

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/retorno de funções
5. Análise comparativa: IA-32 (CISC) e MIPS (RISC)
6. Acesso e manipulação de dados estruturados

Estrutura de uma função (/ procedimento)

- parte visível ao programador em HLL
 - código do corpo da função
 - passagem de parâmetros 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 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...
 - localiz. no cód. p/ 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 p/ IA-32: na memória (*stack*)
- valor a devolver pela função:
 - é uma quantidade escalar, do tipo inteiro ou real
 - localização: em registo (IA-32: 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 com salvaguarda do end. 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*

`%eax, %edx, %ecx`

- *save/restore*: função chamadora

– Três do tipo *callee-save*

`%ebx, %esi, %edi`

- *save/restore*: função chamada

– Dois apontadores (para a *stack*)

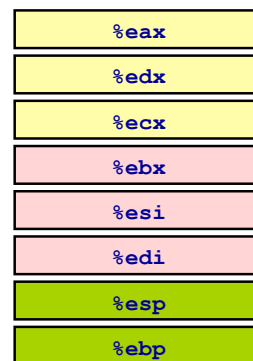
`%esp, %ebp`

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

Caller-Save

Callee-Save

Pointers



Nota: valor a devolver pela função 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 (%ecx),%eax
    movl (%edx),%ebx
    movl %eax,(%edx)
    movl %ebx,(%ecx)
}
    movl -4(%ebp),%ebx
    movl %ebp,%esp
    popl %ebp
    ret
```

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

1. Início de swap

- atualizar frame pointer
- salvaguardar registos?
- reservar espaço p/ locais

2. Corpo de swap

3. Término de swap ...

- libertar espaço de var locais
- recuperar registos?
- recuperar antigo frame pointer
- voltar a call_swap

4. 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

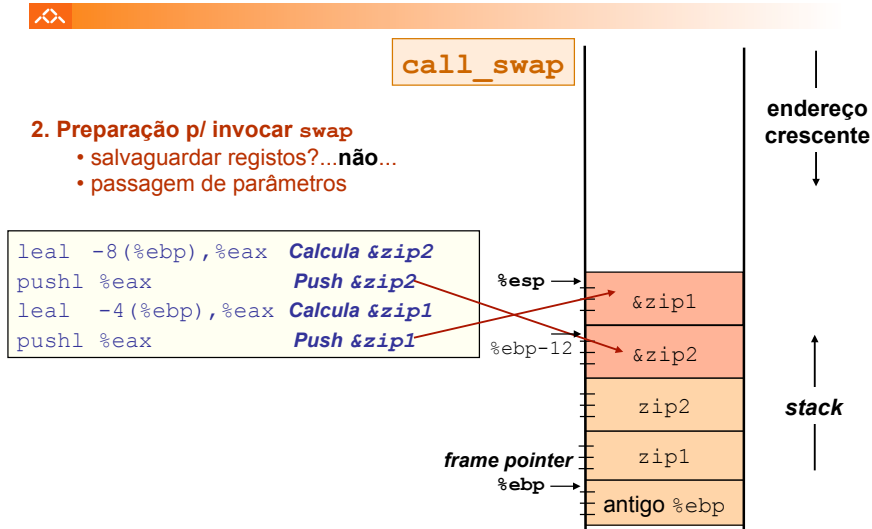
```

