Programación a Nivel-Máquina III: Sentencias switch y Procedimientos IA32

Estructura de Computadores
Semana 4

Bibliografía:

[BRY11] Cap.3 Computer Systems: A Programmer's Perspective. Bryant, O'Hallaron. Pearson, 2011
Signatura ESIIT/C.1 BRY com

Transparencias del libro CS:APP, Cap.3
Introduction to Computer Systems: a Programmer's Perspective
Autores: Randal E. Bryant y David R. O'Hallaron

Guía de trabajo autónomo (4h/s)

- **Lectura:** del Cap.3 CS:APP (Bryant/O'Hallaron)
  - Switch Statements, Procedures.
    - 3.6.7 - 3.7 pp.247-266 (3.7.1 – 3.7.2 sugeridos previamente)
- **Ejercicios:** del Cap.3 CS:APP (Bryant/O'Hallaron)
  - Probl. 3.28 - 3.34 pp.251-52, 257-58, 262, 265-66

Bibliografía:

[BRY11] Cap.3 Computer Systems: A Programmer's Perspective. Bryant, O'Hallaron. Pearson, 2011
Signatura ESIIT/C.1 BRY com

Progr. Máquina III: switch/Procedimientos

- **Sentencias switch**
- **Procedimientos IA 32**
  - Estructura de la Pila
  - Convenciones de Llamada
  - Ejemplos ilustrativos de Recursividad & Punteros


```
long switch_eg
   (long x, long y, long z)
{
    long w = 1;
    switch(x) {
    case 1:
        w = y*z;
        break;
    case 2:
        w = y/z;
        /* Fall Through */
    case 3:
        w += z;
        break;
    case 5:
    case 6:
        w -= z;
        break;
    default:
        w = 2;
    }
    return w;
}
```

Ejemplo de sentencia switch

- **Etiquetas de caso múltiples**
  - Aquí: 5 & 6
- **Caídas en cascada***
  - Aquí: 2
- **Casos ausentes***
  - Aquí: 4

** "fall through cases", "missing cases", en inglés*

Estructura de una Tabla de Saltos

Forma switch

```
switch(x) {
  case val_0:
    Block 0
  case val_1:
    Block 1
    • • •
  case val_n-1:
    Block n–1
}
```

Tabla Saltos*

JTab:

Targ0
Targ1
Targ2
• • •
$Targ_{n-1}$

Destinos salto*

Targ0: **Code Block 0**

Targ1: **Code Block 1**

Targ2: **Code Block 2**

•
•
•

$Targ_{n-1}$: **Code Block *n*–1**

Traducción Aproximada

```
target = JTab[x]; *
goto *target;
```

* "jump table", "jump targets", en inglés

Ejemplo de Sentencia Switch (IA32)

```
long switch_eg(long x, long y, long z)
{
    long w = 1;
    switch(x) {
      . . .
    }
    return w;
}
```

¿Qué rango de valores cubre default?

Inicialización*:

```
  switch_eg:
     pushl  %ebp            # Setup
     movl   %esp, %ebp      # Setup
     movl   8(%ebp), %eax   # %eax = x
     cmpl   $6, %eax        # Compare x:6
     ja     .L2             # If unsigned > goto default
     jmp    *.L7(,%eax,4)   # Goto *JTab[x]
```

Notar que **w** no se inicializa aquí

* "setup" = ajuste inicial, "unsigned >" = "above" = mayor sin signo

Ejemplo de Sentencia Switch (IA32)

```
long switch_eg(long x, long y, long z)
{
    long w = 1;
    switch(x) {
      . . .
    }
    return w;
}
```

Tabla de saltos

```
.section    .rodata
  .align 4
.L7:
  .long     .L2 # x = 0
  .long     .L3 # x = 1
  .long     .L4 # x = 2
  .long     .L5 # x = 3
  .long     .L2 # x = 4
  .long     .L6 # x = 5
  .long     .L6 # x = 6
```

Inicialización:

```
switch_eg:
   pushl  %ebp           # Setup
   movl   %esp, %ebp     # Setup
   movl   8(%ebp), %eax  # eax = x
   cmpl   $6, %eax       # Compare x:6
   ja     .L2            # If unsigned > goto default
   jmp    *.L7(,%eax,4)  # Goto *JTab[x]
```

Salto indirecto

** "Ln" = "Label n" = etiqueta (de salto) n* 7

Explicación Inicialización Ensamblador

- **Estructura de la Tabla**
  - Cada destino salto requiere 4 bytes
  - Dirección base es .L7

Tabla de saltos

```
.section   .rodata
  .align 4
.L7:
  .long    .L2 # x = 0
  .long    .L3 # x = 1
  .long    .L4 # x = 2
  .long    .L5 # x = 3
  .long    .L2 # x = 4
  .long    .L6 # x = 5
  .long    .L6 # x = 6
```

- **Saltos**
  - **Directo:** `jmp .L2`
  - Destino salto indicado por etiqueta .L2
  - **Indirecto:** `jmp *.L7(,%eax,4)`
  - Inicio de la tabla de saltos: .L7
  - Se debe escalar por factor 4 (etiquetas tienen 32-bits = 4 Bytes en IA32)
  - Captar destino (dir. salto) desde la Dirección Efectiva .L7 + eax*4
    - Sólo para $0 \le x \le 6$

8

Tabla de Saltos

Tabla de saltos

```
.section   .rodata
  .align 4
.L7:
  .long    .L2 # x = 0
  .long    .L3 # x = 1
  .long    .L4 # x = 2
  .long    .L5 # x = 3
  .long    .L2 # x = 4
  .long    .L6 # x = 5
  .long    .L6 # x = 6
```

```
switch(x) {
case 1:      // .L3
    w = y*z;
    break;
case 2:      // .L4
    w = y/z;
    /* Fall Through */
case 3:      // .L5
    w += z;
    break;
case 5:
case 6:      // .L6
    w -= z;
    break;
default:     // .L2
    w = 2;
}
```


Tratamiento de Caídas en Cascada

```
long w = 1;
   . . .
switch(x) {
   . . .
case 2:
    w = y/z;
    /* Fall Through */
case 3:
    w += z;
    break;
   . . .
}
```

```
case 3:
        w = 1;
        goto merge;
```

```
    case 2:
        w = y/z;
merge:
        w += z;
```

