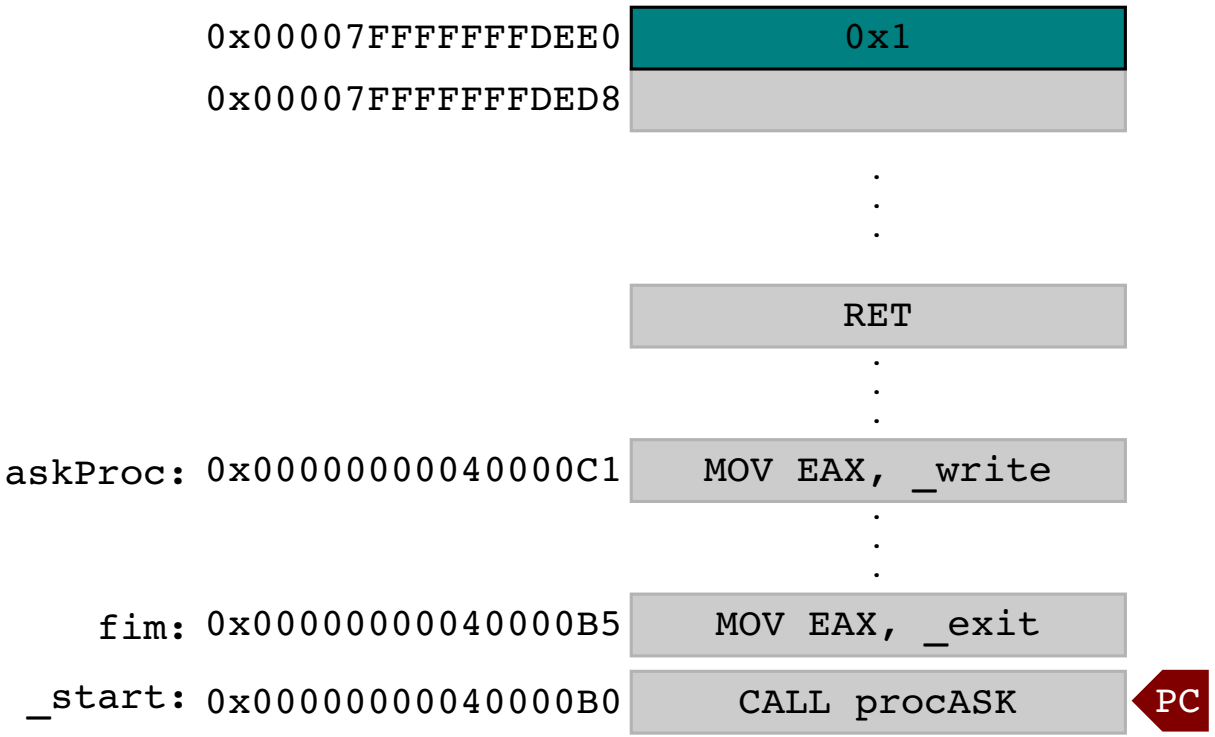
```
13    ...
14  _start:
15      ; PUSH fim
16      ; PC = procAsk
17      call procAsk
18      ...
```

