Introducción a Implementación de Sistemas Operativos

Mg. Ing. Gonzalo E. Sanchez
Esp. Ing. Hanes N. Sciarrone
MSE - 2023

Introducción ISO

- Arquitectura Cortex
- Modelo del programador
- Modelo de excepciones

Arquitectura Cortex

Arquitectura Cortex

Cortex®-M processors

MCU + DSP

Smallest footprint / lowest power

Cortex®-R processors

RTOS

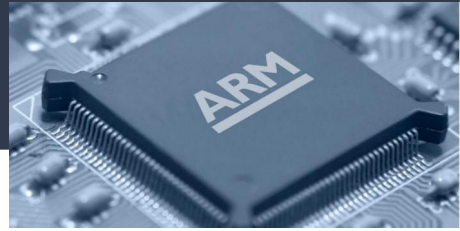
Highest performance / real-time

Cortex®-A processors

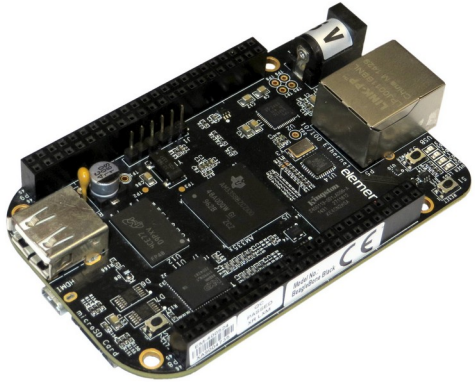
Rich OS

Highest performance

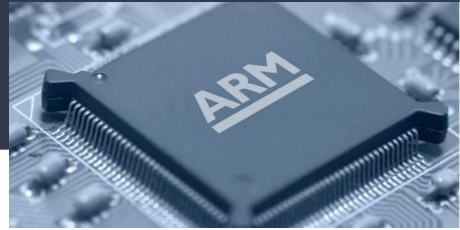
Arquitectura Cortex



- Cortex A (APPLICATION)
  - Procesadores de alto rendimiento
  - Orientados a la implementación de sistemas operativos.

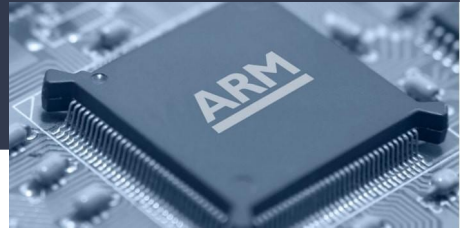


Arquitectura Cortex

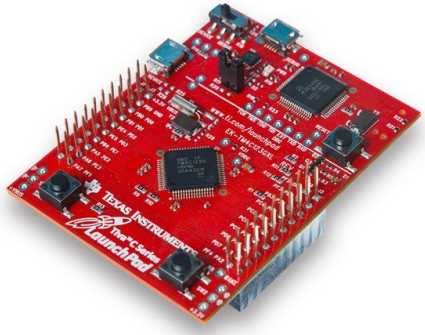


- Cortex R (REAL TIME)
  - Procesadores orientados a sistemas de tiempo real.
  - Necesidad de implementar soluciones de baja latencia y alta capacidad de procesamiento.

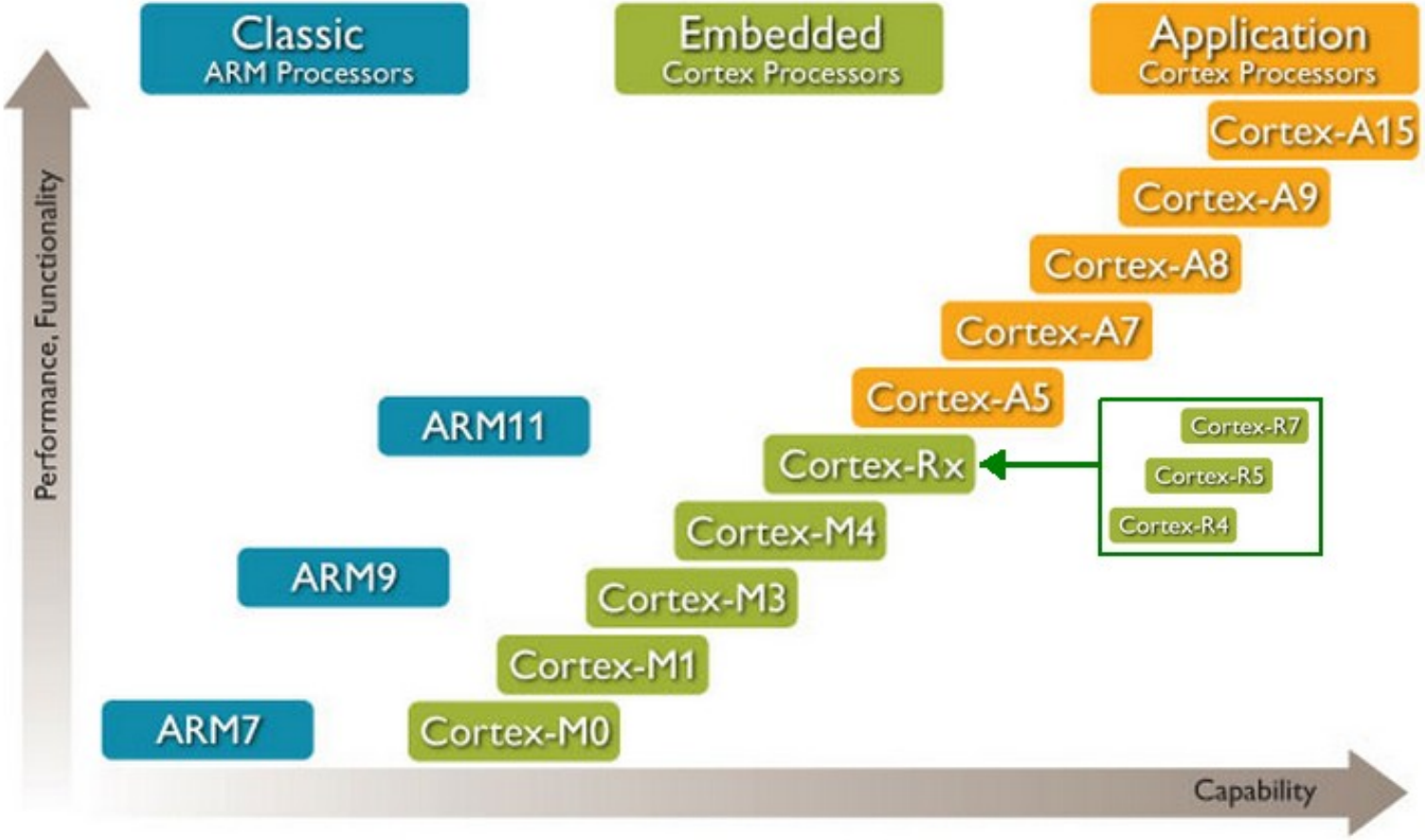
Arquitectura Cortex



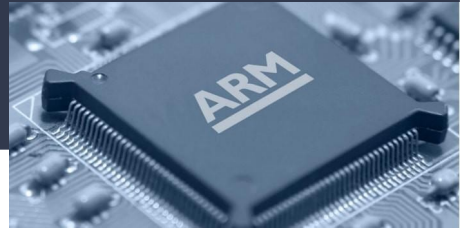
- Cortex M (MICROCONTROLLER)
  - Procesadores orientados a dispositivos de consumo masivo y sistemas embebidos compactos.
  - Diseñados para alta densidad de código y programación en C.



