



# Introducción Cortex– M

Juan Alarcón.  
[jalarcon@electron.frba.utn.edu.ar](mailto:jalarcon@electron.frba.utn.edu.ar)

# Procesadores ARM

■ El nombre ARM proviene de ***Acorn RISC Machine*** renombrada en 1990 a ***Advanced RISC Machine***.

- *Acorn fue la empresa que desarrolló el primer procesador ARM en 1985.*
- *Hace enfasis en la arquitectura RISC (reduced instruction set computer) en vez de CISC (complex instruction set computer).*

# Procesadore ARM

## History of ARM

Joint venture between Acorn Computers and Apple



1990

Designed into first mobile phones and then smartphones



1993 onwards

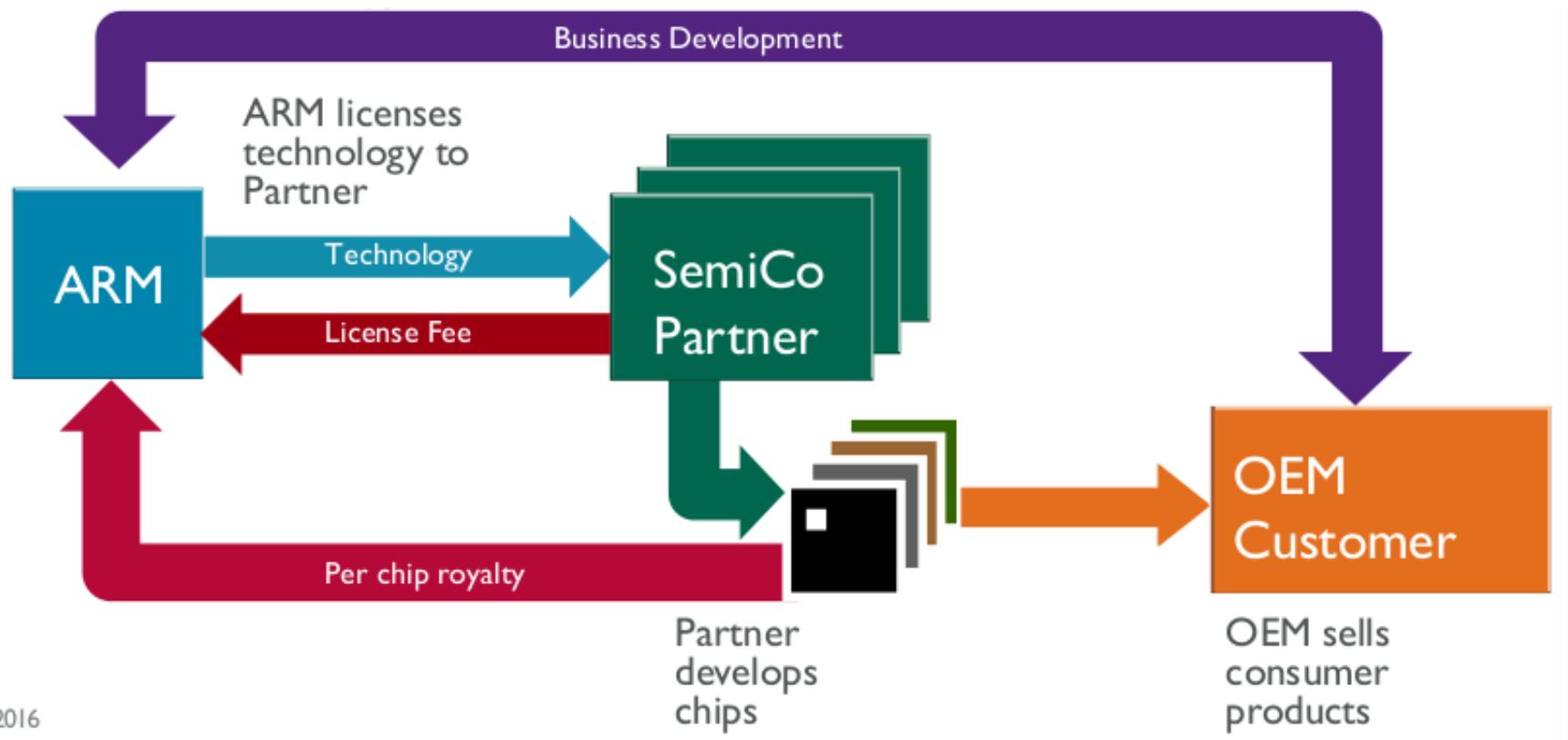
Now all electronic devices can use smart ARM technology



Today

ARM

# Procesadores ARM.



© ARM 2016

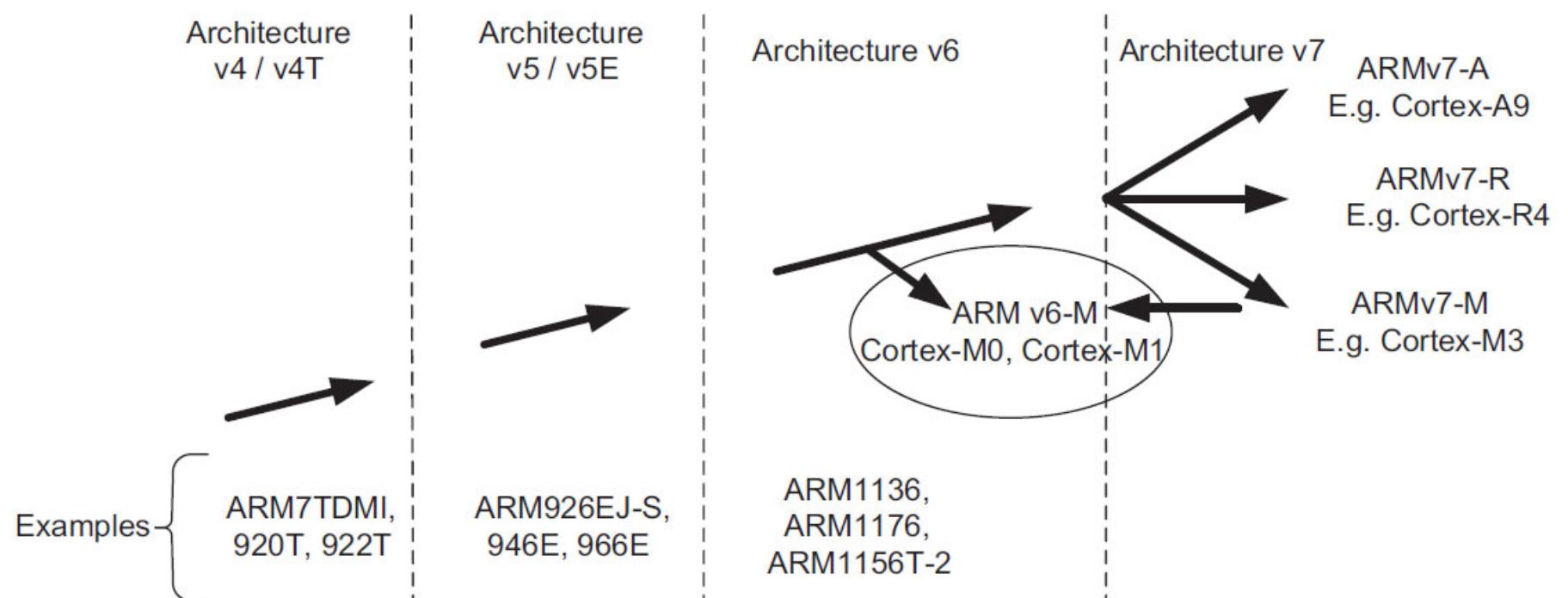
# Procesadores ARM

- Procesadores de 32 bits (registros, bus de datos, bus de memoria).
- Procesadores con set de instrucciones RISC (reduced instruction set computer).
- Arquitectura Harvard (Cortex-M3). Arquitectura Von Neumann (Cortex-M0).
- Definen modos de operación (sistema – usuario definido por hardware).

# Características Generales.

- Arquitectura Load/Store.
- Instrucciones de tamaño fijo.
- Máquina de 3 direcciones.
- La mayoría de las instrucciones son de un único ciclo de reloj.
- Ejecución condicional de instrucciones.
- Barrel shifter de 32 bits.

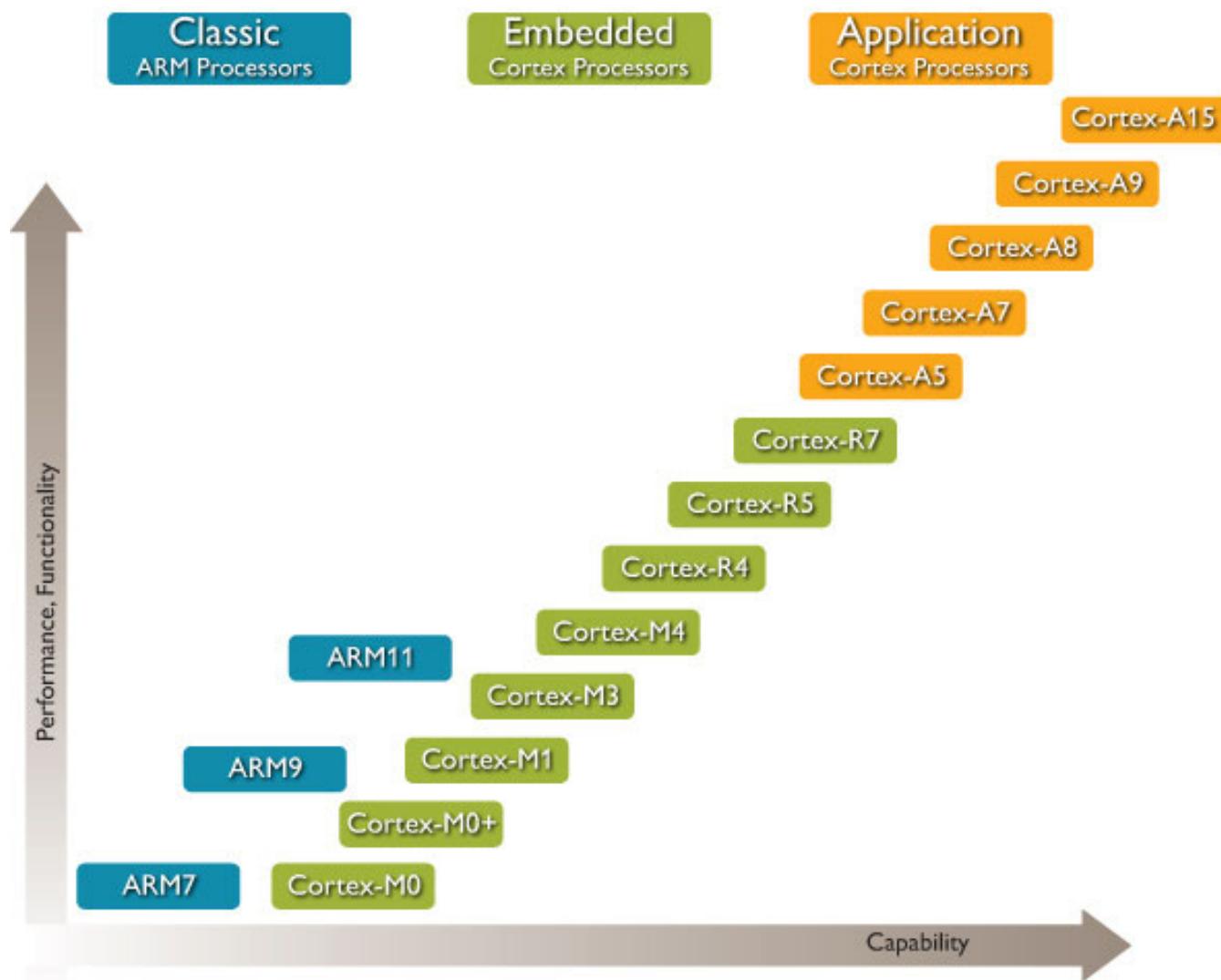
# Arquitecturas. ARM



# ARM - Cortex.

- Los procesadores Cortex son procesadores ARM (<http://www.arm.com>) de arquitectura v7.
- La arquitectura v7 tiene a su vez tres versiones.
  - v7-A. Application.
  - v7-R. Real-Time.
  - v7-M. Microcontroller.

# Procesadores. ARM



# Cortex. Versiones

- **ARMv7-A. Application.** Procesadores que requieren ejecutar aplicaciones complejas de alto rendimiento, tales como Linux, Symbian, Android. Requieren unidad de administración de memoria (MMU). Por ejemplo celulares.
- **ARMv7-R. Real-Time.** Procesadores orientados a aplicaciones de alto rendimiento, alta confiabilidad y tiempo real (dar una respuesta en un período garantizado de tiempo). Por ejemplo controladores de disco duros, sistemas
- **ARMv7-M. Microcontroller.** Procesadores de bajo costo , bajo consumo y baja latencia de interrupciones. Procesadores orientados a usos de microcontroladores.

# Cortex. Versiones

	Cortex-A	Cortex-R	Cortex-M
Architecture type	Support both 64 and 32-bit from Armv8-A, 32-bit in Armv7-A and older architecture	Support both 64 and 32-bit from Armv8-R, 32-bit in Armv7-R and older architecture	32-bit only
Clock frequency range and pipeline	Longer pipeline optimized for high clock frequency range	Medium-length pipeline (e.g., 8-stage in Cortex-R5)	Short to medium length pipeline (2 to 6 stages) for low-power systems
Virtual memory support (required for Linux)	Yes	No (it is permitted in Armv8-R, but not supported in current Cortex-R processors)	No
Virtualization support	Yes	Yes, from Armv8-R (e.g., Cortex-R52)	No
Arm TrustZone security extension	Yes	No	Yes, from Armv8-M, but not in Armv6-M and Armv7-M architectures
Interrupt handling	Based on Generic Interrupt Controller (GIC) with multi-core and virtualization support. Non-deterministic interrupt response speed.	Based on Generic Interrupt Controller with multi-core and virtualization support, or Vectored Interrupt Controller in older Cortex-R. Fast interrupt response.	Based on Nested Vectored Interrupt Controller (NVIC) internal to the processor. Low interrupt latency and easy to use.
ISA for DSP acceleration	Neon Advanced SIMD (128-bit vectored processing). Latest architecture from Armv8.3-A supports Scalable Vector Extension (SVE).	Neon Advanced SIMD support on Armv8-R. Also, support legacy SIMD (32-bit vector processing).	Support legacy SIMD (32-bit vector processing) in Cortex-M4, Cortex-M7, Cortex-M33, and Cortex-M35P

# Cortex. Versiones

- Sí necesito un sistema que corra Linux (Red, Sistema de archivos avanzado, SSH, servicios web), ¿Qué tipo de Cortex tendría que elegir?
- Se va a desarrollar un sistema alimentado a batería que se encarga de adquisición de señales analógicas para enviar de manera inalámbrica ¿Qué tipo de procesador elijo?
- Para el desarrollo del controlador de llama de una caldera de gran porte ¿Qué tipo de procesador elijo?

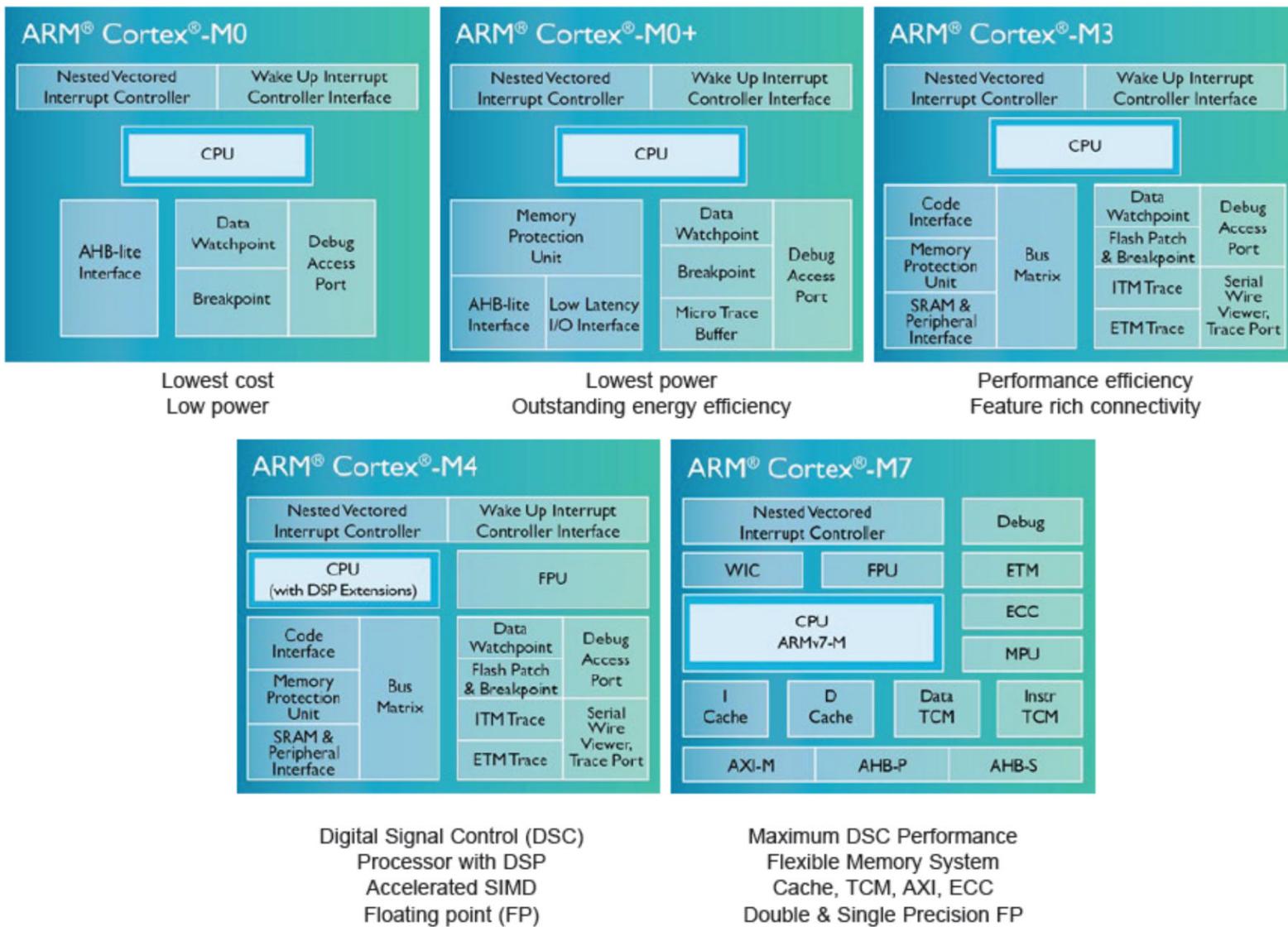
# Cortex-M.

- **Cortex-M0.** Familia de microcontroladores muy pequeños, implementados en la arquitectura ARMv6-M. (LPC1114).
- **Cortex-M0+.** Familia de microcontroladores muy pequeños y de mejor eficiencia energética que los M0 (LPC812).
- **Cortex-M1.** Familia de microcontroladores implementados como soft-cores (IP-cores en FPGA). Pertenece a la arquitectura ARMv6-M. Por ejemplo Altera ARM Cortex-M1.
- **Cortex-M3.** Microcontroladores de arquitectura ARMv7-M. Agregan multiplicación por hardware en un ciclo de reloj, división por hardware y matemática saturada. LPC1769.
- **Cortex-M4.** Microcontroladores de arquitectura ARMv7-M, similar a los Cortex-M3 pero con extensiones para DSP. Multiplicación y suma de 32 bits en un único ciclo de reloj. LPC4300 .
- **Cortex-M7.** Son la familia de mayor rendimiento de los Cortex M. Tienen pipeline de 6 etapas. Instrucciones SIMD y bus de 64bits.

# Cortex-M.

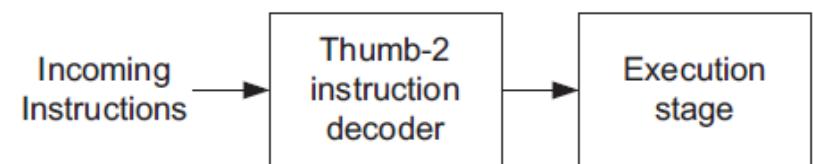
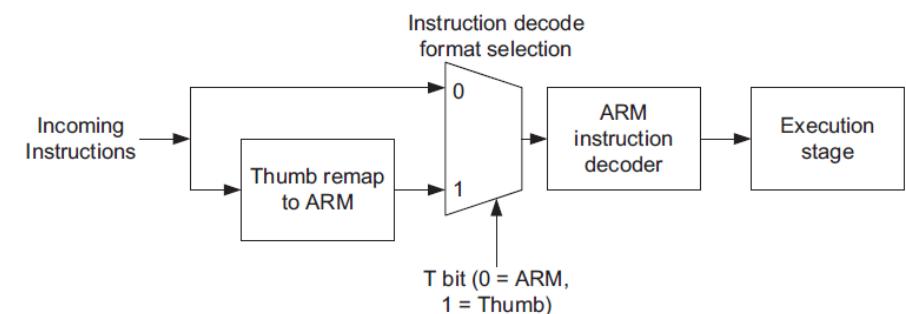
- **Cortex-M0.** Familia de microcontroladores muy pequeños, implementados en la arquitectura ARMv6-M. (LPC1114).
- **Cortex-M0+.** Familia de microcontroladores muy pequeños y de mejor eficiencia energética que los M0 (LPC812).
- **Cortex-M1.** Familia de microcontroladores implementados como soft-cores (IP-cores en FPGA). Pertenece a la arquitectura ARMv6-M. Por ejemplo Altera ARM Cortex-M1.
- **Cortex-M3.** Microcontroladores de arquitectura ARMv7-M. Agregan multiplicación por hardware en un ciclo de reloj, división por hardware y matemática saturada. LPC1769.
- **Cortex-M4.** Microcontroladores de arquitectura ARMv7-M, similar a los Cortex-M3 pero con extensiones para DSP. Multiplicación y suma de 32 bits en un único ciclo de reloj. LPC4300 .
- **Cortex-M7.** Son la familia de mayor rendimiento de los Cortex M. Tienen pipeline de 6 etapas. Instrucciones SIMD y bus de 64bits.

