

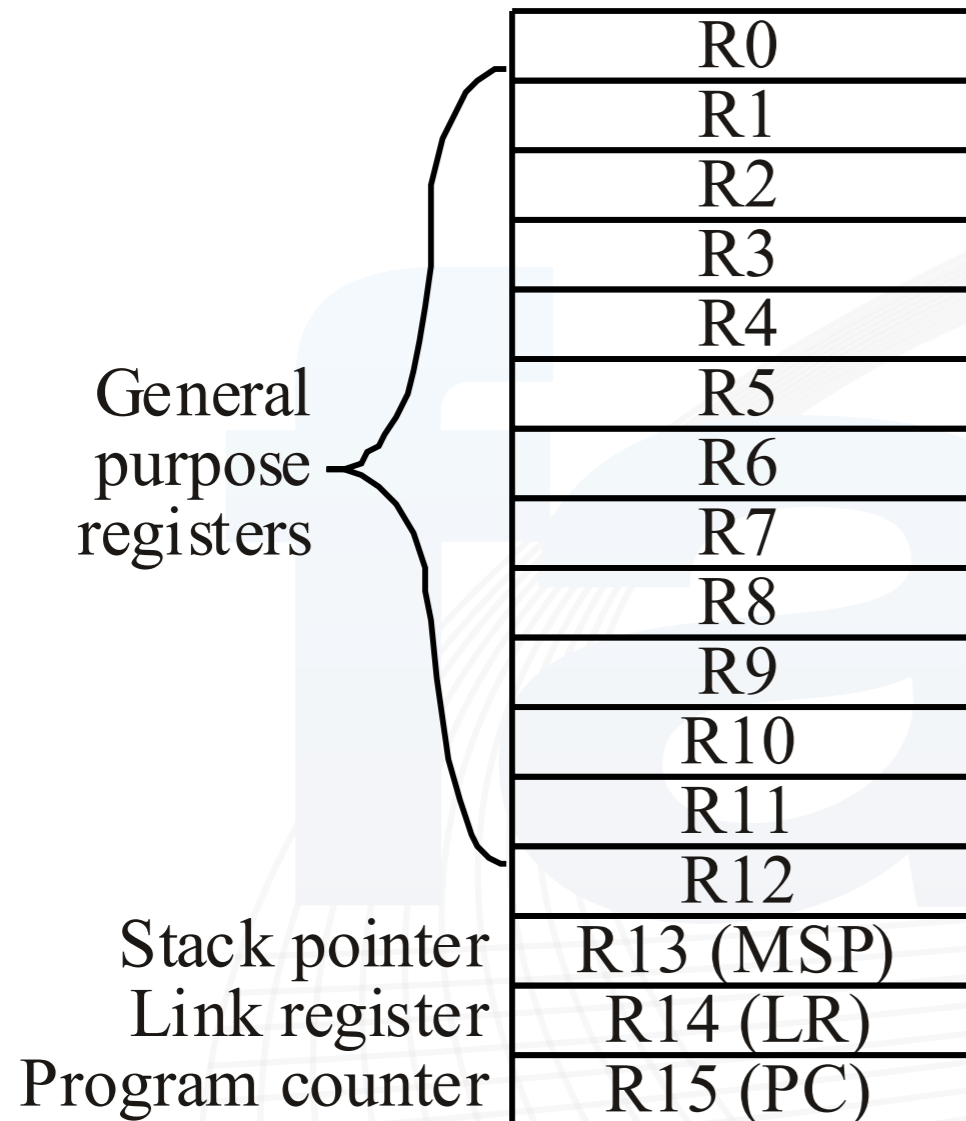
Interrupciones y Excepciones

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

Salto:

- ▶ Toda instrucción que modifique el PC es un salto.

- ▶ Salto Incondicional:

B label // PC \Leftarrow label

destino = PC \pm offset < \pm 16MB (adelante o atrás 16 MB).

- ▶ Salto Condicional (cc = condición):

Bcc label (Ej: **BNE label**)

también relativo, destino < \pm 1MB

- ▶ Saltos con direccionamiento de registro indirecto

BX Rn // PC \Leftarrow 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)

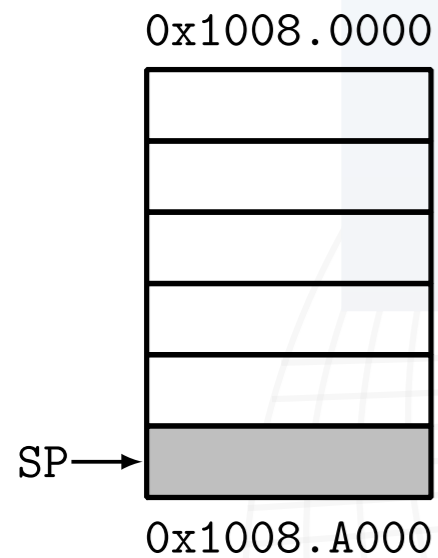
- ▶ Estructura de datos muy utilizada en general.
- ▶ ¿Ejemplos de pilas en el mundo real?
- ▶ Uso importante – sirve para explorar y saber cómo retornar (ejemplo: recorridos de árboles).
- ▶ ¿Cómo se implementa en Sw?

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 inicializar en otra parte con una instrucción LDR.
 - ▶ LDR R13,=direcc_pila

