

## Agenda

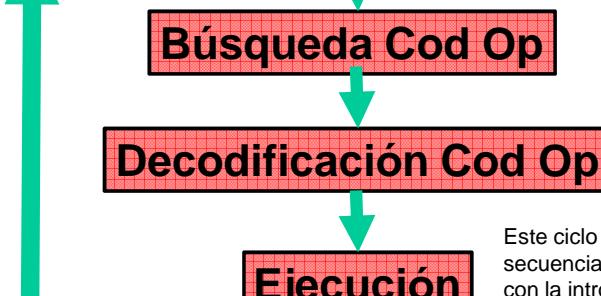
- Introducción
- Presentación de la Familia Cortex M
- Arquitectura de los Cortex M
- Repertorio de Instrucciones y Ejemplos
- Sistema de Memoria
- Excepciones, Interrupciones y el NVIC
- La familia NXP

## Agenda

- **Introducción**
- Presentación de la Familia Cortex M
- Arquitectura de los Cortex M
- Repertorio de Instrucciones y Ejemplos
- Sistema de Memoria
- Excepciones, Interrupciones y el NVIC
- La familia NXP LPC

## Algunos conceptos Fundamentales

Ciclo perpetuo de ejecución



Este ciclo perpetuo y secuencial cambiará con la introducción del pipeline de la familia ARM

## Arquitecturas de las computadoras

- Arquitectura Von Neumann

DIREC.

0000

0001

0002

:

nnnn

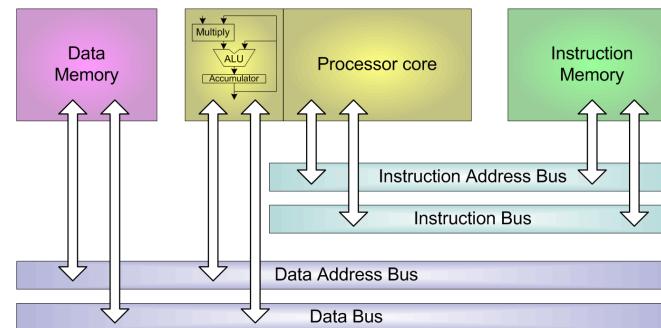
**Memoria de Programa y de Datos indistinta**

BUSES

CPU

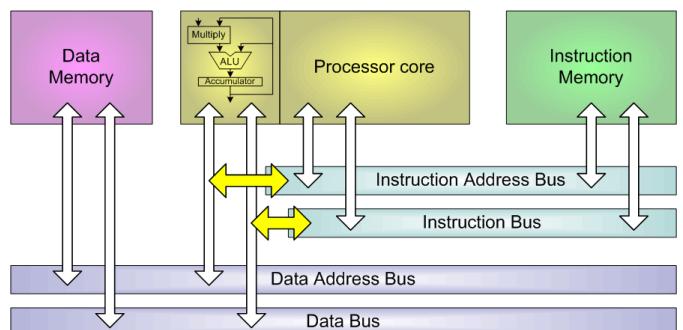
## Arquitecturas de las computadoras

- Arquitectura Harvard



## Arquitecturas de las computadoras

- Arquitectura Harvard Modificada



Introducción y Arquitectura

7

## Una de las formas de clasificar las computadoras

- Complejidad de su repertorio de instrucciones
  - Computadoras CISC
  - Computadoras RISC

Introducción y Arquitectura

8

## Análisis estadístico de Ejecución de programas estándar

Tipo de instrucción	% de Uso
Movimiento de datos	43
Control de flujo (branches)	23
Operaciones Aritméticas	15
Comparaciones	13
Operaciones Lógicas	5
Otras	1

## Concepto RISC

- Se buscó diseñar un procesador que tuviera pocas instrucciones, fundamentalmente las de uso más frecuente y se optimizó su tiempo de ejecución, de manera que la arquitectura resultante fuera muy eficiente en la mayoría de las instrucciones de uso corriente.
- Las instrucciones complejas, no tendrían lugar en el repertorio de instrucciones y deberían ser implementadas por medio de varias instrucciones sencillas.

## Ventajas Arquitectura RISC

1. Mejor aprovechamiento de área de silicio.  
Típicamente 1/3 a ¼ del área requerida por un procesador x86.
2. Menor consumo de energía.
3. Menor tiempo de desarrollo
4. Mayor rendimiento de la energía.  
Típicamente 40 a 50% menor consumo por MHz de reloj.

<http://blogs.arm.com/software-enablement/375-risc-versus-cisc-wars-in-the-prepc-and-pc-eras-part-1/>

<http://blogs.arm.com/software-enablement/377-risc-versus-cisc-wars-in-the-postpc-eras-part-2/>

Introducción y Arquitectura

11

## Inconvenientes Arquitectura RISC

1. No ejecuta código x86
2. Generalmente tienen una pobre densidad de código comparada con CISC
3. Se requerirán múltiples instrucciones RISC para ejecutar una CISC (aunque el tiempo de ejecución del programa completo en RISC sea menor que el tiempo de ejecución en CISC)

Introducción y Arquitectura

12

El repertorio de instrucciones con que nos encontraremos será:

