# Tema 3

## SUBRUTINAS

# ÍNDICE

- Definición e instrucciones básicas
- Soporte para el tratamiento de subrutinas (ejecución de la subrutina y gestión del bloque de activación)
- Interrupciones vs llamadas a procedimiento

# Subrutina: definición

- Los programas de alto nivel hacen uso de estos bloques de código a los que se les pasan unas variables (parámetros) con las que ejecuta una operación y calcula unos resultados (*funciones, procedimientos o métodos*).
- **Subrutina:** porción de código que realiza una operación en base a unos valores dados como parámetros y que puede ser invocado desde cualquier parte del código, incluso desde sí misma (contexto de lenguaje ensamblador)

# Subrutina: ventajas

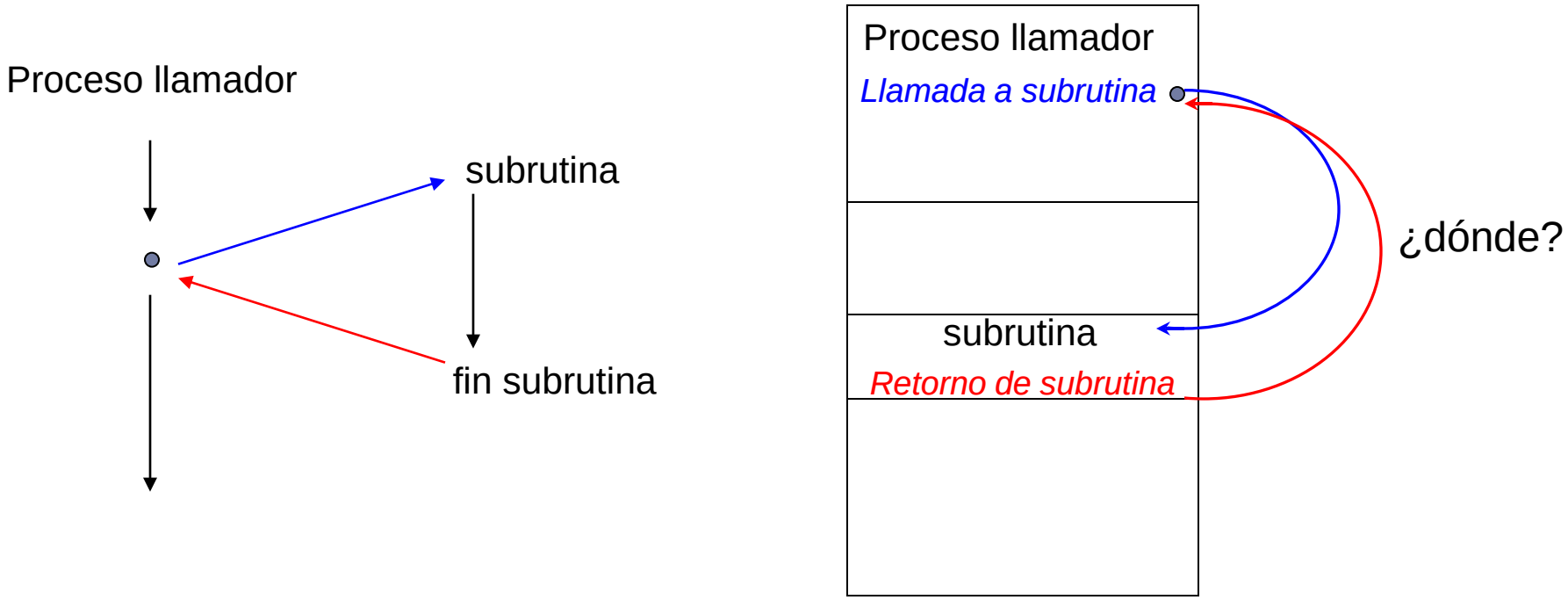
- División del problema
  - *en tareas más fáciles de escribir y depurar*
- Evita código redundante
- Encapsulado del código
  - *las tareas se comunican a través del paso de parámetros y resultados, por lo que un cambio en una de las tareas no implica cambios en el resto del programa*
  - *permite su reutilización en más de un programa (bibliotecas)*

