Programación de IA-32
Modo Real

Control del flujo de la ejecución

Erwin Meza Vega

IA-32 Modo real (1/2)

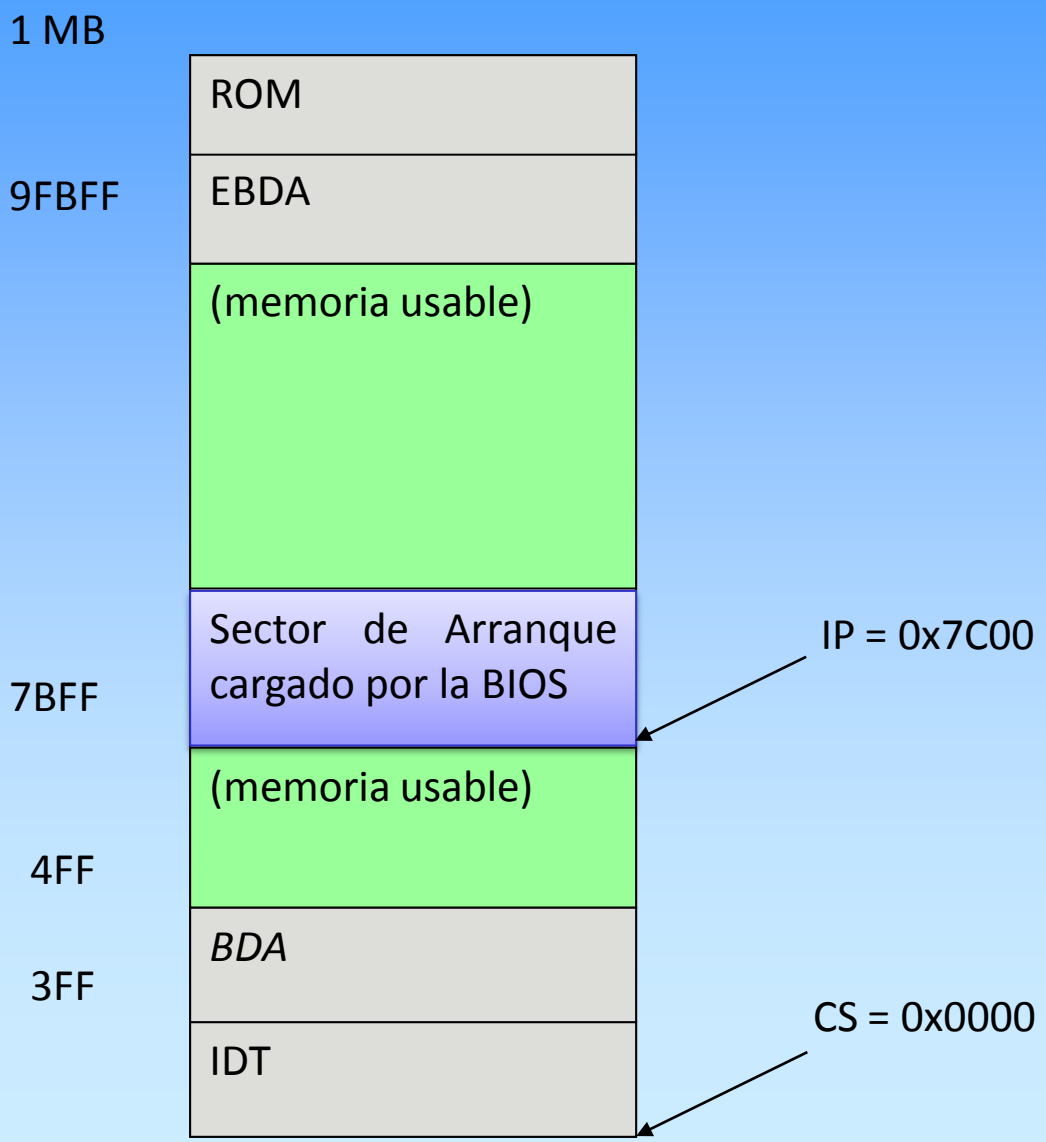
- Acceso sólo a características limitadas del procesador
  - Se comporta como un 8086 muy rápido
- Uso de segmentación de una forma especial
  - Segmentos de 64 KB
  - Los registros de segmento (CS, DS, ES, FS, GS, SS) apuntan a la base de un segmento en memoria, dividida en 16
  - Es necesario establecer explícitamente los valores en los registros de segmento
  - Sólo se puede tener acceso al primer Megabyte de memoria

IA-32 Modo Real (2/2)

- La BIOS lee el sector de arranque del dispositivo (disco), y lo copia en la posición de memoria 0x7C00 (31744)
- El sector de arranque recibe el control de la ejecución.
  - Se deben configurar los registros de segmento CS, DS
  - Se debe crear una pila en una región de la memoria que no esté siendo utilizada (configurar SS = base/16 y SP = tope)
    - Para recordar: la pila crece hacia atrás en la memoria, es decir de la dirección más alta a la más baja.
    - En modo real las operaciones sobre la pila siempre operan sobre 16 bits (2 bytes)
    - **Push**: decrementa SP en 2 y almacena el valor
    - **Pop**: saca el valor apuntado por SP e incrementa SP en 2 (bytes)

Disposición de la memoria

Antes de pasar el control al sector de arranque, la BIOS configura el primer MegaByte de memoria de la siguiente forma:



Flujo de ejecución

El procesador realiza el siguiente ciclo general:

1. FETCH: Lee la instrucción de memoria apuntada por CS:IP
   - Incrementa IP en el número de bytes que ocupa la instrucción leída
2. DECODE: Interpreta la instrucción leída, para determinar la operación y los operandos que intervienen
   - Busca los operandos que intervienen en la instrucción, calcula las direcciones de memoria correspondientes y obtiene los datos requeridos
3. EXECUTE: Ejecuta la instrucción leída e interpretada
   - Puede continuar la ejecución en otra posición de memoria, generar errores (excepciones), modificar FLAGS…
4. Vuelve al paso 1

La ejecución continúa en la instrucción que se encuentra en memoria inmediatamente después de la que se acabó de ejecutar, a menos que en el paso 3 se modifique el flujo de la ejecución

Modificar del flujo de la ejecución

El procesador ofrece varias alternativas para modificar el flujo de ejecución (continuar la ejecución en otra posición de memoria).

Algunas alternativas son:

1. Saltos (instrucciones *jmp, ljmp, jz, je, jne, …)*
2. Llamada a rutinas (instrucciones *call, lcall*)
3. Llamada a una interrupción (instrucción *int*)
4. Retorno de interrupciones (instrucción *iret*) realizados por el procesador o 'simulados'.
5. Errores al ejecutar una instrucción (división por cero, etc). El procesador lanza una "Excepción" (interrupción)

En la práctica todas las alternativas modifican el valor del registro IP (Instruction Pointer)

