



Estrutura do tema ISA do IA-32

1. Desenvolvimento de programas no IA-32 em Linux
2. Acesso a operandos e operações
3. Suporte a estruturas de controlo
4. Suporte à invocação/regresso de funções
5. Análise comparativa: IA-32, x86-64 e MIPS (RISC)
6. Acesso e manipulação de dados estruturados

Suporte a funções e procedimentos no IA-32 (1)



Estrutura de uma função (/ procedimento)

- **função versus procedimento** (ou ainda **rotina**, em Fortran)
 - o nome duma função é usado como se fosse uma variável
 - uma função devolve um valor, um procedimento não
- **parte visível ao programador em HLL**
 - o código do corpo da função
 - a passagem de parâmetros/argumentos para a função ...
... e o valor devolvido pela função
 - o alcance das variáveis: locais, externas ou globais
- **parte não visível em HLL** (gestão do contexto da função)
 - variáveis locais (propriedades)
 - variáveis externas e globais (localização e acesso)
 - parâmetros/argumentos e valor a devolver pela função (propriedades)
 - gestão do contexto (controlo & dados)



Análise do contexto de uma função

- **propriedades das variáveis locais:**
 - visíveis apenas durante a execução da função
 - deve suportar aninhamento e recursividade
 - localização ideal (escalares): em registo, se os houver...
 - localização no código em IA-32: em registo, enquanto houver...
- **variáveis externas e globais:**
 - externas: valor ou localização expressa na lista de argumentos
 - globais: localização definida pelo *linker & loader* (IA-32: na memória)
- **propriedades dos parâmetros/arg's** (só de entrada em C):
 - por valor (c^{te} ou valor da variável) ou por referência (localização da variável)
 - designação independente (f. chamadora / f. chamada)
 - deve ...
 - localização ideal: ...
 - localização no código em IA-32: ...
- **valor a devolver pela função:**
 - é ...
 - localização: ...
- **gestão do contexto ...**

Designação independente dos parâmetros



<pre>void swap(int *xp, int *yp) { int t0 = *xp; int t1 = *yp; *xp = t1; *yp = t0; }</pre>
<pre>void call_swap() { int zip1 = 15213; int zip2 = 91125; (...) swap(&zip1, &zip2); (...) }</pre>



Análise do contexto de uma função

- **propriedades das variáveis locais:**
 - visíveis apenas durante a execução da função
 - deve suportar aninhamento e recursividade
 - localização ideal (escalares): em registo, se os houver...
 - localização no código em IA-32: em registo, enquanto houver...
- **variáveis externas e globais:**
 - externas: valor ou localização expressa na lista de argumentos
 - globais: localização definida pelo *linker* & *loader* (IA-32: na memória)
- **propriedades dos parâmetros/arg's** (só de entrada em C):
 - por valor (*c^{te}* ou valor da variável) ou por referência (localização da variável)
 - designação independente (f. chamadora / f. chamada)
 - deve suportar aninhamento e recursividade
 - localização ideal: em registo, se os houver; mas...
 - localização no código em IA-32: na memória (na *stack*)
- **valor a devolver pela função:**
 - é uma quantidade escalar, do tipo inteiro, real ou apontador
 - localização: em registo (IA-32: *int* no registo *eax* e/ou *edx*)
- **gestão do contexto** (controlo & dados) ...



Análise do código de gestão de uma função

- **invocação e regresso**
 - instrução de salto, mas salvaguarda endereço de regresso
 - em registo (RISC; aninhamento / recursividade ?)
 - em memória/na *stack* (IA-32; aninhamento / recursividade ?)
- **invocação e regresso**
 - instrução de salto para o endereço de regresso
- **salvaguarda & recuperação de registos** (na *stack*)
 - função chamadora ? (nenhum/ alguns/ todos ? RISC/IA-32 ?)
 - função chamada? (nenhum/ alguns/ todos ? RISC/IA-32 ?)
- **gestão do contexto** ...



Utilização dos registos (de inteiros)

– Três do tipo *caller-save*

%eax, %edx, %ecx

- *save/restore*: função chamadora

– Três do tipo *callee-save*

%ebx, %esi, %edi

- *save/restore*: função chamada

– Dois apontadores (para a *stack*)

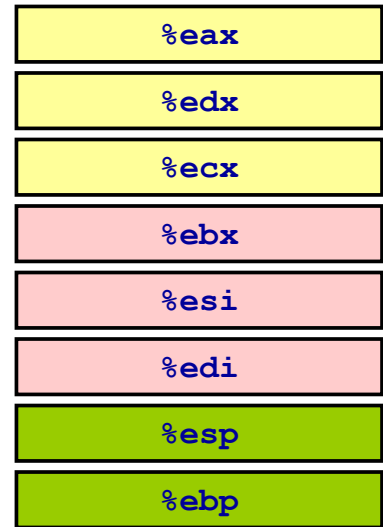
%esp, %ebp

- topo da *stack*, base/referência na *stack*

Caller-Save

Callee-Save

Pointers



Nota: valor a devolver pela função vai em **%eax**

Suporte a funções e procedimentos no IA-32 (3)



Análise do código de gestão de uma função

– invocação e regresso

- instrução de salto, mas salvaguarda endereço de regresso
 - em registo (RISC; aninhamento / recursividade ?)
 - em memória/na *stack* (IA-32; aninhamento / recursividade ?)

– invocação e regresso

- instrução de salto para o endereço de regresso

– salvaguarda & recuperação de registos (na *stack*)

- função chamadora ? (nenhum/ alguns/ todos ? RISC/IA-32 ?)
- função chamada ? (nenhum/ alguns/ todos ? RISC/IA-32 ?)

– gestão do contexto (em *stack*, em *activation record* ou *frame*)

- reserva/libertação de espaço para variáveis locais
- atualização/recuperação do *frame pointer* (IA-32...)



Análise de exemplos

– revisão do exemplo swap

- análise das fases: inicialização, corpo, término
- análise dos contextos (IA-32)
- evolução dos contextos na *stack* (IA-32)

– evolução de um exemplo: Fibonacci

- análise ...

– aninhamento e recursividade

- evolução ...