Arquitectura Cortex

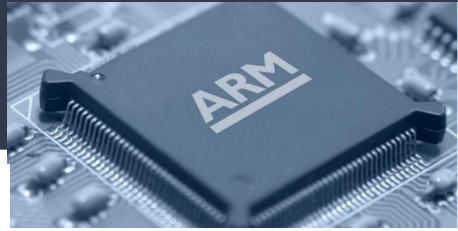


Arquitectura Cortex

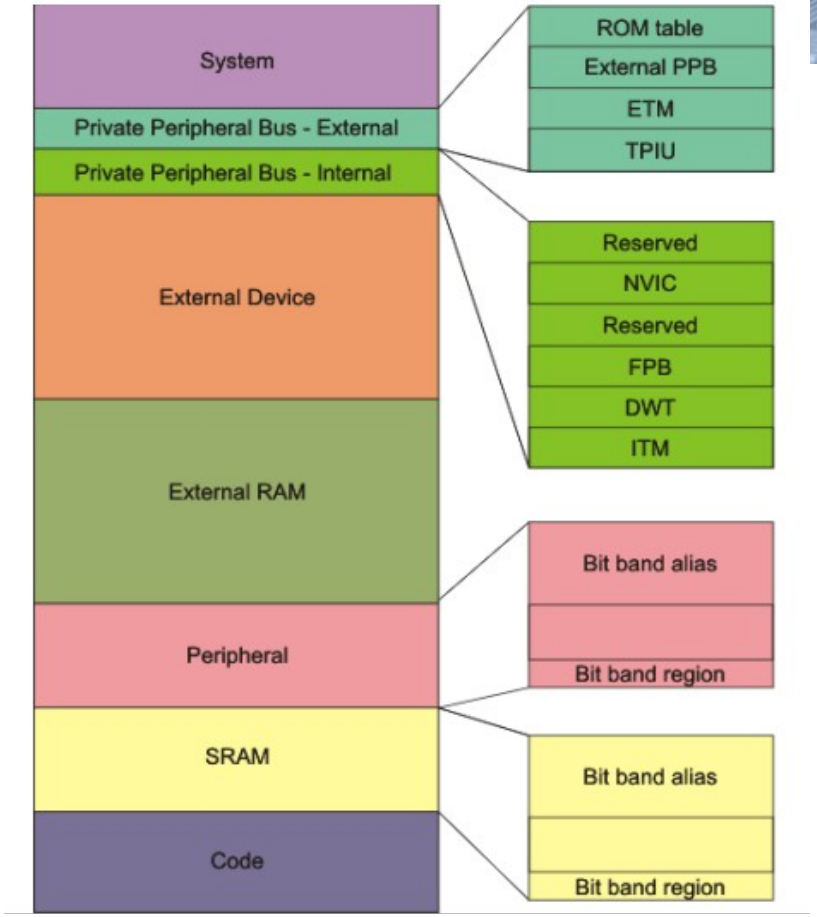


- Cortex M0 / M0+
  - Implementación mínima para bajo consumo y bajo costo.
- Cortex M3 / M4 / M7
  - Mayor performance, agregan funcionalidades para procesamiento digital de señales, unidad de protección de memoria, etc.
  - Cortex M4 se diferencia de M3 al incluir instrucciones SIMD.
  - Cortex M4F se diferencia de M4 al incluir FPU.

Arquitectura Cortex



- Mapa de memoria Cortex M4



Modelo del programador

Modelo del programador

- ARM define como el modelo del programador (o programmer’s model) como la siguiente información:
  - Descripción de registros de núcleo individuales.
  - Stacks.
  - Modos de operación.
  - Niveles de privilegio.
  - Interrupciones y excepciones.
  - Tipos de datos
  - Cortex Microcontroller Software Interface Standard (CMSIS)

Modelo del programador

- Definiremos los puntos más importantes para la implementación de nuestro OS.
- Siguen habiendo revisiones de documentación con respecto al núcleo.
- Es buena práctica utilizar la última documentación disponible.

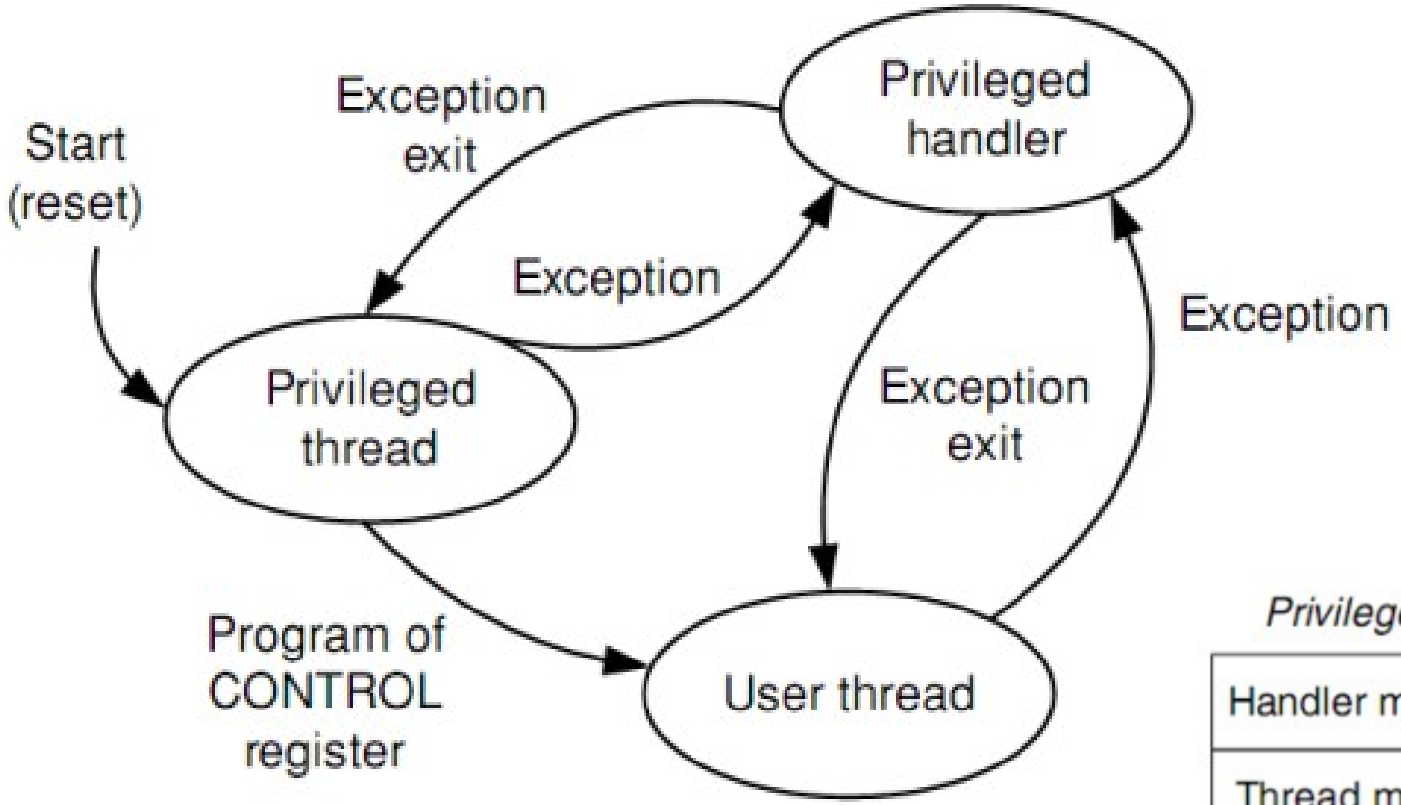
Modos de operación y niveles de privilegio

- Existen dos modos de operación:
  - Modo Thread: Ejecución de software de aplicación.
  - Modo Handler: Ejecución de excepciones.
- Cuando el core sale de un reset, entra en modo Thread.
- Cuando se produce una excepción, entra en modo Handler.
- Las interrupciones son tratadas como excepciones.
- Al terminar de ejecutar una excepción, se vuelve al modo Thread.

