

# LẬP TRÌNH HỆ THỐNG

---

ThS. Đỗ Thị Thu Hiền  
(hiendtt@uit.edu.vn)



TRƯỜNG ĐH CÔNG NGHỆ THÔNG TIN - ĐHQG-HCM  
**KHOA MẠNG MÁY TÍNH & TRUYỀN THÔNG**  
FACULTY OF COMPUTER NETWORK AND COMMUNICATIONS

Tầng 8 - Tòa nhà E, trường ĐH Công nghệ Thông tin, ĐHQG-HCM  
Điện thoại: (08)3 725 1993 (122)

# Machine-level programming: Procedure (Hàm/Thủ tục)



# Cơ chế gọi hàm/thủ tục (procedure)

## ■ 1. Chuyển luồng

- Bắt đầu thực thi hàm được gọi
- Trở về vị trí đã gọi hàm

## ■ 2. Truyền dữ liệu

- Truyền tham số (arguments) cho hàm
- Nhận giá trị trả về của hàm

## ■ 3. Quản lý bộ nhớ

- Cấp phát bộ nhớ khi thực thi hàm

- Thu hồi bộ nhớ khi thực thi xong

## ■ Tất cả đều thực hiện được ở mức máy tính!

## ■ Hàm ở IA32 và x86-64 sẽ có một số khác biệt.

```
P (...) {  
    :  
    :  
    y = Q(x);  
    printf("%d", y);  
}
```

```
int Q(int i)  
{  
    int t = 3*i;  
    int v;  
    :  
    :  
    return v;  
}
```

# Cơ chế gọi hàm/thủ tục (procedure)

```
int main()
{
    int result = func(5,6);
    return result;
}
```

```
int func(int x, int y)
{
    int sum = 0;
    sum = x + y;
    return sum;
}
```

main:

```
pushl %ebp
movl %esp, %ebp
subl $16, %esp
pushl $6 } đưa tham số
pushl $5 } để func lấy
call func
movl %eax, -4(%ebp)
movl -4(%ebp), %eax
                giá trị trả về lưu vào eax
leave
ret
```

func:

```
pushl %ebp
movl %esp, %ebp
subl $16, %esp
movl $0, -4(%ebp)
movl 8(%ebp), %edx
movl 12(%ebp), %eax
addl %edx, %eax
movl %eax, -4(%ebp)
movl -4(%ebp), %eax
leave
ret
```

# Nội dung

---

- Thủ tục (Procedures)
  - Cấu trúc stack
  - Gọi hàm trong IA32
    - Chuyển luồng
    - Truyền dữ liệu
    - Quản lý dữ liệu cục bộ
  - Gọi hàm trong x86-64
  - Minh họa hàm đệ quy
- Bài tập về hàm
- Dịch ngược – Reverse engineering

# IA32 Stack

- Vùng nhớ được quản lý theo quy tắc ngăn xếp

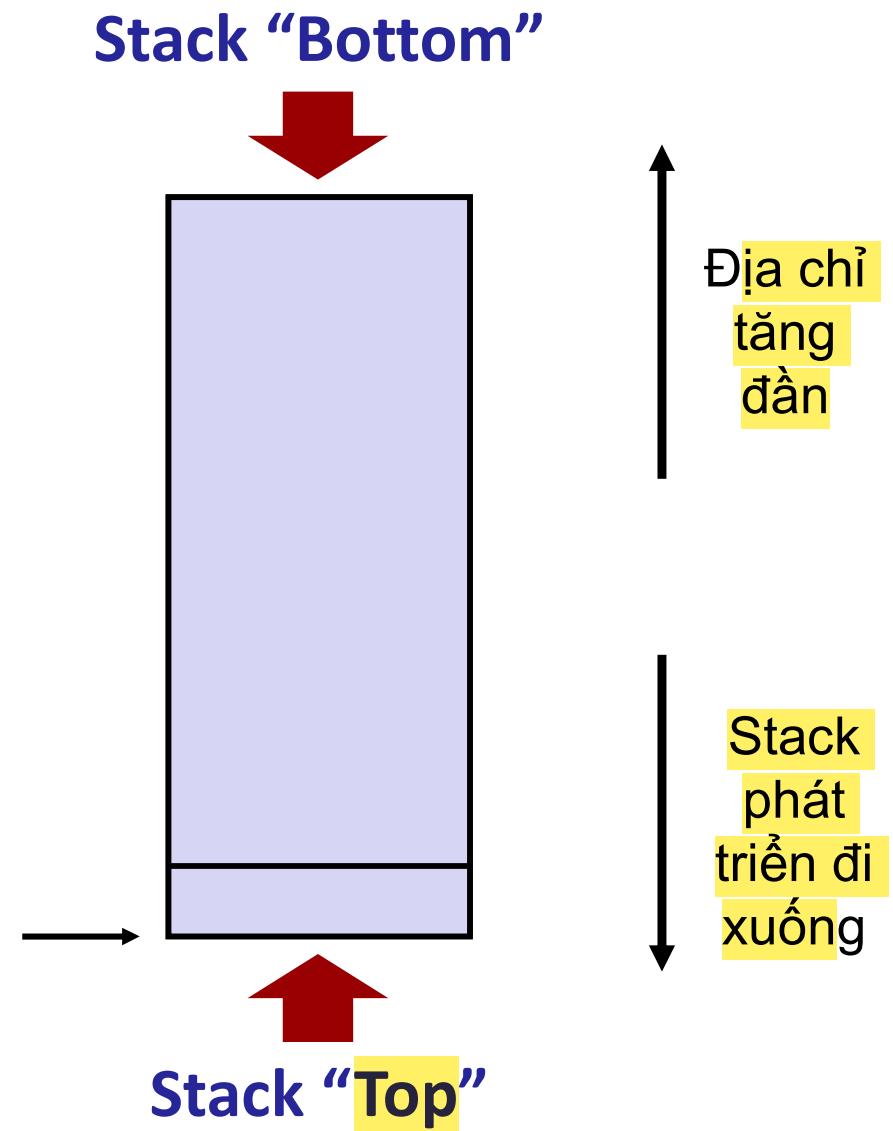
- First In Last Out

- Phát triển dần về phía địa chỉ thấp hơn

- Thanh ghi `%esp` chứa địa chỉ thấp nhất của stack

- địa chỉ của “đỉnh” stack

Con trỏ stack  
(Stack Pointer):  
`%esp`



# IA32 Stack: Push

## ■ Đẩy dữ liệu vào stack

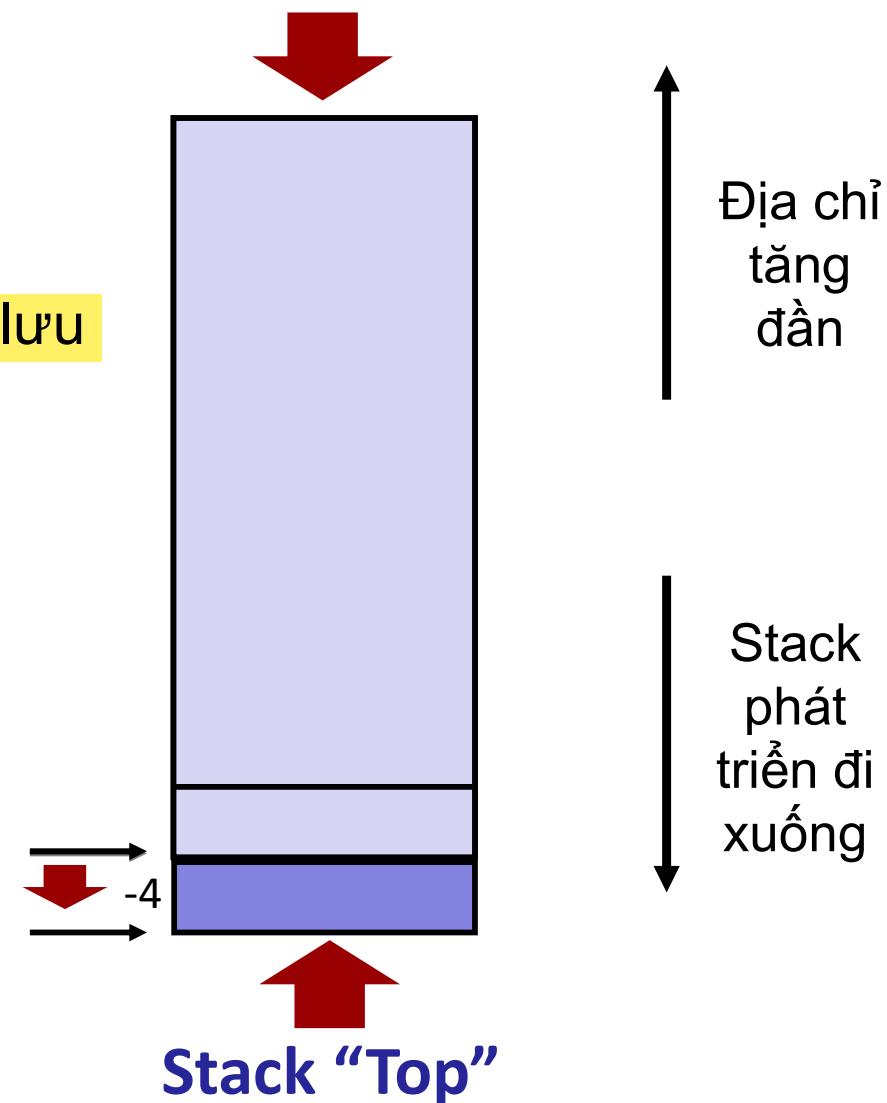
### ■ `pushl Src`

- Lấy giá trị từ Src
- Giảm `%esp` xuống 4 bytes
- Ghi giá trị lấy được vào địa chỉ đang lưu trong `%esp`

stack mở rộng xuống

Stack Pointer: `%esp`

Stack “Bottom”

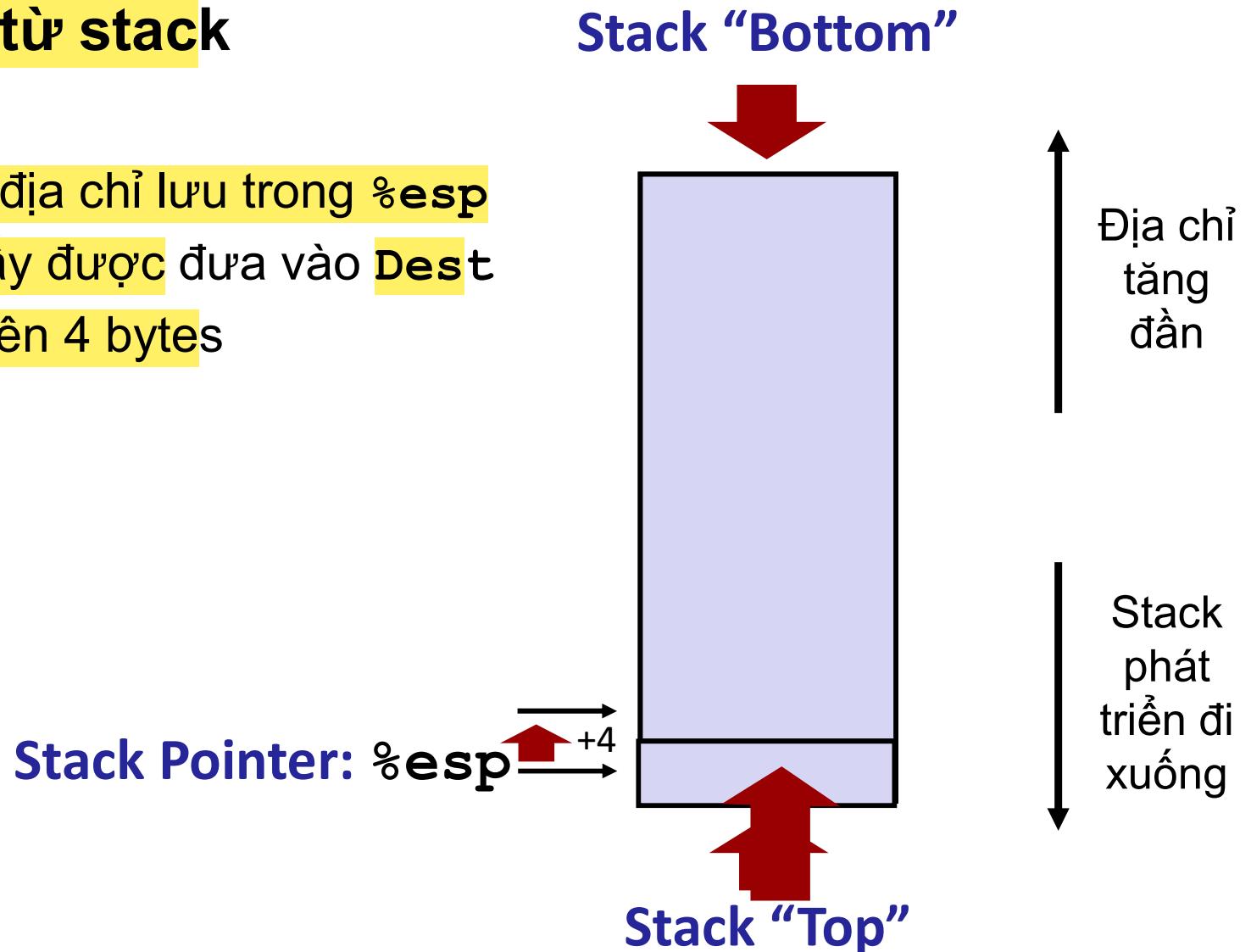


# IA32 Stack: Pop

- **Lấy dữ liệu từ stack**

- **popl Dest**

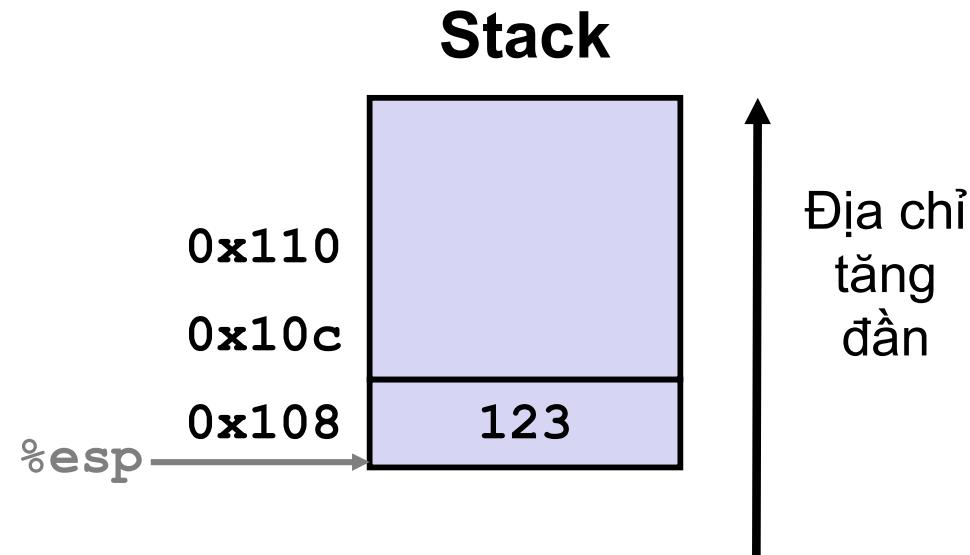
- Lấy giá trị ở địa chỉ lưu trong `%esp`
- Đưa giá trị lấy được đưa vào `Dest`
- Tăng `%esp` lên 4 bytes



# IA32 Stack: Push and Pop – Ví dụ

- **%esp = 0x108**
- **%eax = 0x1234**
- **%ebx = 0xABCD**

Các thanh ghi và stack thay đổi như thế nào khi thực hiện lần lượt các lệnh sau?