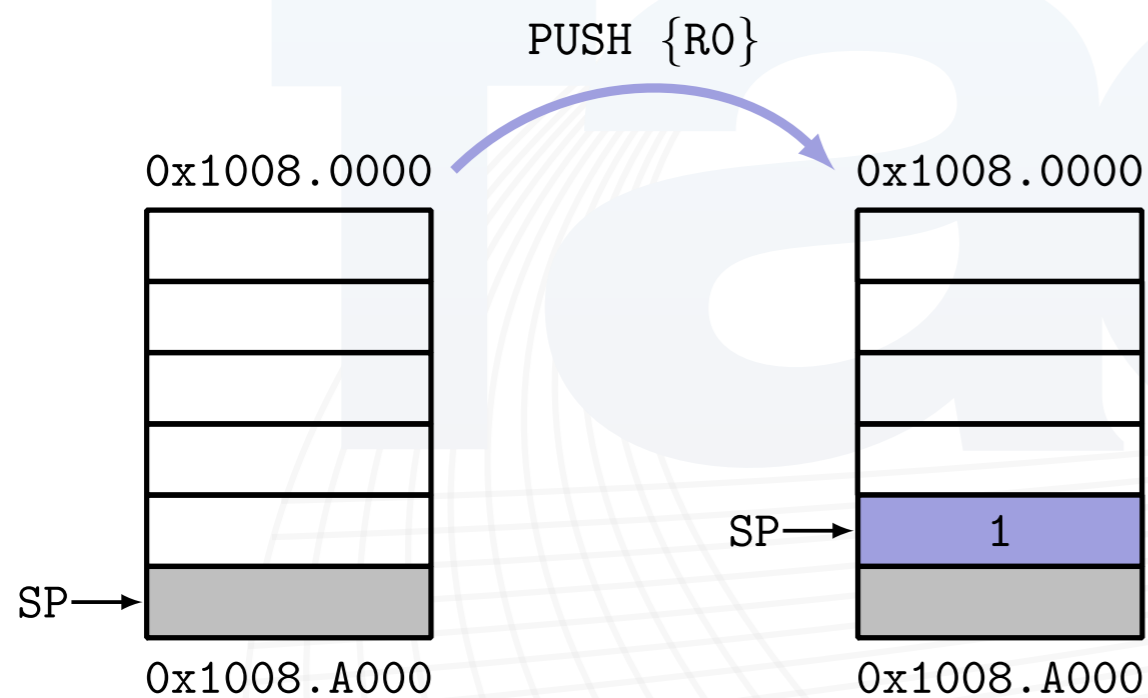
Ejemplo de uso del stack

- ▶ Supongamos $R0=1$, $R1=2$, $R2=3$



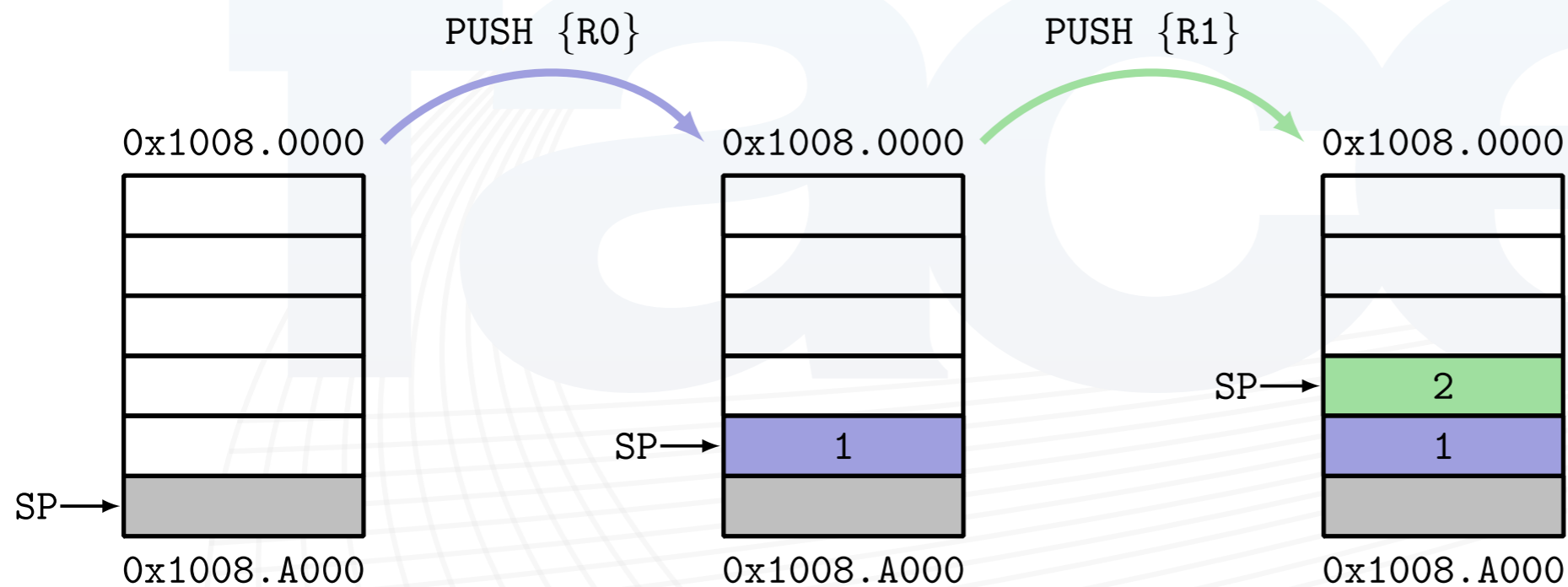
Ejemplo de uso del stack

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



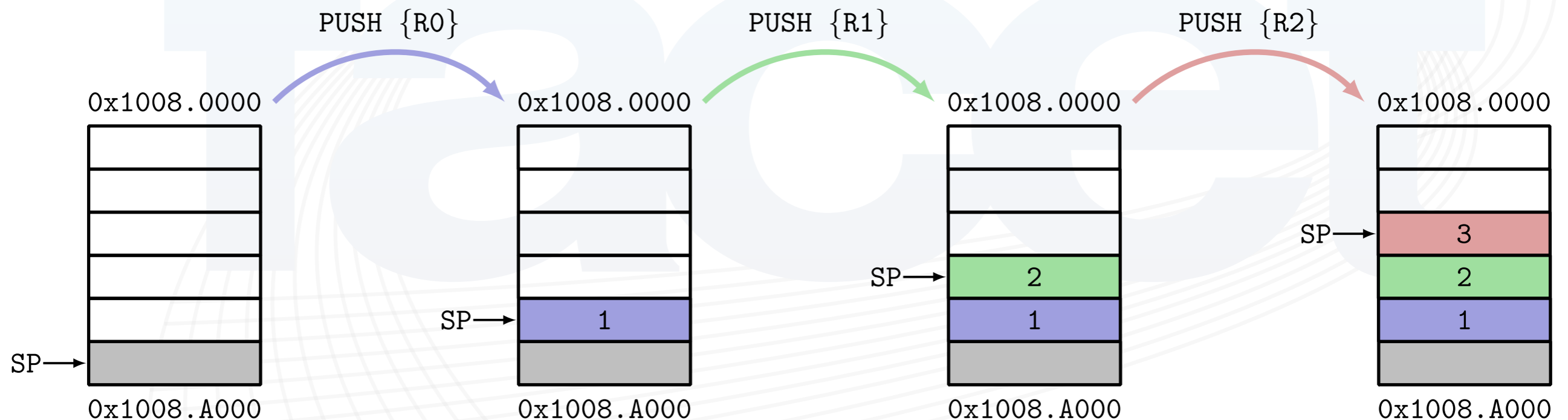
Ejemplo de uso del stack

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



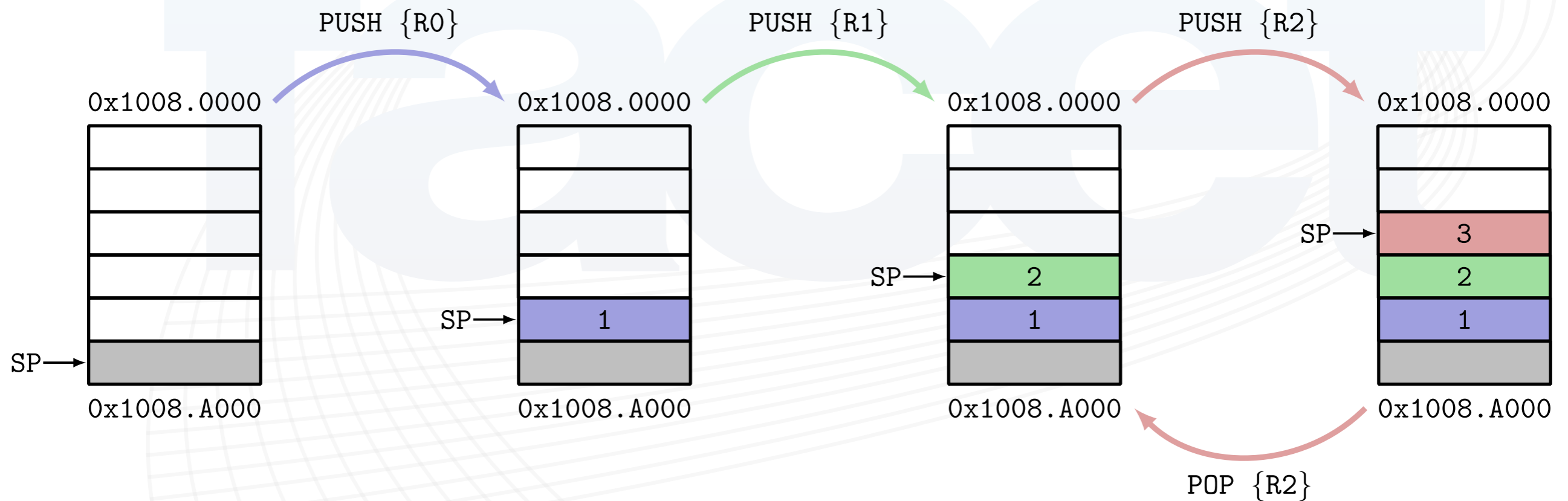
Ejemplo de uso del stack

- ▶ 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}` // $SP = SP+4$, $M(SP) = R2$