# Cortex-M. Aplicaciones

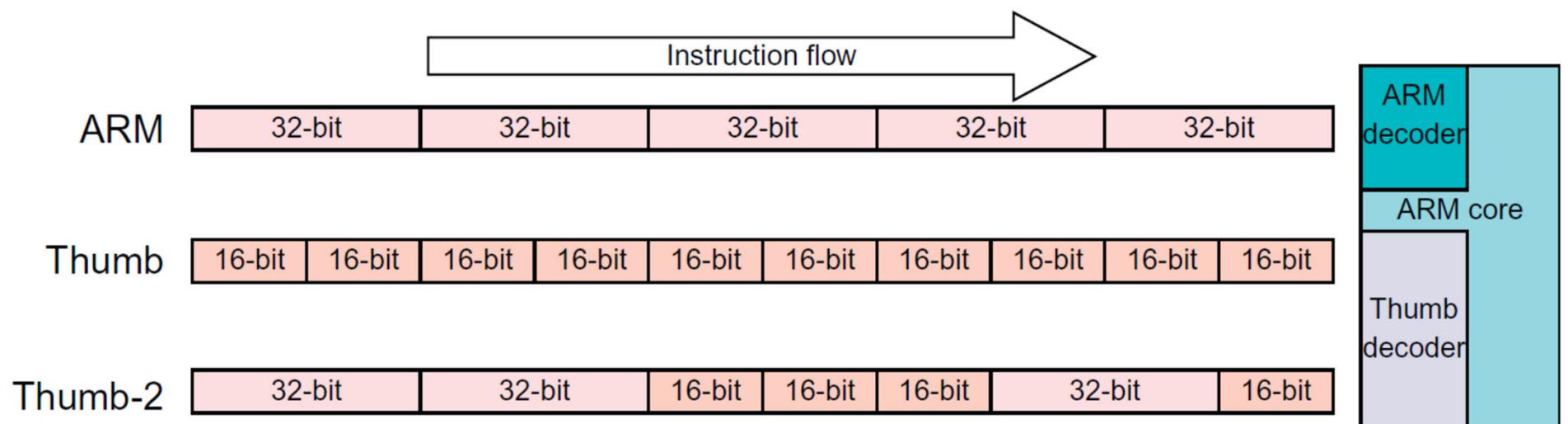


# Cortex-M. Set de Instrucciones

- En los primeros procesadores ARM existía un único set de instrucciones de 32bits. El problema que se presenta para los procesadores pequeños es que se requiere mucha más memoria de programa que para un procesador de 8 o 16bits para la misma tarea
- En el ARM7TDMI se soportaba el set de instrucciones ARM (32 bits) y el Thumb (16 bits) más limitado que el ARM
- Los Cortex-M usan el set de instrucciones Thumb-2 que tiene la mayor parte de las instrucciones de 16 bits y otras de 32bits

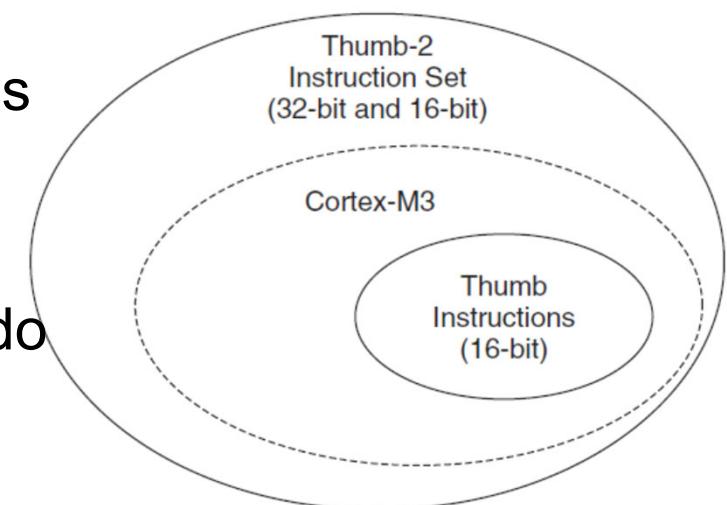


# Set de Instrucciones.

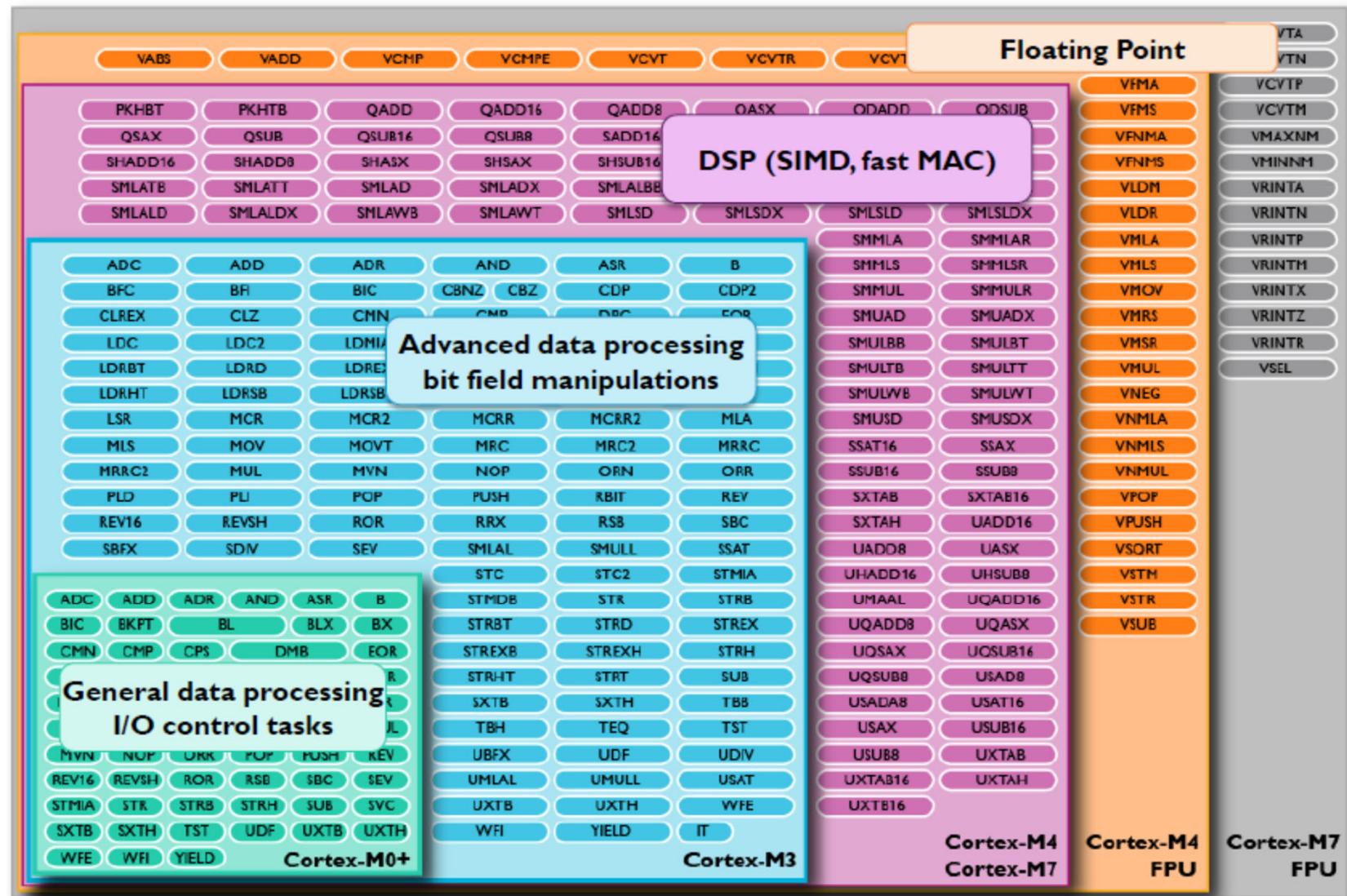


# Set de Instrucciones Thumb-2

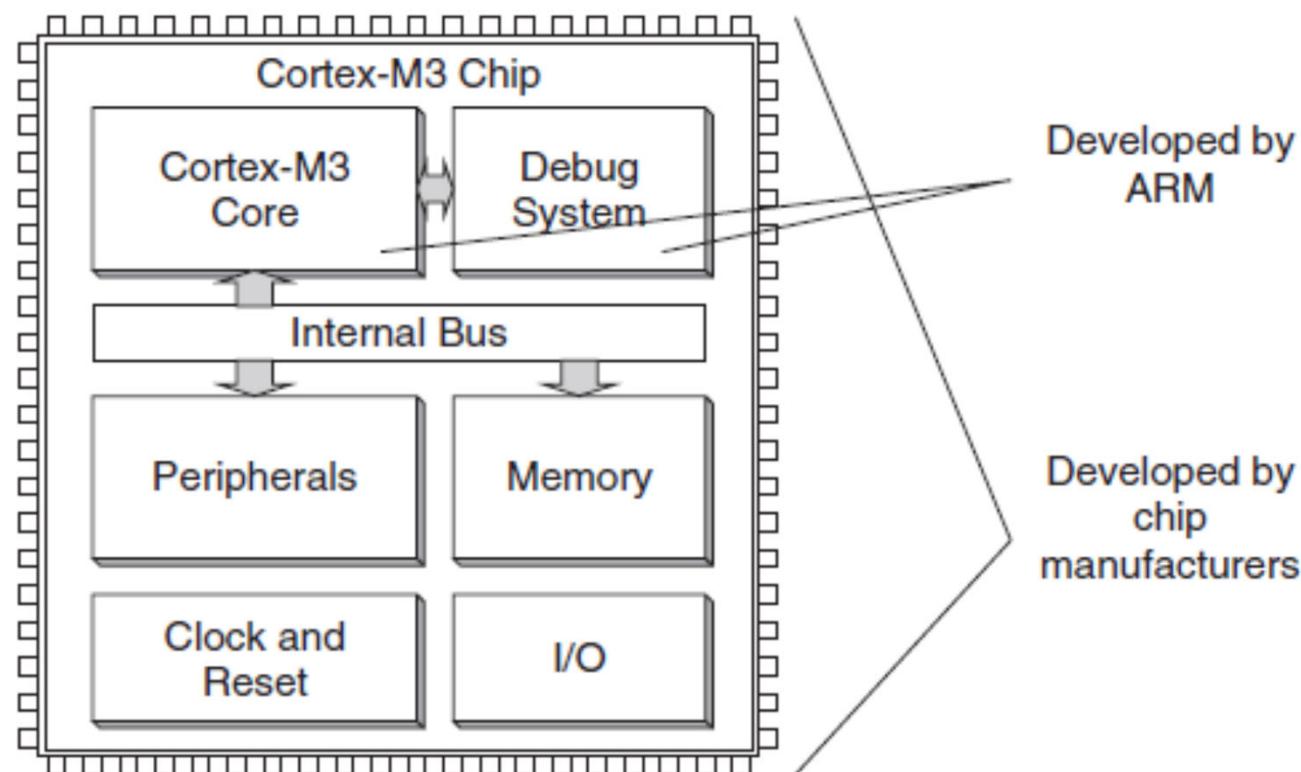
- Thumb-2. es un set de instrucciones de las instrucciones más comunes de ARM de 32bits en 16 bits.
- En ejecución el código de operación se “mapean” a instrucciones de 32 bits, siendo transparente al procesador.
- El uso del set de instrucciones Thumb-2 permite un mejor aprovechamiento de la memoria flash.
- Thumb-2, utiliza un conjunto de instrucciones mixto de 32bits y 16bits, descomprimiendo las de 16 bits a 32bits.



# Set de instrucciones Thumb-2.



# Cortex-M3. Generalidades.



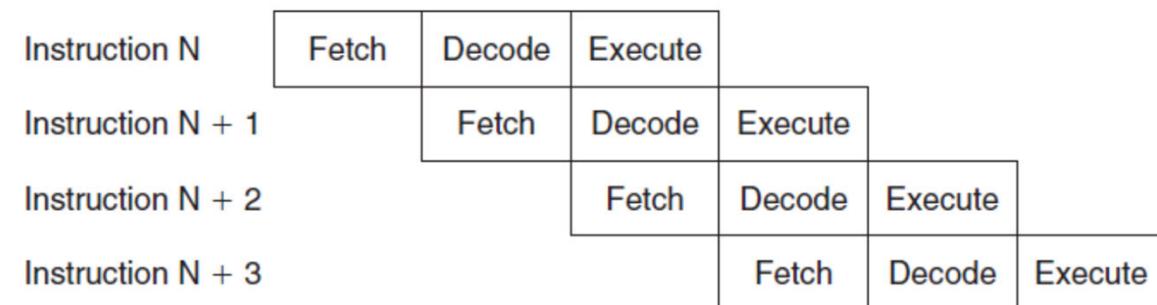
# Cortex-M3. Generalidades.

- Procesador
  - Set de instrucciones Thumb-2.
  - Pipeline de 3 etapas
  - Arquitectura Harvard con búsqueda en memoria de código y en memoria de datos simultánea.
  - Multiplicación de 32 bits en un solo ciclo de reloj.
  - División de 32 bits por hardware en 2-12 ciclos de reloj.
  - Modo Handler y Modo Thread.
  - Soporte de hasta 240 interrupciones por medio del NVIC.
  - Latencia de interrupciones de 12 ciclos de reloj.
- Registros
  - 13 registros de propósito general de 32 bits.
  - Link Register (LR)
  - Program Counter (PC)
  - Program Status Register (xPSR)
  - Dos registros de puntero a pila (SP)



# Cortex-M3. Pipeline

- Para poder alcanzar altas velocidades de clock se divide lógica combinacional en bloques de lógica más pequeña separada por FF, minimizando los caminos críticos.
- Los procesadores Cortex implementan esta técnica en un pipeline de 3 etapas.



# ¿Por qué usar pipeline? Ejemplo (sumar 4 número)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity suma is
  Port ( A : in STD_LOGIC_VECTOR (7 downto 0);
         B : in STD_LOGIC_VECTOR (7 downto 0);
         C : in STD_LOGIC_VECTOR (7 downto 0);
         D : in STD_LOGIC_VECTOR (7 downto 0);
         S : out STD_LOGIC_VECTOR (9 downto 0));
end suma;

architecture Behavioral of suma is
  signal ab, cd : unsigned(8 downto 0);
  signal abcd    : unsigned(9 downto 0);
begin
  ab  <= unsigned("0"&A) + unsigned("0"&B);
  cd  <= unsigned("0"&C) + unsigned("0"&D);
  abcd <= unsigned("0"&ab) + unsigned("0"&cd);
  S    <= std_logic_vector(abcd);
end Behavioral;
```

Clock Information:  
-----  
No clock signals found in this design

Asynchronous Control Signals Information:  
-----  
No asynchronous control signals found in this design

Timing Summary:  
-----  
Speed Grade: -5

Minimum period: No path found  
Minimum input arrival time before clock: No path found  
Maximum output required time after clock: No path found  
Maximum combinational path delay: 13.329ns

# ¿Por qué usar pipeline? Ejemplo (sumar 4 número)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity suma_registrada is
    Port ( clk : in STD_LOGIC;
            A : in STD_LOGIC_VECTOR (7 downto 0);
            B : in STD_LOGIC_VECTOR (7 downto 0);
            C : in STD_LOGIC_VECTOR (7 downto 0);
            D : in STD_LOGIC_VECTOR (7 downto 0);
            S : out STD_LOGIC_VECTOR (9 downto 0));
end suma_registrada;

architecture Behavioral of suma_registrada is
    signal ab, cd : unsigned(8 downto 0);
    signal abcd : unsigned(9 downto 0);

begin
    process(clk)
    begin
        if(rising_edge(clk)) then
            ab <= unsigned("0"&A) + unsigned("0"&B);
            cd <= unsigned("0"&C) + unsigned("0"&D);
            abcd <= unsigned("0"&ab) + unsigned("0"&cd);
        end if;
    end process;
    S <= std_logic_vector(abcd);
end Behavioral;
```

```
=====
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
-----
+-----+-----+
| Clock buffer(FF name) | Load |
+-----+-----+
clk | BUFGP | 28 |
+-----+-----+

Asynchronous Control Signals Information:
-----
No asynchronous control signals found in this design

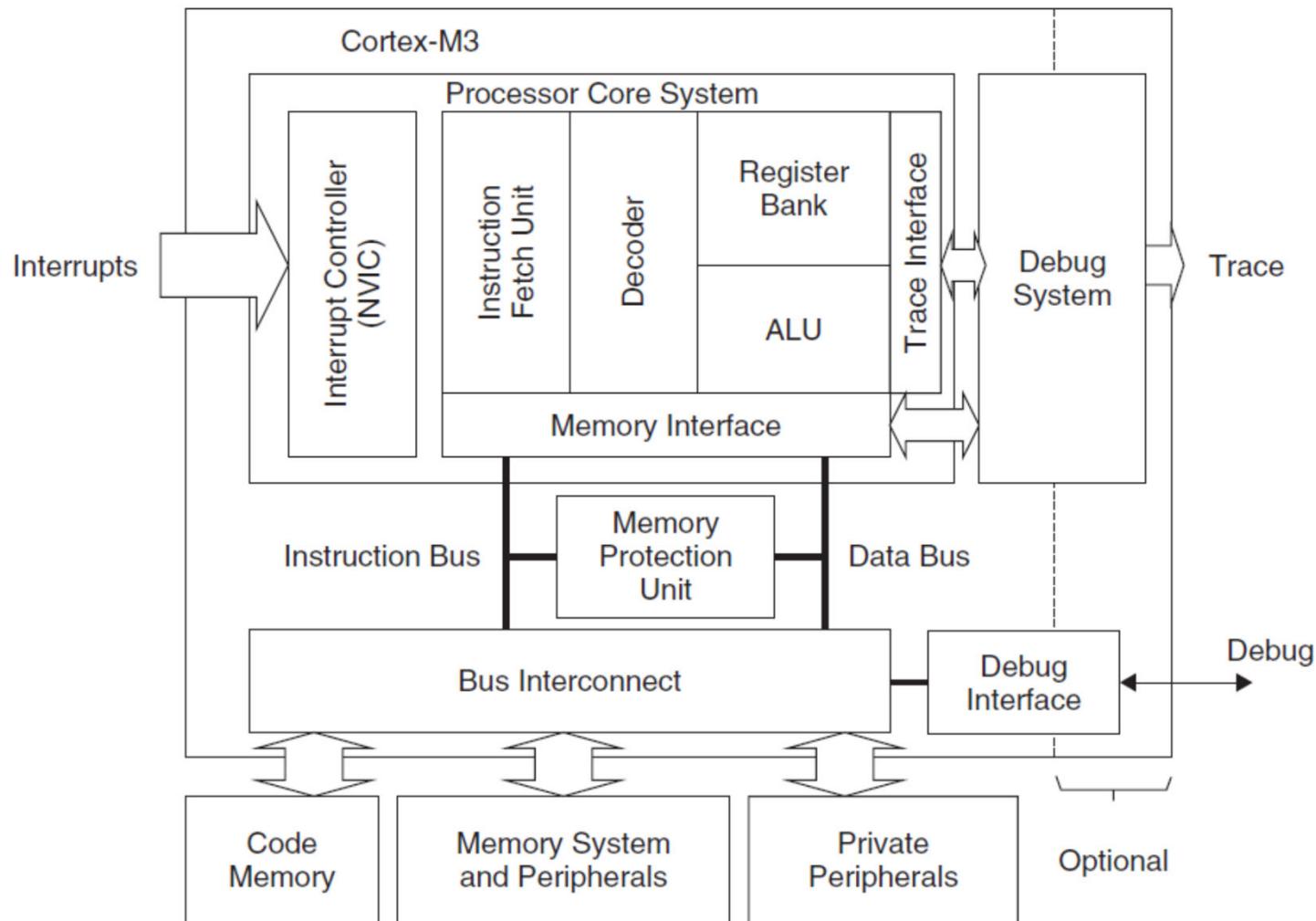
Timing Summary:
-----
Speed Grade: -5

Minimum period: 3.866ns (Maximum Frequency: 258.642MHz)
Minimum input arrival time before clock: 3.900ns
Maximum output required time after clock: 6.216ns
Maximum combinational path delay: No path found

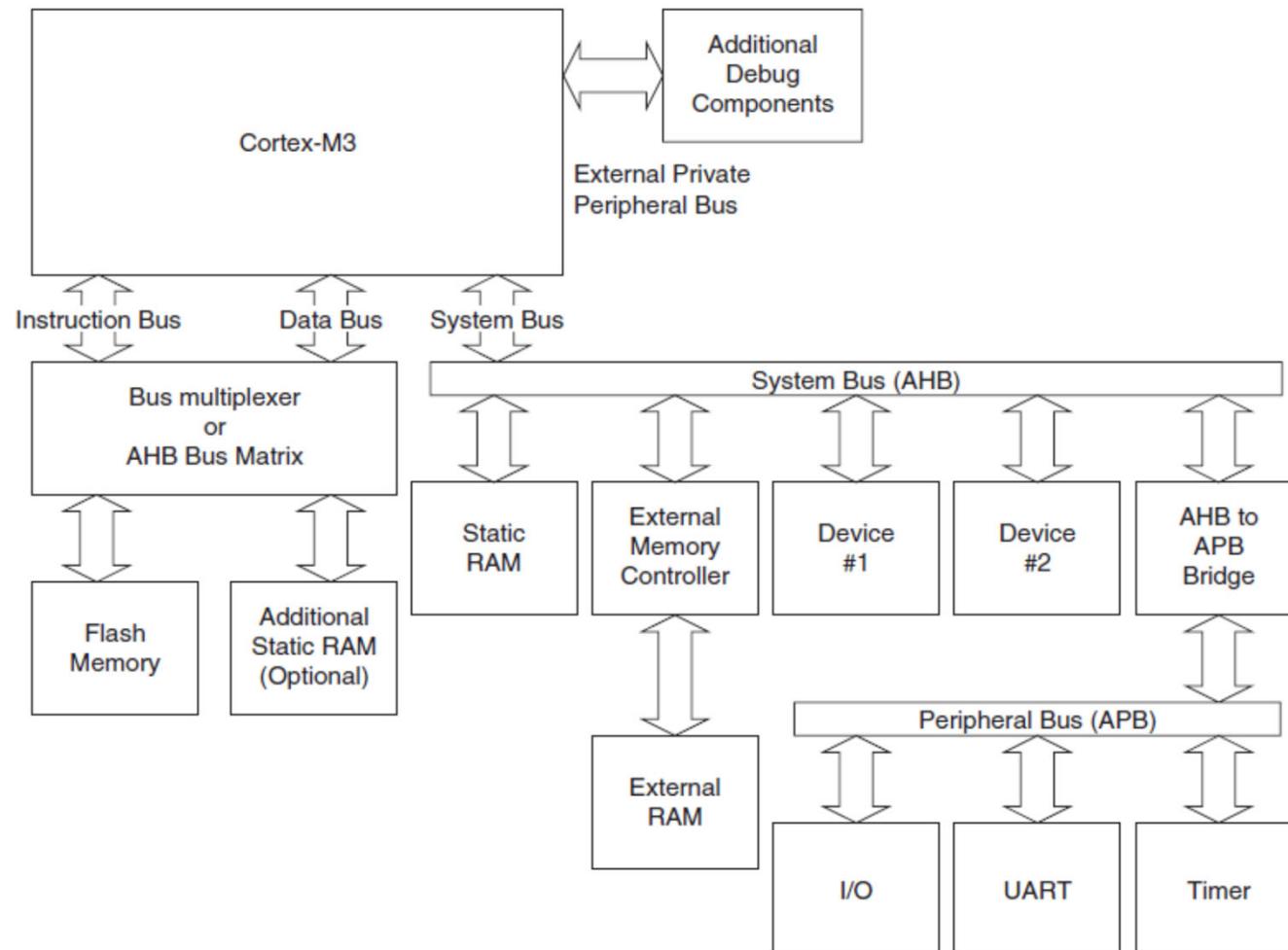
Timing Detail:
-----
All values displayed in nanoseconds (ns)
=====
```

- ¿Cuánto se demora en que el resultado esté a la salida?