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

Memória para a11e01.asm - parte 1

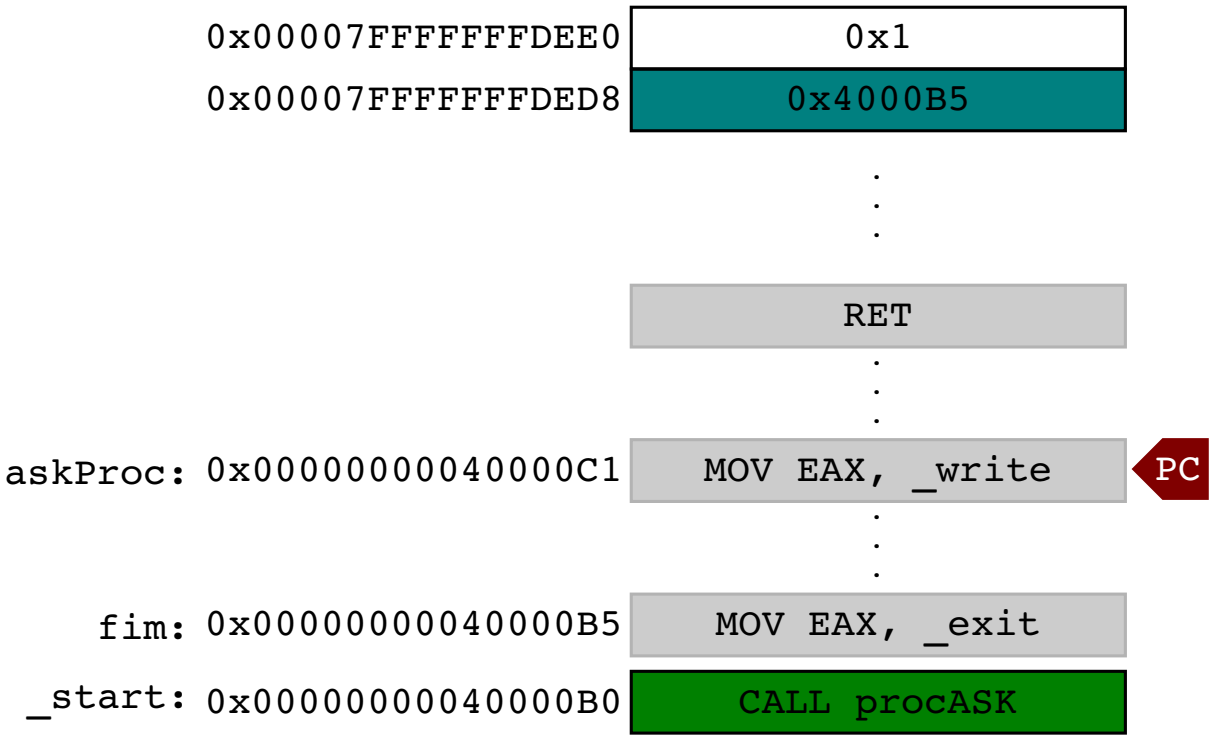


Memória para a11e01.asm - parte 2

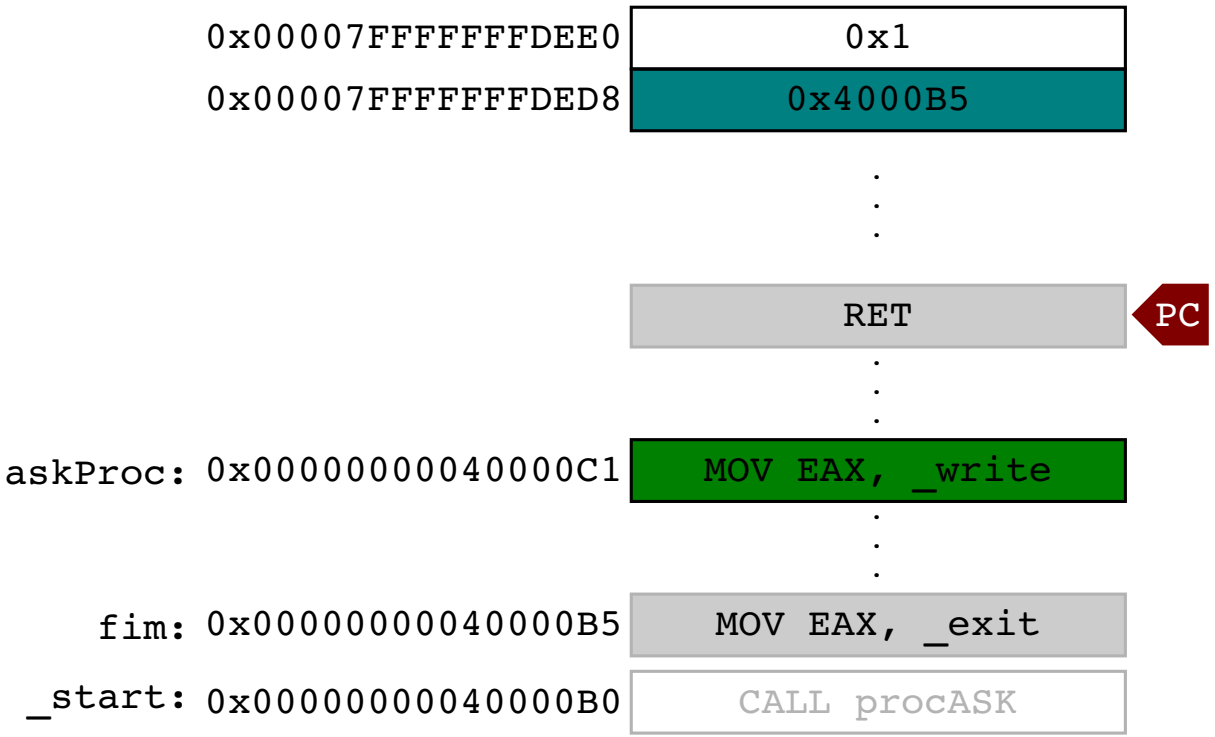


Memória para a11e01.asm - parte 3

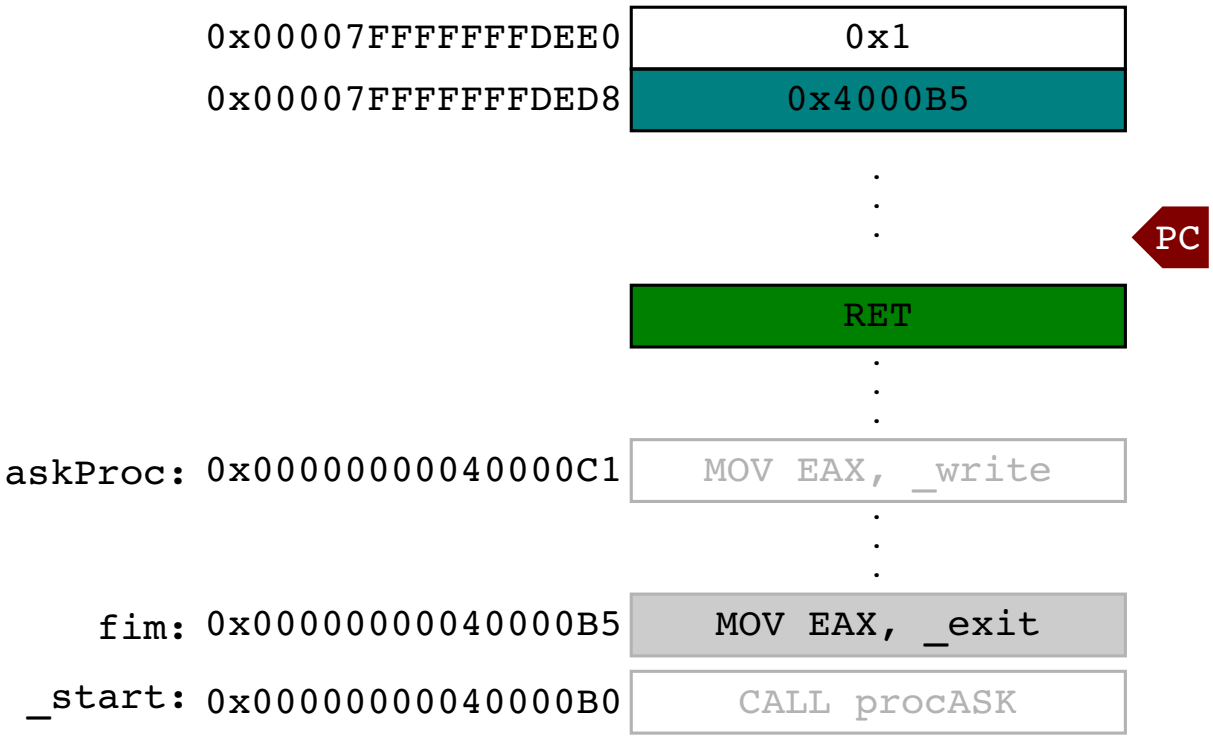