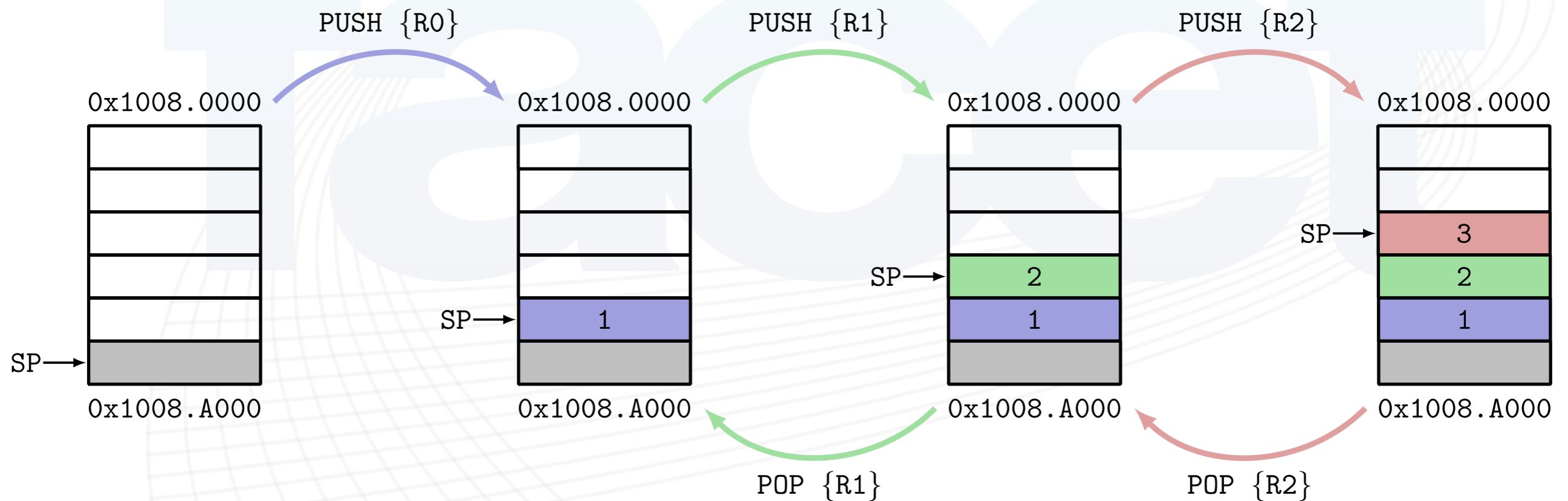
Ejemplo de uso del stack

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



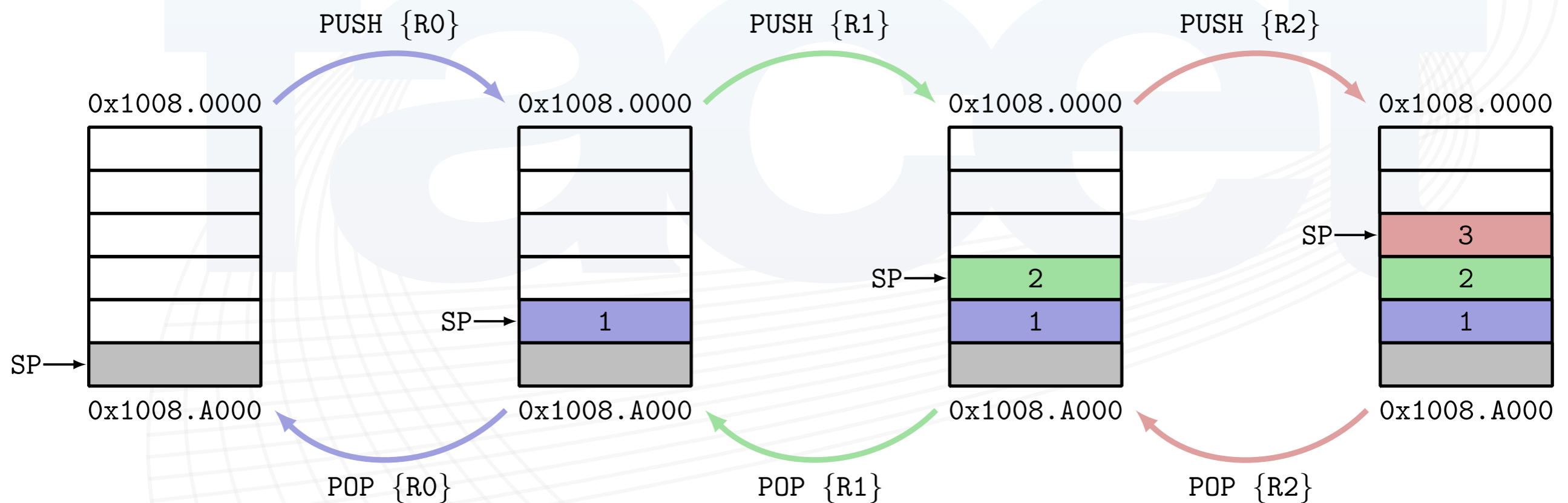
Ejemplo de uso del stack

- ▶ 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

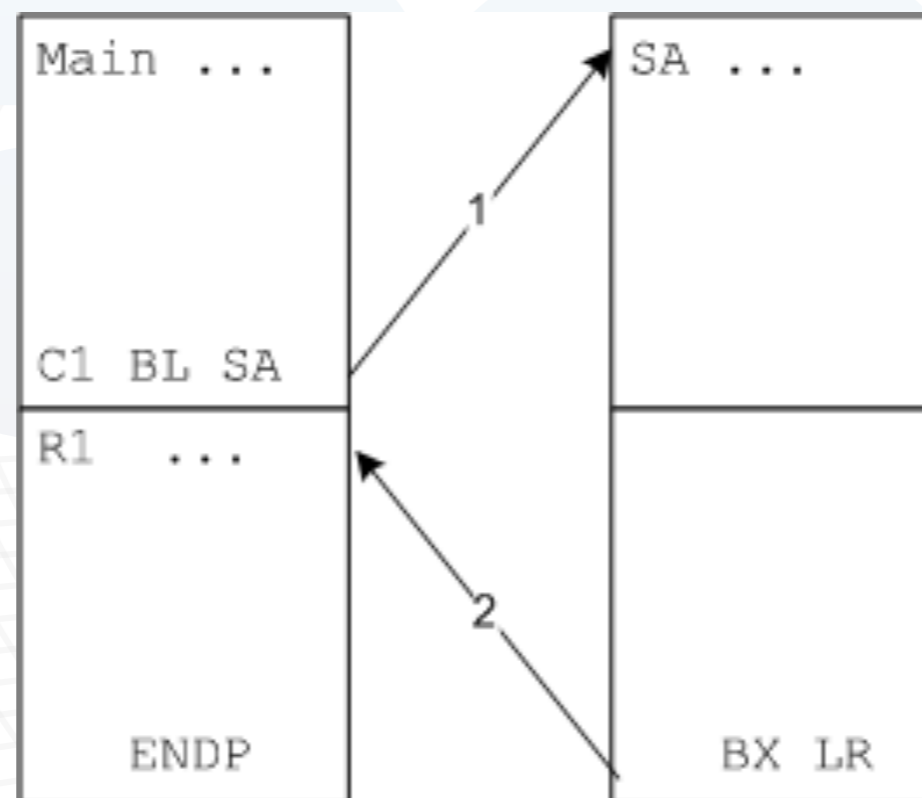
- ▶ Se las convoca de distintos lugares
- ▶ Deben saber de dónde fueron llamadas para retornar a la instrucción siguiente al llamado
 - ▶ Un simple salto no es suficiente...
- ▶ Debe haber un mecanismo para pasar y devolver parámetros

Implementación en ARM

- ▶ Instrucción de llamada: Branch and Link (BL)