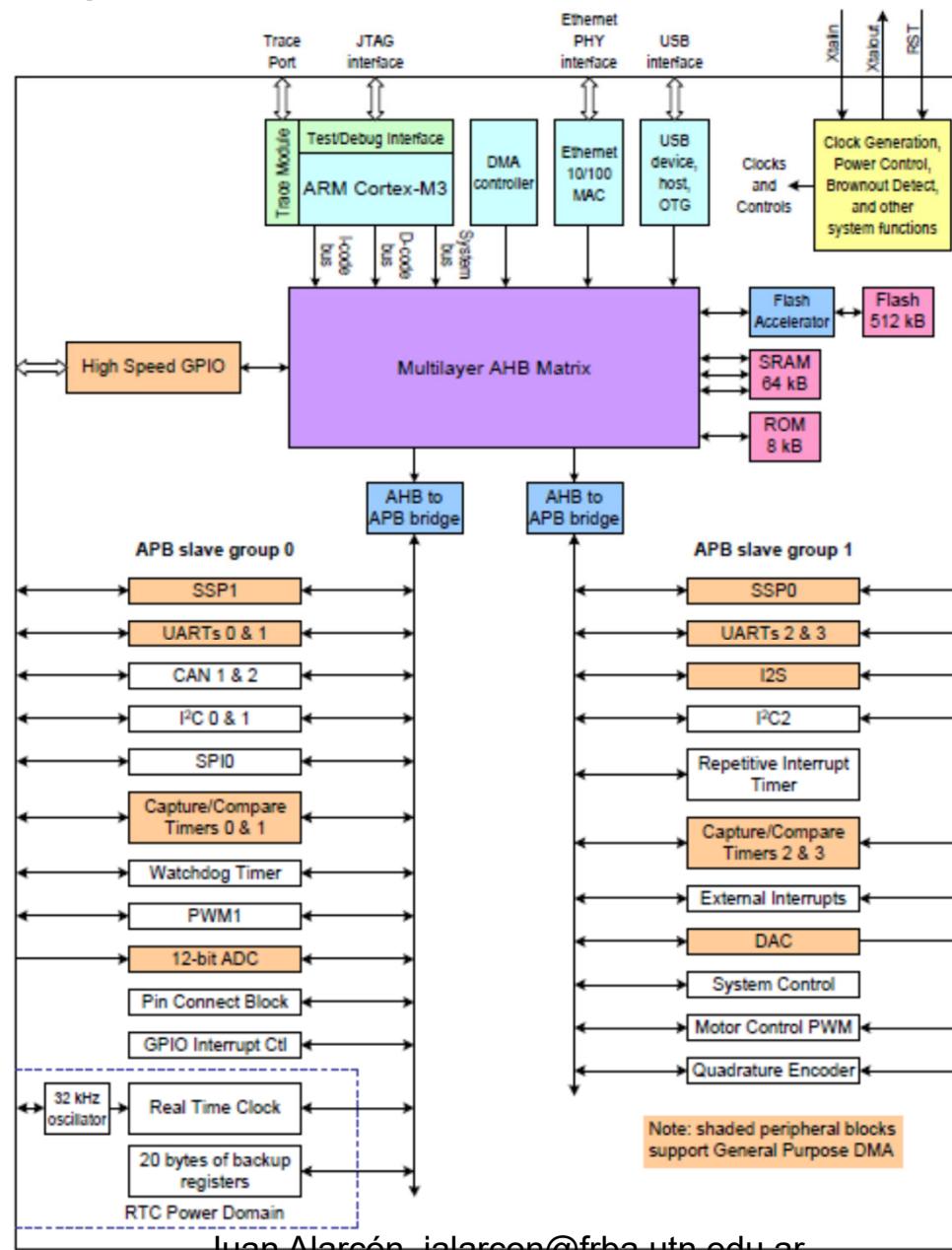
# Núcleo del Cortex-M3.



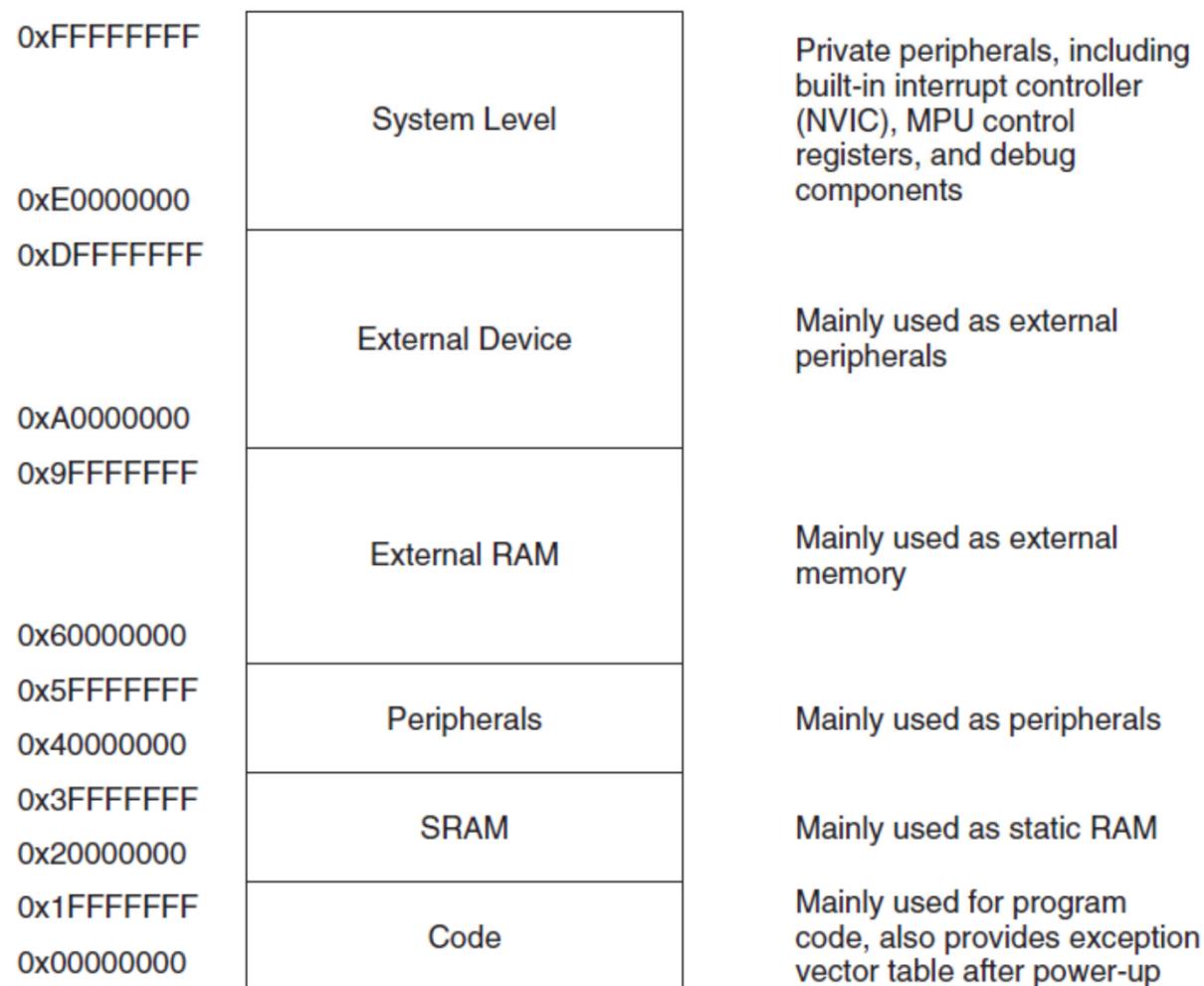
# Buses. Cortex – M3.



# Ejemplo. Cortex-M3



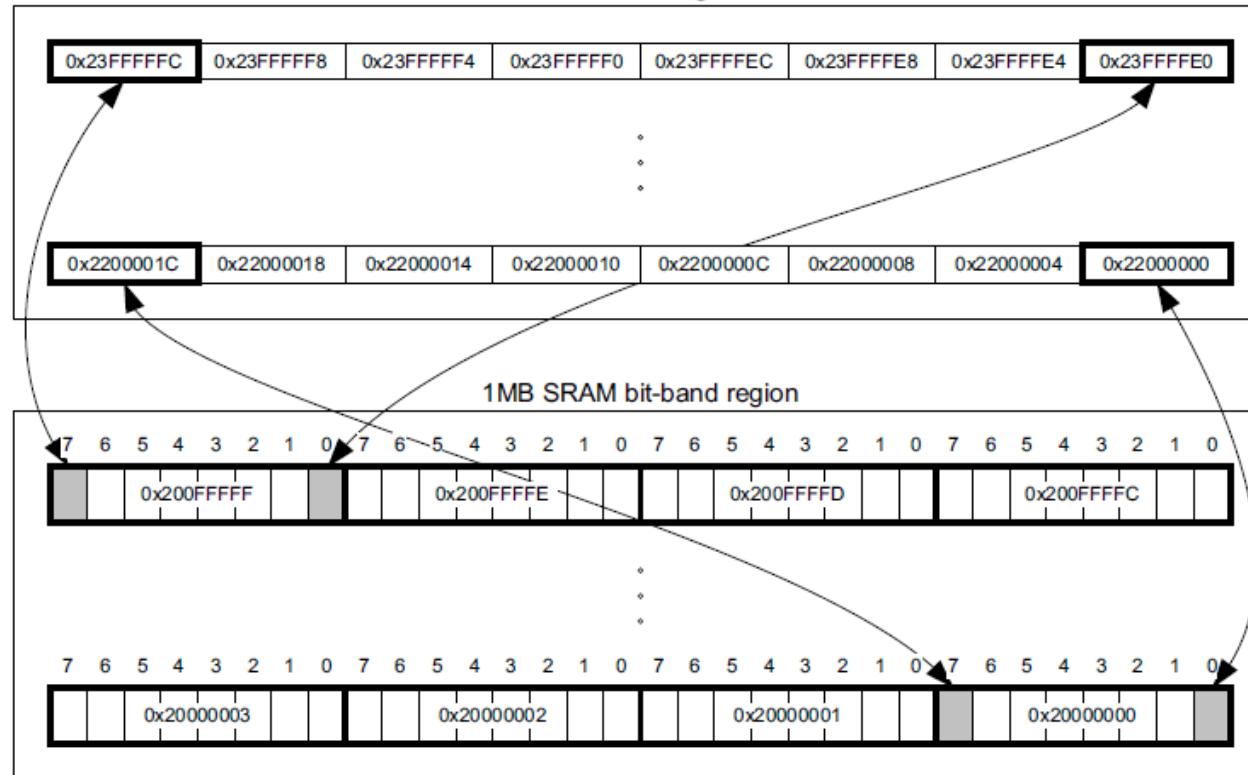
# Mapa de memoria.



- Los Cortex-M3 tienen un mapa de memoria predefinido.
- El procesador direcciona 4Gb.
- Se definen zonas de acceso a nivel de bits (bit banding).

# Bit banding.

32MB alias region



```
bit_word_offset = (byte_offset x 32) + (bit_number x 4)
```

```
bit_word_addr = bit_band_base + bit_word_offset
```

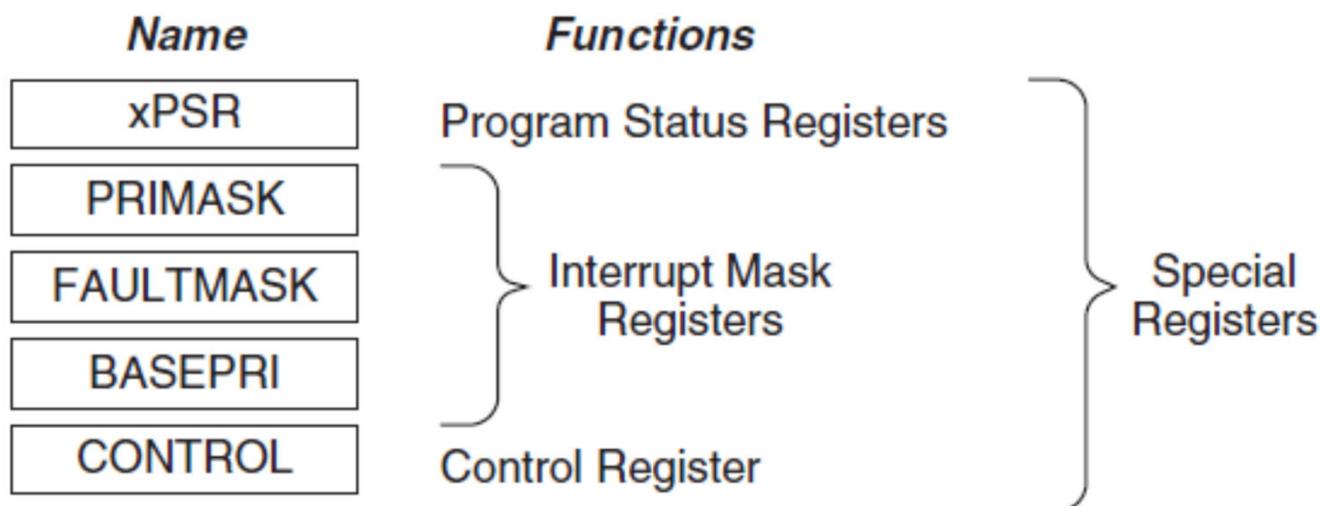
- Dos zonas de 1MB que son direccionables a nivel de bits por medio de dos zonas de “alias” de 32MB.

# Registros.

Name	Functions (and Banked Registers)
R0	General-Purpose Register
R1	General-Purpose Register
R2	General-Purpose Register
R3	General-Purpose Register
R4	General-Purpose Register
R5	General-Purpose Register
R6	General-Purpose Register
R7	General-Purpose Register
R8	General-Purpose Register
R9	General-Purpose Register
R10	General-Purpose Register
R11	General-Purpose Register
R12	General-Purpose Register
R13 (MSP)	Main Stack Pointer (MSP), Process Stack Pointer (PSP)
R13 (PSP)	
R14	Link Register (LR)
R15	Program Counter (PC)

- Algunas de las instrucciones Thumb de 16 bits sólo pueden acceder a los registros bajos (R0-R7).
- MSP es el puntero a la pila para las excepciones y modo privilegiado.
- PSP puntero a la pila del código de aplicación.
- LR. Cuando se llama a una subrutina, LR contiene la dirección de retorno.

# Registros Especiales.



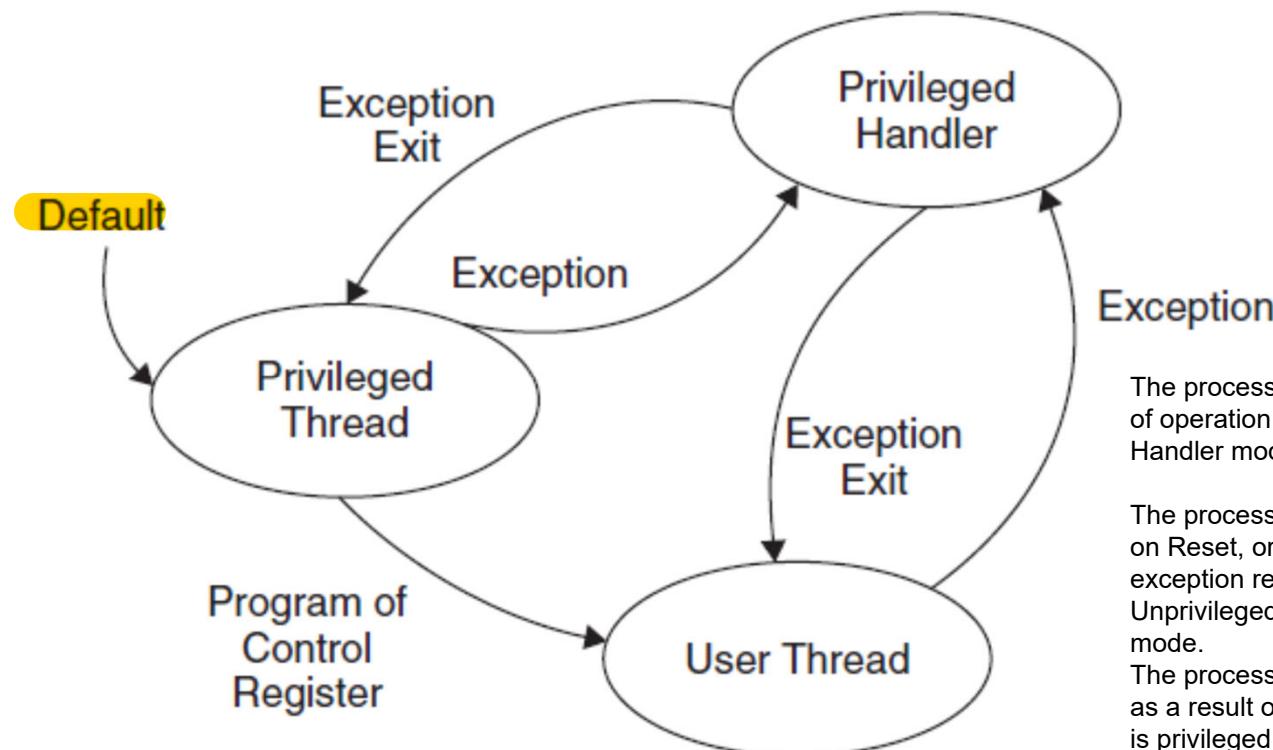
Register	Function
xPSR	Provide ALU flags (zero flag, carry flag), execution status, and current executing interrupt number
PRIMASK	Disable all interrupts except the nonmaskable interrupt (NMI) and HardFault
FAULTMASK	Disable all interrupts except the NMI
BASEPRI	Disable all interrupts of specific priority level or lower priority level
CONTROL	Define privileged status and stack pointer selection

# Registros Especiales.

	31	30	29	28	27	26:25	24	23:20	19:16	15:10	9	8	7	6	5	4:0	
xPSR	N	Z	C	V	Q	ICI/IT	T			ICI/IT		Exception Number					

Bit	Function
CONTROL[1]	Stack status: 1 = Alternate stack is used 0 = Default stack (MSP) is used If it is in the Thread or base level, the alternate stack is the PSP. There is no alternate stack for handler mode, so this bit must be zero when the processor is in handler mode.
CONTROL[0]	0 = Privileged in Thread mode 1 = User state in Thread mode If in handler mode (not Thread mode), the processor operates in privileged mode.

# Modos de operación



The processor supports two modes of operation, Thread mode and Handler mode:

The processor enters Thread mode on Reset, or as a result of an exception return. Privileged and Unprivileged code can run in Thread mode.

