Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería Mecánica Eléctrica

ANÁLISIS ARQUITECTÓNICO DE PERFILES CORTEX-M Y CORTEX-A EN PROCESADORES ARM Y DISEÑO DE GUÍA INTRODUCTORIA EN SU PROGRAMACIÓN DE BAJO NIVEL CON LENGUAJE ENSAMBLADOR

Marie Chantelle Cruz Medina
Asesorado por el MSc. Ing. Iván René Morales Argueta

Guatemala, abril de 2019

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ANÁLISIS ARQUITECTÓNICO DE PERFILES CORTEX-M Y CORTEX-A EN PROCESADORES ARM Y DISEÑO DE GUÍA INTRODUCTORIA EN SU PROGRAMACIÓN DE BAJO NIVEL CON LENGUAJE ENSAMBLADOR

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

MARIE CHANTELLE CRUZ MEDINA
ASESORADO POR EL MSC. ING. IVÁN RENÉ MORALES ARGUETA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERA EN ELECTRÓNICA

GUATEMALA, ABRIL DE 2019

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. José Francisco Gómez Rivera
VOCAL II	Ing. Mario Renato Escobedo Martínez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Luis Diego Aguilar Ralón
VOCAL V	Br. Christian Daniel Estrada Santizo
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro AntonioAguilar Polanco
EXAMINADOR	Ing. Armando Alonso Rivera Carrillo
EXAMINADOR	Ing. Guillermo Antonio Puente Romero
EXAMINADORA	Inga. Ingrid Salomé Rodríguez de Loukota
SECRETARIA	Inga. Lesbia Magalí Herrera López

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

ANÁLISIS ARQUITECTÓNICO DE PERFILES CORTEX-M Y CORTEX-A EN PROCESADORES ARM Y DISEÑO DE GUÍA INTRODUCTORIA EN SU PROGRAMACIÓN DE BAJO NIVEL CON LENGUAJE ENSAMBLADOR

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería Mecánica Eléctrica, con fecha 16 de febrero de 2018.

Marie Chantelle Cruz Medina

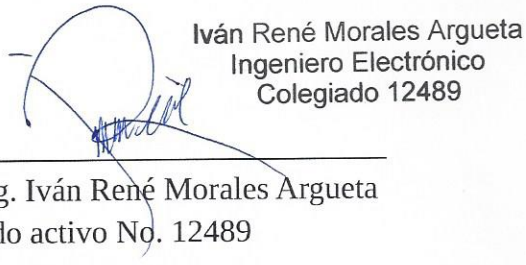
Guatemala, 30 de enero de 2019

Ingeniero
Julio Solares Peñate
Coordinador de Área de Electrónica
Facultad de Ingeniería
Universidad de San Carlos de Guatemala

Señor Coordinador:

Por este medio tengo el gusto de informarle que he concluido con el asesoramiento y revisión del trabajo de graduación con título: **Análisis arquitectónico de perfiles Cortex-M y Cortex-A en procesadores Arm y diseño de guía introductoria en su programación de bajo nivel con lenguaje ensamblador,** desarrollado por la estudiante Marie Chantelle Cruz Medina con carné 201318618. Después de revisar su contenido final doy mi entera aprobación al mismo.

Atentamente,



MSc. Ing. Iván René Morales Argueta
Colegiado activo No. 12489

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

Guatemala, 7 de febrero de 2019

Señor Director
Ing. Otto Fernando Andrino González
Escuela de Ingeniería Mecánica Eléctrica
Facultad de Ingeniería, USAC.

Señor Director:

Por este medio me permito dar aprobación al Trabajo de Graduación titulado **ANÁLISIS ARQUITECTÓNICO DE PERFILES CORTEX-M Y CORTEX-A EN PROCESADORES ARM Y DISEÑO DE GUÍA INTRODUCTORIA EN SU PROGRAMACIÓN DE BAJO NIVEL CON LENGUAJE ENSAMBLADOR,** desarrollado por la estudiante **Marie Chantelle Cruz Medina**, ya que considero que cumple con los requisitos establecidos.

Sin otro particular, aprovecho la oportunidad para saludarlo.

Atentamente,

ID Y ENSEÑAD A TODOS

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

Ing. Julio César Solares Peñate
Coordinador de Electrónica

JULIO CESAR SOLARES P.
INGENIERO MECANICO ELECTRICISTA
COLEGIADO No. 2330

NIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERIA

REF. EIME 09. 2019.

El Director de la Escuela de Ingeniería Mecánica Eléctrica, después de conocer el dictamen del Asesor, con el Visto bueno del Coordinador de Área, al trabajo de Graduación de la estudiante: MARIE CHANTELLE CRUZ MEDINA Titulado: ANÁLISIS ARQUITECTÓNICO DE PERFILES CORTEX-M Y CORTEX-A EN PROCESADORES ARM Y DISEÑO DE GUÍA INTRODUCTORIA EN SU PROGRAMACIÓN DE BAJO NIVEL CON LENGUAJE ENSAMBLADOR procede a la autorización del mismo.

Ing. Otto Fernando Andrino González

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION ESCUELA DE INGENIERIA MECANICA ELECTRICA
FACULTAD DE INGENIERIA

GUATEMALA, 4 DE MARZO 2019.

Universidad de San Carlos
De Guatemala

Facultad de Ingeniería
Decanato

Ref. DTG.186-2019

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería Mecánica Eléctrica del trabajo de graduación titulado: **"ANÁLISIS ARQUITECTÓNICO DE PERFILES CORTEX-M Y CORTEX-A EN PROCESADORES ARM Y DISEÑO DE GUÍA INTRODUCTORIA EN SU PROGRAMACIÓN DE BAJO NIVEL CON LENGUAJE ENSAMBLADOR",** presentado por la estudiante: **Marie Chantelle Cruz Medina,** después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, se autoriza la impresión del mismo.

IMPRÍMASE.

Ing. Pedro Antonio Aguilar Polanco
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERÍA

Guatemala, Abril de 2019

/echm

ACTO QUE DEDICO A:

Dios	Por permitirme llegar hasta este día rodeada de las personas importantes y con ilusión por el futuro.
Mis padres	Lilian Medina y Gustavo Cruz, por su amor, aliento y apoyo incondicional.
Mi hermana	Dulce Cruz, por creer en mí y acompañarme en los tiempos buenos y malos.
Mis padrinos	Iván Morales, David Barrientos y Luis Guillermo García, por su ejemplo inigualable, su cariño, su moral admirable y su guía irremplazable a lo largo de mi desarrollo profesional. Espero que un día sepan que son luz en la vida de muchos.
Mis amigos	Por siempre alentarme, cuidarme y haber estado en los días cansados, en los locos, en los tranquilos y en los divertidos.
Mis compañeros	Porque todo mi esfuerzo en este trabajo fue con la esperanza de dar a alguien el impulso de adentrarse en las áreas del conocimiento que realmente ama.

AGRADECIMIENTOS A:

Mi familia	Por su cariño y apoyo que perduran.
Facultad de Ingeniería	Por hacer posibles mis estudios universitarios en esta carrera y darnos a todos la oportunidad de cumplir nuestras metas.
Escuela de Ciencias Físicas y Matemáticas	Estudiantes y profesores, por recibirme siempre con los brazos abiertos y compartir conmigo su amor por el conocimiento.
IEEE	Por darme los mejores años de mi carrera universitaria, porque encontré amigos y personas que dan siempre todo de sí sin pedir algo a cambio.
ICTP	Por darme la oportunidad de una experiencia inolvidable de la que volví inspirada.
Mis auxiliares	Porque con su paciencia y conocimiento nos enseñaron a perseverar.
Inga. Ingrid de Loukota	Por creer en mí desde el inicio, por su ejemplo, por la oportunidad de trabajar juntas y por su forma de enseñar con amor.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
C	Capacitancia
f	Frecuencia
GB	Gigabytes
GHz	Gigahertz
Hz	Hertz
=	Igualdad
KB	Kilobytes
MB	Megabytes
MHz	Megahertz
mm^2	Milímetro cuadrados
*****	Multiplicación
E+n	Notación científica de base 10 y potencia n
0x, &, 0f_	Número hexadecimal
P	Potencia
-	Resta
s	Segundos
+	Suma
V	Voltaje

GLOSARIO

ADAS	Sistemas avanzados de asistencia al conductor.
AMBA	Arquitectura avanzada de buses desarrollada por Arm para microcontroladores y otros sistemas en un chip.
Barrel shifter	Circuito digital que desplaza un número específico de bits en un grupo sin necesidad de lógica secuencial.
Breakpoint	Pausa intencional en la ejecución de un programa para depuración.
CAN	Protocolo de control en red para múltiples procesadores.
Chromebooks	Ordenador personal que ejecuta el sistema operativo Google Chrome OS.
Cortex-R	Perfil Arm dedicado a procesadores para sistemas con requerimientos de tiempo real.
DAC	Conversor analógico a digital.
Determinismo	Comportamiento de un sistema en que las ejecuciones no son dictadas por el azar.

DMA	Componentes de acceso directo a memoria.
Domótica	Automatización de viviendas en tareas como seguridad, control de consumo de energía, comunicaciones y comodidad.
DP	Precisión doble.
Driver	Controlador específico para un dispositivo.
DSP	Procesamiento digital de señales. Se usa para englobar a las instrucciones necesarias para llevar a cabo esta tarea.
Dual-issue	Control de dos instrucciones a nivel de pipeline por ciclo de reloj. Esto puede escalar a más instrucciones según la capacidad del procesador.
EEPROM	ROM con capacidad de borrado por señal eléctrica.
Ejecución fuera de orden	Técnica para procesadores de alto rendimiento para usar ciclos de reloj al máximo, procurando no desperdiciar tiempo en espera.
FPU	Unidad de punto flotante.
GPIO	Pines de entrada y salida de propósito general.
GSM	Sistema global para comunicaciones móviles.

HDD	Disco duro.
I2C	Especificación de bus para uso entre circuitos integrados.
IEEE	Instituto de ingenieros eléctricos y electrónicos.
Infotainment	Sistemas de audio para automóviles con opciones de entretenimiento e información.
IoT	Internet of Things. Internet de las cosas, sostiene la posibilidad de un sistema de dispositivos todos conectados entre sí a través de la red.
IP	Propiedad intelectual.
Isofrecuencia	Frecuencia única en un rango.
Jazelle	Extensión que premite a arquitecturas Arm ejecutar objetos de Java.
Jitter	Variabilidad en el envío de señales digitales.
JTAG	Estándar para verificación de diseños y prueba de placas de circuito impreso. Definido por el estándar IEEE 1149.1.
Kernel	Núcleo del Sistema operativo.

LIFO	Memoria *Last In First Out*. El ultimo en entrar es el primero en salir.
LCD	Pantalla de cristal líquido.
MAC	Instrucciones de multiplicación y acumulación.
Machine learning	Estudio del uso de sistemas computacionales para ejecución de tareas con base en modelos e inferencias en lugar de instrucciones explícitas.
MPE	Motor de procesamiento de media.
Nanorobótica	Estudio de las tecnologías orientadas a principios de robótica en la escala de los nanómetros.
PWM	Modulación por ancho de pulso.
RAS	Especificación de confiabilidad, disponibilidad y servicio de Arm.
RTOS	Sistema de operación en tiempo real.
SCB	Bloque de control de sistema.
SCU	Unidad de control del sistema.
SDD	Disco de estado sólido.

SIMD	Modo de operación de una entrada y múltiples salidas.
Sistema operativo	Conjunto de declaraciones para el control de un procesador, sobre las que se ejecutan procesos más pequeños.
SP	Precisión simple.
SPI	Interfaz periférica serial.
SRAM	RAM estática.
Superescalar	Descripción de un procesador que es capaz de ejecutar más de una instrucción por ciclo de reloj.
TLB	Buffer de traducción.
UART	*Universal Asynchronous Receiver-Transmitter.*
USB	Bus serial universal.
Watchdog timer	Temporizador utilizado durante la recuperación del procesador desde errores.
Watchpoint	Mecanismo de depuración en que la ejecución se detiene cuando un espacio de memoria determinado se modifica.

WFE	*Wait For Exception.*
WFI	*Wait For Interruption.*
WIC	Controlador de interrupciones para activación de procesador.

RESUMEN

La investigación se llevó a cabo por capítulos:

El primer capítulo se desarrolla como una síntesis breve de los conocimientos fundamentales a considerar, antes de adentrarse en el análisis extensivo.

El segundo capítulo comprende una continuación de conocimientos pero, en este caso, tomando en cuenta conceptos mucho más específicos en el campo de ARM y las características de sus dispositivos.

El tercer y cuarto capítulo implicarán un trabajo de análisis y descripción mucho más complejo, debido a que se enfocarán en detallar cada elemento de las arquitecturas de perfil Cortex-M y Cortex-A, que las distinguen y las hacen óptimas para los objetivos que fueron diseñadas.

El quinto capítulo será una recopilación de conceptos acerca de programación en lenguaje ensamblador, para sentar una base sólida en comprensión de su funcionamiento y los requerimientos en un procesador para su ejecución.

El sexto y séptimo capítulo serán el punto de convergencia de la investigación. En ellos se hará evidente la importancia en la comprensión de cada perfil y las razones por las que fueron creados, dejando una guía adecuada y su programación utilizando como ejemplo dos chips muy utilizados en la ingeniería y computación.

OBJETIVOS

General

Desarrollar un estudio analítico de las características propias de los perfiles Cortex-M y Cortex-A de ARM y las aplicaciones específicas para las que están optimizados como una herramienta de apoyo a la comunidad estudiantil de pregrado de habla hispana.

Específicos

1. Identificar las características propias de los perfiles M y A de la familia Cortex en ARM y las ventajas que implican sus distinciones.

2. Introducir un estudio general de los elementos de lenguaje ensamblador comunes entre dispositivos ARM para su posterior aplicación.

3. Diseñar una guía básica de programación de dispositivos Cortex-M y Cortex-A con lenguaje de bajo nivel como aplicación directa del estudio de cada perfil.

INTRODUCCIÓN

El estudio de arquitecturas de computadoras es, sin duda un campo amplio y en constante cambio. Las motivaciones de un profesional o estudiante en el área para enfocarse en una rama específica pueden encerrar los gustos personales, facilidad de comprensión y (lo más importante), evaluación del impacto de la tecnología en el mundo actual.

Arm ha sido durante años el cerebro detrás del diseño de casi cualquier dispositivo móvil que se utilice actualmente tanto en el área científica como en la vida cotidiana. La forma en que el negocio se modeló desde el inicio de la empresa la llevó a obtener un lugar importante en los dispositivos que se consumen hoy.

Como forma de dar una oferta específica para diversos requerimientos, Arm concentró grupos de procesadores según su fin: así nació Cortex con sus perfiles M (microcontroladores), A (aplicaciones) y R (tiempo real). Para quien interesa especialmente el área de procesadores y cómo estos pueden usarse para tantas aplicaciones en todas las ramas de la ciencia, el estudio de estas arquitecturas supone una inversión para en el futuro conocer cómo se comportan las nuevas tecnologías.

El lenguaje de ensamblador, se presenta como la forma primaria de acercamiento por su relación estrecha a la implementación de hardware en los procesadores. Al terminar este análisis el programador será capaz de construir su propio conocimiento y explorar nuevas ramas que le provoquen interés.

1. CONCEPTOS FUNDAMENTALES

Cuando se estudia un tramo tan específico de la ingeniería de computación como es la arquitectura de procesadores y sus aplicaciones, es importante considerar que existen conocimientos básicos de electrónica que deben manejarse con convicción previamente. Todo esto para asegurar la construcción correcta del conocimiento más complejo.

En este caso se comenzará, entonces, definiendo elementos sustanciales de la electrónica para reforzar su significado y evitar ambigüedades.

1.1. Señal

Comprendida como una función que involucra diversos parámetros y a su vez, depende de por lo menos uno de ellos (usualmente el tiempo). Representa variables físicas y es típico que contenga información sobre el desenvolvimiento de estas.

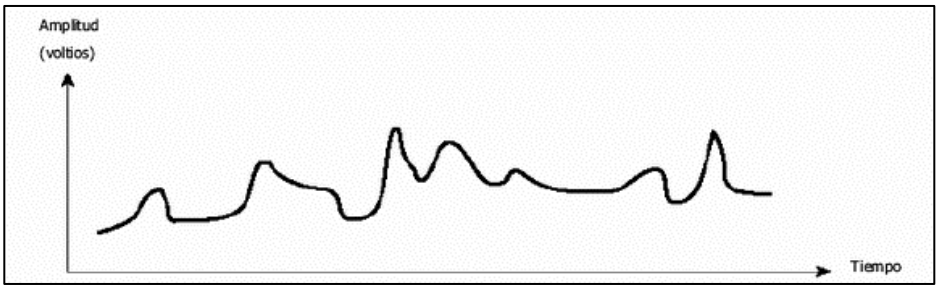
Ejemplos de una señal incluyen los pulsos eléctricos provocados por la transformación del sonido a través de un micrófono o la variación de los fragmentos de una imagen monocromática provocada por la interpretación de matices y brillos.

Aunque existen muchas formas de clasificar las señales, interesa la más usual en el campo de la electrónica.

1.1.1. Clasificación por cantidad de valores que toma la amplitud

Señal analógica: señal cuya amplitud toma todos los valores dentro de su intervalo definido. Considerar, por ejemplo, la figura mostrada a continuación:

Figura 1. **Señal analógica de voltaje dependiente del tiempo**



Fuente: *Transmisores y válvulas inteligentes en la actualidad*. http://www.instrumentacionycontrol.net/cursos-libres/instrumentacion/curso-completo-instrumentacion-industrial/item/232-transmisores-y-v%C3%A1lvulas-inteligentes-en-la-actualidad.html. Consulta: 22 de febrero de 2018.

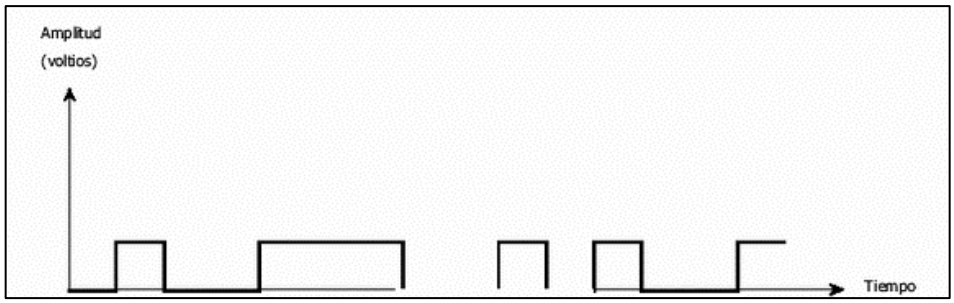
En este caso no es tan importante observar la forma de la onda en su variación con el tiempo, sino los valores que toma en cada instante. Si se trazara una línea perpendicular al eje vertical en un punto aleatorio se hallarán puntos donde también la señal se encuentra definida y esto, lejos de suceder una vez, se repite si se trazaran infinita cantidad de líneas.

Las señales analógicas son consideradas un recurso natural, debido a que se encuentran en el entorno sin necesidad de conversiones.

Algunas señales de este tipo incluyen a las eléctricas, hidráulicas y térmicas.

- Señal digital: señal cuya amplitud sólo puede tomar valores discretos en su rango. Un ejemplo de este tipo se encuentra en la figura siguiente:

Figura 2. **Señal digital de voltaje dependiente del tiempo**



Fuente: *Transmisores y válvulas inteligentes en la actualidad*. http://www.instrumentacionycontrol.net/cursos-libres/instrumentacion/curso-completo-instrumentacion-industrial/item/232-transmisores-y-v%C3%A1lvulas-inteligentes-en-la-actualidad.html. Consulta: 22 de febrero de 2018.

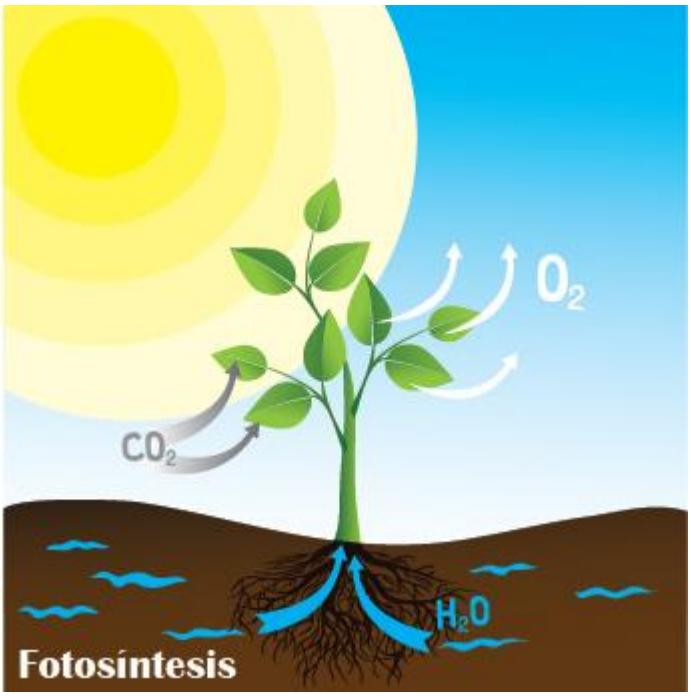
Al igual que en la definición anterior se consideran los valores que la onda toma en amplitud; en el caso específico de la gráfica mostrada se encuentra que son solamente dos, y se le llamará señal binaria. Existen señales digitales con una cantidad mayor de valores en amplitud, pero esta nunca alcanza el infinito (no están definidas continuamente en el eje vertical).

Este tipo de señal necesita de un proceso de conversión; lo que la hace costosa pero más confiable para la transmisión de información.

1.2. Sistema

Conjunto de elementos que se combinan e interactúan armónicamente para cumplir un objetivo. Estos se encuentran definidos por el sujeto observador, que es el que distingue los componentes y cómo se relacionan. Es decir, se trata de una definición relativa.

Figura 3. **Sistema fotosíntesis**



Fuente: Aula365. *La fotosíntesis*. http://www.aula365.com/post/fotosintesis/. Consulta: 22 de febrero de 2018.

La figura 3 brinda el ejemplo claro de que los sistemas no son algo ajeno a la vida cotidiana. Algo tan básico como la continuidad de la vida en la tierra depende del proceso de fotosíntesis en las plantas, para esto es necesario que exista un sistema cuyos elementos principales son la luz del sol, el agua, los

minerales en el suelo, el dióxido de carbono como entradas y el oxígeno como salida principal del proceso.

Cada sistema participa en una jerarquía siendo parte de uno mayor (supersistema), o albergando otros menores. En el ejemplo, un árbol puede ser parte de un sistema más grande que permita la fotosíntesis y también puede él mismo estar formado por otros más pequeños con funciones vitales para su propia existencia.

Al igual que las señales, los sistemas se pueden clasificar de muchas formas. Sin embargo, para los propósitos de estudio de procesadores, interesa el siguiente subgrupo:

1.2.1. Sistema digital

Conjunto de elementos destinado a generar, transmitir o procesar señales digitales. Estos se encuentran primordialmente conformados por dos tipos de componentes:

- Elementos lógicos
    - Software: grupo intangible de información almacenado en memoria en forma de instrucciones que describen la operación general del sistema. Está especialmente diseñado para ser modificable.

Una parte esencial de este grupo son los traductores, estos se encargan de traducir un programa escrito en un lenguaje inicial a uno funcionalmente equivalente. Entre estos se encuentran:

        - Ensambladores: convierten lenguaje ensamblador a lenguaje de máquina.
        - Compiladores: convierten lenguaje de alto nivel a lenguaje de máquina en un solo paso.
        - Intérpretes: convierten el código como un compilador, con la diferencia de la ejecución inmediata de las instrucciones cuando se leen, ellos los hace esencialmente más rápidos.

    - Firmware: tipo especial de software no modificable (a menos que se haga deliberadamente) destinado a propósitos específicos con instrucciones de muy bajo nivel para el control del hardware.

- Elementos físicos

    - Hardware: sección tangible del sistema. Forma parte del mundo perceptible por los sentidos y se caracteriza por ser, comúnmente, reemplazable.

1.3. Sistemas de numeración

Conjunto de símbolos y convenios que permiten representar todas las cantidades existentes.

El estudio de sistemas digitales se caracteriza por la aparición frecuente de representaciones numéricas distintas a la decimal (de base 10), dado que esto abre una conexión directa entre la forma en que la información se trata digitalmente y métodos que resultan cómodos para la comprensión humana. Dos de las más usadas en el área computacional son:

1.3.1. Sistema binario

Sistema de numeración de base 2 en que los números se representan utilizando únicamente las cifras 0 y 1, cada uno llamado bit como contracción de *binary digit*. En electrónica e informática la importancia radica en que las computadoras trabajan básicamente con dos estados de señal eléctrica: alto y bajo, provocando que la forma binaria sea su representación natural.

1.3.2. Sistema hexadecimal

El sistema por excelencia de las computadoras y varias aplicaciones digitales de la electrónica es el binario. Sin embargo, este puede resultar laborioso para un humano cuando se trata de cifras largas.

El sistema hexadecimal tiene base 16 con representación de 10 dígitos de la numeración decimal seguidos de 6 letras del alfabeto latino. Se muestra usualmente como una contracción del binario dada su fácil conversión, ya que cada cuatro cifras binarias equivalen a una hexadecimal; haciendo su uso es bastante extendido.

Habiendo comprendido los cimientos del estudio computacional es posible comenzar a indagar en tramos más específicos, y se hace necesario primordialmente definir a los procesadores y las características esenciales en su configuración:

1.4. Procesador

En ciencias de la computación, procesador se define como un circuito electrónico que ejecuta operaciones sobre información proveída externamente.

Usualmente el término se refiere a la unidad central de procesamiento (CPU), aunque puede referirse también a la implementación de circuitos especializados para el manejo central de dispositivos como los SoC (*System on a Chip*).

El término 'microprocesador' simplemente hará alusión a un procesador que cumple las funciones de un CPU en un solo chip.

Figura 4. **Ejemplo de implementación de microprocesador**

Fuente: informaikta. *Clase de informática: apuntes, herramientas, aplicaciones, seguridad, actividades, curiosidades*. http://informaikta.blogspot.com/p/hardware.html. Consulta: 22 de febrero de 2018.

Según las aplicaciones específicas para las que esté optimizado un procesador, pueden distinguirse algunos grupos generales:

- CPU: unidad central de procesamiento
- GPU: unidad de procesamiento de gráficos
- VPU: unidad de procesamiento de vídeo
- VPU: unidad de procesamiento de visión
- TPU: unidad de procesamiento tensorial
- NPU: unidad de procesamiento neuronal
- PPU: unidad de procesamiento físico

- DSP: procesador digital de señales
- SPU o SPE: elemento sinérgico de procesamiento
- Chip de sonido

Todos estos poseen por lo general características en común que permiten su funcionamiento básico.

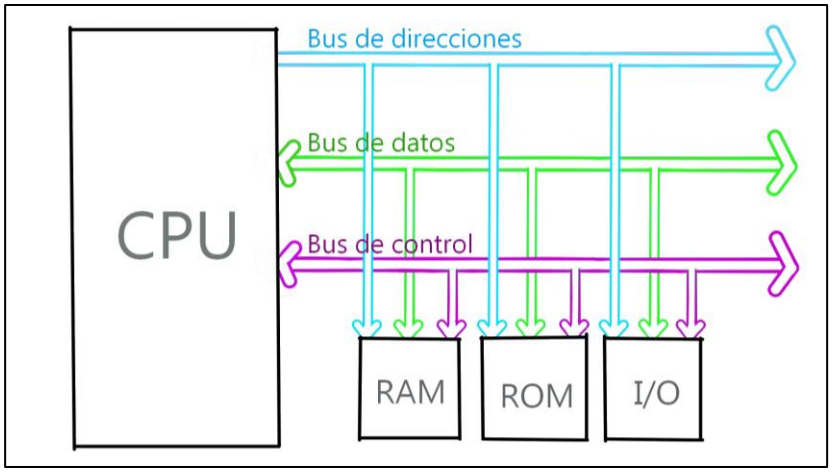
1.4.1. Bus

Conjunto de elementos que facilitan la transmisión de datos entre los componentes de una computadora o fuera de ella. Es parte del hardware, que se forma con cables, pistas y dispositivos físicos.

Los buses se clasifican por la cantidad de información que transmiten simultáneamente (en bits), correspondiendo a la cantidad de vías por las que se envían los datos. La velocidad se define por frecuencia de trabajo (en Hz). En un procesador los buses se clasifican en:

- Bus de control: dicta el acceso de datos y direcciones, ejecutando los mecanismos para evitar zonas de colisión. Maneja señales de órdenes y sincronización.
- Bus de direcciones: en él se maneja la dirección de memoria a la que corresponde la información en el bus de datos, dado que cada segmento de memoria posee una dirección para su fácil acceso.
- Bus de datos: permite la transmisión de información entre el CPU y los demás dispositivos.

Figura 5. **Esquema básico de conexiones entre CPU, memorias y periféricos por medio de buses**



Fuente: elaboración propia, empleando Visio 2013.

El ancho de información que se maneja dentro de los buses suele, por convención, recibir nombres según cuántos bits contiene cada conjunto. Esto puede variar según cada arquitectura de procesador; sin embargo el entendimiento general de estos términos se divide en:

- Bit: unidad fundamental de representación para el sistema binario en sistemas informáticos.
- Byte: conjunto de ocho bits.
- Palabra: es la unidad propia de cada procesador y porque la que se rige su circuitería (registros, buses, entre otros). Los anchos más comunes de palabra para procesadores usados en sistemas embebidos son 32 y 64 bits (ejemplos de ambos grupos se estudiarán más adelante).
- Doble palabra: conjunto con ancho igual al doble de lo ocupado por una palabra. Este término suele utilizarse en procesadores cuya arquitectura

deriva de otra con un ancho de palabra menor al de la actual (es decir, se utiliza para mantener uniformidad en términos). De la misma forma existen los términos 'palabra cuádruple' y 'palabra doble cuádruple'.

1.4.2. Memoria

Dispositivo de almacenamiento de información (datos e instrucciones). Según la forma en que la información es almacenada y leída, se distinguen algunos tipos principales de memoria:

- RAM (*Random Access Memory*): almacenamiento temporal. Cumple la característica de ser volátil, y la información desaparece al desenergizar el procesador. Posee la cualidad de ser bidireccional (lectura y escritura), y es el tipo predilecto para almacenamiento de datos.

    - Pila: región de la RAM que almacena datos temporalmente con una estructura LIFO (*Last In First Out*). Se utiliza comúnmente para el control de flujo de un programa y almacenamiento rápido de variables. La localidad disponible (al inicio de la pila) para escritura o lectura en cada instante es señalada por el puntero de pila (*stack pointer*, SP)
    - *Heap*: espacio similar a la pila con tamaño reconfigurable (usualmente mayor) usado para variables globales en lenguajes de alto nivel.

- ROM (*Read Only Memory*): almacenamiento permanente. Su contenido no es fácilmente alterable, dado que es de tipo no volátil. Por poder ser solamente leída, este tipo de memoria se utiliza para el almacenamiento de instrucciones a ser ejecutadas.

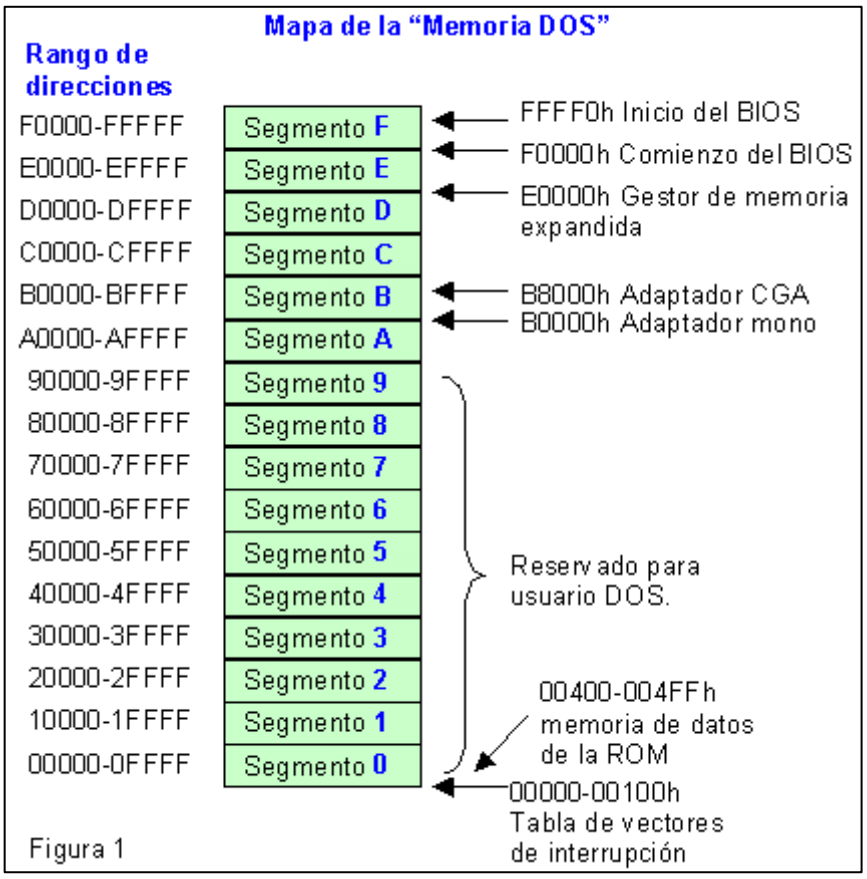
- Caché: sección pequeña y rápida de memoria que actúa de intermediario entre el núcleo y la memoria principal conteniendo información de uso común o reciente. Estas se construyen muy cerca del núcleo, y los accesos a ellas son muy rápidos, aumentando la capacidad de procesamiento del sistema por unidad de tiempo y reduciendo la potencia necesaria para efectuar los accesos a memoria.

La palabra 'cahé' está vinculada en documentación oficial de Arm a la palabra en francés *cacher*, que significa 'esconder'. Hace referencia a que este tipo de memoria se vuelve un espacio en que el procesador almacena datos e instrucciones escondidas del resto del sistema (los demás componentes actúan como si estuvieran leyendo y escribiendo la memoria principal).

Una característica importante en el manejo de memoria es la forma en que esta se alinea (situación que aplica a todos los tipos de memoria). La alineación consiste en asegurar la escritura y lectura de datos en direcciones del tamaño de palabra del procesador. Es decir, si el tamaño de palabra es de 32 bits la información se escribirá o leerá siempre en direcciones múltiplos de 32 (0x0, 0x32, 0x64, 0x96, entre otros) Algunos procesadores pueden solamente acceder a información alineada.

Es común que en las especificaciones de un procesador se encuentre su mapa de memoria, que es una representación esquemática de cómo se segmentan los espacios de memoria (con sus direcciones hexadecimales).

Figura 6. **Ejemplo mapa de memoria**



Fuente: BIOS. *Jerarquía y mapeo de la memoria.* http://creacionesjb.wixsite.com/blogarquitectura/single-post/2015/10/18/Jerarqu%C3%ADa-y-Mapeo-de-la-Memoria-BIOS. Consulta: 20 de mayo de 2018.

1.4.2.1. Memoria virtual

Esta técnica se utiliza para proveer alivio a la escritura extralimitada de la RAM, dado que la mayoría de procesos dependen de ella y su desbordamiento resultaría en fallo del sistema. La forma en que el hardware controla esta situación es tomando cada cierto tiempo secciones de datos (llamadas páginas

por ser siempre del mismo tamaño), para trasladarlas al almacenamiento no volátil y de vuelta con el fin de evitar saturación.

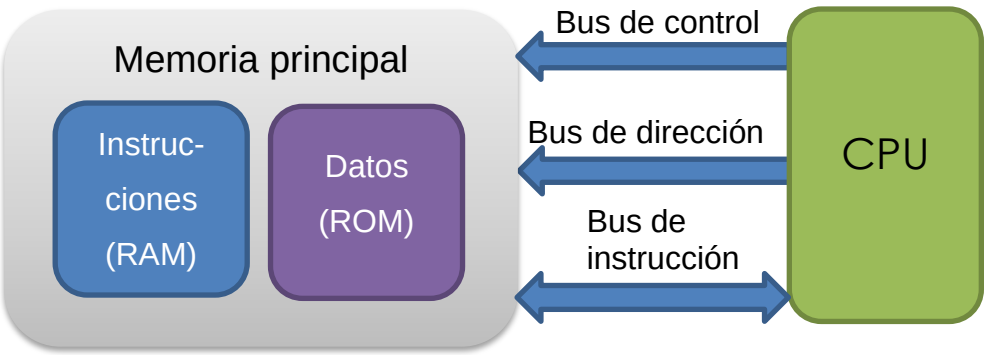
No todas las arquitecturas utilizan este esquema; sin embargo es ampliamente utilizado.

1.4.2.2. Arquitecturas de memoria

Por la forma en que se distribuye el acceso a datos e instrucciones dentro de un procesador, se distinguen dos grupos principales:

- Von Neumann: distribución en que la memoria de datos e instrucciones se encuentran en un mismo bloque. Por lo general la dirección 0x00 se encuentra al inicio de la sección de instrucciones (ROM), y no se ve interrumpido el conteo en los datos (RAM).

Figura 7. **Arquitectura de memoria Von Neumann**

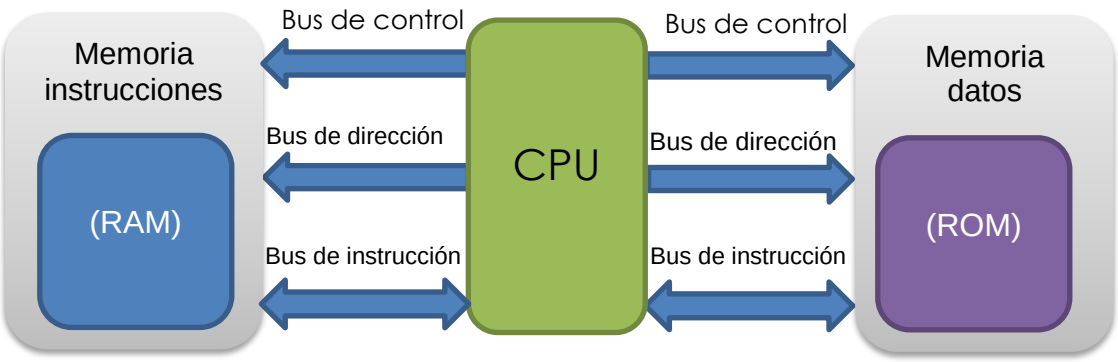


Fuente: elaboración propia.

Esta es una distribución barata para implementar. Sin embargo, dado que los buses se comparten, sólo puede realizarse una transacción por ciclo de reloj.

- Harvard: esta arquitectura separa los espacios de memoria dedicados a instrucciones y datos, con asignación de buses por separado. Cada memoria tiene generalmente su propio conteo de localidades.

Figura 8. **Arquitectura de memoria Harvard**



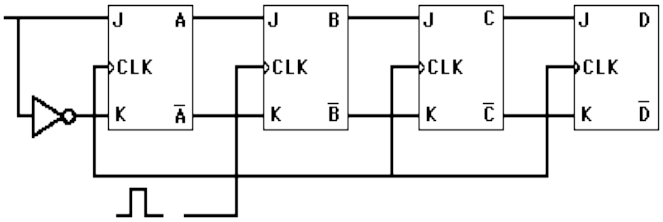
Fuente: elaboración propia.

La ventaja principal de esta arquitectura es la capacidad de leer instrucciones mientras se ejecutan las anteriores, y son ideales para el *pipeline* haciendo el sistema más rápido que con Von Neumann a pesar de que resulta más caro de implementar. Los procesadores Arm están construidos sobre una arquitectura Harvard modificada, que consiste en una memoria principal unificada como en el caso anterior pero con buses separados dedicados a manejo de instrucciones y datos.

1.4.3. Registro

Memoria de poca capacidad y alta velocidad que permite un almacenamiento temporal para acceso rápido de información. Los registros usualmente se miden por su capacidad en bits, siendo estas implementaciones de *flip flops*.

Figura 9. **Esquema básico de registro de cuatro bits, formado con *flip flops* JK**



Fuente: *Registro de desplazamiento*. http://hyperphysics.phy-astr.gsu.edu/hbasees/Electronic/datatran2.html. Consulta: 22 de febrero de 2018.

Los tipos más importantes de registros son:

- Registros de datos
- Registros de memoria
- Registros de propósito general
- Contador de programa
- Registros de punto flotante
- Registros constantes
- Registros de propósito específico (puntero de pila, de estado)
- Registros de banderas

1.4.4. ALU (Unidad Lógico-Aritmética)

Dispositivo digital capaz de ejecutar operaciones aritméticas, lógicas y es una parte fundamental del procesamiento. Su aparición en los procesadores puede ser simple o múltiple, dependiendo de la aplicación para la que estén diseñados.

Las instrucciones que puede llevarse a cabo con una ALU cuentan sobre uno o dos operandos y son básicamente:

- Suma aritmética
- Resta aritmética (complemento a 2)
- Operaciones lógicas
- Producto
- Suma lógica
- Comparación
- Complementación
- Enmascaramiento
- Transferencia
- Rotación

Casi todas estas instrucciones son ejecutables según dos tecnologías:

- Punto fijo: representación introducida a finales de la década de 1980. Consiste en una cantidad fija de dígitos después del punto decimal. La mayoría de sistemas de bajo incluyen esta implementación al no requerir unidades extra de representación. Se compone de bits de magnitud y valores fraccionales.

- Punto flotante: representación numérica con la ventaja del escalamiento de variables. Fue introducida en 1985 por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) por medio del estándar IEEE 754. Esta puede ser presentada como un número de precisión simple (SP) con 32 bits o de precisión doble (DP) con 64 bits. Es una aplicación de resultado exacto y rápido, frecuentemente utilizada para lograr una buena aproximación del número. Se compone de una mantisa multiplicada por una base 2, un exponente que permite escalar y bits dedicados a la especificación de signos tanto en exponente como del número representado.

1.4.5. Arquitectura de set de instrucciones (ISA)

La implementación en silicio de todos los elementos de un procesador y cómo estos se ubican físicamente puede responder a fines específicos según las características que se espera que el sistema contenga. El modelo de esta implementación se describe para cada computadora con un conjunto de instrucciones que corresponden a la forma en que se construye el lenguaje que la máquina ejecuta.

Las primeras computadoras estaban construidas sobre sets pequeños limitados principalmente por los elementos que era posible incorporar a los diseños. Sin embargo, con el paso de los años y la disminución en el tamaño de los transistores, fue posible añadir más características.

Una ISA ofrece la ventaja de dar libertad al diseñador de hardware de disponer los elementos necesarios para implementarla como mejor se adapte a los objetivos y recursos disponibles. Esta descripción de la circuitería necesaria

para llevar al mundo físico un set de instrucciones es conocida como microarquitectura (uArch).

Según los términos del párrafo anterior, una ISA puede contar con innumerables microarquitecturas. Un ejemplo claro de esta condición en la vida cotidiana se evidencia al observar es la existencia de varias implementaciones del set de instrucciones x86 en computadoras portátiles. Los fabricantes Intel y AMD han utilizado ambos esta ISA en sus procesadores realizando distintos diseños (Pentium, AMD Sempron, entre otros), sin embargo, al ejecutar en estos equipos aplicaciones diseñadas para ejecutarse en arquitecturas x86 son igualmente capaces de realizar la tarea.

A pesar de que existen muchos sets de instrucciones, el mundo actual se ha visto principalmente determinado por dos básicos de los que suele derivarse la mayoría de microarquitecturas conocidas: RISC y CISC.

1.4.5.1. RISC (*Reduced Instruction Set Computing*)

Set de instrucciones diseñado para ejecuciones rápidas. Se caracteriza por contener instrucciones simples que se ejecuten en pocos ciclos de reloj (por lo general todas se ejecutan en la misma cantidad de ciclos), esto se debe a que todas son del mismo ancho de bits.

1.4.5.2. CISC (*Complex Instruction Set Computing*)

Set de instrucciones de instrucciones capaces de ejecutar tareas complejas. Al ser específicas, las instrucciones tienen varios anchos de bits en un mismo grupo, y el tiempo en que cada una se ejecuta también es variable. Los sets CISC contienen muchas más instrucciones que los RISC y por la

disponibilidad de operaciones muy específicas se ha usado ampliamente para procesadores robustos, volviéndose base de varias otras ISA como x86 y x64.

1.4.6. *Pipeline*

Llamado también segmentación, es una técnica de paralelismo que toma su nombre de la palabra en inglés equivalente a tubería. La idea central del método, al igual que en un sistema de tuberías, es distribuir la carga del procesamiento de instrucciones en cada elemento del núcleo.

Para utilizar *pipeline* el procesador cuenta con una sección dedicada que divide cada instrucción en pasos que pueden controlarse por distintas secciones al mismo tiempo. En el momento en que el sistema se estabiliza, se dice que se finalizará una instrucción por ciclo de reloj.

Figura 10. **Esquema de etapas de *pipeline***

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Fuente: *Pipelining*. https://www.cs.uaf.edu/2011/spring/cs641/proj1/acornachione/. Consulta: 20 de mayo de 2018.

En la figura 10 se observa cómo al inicio de la segmentación se va sumando la primera etapa de cada instrucción una por una hasta que a partir

del ciclo señalado comienza a terminar de ejecutarse una instrucción por ciclo (estabilidad). El número de ciclos que tarde el sistema en estabilizarse dependerá de la cantidad de etapas que se introduzcan.

2. PROCESADORES ARM

2.1. Historia

Para comprender la visión de Arm hoy, la forma en que la compañía opera y el propósito de las arquitecturas que sus ingenieros diseñan, es necesario repasar los pasos que los llevaron a convertirse de una iniciativa pequeña hace varios años en el gigante de la tecnología que hoy es.

2.1.1. Primeros pasos

ARM Holdings comenzó en la década de 1980 con su sede en Cambridge, Inglaterra. En ese entonces siendo las siglas de Acorn RISC Machines.

Sus cimientos están principalmente en Acorn Computers encargándose de la iniciatva del gobierno británico BBC Micro (con el propósito de entregar una computadora a cada salón de clases en ese país). Sophie Wilson y Steve Furber fueron entonces los científicos encargados de diseñar un procesador de 32 bits que perteneciera completamente a Acorn. Esto llevó al primer diseño en la historia de la compañía: ARM1, creado en 808 líneas de Basic y entregado a Acorn el 26 de abril de 1985. Sin embargo, fue ARM2 el primer chip implementado, a partir de ese momento se conoce a Acorn Archimedes como la primera computadora personal basada en RISC con su liberación en 1987.

La empresa se estableció oficialmente como Advanced RISC Machines Ltd. en noviembre de 1990 en una fusión de esfuerzos entre Acorn Computers, Apple Computer (ahora Apple Inc.) Y VLSI Technology debido a que Apple

quería utilizar la tecnología Arm pero sin basar sus productos en la propiedad intelectual de Acorn.

Apple invirtió el dinero; VLSI, las herramientas y Acorn, los 12 ingenieros con que nació ARM.

Figura 11. **Granero que albergó la primer oficina de Acorn RISC Machines Ltd. en Cambridge, Inglaterra**

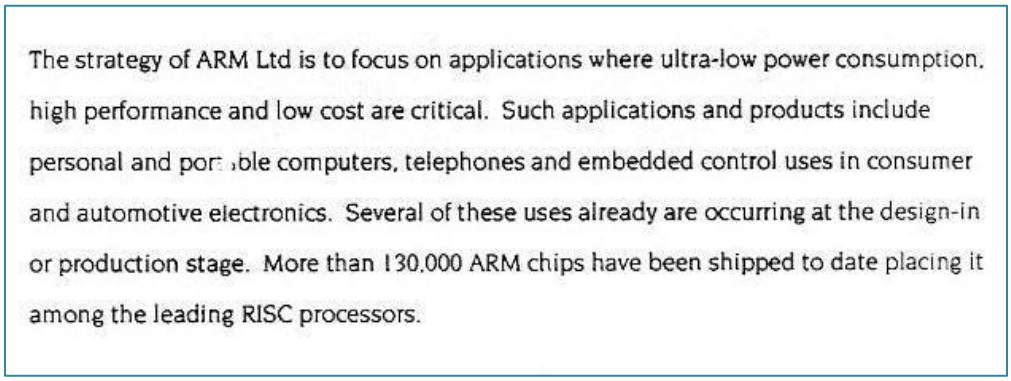
Fuente: A Brief History of ARM. Part 1. https://community.arm.com/processors/b/blog/posts/a-brief-history-of-arm-part-1. Consulta: 28 de febrero de 2018.

Figura 12. **Primer logo de Arm**

Fuente: ARM ADVANCED RISC MACHINES Trademark Information. https://www.trademarkia.com/arm-advanced-risc-machines-74567531.html. Consulta: 28 de febrero de 2018.

A continuación se presenta un extracto en inglés de la primera conferencia de prensa donde se expresa que, desde entonces, la preocupación principal de la empresa fue enfocarse en implementaciones de ultra bajo consumo de potencia, alto rendimiento y bajo costo para su uso en dispositivos móviles.

Figura 13. **Objetivos de Arm en primer conferencia de prensa en 1990**



The strategy of ARM Ltd is to focus on applications where ultra-low power consumption, high performance and low cost are critical. Such applications and products include personal and por: ,ble computers, telephones and embedded control uses in consumer and automotive electronics. Several of these uses already are occurring at the design-in or production stage. More than 130,000 ARM chips have been shipped to date placing it among the leading RISC processors.

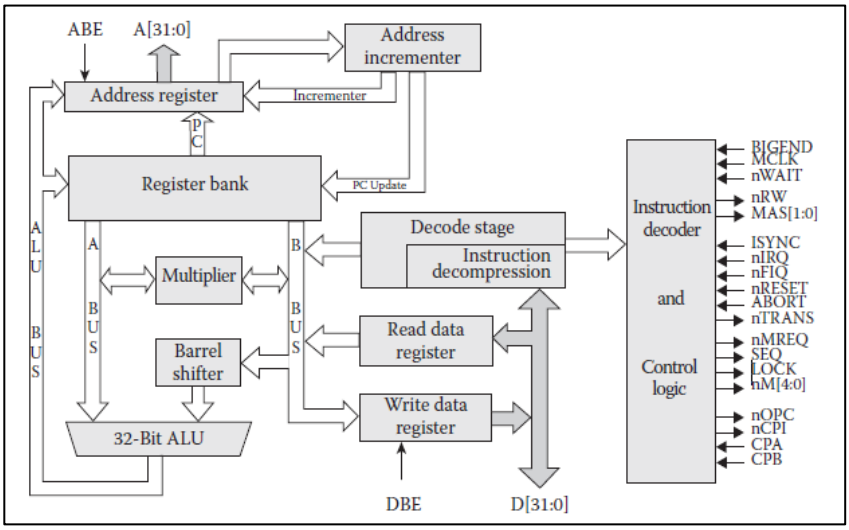
Fuente: *The backstory of how ARM reached a milestone of 86 billion chips in 25 years*. https://yourstory.com/2016/07/arm-holdings-story/. Consulta: 28 de febrero de 2018.

En 1993 el asistente digital personal Apple Newton fue lanzado con una arquitectura ARM, resultando ser una decepción por su elevado precio y fallas en funcionamiento. Debido a la respuesta del negocio, los directivos de la empresa terminaron por comprender algo importante: no les resultaba sostenible apuntar al éxito con productos individuales. Fue por esto que Sir Robin Saxby (director de la compañía en ese tiempo), introdujo el modelo de negocios de propiedad intelectual en el que el procesador ARM tiene una licencia y las compañías afiliadas pagan una cuota para tener derecho a implementarlo en silicio.

2.1.2. El mercado de los dispositivos móviles

Actualmente el poder de procesamiento y el tamaño de dispositivos como *laptops* y teléfonos celulares es algo que se da por sentado. Sin embargo, hace algunas décadas era imposible concebir la idea de escalas tan pequeñas para los procesadores. El punto crucial en la historia de ARM llegó en 1993 con Texas Instruments, con esta afiliación se terminó de comprobar la viabilidad del modelo de negocios recién introducido, ello impulsó los esfuerzos por terminar de formalizarlo y comenzar a trabajar en diseños más económicos llevando a que en 1994 la compañía trabajara en un procesador con 16 bits por instrucción para el Nokia6110 (primer teléfono GSM movido por ARM). Este fue el nacimiento de la tecnología Thumb y el hecho definitivo que llevó a ARM al campo de los teléfonos móviles con una mejora de 35 % en densidad de código, mientras el ARM7TDMI (procesador responsable) tuvo 170 licencias vendidas y con ello, más de 10 billones de unidades fabricadas.

Figura 14. **Diagrama de arquitectura ARM7TDMI**



Fuente: HOHL, William. *ARM Assembly Language Fundamentals and Techniques*. p. 8.

En 1999 se liberó la quinta generación de la arquitectura, introduciendo el procesamiento digital de señales y extensiones de Java *byte code.* Esta se utilizó en sistemas embebidos de gama alta.

2.1.3. Oferta en el nuevo milenio

A inicio de los años 2000, ARM se centraba cada vez más en los sistemas implementados en un solo chip, dado que el tamaño de los procesadores lo hacía posible; sin embargo, sus licencias eran *hard IP* y su aplicación en distintas tecnologías se volvía un problema. Es por esto que en 2001 se anunció el ARM926EJ-S como un diseño sintetizable con 5 etapas de *pipeline*, una MMU (unidad de gestión de memoria) y algunas operaciones con DSP. La licencia del procesador fue entregada a más de 100 proveedores y se ha implementado en varios billones de unidades.

Luego de ARM7 (ARM9E) llegó ARM9, ARM10 y ARM11. Las últimas dos fueron un gran avance en el campo de bajo consumo de potencia y alta capacidad de procesamiento perteneciendo a la sexta generación con mejoras en el set de instrucciones Thumb, soporte para multi núcleos y la introducción del concepto de TrustZone.

En ese momento ARM triplicó la cantidad de sus empleados, maduró su concepto de empresa y decidió diversificar su oferta para cubrir todas las necesidades en la industria a través de ARMv7 (o su séptima generación), incluyendo mejoras en el soporte de punto flotante, extensiones SIMD y lo más importante, un nuevo grupo de procesadores: la familia Cortex.

Cortex fue la respuesta de ARM a la extensión de las aplicaciones demandadas por una industria creciente. Y se dividió, a partir de ese punto, en tres perfiles principales:

- Cortex-A (*applications*): continuando la línea de ARM11, que ofrece la solución a aplicaciones móviles demandantes de alto rendimiento.
- Cortex-R (*real-time*): procesadores especializados en los específicos y exigentes requerimientos de los sistemas de tiempo real.
- Cortex-M (*microcontrollers*): provee diseños de muy bajo consumo de potencia y bajo costo, dedicados a la industria de los microcontroladores.

En 2008 el mercado de los *smartphones* creció rápidamente y con él, la demanda de alto rendimiento y alto cuidado de la batería al mismo tiempo. ARM entonces proveyó el Cortex-A9: un procesador multinúcleo con un dinamismo bueno para adaptarse a todo tipo de usuario, esto se mejoró incluso más con la aparición de la tecnología big.LITTLE en 2011.

Figura 15. **Evolución del mercado principal para los procesadores ARM**

Fuente: Arm Ltd. *Arm Holdings Q3 2017 Roadshows Slides*. Consulta: 3 de marzo de 2018.

2.1.4. Negocios

Una gran parte del éxito de ARM se debe a su modelo de negocios. Las licencias aseguran una ganancia en primera instancia pero son las *royalties* la mayor fuente de ingreso asegurado a largo plazo, porque estas no entran en vigencia hasta cinco años después de vendida una licencia y aumentan su valor conforme el mercado en donde se venden crece. Un ejemplo de esto está en ARM7, que ya no tenía soporte ni era vendido por ARM para el momento de su cumbre en ventas.

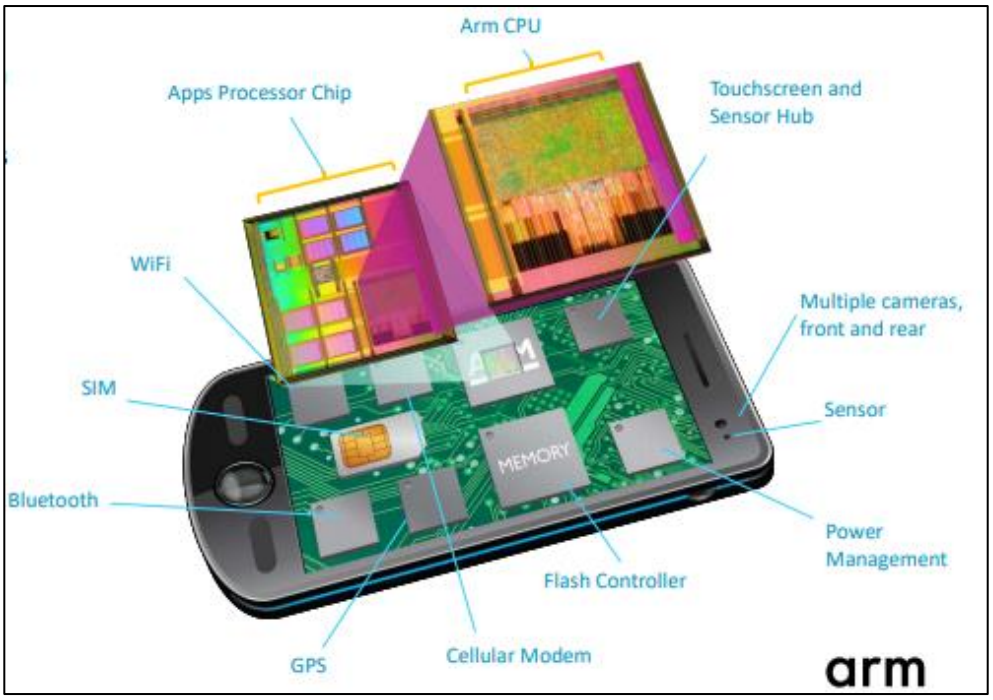
A su vez, los derechos por la propiedad intelectual pueden ser vendidos bajo dos especificaciones:

- Licencia de núcleo: negocio primario de venta de IP cores, utilizados para crear CPUs y SoCs. Los fabricantes deben, entonces, combinar los procesadores ARM con otras partes para producir un dispositivo completo.

- Licencia de arquitectura: este es más específico y dedicado a las necesidades del cliente, se basa en la entrega de sets de instrucciones ARM para la creación de procesadores, cumpliendo con los estándares de la compañía. Empresas que han elegido este tipo de licencia son Apple, Qualcomm y Samsung Electronics.

- Los procesadores basados en licencias ARM son usados en una amplia variedad de aplicaciones desde teléfonos móviles y microcontroladores a servidores, infraestructura de red, sistemas embebidos, industria automotriz, entretenimiento y muchas otras áreas.

Figura 16. **Sistema compuesto de distintos chips complementarios al CPU ARM**

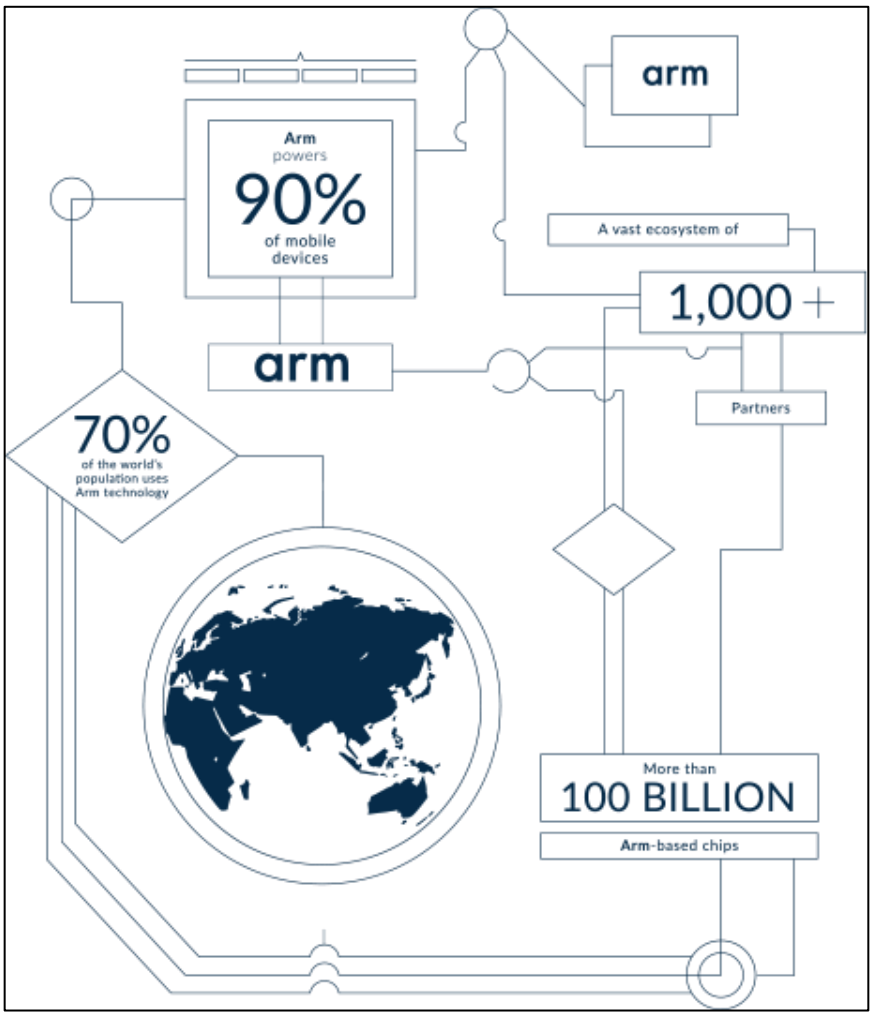


Fuente: Arm Ltd. *Arm Holdings Q3 2017 Roadshows Slides*. Consulta: 3 de marzo de 2018.

Comparado con un estimado de nueve millones de unidades en 1997, los afiliados de ARM han fabricado y entregado alrededor de 100 billones de dispositivos utilizando estos procesadores a la fecha.

La comunidad ARM Connected es el grupo de más de 1 000 compañías trabajando en este ambiente, añadiendo un valor inquebrantable a la arquitectura.

Figura 17. **ARM como un ecosistema**



Fuente: Arm Ltd. https://www.arm.com/company. Consulta: 28 de febrero de 2018.

Finalmente, la compañía tuvo un cambio en su marca al ser adquirida por SoftBank en 2017. Desde este punto, el acrónimo ARM dejó de existir para convertirse simplemente en Arm (a pesar de que la empresa continúa trabajando para los mismos fines), y con este cambio, un nuevo logo haciendo énfasis en que las letras se mantienen por legado a pesar de que ya no representan un significado específico.

Figura 18. **Modificación de logo de Arm Ltd. en 2017**

Fuente: Using the Arm Corporate Logo. https://www.arm.com/company /policies/trademarks/guidelines-corporate-logo. Consulta 05 de marzo de 2018.

2.2. Clasificación

La forma en que ARM clasifica sus generaciones de procesadores se divide principalmente en versiones de las que derivan de una a varias familias. Hasta la fecha se han liberado actualizaciones de la arquitectura que dibujan un árbol genealógico cada vez mayor:

Tabla I. **Resumen de arquitecturas ARM**

Versión	Familias	Características
ARMv1	ARM1	Interrupciones por software. Bus de direcciones de 26 bits.
ARMv2	ARM2, ARM3	Soporte para coprocesadores.
ARMv3	ARM6, ARM7	Introducción de direccionamiento de 32 bits. Mejoras en la velocidad.
ARMv4	Strong ARM, ARM7TDMI, ARM9TDMI	Soporte para set de instrucciones Thumb (16 bits). Implementación de multiplicación de respuesta doble (64 bits).
ARMv5	ARM7EJ, ARM9E, ARM10XE	Mejoras en Thumb. fac Aceleración en ejecución de Java *byte code.*
ARMv6	ARM11	Mejoras en el sistema de memoria. Soporte para instrucciones SIMD (*single instruction multiple data*).
ARMv7	Cortex	División de perfiles A, R y M para optimización.
ARMv8	Cortex	Enfocada principalmente al perfil de aplicaciones. Introducción de direccionamiento de 64 bits.

Fuente: elaboración propia.

2.2.1. Nomenclatura

En la liberación de cada diseño (especialmente en los previos a los Cortex) se añade al nombre ARM siglas que especifican sus características principales. Estas variantes son:

- X = familia
- Y = gestión de memoria
- Z = caché
- T = Thumb
- D = depuración por JTAG
- M = multiplicación rápida
- I = macro celda ICE embebida
- E = instrucción mejorada
- J = Jazelle
- F = unidad de punto flotante
- S = versión sintetizable
- P = protección física
- AE = mejora para sistemas automotrices

2.3. Arquitectura Arm

Siendo ARM una arquitectura RISC, posee características propias de este grupo como:

- Una cantidad grande de registros
- Modo de arquitectura l*oad/store,* donde las operaciones de procesamiento de datos se realizan sobre contenido en registros y no directamente en memoria.

- Modos simples de direccionamiento con las direcciones necesarias para el *load/store* contenidas en registros.
- Campos de instrucción de longitud fija, para simplificar la decodificación de instrucciones.
- Además de estas características, ARM provee por su parte:
- Control sobre ALU y desplazador de bits para maximizar su rendimiento en instrucciones de procesamiento de datos.
- Modos de direccionamiento de auto decremento y auto incremento para optimizar los ciclos.
- Instrucciones *Load and Store Multiple* para maximizar la salida de datos.

Las mejoras que ARM añade a RISC permiten a sus procesadores alcanzar un balance entre alto rendimiento, brevedad de código, bajo consumo de potencia y un área reducida en silicio.

Además, cuentan con características que permiten al circuito realizar todas sus funciones de formas específicas, como se detalla a continuación:

2.3.1. Registros de propósito general

La arquitectura tiene 31 registros de 32 bits (una palabra), de ancho cada uno. Sólo 16 de estos son visibles en el modo de usuario mientras el resto se utilizan en procesamiento.

Dentro de los 16 registros accesibles (con nomenclatura Rn), se encuentran algunos con funciones especiales:

- Puntero de pila: usualmente R13, lleva por siglas SP (*Stack Pointer*). Sirve para insertar y obtener información de la pila.

- Registro de enlace: R14, con siglas LR (*Link Register*). Mantiene la dirección de la siguiente instrucción después de un salto en el programa.
- Contador de programa: R15, con siglas PC (*Program Counter*). Indicador para apuntar a la instrucción dos líneas debajo de la ejecutada en el momento.

Por simplicidad, en la documentación técnica, suele referirse a los tres registros mencionados por sus siglas.

2.3.2. Registros de estado

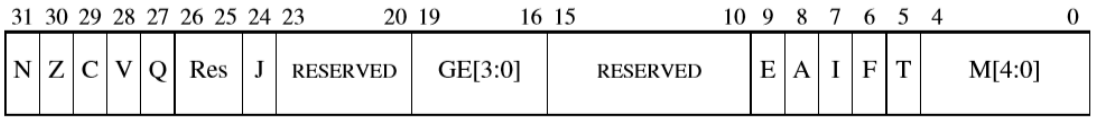
Todos los demás registros del procesador, además de los de propósito general, están incluidos en este grupo. A su vez, los registros de estado se dividen en dos:

- *Current Program Status Register* (CPSR): almacena las condiciones del estado actual del procesador. Contiene:
    - Cuatro banderas de condición (N – negativo, Z – cero, C – acarreo, V - desbordamiento).
    - Una *sticky flag* (Q), a partir de ARMv5. Indica si ha ocurrido saturación en instrucciones de aritmética saturada o un desbordamiento con signo en instrucciones de multiplicación acumulada.
    - Cuatro banderas GE (*Greater than or Equal*) a partir de ARMv6. Indica si los resultados de operaciones con signo son no negativas o si una operación sin signo ha producido un acarreo.
    - Dos bits de deshabilitación de interrupciones (I para IRQ y F para FIQ).

- Una máscara de bits (A), para modo de aborto por imprecisión. Desde ARMv6.
- Cinco bits de codificación del estado actual del procesador. Este campo de modo funciona como máscara de bits para especificar una de las siguientes opciones:
- Modo de usuario: b10000.
- Modo FIQ: b10001.
- Modo IRQ: b10010.
- Modo supervisor: b10011.
- Modo de aborto: b10111.
- Modo indefinido: b11011.
- Modo de sistema: b11111.
- En caso de escribir un número no especificado, el procesador entra en un modo irrecuperable haciendo necesario el *reset*.
- Dos bits para determinar si se ejecutan *opcodes* de instrucciones ARM, Thumb (T) o Jazelle (J).
- Un bit de control de *endianness* (E) para operaciones load/store.

- *Saved Program Status Register* (SPSR): almacena las condiciones del estado actual del procesador inmediatamente después de ocurrida una excepción. Es una forma de preservar el CPSR para su uso posterior.

Figura 19. **Formato de los registros de estado**



Fuente: SEAL, David. *ARM Architecture Reference Manual*. p. 48.

En la figura 19 se observa la forma en que están distribuidos los elementos generalmente en un registro de estado, sea CPSR o SPSR, como se especificó anteriormente.

2.3.3. Modos de procesador

Los procesadores ARM, con excepción del perfil M para sus versiones 6 y 7 tienen soporte para siete modos de trabajo:

- Usuario: único no privilegiado, en él corren por defecto las aplicaciones
- FIQ: entrada de interrupción de alta prioridad (*fast interrupt*)
- IRQ: entrada de interrupción de baja prioridad (*normal interrupt*)
- Supervisor (SVC): utilizado en un *reset* o en la presencia de interrupción por software.
- Aborto: usado para manejar violaciones al acceso de memoria.
- Indefinido: inducido para manejar instrucciones indefinidas.
- Sistema: modo privilegiado con el uso de los mismos registros que en el modo de usuario.

Tabla II. **Prioridad de ejecución de excepciones ARM**

Prioridad	Núm.	Excepción
Mayor	1	Reset
	2	Aborto por datos
	3	FIQ
	4	IRQ
	5	Aborto impreciso (externo)
	6	Aborto por *prefetch*
Menor	7	Instrucción indefinida

Fuente: elaboración propia.

La mayoría de aplicaciones correrá en modo de usuario, dejando al resto como modos privilegiados debido a que suelen proveer una puerta de acceso a recursos protegidos.

Los cambios en modo podrían ser hechos por medio de software; sin embargo, es más usual que su causa sean condiciones externas o excepciones.

Es importante considerar que todos los modos, excepto el de usuario y de sistema son inducidos por excepciones, y tienen sus propios SPSR, SP y LR (solo FIQ hace una copia también a partir de R8 para facilitar la entrada veloz al modo), como una forma de asegurar el regreso al punto exacto en que el programa cesó antes de la excepción y a la vez, para ser capaz de señalar la instrucción que causó el cambio en el modo.