Memória para a11e01.asm - parte 4



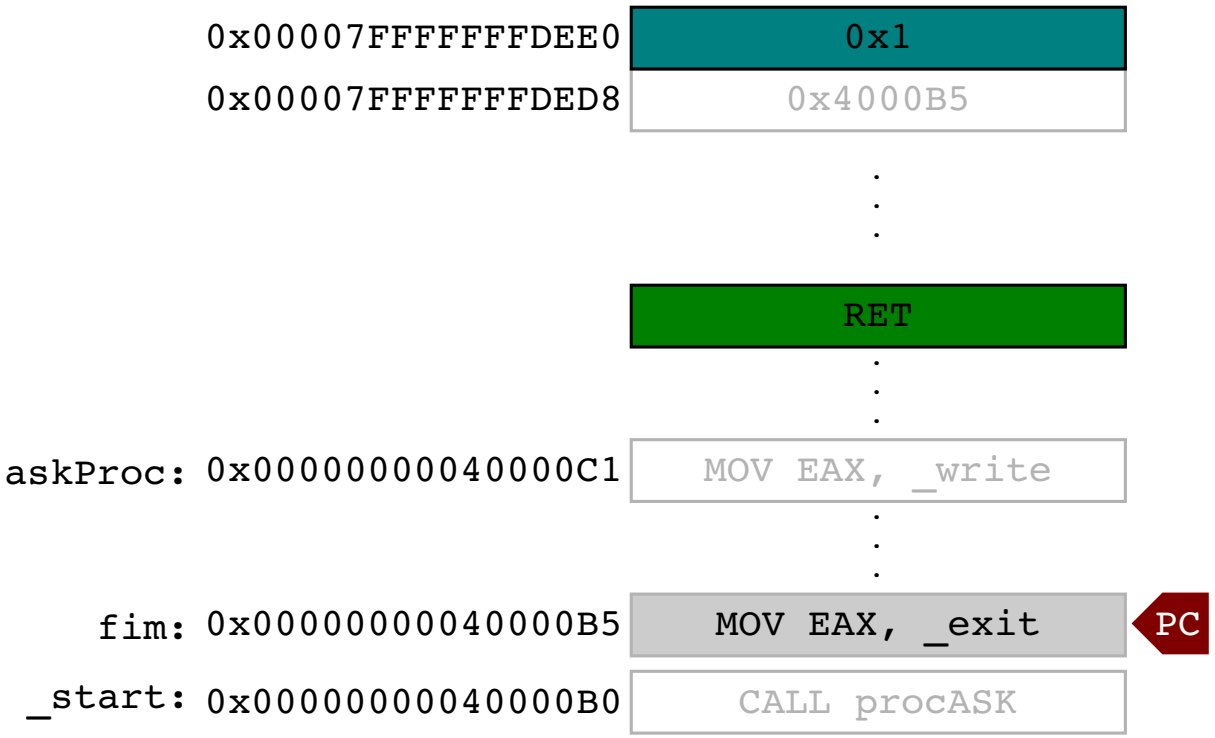
Memória para a11e01.asm - parte 5



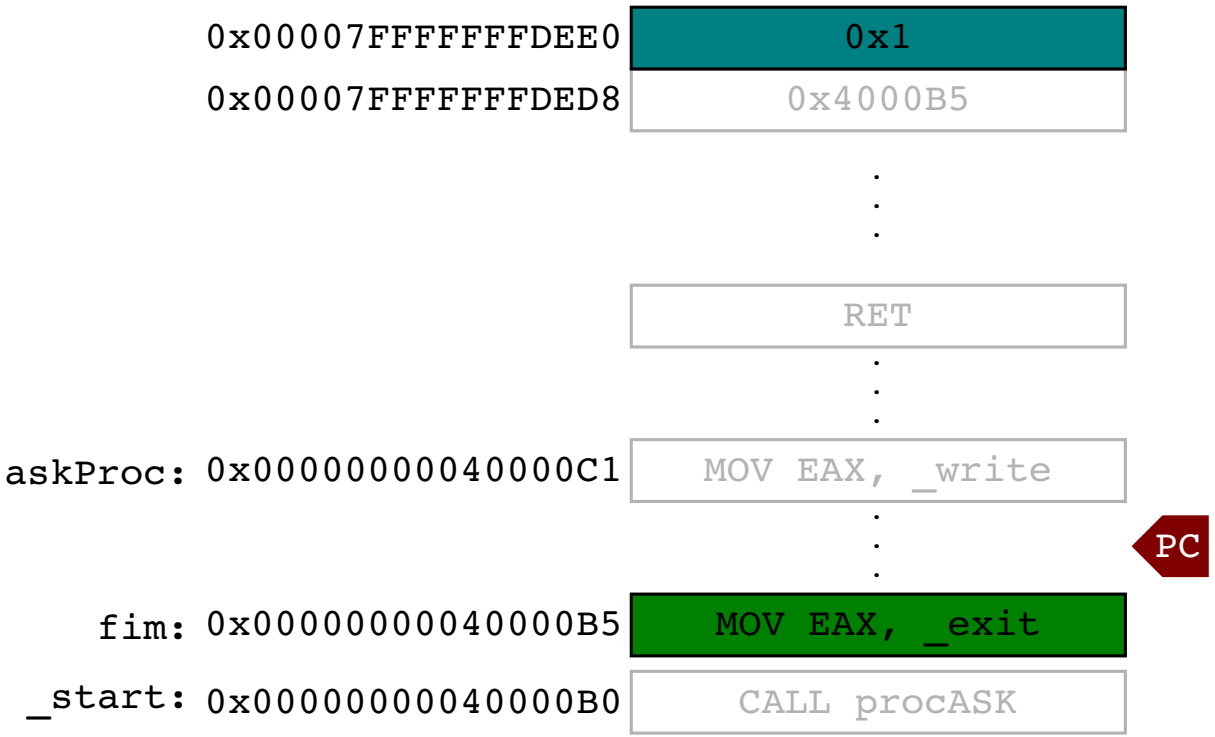
Memória para a11e01.asm - parte 6



Memória para a11e01.asm - parte 7



Memória para a11e01.asm - parte 8



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 a11e01.asm, insira o código abaixo na linha 31

```
31      push rax
```

Assim:

```
28      ...
29      int _kernel
30
31      push rax ; substituindo POP PC (!?)
32      ret
```

  - *segmentation fault*?

