



# Introducción Cortex-R y Cortex-A

# 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 seguros, etc.
- **ARMv7-M. Microcontroller.** Procesadores de bajo costo , bajo consumo y baja latencia de interrupciones. Procesadores orientados a usos de microcontroladores.

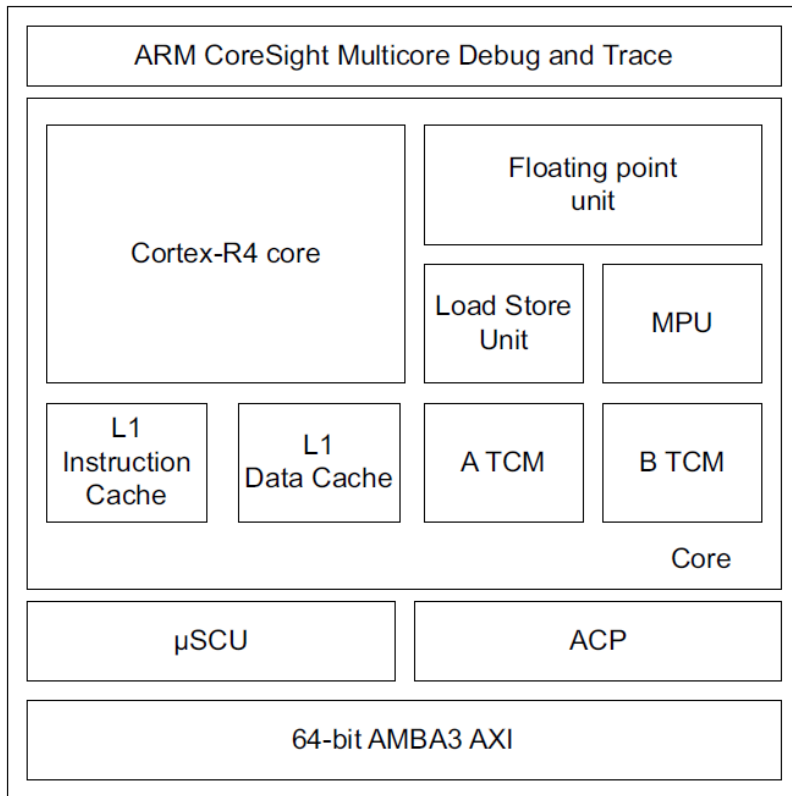
# Cortex. Versiones

	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-R. Versiones

- **Cortex-R4.** El más pequeño de los Cortex-R. Está pensado para aplicaciones de embebidos en entornos de seguridad. Pipeline de 8 etapas, predicción de saltos, bus de 64 bits (AXI3) y controlador de interrupciones GIC. Puede ejecutar instrucciones en modo ARM y modo Thumb-2. La frecuencia típica de reloj es del orden de los 600MHz. Se liberó en 2006.
- **Cortex-R5.** Similar al Cortex-R4 pero con la posibilidad de usar multicore de dos procesadores en modo asimétrico (AMP). Liberado en 2010.
- **Cortex-R7.** El procesador más grande de la familia R. Trabaja en ejecución fuera de orden con pipeline de 11 etapas y pudiendo trabajar con un reloj de 1GHz típicamente. Soporta división por hardware, DSP y punto flotante. También soporta instrucciones en modo ARM y modo Thumb-2

# Procesador Cortex-R4



- En la presentación nos vamos a centrar en el Cortex-R4. Sus principales características son:
  - Arquitectura ARMv7 que soporta set de instrucciones ARM y Thumb-2
  - Pipeline de 8 etapas
  - Unidad de predicción de saltos
  - Unidad de punto flotante
  - Soporte para interrupciones rápidas
  - TCM y cache.
  - Unidad de protección de memoria
  - Interfaz AXI3 de 64 bits

# Cortex-R. Modos del procesador

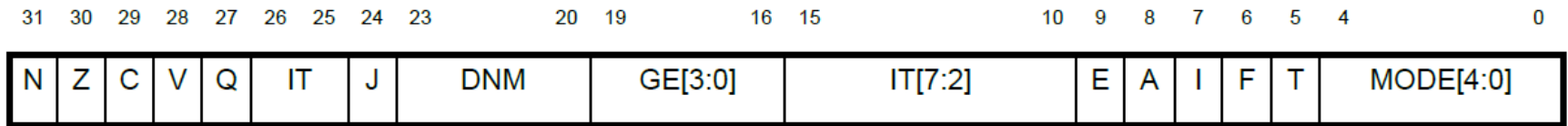
Exception modes	Mode	Description	
	Supervisor (SVC)	Entered on reset and when a Software Interrupt instruction (SWI) is executed	Privileged modes
	Undef	Used to handle undefined instructions	
	Abort	Used to handle memory access violations	
	FIQ	Entered when a high priority (fast) interrupt is raised	
	IRQ	Entered when a low priority (normal) interrupt is raised	
	System	Privileged mode using the same registers as User mode	Unprivileged mode
	User	Mode under which most Applications / OS tasks run	

- El procesador tiene 7 modos de ejecución.
- El único modo sin privilegios es el modo usuario.
- Cada modo va a tener su propio conjunto de registros (salvo system y user que los comparten).
- Los modos van a estar asociados a diferentes condiciones de ejecución.

# Registros del procesador

	User/System mode	FIQ mode	IRQ mode	Supervisor mode	Abort Exception	Undefined Instruction
	R0	R0	R0	R0	R0	R0
	R1	R1	R1	R1	R1	R1
	R2	R2	R2	R2	R2	R2
	R3	R3	R3	R3	R3	R3
	R4	R4	R4	R4	R4	R4
	R5	R5	R5	R5	R5	R5
	R6	R6	R6	R6	R6	R6
	R7	R7	R7	R7	R7	R7
	R8	R8 FIQ	R8	R8	R8	R8
	R9	R9 FIQ	R9	R9	R9	R9
	R10	R10 FIQ	R10	R10	R10	R10
	R11	R11 FIQ	R11	R11	R11	R11
	R12	R12 FIQ	R12	R12	R12	R12
Stack Pointer (SP)	R13 SP	R13 FIQ	R13 IRQ	R13 SVC	R13 ABORT	R13 UNDEF
Link Register (LR)	R14 LR	R14 FIQ	R14 IRQ	R14 SVC	R14 ABORT	R14 UNDEF
Program Counter (PC)	R15 PC	R15 PC	R15 PC	R15 PC	R15 PC	R15 PC
Current Program Status Register (CPSR)	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
Saved Program Status Register (SPSR)		SPSR FIQ	SPSR IRQ	SPSR SVC	SPSR ABORT	SPSR UNDEF

# Registro de estado. CPSR



- **Condition Code Flags**

N = ALU Negative result

Z = ALU Zero result

C = ALU Carry out

V = ALU arithmetic overflow

Q = ALU sticky overflow

- J = Java state bit (always reads 0)

- IT[7:0] = If-Then

- DNM (Do Not Modify)

- GE[3:0] = Greater Than or Equal To

- E = Endianism of Data

- A = Imprecise Abort Disable

- **Interrupt Disable bits**

I = 1, disables the IRQ

F = 1, disables the FIQ

- State bit

T = 0, 32-bit instruction set

T = 1, 16-bit instruction set

- **Mode (defines processor mode)**

M[4:0] = 10000 User mode

M[4:0] = 10001 FIQ mode

M[4:0] = 10010 IRQ mode

M[4:0] = 10011 Supervisor mode

M[4:0] = 10111 Abort mode

M[4:0] = 11011 Undefined mode

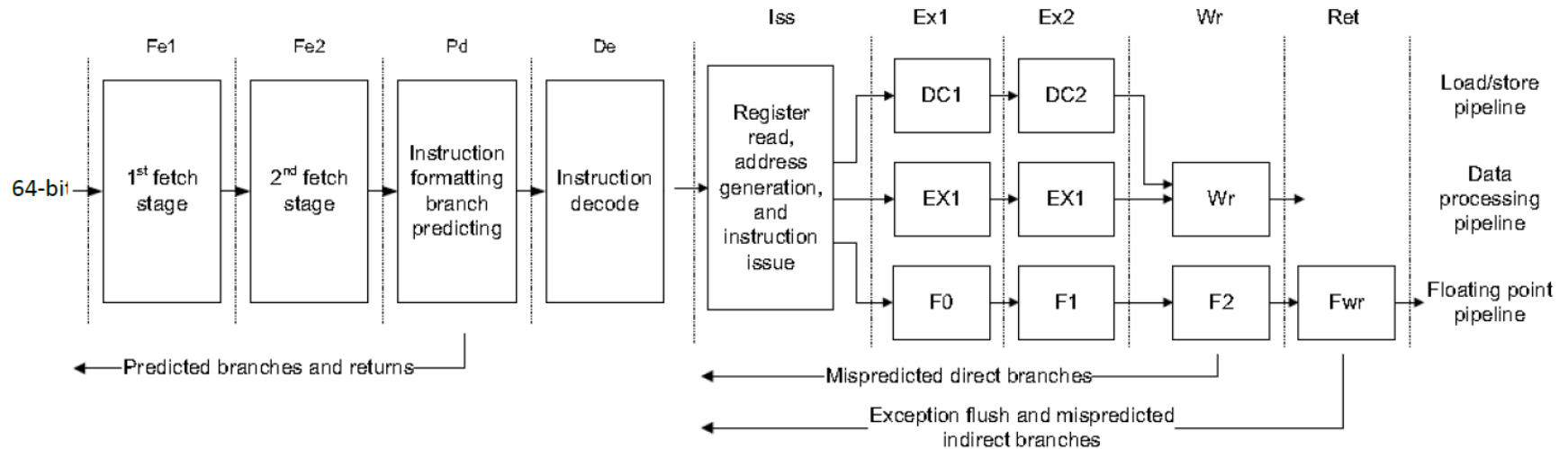
M[4:0] = 11111 System mode



# Pipeline

The following stages make up the pipeline:

- Fetch stages (Fe1, Fe2)
- Pre-Decode and Decode stages (Pd, De)
- Issue stage (Iss)
- Execution stages (Ex1, Ex2, etc.)