Bloques de Código (Parcial)

```
switch(x) {
case 1:       // .L3
      w = y*z;
      break;
 . . .
case 3:       // .L5
    w += z;
    break;
  . . .
default:      // .L2
    w = 2;
}
```

```
.L2:                # Default
  movl  $2, %eax    # w = 2
  jmp   .L8         # Goto done

.L5:                # x == 3
  movl  $1, %eax    # w = 1
  jmp   .L9         # Goto merge

.L3:                      # x == 1
  movl  16(%ebp), %eax    # z
  imull 12(%ebp), %eax    # w = y*z
  jmp   .L8               # Goto done
```


Bloques de Código (Resto)

```
switch(x) {
  . . .
  case 2:  // .L4
      w = y/z;
      /* Fall Through */
  merge:    // .L9
      w += z;
      break;
  case 5:
  case 6:  // .L6
      w -= z;
      break;
}
```

```
.L4:                      # x == 2
  movl  12(%ebp), %edx
  movl  %edx, %eax
  sarl  $31, %edx
  idivl 16(%ebp)          # w = y/z

.L9:                      # merge:
  addl  16(%ebp), %eax    # w += z
  jmp   .L8               # goto done

.L6:                      # x == 5, 6
  movl  $1, %eax          # w = 1
  subl  16(%ebp), %eax    # w = 1-z
```

Código switch (Final)

```
return w;
```

```
.L8:                    # done:
  popl %ebp
  ret
```

- **Características notables**
  - La tabla de saltos ahorra secuenciar a través de los casos
    - Tiempo constante, en vez de lineal
  - Usar la tabla de saltos para gestionar huecos y etiquetas múltiples*
  - Usar secuenciación del programa para gestionar caída en cascada
  - No inicializar w = 1 a menos de que realmente se necesite

* ver pág.4

Implementación switch x86-64

- **Misma idea general, adaptada a código 64-bit**
- **Entradas tabla de 64 bits (punteros)**
- **Casos usan código actualiz. x86-64**

```
    switch(x) {
    case 1:      // .L3
        w = y*z;
        break;
      . . .
    }
```

```
.L3:
  movq    %rdx, %rax
  imulq   %rsi, %rax
  ret
```

Tabla Saltos

```
  .section .rodata
  .align 8
.L7:
  .quad     .L2     # x = 0
  .quad     .L3     # x = 1
  .quad     .L4     # x = 2
  .quad     .L5     # x = 3
  .quad     .L2     # x = 4
  .quad     .L6     # X = 5
  .quad     .L6     # x = 6
```

Código Objeto IA32

- **Inicialización**
  - Etiqueta .L2 se convierte en dirección 0x8048422
  - Etiqueta .L7 se convierte en dirección 0x8048660

Código Ensamblador

```
switch_eg:
  . . .
   ja     .L2           # If unsigned > goto default
   jmp    *.L7(,%eax,4) # Goto *JTab[x]
```

Código Objeto Desensamblado

```
08048410 <switch_eg>:
 . . .
 8048419:77 07                 ja     8048422 <switch_eg+0x12>
 804841b:ff 24 85 60 86 04 08  jmp    *0x8048660(,%eax,4)
```


Código Objeto IA32 (cont.)

- **Tabla de Saltos**
  - No aparece en código desensamblado
  - Se puede inspeccionar usando GDB
  - gdb switch
  - (gdb) x/7xw 0x8048660
    - Examinar 7 (formato) hexadecimal (palabras) "words" (4-bytes cada)
    - Usar el comando "help x" para obtener documentación de formatos

```
0x8048660:    0x08048422    0x08048432    0x0804843b    0x08048429
0x8048670:    0x08048422    0x0804844b    0x0804844b
```

Código Objeto IA32 (cont.)

- Descifrando la Tabla de Saltos

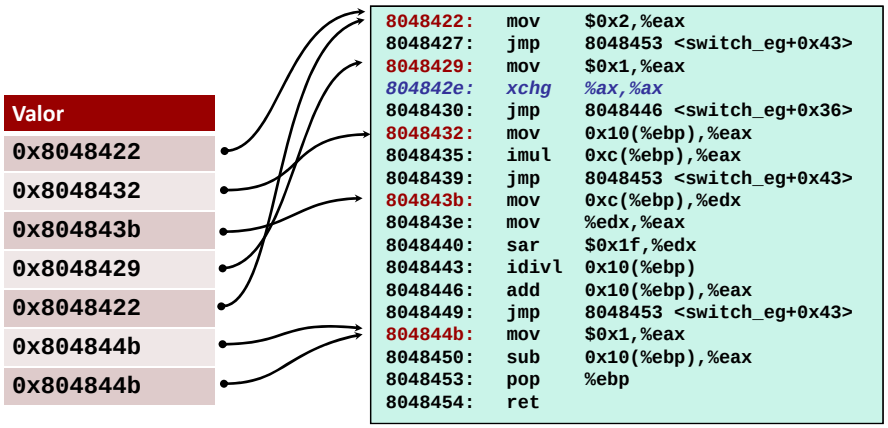
```
0x8048660:    0x08048422    0x08048432    0x0804843b    0x08048429
0x8048670:    0x08048422    0x0804844b    0x0804844b
```

Dirección	Valor	x
0x8048660	0x8048422	0
0x8048664	0x8048432	1
0x8048668	0x804843b	2
0x804866c	0x8048429	3
0x8048670	0x8048422	4
0x8048674	0x804844b	5
0x8048678	0x804844b	6

Destinos desensamblados

```
8048422:   b8 02 00 00 00       mov    $0x2,%eax
8048427:   eb 2a                jmp    8048453 <switch_eg+0x43>
8048429:   b8 01 00 00 00       mov    $0x1,%eax
804842e:   66 90                xchg   %ax,%ax # noop
8048430:   eb 14                jmp    8048446 <switch_eg+0x36>
8048432:   8b 45 10             mov    0x10(%ebp),%eax
8048435:   0f af 45 0c          imul   0xc(%ebp),%eax
8048439:   eb 18                jmp    8048453 <switch_eg+0x43>
804843b:   8b 55 0c             mov    0xc(%ebp),%edx
804843e:   89 d0                mov    %edx,%eax
8048440:   c1 fa 1f             sar    $0x1f,%edx
8048443:   f7 7d 10             idivl  0x10(%ebp)
8048446:   03 45 10             add    0x10(%ebp),%eax
8048449:   eb 08                jmp    8048453 <switch_eg+0x43>
804844b:   b8 01 00 00 00       mov    $0x1,%eax
8048450:   2b 45 10             sub    0x10(%ebp),%eax
8048453:   5d                   pop    %ebp
8048454:   c3                   ret
```

