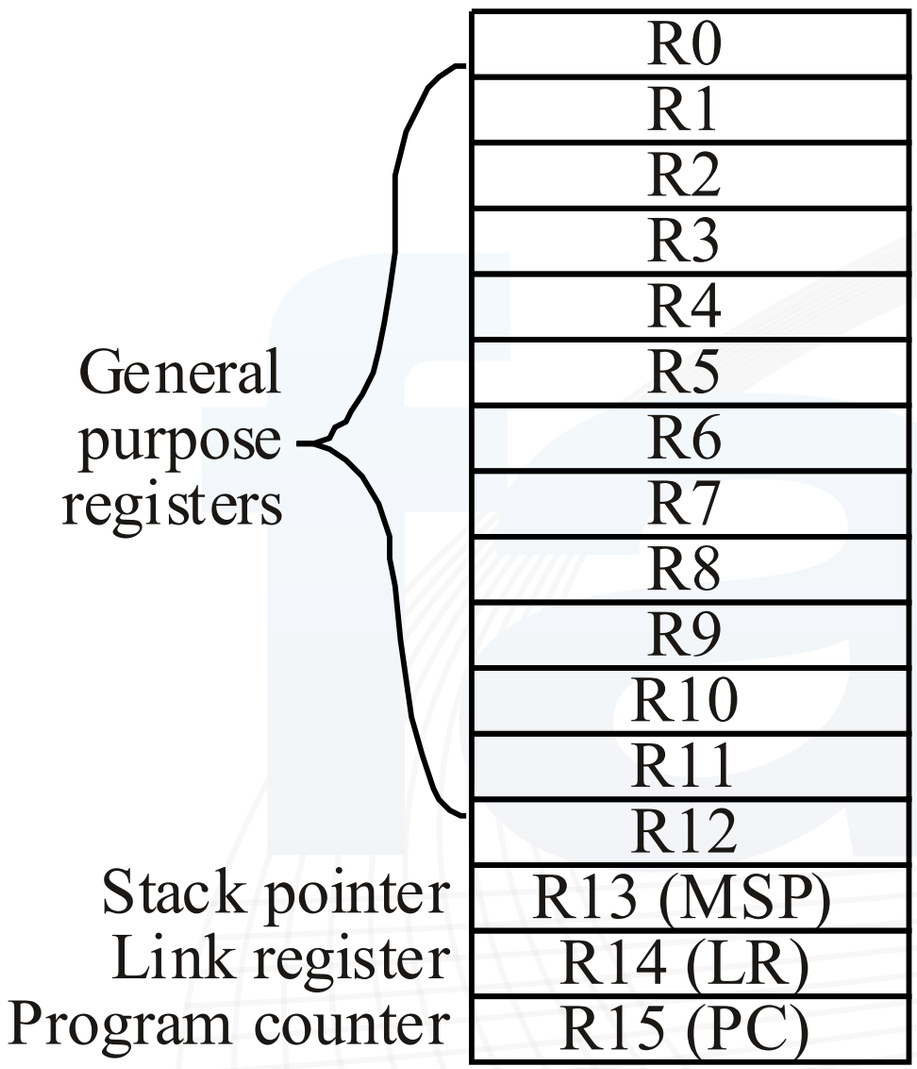
Sistema de Interrupciones

Electrónica IV

Mg.Ing. Esteban Volentini (evolentini@herrera.unt.edu.ar)

https://facetvirtual.facet.unt.edu.ar/course/view.php?id=165

ARM ISA: Registros



- 16 registros.
- 3 de uso especial
  - Más adelante.
- Pueden contener datos o punteros a Memoria
- Ancho: 32 bits.
- R15 es el PC.
- PSR – Program Status Reg
  - N, Z, C, V

Saltos:

- Toda instrucción que modifique el PC es un salto.
- Salto Incondicional:

  ```
  B label                    // PC ⇦ label
  ```

  destino = PC ± offset < ±16MB (adelante o atrás 16 MB).
- Salto Condicional (cc = condición):

  ```
  Bcc label      (Ej: BNE label)
  ```

  también relativo, destino < ±1MB
- Saltos con direccionamiento de registro indirecto

  ```
  BX Rn              //PC ⇦ Rn (El bit 0 de Rn = 1)
  ```

  absoluto, destino toda la memoria

  El bit cero de Rn debe ser 1 para indicar modo THUMB,

  para compatibilidad, sino error en ejecución.
- Se recomienda usar saltos y no usar PC como registro general. Leer set de instrucciones

Uso de la memoria

- El compilador divide la memoria de datos en tres fragmentos:
  - Static: área de datos estática utilizada para variables globales
  - Heap: área de datos dinámica utilizada con las primitivas **`malloc`** y **`free`**.
  - Stack: área de datos dinámica utilizada para variables locales y llamadas a procedimientos.

Pila / Stack (repaso)

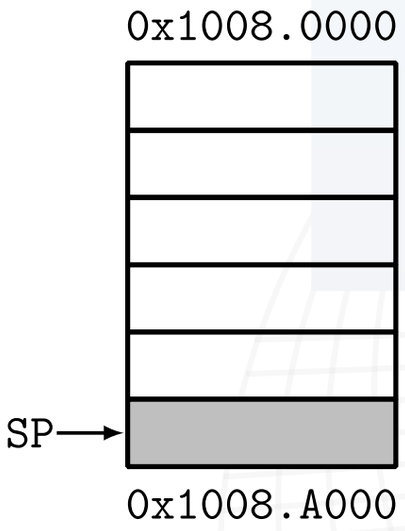
- Estructura de datos muy utilizada en general.
- ¿Ejemplos de pilas en el mundo real?
- Uso importante – sirve para explorar y saber cómo retornar (ejemplo: recorridos de árboles).
- ¿Cómo se implementa en Sw?

Implementación en ARM

- La memoria RAM hace de tabla.
- Emplea un puntero en CPU (SP=R13) para señalar ***el último lugar ocupado en la pila.***
- Instrucciones (PUSH y POP) para ingresar y retirar elementos de la pila.
- Cuando ingresamos un nuevo elemento en la pila el valor de SP se decrementa en cuatro (stack contiene palabras).
  - **Crece en el sentido de las direcciones decrecientes**.
- El puntero a la pila, SP, se inicializa cuando se hace RESET.
- Se puede inicilizar en otra parte con una instrucción LDR.
  - LDR R13,=direcc_pila