# Unidad de punto flotante

- Todos los Cortex pueden tener unidad de punto flotante (Floating Point Unit). La FPU tiene las siguientes características:
  - Compatible con IEEE-754
  - Soporta precisión simple (32 bits) y precisión doble (64 bits)
  - Soporta para precisión simple y doble: suma, resta, multiplicación, división, multiplicar y sumar y raíz cuadrada
  - Conversión de punto fijo a punto flotante y viceversa.
  - Comparaciones
  - Excepciones

# ARM VFP

- La extensión ARM VFP (vector floating point coprocessor) implementa los números en punto flotante de acuerdo a la IEEE-754 (salvo los números subnormales que los convierte a cero y el redondeo al número más cercano).
- Puede implementarse con 32 registros o 16 registros de 64 bits.
- La VFP agrega los registros:
  - ***Floating Point System ID Register (FPSID)***. Que permite determinar que características están presentes en un hardware particular.
  - ***Floating Point Status and Control Register (FPSCR)***. Setea como se redondea y tiene el resultado de las comparaciones principalmente.
  - ***Floating Point Exception Register (FPEXC)***. Configura las excepciones de la VFP
  - **Media and VFP Feature registers 0 and 1 (MVFR0 and MVFR1)**. Configura el uso de instrucciones SIMD y propias de la implementación.

# ARM VFP. Flags


IEEE-754 relationship	ARM APSR flags			
	N	Z	C	V
Equal	0	1	1	0
Less Than (LT)	1	0	0	0
Greater Than (GT)	0	0	1	0
Unordered (At least one argument was NaN)	0	0	1	1

```
VCMP d0, d1
VMRS APSR_nzcv, FPSCR
BNE label
```

- Para poder tomar acciones en función del estado de los números de punto flotante es necesario que la VFP actualice los Flags del APSR.
- Por lo que después de actualizar los flags del procesador estos no van a representar exactamente para lo que fueron implementados pero se van a poder usar los saltos condicionales en función de valores en punto flotante.

# Tabla de Excepciones

Vector Table Offset	Exception	Mode on Entry	Bits in CPSR		
			A	F	I
0x00	Reset	SuperVisor	Set	Set	Set
0x04	UNDEF	Undefined (Mode)	Unchanged	Unchanged	Set
0x08	SVC	SuperVisor	Unchanged	Unchanged	Set
0x0C	PABT	Abort	Set	Unchanged	Set
0x10	DABT	Abort	Set	Unchanged	Set
0x14	Reserved				
0x18	IRQ	IRQ	Set	Unchanged	Set
0x1C	FIQ	FIQ	Set	Set	Set

- Reset Highest Priority
  - DABT
  - FIQ
  - IRQ
  - UNDEF
  - PABT
  - SVC Lowest Priority
- 

- Las prioridades de las excepciones están fijas para los Cortex-R.
- Se puede observar que cuando sucede una FIQ (Fast IRQ) Las IRQ se deshabilitan automáticamente (setea el bit I).
- Las IRQ por defecto no permite anidar interrupciones ya que el procesador deshabilita las IRQ cuando comienza a atender una de ellas.

# Manejo de excepciones

- Cuando ocurre una excepción el procesador:
  1. Copia el CPSR del estado anterior al SPSR del estado nuevo
  2. Actualiza el LR
  3. Pasa al procesador a modo ARM (actualiza CPSR)
  4. Cambia al modo de excepción actual (actualiza CPSR)
  5. Deshabilita las IRQ (actualizando CPSR)
  6. Sí es una FIQ también deshabilita las FIQ
  7. Guarda la dirección de retorno en el LR del estado nuevo.
  8. Carga PC con dirección de la tabla de excepciones correspondiente al estado nuevo
  9. Cambia el banco de registros.
- Al volver de la excepción el handler debe:
  1. Copiar el SPSR del estado viejo al CPSR
  2. Pasar el contenido de LR a PC.

# Manejo de interrupciones

- Cómo la tabla de excepciones es fija y todas las IRQ tienen una única dirección de rutina de atención de interrupción así como todas las FIQ también tienen una única rutina de atención de interrupción pueden darse los siguientes casos:
  1. **Modo de interrupción indexado.** En la IRQ o FIQ se decide, por software, leyendo registros, que interrupción hay que atender.
  2. **Modo de interrupción vectorizado por registro.** En la IRQ se salta al contenido de un registro del controlador de interrupciones que ya tiene la tabla definida.
  3. **Modo de interrupción vectorizado por hardware.** En este modo el vector de la tabla de excepciones no se carga por software, sino que lo carga el controlador de interrupciones y se evita la carga del registro y salto del modo anterior.

# Modo de interrupción indexado

- El evento ocurre en el periférico
- El periférico pide la IRQ/FIQ al controlador de interrupciones.
- El controlador de interrupciones elige y genera la IRQ o FIQ
- El CPU salta a la dirección 0x18 o 0x1C
- Se salta a la rutina de atención de interrupción (ISR) general de la IRQ/FIQ
- Se carga en algún registro la ISR particular
- Se salta a esa ISR



# Modo de interrupción vectorizado por registros

- El evento ocurre en el periférico
- El periférico pide la IRQ/FIQ al controlador de interrupciones.
- El controlador de interrupciones elige y genera la IRQ o FIQ
- El CPU salta a la dirección 0x18 o 0x1C
- En la dirección 0x18 o 0x1C está la instrucción ***LDR PC,[PC,#-0x1B0]*** (por ejemplo) que apunta al registro que tiene la dirección de la ISR particular.
- Se salta a la ISR

# Modo de interrupción vectorizado por hardware

- El evento ocurre en el periférico
- El periférico pide la IRQ/FIQ al controlador de interrupciones.
- El controlador de interrupciones elige y genera la IRQ o FIQ
- El CPU salta a la dirección 0x18 o 0x1C
- El controlador de interrupciones tiene acceso a las posiciones de memoria 0x18 y 0x1C y directamente coloca ahí la dirección de la ISR particular.
- Se salta a la ISR

# Modo Abort

- ***Prefetch Abort***, la CPU trata de ejecutar una instrucción y :
  - La posición de memoria no está implementada en el sistema
  - La región de memoria está protegida por la MPU.
  - Se detectó un error en la lógica de ECC (error correction code)
  
- ***Data Abort***, la CPU trata de leer o escribir datos y :
  - La posición de memoria no está implementada en el sistema
  - La región de memoria está protegida por la MPU.
  - La posición de memoria es de lectura y/o escritura en modo privilegiado (cuando el procesador está en modo usuario).
  - Se detectó un error en la lógica de ECC (error correction code)

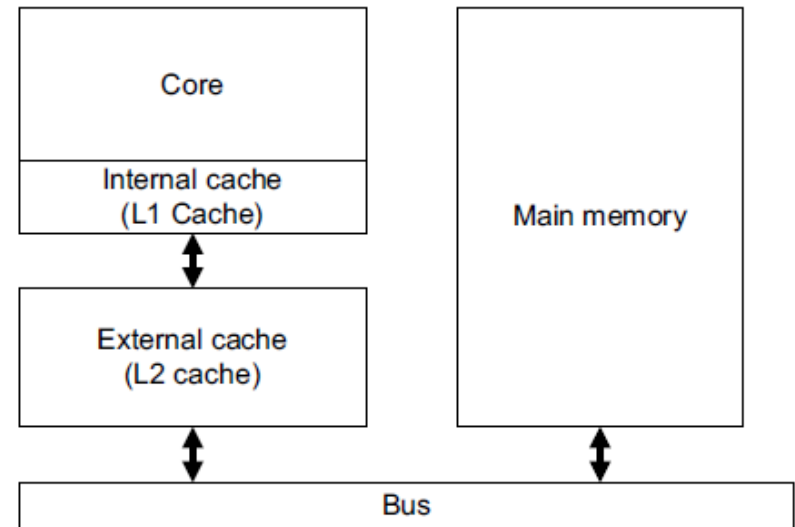
# Assembler. Cambio de modo

- A diferencia de los Cortex-M los Cortex-R pueden trabajar con dos sets de instrucciones:
  - En **modo ARM**. Donde las instrucciones son de 32 bits y mucho más versátiles.
  - En **modo Thumb**. Las instrucciones son de 16 bits y son más acotadas (como las instrucciones condicionales, por ejemplo).
- Para cambiar entre sets de instrucciones se utiliza la instrucción BX o BLX ya que cuando se está en un set de instrucciones no se pueden usar instrucciones del otro.
- BX Rm cambia de modo si:
  - Sí Rm[0] = 0 corre en modo ARM.
  - Sí Rm[0] = 1 corre en modo Thumb

```
CMP      R0, R1
MOVGE    R2, R1 @ R1 is less than or equal to R1
MOVLTE   R2, R0 @ R0 is less than R1
```

# Cache

- Básicamente una memoria cache es una memoria rápida y pequeña que está entre la memoria del sistema y el procesador.
- La memoria cache es transparente al mapa de memoria del procesador
- Su efectividad se basa en que los programas no se ejecutan de manera totalmente aleatoria (algunas zonas de memoria se usan mucho más que otras, por ejemplo en los lazos).
- Los ARM cuando lo implementan, tienen cache separados para instrucciones (cache I) y para datos (cache D)

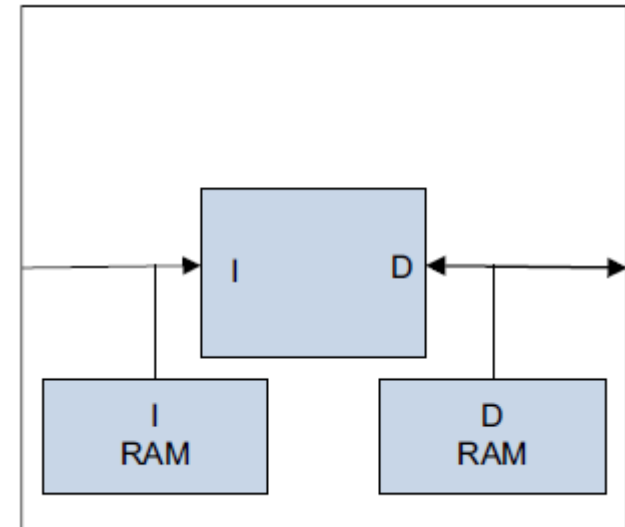


# Inconvenientes del cache

- El cache acelera la ejecución de un programa ya que al tener en una memoria rápida precargado el programa a correr en el procesador se ganan ciclos de ejecución.
- Cuando el cache está lleno o se requiere recargar o saltar a una zona de memoria se va penalizar la ejecución de ese código.
- Por lo que la ejecución de código puede variar temporalmente de manera muy significativa en función del código o los datos que se ejecutaron anteriormente.
- Es decir, no se puede asegurar el tiempo de ejecución de cierto código porque depende del estado del cache atentando contra el determinismo. Para sistemas en tiempo real donde el determinismo es esencial

# Tightly Coupled Memory (TCM).

- La memoria “altamente acoplada” (TCM) también es una memoria RAM de muy alta velocidad, de relativamente poco tamaño (similar a la cache).
- En los procesadores ARM también va a existir TCM de instrucciones (TCM-I) y de datos (TCM-D).
- La principal diferencia entre la TCM y el cache es que **la TCM forma parte del mapa de memoria del procesador.**
- Al formar parte del mapa de memoria el programador decide que ubica en la TCM y se evitan los problemas de determinismo que traía el cache.