Emparejando destinos desensamblados




```
long switch_eg
   (long x, long y, long z)
{
    long w = 1;
    switch(x) {
    case 1:
        w = y*z;
        break;
    case 2:
        w = y/z;
        /* Fall Through */
    case 3:
        w += z;
        break;
    case 5:
    case 6:
        w -= z;
        break;
    default:
        w = 2;
    }
    return w;
}
```

Resumen (Control)

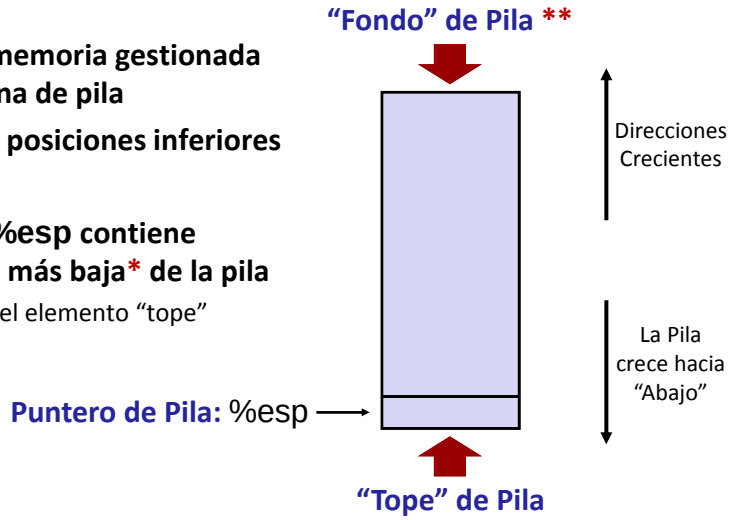
- **Control C**
  - if-then-else
  - do-while
  - while, for
  - switch
- **Control Ensamblador**
  - Salto condicional
  - Movimiento condicional
  - Salto indirecto
  - Compilador genera secuencia código p/implementar control más complejo
- **Técnicas estándar**
  - Bucles convertidos a forma do-while
  - Sentencias switch grandes usan tablas de saltos
  - Sentencias switch poco densas podrían usar árboles decisión

Progr. Máquina III: switch/Procedimientos

- Sentencias switch
- **Procedimientos IA 32**
  - Estructura de la Pila
  - Convenciones de Llamada
  - Ejemplos ilustrativos de Recursividad & Punteros

Pila IA32

- **Región de memoria gestionada con disciplina de pila**
- **Crece hacia posiciones inferiores**
- **El registro %esp contiene la dirección más baja* de la pila**
  - dirección del elemento "tope"

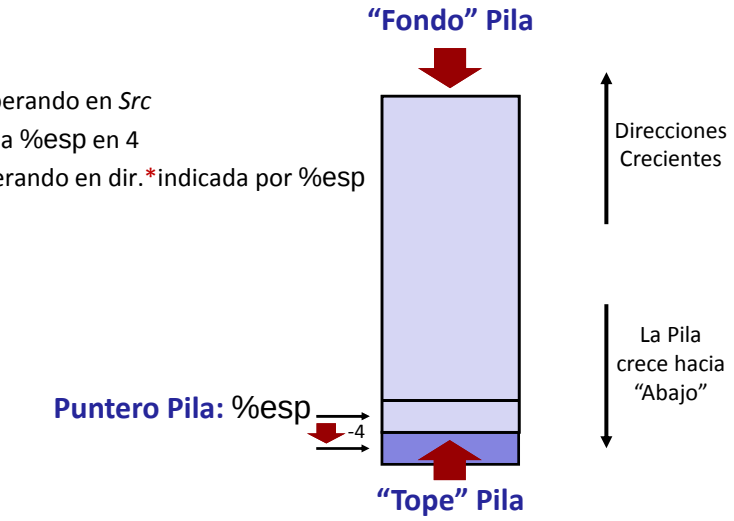


** por el momento, en el instante actual, y es la dirección del byte más bajo de los 4B que ocupa el elemento tope*

*** "top" = tope, "bottom" = fondo* 23

Pila IA32: Push

- **pushl *Src***
  - Capta el operando en *Src*
  - Decrementa %esp en 4
  - Escribe operando en dir.*indicada por %esp

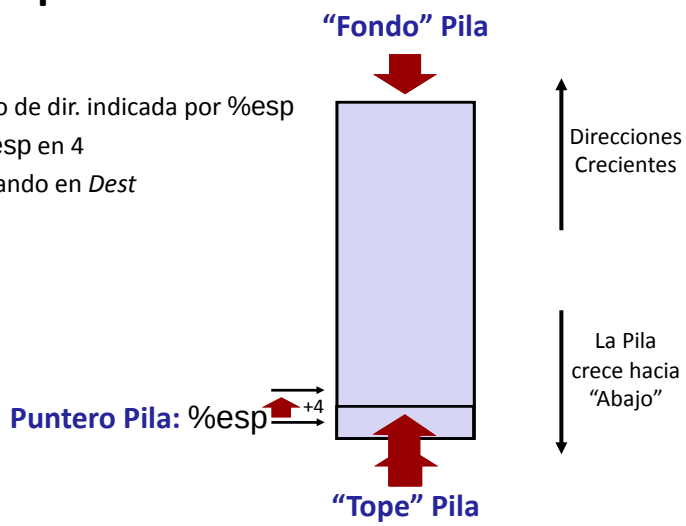


** ocupando por tanto ese byte (el más bajo) y los 3 inmediatamente encima*

*** "push" = empujar* 24

Pila IA32 : Pop

- **popl *Dest***
  - Capta operando de dir. indicada por %esp
  - Incrementa %esp en 4
  - Escribe el operando en *Dest*



** "pop" = saltar (con pequeño ruido explosivo), salir* 25

Flujo de Control para Procedimientos

- **Se usa pila para soportar llamadas y retornos de procedimientos**
- **Llamada a procedimiento: `call label`**
  - Recuerda* la dirección de retorno en la pila
  - Salta a etiqueta ***label***
- **Dirección de retorno:**
  - Dirección de la siguiente instrucción justo después de la llamada (call)
  - Ejemplo de un desensamblado