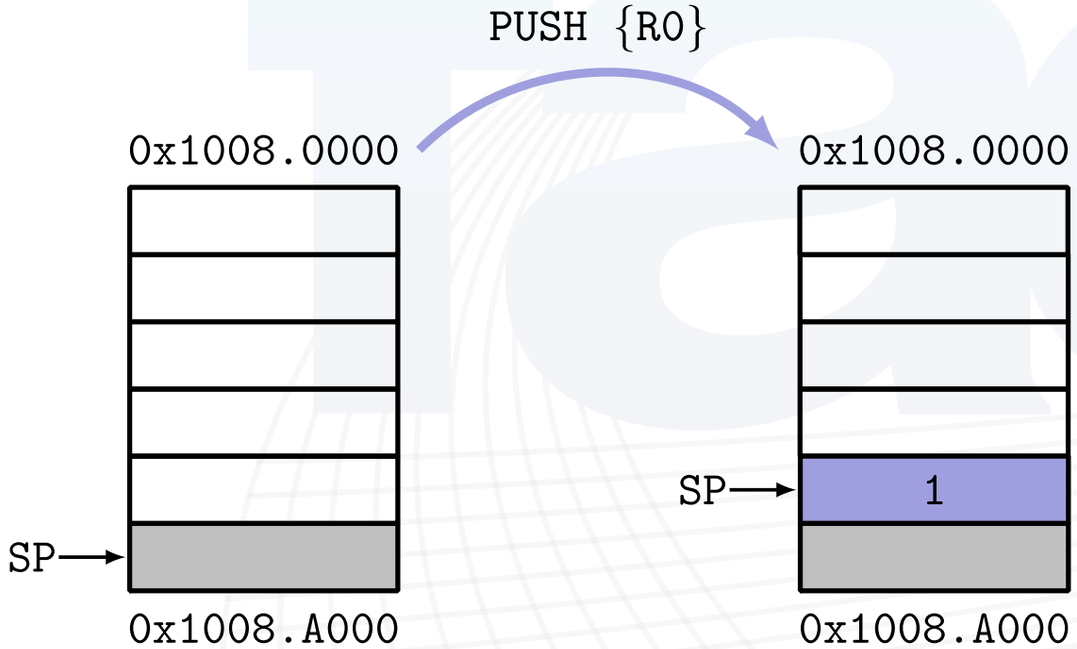
Ejemplo de uso del stack

- Supongamos **R0=1, R1=2, R2=3**



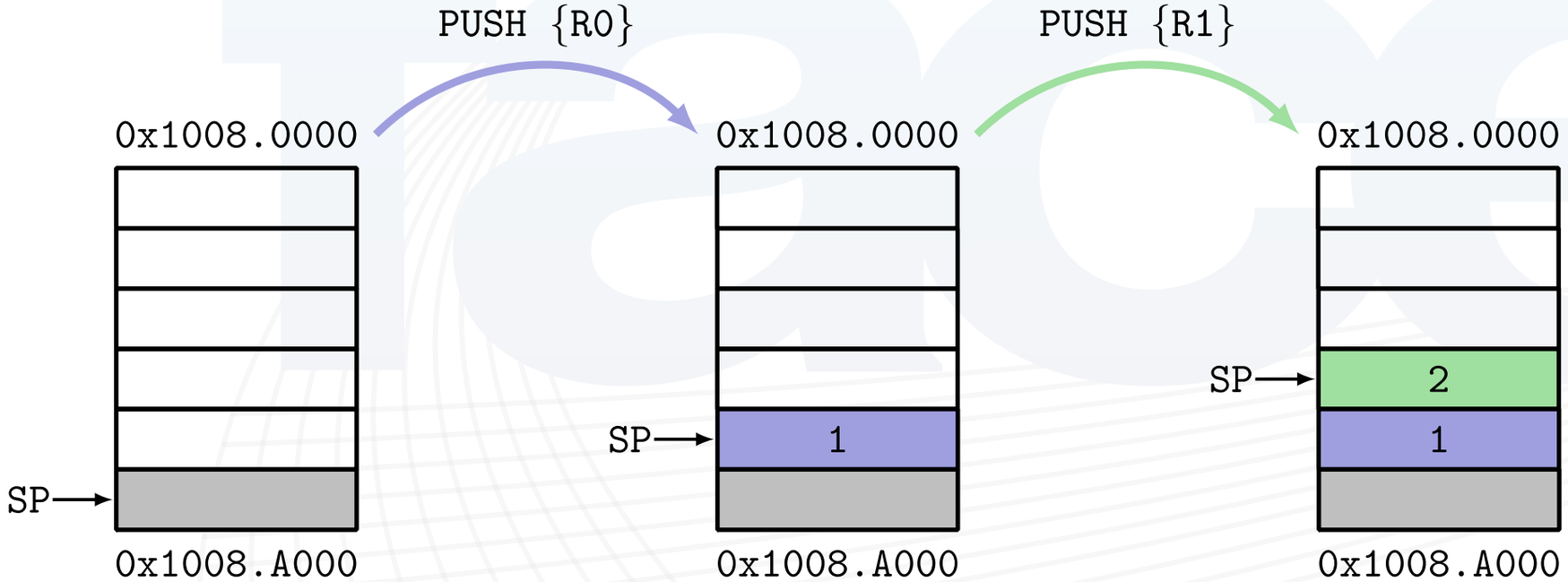
Ejemplo de uso del stack

- Supongamos **R0=1, R1=2, R2=3**
  - **PUSH {R2} // SP = SP+4, M(SP) = R0**



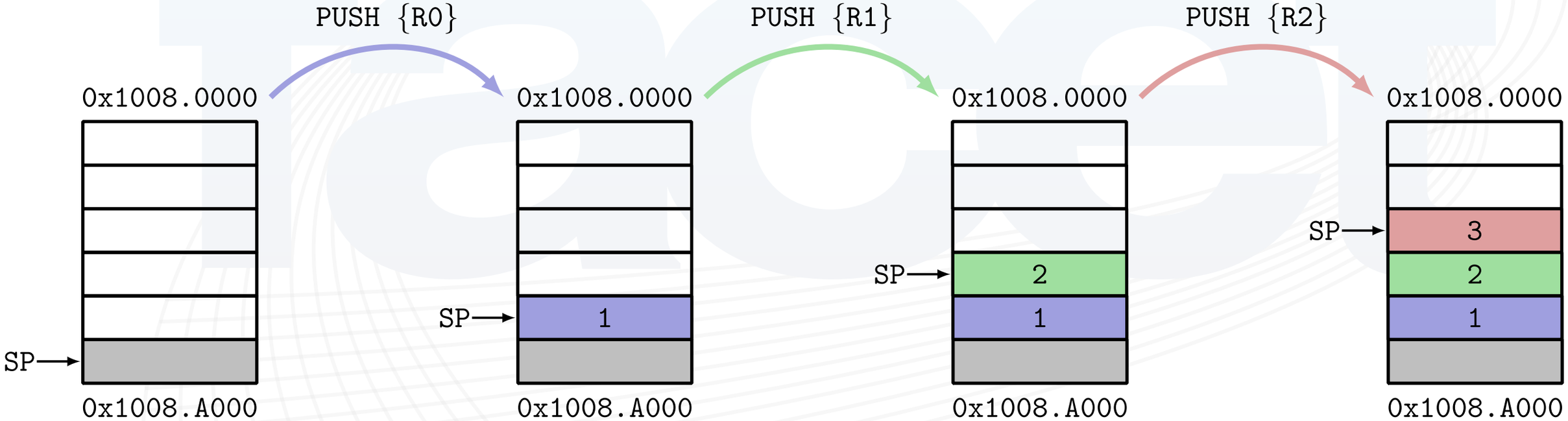
Ejemplo de uso del stack

- Supongamos **R0=1, R1=2, R2=3**
  - **PUSH {R2}** **// SP = SP+4, M(SP) = R0**
  - **PUSH {R1}** **// SP = SP+4, M(SP) = R1**