## 1. push %eax

esp -4 = 0x104  
eax = 0x1234  
ebx = 0xabcd  
stack có thêm 1 giá trị 0x1234

## 2. pop %ebx

eax không đổi  
esp + 4 -> tăng 0x108  
ebx = 0x1234

# IA32 Stack: Push and Pop – Ví dụ 2

- $\%esp = 0x108$
- $\%eax = 0x104$
- $\%ebx = 0xABCD$

Với các lệnh push dưới đây, giá trị bao nhiêu được đưa vào stack?

1. push \$0x100

0x100

esp = 0x104

2. push %eax

0x104

esp 0x100

3. push (%eax)

0xEF

esp = 0XFC

4. push 0x100

0xAB

Địa chỉ	Giá trị
0x108	0xF0
0x104	0xEF
0x100	0xAB

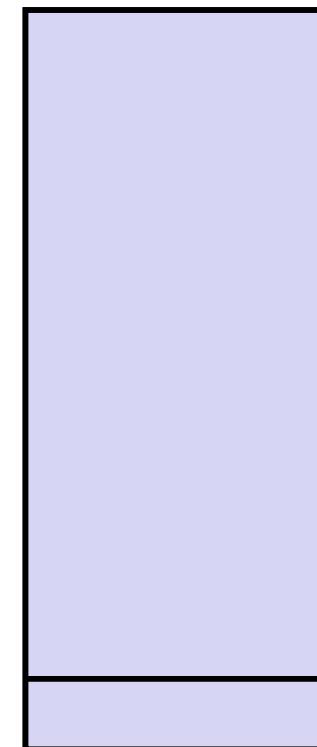
# x86-64 Stack?

- **Thanh ghi %rsp**
- **Các lệnh push hay pop**

- Thay đổi (cộng/trừ) giá trị %rsp 8 bytes

**Stack Pointer: %rsp** →

**Stack “Bottom”**



Địa chỉ  
tăng  
đần

Stack  
phát  
triển đi  
xuống



**Stack “Top”**

# Nội dung

---

- Thủ tục (Procedures)
  - Cấu trúc stack
  - Gọi hàm trong IA32
    - Chuyển luồng
    - Truyền dữ liệu
    - Quản lý dữ liệu cục bộ
  - Gọi hàm trong x86-64
  - Minh họa hàm đệ quy
- Bài tập về hàm
- Dịch ngược – Reverse engineering

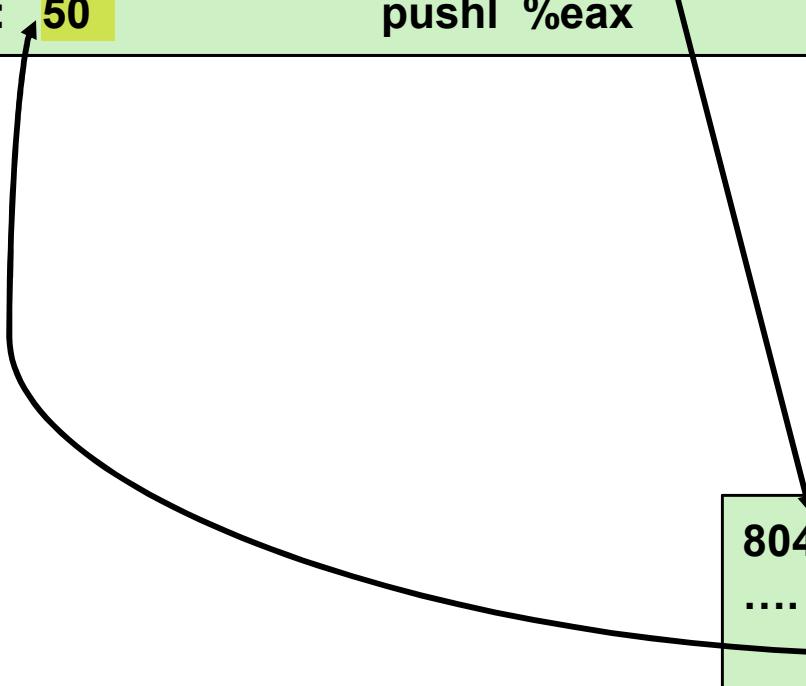
# Chuyển luồng thực thi hàm

```
804854e: e8 3d 06 00 00  call  8048b90 <main>
8048553: 50                pushl %eax
```

```
8048b90 <main>:
```

....

```
ret
```



# Chuyển luồng thực thi hàm

- Mỗi hàm đều có địa chỉ bắt đầu, thường được gán **label**
- Stack hỗ trợ gọi hàm và trả về từ hàm
  - Gọi 1 hàm con
  - Trả về hàm mẹ từ hàm con
- **Gọi hàm: call label**
  - Lưu **địa chỉ trả về** (return address) vào stack (push)
  - Nhảy đến **label** để thực thi
- **Trả về từ hàm: ret**
  - Lấy **địa chỉ trả về** ra từ stack (pop)
  - Nhảy đến **địa chỉ lấy được** để quay về hàm mẹ
- **Địa chỉ trả về (Return address):**
  - Địa chỉ **câu lệnh assembly tiếp theo** của **hàm mẹ cần thực thi ngay** phía sau lệnh **call** hàm con
  - Ví dụ trong mã assembly bên:
    - Địa chỉ trả về = 0x8048553

```
804854e: e8 3d 06 00 00  call 8048b90 <main>
8048553: 50          pushl %eax
```

# Ví dụ: Gọi hàm và Trả về hàm

804854e:	e8 3d 06 00 00	call 8048b90 <main>
8048553:	50	pushl %eax

Địa chỉ trả về? **0x8048553** → nằm trong %eip khi thực thi đến call

call main  $\longleftrightarrow$  - Push địa chỉ trả về vào stack: **push \$0x8048553**  
- Nhảy đến nhãn main: **jmp 8048b90**

- Push địa chỉ trả về vào stack: **push %eip**  
- Nhảy đến nhãn main: **jmp 8048b90**

8048b90: main:

...

8048591: c3 **ret**

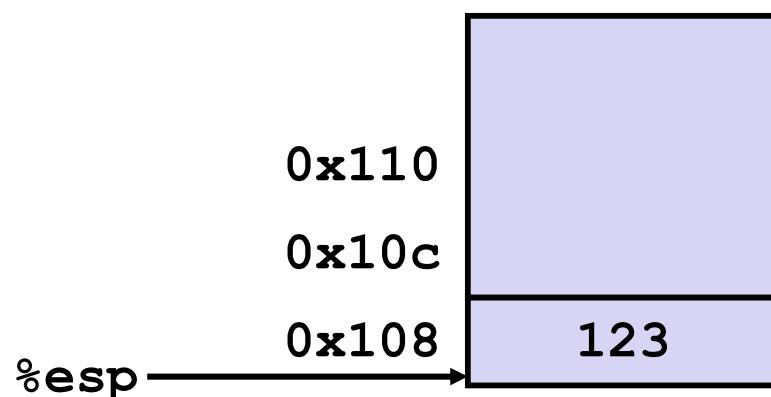
ret  $\longleftrightarrow$  - Pop địa chỉ trả về vào stack: **pop %eip**  
- Nhảy đến lệnh ở địa chỉ trả về: **jmp \*%eip**

# Ví dụ: Gọi hàm

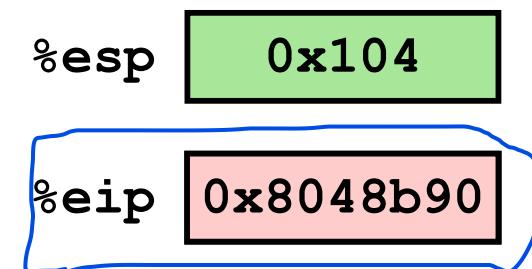
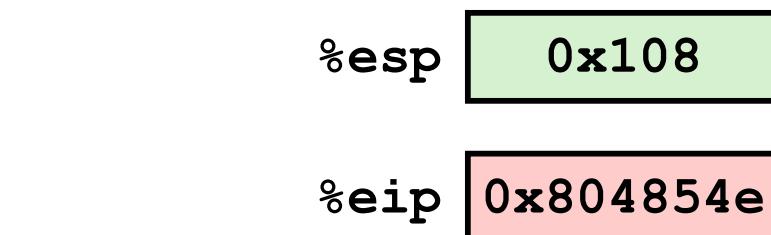
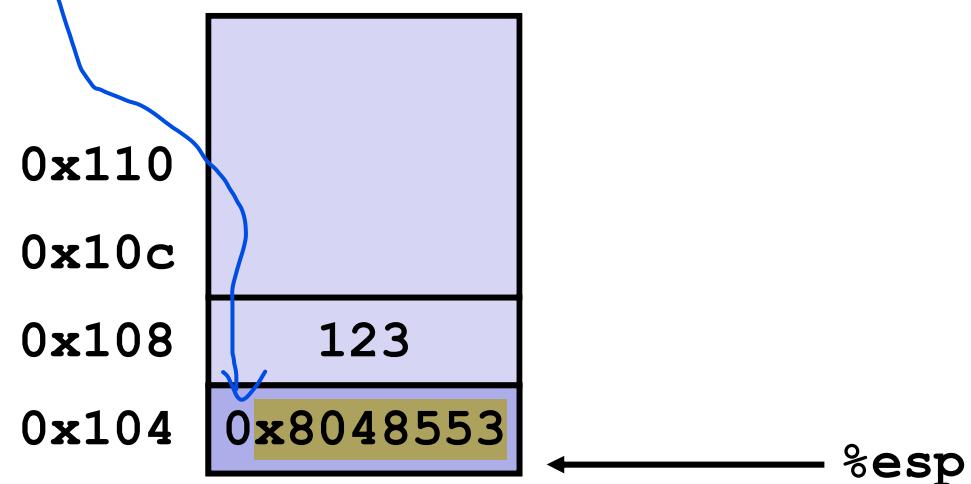
```
804854e: e8 3d 06 00 00    call  8048b90 <main>
8048553: 50                pushl %eax
```

call 0x8048b90 = push %eip  
jmp 0x8048b90

trước call



call 8048b90



%eip: program counter

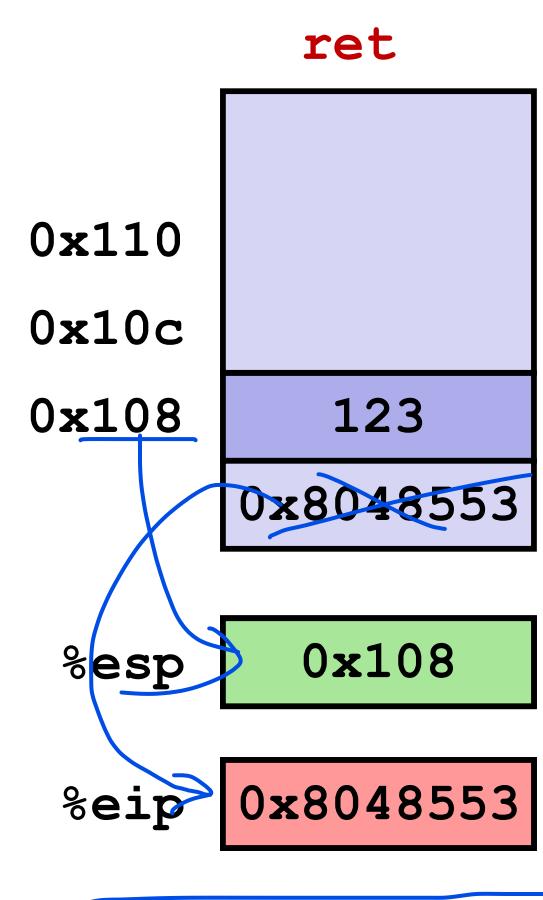
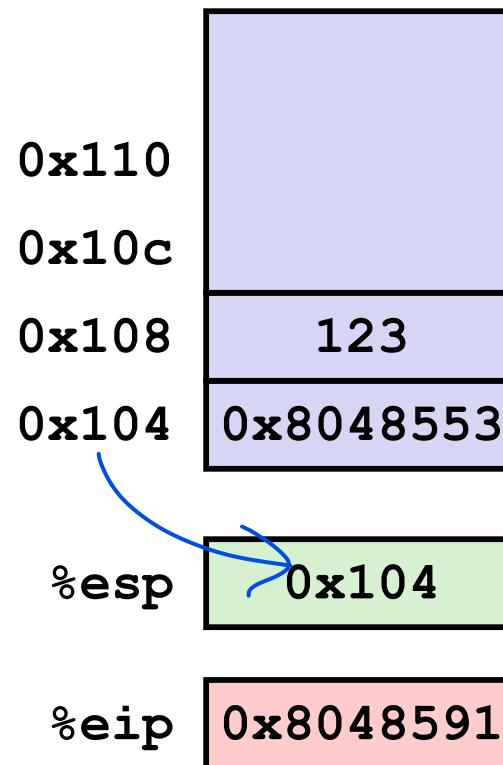
# Ví dụ: Trả về hàm

8048591: c3

ret

ret = pop %eip  
jmp \*%eip

trước ret



`%eip`: program counter

# Gọi và trả về hàm – Ví dụ

```
int main()
{
    int result = func(5,6);
    return result;
}

int func(int x, int y)
{
    int sum = 0;
    sum = x + y;
    return sum;
}
```

<p>main:</p> <pre>pushl %ebp movl %esp, %ebp subl \$16, %esp pushl \$6 pushl \$5 call func movl %eax, -4(%ebp) movl -4(%ebp), %eax leave ret</pre>	<p>func:</p> <pre>pushl %ebp movl %esp, %ebp subl \$16, %esp movl \$0, -4(%ebp) movl 8(%ebp), %edx movl 12(%ebp), %eax addl %edx, %eax movl %eax, -4(%ebp) movl -4(%ebp), %eax leave ret</pre>
--	--

# Hoạt động của hàm dựa trên stack

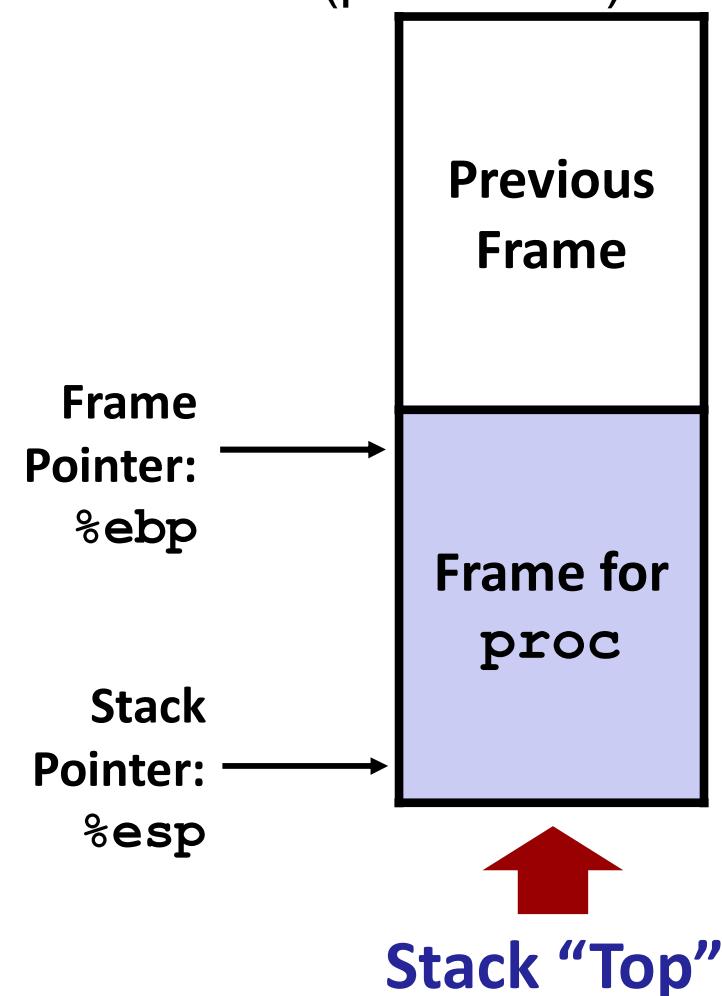
## ■ Stack được cấp phát bằng **Frames**

