



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 (CISC) e MIPS (RISC)
6. Acesso e manipulação de dados estruturados



Análise do contexto de uma função

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



Estrutura de uma função (/ procedimento)

- **parte visível ao programador em HLL**
 - código do corpo da função
 - passagem de parâmetros/argumentos para a função ...
... e valor devolvido pela função
 - alcance das variáveis: locais, externas ou globais
- **parte menos visível em HLL:
a gestão do contexto da função**
 - variáveis locais (propriedades)
 - variáveis externas e globais (localização e acesso)
 - parâmetros e valor a devolver pela função (propriedades)
 - 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/*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*)**
 - actualização/recuperação do *frame pointer* (IA-32...)
 - reserva/libertação de espaço para variáveis locais

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*

```
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);
    (...)
}
```

Utilização dos registos (de inteiros)

–Três do tipo *caller-save*

Caller-Save

`%eax, %edx, %ecx`

- *save/restore*: função chamadora

–Três do tipo *callee-save*

Callee-Save

`%ebx, %esi, %edi`

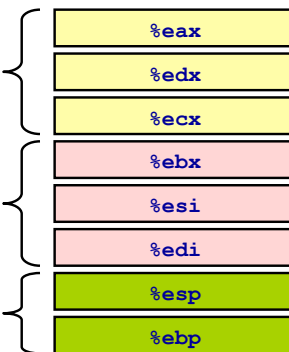
- *save/restore*: função chamada

–Dois apontadores (para a *stack*)

Pointers

`%esp, %ebp`

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



Nota: valor a devolver pela função em `%eax`

```
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 (%edx), %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

```

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);
    (...)
}
    
```

- em call_swap
- na invocação de swap
- na execução de swap
- de volta a call_swap

Que contextos (IA32)?

- passagem de parâmetros
 - via stack
- espaço para variáveis locais
 - na stack
- info de suporte à gestão (stack)
 - endereço de regresso
 - apontador para a stack frame
 - salvaguarda de registos

Construção do contexto na stack, no IA-32

```

call_swap  swap
    
```

- Antes de invocar swap**
- Preparação p/ invocar swap**
 - salvaguardar registos?
 - passagem de parâmetros
- Invocar swap**
 - e guardar endereço de regresso
 - Início de swap**
 - actualizar frame pointer
 - salvaguardar registos?
 - reservar espaço p/ locais
 - Corpo de swap**
 - Término de swap ...**
 - libertar espaço de var locais
 - recuperar registos?
 - recuperar antigo frame pointer
 - voltar a call_swap
- Terminar invocação de swap ...**
 - libertar espaço com parâmetros na stack
 - recuperar registos?

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);
    (...)
}
    
```

Evolução da stack, no IA-32 (2)