Figura 20. **Registros existentes en cada modo de procesador**

Modo					
Usuario/ Sistema	Supervisor	Aborto	Indefinido	Interrupción	Interrupción rápida
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	R8	R8	R8	R8_FIQ
R9	R9	R9	R9	R9	R9_FIQ
R10	R10	R10	R10	R10	R10_FIQ
R11	R11	R11	R11	R11	R11_FIQ
R12	R12	R12	R12	R12	R12_FIQ
R13	R13_SVC	R13_ABORT	R13_UNDEF	R13_IRQ	R13_FIQ
R14	R14_SVC	R14_ABORT	R14_UNDEF	R14_IRQ	R14_FIQ
PC	PC	PC	PC	PC	PC
CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_SVC	SPSR_ABORT	SPSR_UNDEF	SPSR_IRQ	SPSR_FIQ

Registros copiados

Fuente: elaboración propia.

Cuando ocurre una excepción, el procesador ARM detiene la ejecución y comienza el procedimiento en una de varias direcciones de memoria prefijadas (vectores de excepción), ello significa que hay una ubicación de vector para cada excepción.

Excepto por el *reset*, todas las excepciones ocurren de forma similar:

- El procesador cambia al modo de ejecución correspondiente
- Se guarda la dirección de la instrucción siguiente a la que provocó la entrada a la excepción en LR del nuevo modo.
- Se almacena el valor antiguo de CPSR en el SPSR del nuevo modo.
- Se deshabilita IRQ o FIQ.
- Comienza la ejecución desde el vector de excepción.

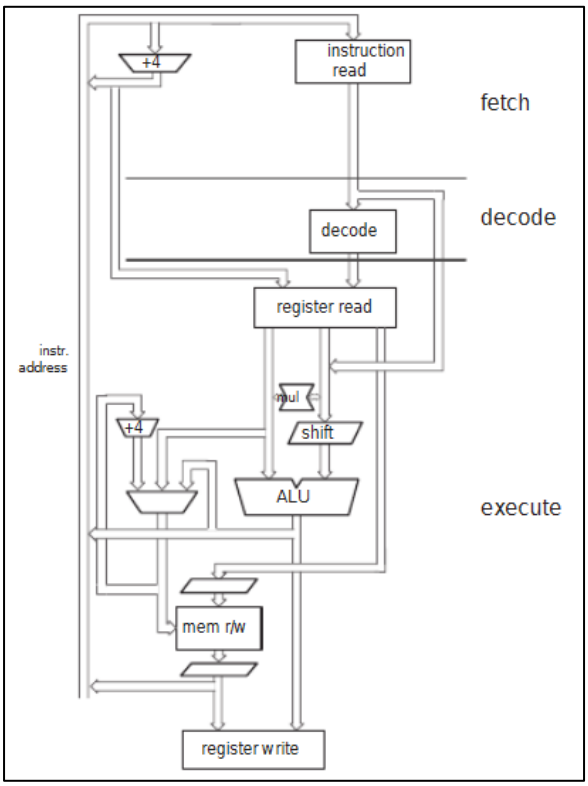
2.3.4. *Pipeline*

Como una técnica de paralelismo a nivel de instrucciones, la implementación de *pipeline* en la arquitectura ARM es un vital en la combinación de alto rendimiento y simplicidad que los diseños tienen por característica principal. Esta característica ha evolucionado a través de los años con diferencias notables entre sus versiones.

2.3.4.1. *Pipeline* de tres etapas

Primer modelo de ARM, permaneció básicamente igual desde ARM1 hasta ARM7TDMI. Se trata de una configuración clásica de *fetch-decode-execute* con la finalización de una instrucción por ciclo de reloj.

Figura 21. ***Pipeline* de 3 etapas**



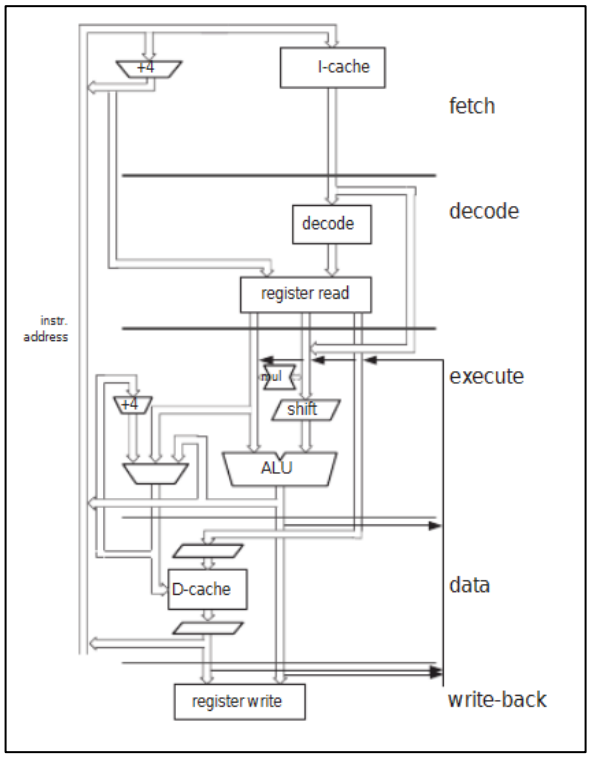
Fuente: Researchgate. *The ARM Architecture*. https://www.researchgate.net/publication/228392292_The_ARM_Architecture. Consulta: 6 de marzo de 2018.

Como se observa en la figura 21, la primera etapa es *Fetch*, que lee la instrucción desde memoria e incrementa el valor del registro de dirección de instrucción y el PC. La segunda etapa (*Decode*), decodifica la instrucción y prepara las señales de control para la ejecución. La tercera etapa (*Execute*), lee los operandos de los registros, realiza las operaciones de ALU, lee o escribe en memoria si es necesario y al final, escribe de vuelta los valores de registros modificados.

2.3.4.2. *Pipeline* de cinco etapas

Esta variante asume la disponibilidad de un solo puerto de memoria, significando que cada instrucción de transferencia de datos provocará una burbuja, debido a que la siguiente instrucción no puede estar en *fetch* mientras se lea o escriba en memoria.

Figura 22. ***Pipeline* de 5 etapas**



Fuente: Researchgate. *The ARM Architecture*. https://www.researchgate.net/publication/228392292_The_ARM_Architecture. Consulta: 6 de marzo de 2018.

Una forma de resolver esto (implementada desde ARM9TDMI), es usar cachés separadas para instrucciones y datos. La lectura de registros se mueve a la etapa *decode*, luego la etapa *execute* se divide en tres (la primera para

aritmética, la segunda para accesos a memoria y la tercera para escribir resultados de vuelta en registros). Los cambios provocan una estructura de pipeline más balanceada; con algunas complicaciones como la necesidad de resolver dependencias de los datos entre etapas sin producir burbujas.

2.3.4.3. *Pipeline* de seis etapas

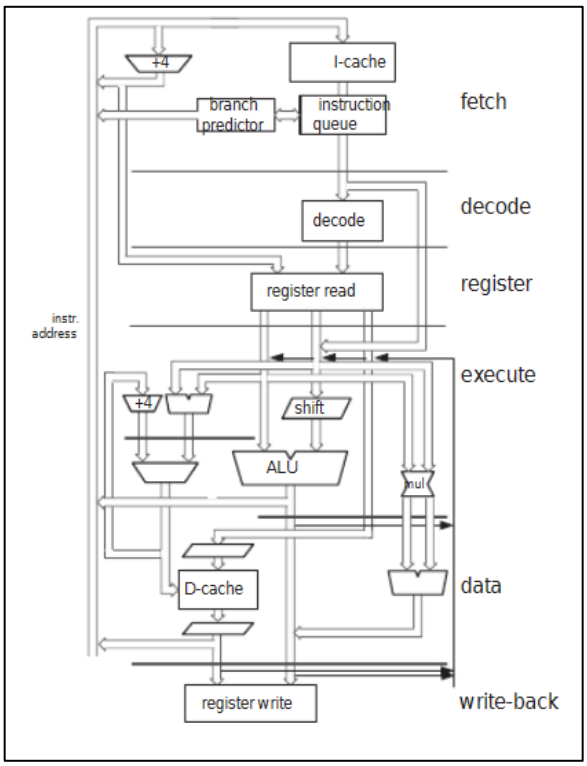
ARM10 fue el siguiente paso en mejoras de la estructura de pipeline.

Debido a que la eficiencia del *pipeline* está principalmente dada por ancho de banda de memoria, se modificó los buses de datos y de instrucción para ser de 64 bits de ancho cada uno. La modificación permite procesar dos instrucciones en la etapa de *fetch* y con esto, la posibilidad de introducir un predictor de saltos para tratar de anticipar el rumbo de las instrucciones hacia el pasado y futuro, reduciendo el procesamiento repetitivo en ciclos que se ejecutan muchas veces.

El siguiente cambio fue descargar la etapa *execute* introduciendo en la etapa data un sumador por separado, para instrucciones de multiplicación-acumulación en lugar de utilizar la ALU principal para sumar. Esto mejora el balance y permite ejecutar a frecuencias de reloj mayores.

De forma similar se introdujo un sumador dedicado al procesamiento de direcciones en la etapa *memory access*. Por último, *instruction decoding* fue establecida como una etapa por separado.

Figura 23. ***Pipeline* de 6 etapas**



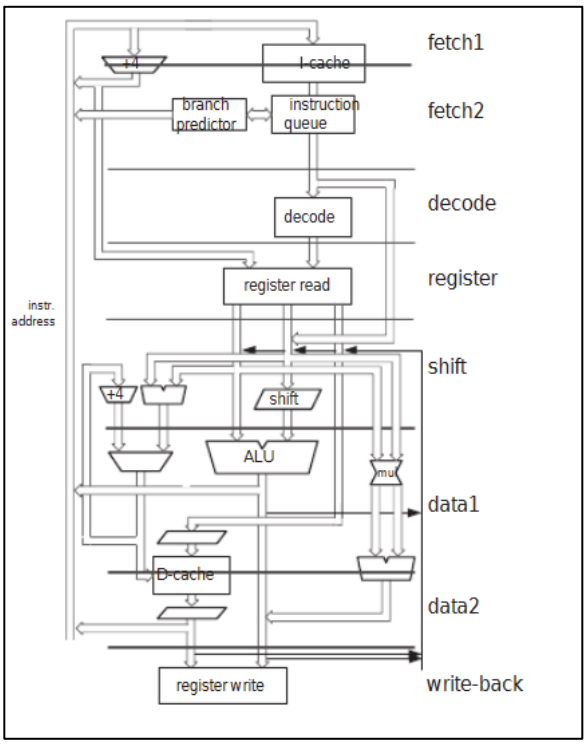
Fuente: Researchgate. *The ARM Architecture*. https://www.researchgate.net/publication/228392292_The_ARM_Architecture. Consulta: 6 de marzo de 2018.

2.3.4.4. *Pipeline* de ocho etapas

A partir de ARM11 se hicieron dos cambios importantes a la estructura. Primero, la operación de desplazamiento de bits fue establecida en una etapa separada. Segundo, las cachés de datos y de instrucciones se distribuyeron en etapas distintas.

Es importante notar que la sección de ejecución de instrucciones de las ocho etapas resultantes operan de forma concurrente en algunas situaciones. *Fetch* y *decode* se ejecutan una después de la otra.

Figura 24. ***Pipeline* de 8 etapas**



Fuente: Researchgate. *The ARM Architecture*. https://www.researchgate.net/publication/228392292_The_ARM_Architecture. Consulta: 6 de marzo de 2018.

Luego de estas versiones de *pipeline* se vuelve complicado describir las mejoras que se han implementado como una sola línea, debido a que componen muchas variantes (especialmente con la aparición de los Cortex), como un efecto del enfoque en ramas específicas de los procesadores. Por esta razón puede encontrarse en la actualidad desde dispositivos del perfil M con cuatro etapas hasta otros del perfil A con diez y hasta once etapas. Las especificaciones necesarias serán mencionadas cuando se trate de un dispositivo individual.

2.3.5. Coprocesadores

Los elementos de procesamiento que sirven como apoyo a la unidad central suelen ser básicamente las mismas a través de diversas arquitecturas. Para Arm los principales se describen a continuación.

2.3.5.1. Control de sistema

Maneja todas las especificaciones estándar de memoria y sistema. Está identificado con el número 15 dentro de los coprocesadores (CP15), y su función principal es controlar y proporcionar información del estado de funciones implementadas en el procesador.

A partir de ARMv4 está incluido con soporte para chequeo automático de caché, TCM (*Tightly Coupled Memory*) y provisiones de coprocesador. Brinda el control para:

- Mecanismos de gestión de memoria como MMU (*Memory Management Unit*), MPU (*Memory Protection Unit*).
- Configuración y control general del sistema.
- Configuración y gestión de caché.
- Precarga de caché L2.
- Monitoreo de rendimiento del sistema.

El coprocesador de control de sistema contiene hasta 16 registros primarios, cada uno con 32 bits de ancho. Estos pueden ser de solo lectura, solo escritura o lectura y escritura.

Tabla III. **Distribución de registros contenidos en coprocesador de control de sistema**

Registro	Uso genérico	Uso específico
0	Códigos de identificación	Identificación del procesador y caché
1	Bits de control	Bits de configuración de sistema
2	Protección y control de memoria	Control de tabla de paginación
3	Protección y control de memoria	Control de acceso a dominio
4	Protección y control de memoria	Reservado
5	Protección y control de memoria	Estado de falla
6	Protección y control de memoria	Dirección de falla
7	Caché y *buffer* de escritura	Control de caché y *buffer* de escritura
8	Protección y control de memoria	Control TLB
9	Caché y *buffer* de escritura	Bloqueo de caché
10	Protección y control de memoria	Bloqueo de TLB
11	Control TCM	Control de DMA
12	Reservado	Reservado
13	Identificación de proceso	Identificación de proceso
14	Reservado	-
15	Definido por implementación	Definido por implementación

Fuente: elaboración propia.

2.3.5.2. Depurador

Coprocesador incluido a partir de ARMv6, antes de esto se consideraba accesorio. Corresponde al coprocesador 14 (CP14). El soporte se ha extendido para brindar:

- Registro de identificación de depurador (DIDR)
- Registro de control y estado de depurador (DSCR)
- Soporte para *breakpoints* y *watchpoints* por hardware
- Un *Debug Communications Channel* (DCC)

Además de la interfaz por software, una interfaz de depuración externa reúne una serie de requerimientos (habilitación de depurador, solicitud y

señalización de reconocimiento). Puede ser usada para controlar los eventos del depurador; para permitir esto, el núcleo necesita ser configurado en uno de los dos modos:

- Depuración detenida: permite que el sistema entre al estado de depuración cuando ocurre un evento. Como consecuencia de estar detenido, el procesador ignora al sistema externo y no puede realizar interrupciones.
- Depuración monitoreada: causa una excepción de depurador como resultado de un evento.

Ambos métodos pueden combinarse con el monitor utilizando alguna característica estándar del sistema (como UART o Ethernet) para comunicarse con el dispositivo a depurar, alternativamente es posible el uso de DCC como un medio de comunicación separado.

Un evento de depurador puede ser:

- Por software
- Generado por una interfaz externa (activación de la señal *External Debug Request* o un comando de solicitud).

Y el procesador responde a él en alguna de las siguientes formas:

- Ignorar el evento
- Entrar al modo de depuración
- Tomar una excepción

Esta respuesta depende de las configuraciones como se muestran en la tabla IV con DBGEN como generador de evento de depurador.

Tabla IV. **Comportamiento del procesador en eventos de depurador**

DBGEN	DSCR [15:14]	Modo	Evento
0	0bxx	Depurador deshabilitado	Ignorar/Abortar
1	0b00	Ninguno	Ignorar/Abortar
1	0bx1	Detenido	Entrada al estado
1	0b10	Monitor	Excepción/Ignorar

Fuente: elaboración propia.

Un evento puede, a su vez, ser alguno de los mencionados a continuación:

- Evento de watchpoint
  - La dirección virtual de depurador coincide con el valor del watchpoint.
  - Todas las condiciones del registro de control de watchpoint (WCR) coinciden.
  - El watchpoint está habilitado.
  - La instrucción que comenzó el acceso a memoria se ve comprometida por la ejecución.

- Evento de *breakpoint*
  - El *breakpoint* está habilitado
  - La instrucción se obtiene y el R13 de CP15 coincide con el valor del *breakpoint.*
  - La instrucción se ve comprometida por la ejecución.

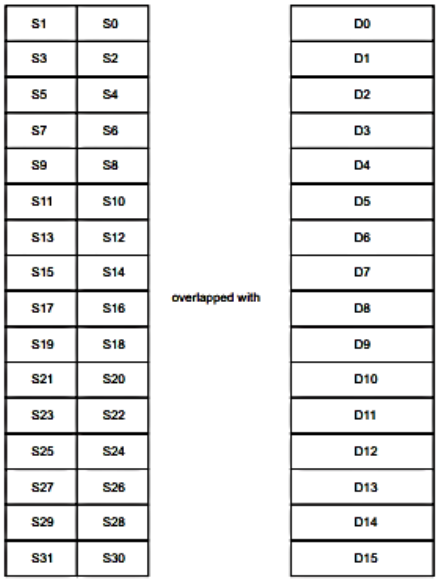
- Evento de *breakpoint* por software
- Evento vector catch

2.3.5.3. VFP (vector floating point)

Se definen los formatos de precisión doble y precisión simple de punto flotante según el estándar IEEE 754 y su uso en la arquitectura VFP.

VFP tiene 32 registros de propósito general (llamados Sn), cada uno con capacidad de un número de punto flotante con precisión simple o un entero de 32 bits. En variaciones D, los registros pueden ser usados en pares (llamados Dn), para almacenar hasta 16 números de punto flotante con precisión doble. La superposición de registros S para formar registros D se muestra en la figura 25.

Figura 25. **Registros de propósito general VFP**

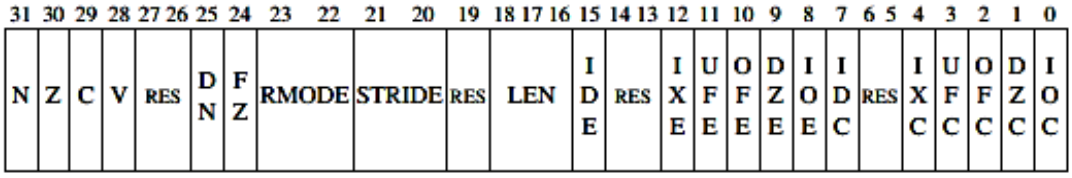


Fuente: SEAL, David. *ARM Architecture Reference Manual*. p. 879.

También pueden existir tres o más registros de sistema:

- FPSID: puede ser leído para determinar cuál implementación de VFP está siendo usada. Hace posible el control a nivel de usuario.
- FPSCR: provee información de control y estado. Los bits de estado contienen resultados de comparación y banderas de acumulación para excepciones de punto flotante. Los bits de control son usados para seleccionar opciones de redondeo.

Figura 26. **Formato de registro de estado de punto flotante**



Fuente: SEAL, David. *ARM Architecture Reference Manual*. p. 885.

En este, la nomenclatura se divide en:

- Los bits [31:28], corresponden a banderas idénticas a las de los demás registros de estado.
- El bit 25 está destinado a control por defecto de respuestas NaN.
- El bit 24 es *Flush-to-zero*.
- Los bits [23:22], están ligados al control de redondeo.
- Los bits [18:16], componen el campo LEN para control de longitud de vectores.
- Los bits 15, [12:7] y [4:0] corresponden a cada una de las excepciones de punto flotante en modos específicos.

- FPEXC: contiene algunos bits para control y estado a nivel de sistema. El resto de bits son usados para comunicación interna entre el hardware y software del VFP.

Además de estos registros, el acceso a VFP es controlado por el coprocesador de control de sistema.

La arquitectura tiene soporte para las cinco excepciones de punto flotante definidas en el estándar IEEE 754:

- Operación inválida
- División en cero
- Desbordamiento positivo
- Desbordamiento negativo
- Inexacto

Implementaciones del VFP pueden ser clasificadas según su inclusión de hardware:

- Implementación de software: como su nombre lo describe, consiste solamente en software, con la aritmética siendo emulada por rutinas ARM. Es también llamada emulador VFP, no es la opción más recomendada por su falta de eficiencia.
- Implementación de hardware: contiene software y hardware diseñado específicamente para optimizar el desempeño.

Existen características mucho más específicas de ARM cuyo análisis es esencial para comprender el funcionamiento de sus procesadores como los tipos de instrucciones, las extensiones Thumb y Jazelle y detalles de sus

coprocesadores. Estas serán introducidas posteriormente por pertenecer a la arquitectura de set de instrucciones y corresponder, por lo tanto, al estudio de instrucciones disponibles para uso en lenguaje de ensamblador.

2.3.6. Memoria y arquitectura de sistema

La arquitectura ARM, como es notorio hasta ahora, ha evolucionado con el paso de varios años provocando que los requerimientos del sistema de memoria varíen considerablemente desde bloques de memoria con mapeo simple a sistemas utilizando algunos o todos los recursos para optimizar la memoria:

- Varios tipos de memoria
- Caché
- Buffers de escritura
- Memoria virtual y otras técnicas de reasignación de valores

El comportamiento de la memoria puede clasificarse por tipos:

- Fuertemente ordenada
- Dispositivo
- Normal

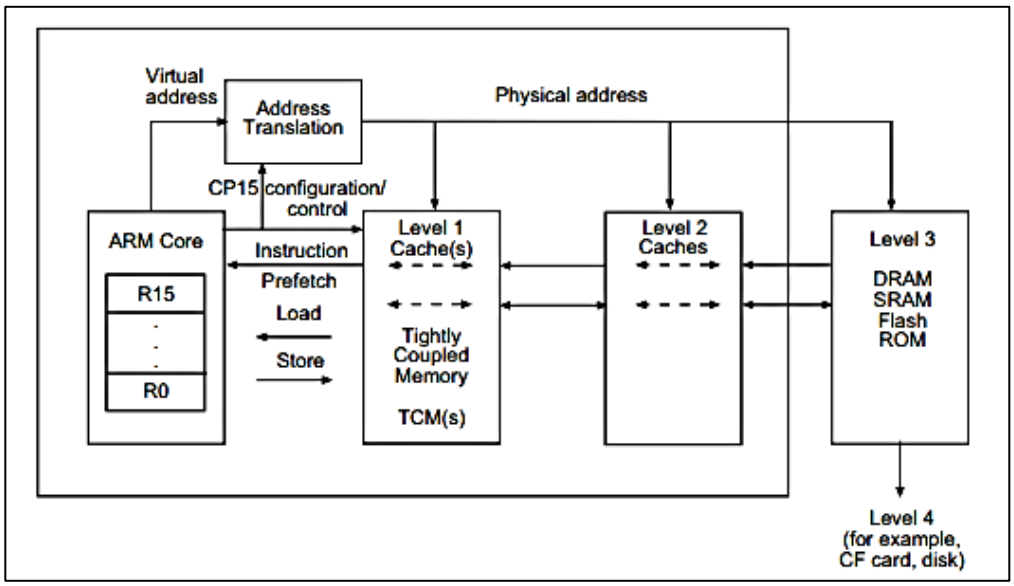
Estos se clasifican, por atributos compartidos, de caché y mecanismos de acceso.

2.3.6.1. Jerarquía de memoria

Buen diseño de sistema es un balance entre muchas variables para alcanzar las metas de eficiencia y costo en general. Una parte importante en este grupo son los recursos de memoria (tipo, tamaño, velocidad de acceso y arquitectura). Por regla general, mientras mayor es la velocidad de acceso, más restringida es la cantidad de recursos disponibles debido a la cercanía física que se debe tener al núcleo. Las caché proveen medios para compartir los recursos más rápidos y costosos del sistema de memoria.

Como se diseña un sistema con distintos tipos de memoria en capas, se le conoce como una jerarquía de memoria. Esto implica que las caché del sistema se adhieren a una estructura enumerada de nomenclatura *level1, level2… leveln* con cada número indicando aumento en tiempos de acceso, y memorias más alejadas del núcleo contarán con números mayores, como se muestra en la figura 27.

Figura 27. **Ejemplo de jerarquía de memoria**



Fuente: SEAL, David. *ARM, Architecture Reference Manual*. p. 648.

- Caché L1

Una caché es un bloque de ubicaciones de memoria de alta velocidad conteniendo información de direcciones y los datos asociados. El propósito es incrementar la velocidad promedio de los accesos a memoria.

Este tipo de memoria opera por dos principios de localidad:

- Espacial. Un acceso es muy probablemente seguido por otros a localidades adyacentes.
- Temporal. El acceso a un área es muy probablemente repetido en un corto periodo.

.

Para minimizar la cantidad de información de control almacenada, la propiedad de localidad espacial se usa para agrupar varias ubicaciones en un bloque lógico conocido como línea de caché y es, típicamente, de 32 bits de ancho.

Cuando se almacena información en estas memorias, el acceso en ocasiones posteriores reduce su tiempo drásticamente, resultando en beneficios para el desempeño en general. El acceso a información que ya se encontraba en la memoria es conocido como *cache hit* y los demás son llamados *cache misses.*

Antes de ARMv6, las caché eran normalmente implementadas virtualmente direccionadas. Con este modelo, las páginas físicas eran mapeadas a una sola virtual o el resultado podría haber resultado impredecible, las copias virtuales de una sola página física no tenían coherencia entre sí. Luego de la mencionada versión, se especifica una arquitectura de caché en que el comportamiento esperado es el asociado con implementaciones físicas.

Su organización puede ser una arquitectura Harvard con caché de instrucciones y de datos separadas o Von Neumann con una sola memoria.

La caché L1 está diseñada para reducir los requerimientos de limpieza y validación, debe soportar múltiples alias de direcciones virtuales para una localidad de memoria en particular. Su nombre se debe simplemente a que es el nivel más cercano al CPU, el resto de sus características están presentes en las demás memorias del mismo tipo, incluyendo cada una capacidad para:

- Limpieza
- Invalidación

- Caché L2

Es el siguiente nivel más cercano al CPU, pero en el caso del nivel dos esta memoria puede ser:

  - Acoplada muy cercana al núcleo
  - Implementada como periféricos mapeados a memoria en el bus de sistema.
  - Es posible introducir otros niveles de caché siempre que cumplan con las características de las demás y tengan coherencia en el sistema. Entre las posibles variantes de esto se distinguen por las implementaciones de:
  - Tamaño de caché y asociación: unificadas (Von Neumann) o separadas (Harvard).
  - Cómo se maneja la obtención de instrucciones: *write-through* y *write-back*.
  - Cómo se maneja la escritura de datos: asignación por lectura y asignación por escritura.

- Buffer de escritura

El término sirve para describir un desacople entre una transacción de escritura y la ejecución de transacciones posteriores.

Es un bloque de memoria de alta velocidad con el propósito de optimizar los almacenamientos en memoria principal. Cuando esto ocurre, los datos son escritos en el *buffer* a una velocidad alta y este completa luego el almacenamiento a la velocidad de la memoria principal. Para mientras, el procesador puede ejecutar adicionales a máxima velocidad.

Las caché y los *buffer* de escritura son un recurso usual en ARM cuando se trata de mejorar la eficiencia del sistema. Las frecuencias de reloj del núcleo se han incrementado hasta ser mayores a la velocidad con que se accede a memoria. Este factor, junto a la reducción de espacio ocupado por silicio fomentan el uso de caché para hacer frente a demandas de sistema crecientes; sin embargo, el costo de implementar memorias muy cercanas al núcleo sigue siendo alto.

Estos tipos de memoria, introducen algunos potenciales problemas principalmente causados por:

- Accesos a memoria ocurriendo en momentos distintos a los que el programador espera.
- La existencia de múltiples localidades físicas donde la información puede ser almacenada.

2.3.6.2. TCM (*Tightly Coupled Memory*)

Diseñada para proveer memoria de baja latencia que pueda ser utilizada por el procesador sin la falta de previsibilidad propia de las caché. Es capaz de soportar rutinas críticas como interrupciones o tareas de tiempo real, donde la indeterminación de la caché sería altamente indeseable.

Es posible también contener cuatro bancos de datos y cuatro de instrucciones correspondientes a TCM, cada uno programado para estar en una ubicación distinta en el mapa físico de memoria.

2.3.6.3. VMSA (*Virtual Memory System Architecture*)

Los sistemas operativos complejos usualmente implementan un sistema de memoria virtual, para asegurar espacios de direcciones separados y protegidos para diferentes procesos.

Los procesos se distribuyen dinámicamente en la memoria junto a otros recursos de sistema mapeado a memoria bajo el control de la MMU (*Memory Management Unit*). Esta permite un control detallado de un sistema de memoria a través de mapeos de direcciones virtuales a físicas, y propiedades de memoria asociadas contenidas en una o más estructuras llamadas TLB (*Translation Lookaside Buffers*).

Los contenidos de los TLB son manejados a través de traducciones por hardware desde tablas contenidas en memoria como diccionarios, un recorrido completo de estas búsquedas es llamado *table walk* con un costo en tiempo de ejecución por acceso a memoria principal, pero con una solución parcial dejando en caché los resultados de búsquedas ya realizadas y así evitar que nuevas solicitudes impliquen una *table walk* completa.

2.3.6.4. PMSA (*Protected Memory System Architecture*)

Basada en MPU, es la implementación de un esquema considerablemente más simple de protección de memoria que MMU. La simplicidad aplica hardware y software.

La principal modificación es que el MPU no usa tablas de traducción; en cambio, los registros de CP15 son utilizados para definir regiones protegidas, eliminando la necesidad de hacer traducciones completas por hardware y de mantener actualizadas las tablas por software. Esto tiene la ventaja de hacer al chequeo de memoria totalmente determinístico; sin embargo el nivel de control ya no es tan fino dado a que se trabaja sobre regiones y no páginas.

La segunda modificación consiste en la eliminación de soporte para mapeo de direcciones virtuales y físicas, estas serán siempre iguales.

En los diseños PMSA son distinguibles las siguientes características:

- La memoria está dividida en regiones
- El control de región de memoria (lectura y escritura) está permitido únicamente para modos privilegiados.
- Si una dirección está definida en múltiples regiones, se utiliza un esquema de prioridad donde la región con un número mayor es preferida.
- El acceso a una dirección no definida causa un aborto de memoria.
- Todas las direcciones son físicas, la traducción no es posible.
- Soporte para espacios de instrucción y datos unificados (von Neumann) o separados (Harvard).

2.3.6.5. FCSE (*Fast Context Switch Extension*)

Modifica el comportamiento del sistema de memoria, permitiendo que múltiples programas corriendo en el procesador usen rangos de direcciones idénticos asegurando que las que presentan al resto del sistema sean distintas.

Cuando se intercambia funciones entre dos procesos por software con rangos de direcciones traslapadas, normalmente se requiere que se hagan cambios en el mapeo de direcciones virtuales y físicas definidas en las tablas de paginación del MMU, causando que el contenido en caché y TLB se vuelva inválido demandando una limpieza. Como resultado, cada intercambio de proceso provoca una saturación por limpieza y reescritura de cada página.

El FCSE implica, entonces, una solución presentando distintas direcciones al resto del sistema aun cuando los procesos usan representaciones idénticas.

2.3.6.6. Extensión de seguridad TrustZone

Esta tecnología que provee aislamiento por hardware para software confiable, estando disponible para Cortex-A y los sistemas basados en Armv8-M. Es utilizada en aplicaciones que requieran confidencialidad para protección de información de alto valor como autenticación, transferencias monetarias, credenciales y cualquier activo que necesite ser protegido de ataques con flexibilidad, para que los diseñadores especifiquen a cada SoC las características que sean apropiadas para su sistema.

La seguridad se alcanza seccionando los recursos de hardware y software del sistema y asignándolos al conjunto del subsistema no seguro o al seguro; así, la tecnología TrustZone se utiliza para proteger firmware, periféricos, entradas, salidas, aislamiento para arranque seguro, actualizaciones confiables e implementaciones seguras que, a la vez, provean respuestas determinísticas. Incluso las operaciones de depuración están dotadas con sensibilidad para acceso a estados seguros y no seguros.

TrustZone hace posible que un solo procesador ejecute código del modo normal y seguro eficiente y simultáneamente, evitando la necesidad de implementar en silicio un procesador dedicado únicamente a la seguridad.

Cada procesador construido físicamente con TrustZone incluido provee dos núcleos virtuales: uno considerado seguro y otro no seguro, con un mecanismo robusto para conmutar entre ambos. El procesador virtual no seguro puede solo acceder a recursos del sistema en el grupo sin requerimientos de seguridad, mientras el procesador virtual seguro puede acceder a todos los recursos.

Sumado a las demás características, la seguridad se extiende al nivel de sistema con el control de interrupciones y excepciones permitiendo que a cada interrupción se les pueda asignar atributos de seguridad y que la información segura se restaure automáticamente en registros para prevenir fugas de información. Como resultado, TrustZone se vuelve una característica deseable en sistemas dirigidos al área de IoT.

Tabla V. **Resumen de características TrustZone**

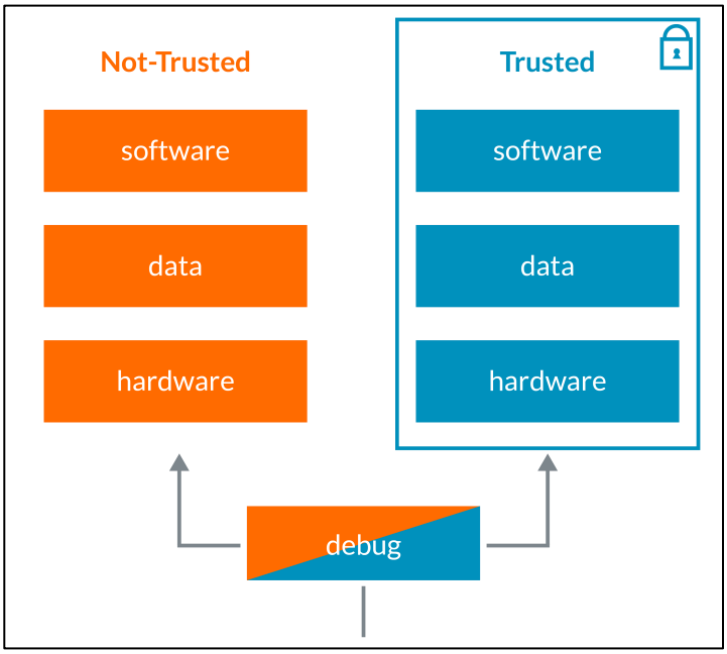
Característica	**TrustZone Armv8-A**	**TrustZone Armv8-M**
Estados adicionales	• Aplicaciones confiables • Sistemas operativos confiables. • Arranque confiable y firmware.	• Modo hilo seguro (instrucciones y datos confiables). • Modo controlador seguro (*drivers* confiables, RTOS, gestores de librerías).
Interrupciones seguras	Sí	Sí (aceleradas)
Transición de estado	Por software	Por hardware (acelerado)
Gestión de memoria	Memoria virtual con atributos de seguridad.	Unidad de atribuciones de seguridad (SAU) y particiones de memoria MPU.
Seguridad de interconexión de sistema.	Sí	Sí

Continuación de la tabla V.

Seguridad en instrucciones, datos y memoria	Sí	Sí
Arranque seguro	Sí	Sí

Fuente: Security on Arm: TrustZone. *Productos*. https://www.arm.com/products/security-on-arm/trustzone. Consulta: 8 de abril de 2018.

Figura 28. **Diagrama de partición TrustZone**



Fuente: Security on Arm. TrustZone. *Productos*. https://www.arm.com/products/security-on-arm/trustzone. Consulta: 8 de abril de 2018.

A pesar de que fue pensada inicialmente para Cortex-A, la ventaja principal de que TrustZone se extienda a Armv8-M es que la protección se presenta en el mercado para todos los rangos de costo. En una implementación

típica, el diseño será pensado para que el código dentro del área segura se mantenga en particiones tan pequeñas como sea posible y así, reducir el área vulnerable a ataques.

Un punto importante de introducir esta tecnología en el área de sistemas embebidos y microcontroladores está en que durante los últimos años se ha disparado la inversión de esfuerzos en el campo de IoT, haciendo que los sistemas se vuelvan más sofisticados y la demanda se incremente, por eso los diseñadores de software necesitan sistemas seguros pero al mismo tiempo, fáciles de implementar.

La forma tradicional de lidiar con la seguridad es haciendo dos particiones por software (una sin privilegios y otra con ellos, utilizando MPU), esto suele ser útil para varios dispositivos IoT; sin embargo una sola vulnerabilidad en el código del área privilegiada puede permitir el acceso no autorizado al sistema completo. En sistemas donde la seguridad resulte crítica, TrustZone extiende las posibilidades a los estados seguro y no seguro:

- Estado no seguro para aplicaciones regulares
- Estado seguro para componentes de software y recursos de seguridad como unidades de almacenamiento seguro, aceleradores criptográficos, TRNG (*True Random Number Generator*), entre otros.

La arquitectura Armv8-M introduce chequeo de límite de pila (para evitar errores por desbordamiento), y mejoras en el diseño de MPU.

2.3.7. Arquitectura de bus AMBA (*Advanced Microcontroller Bus Architecture*)

AMBA es una especificación desarrollada por Arm para conexión y control de bloques en sistemas integrados en un solo chip, y resulta útil en diseño de arquitecturas de buses para control multinúcleo o de múltiples elementos como GPU y procesadores de señales.

Los objetivos de AMBA incluyen mantener estándares independientes de la tecnología, alentar la minimización de área de silicio utilizada para implementación con el fin de mantener bajo consumo de potencia y promover el diseño de sistemas modulares para aumentar las posibilidades de varias implementaciones de propiedad intelectual y librerías.

Las cinco versiones de AMBA hasta la fecha han liberado distintos estándares:

- ASB (*Advanced System Bus*) y APB (*Advanced Peripheral Bus*): las primeras arquitecturas de buses de alto rendimiento y baja potencia respectivamente.
- AHB (*AMBA High-performance Bus*): protocolo de alta frecuencia de ejecuciones en un solo flanco de reloj. La versión AHB5 es la más utilizada en el perfil de microcontroladores y otros sistemas de baja latencia.
- AXI (*Advanced Extensible Interface*): para frecuencias más altas que las soportadas por AHB. Actualmente en su versión AXI5.
- ATB (*Advanced Trace Bus*): parte del hardware de depuración y rastreo.
- ACE (*AXI Coherence Extension*): para uso con AXI en sistemas multinúcleo con el fin de mantener coherencia de caché.

- CHI (*Coherent Hub Interface*): con mejoras para transmisiones a alta velocidad sobre protocolos anteriores y características para reducción de congestión.
- DTI (*Distributed Translation Interface*): para uso con la unidad de gestión de memoria (MMU).
- GFB (*Generic Flash Bus*): para conexión simple entre el sistema y la memoria de almacenamiento no volátil.

3. PERFIL CORTEX-M

En el mercado de los procesadores puede encontrarse, como se ha expuesto con anterioridad, demandas distintas según las necesidades de aplicación.

Es común encontrar requerimientos de muy alto rendimiento para los que el tamaño es una preocupación pero llega a ser una prioridad secundaria comparada con las características que hagan al sistema sumamente poderoso. Por otro lado, puede encontrarse situaciones en las que no se necesita un procesamiento tan complejo porque la aplicación es sencilla pero comienza a ser un objetivo principal consumir la menor cantidad de silicio en el chip para reducir costos, tamaño y reducir drásticamente el consumo de potencia.

Para responder a estas últimas características, los procesadores del perfil M en la familia Cortex de ARM están diseñados con pocas etapas de pipeline y otros atributos que los hacen (especialmente), fáciles de manejar y son populares en el mercado de microcontroladores y sistemas embebidos.

A pesar de ser los más sencillos en la escala de los procesadores ARM, los Cortex-M suelen ser más versátiles que arquitecturas anteriores. Ejemplo de esto son Cortex-M4 y Cortex-M7 (arquitecturas a ser explicadas posteriormente en este capítulo), con un alto rendimiento dentro del grupo.

Las características de Cortex-M incluyen algunos rasgos que las diferencian de las arquitecturas anteriores ARM:

- Aparición de implementaciones líderes en la industria con bajo consumo de potencia y restricciones de área, esto hace a los diseños óptimos para sistemas embebidos.
- *Pipeline* simplificado.
- Solamente cuentan con soporte para instrucciones Thumb de ARM, con extensión para 16 y 32 bits en Thumb2.
- El control de interrupciones es manejado por el NVIC (*Nested Vector Interrupt Controller*), con priorización automática, enmascarado y anidado de interrupciones.
- Operación altamente determinística: Ejecución en uno o pocos ciclos de reloj, operaciones sin caché, controles de interrupción que pueden ser escritos como funciones estándar de lenguaje C y C++. Al mismo tiempo, las respuestas cuentan con baja latencia.
- Depuración y soporte para análisis de rendimiento de software para sistemas dirigidos por eventos.

Estas propiedades permiten a los desarrolladores aprovechar mayor cantidad de características en menor tiempo, a menor costo, con conectividad versátil, seguridad estándar y técnicas recientes para aumento de la eficiencia de la energía.

3.1. Campo de aplicación

Para comprender por qué el mercado de microcontroladores y sistemas embebidos es tan importante, es necesario estudiar las características de estos dispositivos y las aplicaciones en las que se enfocan.

3.1.1. Microcontroladores

Para virtualmente cualquier persona que se dedica el diseño de sistemas digitales es fundamental conocer la existencia de estos dispositivos. Por su simpleza relativa con chips más avanzados es la elección por excelencia en la academia con temas relacionados a electrónica básica.

Tabla VI. **Definición de microcontrolador**

Grupo de circuitos integrados que ejecutan instrucciones reprogramables almacenadas en memoria para desempeñar una tarea específica. Se componen de un CPU, memoria y periféricos (entradas, salidas, conversores).

Fuente: elaboración propia.

Los microcontroladores comenzaron su historia en 1971 con la invención del primer chip a cargo de Texas Instruments TMS 1000 (microcontrolador de 4 bits y funciones ROM y RAM). El diseño fue inicialmente utilizado internamente y luego se comercializó abiertamente.

Alrededor de los mismos años, Intel comenzaba a trabajar en sus microprocesadores y luego de eso, produjo también sus primeros microcontroladores (8048, 8051 y 8040).

Aunque al inicio eran dispositivos demasiado sencillos e incapaces de ejecutar tareas muy complejas, poco a poco los fabricantes se enfocan más en generar microcontroladores mucho más eficientes, especializados e incluso pequeños para su uso en aplicaciones que requieran bajo requerimiento de energía y consumo eficiente de recursos en un solo chip.

Entre las empresas que han trabajado la producción de microcontroladores se encuentran Atmel y Microchip, ambos con infinidad de tarjetas de desarrollo en el mercado que se suman a las basadas en diseños del ya mencionado Texas Instruments.

3.1.2. Sistemas embebidos

Se ha mencionado con anterioridad en este texto que los procesadores de bajo consumo son ampliamente utilizados en sistemas embebidos por su portabilidad y eficiencia en el uso de recursos. A pesar de que estos se encuentran en muchas formas y la línea de separación con otros tipos de sistemas sea difusa, su definición principal se describe en la Tabla a continuación.

Tabla VII. **Definición de sistema embebido**

Sistema integrado en un dispositivo, dedicado a una tarea específica (o a una pequeña cantidad de ellas).

Fuente: elaboración propia.

Las características que identifican a un sistema embebido son principalmente:

- Tiene una función especializada y la repite constantemente
- Está fuertemente restringido a requerimientos de costo, tamaño, potencia y rendimiento.
- Debe reaccionar a eventos y cumplir características de tiempo real en sus respuestas.
- Debe estar basado en un microprocesador.

- Cuenta con unidades de memoria.
- Debe contar con periféricos para conectar entradas y salidas.
- Está formado por software y hardware.

Las especificaciones de bajo costo e implementación de todos los elementos en un solo chip con las que cuentan los microcontroladores los hace una opción económica y eficiente para ser utilizada en sistemas embebidos. Actualmente estos elementos se encuentran en muchísimas aplicaciones de la vida cotidiana como el hogar y los automóviles.

Para los microcontroladores con mayor rendimiento, suele utilizarse el término *System-on-a-Chip* (SoC), y se encuentran dispositivos con implementaciones, incluso multinúcleo.

La mayoría de sistemas embebidos se caracterizan por ser muy simples; sin embargo, también existen aplicaciones complejas como el caso de sistemas de aviación y de seguridad que superan en velocidad o exactitud de decisión a la capacidad humana.

Los fabricantes dedicados a producción de sistemas embebidos incluyen a Apple, IBM, Intel, Texas Instruments y muchas otras empresas; todos basándose mayormente en diseños de ARM.

Tomando en cuenta el campo para el que está dirigida la arquitectura Cortex-M, es mucho más significativo considerar las características de cada una de sus variantes, porque todas cumplirán lineamientos que las hagan eficientes para las aplicaciones a las que se destinan. Algunas de las características compartidas son:

- Uso de NVIC para manejo de interrupciones
- Modos de suspensión definidos por arquitectura: suspensión y suspensión profunda.
- Características de soporte de sistema operativo.
- Soporte para depuración.
- Facilidad de uso.

Es más acertado; sin embargo, considerar estos y otros componentes de las arquitecturas en grupos como se presenta en las subsecciones a continuación. Para tales fines, a partir de este punto se considera que hasta la fecha, los dispositivos Cortex-M cuentan con una clasificación como la expuesta en la tabla. Los detalles de cada grupo se explican más adelante.

Tabla VIII. **Grupos de arquitecturas Cortex-M**

Arquitectura	Diseños	
Armv6-M	Cortex-M0 Cortex-M0+ Cortex-M1	
Armv7-M (Armv7E-M si incluye DSP)	Cortex-M3 Cortex-M4 Cortex-M7	
Armv8-M	Subperfil base	Subperfil principal
	Cortex-M23	Cortex-M33

Fuente: elaboración propia.

3.2. Características de la arquitectura Cortex-M

Por la sencillez de las arquitecturas destinadas a microcontroladores, es usualmente prudente comenzar el análisis de procesadores en el grupo dedicado a ellos para luego avanzar hacia conceptos más complejos, como se verá en el Cap[itulo 4.

3.2.1. Modelo del programador

El perfil M de la familia Cortex comparte una gran parte de sus características esenciales por estar basada en la arquitectura principal de ARM aunque claro, con ciertas variaciones según el propósito para el que está pensado un diseño en especial. Un ejemplo de esto es que los registros de propósito general (desde R0 a R15), se encuentran disponibles en todos los procesadores, así como los registros de estado de programa en donde se encuentren disponibles por implementación como el caso del FPSCR (registro de estado del coprocesador de punto flotante), para Cortex-M4/M7/M33. También existen dos registros especiales llamados FAULTMASK y BASEPRI (ambos para control de interrupciones) que se encuentran disponibles únicamente en Cortex-M3/M4/M7/M33.

Tabla IX. **Registros ARM disponibles en procesadores Cortex-M**

Registros Cortex-M					
Generales	**Unidad de Punto Flotante**			**Especiales**	
R0	**SP**		**DP**	**Nombre**	**Función**
R1	S1	S0	D0	xPSR	Registros de estado de programa
R2	S3	S2	D1		
R3	S5	S4	D2	PRIMASK	Registros de máscaras de interrupción.
R4	S7	S6	D3		
R5	S9	S8	D4	FAULTMASK	
R6	S11	S10	D5	BASEPRI	
R7	S13	S12	D6	CONTROL	Registro de control
R8	S15	S14	D7		
R9	S17	S16	D8		
R10	S19	S18	D9		
R11	S21	S20	D10		
R12	S23	S22	D11		
R13 (SP)	S25	S24	D12		
R14 (LR)	S27	S26	D13		
R15 (PC)	S29	S28	D14		
	S31	S30	D15		
	FPSCR Registro de estado de punto flotante				

	Precisión simple (SP)
	Precisión doble (DP)
	Registros no disponibles en Armv6-M

Fuente: elaboración propia.

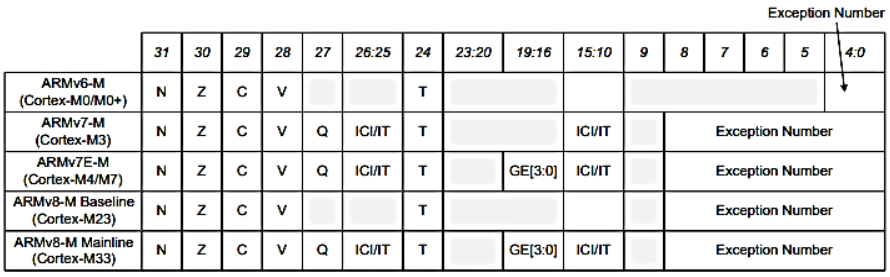
Una característica del modelo de programador en los Cortex-M es que el PSR (*Program Status Register*) se encuentra subdividido en:

- PSR de aplicación (APSR) sin bit Q en ARMv6-M y subperfil base de ARMv8-M.
- PSR de ejecución (EPSR) sin bits ICI/IT en ARMv6-M y subperfil base de ARMv8-M.
- PSR de interrupción (IPSR) con número de interrupciones menor en ARMv6-M.

Cada uno de los tres registros enlistados cumple la función de proveer información acerca del estado del programa, con las características especiales para las que están destinados.

Es importante considerar que los bits GE están disponibles en ARMv7E-M y subperfil principal de ARMv8-M, contando ambas arquitecturas con extensión DSP. Es peculiaridad del perfil Cortex-M contar con los bits ICI para información de instrucciones *store* y *load* multiciclo por excepción y los bits IT, para ejecución condicional de las instrucciones y que pueda volverse al punto correcto de código luego de una interrupción.

Figura 29. **Forma general de los registros de estado de Cortex-M**



Fuente: YIU, Joseph. *White Paper. ARM Cortex-M for Beginners. An overview of the ARM Cortex-M processor family and comparison*. p. 12.

3.2.2. Control de fallas

La capacidad de control de fallas de los procesadores ARM es una característica destacada que los distingue de otras arquitecturas de microcontroladores.

El funcionamiento se basa en la detección de las fallas y activación de las excepciones correspondientes para que el software tome las acciones necesarias.

El control de fallas permite que los sistemas reaccionen mucho más rápido a distintos problemas que otros que se quedan tildados hasta ser reiniciados por temporizadores (como los *watchdog timers*).

Tomando en cuenta que las interrupciones se dan por niveles de prioridad (con preferencia a números menores), dentro de los Cortex-M se encuentran las siguientes especificaciones:

- En la arquitectura Armv6-M, todos los eventos de falla activan el controlador *HardFault* con prioridad -1, que es mayor a todas las excepciones programables, pero justo debajo de las interrupciones no enmascarables.
    - *HardFault*: mecanismo de excepción por defecto para todas las situaciones no contempladas en otros. Es usual su uso en casos de fallas que provoquen que el sistema se vuelva irrecuperable.

- El subperfil base Armv8-M es similar a v6, pero con prioridades -1 o -3 para el controlador *HardFault* con TrustZone.
- Para Armv7-M y subperfil principal de Armv8-M existen varios controles de fallas adicionales al *HardFault*.

- MemManage: controla protección de memoria relacionada con situaciones determinadas por la MPU o restricciones de protección de memoria fija tanto para instrucciones como para datos.
- Falla de bus: funciona como complemento del MemManage, con fallas usualmente generadas en los buses del sistema. Tiene una prioridad configurable.
- Falla de uso: maneja por prioridad configurable fallas causadas por ejecución de instrucciones no relacionadas con memoria. Las situaciones que pueden causar este tipo encierran: instrucciones indefinidas, estado inválido de ejecución de instrucciones, errores en retorno de excepción, acceso deshabilitado o no disponible a coprocesadores, división en cero y acceso desalineado a memoria. Este grupo de excepciones puede ser deshabilitado y en dado caso, el control se moverá a *HardFault.*
- *SecureFault:* para manejo de violaciones de seguridad en la extensión de seguridad TrustZone.
- Monitor de depuración: excepción síncrona categorizada como falla. Aquí los *watchpoints* se comportan como interrupciones, con prioridades configurables.

Las excepciones pueden tanto ser modificadas en prioridad como escaladas por el registro FAULTMASK para alcanzar el nivel de HardFault, si fuera necesario.

Diversos depuradores comerciales incluyen características para diagnosticar eventos de falla usando los FSR (Fault State Registers). Algunos incluso trabajan con el controlador de fallas para sugerir acciones de reparación.

Tabla X. **Resumen control de fallas por arquitectura**

	Armv6-M y subperfil base de Armv8-M	**Armv7-M y subperfil principal de Armv8-M**
HardFault	x	x
MemManage	-	x
Falla de bus	-	x
Falla de uso	-	x
SecureFault	-	x
FSR	solamente de depurador	solamente para subperfil principal de Armv8-M

x: presente
-: ausente

Fuente: YIU, Joseph. *White Paper. ARM Cortex-M for Beginners. An overview of the ARM Cortex-M processor family and comparison*. p. 18.

3.2.3. Control de interrupciones por NVIC

El *Nested Vectored Interrupt Controller* (NVIC), es un controlador de interrupciones y fallas (definidas en el inciso 3.1.2), con acceso por registros en bloques de memoria establecidos, este se encuentra incluido en todos los dispositivos Cortex-M junto con el modelo de excepciones.

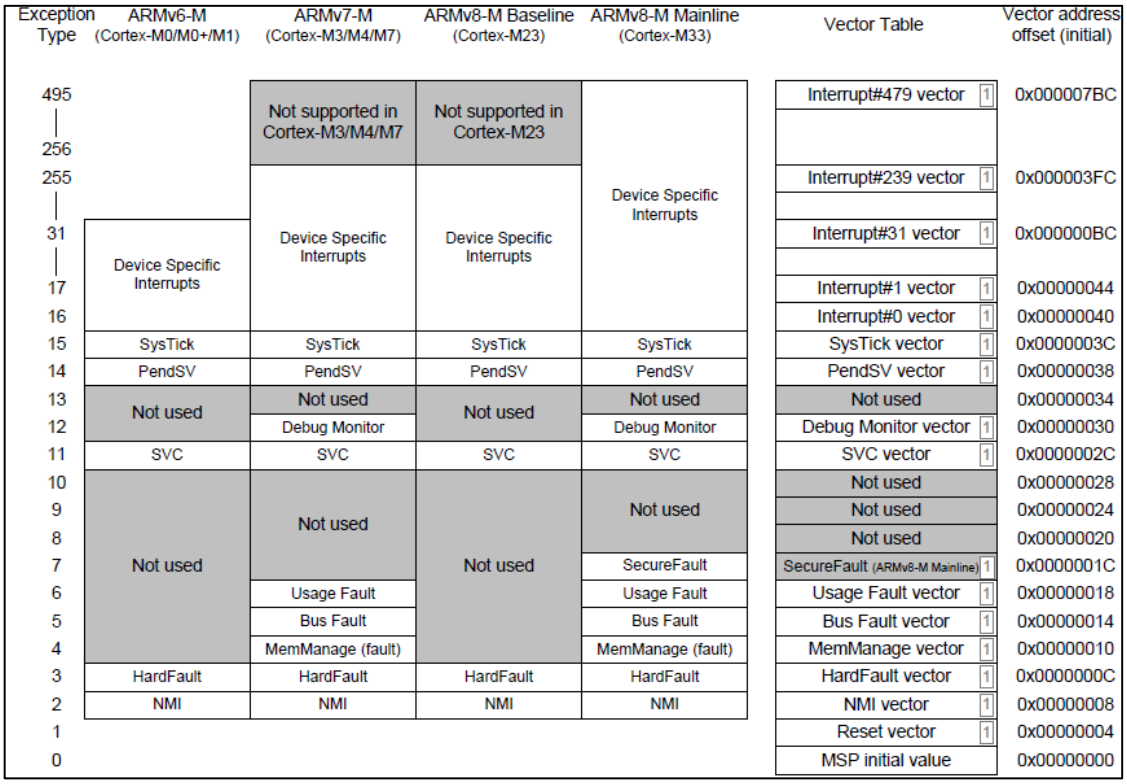
El soporte de este diseño incluye interrupciones con valores de prioridad y número de evento configurables, así como sensibilidad a pulsos y a nivel de señal (activando la *Interrupt Service Routine* o ISR).

La extension de seguridad opcional TrustZone permite que los niveles de prioridad de interrupciones no seguras sean modificados por software seguro y que algunos eventos seguros siempre tengan prioridad mayor a los no seguros.

Existe un espacio de memoria para control de excepciones llamado tabla de vectores, en ella se almacenan las direcciones base de los controladores. La dirección de inicio de esta tabla se define por el registro VTOR (*Vector Table Offset Register*), con las condiciones enlistadas para cada arquitectura:

- Cortex-M0+/M3/M4 tienen la tabla de vectores localizada en el inicio del mapa de memoria (dirección 0x00000000).
- Para Cortex-M7/M23/M33 cuentan con un valor por defecto de VTOR definido por el diseñador. Los procesadores de Armv8-M pueden contar con dos tablas de vectores (una por cada modo de seguridad).
- Cortex-M0 y M1 no implementa VTOR, la dirección de la tabla de vectores siempre es 0x00000000.
- En Cortex-M0+ y M23 el VTOR es opcional.

Figura 30. **Modelo de excepciones de arquitecturas Cortex-M**



Exception Type	ARMv6-M (Cortex-M0/M0+/M1)	ARMv7-M (Cortex-M3/M4/M7)	ARMv8-M Baseline (Cortex-M23)	ARMv8-M Mainline (Cortex-M33)	Vector Table	Vector address offset (initial)
495		Not supported in Cortex-M3/M4/M7	Not supported in Cortex-M23	Device Specific Interrupts	Interrupt#479 vector [1]	0x000007BC
256						
255		Device Specific Interrupts	Device Specific Interrupts		Interrupt#239 vector [1]	0x000003FC
31	Device Specific Interrupts				Interrupt#31 vector [1]	0x000000BC
17					Interrupt#1 vector [1]	0x00000044
16					Interrupt#0 vector [1]	0x00000040
15	SysTick	SysTick	SysTick	SysTick	SysTick vector [1]	0x0000003C
14	PendSV	PendSV	PendSV	PendSV	PendSV vector [1]	0x00000038
13	Not used	Not used	Not used	Not used	Not used	0x00000034
12		Debug Monitor		Debug Monitor	Debug Monitor vector [1]	0x00000030
11	SVC	SVC	SVC	SVC	SVC vector [1]	0x0000002C
10	Not used	Not used	Not used	Not used	Not used	0x00000028
9					Not used	0x00000024
8					Not used	0x00000020
7				SecureFault	SecureFault (ARMv8-M Mainline) [1]	0x0000001C
6		Usage Fault		Usage Fault	Usage Fault vector [1]	0x00000018
5		Bus Fault		Bus Fault	Bus Fault vector [1]	0x00000014
4		MemManage (fault)		MemManage (fault)	MemManage vector [1]	0x00000010
3	HardFault	HardFault	HardFault	HardFault	HardFault vector [1]	0x0000000C
2	NMI	NMI	NMI	NMI	NMI vector [1]	0x00000008
1					Reset vector [1]	0x00000004
0					MSP initial value	0x00000000

Fuente: YIU, Joseph. *White Paper. ARM Cortex-M for Beginners. An overview of the ARM Cortex-M processor family and comparison*. p. 14.

3.2.4. Soporte de sistema operativo

Las arquitecturas Cortex-M están diseñadas tomando en cuenta el soporte para sistemas operativos. Estas características incluyen:

- Puntero de pila sombreado (definición en Tabla XI)
- Excepciones SVC para activar servicios privilegiados
- Excepciones PendSV
- Temporizador SysTick, opcional en Cortex-M0/0+/23

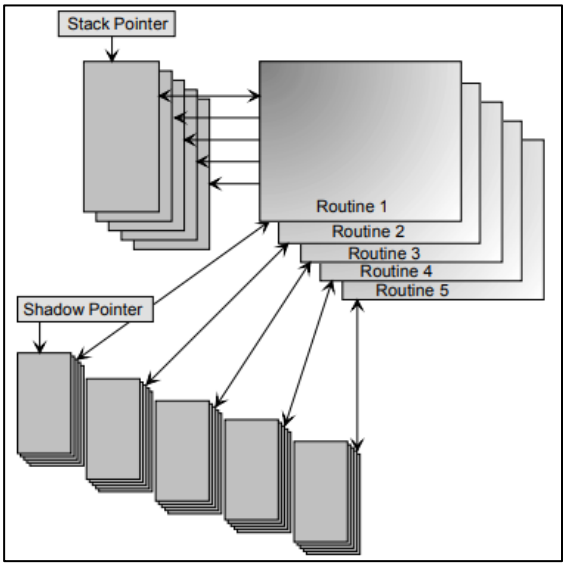
- Nivel de ejecución no privilegiado y MPU en Cortex-M0+/M3/M4/M7/M23/M33.

Tabla XI. **Técnica de sombreado**

Método para crear rutinas de compuertas de propósito general simuladas, sin la complicación de llamadas a subrutinas. Para ejecuciones en donde el puntero de pila (*stack pointer*) cumple su función normal y es aplicada esta técnica, un segundo registro llamado puntero sombreado (*shadow pointer*) se usa para proveer parámetros adicionales y locales variables para distintos segmentos del programa.

Fuente: elaboración propia.

Figura 31. **Rutinas y punteros sombreados**



Fuente: MAURER, Peter. *Chapter 5.j Component-Level Programming*. p. 3.

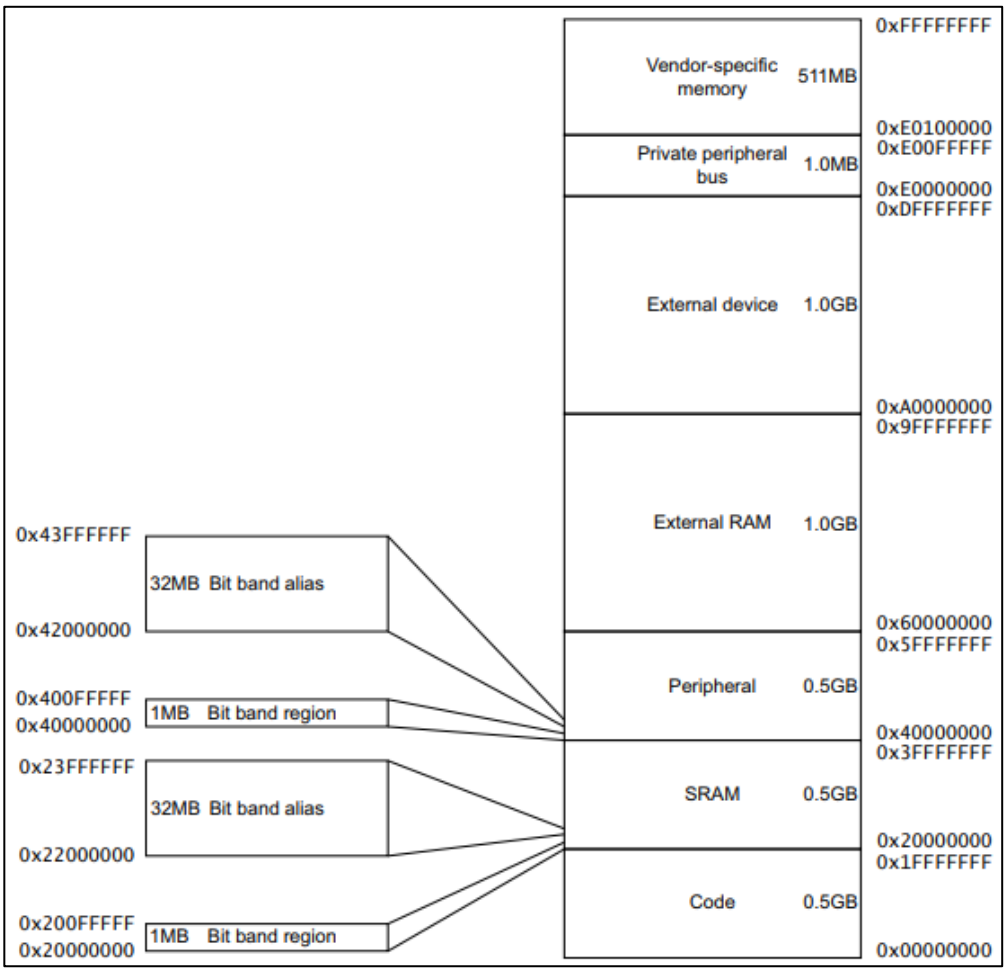
Las características de soporte de sistema operativo son opcionales en Cortex-M1 para lograr que su tamaño se adecúe a dispositivos de lógica programable muy pequeños.

La robustez del sistema se aumenta en Cortex-M0+/3/4/7/23/33 con tareas que pueden ser ejecutadas en niveles no privilegiados y MPU opcional para prevenir violaciones al acceso de memoria.

3.2.5. Mapa de memoria

Los procesadores ARM del perfil Cortex-M tienen un mapa de memoria fijo que provee 4GB de direcciones de memoria, su estructura se muestra a continuación:

Figura 32. **Mapa de memoria Cortex-M**



Fuente: Arm Ltd. *Cortex-M3 Devices Generic User Guide*. p. 25.

El modelo de distribución de memoria y el MPU divide los recursos en regiones, contando cada uno con un tipo de los enlistados:

- Normal: el procesador es capaz de reordenar transacciones para incremento de eficiencia o realizar lecturas especulativas.
- Dispositivo: se conserva el orden de transacciones relativo a otras en dispositivo o del tipo fuertemente ordenado.

- Fuertemente ordenada: es conservado el orden de transacciones relativas a todas las otras.

Los requerimientos de ordenamiento significan que el sistema de memoria puede realizar acciones de *buffer* de escritura a memoria de dispositivo pero no hará lo mismo para memoria fuertemente ordenada.

La memoria, al igual que tipos, puede contar con atributos que incluyen:

- Compartible: la memoria de sistema provee información de sincronización entre maestros de buses en un sistema con otros, como es el caso del controlador DMA.
- *Execute Never* (XN): el procesador previene el acceso a instrucciones. Una excepción de falla se genera solamente en la ejecución de una instrucción en este tipo de regiones de memoria.

El resumen de memorias se presenta en la tabla XII, en ella se toman en cuenta los tipos de memoria y sus atributos como una especificación de la figura 32.

Tabla XII. **Detalles de mapa de memoria Cortex-M**

Rango de direcciones	Región de memoria	Tipo de memoria	Descripción
0x00000000-0x1FFFFFFF	Código	Normal	Región para código y a veces, datos
0x20000000-0x3FFFFFFF	SRAM	Normal	Región para datos. Incluye áreas *bit band* y *bit band alias*.
0x40000000-0x5FFFFFFF	Periféricos	Dispositivo	Incluye áreas *bit band* y *bit band alias*.
0x60000000-0x9FFFFFFF	RAM externa	Normal	Región ejecutable para datos

Continuación de la tabla XII.

0xA0000000- 0xDFFFFFFF	Dispositivo externo*	Dispositivo	Memoria de dispositivo externo
0xE0000000- 0xE00FFFFF	Bus de periféricos privados**	Fuertemente ordenada	Región en que se incluye el NVIC, temporizador de sistema y bloque de control de Sistema.
0xE0100000- 0xFFFFFFFF	Dispositivo	Dispositivo	Específico de implementación

* Atributo: compartible o no compartible.
** Atributo: compartible.

Fuente: Arm Ltd. *Cortex-M3 Devices Generic User Guide*. p. 27.

3.2.6. Temporizador SysTick

Esta característica de los Cortex-M consiste en un contador en decremento de 24 bits con un mecanismo de control flexible.

Algunas de las formas en que puede ser utilizado este contador son:

- Temporizador de RTOS con frecuencia programable e invoca una rutina SysTick.
- Temporizador de alarma de alta velocidad utilizando el reloj principal.
- Temporizador de señal o alarma de frecuencia variable.
- Contador simple, utilizando software para medir tiempo restante y utilizado.
- Control basado en el reloj interno utilizando tiempos faltantes.

El temporizador depende de cuatro registros para controlar su funcionamiento:

- Control para configurar reloj, habilitar contador, habilitar interrupción y determinar el estado del contador.
- Recarga de valor del contador, conocido como valor *wrap*.
- Obtención del valor actual de contador.
- Registro de valor de calibración, indicando un valor de carga preliminar necesario para un reloj de sistema a 100Hz (10 ms).

Es importante considerar que si el núcleo se encuentra en estado de depuración, el contador no incrementará.

Tabla XIII. **Resumen de registros de temporizador SysTick**

Dirección base	L/E	Valor de reinicio	Nombre	Función
0xE000E010	L/E	0x00000000	SYST_CSR	Control y estado
0xE000E014	L/E	NP	SYST_RVR	Recarga de valor
0xE000E018	L/E	NP	SYST_CVR	Valor actual
0xE000E01C	L	IMP	SYST_CALIB	Valor de calibración

L: lectura
E: escritura
IMP: definido por implementación
NP: no predecible

Fuente: SEAL, David. *ARMv7-M Architecture Reference Manual*. p. 509.

3.3. Características de sistema

Con el fin de presentar de forma ordenada los componentes fundamentales de las arquitecturas en el grupo Cortex-M, se presenta a continuación apartados explicando cada uno.

3.3.1. Modos y privilegios

El perfil de microcontroladores cuenta con soporte para dos modos:

- Controlador: se accede a él a partir de una excepción, solamente este modo puede ejecutar un retorno al programa.
- Hilo: activado por reinicio, se accede a él como resultado de retorno desde una excepción.

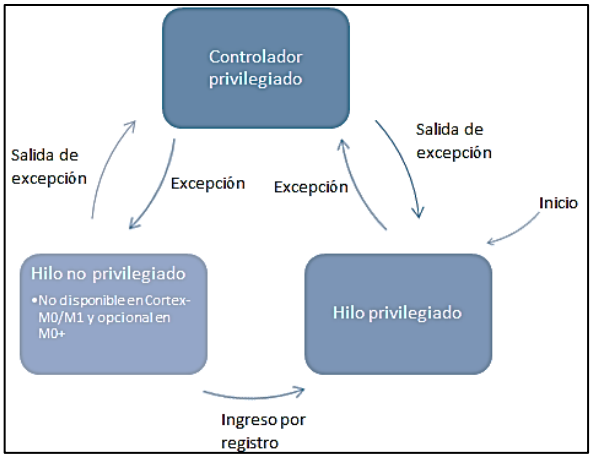
En complemento a los modos, las ejecuciones cuentan también con niveles de permisos de ejecución de código:

- Privilegiado: tiene acceso a todos los recursos
- No privilegiado: limita el acceso a ciertos recursos

Dada la dinámica de modos y privilegios, se distinguen dos punteros de pila según se requerimiento:

- Puntero de pila principal: disponible en modo hilo y controlador
- Puntero de pila de proceso: disponible en modo hilo

Figura 33. **Diagrama de estado de modos y privilegios en dispositivos Cortex-M**



Fuente: YIU, Joseph. *White Paper. ARM Cortex-M for Beginners. An overview of the ARM Cortex-M processor family and comparison*. p. 11.