 - ▶ BL etiqueta \rightarrow 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 ± 16 MBytes
- ▶ También llamada indexada.
 - ▶ BLX Rn \rightarrow LR = PC+4, PC = Rn
- ▶ Instrucción de Retorno
 - ▶ BX LR \rightarrow 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 de BL SA, PC="R1". A continuación LR=PC="R1" y PC="SA". **La subrutina no debe modificar LR.**
- ▶ Con BX LR, PC=LR="R1" y retorna al siguiente luar del llamado a SA

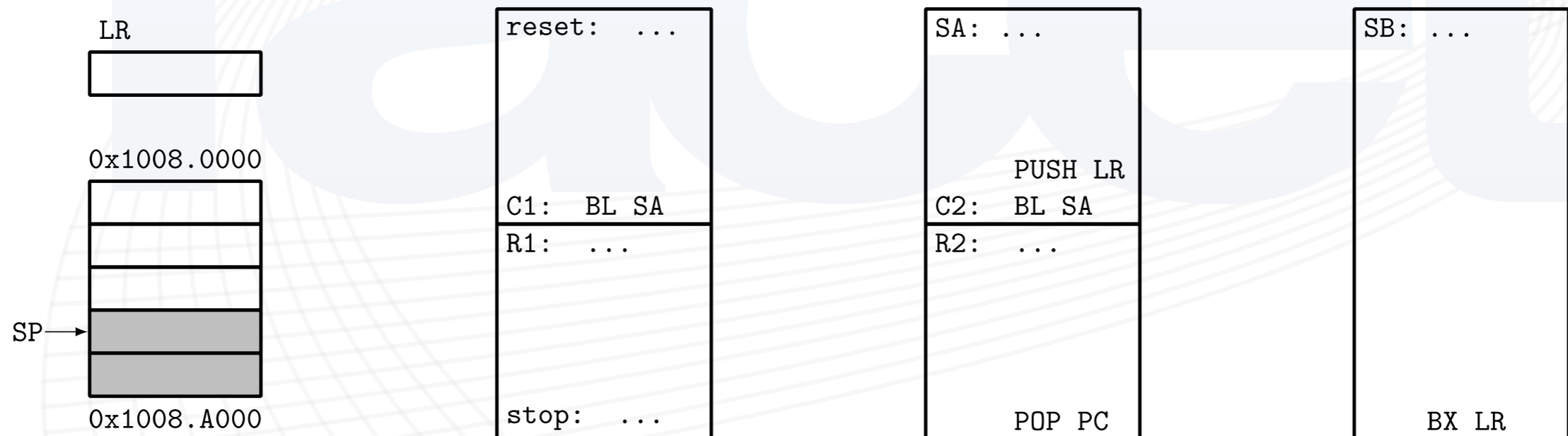


Subrutinas Anidadas

- ▶ ¿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:

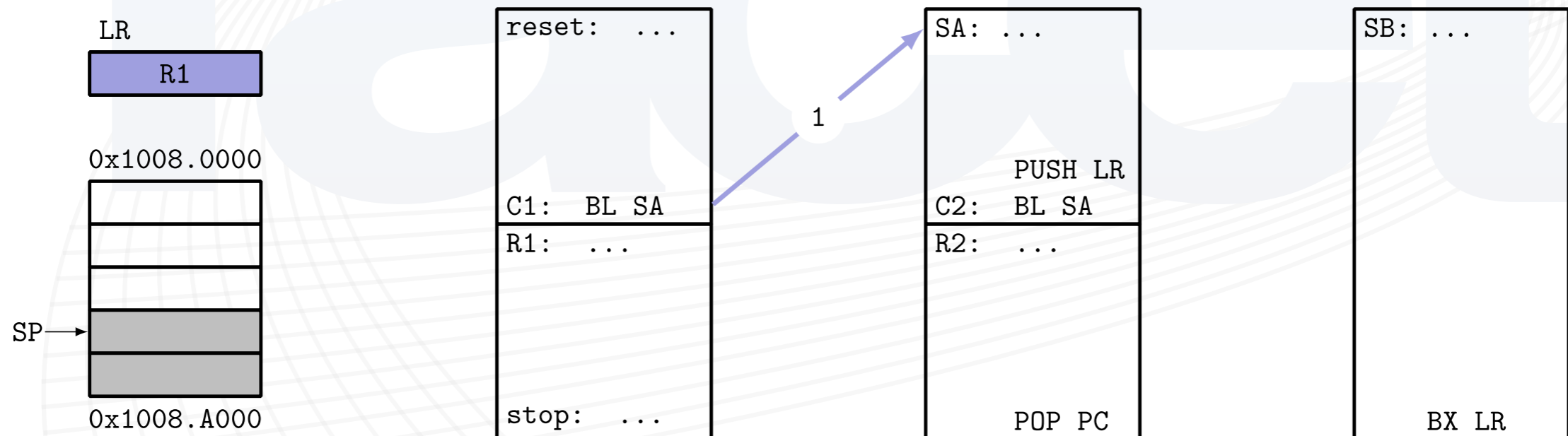
Subrutinas Anidadas

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



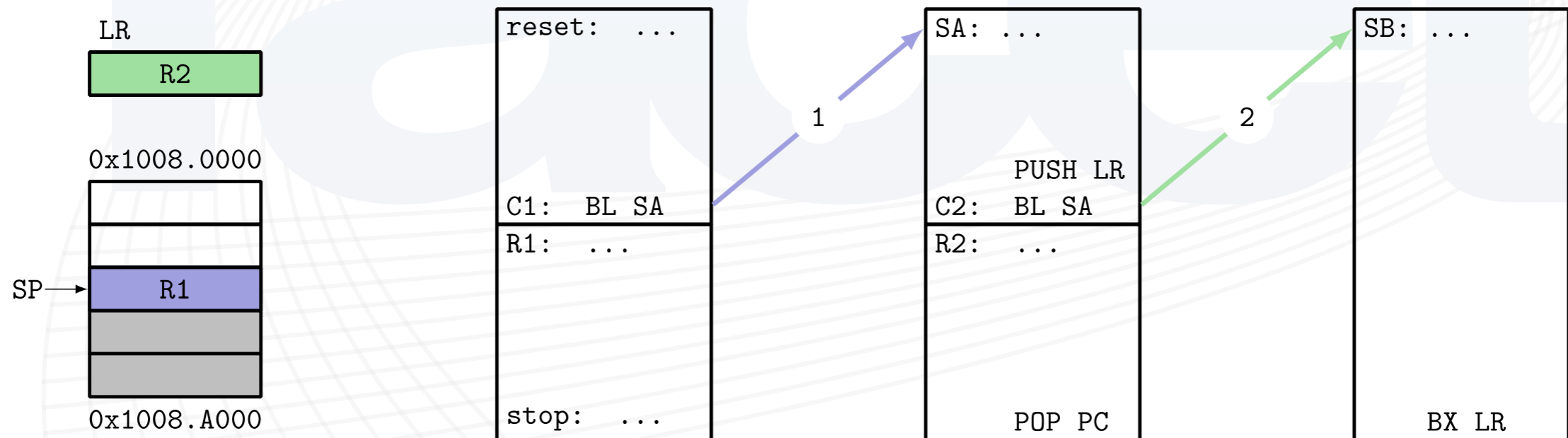
Subrutinas Anidadas

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



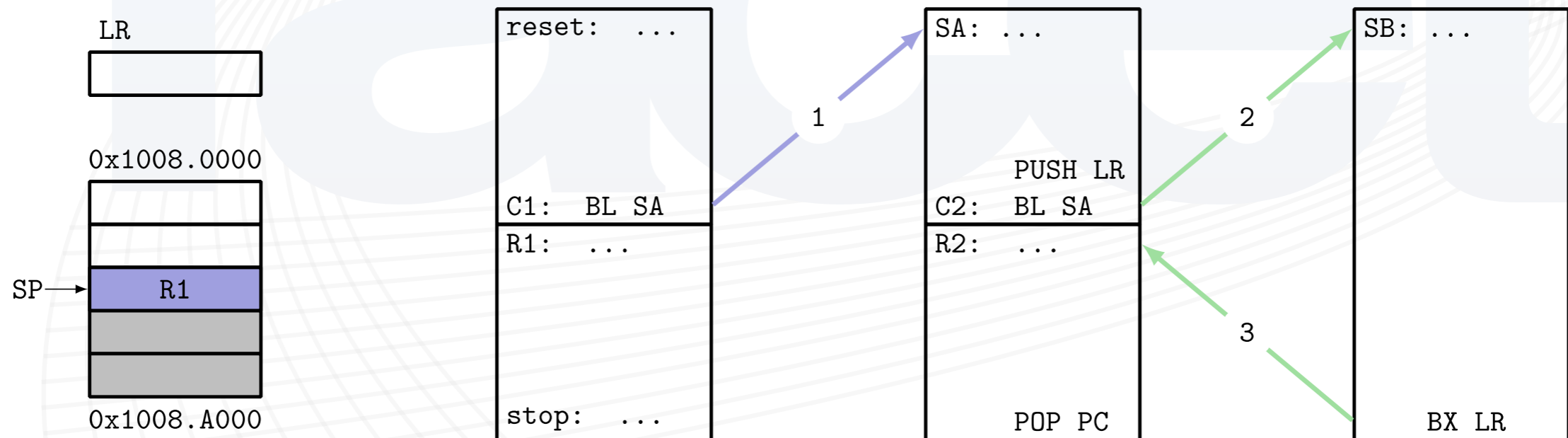
Subrutinas Anidadas

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



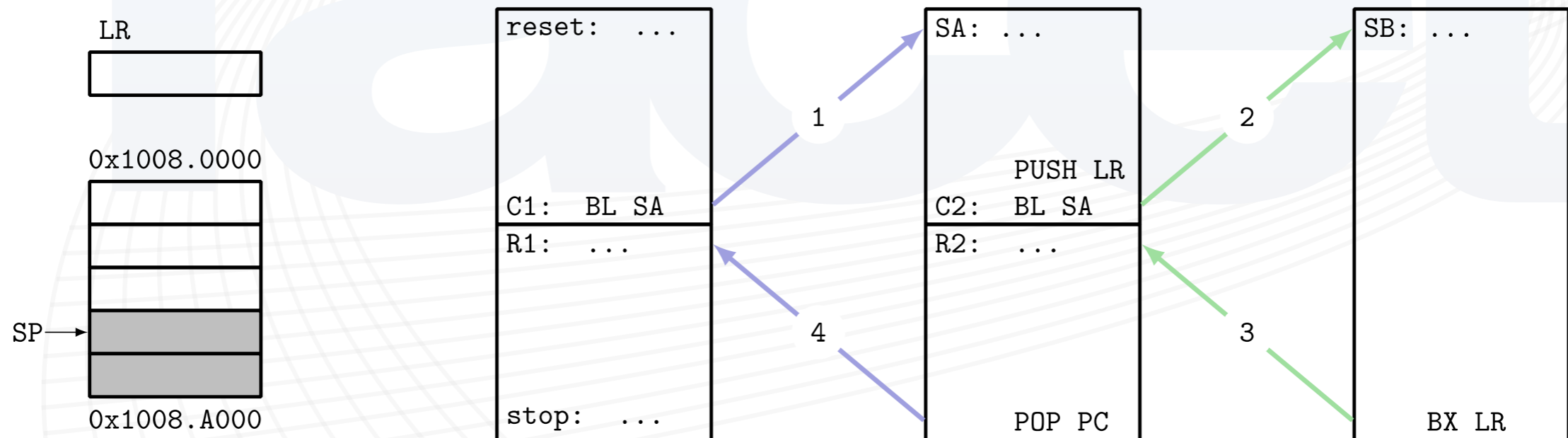
Subrutinas Anidadas

- ▶ ¿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 de la pila
- ▶ Por la convención ABI, el primer parámetro se pasa en Registro R0, el segundo en R1 y así...