# Subrutina: generalidad

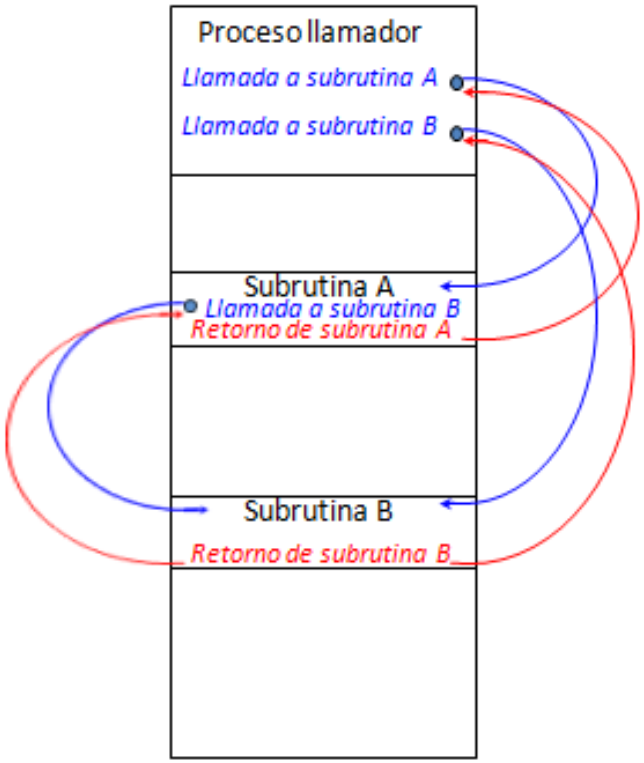
- Resolver un mismo problema ante diferentes datos (parámetros) → dotarla de generalidad:
  - Llamada y retorno de subrutina
  - Paso de parámetros y resultados

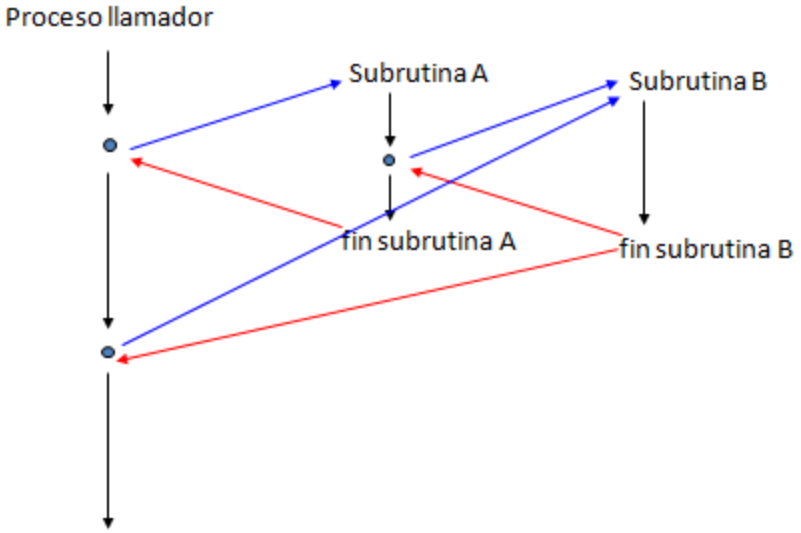
# Subrutina: llamada y retorno

El proceso llamador, llama a la subrutina, pasándole unos parámetros para que realice los cálculos y le devuelva un resultado, tras lo cual seguirá su ejecución



# Subrutina: llamada y retorno




Necesitamos un mecanismo que permita la invocación anidada de subrutinas.