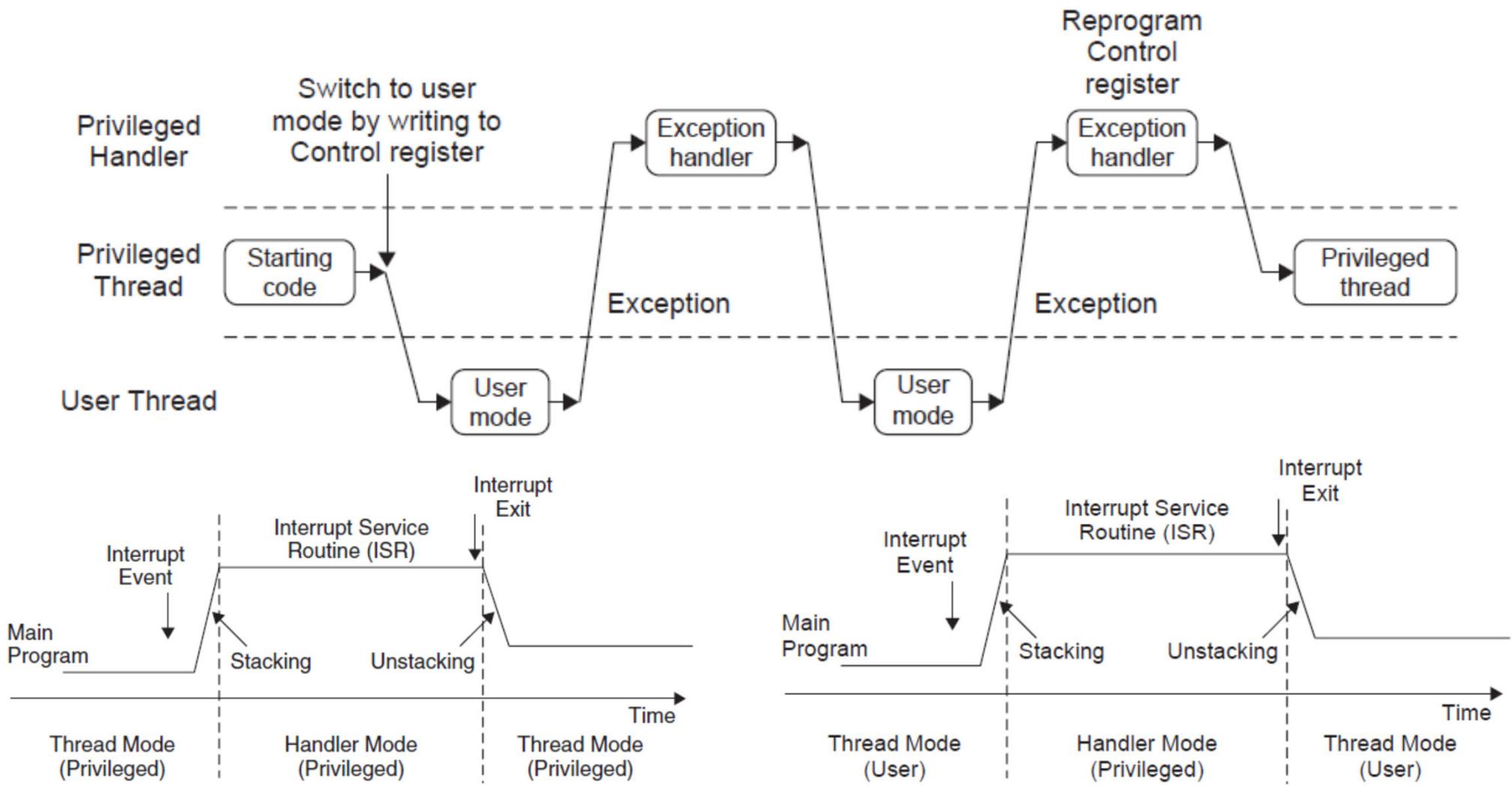
The processor enters Handler mode as a result of an exception. All code is privileged in Handler mode.

*When running an exception*

*When running main program*

Privileged	User
Handler Mode	
Thread Mode	Thread Mode

# Modos de operación (2).



# Excepciones. Interrupciones.

- **Interrupción.** Evento *relacionado con el hardware* que altera el flujo normal del programa en ejecución.
- **Excepción.** Evento que altera el flujo normal de un programa en ejecución. El intento de acceder a memoria no válida es un caso típico de excepción.
- Todos los Cortex-M3 tienen el *mismo* controlador de excepciones e interrupciones el **NVIC** –Nested Vectored Interrupt Controller(Controlador de Interrupciones Anidadas y Vectorizadas).
- El vector de excepciones e interrupciones admite 256 entradas. Las primeras 16 son estándar de la arquitectura, y las restantes 240 son específicas del microcontrolador, definidas por el fabricante.

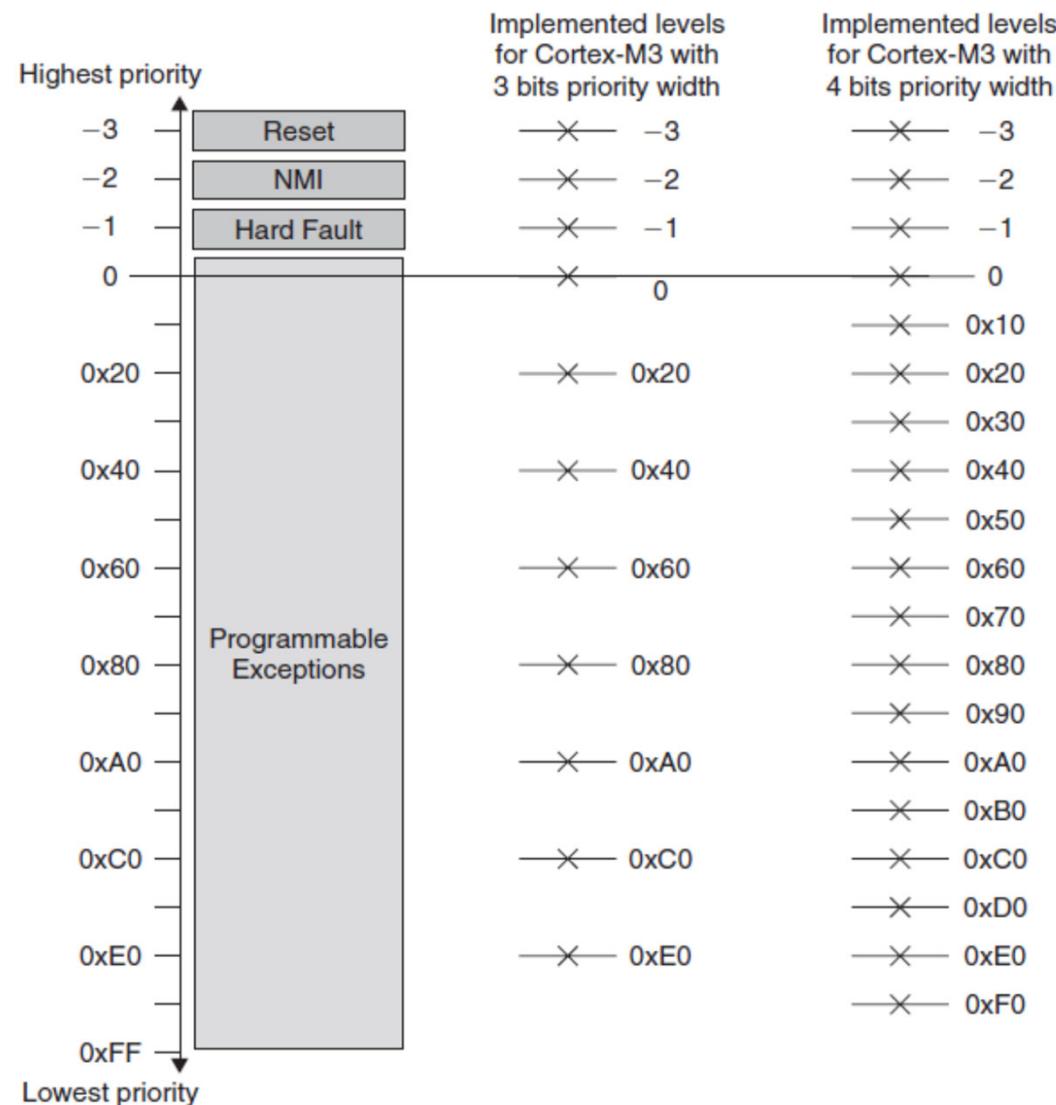
# Excepciones.

Exception Number	Exception Type	Priority (Default to 0 if Programmable)	Description
0	NA	NA	No exception running
1	Reset	-3 (Highest)	Reset
2	NMI	-2	NMI (external NMI input)
3	Hard fault	-1	All fault conditions, if the corresponding fault handler is not enabled
4	MemManage fault	Programmable	Memory management fault; MPU violation or access to illegal locations
5	Bus fault	Programmable	Bus error (prefetch abort or data abort)
6	Usage fault	Programmable	Program error
7–10	Reserved	NA	Reserved
11	SVCall	Programmable	Supervisor call
12	Debug monitor	Programmable	Debug monitor (break points, breakpoints, or external debug request)
13	Reserved	NA	Reserved
14	PendSV	Programmable	Pendable request for system service
15	SYSTICK	Programmable	System tick timer
16	IRQ #0	Programmable	External interrupt #0
17	IRQ #1	Programmable	External interrupt #1
...	...	...	...
255	IRQ #239	Programmable	External interrupt #239

# Vector de Excepciones.

Exception Type	Address Offset	Exception Vector
18-255	0x48-0x3FF	IRQ #2-239
17	0x44	IRQ #1
16	0x40	IRQ #0
15	0x3C	SYSTICK
14	0x38	PendSV
13	0x34	Reserved
12	0x30	Debug Monitor
11	0x2C	SVC
7-10	0x1C-0x28	Reserved
6	0x18	Usage fault
5	0x14	Bus fault
4	0x10	MemManage fault
3	0x0C	Hard fault
2	0x08	NMI
1	0x04	Reset
0	0x00	Starting value of the MSP

# Prioridades.

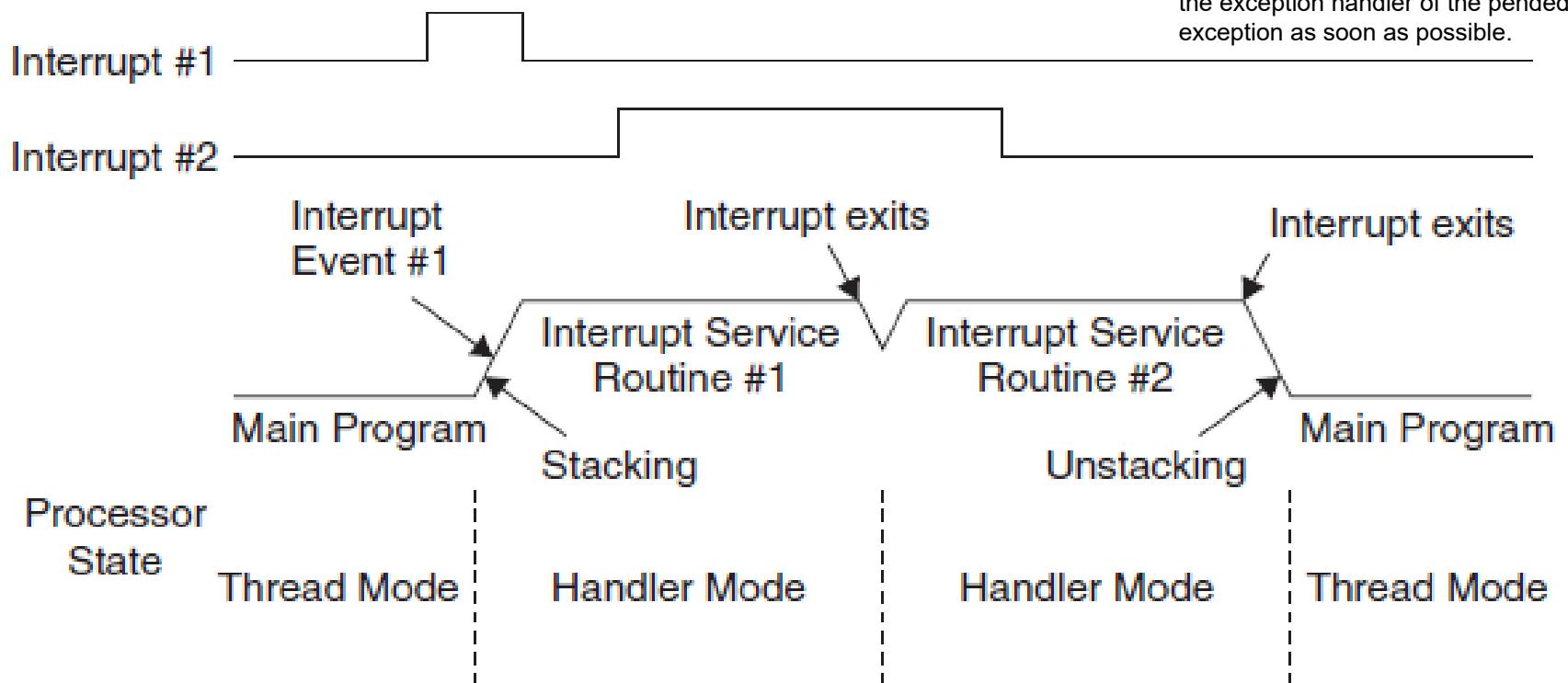


# Stacking

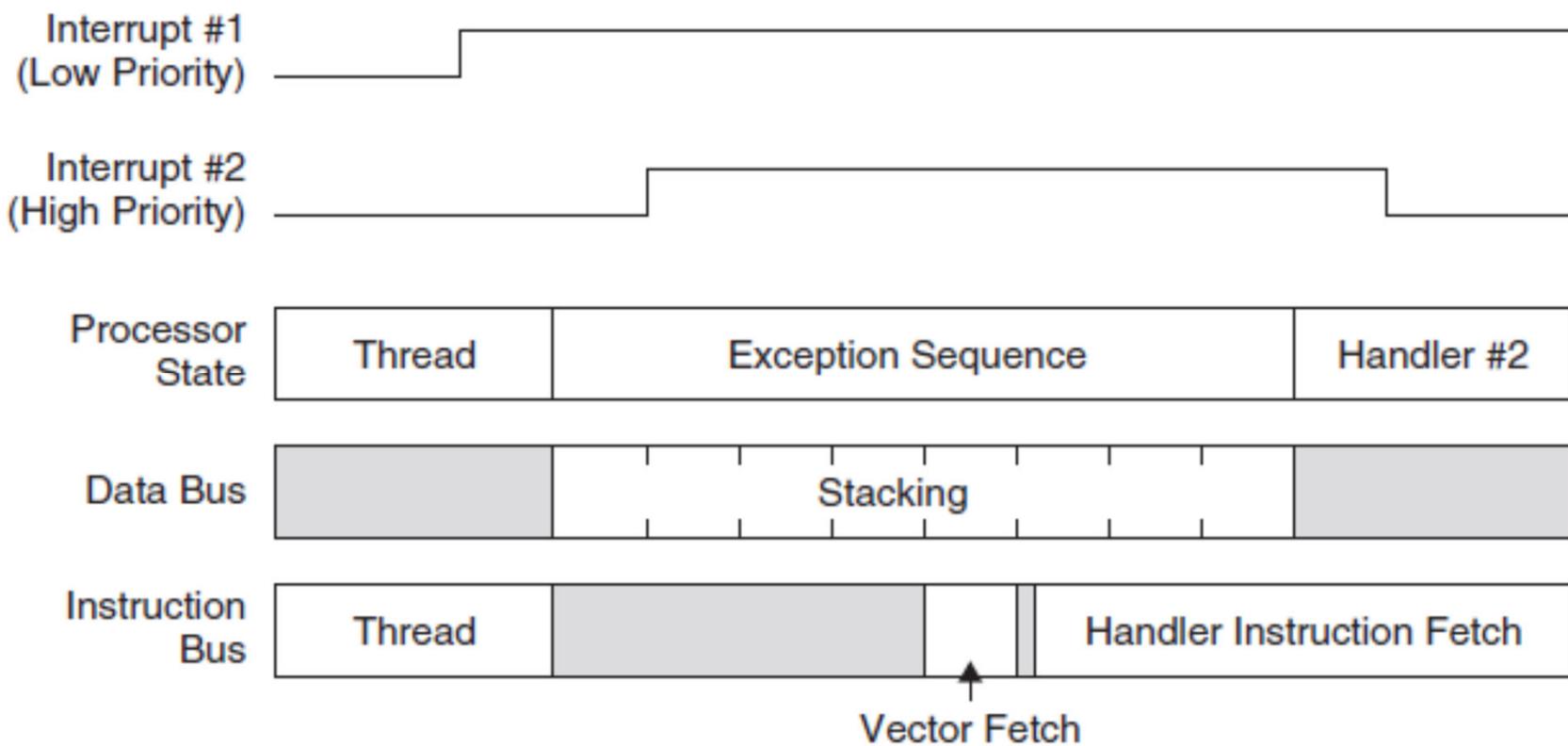
Address	Data	Push Order
Old SP (N) ->	(Previously pushed data)	-
(N-4)	PSR	2
(N-8)	PC	1
(N-12)	LR	8
(N-16)	R12	7
(N-20)	R3	6
(N-24)	R2	5
(N-28)	R1	4
New SP (N-32) ->	R0	3

- Cuando ocurre una excepción se envían a la pila los registros PC, PSR, R0-R3, R12 y LR.
- Si el procesador está en modo privilegiado usa el MSP si está en modo usuario usa el PSP.

When an exception takes place but the processor is handling another exception of the same or higher priority, the exception will enter pending state. When the processor has finished executing the current exception handler, it can then process the pended interrupt. Instead of restoring the registers back from the stack (unstacking) and then pushing them onto the stack again (stacking), the processor skips the unstacking and stacking steps and enters the exception handler of the pended exception as soon as possible.



# Late Arrivals



# Systick.

- El Systick es un timer sencillo integrado con el NVIC.
- Es un contador descendente de 24 bits.
- Se lo va a utilizar generalmente como “ticks” del sistema.

Bits	Name	Type	Reset Value	Description
16	COUNTFLAG	R	0	Read as 1 if counter reaches 0 since last time this register is read; clear to 0 automatically when read or when current counter value is cleared
2	CLKSOURCE	R/W	0	0 = External reference clock (STCLK) 1 = Use core clock
1	TICKINT	R/W	0	1 = Enable SYSTICK interrupt generation when SYSTICK timer reaches 0 0 = Do not generate interrupt
0	ENABLE	R/W	0	SYSTICK timer enable

# Systick (2)

Bits	Name	Type	Reset Value	Description
23:0	RELOAD	R/W	0	Reload value when timer reaches 0

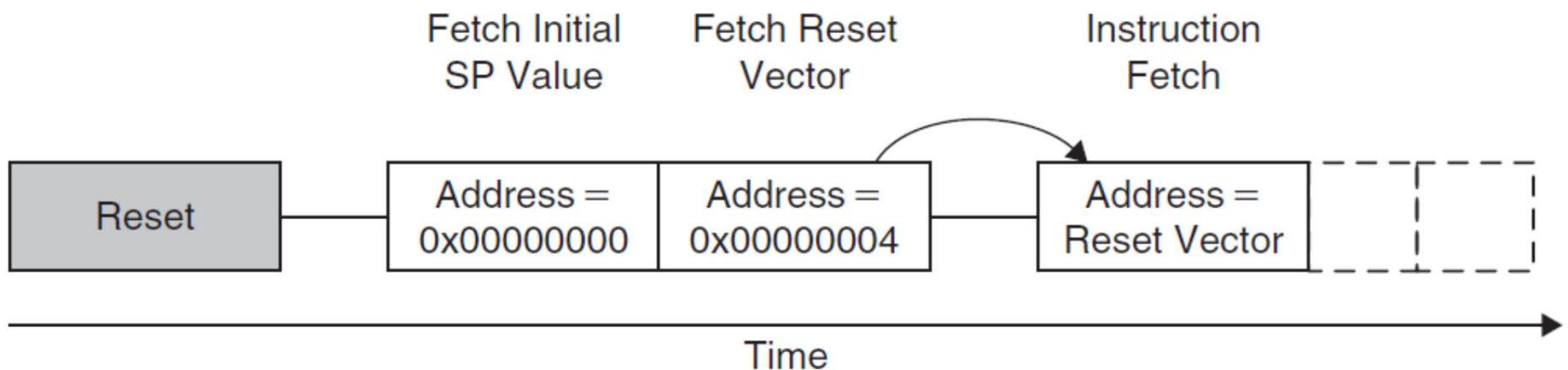
Bits	Name	Type	Reset Value	Description
23:0	CURRENT	R/Wc	0	Read to return current value of the timer. Write to clear counter to 0. Clearing of current value also clears COUNTFLAG in SYSTICK Control and Status Register

Bits	Name	Type	Reset Value	Description
31	NOREF	R	-	1 = No external reference clock (STCLK not available) 0 = External reference clock available
30	SKEW	R	-	1 = Calibration value is not exactly 10 ms 0 = Calibration value is accurate
23:0	TENMS	R/W	0	Calibration value for 10 ms.; chip designer should provide this value via Cortex-M3 input signals. If this value is read as 0, calibration value is not available

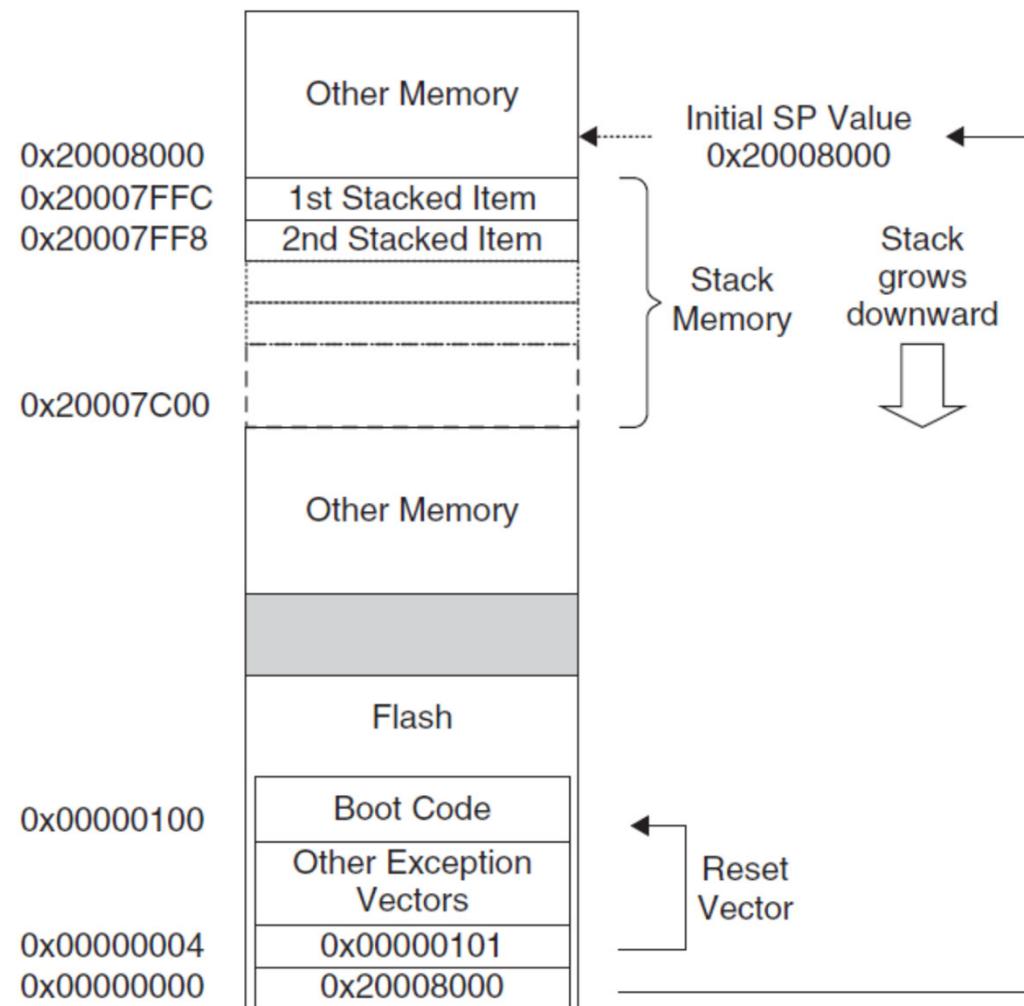
# Secuencia de Reset.