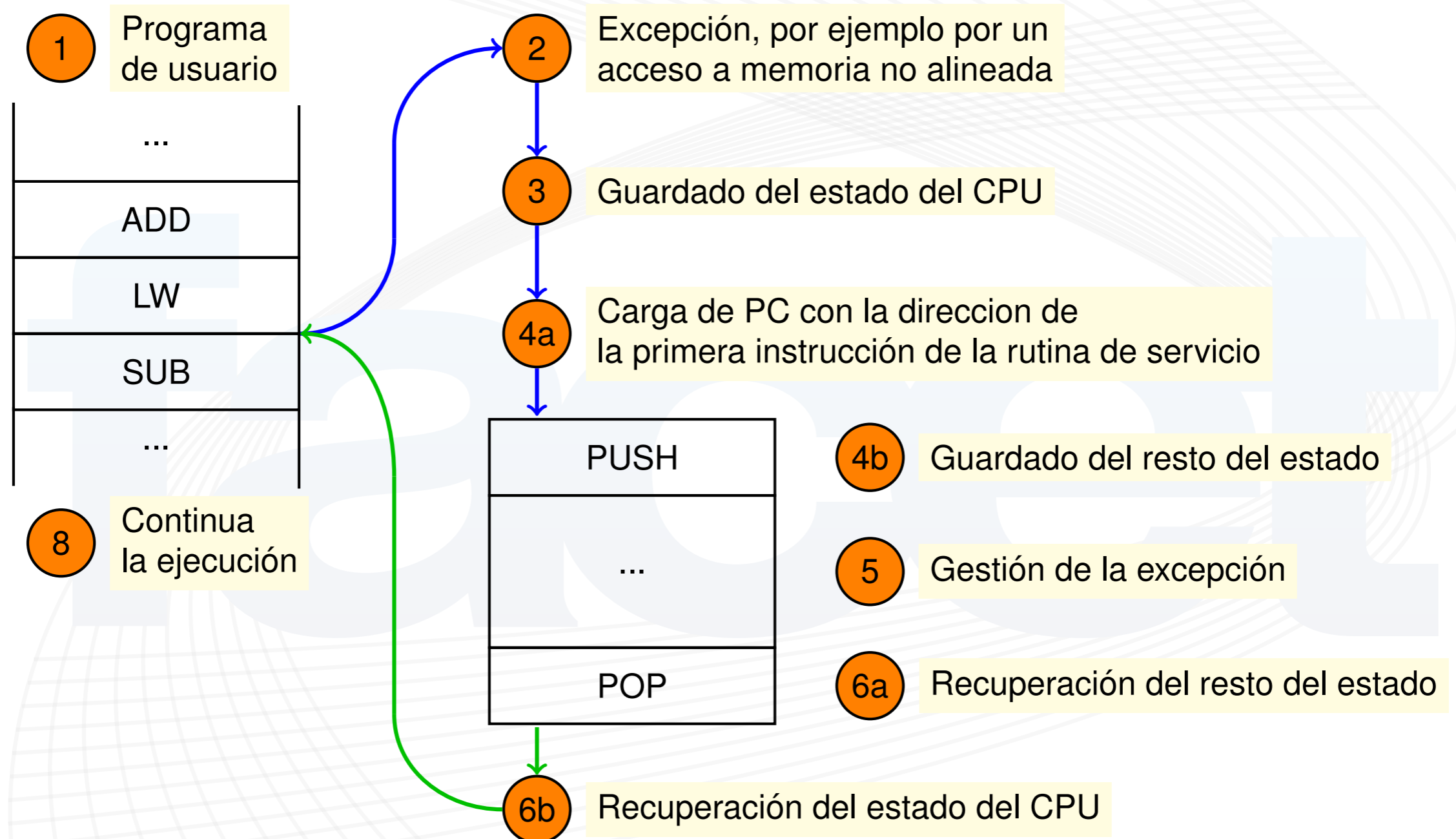
Gestión de errores en un procesador

- ▶ ¿Que ocurre con el procesador cuando ocurre un error durante la ejecución del programa?
- ▶ Lo ideal sería ejecutar una rutina que evalúe el error y tome las acciones correspondientes (no todos los errores son fatales)
- ▶ Para llamar a una rutina es necesario conocer el punto de entrada, es decir la dirección de la primera instrucción
- ▶ Como el error puede ocurrir en cualquier momento ¿a que dirección de memoria debería saltar el procesador?
- ▶ Una opción sería utilizar una dirección de memoria fija, pero esto es muy poco flexible
- ▶ Otra alternativa es utilizar un registro especial para indicarle al procesador donde se ubica la rutina para la gestión de errores

Definiciones

- ▶ Se denomina Excepción a una condición inusual que ocurre durante la ejecución de una instrucción
 - ▶ Internas (instrucción inválida)
 - ▶ Externas (acceso a memoria no implementada)
- ▶ Se denomina Trap a la transferencia de control sincrónica a la rutina de servicio encargada de gestionar una excepción
 - ▶ El proceso es similar al de un llamado a un subrutina, pero en general no se puede predecir cuando ocurrirá una excepción
 - ▶ Si es posible, se resuelve el problema y el programa principal continua la ejecución en forma transparente

Punto de vista del programador



Sistema operativo

- ▶ El programa de aplicación de un sistema embebido se puede dividir en cuatro grandes secciones:
 - ▶ Rutinas de inicialización
 - ▶ Rutinas de gestión de errores
 - ▶ Rutinas para controlar entradas/salidas
 - ▶ Rutinas para resolver el problema de aplicación concreto
- ▶ Las tres primeras tienen más relación con el entorno de ejecución que con el problema a resolver
- ▶ Se pueden separar y reutilizar para resolver diferentes problemas en el mismo entorno de ejecución
- ▶ Constituyen el núcleo de un Sistema Operativo y se convierten en un supervisor del programa de aplicación

Niveles de privilegio

- ▶ Para funcionar correctamente el programa supervisor necesita privilegios sobre el programa de usuario
- ▶ Esto le permite "defenderse" de programas maliciosos y/o errores
- ▶ El procesador inicia con el mayor nivel de privilegio lo que permite al software ejecutar cualquier operación
- ▶ Antes de iniciar el programa de usuario se ejecuta una operación para disminuir los privilegios
- ▶ Ahora el programa de usuario está obligado a pedirle al supervisor que ejecute las operaciones restringidas