Modos de operación y niveles de privilegio

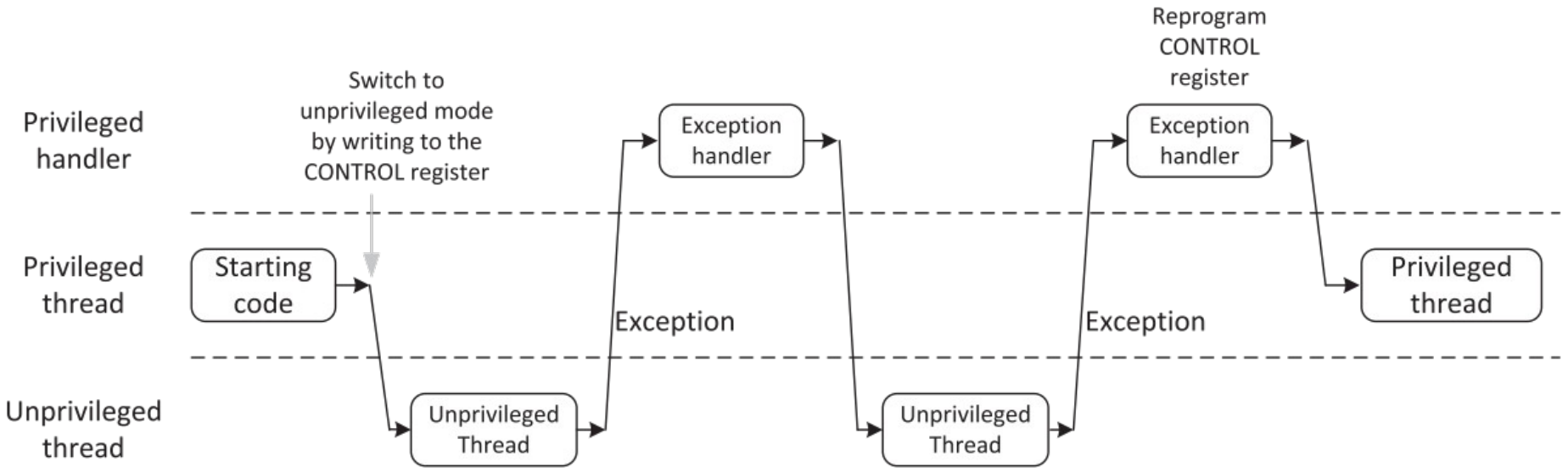
- Existen dos niveles de privilegios:
  - Modo privilegiado.
  - Modo no privilegiado.
- Software en modo no privilegiado no tiene acceso a NVIC, SysTick y el bloque de control del sistema.
- Puede restringirse el acceso a hardware (memoria o periféricos).
- Software en modo privilegiado no tiene restricciones.

Modos de operación y niveles de privilegio



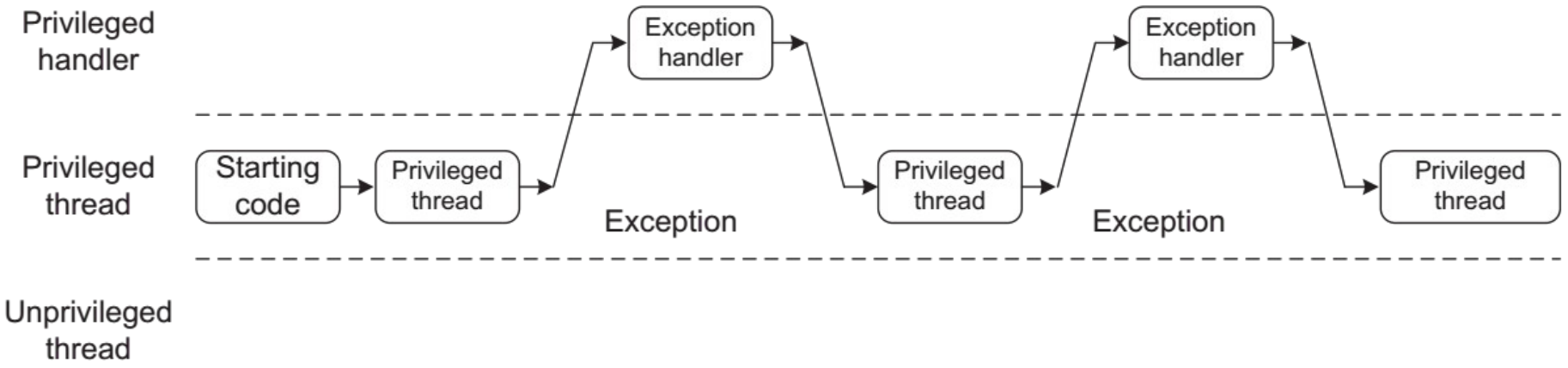
Privileged	*User*
Handler mode	
Thread mode	Thread mode

Modos de operación y niveles de privilegio



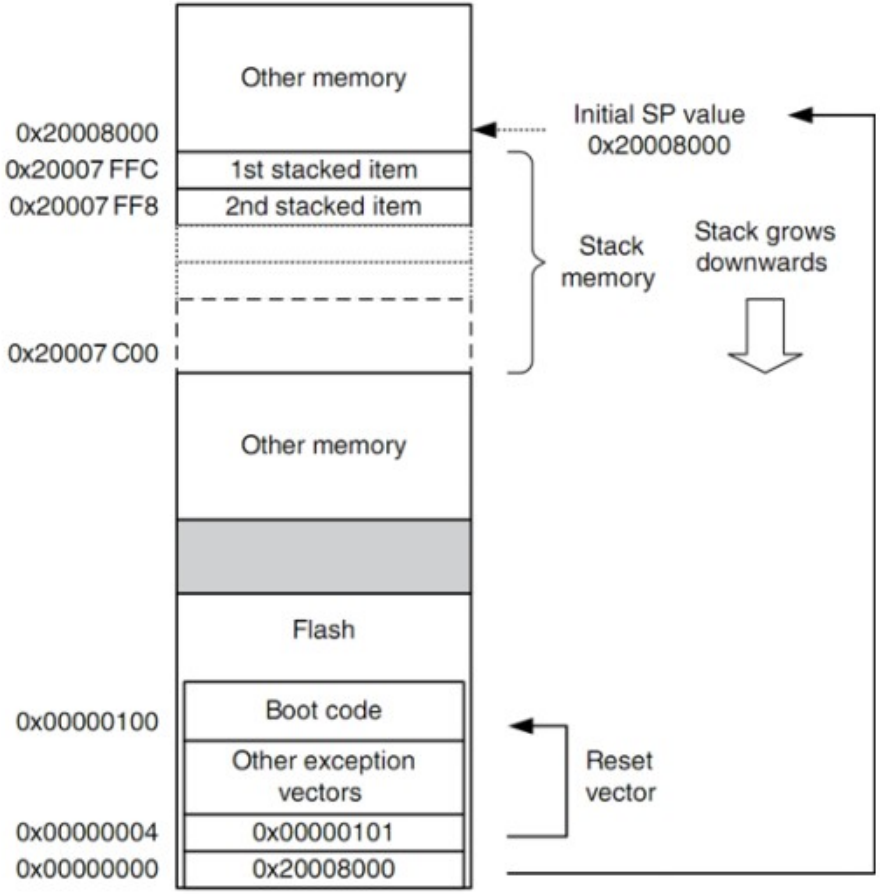
Modos de operación y niveles de privilegio