After the processor exits reset, it will read two words from memory :

- Address 0x00000000: Starting value of R13 (the SP)
- Address 0x00000004: Reset vector (the starting address of program execution; LSB should be set to 1 to indicate Thumb state)

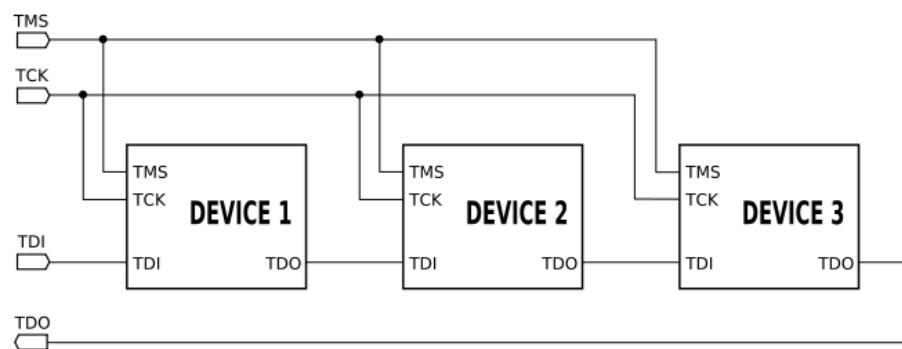


# Secuencia de Reset.

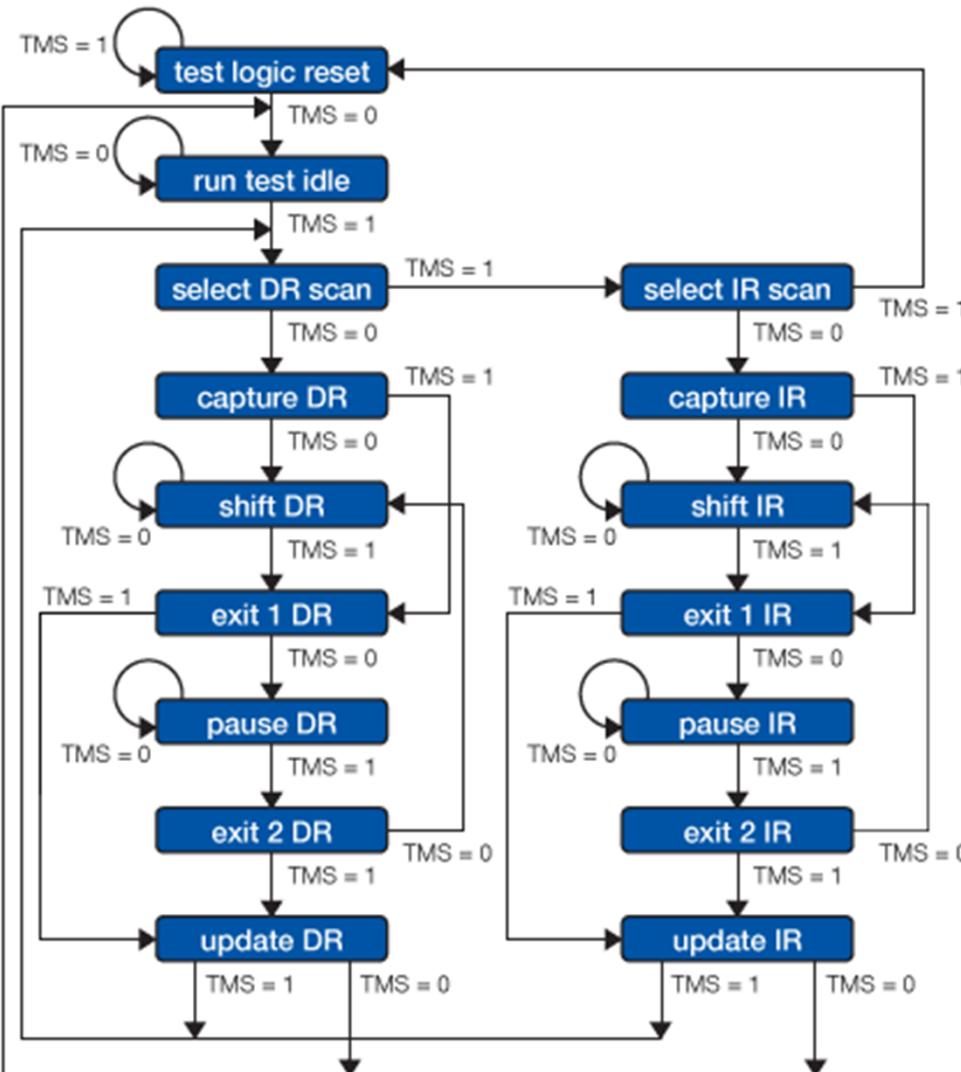


# Debug. JTAG

- JTAG es un protocolo de debug de 4 (lo más usual) o 5 pines.
- JTAG es un protocolo serie que permite "encadenar" varios dispositivos sobre el mismo puerto.
- JTAG es definido por la norma IEEE 1149.1-1990.

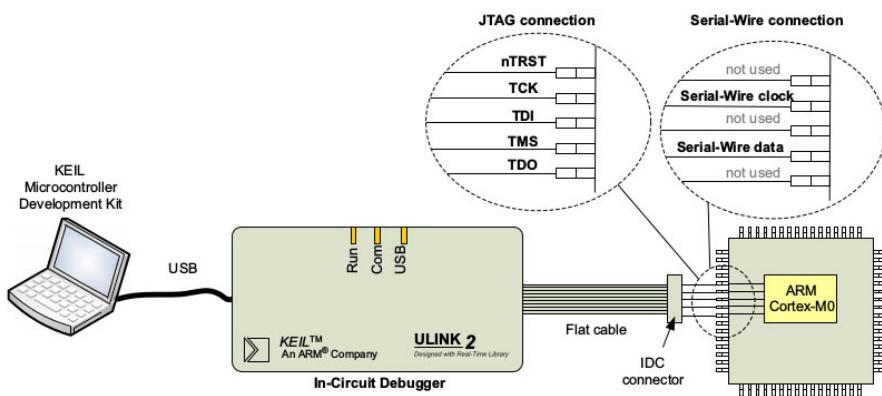


# Debug. JTAG.



- JTAG define un registro de datos(DR) y un registro de instrucción (IR).
- JTAG define algunas instrucciones obligatorias
  - BYPASS
  - EXTEST
  - SAMPLE/PRELOAD
  - IDCODE (no obligatoria)
- Permite definir instrucciones propias (usadas por casi todos los fabricantes para programar dispositivos)

# Debug. SerialWire



- El protocolo SerialWire usa dos pines SWCLK y SWDIO.
- La línea de datos es bidireccional y la de CLK es siempre manejada por el debugger.
- En los microcontroladores LPC se lo puede controlar totalmente por este protocolo.

# Debug en Cortex-M3

- Los procesadores Cortex-M3 provee dos características de debug que pueden ser clasificadas en dos tipos:
  - Invasivo.
  - No invasivo.

# Debug en Cortex-M3

## ■ Debug invasivo:

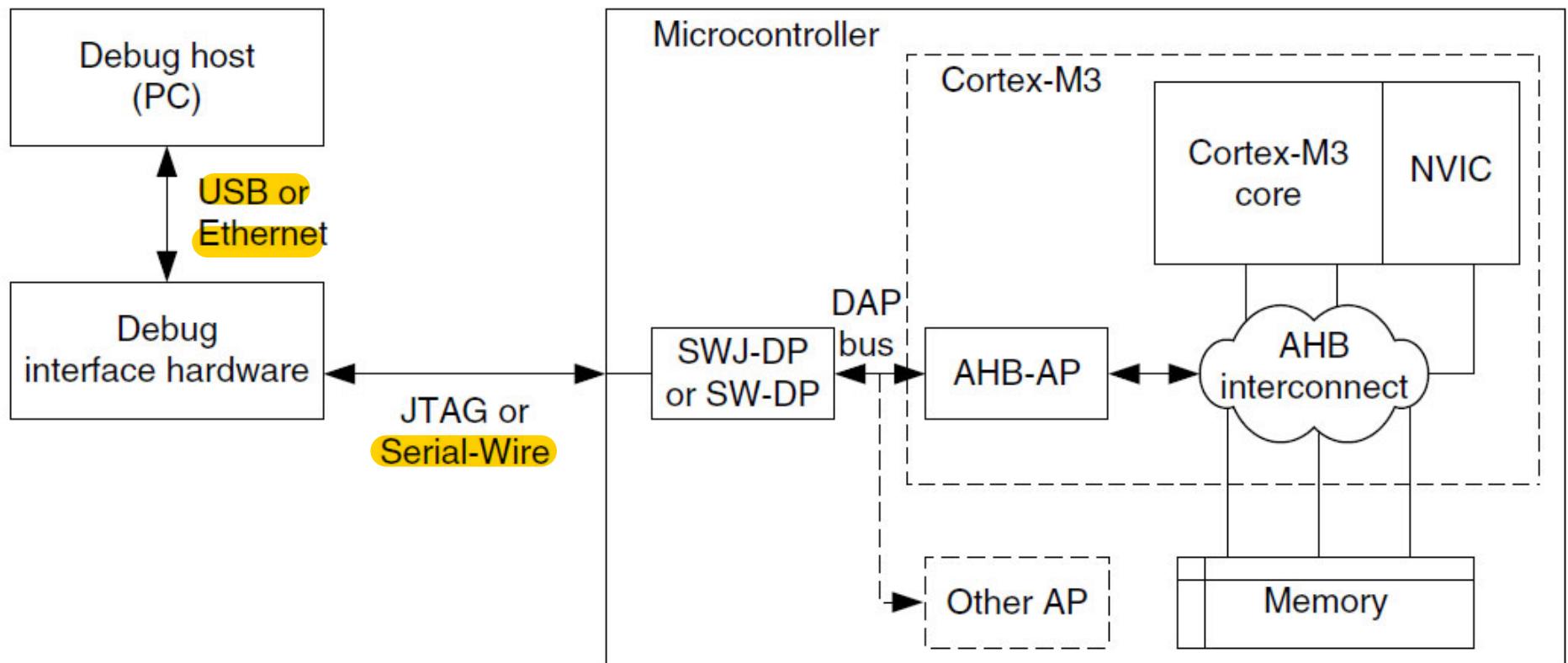
- Ejecución por pasos.
- Breakpoints por hardware.
- Excepciones de debug.
- Acceso a registros (lectura y escritura)
- Acceso y modificación de datos en memoria.

# Debug en Cortex-M3

## ■ Debug no invasivo:

- Acceso a memoria (lectura de memoria).
- Seguimiento:
  - Instrucciones
  - Datos.
- Profiling.

# Interfaz de Debug.



# Modos de debug.

- Los procesadores Cortex-M3 soportan dos modos de debug.
  - Halt. El procesador detiene su ejecución completamente.
  - Debug monitor. El procesador ejecuta una excepción de debug.

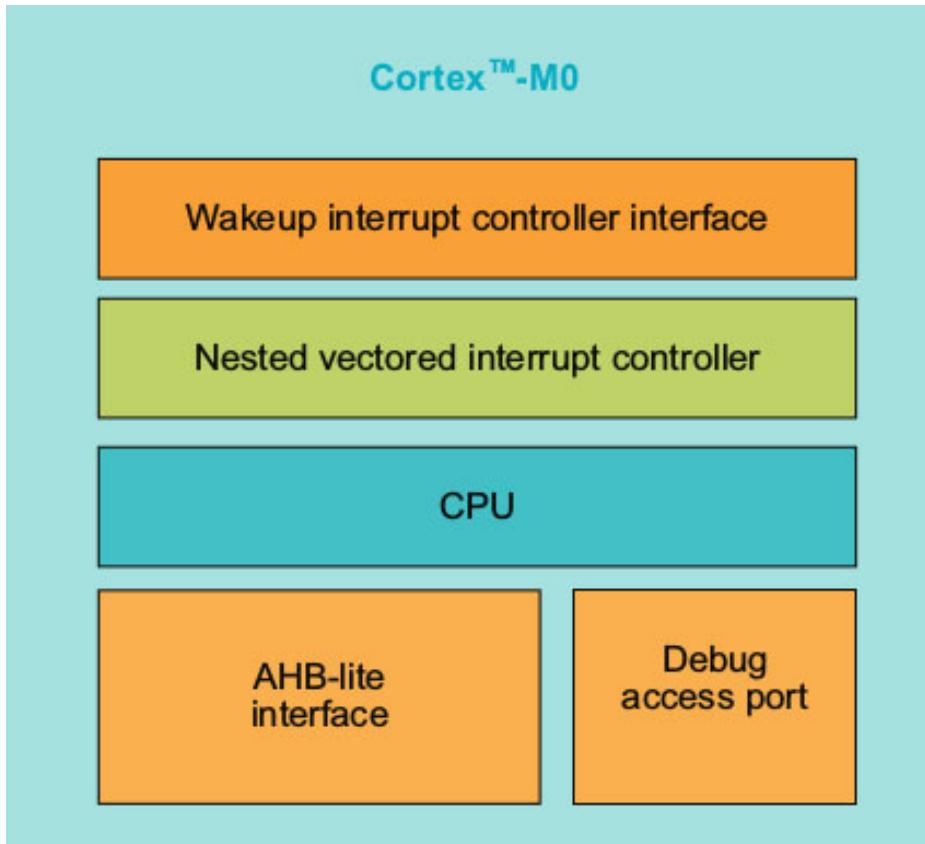
# Modos de debug. Halt.

- La ejecución del procesador **se detiene**.
- El **SYSTICK** se detiene.
- Las **interrupciones son demoradas y/o enmascaradas**.

# Modos de debug. Debug monitor

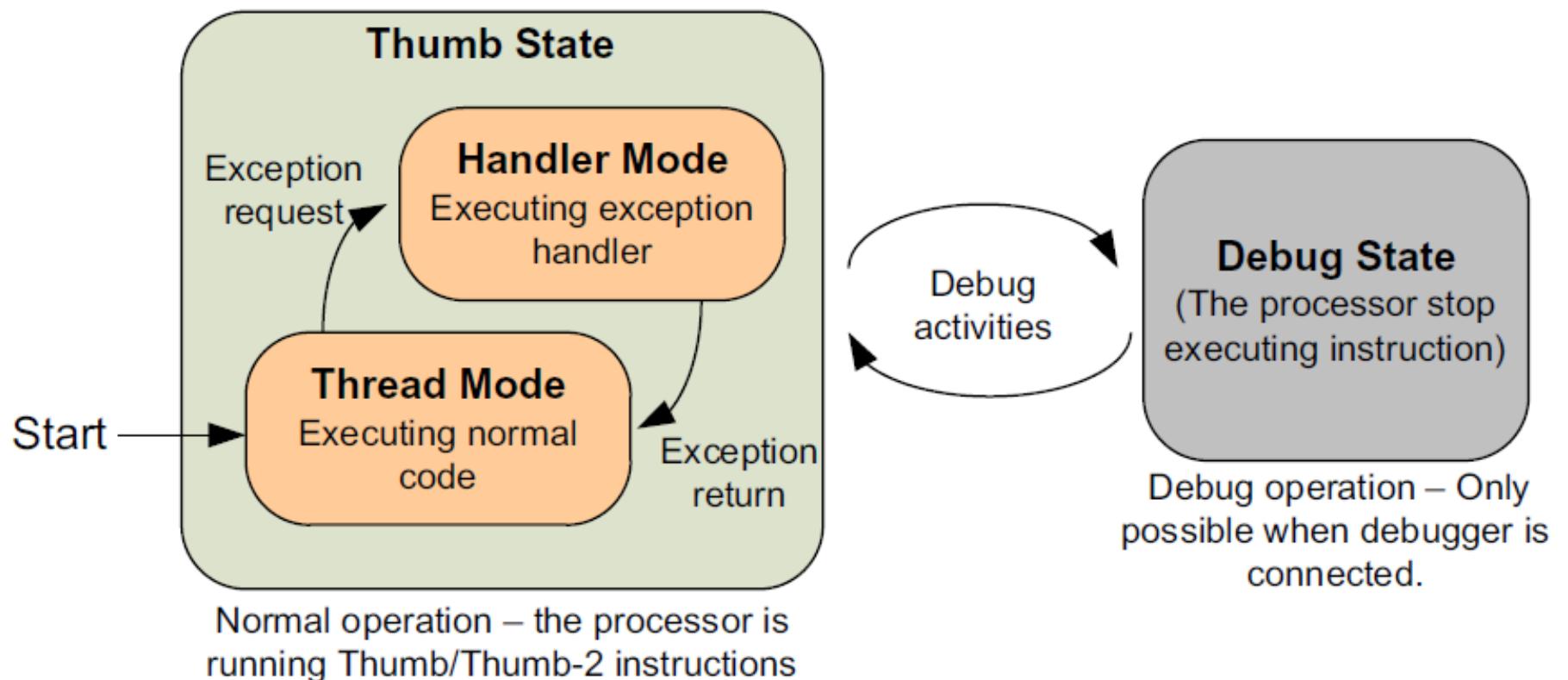
- El procesador ejecuta la interrupción tipo 12.
- El SYSTICK si corriendo.
- Las nuevas interrupciones serán atendidas o demoradas en función de la configuración del NVIC.
- El contenido de la memoria puede ser alterado por el handler de debug.