- 1 hàm (procedure) = 1 stack frame
- Hỗ trợ lưu trữ các thông tin dùng để gọi và trả về hàm (procedure)
  - Địa chỉ trả về
  - Các tham số (arguments)
  - Các biến cục bộ (local variables)

## ■ 1 Frame là vùng nhớ xác định bởi %ebp và %esp

- %ebp trỏ đến vị trí cố định
- %esp lưu động
- Thường truy xuất các dữ liệu trên stack dựa trên %ebp

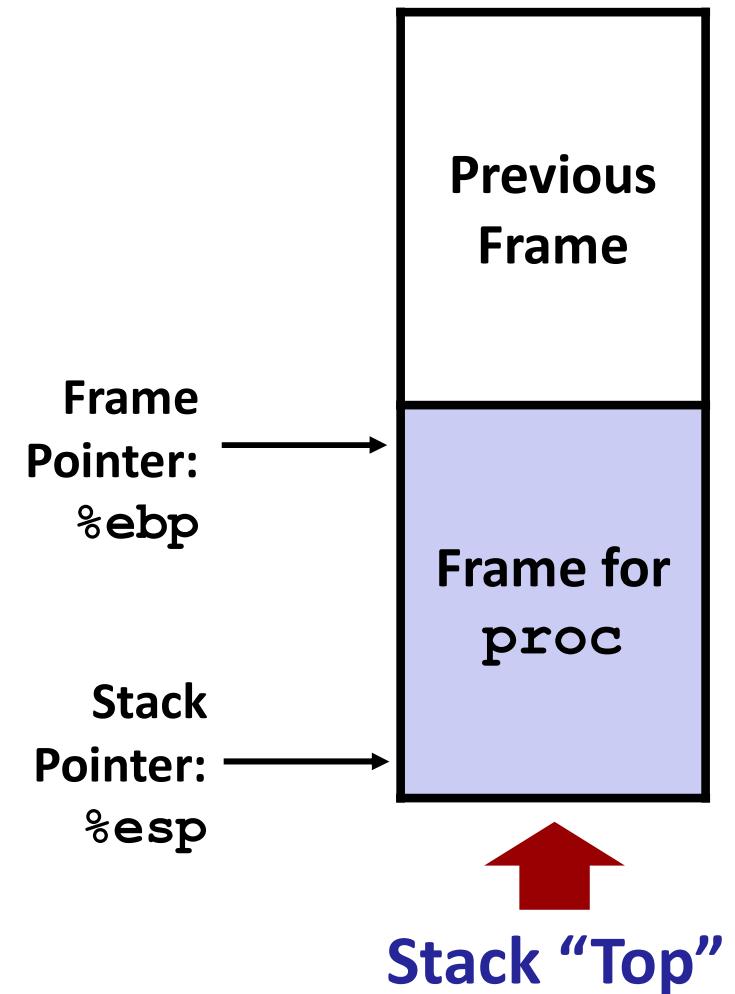
Ví dụ: -4(%ebp)



# Stack Frames trong IA32

## ■ Quy tắc ngăn xếp

- Stack frame của 1 hàm tồn tại trong một khoảng thời gian từ lúc hàm được gọi đến lúc kết thúc.
  - Khi nào hàm được gọi thì stack frame của nó sẽ được tạo.
  - Khi kết thúc, stack frame sẽ được thu hồi.
- Hàm thực thi trước thì stack được cấp phát trước.
  - Stack frame cấp phát sau sẽ nằm ở các địa chỉ thấp hơn.
- Hàm kết thúc trước thì stack thu hồi trước.



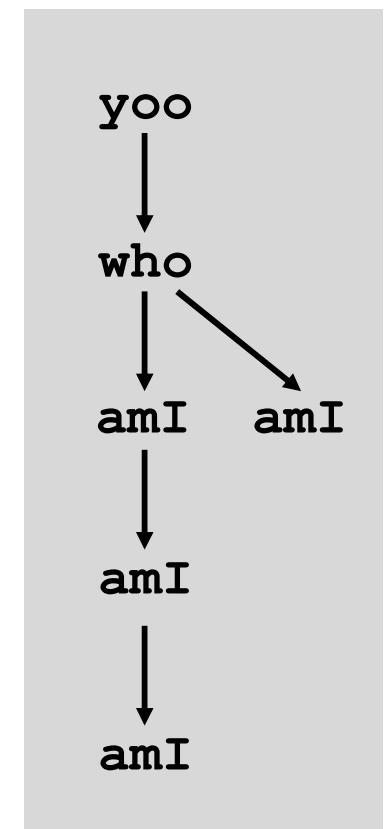
# Ví dụ chuỗi gọi hàm

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

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

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

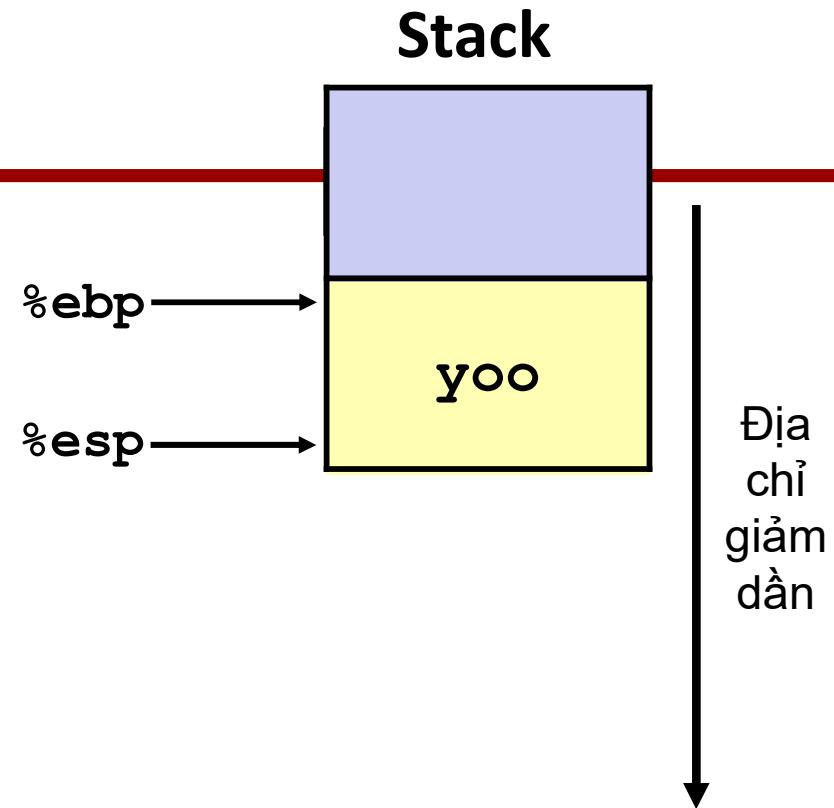
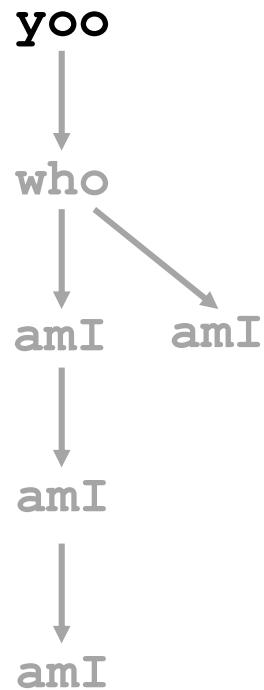
Example  
Call Chain



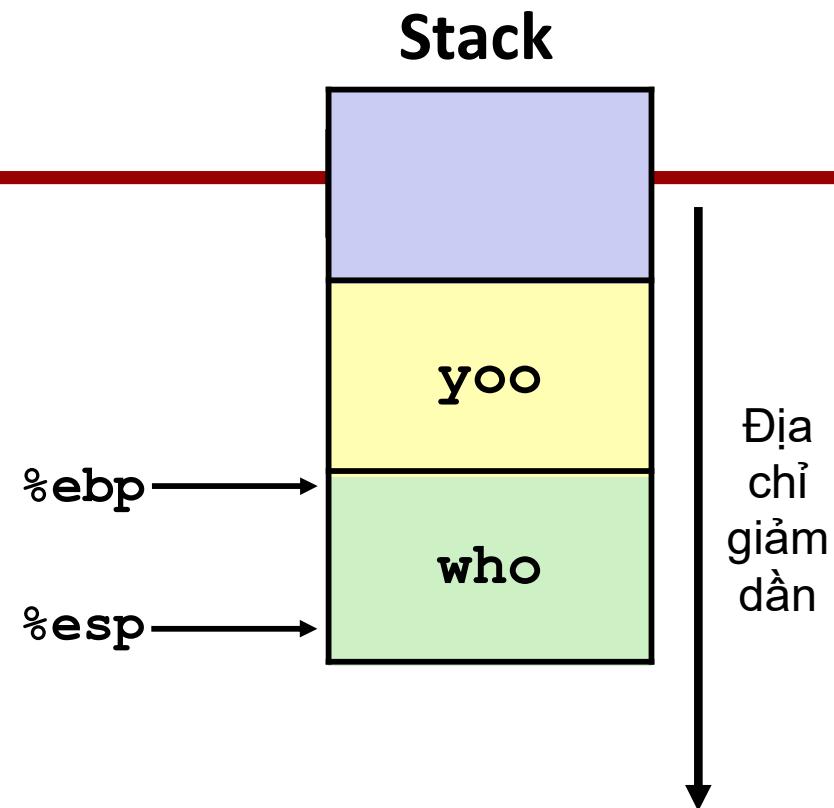
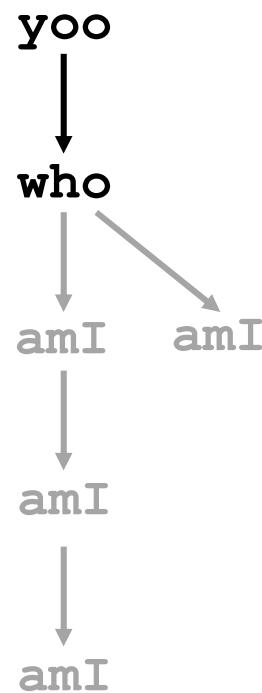
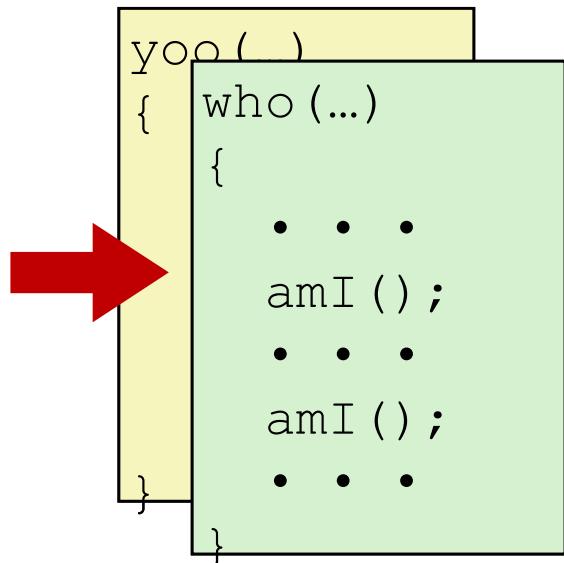
Procedure **amI()** is recursive

# Ví dụ

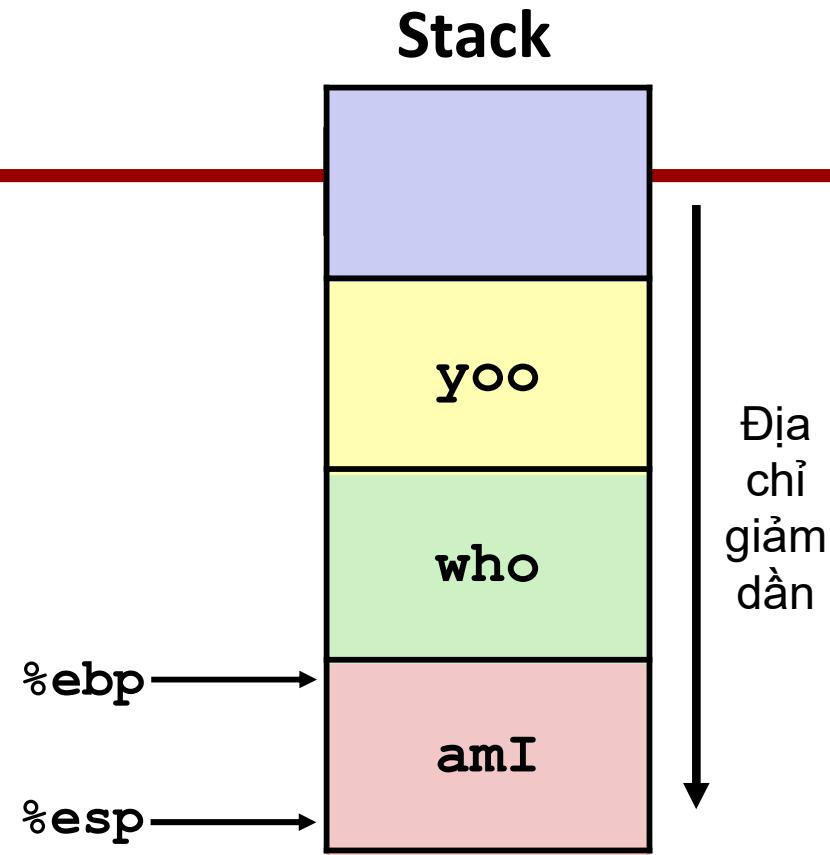
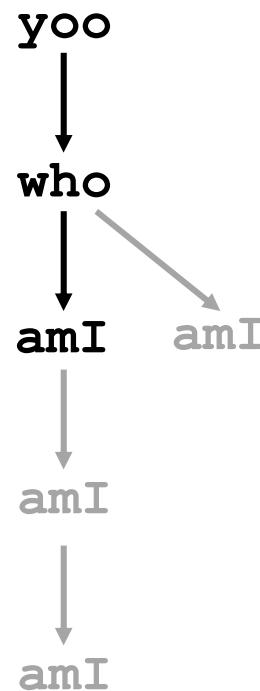
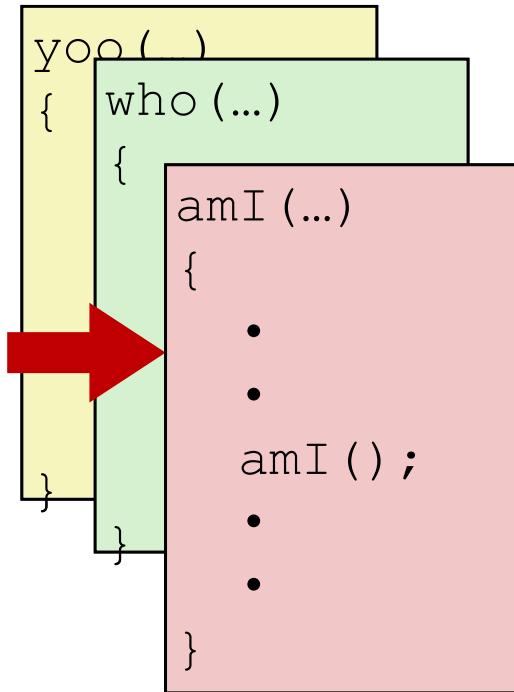
```
yoo (...)  
{  
    •  
    •  
    who () ;  
    •  
    •  
}  
→
```