Análise das fases em swap, no IA-32 (fig. já apresentada)



```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

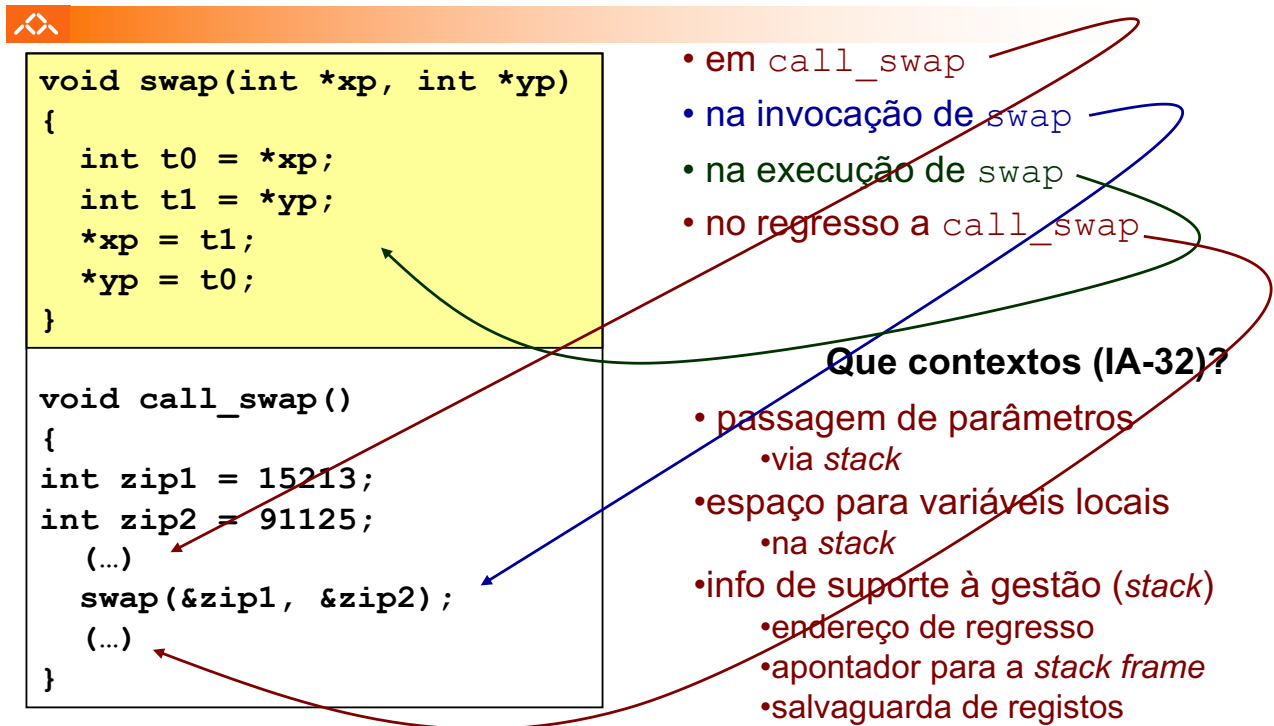
```
swap:
    pushl %ebp
    movl  %esp,%ebp
    pushl %ebx
    }
    movl  12(%ebp),%ecx
    movl  8(%ebp),%edx
    movl  (%ecx),%eax
    movl  (%edx),%ebx
    movl  %eax,(%edx)
    movl  %ebx,(%ecx)
    }
    movl  -4(%ebp),%ebx
    movl  %ebp,%esp
    popl  %ebp
    ret
    }
```

Arranque

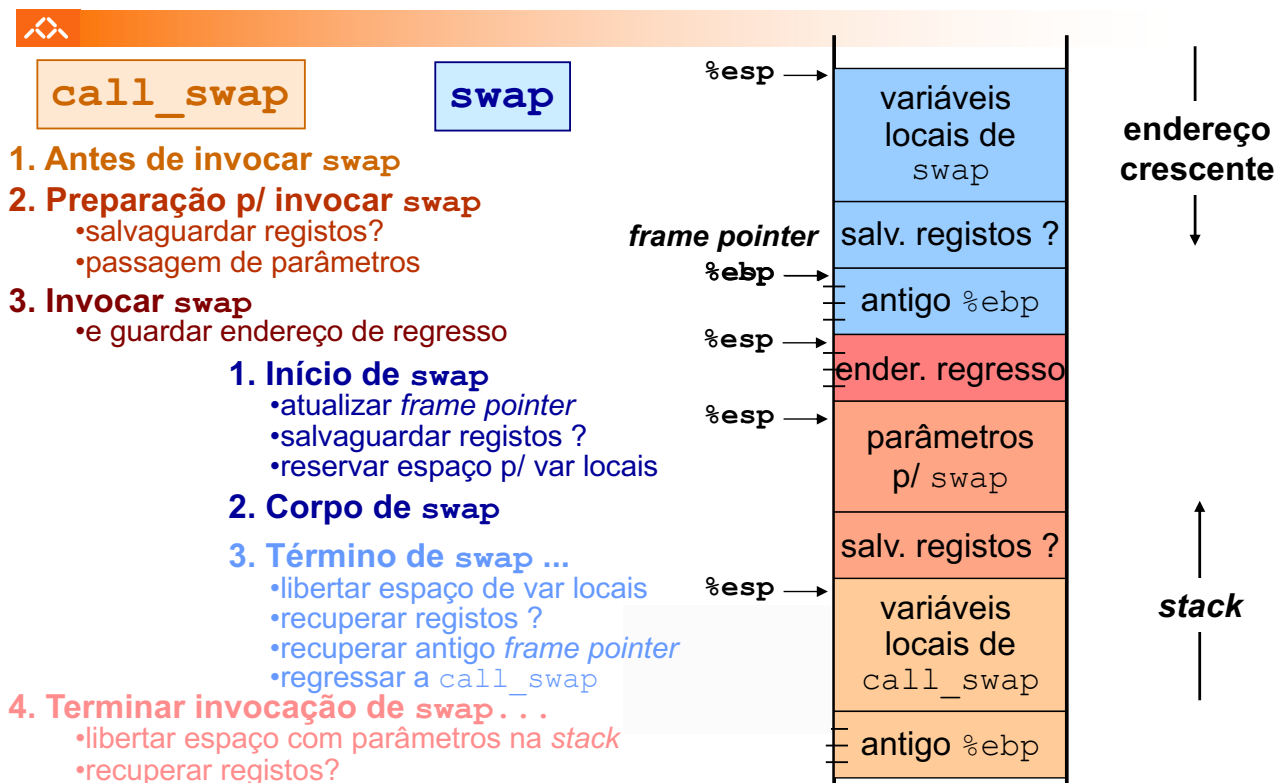
Corpo

Término

Análise dos contextos em swap, no IA-32



Construção do contexto na stack, no IA-32



Evolução da stack, no IA-32 (1)

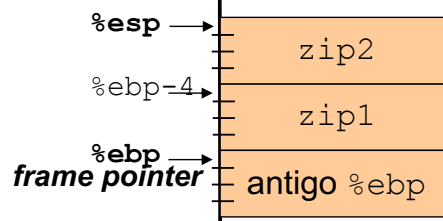


1. Antes de invocar swap

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```
void call_swap()
{
    int zip1 = 15213;
    int zip2 = 91125;
    (...)
    swap(&zip1, &zip2);
    (...)
}
```

call_swap



endereço crescente
↓

↑
stack

Evolução da stack, no IA-32 (2)



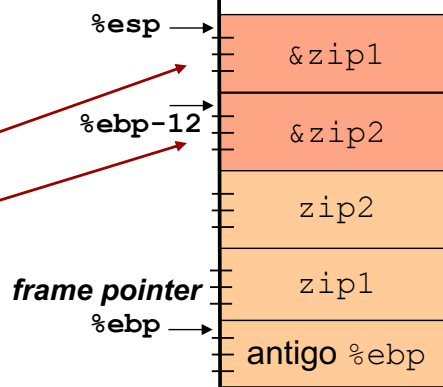
2. Preparação p/ invocar swap

- salvar registos?...**não...**
- passagem de parâmetros

```
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}
```

```
void call_swap()
{
    int zip1 = 15213;
    int zip2 = 91125;
    (...)
    swap(&zip1, &zip2);
    (...)
}
```

call_swap



endereço crescente
↓

↑
stack

Evolução da stack, no IA-32 (3)

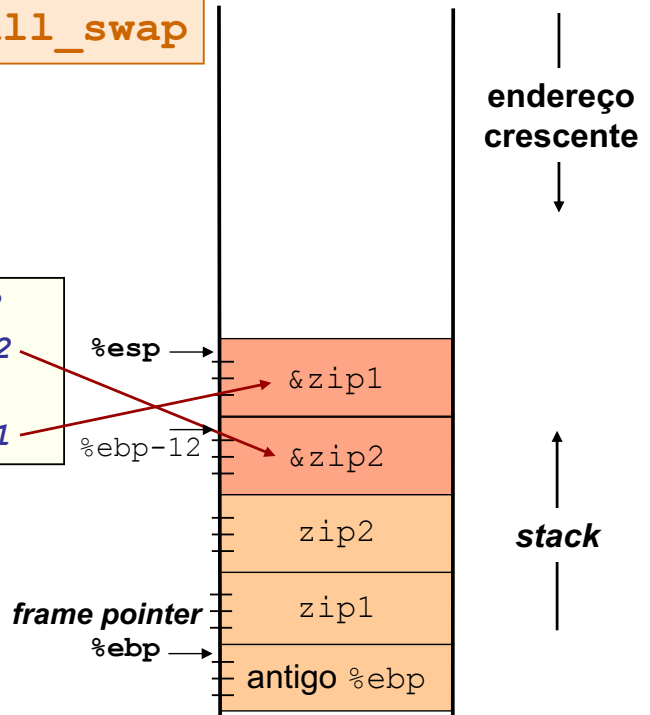


call_swap

2. Preparação p/ invocar swap

- salvar registros?...**não**...
- passagem de parâmetros

```
leal -8(%ebp), %eax  Calcula &zip2
pushl %eax           Empilha &zip2
leal -4(%ebp), %eax  Calcula &zip1
pushl %eax           Empilha &zip1
```