# Cortex-M0.

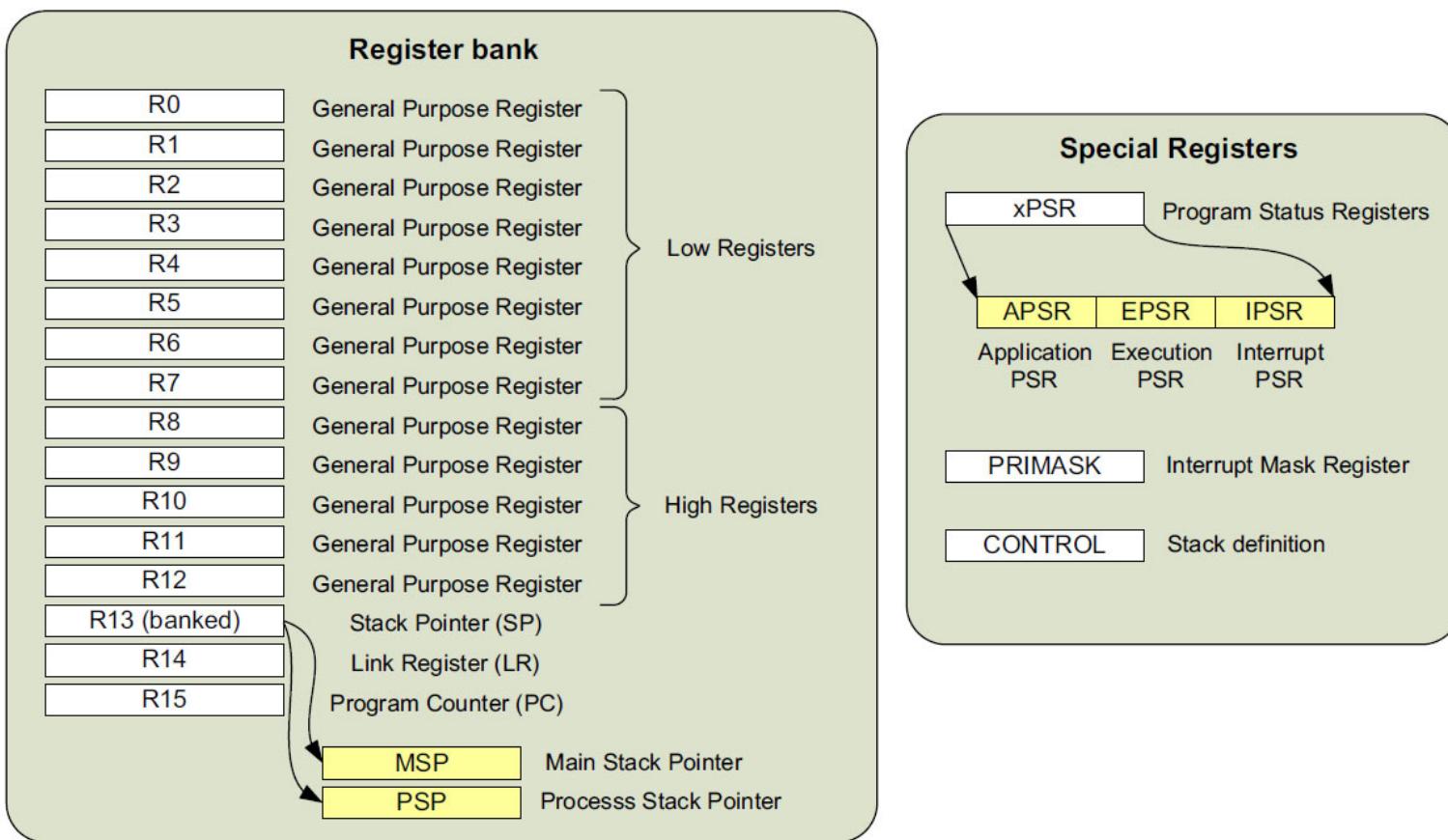


- Es el procesador más pequeño de la familia.
- Arquitectura Von Neumann. Menor rendimiento que los M3.
- Mismo controlador de Interrupciones que el M3 (NVIC) con 32 fuentes de interrupción máximo.
- Menores capacidades de debug que M3.

# Cortex-M0. Modos de operación y estados



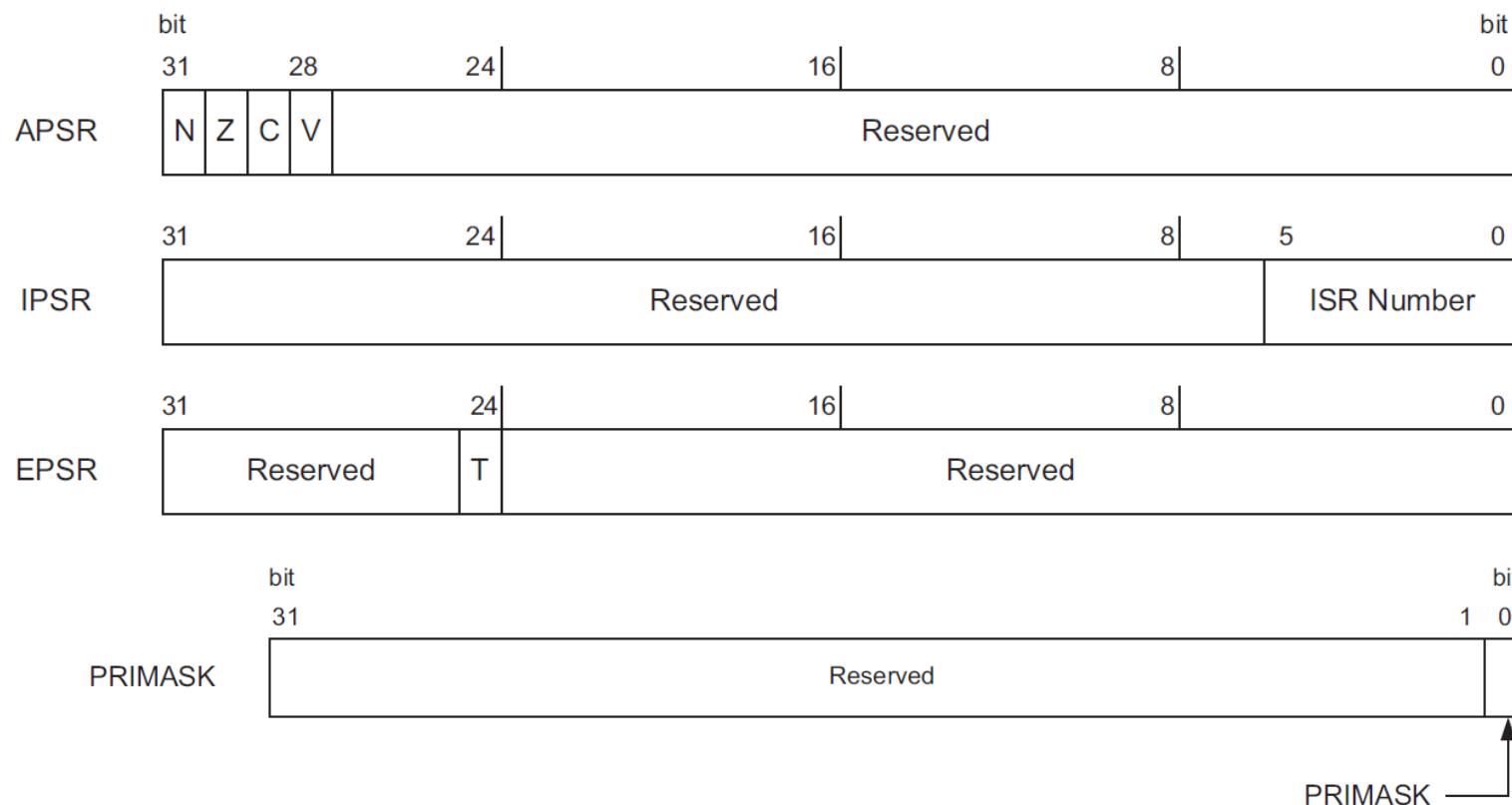
# Cortex-M0. Registros



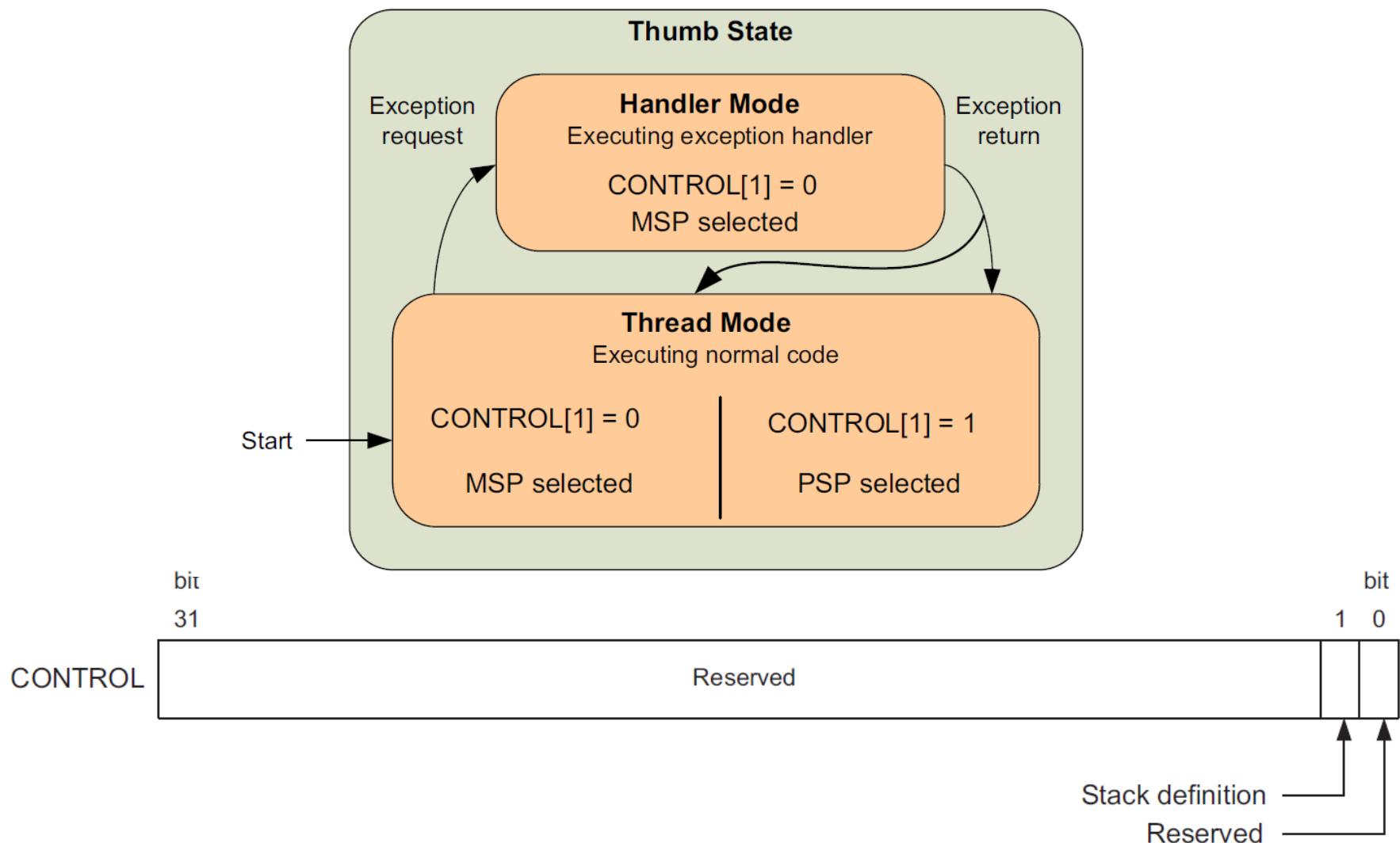
# Cortex-M0. Registros

- R0-R12. Son registros de propósito general. La mayoría de las instrucciones pueden acceder a los registros R0-R7 (por ejemplo MOV).
- R13 - SP es un registro que es usado como puntero a la pila. R13 puede acceder a dos registros (MSP o PSP) en función de la configuración del registro de control en modo thread y siempre MSP en modo handler
- R14 - LR. Contiene la dirección de retorno en el caso de interrupciones o llamadas a función.
- R15 – PC. Contador de programa. Se puede leer y escribir, una escritura en R15 genera un salto (sin actualizar R14) el bit menos significativo no se usa (instrucciones de 16 bits) y debe estar siempre en 1.

# Cortex-M0. Registros especiales



# Cortex-M0. Registros especiales (2)



# Cortex-M0. Set de Instrucciones

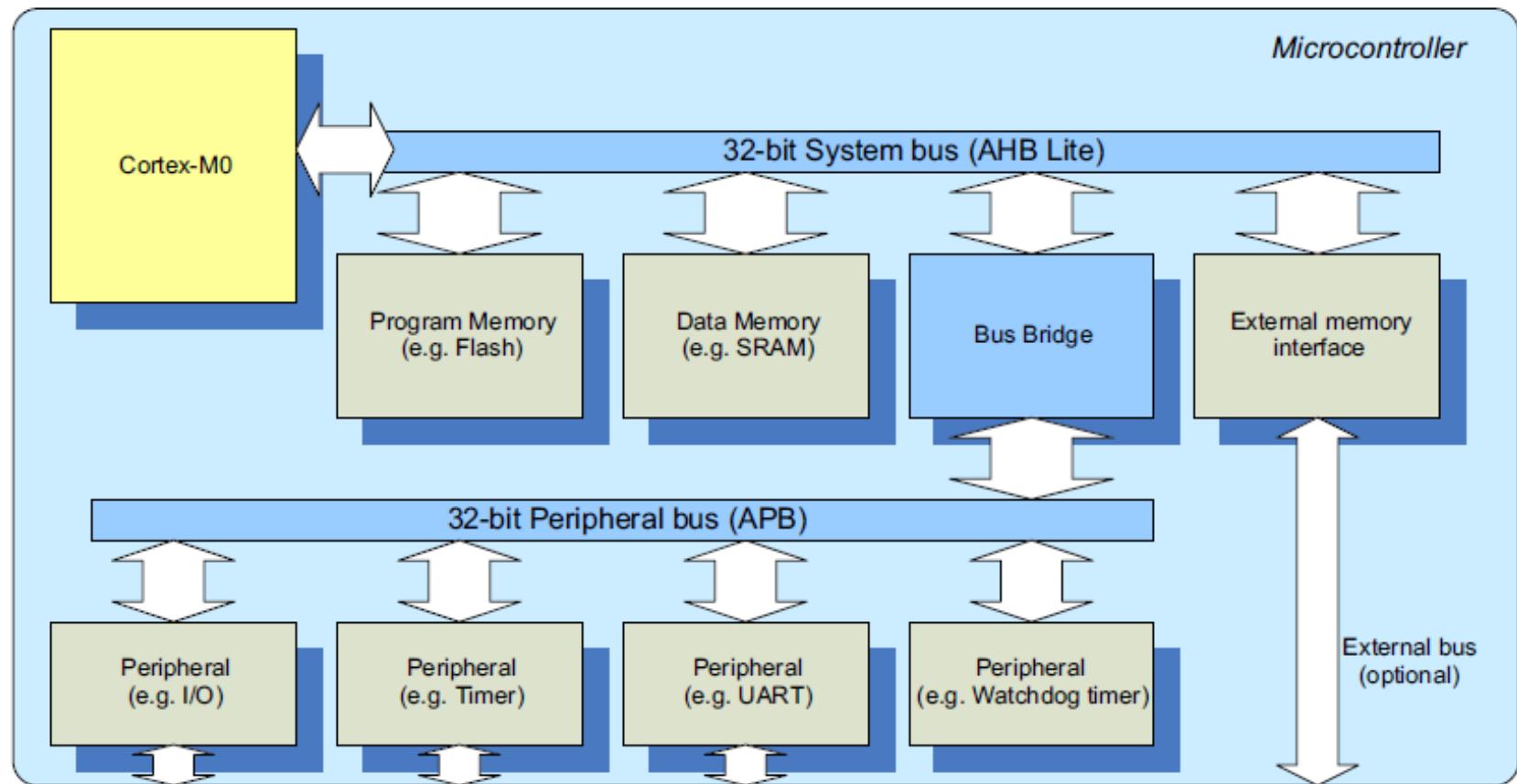
Table 5.1: 16-Bit Thumb Instructions Supported on the Cortex-M0 Processor

16-Bit Thumb Instructions Supported on Cortex-M0									
ADC	ADD	ADR	AND	ASR	B	BIC	BLX	BKPT	BX
CMN	CMP	CPS	EOR	LDM	LDR	LDRH	LDRSH	LDRB	LDRSB
LSL	LSR	MOV	MVN	MUL	NOP	ORR	POP	PUSH	REV
REV16	REVSH	ROR	RSB	SBC	SEV	STM	STR	STRH	STRB
SUB	SVC	SXTB	SXTH	TST	UXTB	UXTH	WFE	WFI	YIELD

Table 5.2: 32-Bit Thumb Instructions Supported on the Cortex-M0 Processor

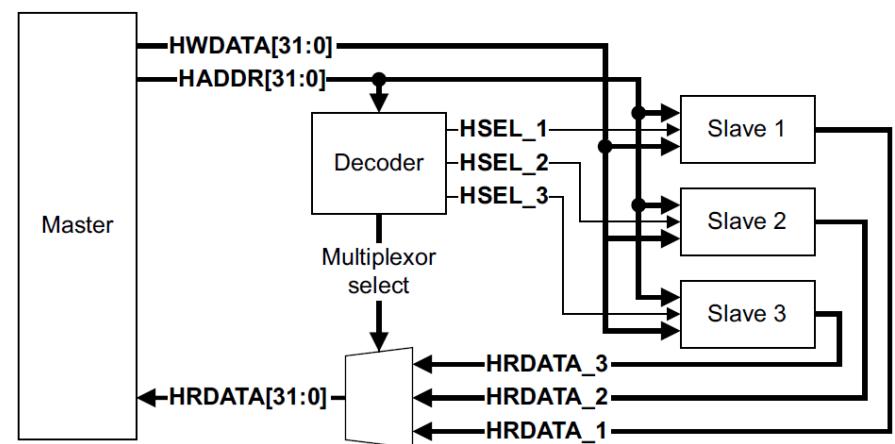
32-Bit Thumb Instructions Supported on Cortex-M0					
BL	DSB	DMB	ISB	MRS	MSR

# Cortex M0. Buses



# Cortex-M0. Interfaz AHB-Lite

- La interfaz AHB-Lite es la que permite la comunicación entre el procesador Cortex-M0 y el resto de los dispositivos de alta velocidad, esencialmente memorias.
- Es una interfaz sincrónica que no permite buses de alta impedancia



# Cortex-M0. Interfaz AHB-Lite

- La interfaz soporta diferentes formas de transferir datos: dato único, secuenciales y con ciclos de espera.
- Todas las transferencias son sincrónicas y las más sencillas requieren al menos dos ciclos de reloj

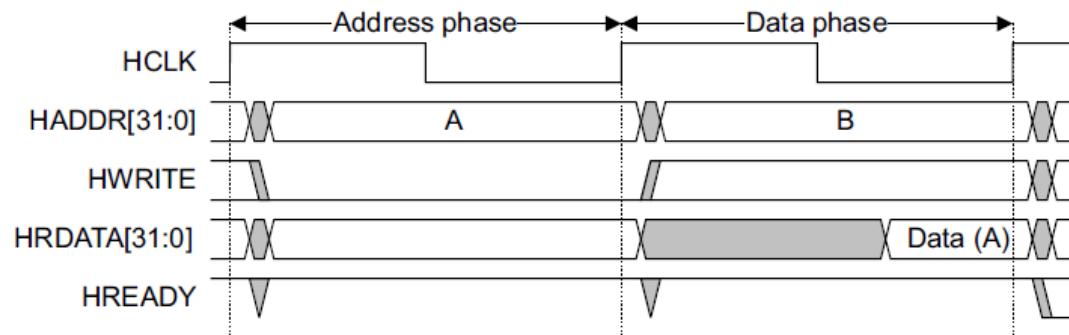


Figure 3-1 Read transfer

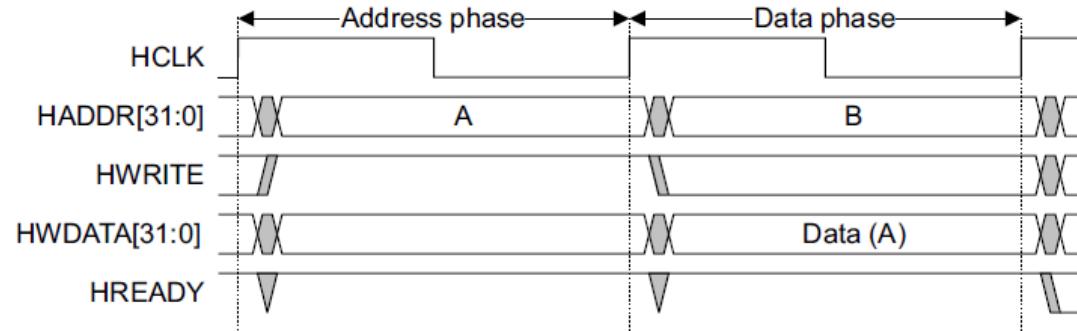
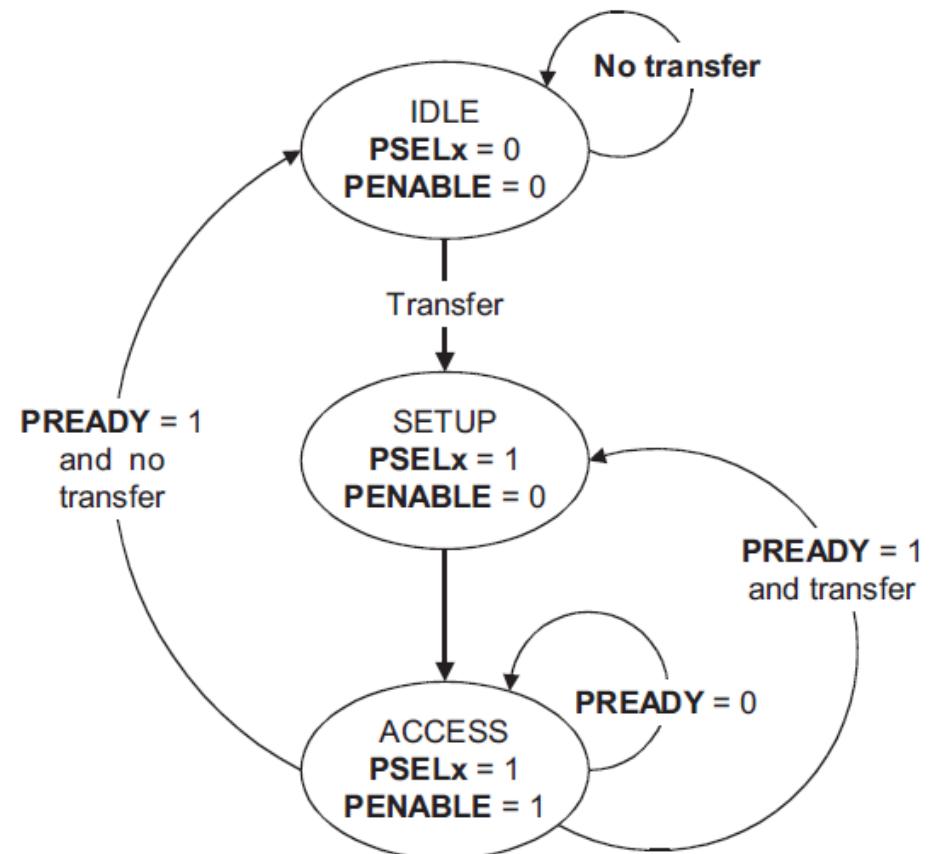