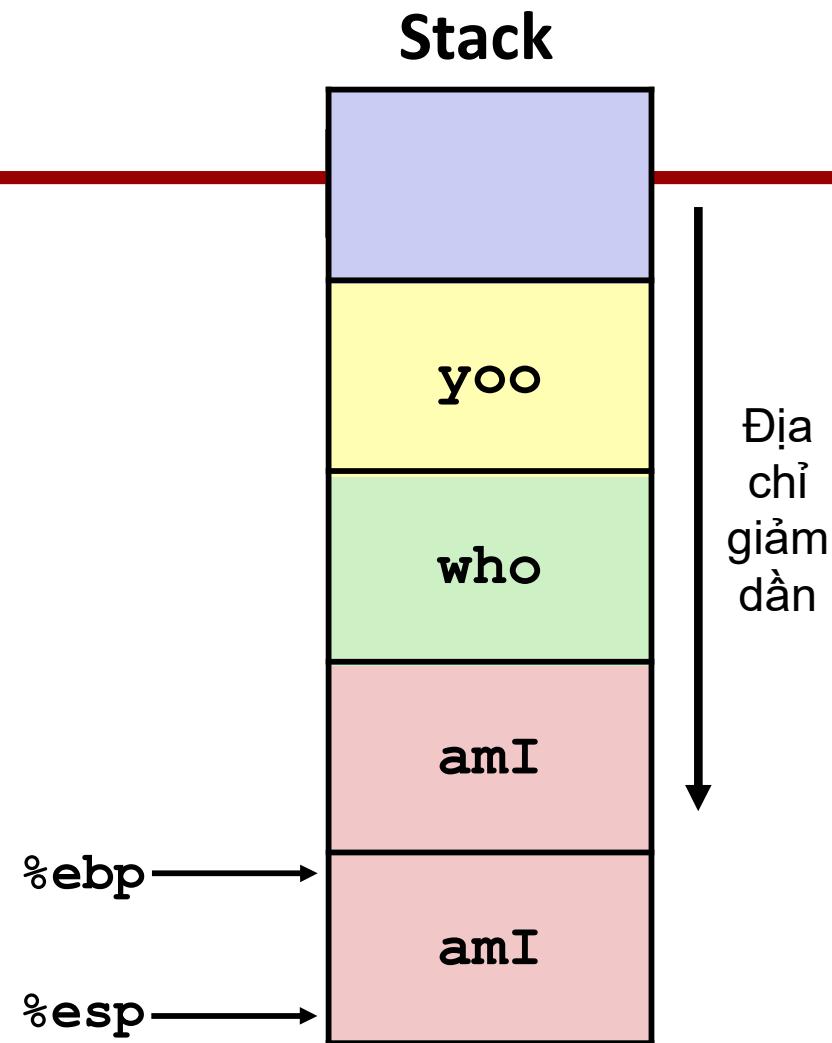
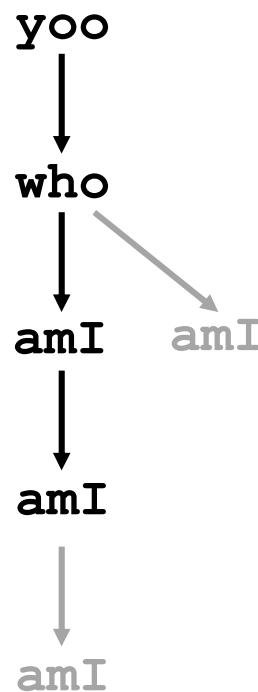
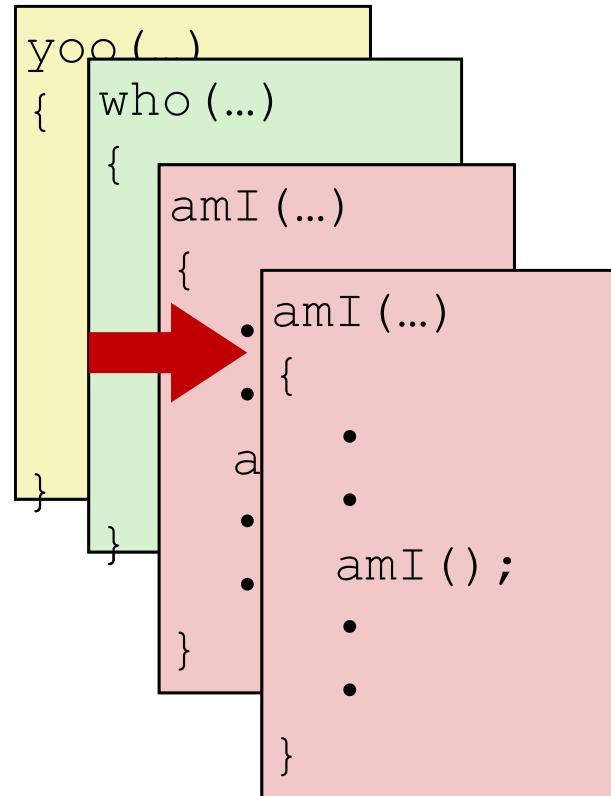
# Ví dụ



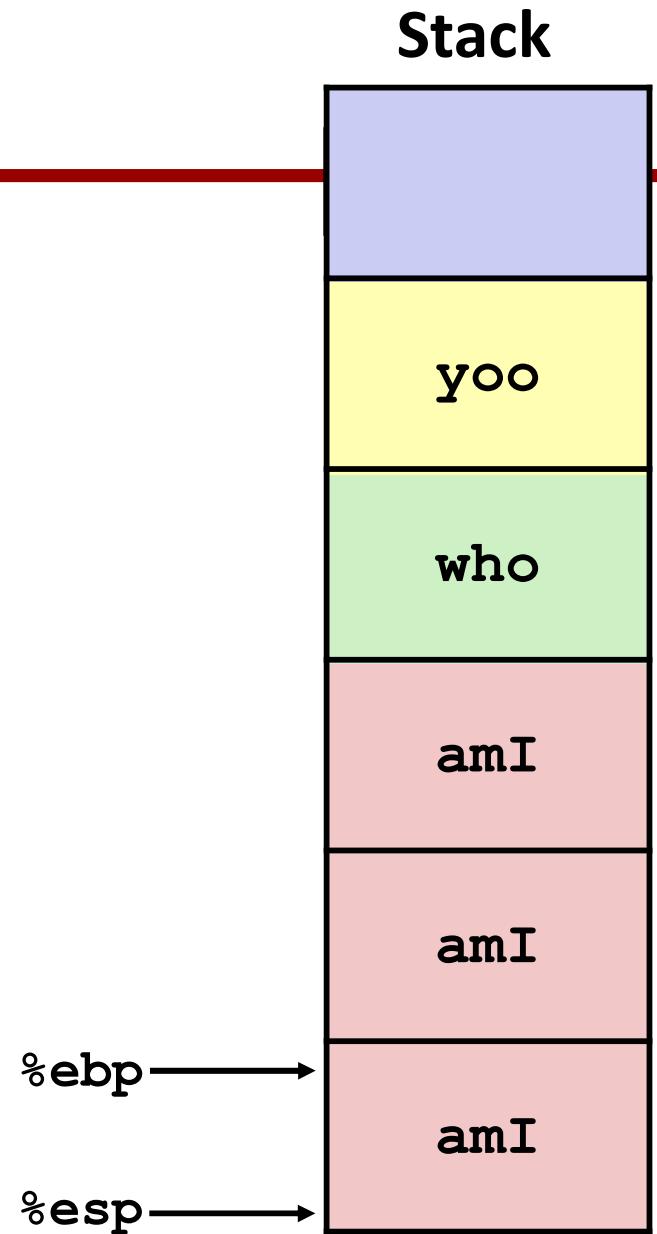
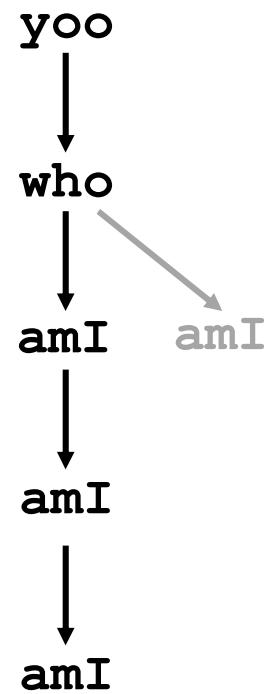
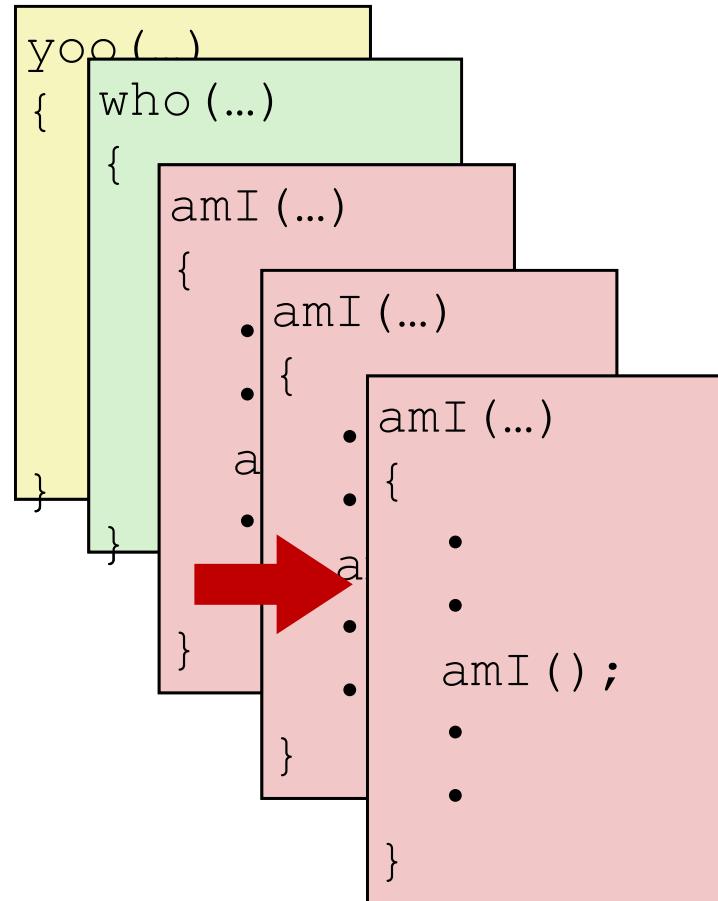
# Ví dụ



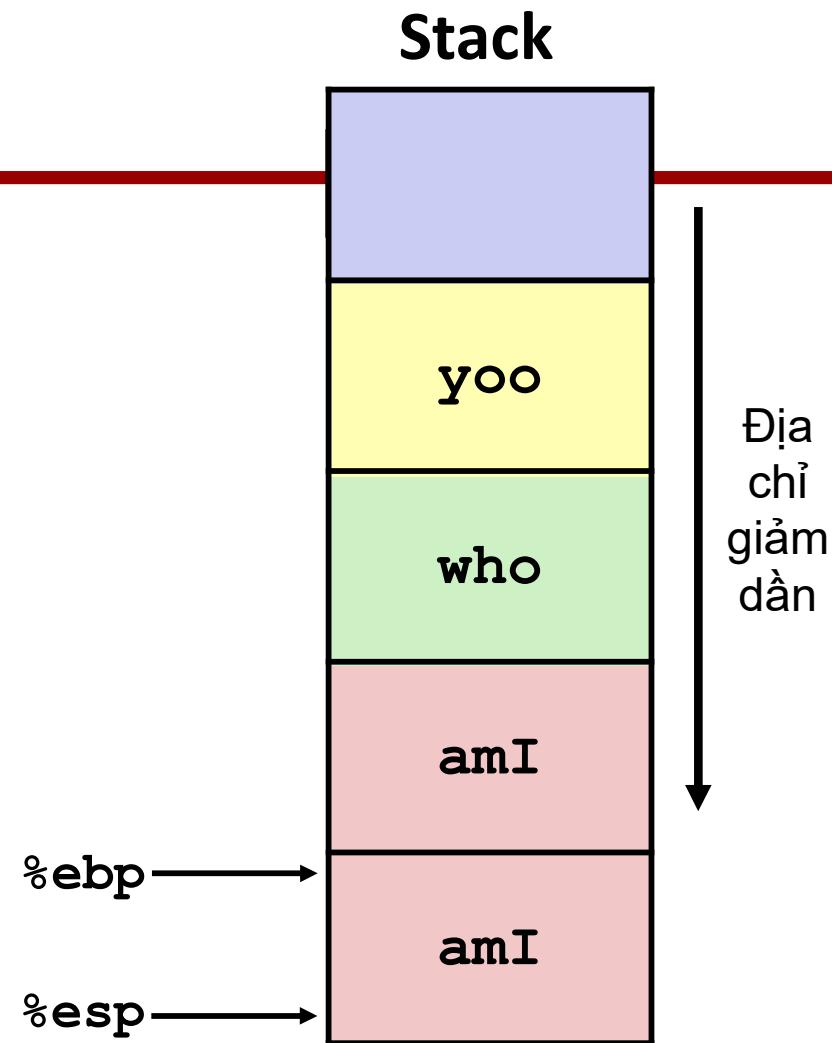
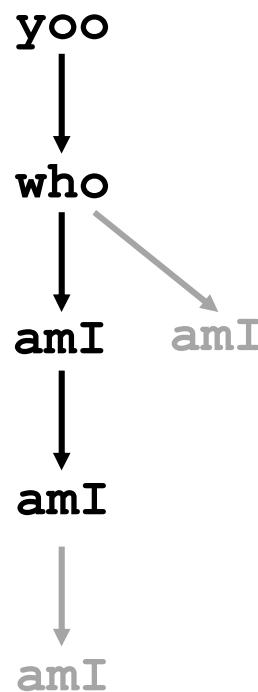
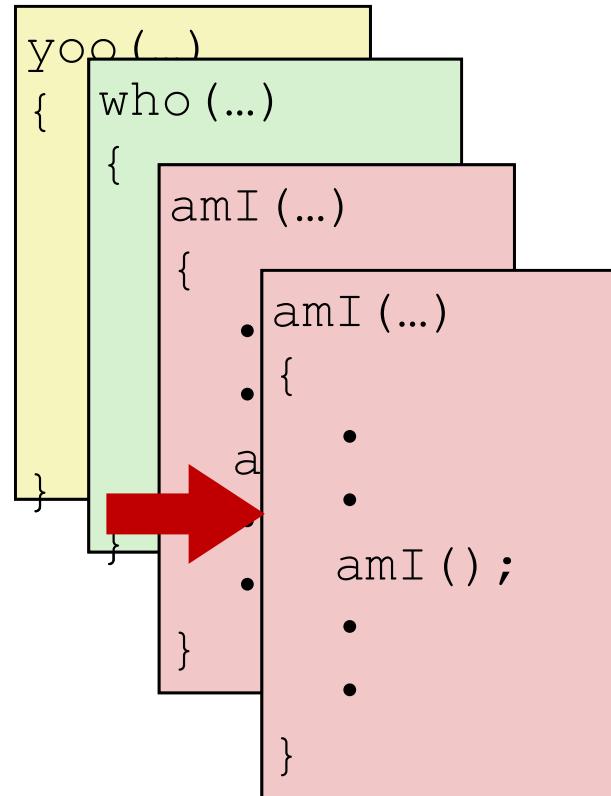
# Ví dụ