3.3.2. Gestión de potencia

Son de bajo costo y simplicidad de uso, los dispositivos Cortex-M se caracterizan por su bajo consumo de potencia. Esto último se debe tanto a los diseños con circuitos reducidos como al soporte de baja potencia en la arquitectura con:

- Instrucciones WFI (Wait For Interrupt) y WFE (Wait For Event)
- Definiciones de modo de suspensión por arquitectura

Características extra de baja potencia en Cortex-M incluyen:

- Modos de suspensión y suspensión profunda

- Suspensión en salida (para aplicaciones dependientes de interrupciones).
- WIC (Wake-up Interrupt Controller).
- *Clock gating,* que permite que los relojes en registros y submódulos sean apagados para reducir el consumo.

3.3.3. *Single-cycle I/O*

Interfaz única en Cortex-M0+ y Cortex-M23, habilita en el procesador la capacidad de ejecutar tareas de entradas y salidas rápidamente.

En los demás procesadores, las interfaces de buses utilizan AHB Lite o AHB5, ambos protocolos que utilizan *pipeline* para operar a altas frecuencias de reloj. Esto tiene la desventaja de necesitar dos ciclos de reloj por instrucción. La interfaz *single-cycle* I/O lidia con esta dificultad añadiendo un bus simple sin *pipeline* que sirve de conexión, para un conjunto pequeño de periféricos específicos como GPIO, permitiendo junto con el resto de características de implementación simple, acceso mucho más rápido que en el resto de arquitecturas.

3.3.4. Bit banding

Característica desarrollada por Arm que mapea una palabra en alguna región de la memoria principal a un solo bit en la región bit band. Esto permite realizar operaciones de escritura y lectura en un solo ciclo de reloj. Usualmente útil en control de periféricos.

Para el perfil Cortex-M se tienen las siguientes consideraciones de implementación:

- Cortex-M3 y M4 tienen *bit band* opcional que permite dos rangos de direcciones (uno en SRAM y otro en periféricos).
- Cortex-M0, M0+ y M1 no cuentan con esta característica, pero puede ser añadida en nivel de sistema.
- Cortex-M7 no tiene soporte para *bit band* dado que las características de caché no pueden ser utilizadas con esta especificación.
- TrustZone en Armv8-M no tiene soporte para *bit band*, esto debido a que la atribución de alias en memoria podría comprometer regiones con atribuciones de seguridad distintas.

3.3.5. Depuración

La arquitectura de depuración en Cortex-M se construye sobre el diseño de depuración Arm *CoreSight*, que tiene soporte para múltiples sistemas y su configurabilidad permite adaptarlo a las necesidades de implementación. Por estas especificaciones, la interfaz de depuración y la de rastreo (*trace interface*), se encuentran desacopladas del procesador, propiciando que las conexiones con el resto del sistema estén especificadas por el diseñador de dispositivo aunque existen configuraciones sugeridas.

La conexión de depuración permite acceder a:

- Registros que controlan depurador y rastreador.
- Mapa de memoria. Para el caso específico en que se haga mientras el procesador está corriendo, se les llama accesos *on-the-fly*.
- Registros generales (solamente cuando el procesador está detenido).
- Historial de rastreo generado por el *Micro Trace Buffer* (MTB) en el caso de Cortex-M0+.
- Memoria flash para su programación.

Para todos los dispositivos de la familia de microcontroladores existe la opción de conectar el depurador por JTAG (como es tradicional), o por protocolo serial de depuración llamado *SerialWire*, que solamente necesita dos pines (ideal para diseños de conteo bajo de pines) y tiene la ventaja adicional de chequeo de paridad.

Por otro lado, la conexión de rastreo permite recolectar información acerca del programa en ejecución en tiempo real. Existen dos tipos:

- Puerto de rastreo, con varios pines de datos y señal de reloj
- Visor *SerialWire* (SWV), con un solo pin para rastreo de datos, eventos o instrumentación.

Distintos componentes permiten capturar información mientras el procesador se encuentra activo, estos son:

- ETM (*Embedded Trace Macrocell*): habilita la recopilación de historial de instrucciones ejecutadas.
- ITM (*Instrumentation Trace Macrocell*): permite que el software envíe mensajes (como impresiones en pantalla) y obtenga información por medio de la conexión de rastreo.
- DWT (Data Watchpoint and Trace): habilita el rastreo selectivo de información acerca de localidades de memoria, *profiling trace* (número de ciclos de reloj que el procesador tarda en ejecutar una tarea) y rastreo de eventos.

El rastreo se encuentra disponible en todos los Cortex-M, con excepción de M0 y M0+. En el caso del segundo, sin embargo, se cuenta con el

mecanismo MTB (Micro Trace Buffer), que permite obtener un historial breve de ejecuciones almacenando instrucciones en una pequeña parte de la SRAM. Esta característica se encuentra también en Armv8-M.

Tabla XIV. **Resumen de especificaciones sugeridas para implementación de depuradores en dispositivos Cortex-M**

	M0/M1	**M0+**	**M3/M4**	**M7**	**M23**	**M33**
Protocolo de conexión de depurador (JTAG/*SerialWire*)	Ambos	Ambos	Ambos*	Ambos	Ambos	Ambos
Protocolo de conexión de rastreador (puerto de rastreo/SWV)	-	-	Ambos	Ambos	Ambos	Ambos
Comparadores de *breakpoint* por hardware	Hasta 4	Hasta 4	Hasta 8	Hasta 8	Hasta 4	Hasta 8
***Breakpoints* por software**	Sí	Sí	Sí	Sí	Sí	Sí
Comparadores de *watchpoints* para datos	Hasta 2	Hasta 2	Hasta 4	Hasta 4	Hasta 4	Hasta 4
Rastreo de instrucciones	-	MTB	ETM	ETM	ETM/ MTB	ETM/ MTB
Rastreo de datos	-	-	DWT	DWT/ ETM	-	DWT
Rastreo de eventos y *profiling trace*	-	-	DWT	DWT	-	DWT
Rastreo de instrumentación	-	-	ITM	ITM	-	ITM
Accesos de memoria on-the-fly	Sí	Sí	Sí	Sí	Sí	Sí
Monitor de depuración	-	-	Sí	Sí	-	Sí
Sincronización de depuración en múltiples núcleos	Sí	Sí	Sí	Sí	Sí	Sí
Muestreo de contador de programa	Sí**	Sí**	Sí***	Sí***	Sí**	Sí***

* Las dos opciones pueden funcionar simultáneamente, el resto de arquitecturas trabaja con una opción a la vez.
** Por conexión de depuración
*** Por conexión de depuración y rastreo

Fuente: YIU, *Joseph. White Paper. ARM Cortex-M for Beginners. An overview of the ARM Cortex-M processor family and comparison*. p. 22.

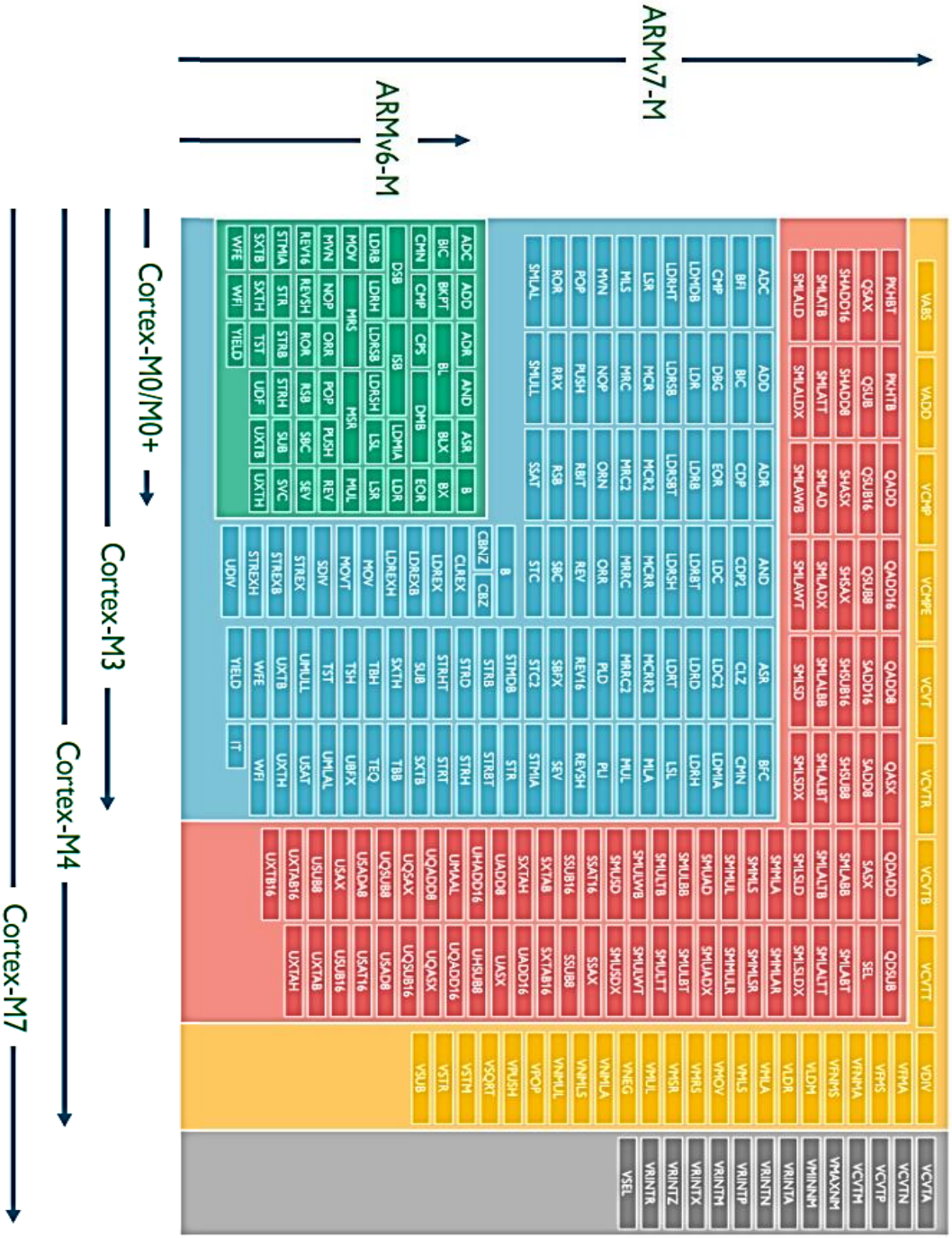
3.3.6. Sets de instrucciones

A pesar de que la mayoría de aplicaciones actualmente están programadas en lenguaje C (alguna alternativa de alto nivel), el estudio del set de instrucciones de los dispositivos en disponibilidad ayuda a comprender las características de los mismos y las tareas para las que son óptimos.

Los procesadores de la serie Cortex-M pueden dividirse por su ISA en los grupos detallados en tabla XIV.

El set de instrucciones se compone de los *opcodes* correspondientes a tareas que la arquitectura es capaz de ejecutar. Por esta razón, es de esperar que diseños complejos cuenten con un set de instrucciones más elaborado que los simples y a su vez, dentro de una misma versión de la arquitectura los procesadores serán compatibles hacia arriba (todos contendrán las instrucciones de sus antecesores y añadirán las propias). Esto es fácilmente apreciable en la imagen a continuación.

Figura 34. **Sets de instrucciones Armv7-M y Armv6-M**



Fuente: YIU, Joseph. *White Paper. ARM Cortex-M for Beginners. An overview of the ARM Cortex-M processor family and comparison*. p. 6.

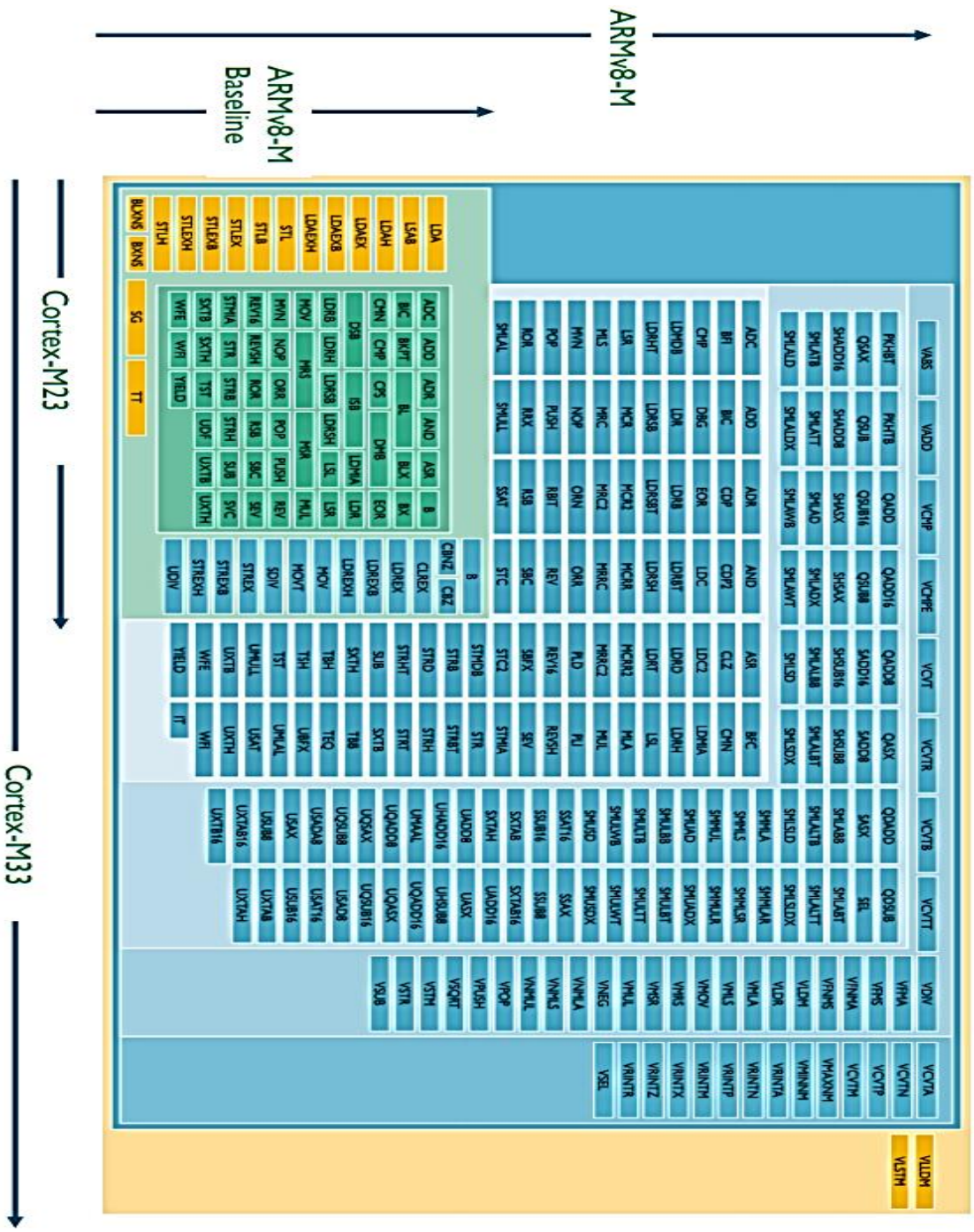
Para una mejor comprensión de la figura 34, es importante tomar en cuenta los grupos de colores. Aquí, el verde identifica a las instrucciones de procesamiento general de datos y tareas de control de entradas y salidas; el celeste, las de procesamiento avanzado de datos; el rosado, las instrucciones DSP y los últimos dos colores, las instrucciones de punto flotante.

Como fue mencionado anteriormente, las arquitecturas avanzadas (Cortex-M7, por ejemplo) podrán fácilmente adoptar código generado para diseños anteriores sin necesidad de modificaciones, porque las instrucciones están contenidas en sus capacidades, aplicando también para casos en que los dispositivos no contienen unidad de punto flotante pero se indica que debe usarse en un programa, en dado caso el compilador insertará las librerías necesarias para ejecutar las instrucciones por software. Todo esto sin embargo, no es posible de modo inverso (compatibilidad hacia abajo).

Al igual que Armv7, las arquitecturas de Armv8 construyen una estructura hacia arriba con instrucciones como se muestra en figura 35. Aquí estarán contenidos, entonces, Cortex-M23 con un despliegue sencillo y Cortex-M33 con soporte para procesamiento mucho más complejo.

Todas estas características serán expuestas más adelante individualmente para cada arquitectura del perfil. De igual forma, las instrucciones y su relación con el lenguaje de ensamblador tendrán un espacio para analizarse con más detalle; por el momento simplemente interesa la comparación general de los sets de instrucciones.

Figura 35. **Sets de instrucciones Armv8**



Fuente: YIU, Joseph. *White Paper. ARM Cortex-M for Beginners. An overview of the ARM Cortex-M processor family and comparison*. p. 7.

Tabla XV. **Resumen de características principales sets de instrucciones Cortex-M**

	M0/M0+	M1	M3	M4	M7	M23	M33
Multiplicación de un solo ciclo	x	x	x	x	x	x	x
Procesamiento a nivel de bits			x	x	x		x
División por hardware			x	x	x	x	x
Acceso a datos desalineados			x	x	x		x
Ejecución condicional			x	x	x		x
Comparación y saltos			x	x	x	x	x
Punto flotante				SP	SP/DP		SP
MAC			x	x	x		x
SIMD				x	x		x
Saturación			x	x	x		x
Acceso exclusivo			x	x	x	x	x
Barrera de memoria	x	x	x	x	x	x	x
TrustZone						x	x

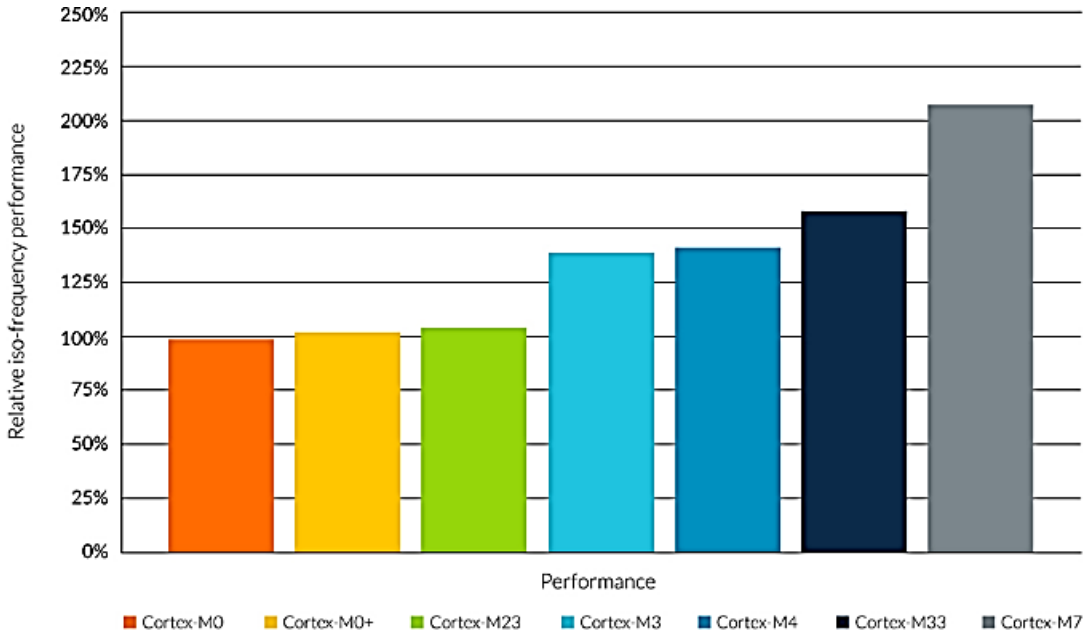
Fuente: YIU, Joseph. *White Paper. ARM Cortex-M for Beginners. An overview of the ARM Cortex-M processor family and comparison*. p. 10.

3.4. Subclasificaciones del perfil M

Como fue mencionado anteriormente, el rendimiento no es el único interés en la selección de procesadores. Existen aplicaciones en las que el consumo de potencia y el costo son críticos.

Por esta razón, el perfil Cortex-M se divide en varios diseños para suplir distintas necesidades. A continuación, se consideran las características que diferencian a cada arquitectura dentro de su grupo tomando en cuenta los aspectos expuestos en los apartados anteriores.

Figura 36. **Desempeño relativo de los procesadores Cortex-M**



Fuente: Cortex. *Processors Cortex-M Series*. https://www.arm.com/products/processors/cortex-m. Consulta: 23 de marzo de 2018.

En la figura 36 se distinguen tres grupos principales según su desempeño relativo en isofrecuencia:

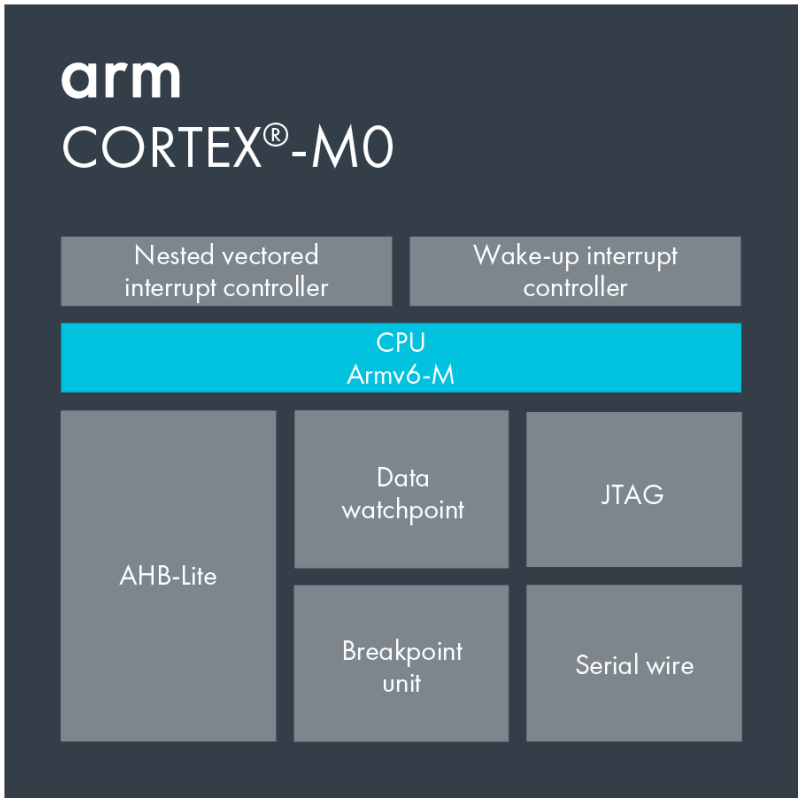
- Cortex-M0, M0+ y M23 están destinados a aplicaciones que requieran costo, consumo de potencia y área mínima.
- Cortex-M3, M4 y M33 se emplean en dispositivos que requieran balance entre las ventajas de desempeño y eficiencia energética.
- Cortex-M7 está hecho para aplicaciones embebidas que requieran alto desempeño.

Para comprender a mayor profundidad cada arquitectura, es fundamental hacer un estudio de cada una individualmente y así identificar las similitudes y características propias de cada una.

3.4.1. Cortex-M0

Procesador de tamaño altamente reducido (a partir de 12K compuertas), dirigido a microcontroladores y sistemas embebidos de costo y consumo de potencia ultra bajos.

Figura 37. **Componentes de arquitectura ARM Cortex-M0**



Fuente: Cortex. *Procesador Cortex-M0*. https://developer.arm.com/products/processors/cortex-m/cortex-m0. Consulta: 23 de marzo de 2018.

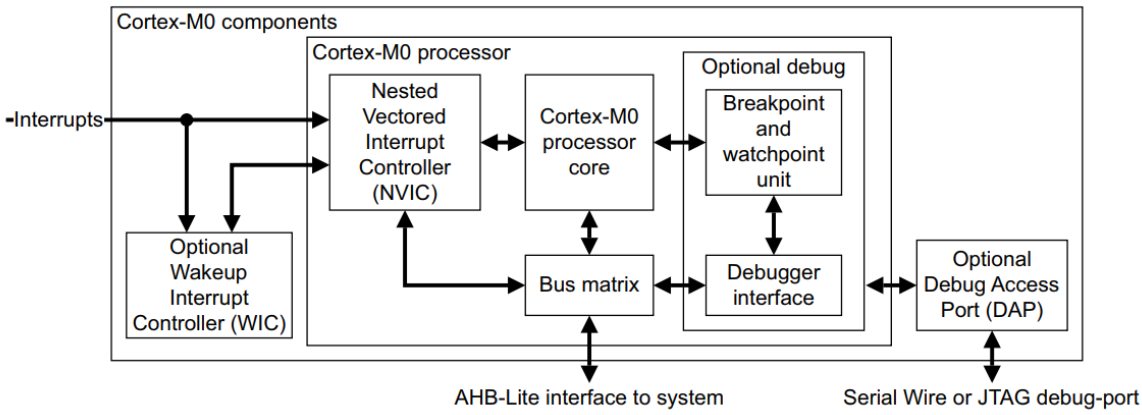
Este diseño tiene un *code footprint* muy bajo (espacio que ocupa el código almacenado en RAM), esto permite a los desarrolladores alcanzar desempeño digno de 32 bits a precios que pueden igualar a los procesadores de 8 bits y densidad de código mucho más alta que las otras familias. Con solamente 56 instrucciones se hace posible manejar su ISA y la arquitectura amigable con lenguaje C. Por ser el más simple en la escala de los Cortex-M, se hace sencilla la adaptabilidad hacia procesadores más avanzados dado que todos tendrán incluidas estas instrucciones, y el código correrá apropiadamente sin necesidad de hacer modificaciones.

Su uso se hace frecuente en sensores MEMS, implementaciones de máquinas de estado, control de motores, monitores de salud portables y monitores ambientales.

Esta arquitectura posee como características principales:

- Soporte de subsets Thumb y Thumb2 en su ISA
- *Pipeline* de tres etapas con arquitectura de memoria Von Neumann
- Región bit-banding
- Interrupciones no enmascarables (NMI) y hasta 32 interrupciones físicas manejadas por NVIC, con opción para interrupción con *jitter* nulo y cuatro niveles de prioridad de interrupción.
- Modos de suspensión con instrucciones WFI (*Wait For Interrupt*) y WFE (*Wait For Event*) habilitadas. Señales de suspensión y suspensión profunda.
- Depuración por JTAG y por puerto serial con hasta 8 *breakpoints* y 4 *watchpoints*.

Figura 38. **Implementación Cortex-M0**



Fuente: Arm Ltd. *Cortex-M0 Devices Generic User Guide*. p. 14.

Los periféricos del núcleo de Cortex-M0 son:

- NVIC
- SCB
- Temporizador SysTick

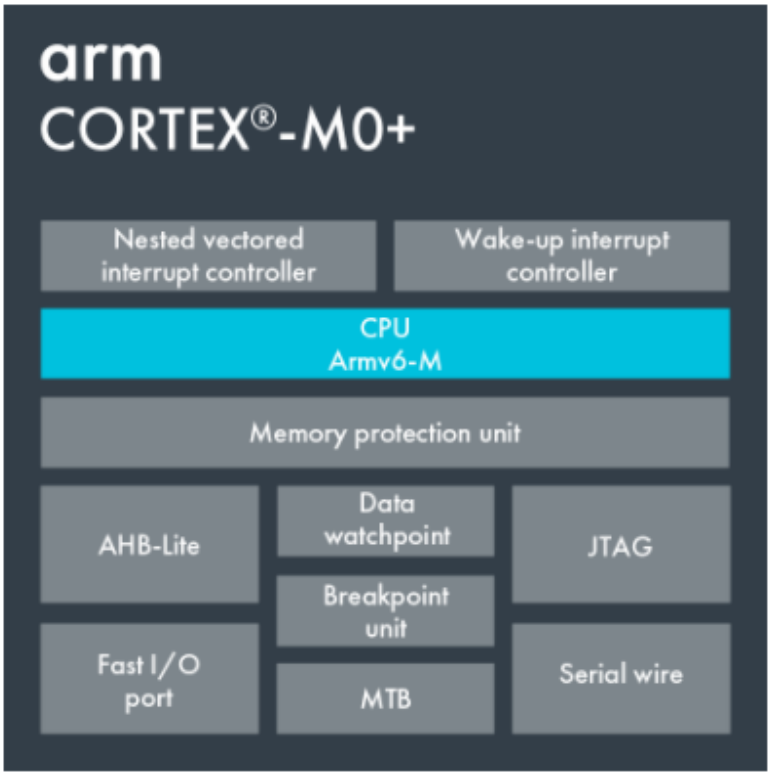
Este procesador junto a Cortex-M0+ y M1 componen el grupo de la arquitectura Armv6-M, con el set de instrucciones más pequeño entre los Cortex (56 *opcodes*), siendo casi todas de 16 bits mientras los datos y registros se mantienen en 32 bits.

La forma reducida de este set de instrucciones permite ejecutar tareas sencillas de control de entradas, salidas y procesamiento básico de datos. Esto puede llegar a ser limitante para aplicaciones complejas; sin embargo permite que la implementación se dé en un número muy bajo de compuertas, reduciendo tanto su tamaño como su costo.

3.4.2. Cortex-M0+

El procesador más eficiente en consumo de energía para sistemas embebidos pequeños. Es similar al Cortex-M0 pero con características adicionales que suman a las ventajas de este, mejoras incluso más grandes en reducción de consumo de potencia e incremento en desempeño.

Figura 39. **Componentes de arquitectura ARM Cortex-M0+**



Fuente: Cortex. *Procesador Cortex-M0+*. https://developer.arm.com/products/processors/cortex-m/cortex-m0+. Consulta: 25 de marzo de 2018.

Al igual que su antecesor, el área de silicio sumamente reducida, bajo consumo de potencia y *code footprint* mínimo permiten desarrollar dispositivos

con desempeño a la altura de 32 bits a precios muy bajos. La compatibilidad hacia dispositivos más avanzados en la familia Cortex es completa.

Esta arquitectura tiene la característica principal de ser la más eficiente en distribución de energía de todos los procesadores Arm, con un consumo de potencia debajo de los 4µW/MHz (en procesos 40LP).

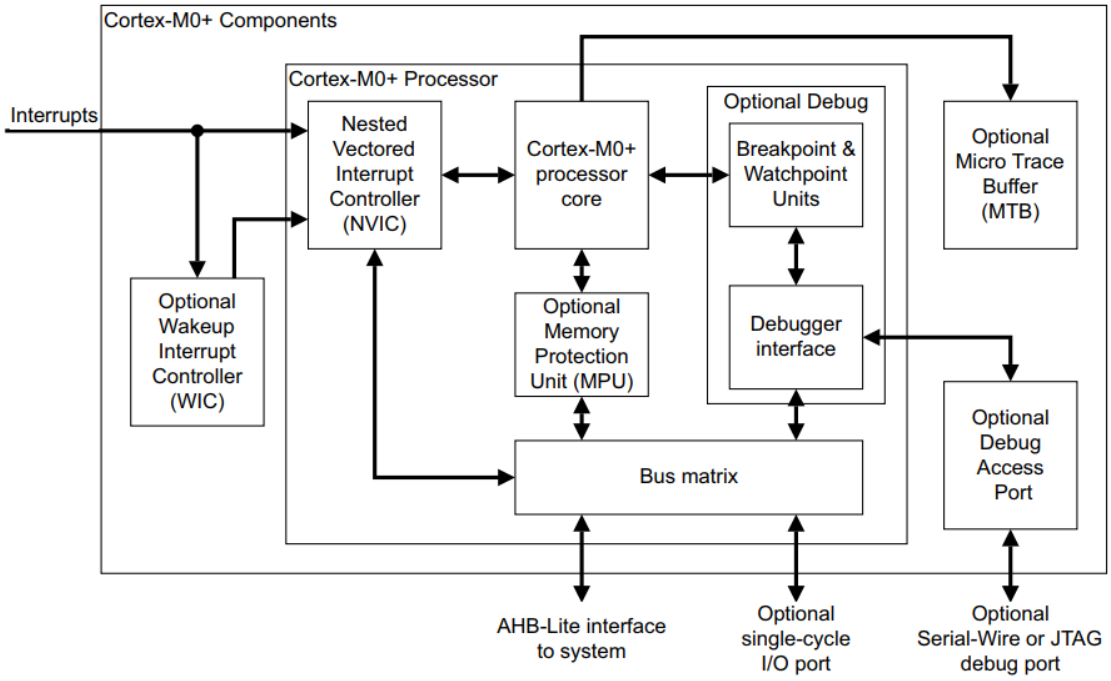
Este tipo de procesadores se encuentra con aplicación en los mismos campos que el Cortex-M0, además de ser implementado en microcontroladores de muy bajo consumo.

Las características que identifican al Cortex-M0+ son:

- Soporte de subsets Thumb y Thumb2
- *Pipeline* de dos etapas con arquitectura de memoria Von Neumann
- Implementación opcional de MPU de 8 regiones con subregiones y regiones de segundo plano.
- Interrupciones no enmascarables y hasta 32 interrupciones físicas manejadas por NVIC, con opción para interrupción con *jitter* nulo y cuatro niveles de prioridad de interrupción.
- Modos de suspensión con instrucciones WFI (*Wait For Interrupt*) y WFE (*Wait For Event*) habilitadas. Señales de suspensión y suspensión profunda y modo de retención opcional.
- Región *Bit-banding.*
- Depuración por JTAG y por puerto serial con hasta 4 *breakpoints* y 2 *watchpoints*.
- MTB opcional con lectura por JTAG.
- Modos de ejecución privilegiado y no privilegiado.

- Ejecución eficiente de código que habilita reloj de procesador de menor frecuencia o tiempo de suspension aumentado.

Figura 40. **Implementación Cortex-M0+**



Fuente: Arm Ltd. *Cortex-M0+ Devices Generic User Guide*. p. 10.

Los periféricos del núcleo de Cortex-M0+ son:

- NVIC
- SCB
- Temporizador SysTick
- MPU opcional
- Puerto de entradas y salidas en un solo ciclo (*single-cycle I/O port*) opcional. Este provee *loads* y *stores* en periféricos fuertemente acoplados.

El set de instrucciones para este procesador comparte las características de Cortex-M0 y M1.

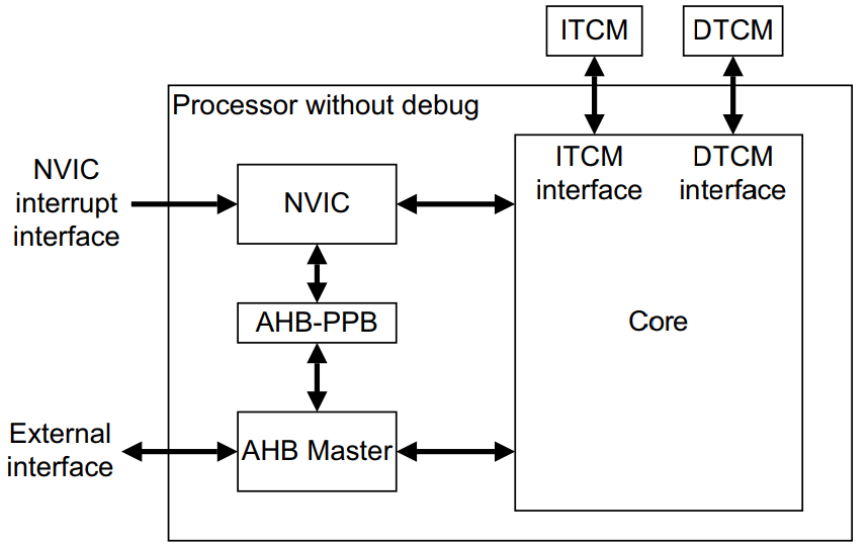
3.4.3. Cortex-M1

Procesador pequeño optimizado para diseños de lógica programable (*soft core*). Cuenta con implementación de TCM usando bloques de memoria en el FPGA, funciona con el mismo set de instrucciones de Cortex-M0 y es también una arquitectura con bajo conteo de compuertas.

Las características que identifican al Cortex-M1 son principalmente:

- Soporte de subsets Thumb y Thumb2.
- Extensión para sistema operativo (OS) opcional.
- Interrupciones no enmascarables y hasta 32 interrupciones físicas manejadas por NVIC, con opción para interrupción con *jitter* nulo y cuatro niveles de prioridad de interrupción.
- *Pipeline* de tres etapas con arquitectura de memoria Von Neumann.
- Puerto de acceso a depuración (DAP).
- Interfaces AHB-Lite de memoria y externas.

Figura 41. **Diagrama de procesador Cortex-M1 sin depurador**



Fuente: Arm Ltd. *Cortex-M1 Technical Reference Manual Revision.* p1. p. 27.

Los periféricos del núcleo de Cortex-M1 son:

- NVIC
- SCB
- AHB-PPB

El procesador Cortex-M1 tiene compatibilidad hacia arriba con Cortex-M3, esto facilita el aumento en eficiencia y migración simple de FPGA a ASIC sin requerir resíntesis.

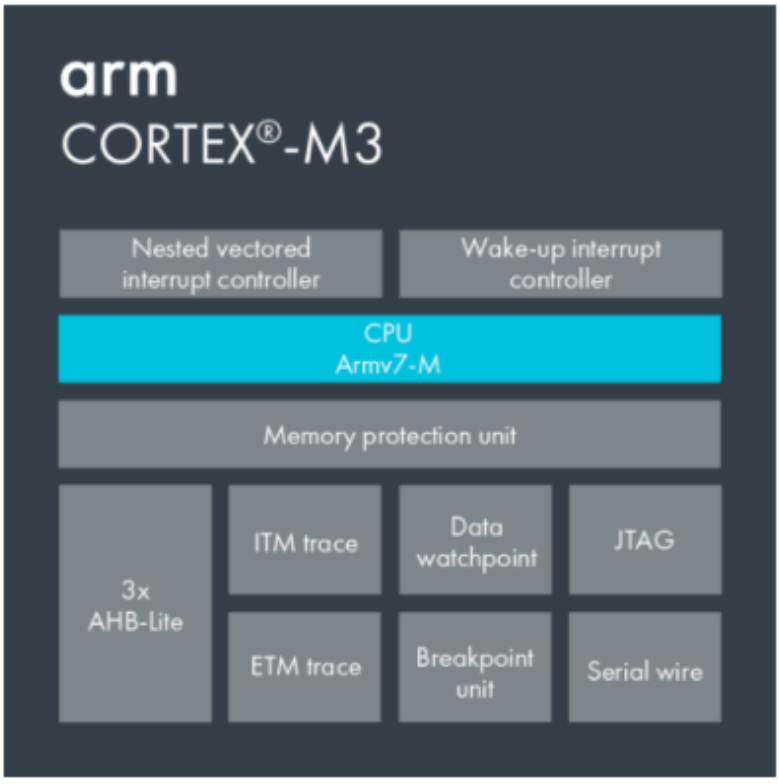
El set de instrucciones para este procesador comparte las características de Cortex-M0 y M0+.

3.4.4. Cortex-M3

Procesador embebido pequeño destinado a usarse en microcontroladores de bajo consumo de potencia y aplicaciones de tiempo real altamente determinísticas. Su ISA lo hace capaz de ejecutar instrucciones complejas más rápido, y es altamente solicitado para diseño de SoC.

Cuenta con un divisor por hardware e instrucciones MAC (*Multiply-Accumulate*).

Figura 42. **Componentes de arquitectura ARM Cortex-M3**



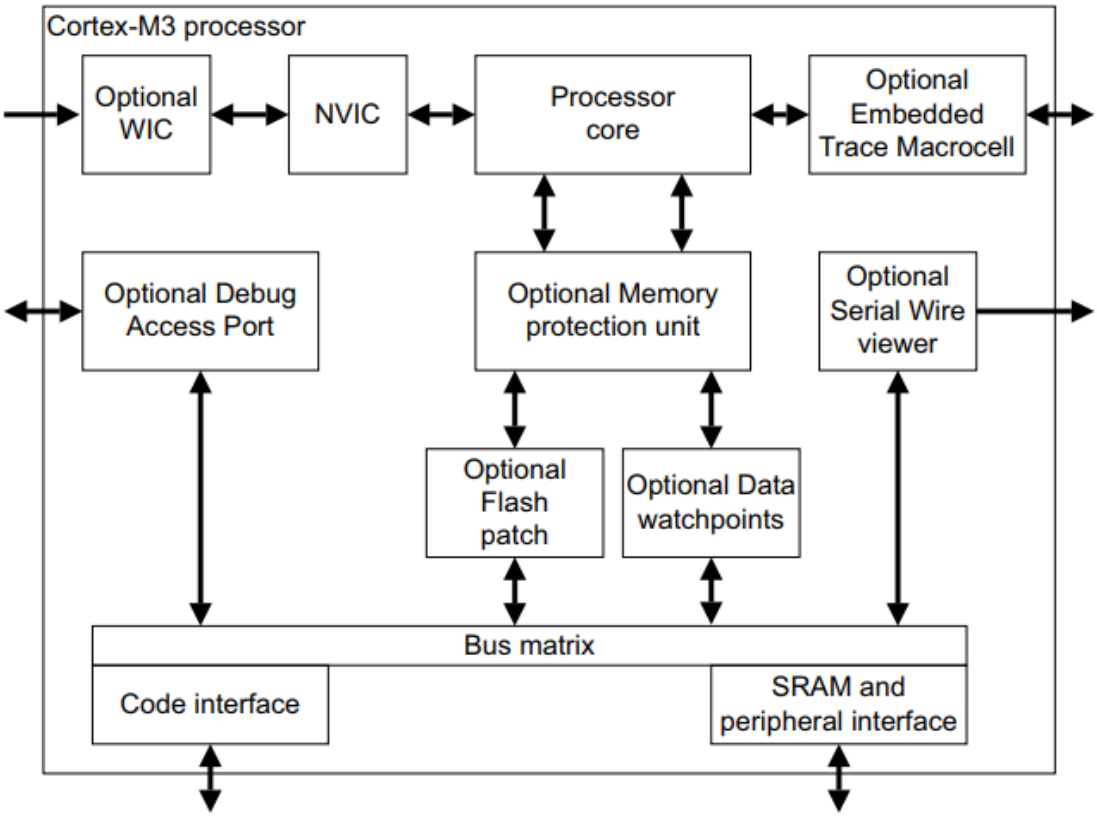
Fuente: Cortex. *Procesador Cortex-M3*. https://developer.arm.com/products/processors/cortex-m/cortex-m3. Consulta: 25 de marzo de 2018.

Esta arquitectura está especialmente desarrollada para dispositivos de alto desempeño y bajo costo en el mercado de sistemas embebidos. Esto encierra microcontroladores, aplicaciones portátiles, IoT, control de motores, conectividad y domótica.

Las características que identifican al Cortex-M3 son principalmente:

- Soporte de subsets Thumb y Thumb2.
- *Pipeline* de tres etapas con arquitectura Armv7E-M Harvard.
- Implementación opcional de MPU de 8 regiones con subregiones y regiones de segundo plano.
- Interrupciones no enmascarables y hasta 240 interrupciones físicas manejadas por NVIC, con opción para interrupción con *jitter* nulo y desde 8 a 256 niveles de prioridad de interrupción.
- Modos de suspensión con instrucciones WFI (*Wait For Interrupt*) y WFE (*Wait For Event*) habilitadas. Señales de suspensión y suspensión profunda y modo de retención opcional.
- Región *Bit-banding.*
- Depuración por JTAG y por puerto serial con hasta 8 *breakpoints* y 4 *watchpoints*.
- Modos de ejecución privilegiado y no privilegiado.
- WIC (*Wait Interrupt Controller*) cuenta hasta con 240 interrupciones *wake-up.*
- Ejecución eficiente de código que habilita reloj de procesador de menor frecuencia o tiempo de suspension aumentado.
- Control de interrupciones de alto desempeño y altamente determinístico para aplicaciones en que el tiempo es crítico.
- Manipulación atómica de bits para habilitar acceso a control de periféricos más veloz.

Figura 43. **Implementación de procesador Cortex-M3**



Fuente: Arm Ltd. *Cortex-M3 Devices Generic User Guide*. p. 11.

Los periféricos del núcleo de Cortex-M3 son:

- NVIC
- SCB
- Temporizador SysTick
- MPU opcional

El procesador Cortex-M3 provee múltiples interfaces utilizando tecnología AMBA para acceso de alta velocidad y baja latencia a memoria. A su vez,

cuenta con soporte para acceso a datos no alineados (a excepción de los anteriores).

Como parte de la arquitectura Armv7-M, este procesador cuenta con un set de instrucciones más elaborado que el de Cortex-M1 que incluye varias instrucciones de 32 bits y brinda acceso más eficiente a los registros altos.

Este set también incluye soporte para:

- Saltos y ejecución condicional con la instrucción IT
- División por hardware
- *Multiply-Acummulate* (MAC)
- Operaciones a nivel de bits

Las instrucciones MAC junto con las de ajuste de saturación en este grupo componen soporte muy básico para operaciones DSP en múltiples ciclos de reloj. La introducción de más opciones de 32 bits habilita el uso del *barrel shifter* con otras operaciones de datos en una sola instrucción.

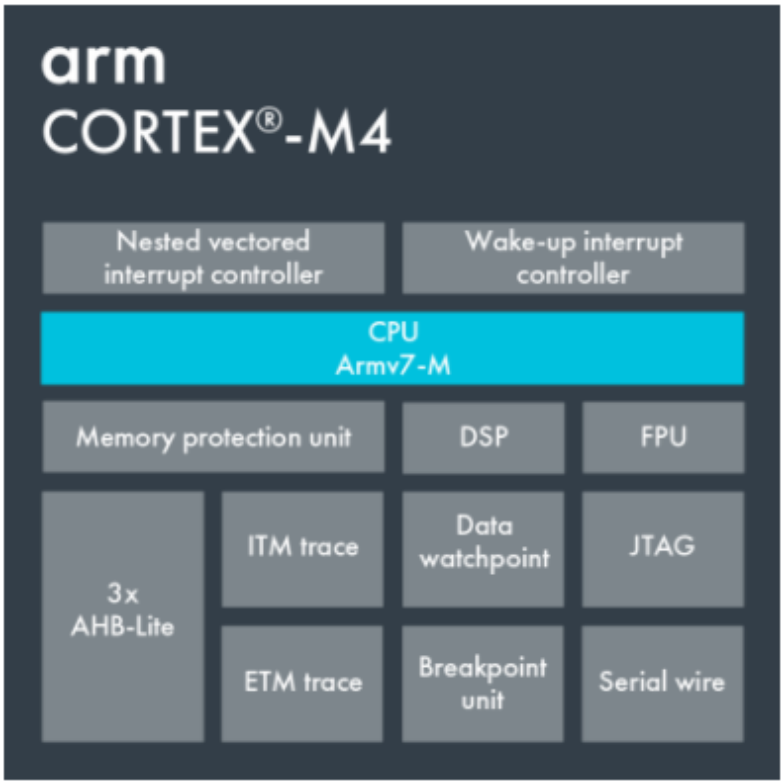
En microcontroladores, el conteo de compuertas de un Cortex-M3 llega a ser más del doble que un M0; sin embargo, aun en estas condiciones el número es insignificante frente al total de silicio que se emplea en el chip en general.

3.4.5. Cortex-M4

Encierra todas las características de Cortex-M3, con instrucciones extra destinadas a procesamiento digital de señales (DSP), como Single Instruction Multiple Data (SIMD), e instrucciones MAC de ejecución en un solo ciclo. Tiene

unidad de punto flotante de precisión simple con soporte de estándar IEEE 754. Estas características lo hacen un procesador de alto rendimiento.

Figura 44. **Componentes de arquitectura ARM Cortex-M4**



Fuente: Cortex. *Procesador Cortex-M4*. https://developer.arm.com/products/processors/cortex-m/cortex-m4. Consulta: 25 de marzo de 2018.

Las características del Cortex-M4 lo hacen óptimo para el mercado de control de señales digitales, debido a que ofrece eficiencia, facilidad de uso, bajo consumo y una mezcla de capacidades de procesamiento de señales y control. Sus aplicaciones típicas incluyen conectividad, IoT, audio, portátiles, domótica, control de potencia y de motores.

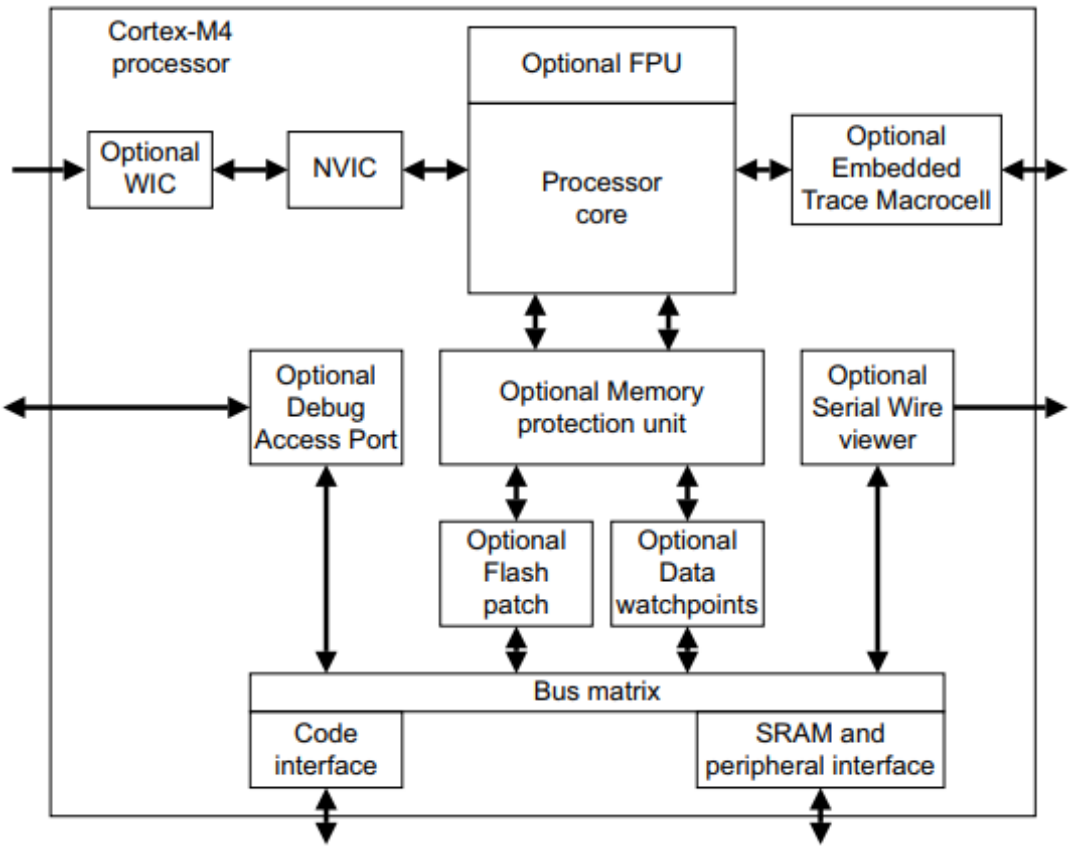
Este procesador gana las ventajas de un microcontrolador con DSP, SIMD y MAC integrados. Dado que las operaciones de punto flotante son manejadas por hardware, los resultados se aceleran hasta diez veces sobre cualquier librería (FPU por software).

Las características que identifican al Cortex-M4 son principalmente:

- Soporte de subsets Thumb y Thumb2.
- *Pipeline* de tres etapas con arquitectura Armv7E-M Harvard.
- Implementación opcional de MPU de 8 regiones con subregiones y regiones de segundo plano.
- Interrupciones no enmascarables y hasta 240 interrupciones físicas manejadas por NVIC, con opción para interrupción con *jitter* nulo y desde 8 a 256 niveles de prioridad de interrupción.
- Modos de suspensión con instrucciones WFI (*Wait For Interrupt*) y WFE (*Wait For Event*) habilitadas. Señales de suspensión y suspensión profunda y modo de retención opcional.
- Región *Bit-banding.*
- Depuración por JTAG y por puerto serial con hasta 8 *breakpoints* y 4 *watchpoints*.
- Modos de ejecución privilegiado y no privilegiado.
- WIC (*Wait Interrupt Controller*) cuenta hasta con 240 interrupciones *wake-up.*
- Extensiones DSP que incluyen MAC *single cycle,* aritmética SIMD y division por hardware de 2 a 12 ciclos.
- Unidad de punto flotante de precision simple opcional, compatible con estándar IEEE 754.
- Instrucciones opcionales de *Embedded Trace Macrocell* (ETM), *Data Watchpoint and Trace* (DWT) e *Instrumentation Trace Macrocell* (ITM).

- Ejecución eficiente de código que habilita reloj de procesador de menor frecuencia o tiempo de suspension aumentado.
- Control de interrupciones de alto desempeño y altamente determinístico para aplicaciones en que el tiempo es crítico.
- Manipulación atómica de bits para habilitar acceso a control de periféricos más veloz.
- Múltiples interfaces utilizando tecnología AMBA para proveer acceso a memoria de alta velocidad y baja latencia.
- Soporte de acceso a datos no alineados.

Figura 45. **Implementación de procesador Cortex-M4**



Fuente: Arm Ltd. *Cortex-M4 Devices Generic User Guide*. p.13.

Los periféricos del núcleo de Cortex-M4 son:

- NVIC
- SCB
- Temporizador SysTick
- MPU opcional
- FPU opcional

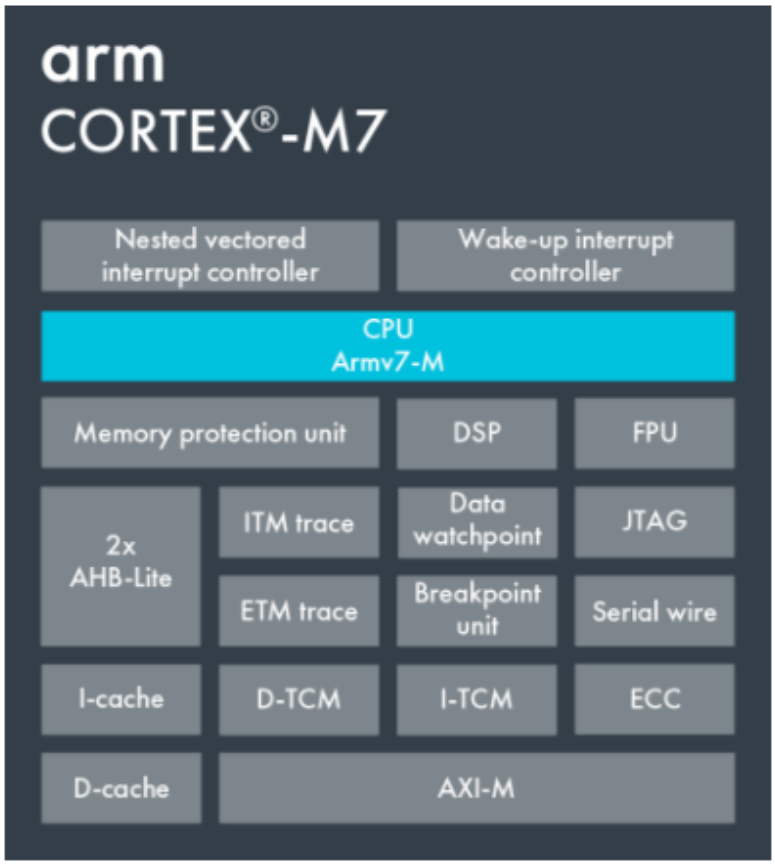
El set de instrucciones de Cortex-M4 contiene todas las características de Cortex-M3 y añade capacidad para aplicar operaciones DSP, SIMD (dos de 16 bits o cuatro de 8 bits), aritmética saturada, instrucciones MAC de un solo ciclo y operaciones de punto flotante con precisión simple (FPv4).

Es importante considerar que las instrucciones SIMD aceleran el procesamiento grandemente pero los compiladores en lenguaje C suelen excluir su uso, y las evaluaciones de Cortex-M3 y Cortex-M4 suelen arrojar resultados similares.

3.4.6. Cortex-M7

Procesador de alto rendimiento para microcontroladores de gama alta y SoCs, es el de más alto desempeño dentro del perfil. Tiene todas las instrucciones de Cortex-M4 con soporte adicional para punto flotante de doble precisión y otras características que lo hacen capaz de soportar aplicaciones demandantes, pero mantenerse en el rango de bajo costo al mismo tiempo.

Figura 46. **Componentes de arquitectura ARM Cortex-M7**



Fuente: Cortex. *Procesador Cortex-M7*. https://developer.arm.com/products/processors/cortex-m/cortex-m7. Consulta: 25 de marzo de 2018.

Esta arquitectura cuenta con interfaces de memoria y sistema flexibles, incluyendo AXI, AHB, cachés y TCM. El rápido acceso a datos y código importantes aumenta la capacidad de responder a eventos críticos.

Incluye soporte para requerimientos automáticos como recuperación de errores a través de ECC (*Error Correction Code*), en la memoria, a través de MBIST (*Memory Built-in Self Test*) y habilitando SEC-DED (*Single Error Correct, Double Error Detect*) para los accesos a memoria.

Su diseño es compatible con los procesadores Cortex-M3 y M4, modelos de programación amigables con lenguaje C.

Cortex-M7 es doblemente redundante, quiere decir que puede operar en modo *lock-step* (usado para aplicaciones de seguridad crítica, con dos copias de la lógica corriendo en desfase por un par de ciclos entre sí para evitar errores vitales de alta velocidad). Al mismo tiempo para aumentar la capacidad de procesamiento, puede ejecutarse algunos pares de instrucciones simultáneamente (procedimiento llamado *dual issue*).

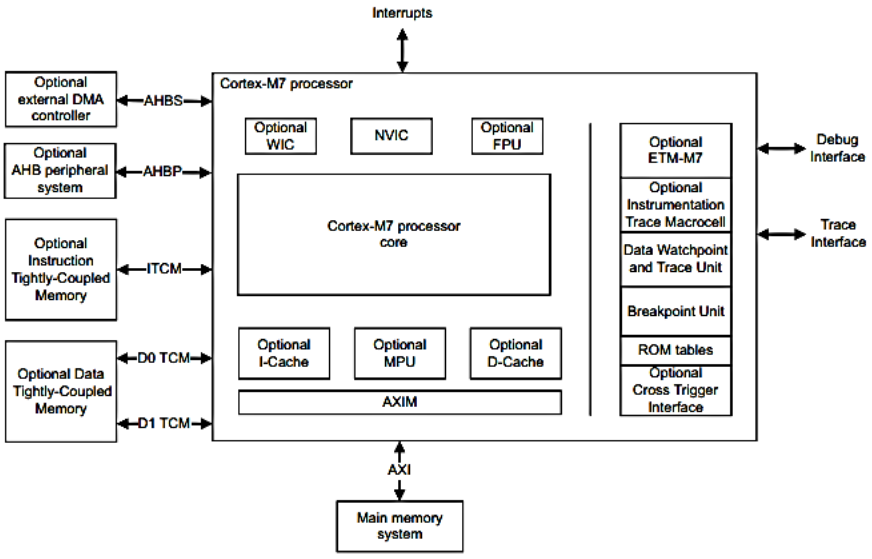
El fabricante de microcontroladores con este procesador puede determinar la configuración de algunas características, y estas pueden variar entre dispositivos y familias. Entre las que identifican al Cortex-M7 se encuentran:

- Soporte de subsets Thumb y Thumb2
- *Pipeline* de seis etapas superescalar con arquitectura Harvard y predicción de saltos.
- Extensiones DSP que incluyen MAC *single cycle,* aritmética SIMD y division por hardware de 2 a 12 ciclos.
- Unidad de punto flotante de precision simple o doble opcional, compatible con estándar IEEE 754.
- Implementación opcional de MPU de 8 o 16 regiones con subregiones y regiones de segundo plano.
- Modos de suspensión con instrucciones WFI (*Wait For Interruption*) y WFE (*Wait For Event*) habilitadas. Señales de suspensión y suspensión profunda y modo de retención opcional.
- Modos de ejecución privilegiado y no privilegiado.

- Interrupciones no enmascarables y hasta 240 interrupciones físicas manejadas por NVIC, con opción para interrupción con *jitter* nulo y desde 8 a 256 niveles de prioridad de interrupción.
- Región *Bit-banding.*
- Depuración por JTAG y por puerto serial con hasta 8 *breakpoints* y 4 *watchpoints*.
- Instrucciones opcionales de *Embedded Trace Macrocell* (ETM), *Data Watchpoint and Trace* (DWT), *Instrumentation Trace Macrocell* (ITM) y *Macro Trace Buffer* (MTB).
- AXI4 de 64 bits y puerto periférico AHB.
- De 0 a 64 KB de caché de instrucciones con asociacividad de dos vías y ECC opcional.
- De 0 a 64 KB de caché de datos con asociacividad de cuatro vías y ECC opcional.
- De 0 a 16 MB de TCM de instrucciones con ECC opcional.
- De 0 a 16 MB de TCM de datos con ECC opcional.
- Instrucciones de depuración integradas.
- Control de interrupciones de alto desempeño y altamente determinístico para aplicaciones en que el tiempo es crítico.
- Soporte de acceso a datos no alineados.

Las aplicaciones típicas del Cortex-M7 incluyen las áreas de procesamiento de audio, industria automotriz, automatización industrial, sensores, procesamiento de video e imágenes, control avanzado de motores, y control de drones.

Figura 47. **Implementación de procesador Cortex-M7**



Fuente: Arm Ltd. *Cortex-M7 Devices Generic User Guide*. p.12.

Los periféricos del núcleo de Cortex-M7 son:

- NVIC
- SCB
- Temporizador SysTick
- MPU opcional
- FPU de precisión doble y simple opcional
- Cachés opcionales de datos e instrucciones integradas

El set de instrucciones de Cortex-M7 es similar al Cortex-M4 con la adición de:

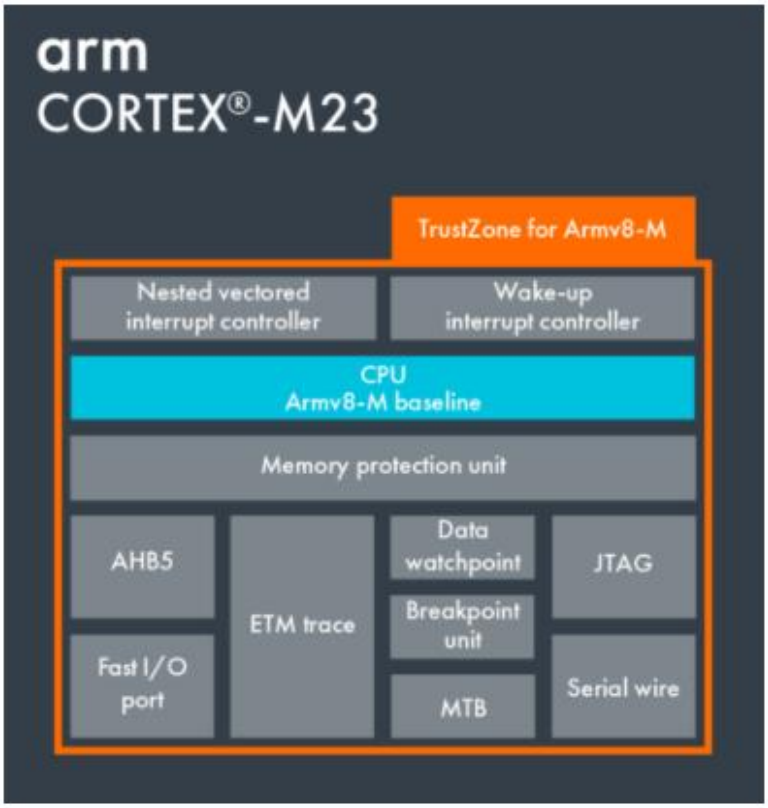
- La arquitectura de punto flotante está basada en FPv5 en lugar de FPv4, y contiene más instrucciones.

- Instrucciones opcionales de punto flotante con precisión doble.
- Instrucción PLD (*Preload Data*) para obtener datos por adelantado.

3.4.7. Cortex-M23

Procesador pequeño de ultra bajo consumo de potencia y bajo costo de diseño, similar al Cortex-M0+ con varias mejoras en el set de instrucciones y a nivel de sistema. También cuenta con soporte para la extensión de seguridad TrustZone.

Figura 48. **Componentes de arquitectura ARM Cortex-M23**



Fuente: Cortex. *Procesador Cortex-M23*. https://developer.arm.com/products/processors/cortex-m/cortex-m23. Consulta: 25 de marzo de 2018.

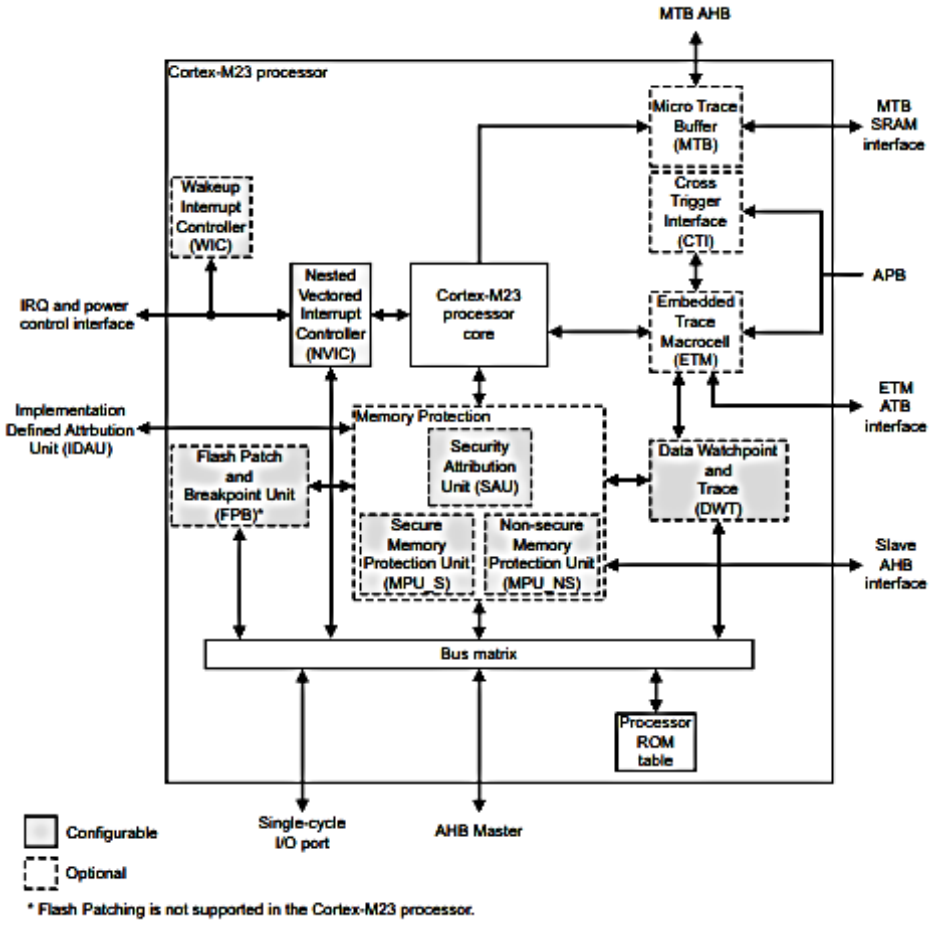
Esta arquitectura es compatible hacia arriba con el Cortex-M33. Sus aplicaciones pretendidas se encuentran en sensores inteligentes, nodos IoT de recolección de energía, control digital de motores, autenticación biométrica, nanorobótica médica y servicios de rastreo de activos.

Las características que identifican al Cortex-M23 son principalmente:

- Soporte de subsets Thumb y Thumb2
- *Pipeline* de dos etapas con arquitectura von Neumann
- Implementación opcional de MPU de 16 regiones con subregiones y regiones de segundo plano.
- Interrupciones no enmascarables y hasta 240 interrupciones físicas manejadas por NVIC, con opción para interrupción con *jitter* nulo y 4niveles de prioridad de interrupción.
- Modos de suspensión con instrucciones WFI (*Wait For Interrupt*) y WFE (*Wait For Event*) habilitadas. Señales de suspensión y suspensión profunda y modo de retención opcional.
- Modos de ejecución privilegiado y no privilegiado.
- Región *Bit-banding.*
- Depuración por JTAG y por puerto serial con hasta 4 *breakpoints* y 4 *watchpoints*.
- Instrucciones opcionales de *Embedded Trace Macrocell* (ETM) *o Macro Trace Buffer* (MTB).
- TrustZone opcional.
- Wake-up Interrupt Controller (WIC) opcional para recuperar el estado del procesador desde retención o cuando todos los relojes han sido detenidos.
- Capacidad de control de interrupciones determinístico.

- Interfaz de Sistema AMBA5 AHB de 32 bits para integración de periféricos de Sistema y memoria.
- Puerto opcional de 32 bits *single-cycle* de entradas y salidas.

Figura 49. **Diagrama de procesador Cortex-M23**



Fuente: Arm Ltd. *Cortex-M23 Technical Reference Manual*. p. 24.

Los periféricos del núcleo de Cortex-M23 son:

- NVIC

- SCB
- Temporizador SysTick
- MPU opcional
- Puerto de entradas y salidas en un solo ciclo (*single-cycle I/O port*) opcional. Este provee *loads* y *stores* en periféricos fuertemente acoplados.

El set de instrucciones de este procesador está contenido en ARMv8-M, formando el subperfil base (superset de Armv6-M). Sus atribuciones extra incluyen:

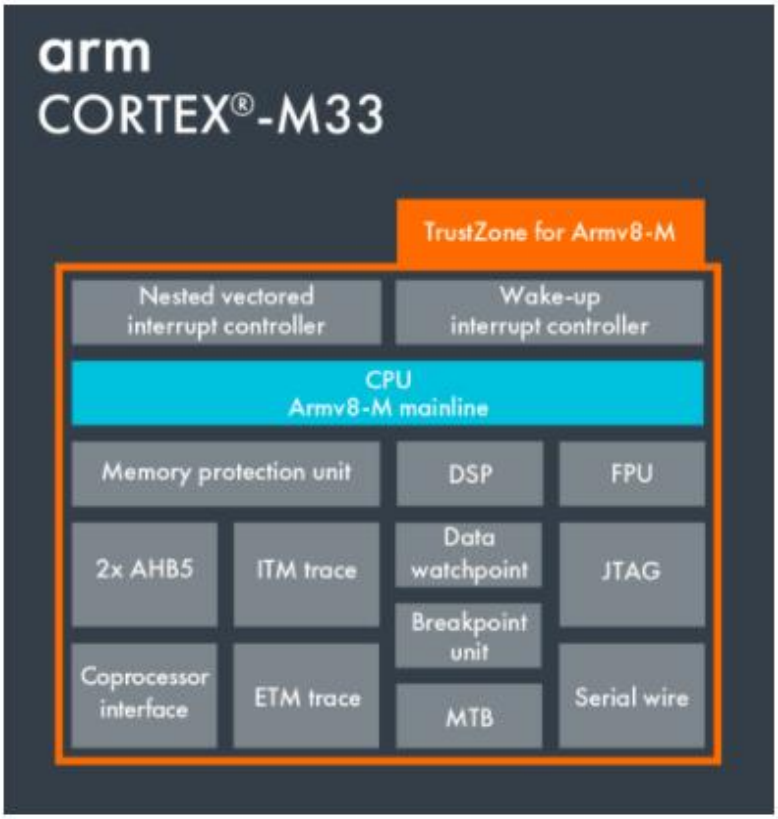
- División por hardware
- Instrucciones de 32 bits para comparación y saltos
- Instrucciones para la extensión de seguridad TrustZone
- Instrucciones de acceso exclusivo (para operaciones semáforo)

Este grupo resulta útil para diseños SoC con múltiples procesadores.