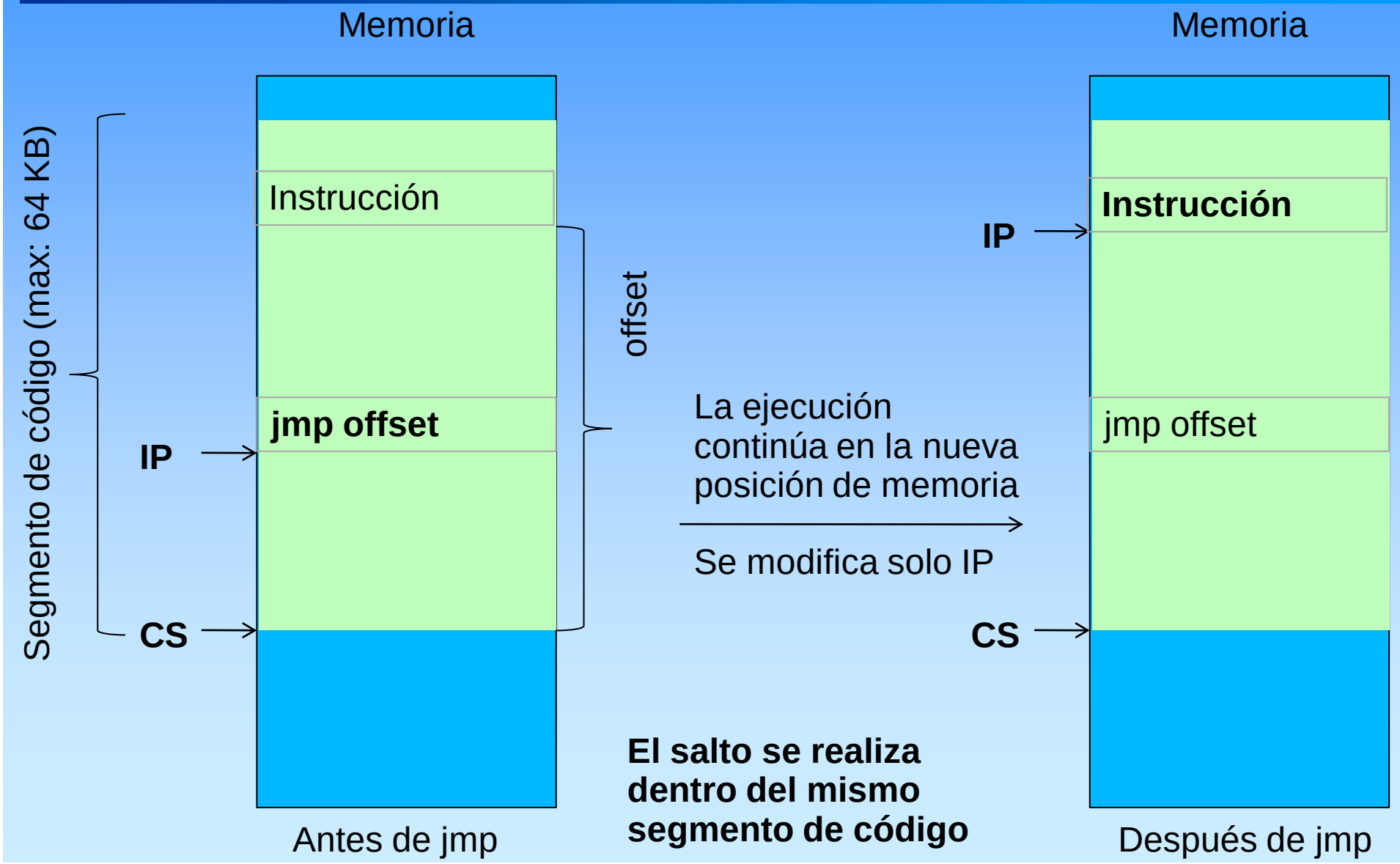
Algunas de éstas también modifican el valor del registro de segmento de código (CS)

Saltos incondicionales

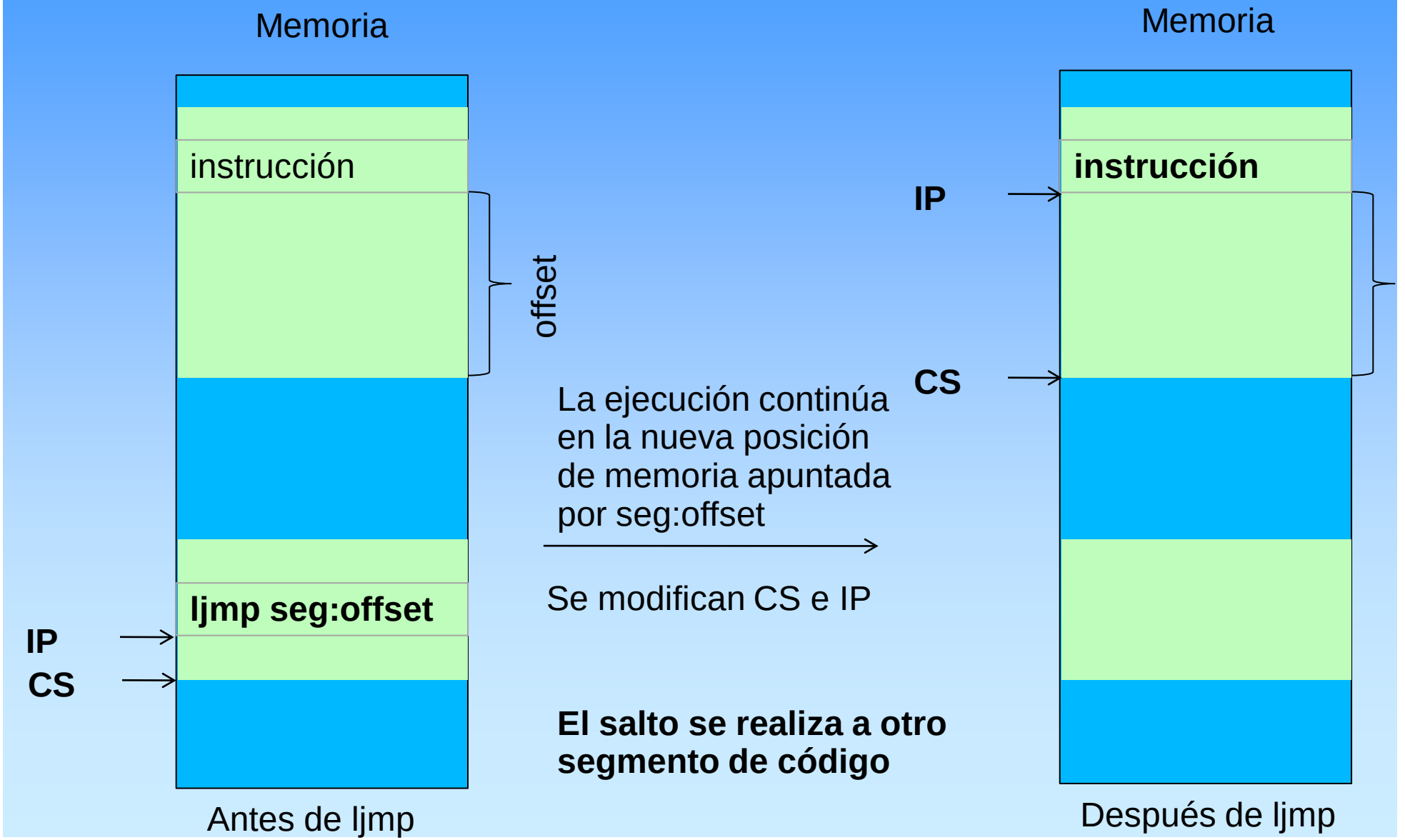
Transfieren el control de la ejecución a otra posición de memoria

- Jmp: Salto a otra posición dentro del mismo segmento de código: Se realiza especificando el desplazamiento dentro del segmento actual. Solo modifica el registro IP.
- Ljmp, Jmp Far: Salto a otro segmento de código. Se realiza especificando una dirección lógica (seg : offset). Este tipo de salto modifica al registro CS y al registro IP.

Instrucción jmp



Instrucción ljmp



Saltos condicionales

Dependen del estado del registro FLAGS. Los bits de este registro se pueden modificar implícitamente cuando se ejecuta una instrucción

- Ej: Cuando la operación aritmética o lógica que se acabó de ejecutar da como resultado cero, se activa el bit ZF (Zero Flag) en el registro FLAGS (ZF = 1)
- En este caso se puede ejecutar la instrucción jz (jump if zero), para transferir el control a otra posición de memoria si la operación dio como resultado cero.

Al igual que en los saltos incondicionales, se debe especificar el nuevo desplazamiento dentro del segmento de código.