- Para la implementación de nuestro OS, no utilizaremos código sin privilegios.
- Esto simplifica el manejo del stack (solo se utiliza MSP).



Stacks

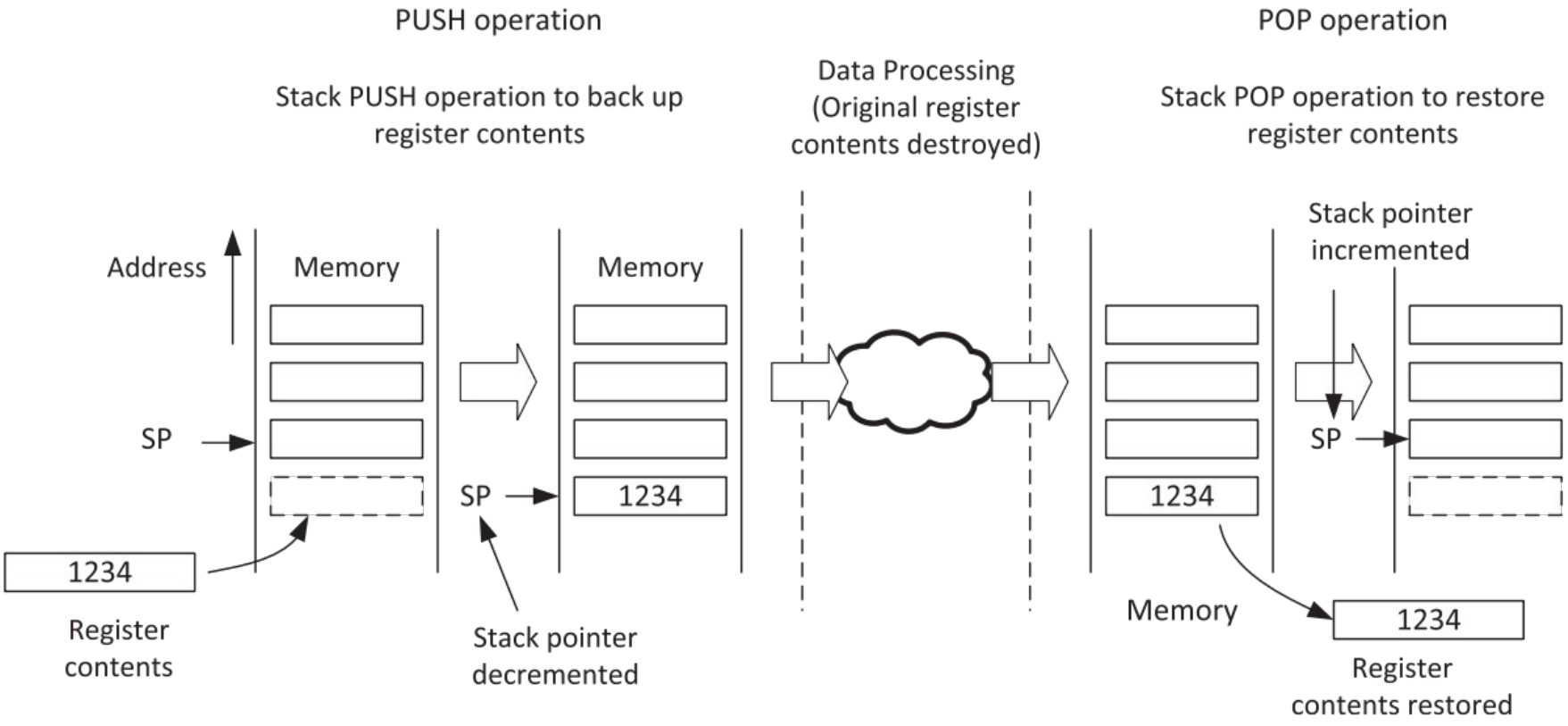
- El procesador utiliza un stack descendente.
- Comienza en direcciones de memoria “altas”.
- Luego de un push, decrementa el stack pointer.
- *full descending stack:* el stack pointer apunta al último elemento ingresado al stack.



Stacks

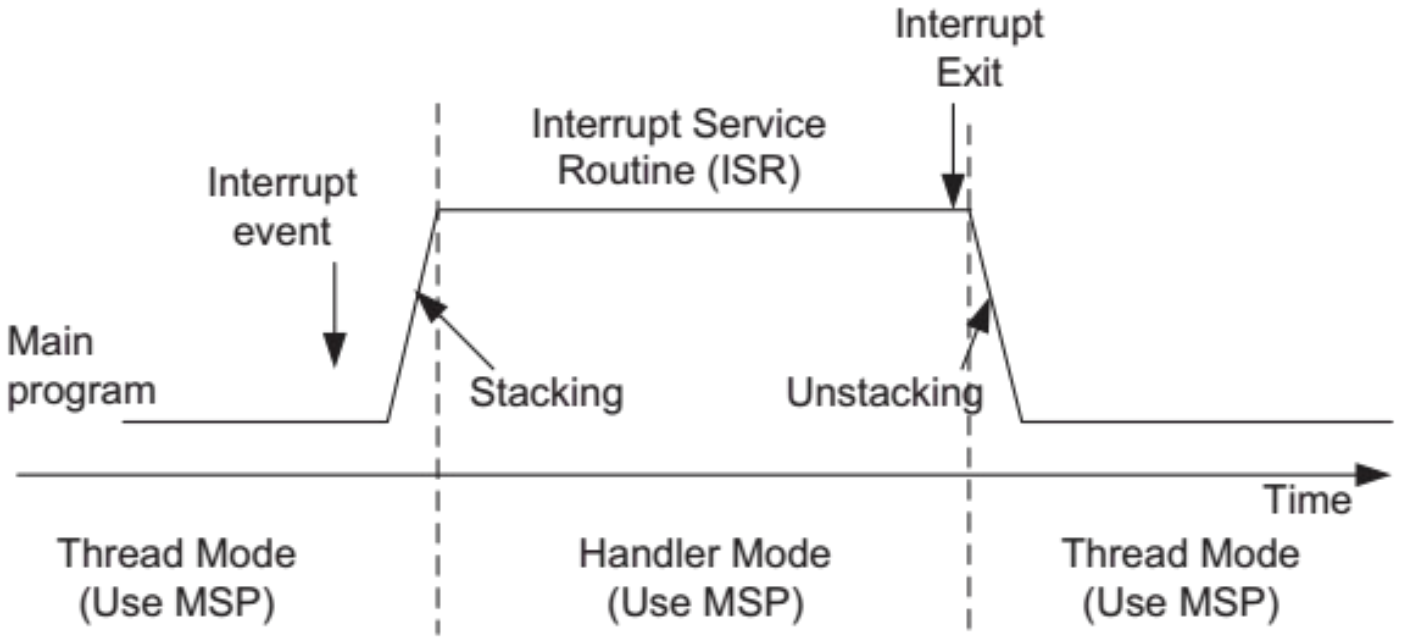
- El procesador implementa dos stacks:
  - Main Stack.
  - Process Stack.
- Cada stack tiene su puntero individual (MSP y PSP).
- El puntero a utilizar esta definido por el registro CONTROL.
- Depende del modo de operación.