Evolução da stack, no IA-32 (4)



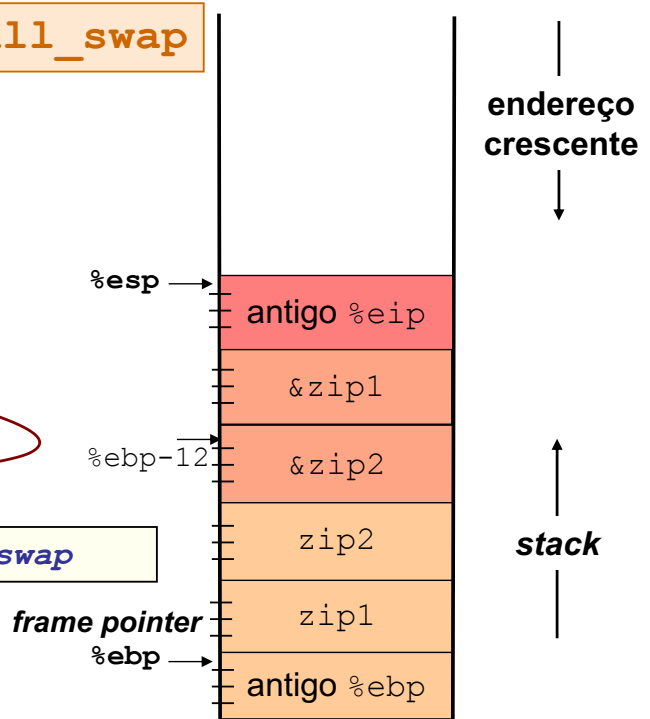
call_swap

3. Invocar swap

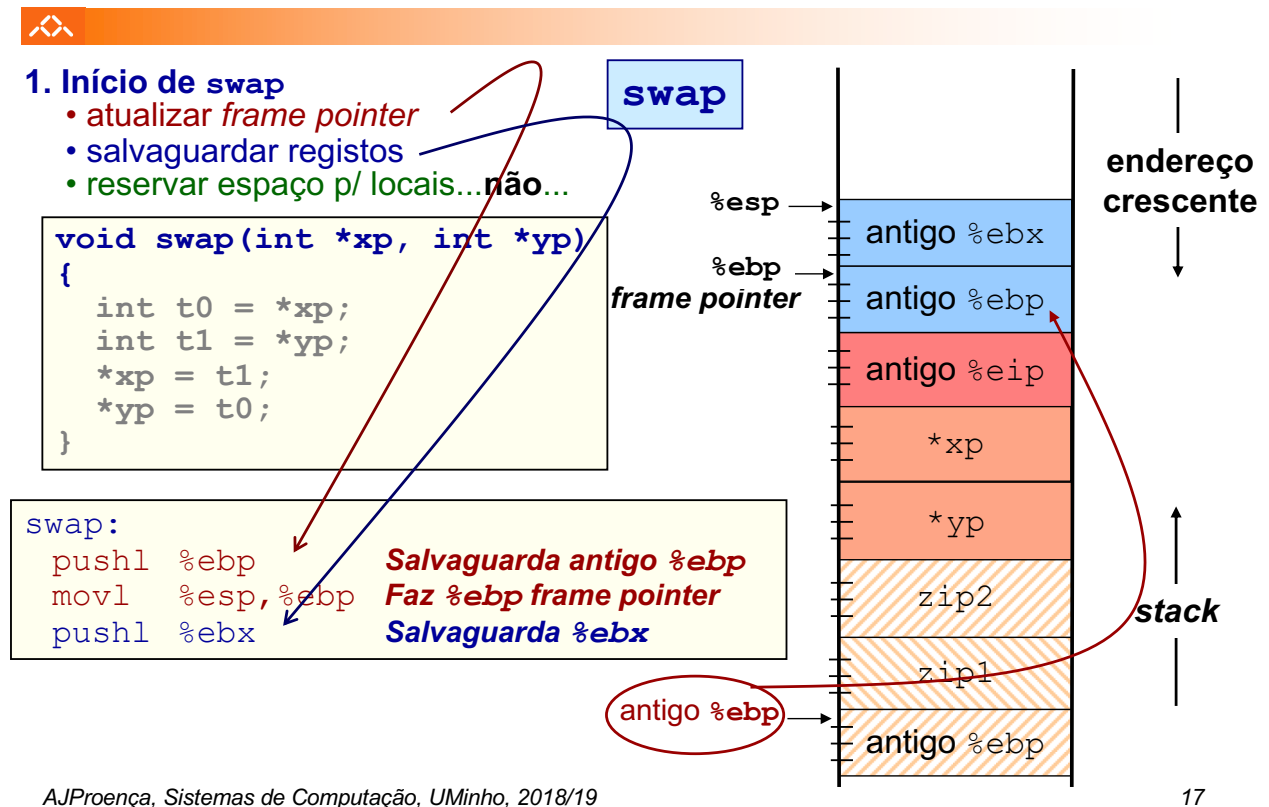
- e guardar endereço de regresso

```
void call_swap()
{
    int zip1 = 15213;
    int zip2 = 91125;
    (...)
    swap(&zip1, &zip2);
    (...)
}
```

call_swap Invoca função swap

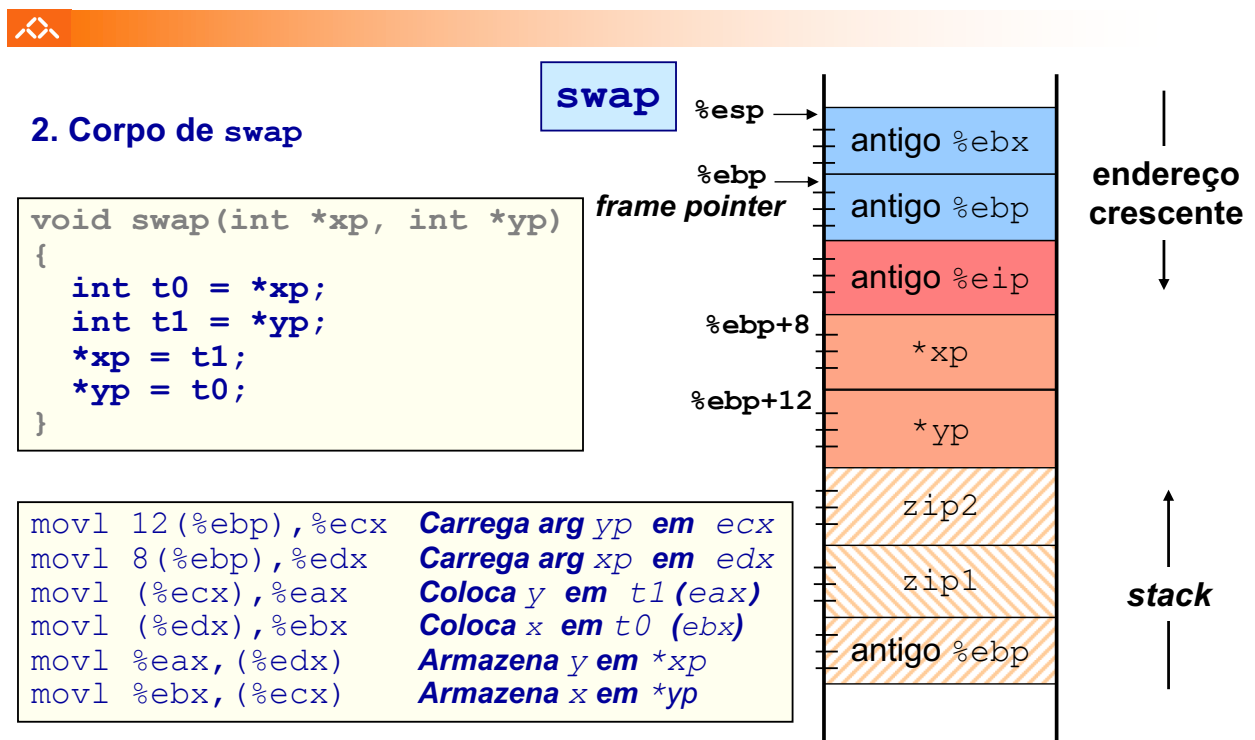


Evolução da stack, no IA-32 (5)