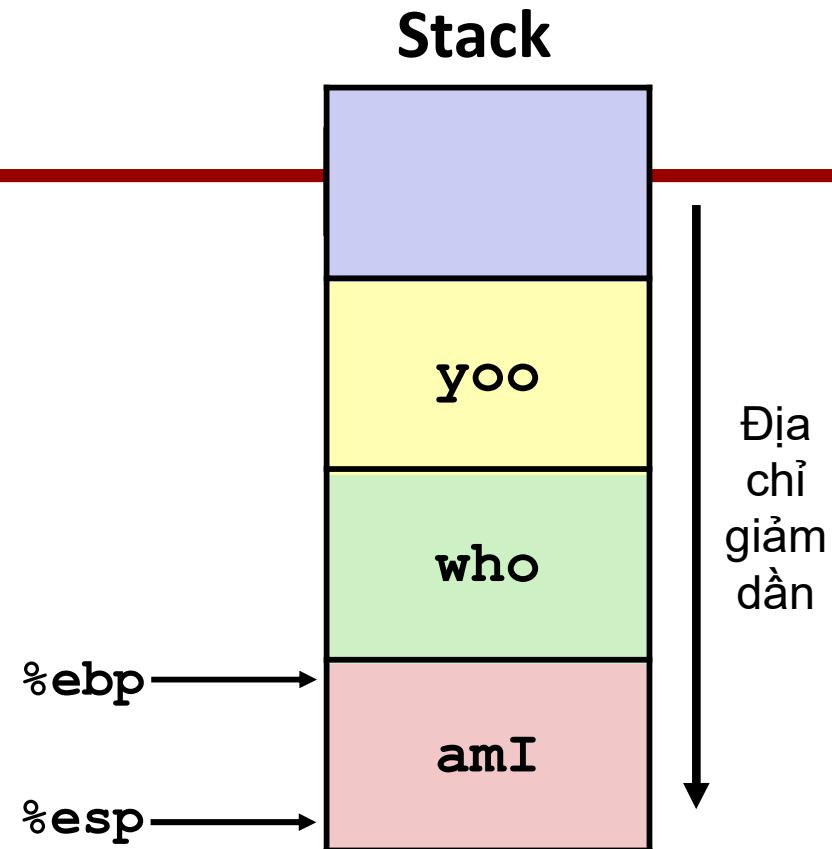
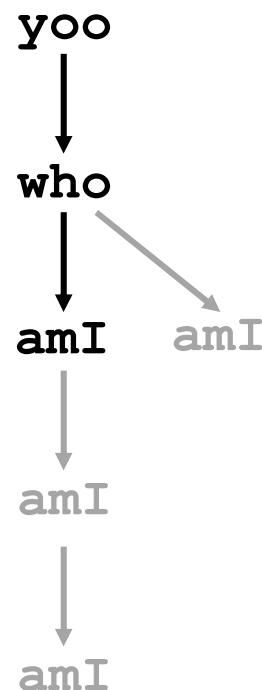
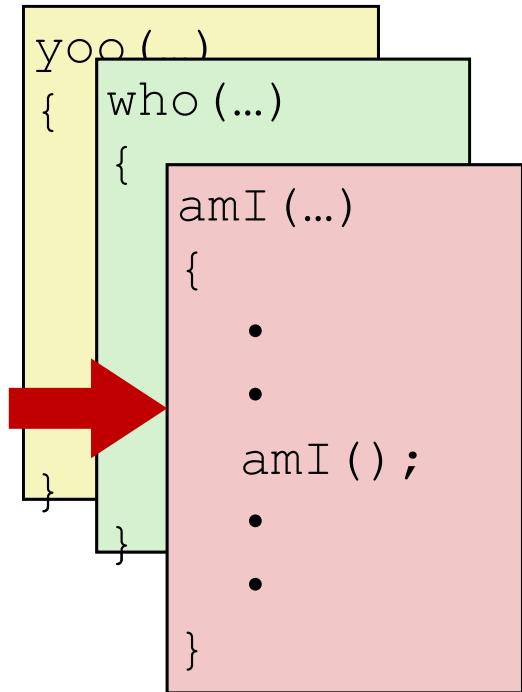
# Ví dụ



# Ví dụ

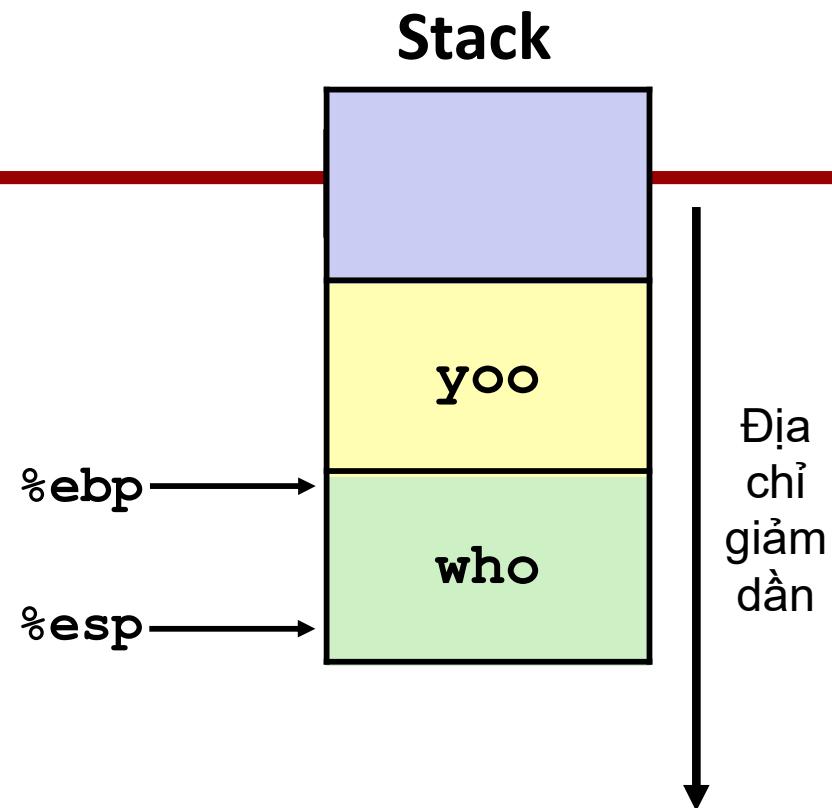
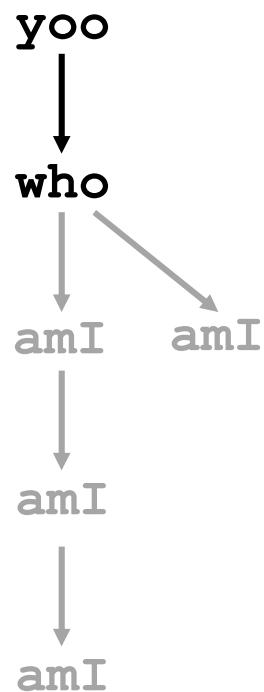


# Ví dụ

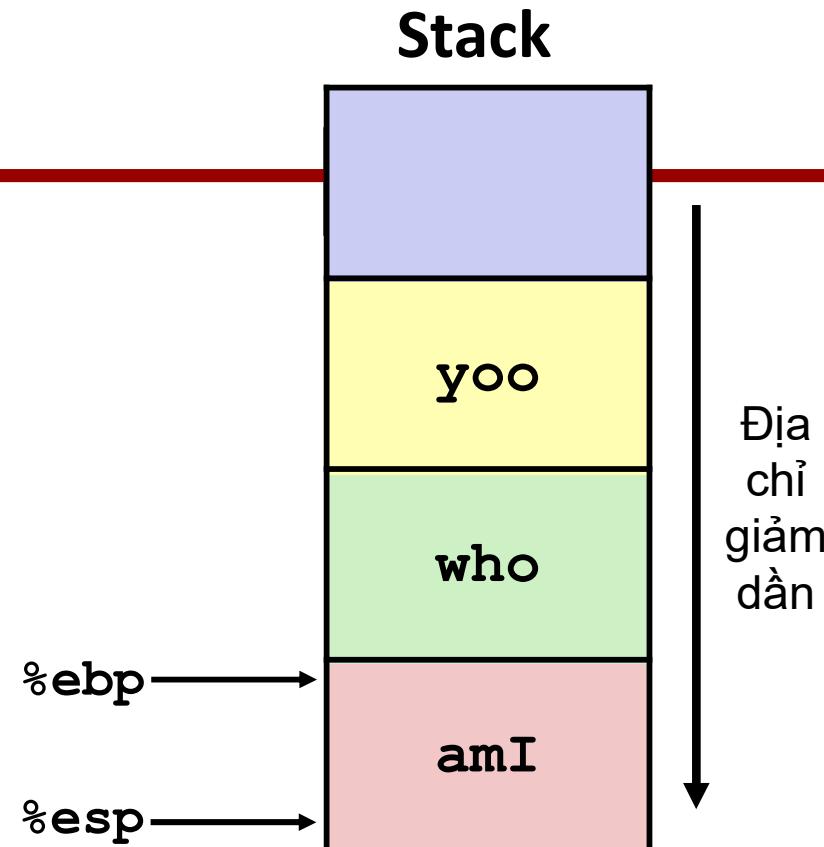
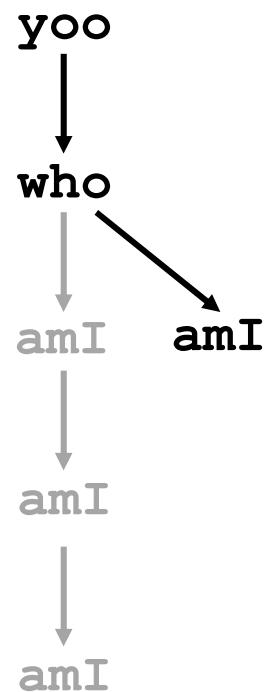
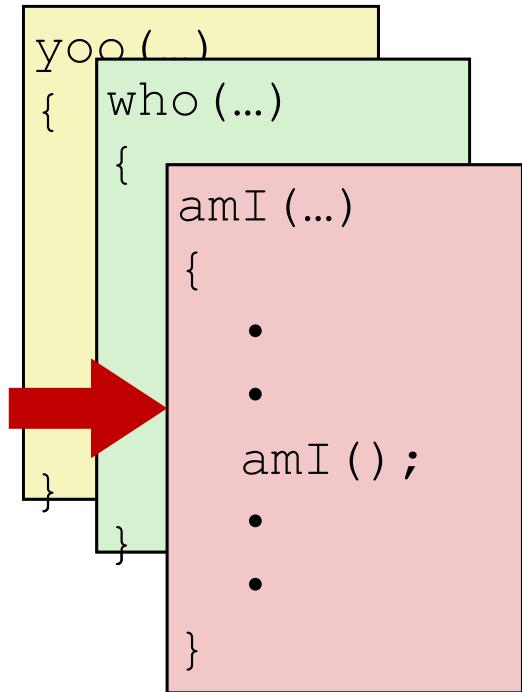


# Ví dụ

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

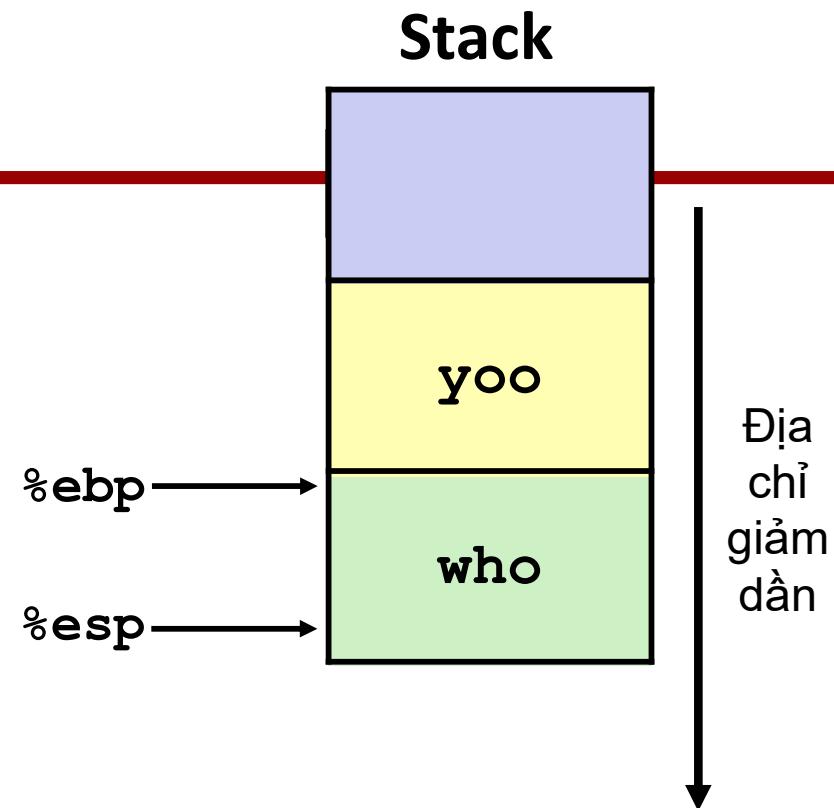
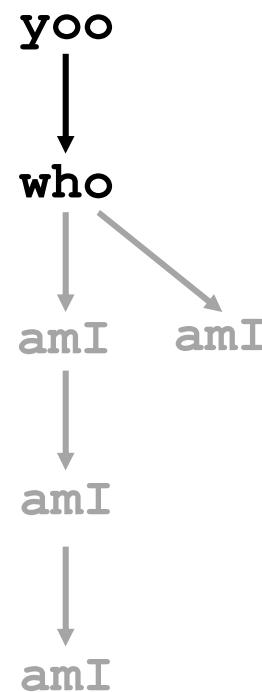


# Ví dụ



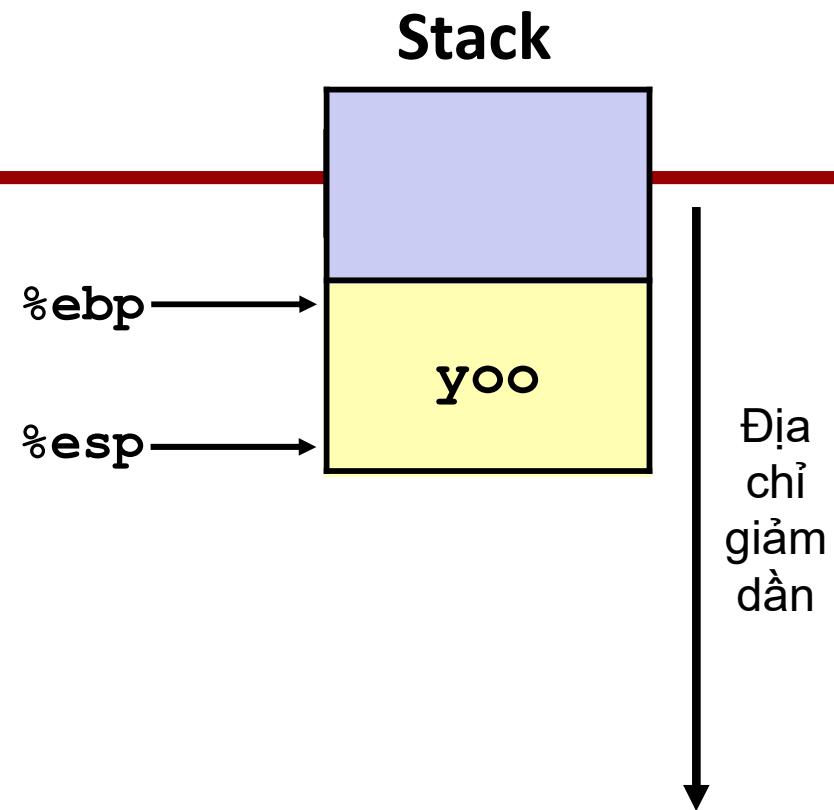
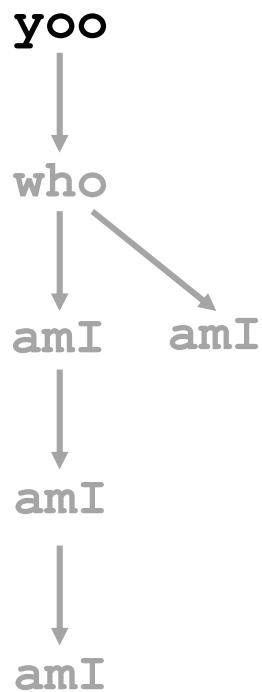
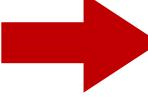
# Ví dụ