Figure 3-2 Write transfer

# Corte-M0. Interfaz APB

- El APB es la interfaz con la que se van a conectar los periféricos de baja velocidad.
- Estos periféricos no se van a conectar directamente al procesador, sino lo van a hacer a través de un dispositivo que conecta el bus APB con el AHB-Lite (Bridge)



# Cortex-M0. Interfaz APB. Escrituras

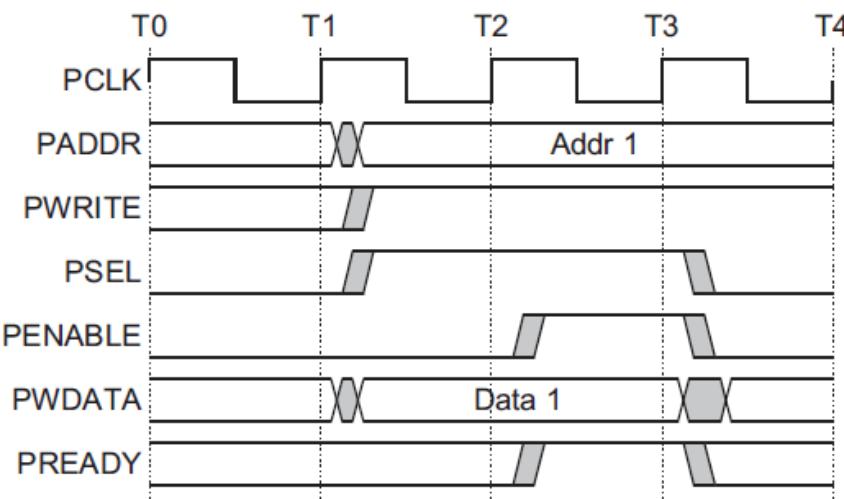


Figure 3-1 Write transfer with no wait states

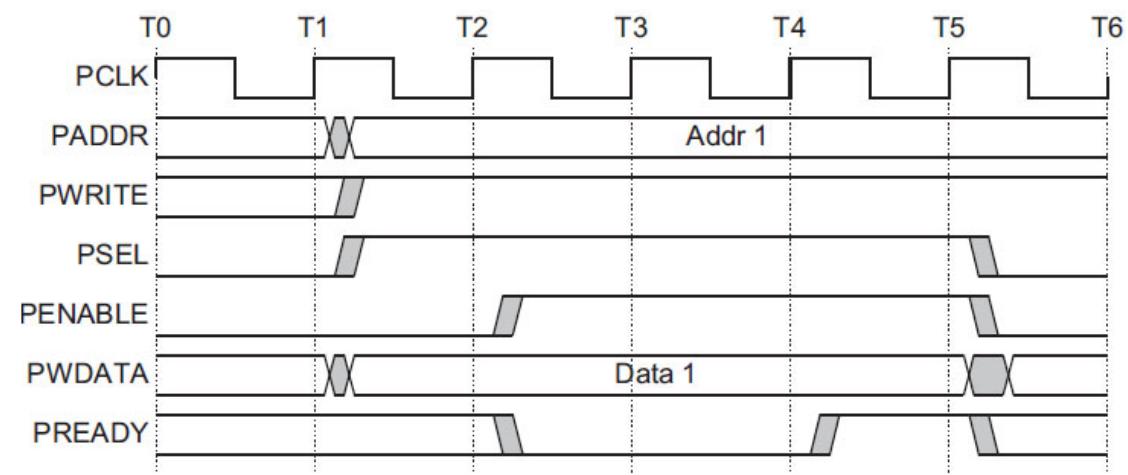


Figure 3-2 Write transfer with wait states

# Cortex-M0. Interfaz APB. Lecturas

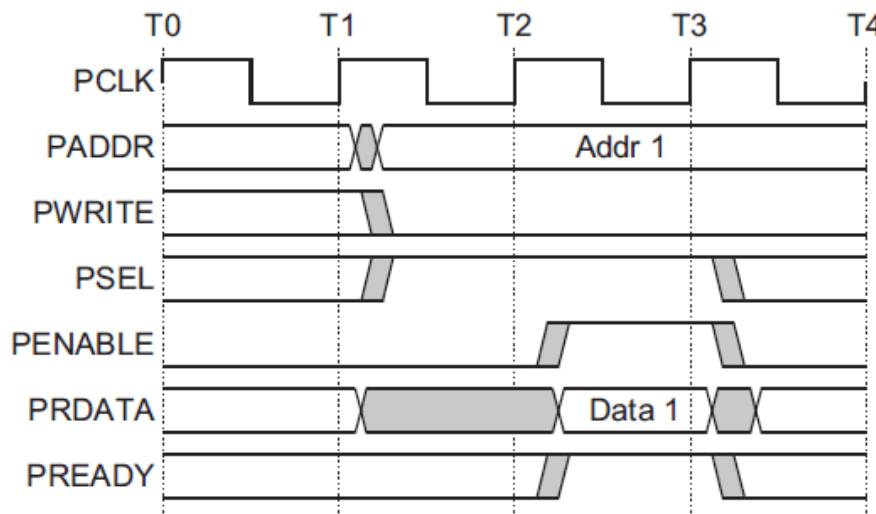


Figure 3-4 Read transfer with no wait states

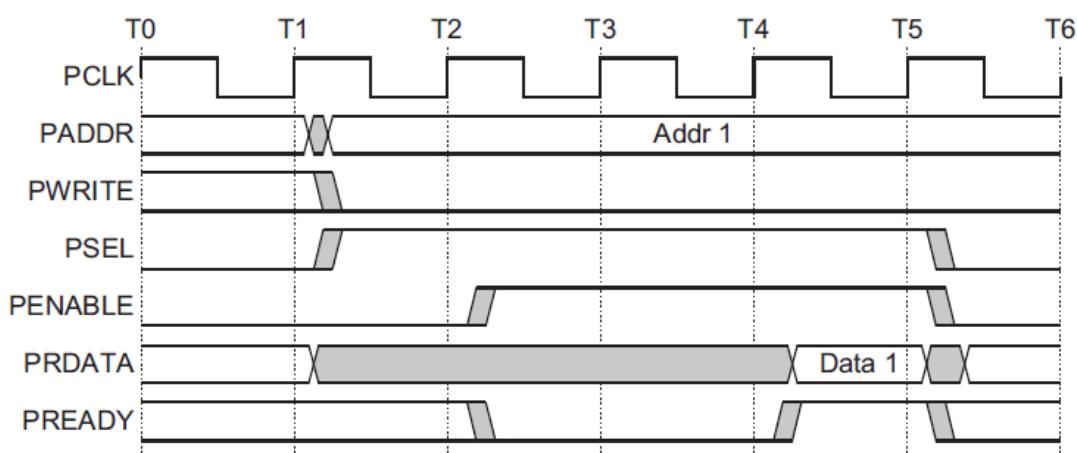
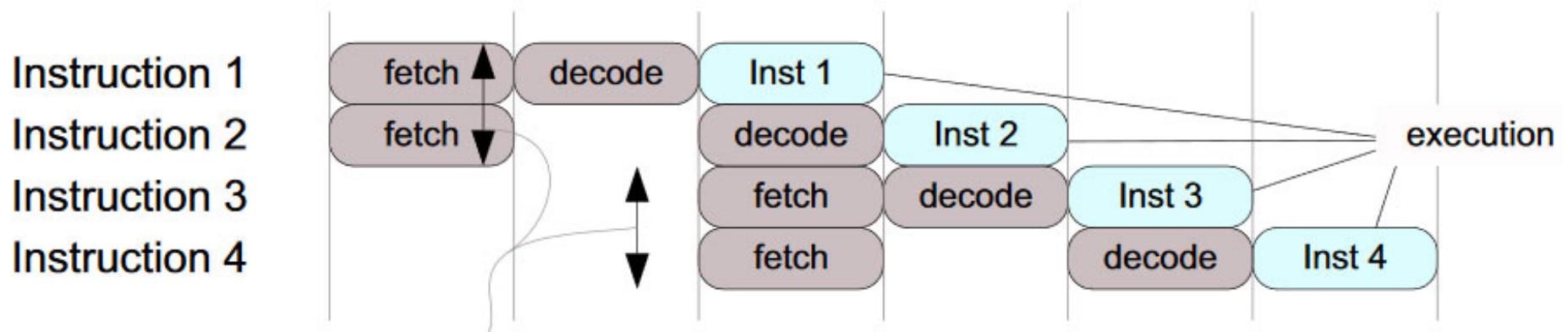


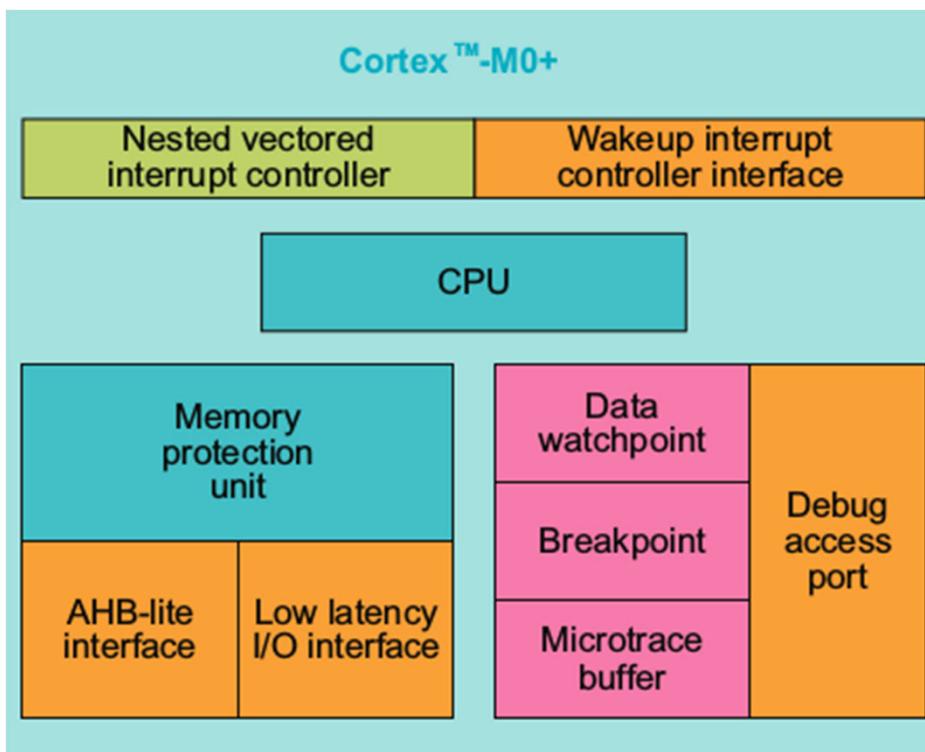
Figure 3-5 Read transfer with wait states

# Cortex-M0. Pipeline



Up to two instructions can be fetched in one transfer. (16-bit instructions)

# Cortex-M0+

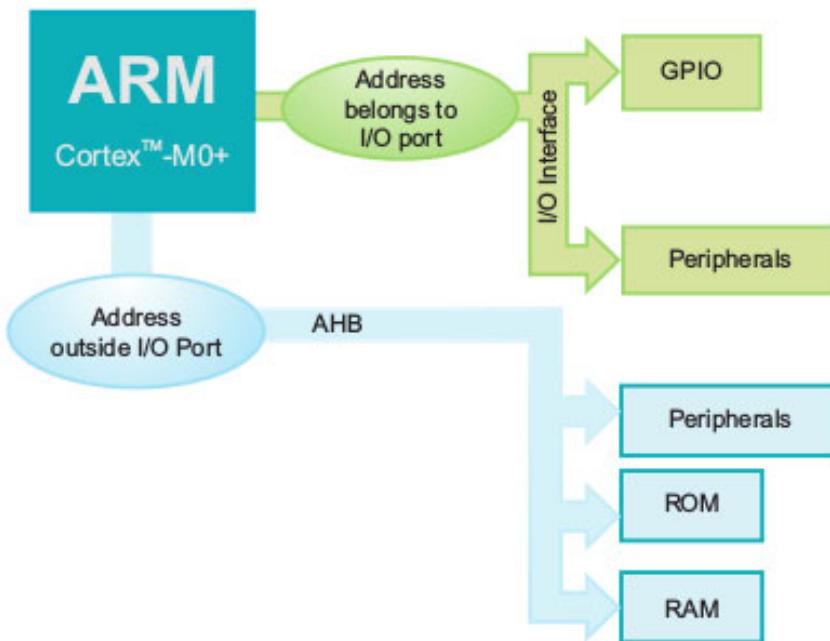


- Es totalmente compatible con el M0.
- Cambios en el pipeline y en la interfaz de I/O.
- Diseñado para utilizar instrucciones de 16bits (casi todas en Cortex M0).
- Mayores capacidades de Debug.

# Cortex-M0+

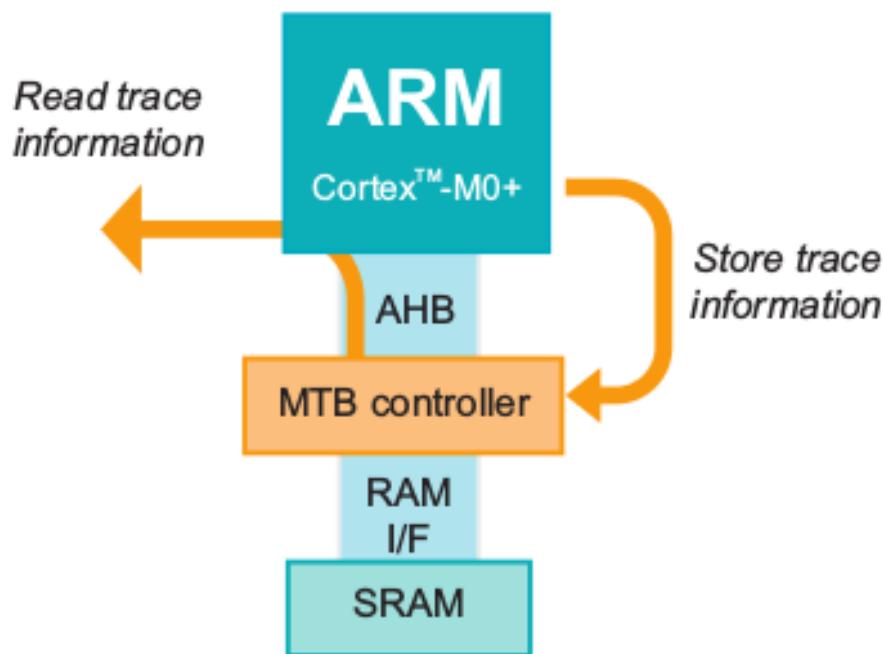


# Cortex-M0+



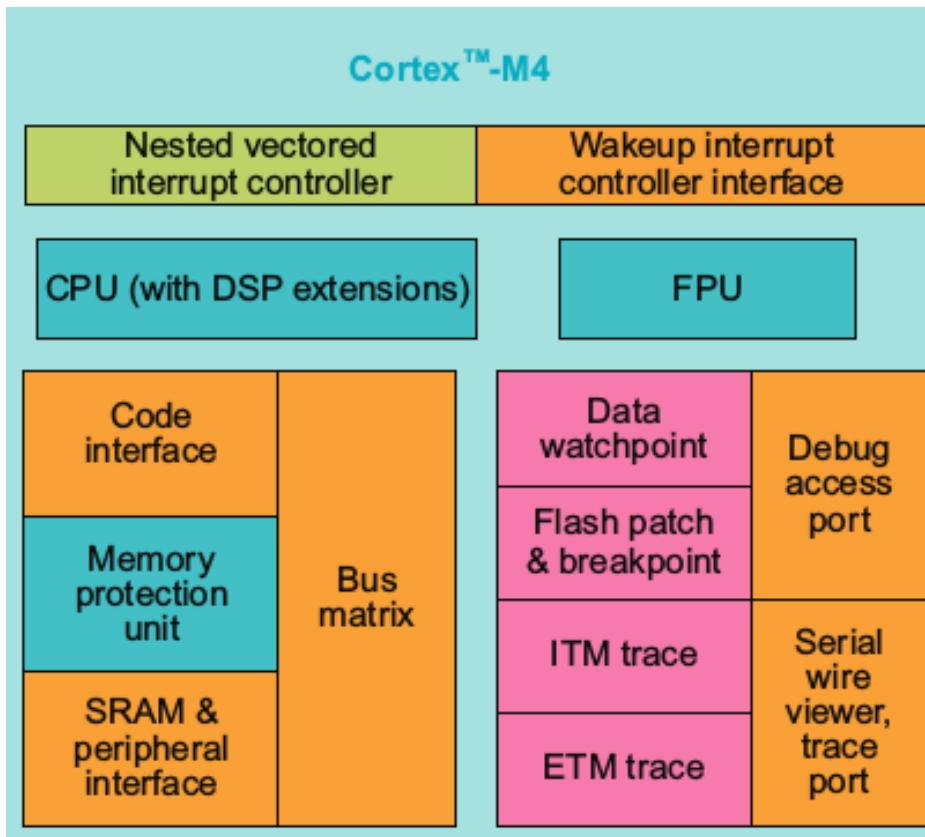
- Los Cortex-M0+ Agregan una interfaz de periféricos aparte del AHB.
- Permite acceder a perifericos y GPIO en un único ciclo de reloj.
- Es transparente a la interfaz de programa.

# Cortex-M0+



- Los Cortex-M0+ tienen una interfaz de debug completa, como los M3 y M4.
- Agregan el controlador MTB (microtrace buffer). Que permite guardar una "cola" de instrucciones ejecutadas en SRAM.

# Cortex-M4



- El Cortex-M0 se puede pensar como un M3 "recortado" mientras que el M4 se puede considerar como un M3 con **DSP**.

- Los Cortex-M4 están orientados a algoritmos de DSP como filtros, FFT, lazos PID.

DSP : Digital Signal Processor/Digital Signal Processing

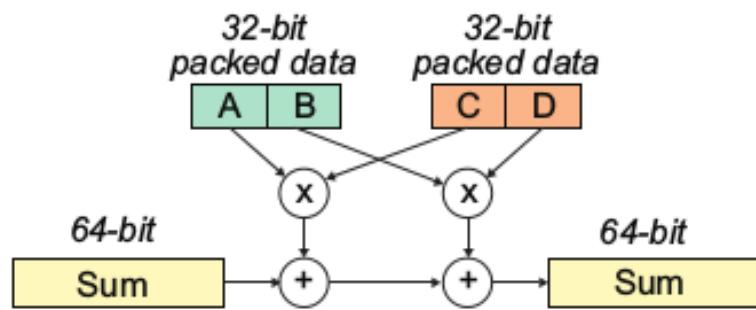
# Cortex-M4.

Single Cycle MAC Instructions on the Cortex-M4

Operation	Instructions
$16 \times 16 = 32$	SMULBB, SMULBT, SMULTB, SMULLT
$16 \times 16 + 32 = 32$	SMLABB, SMLABT, SMLATB, SMLATT
$16 \times 16 + 64 = 64$	SMLALBB, SMLALBT, SMLALTB, SMLALTT
$16 \times 32 = 32$	SMULWB, SMULWT
$(16 \times 32) + 32 = 32$	SMLAWB, SMLAWT
$(16 \times 16) \pm (16 \times 16) = 32$	SMUAD, SMUADX, SMUSD, SMUSDX
$(16 \times 16) \pm (16 \times 16) + 32 = 32$	SMLAD, SMLADX, SMLSD, SMLSX
$(16 \times 16) \pm (16 \times 16) + 64 = 64$	SMLALD, SMLALDX, SMLSQD, SMLSQDX
$32 \times 32 = 32$	MUL
$32 \pm (32 \times 32) = 32$	MLA, MLS
$32 \times 32 = 64$	SMULL, UMULL
$(32 \times 32) + 64 = 64$	SMLAL, UMLAL
$(32 \times 32) + 32 + 32 = 64$	UMAAL
$32 \pm (32 \times 32) = 32$ (upper) $(32 \times 32) = 32$ (upper)	SMMLA, SMMLAR, SMMLS, SMMLSR SMMUL, SMMULR

- Los Cortex-M4 tienen el mismo NVIC que los M3, el mismo sistema de Debug y Buses.
- Agregan instrucciones de Multiplicar y acumular por hardware.
- Los Cortex-M4 agregan unidad de punto flotante (FPU).
- Entre las instrucciones de DSP agregan las SIMD.