17

Evolução da stack, no IA-32 (6)



18

Evolução da stack, no IA-32 (7)



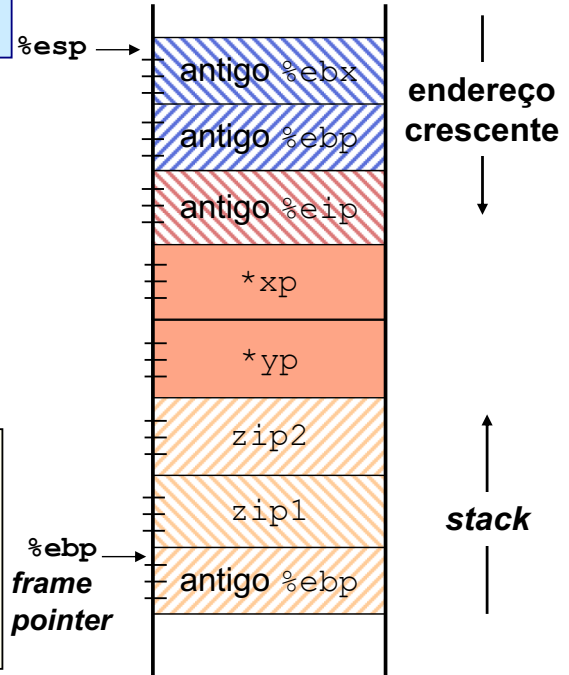
3. Término de swap ...

- libertar espaço de var locais...**não**...
- recuperar registos
- recuperar antigo frame pointer
- regressar a `call_swap`

```
void swap(int *xp, int *yp)
{
    (...)
}
```

```
popl %ebx      Recupera %ebx
movl %ebp, %esp Recupera %esp
popl %ebp      Recupera %ebp
                ou
leave         Recupera %esp, %ebp
ret           Regressa à f. chamadora
```

swap



Evolução da stack, no IA-32 (8)



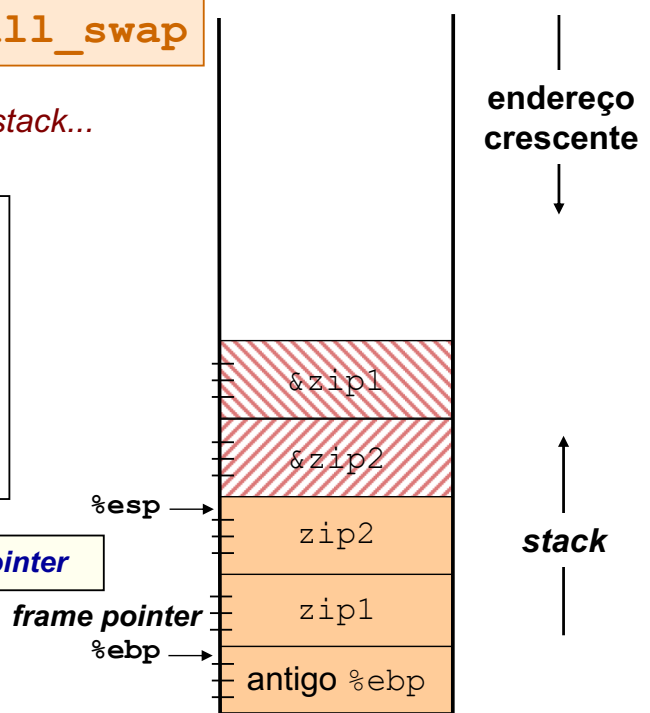
4. Terminar invocação de swap...

- libertar espaço de parâmetros na *stack*...
- recuperar registos?...**não**...

```
void call_swap()
{
    int zip1 = 15213;
    int zip2 = 91125;
    (...)
    swap(&zip1, &zip2);
    (...)
}
```

```
addl $8, %esp    Atualiza stack pointer
```

call_swap





Análise de exemplos

– revisão do exemplo swap

- análise das fases: inicialização, corpo, término
- análise dos contextos (IA-32)
- evolução dos contextos na *stack* (IA-32)

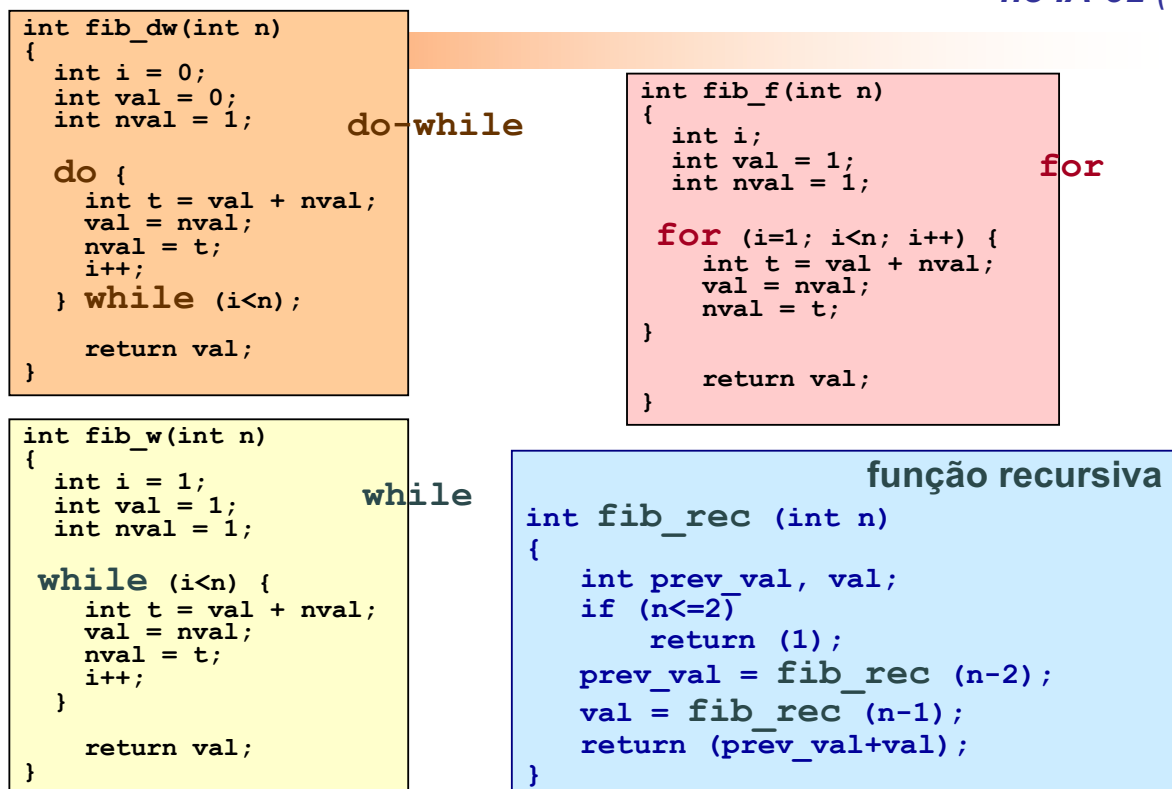
– evolução de um exemplo: Fibonacci

- análise de uma compilação do gcc

– aninhamento e recursividade

- evolução ...