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

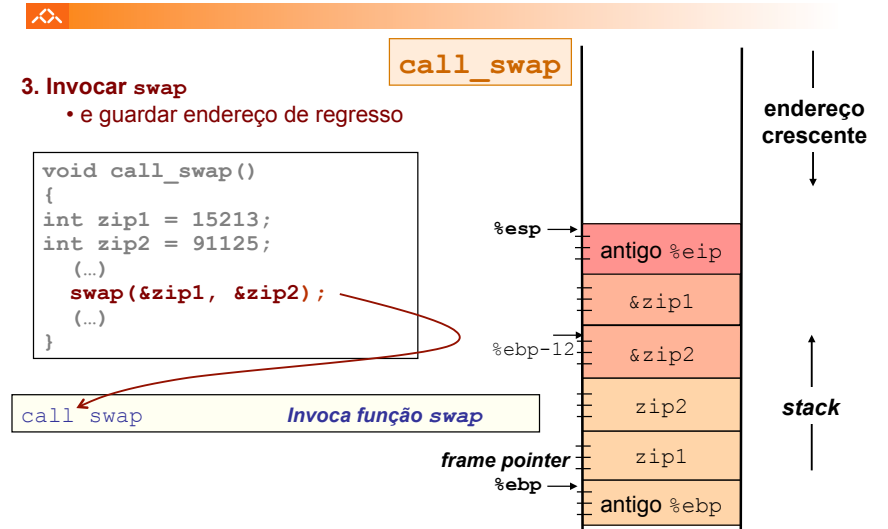
void call_swap()
{
  int zip1 = 15213;
  int zip2 = 91125;
  (...)
  swap(&zip1, &zip2);
  (...)
}

```

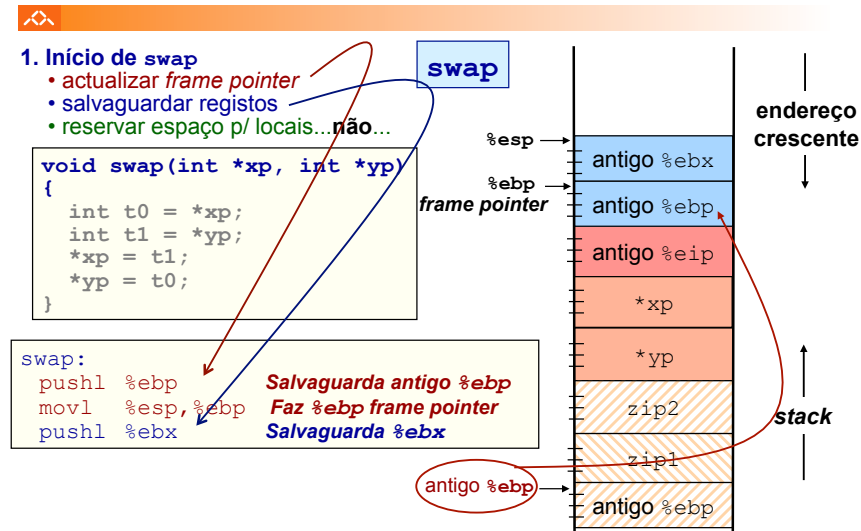
Evolução da stack, no IA-32 (3)



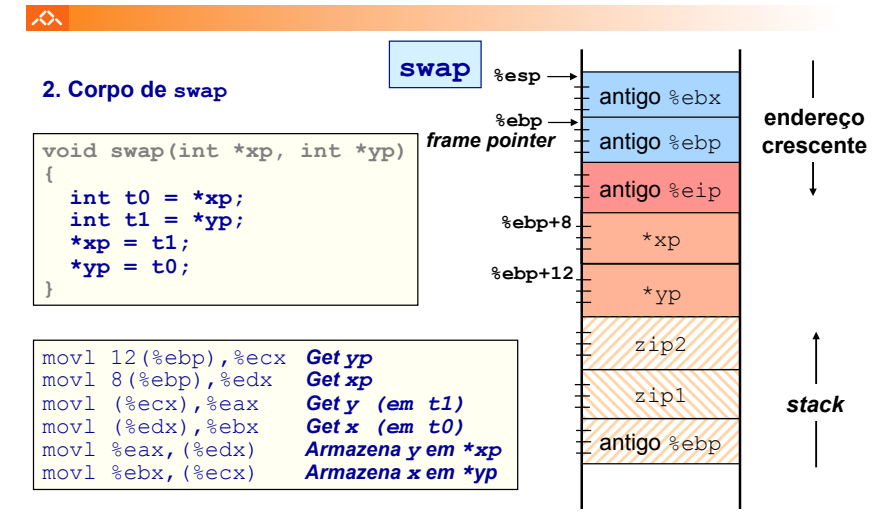
Evolução da stack, no IA-32 (4)



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
leave         Recupera %esp, %ebp
ret          Volta à f. chamadora
```

swap

endereço crescente

↑ stack

%ebp frame pointer

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
```

call_swap

endereço crescente

↑ stack

%esp

frame pointer %ebp

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