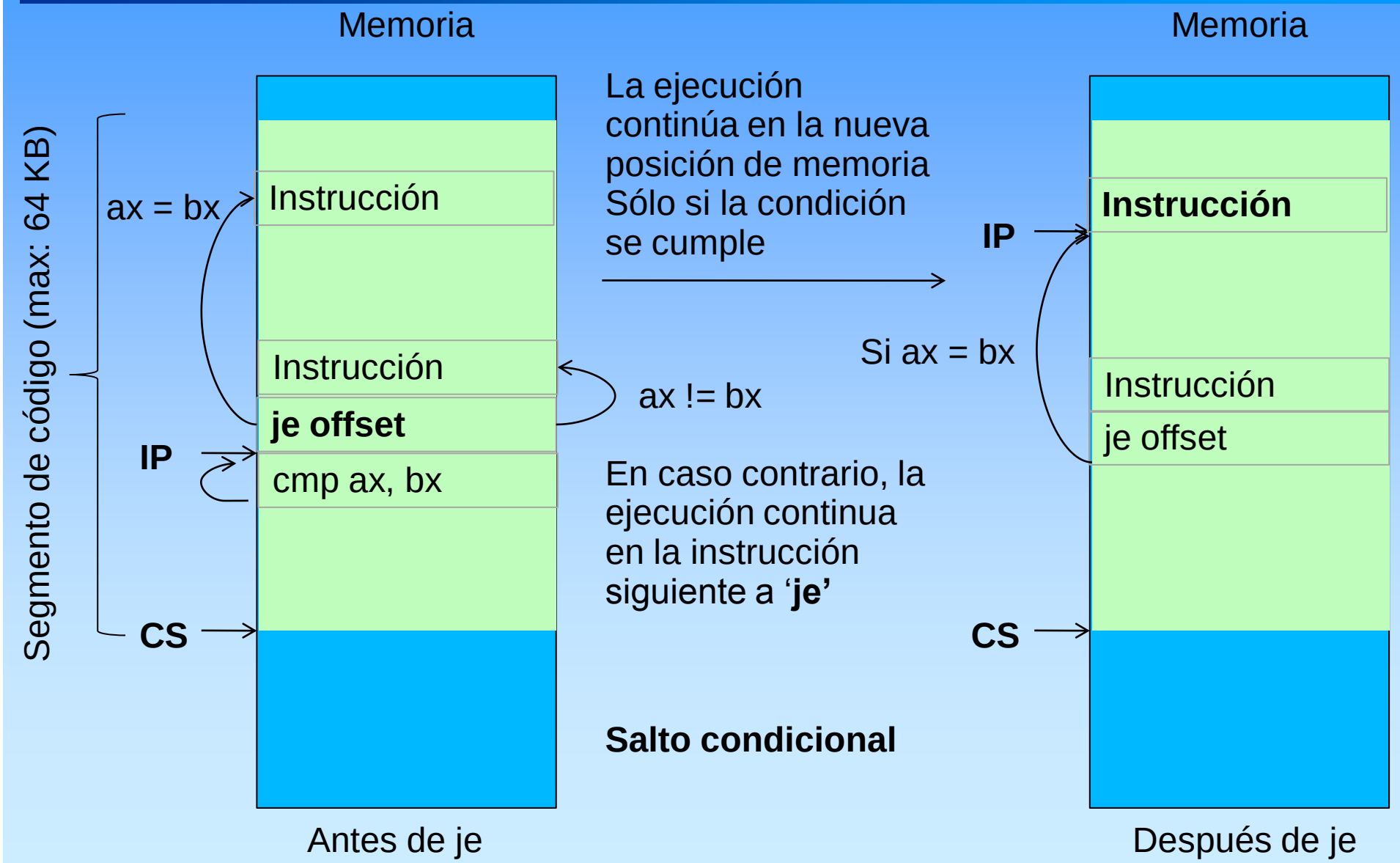
Saltos condicionales

Algunas instrucciones de salto condicional:

- jz: jump if zero, jnz: jump if not zero
- je, jne, jl, jle, jg, jge: Se pueden utilizar para pasar el control a otra posición de memoria si un operando es:
  - igual (e),
  - no igual (ne)
  - menor (l)
  - menor o igual (le)
  - mayor (g)
  - mayor o igual (ge)

  que el otro operando.
- jc: jump if carry. Saltar a otra posición si el bit Carry Flag (Acarreo) se encuentra en 1.

Ejemplo: instrucción je



Llamada a rutinas

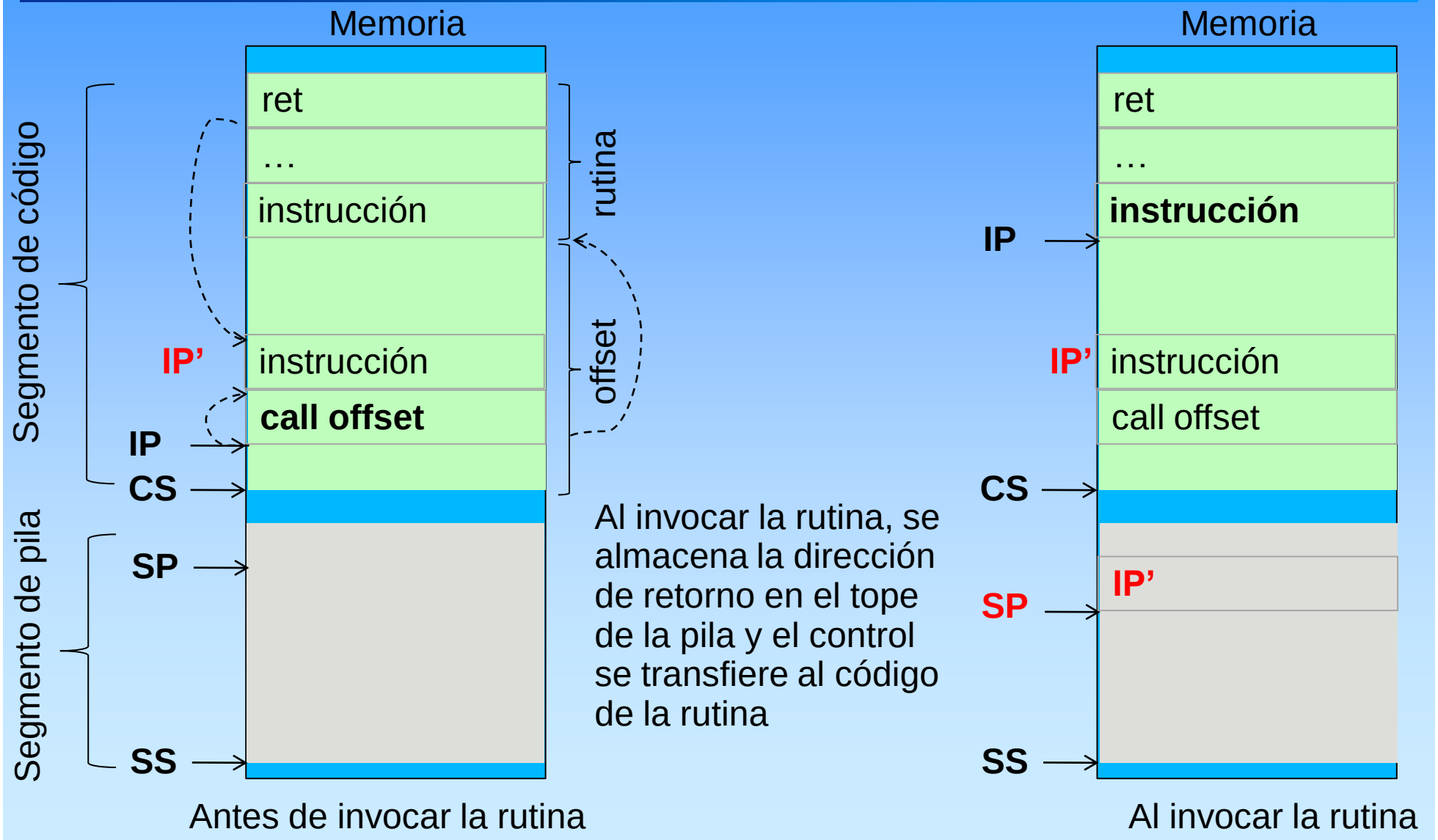
Similares a las instrucciones jmp y ljmp:

- call: Invocar a una rutina dentro del mismo segmento de código. Solo modifica el registro IP.
- lcall, call far: Invocar una rutina en otra posición de memoria, especificada por una dirección lógica (seg : offset). Este tipo de salto modifica al registro CS y al registro IP.