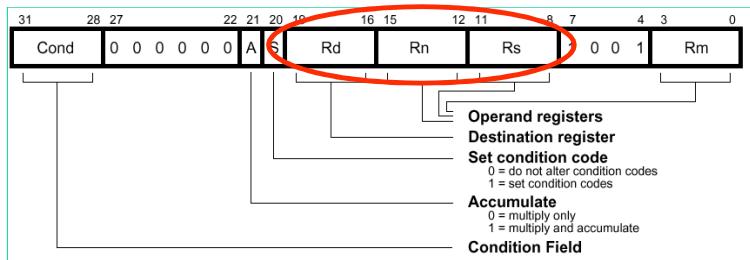
1. Instrucciones de procesamiento de datos
2. Instrucciones de movimientos de datos
3. Instrucciones de control de flujo
4. Instrucciones especiales

Cantidad de direcciones en la palabra de instrucción

Tres Direcciones



## Implementación en un Microcontrolador Cortex



En los microcontroladores de 32 bits se implementa una versión de las máquinas de 3 direcciones en las que las direcciones de los operandos se puede dar por medio de registros de 32 bits que contienen los operandos y que previamente fueron cargados en los mismos

## Suma del contenido de dos posiciones de memoria

Lo que en un CISC sería: Add result, Oper1, Oper2

En RISC puede ser:

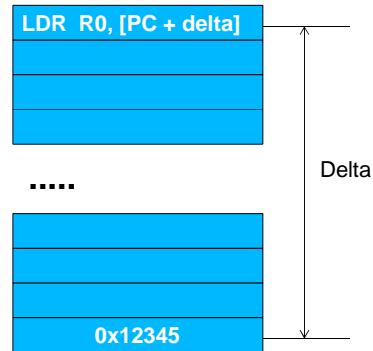
ldr	<i>r0,=Oper1</i>
ldr	<i>r1,[r0]</i>
ldr	<i>r0,=Oper2</i>
ldr	<i>r2,[r0]</i>
add	<i>r3,r2,r1</i>
LDR	<i>r0,=result</i>
str	<i>r3,[r0]</i>

Oper1	DCD	0x12345678
Oper2	DCD	0x23456789
AREA	DATA	
result	space	4

## Carga inmediata de 32 bits en un registro

En realidad el compilador lo traduce a:

Queremos implementar  
**LDR R0,=0x12345**



## Agenda

- Introducción
- **Presentación de la Familia Cortex M**
- Arquitectura de los Cortex M
- Repertorio de Instrucciones y Ejemplos
- Sistema de Memoria
- Excepciones, Interrupciones y el NVIC
- La familia NXP LPC

## ARM Connected Community – 700+



Introducción y Arquitectura

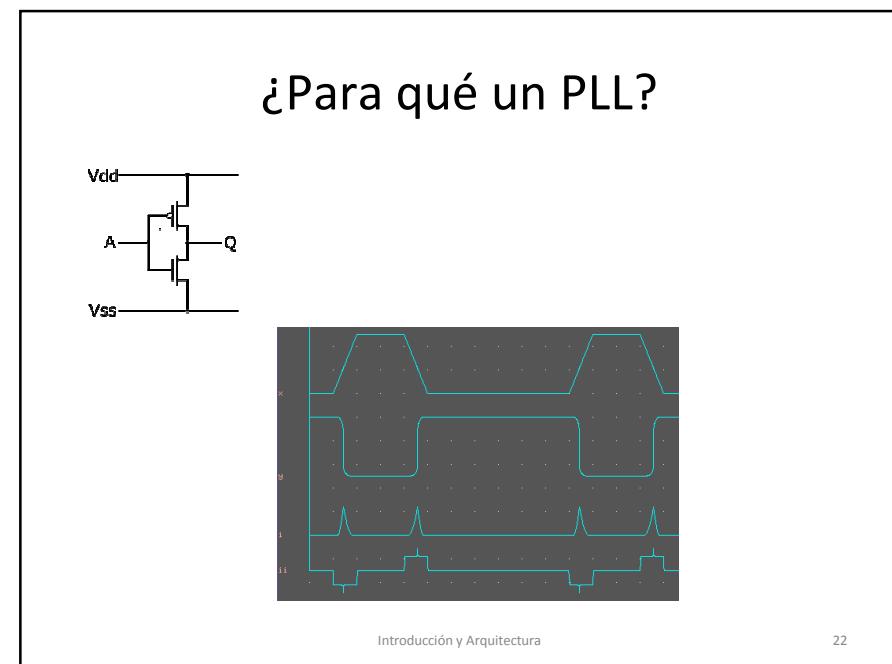
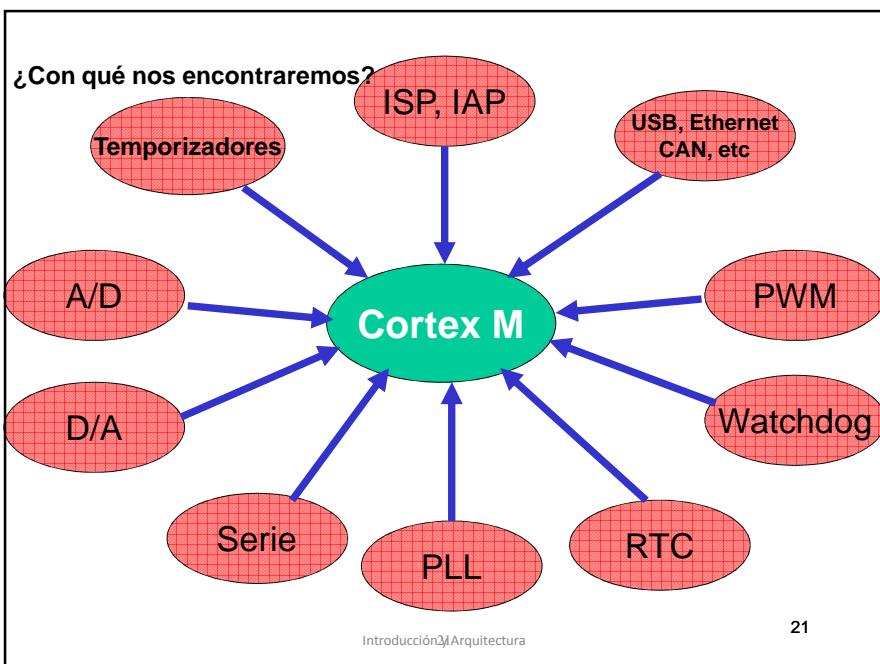
19