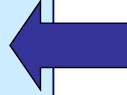
A série de Fibonacci no IA-32 (1)





função recursiva

```
int fib_rec (int n)
{
    int prev_val, val;
    if (n<=2)
        return (1);
    prev_val = fib_rec (n-2);
    val = fib_rec (n-1);
    return (prev_val+val);
}
```



_fib_rec:

```
pushl %ebp
movl  %esp, %ebp
subl  $12, %esp
movl  %ebx, -8(%ebp)
movl  %esi, -4(%ebp)
movl  8(%ebp), %esi
```

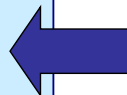
Atualiza frame pointer

*Reserva espaço na stack para 3 int's
Salvaguarda os 2 reg's que vão ser usados;
de notar a forma de usar a stack...*



função recursiva

```
int fib_rec (int n)
{
    int prev_val, val;
    if (n<=2)
        return (1);
    prev_val = fib_rec (n-2);
    val = fib_rec (n-1);
    return (prev_val+val);
}
```



```
...
movl  %esi, -4(%ebp)
movl  8(%ebp), %esi
movl  $1, %eax
cmpl  $2, %esi
jle   L1
leal  -2(%esi), %eax
...
L1:
movl  -8(%ebp), %ebx
```

*Coloca o argumento n em %esi
Coloca já o valor a devolver em %eax
Compara n:2
Se n<=2, salta para o fim
Se não, ...*



função recursiva

```
int fib_rec (int n)
{
    int prev_val, val;
    if (n<=2)
        return (1);
    prev_val = fib_rec (n-2);
    val = fib_rec (n-1);
    return (prev_val+val);
}
```



...		
jle	L1	Se $n \leq 2$, salta para o fim
leal	-2(%esi), %eax	Se não, ... calcula $n-2$, e...
movl	%eax, (%esp)	... coloca-o no topo da stack (argumento)
call	_fib_rec	Invoca a função <code>fib_rec</code> e ...
movl	%eax, %ebx	... guarda o valor de <code>prev_val</code> em <code>%ebx</code>
leal	-1(%esi), %eax	
...		



função recursiva

```
int fib_rec (int n)
{
    int prev_val, val;
    if (n<=2)
        return (1);
    prev_val = fib_rec (n-2);
    val = fib_rec (n-1);
    return (prev_val+val);
}
```

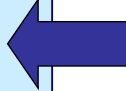


...		
movl	%eax, %ebx	
leal	-1(%esi), %eax	Calcula $n-1$, e...
movl	%eax, (%esp)	... coloca-o no topo da stack (argumento)
call	_fib_rec	Chama de novo a função <code>fib_rec</code>
leal	(%eax,%ebx), %eax	
...		



função recursiva

```
int fib_rec (int n)
{
    int prev_val, val;
    if (n<=2)
        return (1);
    prev_val = fib_rec (n-2);
    val = fib_rec (n-1);
    return (prev_val+val);
}
```



```
...
call    fib_rec
leal    (%eax,%ebx), %eax  Calcula e coloca em %eax o valor a devolver
L1:
movl    -8(%ebp), %ebx
movl    -4(%ebp), %esi     Recupera o valor dos 2 reg's usados
movl    %ebp, %esp        Atualiza o valor do stack pointer
popl    %ebp              Recupera o valor anterior do frame pointer
ret
```

Suporte a funções e procedimentos no IA-32 (4)



Análise de exemplos

– revisão do exemplo swap

- análise das fases: inicialização, corpo, término
- análise dos contextos (IA-32)
- evolução dos contextos na *stack* (IA-32)

– evolução de um exemplo: Fibonacci

- análise de uma compilação do gcc

– aninhamento e recursividade

- evolução dos contextos na *stack*

Exemplo de cadeia de invocações no IA-32 (1)



Estrutura do código

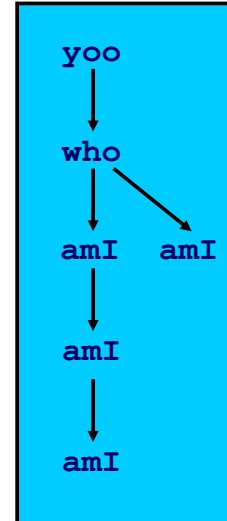
```
yoo (...)  
{  
  .  
  .  
  who () ;  
  .  
  .  
}
```

```
who (...)  
{  
  . . .  
  amI () ;  
  . . .  
  amI () ;  
  . . .  
}
```

```
amI (...)  
{  
  .  
  .  
  amI () ;  
  .  
  .  
}
```

Função **amI** é recursiva

Cadeia de *Call*

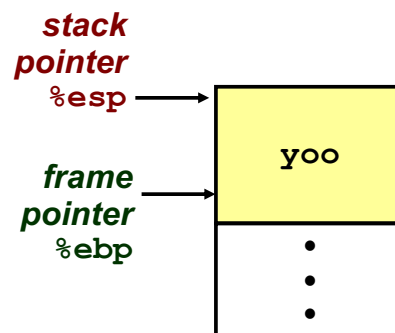
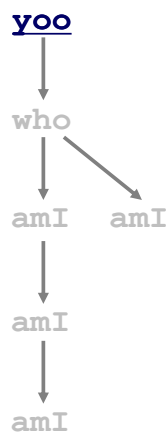


Exemplo de cadeia de invocações no IA-32 (2)