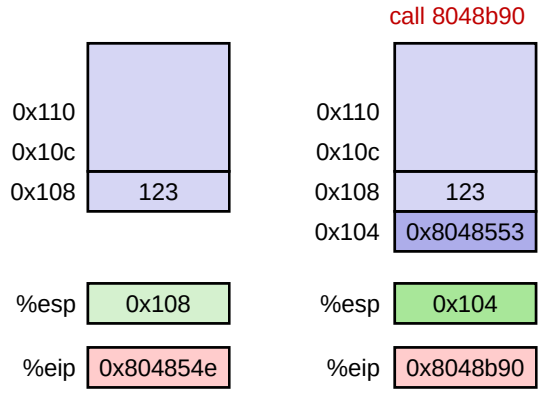
```
804854e:  e8 3d 06 00 00    call   8048b90 <main>
8048553:  50                pushl  %eax
```

  - Dirección de retorno = 0x8048553
- **Retorno de procedimiento: `ret`**
  - Recupera* la dirección (de retorno) de la pila
  - Salta a dicha dirección

** usando "push/pop"* 26

Ejemplo de Llamada a Procedimiento

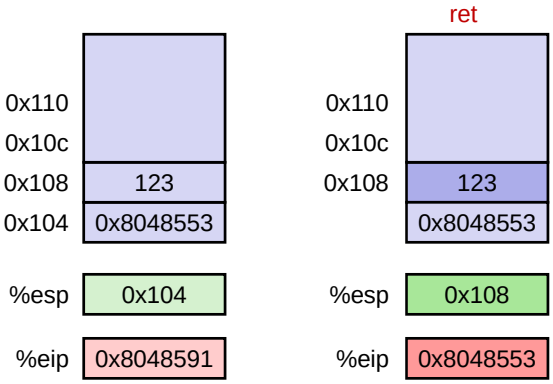
```
804854e:    e8 3d 06 00 00      call   8048b90 <main>
8048553:    50                  pushl  %eax
```



%eip: ***contador de programa***

Ejemplo de Retorno de Procedimiento

```
8048591:    c3                  ret
```



%eip: ***contador de programa***

** notar que dir.ret. sigue escrita en memoria, pero más allá del tope, así que ya no se considera que esté en la pila (que sea elemento válido pila)*

Lenguajes basados en pila*

- **Lenguajes que soportan recursividad**
  - P.ej., C, Pascal, Java
  - El código debe ser "*Reentrante*"
    - Múltiples instanciaciones** simultáneas de un mismo procedimiento
  - Se necesita algún lugar para guardar el estado de cada instanciación
    - Argumentos
    - Variables locales
    - Puntero (dirección) de retorno
- **Disciplina de pila**
  - Estado para un procedimiento dado, necesario por tiempo limitado
    - Desde que se le llama hasta que retorna
  - El invocado† retorna antes de que lo haga el invocante†
- **La pila se reserva en *Marcos††***
  - estado para una sola instanciación procedimiento

† "callee/caller" en inglés
†† "allocated in frames" en inglés
** "block structured" en terminología Intel*
*** "instantiate" = crear nuevos ejemplares*

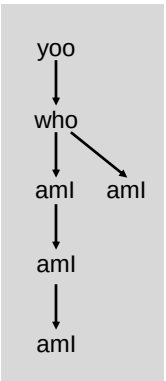
Ejemplo de secuencia de llamadas

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

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

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

Ejemplo Sec. Llamadas

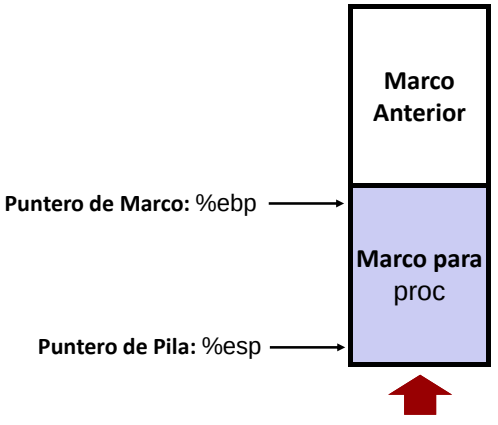


El procedimiento aml() **es recursivo**

** "call chain" en inglés*
"yoo" suena a "you" = tú, "who" = quién, "am I" = soy yo

Marcos de Pila

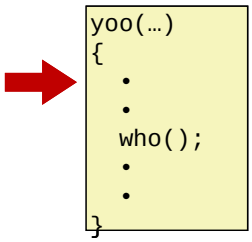
- **Contenido**
  - Variables locales
  - Información para retorno
  - Espacio temporal

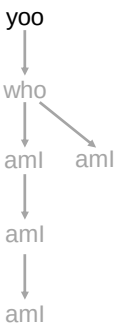


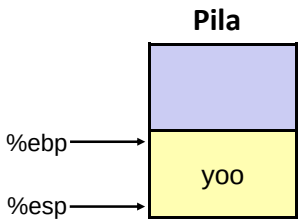
- **Gestión**
  - Espacio se reserva al entrar el procedimiento
    - Código de "Inicialización"*
  - Se libera al retornar
    - Código de "Finalización" *