## Aplicaciones

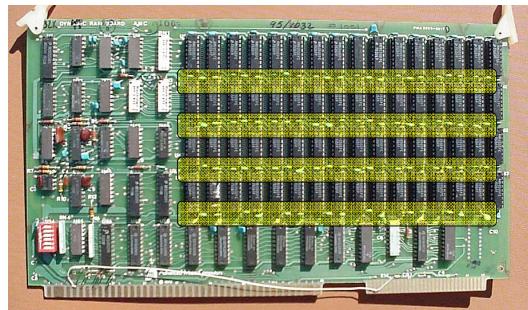


Introducción y Arquitectura

20



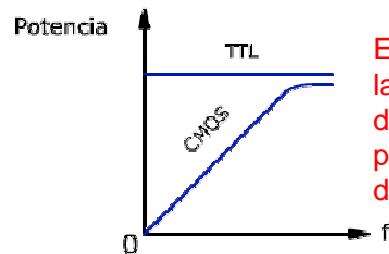
¿Qué efectos producen los picos de corriente y cómo se minimizan?



Introducción y Arquitectura

23

Potencia disipada en función de la frecuencia



El PLL se emplea para disminuir la frecuencia de trabajo en forma dinámica durante la ejecución, por ejemplo a 500 kHz en lugar de 120 MHz

Introducción y Arquitectura

24

## Principios Básicos ARM - Cortex

- Se basa en Arquitectura RISC.
- Optimización del tiempo de ejecución de las instrucciones más frecuentes
- Importante banco de registros
- Arquitectura load-store.
- Pipeline

## Principios Básicos ARM - Cortex

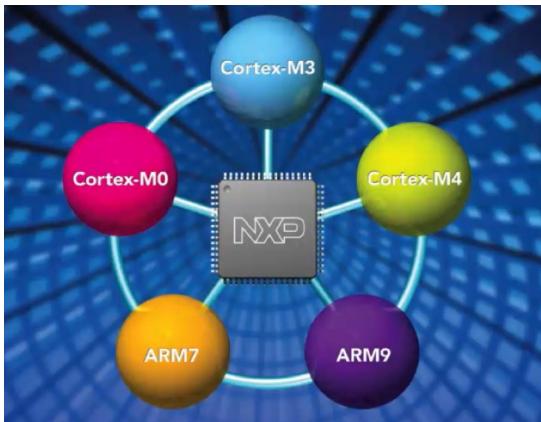
- Instrucciones de tamaño fijo: 16 ó 32-bit
- Ejecución condicional de todas las instrucciones, para maximizar el rendimiento de la ejecución.
- Computadora de 3 direcciones.
- Varios modos de operación.
- Las operaciones aritméticas y lógicas **son entre registros.**

## Características ARM - Cortex

- Todos las familias de procesadores Cortex comparten el mismo repertorio de instrucciones.
- El núcleo del procesador es compartido por todos los fabricantes de silicio. Los periféricos son específicos de cada modelo y suelen NO ser compatibles entre si, por lo cual se han implementado metodologías de homogeneización (CMSIS).
- Existen técnicas de programación que hacen que esas incompatibilidades de hardware sean disimuladas para el programador
- Registros de 32 bits (16 + 1 disponibles). Registros 0 a 7 disponibles en todo momento
- Estructura del bus tipo Von Neuman (ARM7 y Cortex M0), tipo Harvard ( Cortex y ARM9)