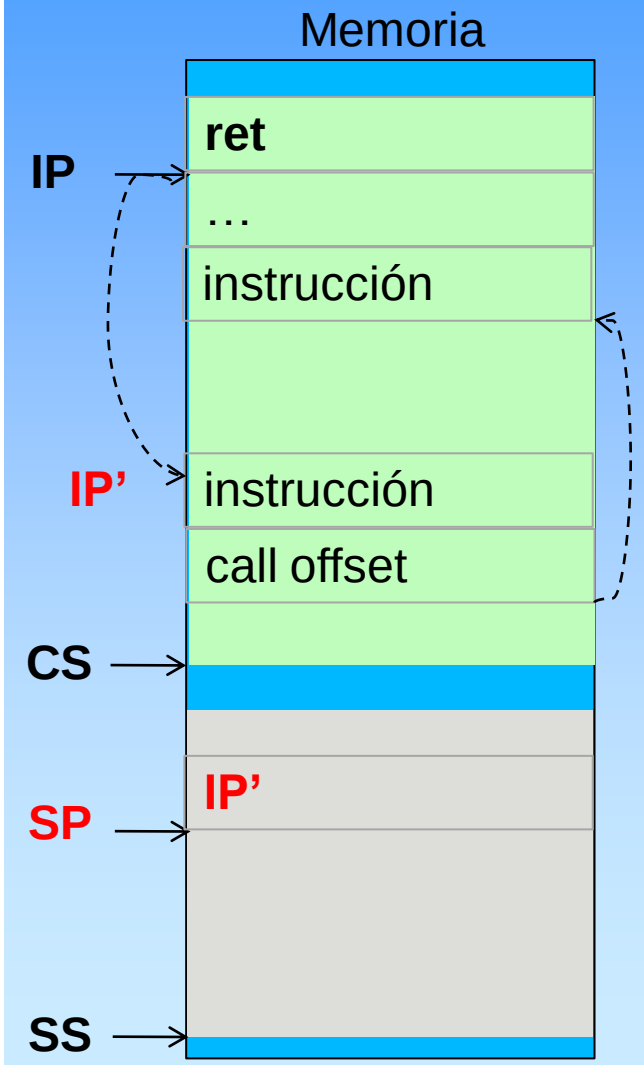
Luego de ejecutar la rutina, se debe ‘retornar’ a la posición desde la cual se invocó. Para lograr este propósito, el procesador almacena automáticamente en la pila:

- En el caso de call, el valor de IP al cual se debe ‘retornar’
- En el caso de lcall, el valor de CS e IP al cual se debe retornar (ya que con lcall se modifica tanto CS como IP).

Llamada a rutinas locales (1/2)



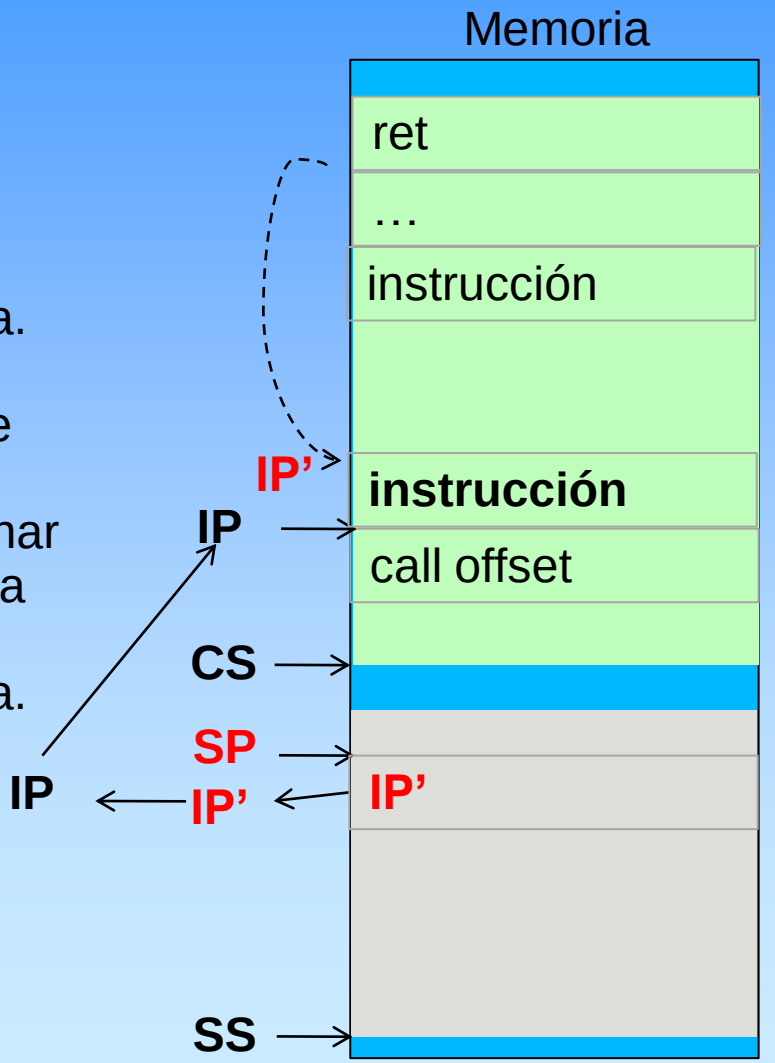
Llamada a rutinas locales (2/2)



Antes de retornar de la rutina

La instrucción 'ret' permite 'retornar' la ejecución a otra posición de memoria.

El procesador extrae del tope de la pila el valor que debe asignar a IP para continuar la ejecución en otra posición de memoria.



Al retornar de la rutina

Paso de parámetros a rutinas

Antes de invocar la rutina (instrucción ***call***) se deben ingresar los parámetros en la pila usando la instrucción **push**.

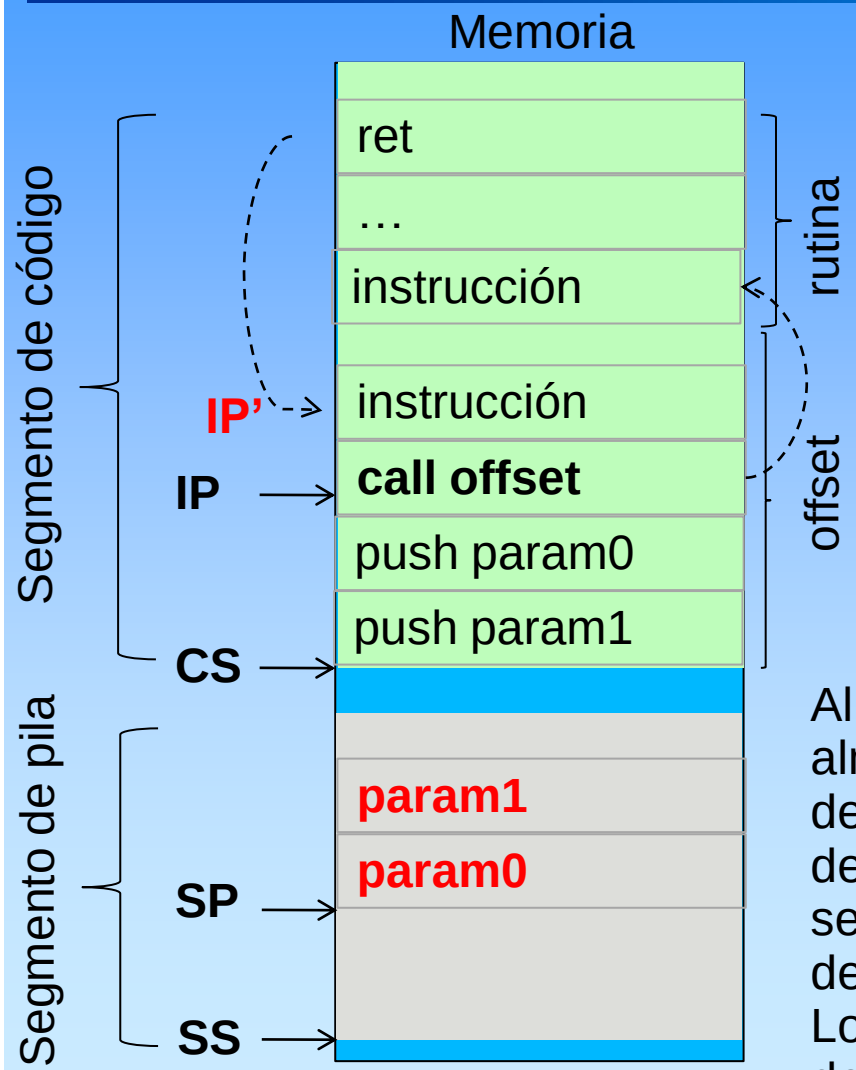
Los parámetros se deben almacenar en la pila en el orden inverso al cual se espera dentro de la rutina