Protocolo x86 para chamadas de sub-programas

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

    1. Salvar *caller-saved registers* (se necessário)
       EAX, ECX e EDX
    2. Passagem de parâmetros para sub-programa via PILHA
    3. **Sempre** usar instrução CALL

    - **Após** instrução CALL

    4. Remover parâmetros da PILHA
    5. Recuperar os valores dos *caller-saved registers*

Protocolo x86 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)
   EBX, EDI, ESI

    - **Depois** da execução do corpo do sub-programa

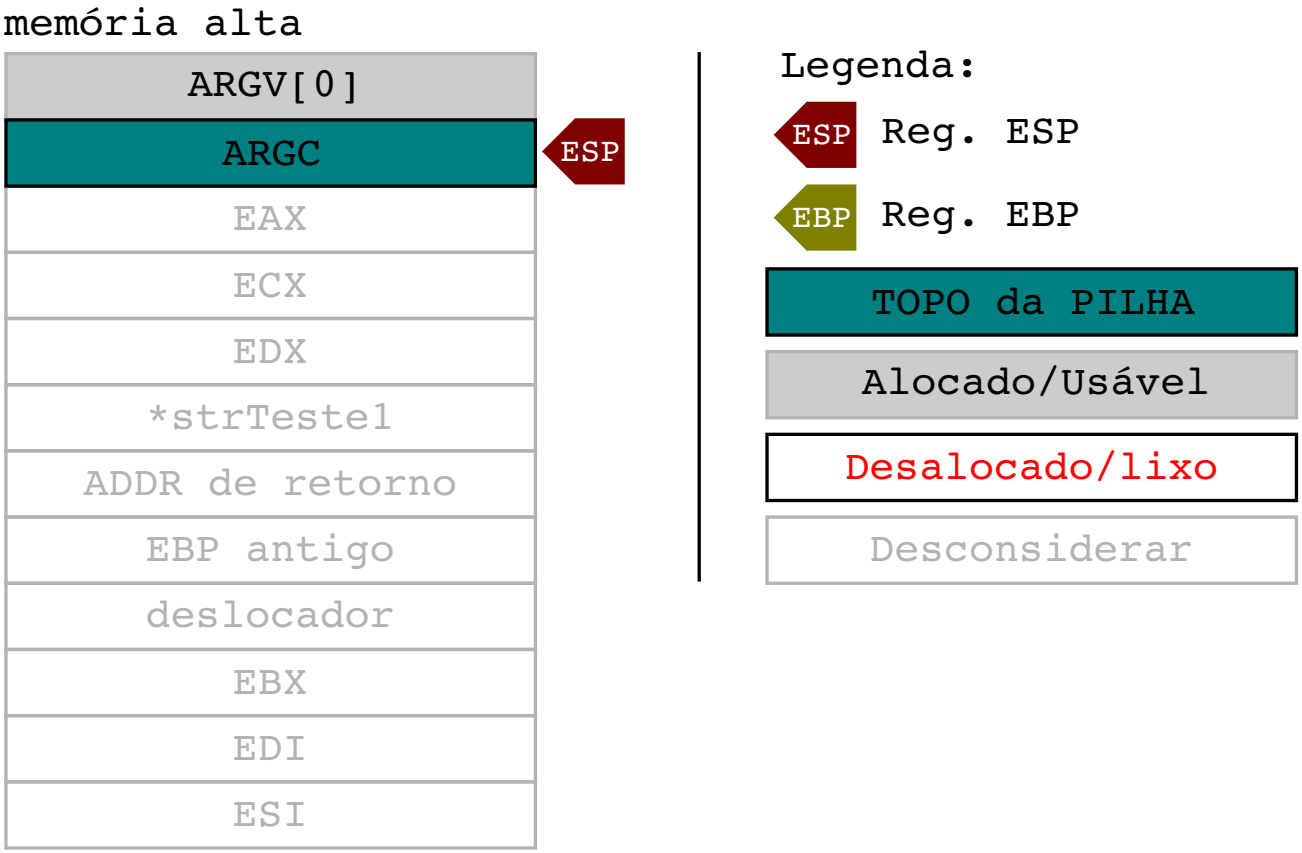
4. Deixar resultado/retorno do sub-programa em EAX
5. Recuperar os *calle-saved registers*
6. Desalocar todas as variáveis locais
7. Garantir endereço de retorno no topo da PILHA
8. **Sempre** retornar com instrução RET

Huge example - a11e02.asm

- Será visto passo-a-passo

```
38      ; passo 2 - Antes da chamada CALL
39      ; empilhar parametros da direita para esquerda
40      push strTeste1
41
42      ; passo 3 - chamada CALL
43      call strLength
44
45      ; retorno do sub-programa em EAX
46      mov [str1L], eax
```