## Componentes, Familias y subfamilias

## Portafolio Procesadores NXP



Introducción y Arquitectura

29

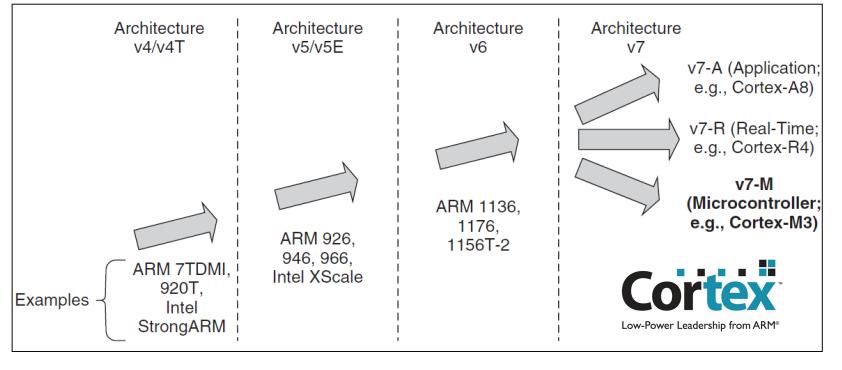
## Evolución

Processor Name	Architecture Version	Memory Management Features	Other Features
ARM7TDMI	ARMv4T		
ARM7TDMI-S	ARMv4T		
ARM7EJ-S	ARMv5E		DSP, Jazelle
ARM920T	ARMv4T	MMU	
ARM922T	ARMv4T	MMU	
ARM926EJ-S	ARMv5E	MMU	DSP, Jazelle
ARM946E-S	ARMv5E	MPU	DSP
ARM966E-S	ARMv5E		DSP
ARM968E-S	ARMv5E		DMA, DSP
ARM966HS	ARMv5E	MPU (optional)	DSP
ARM1020E	ARMv5E	MMU	DSP
ARM1022E	ARMv5E	MMU	DSP
ARM1026EJ-S	ARMv5E	MMU or MPU	DSP, Jazelle
ARM1136J(F)-S	ARMv6	MMU	DSP, Jazelle
ARM1176ZJ(F)-S	ARMv6	MMU + TrustZone	DSP, Jazelle
ARM11 MPCore	ARMv6	MMU + multiprocessor cache support	DSP, Jazelle
ARM1156T2(F)-S	ARMv6	MPU	DSP
Cortex-M3	ARMv7-M	MPU (optional)	NVIC
Cortex-R4	ARMv7-R	MPU	DSP
Cortex-R4F	ARMv7-R	MPU	DSP + Floating point
Cortex-A8	ARMv7-A	MMU + TrustZone	DSP, Jazelle