# Cortex-M4. SIMD.



- Las instrucciones SIMD (single instruction multiple data). Permiten hacer varias operaciones de 8 o 16 bits en un único ciclo de reloj.
- Las SIMD permiten optimizar algoritmos como los de filtros FIR.

$$y[n] = \sum_{k=0}^{N-1} h[k]x[n - k]$$

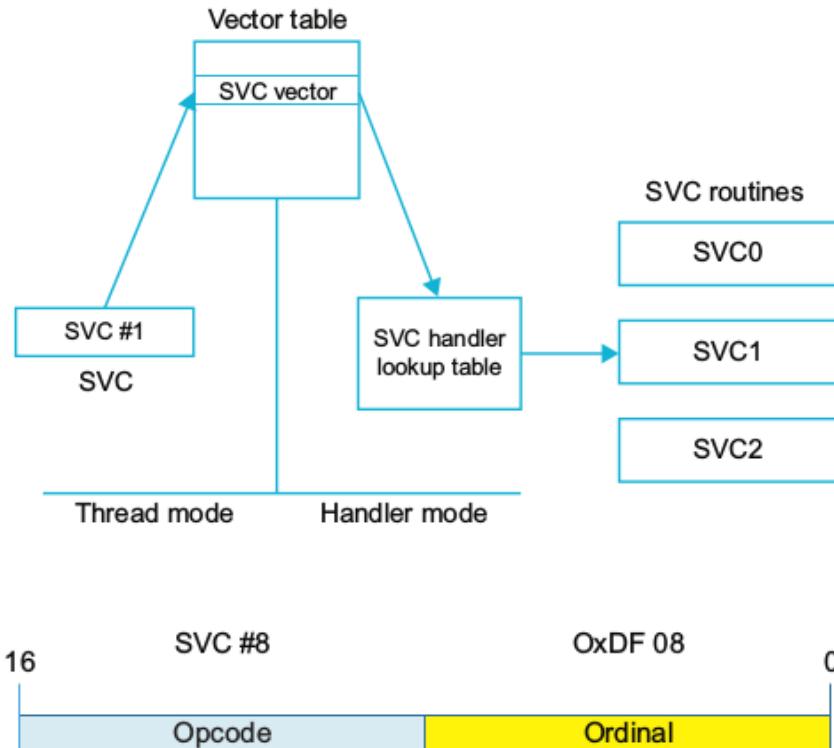
# Administración de Energía.

- Los procesadores Cortex-M pueden entrar en dos modos de bajo consumo. **SLEEP** y **DEEPSLEEP**.
  - **SLEEP.** Detiene el reloj de la CPU pero no de los periféricos. Cualquiera de los periféricos puede despertar al CPU por medio de interrupciones.
  - **DEEPSLEEP.** Detiene el reloj del CPU y de la mayoría de los periféricos. Tipicamente se sostiene el reloj de la SRAM. Al no tener reloj el NVIC es necesario otro dispositivo para volver de este modo.
  - El Wakeup Interrupt Controller (WIC) es un dispositivo sencillo y de bajo consumo que permite volver de un DEEPSLEEP.

# Cortex-M. Modos de operación.

Modes (Thread out of reset)	Operations (Privilege out of reset)	Stacks (Main out of reset)
Handler - Processing of exceptions	Privileged execution full control	Main stack used by OS and exceptions
Thread - No exception is being processed - Normal code execution	Privileged or unprivileged	Main or process

# Cortex-M. Supervisor Call (SVC)

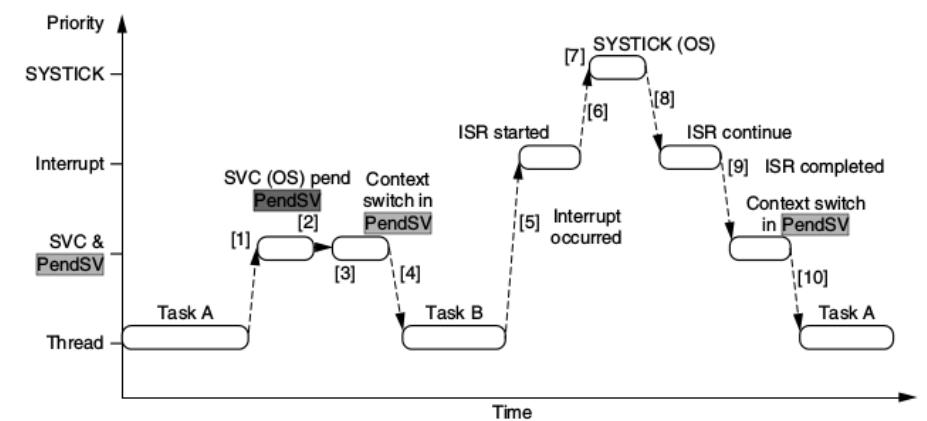
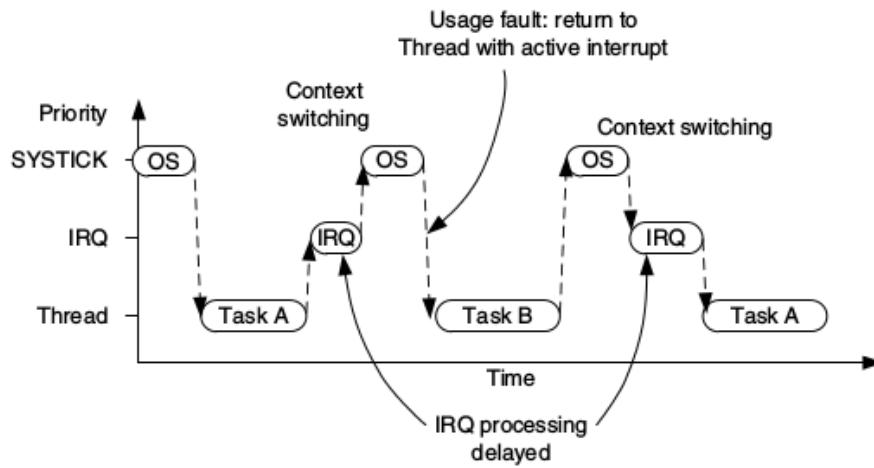


- Una vez configurado en modo thread sin privilegios, el único camino para volver a obtenerlos es una excepción.
- La supervisor CALL es este camino.
- En su código de operación permite codificar un número de 8 bits, lo que permite parametrizar la llamada.

# Cortex-M. Pendable Service Call

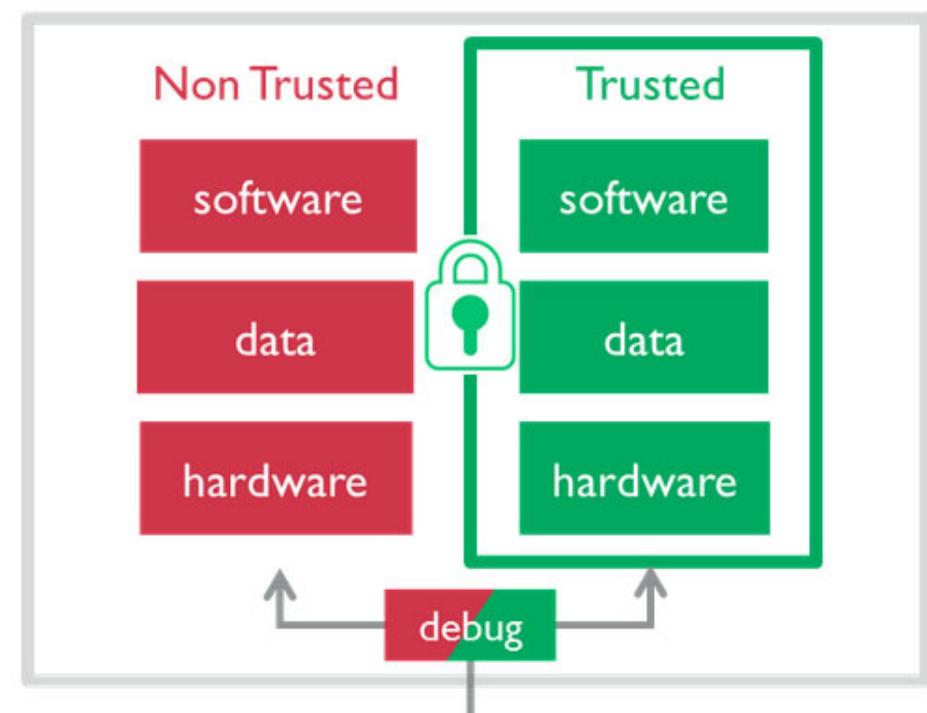
- La Pend\_SVC es una excepción que se tendría que ejecutar si ninguna otra excepción tiene lugar.
- El uso normal de la Pend\_SVC es con sistemas operativos para generar los cambios de contexto.
- La idea detrás de esta excepción es minimizar la cantidad de cambios de contexto del procesador.

# Cortex-M. Pendable Service Call



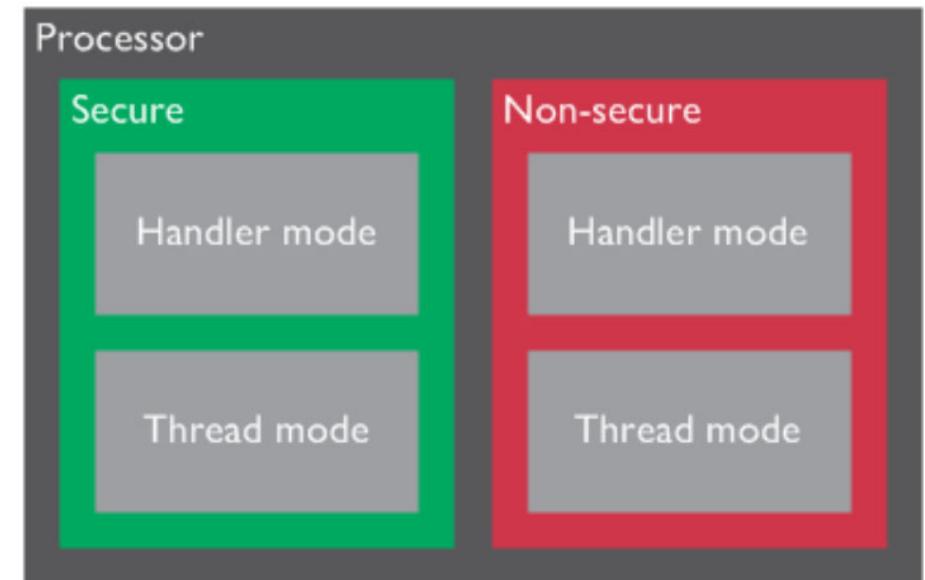
# Cortex-M. ARMv8

- **Cortex-M23.** Procesador más pequeño de con CPU ARMv8 (comparable con los M0/M0+)
- **Cortex-M33.** Procesador con extensiones para DSP + FPU de arquitectura ARMv8. (comparable con los M4/M7).



# Cortex-M. ARMv8. TrustZone

- Se definen dos estados de procesador. Seguro e inseguro.
- El pasaje de un estado del procesador a otro es a través de interfaces bien definidas.
- También se pueden definir zonas de memoria seguras e inseguras con el objeto de aislar datos.
- La aplicación típica es la de tener un firmware seguro y aplicaciones no seguras.



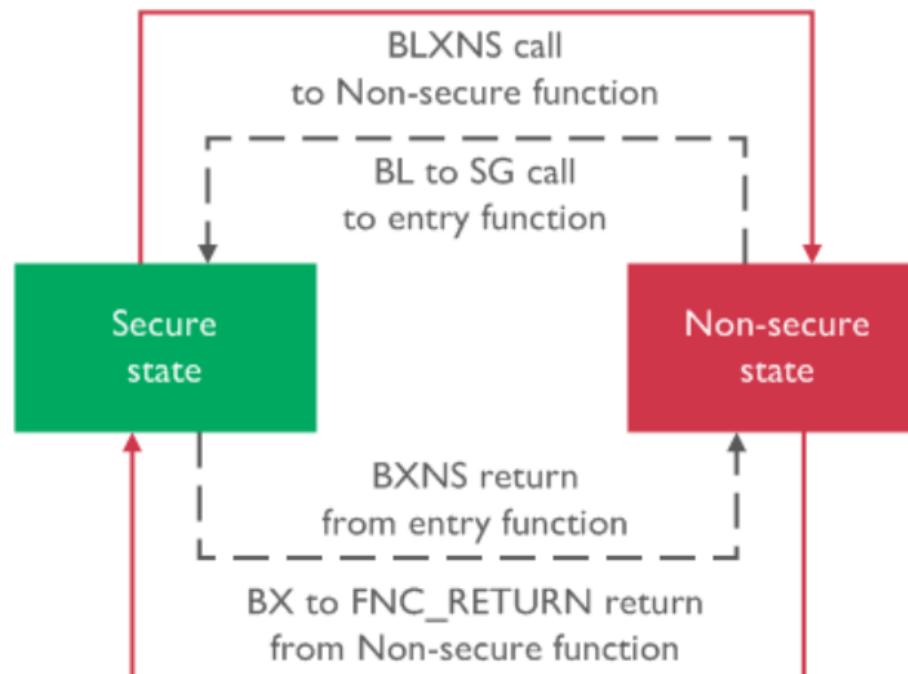
# Cortex-M. ARMv8. TrustZone

- En el espacio de memoria se pueden definir:
  - Zonas de memoria seguras (S): Sólo pueden ser llamadas de estados seguros.
  - Zonas de memoria inseguras “mixtas” (Non-secure Callable): son zonas de memoria seguras que se utilizan para la transición al estado seguro. El uso típico es generar tablas para hacer saltos a zonas seguras.
  - Zonas de memoria no seguras (NS): Son zonas de memoria accesibles por todo el software del procesador.

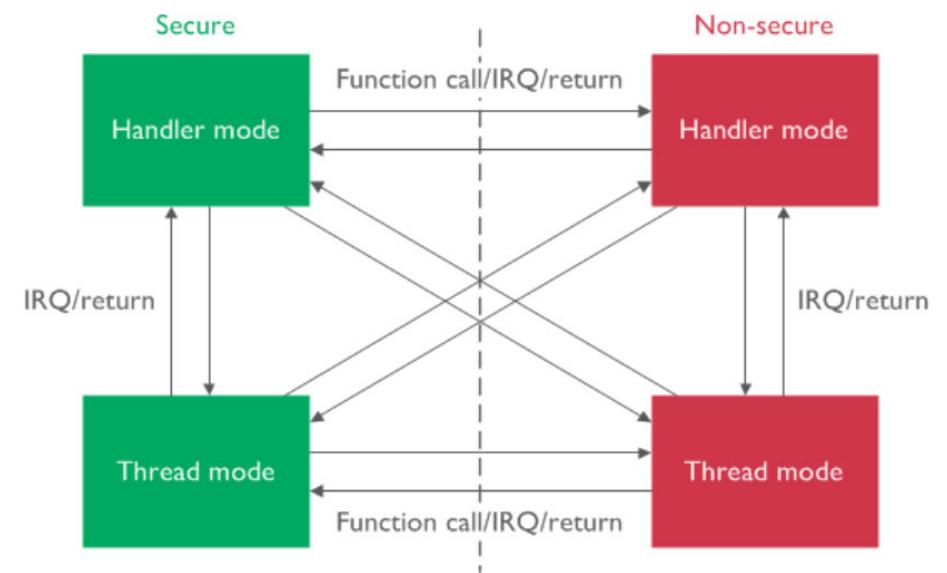
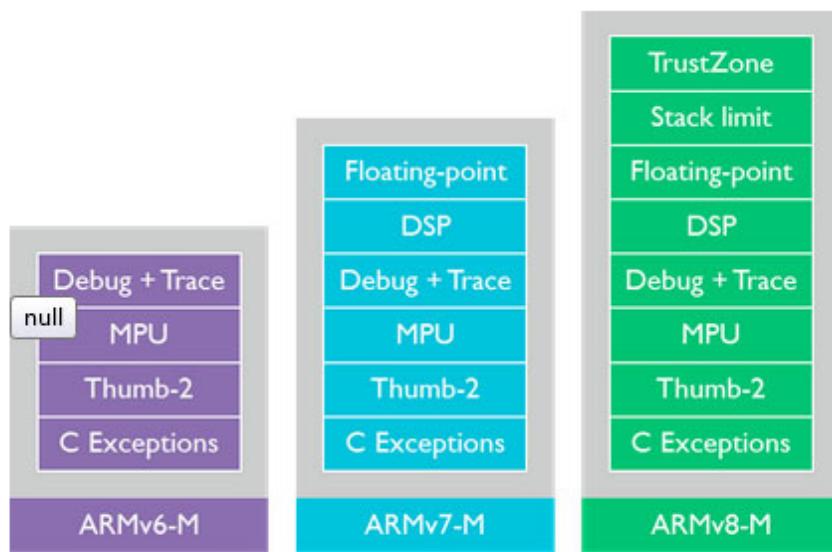
# Cortex-M. ARMv8. TrustZone

SG	Secure gateway. Used for switching from Non-secure to Secure state at the first instruction of Secure entry point.
BXNS	Branch with exchange to Non-secure state. Used by Secure software to branch or return to Non-secure program.
BLXNS	Branch with link and exchange to Non-secure state. Used by Secure software to call Non-secure functions.

The following figure shows the security state transitions.



# Cortex-M. ARMv8. TrustZone



# CMSIS. Archivos.

File	Provider	Description
<code>device.h</code>	Device specific (provided by silicon partner)	Defines the peripherals for the actual device. The file may use several other include files to define the peripherals of the actual device.
<code>core_cm0.h</code>	ARM (for RealView ARMCC, IAR, and GNU GCC)	Defines the core peripherals for the Cortex-M0 CPU and core peripherals.
<code>core_cm3.h</code>	ARM (for RealView ARMCC, IAR, and GNU GCC)	Defines the core peripherals for the Cortex-M3 CPU and core peripherals.
<code>core_cm0.c</code>	ARM (for RealView ARMCC, IAR, and GNU GCC)	Provides helper functions that access core registers.
<code>core_cm3.c</code>	ARM (for RealView ARMCC, IAR, and GNU GCC)	Provides helper functions that access core registers.
<code>startup_device</code>	ARM (adapted by compiler partner / silicon partner)	Provides the Cortex-M startup code and the complete (device specific) Interrupt Vector Table
<code>system_device</code>	ARM (adapted by silicon partner)	Provides a device specific configuration file for the device. It configures the device initializes typically the oscillator (PLL) that is part of the microcontroller device

# CMSIS. Funciones para inicialización.

Function Definition	Description
void SystemInit (void)	Setup the microcontroller system. Typically this function configures the oscillator (PLL) that is part of the microcontroller device. For systems with variable clock speed it also updates the variable SystemCoreClock. SystemInit is called from startup_device file.
void SystemCoreClockUpdate (void)	Updates the variable SystemCoreClock and must be called whenever the core clock is changed during program execution. SystemCoreClockUpdate() evaluates the clock register settings and calculates the current core clock.

Variable Definition	Description
uint32_t SystemCoreClock	Contains the system core clock (which is the system clock frequency supplied to the SysTick timer and the processor core clock). This variable can be used by the user application to setup the SysTick timer or configure other parameters. It may also be used by debugger to query the frequency of the debug timer or configure the trace clock speed. SystemCoreClock is initialized with a correct predefined value.  The compiler must be configured to avoid the removal of this variable in case that the application program is not using it. It is important for debug systems that the variable is physically present in memory so that it can be examined to configure the debugger.

# CMSIS. Systick

Name	Parameter	Description
uint32_t SysTickConfig (uint32_t ticks)	ticks is SysTick counter reload value	Setup the SysTick timer and enable the SysTick interrupt. After this call the SysTick timer creates interrupts with the specified time interval.  Return: 0 when successful, 1 on failure.

# CMSIS. NVIC

Name	Core	Parameter	Description
void NVIC_SetPriorityGrouping (uint32_t PriorityGroup)	M3	Priority Grouping Value	Set the Priority Grouping (Groups , Subgroups)
uint32_t NVIC_GetPriorityGrouping (void)	M3	(void)	Get the Priority Grouping (Groups , Subgroups)
void NVIC_EnableIRQ (IRQn_Type IRQn)	M0, M3	IRQ Number	Enable IRQn
void NVIC_DisableIRQ (IRQn_Type IRQn)	M0, M3	IRQ Number	Disable IRQn
uint32_t NVIC_GetPendingIRQ (IRQn_Type IRQn)	M0, M3	IRQ Number	Return 1 if IRQn is pending else 0
void NVIC_SetPendingIRQ (IRQn_Type IRQn)	M0, M3	IRQ Number	Set IRQn Pending
void NVIC_ClearPendingIRQ (IRQn_Type IRQn)	M0, M3	IRQ Number	Clear IRQn Pending Status
uint32_t NVIC_GetActive (IRQn_Type IRQn)	M3	IRQ Number	Return 1 if IRQn is active else 0
void NVIC_SetPriority (IRQn_Type IRQn, uint32_t priority)	M0, M3	IRQ Number, Priority	Set Priority for IRQn (not threadsafe for Cortex-M0)
uint32_t NVIC_GetPriority (IRQn_Type IRQn)	M0, M3	IRQ Number	Get Priority for IRQn
uint32_t NVIC_EncodePriority (uint32_t PriorityGroup, uint32_t PreemptPriority, uint32_t SubPriority)	M3	IRQ Number, Priority Group, Preemptive Priority, Sub Priority	Encode priority for given group, preemptive and sub priority
NVIC_DecodePriority (uint32_t Priority, uint32_t PriorityGroup, uint32_t* pPreemptPriority, uint32_t* pSubPriority)	M3	IRQ Number, Priority, pointer to Priority Group, pointer to Preemptive Priority, pointer to Sub Priority	Decode given priority to group, preemptive and sub priority
void NVIC_SystemReset (void)	M0, M3	(void)	Resets the System

# Bibliografía.

- The Definitive Guide to the ARM Cortex-M3, Second Edition – Joseph Yiu – Newnes – 2009.
- The Definitive Guide to the ARM Cortex-M0. Joseph Yiu. Elsevier. 2011.
- The Designer's Guide to the Cortex-M Processor Family. Trevor Martin. Elsevier. 2013
- LPC17xx User manual.
- ARM®v7-M Architecture Reference Manual.
- Cortex™-M3 Revision: r1p1 Technical Reference Manual.