```
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);
}
```

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

Actualiza frame pointer

Reserva espaço na stack para 3 int's

Salvuarda os 2 reg's que vão ser usados; de notar a forma de usar a stack...

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);
}
```

```
...
movl   %esi, -4(%ebp)
movl   8(%ebp), %esi   Coloca o argumento n em %esi
movl   $1, %eax         Coloca já o valor a devolver em %eax
cmpl   $2, %esi         n<=2?
jle    L1              Se sim, salta para o fim
leal   -2(%esi), %eax   Se não, ...
...
L1:
movl   -8(%ebp), %ebx
```

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 sim, 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 fib_rec e ...
movl   %eax, %ebx   ... guarda o valor de prev_val em %ebx
leal   -1(%esi), %eax
...
```

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);
}
```

```
...
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 fib_rec
leal   (%eax,%ebx), %eax
...
```

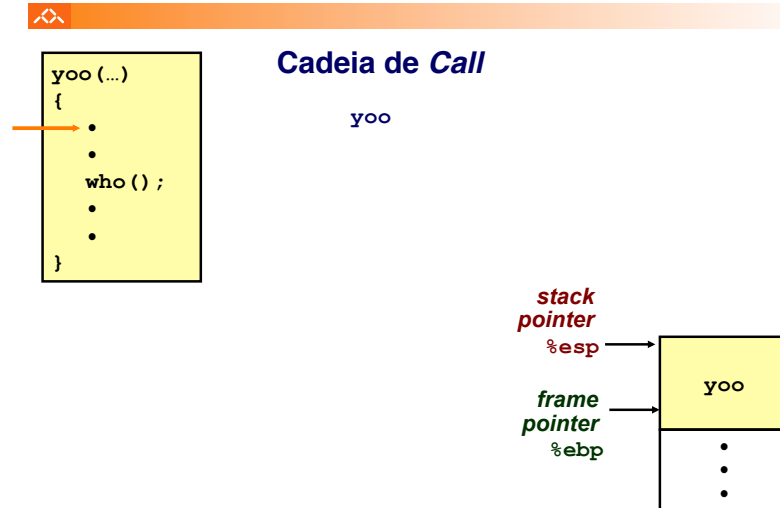
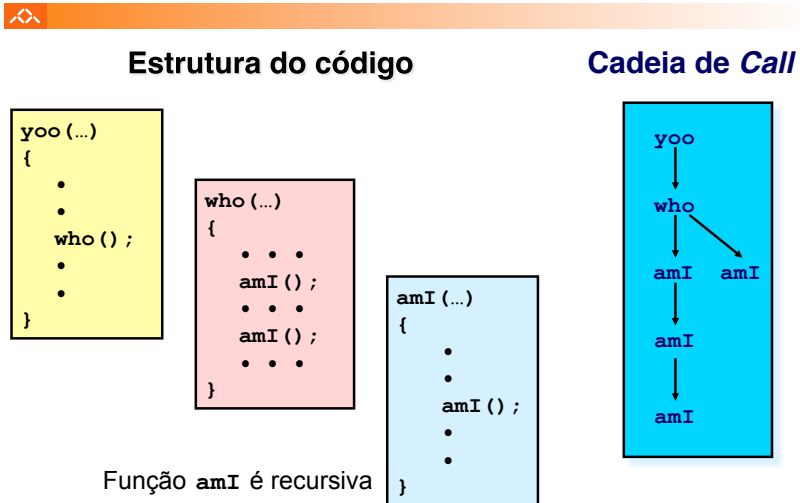
função recursiva

```
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
movl   %ebp, %esp    Recupera o valor dos 2 reg's usados
popl   %ebp         Actualiza o valor do stack pointer
ret                Recupera o anterior valor do frame pointer
```