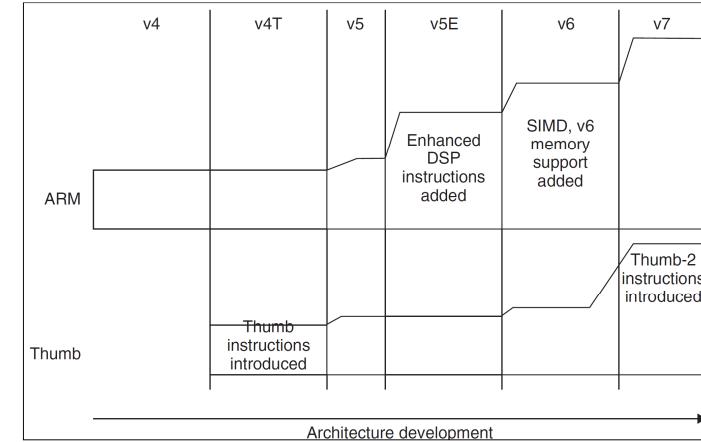
Introducción y Arquitectura

30

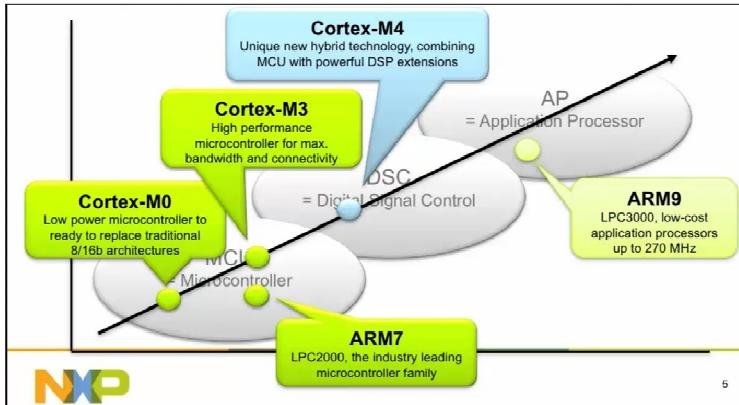
## Evolución



## Evolución



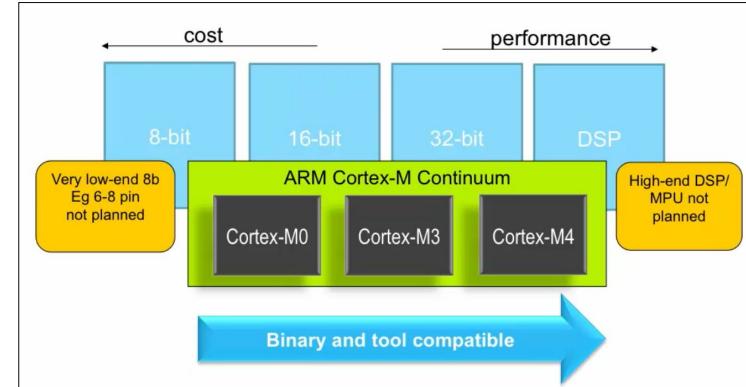
## Evolución



Introducción y Arquitectura

33

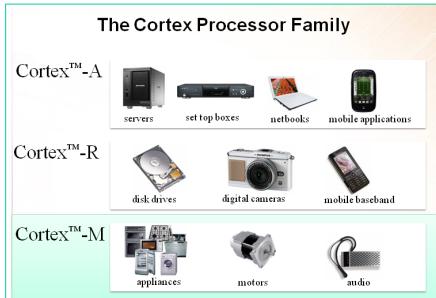
## Costo / Performance



Introducción y Arquitectura

34

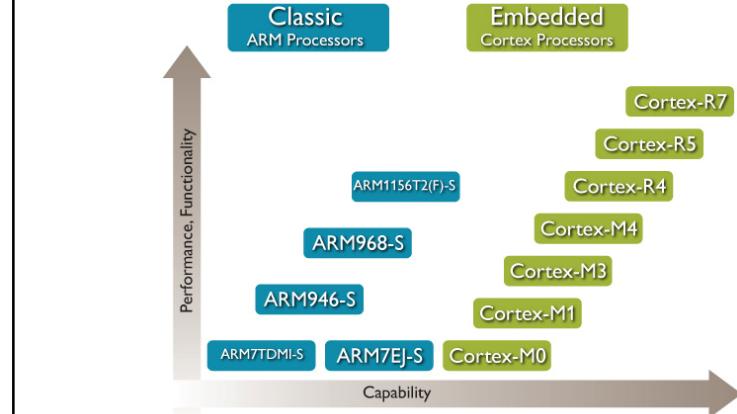
## Aplicaciones de las distintas subfamilias



- ARM Cortex-A, Orientado a Sistemas Operativos complejos y aplicaciones multiusuario.
- ARM Cortex-R, Orientada a sistemas operativos embebidos en tiempo real.
- ARM Cortex-M, Orientada a aplicaciones de bajo costo y FPGAs

<http://www.actel.com/interact/confirmation.aspx?p=E239default.aspx.htm>

## Cortex



## Agenda

- Introducción
- Presentación de la Familia Cortex M
- Arquitectura de los Cortex M**
- Repertorio de Instrucciones y Ejemplos
- Sistema de Memoria
- Excepciones, Interrupciones y el NVIC
- La familia NXP LPC

## Objetivos Cortex M

- Optimizado para uso con flash en un solo ciclo
- Esquema de Interrupciones priorizables basado en hardware
  - Deterministico/Interrupciones con baja latencia
  - Interrupción no enmascarable (NMI)
- Multiplicación en un solo ciclo y división por hardware
- Requerimientos reducidos de memoria.
  - Almacenamiento de datos no-alineado
  - Manipulación de bits
  - Thumb-2

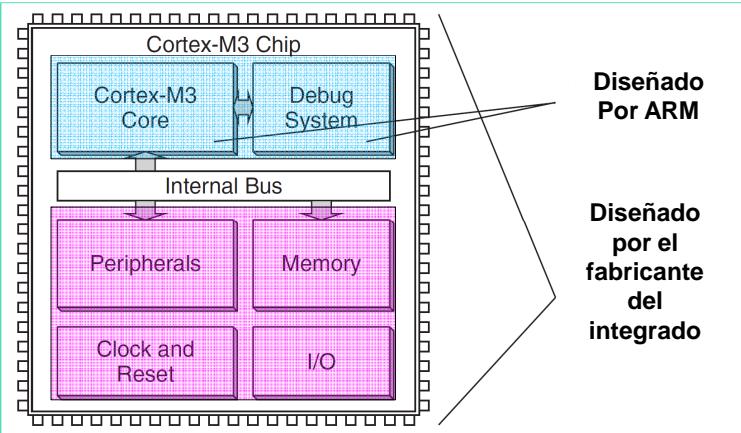
## Objetivos Cortex M

- Manejo de potencia disipada y modos de muy bajo consumo.
- Herramientas de depuración incorporadas en el chip (breakpoints, watchpoints y serial wire viewer).
- Diseño optimizado para la programación en C (aún del reset, interrupciones y excepciones).
- Excelente soporte para la implementación de sistemas operativos.
- Mapa de memoria Fijo
- Bit-banding (operación atómica sobre bits)

## Tamaño de instrucciones y datos

- ARM es una arquitectura de 32-bits.
  - **Byte** significa 8 bits
  - **Halfword** significa 16 bits
  - **Word** significa 32 bits
- El repertorio de instrucciones
  - 32-bit ARM
  - 16-bit Thumb
  - Thumb-2 es una combinación de ambos
- Los que tienen la extensión Jazelle ejecutan código Java

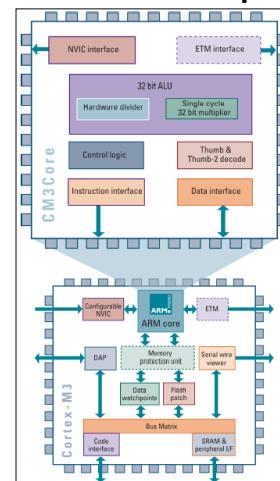
## Cortex M



Introducción y Arquitectura

41

## Arquitectura Cortex M



### Núcleo Cortex-M (core)

Arquitectura Harvard  
Pipeline de tres etapas con especulación de saltos  
Thumb®-2 y Thumb tradicional  
ALU con división por H/W y multiplicación en un ciclo