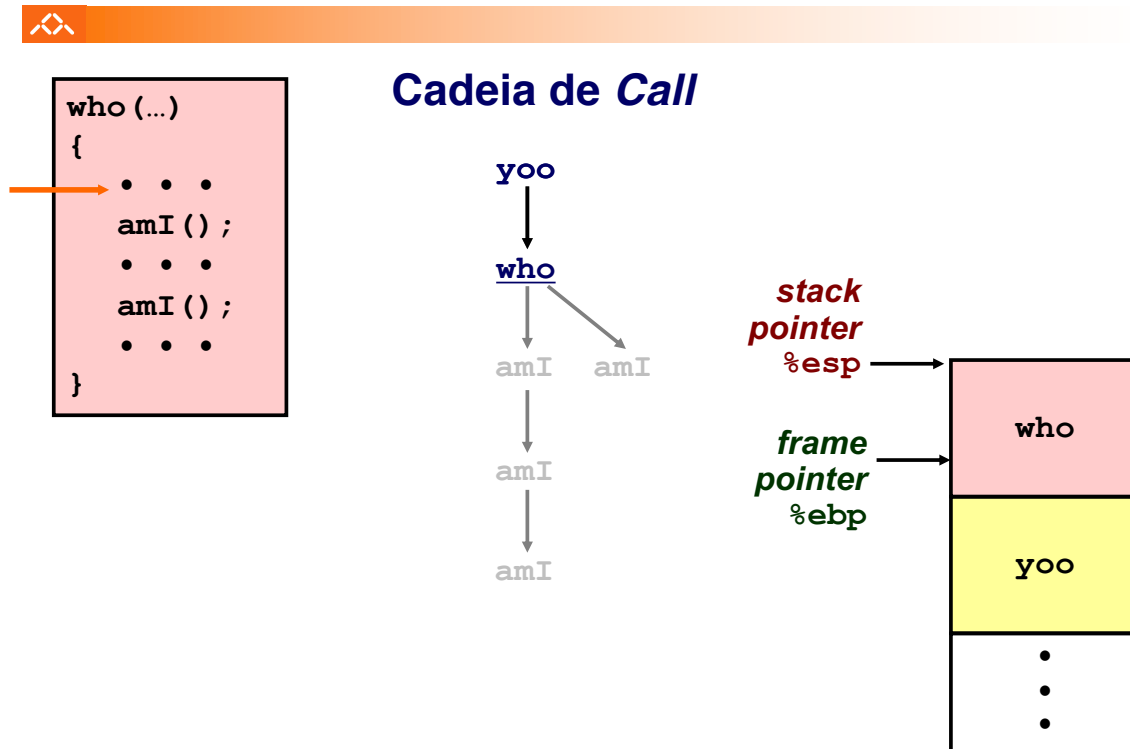


Cadeia de *Call*

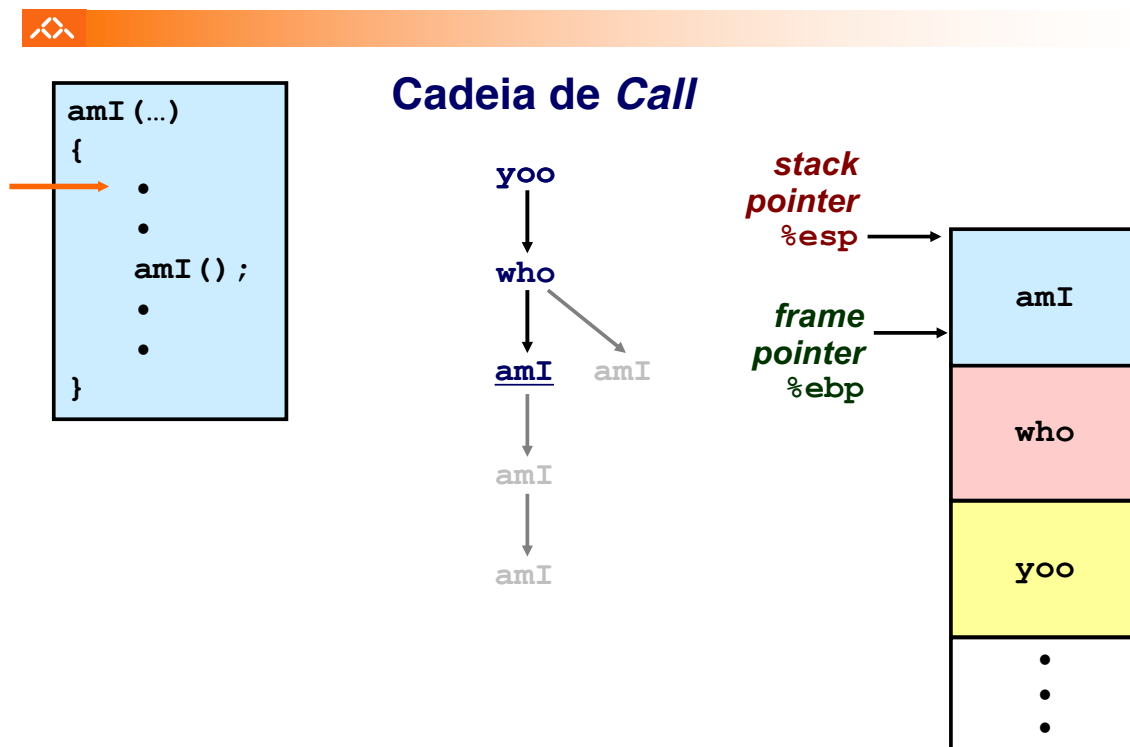
```
yoo (...)  
{  
  .  
  .  
  who () ;  
  .  
  .  
}
```



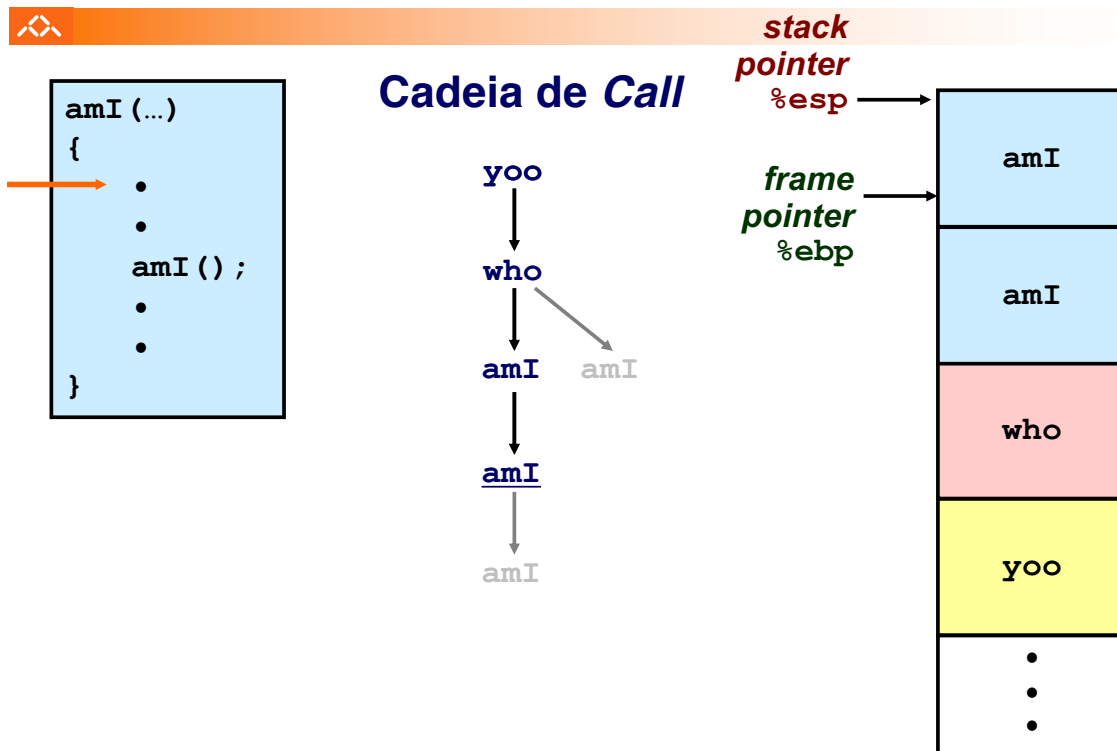
Exemplo de cadeia de invocações no IA-32 (3)



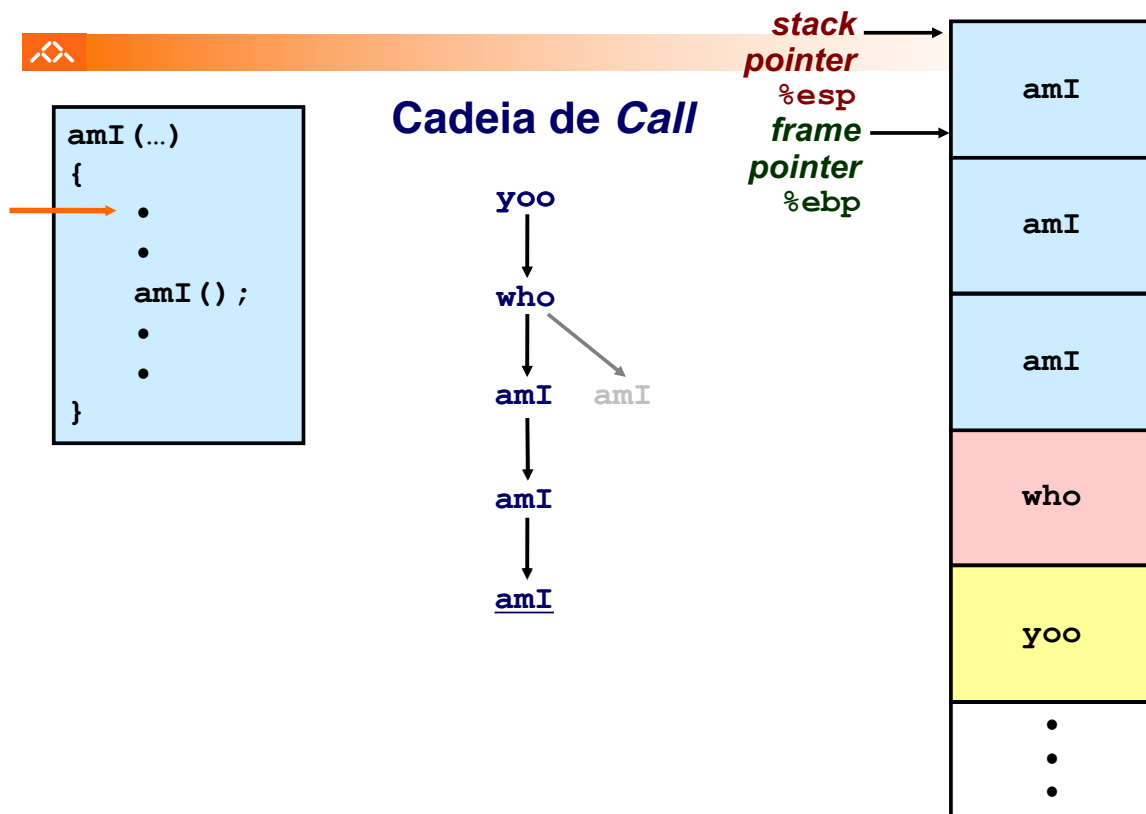
Exemplo de cadeia de invocações no IA-32 (4)