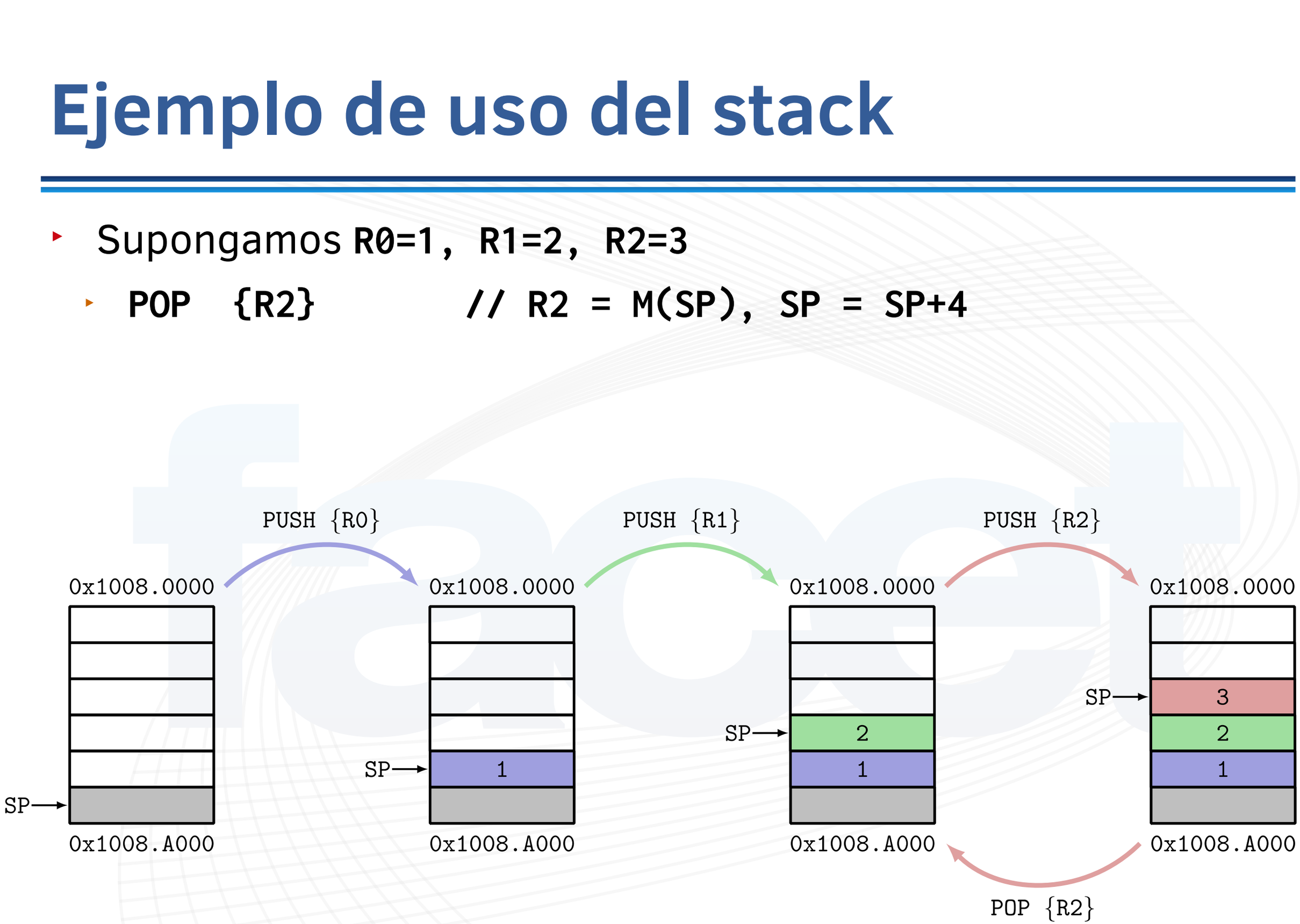
Ejemplo de uso del stack

- Supongamos **R0=1, R1=2, R2=3**
  - **PUSH {R2}** **// SP = SP+4, M(SP) = R0**
  - **PUSH {R1}** **// SP = SP+4, M(SP) = R1**
  - **PUSH {R0}** **// SP = SP+4, M(SP) = R2**