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



# Ví dụ

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



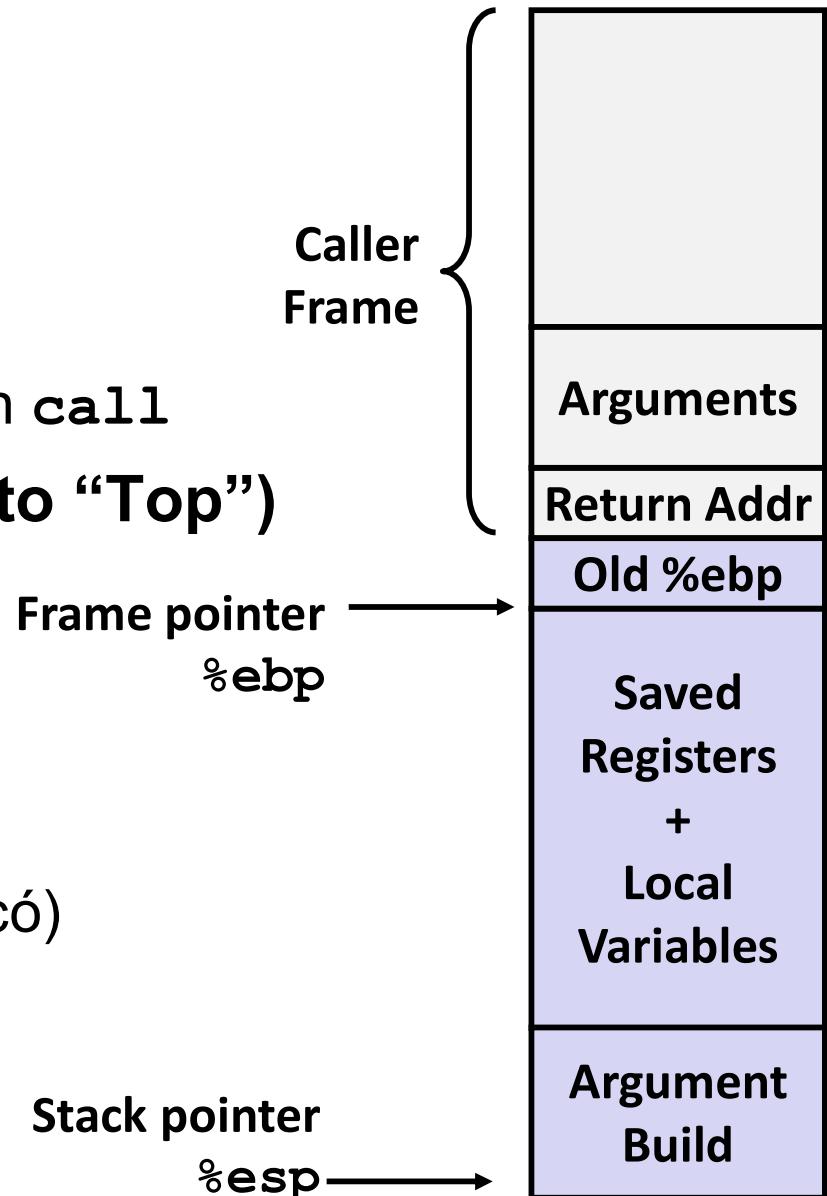
# IA32 Stack Frame chứa thông tin gì?

## ■ Stack frame của hàm mẹ

- Các tham số cho hàm con
  - Đưa vào bằng các lệnh push/mov
- Địa chỉ trả về (Return address)
  - *Tự động đẩy* vào stack khi chạy lệnh call

## ■ Stack Frame của 1 hàm (“Bottom” to “Top”)

- Frame pointer của hàm mẹ (%ebp)
- Những thanh ghi được lưu lại (nếu có)
- Các biến cục bộ của hàm
- “Argument build”  
Tham số cho các hàm muốn gọi (nếu có)



# Quản lý IA32 Stack Frame

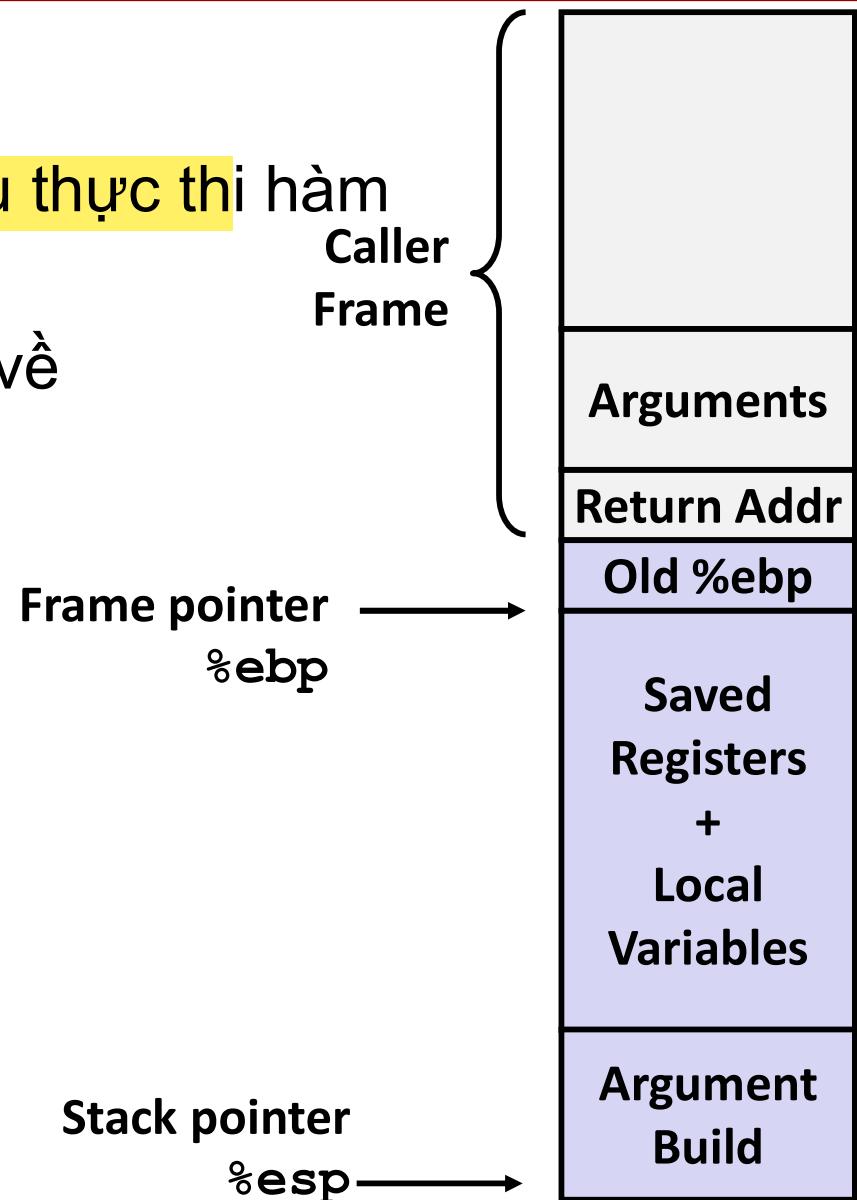
## ■ 2 hoạt động:

- Cấp phát không gian khi bắt đầu thực thi hàm

- “Set-up” code trong assembly

- Thu hồi không gian khi hàm trả về

- “Finish” code trong assembly



# IA32 Stack frame - Set up & Finish

## ■ Stack Frame – Set up

- Thực hiện khi 1 hàm bắt đầu thực thi
- Lưu lại %ebp của hàm trước
- Thiết lập %ebp cho stack frame của nó
- Lưu lại các thanh ghi sẽ sử dụng trong hàm (nếu có)

swap:

```
pushl %ebp  
movl %esp, %ebp  
pushl %ebx
```

} Set Up

```
movl 8(%ebp), %edx  
movl 12(%ebp), %ecx  
movl (%edx), %ebx  
movl (%ecx), %eax  
movl %eax, (%edx)  
movl %ebx, (%ecx)
```

## ■ Stack frame - Finish

- Thực hiện khi 1 hàm chuẩn bị trả về
- Khôi phục giá trị cũ của các thanh ghi đã sử dụng (nếu có)
- Khôi phục %ebp của hàm trước

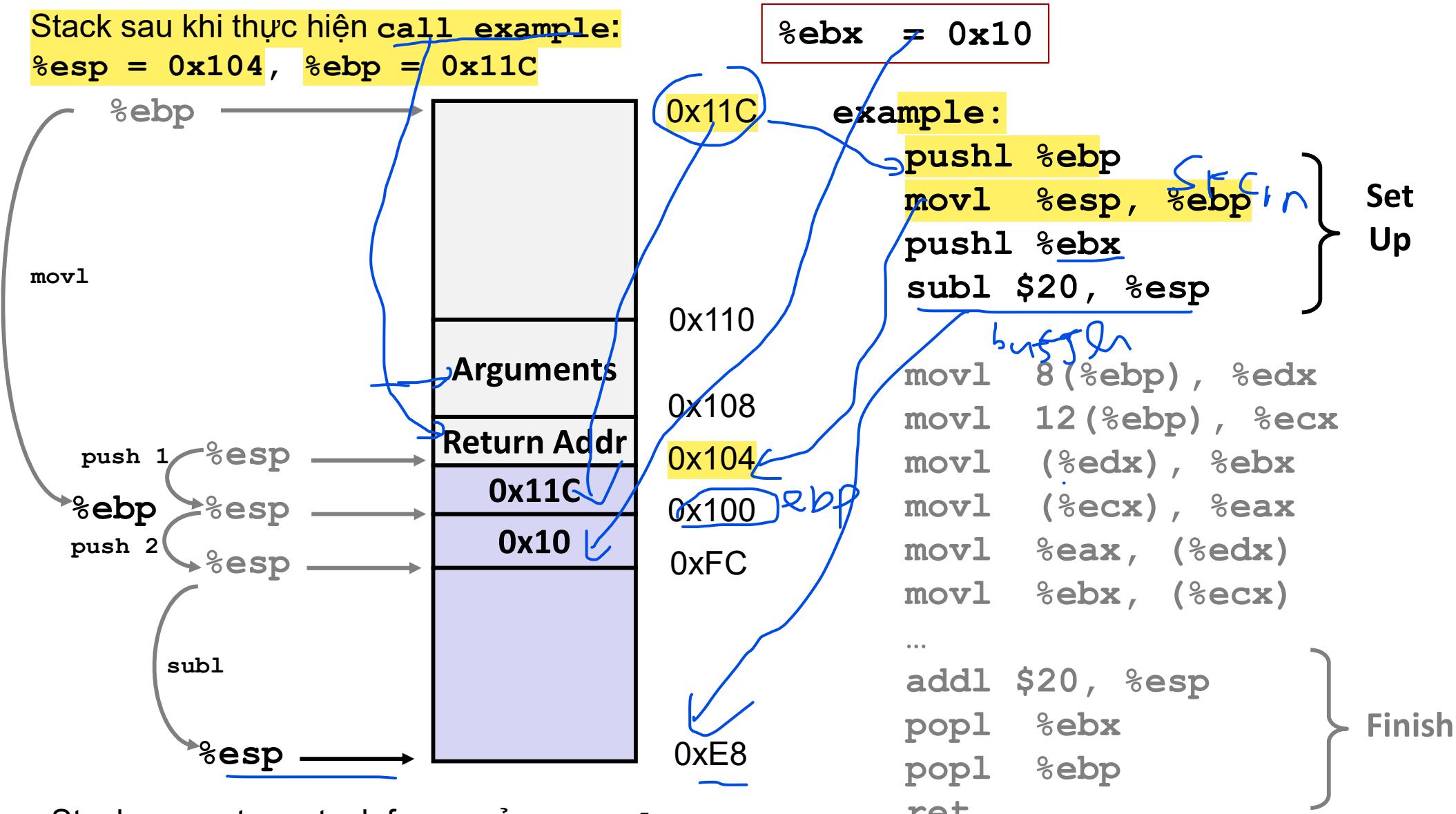
```
popl %ebx  
popl %ebp  
ret
```

} Finish

# Stack frame set up – Ví dụ

Stack sau khi thực hiện `call example`:

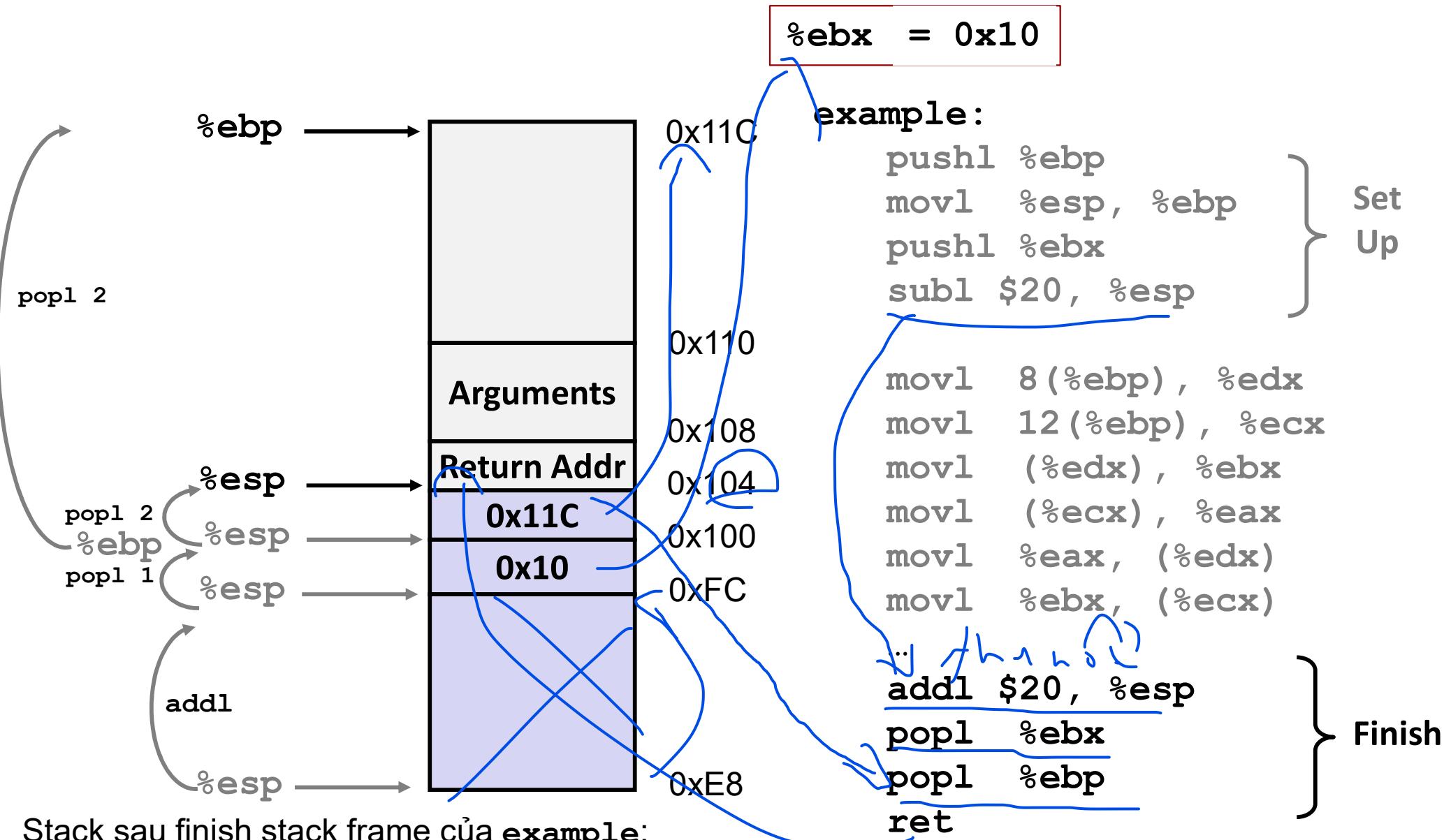
`%esp = 0x104, %ebp = 0x11C`



Stack sau set-up stack frame của `example`:

`%esp = 0xE8, %ebp = 0x100`

# Stack frame Finish – Ví dụ



# Stack frame set up & Finish – Ví dụ

```
int func(int x, int y)
{
    int sum = 0;
    sum = x + y;
    return sum;
}
```

func:

pushl %ebp  
movl %esp, %ebp  
subl \$16, %esp

} Set Up

movl \$0, -4(%ebp)  
movl 8(%ebp), %edx  
movl 12(%ebp), %eax  
addl %edx, %eax  
movl %eax, -4(%ebp)  
movl -4(%ebp), %eax

leave  
ret

} Finish

- Gán %esp = %ebp
- Pop %ebp từ stack

# Nội dung

---

- **Thủ tục (Procedures)**
  - Cấu trúc stack
  - Gọi hàm trong IA32
    - Chuyển luồng
    - Truyền dữ liệu
    - Quản lý dữ liệu cục bộ
  - Gọi hàm trong x86-64
  - Minh họa hàm đệ quy
- Bài tập về hàm
- Dịch ngược – Reverse engineering