2. Preparação p/ invocar swap

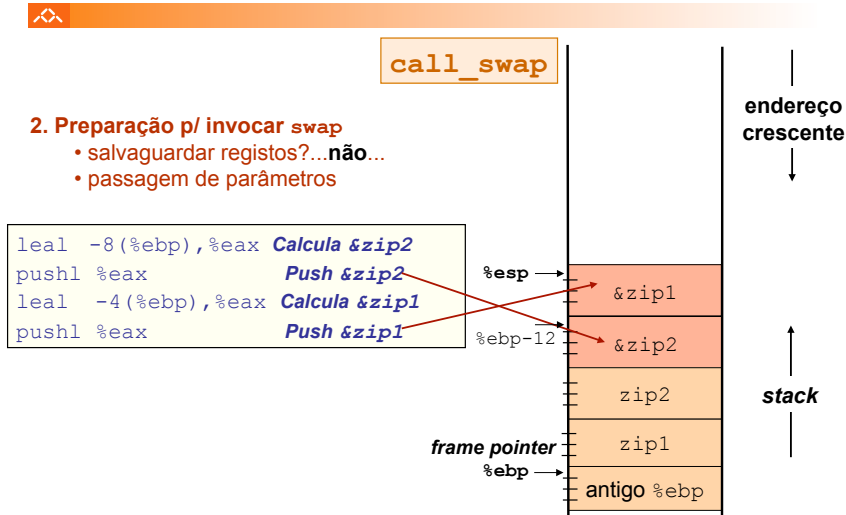
- salvaguardar registos?... não...
- passagem de parâmetros

```

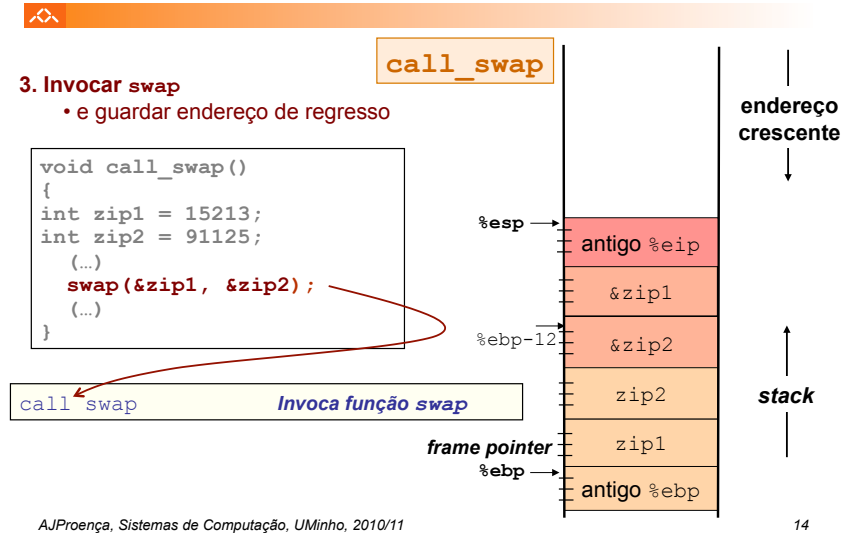
void swap(int *xp, int *yp)
{
    int t0 = *xp;
    int t1 = *yp;
    *xp = t1;
    *yp = t0;
}

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

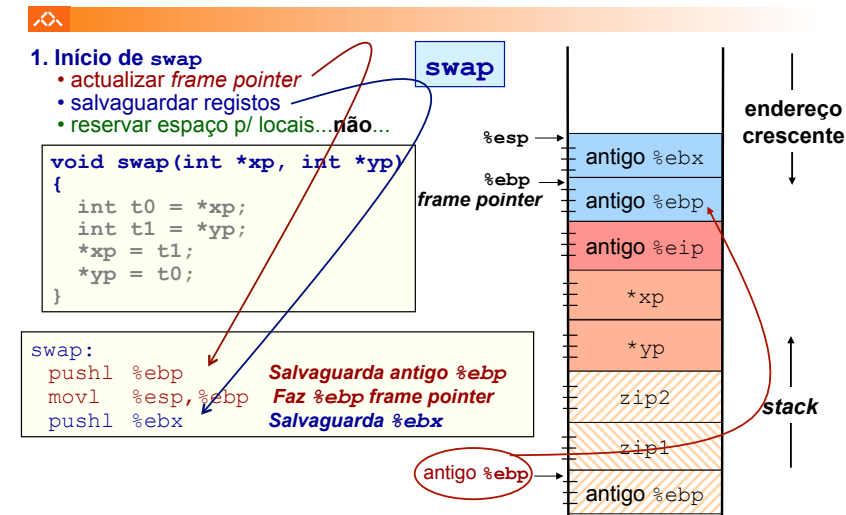
Evolução da stack, no IA-32 (3)



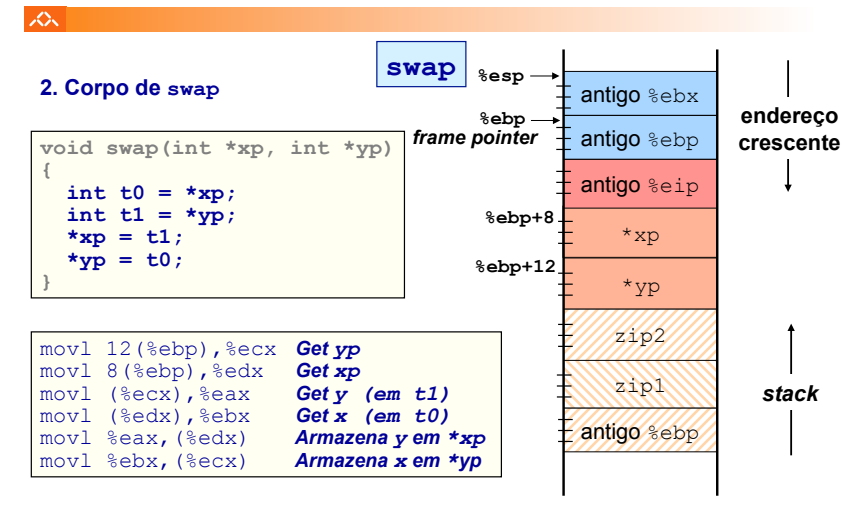
Evolução da stack, no IA-32 (4)



Evolução da stack, no IA-32 (5)



Evolução da stack, no IA-32 (6)



Evolução da stack, no IA-32 (7)

3. Término de swap ...

- libertar espaço de var locais...**não**...
- recuperar registos
- recuperar antigo *frame pointer*
- voltar 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           Volta à f. chamadora
```

