MATA49
Programação de Software Básico

Procedimentos

Leandro Andrade
leandrojsa@ufba.br

Procedimentos

- Assim como nas linguagens de alto nível, os procedimentos tem a função organizar o código e evitar trechos de código redundantes
- Procedimentos são subrotinas que podem ser invocadas durante a execução de um programa

Procedimentos

- Não possui declaração de parâmetros explícitos
  - Os dados são passados através de convenções, que são regras que definem um padrão para passagem de parâmetros

Procedimentos

- Declaração de procedimentos

```
; subprogram get_int
; Parameters:
;   ebx - address of dword to store integer into
;   ecx - address of instruction to return to
; Notes:
;   value of eax is destroyed
get_int:
        call    read_int
        mov     [ebx], eax          ; store input into memory
        jmp     ecx                 ; jump back to caller
```

sub1.asm

Procedimentos

- Invocação de procedimentos:
  - Solução 1: Saltos para o trecho de código da rotina
    - Ex: JMP get_int
    - Porém como retornar ao fim da execução do precedimento?
      - Outro salto?
      - E se houver mais de uma chamada ao procedimento, como saber qual ponto retornar?

Procedimentos

- Invocação de procedimentos:
  - Solução 2: Endereçamento indireto
    - Armazenar em um registrador o endereço de retorno do código

```
mov     ebx, input2
mov     ecx, $ + 7          ; ecx = this address + 7
jmp     short get_int
```

    - Problemas! Cálculo para retorno não é trivial

Procedimentos

- Invocação de procedimentos:
  - Solução 3: Instrução call e ret
    - Call: Realiza um salto incondicional para um subprograma e empliha o o endereço da próxima instrução (a instrução após o call)
    - Sintaxe: call <Nome subprograma>
    - Ret: Desempilha e realiza um salto incondicional para o valor desempilhado
    - Sintaxe: ret

Procedimentos

- Instrução call e ret:

```
        mov     ebx, input1
        call    get_int

        mov     ebx, input2
        call    get_int
```

and change the subprogram `get_int` to:

```
get_int:
        call    read_int
        mov     [ebx], eax
        ret
```

Procedimentos

- Instrução call e ret:
  - Atenção! Quando você empilha alguma informação dentro de uma subrotina é preciso desempilhar antes de executar o **ret**

```
get_int:
        call    read_int
        mov     [ebx], eax
        push    eax
        ret                     ; pops off EAX value, not return address!!
```

Ele desempilha o eax e salta para esse valor
Como não é um endereço válido resulta em
Falha de segmentação!

Procedimentos

- Passagem de parâmetros por pilha
  - Convenção da linguagem C
    - Apresenta boas práticas de como usar parâmetros em Assembly
    - São usadas pelos compiladores C

Procedimentos

- Passagem de parâmetros por pilha
  - Invocando uma função:
    - Antes de chamarmos uma função devemos empilhar os seus parâmetros seguindo uma ordem correta
    - Após a chamada da função desempilhamos esses parâmetros
      - É fundamental fazer isso, pois muito problemas podem surgir sem esse passo

Procedimentos

- Passagem de parâmetros por pilha
  - Invocando uma função:
    - Exemplo:

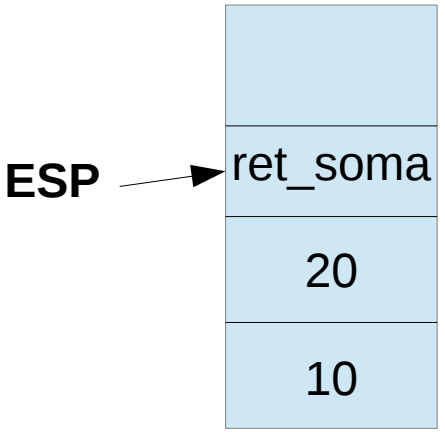
```
soma(int x, int y)...

push 10  ;passagem do parametro1
push 20 ;passagem do parametro2
call  soma
pop ecx ;desempliha o paramentro2
pop ecx ;desempliha o paramentro1
```

Procedimentos

- Convenção da linguagem C
  - Uso de parâmetros na construção da função
    - Um programa que soma dois números:

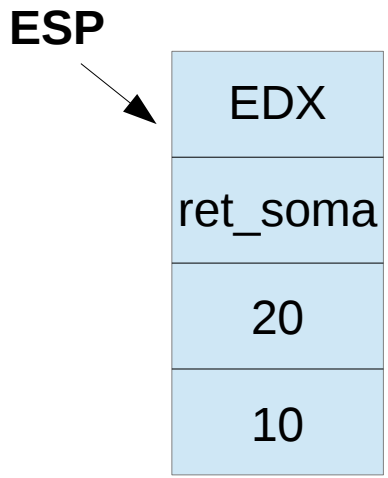
```
soma:
    mov eax, [esp + 8] ;acesso ao paramentro1
    mov ebx, [esp + 4] ;acesso ao paramentro2
    add eax, ebx
    ret
```