# Instrucciones de control de secuencia

| Nombre | Mnemónico | |
|---|---|---|
| Salto | BR | *Branch* |
| Salto incondicional | JMP | *Jump* |
| Avanzar | SKP | *Skip* |
| Llamada a procedimiento | CALL | *Call* |
| Retorno de procedimiento | RET | *Return* |
| Comparación (resta) | CMP | *Compare* |
| Comparación (AND) | TEST | *Test* |

# Subrutina: llamada y retorno

- Llamada a subrutina
  - CALL etiqueta
- Retorno de subrutina
  - RET (no tiene parámetros)

# Subrutina: llamada y retorno

- Llamada a subrutina
  - CALL etiqueta
  - Salvar el actual valor del PC (dir. retorno)
  - Cargar en el PC la dirección de la subrutina
- Retorno de subrutina
  - RET no tiene parámetros
  - Cargar la dirección de retorno en el PC

# Subrutina: llamada y retorno

- ¿Dónde salvamos la dirección de retorno?

~~¿En un registro?~~ Impide el anidamiento de subrutinas

~~¿En memoria?~~ RET requiere una posición conocida (implícita)

¿Pila? Estructura LIFO que crece dinámicamente y cuya última posición siempre está disponible (SP)

*Es responsabilidad del programador que al terminar la ejecución de la subrutina la pila esté como al principio.*

# Subrutina: definición de la subrutina

- En la definición de la subrutina:
  - Número de parámetros
  - Tipo (número entero, cadena de caracteres,…)
  - Cómo se van a pasar
    - por valor (parámetro)
    - por referencia (la dirección)
  - Dónde
    - registros
    - memoria
    - pila

# Subrutina: paso de parámetros

- Paso de parámetros a través de registro
  - El programa llamador y el llamado asumen que los parámetros se almacenan en ciertos registros específicos. Antes de la instrucción de llamada el programa llamador deposita los valores pertinentes en estos registros y la subrutina comienza a procesarlos directamente

→ *desventaja: número limitado de registros de propósito general (muy eficiente con subrutinas con pocos parámetros y un único resultado, ya que el procesador no requiere almacenar datos en memoria)*

# Subrutina: paso de parámetros

- Paso de parámetros a través de memoria
  - Se define una zona de memoria conocida tanto para el programa llamador como para el llamado y en ella se copian el valor de los parámetros y el del resultado para su intercambio entre ambos programas

→ *ventaja: número arbitrario de parámetros*

→ *desventaja: estas zonas han de estar previamente definidas; múltiples espacios de parámetros en el caso de subrutinas anidadas → se requieren tantos espacios como invocaciones pendientes de terminar (igual que la dirección de retorno!)*

# Subrutina: paso de parámetros

- Paso de parámetros a través de la pila
  - Parámetros y resultados son resultados temporales que sólo tienen vigencia en un periodo concreto; además el orden de creación y destrucción de estos parámetros en llamadas anidadas es el adecuado para los mecanismos de gestión de la pila. El orden de almacenamiento ha de ser conocido.