3.4.8. Cortex-M33

Arquitectura de línea principal, similar a Cortex-M3 y Cortex-M4 pero con mayor flexibilidad en diseño de sistema, mejor eficiencia de energía y mayor rendimiento. También tiene soporte para extensión TrustZone.

Figura 50. **Componentes de arquitectura ARM Cortex-M33**



Fuente: Cortex. *Procesador Cortex-M33*. https://developer.arm.com/products/processors/cortex-m/cortex-m33. Consulta: 25 de marzo de 2018.

El Cortex-M33 está diseñado para suplir las necesidades del mercado de embebidos e IoT, con especial enfoque en los que requieran seguridad o control de señales digitales. Este alcanza una mezcla muy buena de determinismo de tiempo real, eficiencia de energía, productividad de software y seguridad de sistema.

Las características que identifican al Cortex-M33 son:

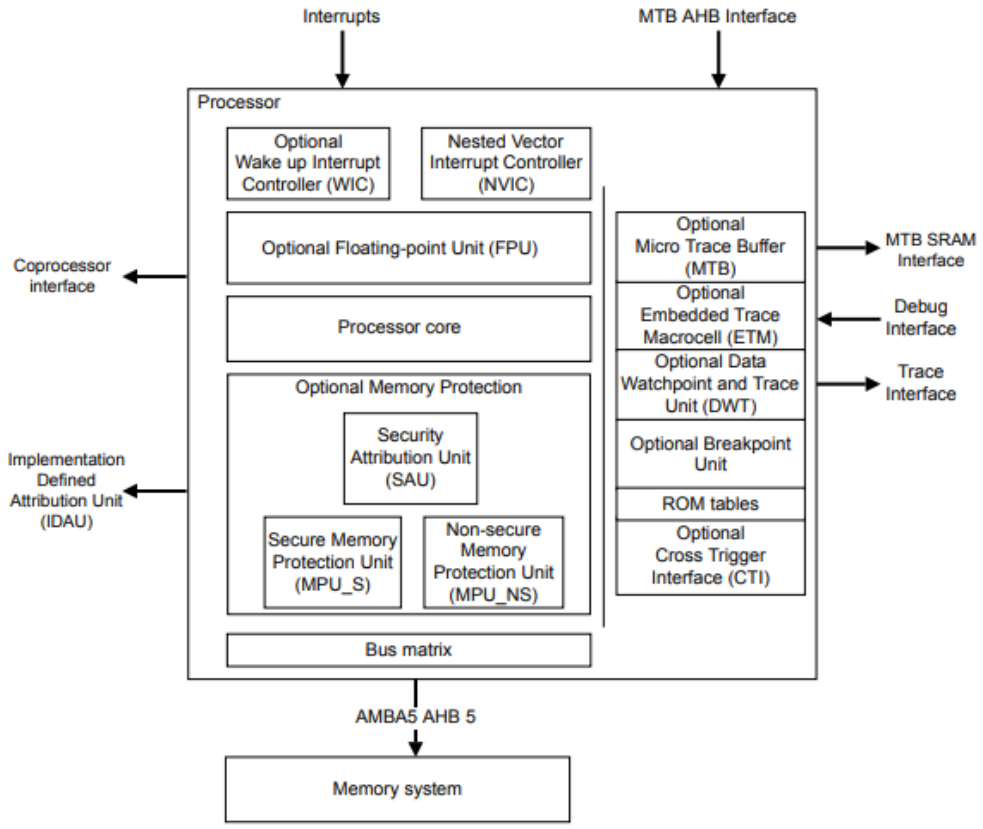
- Soporte de subsets Thumb y Thumb2
- *Pipeline* de tres etapas con arquitectura Harvard
- Extensiones DSP que incluyen MAC *single cycle,* aritmética SIMD y division por hardware de 2 a 12 ciclos.
- Unidad de punto flotante de precision simple opcional, compatible con estándar IEEE 754.
- Implementación opcional de MPU de 16 regiones con subregiones y regiones de segundo plano.
- Modos de suspensión con instrucciones WFI (*Wait For Interrupt*) y WFE (*Wait For Event*) habilitadas. Señales de suspensión y suspensión profunda y modo de retención opcional.
- Modos de ejecución privilegiado y no privilegiado.
- Interrupciones no enmascarables y hasta 480 interrupciones físicas manejadas por NVIC, con opción para interrupción con *jitter* nulo y desde 8 a 256 niveles de prioridad de interrupción.
- Región *Bit-banding.*
- Depuración por JTAG y por puerto serial con hasta 8 *breakpoints* y 4 *watchpoints*.
- Instrucciones opcionales de *Embedded Trace Macrocell* (ETM), *Data Watchpoint and Trace* (DWT), *Instrumentation Trace Macrocell* (ITM) y *Macro Trace Buffer* (MTB).
- Bus opcional dedicado a coprocesadores para hasta 8 unidades de coprocesamiento estrechamente acopladas con motive de computación personalizada.
- Soporte de acceso a datos no alineados.
- Interfaces AMBA para rápido acceso a datos.
- Extensión de seguridad Armv8-M para características de protección de datos e instrucciones. Este aporta modos seguro y no seguro, que

resultan como una alternativa a los modos de hilo y de control tradicionales.

- Control de interrupciones de alto desempeño y altamente determinístico para aplicaciones en que el tiempo es crítico.

Las aplicaciones de esta arquitectura incluyen las áreas de audio, conectividad, domótica, fusión de sensores, IoT y control de motores.

Figura 51. **Implementación de procesador Cortex-M33 con extensión de seguridad**



Fuente: Arm Ltd. *Cortex-M33 Devices Generic User Guide*. p. 14.

Los periféricos del núcleo de Cortex-M33 son:

- NVIC
- System Control Space (SCS), que es la interfaz del programador hacia el procesador con información de implementación y control de sistema.
- Temporizador SysTick
- Security Attribution Unit (SAU), que mejora el Sistema de seguridad definiendo distintos atributos en hasta 8 regiones
- MPU opcional
- FPU de precisión simple opcional

En el set de instrucciones de este procesador se encuentran muchas instrucciones opcionales como resultado de la alta configurabilidad del procesador. Estas incluyen:

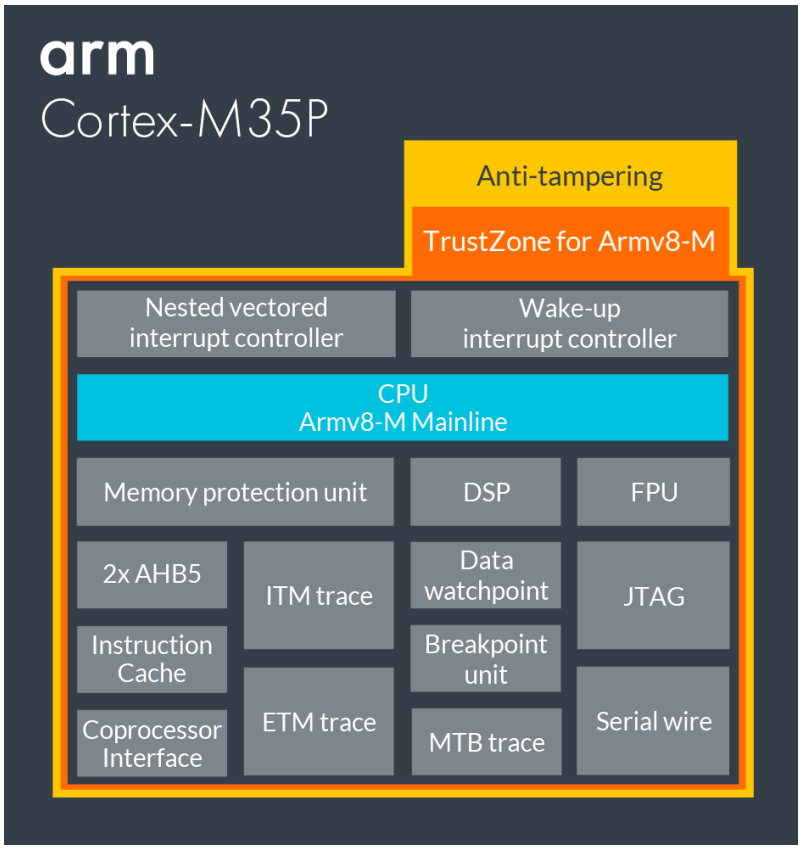
- Instrucciones DSP
- Punto flotante de precisión simple basado en FPv5

Por otro lado, Cortex-M33 introduce nuevas instrucciones en el subperfil principal de Armv8-M como las correspondientes a la extensión de seguridad TrustZone.

3.4.9. Cortex-M35P

Diseño destinado para ser utilizado por desarrolladores de sistemas resistentes a la manipulación física. Se presenta como una arquitectura de alto rendimiento con alto nivel de seguridad y precio accesibles a todo tipo de público por ser parte del perfil de microcontroladores.

Figura 52. **Componentes de arquitectura ARM Cortex-M35**



Fuente: Cortex. *Procesador Cortex-M35*. https://developer.arm.com/products/processors/cortex-m/cortex-m33. Consulta: 14 de septiembre de 2018.

Las características que identifican al Cortex-M35 son principalmente:

- Soporte de subsets Thumb y Thumb2
- *Pipeline* de tres etapas con arquitectura Harvard
- Seguridad física embebida para ataques invasivos y no invasivos
- Extensiones DSP que incluyen MAC *single cycle,* aritmética SIMD de 8 o 16 bits y división por hardware de 2 a 12 ciclos.

- Unidad de punto flotante de precision simple opcional, compatible con estándar IEEE 754.
- Implementación opcional de MPU de 16 regiones con subregiones y regiones de segundo plano.
- Interrupciones no enmascarables y hasta 480 interrupciones físicas manejadas por NVIC desde 8 a 256 niveles de prioridad de interrupción.
- Depuración por JTAG y por puerto serial con hasta 8 *breakpoints* y 4 *watchpoints*.
- Instrucciones opcionales de *Embedded Trace Macrocell* (ETM), *Data Watchpoint and Trace* (DWT), *Instrumentation Trace Macrocell* (ITM) y *Macro Trace Buffer* (MTB).
- Modos de suspensión con instrucciones WFI (*Wait For Interrupt*) y WFE (*Wait For Event*) habilitadas. Señales de suspensión y suspensión profunda y modo de retención opcional.
- Bus opcional dedicado a coprocesadores para hasta 8 unidades de coprocesamiento.
- Extensión de seguridad Armv8-M para características de protección de datos e instrucciones.

Cortex-M35P introduce en el perfil de microcontroladores las características antimanipulación desarrolladas por Arm. Junto con la extension TrustZone y una caché resiliente frente a ataques físicos, esta arquitectura se vuelve en la más robusta desde el punto de vista de seguridad (de software y hardware), con características heredadas de determinismo y eficencia de energía.

Principalmente está orientado a sistemas que deben proteger bienes de alto valor y detectar la forma en que los ataques se realizan para obtener información que ayude a construir mejores técnicas de seguridad.

4. PERFIL CORTEX-A

En el capítulo anterior se expuso las características que hacen a Cortex-M el perfil dirigido a dispositivos de muy bajo consumo de recursos dentro de Arm, haciendo alusión a las aplicaciones que pueden sacrificar cierto porcentaje de rendimiento por mantenerse en el rango bajo de costos y consumo de potencia. En esta sección se considerará la contraparte: aplicaciones para las que sigue siendo importante mantener uso eficiente de recursos pero la principal preocupación es lograr rendimiento muy alto para procesamiento pesado.

El perfil Cortex-A se caracteriza por su soporte para tecnología multinúcleo en distintas configuraciones, presencia de extensiones que permiten manejo más eficiente de cantidades grandes de datos y con la aparición de Armv8-A, el estado de ejecución de 64 bits conocido como AArch64 (y AArch32 para compatibilidad con Armv7). Todos estos detalles, sumados a los que se enlistará más adelante, conforman una línea de procesadores mucho más potentes que los destinados a microcontroladores aunque manteniéndose dentro de límites de consumo de batería que hacen especial a los diseños de Arm.

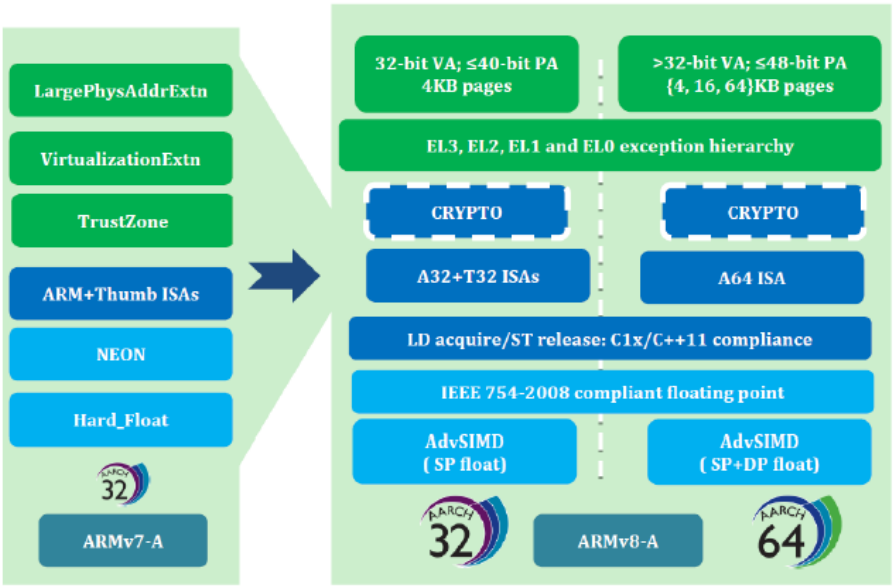
La letra A en Cortex-A hace referencia al campo de aplicaciones (*applications*). Definido como el conjunto de arquitecturas de alto rendimiento con una unidad de gestión de memoria (MMU), que los hace capaces de ejecutar sistemas operativos. Al igual que Cortex-M, el perfil A tiene características compuestas por rasgos de la familia Arm y propias del tipo de procesadores que los hace especializados. Entre las principales para Armv7-A están:

- Núcleo RISC de 32 bits con 16 registros de 32 bits.
- Arquitectura Harvard modificada.
- Tecnología Thumb-2 como estándar.
- Unidades VFP y NEON opcionales (CP10 y CP11).
- Espacio virtual de direccionamiento de 4GB.
- Soporte SMP (*Symmetric Multi-Processing*) en variantes multinúcleo con coherencia de datos en cachés L1.
- Implementación de TLB (*Translation Lookaside Buffer*) para operaciones SMP de alta eficiencia.

Con la llegada de Armv8-A, se introdujeron varios cambios a la arquitectura para ofrecer mejor rendimiento. Estos incluyen:

- Direccionamiento virtual de 64 bits (expandiendo el límite de 4GB).
- 31 registros de 64 bits para propósito general, con el objetivo de incrementar desempeño y reducir accesos a la pila.
- Nuevo modelo de excepciones.
- Mejoras en el uso de caché.
- Criptografía acelerada por hardware para uso de algoritmos como AES, SHA1 y SHA2-256.
- NEON de doble precisión para habilitación de capacidades de procesamiento mucho mayores en caso de buscar implementarse en aplicaciones como computación científica, HPC (*High Performance Computing*) y supercomputadoras.
- Introducción de configuraciones de ocho núcleos para procesadores basados en DynamIQ, en contraste con las de cuatro núcleos disponibles en Armv7-A y la especificación original de Armv8-A.

Figura 53. **Evolución de arquitectura Armv7-A a Armv8-A**



Fuente: SHORE, Chris. *ARMv8-A CPU Architecture Overview*. p. 30.

4.1. Campo de aplicación

Como en el capítulo 3, un punto importante para comprender la importancia del perfil Cortex-A es explorar el campo al que está destinado y las necesidades de potencia y desempeño que estos tipos de procesadores deben tener.

Es importante tomar en cuenta que el perfil de aplicaciones es el que tiene mayor presencia en la actualidad dentro de los diseños Arm. Los dispositivos implementando este tipo de arquitectura han encontrado una amplia satisfacción de los requerimientos principalmente, para dispositivos móviles y avanzando poco a poco a intentar dominar el área de computación de alto rendimiento.

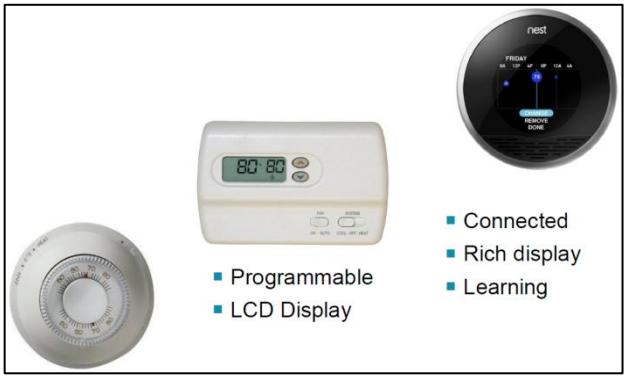
Entre algunos de los socios presentes para desarrollo de chips basados en Armv8-A se encuentran Altera, AMD, Huawei, LG, Nvidia, Samsung, Qualcomm, Broadcom y STMicro.

4.1.1. Sistemas embebidos

Si bien los microcontroladores suelen ser la opción por excelencia para el diseño de sistemas embebidos, existe un sinfín de variaciones para estos en los que se necesitan dispositivos de mayor rendimiento. Para comprender plenamente la definición de este término se recomienda la lectura de la sección 3.1.2.

El uso amplio y versátil de los sistemas embebidos hace que los dispositivos que actúan como cerebros puedan ir desde simples microcontroladores con poca memoria y pines de propósito general, hasta chips con soporte para sistemas operativos, procesamiento pesado de datos o incluso operación en tiempo real. Asumiendo que ya se conocen las generalidades acerca de este campo, se presenta en la figura 54 un caso de la vida real para comprender la presencia de distintos tipos de procesadores en sistemas embebidos.

Figura 54. **Escala de sistemas embebidos en termostatos**

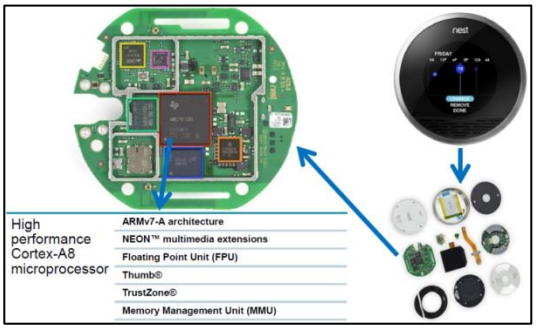


Fuente: Arm Ltd. *Application processors for embedded applications*. p. 7.

En la imagen se observan tres tipos de termostato: uno mecánico para el que cualquier dato se toma y observa sin intervención de procesadores, uno de gama media involucrando electrónica digital con capacidad de programación y visualización por pantalla LCD y por último, un termostato de gama alta con conexión a la red, interfaz amigable y capacidad de aprendizaje con base en los datos que se obtengan en el tiempo.

Es evidente que conforme las aplicaciones se vuelven más complejas por razones de procesamiento, cercanía al usuario u otras, los procesadores que las hacen cobrar vida se vuelven más elaborados. Así podría afirmarse que para el termostato de gama media un microcontrolador sería adecuado para llevar a cabo las tareas requeridas, mientras que para el de gama alta es necesario implementar un procesador que cumpla con las características del perfil Cortex-A (como se observa en la figura 55).

Figura 55. **Termostato de gama alta con procesador Cortex-A8**



Fuente: Arm Ltd. *Application processors for embedded applications*. p. 7.

Dentro del mismo perfil Cortex-A existen variantes respecto a rendimiento que hacen a algunos procesadores más potentes que otros sacrificando (en la mayoría de los casos), el bajo consumo de potencia. Los nuevos diseños están pensados para llevar en la arquitectura lo mejor de ambos mundos, como ha sido propio de Arm desde el inicio.

4.1.1.1. Systems-on-a-Chip (SoC)

A pesar de que la definición de estos dispositivos no es estricta y suele tener interpretaciones distintas, la definición de la Tabla a continuación provee una comprensión general de la idea. Se propone tomarla como base flexible según convenga al campo de estudio específico.

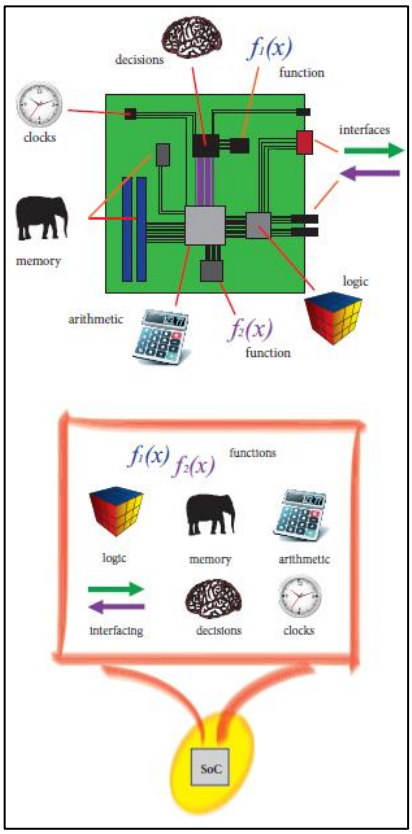
Tabla XVI. **Definición de SoC**

Agrupación de elementos como núcleos, controladores de memoria, periféricos, buses y otros (probablemente conteniendo componentes de radiofrecuencia o analógica) que se disponen en un solo chip con el fin de crear un sistema. Se diferencian de los ASIC (*Aplication Specific Integrated Circuit*) usualmente por su alto nivel de integración.

Fuente: elaboración propia.

Para una fácil comprensión de este término, considerar que usualmente se requiere disponer de varios elementos para llevar a cabo una tarea (entre ellos, el procesador), como es el caso de aplicaciones en que sean necesarios módulos ADC, DAC, PWM, CAN, entre otros. La solución suele imaginarse disponiendo todo lo necesario en una misma placa con varios chips (sistema en un tablero), mientras el SoC cumple las mismas características a una escala mucho menor disponiendo todo en un solo chip.

Figura 56. **Sistemas en un tablero y sistemas en un chip**

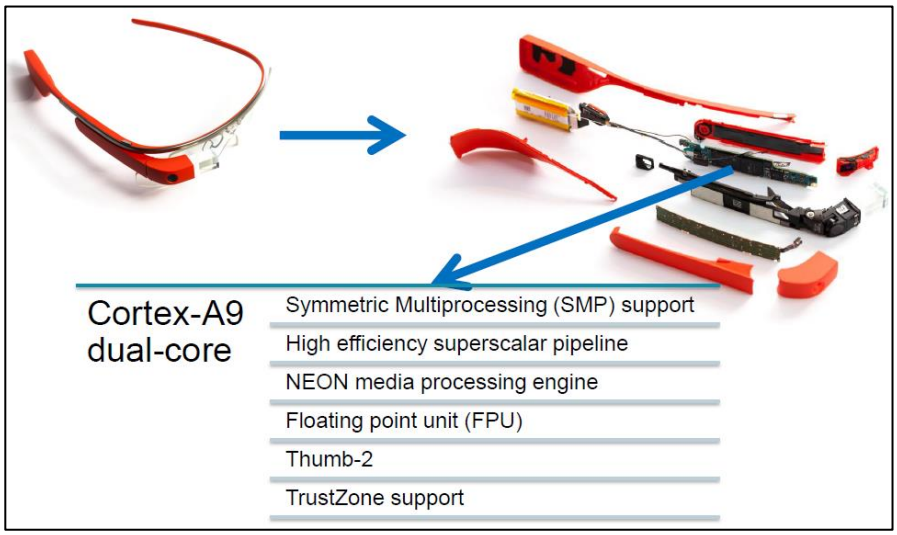


Fuente: CROCKETT, Louise; ELLIOT, Ross; ENDERWITZ, Martin; STEWART, Robert. *The Zynq Book Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. p. 6.

Por supuesto que el término SoC suele considerar tantas configuraciones que los microcontroladores también pueden entrar en este grupo; sin embargo por facilidad en comprensión se los considera como un grupo especial y a partir de este punto, se llama SoC a los chips con capacidad de procesamiento de alto nivel.

Arm ha sido el motor del mercado de dispositivos móviles durante muchos años y cada vez se enfocan más en diseños que sean capaces de soportar cargas muy pesadas y al mismo tiempo, consumir poca energía para mejorar la experiencia del usuario en dispositivos como teléfonos celulares, tabletas, módulos de realidad virtual e interfaces para automóviles. Ejemplos muy populares de aplicación de este tipo de procesadores fueron los *Google Glasses* con un núcleo Cortex-A9 y el Nintendo Switch.

Figura 57. **Ejemplo de dispostivo portátil con procesador Cortex-A**



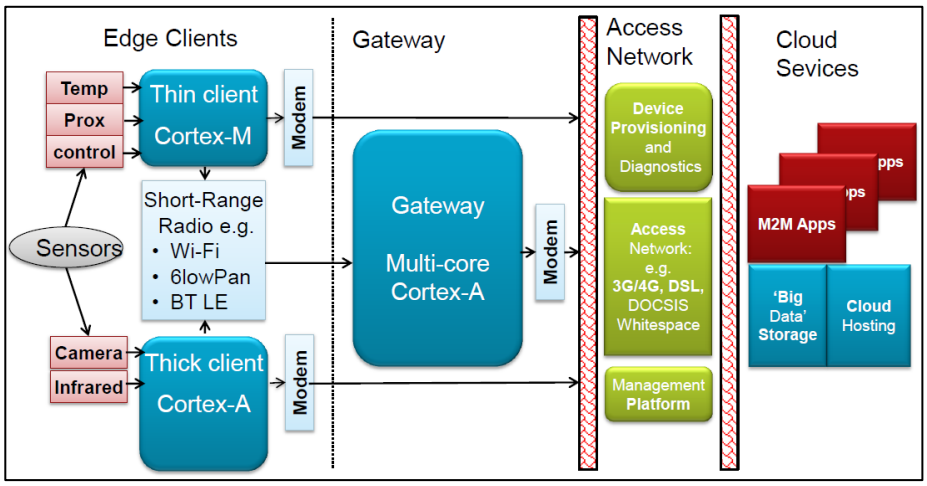
Fuente: Arm Ltd. *Application processors for embedded applications*. p. 10.

Figura 58. **Ejemplo de dispostivo de entretenimiento con procesador Cortex-A**

Fuente: Nintendo. *Nintendo Switch*. https://www.nintendo.com/switch/. Consulta: 15 de junio de 2018.

Para la fecha los procesadores Cortex-A trabajan en conjunto con los Cortex-M en tantos campos que se dice que estos van del sensor a la nube, especialmente para el caso de IoT desde aplicaciones sencillas de tomas de datos hasta dispositivos de inteligencia artificial, pasando por todos los niveles intermedios.

Figura 59. **Procesadores Cortex-M y Cortex-A de los sensores a la nube**



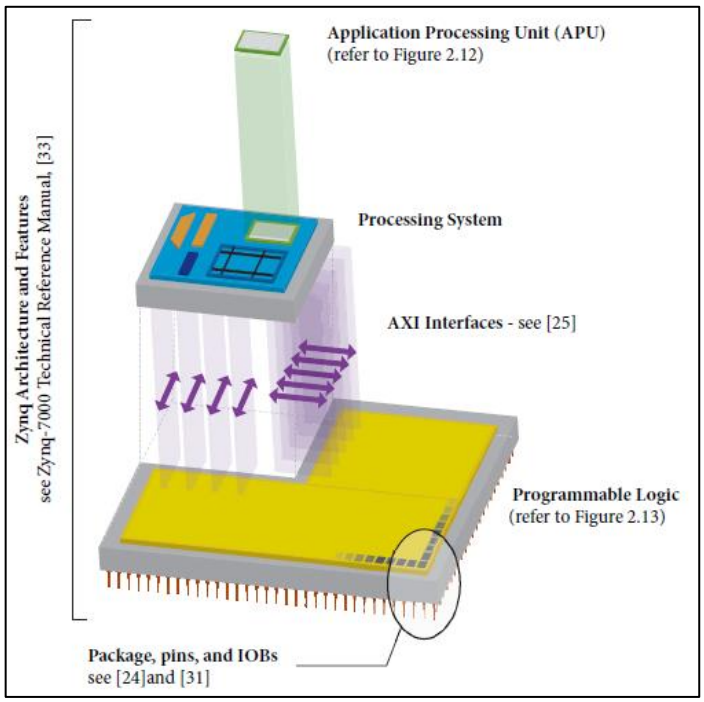
Fuente: Arm Ltd. *Application processors for embedded applications*. p. 15.

4.1.1.2. APSOC

Otra aplicación importante de los Cortex-A aparece en diseños de compañías como Xilinx, que han desarrollado sistemas reprogramables con los beneficios de los FPGA (*Field Programmable Gate Array*), para descripción de hardware y procesadores de aplicación Arm (usualmente Cortex-A) para procesamiento convencional.

Un ejemplo específico de esto es el grupo de dispositivos Zynq, en que se combinan la lógica programable (PL), y el sistema de procesamiento (PS), en un solo chip comunicados por interfaces del estándar AMBA y dispuestos con otros elementos opcionales según la arquitectura.

Figura 60. **Componentes y arquitectura básica de dispositivos Zynq**



Fuente: CROCKETT, Louise; ELLIOT, Ross; ENDERWITZ, Martin; STEWART, Robert. *The Zynq Book Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. p. 41.

4.2. Características de la arquitectura Cortex-A

Habiendo descrito ampliamente los elementos de la arquitectura Cortex-M, resultara mas sencillo para el lector comprender como funciona el grupo de aplicaciones dado que esta ultima se construye sobre el de microcontroladores para ofrecer soluciones de multiprocesamiento y mucho mas alto rendimiento.

Se recomienda la lectura del Capitulo 3; sin embargo, si esto no es posible las definiciones se han dispuesto en el mismo orden para facilitar la lectura de secciones de interés.

4.2.1. Estados de ejecución

En Cortex-A se distinguen dos modos en que pueden operar los procesadores según el programador decida que es más conveniente para determinar el ancho de los registros soportados, el ancho de las instrucciones del set, modelo de excepciones, modelo del programador y arquitectura de sistema de memoria virtual. Estos se dividen en:

4.2.1.1. AARCH32

Estado de ejecución de 32 bits caracterizado por:

- 13 registros de propósito general, PC, SP y LR (todos de 32 bits)
- Uso de registro de enlace de excepciones (ELR) para ejecución y retorno de modo de hipervisor.
- 32 registros de 64 bits para SIMD avanzado y punto flotante.
- Set de instrucciones A32 para ancho fijo de instrucciones de 32 bits.
- Set de instrucciones T32 para ancho variable de instrucciones de 16 o 32 bits.
- Soporte del modelo de excepciones de Armv7-A y mapeo al modelo de Armv8.
- Direccionamiento virtual de 32 bits.

4.2.1.2. AARCH64

Estado de ejecución de 64 bits caracterizado por:

- 31 registros de propósito general que pueden ser utilizados con ancho de 64 bits (X0-X30) o de 32 bits (W0-W30) con PC, SP y ELR.

- 32 registros de 128 bits para soporte de SIMD y punto flotante.
- Definición del modelo de excepciones Armv8 y su jerarquía de privilegios.
- Set de instrucciones A64 para ancho fijo de instrucciones de 32 bits.
- Direccionamiento virtual de 64 bits.
- Se nombra cada registro de sistema con sufijos para indicar el menor nivel de excepción al que se puede acceder.

El cambio de estado entre AArch32 y AArch64 es conocido como interprocesamiento.

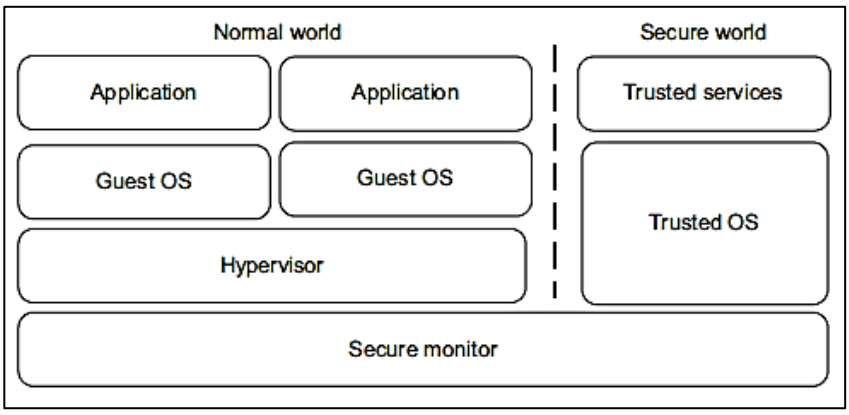
4.2.2. Unidad de gestión de memoria

Denominada MMU por su nombre en inglés (memory management unit). Es una sección de hardware usualmente contenida dentro del CPU que actúa como intermediario para todas las referencias de memoria. Su trabajo consiste principalmente en traducir direcciones de memoria virtual a física, aunque también puede participar en la protección de memoria, control de caché y control de buses en arquitecturas multinúcleo.

4.2.3. Niveles de excepción

Desde la introducción de TrustZone, las ejecuciones se dividieron en estados no seguro (mundo normal), y seguro (mundo seguro), con modos de operación independientes entre sí para manejar software y hardware con seguridad mucho más avanzada que con el modo de procesador privilegiado tradicional. Esto se aborda en la Sección 2.3.6.6. Las extensiones de virtualización añaden un modo de hipervisor (Hyp), a los privilegiados.

Figura 61. **Configuración del hipervisor**

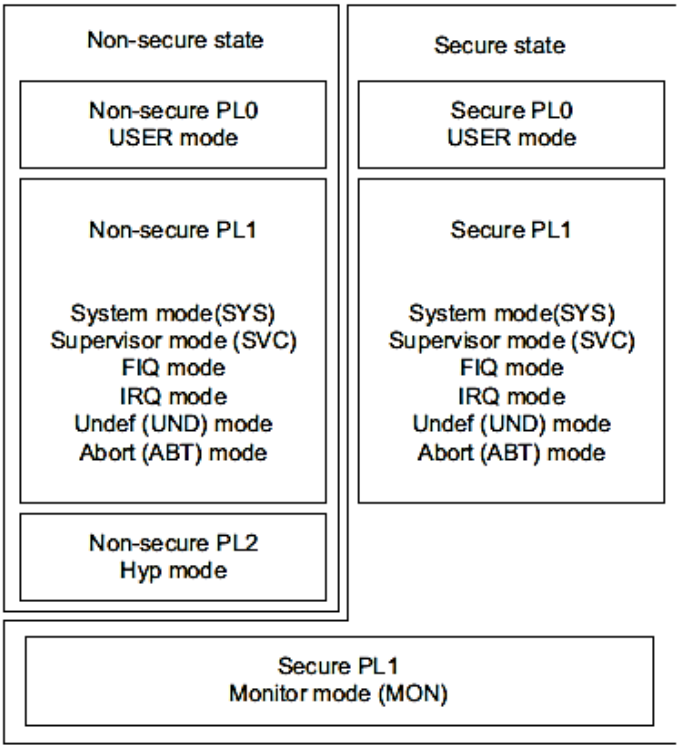


Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide Version 4.0. Arm Ltd*. p. 40.

Si se implementan extensiones de virtualización, el modelo de privilegios se vuelve distinto al de arquitecturas anteriores que se presentan tres niveles de privilegios en el estado no seguro:

- PL0: nivel para software de aplicación que se ejecuta en modo de usuario, este tiene restringido el acceso a algunas características de la arquitectura y la mayoría de sus configuraciones. Se permite sólo el acceso no privilegiado a memoria.
- PL1: software en todos los modos distintos al de usuario y de hipervisor. Común para el sistema operativo.
- PL2: destinado a modo de hipervisor para habilitación y control de sistemas operativos ejecutándose en PL1.

Figura 62. **Niveles de privilegios**



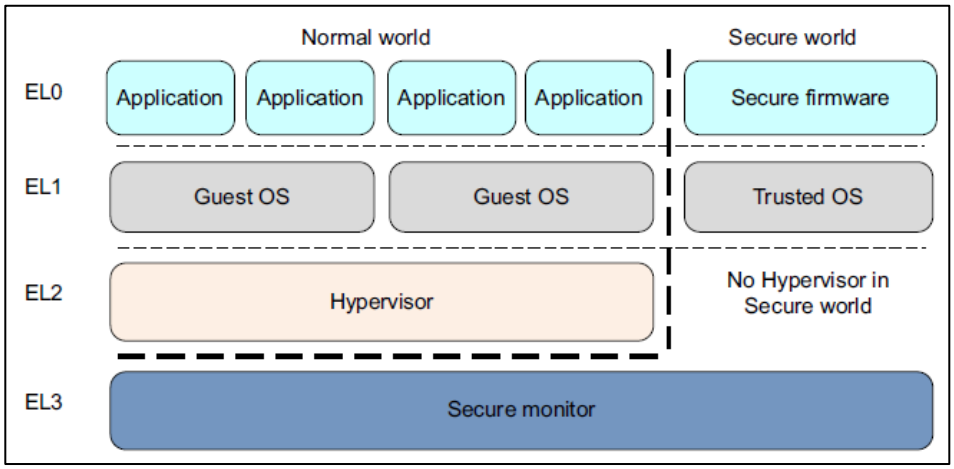
Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide Version 4.0*. p. 42.

En Armv8, la ejecución se da en uno de cuatro niveles de excepción. Para AArch64 el nivel de excepción determina el nivel de privilegio similar a los niveles de privilegio en Armv7:

- EL0: Aplicaciones de usuario
- EL1: Sistema operativo
- EL2: Hipervisor
- EL3: Firmware de bajo nivel

En la figura 63 se observa que los componentes privilegiados que corresponden al mundo normal los kernels de sistemas operativos huéspedes y el hipervisor, y al mundo seguro el firmware privilegiado y sistemas operativos confiables.

Figura 63. **Niveles de excepción en mundo normal y seguro de Armv8**



Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide for ARMv8-A Version 1.0*. p. 29.

4.2.4. Extensiones de virtualización

Con la diversificación en los usos de los procesadores llegan también la necesidad de separar las aplicaciones de software y los ambientes de ejecución por razones de robustez, uso de distintos requerimientos de comportamiento en tiempo real o aislamiento.

La respuesta a esta exigencia es la posibilidad de proveer núcleos virtuales para que el software se ejecute. Esto requiere hardware dedicado (para acelerar la conmutación entre máquinas virtuales), y software de hipervisor.

- El hipervisor

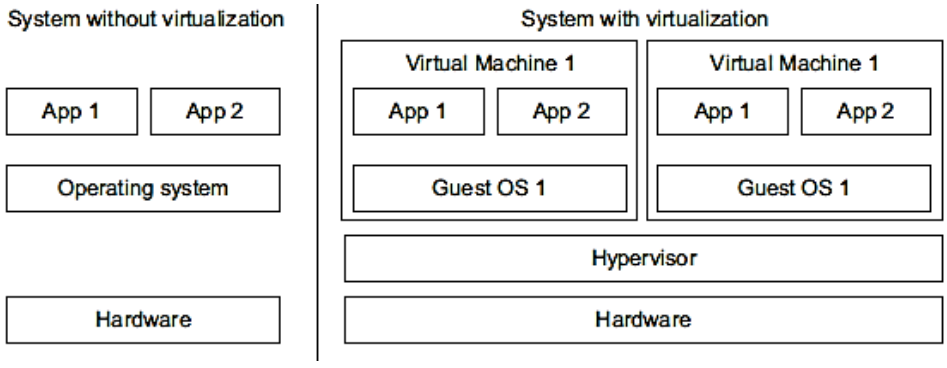
Este término fue usado desde 1960 por IBM para describir software corriendo en *mainframes.* Con el paso del tiempo, la palabra mantuvo una definición similar, considerando que opera a un nivel mayor que el supervisor o el sistema operativo.

Otro término para llamar al hipervisor es *Virtual Machine Monitor* (VMM).

Existen dos tipos de hipervisor en el marco de Arm:

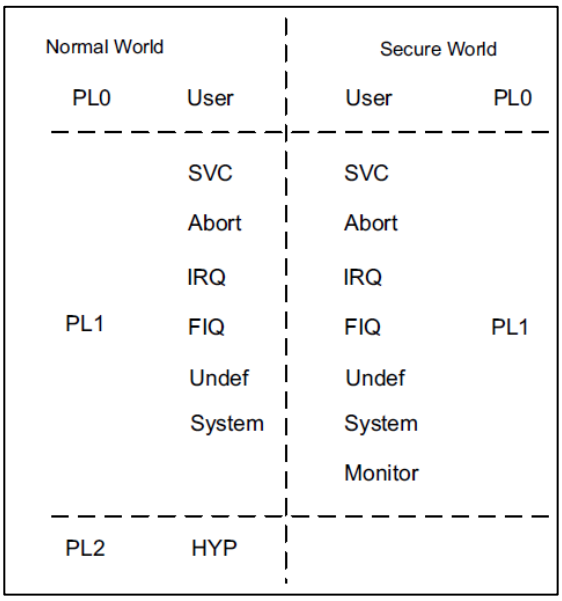
- Tipo 1: descrito con cada máquina virtual que contenga un sistema operativo huésped (el que corre en el sistema), para monitorear su funcionamiento.
- Tipo 2: el hipervisor se comporta como una extensión del sistema operativo anfitrión (el que compone el sistema), con cada sistema operativo huésped contenido en una máquina virtual separada.

Figura 64. **Virtualización *bare metal***



Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide Version 4.0*. p. 290.

Figura 65. **Niveles de privilegio y TrustZone**

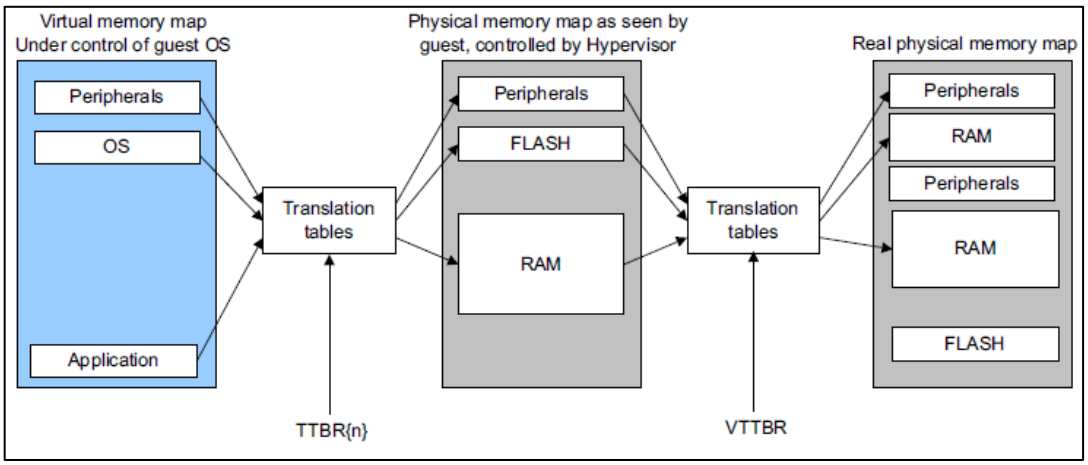


Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide Version 4.0*. p. 295.

Un ejemplo claro del uso de hipervisores es la computación por nube, donde el software puede disponerse en secciones accesibles a cliente o servidor si existen cantidades grandes de datos. Esto al igual que en otras aplicaciones, puede aumentar la cantidad de memoria física direccionable requerida; y llega a ser necesario el uso de LPAE (*Large Physical Address Extension*), para habilitar en cada ambiente el acceso a distintas ventanas de direcciones físicas.

LPAE añade a las características de MMU para traducciones de memoria con el fin de que direcciones de memoria virtual de 32 bits puedan mapearse en un rango de 40 bits de memoria física. Así se obtiene suficiente memoria física para cada máquina virtual aun cuando la demanda total de memoria excede el rango de direccionamiento de 32 bits.

Figura 66. **Etapas de traducción**



Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide Version 4.0*. p. 296.

4.2.5. Extensión de seguridad TrustZone

Un sistema versátil es aquel que se adapta y cuenta con los recursos para responder a necesidades de varios campos conforme las condiciones de los mismos evolucionan.

En el caso de los procesadores, las aplicaciones a las que se destinan arquitecturas como las Cortex-A se encuentran cada vez más sumergidas en el mundo de la contención de información confidencial, manejo de tareas de alto riesgo y otros aspectos que hacen necesario considerar el nivel de confiabilidad del sistema. En este grupo se encuentran dispositivos que protegen bienes como contraseñas, llaves de criptografía, detalles de tarjetas de crédito, entre otros, para evitar ataques de copia, modificación, daño o inhabilitación de la información.

Para estos fines, TrustZone ofrece una vía para particionar los recursos de software y hardware en categorías de acceso con el fin de proteger al sistema de ataques de software y hardware (simples).

Desde Armv6 fue introducida la extension de seguridad y a partir de entonces, se implementó en todos los Cortex-A. Su funcionamiento se explica en la sección 2.3.6.6.

4.2.6. SIMD avanzado (NEON)

La tecnología Arm NEON ofrece implementación de instrucciones SIMD avanzadas con registros por separado a los del núcleo (aunque compartidos con VFP).

NEON provee la posibilidad de ejecutar operaciones SIMD (*Single Instruction Multiple Data*) en los procesadores Cortex-A como un medio para acelerar el rendimiento en aplicaciones de multimedia, porque hacen posible aumentar la velocidad de ejecución de operaciones repetitivas en grupos grandes de datos.

Esta extensión es implementada en hardware, y es opcional en los procesadores Arm; sin embargo lo más común es que esta opción sí se añada al diseño. Sin embargo, la arquitectura no especifica tiempos de ejecución para cada instrucción, y requiere cantidades distintas de ciclos de reloj para ejecutar la misma instrucción en varios procesadores.

- Acerca de SIMD

SIMD es un método utilizado en ciencias de la computación para operar un grupo de datos utilizando una sola instrucción, agrupando los operandos en registros anchos especiales (llamados vectores). Esto permite evitar la ejecución repetitiva de una misma instrucción por separado y por lo contrario, la ejecuta una vez por grupo de datos.

SIMD es comprendido como un tipo de instrucción de procesamiento en paralelo y junto a otros tres tipos, forma parte de la clasificación de las arquitecturas de computadoras según el número de instrucciones y corrientes de datos con las que se trabajan:

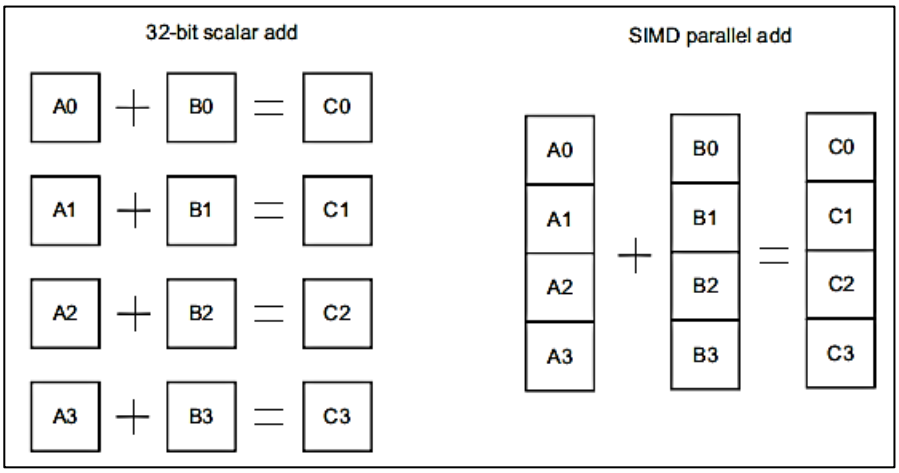
- SIMD (*Single Instruction Multiple Data*): Técnica para procesamiento de varios datos con una sola instrucción.
- SISD (*Single Instruction Single Data*): Se ejecuta un solo hilo de instrucciones que opera en una sola localidad de memoria a la vez. Este modo de funcionamiento es común en los procesadores anteriores a Armv6.
- MISD (*Multiple Instruction Single Data*): Varias unidades ejecutan distintas operaciones en la misma información.
- MIMD (*Multiple Instruction Multiple Data*): distintas instrucciones se ejecutan simultáneamente sobre dos o más piezas de datos. Este tipo es característico de los procesadores multinúcleo.

Para SIMD es común la operación de datos de menos de 32 bits de ancho, como es el caso del procesamiento de imagen con pixeles de 8 bits o codificación de audio con muestras de 16 bits. Para estas aplicaciones, las instrucciones son a menudo simples y repetitivas; y SIMD ofrece una alternativa

funcional para aumentar el desempeño especialmente si se habla de situaciones en que se implemente procesamiento digital de señales.

Como ejemplo en la documentación específica de Arm se muestra la figura 67. En ella se observa dos tipos de operación: una en la que cada suma se hace por separado con cada par de datos para obtener un resultado a la vez (operación escalar), y otra en que los datos son puestos en vectores y luego se procede a sumarlos en una sola operación (SIMD).

Figura 67. **Comparación de operaciones escalares y SIMD**



Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide for ARMv8-A Version 1.0*. p. 97.

NEON comprende instrucciones SIMD de 64 y 128 bits para operaciones de vectores de 128 bits de ancho (comparado con el SIMD de 32 bits de Armv6). Se encuentra disponible en Cortex-A y Cortex-R desde la aparición de Armv7.

Esta tecnología soporta instrucciones para interactuar con memoria externa, trasladar datos entre registros NEON y otros registros Arm.

Las instrucciones NEON operan en los siguientes tipos de elementos:

- Punto flotante de precisión simple (32 bits de ancho)
- Enteros con signo y sin signo de 8, 16, 32 y 64 bits
- Polinomios de 8 y 16 bits

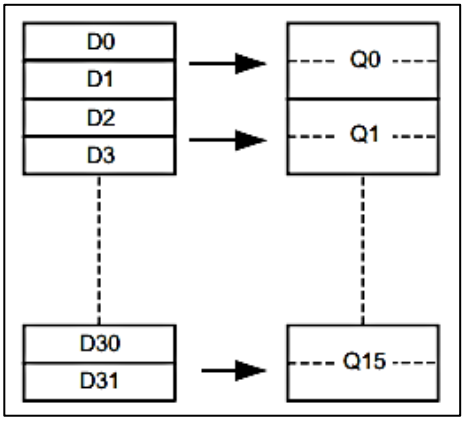
En lenguaje ensamblador, los enlistados arriba se especifican con un prefijo V y un sufijo formado de una letra identificando el tipo y un número indicando el ancho, separados del nemónico por un punto como alguno de los siguientes:

- Entero sin signo: U8, U16, U32, U64
- Entero con signo: S8, S16, S32, S64
- Entero indefinido: I8, I16, I32, I64
- Número de punto flotante: F16, F32
- Polinomio: P8, P16

NEON comparte los registros utilizados con VFP cuando ambas tecnologías se implementan, forzando a la extensión de punto flotante a aparecer en su formato de precisión doble.

La arquitectura NEON cuenta con un banco que puede ser interpretado como de 32 registros de 64 bits de ancho cada uno (D0-D31), o de 16 registros de 128 bits de ancho cada uno (Q0-Q15). En el segundo caso cada registro Q estará formado por dos registros D, como se observa a continuación:

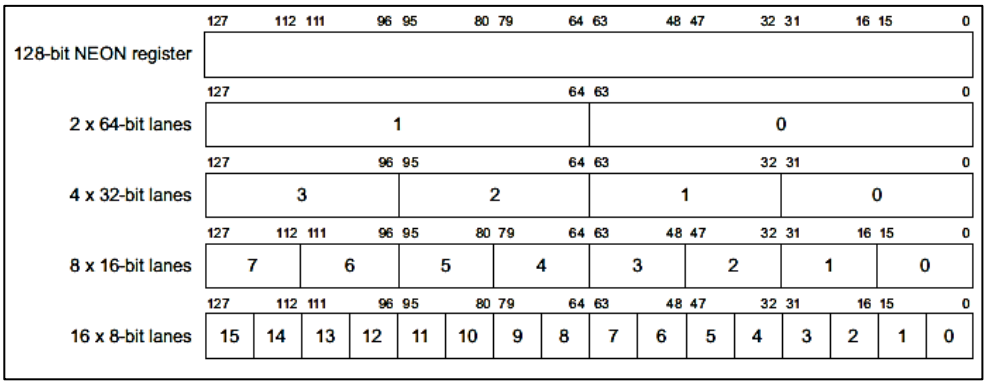
Figura 68. **Distribución de registros NEON**



Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide Version 4.0*. p. 100.

Igualmente, en cuanto a uso de espacios para datos, se mencionó anteriormente que los registros de operadores tienen un ancho de 128 bits y pueden soportar distintos anchos de elementos, ello significa que se pueden operar desde 16 datos de 8 bits cada uno hasta 1 dato de 128 bits.

Figura 69. **Capacidad relativa de registros NEON**



Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide for ARMv8-A Version 1.0*. p. 93.

- Diferencias en AArch64

La tecnología en el estado de 64 bits está basado en AArch32 (lo descrito anteriormente) introduciendo algunos cambios:

  - Se puede contar, en total 32 registros de 128 bits de ancho en lugar de los 16 de Armv7.
  - El prefijo V se elimina de las instrucciones de lenguaje ensamblador.
  - Algunas instrucciones se modificaron para soportar enteros de 64 bits, como comparación, adición, valor absoluto y negación.
  - Se añade soporte para doble precisión y completo funcionamiento de IEEE 754 incluyendo modos de redondeo y manejo de NaN.

4.2.7. Procesadores multinúcleo

Las arquitecturas liberadas por Arm suelen tener la capacidad de presentarse no sólo como núcleos individuales, sino también como sistemas multinúcleo capaces de entregar alto rendimiento y reducción de tiempos de ejecución dado que pueden procesarse múltiples tareas en paralelo.

Tabla XVII. **Definición de multiprocesamiento**

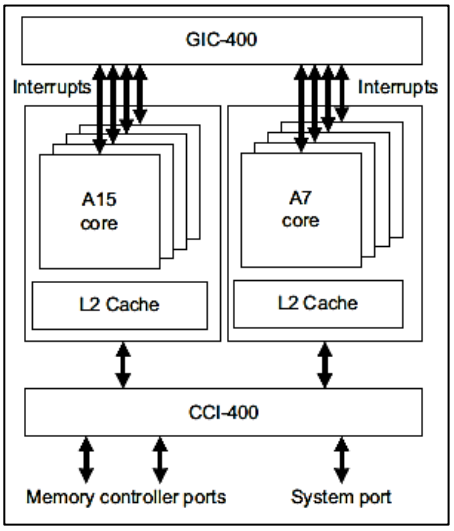
Ejecución de dos o más secuencias de instrucciones simultáneas en un mismo dispositivo con dos o más núcleos que pueden, o no, ser iguales entre sí.

Fuente: elaboración propia.

Las configuraciones de procesadores en Arm suelen tomar formas de núcleos individuales, varios núcleos en un grupo y procesadores con varios grupos de multinúcleo. Las consideraciones del multiprocesamiento incluyen ventajas que hace a los diseñadores escoger topologías de este tipo a menudo:

- El consumo total de energía en un sistema multinúcleo puede llegar a ser menor que el de un solo núcleo procesando todas las instrucciones. Esto junto a la habilidad del sistema de apagar sectores que no estén siendo usados.
- Un sistema de varios núcleos puede llegar a operar a menor frecuencia debido a que las cargas son distribuidas en más elementos, y suelen terminar con mayor rapidez.
- Se habilitan opciones de configuración en que los núcleos tengan, cada uno, tareas distintas.
- La disponibilidad del sistema aumenta con varios núcleos.

Figura 70. **Ejemplo de sistema con varios grupos de procesadores**



Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide Version 4.0*. p. 242.

4.2.7.1. Coherencia de caché

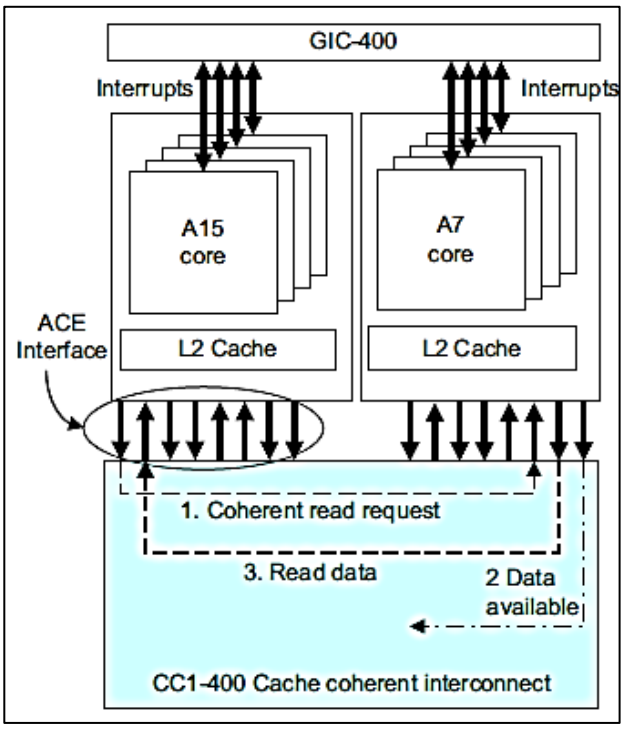
Este término se trata de hacer que los procesadores tengan la misma vista de la memoria que comparten, así cualquier cambio será reconocible para todos al mismo tiempo. Para mantener esta armonía se utilizan los mecanismos:

- Inhabilitación de accesos.
- Coherencia manejada por software, con la desventaja de uso de varios ciclos de reloj y consumo de ancho de banda en buses .
- Coherencia manejada por hardware, siendo la solución más eficiente.

El término SCU (*Snoop Control Unit*) resulta importante al hablar de grupos de procesadores. Este dispositivo controla la coherencia de cache de datos L1 entre los núcleos agrupados sin necesidad de software, permitiendo que la información sea escrita correctamente (y sin sobre escrituras indebidas) esto permite la interacción de interrupciones en todo el grupo.

En caso de existir otros grupos, se habilita el uso de ACP (*Accelerator Coherency Port*) para cumplir la función del SCU a una escala mayor. Similar a este tipo de interfaz se encuentra el CCI (*Cache Coherent Interface*) para coherencia de hardware. Ambas son características ofrecidas por AXI.

Figura 71. **Coherencia de cache en un sistema multinúcleo**



Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide Version 4.0*. p. 251.

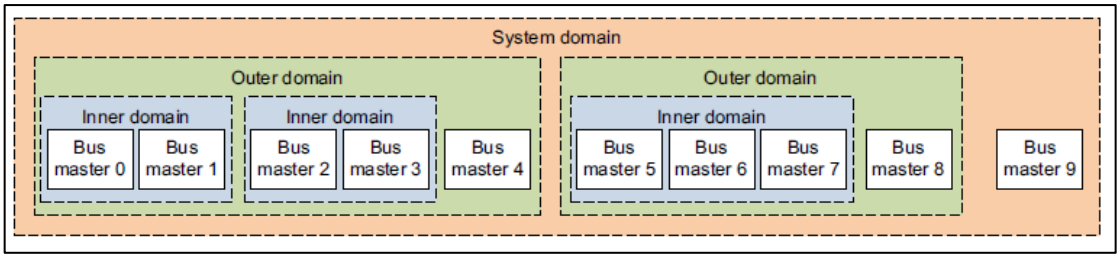
Las formas estándar en que opera la coherencia de caché para procesadores corresponde a combinaciones de las características enlistadas y que el SCU marca en cada línea:

- *Modified* (M), para las versiones más actualizadas sin versiones alternas en otras cachés.
- *Owned* (O) para contenido que posiblemente esté en otra caché. Sólo un núcleo puede tener la información con esta etiqueta.
- *Exclusive* (E), la información se presenta en caché y en memoria principal, sin otras copias en las demás cachés.

- *Shared* (S), la información está en caché y es coherente con memoria principal. Existen copias en otras cachés.
- *Invalid* (I) cuando la línea es inválida.

El SCU funciona por protocolo MOESI con optimización para reducir la cantidad de accesos a memoria externa.

Figura 72. **Dominios de coherencia de control de buses**



Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide for ARMv8-A Version 1.0*. p. 219.

4.2.7.2. Simetría

- Multiprocesamiento simétrico (SMP): este modo se forma alrededor de una arquitectura de software que dicta a cada procesador las tareas que debe ejecutar. Todas las cargas pueden correr en cualquier núcleo y el programador de sistema operativo puede cambiar atribuciones de forma dinámica. Se cuenta con coherencia de memoria y cualquier cambio es visible a todos los núcleos.
- Multiprocesamiento asimétrico (AMP): se asignan roles individuales a los núcleos para que ejecuten tareas distintas. Parece a un núcleo individual trabajando con aceleradores, cada uno teniendo vistas distintas de la memoria sin requerimiento de coherencia por hardware. Este tipo se

implementa en aplicaciones en que sea necesaria la seguridad, ejecución en tiempo real o distribución de tareas individuales.

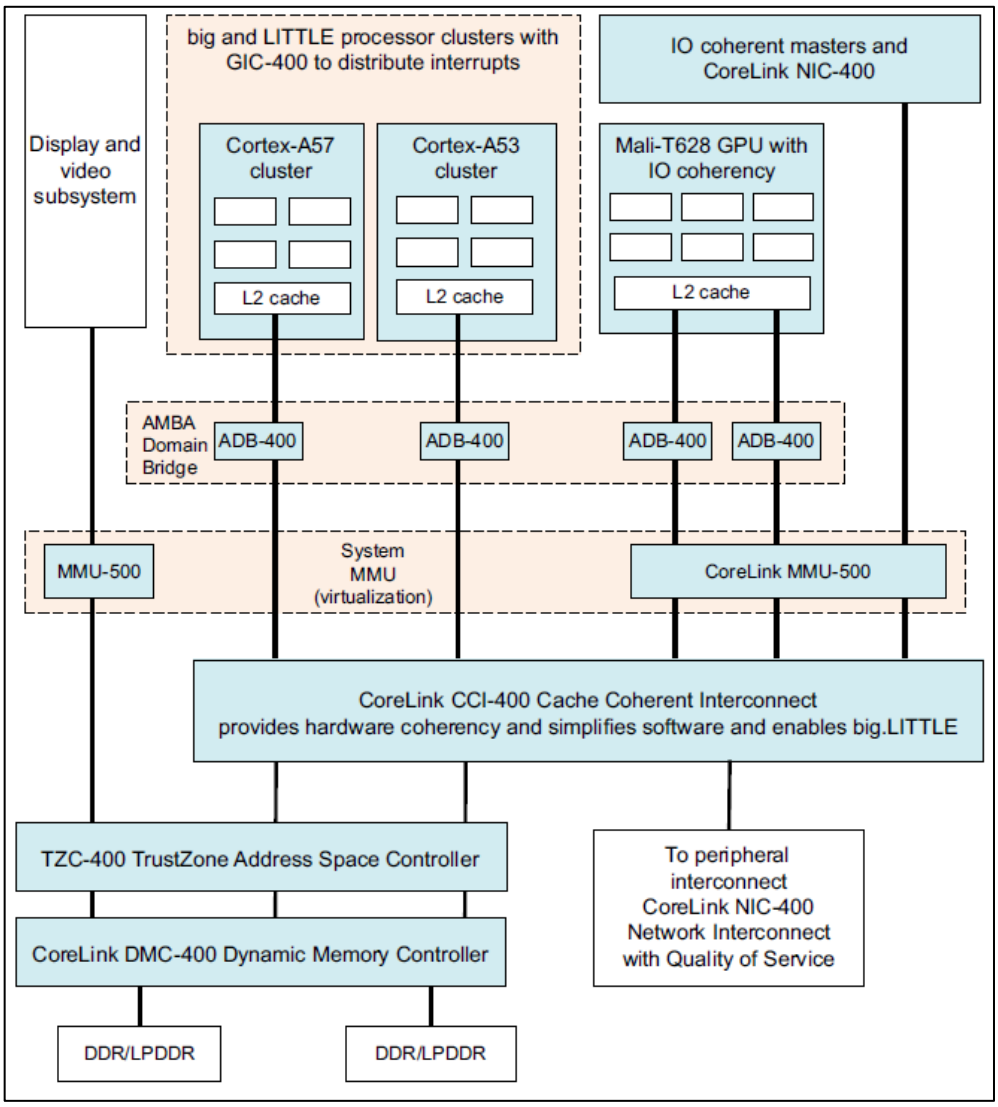
- Multiprocesamiento heterogéneo (HMP): sistema compuesto de grupos de núcleos 100 % idénticos en ISA pero distintos en microarquitectura, con completa coherencia de caché. El ejemplo más evidente de esto es la tecnología big.LITTLE.

4.2.7.3. Interconexión

Para manejar la información entre núcleos, Arm provee opciones determinadas a mantener coherencia, entre las que se encuentran:

- Interconexión con caché coherente CoreLink CCI-400
- Red con caché coherente CoreLink CCN-504
- Red con caché coherente CoreLink CCN-508
- MMU de sistema CoreLink MMU-500
- Controlador de espacio de direcciones TrustZone CoreLink TZC-400
- Controlador de memoria dinámica CoreLink DMC-400
- Interconexión de red CoreLink NIC-400

Figura 73. **Ejemplo de aplicación móvil con CoreLink**



Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide for ARMv8-A Version 1.0*. p. 222.

La figura 73 muestra, como ejemplo de interconexión, un sistema formado por Cortex-A57 y Cortex-53 en configuración big.LITTLE conectados a CCI-400 por interfaz AMBA 4 ACE para coherencia de hardware y de la unión del GPU Mali-T628.

4.2.8. big.LITTLE

Tecnología introducida en Armv7 con el fin de crear dispositivos capaces de balancear los requerimientos de alto rendimiento (para aplicaciones como videojuegos) y eficiencia en consumo de energía (para cuidado de baterías).

Los diseños tradicionales de procesadores no proveen una arquitectura capaz de cubrir por sí misma ambos campos, significa que usualmente se sacrifica la vida útil de las baterías por rendimiento aceptable en el núcleo que, al mismo tiempo, se ve aceptado por los límites de temperatura a los que el procesador puede trabajar apropiadamente.

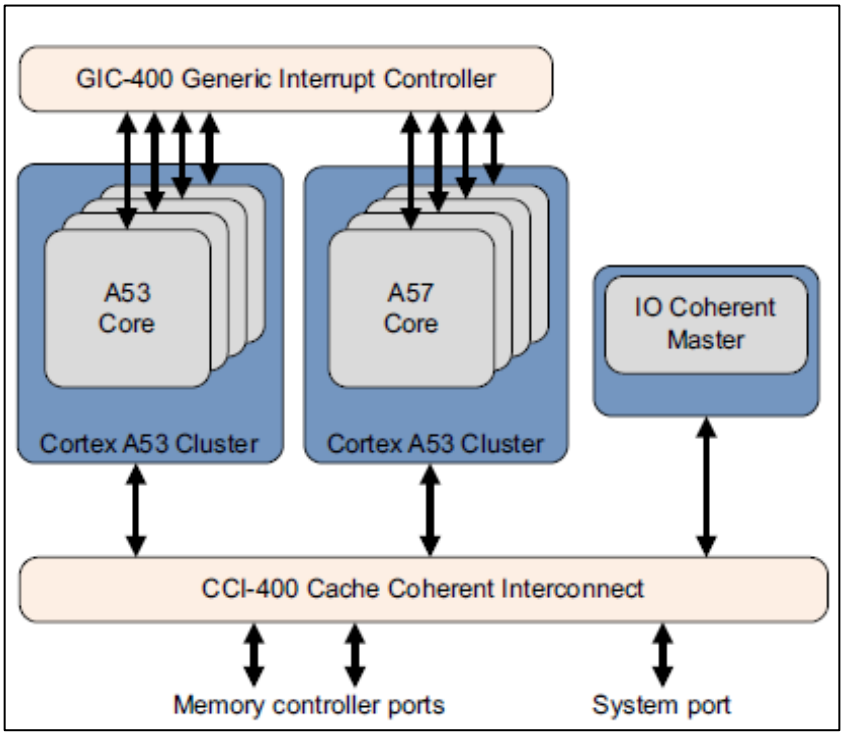
Frente a estas dificultades, big.LITTLE ofrece una solución estableciendo pares de núcleos que incluyen uno eficiente en consumo de energía (LITTLE) y uno de alto rendimiento (big), comprende un ejemplo de sistema de procesamiento heterogéneo (HMP) con la característica adicional de que sus procesadores son de propósito general con distinta microarquitectura pero son compatibles en el set de instrucciones.

Los modelos big.LITTLE requieren transferencia de información transparente entre los grupos big y LITTLE. Esto es alcanzado a través de interconexiones de coherencia de cachés como CoreLink CCI-400/500/550, en caso contrario las transferencias sucederían en memoria principal sacrificando velocidad y consumo de potencia.

Otra característica que requiere interconexión en esta tecnología es el controlador compartido de interrupciones, como CoreLink GIC-400/500.

Con el amplio catálogo de Cortex-A, es posible obtener muchas configuraciones big.LITTLE. La figura 74 presenta un ejemplo con Cortex-A57 como big y Cortex-A53 como LITTLE.

Figura 74. **Ejemplo de sistema big.LITTLE**



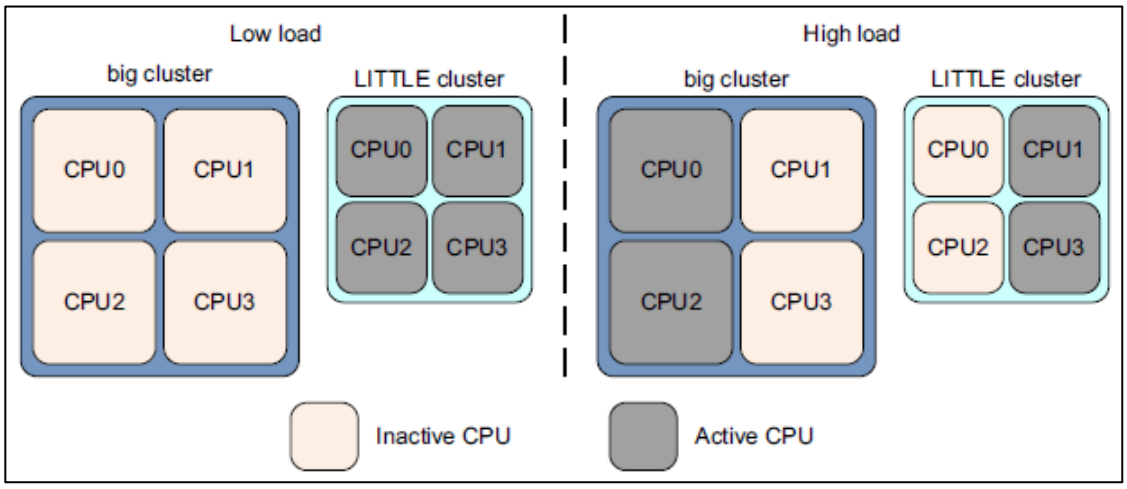
Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide for ARMv8-A Version 1.0*. p. 233.