Procedimentos

- Convenção da linguagem C
  - É possível que o programador empilhe valores durante a execução do sub-programa e isso pode alterar a forma de obter os parâmetros com o ESP
  - Considerando o exemplo anterior:

```
soma:
    push edx ; por alguma razão o EDX é empilhado
    mov eax, [esp + 8] ;O valor correto seria [esp + 12]
    mov ebx, [esp + 4] ;O valor correto seria [esp + 8]
    add eax, ebx
    pop ecx
    ret
```



Procedimentos

- Convenção da linguagem C
  - Solução:

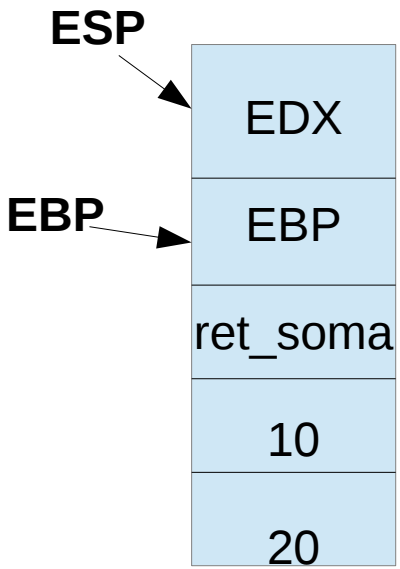
    Ao iniciar qualquer sub-rotina empilhar o **EBP** e atribuir (mov) o EBP o valor do ESP. Assim usando o EBP para pegar os parâmetros

    A não utilização desta convenção motiva muitos erros no uso dos parâmetros

Procedimentos

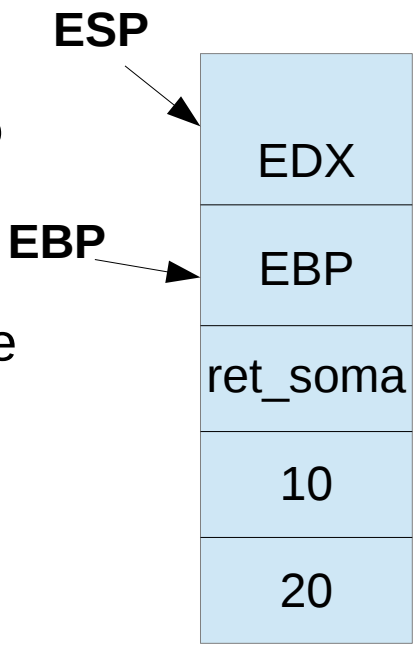
- Convenção da linguagem C
  - Exemplo anterior:

```
soma:
    push ebp
    mov ebp, esp
    push edx ; não altera mais o acesso aos parâmetros pois são feitos com o ebp
    mov eax, [ebp + 12] ;acesso ao paramentro1
    mov ebx, [ebp + 8] ;acesso ao paramentro2
    add eax, ebx
    pop ecx   ; desempilha o edx e armazena no ecx
    pop ebp  ; desempilha o ebp e armazena no ebp
    ret
```



Procedimentos

- Convenção da linguagem C
  - Em outras palavras fazer isso nos permite posicionar o EBP como **um ponto de referência** para acessar nosso parâmetros
  - **Por que empilhar o EBP e não simplesmente fazê-lo apontar para o primeiro parâmetro?**
    - Quando empilhamos o EBP e no final do código da função executamos “pop EBP” nos permite o uso de chamada recursivas, ou até mesmo funções que chama outras funções.
    - Assim o EBP pode ser usado para chamadas de funções sequencias



Procedimentos

- Convenção da linguagem C
  - Assim podemos convencionar que toda função deve seguir a seguinte estrutura:

```
funcao:
   push ebp
   mov ebp, esp

   ; codigo da funcao

   pop ebp
ret
```

Procedimentos

- Convenção da linguagem C
  - Retorno da função
    - Por convenção os retornos das funções são passados no registrador eax

Procedimentos

- Convenção da linguagem C
  - Exercício:

    Faça uma função que calcula o delta da fórmula de Bhaskara e faça um código que invoca esta função para seguintes valores: a=5; b= 10; c=15
  - Lembrado que delta = b^2 - 4ac