# Unidad de protección de memoria

- Muchos sistemas de tiempo real trabajan con sistemas operativos multitarea. Los sistemas operativos proveen la abstracción de que las tareas son independientes entre sí, pero ¿Qué pasa si una tarea sale de control?
- Generalmente el sistema operativo se apoya en el hardware. Para la ejecución de código en los modos con y sin privilegios del procesador (las tareas no suelen tenerlos) y para que la tarea no use zonas de memoria no permitidas en la **unidad de protección de memoria MPU (Memory Protection Unit)**.
- La MPU permite definir zonas de memoria con diferentes accesos y controlar que se cumplan por hardware.

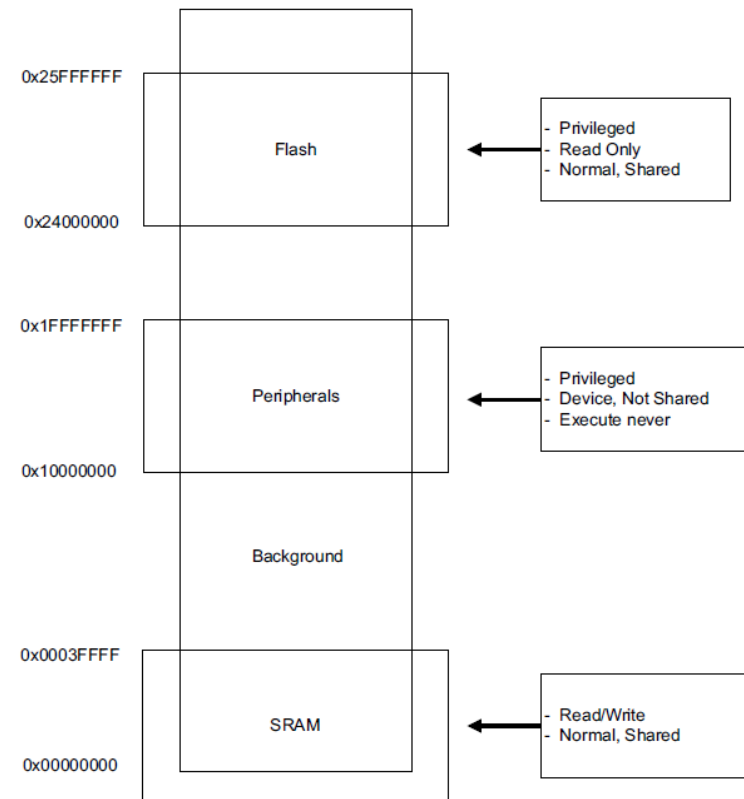


# Zonas de memoria

- La MPU de los Cortex R4/R5 puede definir hasta 12 zonas de memoria.
- En caso de un acceso indebido a memoria se genera una excepción tipo Abort.
- Cada zona de memoria define:
  - Una dirección base de memoria
  - Un tamaño de la zona de memoria
  - El tipo de memoria y el control de acceso
- En la implementación de un sistema con zonas de memoria en Cortex-R hay que tener en cuenta:
  - Las zonas tienen prioridades y estas son independientes de los privilegios de cada zona.
  - Las regiones se pueden solapar y los privilegios resultantes son los de la zona de más prioridad.
  - Las regiones tienen que tener entre 32bytes y 4GB
  - Un acceso a una posición de memoria no alcanzada por ninguna zona genera una excepción tipo Abort

# Sistema con zonas de memoria

- Es común definir una zona de prioridad más baja que todas las demás que cubre los 4GB que provee acceso de modo privilegiado.
- Luego se definen las zonas de memoria en función de los periféricos, RAM y FLASH por ejemplo.



# Registros MPU.

- MPU Base Address register. Define el comienzo de la zona (alineada a 32 bytes)
- MPU Region Size and Enable Register. Habilita la MPU y define el tamaño de la zona.
- MPU Region Access Control Register. Define el tipo de acceso a las zonas de memoria

# Permisos MPU

AP	Privileged	Unprivileged	Description
00	No access	No access	Permission fault
01	Read/Write	No access	Privileged Access only
10	Read/Write	Read	No user-mode write
11	Read/Write	Read/Write	Full access
00	-	-	Reserved
01	Read	No access	Privileged Read only
10	Read	Read	Read only
11	-	-	Reserved

# Inicio Cortex-R

- Cuando se inicia el procesador, en un Cortex-R puede comenzar en la dirección 0x00000000 o bien en la dirección 0xFFFF0000. La rutina de atención de reset debe:
  - ☐ En un sistema multicore, poner a dormir todos los cores menos el que inicializa el sistema.
  - ☐ Inicializar el vector de excepciones.
  - ☐ Inicializa la memoria, incluyendo la MPU
  - ☐ Inicializar los stacks de todos los modos y registros.
  - ☐ Inicializar toda la entrada/salida crítica
  - ☐ Inicializa la VFP
  - ☐ Habilitar las interrupciones
  - ☐ Cambiar el modo de operación
  - ☐ Llamar a la función main()

# Modos de bajo consumo

- Los procesadores ARM soportan los siguientes modos de bajo consumo:
  - **Standby.** En este modo el procesador permanece alimentado, pero sin clock. Se entra en este modo usando **WFI** o **WFE** y se sale con una interrupción, son pocos los ciclos de reloj requeridos para entrar o salir.
  - **Retention.** Este modo es similar a standby pero no requiere reiniciar el procesador. Desde el punto de vista del usuario es similar a standby
  - **Power Down.** En este modo se le quita la alimentación al procesador y generalmente se sale de este modo por algún hardware específico o por un reset.
  - **Dormant mode.** En este modo se le quita la alimentación al procesador, pero se mantiene la RAM y las TCM

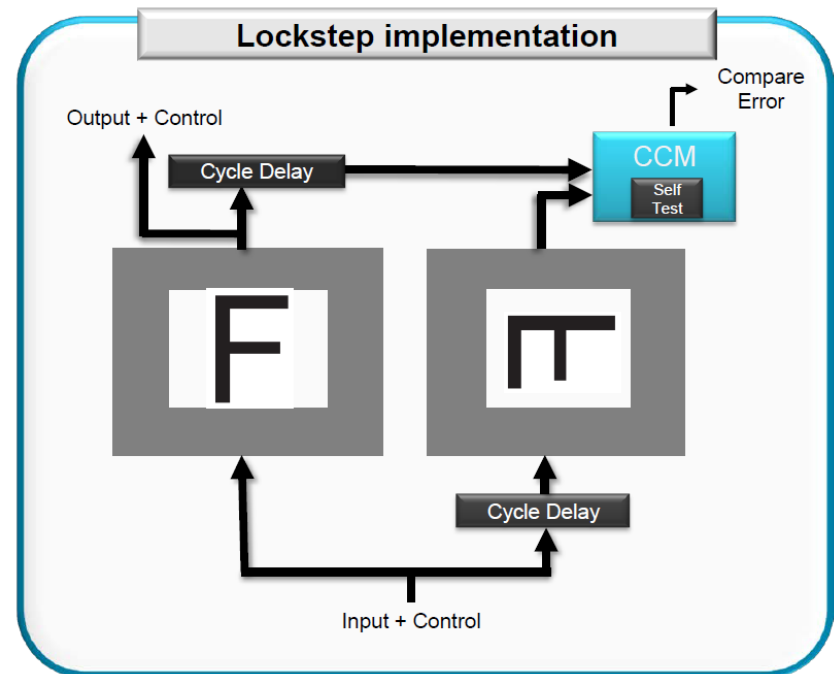
# Ejemplo de aplicación. RM46852

## ■ Características generales

- Procesador Cortex-R4F
- Unidad de punto flotante
- 1.25MB de Flash ECC
- 192KB de RAM ECC
- FCLK 220MHz

## ■ Características para seguridad

- Dos procesadores corriendo en “lockstep” es decir dos procesadores sincrónicos y hardware que compara sus estados.
- MPU de doce zonas de memoria.
- RAM y FLASH con corrección de errores ECC.
- Sin cache FLASH y RAM implementadas como TCM



# Herramientas de desarrollo

- Code Composer Studio. IDE que utiliza el toolchain GNU basado en Eclipse.  
<http://www.ti.com/tool/CCSTUDIO>
- Hardware Abstraction Layer Code Generator for Hercules MCUs  
<http://www.ti.com/tool/HALCOGEN>
- Hercules RM46x LaunchPad Development Kit  
<http://www.ti.com/tool/LAUNCHXL2-RM46>



# Código de ejemplo

## main.c

```
/* USER CODE BEGIN (2) */
uint32_t flag;
/* USER CODE END */

int main(void)
{
/* USER CODE BEGIN (3) */
    flag=0;
    gpioInit();
    rtiInit();
    rtiEnableNotification(rtiNOTIFICATION_COMPARE0);
    rtiStartCounter(rtiCOUNTER_BLOCK0);
    gpioSetBit(gioPORTB, 1, 0);
    gpioSetBit(gioPORTB, 2, 1);
    _enable_interrupt_();
    while(1)
    {
        if(flag)
        {
            flag = 0;
            gpioToggleBit(gioPORTB, 1);
            gpioToggleBit(gioPORTB, 2);
        }
    }
/* USER CODE END */

    //return 0;
}
```