- Modelos de ejecución big.LITTLE

    - Migración: extensión hacia la gestión de potencia y desempeño a través de técnicas como DVFS (*Dynamic Voltage and Frequency Scaling*). Esto puede darse en alguno de los siguientes tipos:

- Migración de grupo: un solo grupo se activa a la vez, excepto en el breve momento de conmutación. Por ahorro de energía, la mayoría del código corre en LITTLE y cambia a big en el momento requerido. Este modelo exige el mismo número de núcleos en ambos grupos. No es el método más utilizado debido a que no se ejecuta eficientemente con cargas desbalanceadas entre núcleos de un mismo grupo.
- Migración de CPU: cada núcleo big es emparejado con un LITTLE, con sólo uno de ellos activo a la vez según la carga y el otro apagado. Este modelo también exige la misma cantidad de núcleos en ambos grupos; sin embargo, maneja eficientemente las cargas desbalanceadas como se observa en la figura 75.

Figura 75. **Migración de CPU**

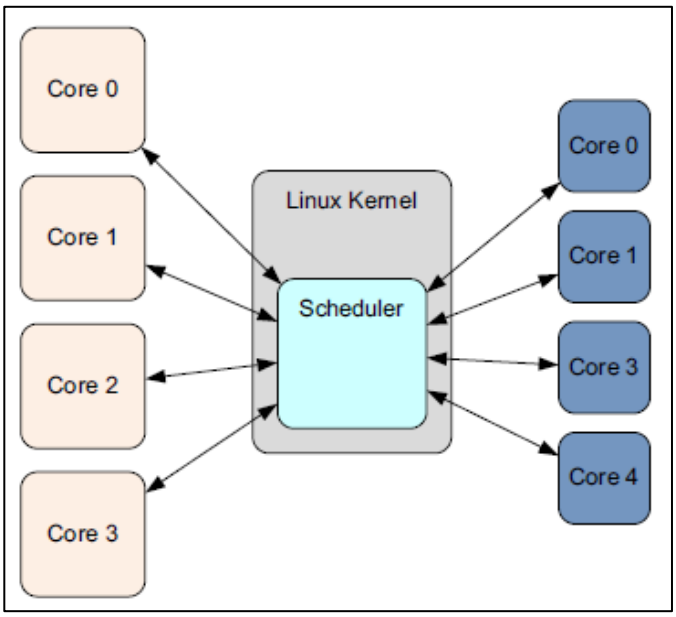


Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide for ARMv8-A Version 1.0*. p. 236.

- Programación global de tareas: en este modelo el programador de tareas del sistema operativo toma en cuenta las diferencias en capacidad de procesamiento entre los núcleos para atribuir al apropiado determinados hilos en la ejecución del código, según sus exigencias, manteniendo el resto apagados. Entre las ventajas sobre el modelo por migración tiene:

    - El sistema puede tener distintos números de núcleos big o LITTLE y a su vez, puede activarse cualquier cantidad de ellos en caso se dé un momento de máxima exigencia y sea necesario ofrecer la máxima capacidad de procesamiento.
    - Se puede aislar al grupo big para aplicación exclusiva en tareas de alta exigencia, permite completar el procesamiento pesado más rápido que teniendo hilos en segundo plano.
    - Es posible manejar interrupciones individualmente.

Figura 76. **Programación global de tareas**



Fuente: Arm Ltd. *ARM Cortex-A Series Programmer's Guide for ARMv8-A Version 1.0*. p. 236.

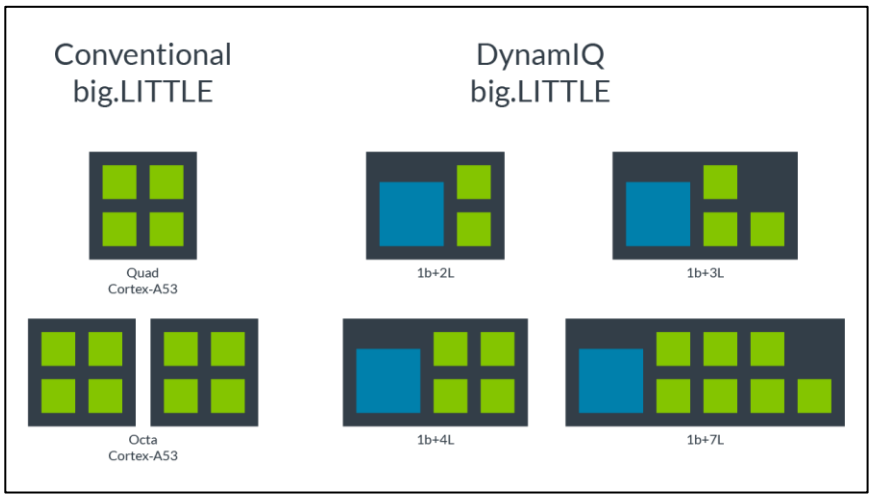
4.2.9. DynamIQ

La configuración multinúcleo en procesadores Cortex-A sin duda es gran parte del alto nivel de rendimiento que tiene la arquitectura. Con la tecnología big.LITTLE se hizo posible afrontar el mercado de smartphones con gran éxito introduciendo computación heterogénea y con la entrada de Arm DynamIQ se apunta aún más a las estructuras multinúcleo con CPUs big y LITTLE en un solo grupo optimizado para alto rendimiento y cuidado de batería.

Las características que distinguen a DynamIQ son:

- Diseño en un solo grupo: los conjuntos DynamIQ soportan desde uno a ocho núcleos en el mismo espacio, aumentando la capacidad de procesamiento.
- Computación inteligente: se optimiza la capacidad de procesamiento destinado a *machine learning* e inteligencia artificial.
- Soporte para aceleradores: se habilita la opción de acoplar los núcleos aceleradores y módulos de entradas y salidas con el propósito de ayudar en operaciones como aceleración criptográfica y procesamiento de paquetes con alto rendimiento y baja latencia.
- Opciones integradas para cuidado de batería: para gestión de la potencia se añaden a la arquitectura mecanismos para transición rápida entre estados de potencia y adaptación del tamaño de memoria disponible a los núcleos según el tipo de aplicación. Se incluye mejoras en el manejo de voltaje dinámico para responder a las necesidades de dispositivos con límites térmicos específicos.
- RAS avanzado y características de seguridad: la arquitectura es optimizada para funcionamiento en sectores en que sea necesario un alto nivel de seguridad y respuesta confinable en situaciones de inteligencia artificial.
- big.LITTLE en DynamIQ: la combinación de CPUs de alto rendimiento (big) y de alta eficiencia de energía (LITTLE) se beneficia de DynamIQ con diseños más fuertemente acoplados y flexibilidad en la topología, permitiendo distribuciones como 1 núcleo big y 7 LITTLE en el mismo grupo. Esto se aprecia mejor en la figura 77.

Figura 77. **big.LITTLE y DynamIQ**



Fuente: Arm DynamIQ Technology. *Tecnologías*.
https://developer.arm.com/technologies/dynamiq. Consulta: 15 de junio de 2018.

4.2.10. Gestión de potencia

Como se ha explicado anteriormente, los procesadores Arm se caracterizan por poner a andar dispositivos portátiles (en su mayoría), y siendo Cortex-A el perfil de más alto rendimiento se necesitan técnicas que ayuden a aumentar la vida de la batería utilizada tomando decisiones de consumo de energía. Otras razones para reducir el consumo podrían involucrar ambientalismo, costo y temperatura.

Para considerar los métodos que Arm utiliza con el fin de controlar el uso de energía, se debe especificar que su consumo se divide en dos componentes:

- Estático: también llamado fuga, suele ser el consumo proporcional al área de silicio a la que se le aplica voltaje.

- Dinámico: ocurre con las conmutaciones de transistores. Se comprende como una función de la velocidad de reloj y la cantidad de transistores cambiando su estado por ciclo (relojes más veloces y núcleos más complejos provocan mayor consumo).

Los sistemas que cuentan con gestión de potencia cambian los estados del procesador para adaptarse a las cargas y disminuir el consumo en etapas de baja exigencia. Las técnicas principales para esto son:

4.2.10.1. Gestión de inactividad

En estado de inactividad el OSPM (*Operating System Power Management*) cambia el núcleo a algún modo de baja potencia con consideraciones de latencia según la prontitud con que el procesador se necesite de nuevo.

Los modos elegidos suelen depender de todos los componentes en el chip (no sólo el núcleo) y se manejan principalmente cortando temporalmente suministros de potencia o deteniendo el reloj para suprimir el consumo dinámico.

Tabla XVIII. **Niveles de gestión de potencia Arm**

Modo	Definición
Espera	El núcleo es dejado encendido con casi todos sus relojes detenidos, dejando la mayor parte de consumo como estático. Las instrucciones utilizadas para inducir este estado son WFI (*Wait For Interrupt*) y WFE (*Wait For Event*).
Retención	El núcleo y configuraciones de depurador son apagados parcialmente con el último estado guardado para reanudar la operación.
Apagado	El suministro de energía es cortado totalmente, se hace necesario guardar el último estado y ejecutar un reinicio para reanudar. Se considera un modo destructivo del contexto.
Latente	Implementación del modo apagado con el núcleo apagado pero las cachés encendidas (usualmente en modo de retención). El reinicio suele ser más rápido.
Hotplug	Los núcleos con apagados y encendidos de forma dinámica. Su ventaja principal es dejar disponible la capacidad de computación realmente requerida.

Fuente: elaboración propia.

4.2.10.2. Escalado de frecuencia y voltaje dinámico (DVFS)

Para sistemas en que la carga es variable, la capacidad de reducir o incrementar el rendimiento del procesador puede resultar útil. DVFS es una técnica basada en la relación lineal entre consumo y frecuencia de operación con la expresión:

Tabla XIX. **Expresión de DFVS**

$P=C*V^2*f$

Fuente: elaboración propia.

Donde P es la potencia dinámica, C es la capacitancia de conmutación en el circuito lógico que se considera, V es el voltaje operacional y f es la frecuencia de operación.

El ahorro de energía es, entonces, alcanzado disminuyendo la frecuencia, con esto el núcleo puede trabajar a voltajes menores. Este último aspecto reduce la potencia estática y dinámica.

Existe una relación específica de la implementación entre el voltaje de operación y el rango de frecuencias en que puede operar formando una tuplas en cada punto conocidas como puntos de desempeño de operación (OPP). Por lo tanto, para un sistema existe un rango de OPPs que forman la llamada curva de DVFS.

La ventaja de los dispositivos que implementan DVFS es que no solo ahorran energía, sino también se mantienen dentro de límites de temperatura.

4.2.11. Procesamiento de gráficos Mali

Arm cuenta con arquitecturas de procesamiento destinado a gráficos (GPU) que suelen funcionar junto a arquitecturas Cortex-A en varios chips, aunque algunas veces pueden hacerlo por separado. Son las GPU más utilizadas en el mercado debido a que muchas empresas las han utilizado por varios años para ofrecer una experiencia de usuario con imágenes de alta calidad.

Como es usual para Arm, existen GPU Mali para todos los requerimientos. Por esta razón se dividen en tres grupos:

- De alto desempeño: sector diseñado para el más alto nivel de gráficos en dispositivos de gama alta con énfasis en la eficiencia de energía.
- De alta eficiencia de área: destinados a ocupar la menor cantidad de espacio en silicio para reducción de costo. Son utilizadas en gama media y baja.
- De ultra baja potencia: cumplen el objetivo de mantener presupuestos de potencia muy estrictos bajo control. Suelen utilizarse para IoT y portátiles como relojes inteligentes.

Tabla XX. **GPU Arm disponibles por categoría**

Alto desempeño	Alta eficiencia de área	Ultra baja potencia
Mali-G76	Mali-G52	Mali-G31
Mali-G72	Mali-G51	Mali-470
Mali-G71	Mali-T820	Mali-450
Mali-T860	Mali-T830	Mali-400
Mali-T880	Mali-T760	
	Mali-T720	

Fuente: elaboración propia.

4.2.12. Depurador

Las características básicas de depuración para Cortex-A son similares a las de Cortex-M por ser ambos perfiles parte del conjunto Arm con eventos de depuración (*breakpoints* y *watchpoints*) y hardware de rastreo (ETM).

A partir de Armv8 se optimizó el soporte para depuración en el dispositivo con uso del modelo de excepciones y para depuración con algún dispositivo externo.

CoreSight, expande la capacidad del ETM para adaptarse a sistemas multinúcleo (en AMP y SMP) en que los eventos son compartidos entre todos los núcleos para sincronización de depuración. La herramienta provee la capacidad de identificar cuellos de botella y la vista general del tiempo consumido en cada tarea con los siguientes componentes:

- *Debug Access Port* (DAP) para uso de dispositivos externos.
- *Embedded Cross Trigger* (ECT) para enlazar múltiples dispositivos en un sistema.
- Cable serial CoreSight.
- Macrocelda de rastreo de sistema (STM).
- Controlador de memoria de rastreo (TMC).

El depurador DS-5 es una de las alternativas más utilizadas para depurar procesadores Arm por sus características de manipulación de eventos, uso de imágenes, control de variables y soporte para sistemas operativos Linux y Android (con funciones como depurado hacia adelante y en reversa). Este componente permite ejecutar comandos en el IDE Eclipse, archivos y consola para control de funcionamiento.

4.3. Sets de instrucciones

De la misma forma en que el resto de arquitecturas Arm se agrupan, Cortex-A contiene en cada versión diseños que se adaptan a las actualizaciones y las necesidades específicas del mercado.

Tabla XXI. **Grupos de arquitecturas Cortex-A**

Arquitectura	Diseños
Armv7-A	Cortex-A5 Cortex-A7 Cortex-A8 Cortex-A9 Cortex-A12 Cortex-A15 Cortex-A17
Armv8-A	Cortex-A32 Cortex-A35 Cortex-A53 Cortex-A55 Cortex-A57 Cortex-A65AE Cortex-A72 Cortex-A73 Cortex-A75 Cortex-A76 Cortex-A76AE

Fuente: elaboración propia.

La clasificación de la tabla XXI, se considera que los Cortex-A se agrupan según su capacidad de procesamiento en series que pueden, o no, coincidir con la numeración dada anteriormente.

4.3.1. Serie Cortex-A7x

Destinados a alto rendimiento están:

- Cortex-A76
- Cortex-A76AE
- Cortex-A75
- Cortex-A73
- Cortex-A72

- Cortex-A57
- Cortex-A65AE
- Cortex-A17
- Cortex-A15

4.3.2. Serie Cortex-A5x

En gama media se encuentran:

- Cortex-A55
- Cortex-A53
- Cortex-A9
- Cortex-A8

4.3.3. Serie Cortex-A3x

Los más pequeños y de menor consumo son:

- Cortex-A35
- Cortex-A32

4.4. Subclasificaciones

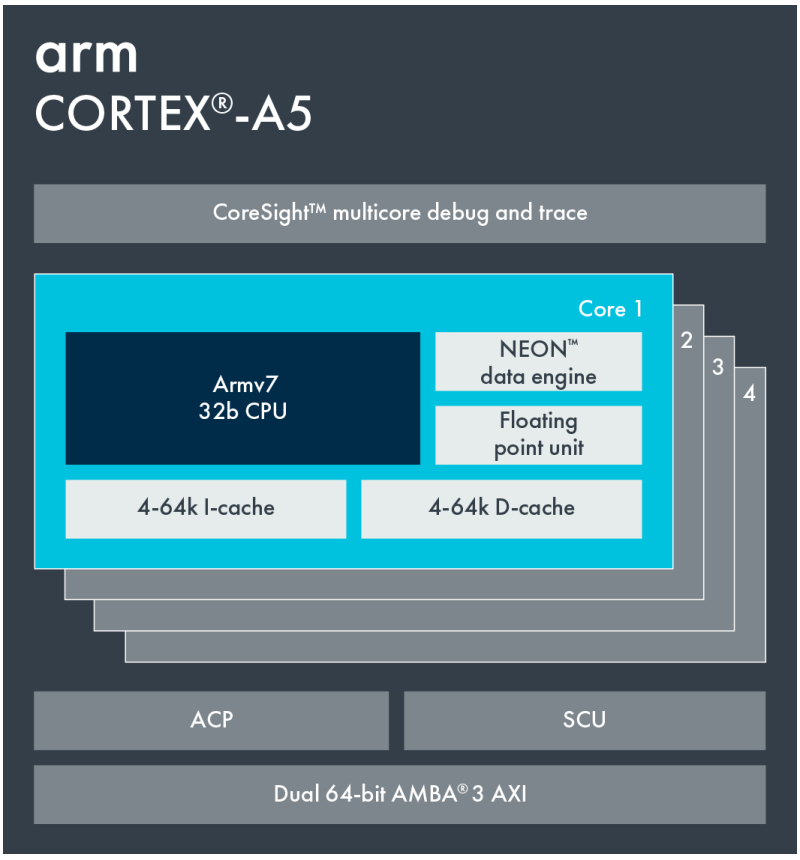
4.4.1. Cortex-A5

El procesador más pequeño y de menor consumo de potencia en el grupo de Armv7-A, lo hace apropiado para aplicaciones que necesiten manejo eficiente de memoria virtual y funcionalidad de alto nivel en su núcleo pero, al

mismo tiempo, cuidado en el uso de la energía. Cuenta con opción de licencias *uniprocessor* (UP) y *multiprocessor* (MP).

Dentro de Arm destaca por tener todas las características de Cortex-A9 a un tercio del área en silicio, disminuyendo su costo, disipación de calor y aumento de la vida en baterías para portátiles.

Figura 78. **Componentes de arquitectura ARM Cortex-A5**



Fuente: Cortex. *Procesador Cordex-A5*. https://developer.arm.com/products/processors/cortex-a/cortex-a5. Consulta: 7 de mayo de 2018.

Esta arquitectura posee como características principales:

- Capacidad de funcionamiento *multicore* (de 1 a 4 núcleos).
- Soporte de subsets Thumb y Thumb2 (con ThumbEE) en su ISA.
- Soporte para SIMD por extensión NEON avanzada.
- Extensión DSP opcional.
- Punto flotante opcional por VFPv4 conforme estándar IEEE754.
- Gestión de memoria por *Memory Management Unit* (MMU) de Armv7.
- Pipeline de 8 etapas con predicción dinámica de saltos y arquitectura de memoria Harvard.
- Extensión de seguridad TrustZone.
- Interfaz AXI de 64 bits.
- Arquitectura de depuración Armv7 por interfaz APB CoreSight con hasta 3 *breakpoints* y 2 *watchpoints.*
- Soporte de rastreo mediante interfaz ETM.
- MPE (*Media Processing Engine*) opcional con tecnología NEON.
- Aceleración por hardware opcional para Jazelle.
- Interfaz DFT (*Design For Test*).
- Extensiones de virtualización compatibles con Cortex-A9.
- Tamaño de caché de instrucciones configurable para 4KB, 8KB, 16KB, 32KB, o 64KB.
- Tamaño de caché de datos configurable para 4KB, 8KB, 16KB, 32KB, o 64KB.

Las aplicaciones de este procesador abarcan los campos de televisión digital, domótica, portátiles, relojes inteligentes, nodos IoT y reproductores de audio y DVD.

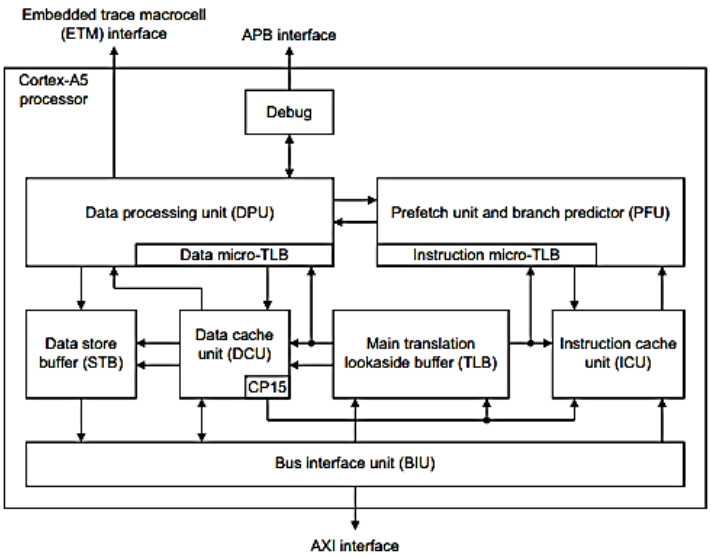
La arquitectura Cortex-A5 es totalmente compatible con Cortex-A9, teniendo entre 70 % y 80 % de su rendimiento y casi el doble de eficiencia en consumo de potencia.

Todo esto es alcanzado principalmente por alta configurabilidad, siendo un ejemplo claro el caso en que el diseño es su configuración más pequeña (con caches de 4KB) resultando en un diseño de 0,2 mm^2 de tamaño total en tecnología de 28 nm.

El procesador Cortex-A5 está diseñado para implementarse en un SoC, contando con la capacidad de incorporar propiedad intelectual Arm más allá de la del núcleo (de gráficos, de sistema, física). Las IP relacionadas con este procesador incluyen:

- GPU Mali-400
- Familia de interconexión de red CoreLink
- Controladores de interrupciones
- CoreSight SoC-400
- POP IP

Figura 79. **Implementación Cortex-A5**



Fuente: Arm Ltd. *Cortex-A5 Revision:r0p1 Technical Reference Manual*. p. 29.

Los componentes de implementación de Cortex-A5 incluyen:

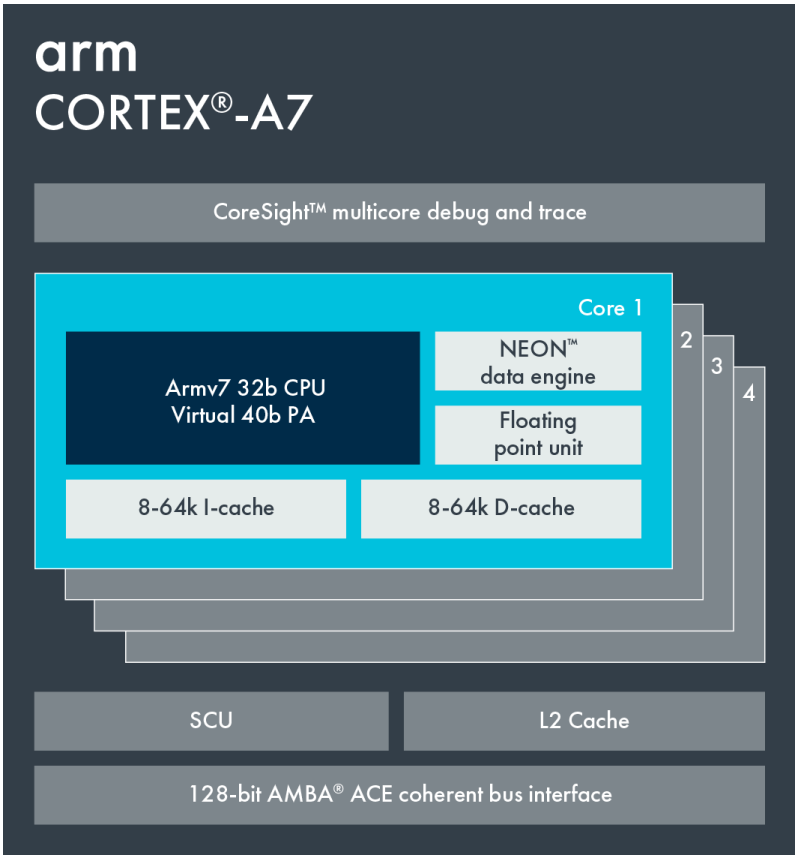
- Unidad de procesamiento de datos (DPU)
- Coprocesador de control de sistema
- Sistema de memoria lateral de instrucciones
- Sistema de memoria lateral de datos
- Sistema de memoria L1
- Interfaces AXI L2
- Motor de procesamiento de media
- Unidad de punto flotante
- Depurador
- Monitoreo de desempeño
- Extensiones de virtualización

4.4.2. Cortex-A7

Este procesador se encuentra en un punto medio entre Cortex-A5 (con un rendimiento mayor en 20 %) y el conjunto Cortex-A15/A17 (conteniendo sus características de gama alta).

Cortex-A7 es un procesador destinado a dispositivos móviles portátiles y otros productos dirigidos a consumidor basados en interfaz de usuario.

Figura 80. **Componentes de arquitectura ARM Cortex-A7**



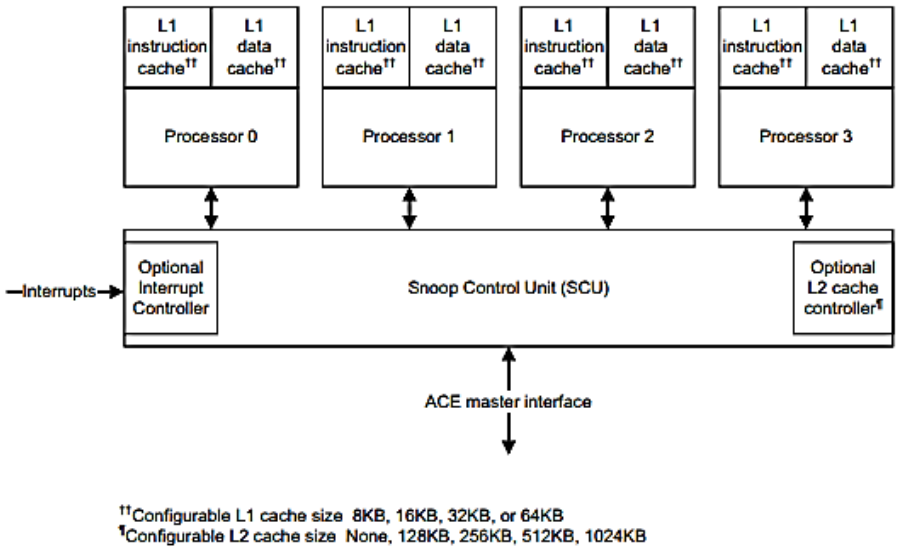
Fuente: Cortex. *Procesador Cordex-A7*. https://developer.arm.com/products/processors/cortex-a/cortex-a7. Consulta: 7 de mayo de 2018.

Esta arquitectura posee como características principales:

- Capacidad de funcionamiento *multicore* (de 1 a 4 núcleos por grupo) con soporte para 1 o 2 grupos.
- Soporte de subsets Thumb y Thumb2 (con ThumbEE) en su ISA.
- Soporte para SIMD por extensión NEON avanzada.
- Extensión DSP.
- Punto flotante opcional por VFPv4 conforme estándar IEEE754 con 16 registros de precisión doble sin trabajo en conjunto con extensión NEON y 32 registros con ella.
- Gestión de memoria por *Memory Management Unit* (MMU) de Armv7.
- *Pipeline* de 8 etapas con predicción dinámica de saltos y arquitectura de memoria Harvard.
- *Dual-issue* mejorado.
- Extensión de seguridad TrustZone.
- Interfaz AMBA 4 AXI de 128 bits.
- Arquitectura de depuración Armv7 por interfaz APB CoreSight con hasta 6 *breakpoints* y 4 *watchpoints.*
- Soporte de rastreo mediante interfaz ETM.
- Interfaz DFT (*Design For Test*).
- Soporte para virtualización por hardware y LPAE (Large Physical Address Extensions), que permiten acceso a 1TB de memoria con direcciones físicas de hasta 40 bits.
- Tamaño de caché de instrucciones configurable para 8KB, 16KB, 32KB, o 64KB.
- Tamaño de caché de datos configurable para 8KB, 16KB, 32KB, o 64KB.
- Caché L2 opcional.

La arquitectura Cortex-A7 se presenta como un *MPCore* (múltiples procesadores), permitiendo la existencia desde 1 a 4 núcleos ordenados por SMP (*Symmetric Multiprocessing*) en un solo grupo de procesadores o varios grupos con coherencia a través de tecnología AMBA 4. La configuración con cuatro procesadores puede observarse a continuación:

Figura 81. **Configuración *quad-core* de Cortex-A7**



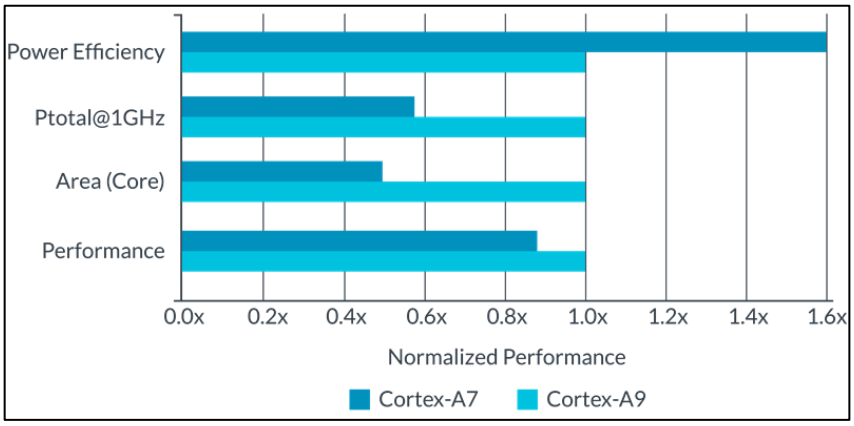
Fuente: Arm Ltd. *Cortex-A7 MPCore Revision:r0p5 Technical Reference Manual*. p. 13.

Como se mencionó anteriormente, esta arquitectura está orientada a portátiles con balance apropiado entre desempeño y consumo de potencia. Aplicaciones en el grupo incluyen SBCs (*Smartwatch Single board computers)*, portátiles, dispositivos nodos IoT alámbricos e inalámbricos e infrastructura de red.

Fue el primer diseño de procesador LITTLE compatible arquitectónicamente con Cortex-A15 y 17 para combinaciones de procesadores big.LITTLE.

En tecnología de 28 nm la arquitectura ocupa un área de 0,45 mm^2 en la configuración que incluye unidad de punto flotante, NEON y cache L1 de 32KB. Esto, junto a los beneficios de caché L2 de bajo consumo y baja latencia permiten entregar un rendimiento similar a Cortex-A9 con niveles mucho más altos de eficiencia de energía a frecuencias de reloj entre 1,2 y 1,6 GHz.

Figura 82. **Comparación de características Cortex-A7 y A9**

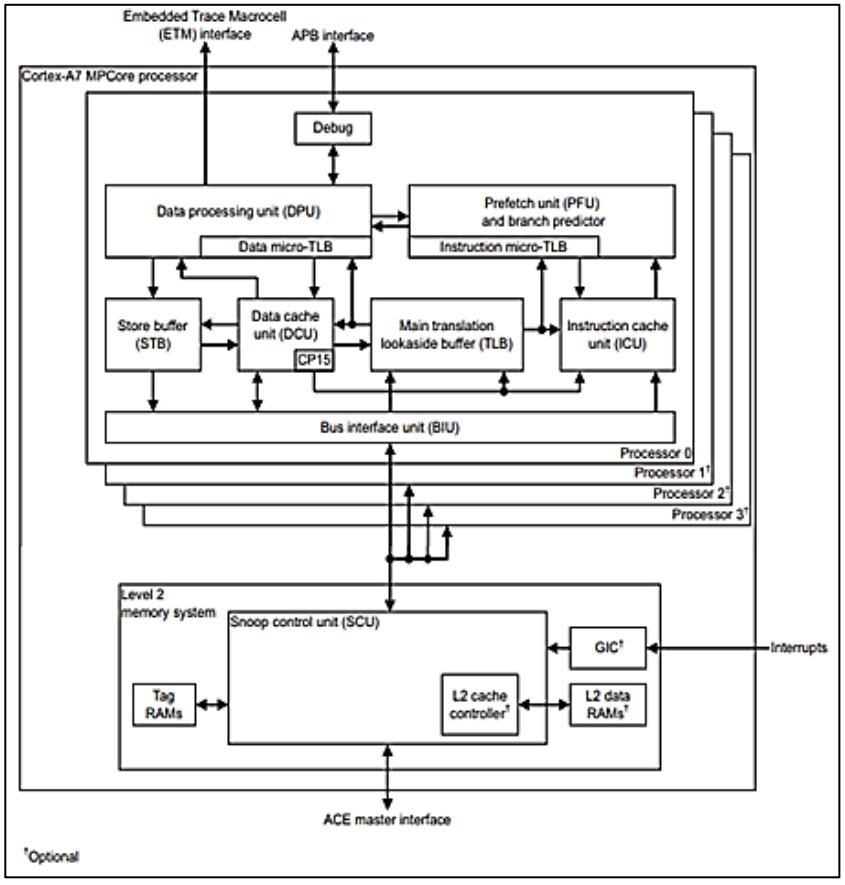


Fuente: Cortex. *Procesador Cordex-A7*. https://developer.arm.com/products/processors/cortex-a/cortex-a7. Consulta: 7 de mayo de 2018.

El procesador Cortex-A7 está diseñado para implementarse en un SoC, contando con la capacidad de incorporar propiedad intelectual Arm más allá de la del núcleo (de gráficos, de sistema, física). Las IP relacionadas con este procesador incluyen:

- GPUs Mali
- Procesadores de monitoreo Mali
- Procesador de vídeo Mali-V500
- Familia de interconexión de red CoreLink
- Controladores de sistema CoreLink
- Controladores de interrupciones
- CoreSight SoC-400
- POP IP

Figura 83. **Implementación Cortex-A7**



Fuente: Arm Ltd. *Cortex-A7 MPCore Revision: r0p5 Technical Reference Manual*. p. 24.

Los componentes de Cortex-A7 incluyen:

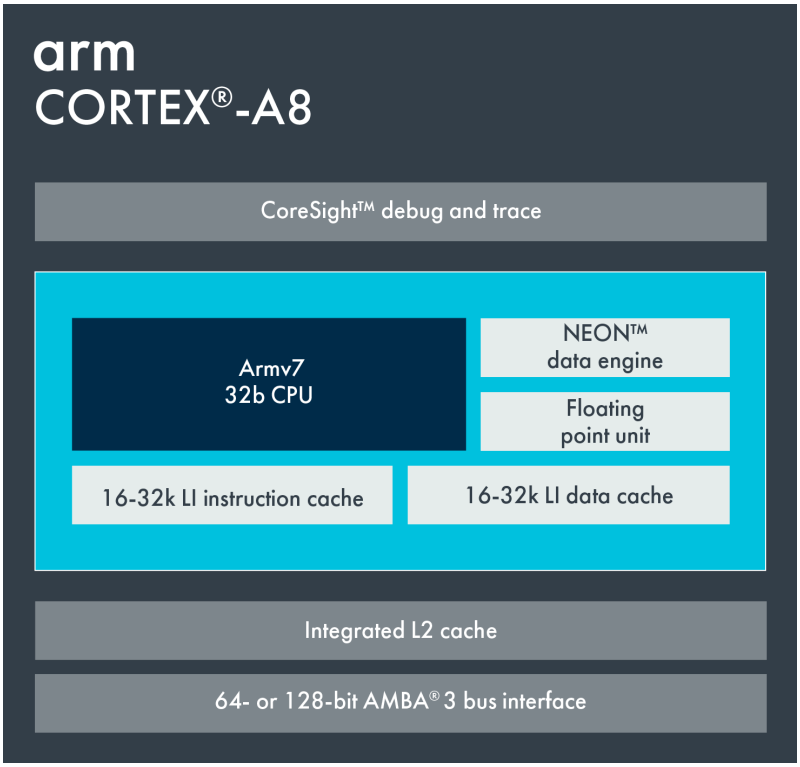
- Unidad de procesamiento de datos (DPU)
- Coprocesador de control de sistema
- Sistema de memoria lateral de instrucciones
- Sistema de memoria lateral de datos
- Sistema de memoria L1
- Motor de procesamiento de media
- Unidad de punto flotante
- Sistema de memoria L2 con *Snoop Control Unit* (SCU)
- Depurador
- Monitoreo de desempeño
- Extensiones de virtualización
- Extensión ETM

4.4.3. Cortex-A8

El procesador Cortex-A8 es un diseño de alto rendimiento, bajo consumo y soporte para memoria virtual con frecuencias de reloj desde 600MHz hasta más de 1GHz.

Fue la primera arquitectura de Armv7-A y aún se utiliza en aplicaciones embebidas diversas dado que ofreció una alternativa mejorada sobre Arm11 con la inclusión de NEON, TrustZone y Thumb-2 (que fueron adiciones clave).

Figura 84. **Componentes de arquitectura ARM Cortex-A8**



Fuente: Cortex. *Procesador Cordex-A8*. https://developer.arm.com/products/processors/cortex-a/cortex-a8. Consulta: 11 de mayo de 2018.

Esta arquitectura posee como características principales:

- Capacidad de funcionamiento únicamente en núcleo individual.
- Soporte de subsets Thumb y Thumb2 (con ThumbEE) en su ISA.
- Soporte para SIMD por extensión NEON avanzada con *pipeline* de 10 etapas.
- Extensión de seguridad TrustZone.
- Punto flotante opcional por VFPv3 conforme estándar IEEE754.

- Gestión de memoria por *Memory Management Unit* (MMU) de Armv7 con TLBs separadas para instrucciones y datos de 32 entradas cada una.
- *Pipeline* superescalar de 13 etapas o más con predicción dinámica de saltos y arquitectura de memoria Harvard.
- Arquitectura de depuración Armv7 por interfaz APB CoreSight de 32 bits.
- Soporte para Jazelle-RCT.
- Interfaz AMBA AXI de 64 o 128 bits.
- Tamaño de caché L1 de instrucciones y de datos configurable para 16KB, 32KB.
- Tamaño de caché L2 de datos configurable para 0KB,128KB, 256KB, 512KB o 1MB.
- Caché L2 con paridad y ECC (*Error Correction Code*) opcional.
- Soporte de rastreo mediante interfaz ETM.
- Gestión de potencia dinámica y estática con inclusión de IEM (*Intelligent Energy Management*).

Cortex-A8 fue diseñado para suplir las necesidades de un mercado de *smartphone*s con expansión veloz y exigencia en características especializadas para desempeño de móviles. La telefonía no es el único campo cubierto, otras aplicaciones incluyen dispositivos de bajo costo para consumo, dispositivos de red caseros, computación embebida, impresoras y almacenamiento (discos duros y de estado sólido).

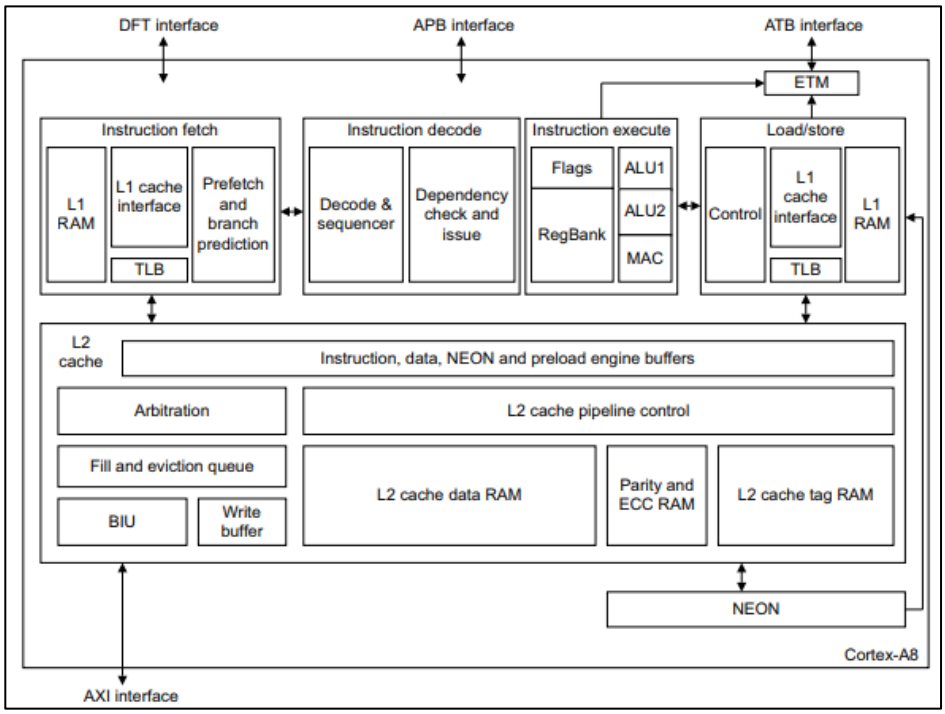
La arquitectura se presentó como el primer diseño superescalar de Arm que, junto a la simetría del pipeline, ofrece capacidad de *dual-issue* y trabajo a altas frecuencias.

El procesador Cortex-A8 está diseñado para implementarse en un SoC, contando con la capacidad de incorporar propiedad intelectual Arm más allá de

la del núcleo (de gráficos, de sistema, física). Las IP relacionadas con este procesador incluyen:

- GPU Mali-400
- Procesadores de monitoreo Mali
- Procesador de vídeo Mali-V500
- Familia de interconexión de red CoreLink
- Controladores de sistema CoreLink
- Controladores de interrupciones
- CoreSight SoC-400
- POP IP

Figura 85. **Implementación Cortex-A8**



Fuente: Arm Ltd. *Cortex-A8 Revision:r3p2 Technical Reference Manual*. p. 29.

Los componentes de Cortex-A8 incluyen:

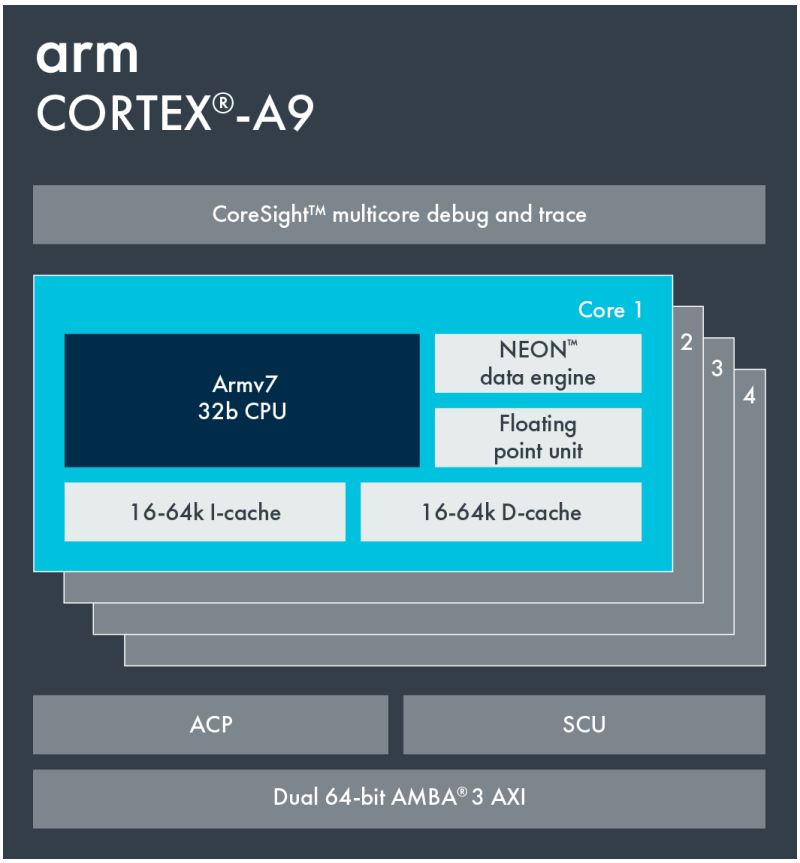
- Bloque de obtención de instrucciones
- Bloque de decodificación de instrucciones
- Bloque de ejecución de instrucciones
- Bloque de carga y almacenamiento
- Sistema de memoria lateral de datos
- Sistema de memoria L2
- Unidad de punto flotante
- Extensión NEON
- Extensión ETM

4.4.4. Cortex-A9

El procesador Cortex-A9 es un diseño MP optimizado en potencia y desempeño, lo hace uno de los más utilizados en el catálogo de Arm.

Cuenta con características flexibles y configurables que le permiten alcanzar rendimiento 50 % mayor al de Cortex-A8 en configuración de un solo núcleo, haciéndolo útil para aplicaciones de 32 bits móviles.

Figura 86. **Componentes de arquitectura ARM Cortex-A9**



Fuente: Cortex. *Procesador Cordex-A9*. https://developer.arm.com/products/processors/cortex-a/cortex-a9. Consulta: 11 de mayo de 2018.

Esta arquitectura posee como características principales:

- Capacidad de funcionamiento *multicore* (de 1 a 4 núcleos).
- Soporte de subsets Thumb y Thumb2 (con ThumbEE) en su ISA.
- Soporte para SIMD por extensión NEON avanzada opcional.
- Extensión DSP.
- Punto flotante opcional por VFPv3 conforme estándar IEEE754.

- Gestión de memoria por *Memory Management Unit* (MMU) de Armv7.
- Soporte para Jazelle-RCT/DBX.
- Extensión de seguridad TrustZone.
- Arquitectura de depuración Armv7 por interfaz APB CoreSight con hasta 6 *breakpoints* y 4 *watchpoints.*
- *Pipeline* superescalar de 9 a 12 etapas de ejecución fuera de orden con predicción dinámica de saltos y arquitectura de memoria Harvard.
- *Dual-issue* mejorado.
- Tamaño de caché de instrucciones configurable para 16KB, 32KB, o 64KB.
- Tamaño de caché de datos configurable para 16KB, 32KB, o 64KB.
- Interfaz AMBA 3 AXI de 64 bits.
- Soporte de coherencia de sistema utilizando ACP (*Accelerator Coherency Port*).
- Soporte de rastreo mediante interfaz PTM (Program Trace Macrocell).

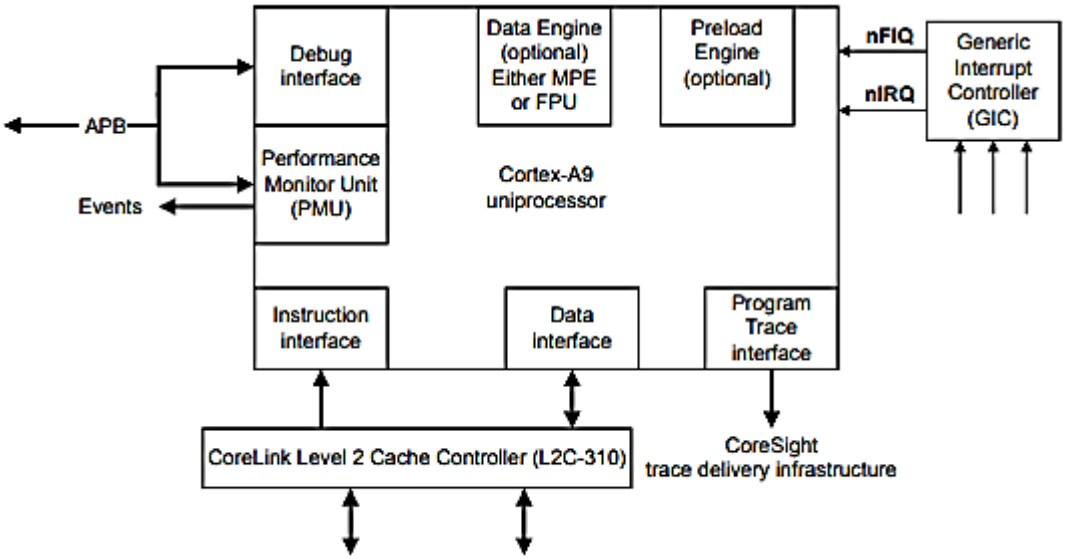
Las aplicaciones de esta arquitectura incluyen, en mayoría, procesadores para telefonía móvil aunque también se suman al grupo dispositivos de bajo costo para consumo, dispositivos de computación embebida de 32 bits y dispositivos de red caseros.

El procesador Cortex-A9 está diseñado para implementarse en un SoC, contando con la capacidad de incorporar propiedad intelectual Arm más allá de la del núcleo (de gráficos, de sistema, física). Las IP relacionadas con este procesador incluyen:

- GPUs Mali
- Procesadores de monitoreo Mali
- Procesador de vídeo Mali-V500

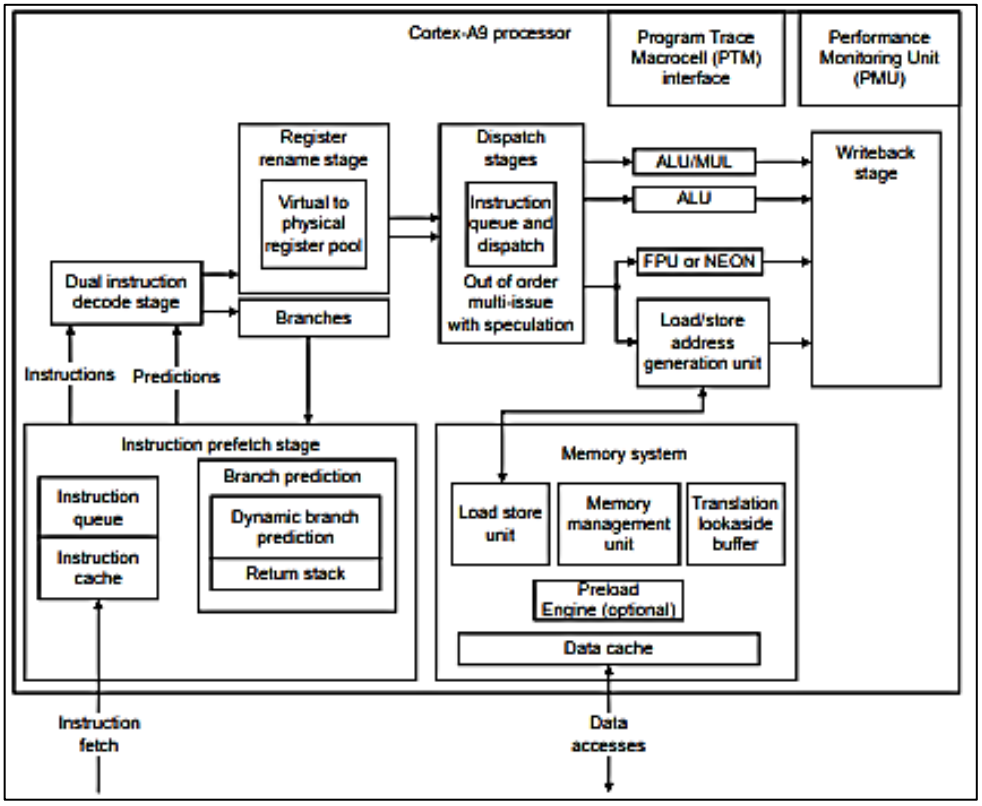
- Familia de interconexión de red CoreLink
- Controladores de sistema CoreLink

Figura 87. **Implementación Cortex-A9 como uniprocesador**



Fuente: Arm Ltd. *Cortex-A9 Revision:r4p1 Technical Reference Manual*. p. 19.

Figura 88. **Implementación Cortex-A9**



Fuente: Arm Ltd. *Cortex-A9 Revision:r4p1 Technical Reference Manual*. p. 32.

Los componentes de Cortex-A9 incluyen:

- Motor de procesamiento de media.
- Unidad de punto flotante.
- Sistema de memoria L2 con *Snoop Control Unit* (SCU).
- Control de interrupciones mediante PL390 (*PrimeCell Generic Interrupt Controller*) con soporte para dispositivos anteriores.
- Controlador de caché L2 CoreLink.
- Depurador.

- Monitoreo de desempeño.
- Extensiones de virtualización.
- Extensión PTM.
- Sistema de memoria L1.

4.4.5. Cortex-A12

Las consideraciones para este procesador se hacen junto a las de Cortex-A17.

Al inicio ambas arquitectura fueron presentadas como diseños individuales con características por separado. Sin embargo, debido a la constante actualización como respuesta a la demanda del mercado, se introdujo en la segunda revisión de Cortex-A12 mejoras que le permitían alcanzar niveles de desempeño similares al A17, haciéndolos prácticamente indistinguibles.

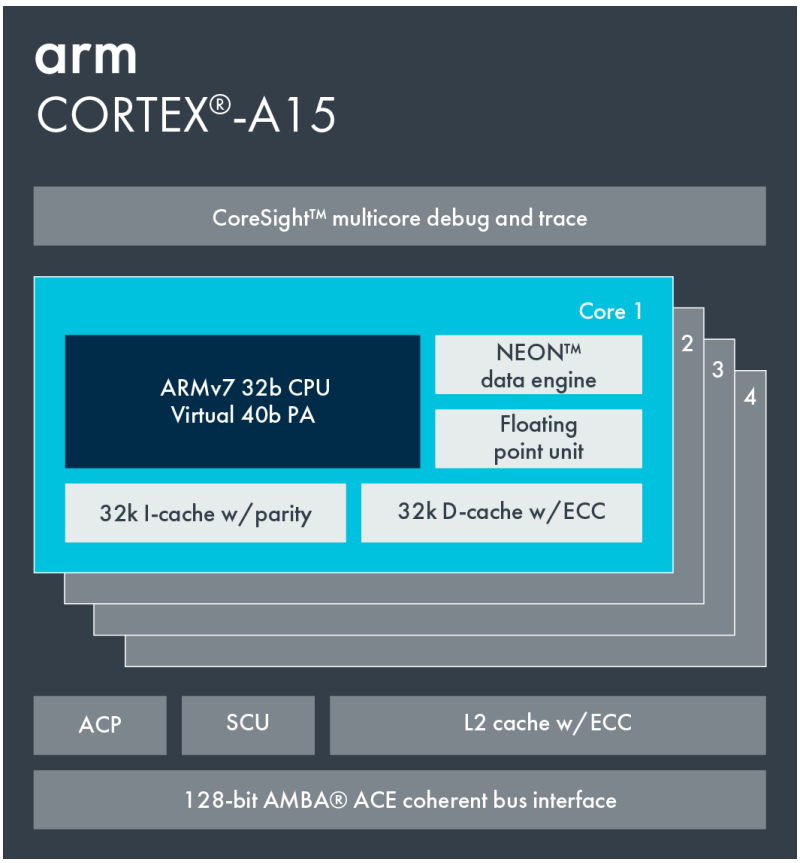
Por lo mencionado, y para facilitar el uso de documentación, Arm decidió entonces aplicar el nombre Cortex-A17 a ambas arquitecturas, y las referencias a Cortex-A12 desaparecieron y el soporte necesario se encuentra en los manuales de referencia de A17.

4.4.6. Cortex-A15

Uno de los diseños más exitosos de Arm con 50 millones de unidades comprendidas en dispositivos móviles y aplicaciones, siendo el de mayor rendimiento en el grupo. Fue el primer diseño en funcionar con Cortex-A7 en configuración big.LITTLE, habilitando la posibilidad de ofrecer múltiples soluciones; sin embargo, aun en configuración de un solo núcleo, el Cortex-A15

alcanza un rendimiento 50 % mayor a Cortex-A9 en funciones clave permitiendo implementarlo en dispositivos que necesiten ejecuciones a altas frecuencias.

Figura 89. **Componentes de arquitectura ARM Cortex-A15**



Fuente: Cortex. *Procesador Cordex-A15*. https://developer.arm.com/products/processors/cortex-a/cortex-a15. Consulta: 11 de mayo de 2018.

Esta arquitectura posee como características principales:

- Capacidad de funcionamiento *multicore* (de 1 a 4 núcleos por grupo) con soporte para 1 hasta 4 grupos.
- Soporte de subsets Thumb y Thumb2 (con ThumbEE) en su ISA.

- Soporte para SIMD por extensión NEON avanzada opcional.
- Extensión DSP.
- Punto flotante opcional por VFPv4 conforme estándar IEEE754.
- Gestión de memoria por *Memory Management Unit* (MMU) de Armv7.
- Extensión de seguridad TrustZone.
- Arquitectura de depuración Armv7 por interfaz APB CoreSight con hasta 6 *breakpoints* y 4 *watchpoints.*
- Soporte de virtualización por hardware.
- *Pipeline* superescalar de 15 etapas de ejecución fuera de orden con predicción dinámica de saltos y arquitectura de memoria Harvard.
- *Triple-issue* mejorado.
- Soporte para virtualización por hardware y LPAE (Large Physical Address Extensions), que permiten acceso a 1TB de memoria con direcciones físicas de hasta 40 bits.
- Cachés L1 de instrucciones y datos de 32KB cada una.
- Caché L2 de tamaño configurable entre 512KB, 1MB, 2MB o 4MB.
- Compatibilidad para caché L3 opcional.
- Protección para caché L1 y L2 mediante ECC opcional.
- Instrucciones MAC fusionadas.
- Instrucciones depurador en modo de hipervisor.
- Interfaz AMBA 4 AXI de 64 bits.
- Soporte de coherencia de sistema utilizando ACP (*Accelerator Coherency Port*).

El grupo de los Cortex-A15 está diseñado para aplicaciones que requieran procesamiento de 32 bits de alto rendimiento con las ventajas que Arm ofrece en gestión de potencia. Estos mercados incluyen equipo de entretenimiento casero de gama alta, dispositivos de red alámbricos e inalámbricos,

smartphones de bajo costo, automatización industrial, controladores de almacenamiento embebido y computación móvil.

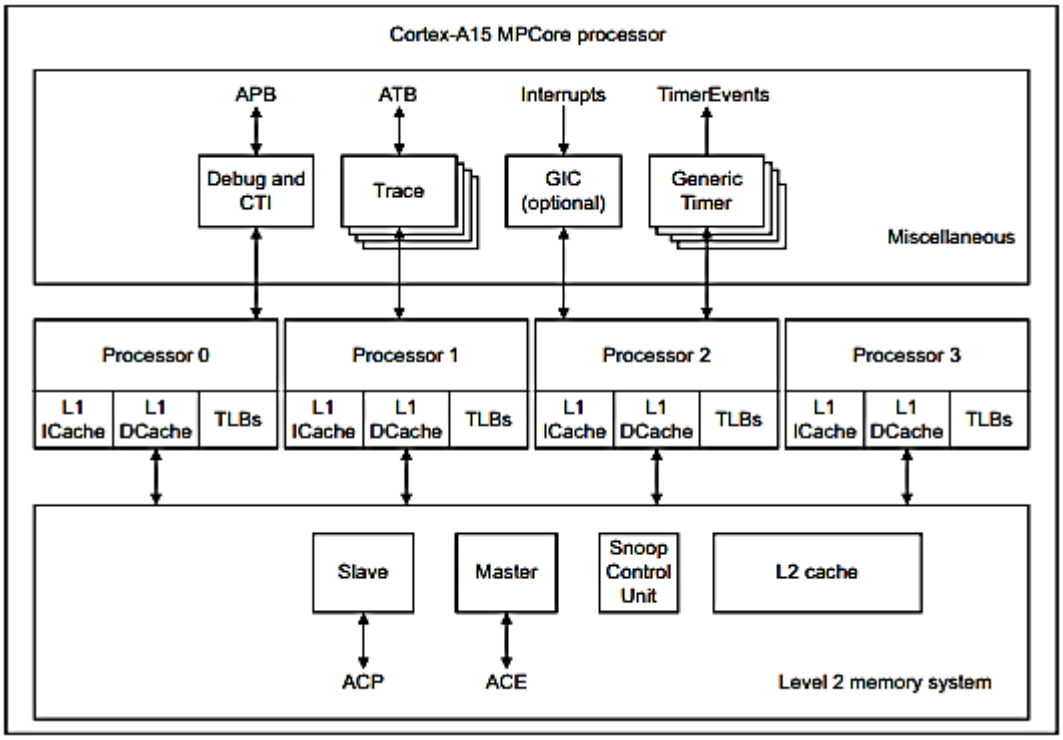
El *pipeline* de esta arquitectura permite que el procesador responda a las exigencias de computación móvil en la actualidad, donde usualmente se debe dar solución a dos grupos opuestos: alta sensibilidad o *frame rate* (para aplicaciones como videojuegos o navegación por internet) y cuidado de la vida de la batería (para experiencia de usuario ininterrumpida).

El procesador Cortex-A15 está diseñado para implementarse en un SoC, contando con la capacidad de incorporar propiedad intelectual Arm más allá de la del núcleo (de gráficos, de sistema, física). Las IP relacionadas con este procesador incluyen:

- GPU Mali-450
- Procesadores de monitoreo Mali
- Procesador de vídeo Mali-V500
- Familia de interconexión de coherencia en caché CoreLink
- Controladores de memoria
- Controladores de sistema CoreLink
- Controladores de interrupciones
- CoreSight SoC-400
- POP IP

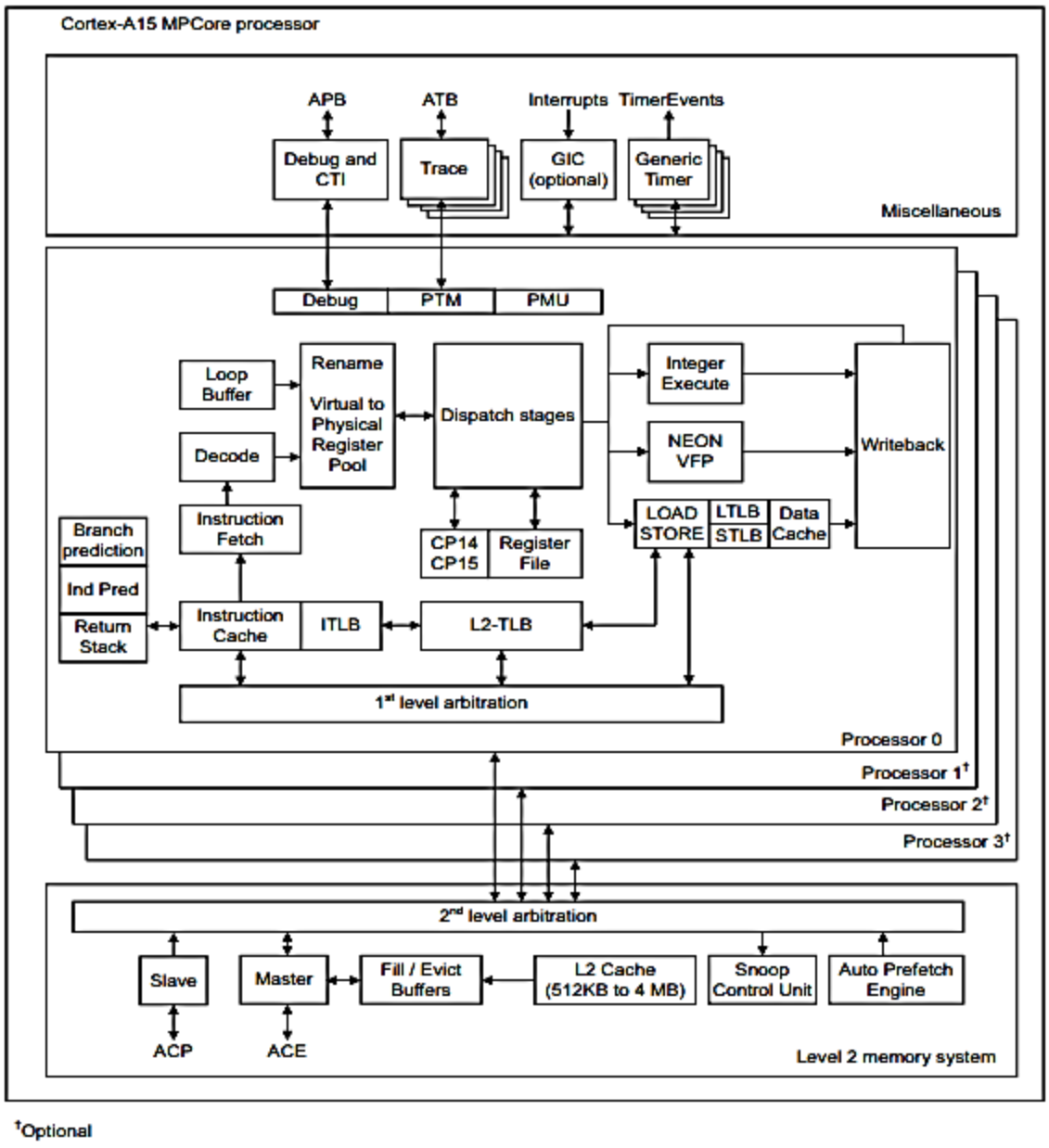
La arquitectura Cortex-A15 se presenta como un *MPCore* (múltiples procesadores), permitiendo la existencia desde 1 a 4 núcleos ordenados por SMP (*Symmetric Multiprocessing*) en un solo grupo de procesadores o varios grupos con coherencia a través de tecnología AMBA 4. La configuración con cuatro procesadores se observa en la figura 90.

Figura 90. **Implementación *quad-core* de Cortex-A15**



Fuente: Arm Ltd. *Cortex-A15 MPCore Revision:r4p0 Technical Reference Manual*. p. 13.

Figura 91. **Implementación Cortex-A15**



Fuente: Arm Ltd. *Cortex-A15 MPCore Revision:r4p0 Technical Reference Manual*. p. 27.

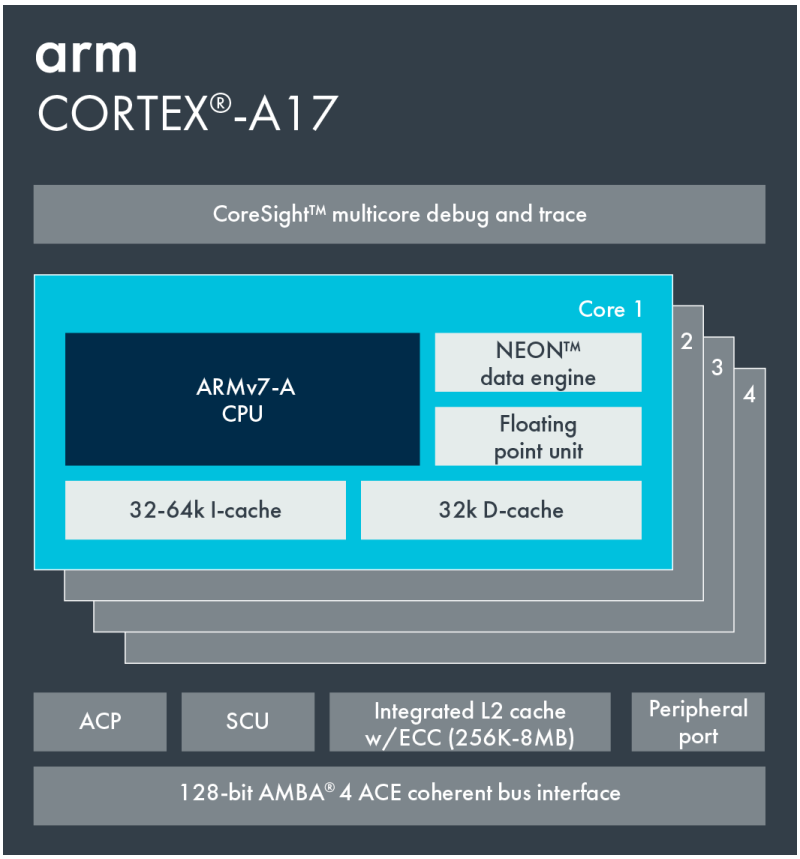
Los componentes de Cortex-A15 incluyen:

- Bloque de obtención de instrucciones
- Bloque de decodificación de instrucciones
- Bloque de despacho de instrucciones
- Bloque de ejecución entera
- Bloque de carga y almacenamiento
- Sistema de memoria L2
- Unidad NEON y VFP
- Controlador genérico de interrupciones
- Temporizador genérico
- Depuración y rastreo

4.4.7. Cortex-A17

Este diseño está orientado a aplicaciones de presupuesto de potencia de gama media, dado que cuenta con características similares a Cortex-A9 pero rendimiento mayor. Presente en el mercado de embebidos de 32 bits en masa.

Figura 92. **Componentes de arquitectura ARM Cortex-A17**



Fuente: Cortex. *Procesador Cortex-A17*. https://developer.arm.com/products/processors/cortex-a/cortex-a17. Consulta: 11 de mayo de 2018.

Esta arquitectura posee como características principales:

- Capacidad de funcionamiento *multicore* (de 1 a 4 núcleos por grupo) con soporte para 1 hasta 2 grupos.
- Soporte de subsets Thumb y Thumb2 (con ThumbEE) en su ISA.
- Soporte para SIMD por extensión NEON avanzada opcional.
- Extensión DSP.
- Punto flotante opcional por VFPv4 conforme estándar IEEE754.

- Gestión de memoria por *Memory Management Unit* (MMU) de Armv7.
- Extensión de seguridad TrustZone.
- Arquitectura de depuración Armv7 por interfaz APB CoreSight con hasta 6 *breakpoints* y 4 *watchpoints.*
- Soporte de virtualización por hardware.
- Soporte para virtualización por hardware y LPAE (Large Physical Address Extensions), que permiten acceso a 1TB de memoria con direcciones físicas de hasta 40 bits.
- Instrucciones MAC fusionadas.
- Instrucciones depurador en modo de hipervisor.
- Caché L2 de tamaño configurable entre 256KB, 512KB, 1MB, 2MB, 4MB u 8MB.
- Interfaz AMBA 4 AXI.
- *Pipeline* superescalar de 11 etapas o más de ejecución fuera de orden con predicción dinámica de saltos y arquitectura de memoria Harvard.

Las aplicaciones de Cortex-A17 incluyen *smartphones*, *infotainment* en industria automotriz, televisiones inteligentes, tablets, dispositivos de consumo casero y decodificadores de televisión digital.