→ *¿cómo acceder a esta zona de parámetros?*

SP + direccionamiento indexado

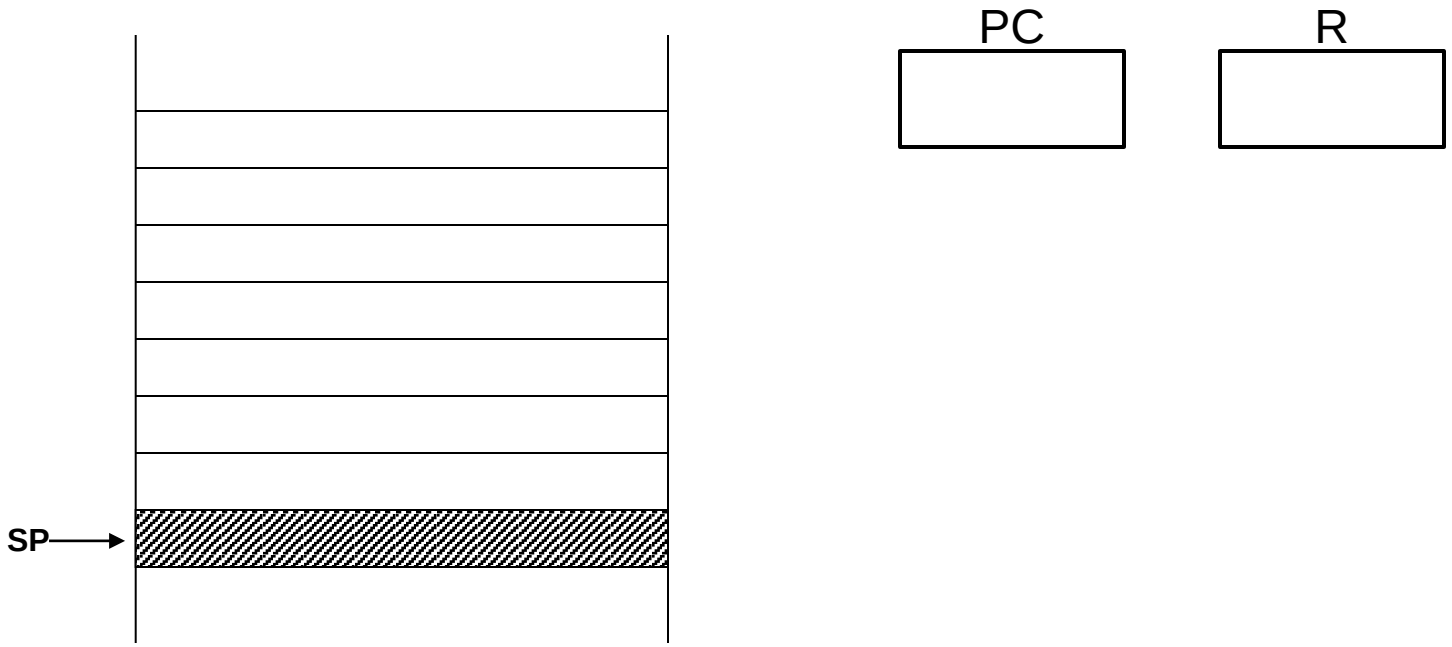
La pila puede variar durante la ejecución de la subrutina!

# Subrutina: paso de parámetros

- Para poder acceder a la zona de parámetros y resultados con desplazamientos constantes las primeras instrucciones copian el valor de SP en otro registro (previa copia de su valor en la pila) → ***puntero al bloque de activación***
- Al terminar la ejecución de la subrutina se deshace la estructura de datos creada en la pila en orden inverso.
- ***Bloque de activación***: porción de memoria de la pila que contiene el espacio para los resultados, los parámetros, la dirección de retorno y la copia del valor original del registro en el que se guardará el SP

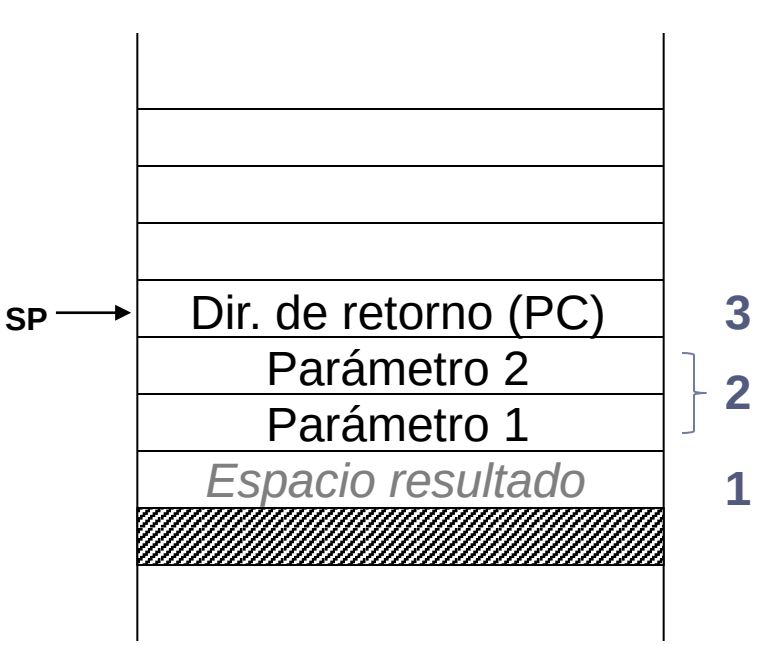
# Subrutina: bloque de activación

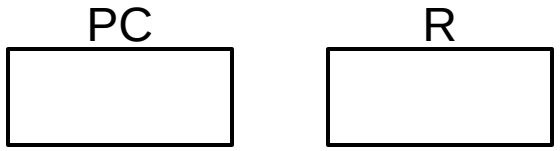
- Gestión del bloque de activación: programa llamador