## rti.c

```
/* SourceId : RTI_SourceId_022 */
/* DesignId : RTI_DesignId_022 */
/* Requirements : HL_SR95 */
void rtiCompare0Interrupt(void)
{
/* USER CODE BEGIN (74) */
    static uint32_t demora = TICS_LED;
/* USER CODE END */

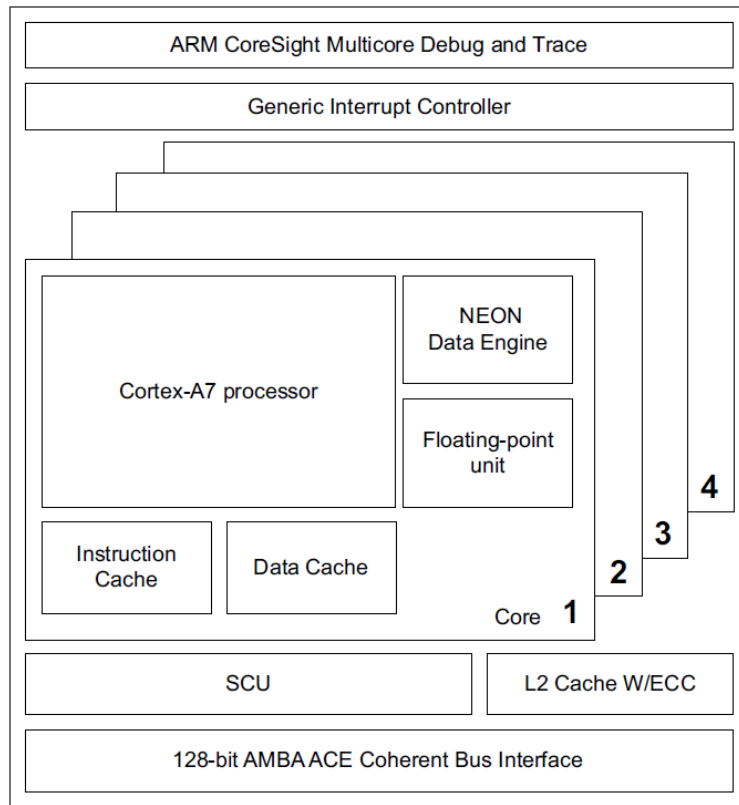
    rtiREG1->INTFLAG = 1U;
    rtiNotification(rtiNOTIFICATION_COMPARE0);

/* USER CODE BEGIN (75) */
    if(!--demora)
    {
        demora = TICS_LED;
        flag=1;
    }
/* USER CODE END */
}
```

# Cortex-A. Versiones

	Processor					
	Cortex-A5	Cortex-A7	Cortex-A8	Cortex-A9	Cortex-A12	Cortex-A15
Release date	Dec 2009	Oct 2011	July 2006	March 2008	June 2013	April 2011
Typical clock speed	~1GHz	~1GHz on 28nm	~1GHz on 65nm	~2GHz on 40nm	~2GHz on 28nm	~2.5GHz on 28nm
Execution order	In-order	In-order	In-order	Out of order	Out of order	Out of order
Cores	1 to 4	1 to 4	1	1 to 4	1 to 4	1 to 4
Peak integer throughput	1.6DMIPS/MHz	1.9DMIPS/MHz	2DMIPS/MHz	2.5DMIPS/MHz	3.0DMIPS/MHz	3.5DMIPS/MHz
VFP architecture	VFPv4	VFPv4	VFPv3	VFPv3	VFPv4	VFPv4
NEON architecture	NEON	NEONv2	NEON	NEON	NEONv2	NEONv2
Half precision extension	Yes	Yes	No	Yes	Yes	Yes
Hardware Divide	No	Yes	No	No	Yes	Yes
Fused Multiply Accumulate	Yes	Yes	No	No	Yes	Yes
Pipeline stages	8	8	13	9 to 12	11	15+
Instructions decoded per cycle	1	Partial dual issue	2 (Superscalar)	2 (Superscalar)	2 (Superscalar)	3 (Superscalar)
Return stack entries	4	8	8	8	8	48
LPAA	No	Yes	No	No	Yes	Yes
Floating Point Unit	Optional	Yes	Yes	Optional	Yes	Optional
AMBA interface	64-bit AMBA 3	128-bit AMBA 4	64 or 128-bit AMBA 3	2x 64-bit AMBA 3	128-bit AMBA 4	128-bit AMBA 4
Generic Interrupt Controller (GIC)	Included	Optional	Not included	Included	External	Optional
Trace	Optional ETM	Optional ETM separate macrocell	Integrated ETM	Integrated PTM	Integrated PTM	Integrated PTM

# Procesadores Cortex-A.



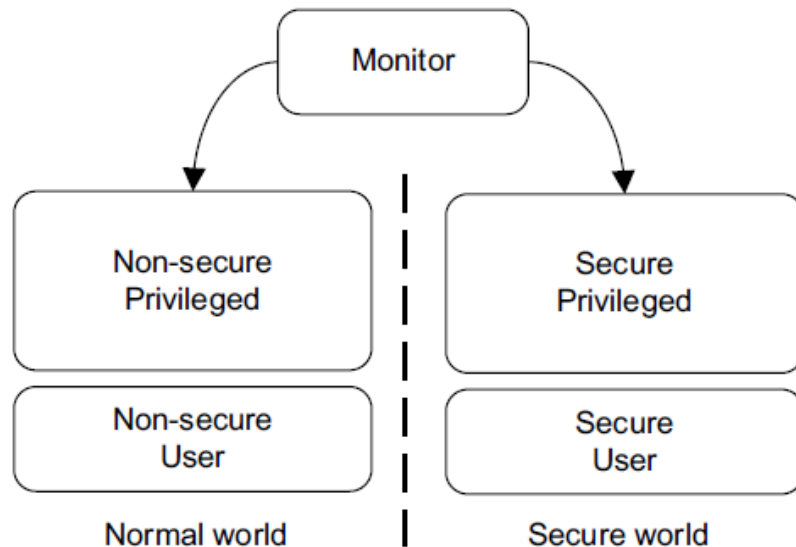
- Las principales características de los procesadores Cortex-A se pueden describir como:
  - Core RISC de 32 bits
  - Arquitectura ARMv7 que soporta set de instrucciones ARM y Thumb-2
  - Unidad de predicción de saltos
  - Unidad de punto flotante
  - Unidad de operaciones SIMD (NEON)
  - Cache de instrucciones y datos.
  - Unidad de administración de memoria y uso de paginado y memoria virtual.

# Cortex-A. Modos del procesador

Exception modes	Mode	Description	
	Supervisor (SVC)	Entered on reset and when a Software Interrupt instruction (SWI) is executed	Privileged modes
	Undef	Used to handle undefined instructions	
	Abort	Used to handle memory access violations	
	FIQ	Entered when a high priority (fast) interrupt is raised	
	IRQ	Entered when a low priority (normal) interrupt is raised	
	System	Privileged mode using the same registers as User mode	Unprivileged mode
	User	Mode under which most Applications / OS tasks run	

- El procesador tiene 7 modos de ejecución.
- El único modo sin privilegios es el modo usuario.
- Cada modo va a tener su propio conjunto de registros (salvo system y user que los comparten).
- Los modos van a estar asociados a diferentes condiciones de ejecución.

# Cortex-A. Zonas seguras.

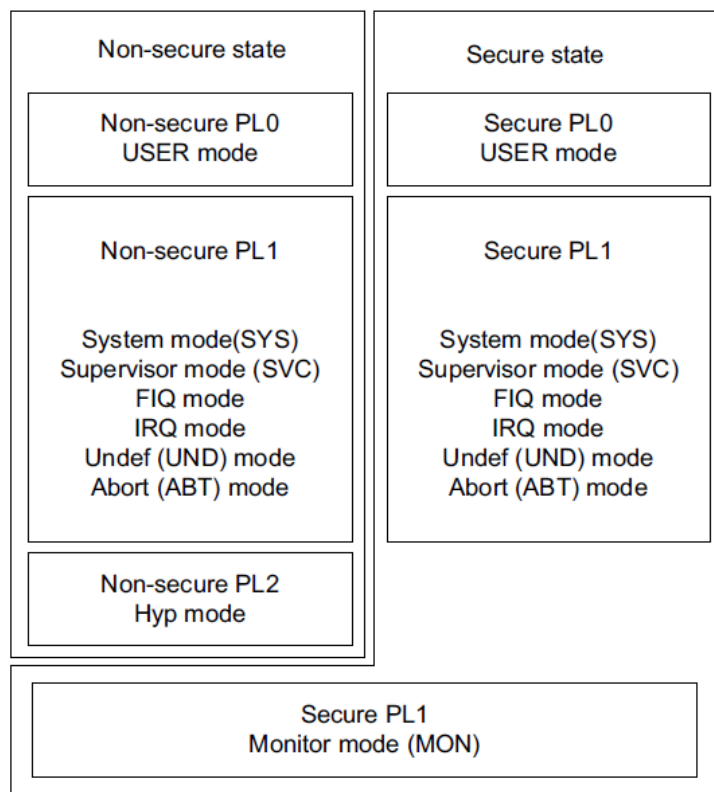


- Las zonas seguras (TrustZone Security Extensions) crean dos estados del procesador: “seguro e inseguro” que son independientes de los modos del procesador.
- Agrega un modo monitor que permite pasar de manera controlada de un modo de seguridad al otro.
- Básicamente se divide el hardware y software en dos y todo los recursos “no seguros” no tienen acceso a los recursos seguros.

# Cortex-A. Niveles de privilegio

- Cada modo del procesador tiene asociado un nivel de privilegio. Para los Cortex-A se los define:
  - PL0. Es el nivel sin privilegio. No tiene acceso a los registros de configuración y puede ejecutar accesos sin privilegios a memoria. El modo usuario (User Mode) tiene nivel de privilegio 0.
  - PL1. Es el modo que normalmente tienen los sistemas operativos y en el que se ejecutan todos los modos que no son el User ni el Hyp.
  - PL2. Si el procesador tiene implementadas las extensiones para Virtualización, el modo Hypervisor (modo que controla los sistemas operativos virtuales que corren en PL1) lo usa.
- Los niveles de privilegio no dependen de los estados seguros.