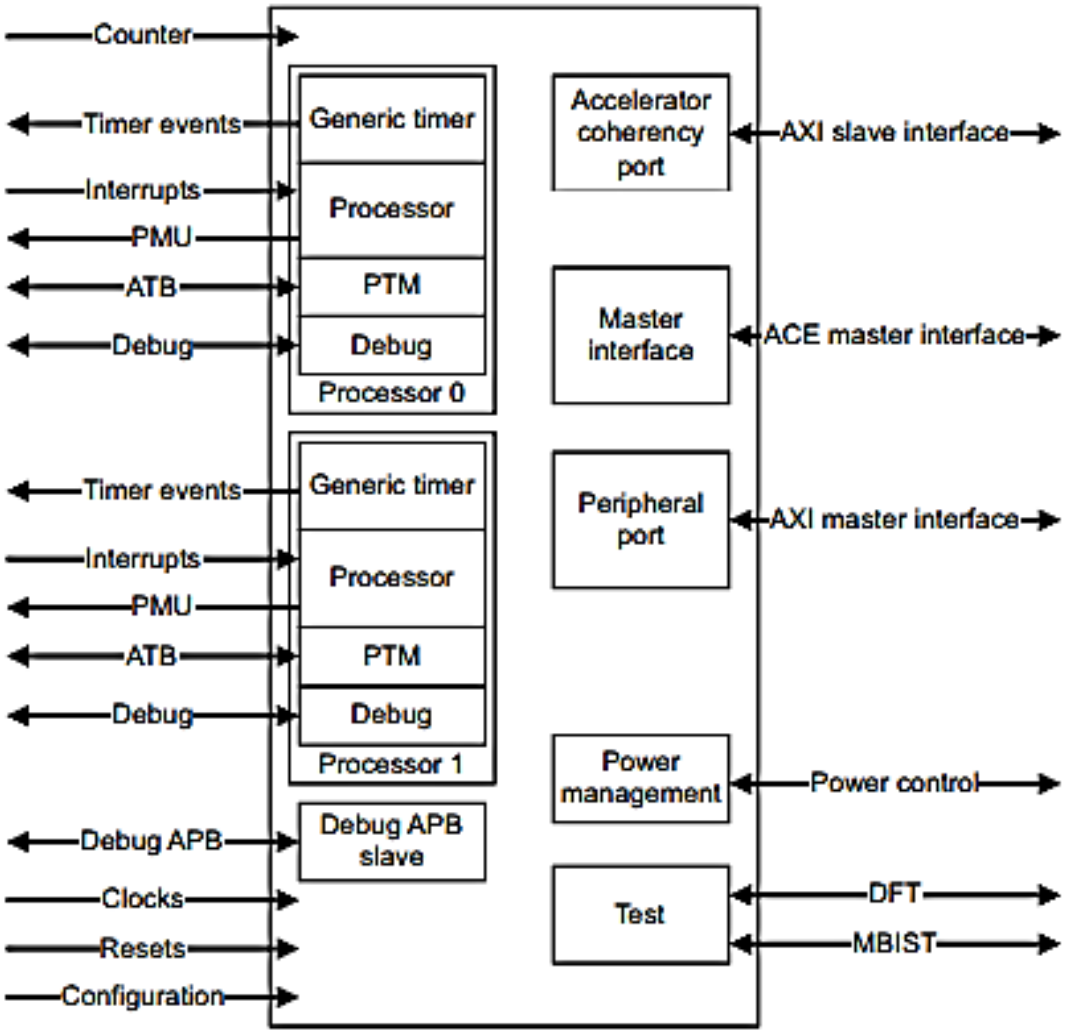
Cortex-A17 ofrece 60 % de desempeño sobre Cortex-A9 en implicaciones NEON y FPU, permitiendo que el rendimiento en general aumente para aplicaciones de procesamiento alto (por ejemplo en audio y video). Esto lo hace la opción más rápida para exigencias de gama media con frecuencias sobre 2,5 GHz en tecnología de 28 nm.

El procesador Cortex-A17 está diseñado para implementarse en un SoC, contando con la capacidad de incorporar propiedad intelectual Arm más allá de

la del núcleo (de gráficos, de sistema, física). Las IP relacionadas con este procesador incluyen:

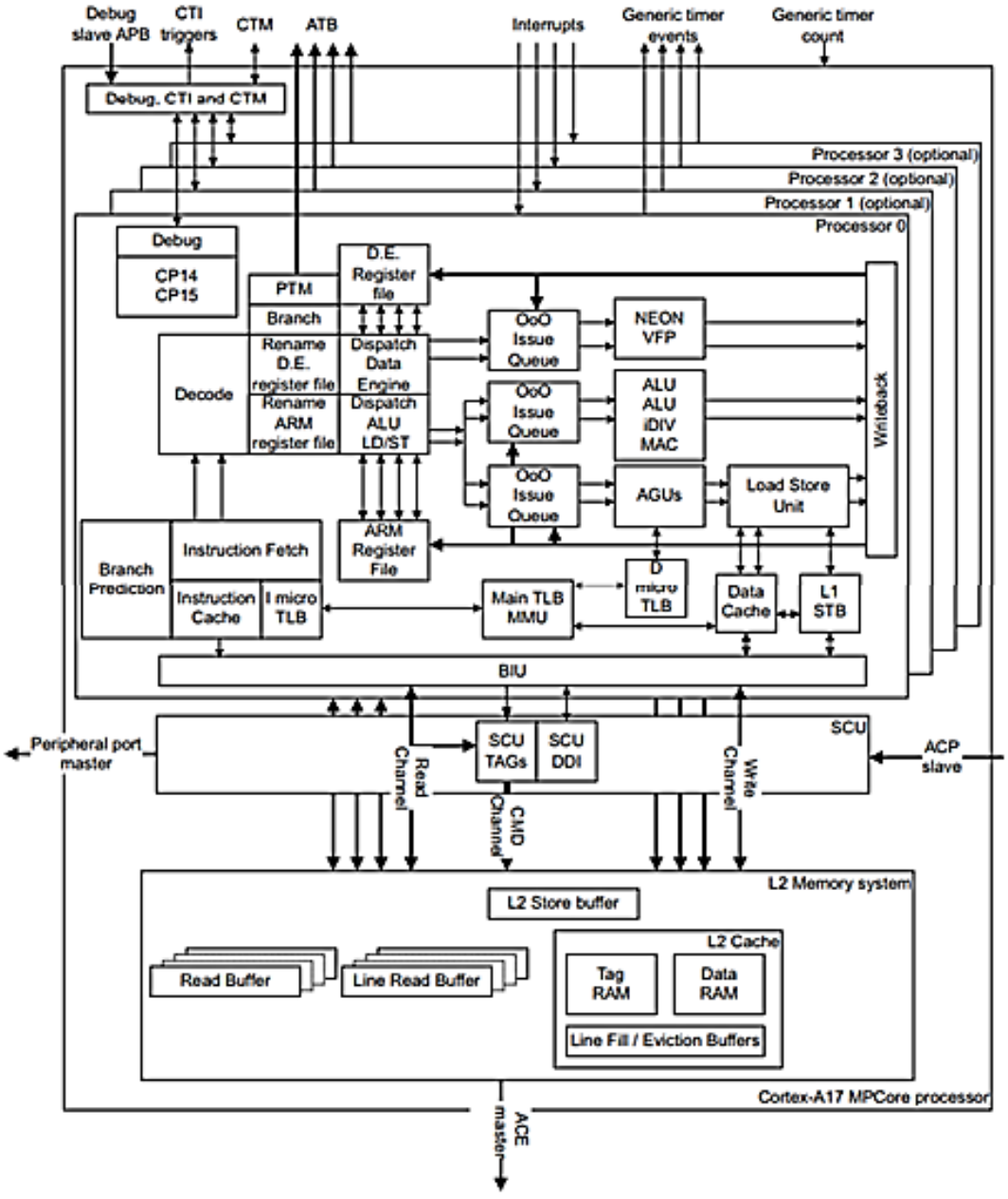
- GPU Mali-450
- Procesadores de monitoreo Mali
- Procesador de video Mali-V500
- Interconexión CoreLink
- Controladores de sistema CoreLink
- Controladores de interrupciones
- CoreSight SoC-400
- POP IP
- CoreLink DMC-500/520

Figura 93. **Diagrama de bloques arquitectura Cortex-A17**



Fuente: Arm Ltd. *Cortex-A17 MPCore Revision:r1p1 Technical Reference Manual*. p.13.

Figura 94. **Implementación Cortex-A17**



Fuente: Arm Ltd. *Cortex-A17 MPCore Revision:r1p1 Technical Reference Manual*. p. 23.

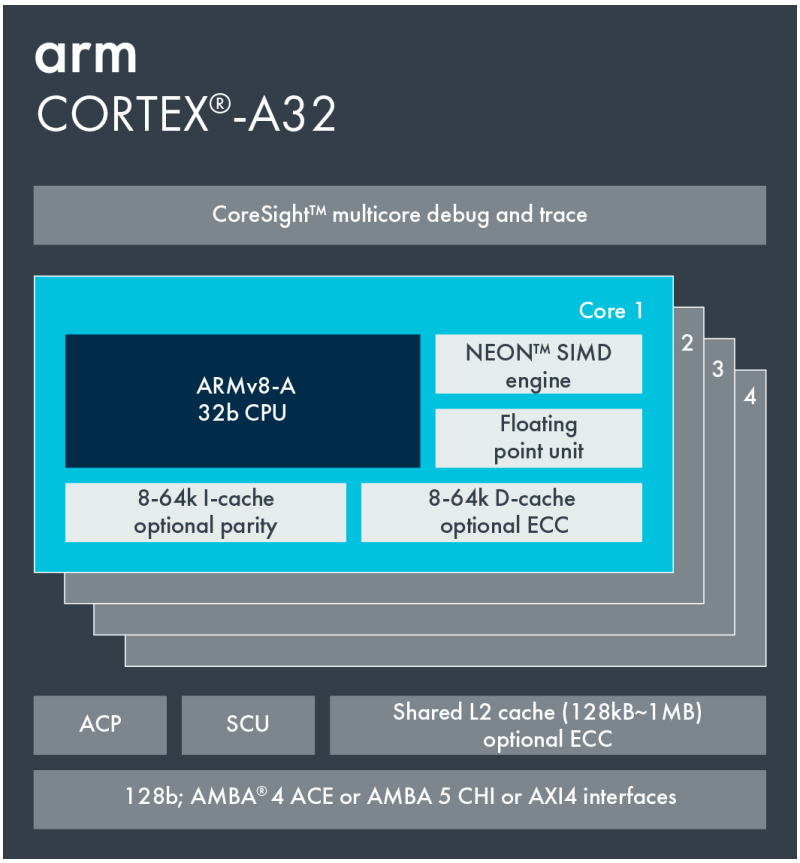
Los componentes de Cortex-A17 incluyen:

- MMU
- Unidad de procesamiento de datos
- Depurador
- Interfaz de rastreo de programa PTM
- Unidad de monitoreo de desempeño (PMU)
- Sistema de memoria lateral de instrucciones
- Sistema de memoria lateral de datos
- Controlador genérico de interrupciones
- Temporizador genérico

4.4.8. Cortex-A32

El procesador Cortex-A32 es el más pequeño de Arm y de menor consumo en 32 bits. Con un *pipeline* optimizado, ofrece desempeño a nivel de Armv8 con sus características y mejoras en *footprint* en comparación a Armv7.

Figura 95. **Componentes de arquitectura ARM Cortex-A32**



Fuente: Cortex. *Procesador Cortex-A32*. https://developer.arm.com/products/processors/cortex-a/cortex-a32. Consulta: 19 de mayo de 2018.

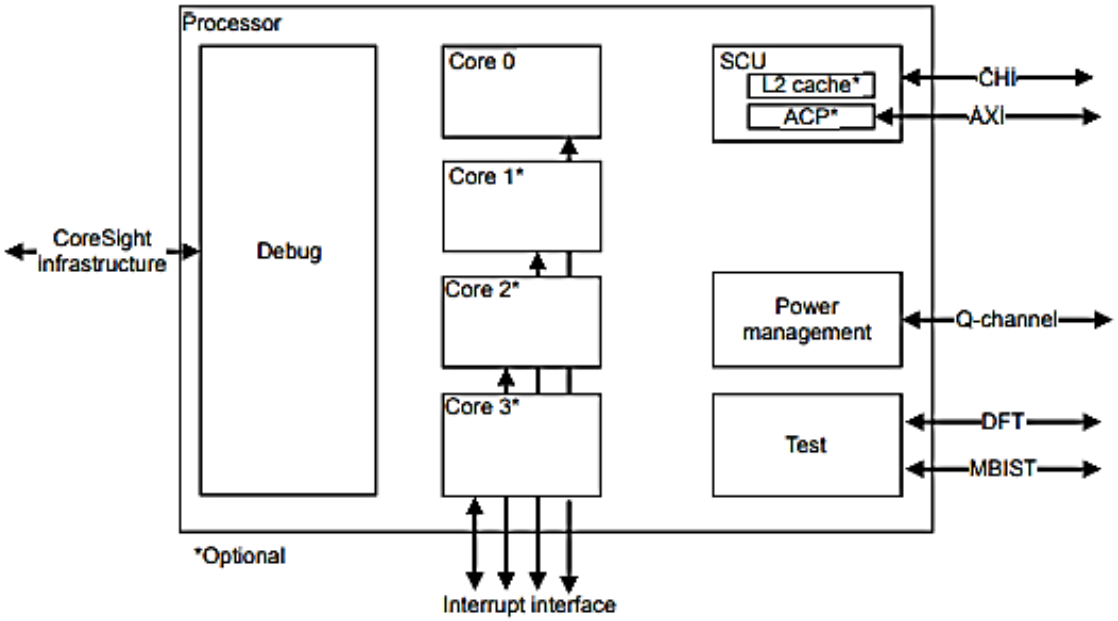
Esta arquitectura posee como características principales:

- Capacidad de funcionamiento *multicore* (de 1 a 4 núcleos por grupo) con soporte para 1 hasta 2 grupos.
- Implementación de sets de instrucciones ARMv8-A A32 y T32 para compatibilidad completa con Armv7-A.
- Soporte para SIMD por extensión NEON avanzada opcional.
- Extensión DSP.

- Punto flotante opcional por VFPv4 conforme estándar IEEE754.
- Extensión de seguridad TrustZone.
- Arquitectura de depuración Armv8 CoreSight DK-A32 con hasta 6 *breakpoints* y 4 *watchpoints.*
- Soporte de virtualización por hardware.
- *Pipeline* de 8 etapas o más de ejecución ordenada con predicción dinámica de saltos y arquitectura de memoria Harvard.
- Sistemas de memorias L1 laterales de instrucciones y datos con MMU.
- Caché L1 de instrucciones de tamaño configurable entre 8KB, 16KB, 32KB, 64KB.
- Caché L1 de datos de tamaño configurable entre 8KB, 16KB, 32KB, 64KB.
- Caché L2 opcional de tamaño configurable entre 128KB, 256KB, 512KB, 1024KB.
- Extensión opcional de criptografía disponible con el motor de datos.
- Unidad de rastreo ETM opcional.
- Interfaz AMBA 4 AXI.

La arquitectura Cortex-A32 se presenta como un *MPCore* (múltiples procesadores), permitiendo la existencia desde 1 a 4 núcleos ordenados por SMP (*Symmetric Multiprocessing*), en un solo grupo de procesadores o varios grupos con coherencia a través de tecnología AMBA 4. La configuración con cuatro procesadores puede observarse en la figura 96.

Figura 96. **Implementación *quad-core* Cortex-A32**

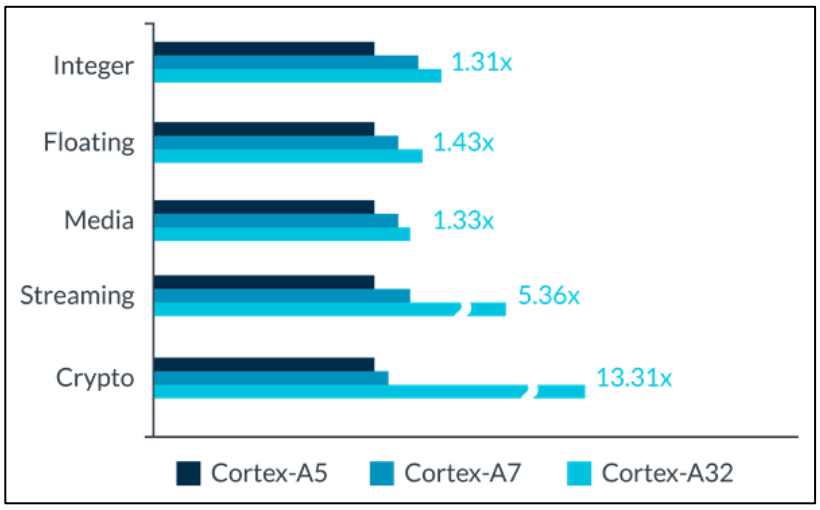


Fuente: Arm Ltd. *ARM Cortex-A32 Processor Revision:r0p1 Technical Reference Manual*. p. 26.

Las aplicaciones de Cortex-A32 incluyen procesadores embebidos en aplicaciones industriales, SBCs, domótica, nodos IoT de baja potencia con soporte para sistema operativo y portátiles.

Esta arquitectura es un diseño mejorado de Cortex-A7 para aumentar la eficiencia en consumo de potencia en estados de espera del CPU. La figura 97 muestra una comparación de desempeño entre Cortex-A32, A5 y A7 para las mismas frecuencias de reloj y configuraciones de procesador.

Figura 97. **Comparación de desempeño Cortex-A32, A5 y A7**

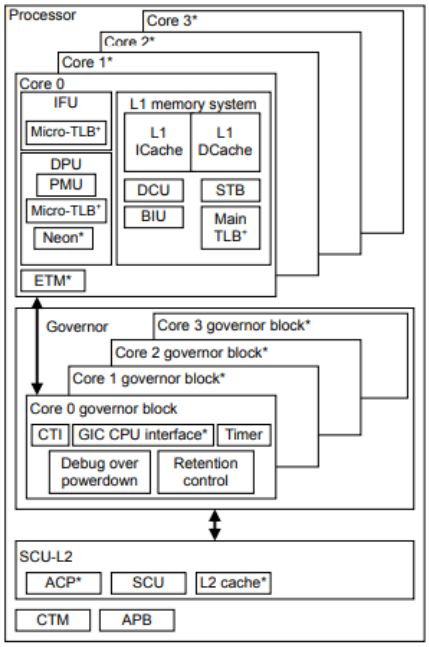


Fuente: Cortex. *Procesador Cortes-A32*. https://developer.arm.com/products/processors/cortex-a/cortex-a32. Consulta: 19 de mayo de 2018.

El procesador Cortex-A32 está diseñado para implementarse en un SoC, contando con la capacidad de incorporar propiedad intelectual Arm más allá de la del núcleo (de gráficos, de sistema, física). Las IP relacionadas con este procesador incluyen:

- GPU Mali-450
- Procesadores de monitoreo Mali-DP550
- Procesador de video Mali-V500
- Interconexión CoreLink
- Controladores de interrupciones
- Controladores de memoria
- Familia de interconexión de coherencia en caché CoreLink
- CoreSight SoC-400
- POP IP

Figura 98. **Implementación Cortex-A32**



Fuente: Arm Ltd. *ARM Cortex-A32 Processor Revision:r0p1 Technical Reference Manual*. p. 36.

Los componentes de Cortex-A32 incluyen:

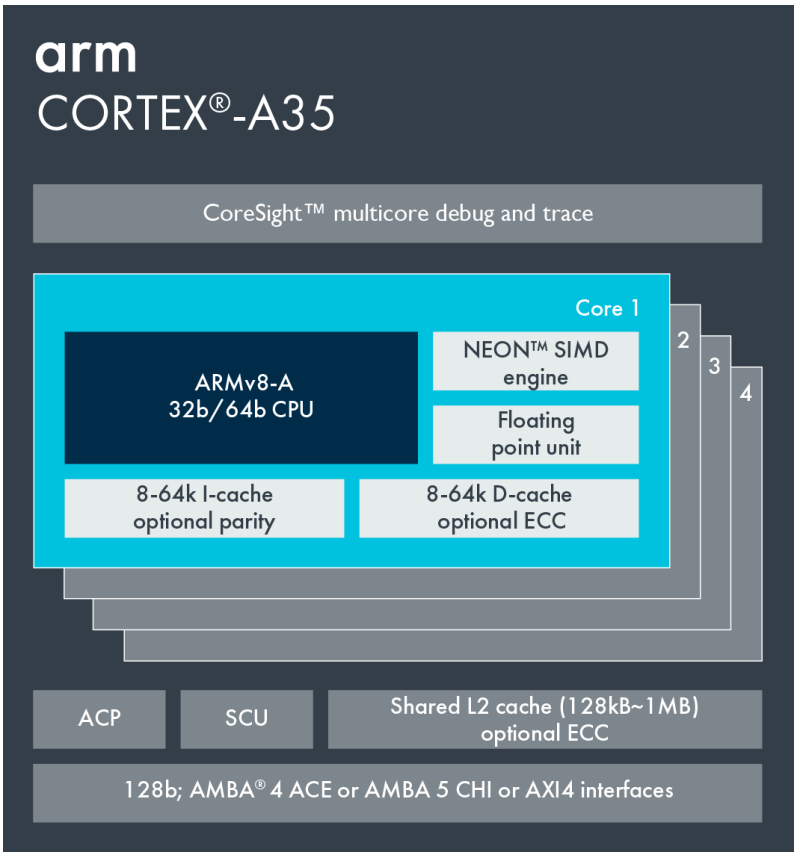
- Bloque de sistema de memoria SCU-L2
- Componentes CoreSight con protección de caché, ROM de depuración y extensión de criptografía.
- Unidad de obtención de instrucciones (IFU).
- Unidad de procesamiento de datos (DPU).
- MMU.
- Sistema de memoria L2.
- Sistema de memoria lateral de datos.

- Bloque gobernador, que se encuentra fuera del núcleo e incluye las funciones que deben continuar operando cuando el procesador está en modo de retención.

4.4.9. Cortex-A35

El procesador Cortex-A35 es el diseño de Arm más eficiente en consumo de potencia que, a la vez, soporta codificación de 32 y 64 bits.

Figura 99. **Componentes de arquitectura ARM Cortex-A35**



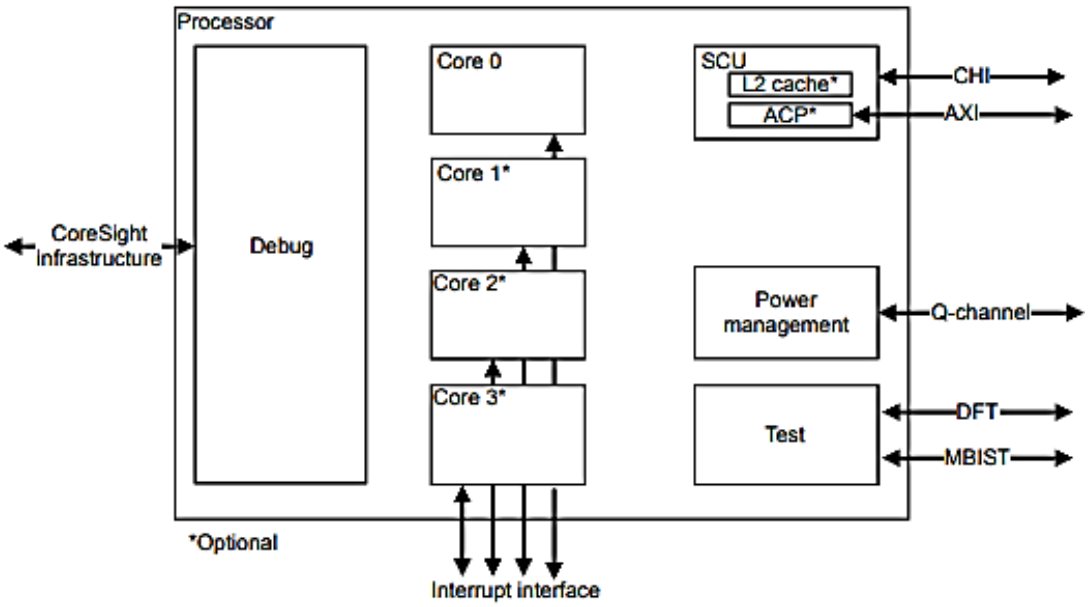
Fuente: Cortex. *Procesador Cortex-A35*. https://developer.arm.com/products/processors/cortex-a/cortex-a35. Consulta: 19 de mayo de 2018.

Esta arquitectura posee como características principales:

- Capacidad de funcionamiento *multicore* (de 1 a 4 núcleos por grupo), con soporte para 1 hasta 2 grupos.
- Implementación de sets de instrucciones ARMv8-A A32 y T32 para compatibilidad completa con Armv7-A.
- Implementación AArch64 para soporte de 64 bits y nuevas características.
- Extensión de seguridad TrustZone.
- Soporte para SIMD por extensión NEON avanzada opcional.
- Extensión DSP.
- Punto flotante opcional por VFPv4 conforme estándar IEEE754.
- Soporte de virtualización por hardware.
- Arquitectura de depuración Armv8 CoreSight DK-A53 con hasta 6 *breakpoints* y 4 *watchpoints.*
- *Pipeline* de 8 etapas o más de ejecución ordenada con predicción dinámica de saltos y arquitectura de memoria Harvard.
- Sistemas de memorias L1 laterales de instrucciones y datos con MMU.
- Caché L1 de instrucciones de tamaño configurable entre 8KB, 16KB, 32KB, 64KB.
- Caché L1 de datos de tamaño configurable entre 8KB, 16KB, 32KB, 64KB.
- Caché L2 opcional de tamaño configurable entre 128KB, 256KB, 512KB, 1024KB.
- Extensión opcional de criptografía disponible con el motor de datos.
- Unidad de rastreo ETM opcional.
- Interfaz AMBA 4 AXI.

La arquitectura Cortex-A35 se presenta como un *MPCore* (múltiples procesadores), permitiendo la existencia desde 1 a 4 núcleos ordenados por SMP (*Symmetric Multiprocessing*), en un solo grupo de procesadores o varios grupos con coherencia a través de tecnología AMBA 4. La configuración con cuatro procesadores puede observarse en la figura 100.

Figura 100. **Implementación *quad-core* Cortex-A35**



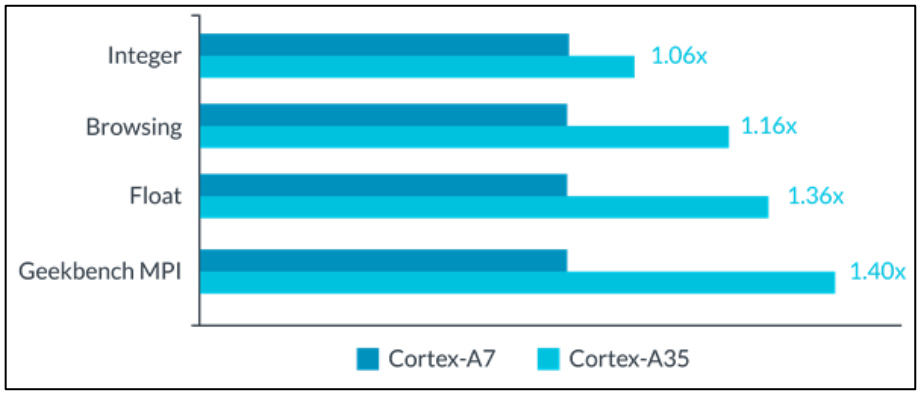
Fuente: Arm Ltd. *ARM Cortex-A35 Processor Revision:r0p2 Technical Reference Manual*. p. 28.

Las aplicaciones de esta arquitectura incluyen el mercado de móviles, embebidos, nodos IoT, domótica, portátiles y aplicaciones industriales.

Cortex-A35 implica características de gestión de potencia que se suman a las de Cortex-A7, mejorando la eficiencia de la batería en dispositivos móviles. Su compatibilidad se extiende a otros procesadores Armv8-A con los que se puede como procesador LITTLE en un sistema big.LITTLE.

Este diseño entrega mejor desempeño que las arquitecturas anteriores a mejores niveles de eficiencia en consumo de potencia. La figura 101 representa la comparación de este procesador con Cortex-A7 en las mismas configuraciones y frecuencia de reloj.

Figura 101. **Comparación de desempeño Cortex-A35 y A7**



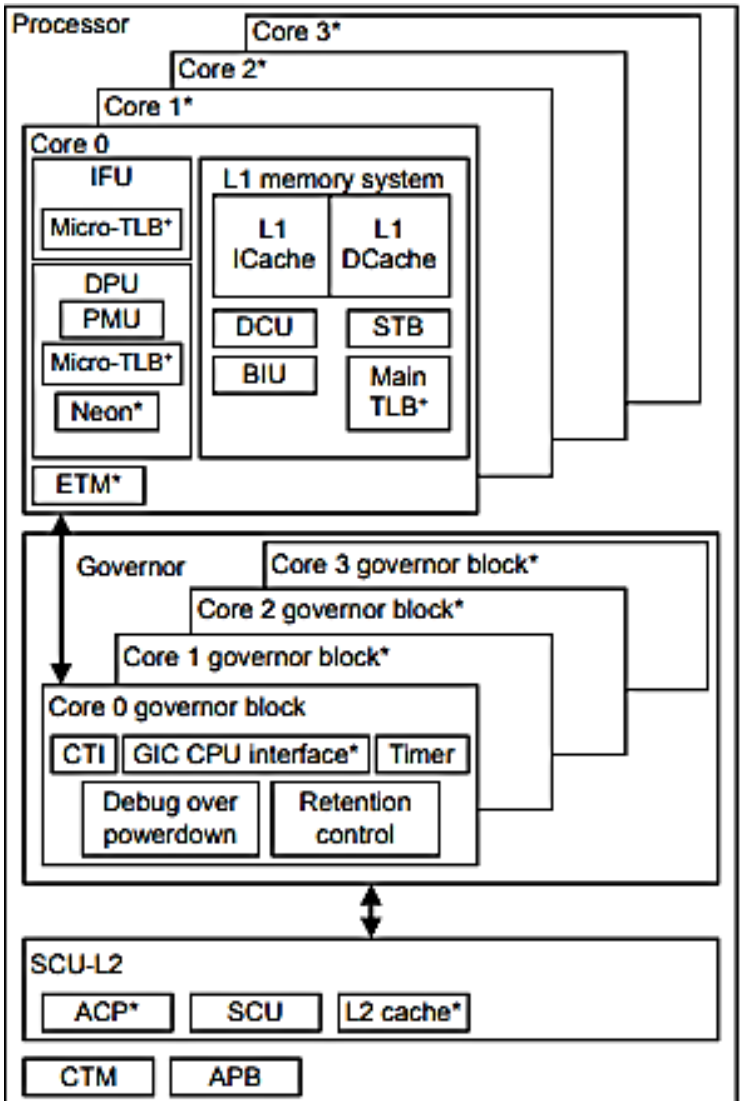
Fuente: Cortex. *Procesador cortex-A35*. https://developer.arm.com/products/processors/cortex-a/cortex-a35. Consulta: 19 de mayo de 2018.

El procesador Cortex-A35 está diseñado para implementarse en un SoC, contando con la capacidad de incorporar propiedad intelectual Arm más allá de la del núcleo (de gráficos, de sistema, física). Las IP relacionadas con este procesador incluyen:

- GPUs Mali-T820 y Mali-T830
- Procesador de monitoreo Mali-DP550
- Procesador de vídeo Mali-V500
- Interconexión CoreLink
- Controladores de interrupciones
- Familia de interconexión de coherencia en caché CoreLink

- TrustZone CryptoCell
- CoreSight SoC-400
- POP IP

Figura 102. **Implementación Cortex-A35**



Fuente: Arm Ltd. *ARM Cortex-A35 Processor Revision:r0p2 Technical Reference Manual*. p. 38.

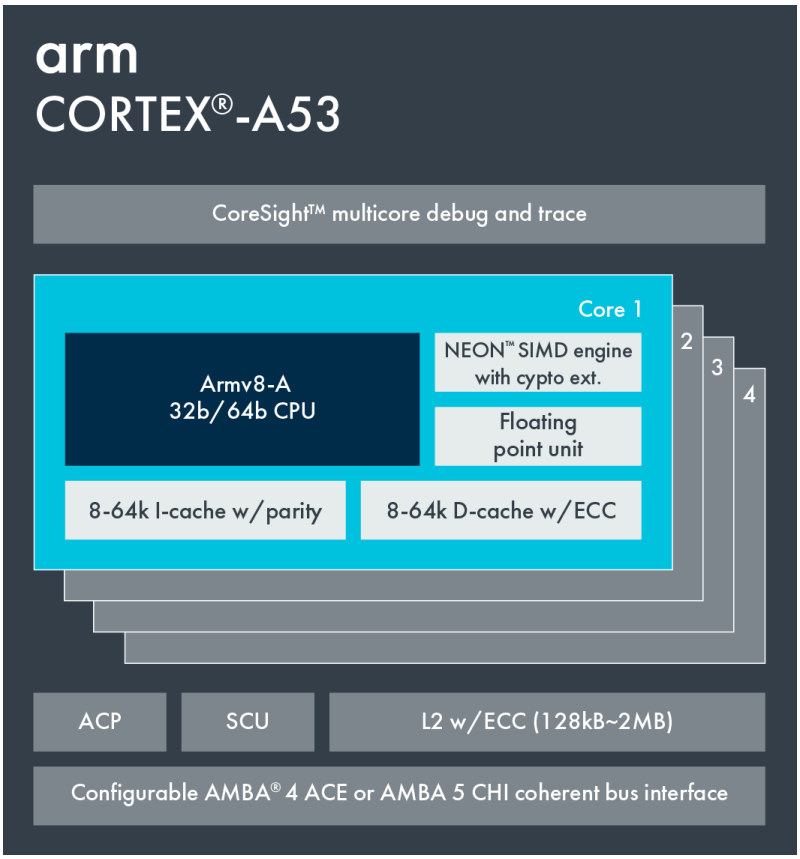
Los componentes de Cortex-A35 incluyen:

- Bloque de sistema de memoria SCU-L2
- Componentes CoreSight con protección de caché, ROM de depuración y extensión de criptografía.
- Unidad de obtención de instrucciones (IFU).
- Unidad de procesamiento de datos (DPU).
- MMU.
- Sistema de memoria L2.
- Sistema de memoria lateral de datos.
- Bloque gobernador, que se encuentra fuera del núcleo e incluye las funciones que deben continuar operando cuando el procesador está en modo de retención.

4.4.10. Cortex-A53

El procesador Cortex-A53 es un diseño de gama media y alta eficiencia introducido en aplicaciones móviles de redes, almacenamiento y otras que se benefician de su versatilidad al poderse emparejar con cualquier núcleo Armv8.0 para formar una configuración big.LITTLE (Cortex-A57, Cortex-A72, otros Cortex-A53 y Cortex-A35). Esta arquitectura es una de las más utilizadas en el catálogo de Arm.

Figura 103. **Componentes de arquitectura ARM Cortex-A53**



Fuente: Cortex. *Procesador Cortex-A53*. https://developer.arm.com/products/processors/cortex-a/cortex-a53. Consulta: 23 de mayo de 2018.

Esta arquitectura posee como características principales:

- Capacidad de funcionamiento *multicore* (de 1 a 4 núcleos por grupo) con soporte para 1 hasta 2 grupos.
- Implementación de sets de instrucciones ARMv8-A A32 y T32 para compatibilidad completa con Armv7-A.
- Implementación AArch64 para soporte de 64 bits y nuevas características.

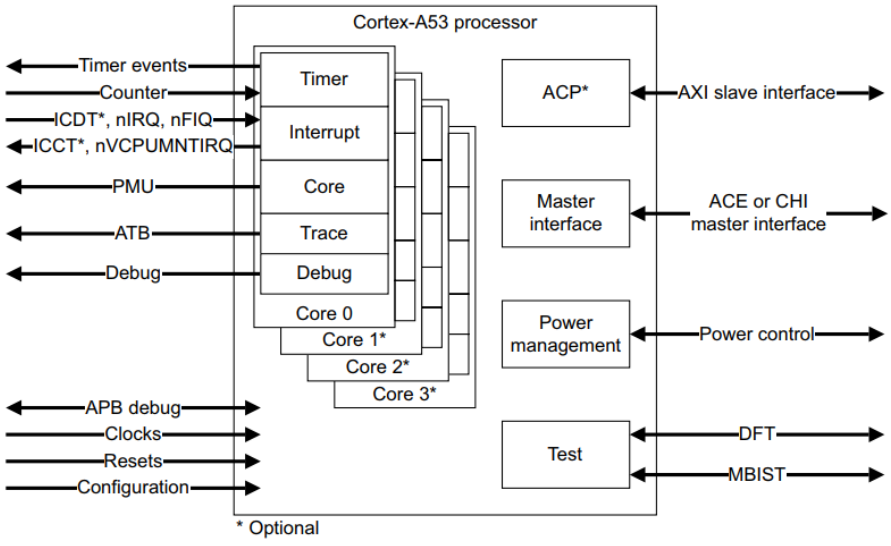
- Extensión de seguridad TrustZone.
- Soporte para SIMD por extensión NEON avanzada opcional.
- Extensión DSP.
- Punto flotante opcional por VFPv4 conforme estándar IEEE754.
- Soporte de virtualización por hardware.
- Arquitectura de depuración Armv8 CoreSight DK-A53 con hasta 6 *breakpoints* y 4 *watchpoints.*
- Capacidad *dual-issue.*
- *Pipeline* de 8 etapas o más de ejecución ordenada con predicción dinámica de saltos avanzada y arquitectura de memoria Harvard.
- Sistemas de memorias L1 laterales de instrucciones y datos con MMU.
- Caché L1 de instrucciones de tamaño configurable entre 8KB, 16KB, 32KB, 64KB.
- Caché L1 de datos de tamaño configurable entre 8KB, 16KB, 32KB, 64KB.
- Caché L2 opcional de tamaño configurable entre 128KB, 256KB, 512KB, 1024KB o 2048KB.
- Extensión opcional de criptografía disponible con el motor de datos.
- Unidad de rastreo ETM opcional.
- Interfaz AMBA 4 AXI.

La capacidad de esta arquitectura de implementar los estados de ejecución AArch32 y AArch64 permite obtener desempeño muy alto y compatibilidad con diseños Armv7. Junto a las características para aumentar la eficiencia en consumo de potencia, Cortex-A53 se presta para solucionar requerimientos en aplicaciones como procesamiento en *smartphones* de gama muy alta y media, aplicaciones aeroespaciales, redes, almacenamiento (HDD, SSD) e *infotainment* en industria automotriz.

El procesador Cortex-A53 está diseñado para implementarse en un SoC, contando con la capacidad de incorporar propiedad intelectual Arm más allá de la del núcleo (de gráficos, de sistema, física). Las IP relacionadas con este procesador incluyen:

- GPUs Mali-T860 y Mali-T880
- Procesador de monitoreo Mali-DP550
- Procesador de video Mali-V500
- Interconexión CoreLink
- Controladores de interrupciones
- Controladores de sistema CoreLink
- Controladores de memoria

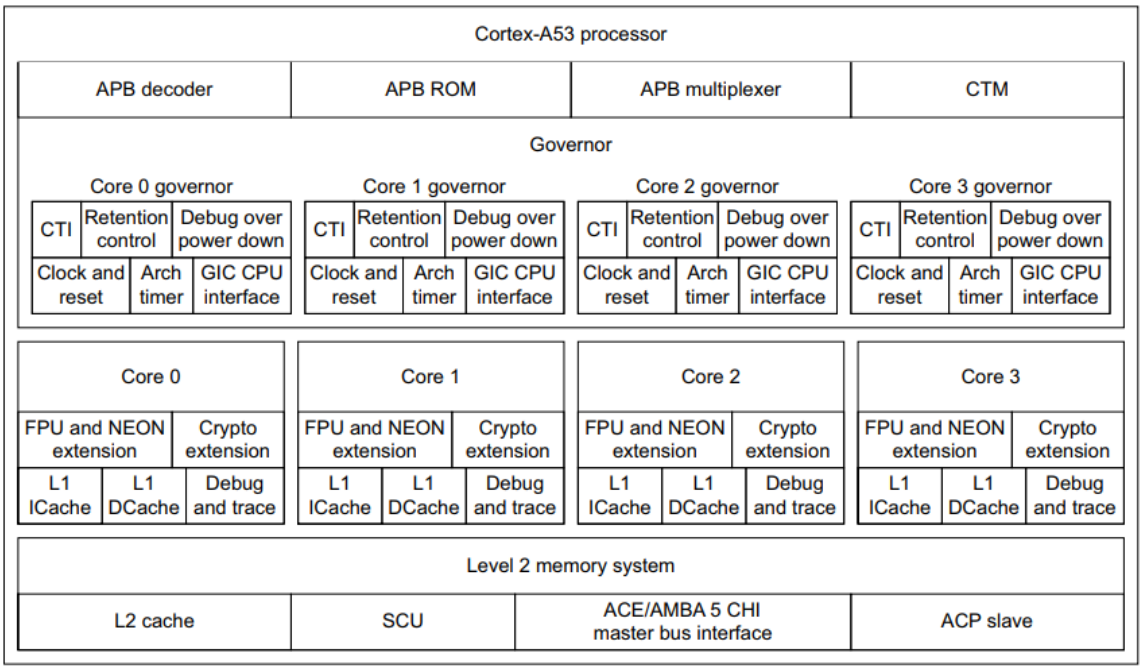
Figura 104. **Diagrama de bloques Cortex-A53**



Fuente: Arm Ltd. *Cortex-A53 MPCore Revision:r0p4 Technical Reference Manual*. p. 13.

La arquitectura Cortex-A53 se presenta como un *MPCore* (múltiples procesadores), permitiendo la existencia desde 1 a 4 núcleos ordenados por SMP (*Symmetric Multiprocessing*), en un solo grupo de procesadores o varios grupos con coherencia a través de tecnología AMBA 4. La configuración con cuatro procesadores puede observarse en la figura 105.

Figura 105. **Implementación *quad-core* Cortex-A53**



Fuente: Arm Ltd. *Cortex-A53 MPCore Revision:r0p4 Technical Reference Manual*. p. 25.

Los componentes de Cortex-A53 incluyen:

- Unidad de obtención de instrucciones (IFU)
- Unidad de procesamiento de datos (DPU)
- MMU

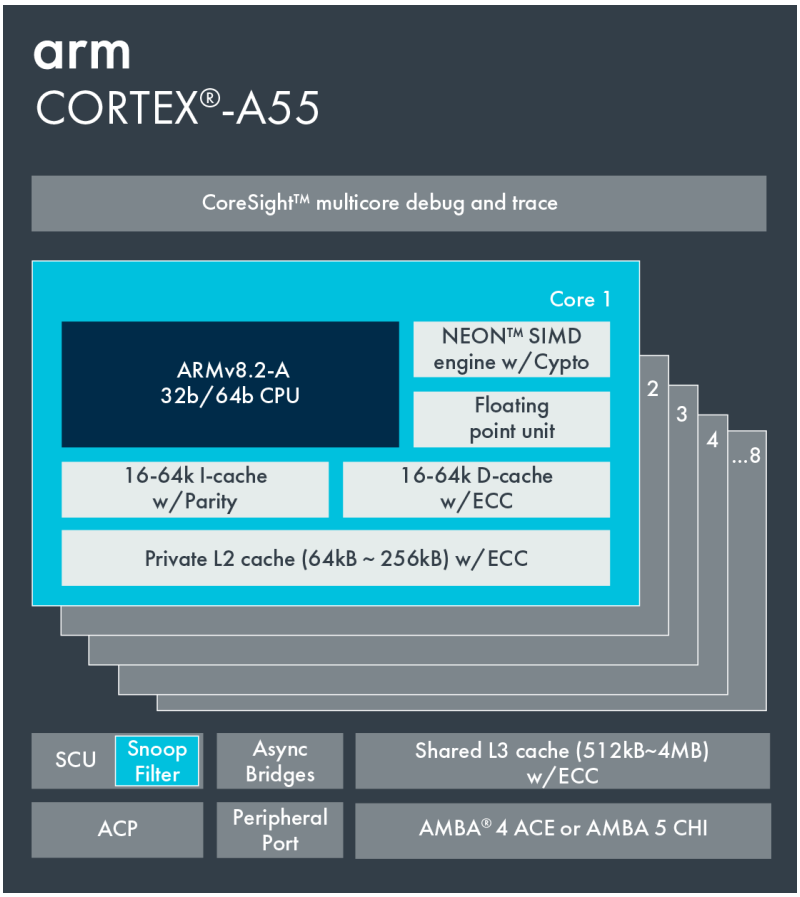
- Sistema de memoria L2.
- Sistema de memoria lateral de datos.
- Componentes CoreSight con protección de caché, ROM de depuración y extensión de criptografía.
- Bloque gobernador, que se encuentra fuera del núcleo e incluye las funciones que deben continuar operando cuando el procesador está en modo de retención.

4.4.11. Cortex-A55

Cortex-A55 es la arquitectura Arm de gama media más eficiente construida en la primera generación de tecnología DynamIQ. Junto a las extensiones de Armv8-A dedicadas a *machine learning*, su escalabilidad y mejoras sobre Cortex-A53, este procesador se utiliza en aplicaciones centrales y de *edge computing.*

Puede ser utilizado como núcleo individual o como sección LITTLE junto a Cortex-A75 para formar un big.LITTLE.

Figura 106. **Componentes de arquitectura ARM Cortex-A55**



Fuente: Cortex. *Procesador Cortex-A55*. https://developer.arm.com/products/processors/cortex-a/cortex-a55. Consulta: 23 de mayo de 2018.

Comparado a Cortex-A53, el A55 entrega mayor desempeño incluyendo mejoras porcentuales de:

- 18 % o más para operaciones en enteros
- 20 % o más para operaciones en punto flotante
- 40 % o más en SIMD NEON
- 15 % o más en ejecución JavaScript

- 200 % en exigencia de memoria

Estas características permiten adaptarse a interacciones de usuario exigentes en dispositivos de pantalla táctil, sistemas de infraestructura de anchos de banda altos y sistemas autónomos automotrices de alta velocidad de reacción. El diseño está pensado para adaptarse a todas las aplicaciones posibles abarcando principalmente compuertas IoT, dispositivos de consumo, realidad virtual y portátiles.

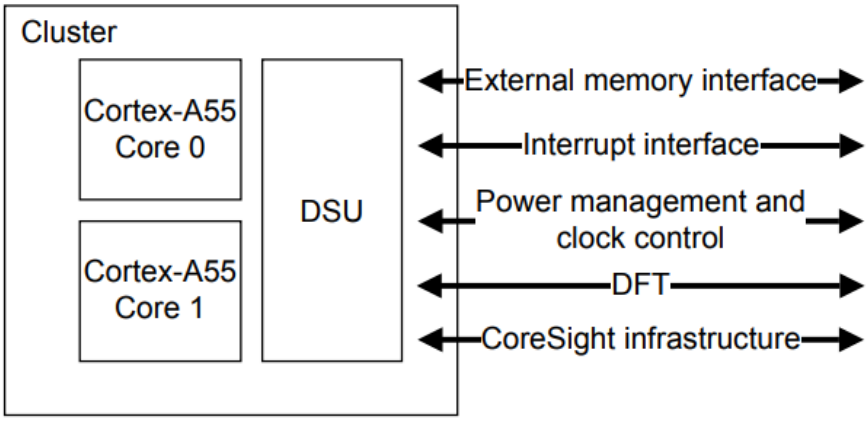
Esta arquitectura posee como características principales:

- Capacidad de funcionamiento *multicore* (de 1 a 8 núcleos por grupo).
- Implementación de sets de instrucciones ARMv8-A A32 y T32 para compatibilidad completa con Armv7-A.
- Implementación AArch64 para soporte de 64 bits y nuevas características.
- Extensión de seguridad TrustZone.
- Soporte para SIMD por extensión NEON avanzada opcional.
- Extensión DSP.
- Sistemas de memorias L1 laterales de instrucciones y datos con MMU y ECC.
- Caché L1 de instrucciones de tamaño configurable entre 16KB, 32KB, 64KB.
- Caché L1 de datos de tamaño configurable entre 16KB, 32KB, 64KB.
- Caché L2 opcional (con latencia reducida en 50 % en accesos) de tamaño configurable entre 64KB, 128KB, 256KB.
- Caché L3 opcional de tamaño configurable entre 512KB, 1MB, 2MB, 3MB, 4MB.
- Extensión opcional de criptografía disponible con el motor de datos.

- Arquitectura de depuración Armv8 CoreSight v3 con hasta 6 *breakpoints* y 4 *watchpoints.*
- Direccionamiento físico de 40 bits.
- Extensiones Armv8.1, Armv8.2, Armv8.3 y Armv8.4.
- Unidad de rastreo ETM opcional.
- Unidad compartida DynamIQ para conectar los núcleos a un sistema de memoria externo.
- Punto flotante opcional por VFPv4 conforme estándar IEEE754.
- *Pipeline* de 8 etapas o más de ejecución ordenada con predicción dinámica de saltos avanzada y arquitectura de memoria Harvard.
- Interfaz AMBA 4 AXI.

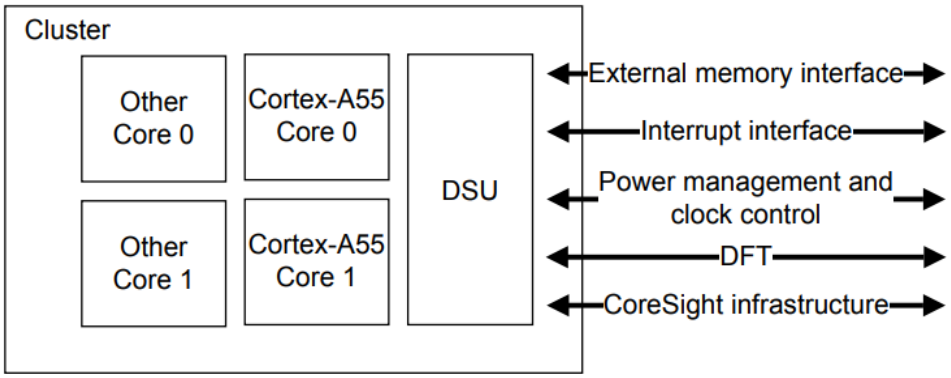
Actualmente la arquitectura tiene soporte en reversa para el software existente en Arm y representa un enlace con las características más recientes con instrucciones NEON, para *machine learning* y características avanzadas de seguridad.

Figura 107. **Implementación *dual-core* Cortex-A53**



Fuente: Arm Ltd. *ARM Cortex-A55 Processor Revision:r1p0 Technical Reference Manual*. p. 26.

Figura 108. **Implementación Cortex-A53 con otro núcleo integrados en grupo compartido L3**



Fuente: Arm Ltd. *ARM Cortex-A55 Processor Revision:r1p0 Technical Reference Manual*. p. 26.

Los componentes de Cortex-A55 incluyen:

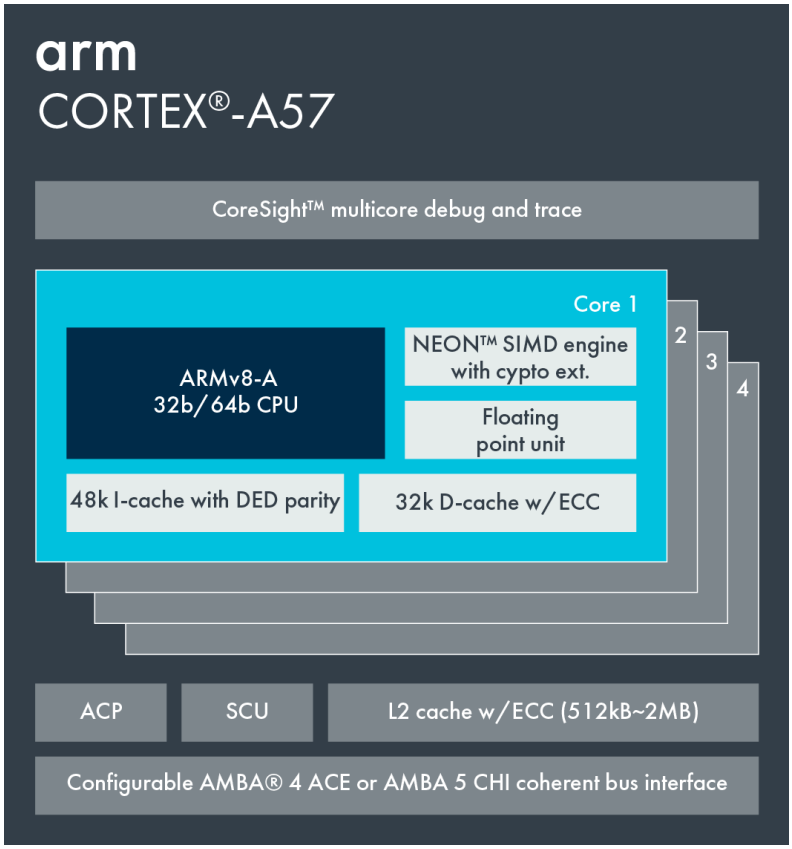
- Unidad de obtención de instrucciones (IFU).
- Unidad de procesamiento de datos (DPU).
- MMU.
- Sistema de memoria L2.
- Sistema de memoria lateral de datos.
- Componentes CoreSight con protección de caché, ROM de depuración y extensión de criptografía.
- Unidad compartida DynamIQ.

4.4.12. Cortex-A57

El procesador Cortex-A57 es una arquitectura de alto rendimiento que puede ser emparejado con Cortex-A53 en configuración big.LITTLE para

aplicaciones móviles. Se comprende como un diseño mejorado de Cortex-A15 con mayor desempeño a niveles mayores de eficiencia en consumo de potencia.

Figura 109. **Componentes de arquitectura ARM Cortex-A57**



Fuente: Cortex. *Procesador Cortex-A57*. https://developer.arm.com/products/processors/cortex-a/cortex-a57. Consulta: 23 de mayo de 2018.

Esta arquitectura posee como características principales:

- Capacidad de funcionamiento *multicore* (de 1 a 4 núcleos por grupo)

- Implementación de sets de instrucciones ARMv8-A A32 y T32 para compatibilidad completa con Armv7-A.
- Implementación AArch64 para soporte de 64 bits y nuevas características.
- Extensión de seguridad TrustZone.
- Soporte para SIMD por extensión NEON avanzada opcional.
- Extensión DSP.
- Punto flotante opcional por VFPv4 conforme estándar IEEE754.
- Soporte de virtualización por hardware.
- Arquitectura de depuración Armv8 CoreSight DK-A57 con hasta 6 *breakpoints* y 4 *watchpoints.*
- Capacidad *triple-issue.*
- Sistemas de memorias L1 laterales de instrucciones y datos con MMU y ECC.
- Caché L1 de instrucciones de tamaño fijo 48KB.
- Caché L1 de datos de tamaño fijo 32KB.
- Caché L2 compartida de tamaño configurable entre 512KB, 1MB, 2MB.
- Interfaz AMBA 4 AXI.
- Unidad de rastreo ETM opcional.
- Extensión de criptografía.
- *Pipeline* de 15 etapas de ejecución fuera de orden con predicción dinámica de saltos avanzada y arquitectura de memoria Harvard.

Las objetivos de este procesador incluyen aplicaciones de altas exigencies en *smartphones* de gama media, redes, almacenamiento, compuertas IoT, ADAS en automotriz, área aeroespacial, industrial y military.

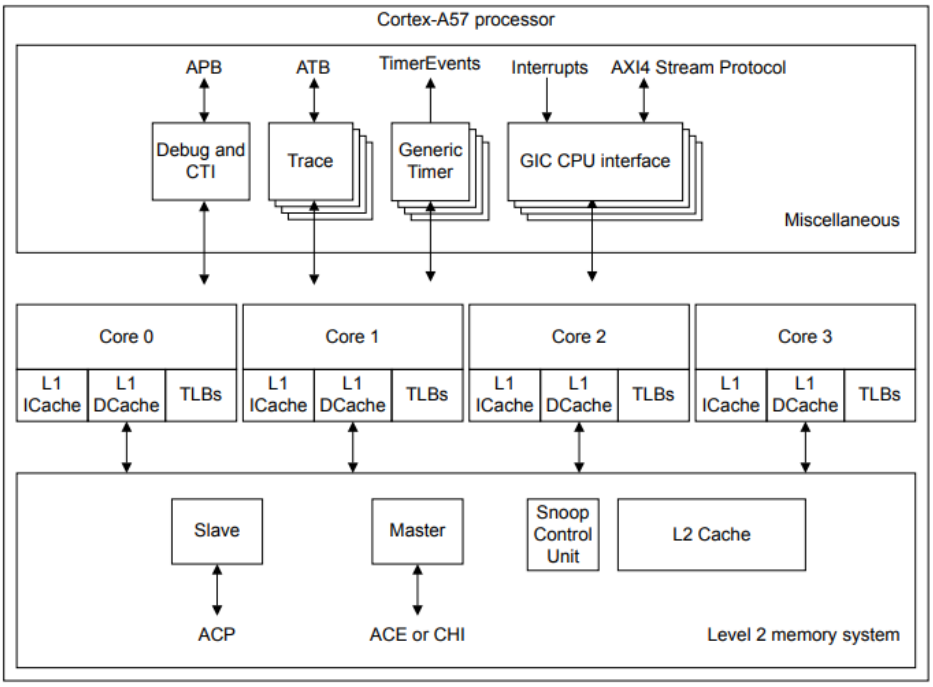
El procesador Cortex-A57 está diseñado para implementarse en un SoC, contando con la capacidad de incorporar propiedad intelectual Arm más allá de

la del núcleo (de gráficos, de sistema, física). Las IP relacionadas con este procesador incluyen:

- GPUs Mali-T860 y Mali-T880
- Procesador de monitoreo Mali-DP550
- Procesador de vídeo Mali-V500
- Controladores de interrupciones.
- Controladores de sistema CoreLink
- Familia de interconexión de coherencia en caché CoreLink
- CoreLink DMC-500
- CoreLink DMC-520
- CoreSight SoC-400
- POP IP

La arquitectura Cortex-A57 se presenta como un *MPCore* (múltiples procesadores), permitiendo la existencia desde 1 a 4 núcleos ordenados por SMP (*Symmetric Multiprocessing*) en un solo grupo de procesadores o varios grupos con coherencia a través de tecnología AMBA 5 CHI o AMBA 4 ACE. La configuración con cuatro procesadores puede observarse en la figura 110.

Figura 110. **Implementación *quad-core* Cortex-A57**



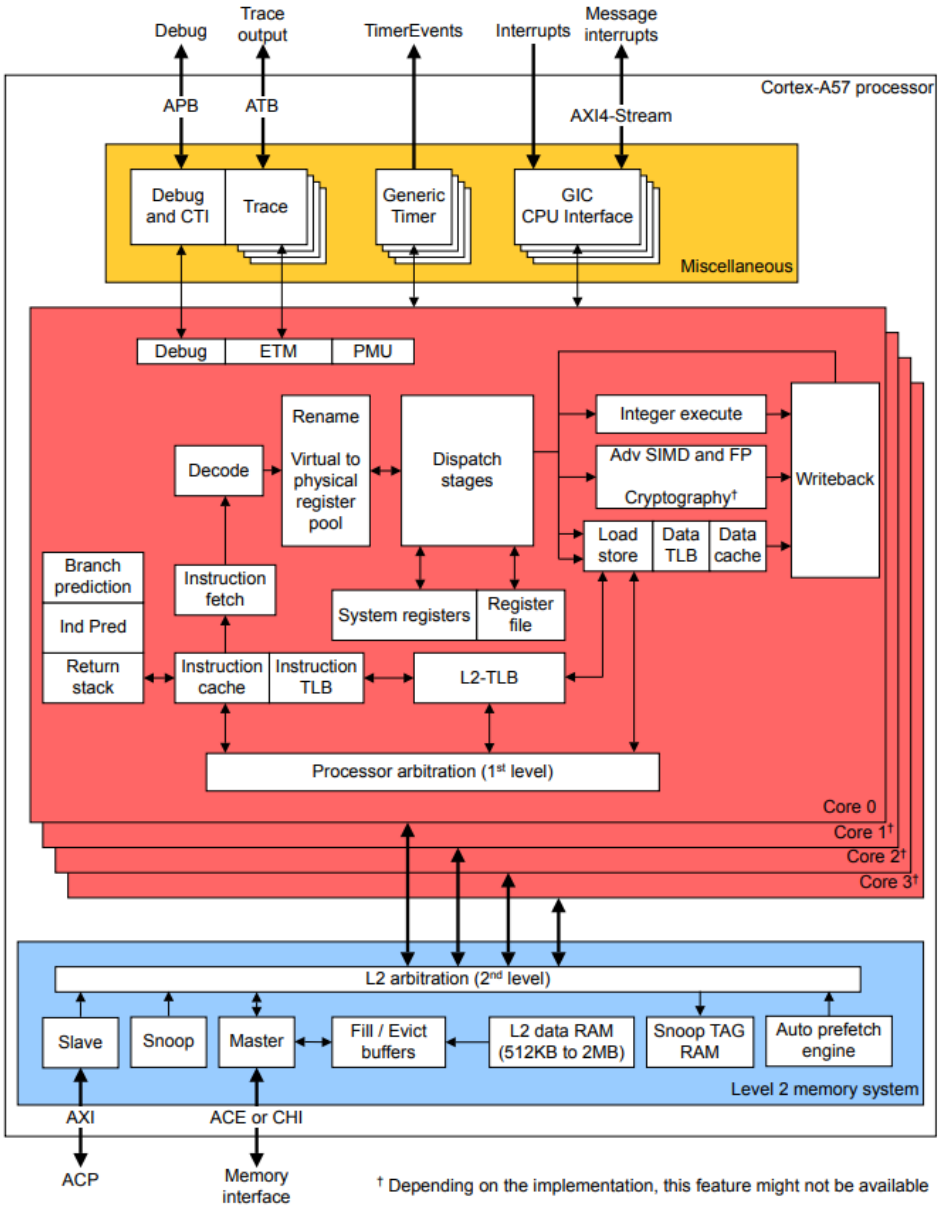
Fuente: Arm Ltd. *Cortex-A57 MPCore Revision:r1p3 Technical Reference Manual*. p. 14.

Los componentes de Cortex-A57 incluyen:

- Bloque de obtención de instrucciones
- Bloque de decodificación de instrucciones
- Bloque de despacho de instrucciones
- Bloque de ejecución entera
- Bloque de carga y almacenamiento
- Sistema de memoria L2
- Unidad NEON y VFP
- Controlador genérico de interrupciones
- Temporizador genérico

- Depuración y rastreo

Figura 111. **Diagrama de bloques Cortex-A57**



Fuente: Arm Ltd. *Cortex-A57 MPCore Revision:r1p3 Technical Reference Manual*. p. 25.

4.4.13. Cortex-A65AE

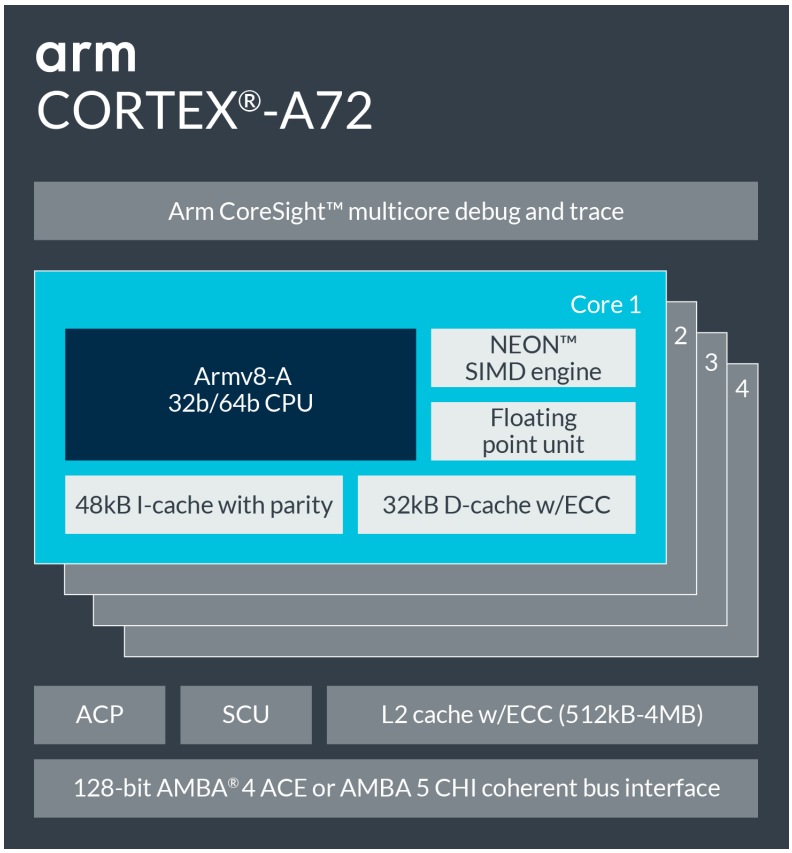
Esta arquitectura se presenta como una extension de Cortex-A53 mejorada para uso en aplicaciones automotrices (de ahí las siglas AE por *Automotive Enhancement*). Está diseñada para dispositivos con una carga alta de procesamiento y tareas críticas de seguridad, y está basada en tecnología DynamIQ con características flexibles multinúcleo.

Como parte del programa Arm Safety Ready (listo para la seguridad), Cortex-A65AE ha sido ampliamente probado y mejorado para responder adecuadamente a niveles rigurosos de seguridad. Cuenta con una característica de tolerancia de fallas llamada DCLS (Dual Core Lock-Step).

4.4.14. Cortex-A72

Cortex-A72 es una arquitectura de alto rendimiento que puede ser puesto en configuración big.LITTLE con Cortex-A53 para aplicaciones móviles de alta densidad de procesamiento con ejecución eficiente de instrucciones a frecuencias sobre los 3GHz.

Figura 112. **Componentes de arquitectura ARM Cortex-A72**



Fuente: Cortex. *Procesador Cortex-A72*. https://developer.arm.com/products/processors/cortex-a/cortex-a72. Consulta: 23 de mayo de 2018.

Esta arquitectura posee como características principales:

- Capacidad de funcionamiento *multicore* (de 1 a 4 núcleos por grupo).
- Implementación de sets de instrucciones ARMv8-A A32 y T32 para compatibilidad completa con Armv7-A.
- Implementación AArch64 para soporte de 64 bits y nuevas características.
- Extensión de seguridad TrustZone.

- Soporte para SIMD por extensión NEON avanzada opcional.
- Extensión DSP.
- Punto flotante opcional por VFPv4 conforme estándar IEEE754.
- Soporte de virtualización por hardware.
- Arquitectura de depuración Armv8 CoreSight DK-A72 con hasta 6 *breakpoints* y 4 *watchpoints.*
- Capacidad *triple-issue.*
- Sistemas de memorias L1 laterales de instrucciones y datos con MMU y ECC.
- Caché L1 de instrucciones de tamaño fijo 48KB.
- Caché L1 de datos de tamaño fijo 32KB.
- Caché L2 compartida de tamaño configurable entre 512KB, 1MB, 2MB o 4MB.
- Interfaz AMBA 4 AXI.
- Unidad de rastreo ETM opcional.
- Extensión de criptografía.
- *Pipeline* 15 etapas de ejecución fuera de orden con predicción dinámica de saltos avanzada y arquitectura de memoria Harvard.

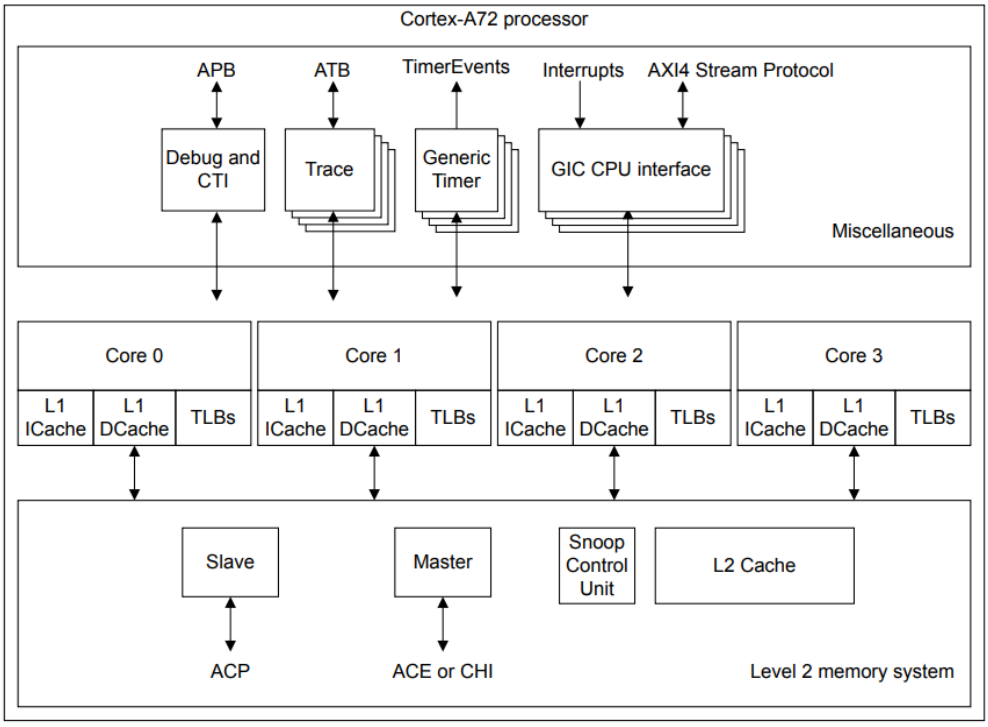
Las aplicaciones de este procesador incluyen los mismos campos que Cortex-A53 con mejoras sobre las características de Cortex-A57. En el área de los smartphones específicamente, esta arquitectura entrega rendimiento más de 3 veces mayor al de Cortex-A15.

El procesador Cortex-A72 está diseñado para implementarse en un SoC, contando con la capacidad de incorporar propiedad intelectual Arm más allá de la del núcleo (de gráficos, de sistema, física). Las IP relacionadas con este procesador incluyen:

- GPUs Mali-T860 y Mali-T880
- Procesador de monitoreo Mali-DP550
- Procesador de vídeo Mali-V500
- Controladores de sistema CoreLink
- Familia de interconexión de coherencia en caché CoreLink
- CoreLink DMC-500
- CoreLink DMC-520
- CoreSight SoC-400
- POP IP

La arquitectura Cortex-A72 se presenta como un *MPCore* (múltiples procesadores), permitiendo la existencia desde 1 a 4 núcleos ordenados por SMP (*Symmetric Multiprocessing*), en un solo grupo de procesadores o varios grupos con coherencia a través de tecnología AMBA 5 CHI o AMBA 4 ACE. La configuración con cuatro procesadores puede observarse en la figura 113.