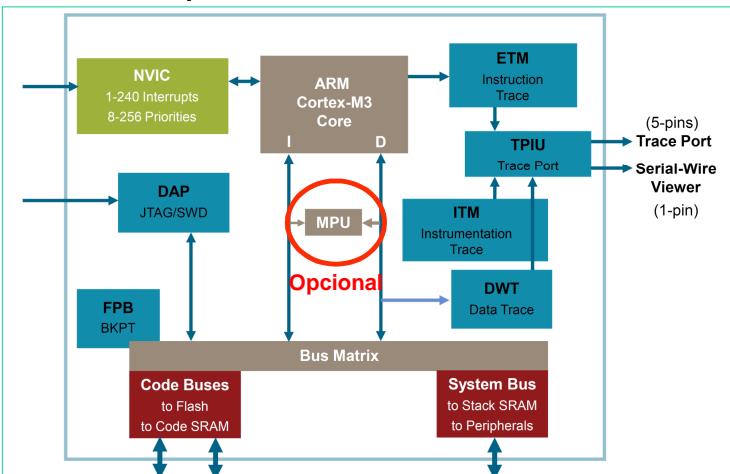
### Procesador Cortex-M

Controlador de interrupciones Configurable  
Bus matrix  
Componentes de depuración avanzados  
Opcionales: MPU & ETM

Introducción y Arquitectura

42

## Arquitectura Cortex M

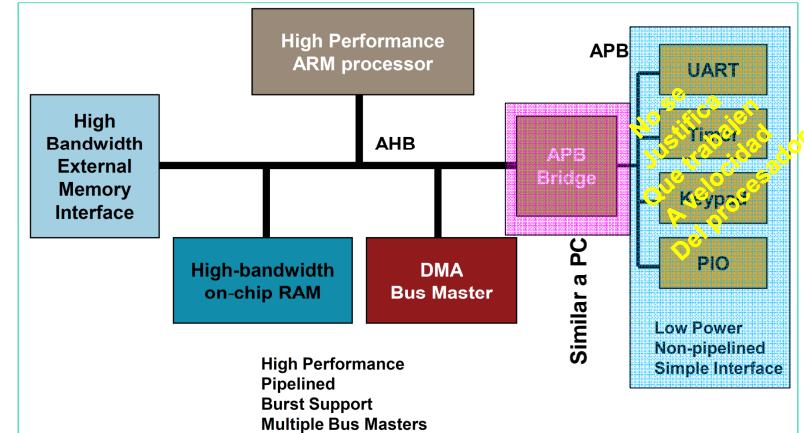


Introducción y Arquitectura

43

## Un ejemplo: AMBA

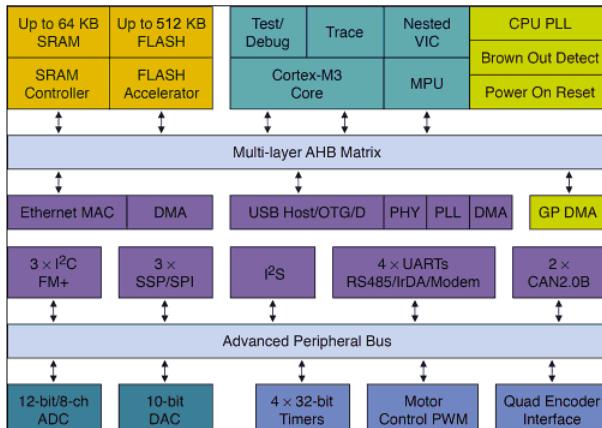
(Advanced Microcontroller Bus Architecture)



Introducción y Arquitectura

44

## Ejemplo de Implementación Arquitectura NXP LPC



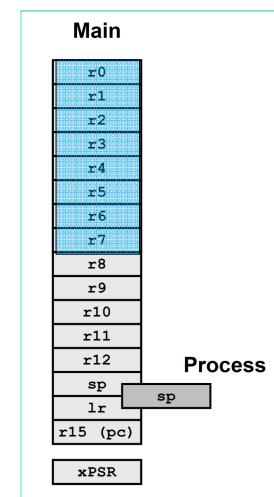
Introducción y Arquitectura

45

## Registros de Cortex M

Modelo amigable para compiladores

- Arquitectura Load/Store
- Registros de 32-bit
- Esquema Flexible de registros
- Espacio de direccionamiento Linear de 32-bits



Introducción y Arquitectura

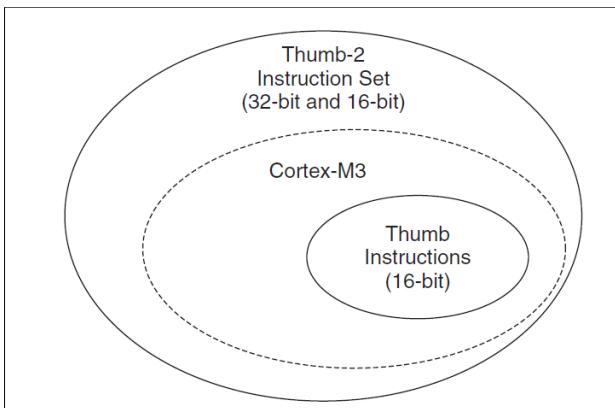
47

## Registros específicos

- R15 = PC
- R14 = LR = Link register. Almacena la dirección de retorno en subrutinas y excepciones
- R13 = SP (con doble significación)
- Program Status Registers (PSRs)
- Interrupt Mask Registers (PRIMASK, FAULTMASK, BASEPRI)
- Control Register (CONTROL)