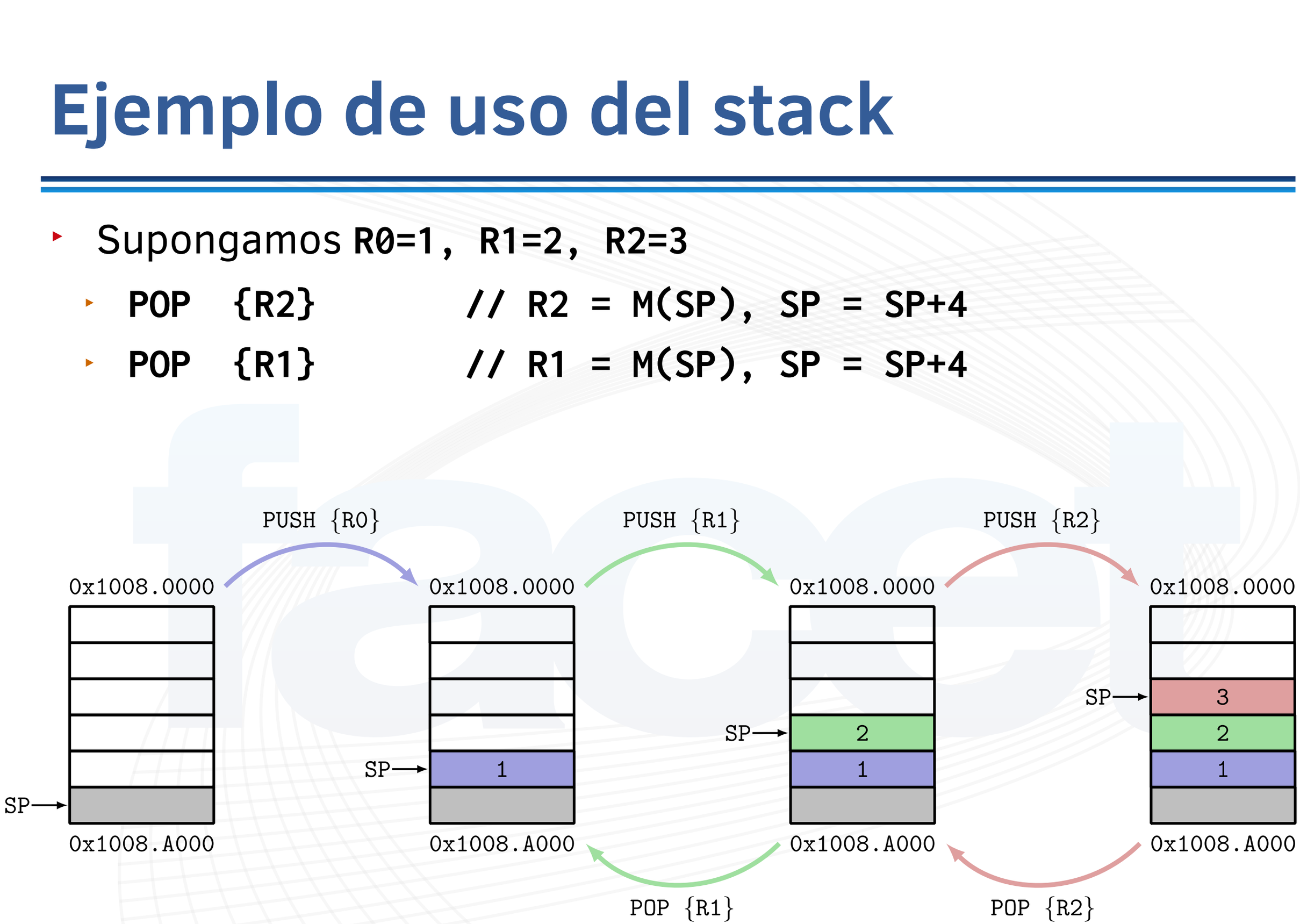
Ejemplo de uso del stack

- Supongamos **R0=1, R1=2, R2=3**
  - **POP {R2}** **// R2 = M(SP), SP = SP+4**



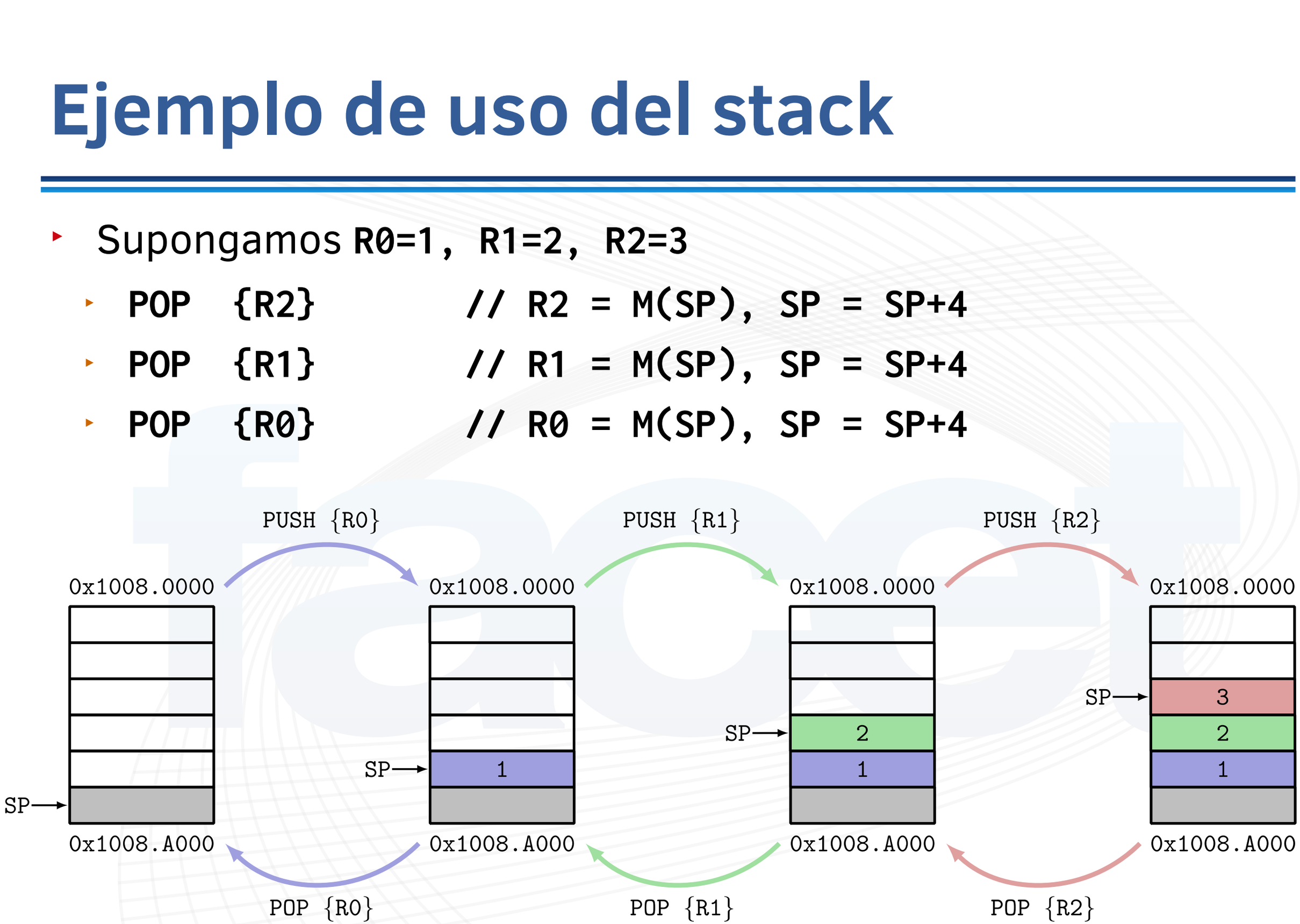
Ejemplo de uso del stack

- Supongamos **R0=1, R1=2, R2=3**
  - **POP {R2}** **// R2 = M(SP), SP = SP+4**
  - **POP {R1}** **// R1 = M(SP), SP = SP+4**



Ejemplo de uso del stack

- Supongamos **R0=1, R1=2, R2=3**
  - **POP {R2}** **// R2 = M(SP), SP = SP+4**
  - **POP {R1}** **// R1 = M(SP), SP = SP+4**
  - **POP {R0}** **// R0 = M(SP), SP = SP+4**



Subrutinas

- Generalmente los programas contienen bloques de código que se repiten.
- En estos bloques de código pueden variar algunos de los operandos (parámetros).
- Se puede ahorrar memoria (y tiempo de desarrollo) si estos bloques se escriben una vez y se ejecutan cada vez que se los requiere.
- Además estos bloques se podrían utilizar en otros programas.

Requerimientos

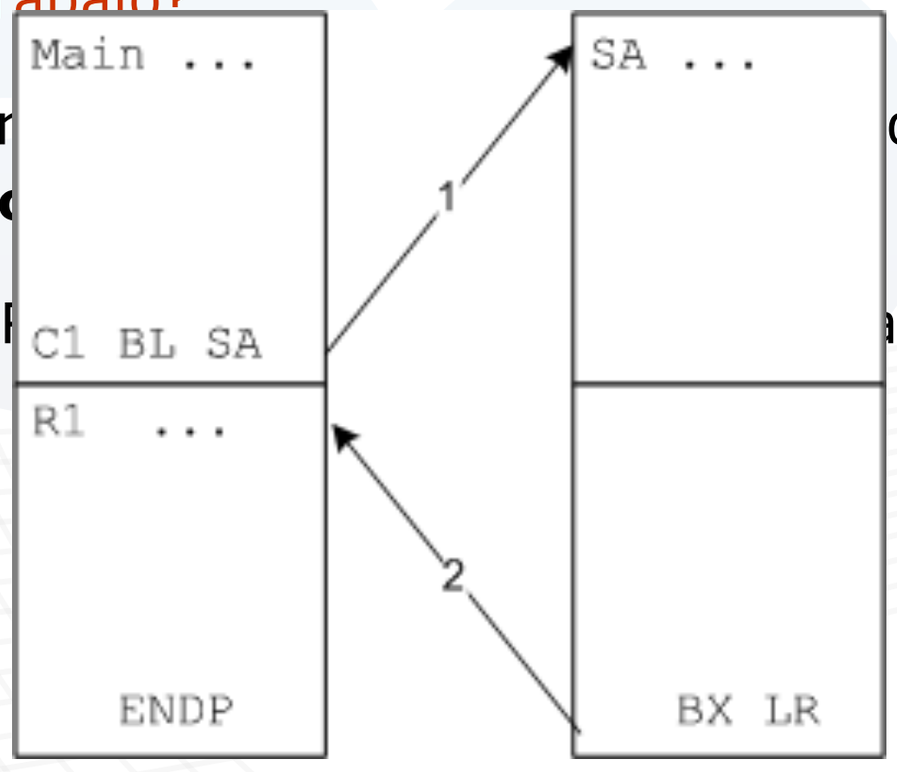
- Se las convoca de distintos lugares
- Deben saber de dónde fueron llamadas para retornar a la instrucción siguiente al llamado
  - Un simple salto no es suficiente…
- Debe haber un mecanismo para pasar y devolver parámetros

Implementación en ARM

- Instrucción de llamada: Branch and Link (BL)
  - **`BL etiqueta → LR = PC + 4, PC = etiqueta`**
  - Guarda PC+4 en registro LR (R14), “link register”
  - El salto es relativo al valor actual del PC
  - El alcance del destino es ±16MBytes
- También llamada indexada.
  - **`BLX Rn → LR = PC+4, PC = Rn`**
- Instrucción de Retorno
  - **`BX LR → PC = LR`**
  - Si al entrar en la Subrutina se hace PUSH LR, la instrucción POP PC también permite retornar.

Ejemplo

- ¿Esta implementación permite a un programa convocar a otra subrutina para hacer parte del trabajo?
- Si, en la ejecución[illegible]ción LR=PC="R1" y PC="SA". **La subrutina no[illegible]**
- Con BX LR, PC=LR[illegible]r del llamado a SA

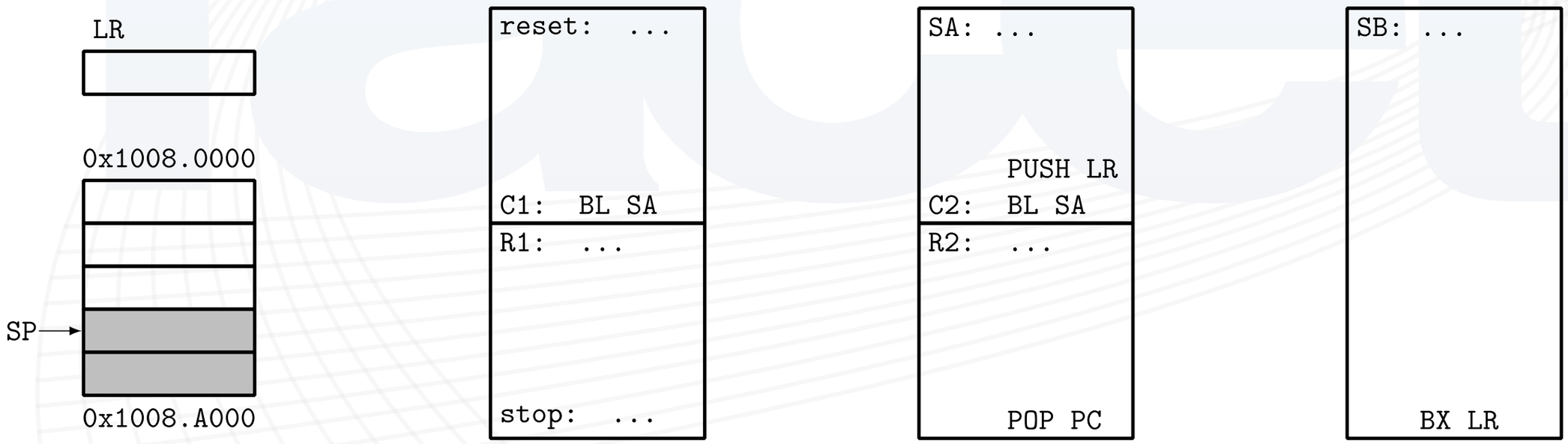


Subrutinas Anidadas

- ¿Esta implementación permite a una subrutina convocar a otra subrutina para hacer parte del trabajo?
- No, se pierde el contenido de LR en la llamada anidada.
- Para resolverlo la subrutina debe guardar LR en stack, PUSH LR.
- Cuando esta subrutina retorna a quien la llamó debe recuperar LR mediante POP LR y ejecutar luego BX LR.
- Una alternativa más corta a lo anterior es usar POP PC (entonces si LR estaba en Stack pasa directamente a PC y se retorna).
- Veamos cómo funciona:

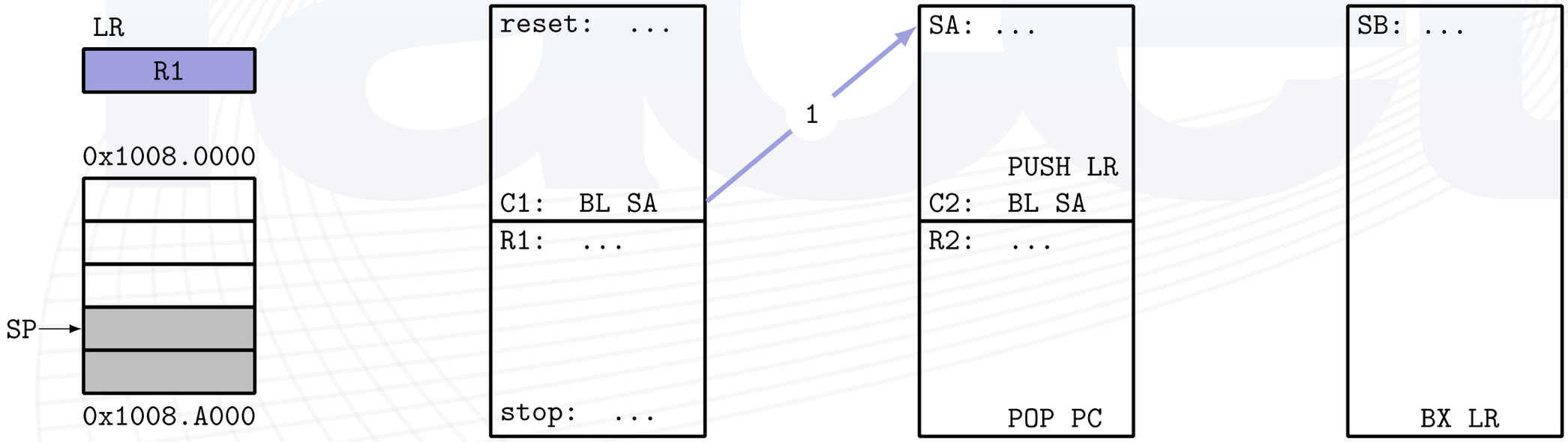
Subrutinas Anidadas

- ¿Esta implementación permite a una subrutina convocar a otra subrutina para hacer parte del trabajo?
- Si, el mecanismo de Stack lo permite sin problemas.



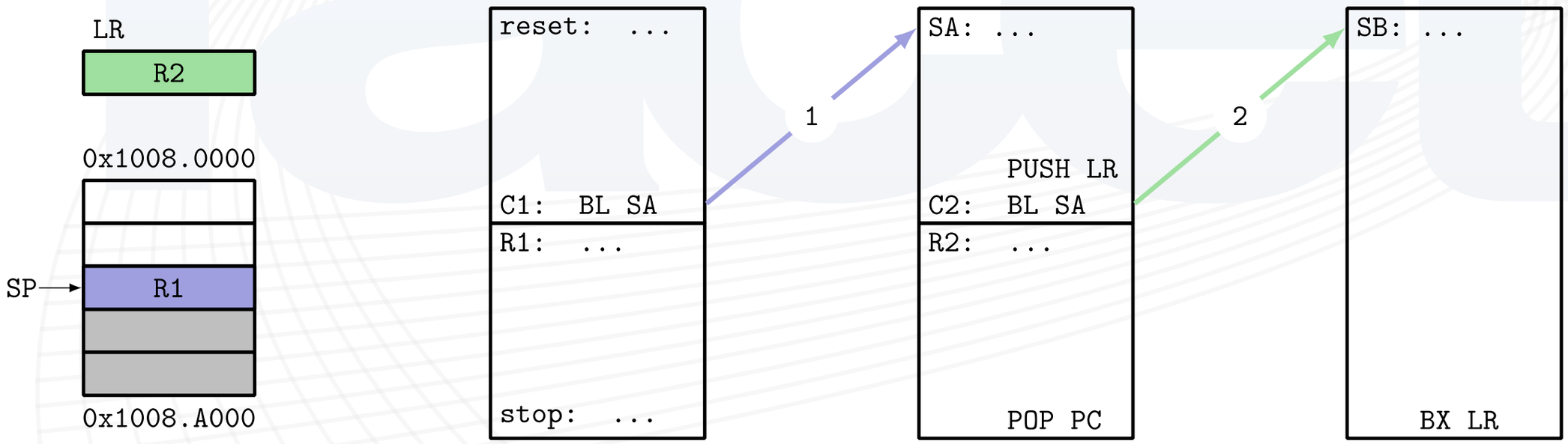
Subrutinas Anidadas

- ¿Esta implementación permite a una subrutina convocar a otra subrutina para hacer parte del trabajo?
- Si, el mecanismo de Stack lo permite sin problemas.



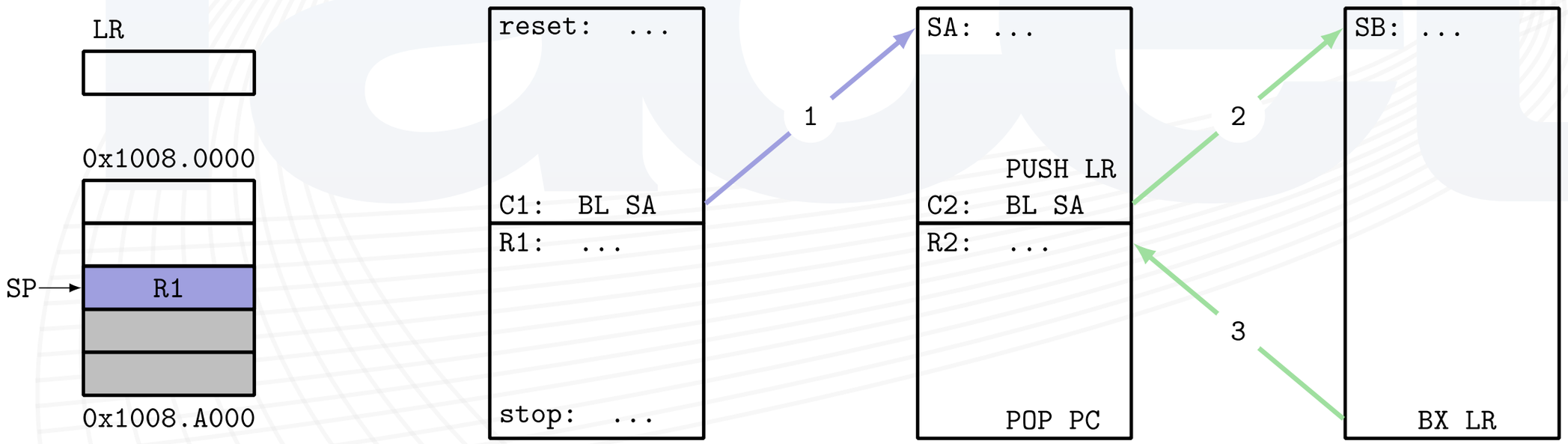
Subrutinas Anidadas

- ¿Esta implementación permite a una subrutina convocar a otra subrutina para hacer parte del trabajo?
- Si, el mecanismo de Stack lo permite sin problemas.