# Subrutina: bloque de activación

- Gestión del bloque de activación: programa llamador

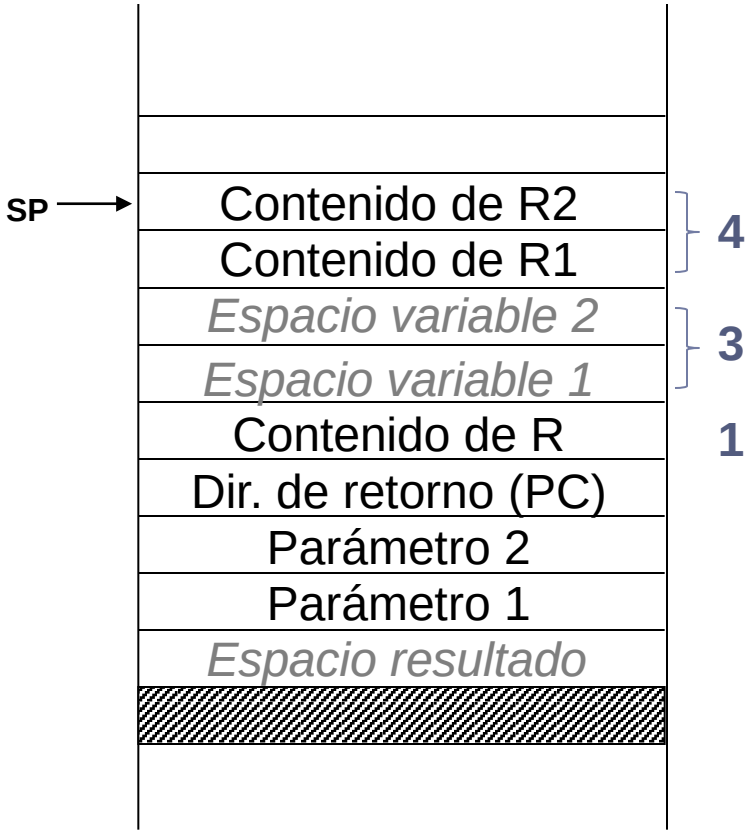


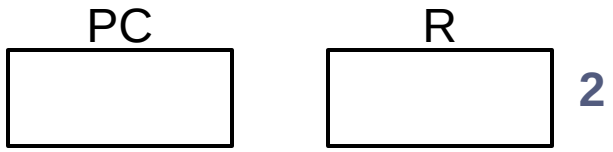




1. Reserva espacio para el resultado
2. Pasa los parámetros (en orden)
3. Llama a la subrutina (guarda en la pila la dirección de retorno y carga en el PC la dirección de la subrutina)

# Subrutina: bloque de activación

- Gestión del bloque de activación: programa llamado (subrutina)