## Modos ARM, Thumb y Thumb-2

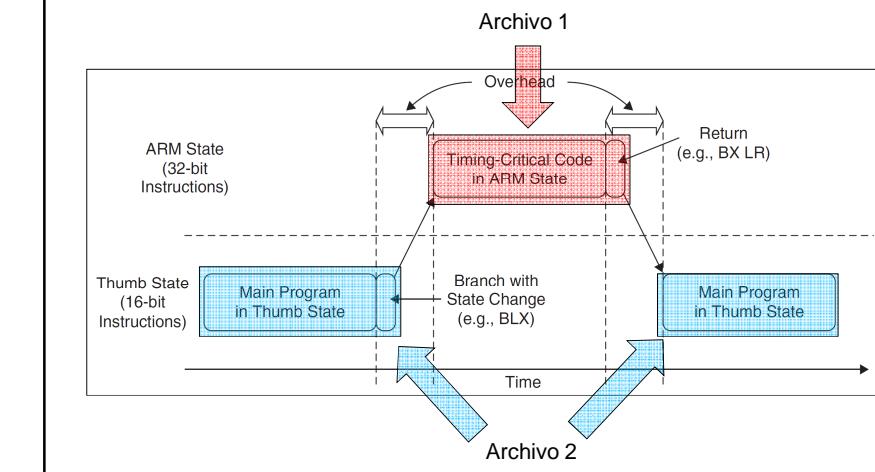
## Thumb-2 y Thumb



Introducción y Arquitectura

50

## Cambio de modos en ARM



Introducción y Arquitectura

51

## Arm 32 y Thumb 2 en Cortex

- No hay overhead de comutación de estado, ahorrando tiempo de ejecución y espacio de instrucción
- No hay necesidad de separar el código ARM del Thumb y se puede hacer todo el programa en un solo archivo, lo que hace el desarrollo y mantenimiento del firmware más fácil
- Es más fácil obtener la mejor eficiencia y rendimiento, a su vez, por lo que es más fácil escribir software, porque no hay necesidad de preocuparse por el cambio de código entre el ARM y Thumb para tratar de obtener la mejor densidad / rendimiento

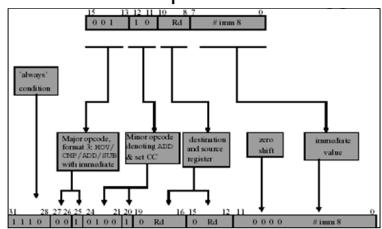
## Thumb-2

- Permite mezclar instrucciones de 16 bits de longitud con instrucciones de 32 bits.
- La mayor parte de las instrucciones de los Cortex-M son de 16 bits.
- Internamente se procesan como instrucciones de 32 bits, por lo que existe una etapa de decompresión de las instrucciones que las lleva de 16 a 32 bits.

## Thumb-2

Debemos destacar dos temas muy importantes:

1. A pesar de que la mayoría de las instrucciones de un Cortex M son de 16 bits, internamente se procesan como 32 bits “descomprimiéndose”

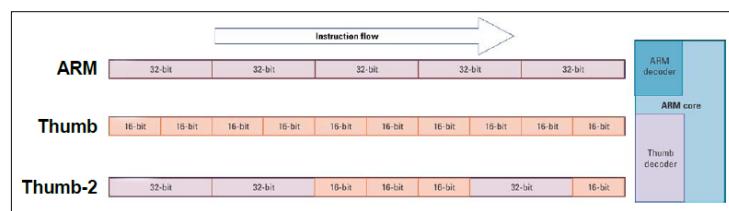


2. Aunque las instrucciones sean de 16 operan sobre registros de 32 bits (salvo las instrucciones específicas que trabajan sobre 16 u 8 bits)

## Instrucciones en Thumb-2

- En ARM Unified Assembler Language Muchas instrucciones se representan nemónicamente de igual manera para 32 ó 16 bits.
- Si no indicamos lo contrario, y de ser posible los compiladores habitualmente eligen la opción de 16 bits.
- Ej: ADDS R0, #1 será implementada en 16 bits
- Si deseamos se más explícitos (strong Typing)
- ADDS.W R0,#1 ; Wide = 32 bits
- ADDS.N R0,#1 ; Narrow = 16 bits