# Cortex-A. Modos y niveles de privilegio



Mode	Encoding	Function	Security state	Privilege level
User (USR)	10000	Unprivileged mode in which most applications run	Both	PL0
FIQ	10001	Entered on an FIQ interrupt exception	Both	PL1
IRQ	10010	Entered on an IRQ interrupt exception	Both	PL1
Supervisor (SVC)	10011	Entered on reset or when a Supervisor Call instruction (SVC) is executed	Both	PL1
Monitor (MON)	10110	Implemented with Security Extensions. See <a href="#">Chapter 21</a>	Secure only	PL1
Abort (ABT)	10111	Entered on a memory access exception	Both	PL1
Hyp (HYP)	11010	Implemented with Virtualization Extensions. See <a href="#">Chapter 22</a>	Non-secure	PL2
Undef (UND)	11011	Entered when an undefined instruction executed	Both	PL1
System (SYS)	11111	Privileged mode, sharing the register view with User mode	Both	PL1

# Registros del procesador

R0	R0	R0	R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7	R7	R7	R7
R8	R8	R8_fiq	R8	R8	R8	R8	R8	R8
R9	R9	R9_fiq	R9	R9	R9	R9	R9	R9
R10	R10	R10_fiq	R10	R10	R10	R10	R10	R10
R11	R11	R11_fiq	R11	R11	R11	R11	R11	R11
R12	R12	R12_fiq	R12	R12	R12	R12	R12	R12
R13 (sp)	R13 (sp)	SP_fiq	SP_svc	SP_abt	SP_svc	SP_und	SP_mon	SP_hyp
R14 (lr)	R14 (lr)	LR_fiq	LR_svc	LR_abt	LR_svc	LR_und	LR_mon	R14 (lr)
R15 (pc)	R15 (pc)	R15 (pc)	R15 (pc)	R15 (pc)	R15 (pc)	R15 (pc)	R15 (pc)	R15 (pc)

(A/C)PSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
		SPSR_fiq	SPSR_irq	SPSR_abt	SPSR_svc	SPSR_und	SPSR_mon	SPSR_hyp
								ELR_hyp

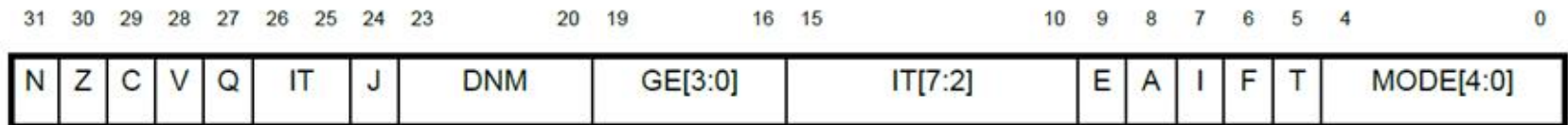
  

User	Sys	FIQ	IRQ	ABT	SVC	UND	MON	HYP
------	-----	-----	-----	-----	-----	-----	-----	-----

Banked



# Registro de estado. CPSR



- **Condition Code Flags**

N = ALU Negative result

Z = ALU Zero result

C = ALU Carry out

V = ALU arithmetic overflow

Q = ALU sticky overflow

- J = Java state bit (always reads 0)

- IT[7:0] = If-Then

- DNM (Do Not Modify)

- GE[3:0] = Greater Than or Equal To

- E = Endianism of Data

- A = Imprecise Abort Disable

- **Interrupt Disable bits**

I = 1, disables the IRQ

F = 1, disables the FIQ

- **State bit**

T = 0, 32-bit instruction set

T = 1, 16-bit instruction set

# Assembler. Cambio de modo

- A diferencia de los Cortex-M los Cortex-A pueden trabajar con dos sets de instrucciones:
  - En **modo ARM**. Donde las instrucciones son de 32 bits y mucho más versátiles.
  - En **modo Thumb**. Las instrucciones son de 16 bits y son más acotadas (como las instrucciones condicionales, por ejemplo).
- Para cambiar entre sets de instrucciones se utiliza la instrucción BX o BLX ya que cuando se está en un set de instrucciones no se pueden usar instrucciones del otro.
- BX Rm cambia de modo si:
  - Sí Rm[0] = 0 corre en modo ARM.
  - Sí Rm[0] = 1 corre en modo Thumb

```
CMP      R0, R1
MOVGE    R2, R1 @ R1 is less than or equal to R1
MOVLT    R2, R0 @ R0 is less than R1
```

# ARM VFP

- La extensión ARM VFP (vector floating point coprocessor) implementa los números en punto flotante de acuerdo a la IEEE-754 (salvo los números subnormales que los convierte a cero y el redondeo al número más cercano).
- Puede implementarse con 32 registros o 16 registros de 64 bits.
- La VFP agrega los registros:
  - ***Floating Point System ID Register (FPSID)***. Que permite determinar que características están presentes en un hardware particular.
  - ***Floating Point Status and Control Register (FPSCR)***. Setea como se redondea y tiene el resultado de las comparaciones principalmente.
  - ***Floating Point Exception Register (FPEXC)***. Configura las excepciones de la VFP
  - **Media and VFP Feature registers 0 and 1 (MVFR0 and MVFR1)**. Configura el uso de instrucciones SIMD y propias de la implementación.

# ARM VFP. Flags

IEEE-754 relationship	ARM APSR flags			
	N	Z	C	V
Equal	0	1	1	0
Less Than (LT)	1	0	0	0
Greater Than (GT)	0	0	1	0
Unordered (At least one argument was NaN)	0	0	1	1

```
VCMP d0, d1
VMRS APSR_nzcv, FPSCR
BNE label
```

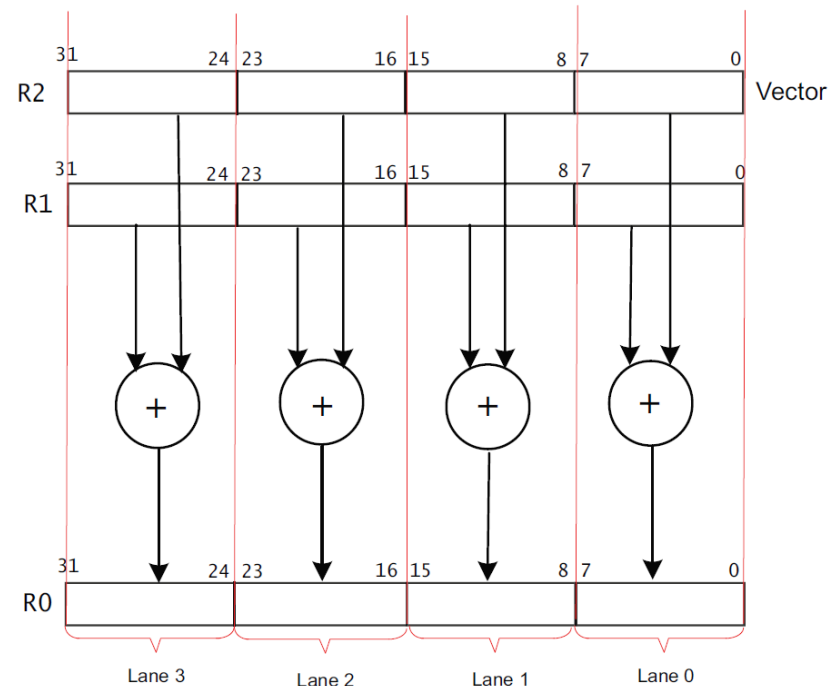
- Para poder tomar acciones en función del estado de los números de punto flotante es necesario que la VFP actualice los Flags del APSR.
- Por lo que después de actualizar los flags del procesador estos no van a representar exactamente para lo que fueron implementados pero se van a poder usar los saltos condicionales en función de valores en punto flotante.

# Instrucciones SIMD

- Los Cortex-A suelen implementar una extensión llamada NEON que permite la ejecución de instrucciones SIMD (Single Instruction Multiple Data). Las instrucciones se pueden clasificar como:
  - **SISD (Single Instruction Single Data)**. Una única instrucción que hacer una única operación. La enorme mayoría de las instrucciones son SISD.
  - **SIMD (Single Instruction Multiple Data)**. Una única instrucción que permite hacer múltiples operaciones.
  - **MISD (Multiple Instruction Single Date)**. Las computadoras tolerantes a fallas ejecutan la misma instrucción en varios cores para detectar posibles fallas.
  - **MIMD (Multiple Instruction Multiple Data)**. Múltiples procesadores realizan operaciones sobre múltiples datos de manera sincornizada (Multicore con SIMD por ejemplo)

# SIMD. Ejemplo: Suma en 8 bits.

- Un ejemplo común de SIMD es la suma de variables de 8bits (Por ejemplo para imágenes). De manera tradicional se necesitan cuatro operaciones ADD para sumar cuatro valores de 8bits.
- Si se utiliza la instrucción:
  - UADD8 R0, R1, R2
- En una única instrucción se realizan 4 sumas optimizando el tiempo de ejecución.
- Es importante resaltar que las operaciones son independientes y no hay acarreo intermedios (a diferencia de una suma con ADD).

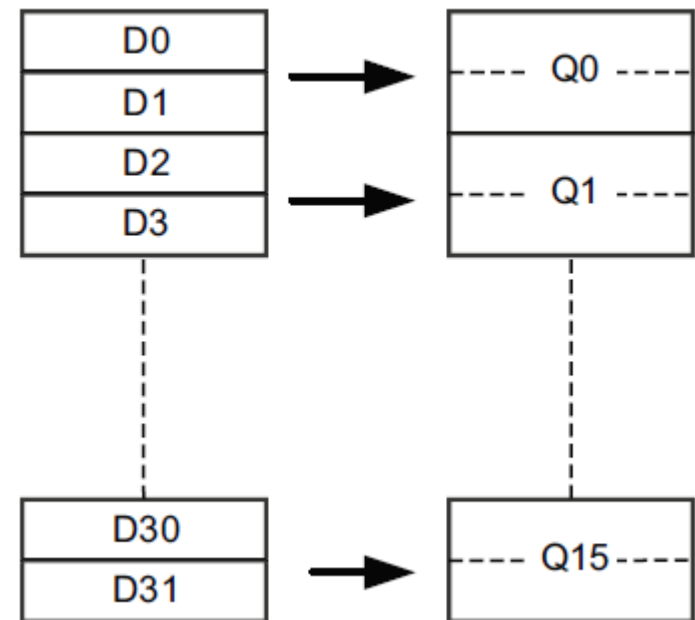
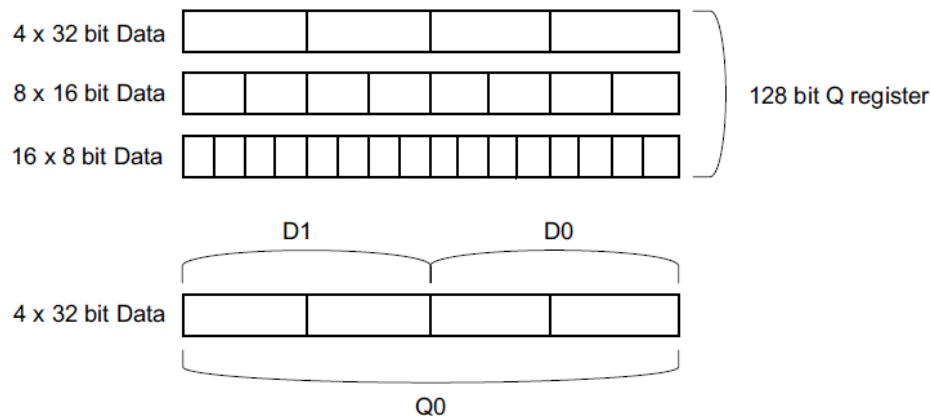