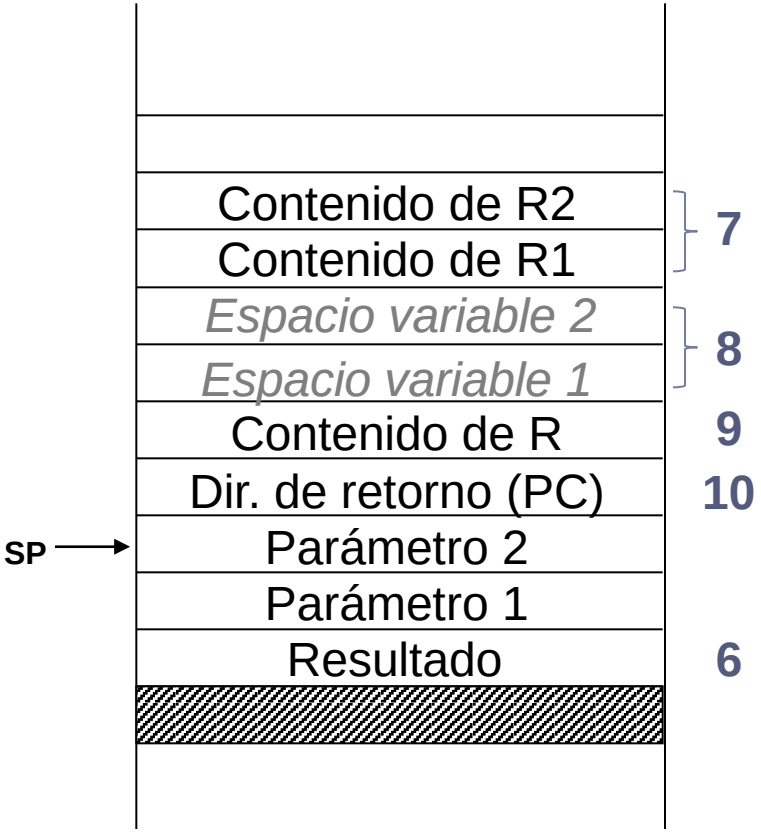


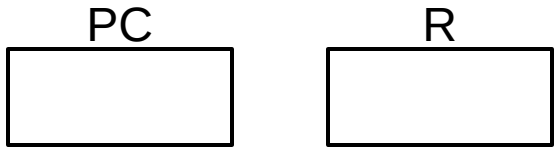




1. Guarda en la pila el contenido del registro que usará como puntero al bloque de activación (R)
2. Guarda en R el valor actual del SP
3. Reserva espacio para variables locales
4. Guarda el contenido de los registros que utilizará durante la ejecución de la subrutina
5. Ejecuta la subrutina (direccionamiento indexado utilizando R)

# Subrutina: bloque de activación

- Gestión del bloque de activación: programa llamado (subrutina)

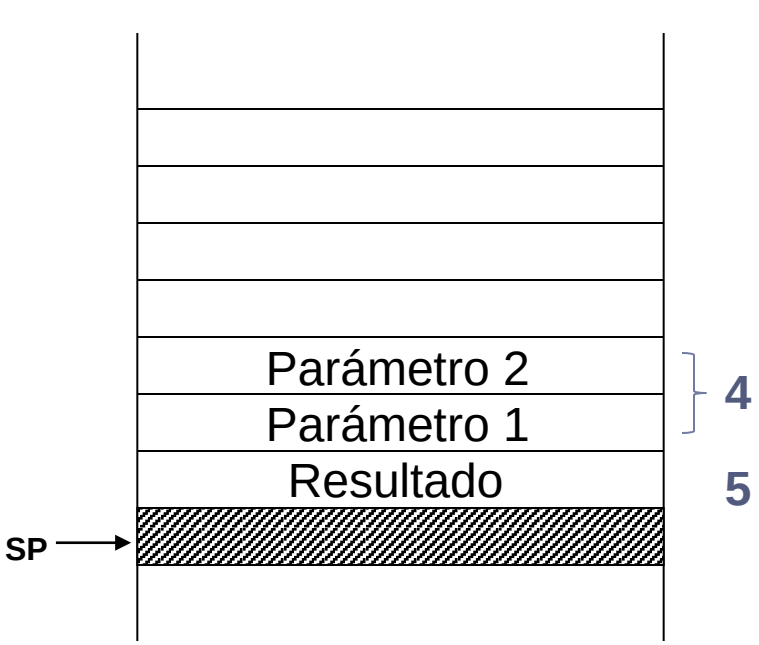


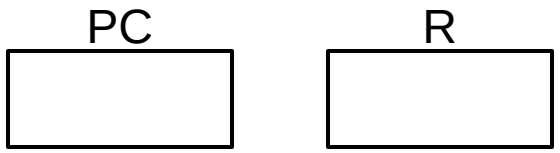




6. Guarda el resultado en el sitio reservado para el
7. Devolver a los registros que se utilizaron durante la ejecución de la subrutina el valor que tenían
8. Liberar el espacio para las variables locales (actualizar SP)
9. Devolver a R su valor inicial
10. Retorno de la subrutina (cargar en PC la dirección de retorno)