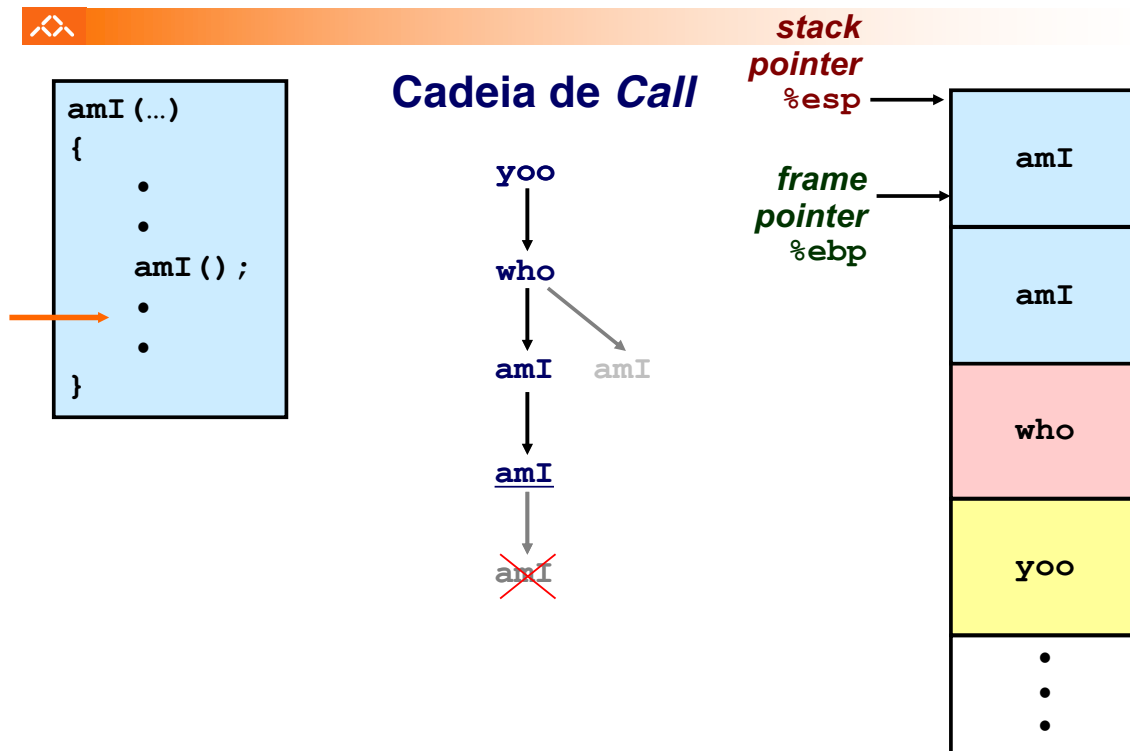
Exemplo de cadeia de invocações no IA-32 (5)



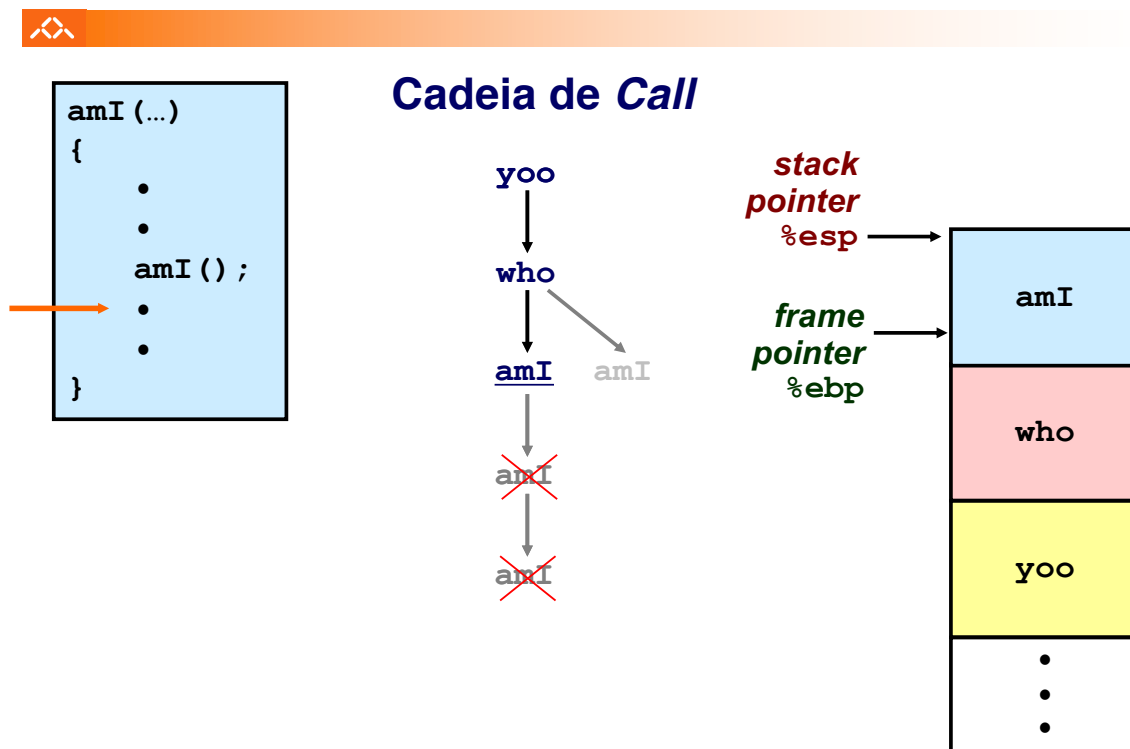
Exemplo de cadeia de invocações no IA-32 (6)