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)  Actualiza stack pointer
```

A série de Fibonacci no IA-32 (1)

```
int fib_dw(int n)
{
    int i = 0;
    int val = 0;
    int nval = 1;

    do {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    } while (i < n);

    return val;
}
```

do-while

```
int fib_f(int n)
{
    int i;
    int val = 1;
    int nval = 1;

    for (i=1; i<n; i++) {
        int t = val + nval;
        val = nval;
        nval = t;
    }

    return val;
}
```

for

```
int fib_w(int n)
{
    int i = 1;
    int val = 1;
    int nval = 1;

    while (i < n) {
        int t = val + nval;
        val = nval;
        nval = t;
        i++;
    }

    return val;
}
```

while

```
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);
}
```

função recursiva

A série de Fibonacci no IA-32 (2)

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      Actualiza frame pointer
    subl $12, %esp      Reserva espaço na stack para 3 int's
    movl %ebx, -8(%ebp)  Salva os 2 reg's que vão ser usados;
    movl %esi, -4(%ebp)  de notar a forma de usar a stack...
    movl 8(%ebp), %esi
```

A série de Fibonacci no IA-32 (3)

A série de Fibonacci no IA-32 (4)

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);
}
```

Assembly:

```
...
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
n<=2?
Se sim, 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);
}
```

Assembly:

```
...
jle   L1
leal  -2(%esi), %eax
movl  %eax, (%esp)
call  _fib_rec
movl  %eax, %ebx
leal  -1(%esi), %eax
...
```

*Se sim, salta para o fim
Se não, ... calcula n-2, e...
... coloca-o no topo da stack (argumento)
Invoca a função fib_rec e ...
... guarda o valor de prev_val em %ebx*

A série de Fibonacci no IA-32 (5)

A série de Fibonacci no IA-32 (6)

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);
}
```

Assembly:

```
...
movl  %eax, %ebx
leal  -1(%esi), %eax
movl  %eax, (%esp)
call  _fib_rec
leal  (%eax,%ebx), %eax
...
```

*Calcula n-1, e...
... coloca-o no topo da stack (argumento)
Chama de novo a função fib_rec*

função recursiva

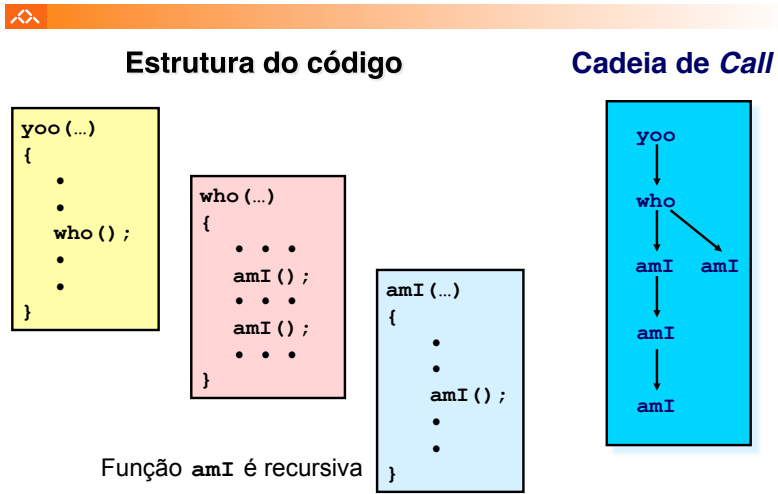
```
int fib_rec (int n)
{
  int prev_val, val;
  if (n<=2)
    return (1);
  prev_val = fib_rec (n-2);
  val = fib_rec (n-1);
  return (prev_val+val);
}
```

Assembly:

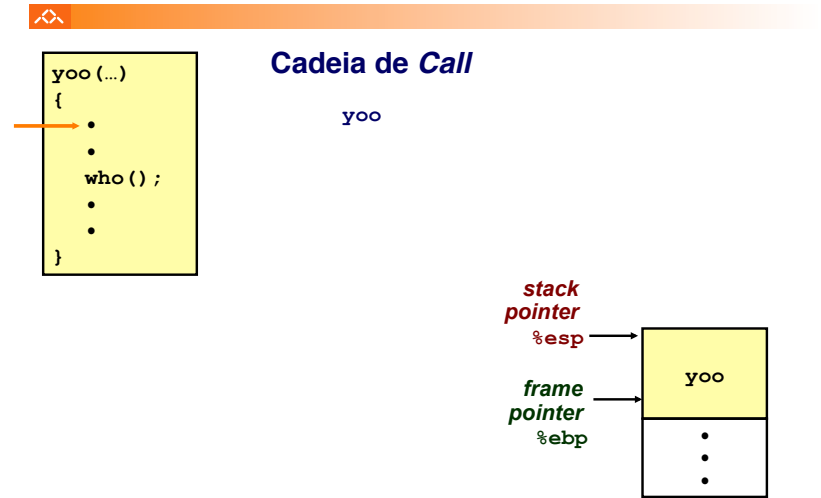
```
...
call  _fib_rec
leal  (%eax,%ebx), %eax
L1:
movl  -8(%ebp), %ebx
movl  -4(%ebp), %esi
movl  %ebp, %esp
popl  %ebp
ret
```

*Calcula e coloca em %eax o valor a devolver
Recupera o valor dos 2 reg's usados
Atualiza o valor do stack pointer
Recupera o anterior valor do frame pointer*

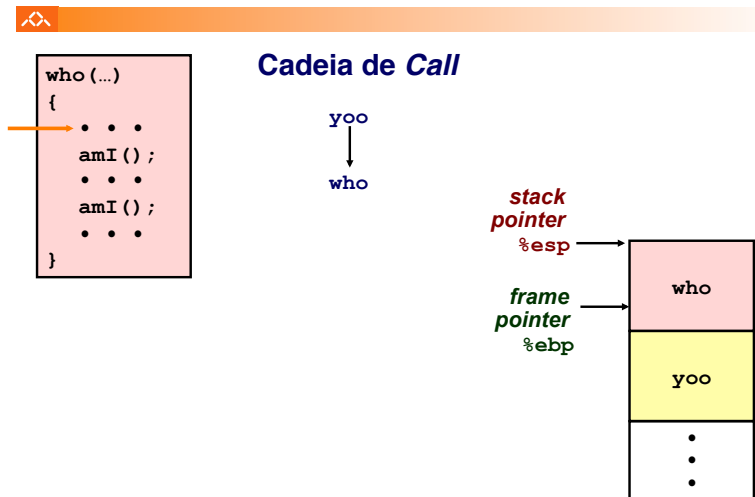
Exemplo de cadeia de invocações
no IA-32 (1)



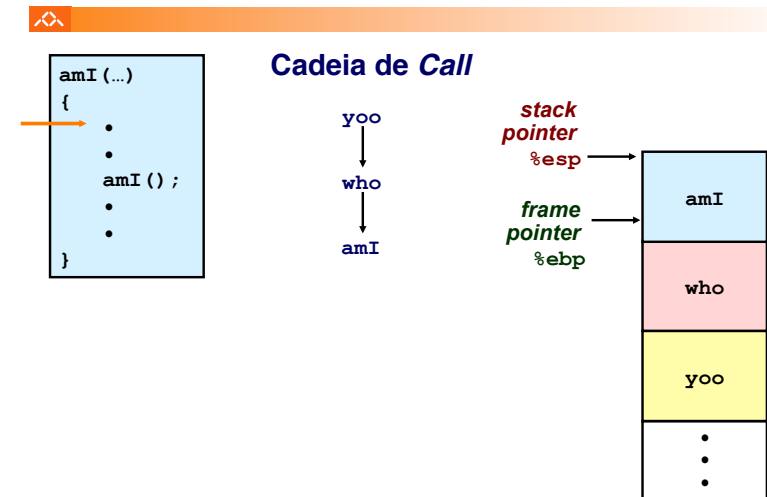
Exemplo de cadeia de invocações
no IA-32 (2)