# Subrutina: bloque de activación

- Gestión del bloque de activación: programa llamador

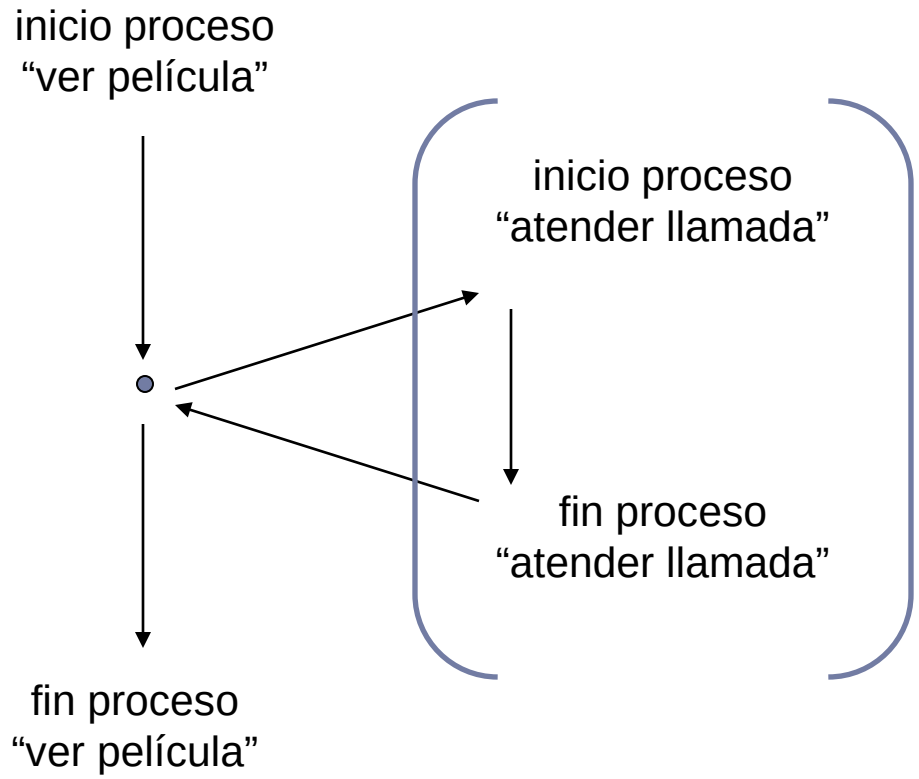



4. Sacar los parámetros de la pila (liberar el espacio)
5. Recuperar el resultado

# Interrupciones

- Estás en casa viendo una película cuando suena el timbre. ¿Qué haces?



# Interrupciones

- Cuando se da una interrupción, se rompe la ejecución normal del programa para ejecutar la rutina de servicio a la interrupción (RSI), volviendo tras esta a la ejecución normal del programa. Aunque se parece a la llamada a procedimiento, no es igual:
  1. No la provoca una instrucción, sino una señal (externa/interna) no previsible.
  2. La dirección del programa de atención se facilita mediante hardware (no hay campo de dirección).
  3. Ante una interrupción hay que guardar la información del bloque de registros (o de parte), no sólo el PC.

# Interrupciones

- **Interrupción hardware**
  - Cuando un recurso hw requiere atención → asíncrona (primero terminar la instrucción máquina en curso, luego atenderla): *temporizador, fallo alimentación, dispositivos de E/S (multiproceso + recurso compartido (gestionado por el S.O.) → "polling" ineficiente → interrupción hw)…*