Stacks

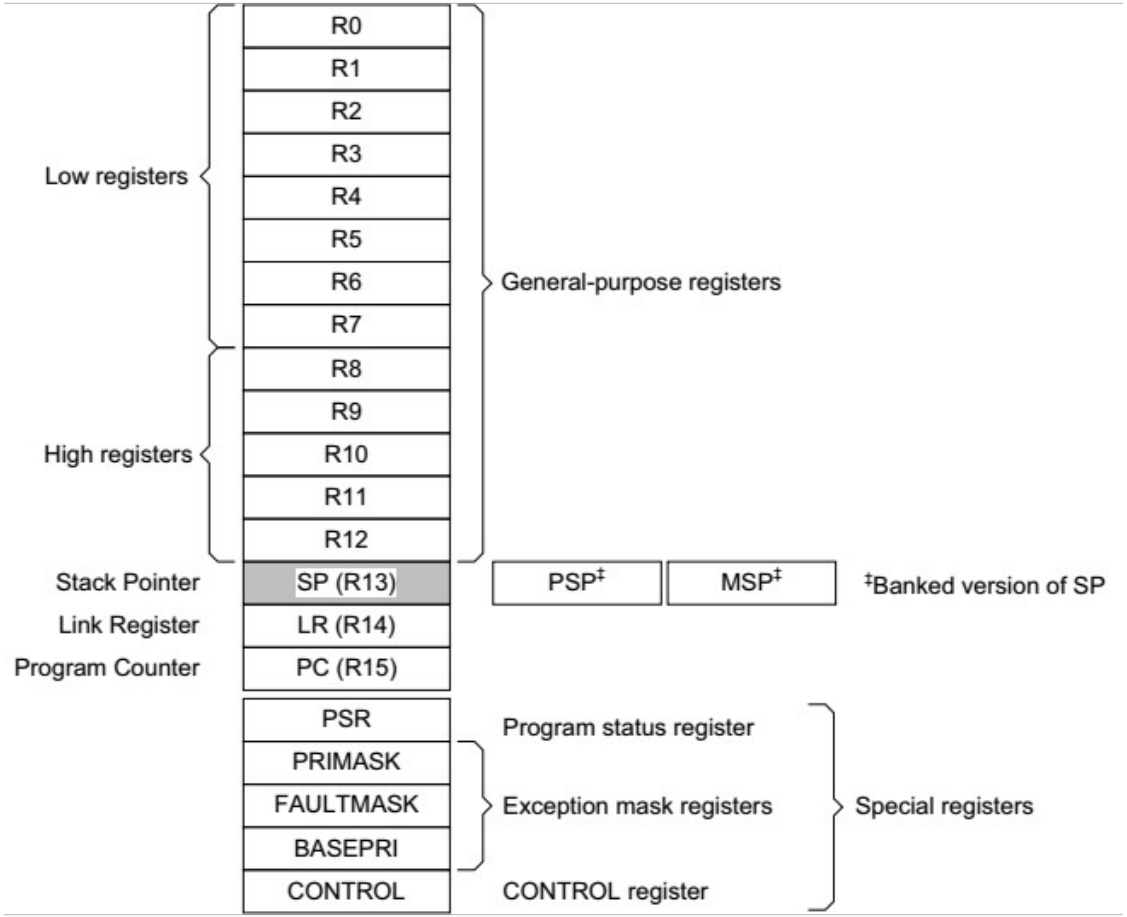

PUSH operation
Stack PUSH operation to back up register contents
Data Processing (Original register contents destroyed)
POP operation
Stack POP operation to restore register contents
Address
Memory
Memory
SP
SP
1234
1234
Register contents
Stack pointer decremented
Stack pointer incremented
SP
1234
Memory
1234
Register contents restored

Stacks

Caso de uso para la materia:



Registros


R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
SP (R13)
LR (R14)
PC (R15)
PSR
PRIMASK
FAULTMASK
BASEPRI
CONTROL
Low registers
High registers
General-purpose registers
Stack Pointer
Link Register
Program Counter
PSP‡
MSP‡
‡Banked version of SP
Program status register
Exception mask registers
CONTROL register
Special registers

Registros

Name	Type[a]	Required privilege[b]	Reset value
R0-R12	RW	Either	Unknown
MSP	RW	Privileged	See description
PSP	RW	Either	Unknown
LR	RW	Either	0xFFFFFFFF
PC	RW	Either	See description
PSR	RW	Privileged	0x01000000
ASPR	RW	Either	Unknown
IPSR	RO	Privileged	0x00000000
EPSR	RO	Privileged	0x01000000
PRIMASK	RW	Privileged	0x00000000
FAULTMASK	RW	Privileged	0x00000000
BASEPRI	RW	Privileged	0x00000000
CONTROL	RW	Privileged	0x00000000

Registros

- **R0-R12**: Registros de propósito general para operaciones de datos (32bits).

- **SP (R13):** Registro Stack Pointer. Puede apuntar a MSP o PSP dependiendo del bit CONTROL[1] = SPSEL.

- **LR (R14):** Link Register guarda la información de retorno para subrutinas, llamadas a funciones y excepciones.

Registros

- **PC (R15):** Program Counter.
  - Contiene la dirección actual de programa. En el reset se carga con 0x00000004.
  - El bit[0] siempre se carga en xPSR[24] (Thumb bit).