* "ajuste inicial/fin" en desensamblados y dibujos pila anteriores

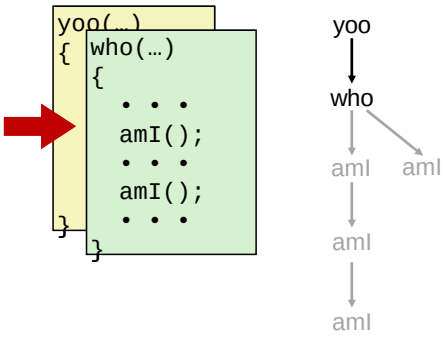
Ejemplo

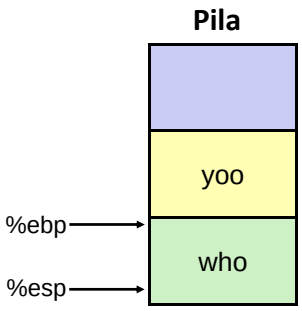




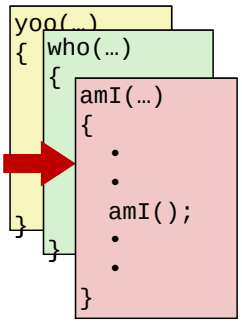


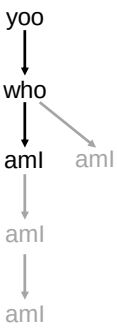
Ejemplo

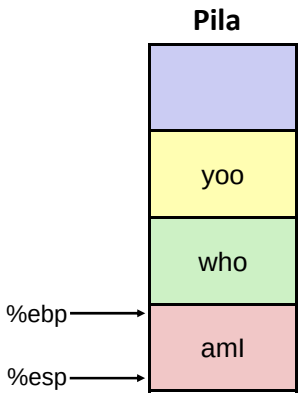




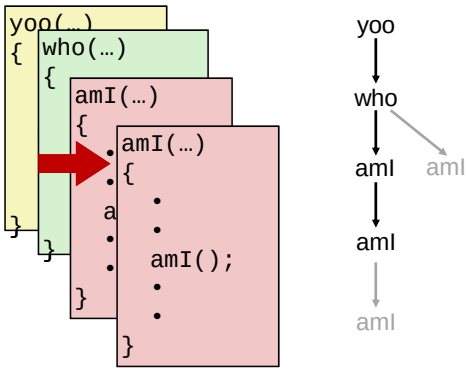
Ejemplo

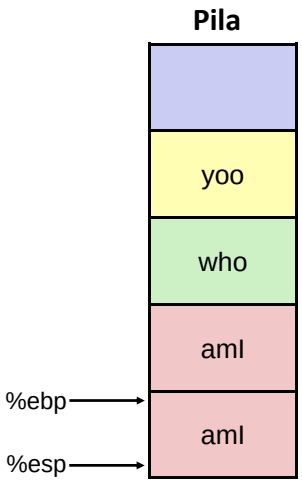




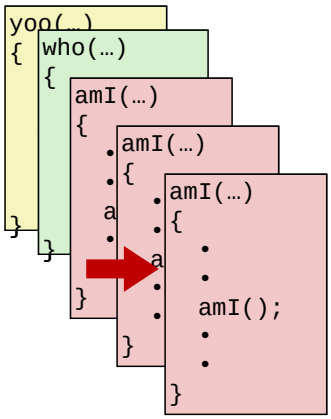


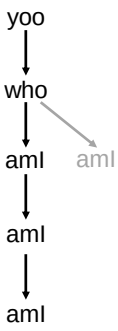
Ejemplo

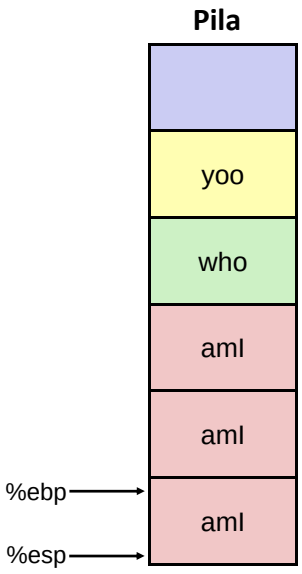




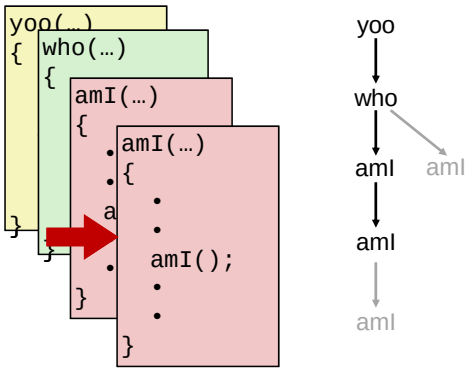
Ejemplo

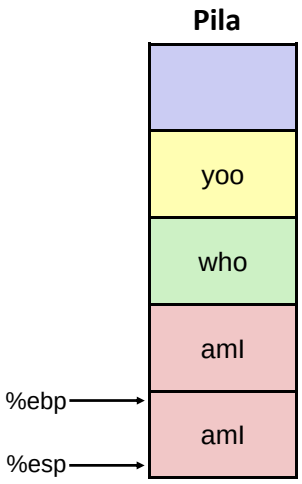




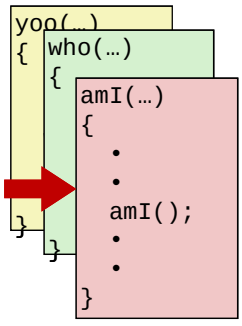


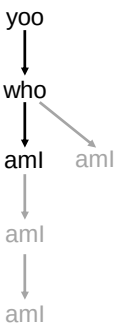
Ejemplo

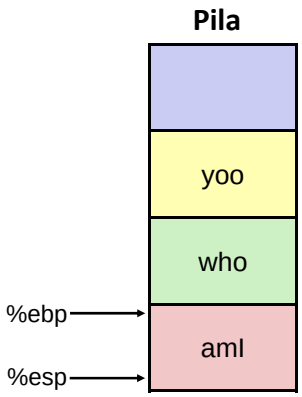




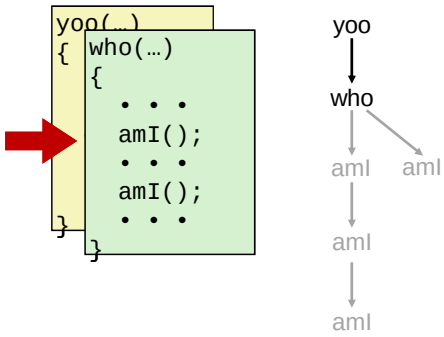
Ejemplo

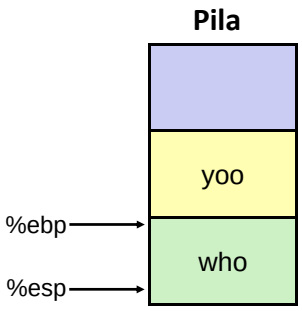




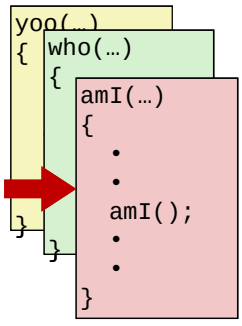


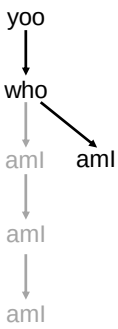
Ejemplo

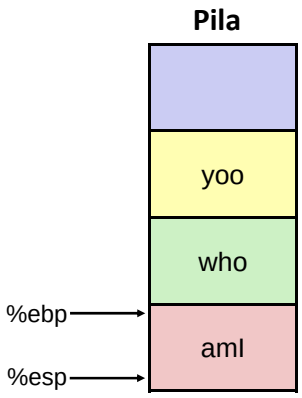




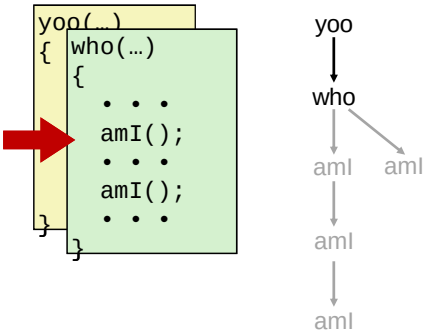
Ejemplo

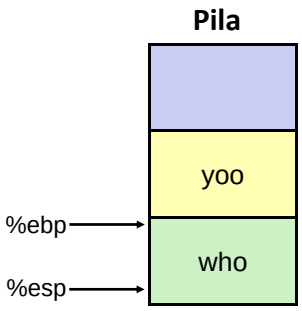




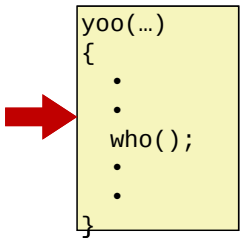


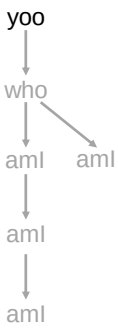
Ejemplo

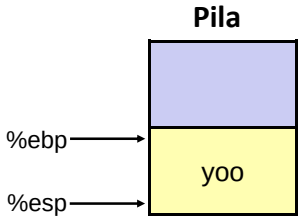




Ejemplo

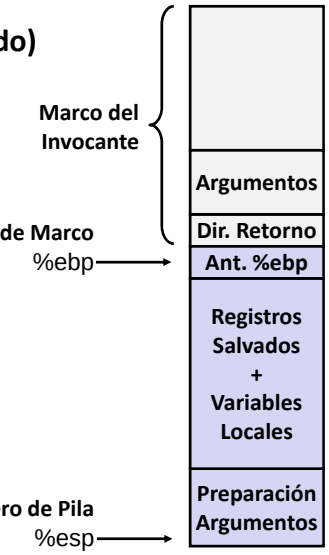






Marco de Pila IA32/Linux

- **Marco de Pila Actual (de "Tope" a Fondo)**
  - "Preparación de argumentos": Parámetros para función a punto llamar
  - Variables locales Si no se pueden mantener en registros*
  - Contexto (de registros) salvado
  - Antiguo puntero de marco
- **Marco de Pila del Invocante**