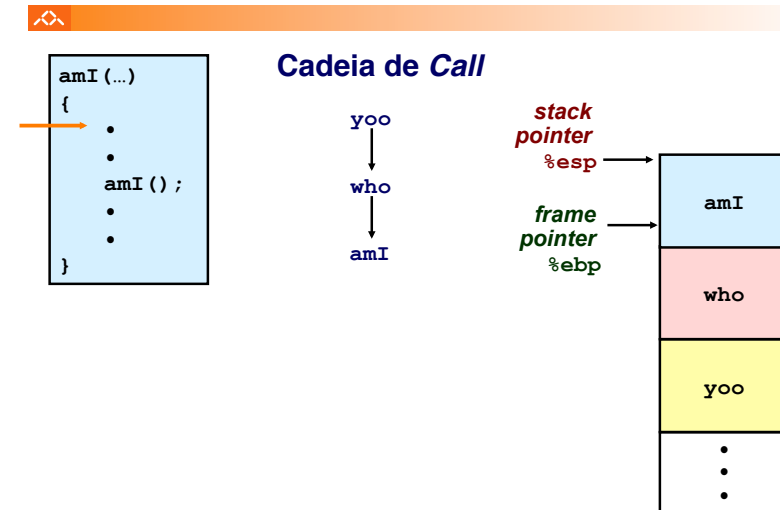
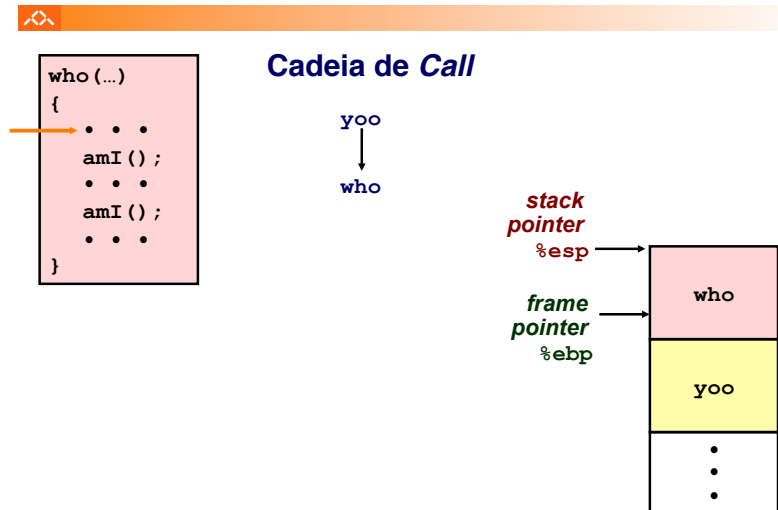
Exemplo de cadeia de invocações
no IA-32 (1)

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

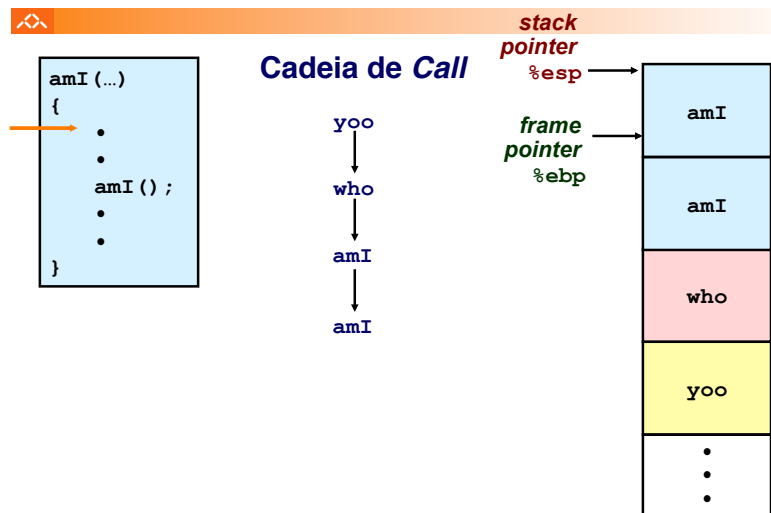


Exemplo de cadeia de invocações
no IA-32 (3)