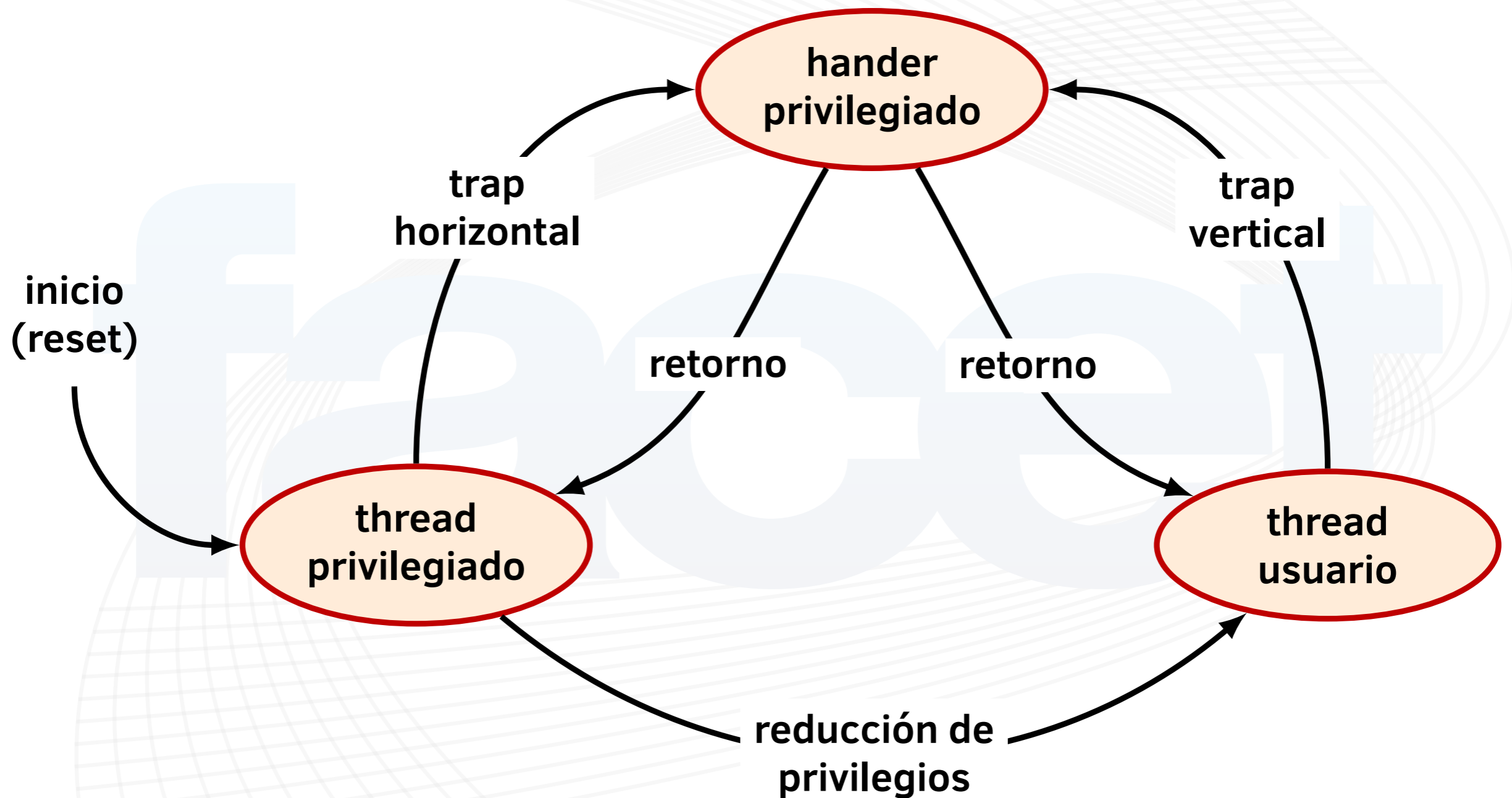
Niveles de privilegio en ARM

- ▶ Se definen dos niveles de privilegio
 - ▶ En modo Supervisor
 - ▶ Se pueden ejecutar todas las instrucciones
 - ▶ Se puede escribir todos los registros de control
 - ▶ Esta permitido el accesos a todas las áreas de memoria
 - ▶ En modo Usuario
 - ▶ No se permite ejecutar algunas instrucciones
 - ▶ No se permite escribir algunos registros de control
 - ▶ No se permite acceder a algunas áreas de memoria
 - ▶ En general no se permite acceder a los dispositivos
 - ▶ Se puede utilizar un puntero de pila independiente (MSP y PSP)

Niveles de privilegio en ARM

- ▶ Se denomina thread a un programa secuencial normal (puede incluir llamados a subrutinas y lazos)
- ▶ Se denomina handler a la rutina de servicio que tiene por función atender una excepción
- ▶ Se denomina trap horizontal a un cambio de contexto que no implica la elevación de los privilegios, por ejemplo de un thread privilegiado a un handler
- ▶ Se denomina trap vertical a un cambio de contexto que eleva los privilegios, por ejemplo de un thread de usuario a un handler

Niveles de privilegio



Supervisor Call (SVC)

- ▶ Similar a un llamado a subrutina pero es una excepción voluntaria y provoca un trap vertical
 - ▶ No es necesario conocer el punto de entrada
 - ▶ Se produce una elevación de privilegios
- ▶ Como es provocada por el programa se pueden pasar argumentos por registros
 - ▶ Permite definir un conjunto de servicios que el supervisor ofrece al programa de usuario
 - ▶ Simplifican las tareas del programa de usuario
 - ▶ Permiten al supervisor compartir y gestionar los recursos

Interrupciones para pedir servicios

- ▶ Una Interrupción es una señal externa asincrónica que permite a un dispositivo solicitar un servicio al procesador
 - ▶ Se puede pensar que una interrupción asincrónica produce una excepción sincrónica en una instrucción en particular
 - ▶ A consecuencia de esta excepción se produce un trap para ejecutar la rutina de servicio correspondiente
- ▶ Así las interrupciones son asincrónicas, pero la atención sigue siendo sincrónica como en las excepciones
- ▶ Es habitual que las interrupciones puedan enmascarse, es decir ignorarlas durante un tiempo
 - ▶ En este el pedido de interrupción sigue activo pero no se produce la excepción y, por lo tanto, tampoco el trap correspondiente

Simultáneas o anidadas

- ▶ Son simultáneas las interrupciones que ocurren en el intervalo de una instrucción
 - ▶ También son simultáneas las que quedaron sin atender de una ronda previa
 - ▶ Se resuelve asignando prioridades a cada interrupción
- ▶ Durante la atención de una interrupción podría llegar un nuevo pedido de interrupción
 - ▶ Si se atiende entonces las rutinas de servicio se anidan como sucede con las subrutinas
 - ▶ Las prioridades para el anidamiento son necesariamente diferentes a las prioridades para la simultaneidad

Excepciones en ARM

- ▶ Cada excepción se identifica con un número:
 - ▶ Las primeras 15 corresponden a eventos internos del núcleo
 - ▶ El resto corresponden a interrupciones externas
- ▶ Hasta 255 excepciones y hasta 240 interrupciones
 - ▶ El número de excepción es el número de interrupción mas 16
- ▶ Hay vector (o tabla) con la dirección de inicio de cada rutina de servicio
- ▶ Se busca en la entrada correspondiente utilizando el numero de excepción como indice
 - ▶ Existe un registro VTOR para señalar el inicio de la tabla

Niveles de prioridades en ARM

- ▶ 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 número de excepción.

Niveles de prioridades en ARM

- ▶ 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.