En C: rutina (param0, param1, … , paramN)

Se deben ingresar de la siguiente forma:

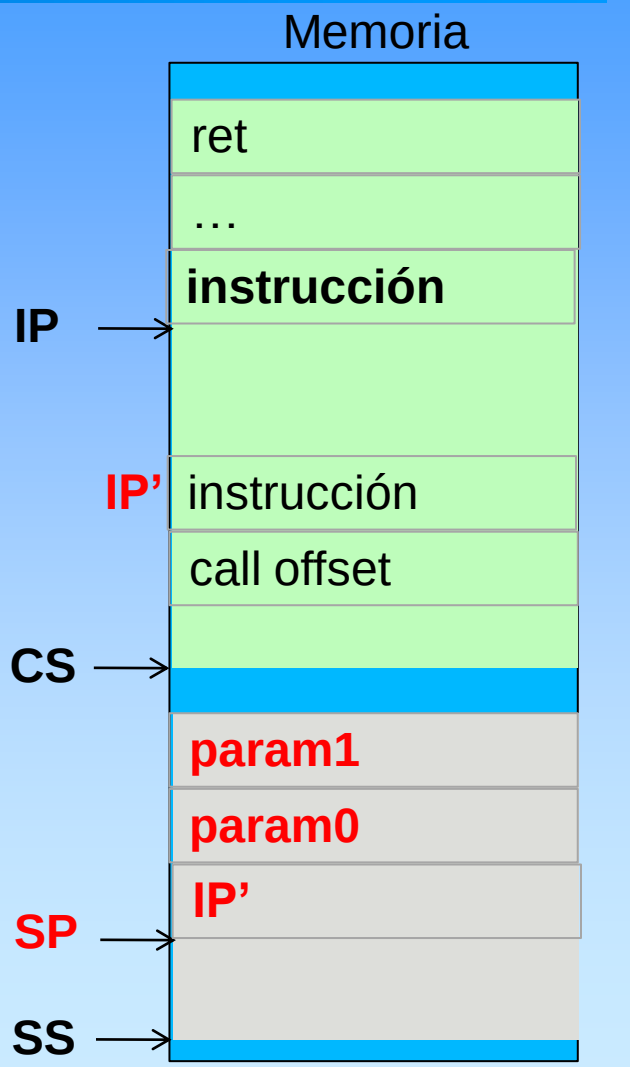
```
push paramN
....
push param1
push param0
call rutina
```

Paso de parámetros a rutinas (1/2)



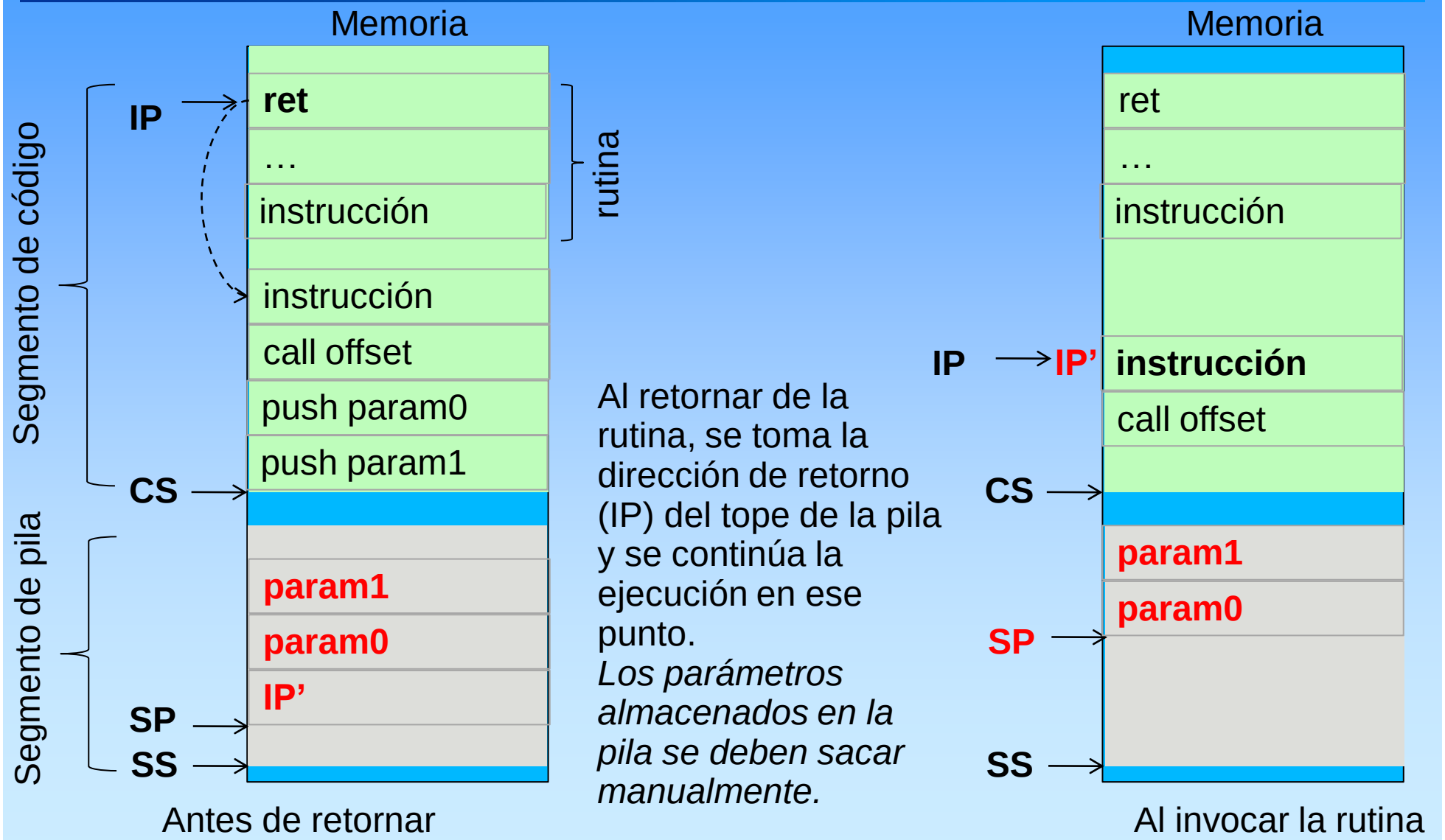
Antes de invocar la rutina

Al invocar la rutina, se almacena la dirección de retorno en el tope de la pila y el control se transfiere al código de la rutina
Los parámetros se deben encontrar en la pila *en orden inverso*



Al invocar la rutina

Paso de parámetros a rutinas (2/2)