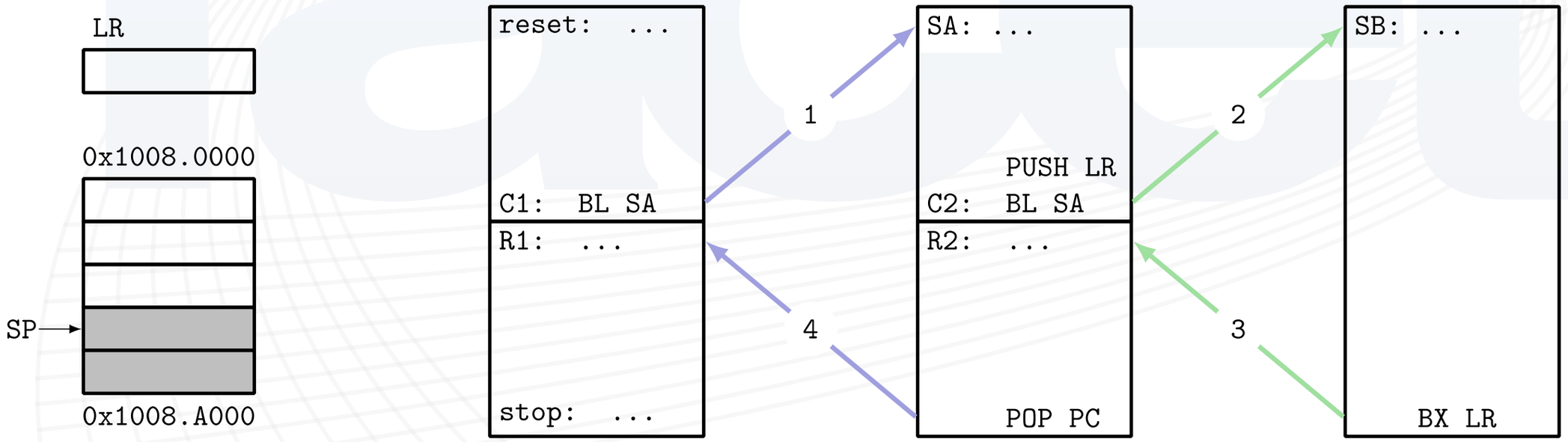
Subrutinas Anidadas

- ¿Esta implementación permite a una subrutina convocar a otra subrutina para hacer parte del trabajo?
- Si, el mecanismo de Stack lo permite sin problemas



Subrutinas Anidadas

- ¿Esta implementación permite a una subrutina convocar a otra subrutina para hacer parte del trabajo?
- Si, el mecanismo de Stack lo permite sin problemas.
- ¿Qué pasa si en las rutinas se modifica el valor de retorno o el valor de SP?



Reglas para el uso del stack

- Se debe hacer uso balanceado del stack, es decir debe haber el mismo número de push y pop
- Los accesos push o pop no deben realizarse fuera del área asignada de antemano por el diseñador
- Ese área debe estar en Memoria RAM
  - No superponerse con otros datos
  - No superponerse con FLASH
  - No superponerse con partes no ocupadas

ARM Arch. Procedure Call Standard (AAPCS)

- Convención ABI para todas las arquitecturas ARM
- Registros R0, R1, R2, y R3 para pasar los primeros cuatro parámetros de entrada, el resto en la pila
- ABI obliga a poner el parámetro de retorno en R0
- Las funciones son libres para modificar R0–R3 y R12 es reservado para uso del compilador o ensamblador
- Si una función necesita R4-R11, deberá guardar primero los valores de los registros a usar en la pila, y antes de retornar, debe recuperar los viejos valores del la pila
- Por la convención ABI, el primer parámetro se pasa en Registro R0, el segundo en R1 y así...

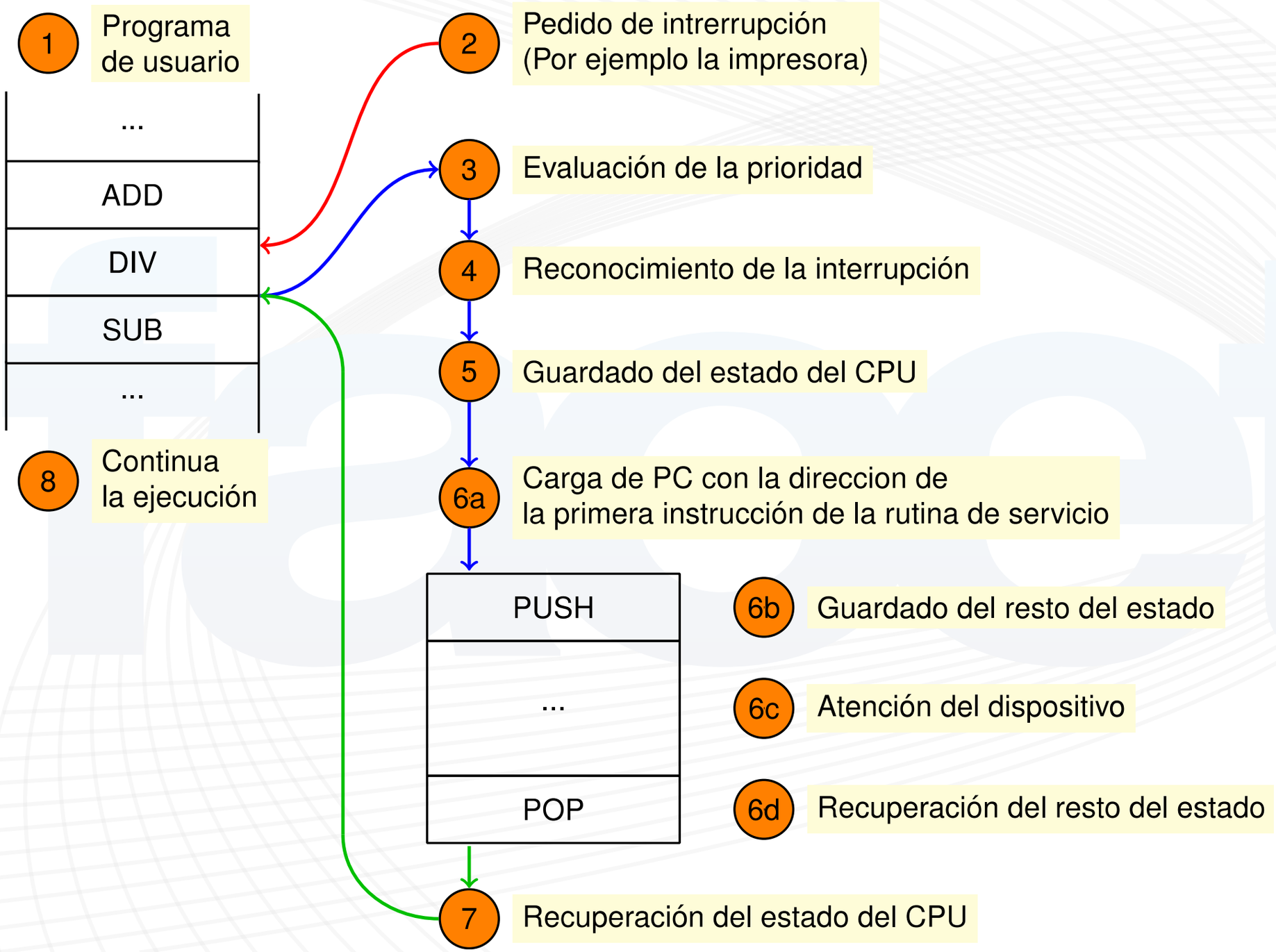
Interrupciones en la vida real

1) Un estudiante está estudiando (Ejecuta una tarea)
2) Lo llaman por teléfono (Pedido de Interrupción)
3) Decide si atenderá la llamada (Consulta de Prioridades)
4) Indica que sí atenderá (Reconocimiento de la Interrupción)
5) Pone un señalador en su libro (Guarda el estado de su tarea)
6) Atiende el teléfono (Sirve a la interrupción)
7) Abre el libro en el señalador (Recupera el estado de su tarea)
8) Retoma el estudio (Continua la tarea original)