# ARM NEON

- NEON es una extensión de ARM que:
  - Permite hacer operaciones de tipo SIMD tanto en modo ARM como en Thumb.
  - Se implementa con un conjunto de registros de 64bits o 128bits.
  - Puede trabajar con la VFP.
  - Maneja los siguientes tipos de datos:
    - Punto flotante de 32bits.
    - Magnitudes o enteros de 8, 16, 32, 64 bits.
    - Polinomios de 8 y 16bits (CRC)

# NEON. Registros

- Los registros de NEON se pueden ver como 16 registros de 128bits (Q0 – Q15) o bien como 32 registros de 64bits (D0 – D31)





# NEON. Tipo de Instrucciones

- **Normal.** Instrucciones que pueden operar en cualquier tipo de vector con vectores de entrada de igual tamaño y de salida del mismo tamaño.
- **Long.** Se parte de vectores de igual tamaño y se obtiene del vectores del doble de tamaño (entrada doubleword, salida qword).
- **Wide.** Se parte de un vector de entrada dword y otro qword con resultado un vector qword.
- **Narrow.** Se parte de dos vectores qword y se obtiene como resultado un vector dword

# NEON. Tipo de Instrucciones (2)

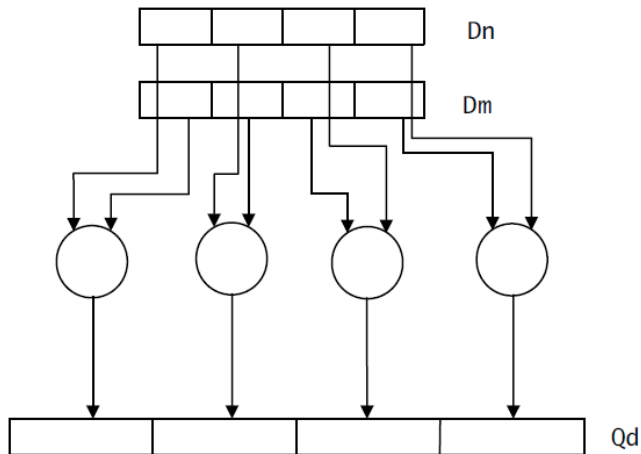


Figure 7-5 NEON long instructions

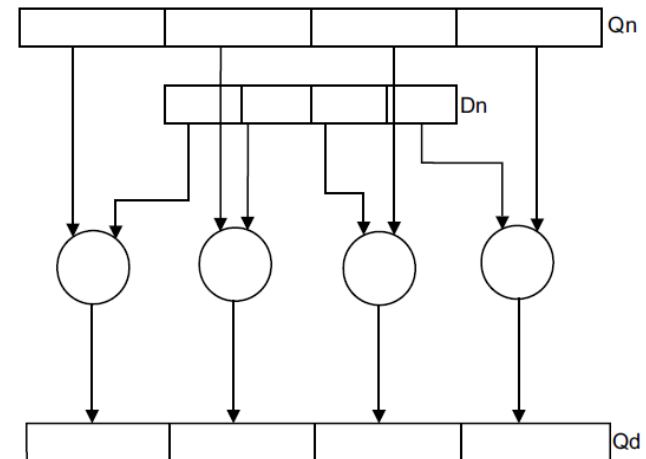


Figure 7-6 NEON wide instructions

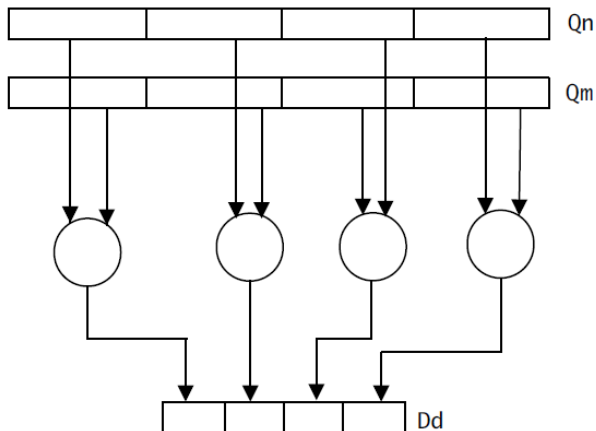
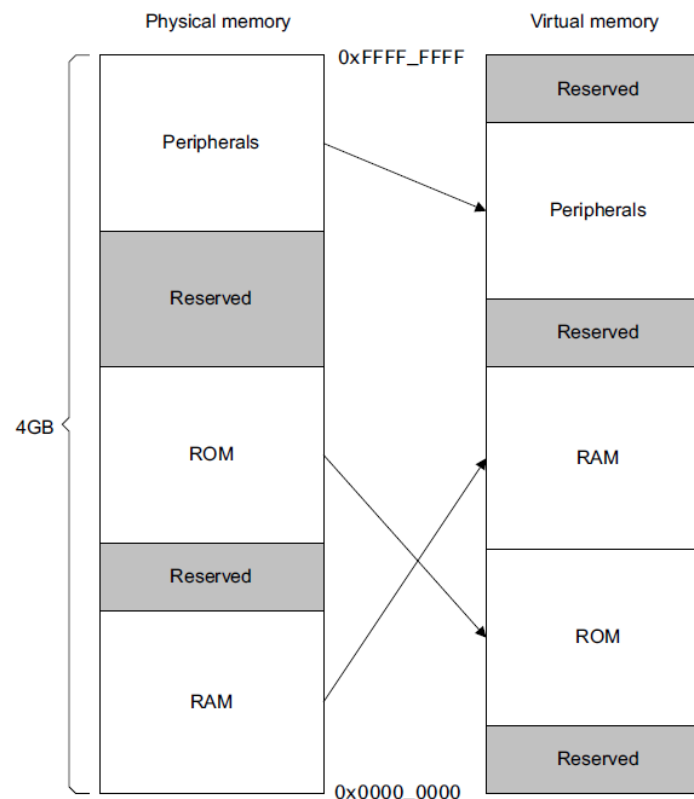


Figure 7-7 NEON narrow instructions

# Unidad de administración de memoria (MMU)

- La unidad de administración de memoria es el hardware que genera las protecciones de memoria como la MPU, pero no es su rol más importante.
- La principal función de la MMU es implementar la **memoria virtual**.
- La idea clave detrás de la memoria virtual es crear mapas de memoria lógicos que luego son “traducidos” a la memoria física.
- Todos los sistemas operativos implementan el esquema de memoria virtual para la gestión de procesos.



# Uso de la memoria virtual en sistemas operativos

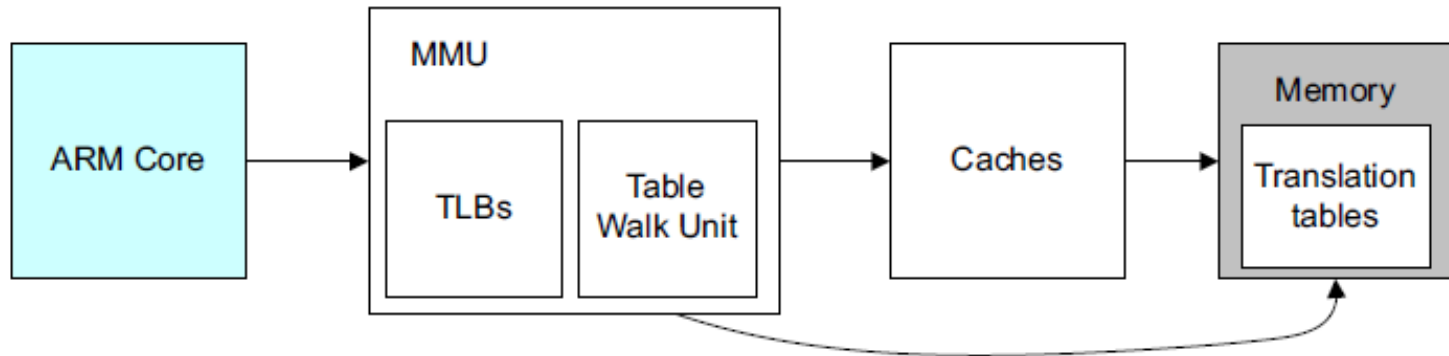
## ■ Ventajas del uso de memoria virtual

- Cada proceso ve su propio espacio de memoria (incluso dos procesos pueden compartir memoria física o si no están corriendo a la vez tener el mismo mapa de memoria)
- Los procesos no pueden interferirse entre sí. Ya que no pueden alcanzar la memoria de los otros.
- Compartir datos entre procesos también es sencillo en función de la definición de los mapas de memoria.
- Se puede hacer una gestión unificada de la memoria física.