# Truyền tham số trong Stack frame IA32

## ■ Hàm mẹ (caller) đưa tham số vào stack cho hàm con (callee)

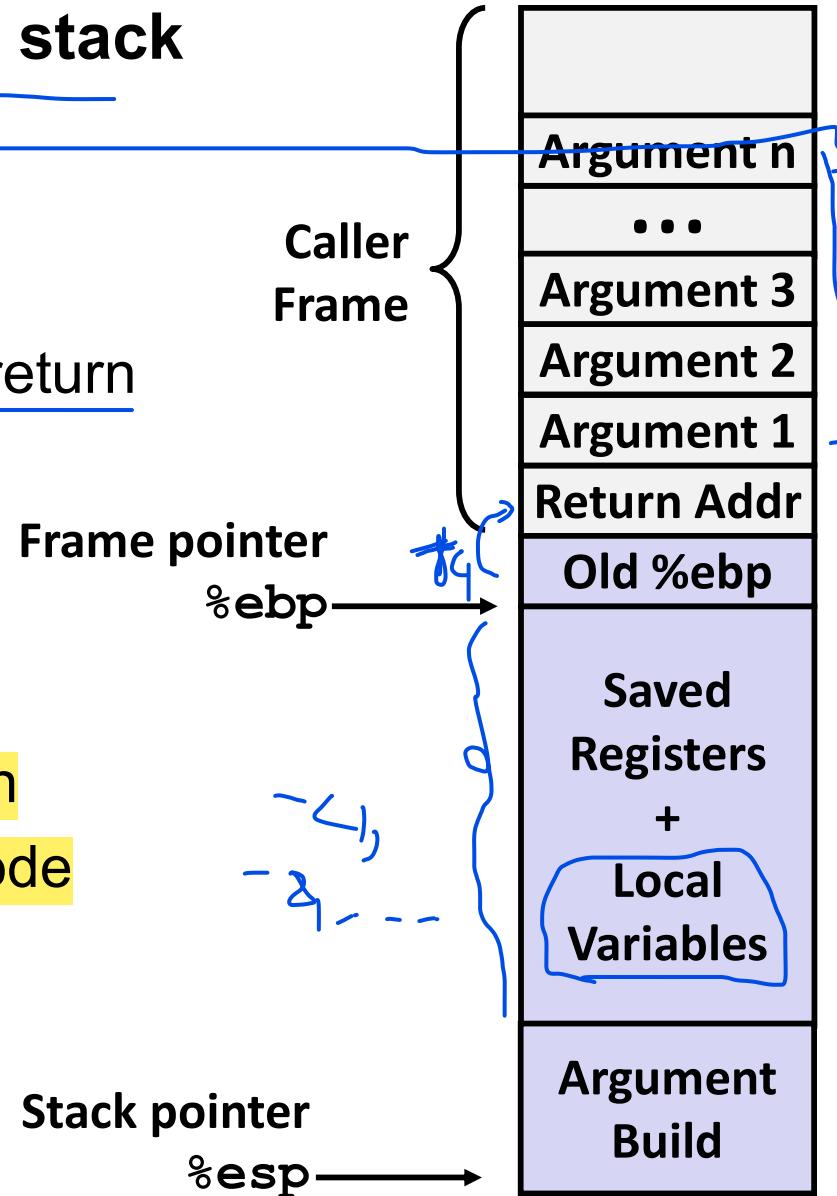
- Trước khi thực thi call label
  - Lệnh push/mov
  - Nằm ngay phía trước địa chỉ trả về (return address) trong stack
- Thứ tự: reverse order

↖ Gđ.

## ■ Hàm con (callee) truy xuất tham số

- Dựa trên vị trí so với %ebp của hàm con
  - %ebp sau khi hoàn thành “set up” code

tham số 1:  $ebp + 8$   
 $n: ebp + a*(n+1)$



# Truyền tham số cho hàm – Ví dụ 1

```
int main()
{
    int result = func(5,6);
    return result;
}
```

```
int func(int x, int y)
{
    int sum = 0;
    sum = x + y;
    return sum;
}
```

main:

```
pushl %ebp
movl %esp, %ebp
subl $16, %esp
pushl $6
pushl $5
call func
addl $8, %esp
movl %eax, -4(%ebp)
movl -4(%ebp), %eax
leave
ret
```

gần call  
là tham số đầu

func:

```
pushl %ebp
movl %esp, %ebp
subl $16, %esp biến cục bộ
movl $0, -4(%ebp)
movl 8(%ebp), %edx
movl 12(%ebp), %eax
addl %edx, %eax
movl %eax, -4(%ebp)
movl -4(%ebp), %eax
leave
ret
```

tham số

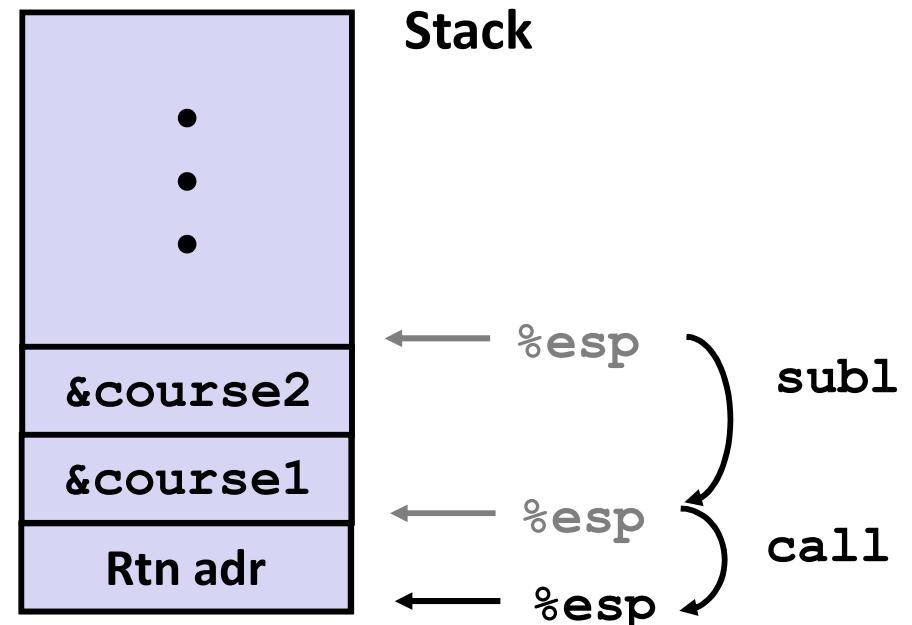
# Truyền tham số cho hàm: Ví dụ 2 - swap

```
int course1 = 15213;  
int course2 = 18243;  
  
void call_swap() {  
    swap(&course1, &course2);  
}
```

Gọi swap từ hàm call\_swap

```
call_swap:  
    . . .  
    subl    $8, %esp  
    movl    $course2, 4(%esp)  
    movl    $course1, (%esp)  
    call    swap  
    . . .
```

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



# Truyền tham số: Ví dụ swap

```
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 8(%ebp), %edx  
movl 12(%ebp), %ecx  
movl (%edx), %ebx  
movl (%ecx), %eax  
movl %eax, (%edx)  
movl %ebx, (%ecx)
```

```
popl %ebx  
popl %ebp  
ret
```

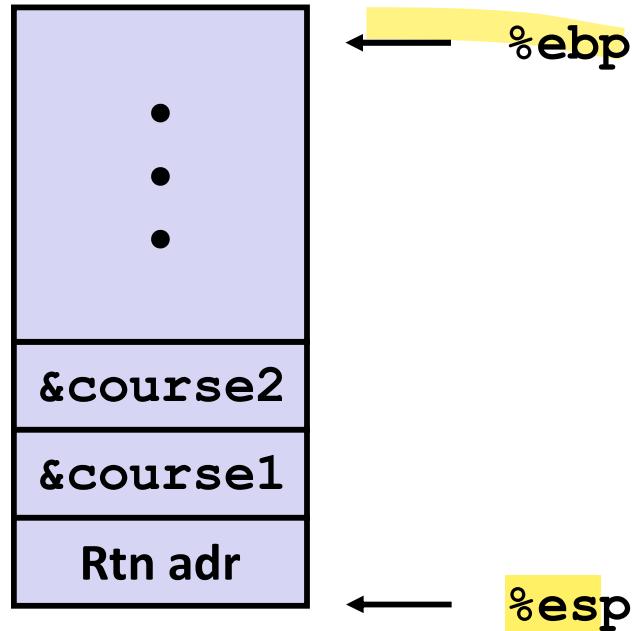
} Set Up

} Body

} Finish

# swap Setup #1

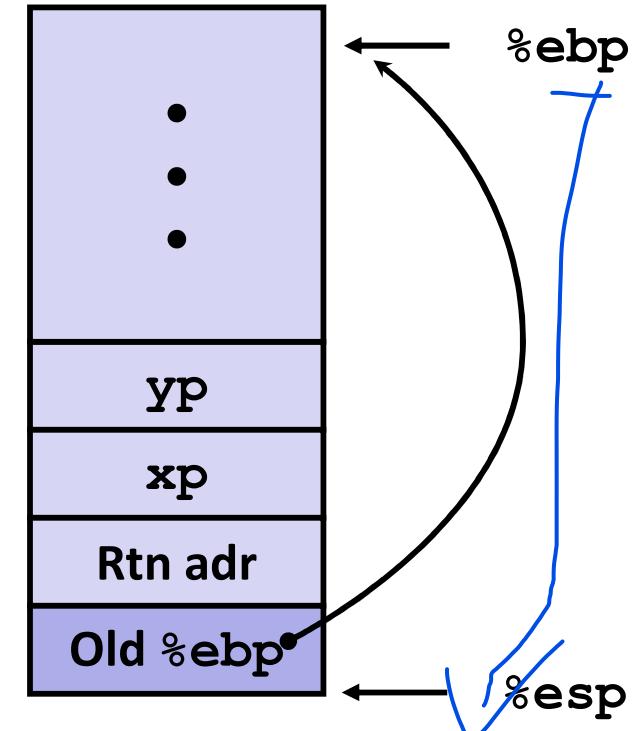
Entering Stack



swap:

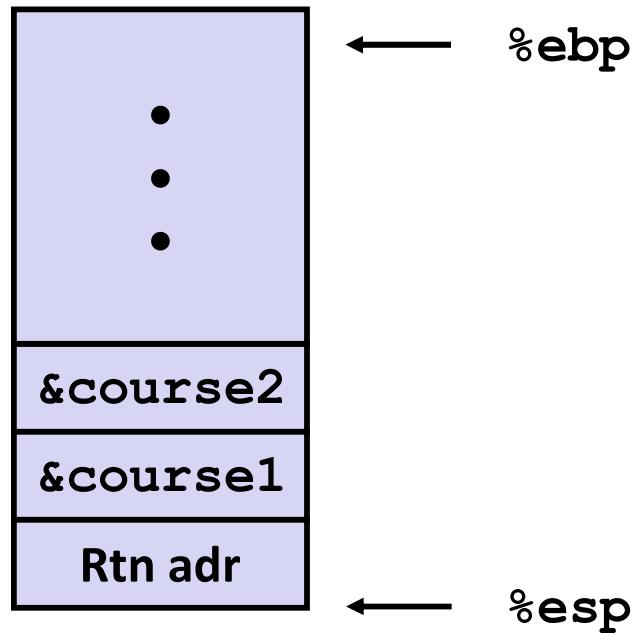
```
pushl %ebp
movl %esp,%ebp
pushl %ebx
```

Resulting Stack

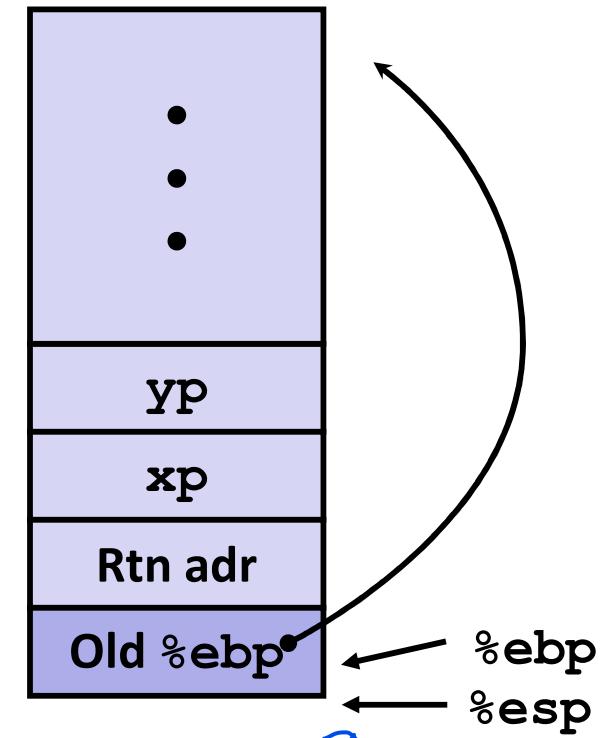


# swap Setup #2

Entering Stack



Resulting Stack

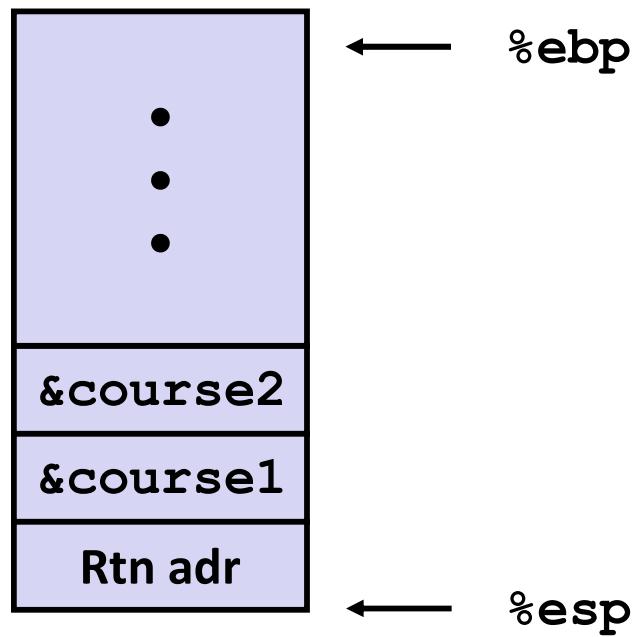


swap:

```
pushl %ebp
movl %esp,%ebp
pushl %ebx
```

# swap Setup #3

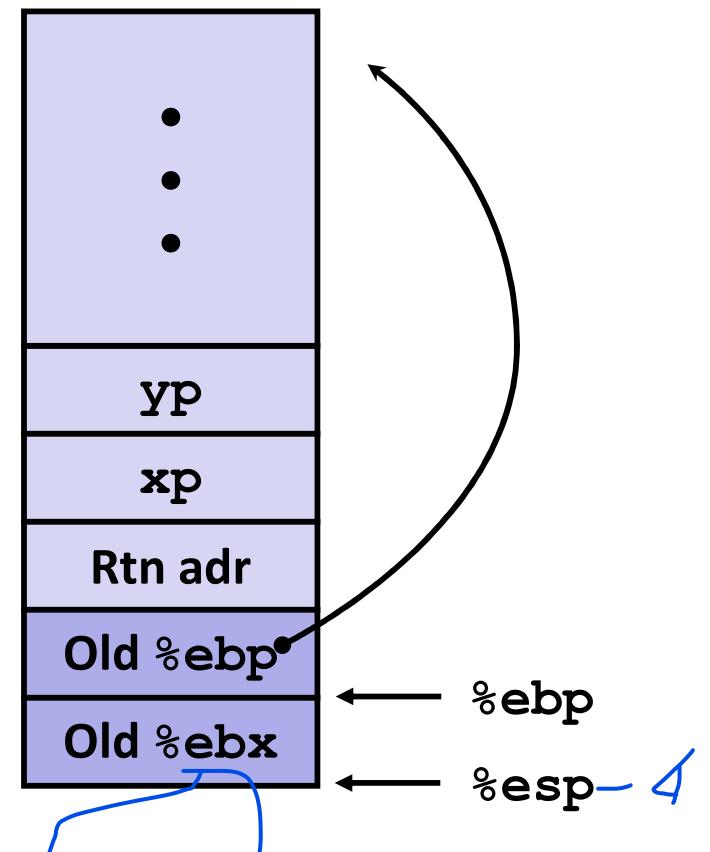
Entering Stack



swap:

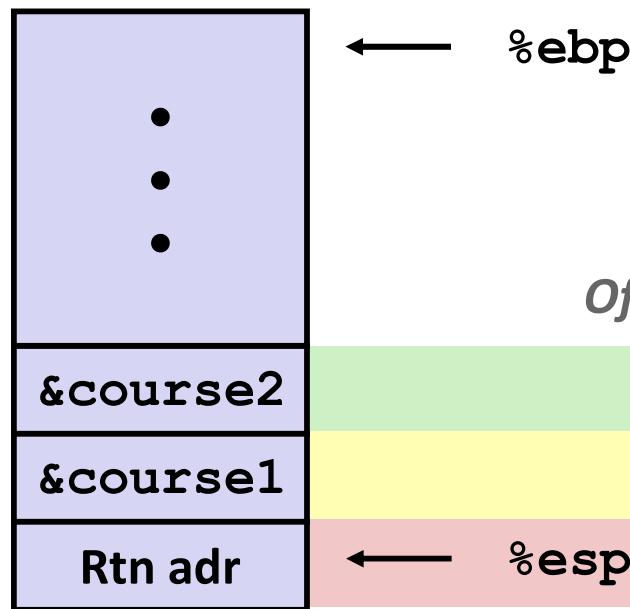
```
pushl %ebp  
movl %esp,%ebp  
pushl %ebx
```

Resulting Stack

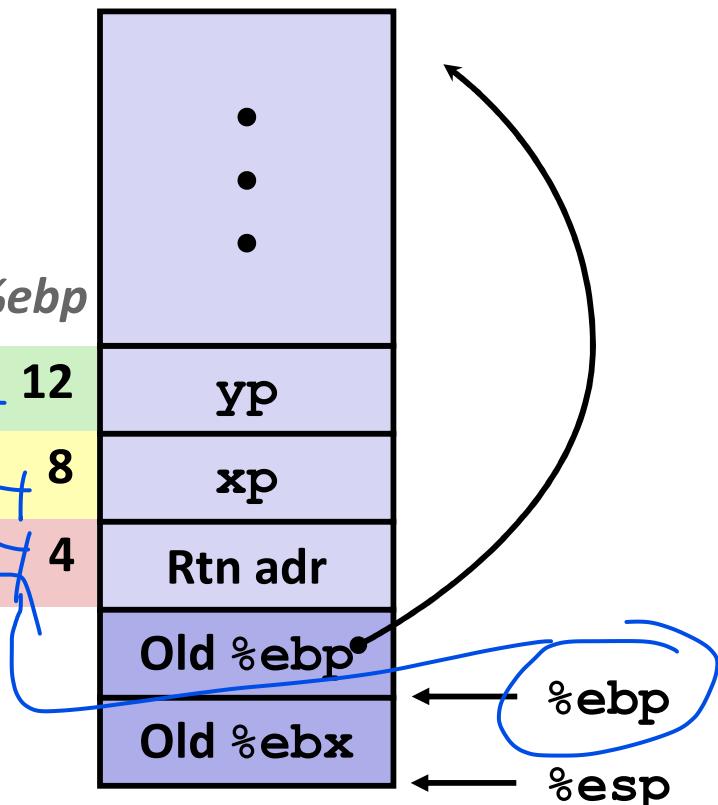


# swap: Lấy các tham số

Entering Stack



Resulting Stack



```
movl 8(%ebp),%edx    # get xp  
movl 12(%ebp),%ecx    # get yp  
. . .
```

# Giá trị trả về từ hàm

## ■ Hàm có trả về giá trị

- Trong C: qua lệnh **return x**.

```
int Q(int i)
{
    int t = 3*i;
    int v[10];
    .
    .
    return v[t];
}
```

## ■ Giá trị trả về của hàm trong assembly

- Thường lưu trong thanh ghi %eax

```
1 int func(int x, int y)
2 {
3     return x + y;
4 }
```



1	func:
2	pushl %ebp
3	movl %esp, %ebp
4	movl 8(%ebp), %edx
5	movl 12(%ebp), %eax
6	addl %edx, %eax
7	popl %ebp
8	ret

# Giá trị trả về từ hàm – Ví dụ

```
int main()
{
    int result = func(5,6);
    return result;
}
```

main:

```
pushl %ebp
movl %esp, %ebp
subl $16, %esp
pushl $6
pushl $5
call func
addl $8, %esp
movl %eax, -4(%ebp)
movl -4(%ebp), %eax
leave
ret
```

```
int func(int x, int y)
{
    int sum = 0;
    sum = x + y;
    return sum;
}
```

func:

```
pushl %ebp
movl %esp, %ebp
subl $16, %esp
movl $0, -4(%ebp)
movl 8(%ebp), %edx
movl 12(%ebp), %eax
addl %edx, %eax
movl %eax, -4(%ebp)
movl -4(%ebp), %eax
leave
ret
```

# Nội dung

---

- **Thủ tục (Procedures)**
  - Cấu trúc stack
  - Gọi hàm trong IA32
    - Chuyển luồng
    - Truyền dữ liệu
    - Quản lý dữ liệu cục bộ
  - Gọi hàm trong x86-64
  - Minh họa hàm đệ quy
- Bài tập về hàm
- Dịch ngược – Reverse engineering

# Sử dụng thanh ghi cho trong hàm

- Giả sử **yoo** là hàm mẹ, gọi hàm **who**
- Có thể dùng thanh ghi để lưu trữ tạm?

**yoo:**

```
• • •  
    movl $15213, %edx  
    call who  
    addl %edx, %eax  
• • •  
    ret
```

**who:**

```
• • •  
    movl 8(%ebp), %edx  
    addl $18243, %edx  
• • •  
    ret
```

- Giá trị của thanh ghi **%edx** bị ghi đè trong hàm **who**
- Có thể gây ra vấn đề → cần lưu lại!

# Quy ước lưu các thanh ghi

## ■ Giả sử yoo gọi who:

- yoo là hàm mẹ (caller)
- who là hàm con (callee)

## ■ Quy ước

- “Caller Save”
  - Hàm mẹ lưu lại các giá trị tạm thời trong stack frame của nó trước khi gọi hàm con
- “Callee Save”
  - Hàm con lưu lại các giá trị tạm thời trong stack của nó trước khi sử dụng

# Sử dụng các thanh ghi IA32/Linux + Windows

## ■ **%eax, %edx, %ecx**

- Hàm mẹ lưu trước khi gọi nếu giá trị sẽ được sử dụng tiếp

## ■ **%eax**

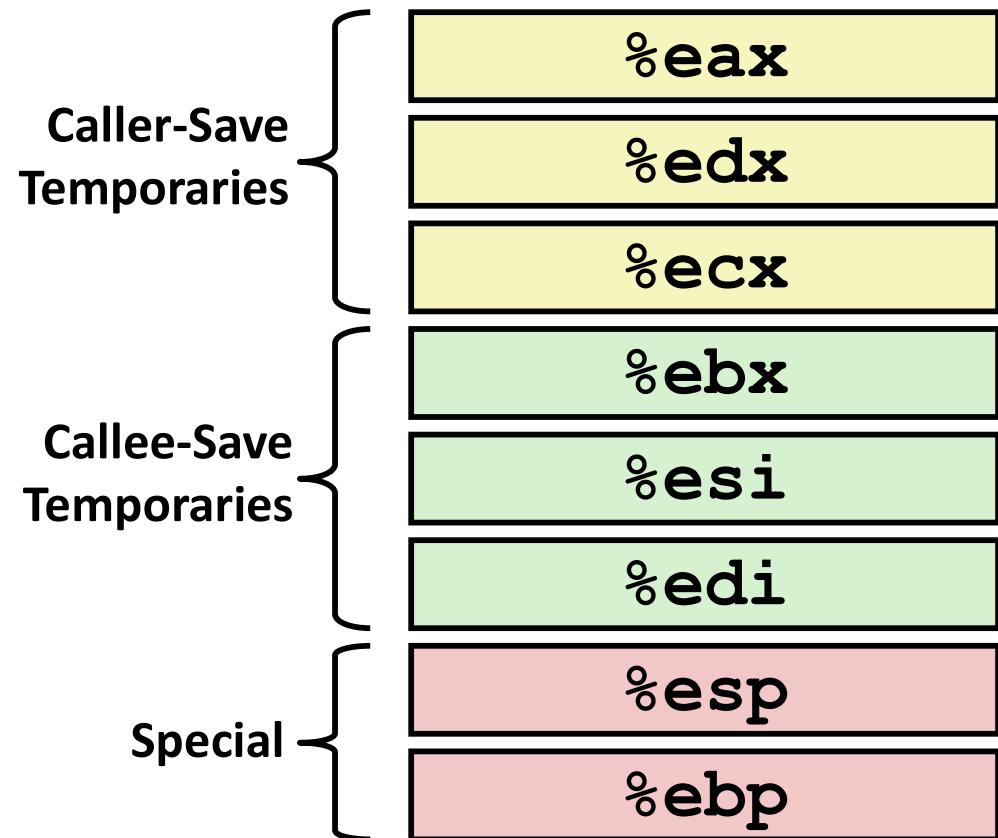
- được sử dụng để trả về giá trị số nguyên

## ■ **%ebx, %esi, %edi**

- Hàm con sẽ lưu nếu muốn sử dụng

## ■ **%esp, %ebp**

- Trường hợp đặc biệt cần hàm con lưu
- Khôi phục lại giá trị ban đầu trước khi thoát hàm



# Khởi tạo biến cục bộ: Ví dụ

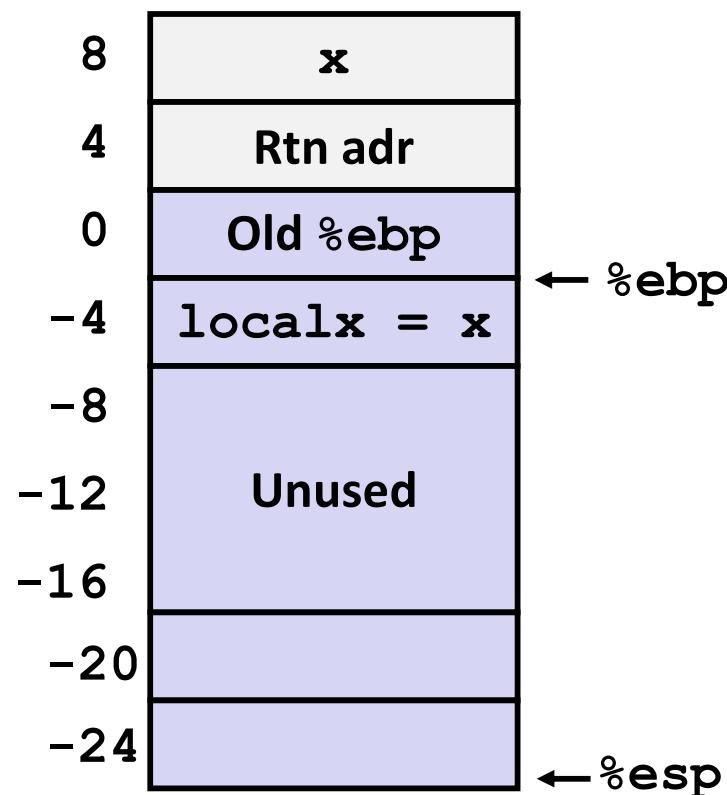
## ■ Biến cục bộ

- Cấp phát vùng nhớ trong stack để lưu các biến cục bộ của hàm
- Truy xuất dựa trên %ebp
  - Địa chỉ thấp hơn so với %ebp

```
int add3(int x) {  
    int localx = x;  
    incr(&localx, 3);  
    return localx;  
}
```

First part of add3

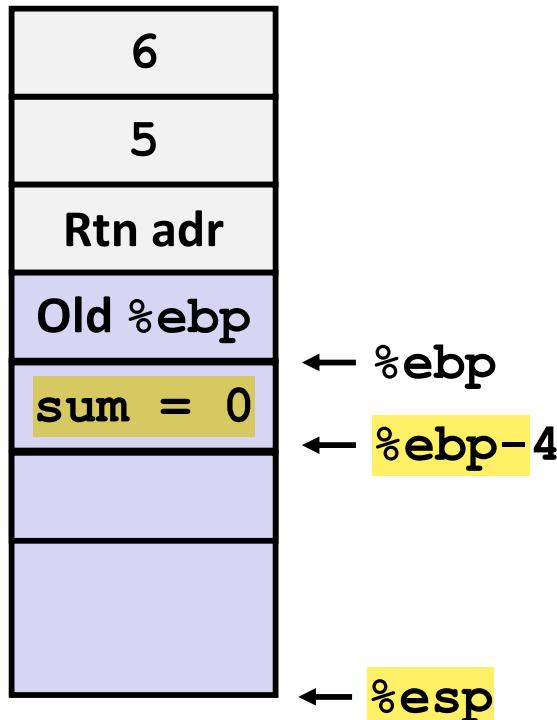
```
add3:  
    pushl %ebp  
    movl %esp, %ebp  
    subl $24, %esp      # Alloc. 24 bytes  
    movl 8(%ebp), %eax  
    movl %eax, -4(%ebp) # Set localx to x
```



# Biến cục bộ – Ví dụ

```
int main()
{
    int result = func(5,6);
    return result;
}
```

```
int func(int x, int y)
{
    int sum = 0;
    sum = x + y;
    return sum;
}
```



func:

pushl	%ebp
movl	%esp, %ebp
subl	\$16, %esp
movl	\$0, -4(%ebp)
movl	8(%ebp), %edx
movl	12(%ebp), %eax
addl	%edx, %eax
movl	%eax, -4(%ebp)
movl	-4(%ebp), %eax
leave	
ret	

# Gọi hàm (IA32): Tổng kết

## ■ Stack đóng vai trò quan trọng trong gọi/trả về hàm

- Lưu trữ địa chỉ trả về
- Các tham số (trong stack frame hàm mẹ)
- Có thể lưu các giá trị trong stack frame hoặc các thanh ghi
- Giá trị trả về ở thanh ghi %eax

# Bài tập gọi hàm 1

main:

```
pushl %ebp  
movl %esp, %ebp  
subl $16, %esp  
movl $1, -4(%ebp)  
movl $2, -8(%ebp)  
movl $0, -12(%ebp)  
pushl -4(%ebp)  
pushl -8(%ebp)  
call function  
addl $8, %esp  
movl %eax, -12(%ebp)  
movl $0, %eax  
leave  
ret
```

1. Hàm nào là caller/callee?

main caller  
func callee

2. Mỗi hàm có bao nhiêu biến cục bộ? Giá trị như thế nào?

caller: 1,2,0  
callee: a =, res -=

function:

```
pushl %ebp  
movl %esp, %ebp  
subl $16, %esp  
movl $10, -4(%ebp) # a local  
movl -4(%ebp), %edx  
movl 8(%ebp), %eax # x  
addl %eax, %edx  
movl 12(%ebp), %eax # y  
imull %edx, %eax  
movl %eax, -8(%ebp)  
movl -8(%ebp), %eax # result  
leave  
ret
```

3. Hàm function nhận bao nhiêu tham số?

2: 1,2

4. Hàm main đã truyền các tham số có giá trị cho function?

2: 1,2

5. Hàm function làm gì? Với các giá trị tham số đã tìm thấy ở Câu 4, tìm giá trị được function trả về cho main?