Figura 113. **Implementación *quad-core* Cortex-A72**



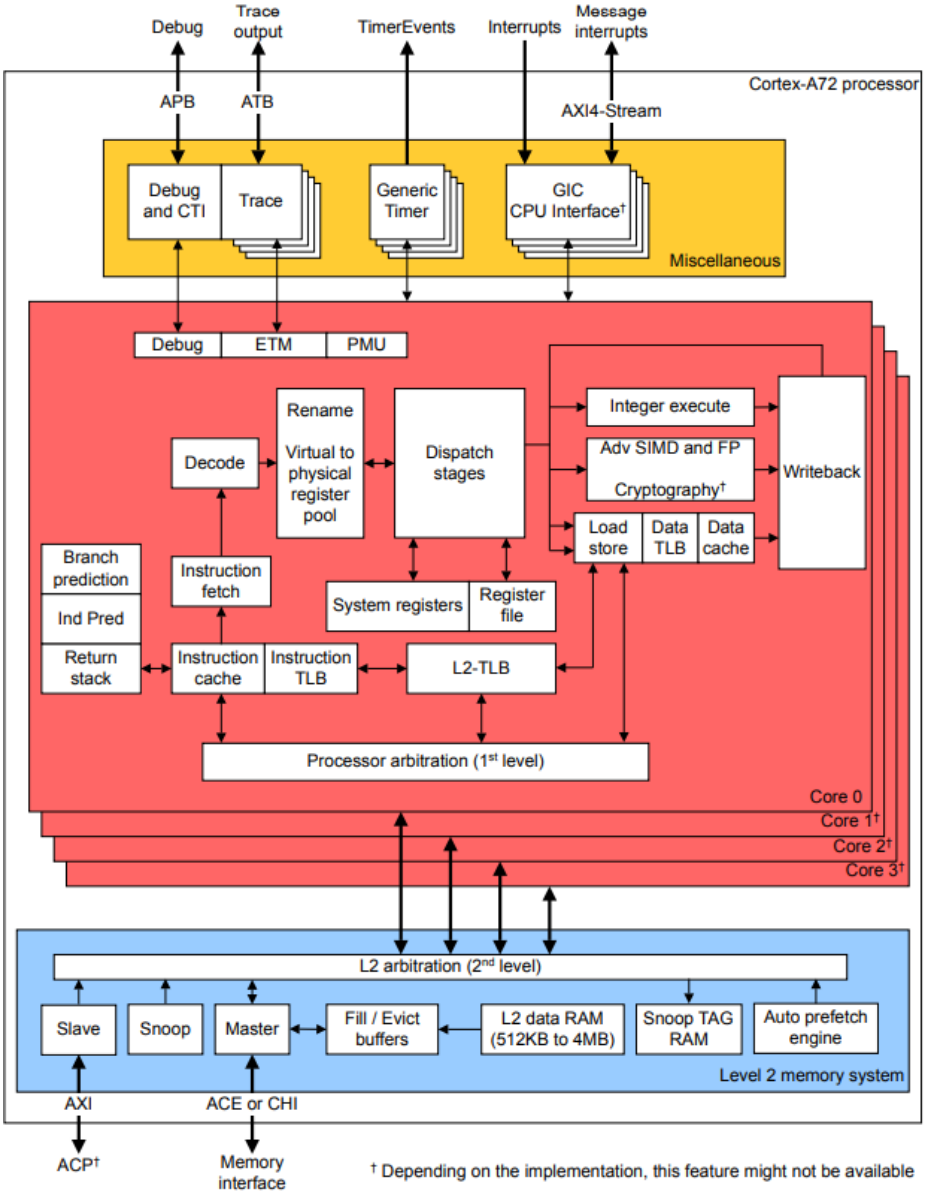
Fuente: Arm Ltd. *Cortex-A72 MPCore Revision:r0p3 Technical Reference Manual*. p. 14.

Los componentes de Cortex-A72 incluyen:

- Bloque de obtención de instrucciones
- Bloque de decodificación de instrucciones
- Bloque de despacho de instrucciones
- Bloque de ejecución entera
- Bloque de carga y almacenamiento
- Sistema de memoria L2
- Unidad NEON y VFP
- Controlador genérico de interrupciones

- Temporizador genérico
- Depuración y rastreo

Figura 114. **Diagrama de bloques Cortex-A72**

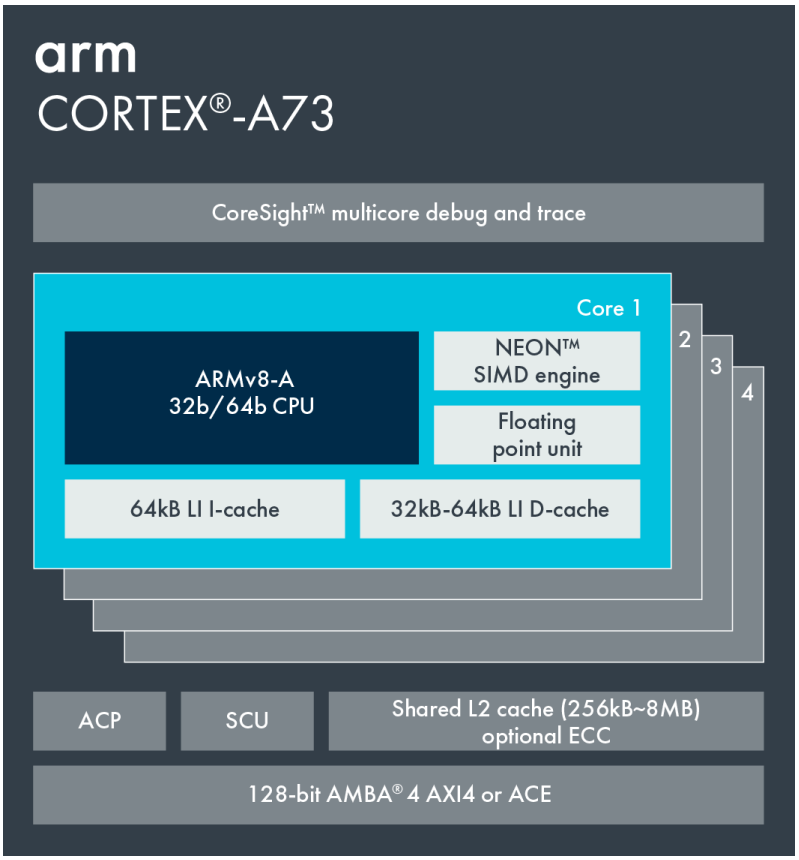


Fuente: Arm Ltd. *Cortex-A72 MPCore Revision:r0p3 Technical Reference Manual*. p. 25.

4.4.15. Cortex-A73

El procesador premium más pequeño en Armv8-A con áreas menores 0,65 mm^2 por núcleo. Con 30 % de incremento en rendimiento sobre Cortex-A72, puede ser implementado en big.LITTLE usualmente con un procesador Cortex-A53 o Cortex-A35.

Figura 115. **Componentes de arquitectura ARM Cortex-A73**



Fuente: Cortex. *Procesador Cortex-A73*. https://developer.arm.com/products/processors/cortex-a/cortex-a73. Consulta: 23 de mayo de 2018.

La arquitectura Cortex-A73 se presenta como un *MPCore* (múltiples procesadores), permitiendo la existencia desde 1 a 4 núcleos ordenados por SMP (*Symmetric Multiprocessing*) en un solo grupo de procesadores o varios grupos con coherencia a través de tecnología AMBA 4 ACE.

Esta arquitectura posee como características principales:

- Capacidad de funcionamiento *multicore* (de 1 a 4 núcleos por grupo).
- Implementación de sets de instrucciones ARMv8-A A32 y T32 para compatibilidad completa con Armv7-A.
- Implementación AArch64 para soporte de 64 bits y nuevas características.
- Extensión de seguridad TrustZone.
- Soporte para SIMD por extensión NEON avanzada opcional.
- Extensión DSP.
- Punto flotante opcional por VFPv4 conforme estándar IEEE754.
- Soporte de virtualización por hardware.
- Arquitectura de depuración Armv8 CoreSight con hasta 6 *breakpoints* y 4 *watchpoints.*
- Sistemas de memorias L1 laterales de instrucciones y datos con MMU y ECC.
- Caché L1 de datos de tamaño configurable de 32KB o 64KB.
- Caché L2 de tamaño configurable entre 256KB, 512KB, 1MB, 2MB, 4MB u 8MB.
- Interfaz AMBA 4 AXI.
- Unidad de rastreo ETM opcional.
- Extensión de criptografía.

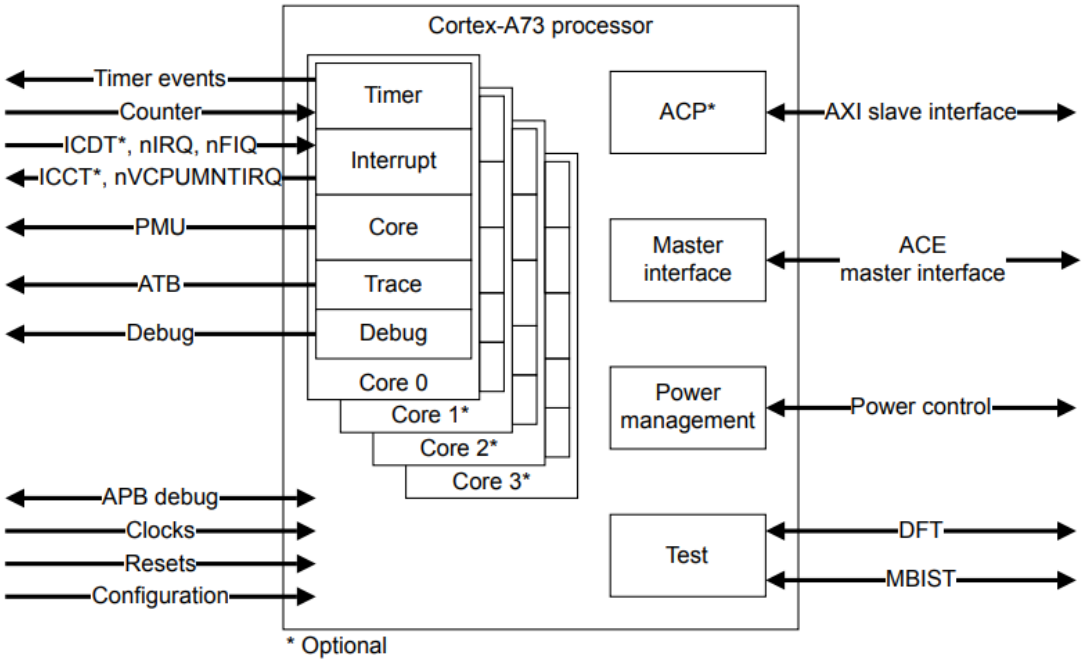
- *Pipeline* 11 o 12 etapas de ejecución fuera de orden con predicción dinámica de saltos avanzada y arquitectura de memoria Harvard.

Esta arquitectura está orientada a los móviles y dispositivos de consumo. Con la capacidad de implementarse a frecuencias por encima de los 2,8 GHz, es capaz de incrementar el rendimiento para soportar el trabajo en dispositivos con pantallas grandes; esto lo ubica en las áreas de *smartphones* de gama media y muy alta, televisión digital, equipos de redes caseros, *infotainment* en industria automotriz y decodificadores de televisión digital.

El procesador Cortex-A73 está diseñado para implementarse en un SoC, contando con la capacidad de incorporar propiedad intelectual Arm más allá de la del núcleo (de gráficos, de sistema, física). Las IP relacionadas con este procesador incluyen:

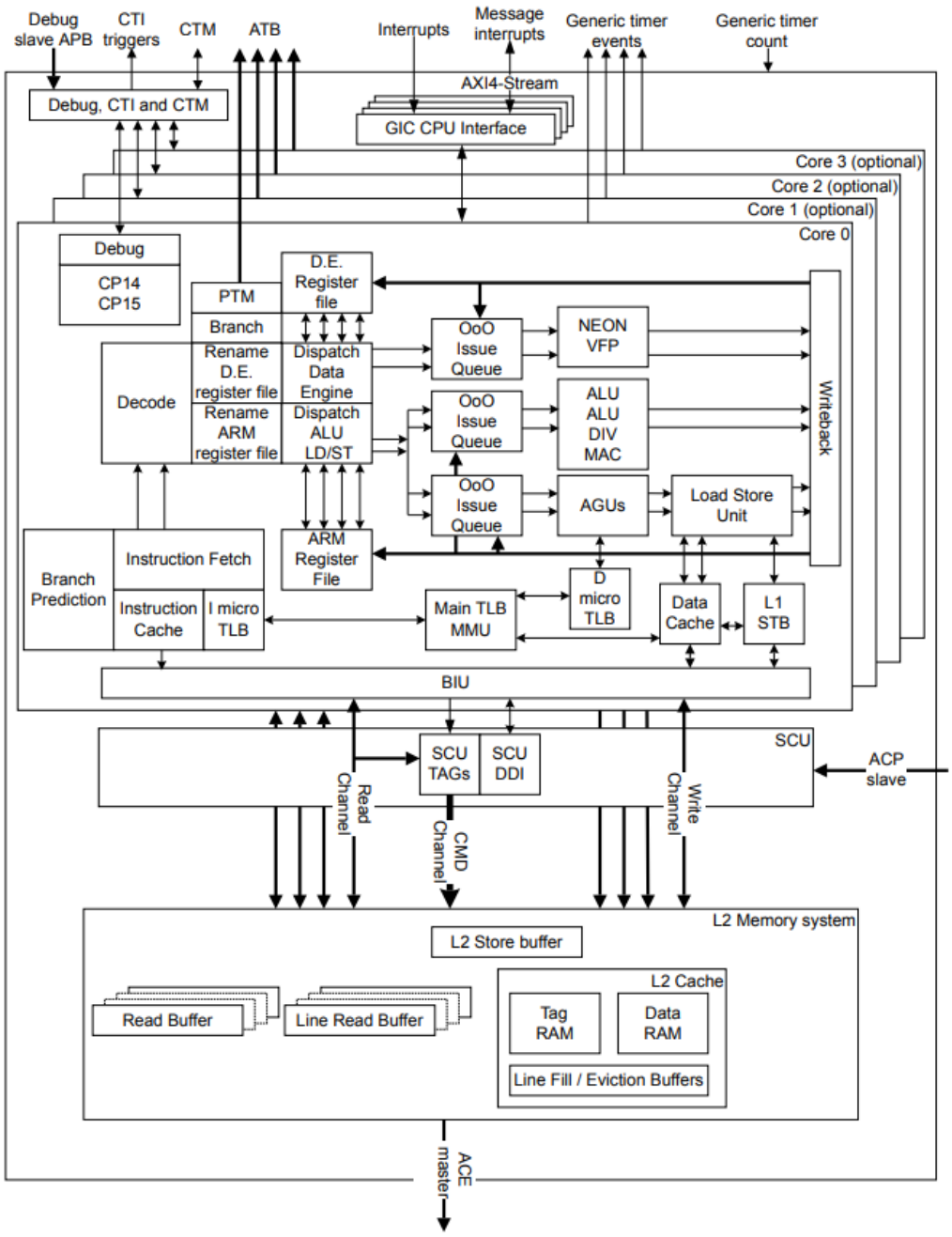
- GPU Mali-G71
- Procesador de monitoreo Mali-DP550
- Procesador de video Mali-V500
- CoreLink CCI-550
- Controladores de sistema CoreLink
- Controladores de interrupciones CoreLink
- TrustZone System IP
- CoreLink DMC-500
- CoreLink DMC-520
- CoreSight SoC-400
- POP IP

Figura 116. **Configuración procesador Cortex-A73**



Fuente: Arm Ltd. *Cortex-A73 MPCore Revision:r0p2 Technical Reference Manual*. p.15.

Figura 117. **Diagrama de bloques Cortex-A73**



Fuente: Arm Ltd. *Cortex-A73 MPCore Revision:r0p2 Technical Reference Manual*. p. 27.

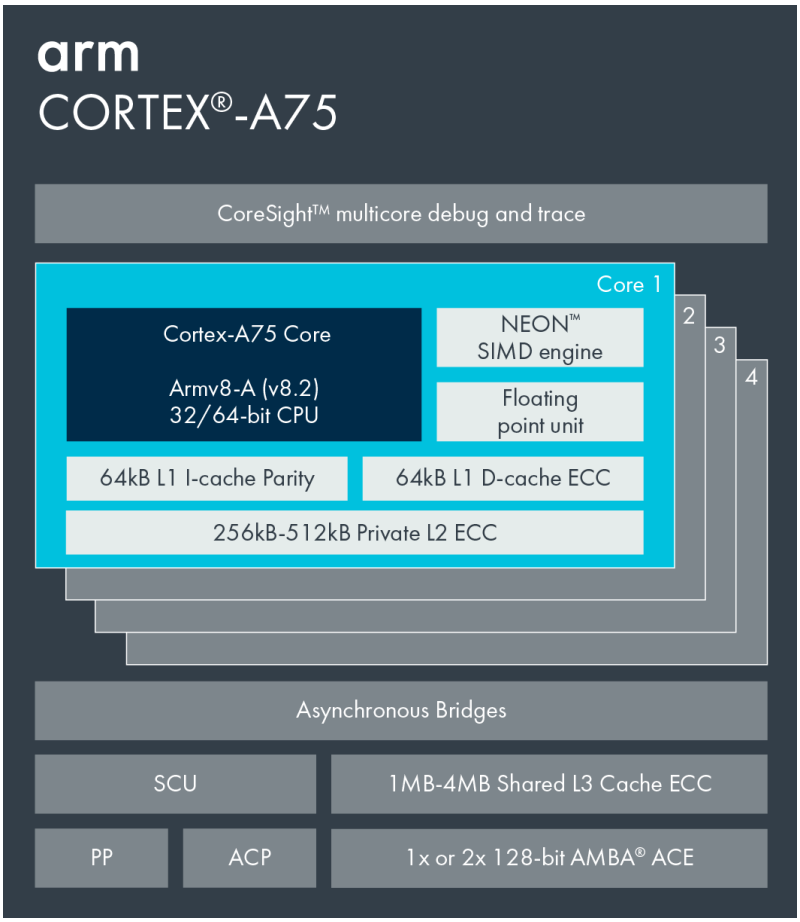
Los componentes de Cortex-A73 incluyen:

- Bloque de obtención de instrucciones
- Bloque de decodificación de instrucciones
- Bloque de despacho de instrucciones
- Bloque de ejecución entera
- Bloque de carga y almacenamiento
- Sistema de memoria L2
- Unidad NEON y VFP
- Controlador genérico de interrupciones
- Temporizador genérico
- Depuración y rastreo

4.4.16. Cortex-A75

Construido en tecnología DynamIQ, el procesador Cortex-A75 es un diseño escalable con alta capacidad de respuesta orientado a todo tipo de aplicaciones como una actualización de las arquitecturas Cortex-A72 y Cortex-A73 con mejoras para trabajo en *machine learning* y otras aplicaciones avanzadas, lo que lo posiciona entre los más potentes a la fecha.

Figura 118. **Componentes de arquitectura ARM Cortex-A75**



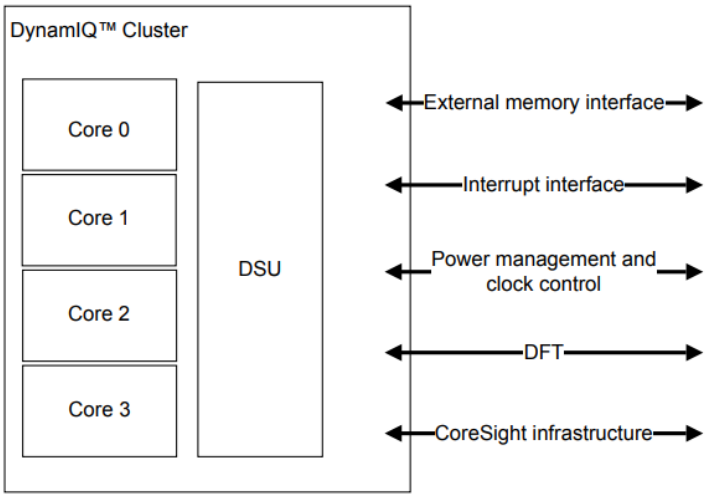
Fuente: Cortex. *Procesador Cortex-A75*. https://developer.arm.com/products/processors/cortex-a/cortex-a75. Consulta: 23 de mayo de 2018.

Cuando actúa como procesador *big* con Cortex-A55 como LITTLE, se puede obtener una configuración big.LITTLE eficiente para dispositivos con limitaciones térmicas, permitiendo suplir las necesidades en campos de *smartphones,* dispositivos inteligentes caseros como DTV, servidores, dispositivos de pantalla ancha (como laptops y Chromebooks) y aplicaciones en industria automotriz (de interacción con el usuario y de seguridad en conjunto con procesadores Cortex-R).

Esta arquitectura posee como características principales:

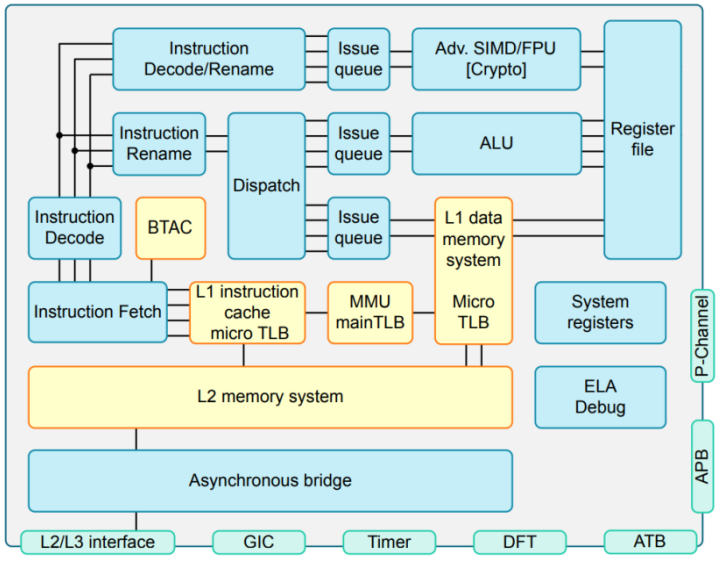
- Capacidad de funcionamiento *multicore* (de 1 a 4 núcleos por grupo).
- Implementación de sets de instrucciones ARMv8-A A32 y T32 para compatibilidad completa con Armv7-A.
- Implementación AArch64 para soporte de 64 bits y nuevas características.
- Extensión de seguridad TrustZone.
- Soporte para DSP y SIMD por extensión NEON avanzada opcional.
- Extensión opcional de criptografía disponible con el motor de datos.
- Direccionamiento físico de 44 bits.
- Sistemas de memorias L1 laterales de instrucciones y datos de 64KB cada una con MMU y ECC.
- Cachés L2 privadas y unificadas para datos e instrucciones de tamaño configurable entre 256KB o 512KB.
- Caché L3 de tamaño configurable entre 512KB, 1MB, 2MB, 3MB o 4MB.
- Arquitectura de depuración Armv8 CoreSight v3 con hasta 6 *breakpoints* y 4 *watchpoints.*
- Extensiones Armv8.1, Armv8.2 (incluyendo RAS), Armv8.3 (LDAPR) y Armv8.4.
- Unidad de rastreo ETM opcional.
- Unidad compartida DynamIQ para conectar los núcleos a un sistema de memoria externo.
- Punto flotante opcional por VFPv4 conforme estándar IEEE754.
- *Pipeline* de 11 a 13 etapas de ejecución fuera de orden con predicción dinámica de saltos avanzada y arquitectura de memoria Harvard.
- Interfaz AMBA 4 AXI.

Figura 119. **Diagrama de bloques Cortex-A75**



Fuente: Arm Ltd. *ARM Cortex-A75 Processor Revision:r3p0 Technical Reference Manual*. p. 28.

Figura 120. **Vista general procesador Cortex-A75**



Fuente: Arm Ltd. *ARM Cortex-A75 Processor Revision:r3p0 Technical Reference Manual*. p. 36.

Los componentes de Cortex-A75 incluyen:

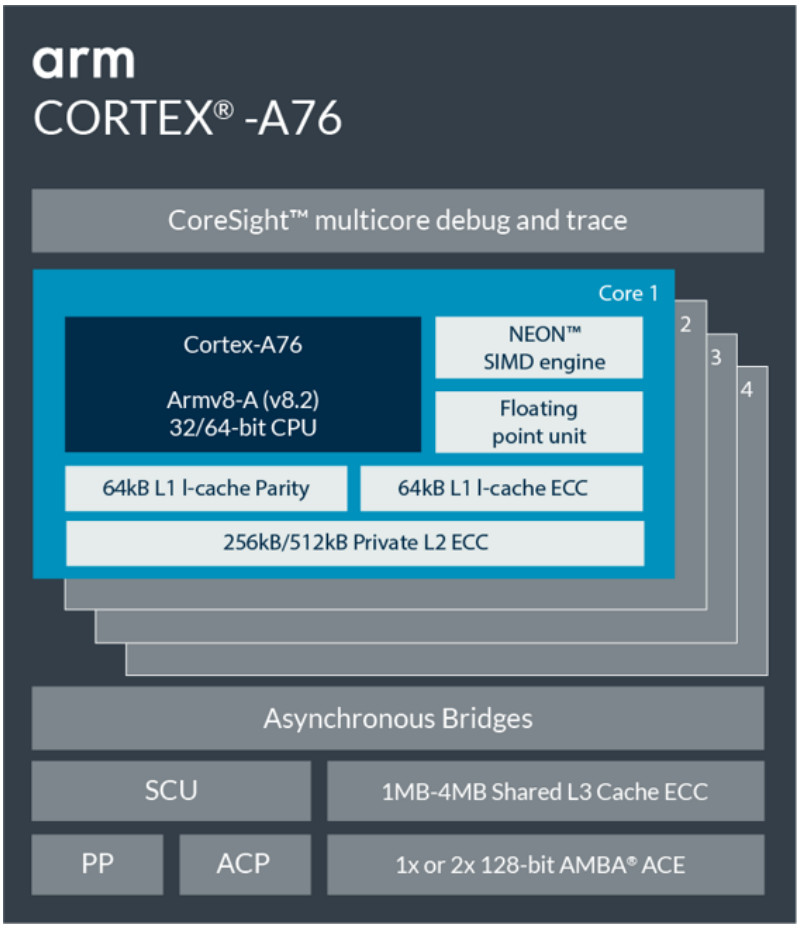
- Bloque de obtención de instrucciones
- Bloque de decodificación de instrucciones
- Bloque de despacho de instrucciones
- Bloque de ejecución
- Sistema de memoria L2
- Sistema de memoria L1
- Renombrado de registros

4.4.17. Cortex-A76

Esta arquitectura es la continuación de la línea de gama alta construida por DynamIQ, funcionando con Cortex-A55 en configuración big.LITTLE.

Está diseñada para servir de núcleo en aplicaciones al nivel de computadoras portátiles y otros dispositivos inteligentes móviles de alta disponibilidad. Su principal mejora es el aumento de rendimiento respecto a Cortex-A75 (en 35 %), aumentando al mismo tiempo el cuidado de la vida de la batería.

Figura 121. **Componentes de arquitectura ARM Cortex-A76**



Fuente: Cortex. *Procesador Cortex-A76*. https://developer.arm.com/products/processors/cortex-a/cortex-a76. Consulta: 15 de junio de 2018.

Esta arquitectura posee como características principales:

- Capacidad de funcionamiento *multicore* (de 1 a 4 núcleos por grupo).
- Implementación de sets de instrucciones ARMv8-A A32 y T32 para compatibilidad completa con Armv7-A.

- Implementación AArch64 para soporte de 64 bits y nuevas características.
- Extensiones Armv8.1, Armv8.2 (incluyendo RAS), Armv8.3 (LDAPR) y Armv8.4.
- Soporte para DSP y SIMD por extensión NEON avanzada opcional.
- Extensión opcional de criptografía disponible con el motor de datos.
- Extensión de seguridad TrustZone.
- Cachés L1 de datos e instrucciones de 64KB cada una.
- Cachés L2 privadas y unificadas para datos e instrucciones de tamaño configurable entre 256KB o 512KB.
- Caché L3 de tamaño configurable entre 512KB, 1MB, 2MB, 3MB o 4MB.
- Arquitectura de depuración Armv8 CoreSight v3.
- Unidad de rastreo ETM opcional.
- Punto flotante opcional conforme estándar IEEE754.
- *Pipeline* de 11 a 13 etapas de ejecución fuera de orden con predicción dinámica de saltos avanzada y arquitectura de memoria Harvard.
- Direccionamiento físico de 40 bits.
- Interfaz AMBA 4 AXI.

Por el momento no existe aún mucha documentación acerca de la arquitectura Cortex-76; sin embargo, las aplicaciones pensadas para este procesador incluyen *smartphones,* IoT para el hogar, dispositivos de pantallas largas (como *laptops* y Chromebooks), computadoras embebidas (como SBC), y aplicaciones en industria automotriz.

Lo curioso en la aparición de este diseño es la entrada de Arm en el campo de las computadoras personales portátiles que en caso de funcionar adecuadamente, marcaría la entrada de la compañía en este mercado y por lo

tanto, la extensión completa de la arquitectura Arm. Las primeras implementaciones están calendarizadas para lanzarse en 2019.

De forma similar al Cortex-A65AE, existe una versión mejorada de esta arquitectua con el fin de servir a propósitos de aplicaciones automotrices. Cortex-A76AE cuenta con características que lo hacen adecuado para tareas de alta exigencia de procesamiento y de seguridad. Está orientado a ser usado en sistemas de conducción autónoma y los más recientes sistemas de asistencia avanzada al conductor (ADAS).

5. LENGUAJE DE ENSAMBLADOR

En ciencias de la computación es común encontrar traductores, que pertenecen al grupo de software y cuya función principal es precisamente traducir código de un lenguaje a otro. En el caso específico de este capítulo se dedica el estudio al lenguaje dedicado para uso en traductores del tipo ensamblador que mapea una a una las sentencias escritas por el programador a su equivalente en lenguaje de máquina, y es el lenguaje de más bajo nivel conocido.

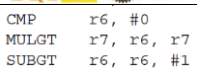
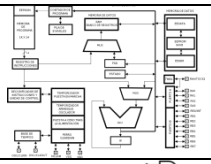
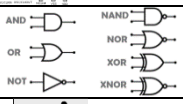
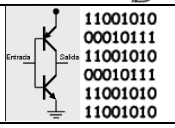
Tabla XXII. **Definición de lenguaje de ensamblador (asm)**

Lenguaje de programación de bajo nivel orientado a computadores y otros dispositivos programables. Las instrucciones de código de máquina de la arquitectura están fuertemente ligadas a representaciones simbólicas por medio de mnemónicos y otros elementos de este tipo de lenguaje, y varía entre procesadores. Suele utilizarse para controladores de hardware en la actualidad.

Fuente: elaboración propia.

Al hablar de lenguajes de programación, se considera bajo nivel a cualquier elemento que se acerque más al nivel de la máquina (representaciones en bits), y alto nivel a los cercanos a la lógica del programador (componentes complejos), y son más sencillos de comprender por un humano.

Tabla XXIII. **Jerarquía de computación**

Aplicaciones/OS	
Lenguajes de programación	
ISA	
Microarquitectura	
Compuertas	
Transistores	

Fuente: HOHL, William; HINDS, Christopher. *ARM Assembly Language Fundamentals and Techniques*. p. 14.

Los dispositivos que pertenezcan a distintos fabricantes serán, por lo general, capaces de ejecutar las mismas instrucciones básicas (con excepción de aquellas muy específicas); sin embargo la forma en la que ejecutan las operaciones pueden ser distinta debido a que las arquitecturas difieren y no todas cuentan con la misma distribución de silicio, componentes, capacidad de memoria, entre otros. Por esta razón, las instrucciones en lenguaje ensamblador serán específicas para la familia de dispositivos con que se trabaje. En el caso de este capítulo (y los posteriores), interesa solamente lo referente a la arquitectura Arm, y el contenido se limitará a este grupo de procesadores.

5.1. Historia

La aparición del lenguaje de ensamblador está fuertemente ligada al inicio del desarrollo de los computadores *stored-program* (que almacenan código en memoria en lugar de dispositivos externos), en 1949 con mnemónicos de una sola letra. IBM lo adoptó en 1955 con la creación del lenguaje SOAP (*Symbolic Optimal Assembly Program*), y a partir de entonces, comenzó a utilizarse ampliamente debido a que alivianaba a los programadores la carga de memorizar códigos numéricos, para escribir instrucciones y seguía siendo óptimo para los dispositivos de 8 bits, hasta que en 1980 su uso disminuyó con la llegada de los lenguajes de alto nivel y aún más con la aparición de procesadores de 16, 32 y 64 bits (donde el tamaño del código no suele ser la principal preocupación).

En la época dorada del lenguaje de ensamblador, los programas eran escritos para procesadores con todo tipo de objetivos y abarcaban desde el *Apollo Guidance Computer* (AGC), hasta dispositivos de entretenimiento como Atari ST y la mayoría de videojuegos que corrían en la consola NES (*Nintendo Entertainment System*).

Figura 122. **Videojuegos programados en lenguaje ensamblador**

Fuente: Retro games. *The 50 best games of the '80s*. https://www.gamesradar.com/best-retro-games. Consulta: 11 de junio de 2018.

En la actualidad el uso de lenguaje ensamblador es poco frecuente y suele reservarse a los ámbitos de educación e investigación, con excepción de algunos casos especiales en los que sea necesaria la manipulación directa de hardware, en los que se incluye:

- Sistemas embebidos con memorias muy pequeñas y orientación a tareas simples muy específicas.
- Sistemas con limitaciones estrictas en recursos.
- Elementos de ejecución estrecha con el hardware como *bootloaders*, *drivers* y virus.
- Diseño de compiladores.
- Aplicaciones que requieren uso de instrucciones específicas a las que no se accede sencillamente a través de un compilador.
- Sistemas que necesitan ser acelerados respecto a su funcionamiento con lenguajes de alto nivel.

- Sistemas de algoritmos criptográficos con exigencias en tiempos de ejecución muy rígidas para evitar ataques temporales.

Tabla XXIV. **Consideraciones en el uso de lenguaje de ensamblador**

Ventajas	Desventajas
Ejecución total rápida por la cercanía lógica a los componentes del procesador y baja abstracción. Eficiencia en tamaño. Flexibilidad en uso del hardware. Aprovechamiento máximo de recursos. Ausencia de etapas de interpretación y traducción presentes en compiladores.	Las exigencias hacia el programador son mayores, el código requerirá mayor inversión de tiempo y conocimiento de la arquitectura. Archivos de código fuente suelen ser largos. Falta de portabilidad a otras arquitecturas.

Fuente: elaboración propia.

5.2. Herramientas de desarrollo Arm

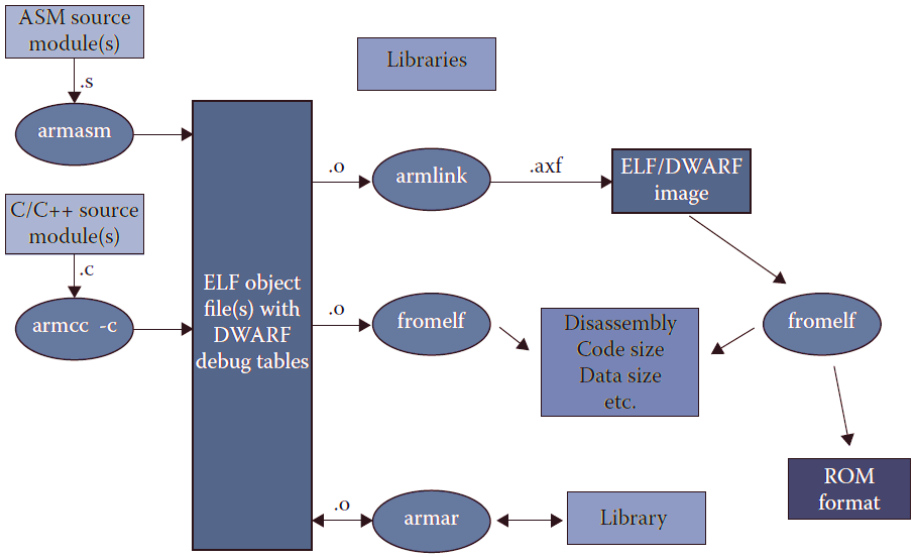
Como se mencionó anteriormente, a partir de cierto punto los lenguajes de alto nivel se volvieron más comunes que los correspondientes a los de ensamblador. La historia de la computación fue desde programadores que tenían que lidiar con código binario, pasó por lenguajes tempranos como COBOL, FORTRAN y ALGOL hasta verse desplazados por C, C++, Java y Python.

Todos estos lenguajes tienen en común la característica de estar descritos en inglés (un idioma de origen humano), requiriendo que el código se traduzca al set de instrucciones del procesador a través de un compilador. En el caso de mantener la necesidad de escribir en lenguaje de ensamblador a código de máquina entonces se utilizará un tipo similar de traductor similar al anterior, llamado precisamente ensamblador, para producir código entendible directamente por el hardware. Estas traducciones requerirán que el

programador cumpla reglas en la escritura de código como espaciado, sintaxis, uso de marcadores, entre otros.

Para comprender del proceso que sigue el código escrito para llegar al dispositivo, considerar la figura 123:

Figura 123. **Flujo de herramientas Arm**



Fuente: HOHL, William; HINDS, Christopher. *ARM Assembly Language Fundamentals and Techniques*. p. 26.

En este diagrama se distinguen las siguientes etapas:

- El archivo fuente de código pasa a través del ensamblador para producir un archivo objeto (.o). El compilador hará lo mismo con código escrito en lenguajes como C o C++.
- El enlazador se usa para combinar los archivos objeto en un solo programa ejecutable, incluyendo librerías en caso de ser necesario.

- El depurador provee acceso a registros en el chip, información de memoria y la habilidad de manipular *breakpoints* y *watchpoints*.

5.2.1. Herramientas *Open Source*

En el ambiente académico es común la elección de herramientas que no impliquen pago o trámites complicados para obtención de licencias. La mayoría de compañías no suelen tener problema para ofrecer algunos componentes fácilmente; sin embargo, existen organizaciones sin fines de lucro que proveen paquetes libres. En este grupo se encuentra Linaro (con desarrollo específico para Arm), incluyendo kernel para Linux y la cadena de herramientas GCC, que suelen ser útiles tanto para aplicaciones en máquina virtual o *bare metal.*

5.2.2. Arm KEIL

Este grupo de herramientas está diseñado para generar código optimizado para sets de instrucciones Arm, contando con paquetes especializados como RVMDK (*Real View Microcontroller Development Kit*), para visualización de lenguaje de alto nivel con su equivalente en lenguaje de ensamblador en la misma pantalla y DS-MDK para soporte no sólo de microcontroladores, sino de procesadores de la línea Cortex-A.

Las herramientas de Keil contienen:

- Compiladores para C y C++.
- Macro ensambladores.
- Enlazador.
- Depurador con simulador de CPU y periféricos para los chips Arm más populares.

- Entorno de desarrollo integrado (IDE) y Vision con editor de código equipado y gestor de proyectos.
- Perfilador de ejecución y analizador de desempeño.
- Opciones de conversión de archivos.
- Documentación conteniendo manuales, hojas de datos y guías de usuario.

5.2.3. Code Composer Studio (CCS)

Grupo de herramientas propio de Texas Instruments (TI), como uno de los socios más grandes de Arm. CCS está hecho para Arm y el resto de productos manejados por TI, e incluye algunas características extra. Está diseñado para soportar tanto microcontroladores como SoCs grandes (como las líneas DaVinci y Sitara).

El entorno de desarrollo está basado en Eclipse, y es frecuentemente preferido por la comunidad *open source*.

Code Composer Studio incluye entre sus herramientas:

- Compiladores para todas las familias de dispositivos TI
- Editor de código fuente
- Ambiente para construcción de proyectos
- Depurador
- Perfilador de código
- Simuladores
- Sistema de operación en tiempo real

5.3. Código de máquina

El lenguaje de máquina consiste en las instrucciones que puede ejecutar un procesador directamente con sus componentes de hardware, debe tenerse en cuenta que este no está en un nivel de abstracción que corresponda a lo entendible por los humanos.

Figura 124. **Ejemplo de instrucción en código de máquina**

Fuente: elaboración propia.

Si la trama de bits es equivalente a una instrucción, resulta evidente que no es un tipo de programación intuitivo. Por su parte, entonces, el lenguaje de ensamblador asigna a cada instrucción un equivalente en sílabas llamadas mnemónicos (que evocan en el programador la idea de la instrucción que se ejecuta). Los operadores binarios se suelen transformar en equivalentes hexadecimales por facilidad en escritura. Como un ejemplo de estos, observar la figura 125:

Figura 125. **Ejemplo de mnemónico en lenguaje de ensamblador**

Fuente: elaboración propia.

Aquí, el código de máquina sube un nivel en abstracción para conformar la sintaxis simplificada en la imagen. Se distingue que el mnemónico utilizado es

MOV (común en muchos sets de instrucciones), que corresponde a un modo de representar la instrucción *move*; así, la figura 125 representa la instrucción 'mover el valor 0x061 al registro Rn'.

Igualmente, este ejemplo sirve para distinguir un punto importante del lenguaje de ensamblador: las partes de una instrucción.

Figura 126. **Partes de una instrucción en lenguaje de ensamblador**

Fuente: elaboración propia.

Por la correspondencia entre lenguaje de ensamblador y de máquina, la imagen anterior representa las partes comunes de una instrucción para ambos:

- Un mnemónico para representar la instrucción que se ejecuta. Este puede ir acompañado de prefijos o sufijos (una de las opciones o ambas) para extender el funcionamiento.
- Operandos. Que pueden ir desde cero hasta la cantidad que permita el set de instrucciones (usualmente dos, uno fuente y otro destino).
- Comentarios que, como en todo lenguaje de programación, no son reconocidos por el ensamblador o compilador.

5.3.1. Componentes del código de máquina

Para ejecutar instrucciones, el lenguaje de bajo nivel se vale de elementos que permiten interactuar con el hardware y manipular su comportamiento. Se

considera que los tres principales son las subsecciones siguientes y que, a partir de ellos es posible ejecutar todo tipo de instrucciones.

5.3.1.1. Registros

La existencia de estos tipos de memoria se han discutido ampliamente para el ambiente Arm a lo largo de secciones anteriores, y en esta simplemente se hará un recuento de sus características más relevantes para el programador.

- Son memorias veloces de poca capacidad, conformadas por arreglos de *flip flops*.
- A diferencia de la memoria que puede adaptarse en tamaño al código, los registros están siempre habilitados y presentes en hardware.
- Existen algunos registros disponibles para uso ilimitado, mientras otros se encuentran restringidos por comprometer el funcionamiento interno del procesador (como es el caso de los registros de estado, contador de programa*, link register* y puntero de pila).
- Los coprocesadores también cuentan con registros propios a los que se puede acceder. Como es el caso de las unidades de punto flotante.
- Suelen identificarse por una letra mayúscula (R, S, D, W, etc.) y un número por orden.
- Sobre ellos actúan las instrucciones. Es decir, funcionan como operandos.

5.3.1.2. Posiciones de memoria y *offsets*

Por ser de muchísima mayor capacidad, las localidades que contiene una memoria son más que las de los registros.

Suelen haber secciones totalmente restringidas al programador para evitar daños en el dispositivo; sin embargo, las posiciones disponibles funcionan como operandos (es decir, se puede ejecutar instrucciones sobre la información que se almacena en ellas). Esto resulta útil no sólo para manipular los datos que se encuentran dentro del procesador, sino para la información de coprocesadores y de periféricos, en caso de que estos se encuentren mapeados a memoria.

Para un uso eficiente de la memoria y reducción del código se utiliza la definición de la tabla XXV.

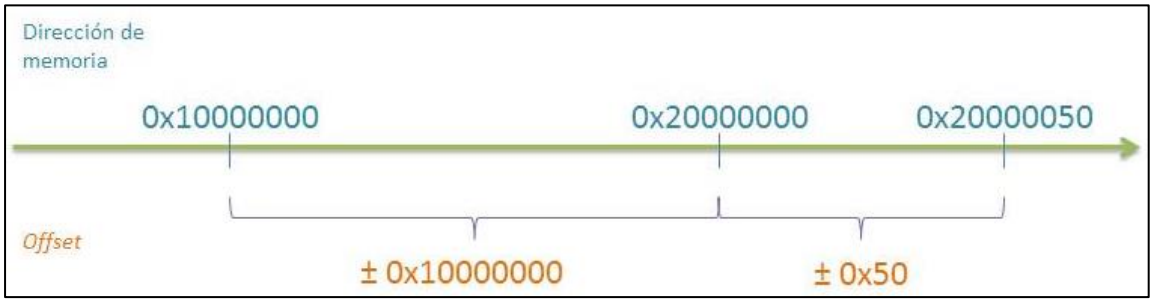
Tabla XXV. **Definición de *offset* (ciencia de la computación)**

En el área de informática se conoce el término *offset* como un desplazamiento entre una localidad inicial y una final, especialmente en el caso de memorias.

Fuente: elaboración propia.

Para mayor comprensión del término, considerar la figura 127:

Figura 127. ***Offsets* en localidades de memoria**



Fuente: elaboración propia.

Aquí se aprecia en la parte superior del eje las posiciones de memoria y en la inferior los *offsets* (representados en números hexadecimales). Pensar, por ejemplo que se desea posicionar el resultado de una operación en la localidad 0x20000000 (dirección absoluta). Se podrá acceder especificando la dirección como tal o indicando al ensamblador que el lugar al que deseamos acceder es 0x10000000 (la dirección base, quizás algo disponible en caché por uso frecuente) más el *offset* 0x10000000. En este caso, quizás para el programador lleva exactamente la misma cantidad de tiempo escribir dos números hexadecimales de ocho dígitos; la utilidad llega frecuentemente para interactuar con varias posiciones cercanas para las que en el código simplemente se especifica la dirección base y luego para cada localidad por separado se opera el offset requerido, como sería el caso de que se especifique una dirección base 0x20000000 y varias direcciones absolutas en 0x20000050,0x20000060 y 0x20000070, para las que entonces podría indicarse que los *offsets* necesarios son 0x50, 0x60 y 0x70 respectivamente. Lo último simplifica la escritura de código, y es un recurso muy frecuentado.

5.3.1.3. Modos de direccionamiento

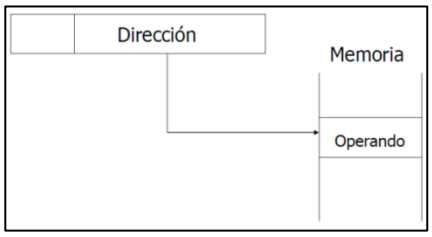
Estos se comprenden como técnicas en que se manipula de distinta forma los operadores para alcanzar objetivos como:

- Disminuir el ancho total de una instrucción
- Escribir código para direcciones dinámicas o desconocidas para el programador.
- Manejo eficiente de arreglos.

Existen muchas formas de alcanzar esto dependiendo de los recursos con los que se cuente. Sin embargo, se distinguen algunos básicos:

- Inmediato: el operando se obtiene de memoria al mismo tiempo que la instrucción, porque estos se dan al mismo tiempo, evitando requerir acceso a localidades extra o saltos en memoria. Su uso está dado cuando se define el valor de una constante o se reinicia una variable.
- Directo a memoria: en la instrucción se especifica la dirección absoluta del operando para encontrarlo en memoria.

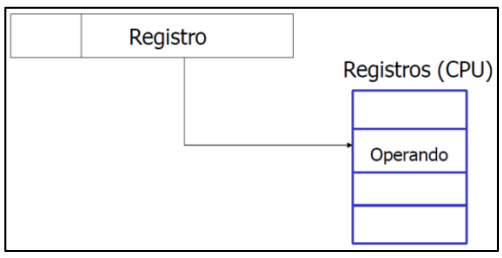
Figura 128. **Modo de direccionamiento directo a memoria**



Fuente: FERRARIO, Nicolás. *Organización de computadoras: Clase 7*. p. 33.

- Directo a registro: de funcionamiento similar al anterior, con la diferencia de que se especifica la dirección de un registro y no una localidad de memoria, y resulta más rápido.

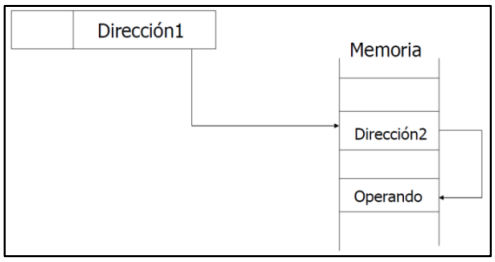
Figura 129. **Modo de direccionamiento directo a registro**



Fuente: FERRARIO, Nicolás. *Organización de computadoras: Clase 7*. p. 35.

- Indirecto por memoria: se especifica la dirección de la localidad donde se encuentra la dirección del operador que se busca. Esto con la intención de poder apuntar a direcciones largas, con la desventaja de tener que ejecutar varios saltos en memoria.

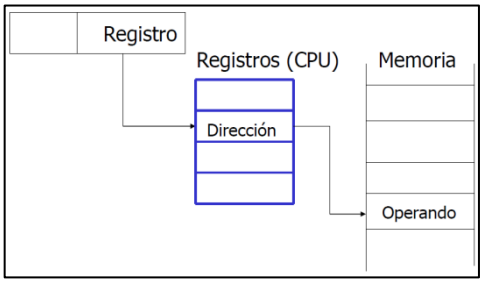
Figura 130. **Modo de direccionamiento indirecto por memoria**



Fuente: FERRARIO, Nicolás. *Organización de computadoras: Clase 7*. p. 37.

- Indirecto por registro: se especifica un registro que contiene la dirección del operando que se busca. Con esto se reduce los accesos de memoria y por lo tanto, la ejecución total es más rápida con la ventaja también de poder apuntar a direcciones grandes. Este es el modo de funcionamiento de los punteros comunes en lenguajes de alto nivel como C.

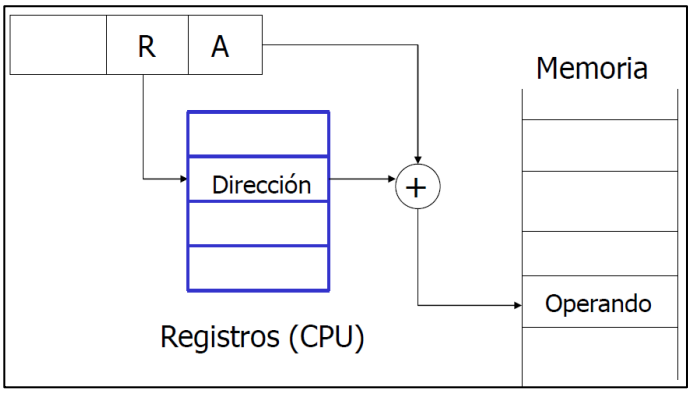
Figura 131. **Modo de direccionamiento indirecto por registro**



Fuente: FERRARIO, Nicolás. *Organización de computadoras: Clase 7*. p. 39.

- De pila: se utiliza el puntero de pila para acceder a la sección de datos tipo LIFO.
- Implícito: se especifica solamente el campo de instrucción (sin parámetros), ya que puede no acceder a memoria o hacerlo de forma específica que no requiera detallarlo.
- Por desplazamiento: el modo en que se especifica la dirección donde se encuentra el operando implica el uso de *offsets*.

Figura 132. **Modo de direccionamiento por desplazamiento**



Fuente: FERRARIO, Nicolás. *Organización de computadoras: Clase 7*. p. 41.

Los tipos más comunes del modo por desplazamiento son:

- Relativo a contador de programa.
- Relativo a registro.
- Indexado (disponible en algunas arquitecturas con registros del tipo índice).

5.4. Lenguaje de ensamblador en Arm

Si bien cada ensamblador tiene características propias y el lenguaje que va con él se transforma, el aprendizaje de este nivel de programación requiere, en ciertos momentos, iniciar la exploración para conocer por experiencia ciertos elementos y luego entonces volver a lo básico para reafirmar la teoría y continuar. Se trata más de un aprendizaje pendular más que uno estrictamente lineal. Por esta razón, es probable que quien se inicia en el campo requiera volver a estas secciones mientras se avanza en la práctica (especialmente cuando se conozca más de un ensamblador). Se limitará el texto es a las herramientas relacionadas a Arm.

5.4.1. Lineamientos para programación en ensamblador

Debido a que el lenguaje de ensamblador se encuentra en el nivel más bajo dentro de las herramientas disponibles para programación, suele significar un reto aprenderlo de cero. Sin embargo; siguiendo los pasos adecuados y teniendo una base adecuada en teoría, es posible dominar el lenguaje sin problemas al final del proceso.

Con el fin de preparar al aprendiz para iniciar en el mundo del lenguaje de ensamblador, los autores William Hohl y Christopher Hinds proponen en su libro *ARM Assembly Language Fundamentals and Techniques* (texto recomendado por la comunidad Arm), algunos lineamientos a tomar en cuenta cada vez que se practique o realicen módulos nuevos. Estos se traducen y resumen así:

- Dividir el problema en trozos. Cuando se presentan problemas extensos es mejor probar soluciones a tareas pequeñas para asegurar su

funcionamiento y luego poder unificarlas, esto reduce el desgaste de buscar secciones problemáticas en códigos muy largos.

- Correr una prueba en el código finalizado. A pesar de que el código parezca tener una lógica correcta, es posible que existan casos especiales en los que no funcione adecuadamente y el programador no esté considerando. Se recomienda probar casos extremos en donde el funcionamiento adecuado se vea comprometido.
- Procurar el uso de todas las herramientas disponibles con el fin de visualizar con facilidad el funcionamiento del código paso a paso, se recomienda implementar características de los entornos como *breakpoints, watchpoints*, detalles de registros, banderas y memoria.
- Siempre asumir que el código será leído por alguien más. Es importante definir nombres y etiquetas con los que pueda comprenderse por intuición cómo el código trabaja en el momento en que este se escribe. Esto evita que incluso el mismo programador recurra tiempo después a su código almacenado y no se encuentre en capacidad de comprenderlo por ambigüedad.
- Es preferible un código simple y funcional sobre uno sofisticado con problemas de ejecución.
- Dejar los códigos óptimos a la experiencia. Cuando se inicia en el lenguaje es probable que los módulos no sean de lo más eficientes, sin embargo las técnicas para lograrlo se aprenden con el tiempo. Es preferible comenzar teniendo en mente la solución del problema.
- Abandonar el miedo a intentar cosas nuevas. Si bien, el lenguaje de ensamblador requiere conocimiento técnico del dispositivo con que se trabaja; nunca es un error explorar con código y aprender por empirismo cómo se comportan los elementos disponibles.
- Para ciertos programadores es útil implementar diagramas de flujo antes de abordar un problema. Esto depende de la comodidad de quien escribe

el código y la ayuda que signifique para visualizar adecuadamente la situación a la que se enfrenta.

- Prestar atención a la inicialización. Con este tipo de lenguajes es importante no asumir que los valores esperados estarán como conviene al inicio del módulo, en cambio se recomienda verificarlo y observar si estos necesitan ser reiniciados en algún punto del código. Este punto aplica para constantes, variables y registros.

5.4.2. Sets de instrucciones en el ensamblador

Las herramientas ARM correspondientes al desarrollo de software para Armv7 y Armv8 tienen soporte para los siguientes ensambladores:

- El armasm, que es el principal y maneja código de A64, A32, T32, Thumb y ARM con sintaxis armasm.
- Los ensambladores embebidos en los compiladores de C y C++ (uno con optimización y otro sin ella).
- El integrado armclang para sintaxis GNU.
- Un ensamblador para optimización construido para lenguaje de ensamblador en sintaxis GNU usado con código de C y C++.

El ensamblador de Arm soporta instrucciones, directivas y macros definidas por el usuario en:

- *Unified Assembly Language* (UAL) para ARM, Thumb, A32 y T32.
- Lenguaje de ensamblador para código A64.
- Instrucciones NEON en ARM, Thumb, A32, T32 y A64.
- Instrucciones de punto flotante en ARM, Thumb, A32, T32 y A64.

- Instrucciones de tecnología MMX para ensamblar código destinado al procesador PXA270 de Intel.

Tabla XXVI. **Unified Assembler Language**

El UAL es un lenguaje en común entre ARM, Thumb, A32 y T32 que permite unificar la sintaxis de todos los sets en una sola. Resulta conveniente para portabilidad entre dispositivos y hacia versiones posteriores. El lenguaje Thumb antes de UAL no cuenta con soporte para instrucciones de 32 bits.

Fuente: elaboración propia.

5.4.2.1. Arm, Thumb y ThumbEE

El set ARM fue el primero en aparecer en este grupo con instrucciones de 32 bits. Según la arquitectura fue avanzando apareció Thumb desde Armv4T con instrucciones de 16 bits. La mayoría de la funcionalidad de ARM se encuentra disponible en el segundo set, aunque algunas operaciones necesiten más instrucciones para llevarse a cabo. Esto último se considera un pago por la mejora en densidad de código.

A partir de Armv6T2 se introdujo la tecnología Thumb-2, que sumó a las ventajas de Thumb que se soporten instrucciones de 32 bits. Esto provee casi el mismo nivel de rendimiento que ARM con, de nuevo, mejoras en la densidad de código.

Thumb-2 fue incluida en Armv7; sin embargo, el perfil M sólo soporta el set Thumb mientras los perfiles A y R soportan ARM y Thumb.

ThumbEE (Thumb Execution Environment), fue definido desde Armv7 y es un set de instrucciones con base en Thumb y algunas modificaciones para mejorar la funcionalidad de código generado dinámicamente.

5.4.2.2. A32 y T32

Los términos A32 y T32 son alias para ARM y Thumb, respectivamente, y sus definiciones son similares.

A32 no es soportada por el perfil M en ninguna versión, mientras A y R soportan A32 y T32 en estado de ejecución AArch32.

5.4.2.3. A64

Las instrucciones de son de 32 bits de ancho. Este set fue introducido en Armv8 y sólo está disponible en estado de ejecución AArch64.

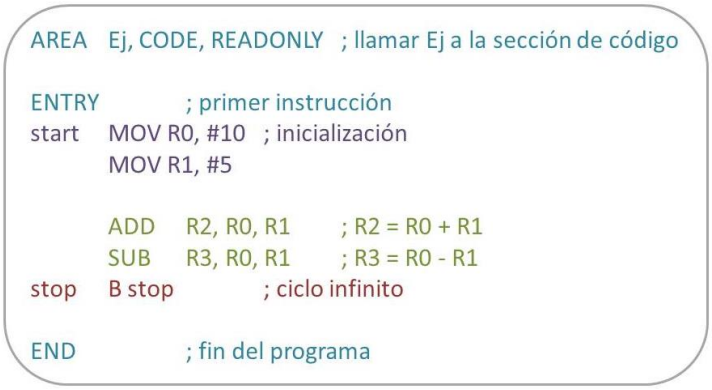
Su funcionamiento es similar al de A32 y T32, con la ventaja de dar acceso a espacios de direccionamiento virtual mayores.

5.4.3. Estructura de módulos

Como se menciona al inicio de la sección 5.4, la sintaxis para cada ensamblador suele ser distinta y resulta necesario que el programador se familiarice con cada uno en caso de una migración. A pesar de esto es posible adentrarse en las características del lenguaje de forma general. Las consideraciones específicas se mencionarán en los capítulos posteriores para que sea posible replicar la práctica propuesta en Arm.

Considerar, como primer paso, el código de la figura 133.

Figura 133. **Estructura de módulos en lenguaje de ensamblador**



Fuente: elaboración propia.

Para este punto no interesa tanto comprender totalmente cómo el programa se ejecuta como las secciones que lo conforman. El código de la figura 133 se compone de las siguientes etapas:

- Al inicio, en celeste, las directivas. Que son instrucciones que se dan al ensamblador acerca de cómo manejar el código.
- Luego, en morado, la primera parte del programa en que se cargan a los registros R0 y R1 valores constantes.
- En verde, la segunda parte del programa utilizando los registros para operar una suma y una resta almacenando los resultados en R2 y R3.
- En rojo, una línea para inducir un ciclo infinito sin ejecutar nada al final de las operaciones.
- Por último, de nuevo en celeste, la directiva que indica el fin del código.

Es de esperar que los módulos en lenguaje ensamblador estén estructurados de una manera similar y que, como se observa en la imagen, cada sentencia (línea de código) tenga una estructura más o menos así:

{etiqueta} {instrucción/directiva/pseudoinstrucción} {;comentario}

Con cada parte dentro de llaves teniendo un carácter opcional, esto hace obligatorio que las instrucciones, directivas y pseudoinstrucciones sean precedidas por un espacio (que acostumbra ser una tabulación por orden visual), en caso contrario se marcará como error.

Para la escritura de código en UAL el programador debe tener en cuenta que las palabras reservadas del lenguaje se escriben todas en mayúsculas o todas en minúsculas, no permitiendo combinaciones como “Add, aDD” o similares. Para Keil cada línea tiene un límite de 4095 caracteres y CCS, 400.

5.4.4. Comentarios

En herramientas Keil, el primer punto y coma de la línea indica el inicio de un comentario, a menos que este se encuentre en una cadena de caracteres. Por otro lado, en herramientas de TI, se permite un asterisco al inicio de la línea para identificar un comentario o un punto y coma en cualquier otro punto.

5.4.5. Etiquetas

Las etiquetas son nombres que se asignan a direcciones de memoria en código específicas de tal forma que resulte más sencillo acceder a ellas, para el programador al estar escritas en lenguaje humano. Se permite la definición de estas una sola vez para evitar accesos erróneos, con excepción de las etiquetas numéricas locales que comienzan con números en el rango de 0 a 99. Todas deben escribirse al inicio de la línea para la mayoría de ensambladores.

Su uso principal se distingue al trabajar con instrucciones para jugar con el flujo del programa, atribuyendo una etiqueta a cada grupo de instrucciones. Ejemplos de esto se verán más adelante.

5.4.6. Literales

Para el caso de UAL, se utilizarán los siguientes formatos de constantes:

- Números naturales (3, 150, 2000).
- Hexadecimales con prefijo 0x o & para cualquier cantidad de dígitos, 0f_ para 8 dígitos y 0d_ para 16 dígitos (0xFF, &0EC1, 0f_11111EEE, 0d_0FFFFF0F0F0000FF).
- Números de cualquier base n en el rango de 2 a 9 con exponentes en el rango de 0 a (n-1), exclusivo de Keil (4_20, 5_340, 9_18).
- Números de punto flotante (5.1, -2E-7, 18.6E2) con rango en precisión simple de 1.17549435E-38 a 3.40282347E+38 y en precisión doble de 2.22507385850720138E-308 a 1.7976931348623157E+308.
- Valores booleanos ({TRUE} o {FALSE}).
- Valores de un caracter (‘a’, ‘t’, ‘d’).
- Cadenas de caracteres (prueba, código, fin).

Las constantes numéricas son de 32 bits en A32, T32, ARM y Thumb con rango de 0 a $(2^{32}-1)$, para números sin signo o de -2^{31} a $(2^{31}-1)$, para números con signo. En A64 las constantes son de 64 bits de ancho con rango de 0 a $(2^{64}-1)$, para números sin signo o de -2^{63} a $(2^{63}-1)$, para números con signo.

Aunque en algunos ensambladores es opcional, se aconseja que las constantes asignadas inmediatamente sean precedidas por un # siempre, de la forma: MOVT R1, #256. Ejemplos de esto se verán más adelante.

5.4.7. Nombres predefinidos de registros

La mayoría de ensambladores tienen, entre las palabras reservadas, nombres intuitivos de registros para que hacer más sencilla la escritura y lectura del código.

Para Arm, los nombres predefinidos más utilizados son:

Tabla XXVII. **Nombres de registros predefinidos para uso en UAL**

Nombre de registro	Uso
r0-r15 o R0-R15	Registros de propósito general.
a1-a4	Registros de argumento, resultado o incialización. Sinónimos de R0-R3.
v1-v8	Registros de variable. Sinónimos de R4-R11.
sb o SB	Registro base estático. Sinónimo para R9.
ip o IP	Registro de llamdo interno de procedimientos. Sinónimo de R12.
sp o SP	Puntero de pila. Sinónimo de R13.
lr o LR	Registro de enlace. Sinónimo de R14.
pc o PC	Contador de programa. Sinónimo para R15.
s0-s31 o S0-S31	Registros de punto flotante con precisión simple.
cpsr o CPSR	Registros de estado de programa actual.
spsr o SPSR	Registros de estado de programa guardado.
apsr o APSR	Registros de estado de programa de aplicación.
fpscr o FPSCR	Registros de estado de programa de actual de FPU.

Fuente: elaboración propia.

5.4.8. Funciones matemáticas

Code Composer Studio tiene la ventaja de contar con funciones que, si bien no deben utilizarse en el código, ofrecen una forma de realizar chequeos o declaraciones rápidas de secciones de datos. Algunas de estas son:

- $$cos(x), donde x es un número de punto flotante
- $$sin(x), donde x es un número de punto flotante
- $$log(x), donde x debe ser mayor a 0
- $$max(x1, x2), que devuelve el mayor de los dos valores
- $$sqrt(x), donde x debe ser mayor o igual a 0

5.4.9. Operadores

Para operaciones primitivas que no pertenecen a las instrucciones pero puedan resultar útiles en la manipulación de los datos, se emplean los operadores (por lo general binarios):

Tabla XXVIII. **Operadores para uso en UAL**

Función	Símbolo	Ejemplo
Suma	+	A+B
Resta	-	A-B
Multiplicación	*	A*B
División	/	A/B
A módulo B	:MOD:	A:MOD:B
Rotar A hacia la izquierda B bits	:ROL:	A:ROL:B
Rotar A hacia la derecha B bits	:ROR:	A:ROR:B
Desplazar A hacia la izquierda B bits	:SHL:	A:SHL:B
Desplazar A hacia la derecha B bits	:SHR:	A:SHR:B
AND	:AND:	A:AND:B
OR exclusiva	:EOR:	A:EOR:B
OR	:OR:	A:OR:B
AND lógica	:LAND:	A:LAND:B
OR lógica	:LOR:	A:LOR:B
Concatenar B al final de A	:CC:	A:CC:B
Igual a	=	A=B
Mayor que	>	A>B
Mayor o igual que	>=	A>=B
Menor que	<	A<B
Menor o igual que	<=	A<=B
No igual a	/=, <>	A/=B, A<>B
Negación	:NOT:, ~	:NOT:A, ~A

Fuente: elaboración propia.

5.4.10. Directivas

Las directivas son palabras reservadas en lenguaje de programación que sirven para indicar al ensamblador cómo tratar el código escrito. No forman parte del set de instrucciones y nunca llegan realmente al procesador, porque su única función es guiar el proceso de ensamblaje. Debido a que estas corresponden a las herramientas utilizadas, se distinguen grupos para los propósitos de este texto:

5.4.10.1. Directivas en herramientas Keil

- Definición de bloques de datos o código: con el propósito de asignar correctamente datos en memoria de tipo RAM y código en tipo ROM, se especifica a la herramienta las secciones a las que se desea atribuir estas características. La directiva principal a utilizar en este caso es la palabra AREA, en la forma de la figura 134.

Figura 134. **Directiva de área**

Fuente: elaboración propia.

Donde 'nombre' puede ser casi cualquier secuencia de caracteres, con algunas excepciones como |.text | (que hace que el código pueda llamarse desde módulos escritos en C). Los nombres que comienzan con números deben encerrarse en barras (e.g. |1_seccion|).

Cada programa debe contener, por lo menos, una sección marcada con directiva AREA con alguno de los atributos mostrados en la tabla XXIX:

Tabla XXIX. **Atributos de la directiva AREA**

Atributo	Función
ALIGN=n	Alínea la sección en un límite de 2^n bytes.
CODE	La sección corresponde a código de máquina (instrucciones).
DATA	La sección corresponde a datos.
READONLY	Indica que el área puede ubicarse en ROM, atributo por defecto en secciones CODE.
READWRITE	El área puede ubicarse en RAM, por defecto para secciones DATA.

Fuente: elaboración propia.

- Definición de nombres de registros: la directiva encargada de esta función es RN, se utiliza para mejorar la comprensión del código escrito. Su estructura es:

Figura 135. **Directiva para nombre de registro**

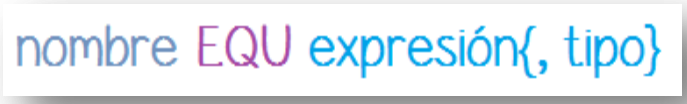
Fuente: elaboración propia.

Donde 'nombre' puede ser casi cualquier secuencia de caracteres, con excepción de los predefinidos de la tabla XXIX y el campo expresión puede contener cualquier número en el rango de 0 a 15.

- Igualación de símbolo y constante: es común que en los lenguajes de programación aparezcan constantes con nombres asignados que las

hacen fáciles de comprender en el funcionamiento del código (como la función #define en C). La sintaxis de la directiva EQU se observa a continuación:

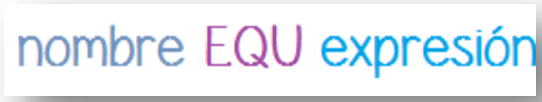
Figura 136. **Directiva para igualación de constante**



Fuente: elaboración propia.

Donde 'nombre' es el que se dará a la constante para usarse en el código, el campo 'expresión' es una dirección relativa al programa, una dirección absoluta o una constante de 32 bits y 'tipo' es un parámetro opcional entre ARM, THUMB, CODE16, CODE32 y DATA por eso la directiva suele llevar nombre y expresión.

Figura 137. **Directiva para igualación de constante en forma común**



Fuente: elaboración propia.

- Alineación de código y datos a límites apropiados: según sea más óptimo para el procesador o el programa escrito, puede requerirse alineaciones específicas de la información. La directiva ALIGN realiza la operación ajustando las localidades actuales con ceros.

Figura 138. **Directiva de alineación**

Fuente: elaboración propia.

Donde 'expresión' es cualquier número que resulte de elevar 2 a una potencia entre 0 y 31 y *offset* puede ser cualquier número. La localidad actual entonces se alinea a la siguiente dirección que corresponda a (*offset* + (n*expresión). Si esto no se especifica, ALIGN opera en el límite de la siguiente palabra (4 bytes).

- Finalización de archivo fuente: al terminar el código en un módulo es necesario indicarlo al ensamblador. Esto se hace simplemente con la directiva END en la última línea.

5.4.10.2. Directivas en herramientas Texas Instruments

- Definición de secciones: para declaraciones similares a Keil de información en datos o instrucciones se utiliza la directiva de la figura 139:

Figura 139. **Directiva de sección**

Fuente: elaboración propia.

La sección por defecto del compilador para ubicar el código se llama .text, que es donde se encuentra el programa de ensamblador. Otras opciones de sección son:

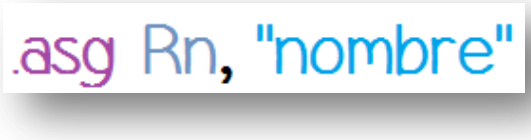
Tabla XXX. **Directivas de sección TI**

Directiva	Función
.bss	Reserva espacio en la sección .bss.
.usect	Reserva espacio en una sección con nombre específico y no inicializada.
.text	Sección por defecto para ubicar código por el compilador.
.data	Usada para variables o tablas pre incializadas.
.sect	Define el nombre de una sección similar a .text o .data

Fuente: elaboración propia.

- Asignación de nombres a registros: con función similar a RN en Keil, para TI se utiliza la forma de la figura 140.

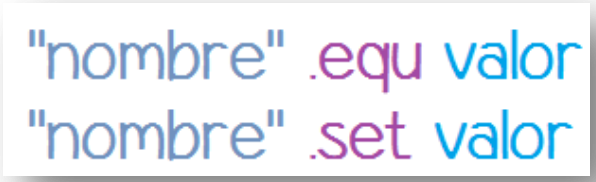
Figura 140. **Asignación de nombres a registros**



Fuente: elaboración propia.

- Igualación de constantes: para asignar nombres simbólicos a constantes que se usarán en el código se utilizan las dos formas de la figura 141.

Figura 141. **Directivas de asignación de nombre a constantes, 1**



Fuente: elaboración propia.

Estas directivas tienen la característica de soportar asignación de cualquier valor numérico constante o servir de alternativa a .asg.

- Alineación: para alinear el PC de la sección a un límite en particular, se utiliza la forma de la figura 142.

Figura 142. **Directivas de asignación de nombre a constantes, 2**

Fuente: elaboración propia.

Donde n es la cantidad de bytes a la que se desea alinear. Si este número no aparece, el límite es un byte.

- Finalización del archivo fuente: para terminar el código de un módulo se ubica la directiva .end en la última línea.