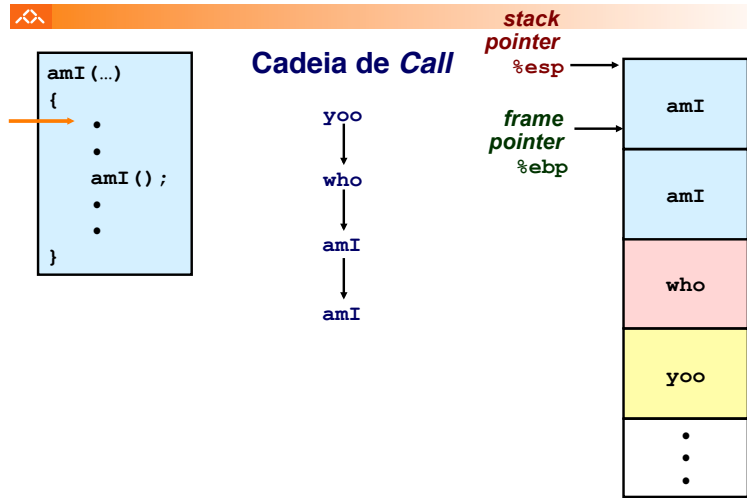
Exemplo de cadeia de invocações
no IA-32 (3)



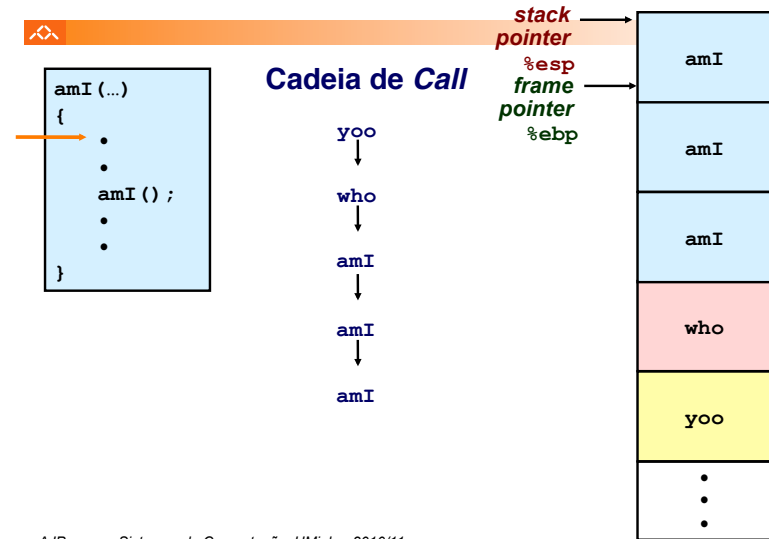
Exemplo de cadeia de invocações
no IA-32 (4)



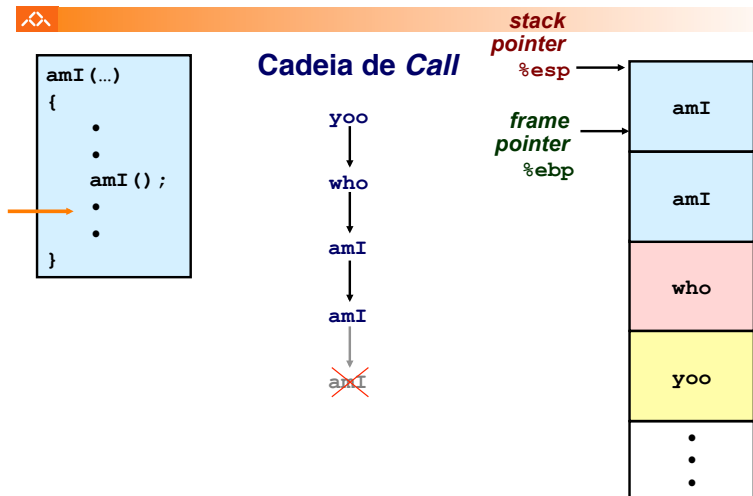
Exemplo de cadeia de invocações no IA-32 (5)



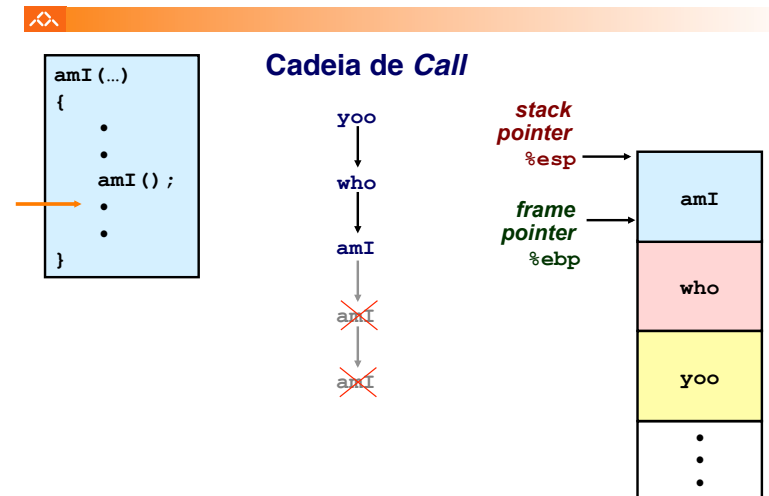
Exemplo de cadeia de invocações no IA-32 (6)