  - Dirección de retorno
    - Salvada por la instrucción call
  - Argumentos para esta llamada (invocación)
  - …



* Si el compilador no puede resolver la compilacion del procedimiento usando sólo registros

Volviendo a ver swap

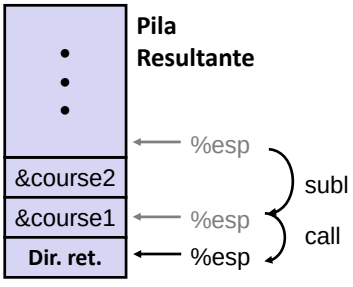
```
int course1 = 15213;
int course2 = 18243;

void call_swap() {
  swap(&course1, &course2);
}
```

Llamando a swap **desde** 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;
}
```



Volviendo a ver 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
```

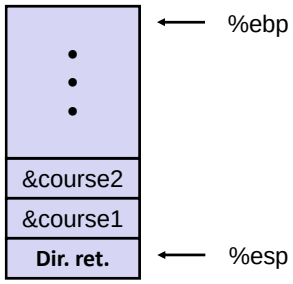
Ajuste Inicial

Cuerpo

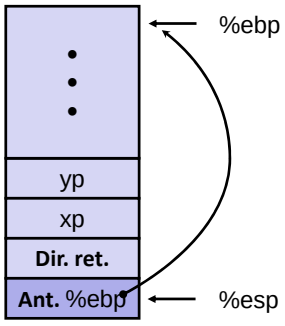
Fin

swap Ajuste #1

Pila a la Entrada



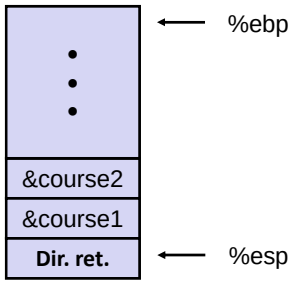
Pila Resultante



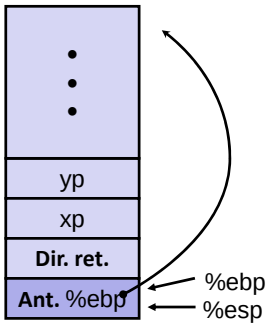

```
swap:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
```

swap Ajuste #2

Pila a la Entrada



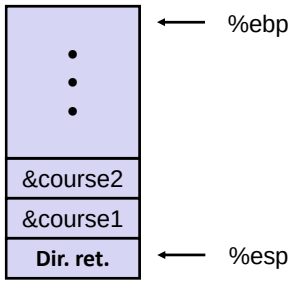
Pila Resultante



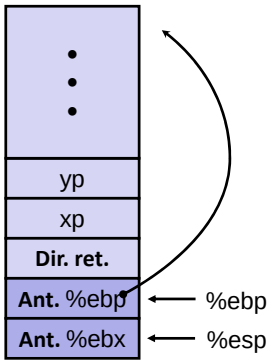

```
swap:
    pushl %ebp
    movl %esp,%ebp
    pushl %ebx
```


swap Ajuste #3

Pila a la Entrada



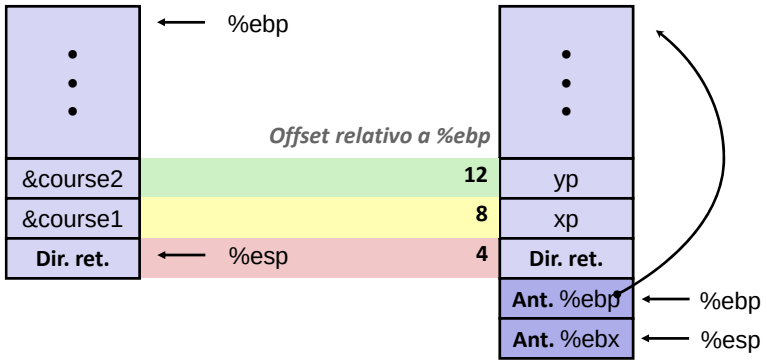
Pila Resultante




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

swap Cuerpo

Pila a la Entrada

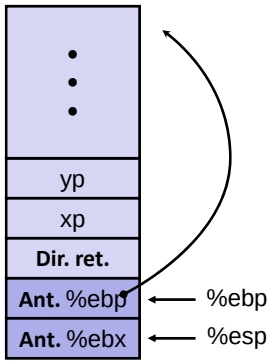


Pila Resultante

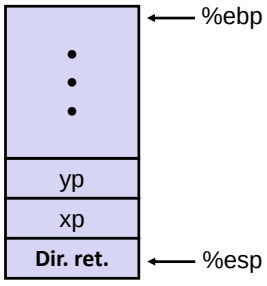

```
movl 8(%ebp),%edx   # obtener xp
movl 12(%ebp),%ecx  # obtener yp
. . .
```


swap Fin

Pila antes de Fin



Pila Resultante



- **Observación**
  - Se salva y recupera registro %ebx
  - No sucede así con %eax, %ecx, %edx