Interrupciones en un procesador

1) El procesador ejecuta un programa (Ejecuta una tarea)
2) Un puerto de comunicaciones activa /IRQ porque recibió datos (Pedido de Interrupción)
3) Prioridad del Programa < Prioridad de la Interrupción (Consulta de Prioridades - suponemos que sí)
4) Señal de salida INTA (Reconocimiento de la Interrupción)
5) Guarda el valor actual del PC y ciertos registros (Guarda el estado de la tarea actual)
6) Atiende la interrupcion
   - Carga el PC con la dirección de la rutina de servicio (Sirve a la interrupción)
   - La rutina de servicio guarda el resto del estado al comenzar
   - Ejecuta los pasos necesarios para completar la nueva tarea
   - La rutina de servicio recupera el estado guardado al comienzo
7) Al terminar recupera el valor del PC y ciertos registros (estado de su tarea)
8) Continua la ejecución del programa (Continua la tarea original)

Punto de Vista del Programador



Características de las INT

- Interrupciones
  - Causada por eventos externos
  - Asíncronas con la ejecución del programa.
  - Reclaman un servicio, ISR (S.O. en general)
- Transparentes al Programa Principal
  - No tiene forma de saber cuándo llega.
  - No puede prepararse para ello.
  - ¿Cómo sabe que llegó una interrupción?

Pregunta Importante

- ¿Qué diferencias hay entre interrupciones y subrutinas?
- Las interrupciones son ASINCRONICAS.
- ¿Cómo sabe en donde está la rutina de interrupción?
- Mediante el Num de INT accede a una tabla de direcciones

Excepciones

- Interrupciones: Causada por eventos externos
  - Asíncronas con la ejecución del programa
  - Reclaman un servicio
  - Se ejecutan entre instrucciones del programa
    - Simple: suspenda y continúe el programa del usuario
- Traps: Causados por eventos internos
  - Condiciones excepcionales o de error.
  - Son sincrónicas con la ejecución del programa.
  - El problema debe ser remediado por un supervisor.
  - Si no se puede, se aborta el programa

Simultáneas

- Son simultáneas las que ocurren en el intervalo de una instrucción
- También son simultáneas las que quedaron sin atender de una ronda previa.
- Se resuelve por prioridades.
- El sistema de prioridades debería ser equitativo.

Anidamiento

- Hablo por teléfono y suena el timbre...
- Define el Diseñador.
  - Usar solo en caso necesario, es complejo.
  - En general se las puede enmascarar.
- Si Admite, entonces
  - El Hw permite solo con mayor prioridad.

NVIC - Visión Lógica

- Hasta 255 excepciones
  - Hasta 240 interrupciones
  - En LPC43xx hasta 53 interrupciones
- Cada excepción se identifica con un número
  - Núm. de Interrupción = Núm. de Excepción – 16.
  - Hay vector (o tabla) con la dirección de inicio de cada rutina de servicio
  - Se busca en la entrada correspondiente al Número de Excepción

Condiciones de Interrupción - I

- deben cumplirse cuatro condiciones necesarias simultáneamente:
  - Habilitado en el CPU: I=0 in PRIMASK.
  - Habilitado en el dispositivo: el bit del Hw=1
  - Nivel Prioridad: Nivel de IRQ “menor” que BASEPRI
  - Pedido: Una acción externa de Hw setea la bandera de pedido.

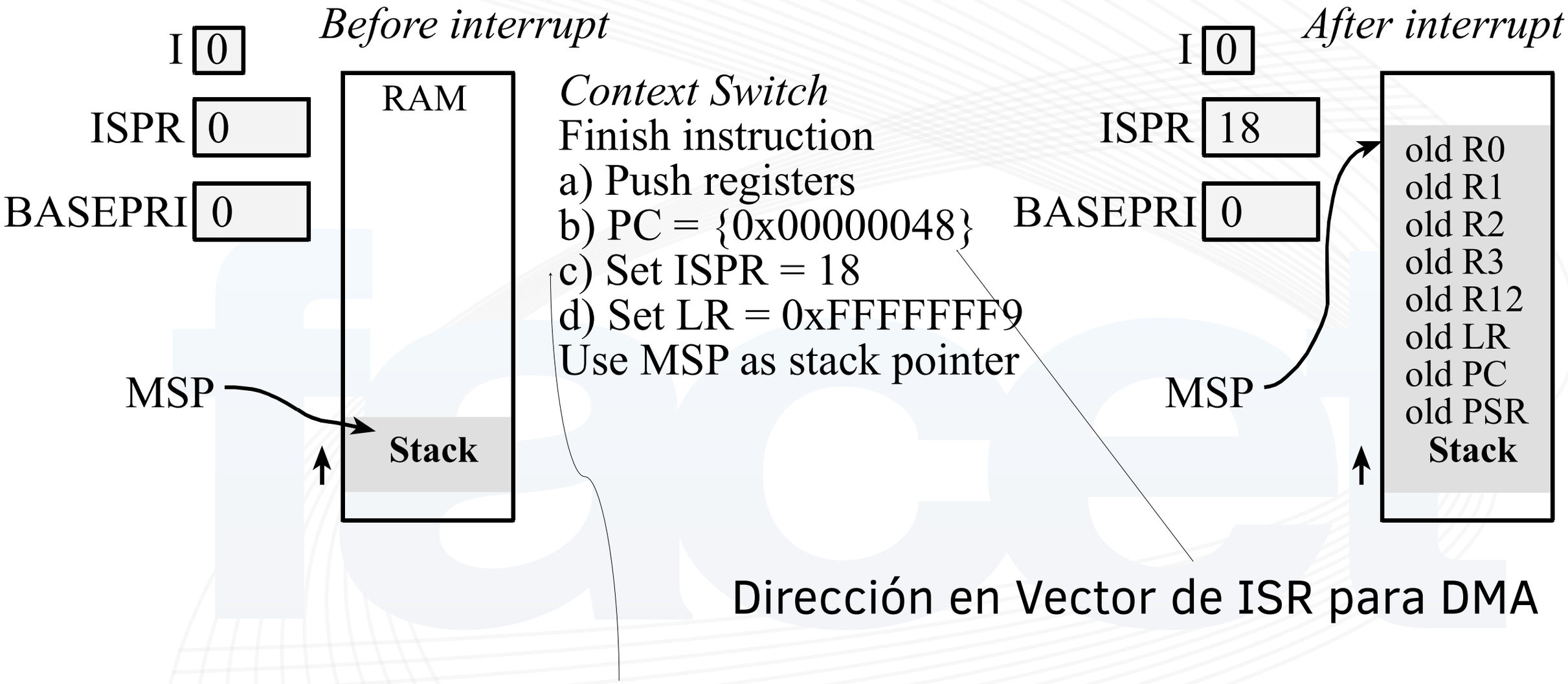
Procesamiento de la INT - ISR

- El Hw realiza lo siguiente:
  1. Esperar fin de la instrucción actual.
  2. Suspender ejecución y push de 8 registros (R0-R3, R12, LR, PC, PSR) en stack.
  3. LR=0xFFFFFFF9 (indica que hubo una Interrupción)
  4. Escribir en PSR el número de interrupción.
  5. PC ← Dirección inicial de ISR.

Ejecución de la ISR

- Acknowledge: Pone a cero el flag que disparó la interrupción.
- Salva registros no guardados automáticamente si los usa, en Stack.
- Servicio a la interrupción.
- Restaura Registros del punto (2)
- Se retorna del Prog. Principal con BX LR

Ej: Cambio de Contexto (INT#18)



Es fundamental que todo quede como antes de la interrupción!!!