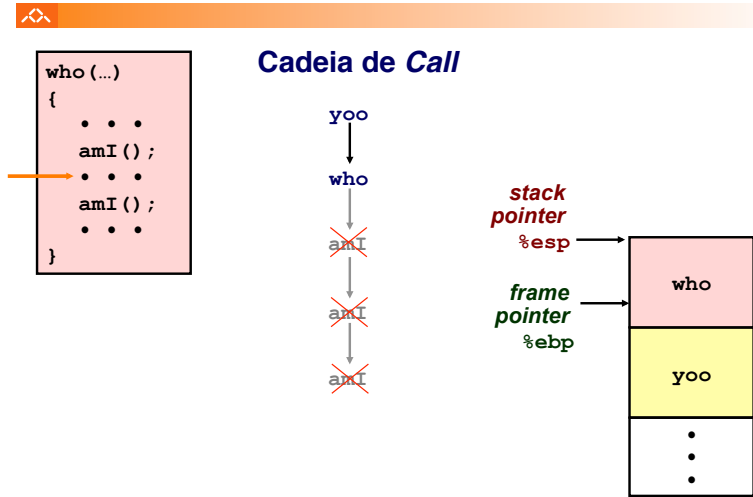
Exemplo de cadeia de invocações no IA-32 (7)



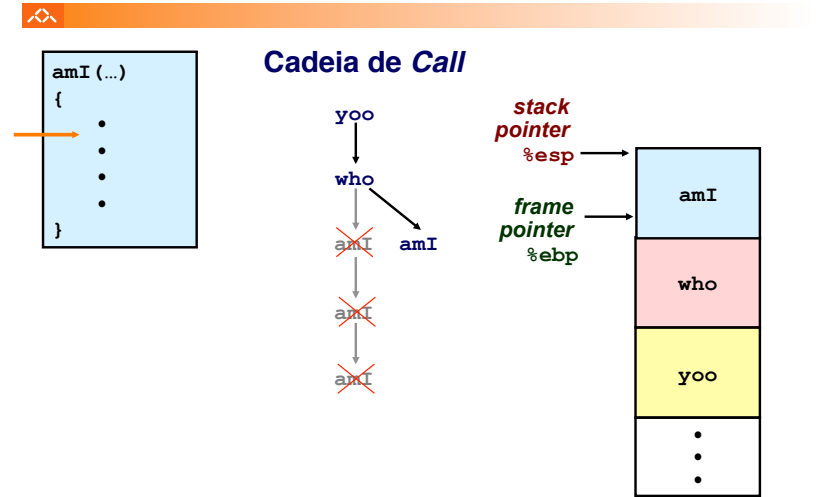
Exemplo de cadeia de invocações no IA-32 (8)



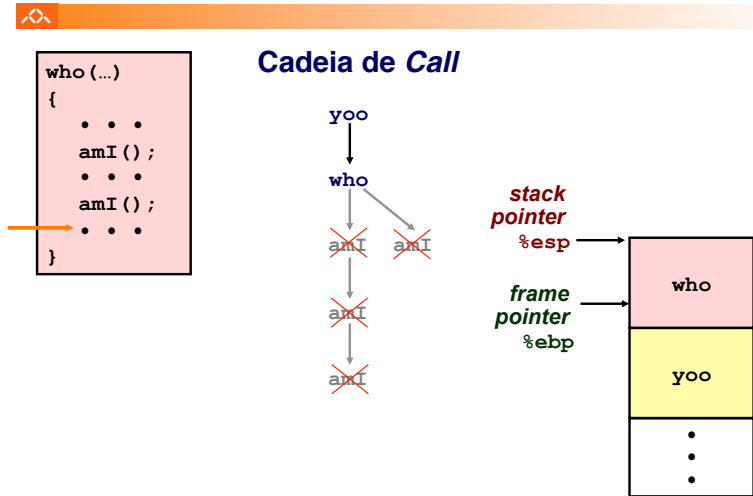
Exemplo de cadeia de invocações no IA-32 (9)



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)