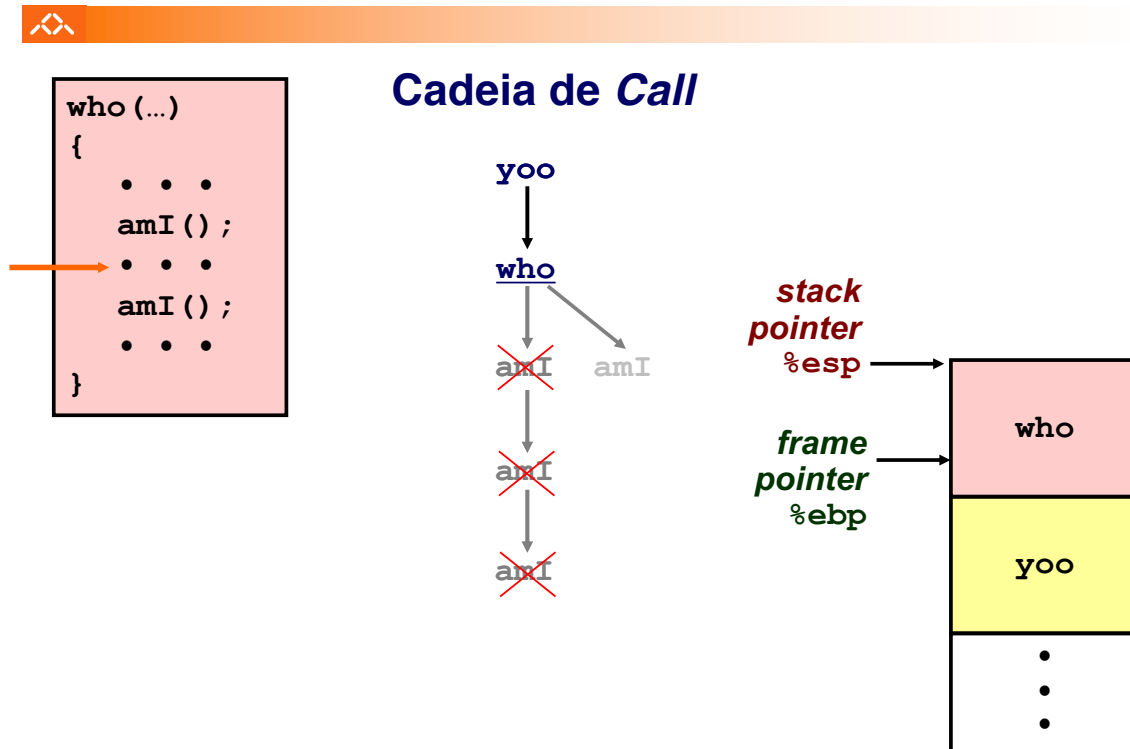
Exemplo de cadeia de invocações no IA-32 (7)



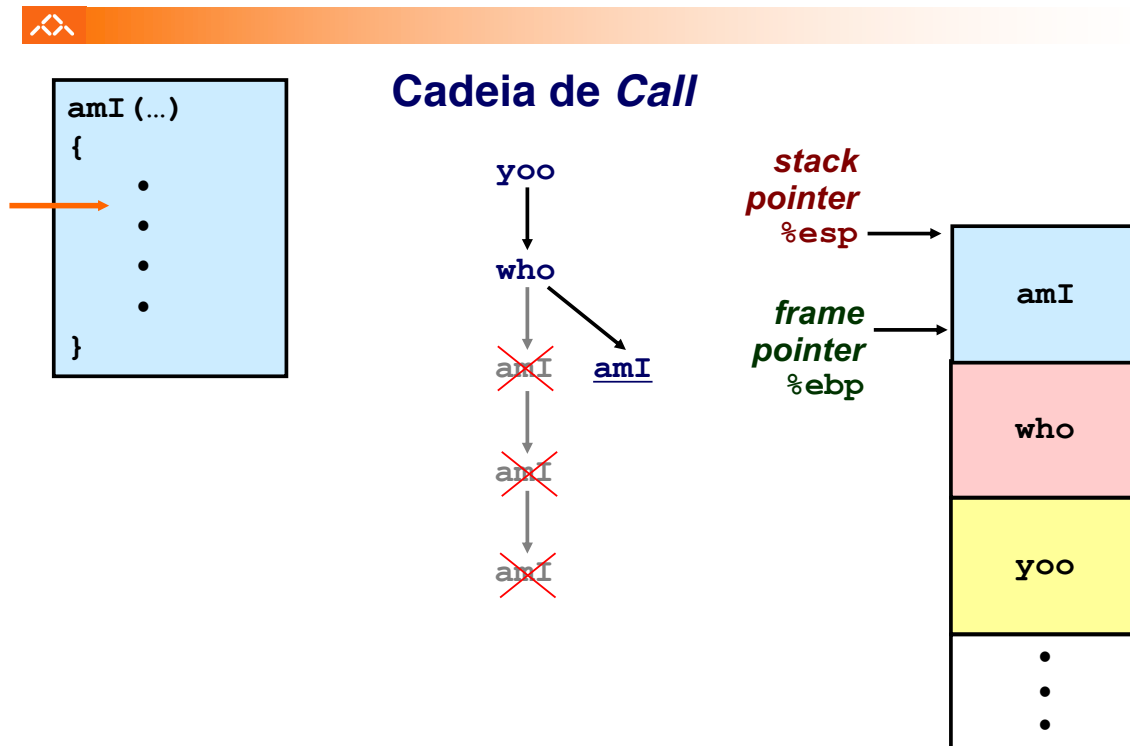
Exemplo de cadeia de invocações no IA-32 (8)



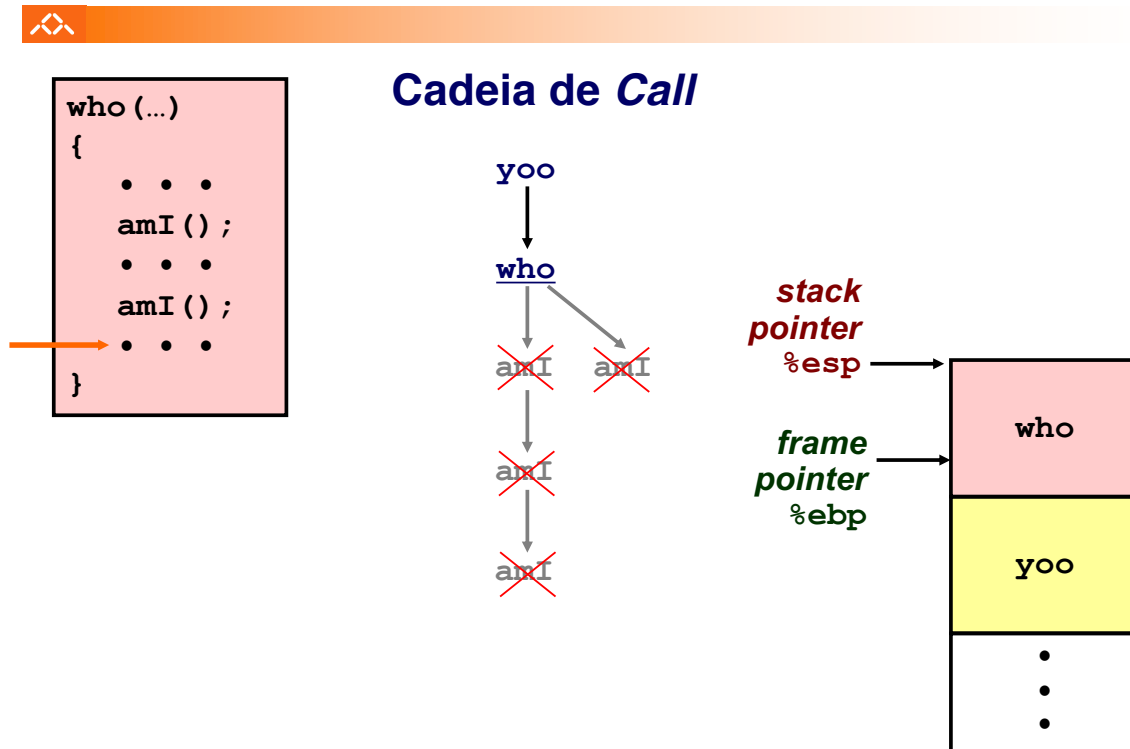
Exemplo de cadeia de invocações no IA-32 (9)



Exemplo de cadeia de invocações no IA-32 (10)



Exemplo de cadeia de invocações no IA-32 (11)



Exemplo de cadeia de invocações no IA-32 (12)