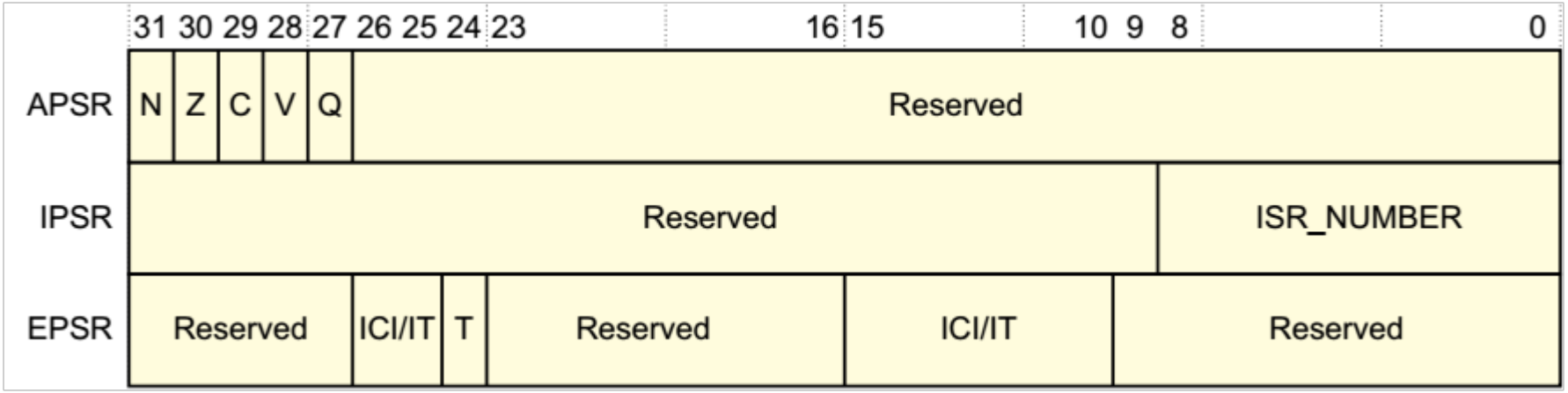
- **PSR:** Program Status Register, combina tres registros:
  - Application Program Status Register (APSR).
  - Interrupt Program Status Register (IPSR).
  - Execution Program Status Register (EPSR).

Registros

- Los registros APSR, EPSR e IPSR se componen de campos de bits mutuamente excluyentes.

- Pueden ser accedidos:
  - Individualmente.
  - De a pares.
  - Como un solo registro (xPSR).

Registros


31 30 29 28 27 26 25 24 23 16 15 10 9 8 0
APSR
N
Z
C
V
Q
Reserved
IPSR
Reserved
ISR_NUMBER
EPSR
Reserved
ICI/IT
T
Reserved
ICI/IT
Reserved

Registros

- **APSR:** Contiene flags de estado de ejecución de instrucciones anteriores.

Bits	Name	Function
[31]	N	Negative flag
[30]	Z	Zero flag
[29]	C	Carry or borrow flag
[28]	V	Overflow flag
[27]	Q	DSP overflow and saturation flag
[26:20]	-	Reserved
[19:16]	GE[3:0]	Greater than or Equal flags. See *SEL* on page 3-70 for more information.
[15:0]	-	Reserved

Registros

- **IPSR:** Contiene el número de la excepción ocurrida.

Bits	Name	Function
[31:9]	-	Reserved
[8:0]	ISR_NUMBER	This is the number of the current exception: 0 = Thread mode 1 = Reserved 2 = NMI 3 = HardFault 4 = MemManage 5 = BusFault 6 = UsageFault 7-10 = Reserved 11 = SVCall 12 = Reserved for Debug 13 = Reserved 14 = PendSV 15 = SysTick 16 = IRQ0. . . . n+15 = IRQ(n-1)[a]

Registros

- **EPSR:** Contiene el estado de los bits de ejecucion para:
  - La instrucción If-Then (IT)
  - Instrucciones interrumpibles-continuables (ICI). Afecta instrucciones Store-Multiple y Load-Multiple.
- Además contiene el bit THUMB (EPSR[24]).
- Cortex M4 solo soporta ejecución de funciones Thumb.
- Especial atención. Modo THUMB: EPSR[24] = 1

Registros

- **THUMB State:** Las siguientes condiciones pueden hacer un clear de este bit:
  - Ejecución de instrucciones BLX, BX y POP{PC}.
  - Restitución de un valor guardado en stack del valor xPSR al volver de una excepción.
  - bit[0] en el valor del vector en una interrupción o un reset .
- Direcciones almacenadas en la tabla vector son impares gracias al bit THUMB en cortex M4.

Registros

- **CONTROL:** Este registro controla el stack utilizado y el nivel de privilegio para ejecución de software.
- En el caso de estar implementado, indica si la FPU está activa.
- CONTROL[1] = SPSEL
  - SPSEL = 0 ⇒ Se utiliza MSP. Este es el valor luego de un reset.
  - SPSEL = 1 ⇒ Se utiliza PSP.
- CONTROL[0] = nPRIV
  - nPRIV = 0 ⇒ Modo privilegiado.
  - nPRIV = 1 ⇒ Modo no privilegiado.

Modelo de excepciones

Modelo de excepciones

- El modelo de excepciones describe los siguientes aspectos:
  - Estados de excepciones.
  - Tipos de excepciones.
  - Handlers.
  - Tabla de vectores.
  - Prioridades de excepción.
  - Grupos de prioridades de interrupciones.
  - Entrada y retorno de excepciones.

Estados de excepciones

- Cada excepción puede tener los siguientes estados:
  - **Inactiva:** No solo no está activa, sino que tampoco está pendiente
  - **Pendiente:** A la espera de ser atendida por el procesador.
  - **Activa:** Está siendo atendida por el procesador
  - **Activa y pendiente:** Está siendo atendida por el procesador y hay una excepción pendiente de la misma fuente.
- Si hay dos excepciones anidadas, ambas están en estado activo.

Tipos de excepciones

- Los distintos tipos de excepción son:
  - **Reset:** El modelo de excepciones trata al reset (POR y warm reset) como un tipo especial de excepción.
  - **NMI:** Non Maskable Interrupt, es la excepción de más alta prioridad debajo del reset. Siempre está habilitada.
  - **HardFault:** Se produce por un error al procesar una excepción, o porque surge una excepción que no puede ser manejada con otro mecanismo.
  - **MemManage:** Se produce siempre relacionada a regiones de memoria protegida.

Tipos de excepciones

- Los distintos tipos de excepción son:
  - **BusFault:** Se produce por una falla relacionada a la ejecución de una instrucción o transacción de datos en memoria.
  - **UsageFault:** Se produce por una falla relacionada a la ejecución de una instrucción. Incluye instrucciones indefinidas, estado inválido de ejecución (thumb) y errores en retornos de excepciones.
  - **SVCall:** supervisor call, se produce al ejecutar la instruccion SCV
  - **PendSV:** Es una petición originada en una interrupción. Se recomienda utilizarla para hacer cambios de contexto en OS cuando no hay otra interrupción activa.

Tipos de excepciones

- Los distintos tipos de excepción son:
  - **SysTick:** Excepción generada a partir del timer del sistema. Utilizado como base de tiempos en OS.
  - **IRQ:** Interrupciones generadas por un periférico, o por software. Dependen del fabricante del silicio, no del procesador (no definidas por ARM).