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

Desensamblado de swap

```
08048384 <swap>:
 8048384:  55                    push   %ebp
 8048385:  89 e5                 mov    %esp,%ebp
 8048387:  53                    push   %ebx
 8048388:  8b 55 08              mov    0x8(%ebp),%edx
 804838b:  8b 4d 0c              mov    0xc(%ebp),%ecx
 804838e:  8b 1a                 mov    (%edx),%ebx
 8048390:  8b 01                 mov    (%ecx),%eax
 8048392:  89 02                 mov    %eax,(%edx)
 8048394:  89 19                 mov    %ebx,(%ecx)
 8048396:  5b                    pop    %ebx
 8048397:  5d                    pop    %ebp
 8048398:  c3                    ret
```

Código Llamada

```
 80483b4:  movl   $0x8049658,0x4(%esp) # Copiar &course2
 80483bc:  movl   $0x8049654,(%esp)    # Copiar &course1
 80483c3:  call   8048384 <swap>       # Llamar swap
 80483c8:  leave                       # Preparar retorno
 80483c9:  ret                         # Retornar
```

** LEAVE deshace ENTER previo: ESP←EBP, POP EBP.*

Progr. Máquina III: switch/Procedimientos

- Sentencias switch
- **Procedimientos IA 32**
  - Estructura de la Pila
  - Convenciones de Llamada
  - Ejemplos ilustrativos de Recursividad & Punteros

Convenciones de Preservación de Registros*

- **Cuando el procedimiento yoo llama a who:**
  - yoo es el ***que llama (invocante, llamante)***
  - who es el ***llamado (invocado)***
- **¿Se puede usar un registro para almacenamiento temporal?**

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

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

  - Contenidos del registro %edx sobrescritos por who
  - Podría causar problemas → ¡debería hacerse algo!
    - Necesita alguna coordinación

** "register saving conventions" en inglés* 53

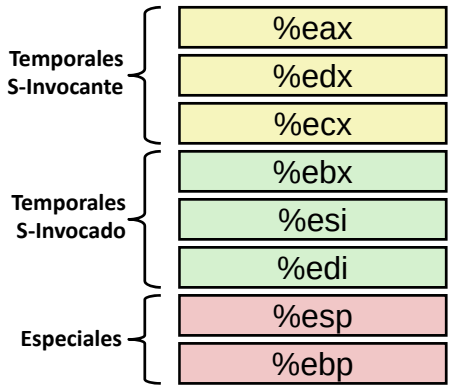
Convenciones de Preservación de Registros

- **Cuando el procedimiento yoo llama a who:**
  - yoo es el ***que llama (invocante, llamante)***
  - who es el ***llamado (invocado)***
- **¿Se puede usar un registro para almacenamiento temporal?**
- **Convenciones***
  - ***"Salva Invocante"***
    - El que llama salva valores temporales en su marco antes de la llamada
  - ***"Salva Invocado"***
    - El llamado salva valores temporales en su marco antes de reusar regs.

** "caller/callee save" en inglés* 54

Uso de Registros en IA32/Linux+Windows

- **%eax, %edx, %ecx**
  - Invocante salva antes de llamada si los valores se usan después
- **%eax**
  - tb. usado para devolver v. entero
- **%ebx, %esi, %edi**
  - Invocado salva si quiere usarlos
- **%esp, %ebp**
  - forma especial de invocado-salva
  - Restaurados a su valor original al salir del procedimiento



** Consultar Wikipedia: "X86 calling conventions", cdecl/stdcall* 55

Progr. Máquina III: switch/Procedimientos

- Sentencias switch
- **Procedimientos IA 32**
  - Estructura de la Pila
  - Convenciones de Llamada
  - Ejemplos ilustrativos de Recursividad & Punteros

56

Función Recursiva

```
/* Recursive popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

- **Registros**
  - %eax, %edx usados sin salvarlos primero
  - %ebx usado, pero salvado al principio & restaurado al final

```
pcount_r:
    pushl %ebp
    movl  %esp, %ebp
    pushl %ebx
    subl  $4, %esp
    movl  8(%ebp), %ebx
    movl  $0, %eax
    testl %ebx, %ebx
    je  .L3
    movl  %ebx, %eax
    shrl  %eax
    movl  %eax, (%esp)
    call  pcount_r
    movl  %ebx, %edx
    andl  $1, %edx
    leal  (%edx,%eax), %eax
.L3:
    addl  $4, %esp
    popl  %ebx
    popl  %ebp
    ret
```

** "population count", ver pág.38 lección anterior*

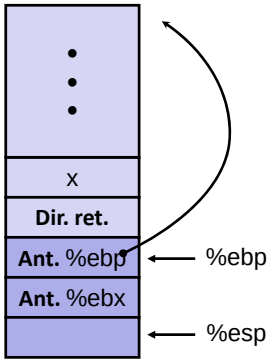
Función Recursiva #1

```
/* Recursive popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

- **Acciones**
  - Salvar en pila antiguo valor %ebx
  - Reservar espacio para argumento de llamada recursiva
  - Almacenar x en %ebx

%ebx | x

```
pcount_r:
    pushl %ebp
    movl  %esp, %ebp
    pushl %ebx
    subl  $4, %esp
    movl  8(%ebp), %ebx
     • • •
```



Función Recursiva #2

```
/* Recursive popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
     . . .
    movl  $0, %eax
    testl %ebx, %ebx
    je  .L3
     . . .
.L3:
     . . .
    ret
```

- **Acciones**
  - Si x == 0, retornar
    - con %eax puesto a 0

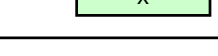
59

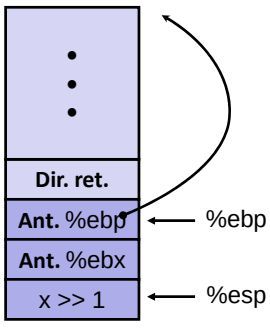
Función Recursiva #3

```
/* Recursive popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
     . . .
    movl  %ebx, %eax
    shrl  %eax
    movl  %eax, (%esp)
    call  pcount_r
     . . .
```