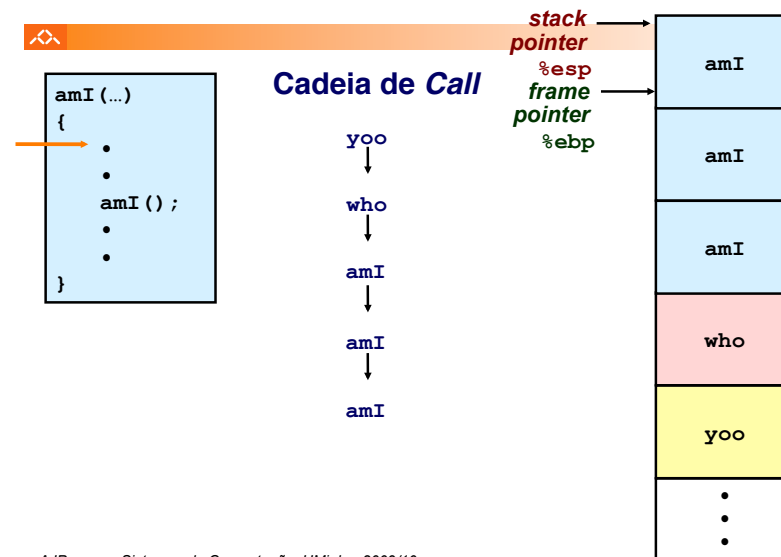
Exemplo de cadeia de invocações
no IA-32 (4)



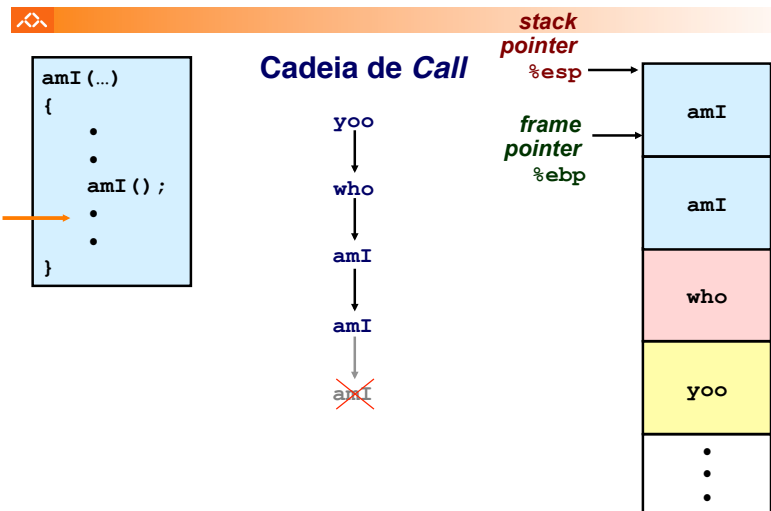
Exemplo de cadeia de invocações no IA-32 (5)



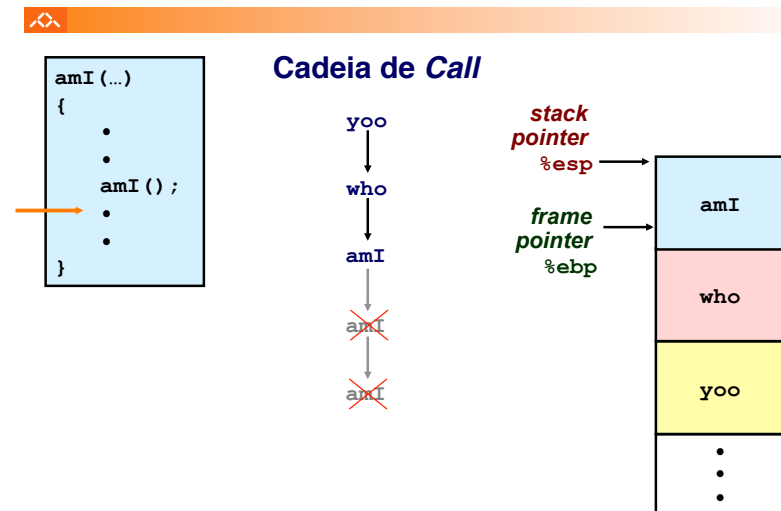
Exemplo de cadeia de invocações no IA-32 (6)