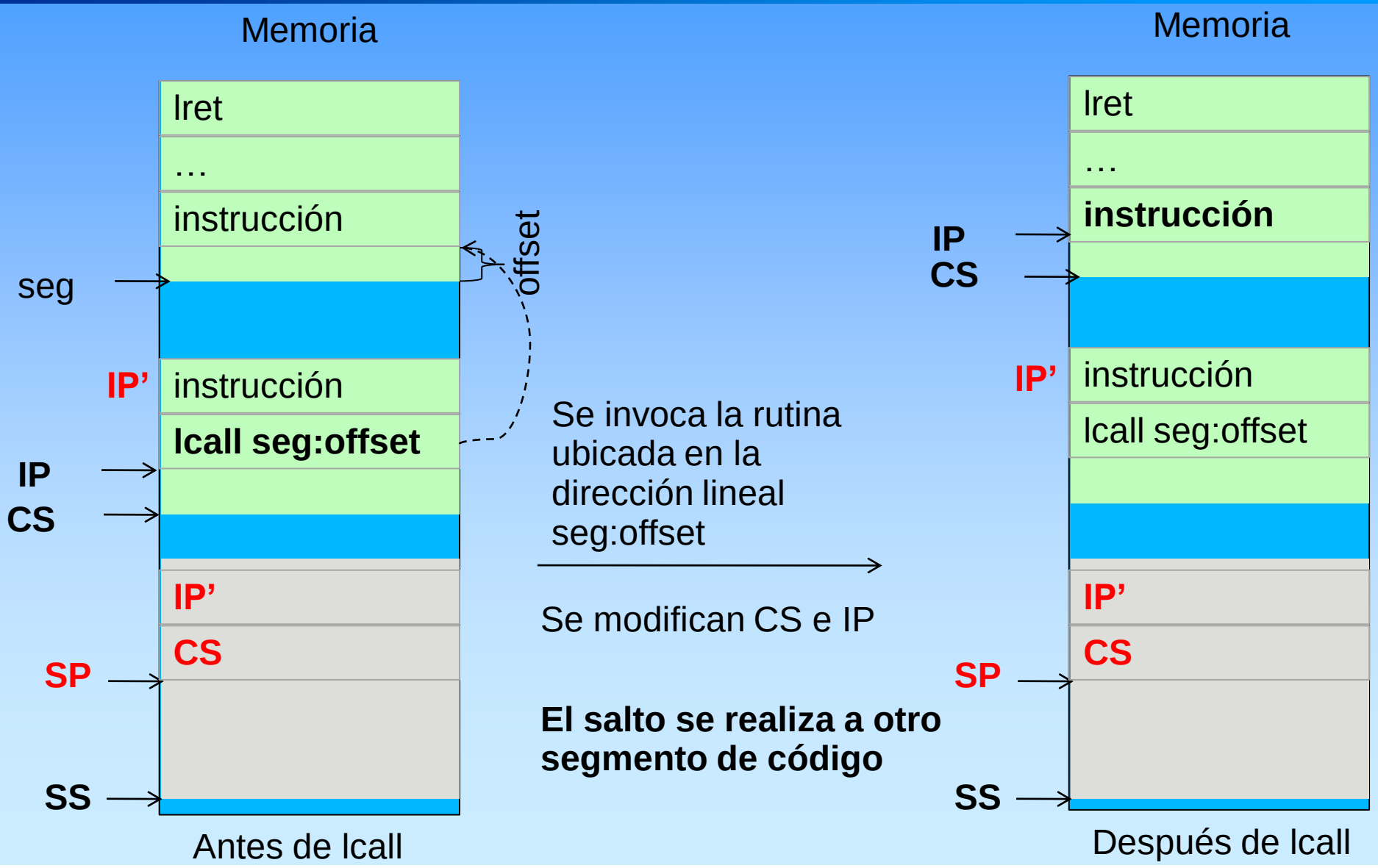
Paso de parámetros a rutinas

Luego de retornar de la rutina, se deben extraer los parámetros almacenados en la pila. Esto se puede lograr usando la instrucción **pop** tantas veces como parámetros se haya incluido en la pila, o desplazando el apuntador de la pila hacia arriba en la memoria:

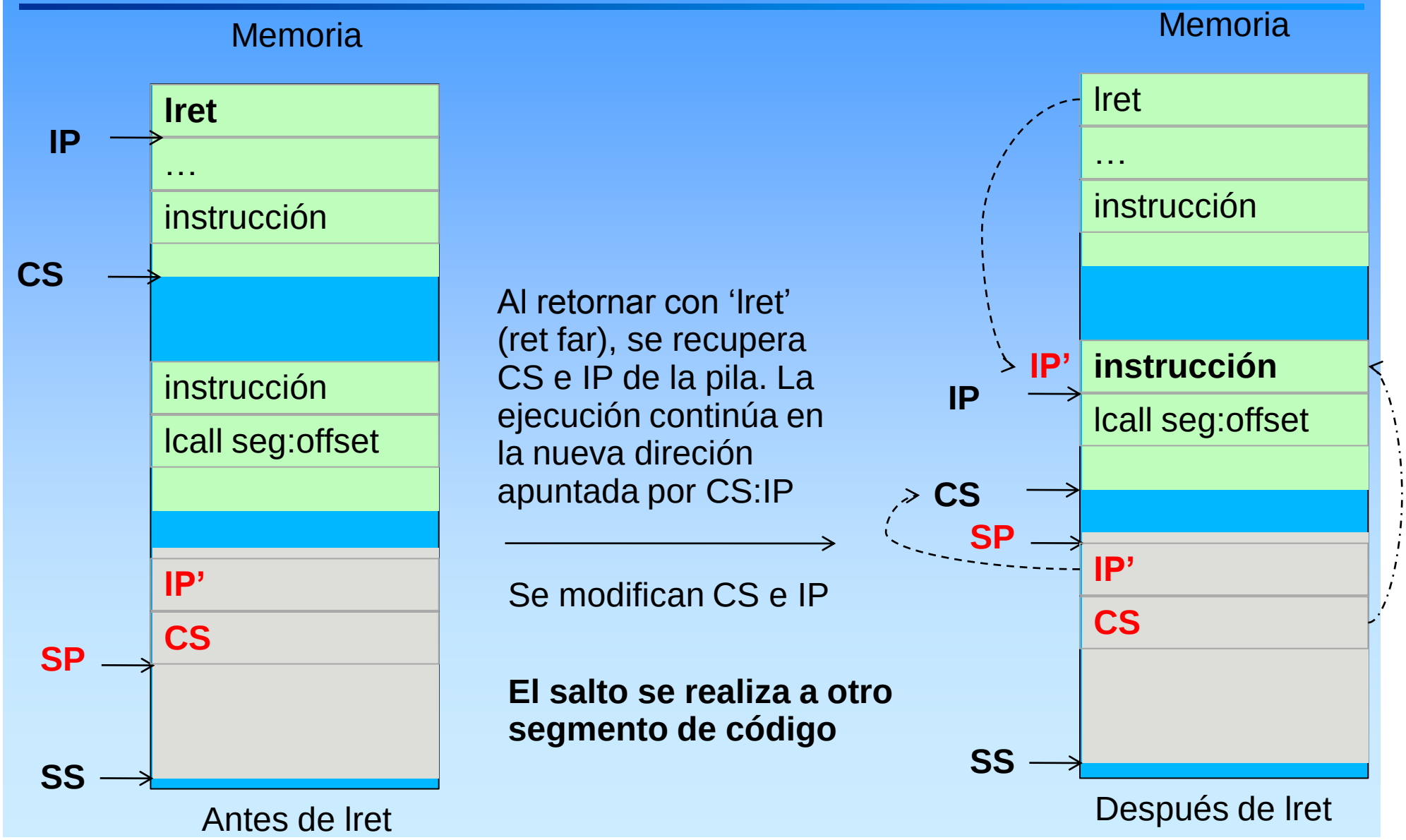
```
add SP, M
```

Donde M se obtiene de multiplicar el número de parámetros almacenados en la pila por 2 (debido a que cada parámetro en la pila ocupa 1 word : 2 bytes).