## ■ Uso de memoria virtual en sistemas operativos

- Unix la implementó en 3BSD en 1979
- Microsoft la implementó en 1992 para Windows NT
- Apple la implementó en OS X en 2002

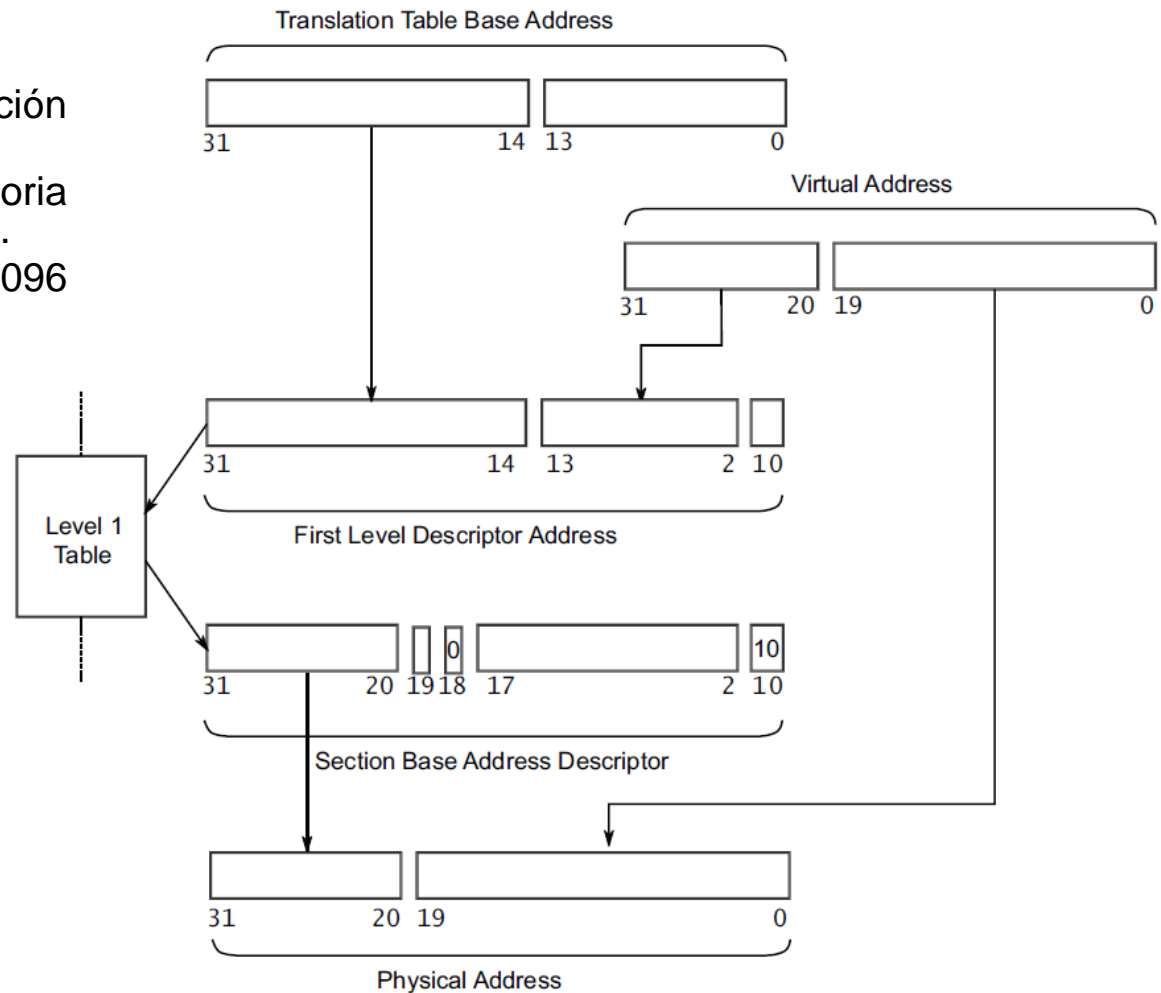
# Partes de la MMU



- Una vez que está activa la MMU todos los direccionamientos pasan a través de ella.
- La función de la MMU es tomar la dirección virtual y de ella los bits más significativos, pasarlos por una o varias tablas y obtener la dirección física a acceder.
- La MMU separa la memoria en páginas (que pueden variar significativamente de implementación a implementación, desde 1KB a 16MB por ejemplo)
- La TLB (Translation lookaside buffer) es un caché de las páginas recientemente accedidas y está dentro de la MMU

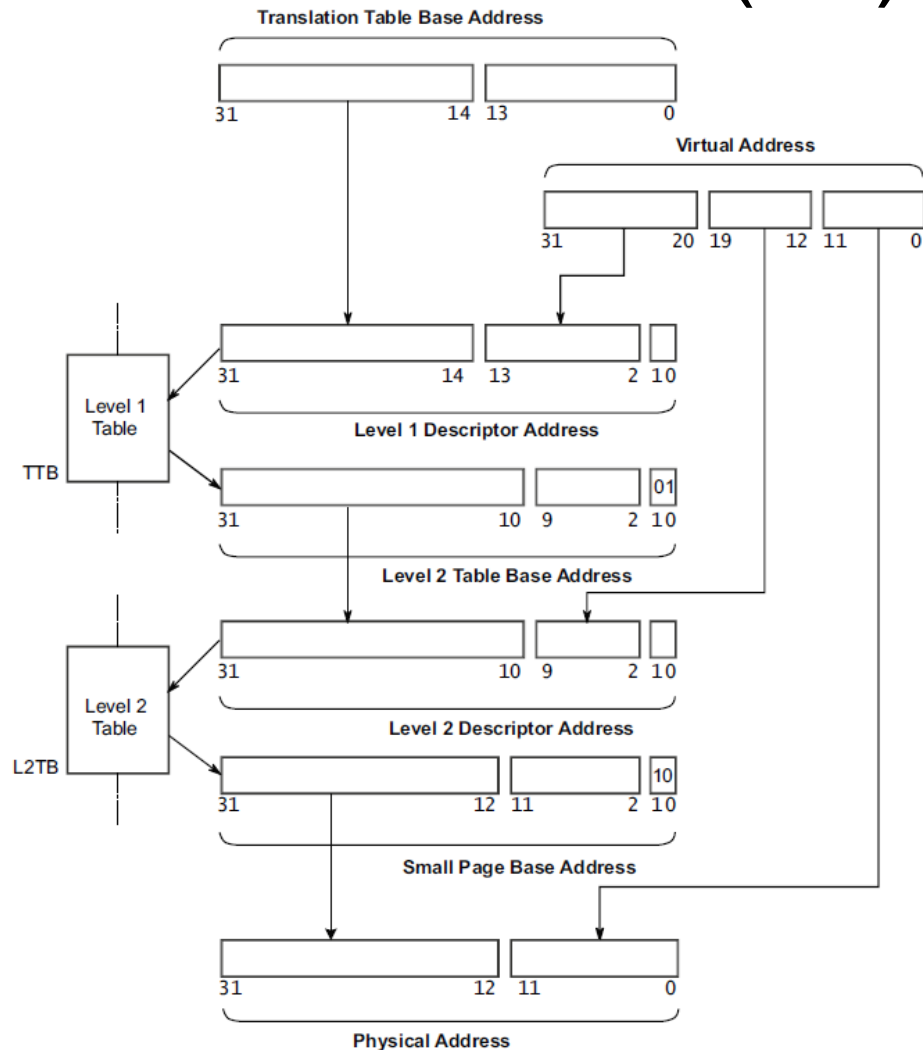
# Primer nivel de traducción (L1)

- El primer nivel de traducción de memoria se llama L1.
- Este nivel divide la memoria en 4096 espacios de 1MB.
- La tabla L1 tiene 4096 entradas de 32 bits.



# Segundo nivel de traducción (L2)

- El segundo nivel es una tabla en la que se ingresa luego de pasar por la tabla L1.
- Usando L1 y L2 se tiene:
  - Descriptores de página de 32 bits.
  - Hasta dos niveles de indirección.
  - Memoria física de 32bits.
  - Secciones 16MB o 1MB
  - Páginas de 64KB o 4KB.



# MMU. Atributos de memoria

APX	AP	Privileged	Unprivileged	Description
0	00	No access	No access	Permission fault
0	01	Read/Write	No access	Privileged Access only
0	10	Read/Write	Read	No user-mode write
0	11	Read/Write	Read/Write	Full access
1	00	-	-	Reserved
1	01	Read	No access	Privileged Read only
1	10	Read	Read	Read only
1	11	-	-	Reserved



# Tabla de Excepciones

Exception	Mode	CPSR interrupt mask
Reset	Supervisor	F = 1 I = 1
UNDEFINED instruction	Undef	I = 1
Supervisor Call	Supervisor	I = 1
Prefetch Abort	Abort	I = 1
Data Abort	Abort	I = 1
Not used	HYP	-
IRQ interrupt	IRQ	I = 1
FIQ interrupt	FIQ	F = 1 I = 1

- La tabla de excepciones se puede ubicar en dos direcciones de memoria:
  - ☐ 0x00000000
  - ☐ 0xFFFF0000
- Todos los Cortex-A permiten usar ambas.
- Los Cortex-A que implementan las extensiones de seguridad define dos vectores de manera separada

# Tabla de Excepciones (2)

Normal Vector offset	High vector address	Non-secure	Secure	Hypervisor <sup>a</sup>	Monitor
0x0	0xFFFF0000	Not used	Reset	Reset	Not used
0x4	0xFFFF0004	UNDEFINED instruction	UNDEFINED instruction	UNDEFINED instruction from Hyp mode.	Not used
0x8	0xFFFF0008	Supervisor Call	Supervisor Call	Secure Monitor Call	Secure Monitor Call
0xC	0xFFFF000C	Prefetch Abort	Prefetch Abort	Prefetch Abort from Hyp mode.	Prefetch Abort
0x10	0xFFFF0010	Data Abort	Data Abort	Data Abort from Hyp mode,	Data Abort
0x14	0xFFFF0014	Not used	Not used	Hyp mode entry	Not used
0x18	0xFFFF0018	IRQ interrupt	IRQ interrupt	IRQ interrupt	IRQ interrupt
0x1C	0xFFFF001C	FIQ interrupt	FIQ interrupt	FIQ interrupt	FIQ interrupt

# Manejo de excepciones

- Cuando ocurre una excepción el procesador:
  1. Copia el CPSR del estado anterior al SPSR del estado nuevo
  2. Actualiza el LR
  3. Cambia al modo de excepción actual (actualiza CPSR)
  4. Carga PC con dirección de la tabla de excepciones correspondiente al estado nuevo
  5. Cambia el banco de registros.
- Al volver de la excepción el handler debe:
  1. Copiar el SPSR del estado viejo al CPSR
  2. Actualizar el PC.