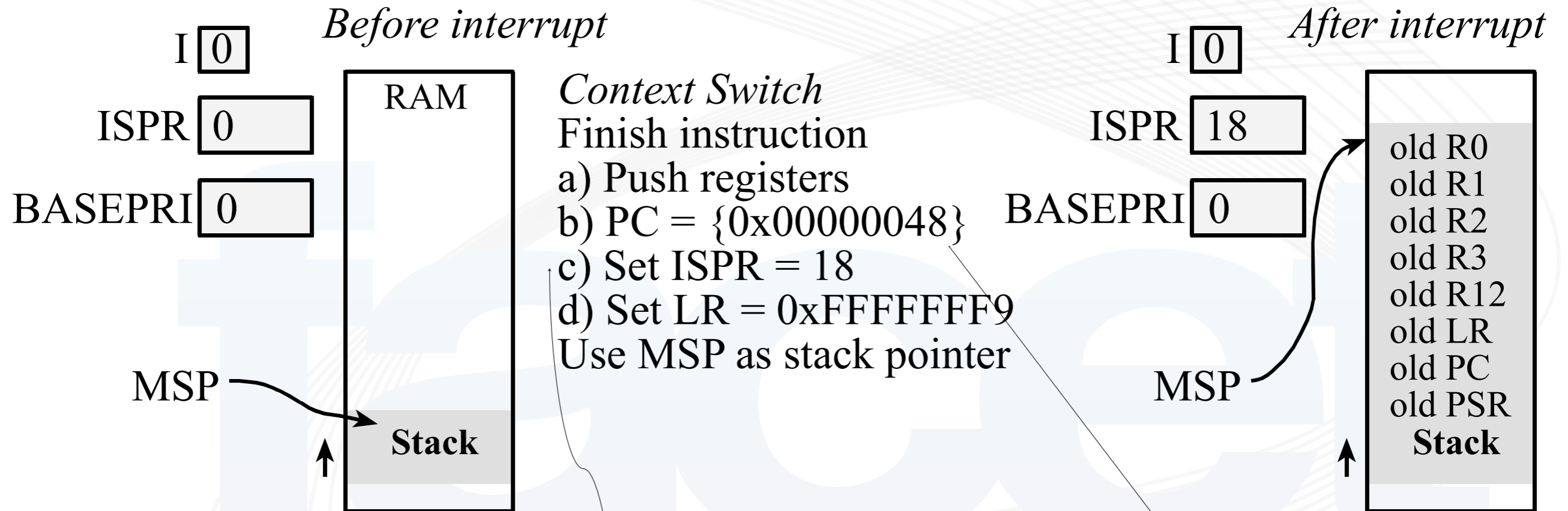
Excepciones en ARM

- ▶ 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 memoria
 - ▶ Bus Fault: Falla en el Bus
 - ▶ Usage Fault: Falla en programa: instrucción no definida, división por cero, etc.
- ▶ SVCcall: Pedidos del programa al SO, servicios privilegiados
- ▶ SysTick Interrupt: generada por un timer interno del núcleo

Detalle de los traps en ARM

- ▶ Suspender ejecución y apilar 8 de los registros:
 - ▶ R0-R3, R12, LR, PC, PSR
- ▶ Escribir en LR un valor que indica el estado del procesador antes de la excepción:
 - ▶ 0xFFFFFFFF9, si estaba en un thread en modo supervisor
 - ▶ 0xFFFFFFFF1, si estaba en un handler
 - ▶ 0xFFFFFFFFD, si estaba en un thread en modo usuario
- ▶ Escribir en PSR el número de excepción
- ▶ Escribir en PC la dirección inicial de la rutina de servicio
 - ▶ Usar un registro (VTOR) como puntero a una tabla
 - ▶ Usar el número de excepción como índice en la tabla

Ej: Cambio de Contexto (INT#18)



Dirección en Vector de ISR para DMA

Número de excepción 18 corresponde a DMA

Condiciones de interrupción

- ▶ 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.

Ejecución de la ISR

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

Esperar tiempo fijo

- ▶ Hasta aquí lo hacíamos con lazos
- ▶ Presentaremos un dispositivo que viene en el núcleo del ARM: SysTick Timer
- ▶ Se puede utilizar para esperas únicas de un lapso de tiempo conocido
- ▶ EN general se usa para obtener una base de tiempo con una señal periódica

SysTick Timer

- ▶ Contador de 24 bits.
- ▶ Decrementa a la frecuencia del bus.
- ▶ Inicia en un valor n y decrementa hasta cero
 - ▶ Cuando llega a 0 se carga con n de nuevo
- ▶ Provoca una señal cada $n + 1$ pulsos:
 - ▶ $\text{next_value} = (\text{current_value} - 1) \bmod (n + 1)$
 - ▶ Secuencia: $n, n-1, n-2, \dots, 2, 1, 0, n, n-1, \dots$
- ▶ Al pasar por 0 produce una señal que puede configurarse como interrupción

Registros de SysTick Timer

Address	31-24	23-17	16	15-3	2	1	0	Nombre
\$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	SYST-CSR - control/stat.
\$E000E014	0	24-bit RELOAD value						SYST-RVR – Reload Val.
\$E000E018	0	24-bit CURRENT value of SysTick counter						SYST-CVR – Current Val.

- ▶ **SYST-CSR: Registro de control y estado**
 - ▶ **ENABLE:** En 0 detiene el reloj y en 1 lo arranca
 - ▶ **CLK_SRC:** Permite elegir varias fuentes de reloj
 - ▶ **COUNT** es un flag – solo de lectura (se ignoran las escrituras)
 - ▶ Se pone a 1 cada vez que el contador pasa por 0
 - ▶ Se pone a 0 cuando se lee el contador
 - ▶ Si pide interrupción, con la respuesta se pone a 0 automáticamente
 - ▶ **INTEN=1** para habilitar interrupción cuando **COUNT=1**

Registros de SysTick Timer

Address	31-24	23-17	16	15-3	2	1	0	Nombre
\$E000E010	0	0	COUNT	0	CLK_SRC	INTEN	ENABLE	SYST-CSR - control/stat.
\$E000E014	0	24-bit RELOAD value						SYST-RVR – Reload Val.
\$E000E018	0	24-bit CURRENT value of SysTick counter						SYST-CVR – Current Val.

- ▶ **SYST-RVR: Registro con el valor de recarga**
 - ▶ El programa escribe allí el valor máximo n para el contador
 - ▶ Cuando el contador llega a 0 se vuelve a cargar con este valor n
- ▶ **SYST-CVR: Registro con el valor actual de la cuenta**
 - ▶ Escribir cualquier cosa pone la cuenta actual en 0
 - ▶ Al leer o escribir se borra COUNT en el registro SYST-CSR

Inicialización en 5 pasos

- ▶ Escribir `ENABLE=0` en `SYST-CSR` para detener el contador
- ▶ Escribir valor de `RELOAD (n)` en `SYST-RVR`
- ▶ Resetear Contador y `COUNT=0`:
 - ▶ escribiendo cualquier valor en `SYST-CVR`
- ▶ Escribir `CLK_SRC=x` para elegir la fuente
 - ▶ Por ejemplo `CLK_SRC=1` para el clock interno de bus
- ▶ Escribir `ENABLE=1` y `INTEN=1` en `SYST-CSR` para para iniciar la cuenta y habilitar interrupción