Antes do passo-a-passo - estado inicial da PILHA



Passo-a-passo com exemplo a11e02.asm - parte 01

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

1. Salvar *caller-saved registers* (se necessário) EAX, ECX e EDX

```
32      ...
33      ; passo 1 - Antes da chamada CALL
34      ; salvar registradores EAX, ECX e EDX
35      push eax
36      push ecx
37      push edx
38      ...
```

Passo-a-passo com exemplo a11e02.asm - PILHA

Passo-a-passo com exemplo a11e02.asm - parte 02

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

    2. Passagem de parâmetros para sub-programa via PILHA

```
38      ...
39      ; passo 2 - Antes da chamada CALL
40      ; empilhar parametros da direita para esquerda
41      push strTeste1
42      ...
```

Passo-a-passo com exemplo a11e02.asm - PILHA

Passo-a-passo com exemplo a11e02.asm - parte 03

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

```
42      ...
43      ; passo 3 - chamada CALL
44      call strLength
45      ...
```

Passo-a-passo com exemplo a11e02.asm - PILHA

Passo-a-passo com exemplo a11e02.asm - parte 04

- Convenção de chamadas
  - **Chamado** (*callee*)
    - **Antes** de executar corpo de sub-programa

1. Criar *stack-frame*

```
66      ...
67      ; passo 1 - Antes do corpo do sub-programa
68      ; criar stack-frame
69      push ebp
70      mov ebp, esp
71      ...
```

Passo-a-passo com exemplo a11e02.asm - PILHA

Passo-a-passo com exemplo a11e02.asm - parte 05

- Convenção de chamadas
  - **Chamado** (*callee*)
    - **Antes** de executar corpo de sub-programa

    2. Alocar espaço na PILHA para variáveis locais

```
71      ...
72      ; passo 2 - Antes do corpo do sub-programa
73      ; criar espaco para variaveis locais
74      sub esp, 4  ; 4 bytes para variavel inteira local
75      ; [esp-4] eh o deslocador ate encontrar '0'
76      ...
```

Passo-a-passo com exemplo a11e02.asm - PILHA

Passo-a-passo com exemplo a11e02.asm - parte 06

- Convenção de chamadas
  - **Chamado** (*callee*)
    - **Antes** de executar corpo de sub-programa

    2. *calle-saved registers* (se necessário)
       EBX, EDI, ESI

```
76      ...
77      ; passo 3 - Antes do corpo do sub-programa
78      ; salvar registradores EBX, EDI e ESI
79      push ebx
80      push edi
81      push esi
82      ...
```

Passo-a-passo com exemplo a11e02.asm - PILHA

Passo-a-passo com exemplo a11e02.asm - parte 07

- Convenção de chamadas
  - **Chamado** (*callee*)
    - sub-programa
    - parâmetro *char[ ] → [ebp+8]
    - variável local → [ebp-4]

```
83      ...
84      mov ecx, [ebp+8] ; char *c[ ] - parametro 1
85      mov dword [ebp-4], 0
86      laco:
87         mov edx, [ebp-4]   ; deslocador
88         mov al, [ecx+edx]  ; char[deslocador]
89         ; inc edx            ; para contar '\0'
90         cmp al, 0          ; char eh zero?
91         je finaliza
92         inc edx            ; desloc++
93         mov [ebp-4], edx   ; guarda deslocador
94         jmp laco
95      ...
```

Passo-a-passo com exemplo a11e02.asm - PILHA

Passo-a-passo com exemplo a11e02.asm - parte 08

- Convenção de chamadas
  - **Chamado** (*callee*)
    - **Antes** de executar corpo de sub-programa
    4. Deixar resultado/retorno do sub-programa em EAX

```
98      ...
99      ; passo 4 - depois do corpo do sub-programa
100     ; copiar resultado para EAX
101     mov eax, [ebp-4]
102     ...
```