## Thumb-2

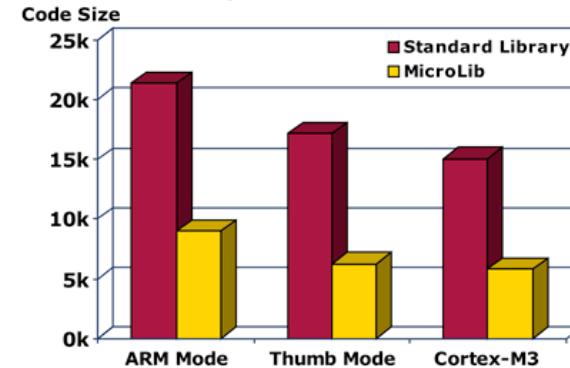


Introducción y Arquitectura

56

## Comparación

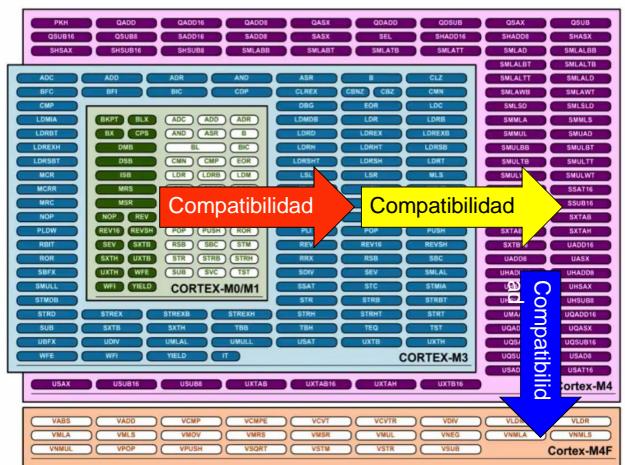
Dhrystone 2.1 Benchmark



Introducción y Arquitectura

57

## Compatibilidad entre subfamilias



Introducción y Arquitectura

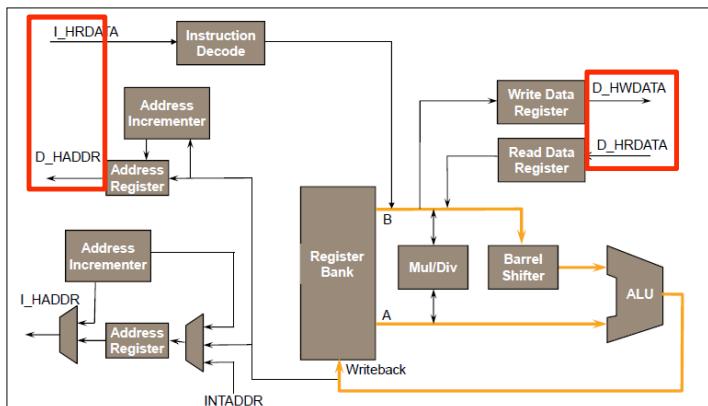
58

## Caminos de Datos y pipeline

Introducción y Arquitectura

59

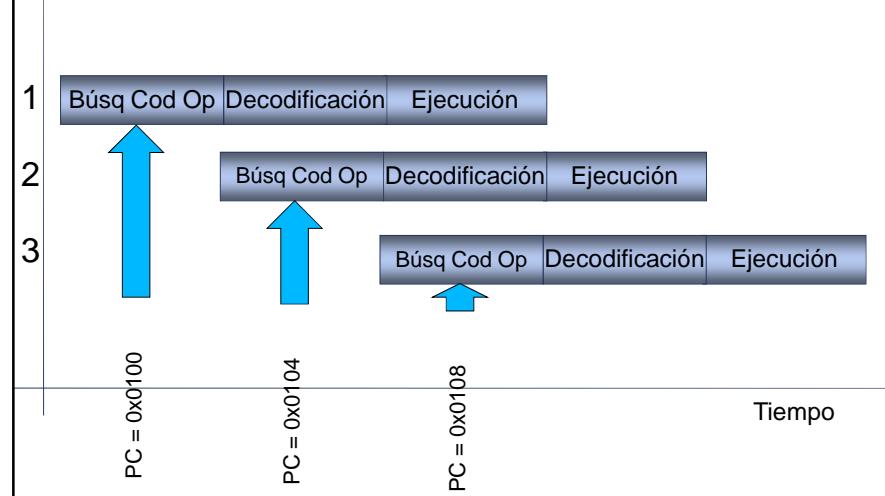
## Caminos de datos



Introducción y Arquitectura

60

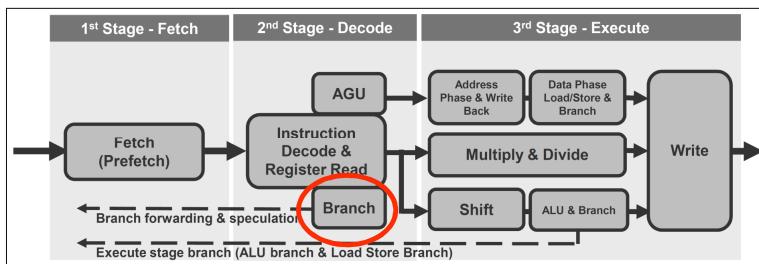
## Pipeline de 3 etapas



Introducción y Arquitectura

61

## Pipeline en Cortex. Funciones ampliadas



- Pipeline de 3-etapas + especulación de saltos
  - Cuando se encuentra una instrucción de salto, la etapa de decodificación también incluye una búsqueda de código de operación especulativa del salto

## Utilización Óptima del Pipeline

6 ciclos = 6 Instrucciones									
Cycle	1	2	3	4	5	6	7	8	9
Operation									
ADD	F	D	E						
SUB	F	D	E						
ORR	F	D	E						
AND		F	D	E					
ORR		F	D	E					
EOR		F	D	E					

F - Fetch   D - Decode   E - Execute

## Caso más desfavorable

Address	Operation	1	2	3	4	5	6	7	8	9
0x8000	BX r5	F	D	E						
0x8002	SUB		F	D						
0x8004	ORR		F							
0x8FEC	AND		F	D	E					
0x8FEE	ORR		F	D	E					
0x8FF0	EOR		F	D	E					
Ejecutar el Branch tarda 3 ciclos.										
F - Fetch    D - Decode    E - Execute										

## LDR

Debe completarse la instrucción LDR antes de poder ejecutar la sig. instrucción

Operation	1	2	3	4	5	6	7	8	9
ADD	F	D	E						
SUB		F	D	E					
LDR		F	D	Ea	Ed				
AND		F	D	S	E				
ORR		F	S	D	E				
EOR		F	D	E					
F - Fetch    D - Decode    E - Execute    S - Stall									
Ea - LDR address phase    Ed - LDR data phase									

That's All Folks