- **Acciones**
  - Almacenar (pasar) x >> 1 en pila
  - Hacer llamada recursiva
- **Efecto**
  - %eax puesto a resultado función
  - %ebx aún tiene valor de x





60

Función Recursiva #4

```
/* Recursive popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
  . . .
movl  %ebx, %edx
andl    $1, %edx
leal  (%edx,%eax), %eax
  . . .
```

- **Asumir**
  - %eax contiene valor de llam. recursiva
  - %ebx contiene x
- **Acciones**
  - Calcular (x & 1) + valor retornado
- **Efecto**
  - %eax puesto a resultado función

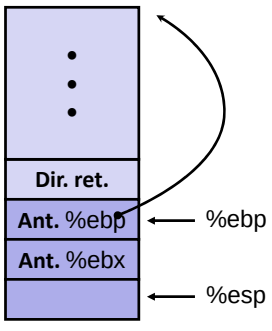
61

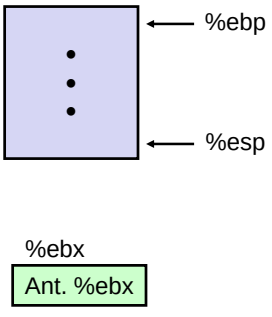
Función Recursiva #5

```
/* Recursive popcount */
int pcount_r(unsigned x) {
  if (x == 0)
    return 0;
  else return
    (x & 1) + pcount_r(x >> 1);
}
```

```
    . . .
L3:
    addl  $4, %esp
    popl  %ebx
    popl  %ebp
    ret
```

- **Acciones**
  - Restaurar valores de %ebx y %ebp
  - Restaurar %esp





62

Observaciones Sobre la Recursividad

- **Manejada sin Especiales Consideraciones**
  - Marcos pila implican que cada llamada a función tiene almcnmto. privado
    - Registros & variables locales salvadas
    - Dirección de retorno salvada
  - Convenciones preservación registros previenen que una llamada a función corrompa los datos de otra
  - Disciplina de pila sigue el patrón de llamadas / retornos
    - Si P llama a Q, entonces Q retorna antes que P
    - Primero en entrar, último en salir (Last-In, First-Out)*
- **También funciona con recursividad mutua****
  - P llama a Q; Q llama a P

** pila = lista LIFO*
*** funciona incluso con reentrancia*

Código para Punteros

Generando un Puntero

```
/* Compute x + 3 */
int add3(int x) {
  int localx = x;
  incrk(&localx, 3);
  return localx;
}
```

Referenciando un Puntero

```
/* Increment value by k */
void incrk(int *ip, int k) {
  *ip += k;
}
```

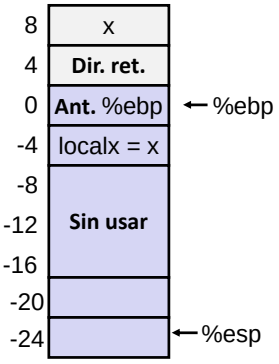
- **`add3` crea un puntero y se lo pasa a `incrk`**

** Los autores que dicen "references" (no "pointers"), llaman "dereferencing" (no "referencing") a la operación de seguir el puntero*

Creando e Inicializando Variable Local

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

- **La variable localx debe almacenarse en la pila**
  - Porque: Hay que crear puntero a ella
- **Calcular puntero como -4(%ebp)**



Parte inicial de add3

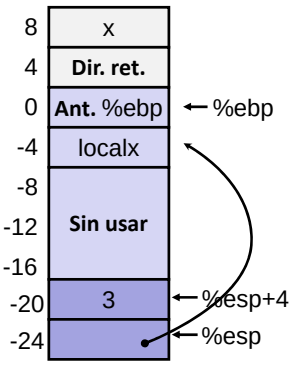
```
add3:
   pushl%ebp
   movl %esp, %ebp
   subl $24, %esp      # Alloc. 24 bytes
   movl 8(%ebp), %eax
   movl %eax, -4(%ebp)# Set localx to x
```

65

Creando Puntero para Argumento

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

- **Usar instrucción leal para calcular dirección de localx**



Parte intermedia de add3

```
   movl $3, 4(%esp)    # 2nd arg = 3
   leal -4(%ebp), %eax# &localx
   movl %eax, (%esp) # 1st arg = &localx
   call incrk
```

66

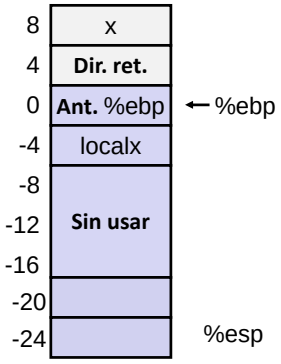
Recuperando variable local

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

- Recuperar localx de la pila como valor de retorno

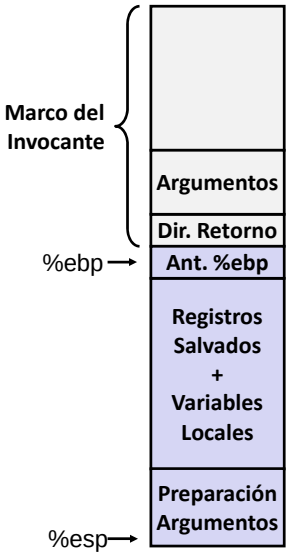
Parte final de add3

```
movl -4(%ebp), %eax # Return val= localx
leave
ret
```



Resumen (Procedimientos en IA 32)

- **Puntos Importantes**
  - Pila es la estructura de datos correcta para llamada / retorno procedimientos
    - Si P llama a Q, entonces Q retorna antes que P
- **Recursividad (& recursividad mutua) con mismas convenciones de llamada normales**
  - Se pueden almacenar valores tranquilamente en el marco de pila local y en registros salva-invocado
  - Poner argumentos de la función en tope de pila
  - Devolver resultado en %eax
- **Punteros son direcciones de valores**
  - Global o en pila



Guía de trabajo autónomo (4h/s)

- **Estudio: del Cap.3 CS:APP (Bryant/O'Hallaron)**
  - Switch Statements, Procedures.
    - 3.6.7 - 3.7 pp.247-266
    - Probl. 3.28 - 3.34 pp.251-52, 257-58, 262, 265-66

Bibliografía:

[BRY11] Cap.3 Computer Systems: A Programmer's Perspective. Bryant, O'Hallaron. Pearson, 2011
Signatura ESIIT/C.1 BRY com