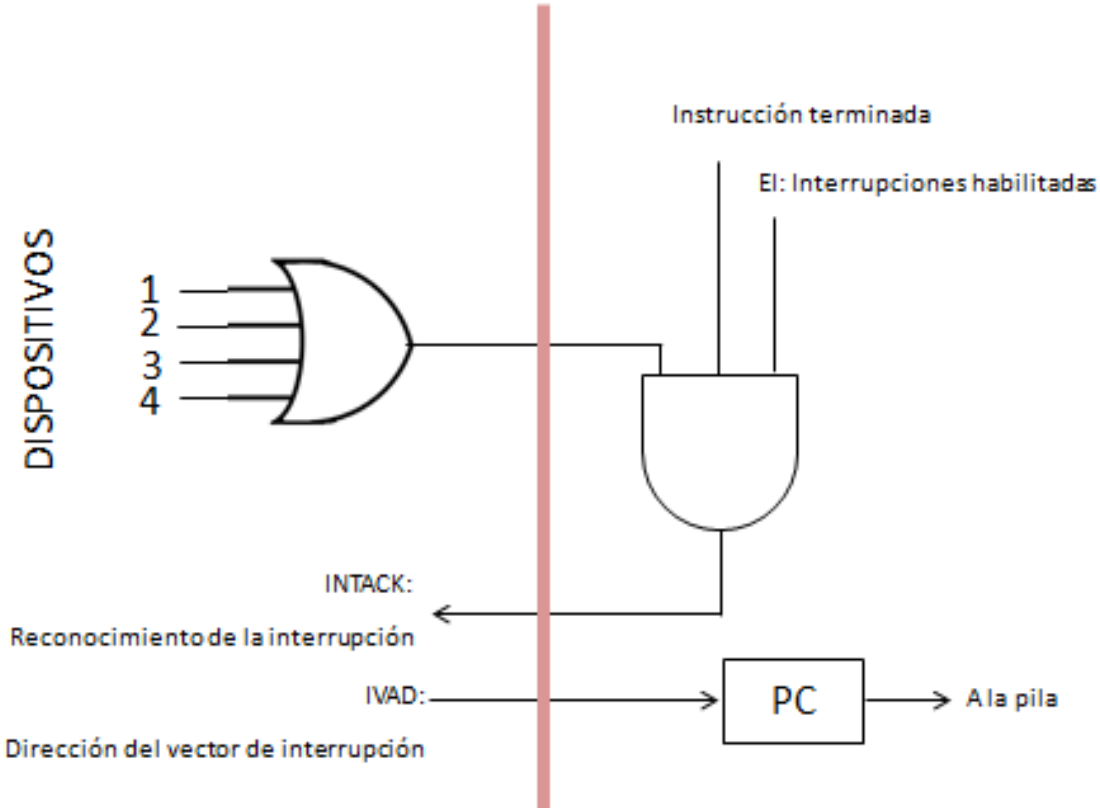
- **Interrupción software** o excepción
  - Introducidas por el programador (sincronizada) (INT).

- Trampas (***traps***)
  - Causadas por la CPU ante una condición de error: *utilización errónea o ilegal de un conjunto de datos, división por cero, direccionamiento no permitido…* Introducidas por el programador aunque no intencionadamente (sincronizada).

# Interrupciones

- Según importancia pueden ser
  - Enmascarables
  - No enmascarables
- Interrupción hw ¿cómo sabe la CPU qué dispositivo ha interrumpido?
  1. Multinivel (cada dispositivo una entrada; sencillo pero muy caro)
  2. Línea única + "polling" (pregunta uno a uno)
  3. Vectorizadas (además de la señal de interrupción, ponen en el bus de datos un *vector* que las identifica) (Periférico o controlador y gestión prioridad tema 5)

# Ejemplo interrupción externa




*En este ejemplo la interrupción pone directamente la dirección de la rutina de servicio (RSI); si sólo indicase el vector que la identifica la CPU lo usaría como índice en una tabla dónde tiene las direcciones de las diferentes RSI.*

# Ejemplo interrupción externa

- Pasos tras la interrupción:
  - SP←SP-1 actualizar SP (hacer sitio en la pila)
  - M[SP]←PC guardar dirección de retorno
  - SP←SP-1 actualizar SP (hacer sitio en la pila)
  - M[SP]←PSR guardar la palabra de estado
  - EI←0 deshabilitar interrupciones
  - INTACK←1 acusar recepción
  - PC←IVAD dirección RSI