Llamada a rutinas en otro segmento (1/2)



Llamada a rutinas en otro segmento (2/2)



Llamada a interrupción

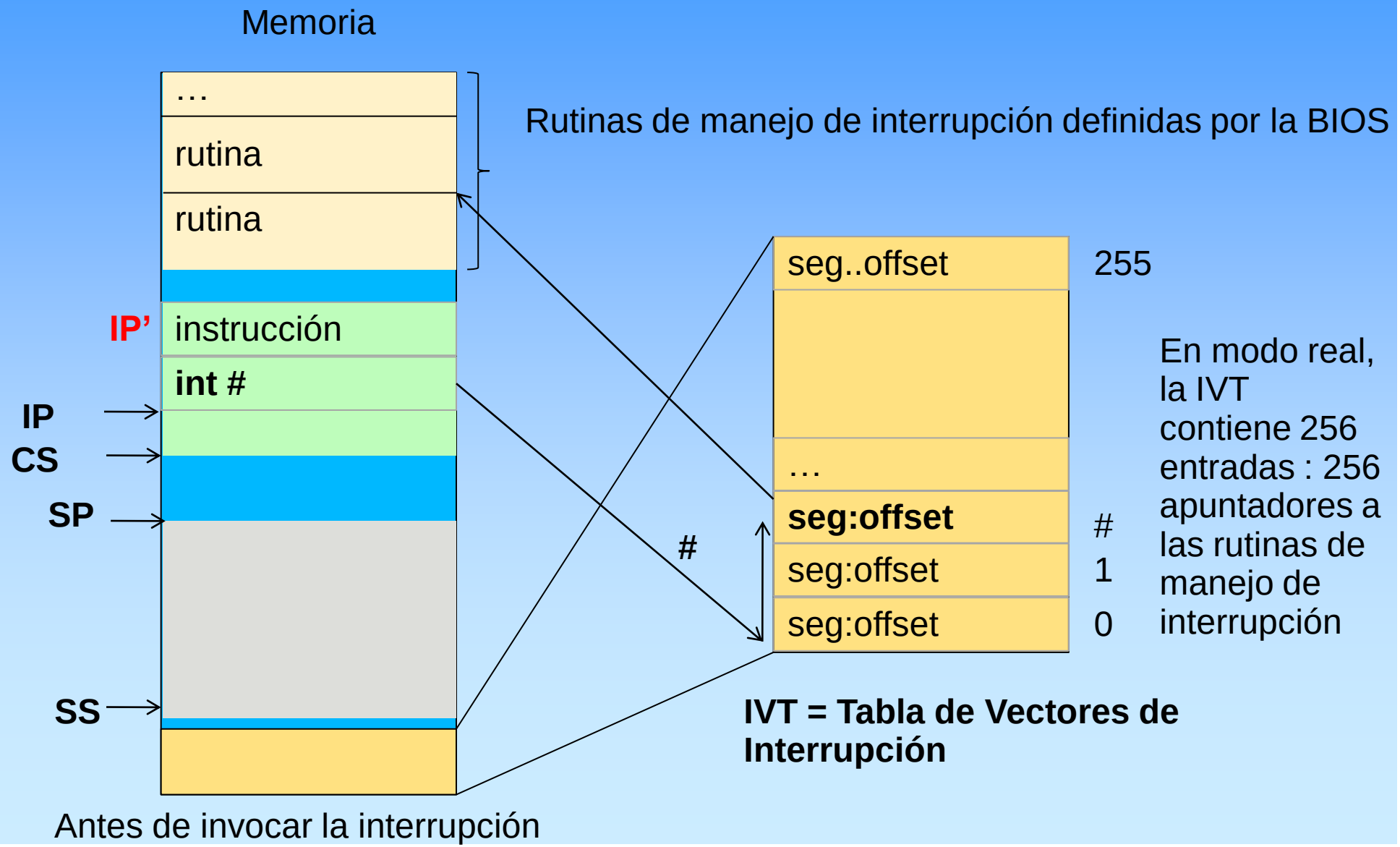
Es posible invocar una interrupción por software, mediante la instrucción ***int***.

- Al recibir una interrupción, el procesador automáticamente almacena en la pila el valor de FLAGS, CS e IP (la posición actual de ejecución)
- Luego el procesador transfiere el control a una rutina de manejo de interrupción.
- Cuanto la rutina de manejo de interrupción termina, ejecuta la instrucción 'iret'. Esta instrucción extrae del tope de la pila los valores de FLAGS, CS e IP en los cuales se debe continuar con la ejecución.

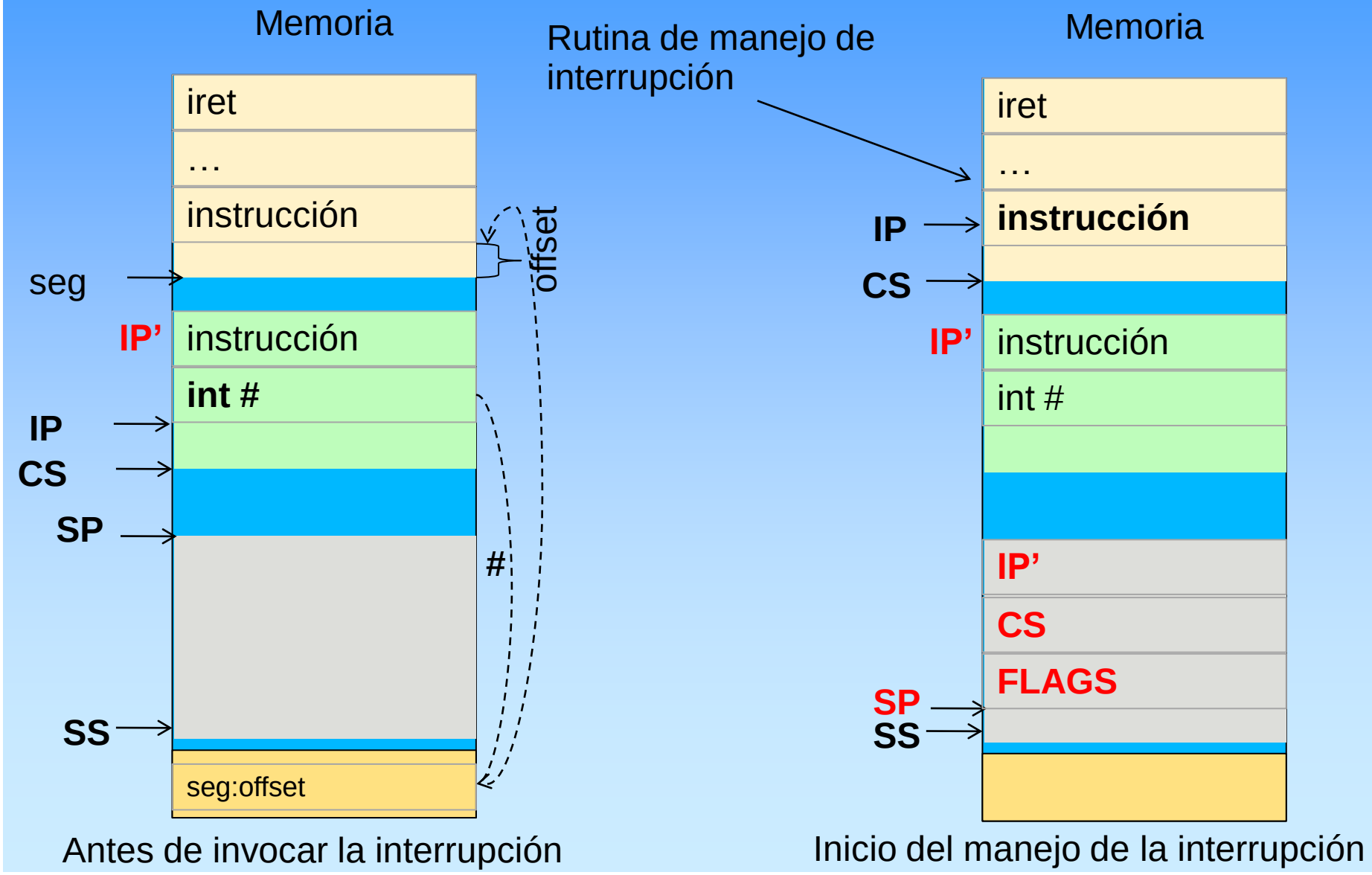
En modo real, las rutinas de manejo de interrupción son configuradas por la BIOS al inicio del sistema

- La BIOS configura en la posición 0 de la memoria una tabla (arreglo) llamada 'Tabla de Descriptores de Interrupción' (IDT). Cada entrada de esta tabla contiene las direcciones lógicas (seg:offset) en las cuales se encuentran las rutinas de manejo de interrupción.