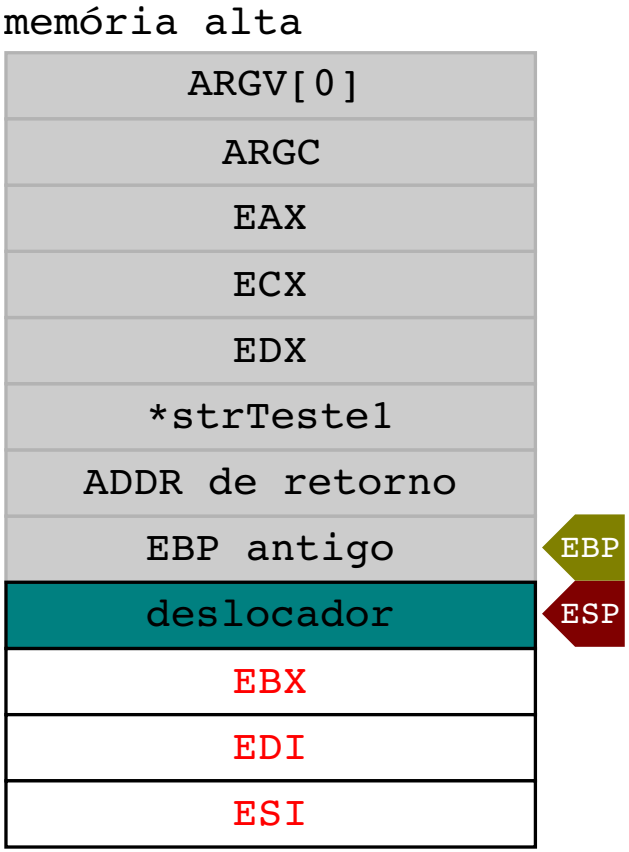
Passo-a-passo com exemplo a11e02.asm - parte 09

- Convenção de chamadas
  - **Chamado** (*callee*)
    - Recuperar os *calle-saved registers*

    5. Deixar resultado/retorno do sub-programa em EAX

```
102     ...
103     ; passo 5 - depois do corpo do sub-programa
104     ; recuperar registradores EBX, EDI e ESI
105     pop esi ; ultimo empilhado
106     pop edi
107     pop ebx ; primeiro empilhado
108     ...
```

Passo-a-passo com exemplo a11e02.asm - PILHA



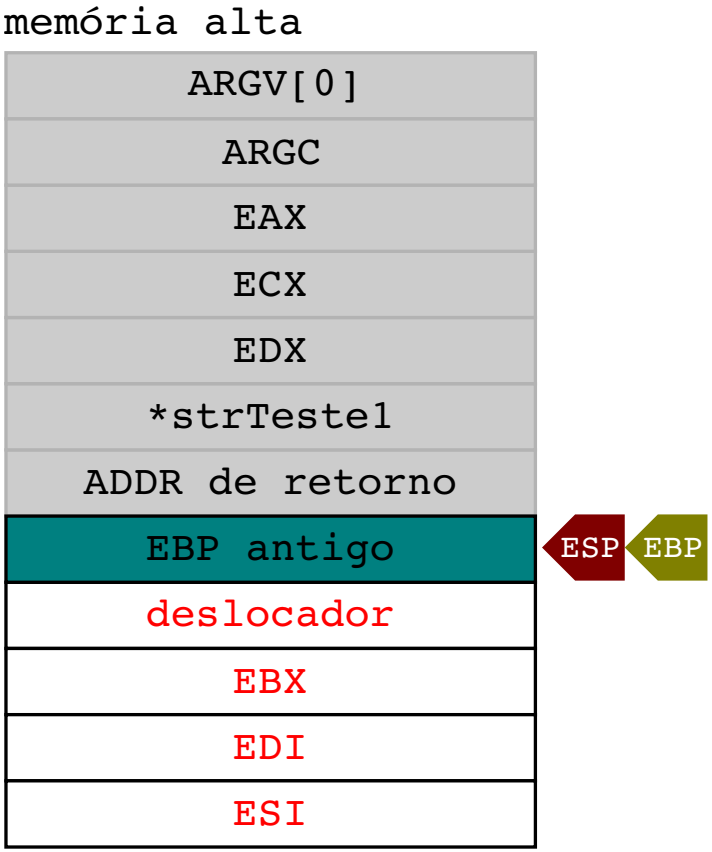
Passo-a-passo com exemplo a11e02.asm - parte 10

- Convenção de chamadas
  - **Chamado** (*callee*)
    - Recuperar os *calle-saved registers*

    6. Desalocar todas as variáveis locais

```
108     ...
109     ; passo 6 - depois do corpo do sub-programa
110     ; desalocar variaveis locais
111     mov esp, ebp
112     ...
```

Passo-a-passo com exemplo a11e02.asm - PILHA



Passo-a-passo com exemplo a11e02.asm - parte 11

- Convenção de chamadas
  - **Chamado** (*callee*)
    - Recuperar os *calle-saved registers*

    7. Garantir endereço de retorno no topo da PILHA

```
108     ...
109     ; passo 7 - depois do corpo do sub-programa
110     ; recuperar ebp antigo, garantir endereco de retorno
111     pop ebp
112     ...
```

Passo-a-passo com exemplo a11e02.asm - PILHA

Passo-a-passo com exemplo a11e02.asm - parte 12

- Convenção de chamadas
  - **Chamado** (*callee*)
    - Recuperar os *calle-saved registers*