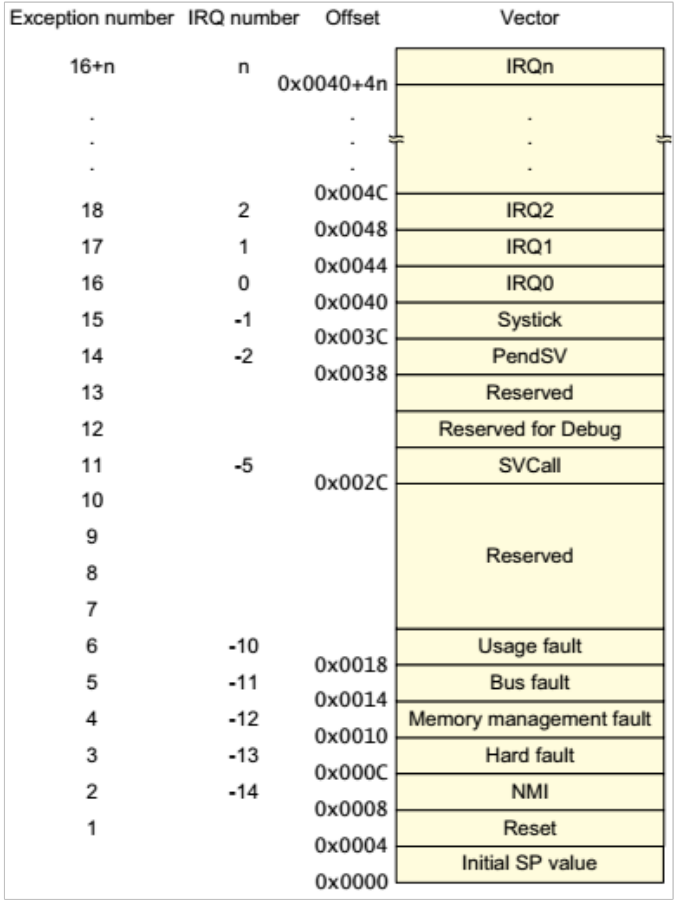
Handlers

- El procesador maneja las interrupciones utilizando:
  - Interrupt Service Routines: para todas las IRQ.
  - Fault Handlers: para HardFault, MemManage fault, UsageFault, and BusFault.
  - System Handlers: para NMI, PendSV, SVCall y SysTick.

- NOTA: Alguna documentación asume que todos los Fault Handlers son System Handlers.

Tabla de vectores

- La tabla de vectores contiene el valor de reset del SP.
- Determina las direcciones de todos los handlers.
- El LSB de cada vector debe ser 1.
- Esto indica que el handler contenido está codificado en THUMB.



Prioridades de excepción

- Todas las excepciones tienen prioridad asociada.
- La prioridad es configurable, excepto para RESET, HardFault y NMI.
- Un valor cercano a cero indica alta prioridad.
- Luego de un reset, todas las excepciones configurables tienen prioridad 0 (la más alta configurable).
- En teoría es posible tener prioridades en el rango de 0-255.
- Cada fabricante determina cuántos bits se disponen para esto.

Prioridades de excepción

- El NVIC soporta grupos de prioridades de excepciones.
- Pueden agruparse interrupciones de periféricos en un mismo nivel.
- El grupo de prioridad determina si una excepción puede expropiar el CPU a otra excepción.
- Se toma la prioridad de la excepción y se enmascara según la cantidad de bits configurada.

Prioridades de excepción

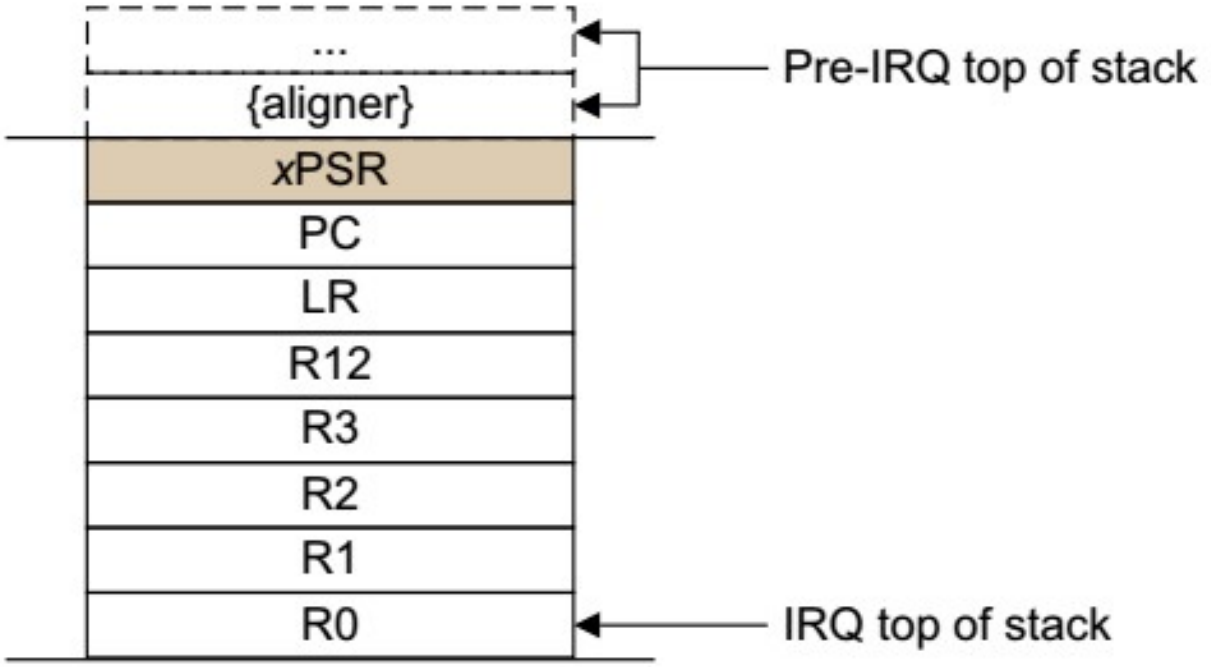
- El registro de prioridad para cada excepción solo tiene 8 bits.
- Se divide en dos partes siempre: grupo de prioridad y subprioridad.
- En un MCU con solo 3 bits implementados, se puede lograr 4 grupos con 2 subprioridades.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Preempt priority		Sub-priority	Not Implemented				

Entrada y retorno de excepciones

- Si una excepción de suficiente prioridad ocurre, esta expropia el CPU.
- Al expropiar el CPU, se hace un PUSH de cierta información al stack.
- Esto conforma lo que se llama un **stack frame**.
- El tamaño del stack frame dependerá de si está activa o no la FPU.

Entrada y retorno de excepciones



Exception frame without
floating-point storage

Entrada y retorno de excepciones

- Inmediatamente luego de hacer PUSH del stack frame, el SP apunta al último elemento de memoria en el stack.
- El stack frame contiene la dirección de retorno (PC).
- El CPU escribe el valor EXEC_RETURN en el registro LR.
- Esto indica que stack pointer corresponde al stack frame (MSP o PSP).
- También indica en qué modo de operación estaba el procesador antes de entrar al handler de la excepción.