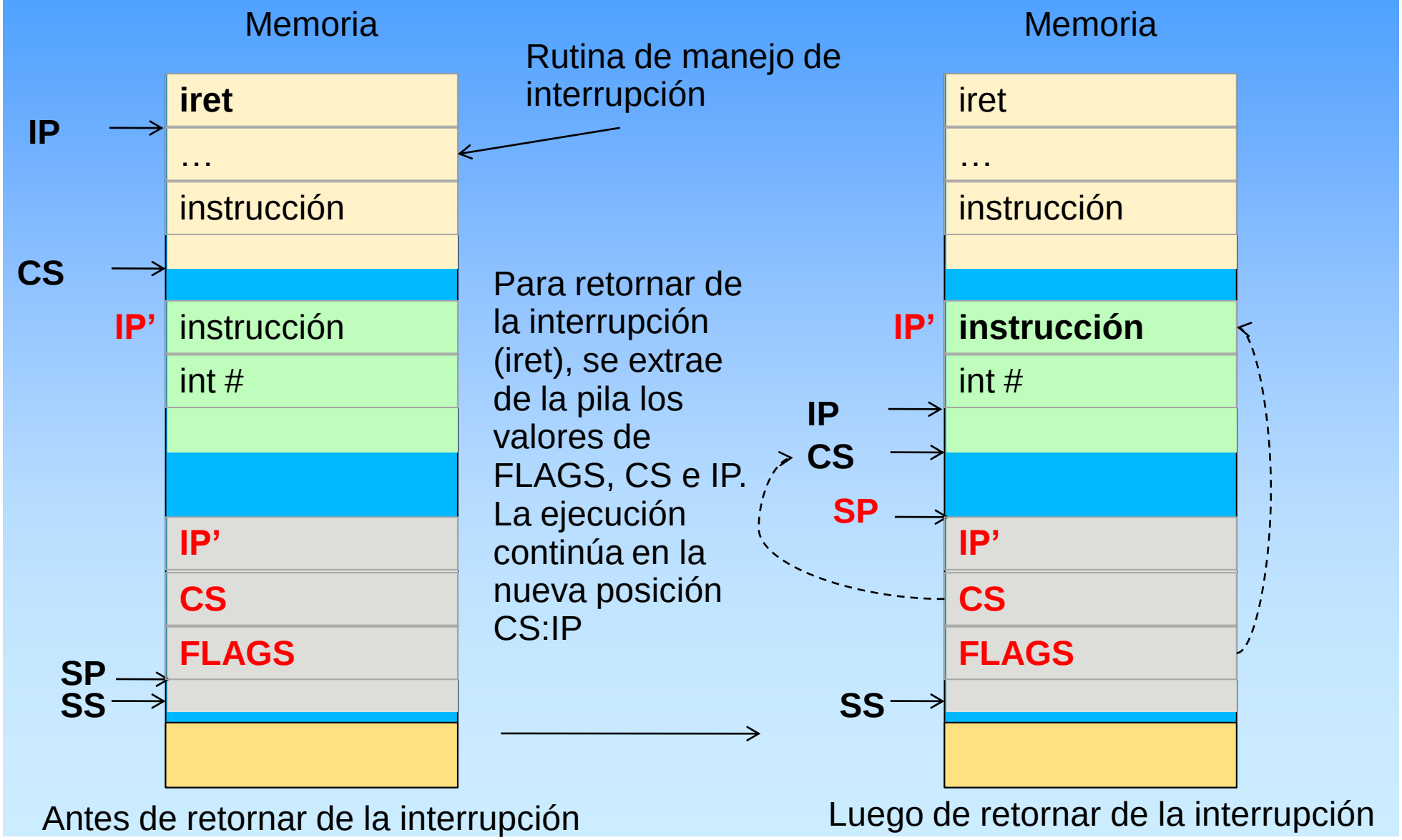
Interrupción por software (1/3)



Interrupción por software (2/3)



Interrupción por software (3/3)



Simular un retorno de interrupción

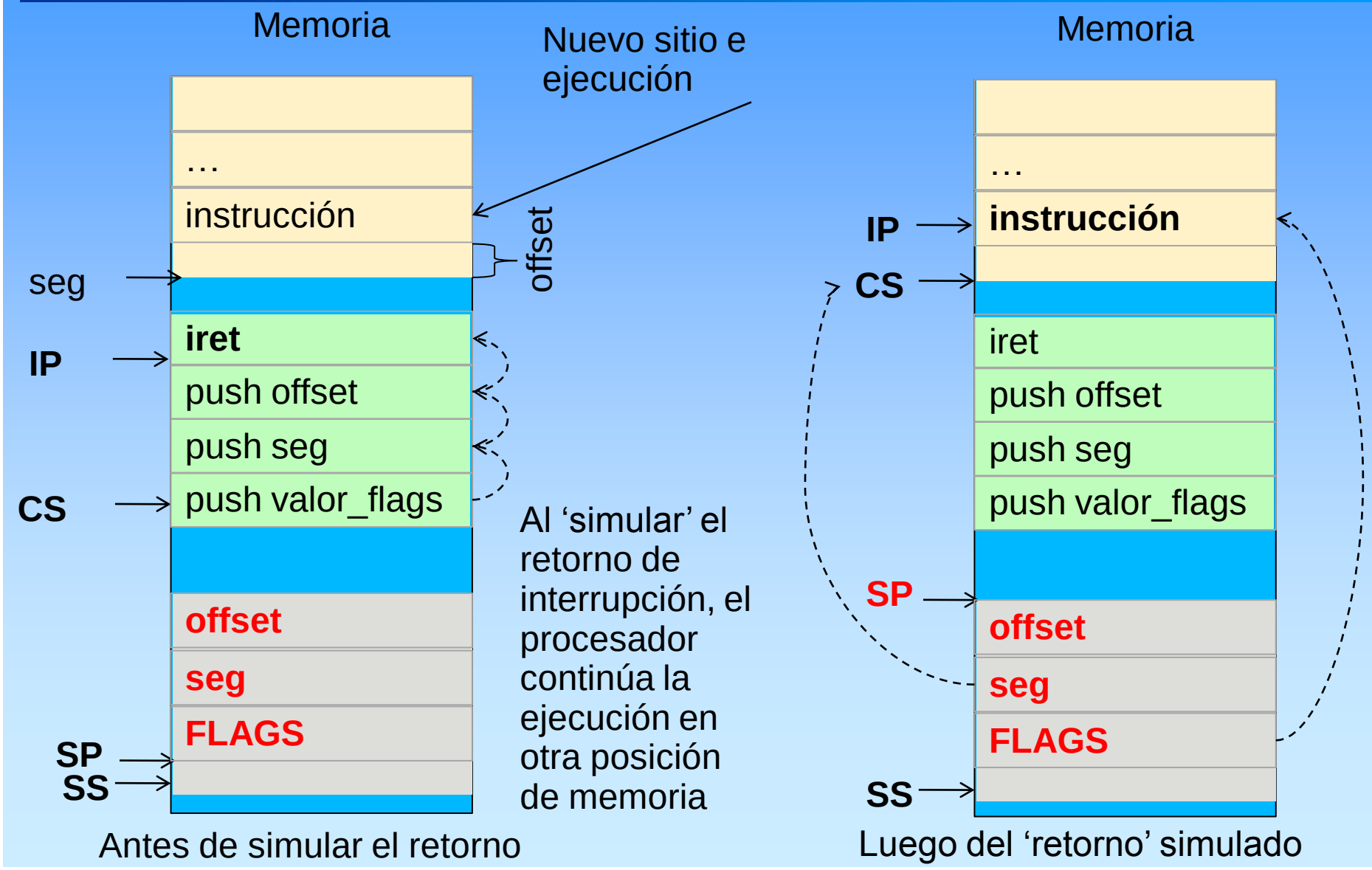
Conociendo el mecanismo usado por el procesador para retornar de una interrupción (que almacena automáticamente FLAGS, CS e IP), es posible pasar el control a otra posición de memoria ‘simulando’ que ha ocurrido una interrupción y ejecutando la instrucción ‘iret’

- Se debe almacenar en la pila el valor de FLAGS, el nuevo valor de CS y el nuevo valor de IP al cual se desea pasar el control.
- Luego se invoca la instrucción ‘iret’

Luego de pasar al nuevo punto de ejecución no es posible retornar al punto anterior.

Un mecanismo similar a éste se usa en modo protegido para implementar la multitarea. Además de FLAGS, CS e IP se almacenan y recuperan los valores de los registros de propósito general (eax, ebx, .. etc.).

Simular un retorno de interrupción



Referencias

Manuales de Intel

http://www.intel.com/products/processor/manuals/