Procedimentos

- Convenção da linguagem C
  - Exercício:

```
delta:
    push ebp
    mov ebp, esp

    mov eax, [ebp+16]
    mov ebx, [ebp+12]
    mov ecx, [ebp+8]

    imul ebx, ebx
    imul eax, 4
    imul eax, ecx
    sub ebx, eax
    mov eax, ebx

    pop ebp
    ret
```

```
push 5
push 10
push 15
call delta
pop ecx
pop ecx
pop ecx
; em vez de fazer 3 pops basta
; fazer add esp, 12
```

Procedimentos

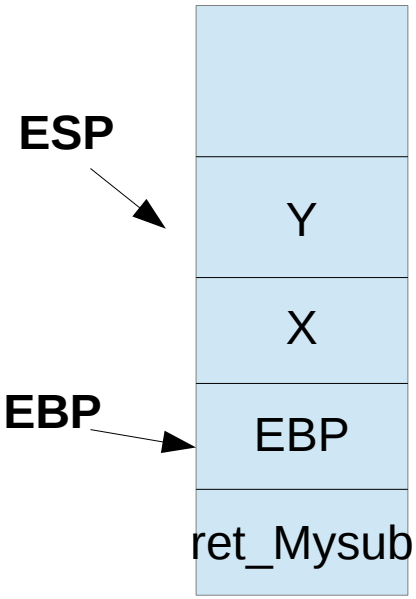
- Convenção da linguagem C
  - Variáveis Locais
    - As variáveis locais são definidas na na pilha de execução do programa
    - É reservado um espaço para armazená-las acima do registrador EBP
    - No procedimento abaixo precisamos reservar 8 bytes para armazenar as duas variáveis

```
void MySub(){
    int X = 10;
    int Y = 20;
}
```

Procedimentos

- Convenção da linguagem C
  - Variáveis Locais
    - Assim temos o seguinte código:

```
void MySub(){
    int X = 10;
    int Y = 20;
}
```




```
MySub:
    push ebp
    mov ebp,esp

    sub esp,8  ; aqui é reservado o espaço
de 8 bytes
    mov [ebp-4],eax ; x = eax
    mov [ebp-8],ebx ; y = ebx

    mov esp,ebp ;remove as vars locais
    pop ebp
ret
```

Procedimentos

- Convenção da linguagem C
  - Instrução ENTER
    - Reserva o espaço para variáveis locais e salva o EBP na pilha
    - Mais especificadamente executa 3 ações:
      - push EBP
      - mov EBP, ESP
      - sub ESP, numbytes (reserva o espaço para variáveis locais)

Procedimentos

- Convenção da linguagem C
  - Instrução ENTER
    - Sintaxe:
      - ENTER num_bytes, nesting_level
        - **num_bytes**: número de bytes reservados para variáveis locais
        - nesting_level: niveis de aninhamentos (para nosso programas sempre será zero)
      - Os parâmetros são sempre números inteiros

Procedimentos

- Convenção da linguagem C
  - Instrução ENTER
    - Observado o exemplo anterior temos o seguinte código usando a instrução ENTER

```
MySub:
    push ebp
    mov ebp,esp

    sub esp,8
    mov [ebp-4],eax
    mov [ebp-8],ebx
```



```
MySub:
    enter 8,0

    mov [ebp-4],eax
    mov [ebp-8],ebx
```

Procedimentos

- Convenção da linguagem C
  - Instrução LEAVE
    - Termina o espaço reservado para variáveis locais e retira o EBP da pilha
    - Executa as seguintes ações:
      - mov esp, ebp
      - pop ebp
    - Sintaxe:
      - Não possui parâmetros

Procedimentos

- Convenção da linguagem C
  - Instrução LEAVE
    - Exemplo:

```
MySub:
    push ebp
    mov ebp,esp

    sub esp,8
    mov [ebp-4],eax
    mov [ebp-8],ebx

    mov esp,ebp
    pop ebp
ret
```



```
MySub:
    enter 8,0

    mov [ebp-4],eax
    mov [ebp-8],ebx

    leave
ret
```

Procedimentos

- Chamadas recursivas a procedimentos
  - Ocorre quando uma sub-rotina chama ela mesmo
  - É necessário ter atenção com o empilhamento de parâmetros e dados
  - Exemplo:

```
proc:
    ...
    call proc
ret
```

Procedimentos

```
1   ; finds n!
2   segment .text
3         global _fact
4   _fact:
5         enter  0,0
6
7         mov    eax, [ebp+8]     ; eax = n
8         cmp    eax, 1
9         jbe    term_cond        ; if n <= 1, terminate
10        dec    eax
11        push   eax
12        call   _fact            ; eax = fact(n-1)
13        pop    ecx              ; answer in eax
14        mul    dword [ebp+8]    ; edx:eax = eax * [ebp+8]
15        jmp    short end_fact
16  term_cond:
17        mov    eax, 1
18  end_fact:
19        leave
20        ret
```