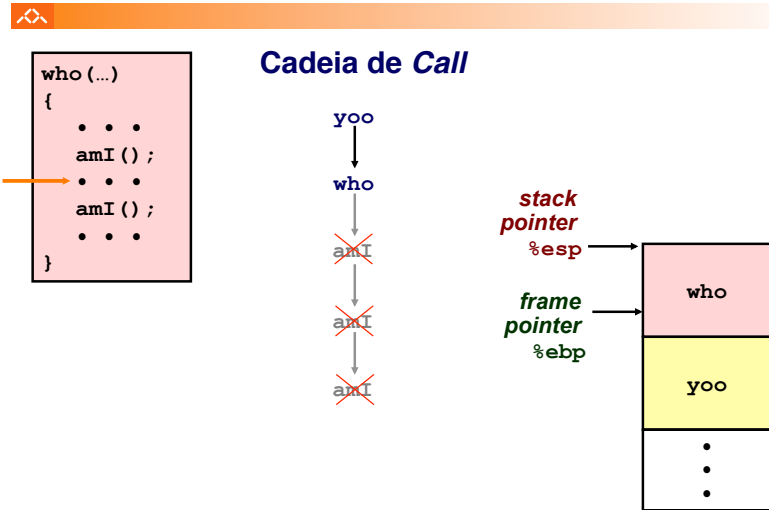
Exemplo de cadeia de invocações no IA-32 (7)



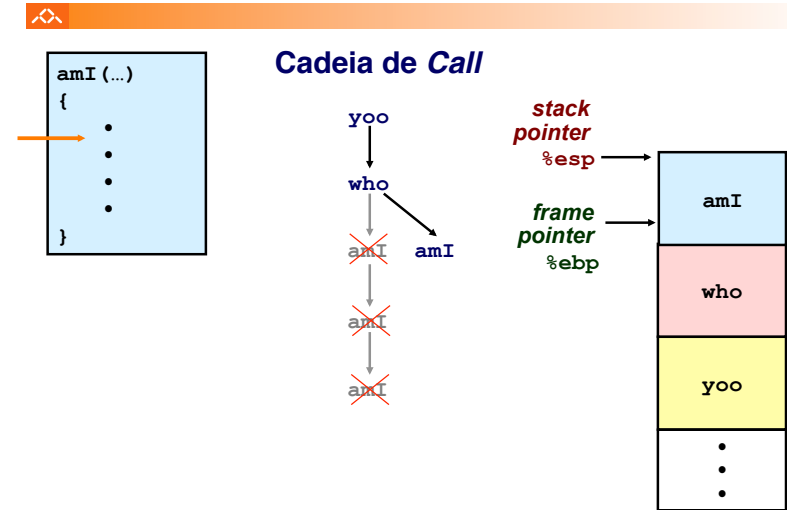
Exemplo de cadeia de invocações no IA-32 (8)



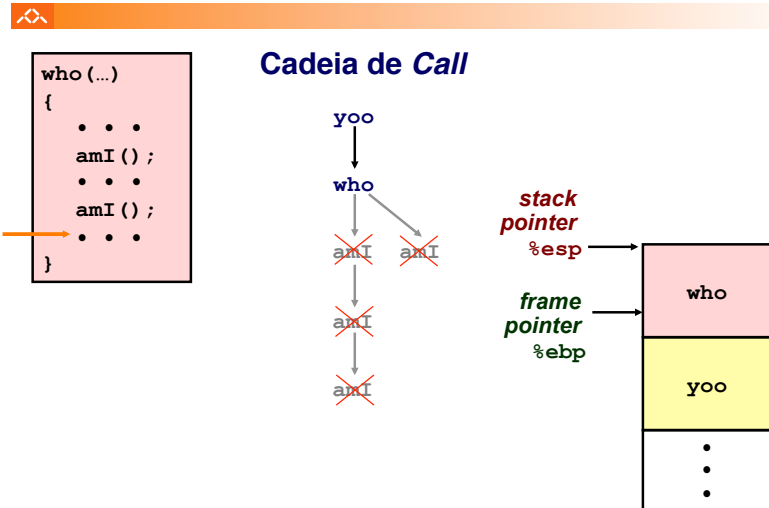
Exemplo de cadeia de invocações no IA-32 (9)



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)