Entrada y retorno de excepciones

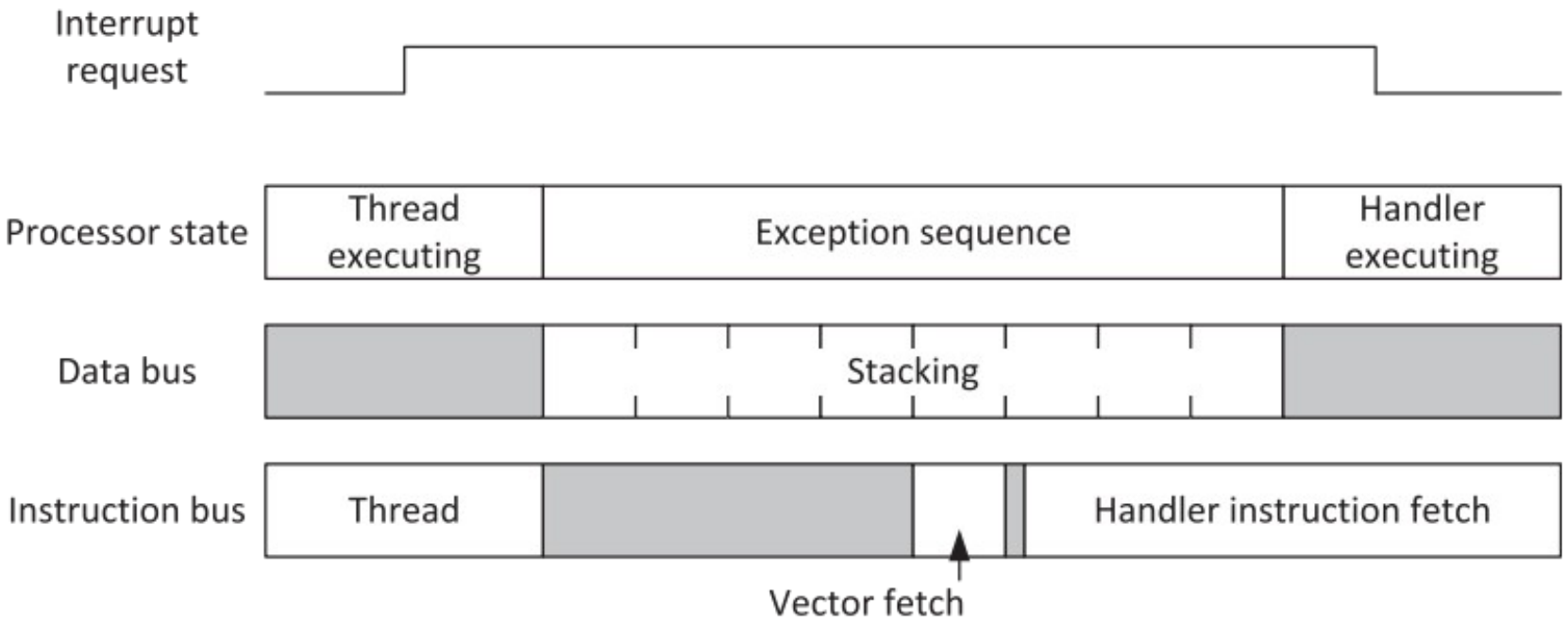
- Al terminar de atender la excepción, se carga el LR en el PC.
- Recordar que LR está cargado con el valor EXEC_RETURN.
- Este valor no es una dirección válida, sino un valor especial.
- Los bits [31:5] de EXEC_RETURN estan seteados a 1.
- Esto al ser cargado en el PC indica al CPU que la excepción fue atendida.
- El CPU comienza la secuencia de retorno de excepcion.

Entrada y retorno de excepciones

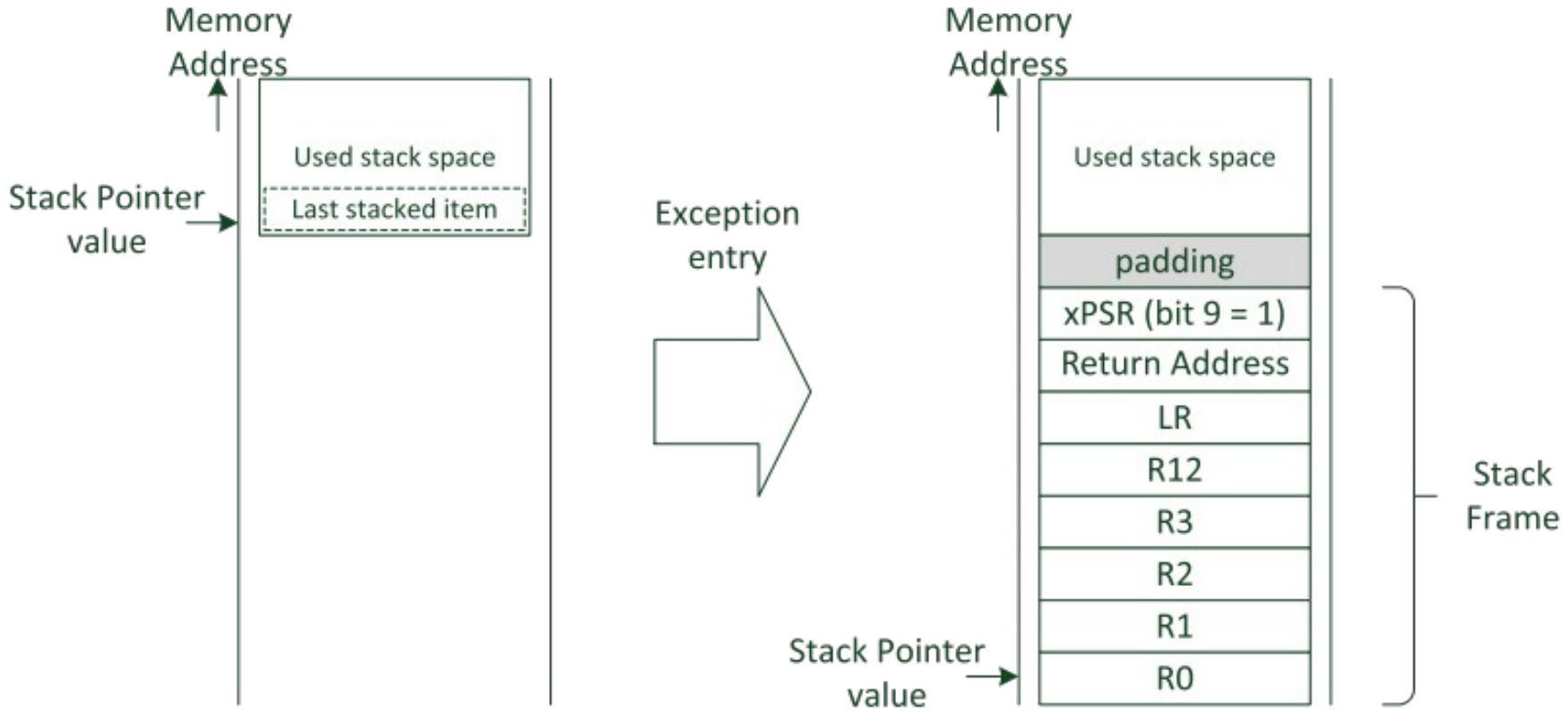
	Floating Point Unit was used before Interrupt (FPCA = 1)	**Floating Point Unit was not used before Interrupt (FPCA = 0)**
Return to Handler mode (always use Main Stack)	0xFFFFFFE1	0xFFFFFFF1
Return to Thread mode and use the Main Stack for return	0xFFFFFFE9	0xFFFFFFF9
Return to Thread mode and use the Process Stack for return	0xFFFFFFED	0xFFFFFFFD

Secuencia de stacking y unstacking

Entrada y retorno de excepciones



Entrada y retorno de excepciones



Entrada y retorno de excepciones

- Al terminar de atender la excepción, se hace POP del stack frame.
- **NOTAS**:
  - bit[9] de xPSR debe ser 1 para habilitar alineación double-word.
  - Esto es requerido para cumplir el AAPCS.
  - Esta alineación puede ser desactivada (no se hace padding).
  - Para esta materia, utilizaremos el estándar AAPCS.

Gracias.