    8. **Sempre** retornar com instrução RET

```
112     ...
113     ; passo 8 - depois do corpo do sub-programa
114     ; sempre, sempre, sempre retorne com RET
115     ret
116     ...
```

Passo-a-passo com exemplo a11e02.asm - PILHA

Passo-a-passo com exemplo a11e02.asm - parte 13

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

    4. Remover parâmetros da PILHA

```
48      ...
49      ; passo 4
50      ; remover parametros da PILHA
51      add esp, 4 ; +4 para cada parametro empilhado
52      ...
```

Passo-a-passo com exemplo a11e02.asm - PILHA

Passo-a-passo com exemplo a11e02.asm - parte 14

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

5. recuperar os valores dos *caller-saved registers*

```
52      ...
53      ; passo 5
54      ; recuperar registradores EAX, ECX e EDX
55      pop edx  ; ultimo empilhado
56      pop ecx
57      pop eax  ; primeiro empilhado
58      ...
```

Passo-a-passo com exemplo a11e02.asm - PILHA

Finalizando o passo-a-passo com exemplo a11e02.asm

- Material de apoio para seção *x*86

  ▸ x86 Assembly Guide

  - Acessado em 29/06/2016
  - Obs.: O guia utiliza modelo de memória FLAT do 486
    Não abordado na disciplina
    Algumas coisas podem parecer estranhas
- Na sequência: sub-programas em x64

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 - a11e03.asm

- Não será estudado passo-a-passo, pois é semelhante ao exemplo a11e02.asm

```
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    ...
```

Chamada de funções externas (C)

- Utilização de funções externas
  - Declaração

```
extern nomeDaFuncao
```

  - Chamada utilizando CALL

```
CALL nomeDaFuncao
```

  - Passagem de parâmetro utilizando protocolo da arquitetura

- Alteração no código .asm

```
section .text
  global _start
```

é alterado para:

```
section .text
  global main
```

Exemplo a11e04.asm - printf em x86

- Chamada para *printf* em arquitetura x86

```
22    ...
23    push umaStr ; endereco para string 'umaStr'
24    push dword [umInt] ; conteudo do inteiro 'umInt'
25    push strCtrl ; string de controle para printf
26    call printf
27    ...
```

- Montar:

```
nasm -f elf32 a11e04.asm
```

- Linkar:

```
gcc -m32 a11e04.o -o a110e04.x
```

Exemplo a11e05.asm - printf em x64

- Chamada para *printf* em arquitetura x64

```
26    ...
27    mov rdi, strCtrl ; string de controle para printf
28    mov esi, [umInt] ; conteudo do inteiro 'umInt'
29    mov rdx, umaStr  ; endereco para string 'umaStr'
30    call printf
31    ...
```

- Montar:

```
nasm -f elf64 a11e04.asm
```

- Linkar:

```
gcc -m64 a11e04.o -o a110e04.x
```

Exemplo a11e05.asm - printf em x64

- Importante: Parâmetros para Ponto-Flututante
  - 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      ...
```

Exercícios de Fixação

- EF1101: Usando o exemplo **a11e02.asm**, realize a chamada da função

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

passando como parâmetro *strTeste2* e *strTeste3*

  - Escreva o código das chamadas a partir da linha 58
  - Guarde o resultado das chamadas em *str2L* e *str3L*

- EF1102: Refaça o exercício EF1101 para arquitetura x64

Exercícios de Fixação

- EF1103: 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 pode ser x86 ou x64
    - Escolha uma arquitetura e use o protocolo desta

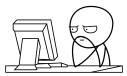
Exercícios de Fixação

- EF1103: 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:

```
$: ./EF1103.x
Entrada: <texTODOUsuario>
Convert: <TEXTODOUSUARIO>
Entrada: <OutROtEXTo>
Convert: <outrotexto>
Entrada: <ExIt>
2 converts, Farewell my friend!
$:
```

Exercícios de Fixação

- EF1104: 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:

```
$ ./EF1104.x
Fatorial de <15>
eh <1.307.674.368.000>
$:
```

    - 15 é entrada do usuário
    - 1.307.674.368.000 é o resultado

Relatório

- Somente relatório
  - O modelo de relatório para a disciplina de LM está disponível em anexo da Aula 01
    - Arquivo **modeloRelatorioLM.odt**
    - A data do relatório é a data de entrega (ver moodle)
  - Somente serão aceitos os relatórios em formato .pdf com nome do arquivo seguindo o padrão: **TY.PXX.nome.sobrenome.pdf**
    - TY é o número da turma prática (1, 2, 3 ou 4)
    - PXX é o número da prática, neste caso: P11
  - Ex.: aluno Malcolm "Mal" Reynolds da turma prática 9 (de 2517):
    - T9.P11.Malcolm.Reynolds.pdf