# Manejo de interrupciones

- Cómo la tabla de excepciones es fija y todas las IRQ tienen una única dirección de rutina de atención de interrupción así como todas las FIQ también tienen una única rutina de atención de interrupción los Cortex-A tienen un controlador de interrupciones denominado GIC (Generic Interrupt Controller).
- El GIC puede configurar para cada interrupción:
  - ☐ Prioridad.
  - ☐ Si es sensible por flanco o nivel
  - ☐ A que procesador interrumpe
  - ☐ Habilitarla o deshabilitarla
  - ☐ Si es de modo seguro o no.

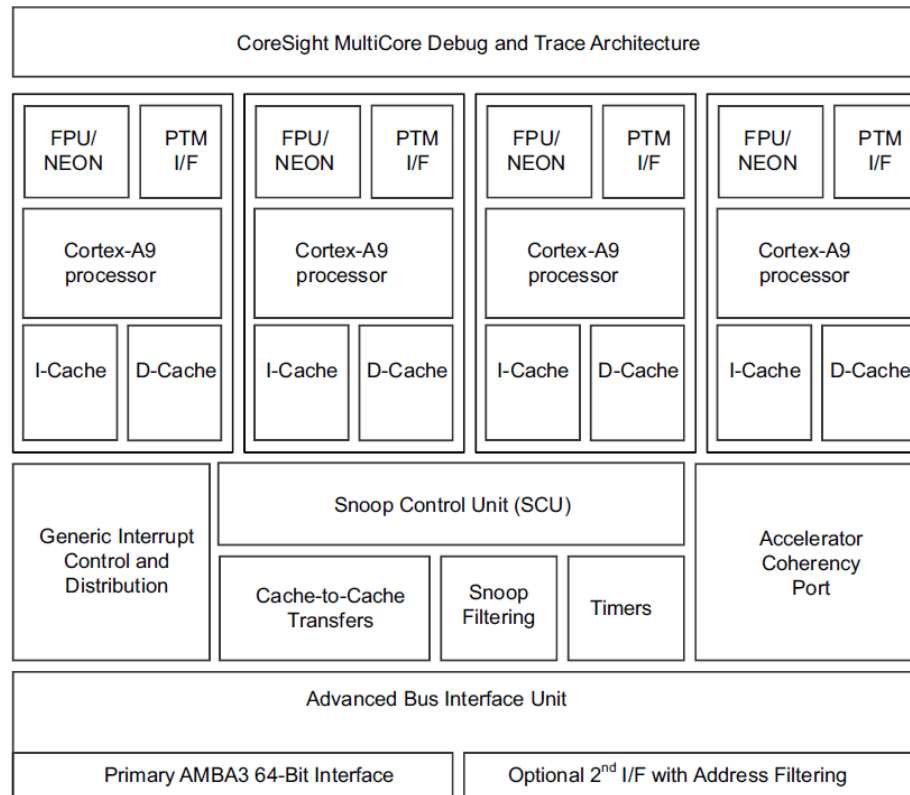
# Inicio Cortex-A

- Cuando se inicia el procesador, en un Cortex-R puede comenzar en la dirección 0x00000000 o bien en la dirección 0xFFFF0000. La rutina de atención de reset debe:
  - ☐ En un sistema multicore, poner a dormir todos los cores menos el que inicializa el sistema.
  - ☐ Inicializar el vector de excepciones.
  - ☐ Inicializa la memoria, incluyendo la MPU
  - ☐ Inicializar los stacks de todos los modos y registros.
  - ☐ Inicializar toda la entrada/salida crítica
  - ☐ Inicializa NEON y VFP
  - ☐ Habilitar las interrupciones
  - ☐ Cambiar el modo de operación
  - ☐ Llamar a la función main()

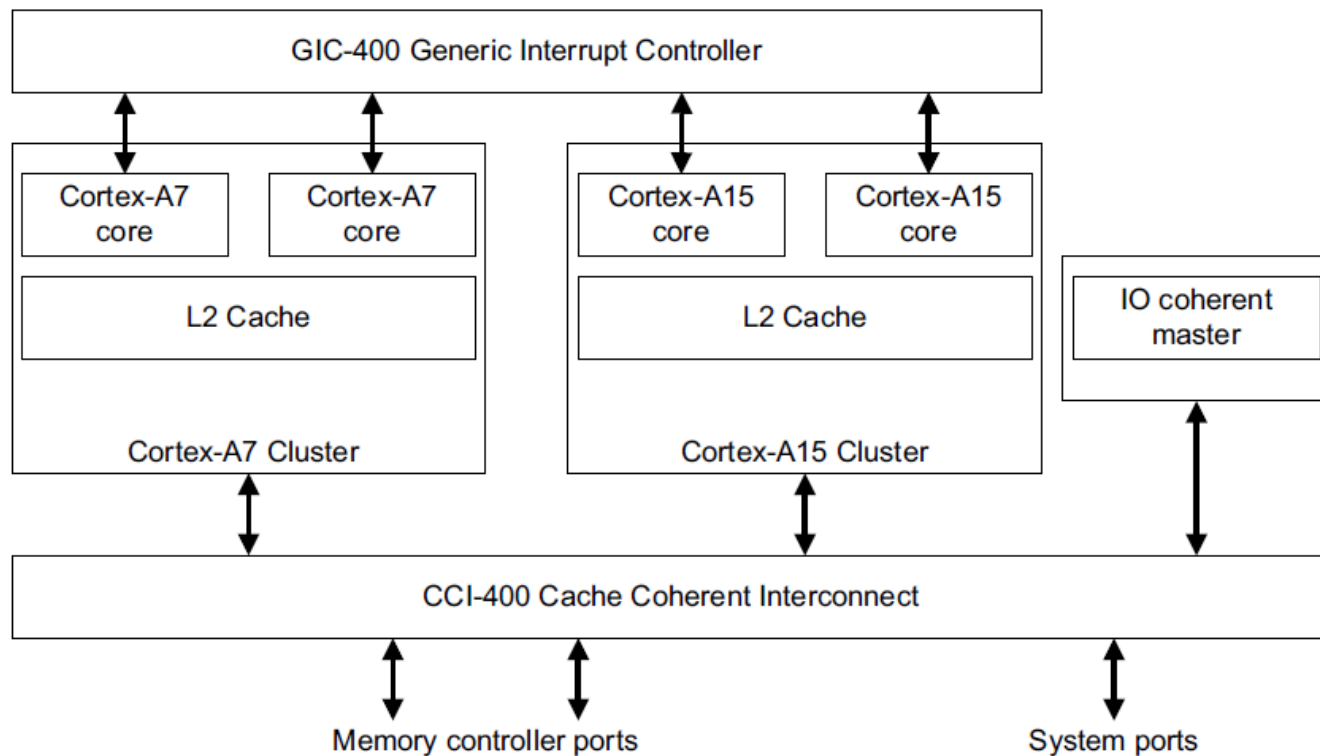
# Multiprocesamiento

- **Multi procesamiento simétrico (SMP – symmetric multi-processing).** Cada procesador del clúster ve la misma memoria física y hardware compartido. Desde el punto de vista de la aplicación un proceso puede correr en cualquiera de los procesadores.
- **Multi procesamiento asimétrico (AMP – asymmetric multi-processing).** Cada procesador del clúster tiene una función específica y no necesariamente “ven” la misma memoria y hardware.
- **Multiprocesamiento heterogéneo (HMP – Heterogeneous multi-processing).** En ARM es una configuración como la de SMP con procesadores con arquitectura diferente.

# Ejemplo. Hardware que soporta SMP (Cortex-A9)

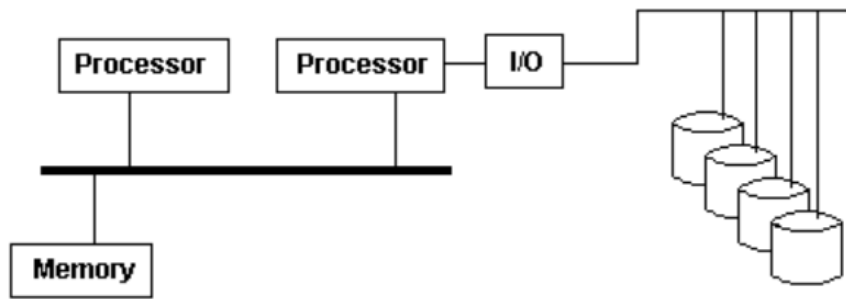


# Ejemplo. Sistema HMP ARM



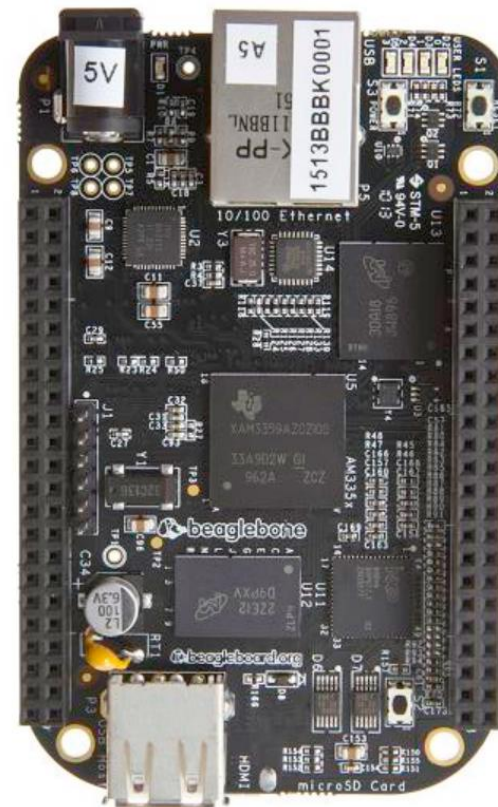


# Ejemplo. AMP



- Los sistemas AMP por lo general eran sistemas con procesadores dedicados a alguna tarea específica.
- Por ejemplo manejo de entradas salidas, hardware de propósito especial (criptografía, tiempo real, etc.)

# Plataformas con Cortex-A



# Bibliografía.

- ARM Cortex-R Series. Programmer's Guide. Version 1.0.  
<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0042a/index.html>
- ARM Cortex-A Series. Programmer's Guide. Version 1.0.  
<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0013d/index.html>
- Hercules Web Page: [www.ti.com/hercules](http://www.ti.com/hercules)
- Hercules Safety Microcontrollers Training Series  
[training.ti.com/Hercules](http://training.ti.com/Hercules)