M4: Excepciones del Núcleo

- Tres con Prioridades Fijas, el resto configurable
  - Reset: varias causas (POR, BOD, …), prioridad=-3
  - NMI: Solo Reset tiene más prioridad. Prioridad=-2
  - Hard Fault: Falla de Hw no especificada. Prioridad=-1
- Faults – además de Hard Fault:
  - Memory Management: Violación de Protección de M (MPU)
  - Bus Fault: Falla en el Bus.
  - Usage Fault: Falla en programa: instrucción no definida, división por cero, etc.
- SVCall: Pedidos del programa al SO, serv. privilegiados.
- SysTick Interrupt: Pedida por un timer interno del núcleo del ARM. Veremos más adelante.

NVIC Niveles de prioridades

- Prioridades numéricas menores son “más altas”.
- Se define el concepto de grupo y subgrupo de prioridad.
- Se permite el anidamiento de excepciones con diferentes grupos de prioridad
- El subgrupo permite definir entre interrupciones simultáneas del mismo grupo.
- Si dos interrupciones tienen el mismo grupo y subgrupo entonces se define por el numero de excepción.

NVIC Niveles de prioridades

- La prioridad se configura con un registro de ocho bits que se puede partir en dos campos:
  - La parte más significativa define el grupo de prioridad.
  - El resto define el subgrupo de prioridad.
- La división en grupo y subgrupo se puede configurar
  - Se pueden asignar de 0 a 7 bits para el grupo
  - El resto corresponde al subgrupo
  - Máximo 128 niveles de grupo y hasta 256 niveles de subgrupo.
- Los registros de prioridad se pueden fabricar con menos bits
  - El fabricante puede usar de 3 a 8 bits para cada registro
  - El procesador NXP4337 tiene implementado solo 3 bits
  - En este caso los bits no implementados se consideran siempre como cero.

NVIC: pendientes y activos

- Una interrupción puede anidar a otra.
- Si no puede, pone bit pendiente=1 en un registro que vale para 32 IRQs.
- Cuando se ejecuta, pone pendiente=0, activo=1.
- Si anida, la que se interrumpió sigue con su bit activo en 1.
- Ojo. Si I=0 o no habilitada en NVIC, y llega el pedido, Pend=1!!

Modos: Usuario / Supervisor

- Nivel de Privilegio: Supervisor/Usuario
- En modo Supervisor
  - Todas las instrucciones permitidas.
  - Todos los dispositivos y áreas de M.
- En modo Usuario
  - No se permite acceder a los dispositivos.
  - No se permite acceder a algunas áreas.
  - Para asegurar que un usuario no provoque problemas. ¿Cuáles?

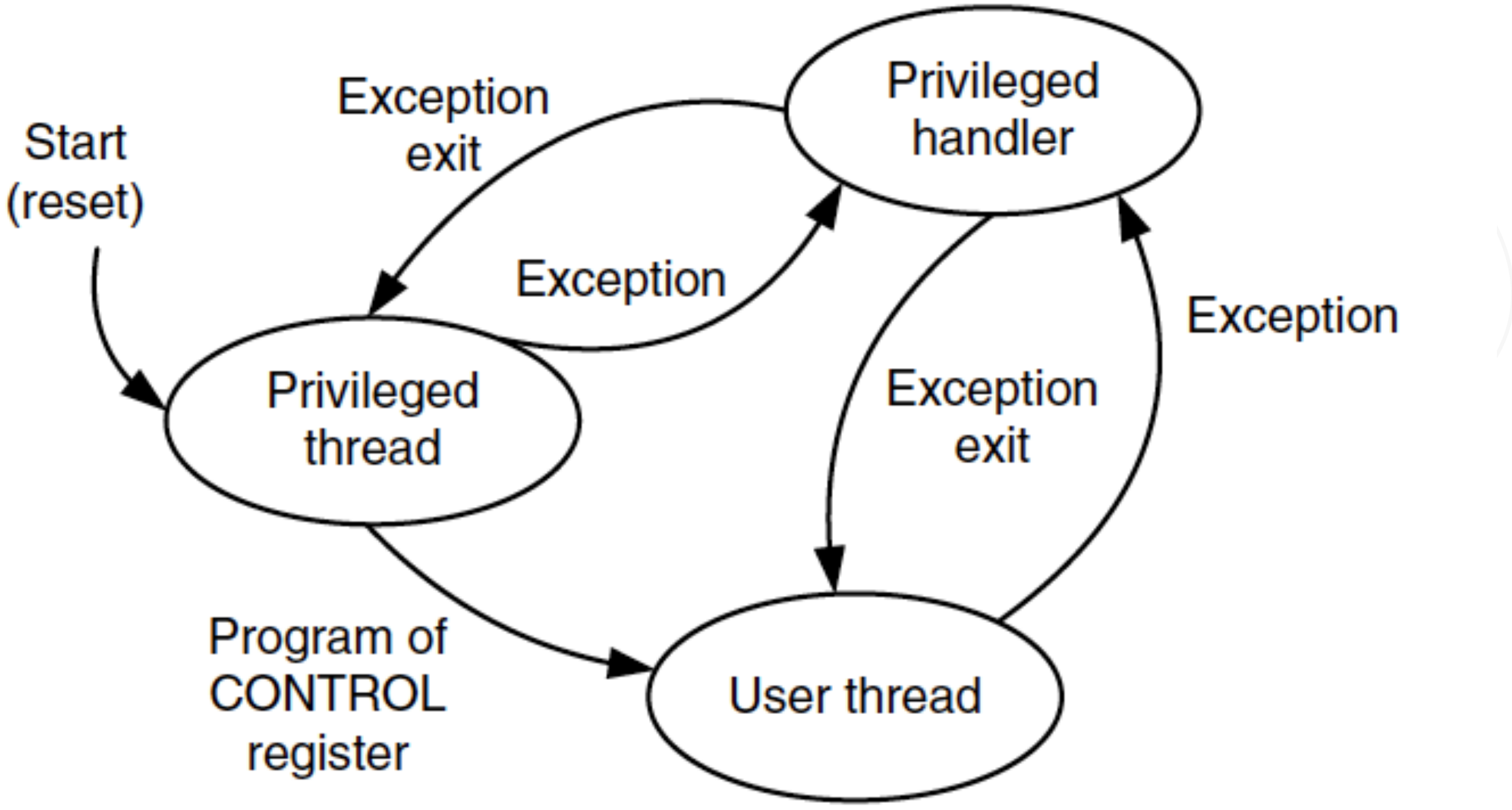
Modos: Thread/Handler

- Implementación ARM-M4
- Modo Thread
  - Cuando un programa está corriendo.
- Modo Handler
  - Cuando una interrupción se ejecuta.

Modos del Procesador

- Solo en nivel Supervisor.
- Modo Supervisor Thread.
  - Corre un programa, por ej. SO.
  - Así arranca con el reset.
  - Trabajaremos en este modo, ya que somos desarrolladores.
- Modo Handler.
  - Para respuesta a Excepciones.
    - Interrupción se considera como excepción.

Operación: Modos y Privilegio



Implementación Cortex-M4

- Las ISR siempre en nivel supervisor. Se les llama “handler” porque en general manejan un dispositivo (modo handler).
- El programa puede estar en nivel Supervisor o Usuario.
  - En general en nivel usuario.
    - En este caso tiene un stack distinto PSP (puntero)
    - El supervisor tiene MSP (puntero). ¿Por qué?
  - ¿Cómo corre el SO?
    - En nivel supervisor.

Detalle Anidamiento

- Para retornar de ISR siempre BX LR
- LR se carga automáticamente
  - FFFFFFF9 desde Supervisor Thread.
  - FFFFFFF1 desde Handler.
  - FFFFFFFD desde USER Thread – PSP
- Dos registros de pila, intercambiables
  - PSP – pila para usuario.
  - MSP – pila para supervisor.
  - Trabajaremos desde Supervisor Thread **¿por qué?**

Supervisor Call (SVC)

- Similar a una Interrupción, pero provocada por una Instrucción
- En realidad es una excepción
- Salta a una rutina de Excepción, de acuerdo a un vector propio
- Se usa para pedir servicios al Sistema Operativo desde “thread usuario”