5.4.11. Macros

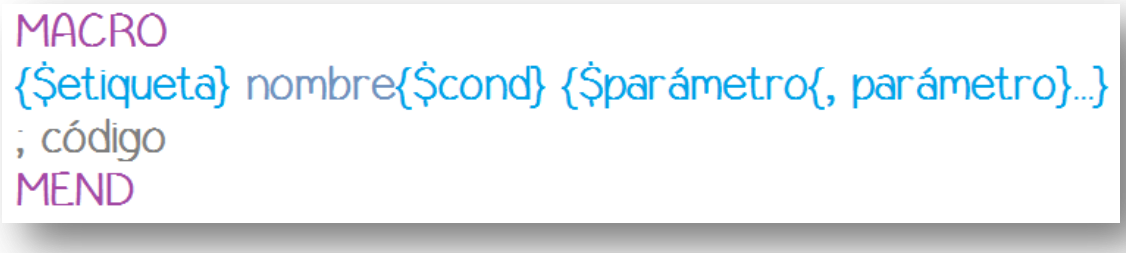
En todo lenguaje de programación existe el concepto de funciones definidas por el programador que contienen operaciones específicas y luego pueden llamarse desde otra parte del código, para repetir el funcionamiento reduciendo el tamaño del archivo y la repitencia.

El uso de macros dependerá de la conveniencia, considerando que la ventaja principal es disminuir el código, pero la desventaja es la ineficiencia en memoria si ocupan un lugar grande y se usa muy frecuentemente.

La diferencia de estas definiciones con los llamados de subrutinas es que el ensamblador reemplaza el código de una macro al lenguaje apropiado, y su uso solamente compete al programador y el procesador recibe el código con las funciones repetidas cada vez que se utiliza la macro.

En herramientas Keil las macros se definen por las directivas MACRO y MEND con la forma:

Figura 143. **Definición de una macro**



Fuente: elaboración propia.

Donde 'etiqueta' es un símbolo opcional que se da cuando la macro es llamada, 'cond' es un campo condicional y 'parámetro' son los valores sobre los que actúa la función a definir y se sustituyen cuando esta se llama.

5.4.12. Tipos de instrucciones

Las guías de uso de ensambladores Arm distinguen tipos de instrucciones en las que pueden agruparse todas las existentes en sets de instrucciones de la arquitectura como se enlistan a continuación.

- Ramificación y control: en este grupo están las instrucciones para efectuar saltos entre rutinas, controlar ciclos, estructuras condicionales y cambiar modos de procesador.
- Procesamiento de datos: estas instrucciones operan en los registros de propósito general con funciones aritméticas y lógicas.
- Carga y almacenamiento en registro: conjunto destinado al intercambio de información entre un registro y memoria.
- Carga y almacenamiento en múltiples registros: con la misma función que el tipo anterior, con la diferencia de actuar sobre conjuntos de registros de propósito general, en lugar de sólo uno.
- Acceso a registro de estado: para intercambio de información entre registro de estado y uno de propósito general.
- De coprocesador: conjunto destinado a la extensión de la arquitectura, usualmente habilitando el uso de registros de coprocesador de control de sistema (CP15).

5.4.13. Condicionamiento

Como se explicó en la sección anterior, existen instrucciones que se ejecutan con base en la condición de las banderas en el APSR (modificado por una instrucción anterior). Este tipo de operación requiere un análisis individual porque es la base del control de la línea de ejecución de un programa en la mayoría de los casos con la implementación de instrucciones de ramificación (*branch*). El uso estas instrucciones dependerá de la conveniencia debido a que suele tomar tres ciclos de reloj volver a llenar una estructura de *pipeline* cada vez que se ejecuta un *branch,* con excepción de los procesadores con predicción de saltos (que solo lo harán cuando el mecanismo no acierte).

Para escribir instrucciones condicionales se añade a los mnemónicos sufijos de condición, indicando al procesador que debe revisar las banderas, para decidir si debe o no ejecutar la instrucción, escribir valores, afectar banderas o generar excepciones. La tabla XXXI muestra los sufijos que pueden utilizarse, las banderas de las que dependen y su significado:

Tabla XXXI. **Sufijos de condición**

Sufijo	Bandera	T1
EQ	Z=1	Igual
NE	Z=0	No igual
CS o HS	C=1	Mayor o igual
CC o LO	C=0	Menor
MI	N=1	Negativo
PL	N=0	Positivo o cero
VS	V=1	Desbordamiento
VC	V=0	No desbordamiento
HI	C=1 y Z=0	Mayor
LS	C=0 y Z=1	Menor o igual
GE	N y V iguales	Mayor o igual con signo
LT	N y V distintos	Menor con signo
GT	Z=0, N y V iguales	Mayor con signo
LE	Z=1, N y V distintos	Menor o igual con signo

Fuente: elaboración propia.

Cabe aclarar que los sufijos (y prefijos), que se añaden a mnemónicos en lenguaje ensamblador modifican el significado de una instrucción tanto como lo hacen los existentes en el idioma español. Considerar, por ejemplo, la palabra nacional que cumple la función central de la idea y cómo el prefijo inter modifica su significado para formar internacional; esto sucede con muchas otras palabras como vicepresidente, submarino, carnívoro y responsable (las últimas dos como ejemplo de sufijos). Así en lenguaje ensamblador estas extensiones de los mnemónicos no modificarán el funcionamiento básico de la instrucción, sino simplemente las hacen específicas para lo que se desea ejecutar, ocurriendo de igual forma para prefijos que indican ancho de instrucción o uso de VFP.

Además, de los sufijos, existen a partir de Armv6T2 mnemónicos que ya son condicionales:

- Instrucciones CBZ (salto condicional en cero) y CBNZ (salto en no cero)
- Instrucción IT (si-entonces) de 16 bits para instrucciones Thumb

Tabla XXXII. **Diferencias principales de sintaxis entre UAL y A64**

UAL	A64
Instrucciones generales	
La condicionalidad de se marca uniendo sufijos directamente al mnemónico (e.g. BEQ).	La condicionalidad se especifica con un sufijo separado por un punto del mnemónico (e.g. B.EQ).
Los sufijos .W y .N indican ancho de instrucción de 32 o 16 bits respectivamente.	El ancho de instrucción es de 32 bits fijos, no se admiten .W y .N.
Los registros principales tiene nombres de R0 a R15.	El tamaño de los datos se especifica en registros de 32 bits (W0-W31) o 64 bits (X0-X31).
Los registros R13, R14 y R15 son llamados SP, LR y PC.	El registro 31 es SP, WZR o XZR. El 30 es llamado LR.

Continuación de la tabla XXXII.

SIMD y VFP	
Todos los mnemónicos comienzan con V (e.g. VMAX).	La primera letra de la instrucción indica tipo de dato a operar: S para signo, U para sin signo y F para punto flotante.
Un calificador de mnemónico especifica tipo y ancho de elementos en un vector.	El calificador se ubica en el registro.
Los vectores de 128 bits con llamados de Q0 a Q15 y los de 64 bits de D0 a D31.	Todos los vectores son llamados de V0 a V31. Q, D, S, H y B con usados para indicar escalares.
Existe condicionalidad.	No existen instrucciones de punto flotante o SIMD avanzado condicionalmente ejecutadas.

Fuente: elaboración propia.

6. PROGRAMACIÓN DE PROCESADOR CORTEX-M4F UTILIZANDO EL MICROCONTROLADOR TM4C123GH6PM

La sección desarrollada a continuación se presenta como una guía introductoria al manejo de lenguaje de ensamblador para programación de procesadores Cortex-M4F.

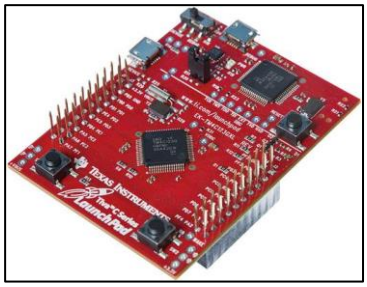
Las instrucciones básicas disponibles han sido divididas en grupos fundamentales para comprensión ordenada y progresiva. Cada conjunto cuenta con ejercicios, para apoyar la teoría de cada subsección y los capítulos 2, 3 y 5, se recomienda la consulta del set de instrucciones del procesador para información extra.

6.1. Tarjeta de desarrollo Tiva C

En el ámbito educativo es ampliamente conocido el microcontrolador TM4C123GH6PM como un dispositivo con alta capacidad (32 bits), fabricado por Texas Instruments con características que lo hacen accesible a la comunidad estudiantil. Con un procesador Cortex-M4F, la tarjeta de desarrollo Tiva C cuenta con las ventajas de un núcleo de gama media dentro del perfil destinado a microcontroladores (ver capítulo 3), con capacidad de ejecutar operaciones DSP, de punto flotante y bajo consumo de potencia. Además, el microcontrolador cuenta con circuitería externa para tareas especializadas que lo hacen una opción confiable para implementación y aprendizaje. La tarjeta complementa las características con elementos extras como segmento de depuración, reloj de cristal externo para precisión y led contenidos en la misma placa.

La suma de todas las características relacionadas con el precio de la Tiva C la convirtieron en una opción ampliamente utilizada para el aprendizaje de microcontroladores y sirviendo como una introducción a programación avanzada en lenguaje de ensamblador y C, se desarrolla la sección teniendo en cuenta las características de la tarjeta de desarrollo mencionada sin perder de vista que el tema principal es la programación del procesador, y podrían utilizarse los mismos fundamentos para otros microcontroladores y tarjetas.

Figura 144. **Tarjeta de desarrollo Tiva C con microcontrolador TM4C123GH6PM**



Fuente: Cortex. *Procesador Cortex-A76*. https://www.conrad.de/de/entwicklungsboard-texas-instruments-ek-tm4c123gxl-515674.html. Consulta: 9 de noviembre de 2018.

6.1.1. Periféricos

Tabla XXXIII. **Definición de periférico**

Circuito independiente dedicado a una función específica que sirve a las instrucciones de alguna unidad de procesamiento central. Al hablar de un microcontrolador, puede diferenciarse entre los periféricos del núcleo del procesador, los del chip que contiene al procesador, los del microcotrloador e incluso, los de la tarjeta en que el sistema pueda estar implementado.

Fuente: elaboración propia.

Los periféricos del TM4C123GH6PM se encuentran conectados a los pines de uso múltiple en la tarjeta de desarrollo Tiva C. Por esta razón, algunos de ellos cuentan con funciones alternativas según especifique el programador (podrían, por ejemplo, funcionar como entrada o salida y como UART). Detalles con estos circuitos se estudiarán al final de este capítulo, por ahora simplemente se enlistan los disponibles en la tarjeta:

- GPIO (*General Purpose Inputs and Outputs*): Pines de entrada y salida de propósito general. Cada pin puede ser alcanzado a través de dos buses. *Advanced Peripheral Bus* (APB), para compatibilidad hacia abajo con dispositivos anteriores y *Advanced High-Performance Bus* (AHB) para mejor rendimiento.
- SSI (*Synchronous Serial Interface*).
- UART (*Universal Asynchronous Receiver-Transmitter*).
- I2C (*Inter-Integrated Circuit*).
- PWM (*Pulse Width Modulator*).
- QEI (*Quadrature Encoder Interface*).
- Temporizadores de 16, 32 y 64 bits.
- ADC (*Analog Digital Converter*).
- CAN (Controller Area Network).
- USB (*Universal Serial Bus*).

6.1.2. Memoria

El microcontrolador tiene una distribución fija de 4GB de memoria con posibilidad de *bit-banding* en SRAM y periféricos, también se reserva una región para el bus periférico privado (PPB), por ser de uso exclusivo en el núcleo.

Los periféricos se encuentran mapeados a memoria, esto quiere decir que para acceder a ellos basta con ingresar en la localidad de memoria correspondiente. Será como asignar a cada espacio físico del periférico un nombre hexadecimal que corresponde a la dirección en que se almacena toda su información. Para esto es posible utilizar instrucciones de carga y almacenamiento, como se estudiará más adelante en este capítulo.

El mapa de memoria del microcontrolador se especifica con rangos hexadecimales para cada región:

Tabla XXXIV. **Mapa de memoria para microcontrolador TM4C123GH6PM**

Inicio	Fin	Descripción
Memoria		
0x00000000	0x0003FFFF	Flash en el chip
0x00040000	0x1FFFFFFF	Reservado
0x20000000	0x20007FFF	SRAM en el chip con bit-banding
0x20008000	0x21FFFFFF	Reservado
0x22000000	0x220FFFFF	Alias de bit-banding para SRAM
0x22100000	0x3FFFFFFF	Reservado
Periféricos		
0x40000000	0x40000FFF	Temporizador *watchdog* 0
0X40001000	0X40001FFF	Temporizador *watchdog* 1
0X40002000	0X40003FFF	Reservado
0X40004000	0X40004FFF	Puerto A GPIO
0X40005000	0X40005FFF	Puerto B GPIO
0X40006000	0X40006FFF	Puerto C GPIO
0X40007000	0X40007FFF	Puerto D GPIO
0X40008000	0X40008FFF	SSI 0
0X40009000	0X40009FFF	SSI 1
0X4000A000	0X4000AFFF	SSI 2
0X4000B000	0X4000BFFF	SSI 3
0X4000C000	0X4000CFFF	UART 0
0X4000D000	0X4000DFFF	UART 1
0X4000E000	0X4000EFFF	UART 2
0X4000F000	0X4000FFFF	UART 3
0X40010000	0X40010FFF	UART 4
0X40011000	0X40011FFF	UART 5
0X40012000	0X40012FFF	UART 6
0X40013000	0X40013FFF	UART 7
0X40014000	0X4001FFFF	Reservado

Continuación de la tabla XXXIV.

Periféricos		
0x40020000	0x40020FFF	I2C 0
0x40021000	0x40021FFF	I2C 1
0x40022000	0x40022FFF	I2C 2
0x40023000	0x40023FFF	I2C 3
0x40024000	0x40024FFF	Puerto E GPIO
0x40025000	0x40025FFF	Puerto F GPIO
0x40026000	0x40027FFF	Reservado
0x40028000	0x40028FFF	PWM 0
0x40029000	0x40029FFF	PWM 1
0x4002A000	0x4002BFFF	Reservado
0x4002C000	0x4002CFFF	QEI 0
0x4002D000	0x4002DFFF	QEI 1
0x4002E000	0x4002FFFF	Reservado
0X40030000	0X40030FFF	Temporizador 0 (16/32 bits)
0X40031000	0X40031FFF	Temporizador 1 (16/32 bits)
0X40032000	0X40032FFF	Temporizador 2 (16/32 bits)
0X40033000	0X40033FFF	Temporizador 3 (16/32 bits)
0X40034000	0X40034FFF	Temporizador 4 (16/32 bits)
0X40035000	0X40035FFF	Temporizador 5 (16/32 bits)
0X40036000	0X40036FFF	Temporizador 0 (32/64 bits)
0X40037000	0X40037FFF	Temporizador 1 (32/64 bits)
0X40038000	0X40038FFF	ADC 0
0X40039000	0X40039FFF	ADC1
0X4003A000	0X4003BFFF	Reservado
0X4003C000	0X4003CFFF	Comparador analógico
0X4003D000	0X4003FFFF	Reservado
0x40040000	0x40040FFF	Controlador CAN 0
0x40041000	0x40041FFF	Controlador CAN 1
0x40042000	0x4004BFFF	Reservado
0x4004C000	0x4004CFFF	Temporizador 2 (32/64 bits)
0x4004D000	0x4004DFFF	Temporizador 3 (32/64 bits)
0x4004E000	0x4004EFFF	Temporizador 4 (32/64 bits)
0x4004F000	0x4004FFFF	Temporizador 5 (32/64 bits)
0x40050000	0x40050FFF	USB
0x40051000	0x40057FFF	Reservado
0x40058000	0x40058FFF	Puerto A GPIO (apertura AHB)
0x40059000	0x40059FFF	Puerto B GPIO (apertura AHB)
0x4005A000	0x4005AFFF	Puerto C GPIO (apertura AHB)
0x4005B000	0x4005BFFF	Puerto D GPIO (apertura AHB)
0x4005C000	0x4005CFFF	Puerto E GPIO (apertura AHB)
0x4005D000	0x4005DFFF	Puerto F GPIO (apertura AHB)
0x4005E000	0x400AEFFF	Reservado
0x400AF000	0x400AFFFF	EEPROM y *locker* de llaves
0x400B0000	0x400F8FFF	Reservado
0x400F9000	0x400F9FFF	Módulo de excepción del sistema
0x400FA000	0x400FBFFF	Reservado
0x400FC000	0x400FCFFF	Módulo de hibernación
0x400FD000	0x400FDFFF	Control de memoria Flash
0x400FE000	0x400FEFFF	Control de sistema
0x400FF000	0x400FFFFF	Micro DMA

Continuación de la tabla XXXIV.

0x40100000	0x41FFFFFF	Reservado
0x42000000	0x43FFFFFF	Alias de bit-banding para 0x40000000-0x400FFFFF
0x44000000	0xDFFFFFFF	Reservado
0XE0000000	0xFFFFFFFF	PPB

Fuente: Tiva*. TM4C123GH6PM datasheet*. p. 94.

6.1.3. El archivo Startup

Al observar detenidamente la distribución de memoria es notorio que cada elemento en el microcontrolador tiene asignada una dirección en memoria conteniendo su información para lectura o escritura. Existen componentes establecidos por defecto mientras otras especificaciones pueden ser dadas por el programador, como el tamaño de pila (sabiendo que esta se ubicará en el inicio de la RAM).

Un archivo *Startup* se compone en:

- Declaración de área de pila
- Declaración de área de *heap* (memoria reservada para almacenamiento mientras el programa corre).
- Tabla de vectores.
- Código de controlador de reinicio.
- Código de otros controladores de excepciones.

Por el contenido de este tipo de archivos, su lectura es fundamental para la preparación del dispositivo antes de cargar el programa ya que el *Startup* indicará cómo se manejarán interrupciones, ubicación de pila, controladores especificados por el usuario y qué sucederá en el programa inmediatamente

después de un reinicio o la primera vez que el código se carga en el procesador. Es importante recordar, que este archivo será lo primero en leerse antes de cualquier otro segmento de código y que se ejecuta una sola vez por reinicio o carga, especificando en el control de reinicio el punto exacto de código al que se saltará después en el archivo de código principal, para ejecutar el resto del programa por lo general utilizando alguna instrucción *branch*. En el caso de los ejemplos más adelante en este capítulo se verá de la forma B Start, y cada uno incluirá una etiqueta llamada Start como la primera línea del programa que se ejecutará.

6.1.4. Keil versus CCS para Cortex-M

En el capítulo 5 se evaluaron algunas diferencias entre la escritura de directivas para herramientas Keil y Code Composer Studio. El programador puede decidir cuál set se adapta a sus necesidades y comodidad tomando en cuenta las características de cada uno:

- Code Composer Studio es desarrollado por Texas Instruments, y tiene amplio soporte en dispositivos fabricados por esta empresa. El entorno de programación es muy amigable y por su base en Eclipse resulta familiar a muchos programadores. Cuenta con espacios de depuración y ventanas de monitoreo. Cuenta con algunas funciones para uso en ensamblador que, si bien no son la forma más óptima de programar, simplifican de alguna forma el diseño de código.
- Keil uVision tiene soporte creado específicamente para Arm, y no sólo es mayor que el de CCS sino que para el programador supone una forma de adaptarse a la programación de todos los dispositivos Arm, (no sólo los de un fabricante). Su entorno de programación también es amigable al usuario con visualización de ventanas para monitoreo, depuración y

simulación. Algunas versiones soportan procesadores Cortex-A y Cortex-M, ampliando su utilidad. Por último, una de las razones de más peso para su elección es que el ensamblador es capaz de transformar ciertas formas incorrectas de escritura de instrucciones a un formato legal automáticamente, pudiendo observarse el cambio a través del desensamblador.

Por las razones expuestas se ha decidido utilizar exclusivamente Keil uVision v5 en este capítulo dejando claro que la transición a CCS, simplemente requiere repasar la documentación que Texas Instruments ha liberado al respecto y está al alcance del público general.

6.1.5. El segundo operador flexible en Arm

Existen operaciones de procesamiento de datos que contienen un segundo operando en forma de constante (representado por lo general como Op2). Este solamente puede tener las formas enlistadas con X y Y como valores hexadecimales:

- Cualquier constante producida por el desplazamiento de un valor de 8 bits en una palabra de 32 bits.
- Cualquier constante de la forma 0x00XY00XY.
- Cualquier constante de la forma 0xXY00XY00.
- Cualquier constante de la forma 0xXYXYXYXY.

Algunas instrucciones permiten algunos otros valores (como el caso de la suma, que permite hasta 0xFFF o su equivalente en números naturales conocido como #imm12), restricciones son la principal causa de elegir utilizar la pseudoinstrucción LDR Rt, =imm32 (sección 6.2), donde imm32 significa una

constante de 32 bits, o formas compuestas de instrucciones de movimiento de datos.

El ensamblador de Keil se encuentra en capacidad de transformar algunas formas erróneas de escritura de este operando a instrucciones legales.

6.2. Carga y almacenamiento

El primer grupo a explorar en UAL es el de las instrucciones destinadas a cargar valores, moverlos entre localidades y almacenarlos. Su importancia es equivalente a decir que para efectuar 2+3 se necesitan los dígitos 2 y 3, este conjunto se encarga de ubicar los datos en el lugar que se necesita para operar o guardar en memoria. Tomando en cuenta la definición de las arquitecturas Arm en el capítulo 2, estas se comportan de la forma *load/store*, esto quiere decir que su diseño hace necesario acceder a memoria para obtener información y luego operarla (y viceversa). Considerando las instrucciones más utilizadas, se presenta su resumen:

Tabla XXXV. **Instrucciones básicas UAL para carga y almacenamiento Cortex-M4**

Mnemónico	Instrucción	Estructura
LDM	Cargar múltiples registros	Rn, lista
LDR	Cargar registro	Rt, [Rn{, #offset}] Rt, =imm32
MOV	Mover	Rd, Op2 Rd, #imm16
MOVW	Mover a los 16 bits menos significativos	Rd, #imm16
MOVT	Mover a los 16 bits más significativos	Rd, #imm16
MVN	Mover NOT	Rd, Op2
MOV32	Mover constante de 32 bits	Rd, #imm32
STM	Almacenar múltiples registros	Rn, lista
STR	Almacenar registro de palabra	Rt, [Rn{, #offset}]

Fuente: elaboración propia.

Para simplificar el análisis de estas, considerar tres subgrupos:

- Las instrucciones de carga toman valores en memoria y los escriben en registros de propósito general. LDR lo hace con un valor y LDM con varios de ellos.
- Un caso especial de este tipo es la llamada pseudoinstrucción.
- LDR Rt, =imm32 que se utiliza para cargar constantes en registros como una alternativa al siguiente subgrupo.
- Las instrucciones para mover se utilizan para atribuir valores inmediatos (imm) a registros o constantes en la forma de segundo operador. Son las instrucciones más utilizadas para esta finalidad, con diversas opciones según la necesidad del programador.
- Las instrucciones de almacenamiento toman valores en registros de propósito general y los escriben en memoria. STR y STM tienen, por lo tanto, similitud a LDR y LDM con funcionamiento inverso.

6.2.1. Ejercicio 1

El primer programa explora el uso de las instrucciones para mover valores, es el más sencillo de comprender junto con la explicación de las directivas en el capítulo anterior.

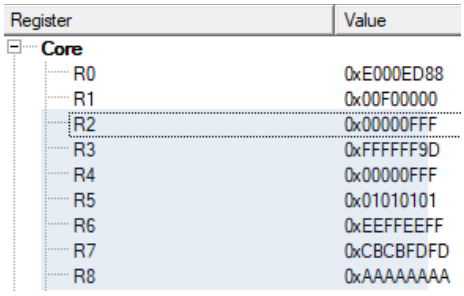
Figura 145. **Programa 1**

```
    AREA    codigo, CODE, READONLY,ALIGN=2
    THUMB
    EXPORT Start

Start   ; Punto de entrada a partir del archivo Startup.

; Para enteros de hasta 12 bits o de la forma de segundo operador flexible.
    MOV     R2, #4095
    MOV     R3, #-99
    MOV     R4, #0XFFF
    MOV     R5, #0x01010101
    MVN     R6, #0x11001100     ; Escribe 0xEEFFEEFF.
; Para enteros de hasta 32 bits.
    MOVW    R7, #0xFDFD         ; Bits menos significativos de R7.
    MOVT    R7, #0xCBCB         ; Bits más significativos de R7.
    MOV32   R8, #2863311530

    ALIGN
    END                         ; Final del programa.
```

Fuente: elaboración propia.

Con resultados visibles en el depurador como se observa en la figura 146.

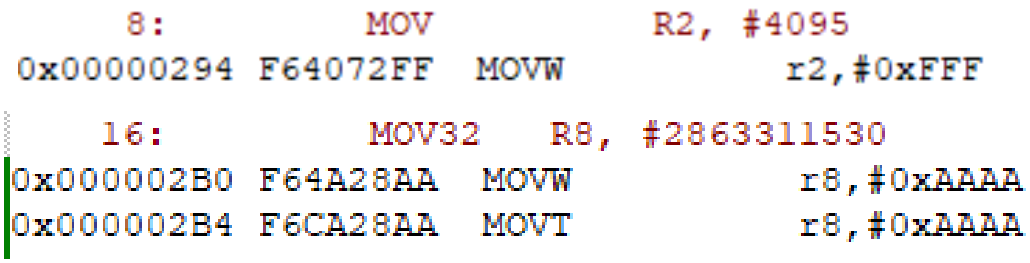
Figura 146. **Vista de registros programa 1**



Fuente: elaboración propia.

La instrucción MOVW en algunos sets puede escribirse simplemente como MOV, y el uso de la segunda se extenderá a 16 bits; sin embargo siempre es recomendable consultar en la documentación. En el caso de este programa se utilizan ambos mnemónicos con la ventaja de que el ensamblador se encarga de elegir la instrucción más adecuada, como se observa en la ventana de desensamblador:

Figura 147. **Vista de desensamblador programa 1**



Fuente: elaboración propia.

En la figura 147 se distingue cómo el ensamblador tomó la instrucción MOV R2, #4095 y la convirtió en MOVW R2, #0xFFF por conformidad con el ancho de los operadores. También convirtió MOV32 es su forma extendida (MOVW+MOVT).

Como se mencionó antes, el ensamblador de Keil ofrece opciones para facilitar al programador el diseño de código.

A partir de este punto, se conocen las formas de asignar valores a registros y se construirá poco a poco programas más elaborados con finalidades concretas.

6.2.2. Ejercicio 2

Este programa está diseñado para demostrar el funcionamiento fundamental de las instrucciones de carga y almacenamiento, tomando en cuenta existen formas más complejas de utilizarlas y que exigen estudio amplio de la documentación adecuada.

Como primer paso se muestra el código completo:

Figura 148. **Programa 2**

```
DIRECCION   EQU 0x20000008 ; Dirección en RAM.

    AREA    codigo, CODE, READONLY,ALIGN=2
    THUMB
    EXPORT Start

Start ; Punto de entrada a partir del archivo Startup.
    MOV32 R1, DIRECCION

    ; Uso de pseudoinstrucción alternativa LDR Rn, =constante.
    LDR R2, = 108900
    LDR R3, = 12345
    LDR R4, = 8000
    LDR R5, = 752

    ; Almacenamiento de valores a RAM individualmente con aumento
    ; de 4 bytes en dirección cada vez.
    STR R2, [R1], #4
    STR R3, [R1], #4
    ; Almacenamiento colectivo de R4 y R5 en RAM.
    STM R1, {R4, R5}
    MOV32 R1, DIRECCION ; Reinicio de dirección.

    ; Carga colectiva desde RAM a registros R6, R7 y R8.
    LDM R1!,{R6,R7,R8}
    ; Carga individual desde RAM a R9 con la última dirección.
    LDR R9, [R1]

    ALIGN
    END
```

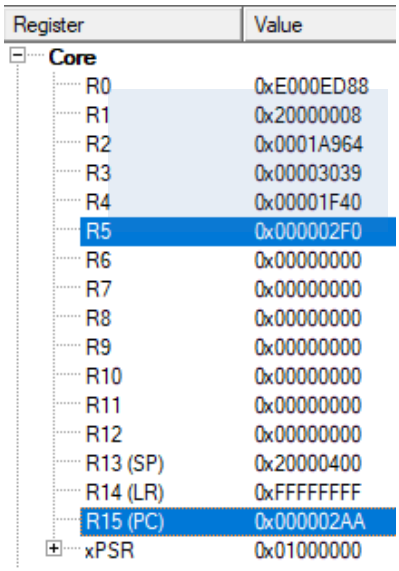
Fuente: elaboración propia.

Puede distinguirse a simple vista que se introduce el uso de la directiva EQU para igualar la etiqueta DIRECCION al valor 0x20000008 que, lejos de ser aleatorio, se eligió por estar dentro de la RAM del dispositivo a usar y se utiliza más de una vez en el programa. Si, por caso contrario se eligiera una dirección dentro de la ROM sería posible leer valores pero no escribirlos.

Para cargar valores se utiliza la alternativa a las opciones del Programa 1. Nótese que con esta sintaxis las constantes no están precedidas por un signo numeral (#), en cambio como parte de la pseudoinstrucción se escribe un signo igual (=).

En el depurador la carga de valores en registros se verá como se muestra a continuación con valores convertidos automáticamente a su forma hexadecimal:

Figura 149. **Primera vista de registros programa 2**



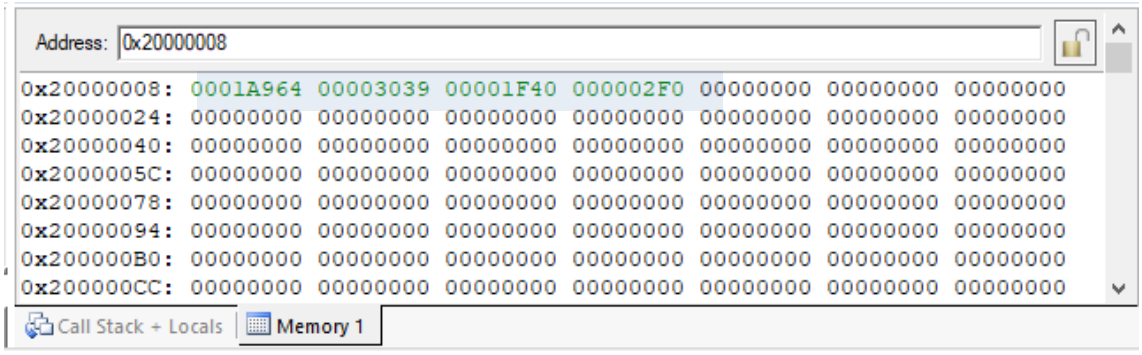
Fuente: elaboración propia.

El programa luego seguirá los siguientes pasos:

- Se carga el contenido de R2 a la dirección de memoria escrita en R1 (0x20000008), luego de escribir se suma 4 a R1. Recordar que cada aumento o decremento en direcciones de memoria tiene por unidad los bytes, y un aumento de 4 es igual a 32 bits (siguiente palabra).
- Se carga el contenido de R3 a la nueva dirección en R1 (0x2000000C), luego de escribir se suma 4 nuevamente a R1.
- Se carga en la nueva dirección escrita en R1 (0x20000010) el contenido de R4 y R5.

Para este punto del programa, la memoria tendrá los valores:

Figura 150. **Vista de memoria programa 2**



Fuente: elaboración propia.

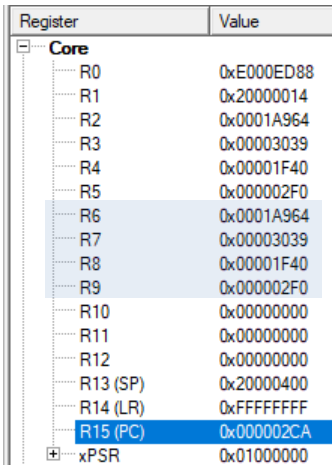
Luego, para efectuar instrucciones de lectura desde memoria en las mismas localidades, el programa efectúa:

- Se reinicia el valor de R1 a 0x20000008.

- Se lee a partir del valor de R1 tres palabras en memoria y se escriben correspondientemente en R6, R7 y R8. El signo de exclamación (!) señala que se sumará al valor de R1 la cantidad de bytes que haya avanzado para efectuar la instrucción (que en este caso son 12).
- Se lee el valor guardado en la nueva dirección escrita en R1 (0x20000010) y se carga al registro R9.

Al terminar el proceso, los registros en el depurador se verán entonces:

Figura 151. **Segunda vista de registros programa 2**



Fuente: elaboración propia.

6.3. Operaciones de pila

Con similitud a las instrucciones de carga y almacenamiento se encuentran las de la tabla XXXVI que se comportan de la misma forma que LDM y STM utilizando R13 (puntero de pila, SP), como dirección base para las operaciones y una lista de registros como contenido para luego escribir de vuelta en SP la última dirección modificada. Es útil recordar que la pila se ubica

dentro de la RAM, que su dirección de inicio es igual a la base de la SRAM (0x20000000), su espacio se especifica en el archivo *Startup* y que es una memoria del tipo LIFO (*Last In, First Out*).

Tabla XXXVI. **Instrucciones básicas para interacción con pila Cortex-M4**

Mnemónico	Instrucción	Estructura
POP	Obtener registros de pila	{lista de registros}
PUSH	Almacenar registros en pila	{lista de registros}

Fuente: elaboración propia.

6.4. Aritmética y lógica

Luego de que el programador se encuentra capacitado para manipular la disponibilidad de los datos que se utilizarán en el sitio adecuado, puede comenzar a explorarse las operaciones para transformar la información.

En este grupo se enlistan las instrucciones de aritmética básica y lógica que, como en el resto de casos, pueden tener variaciones más complejas para fines específicos (como es el caso de las operaciones de matemática saturada). En esta subsección se presentan las que son consideradas fundamentales y de uso más frecuente.

Tabla XXXVII. **Instrucciones básicas UAL para aritmética y lógica Cortex-M4**

Mnemónico	Instrucción	Estructura
ADD	Suma	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
MUL	Multiplicación con resultado de 32 bits	{Rd,} Rn, Rm
SDIV	División con signo	{Rd,} Rn, Rm
SUB	Sustracción	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
UDIV	División sin signo	{Rd,} Rn, Rm
UMULL/ SMULL	Multiplicación sin/con signo y resultado de 64 bits	RLo, RHi, Rn, Rm
AND	Compuerta AND lógica	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
BIC	Limpieza de bits	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
EOR	Compuerta OR exclusiva (XOR)	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
ORR	Compuerta OR lógica	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
ORN	Compuerta NOR lógica	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
ROR	Rotación a la derecha	Rd, Rn, <Rsl#n>

Fuente: elaboración propia.

6.4.1. Ejercicio 3

Con la ventaja de que las instrucciones aritméticas y lógicas básicas tienen el mismo funcionamiento en cualquier contexto (el resultado en papel, ábacos y compuertas es el mismo que en programación), el ejercicio propuesto para su comprensión se simplifica tomando en cuenta los mnemónicos de la tabla XXXVII y su definición.

Figura 152. **Programa 3**

```
    AREA    codigo, CODE, READONLY,ALIGN=2
    THUMB
    EXPORT Start

Start

; Para operaciones aritméticas con resultado de 32 bits.
    LDR     R0, =9
    LDR     R1, =-2

    ADD     R2, R0, #6  ; R2=R0+4
    SUB     R1, #3      ; R1=R1-3
    MUL     R3, R2, R1  ; R3=R2*R1
    SDIV    R2, R1      ; R2=R2/R1

; Para multiplicación con resultado de 64 bits.
    MOV32   R4, 0xFFFFFFFF
    MOV32   R5, 0x3
    UMULL   R6, R7, R5, R4  ;(R6, R7)=R5*R4

; Para operaciones lógicas.
    MOV     R8, #0x10101010
    MOV32   R9, #0x11111101

    AND     R10, R8, R9 ; R10=R8&R9
    EOR     R11, R8, R9 ; R11=R8^R9
    ORR     R12, R8, R9 ; R12=R8|R9
    BIC     R8, R9      ; Limpiar en R8 los bits señalados por R9.
    ROR     R9, #8      ; Rotar R9 8 bits a la derecha.

    ALIGN
    END
```

Fuente: elaboración propia.

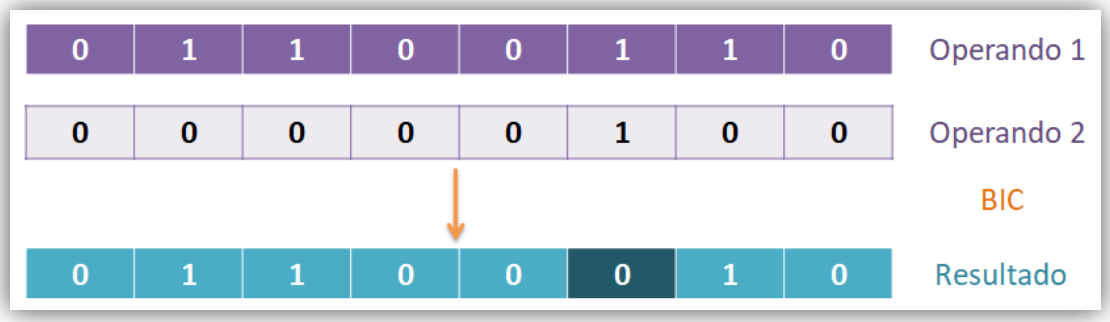
Con el primer grupo resulta importante observar que las instrucciones pueden prescindir de la escritura de uno de sus términos cuando el registro destino y primer operando son el mismo. Para valores en los que el signo deba mantenerse se aplican las operaciones adecuadas (SDIV contra UDIV).

Para multiplicaciones con resultados de hasta 64 bits, el set de instrucciones ofrece la opción asociada al mnemónico UMULL (en el caso de

operación sin signo), que actúa similar al grupo aritmético base con la excepción de que su resultado se sitúa en dos registros por exceder los 32 bits. El primer registro contendrá los bits menos significativos y el segundo, los más significativos.

Por último, el grupo de las instrucciones lógicas actúa conforme las reglas de las operaciones convencionales añadiendo al conjunto la posibilidad del mnemónico BIC que simplemente opera como se muestra en la figura 153.

Figura 153. **Instrucción BIC**



Fuente: elaboración propia.

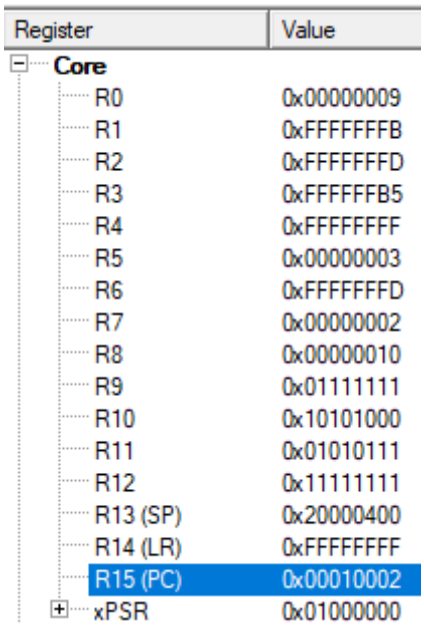
La instrucción BIC limpia los bits del primer operando que estén marcados por un 1 en el segundo, el resto no será modificado.

Figura 154. **Verdad de instrucción BIC**

Operando 1	Operando 2	Resultado
1	0	1
0	0	0
1	1	0
0	1	0

Fuente: elaboración propia.

Figura 155. **Vista de registros programa 3**



Fuente: elaboración propia.

6.4.2. Método leer modificar escribir

Jonathan Valvano y Ramesh Yerralbi introducen en su curso *Embedded systems: shape the world* el concepto de programación amigable como una forma de manipulación de los datos que procura modificar solamente los segmentos que se pretenden sin afectar el resto de la información. Esto resulta especialmente útil en el momento en que el lenguaje de ensamblador se utiliza para códigos de interacción directa con memoria ya sea de forma interna o por programación de los periféricos en un dispositivo, y sus beneficios quizás no sean tan evidentes en este punto pero sí lo serán en el desarrollo de la sección 6.8.

Tabla XXXVIII. **Acerca de la programación amigable**

En el diseño de sistemas embebidos es importante mantener la eficiencia del código al máximo. Para esto existen técnicas que evitan la sobre escritura innecesaria de bits en registros de interés. El mantra de la programación amigable se basa en el "*do not harm*", que implica la modificación solamente de los bits necesarios.

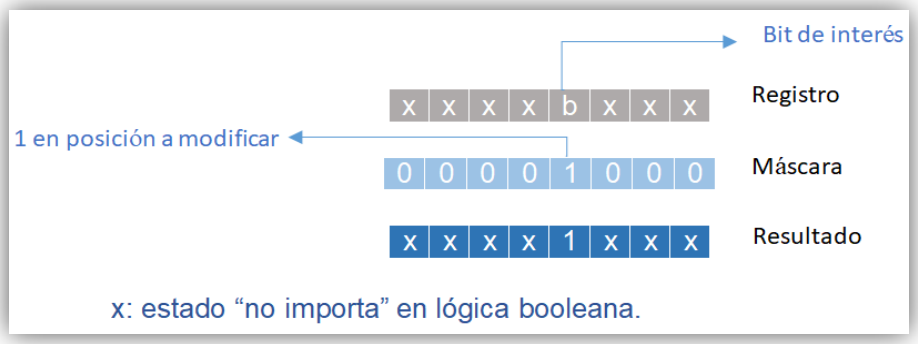
Fuente: elaboración propia.

Esta línea de diseño utiliza dos métodos, según el valor que se desee asignar a un bit.

6.4.2.1. O*r to set*

Utiliza una operación lógica OR entre el registro a modificar y una máscara de dígitos 1 para hacer *set* (escribir 1), en la posición deseada.

Figura 156. ***Or to set***

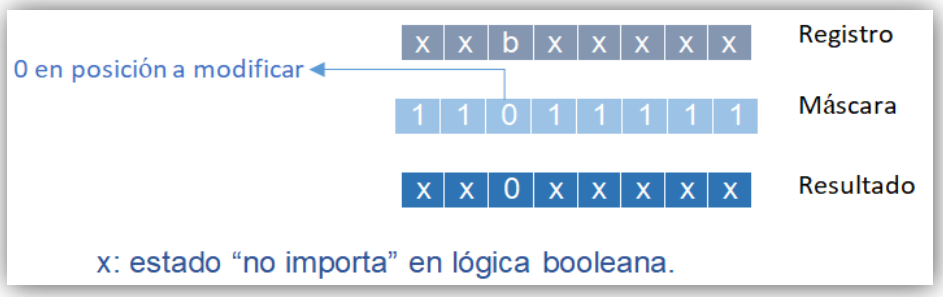


Fuente: elaboración propia.

6.4.2.2. *AND to clear*

Utiliza una operación lógica AND entre el registro y una máscara de dígitos 0 para hacer *clear* (escribir 0) en la posición deseada.

Figura 157. ***And to clear***



Fuente: elaboración propia.

Con base en estas dos técnicas de escritura de bits, el método leer modificar escribir utiliza cuatro pasos para control de periféricos:

- Asignar a un registro la dirección de memoria que interesa modificar.

- Cargar a un registro temporal el contenido de la localidad de memoria a la que corresponde la dirección del paso anterior.
- Realizar la operación para escribir dígitos en el contenido del registro.
- Almacenar el contenido modificado de vuelta en la localidad de memoria a la que corresponde la dirección del primer paso.

Según la opción requerida en el tercer paso, el método se divide entonces en:

- Para escribir dígitos 1 se recurre al leer-OR-escribir con el mnemónico ORR.
- Para escribir dígitos 0 se recurre al leer-AND-escribir con el mnemónico BIC.

Figura 158. **Ejemplo de formas leer modificar escribir**

Fuente: elaboración propia.

Notar que las instrucciones utilizadas en los ejemplos de la figura 158 han sido explicadas en los ejercicios anteriores, y esta se presenta como una aplicación de las mismas en el entorno de los sistemas embebidos.

Resulta importante observar que la máscara en las instrucciones lógicas está escrita en números hexadecimales (0x80 corresponde a modificar el tercer bit), esto se hace por simplicidad en las escritura y requiere simplemente una conversión de números binarios a hexadecimales.

Además, en lugar de una compuerta AND se utiliza la instrucción BIC con el fin de que ambas versiones de modificación funcionen con máscaras de dígitos 1 y así facilitar el trabajo del programador.

6.5. Condicionales y subrutinas

Hasta este punto son conocidas las instrucciones para operaciones básicas de trasformación y movilización de datos. Para el programador con cierto grado de experiencia en lenguajes de alto nivel resulta evidente que no se ha presentado una forma equivalente de escribir figuras condicionales (*if-else*), o ciclos (*for* y *while*), en lenguaje de ensamblador. Estos son, sin duda, elementos esenciales para la escritura de código funcional; por ser UAL un lenguaje de muy bajo nivel no existen declaraciones específicas dentro de la sintaxis para crear estas formas. Se opta entonces por escribirlas paso a paso con el uso de instrucciones condicionales y sufijos (será importante apoyarse en la teoría de la sección 5.4.13).

Debido a la cantidad de instrucciones que pueden utilizarse existen muchas formas de escribir código para un mismo fin. Dependerá del estilo del programador, su conocimiento de la ISA, la eficiencia y comodidad para decidir

cuál vía tomar. A pesar de esto puede evaluarse una forma básica para el aprendizaje de este grupo con las descripciones de la tabla XXXIX.

Tabla XXXIX. **Instrucciones básicas UAL para condicionamiento Cortex-M4**

Mnemónico	Instrucción	Estructura
B	Salto	etiqueta
BL	Salto con enlace	etiqueta
BLX	Salto indirecto con enlace	Rm
BX	Salto indirecto	Rm
CBNZ	Comparar y saltar si no es igual a cero	Rn, etiqueta
CBZ	Comparar y saltar si es igual a cero	Rn, etiqueta
CMN	Comparar negativo	Rn, Op2
CMP	Comparar	Rn, Op2
NOP	No operar	

Fuente: elaboración propia.

Este grupo está compuesto principalmente por las conocidas como instrucciones *branch* por el término en inglés que indica que pueden armarse ramas en el hilo principal de programa, para crear saltos a líneas no consecutivas (base de la instrucción *goto* en lenguaje C). El uso de sufijos permite evaluar cualquier condición necesaria para controlar el flujo de programa y las instrucciones de comparación actúan sobre las banderas modificadas por instrucciones anteriores, terminando de completar el conjunto destinado a cubrir todas las posibles variantes en que un programa puede desarrollarse.

6.5.1. Estructuras condicionales

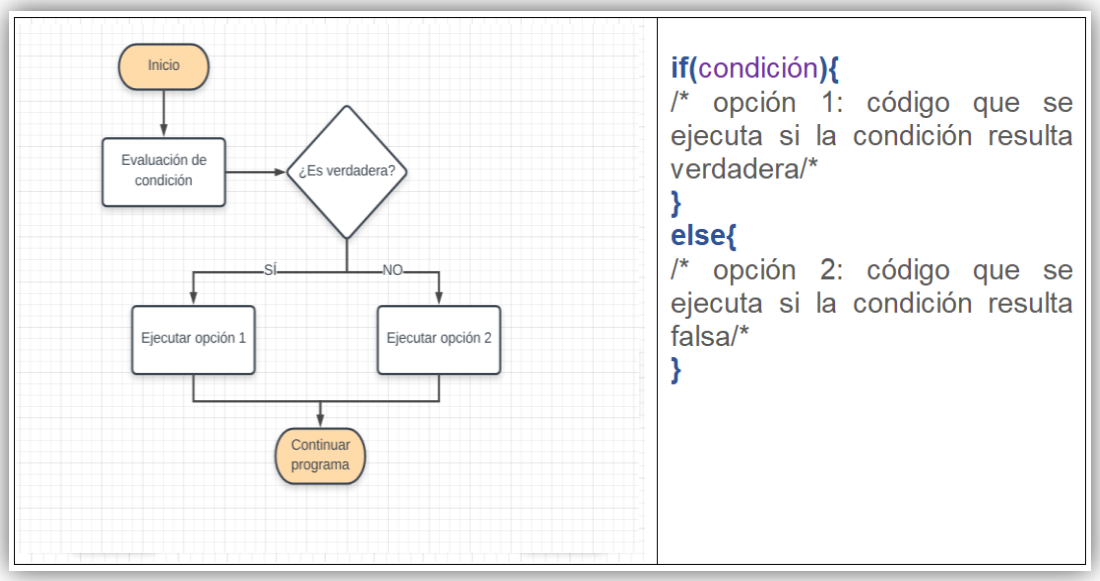
En el mundo de la programación se entiende a estas como figuras que comparan variables con alguna referencia (otra variable o una constante), para

determinar el curso del programa según el resultado. Por su importancia y simplicidad, se considerarán en esta sección la estructura *if-else* y el ciclo *while*.

El nombre de la estructura *if-else* se traduce al español como “si por lo contrario” esto da una idea de que su funcionamiento se basa en evaluar una condición booleana (con resultado verdadero o falso), y según el resultado de esta ejecutar un segmento de código.

Sin este tipo de condicionamiento, sería imposible diseñar código que se ejecute de forma versátil; se limitaría a una sola respuesta. Por esta razón todos los lenguajes de programación y de descripción de hardware ofrecen alguna variante de *if-else* en su propia sintaxis. Lo explicado en los últimos dos párrafos se observa con mayor claridad en la figura 159.

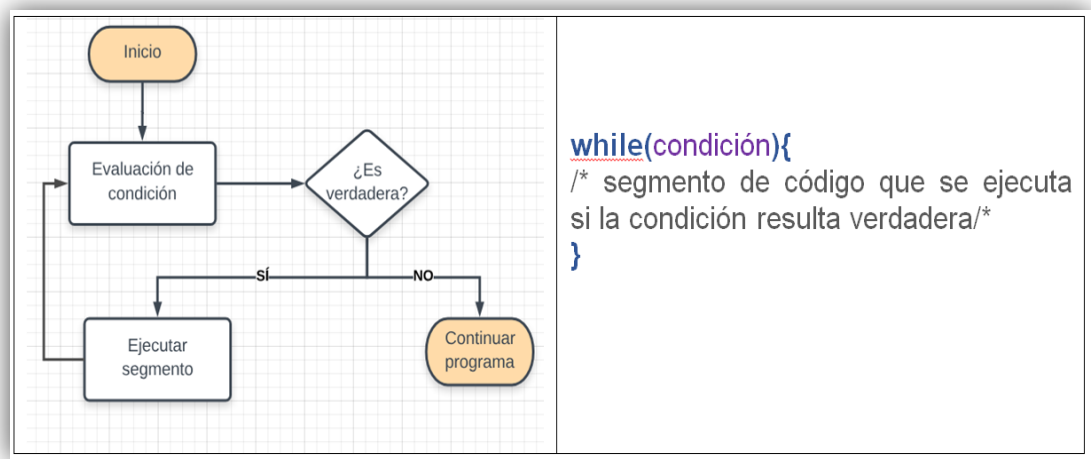
Figura 159. **Diagrama de flujo estructura *if-else* y sintaxis en lenguaje C**



Fuente: elaboración propia.

Aunque *if-else* soluciona la mayoría de los problemas de condicionamiento, debe considerarse que se ejecuta una vez por cada una que se ejecuta el código; sin embargo, existen situaciones en que se requiere que un segmento de código se ejecute ininterrumpidamente mientras cierta condición lo indique. Para esto se utilizan los ciclos *while,* que con su traducción al español mientras indica que siempre que la condición evaluada resulte verdadera, el código dentro del ciclo se ejecutará; en caso contrario, se saltará el segmento para continuar el programa.

Figura 160. **Diagrama de flujo de ciclo *while* y sintaxis en lenguaje C**



Fuente: elaboración propia.

Por generalidad las estructuras equivalentes a las descritas anteriormente son conocidas en lenguaje de ensamblador como subrutinas, denotando que son fragmentos de código que realizan una acción dentro del programa completo, contribuyendo al dinamismo.

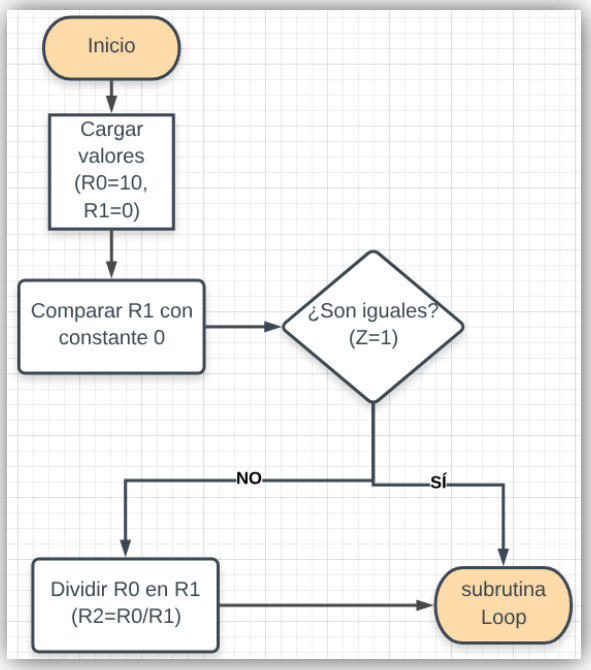
Aunque las subrutinas pueden presentarse de muchas formas y llegar al mismo resultado utilizando distintas técnicas, el funcionamiento básico se

mantiene relativamente constante para objetivos estándar como el de simular estructuras *if-else* o *while,* como se observa en los ejercicios a continuación.

6.5.2. Ejercicio 4

Con motivo de explorar la estructura *if-else* en lenguaje de ensamblador, considerar la figura 161 en que se plasma el diagrama de flujo de un programa que verifica el contenido de dos registros, antes de efectuar una división para evitar divisiones sobre cero.

Figura 161. **Diagrama de flujo para evaluación de denominador**



Fuente: elaboración propia.

Siguiendo el flujo se observa que el programa cargará dos constantes a registros (R0 y R1), luego revisará si el denominador es igual a cero (la bandera

cero será igual a 1 si la comparación es cierta), y según el resultado podrá tomar uno de dos caminos:

- Si el denominador no es igual a cero (Z=0), la división se lleva a cabo y el cociente se almacena en R2.
- Si el denominador es igual a cero (Z=1), el programa salta a la subrutina Loop y la división no se ejecuta.

Figura 162. **Programa 4**

```
    AREA codigo, CODE, READONLY, ALIGN=2
    THUMB
    EXPORT Start

Start
    ; Cargar valores.
    LDR R0, =10
    LDR R1, =0

    ; Comparar el contenido de R1 con 0. Si son iguales entonces Z=1.
    CMP R1, #0

    BEQ Loop ; Saltar a Loop si R1=0 (no es posible dividir).

    SDIV R2, R0, R1 ; Sólo se llega a esta línea si R1!=0, se divide.

Loop
    B Loop ; Ciclo sin salida.

    ALIGN
    END
```

Fuente: elaboración propia.

La rutina Loop al final simplemente cumple la función de un ciclo sin salida y sin fin como escapatoria luego de ejecutar todo el programa.

Es fundamental comprender y observar que la comparación se hace añadiendo el sufijo EQ (igual), a la instrucción B (saltar), por eso el mnemónico

BEQ se leerá como saltar si a es igual a b y que compone una línea de código que sólo se ejecutará si la bandera correspondiente se encuentra levantada. Este es un punto en que se encuentra la individualidad del programador, porque esta acción puede escribirse de varias formas, como se observa en la figura 163 sustituyendo el uso de CMP y BEQ con una sola instrucción CBZ (comparar y saltar si es igual a cero).

Figura 163. **Versión alternativa de programa 4**

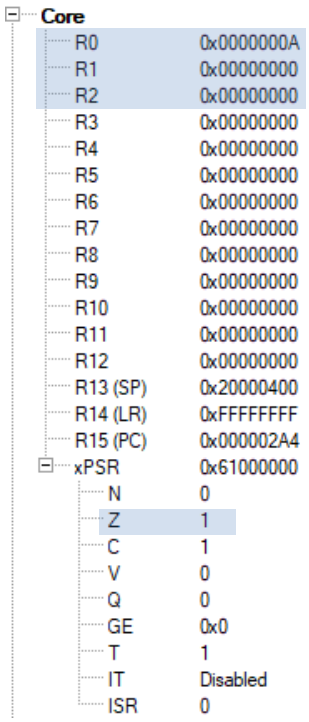
```
    AREA codigo, CODE, READONLY, ALIGN=2
    THUMB
    EXPORT Start

Start
    ; Cargar valores.
    LDR R0, =10
    LDR R1, =0

    ; Comparar el contenido de R1 con 0. Si son iguales entonces Z=1.
    CBZ R1, Loop ; Saltar a Loop si R1=0 (no es posible dividir).

    SDIV R2, R0, R1 ; Sólo se llega a esta línea si R1!=0, se divide.

Loop
    B Loop ; Ciclo sin salida.

    ALIGN
    END
```

Fuente: elaboración propia.

Si bien, CBZ se limita solamente a situaciones en que interese comparar un registro con cero, resulta evidente que en los casos en que sea viable su uso esto ahorra por lo menos una línea de código (situación que resulta ventajosa en el caso de que se desee reducir el programa).

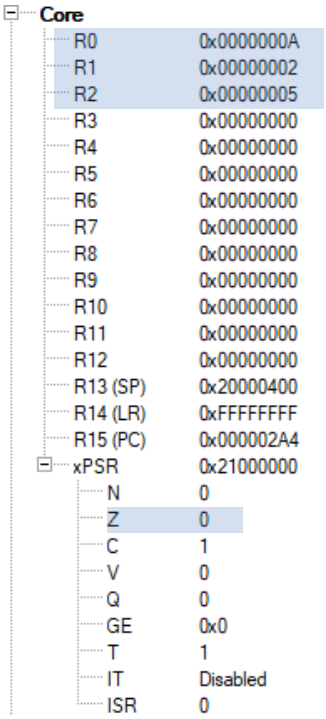
En las figuras 164 y 165 se encuentra, de forma correspondiente, el comportamiento del programa en vista de los registros que modifica. Para el caso de R0=10 y R1=0, la condición resulta verdadera (Z=1) y la división no se ejecuta (R2=0); para el caso de R=10 y R1=2, la condición resulta falsa (Z=0) y la división se ejecuta (R2=5).

Figura 164. **Vista de registros y banderas programa 4, condición verdadera**



Fuente: elaboración propia.

Figura 165. **Vista de registros y banderas programa 4, condición falsa**

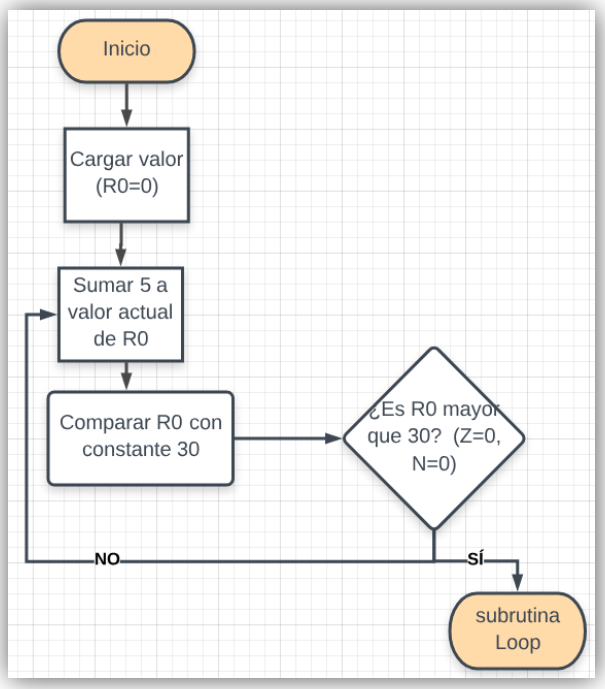


Fuente: elaboración propia.

6.5.3. Ejercicio 5

Al igual que con el ejercicio anterior, para explorar el ciclo *while* en lenguaje de ensamblador, considerar la figura 166, donde se presenta el diagrama de flujo de un programa que incremento el valor de un registro de cinco en cinco unidades siempre que el mismo no sobrepase el 30. En el momento en que el incremento llega a su límite, el programa sale del ciclo y recurre a una subrutina Loop.

Figura 166. **Diagrama de flujo para contador creciente**



Fuente: elaboración propia.

Antes de continuar con el programa 5, es importante recordar que existen muchos más sufijos, además, de EQ que ayudan a dirigir el funcionamiento de un programa. Usando la teoría desarrollada en el capítulo 5 se deduce que los sufijos actuarán según el estado de ciertas banderas, que las instrucciones de comparación modifican según tres situaciones básicas (igual, mayor que, menor que), o alguna variación de ellas (mayor o igual, menor o igual, no igual, entre otros). Estas evaluaciones se hacen simplemente restando los valores que interesa comparar, pueden hacerse ejemplos para verificar como comparar 10 con 10 (10-10=0) resultando en una bandera cero levantada para indicar que 10 es igual a 10 o 3 con 5 (3-5=-2), resultando en una bandera negativo levantada para indicar que 3 es menor que 5.

Por esta razón, siempre que se recurra al uso de sufijos no debe perderse de vista las especificaciones de la tabla XXXI. Para este caso se ha decidido utilizar el sufijo GT (mayor con signo), que indicará un resultado verdadero cuando las banderas negativo y desbordamiento sean iguales y la bandera cero esté baja (Z=0, N=V).

Figura 167. **Programa 5**

```
    AREA codigo, CODE, READONLY, ALIGN=2
    THUMB
    EXPORT Start

Start
    LDR R0, =0

Sumar
    ADD R0, #5 ; Sumar 5 al valor anterior de R0.

    CMP R0, #30 ; Comparar R0 con constante 30.
    BGT Loop ; Si R0>30 salir del ciclo.
    B Sumar ; Se llegará a esta línea sólo si R0<=30

Loop
    B Loop ; Ciclo sin salida

    ALIGN
    END
```

Fuente: elaboración propia.

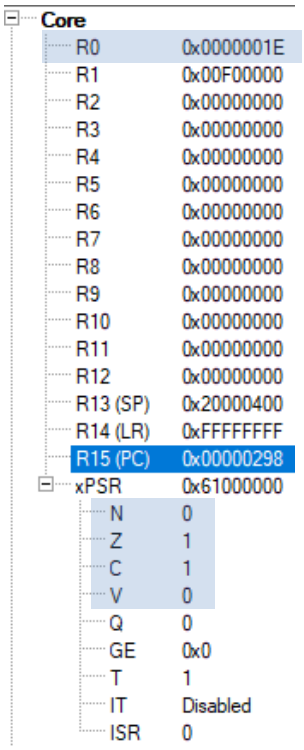
Para evaluar el programa debe considerarse que la etiqueta Sumar marca el inicio de la subrutina que incrementa valores en el registro de interés. Cada vez que el programa regresa a Sumar, suma 5 al valor anterior de R0, compara R0 con la constante 30, evalúa las banderas y decide:

- Si R0 es mayor que 30 (Z=0 y C=V), se sale del ciclo y se ejecuta Loop.

- Si R0 es menor o igual que 30, el programa no ejecuta la línea BGT Loop y pasa a la siguiente, que es un salto de vuelta al inicio de Sumar.

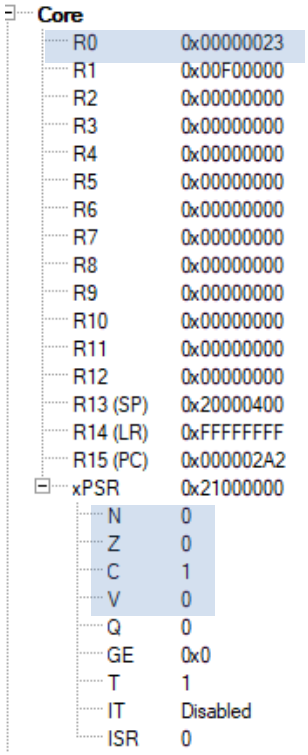
Con el fin de observar cómo se comportan las banderas en la comparación antes y después de que la suma alcance su límite, se presentan la figura 168 (R0=30) y la figura 169 (R0=35) para vista de registros y banderas.

Figura 168. **Vista de registros y banderas programa 5, condición falsa**



Fuente: elaboración propia.

Figura 169. **Vista de registros y banderas programa 5, condición verdadera**



Fuente: elaboración propia.

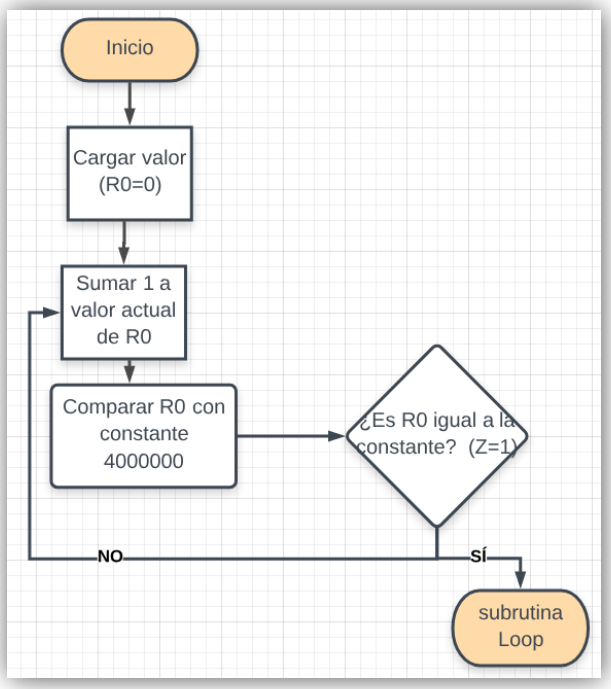
6.5.4. Ejercicio 6

Una aplicación simple y muy útil de las instrucciones de condición es la construcción de retardos (*delays*). Estos son utilizados usualmente para dar tiempo a un proceso de terminarse antes de ejecutar otra instrucción, ejemplo de esto son los *blinking* led, los retardos para evitar el rebote producido por botones, alarmas, entre otros.

Los retardos en su forma más óptima se realizan utilizando un periférico temporizador para evitar ocupar el CPU durante varios ciclos con una tarea fácil. Como ejercicio se propone una versión simple de un retardo de un segundo utilizando solamente subrutinas.

Si el programador maneja con seguridad el uso de estructuras condicionales para este punto, le resultará evidente que un retardo es simplemente un contador, cuyo límite sea igual a la cantidad de instrucciones que tengan que ejecutarse para ocupar al procesador una cantidad de ciclos igual al tiempo deseado. Es decir que tomando en cuenta que se utiliza el reloj interno del TM4C123GH6PM (con frecuencia de 16 MHz), haría falta ejecutar 16 millones de instrucciones para ocupar al sistema durante un segundo. Si se realizara una subrutina *while* que contenga, entonces, cuatro instrucciones y por cada vez que se ejecuta la misma se incrementa el contador en uno; haría falta contar hasta 4 millones para obtener 16 millones de instrucciones ejecutadas. Esto se visualiza en el diagrama de la figura 170.

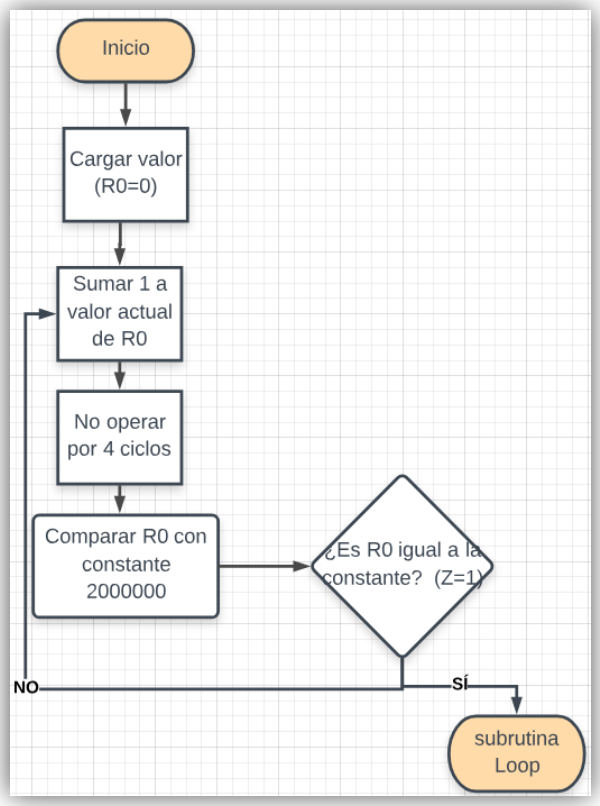
Figura 170. **Diagrama de flujo para rutina de retardo**



Fuente: elaboración propia.

Una alternativa a la lógica anterior se da considerando la instrucción NOP, que por su traducción literal indica que el procesador no operará por un ciclo de reloj (puede interpretarse como hacer nada durante un ciclo). Esto es útil para situaciones en que se desee aguardar por algunos ciclos mientras algún proceso se ejecuta, como en los retardos. En el caso específico del ciclo *while* propuesto, se usarán cuatro NOP seguidos para aumentar la cantidad de instrucciones dentro de la subrutina Sumar al doble del caso anterior, y así reducir a la mitad la cantidad de veces que tiene que repetirse el ciclo para que el retardo dure un segundo.

Figura 171. **Diagrama de flujo para rutina de retardo con instrucción NOP**



Fuente: elaboración propia.

El diagrama de flujo de la figura 171 sirve para mostrar el funcionamiento del programa 6:

- Se cargan los valores a usar en registros.
- Se entra a la rutina Delay que incrementará el valor de R0 en uno cada vez que se ingrese al ciclo.
- Cuatro ciclos sin operar.
- Se compara el valor de R0 actual con 2 millones.
- Con la instrucción compuesta BNE (saltar si a no es igual a b), se regresa al inicio de Delay si el contador no ha alcanzado su máximo o

evita el salto si la comparación es verdadera avanzando al ciclo sin fin Loop.

Por supuesto, los valores para retardo en este caso pueden modificarse para ajustarse a distintos tiempos, de igual forma en que la lógica puede ser escrita según convenga al programador.

Figura 172. **Programa 6**

```
    AREA codigo, CODE, READONLY, ALIGN=2
    THUMB
    EXPORT Start

Start
    LDR R0, =0
    LDR R1, =2000000

Delay
    ADD R0, #1 ; Sumar 1 al valor anterior de R0.
    NOP
    NOP
    NOP
    NOP

    CMP R0, R1; Comparar R0 con constante.
    BNE Delay ; Si R0!=2000000 repetir subrutina Delay.

Loop
    B Loop ; Ciclo sin salida

    ALIGN
    END
```

Fuente: elaboración propia.

6.6. Macros

Para el diseño de código, es ampliamente reconocido que existen herramientas que ayudan a facilitar el trabajo de los programadores con el fin de que este no se extienda más de lo necesario y puedan enfocar su energía en

tareas importantes. Las macro son parte de estas tácticas que, como el resto, pueden ser utilizadas según la conveniencia.

Tabla XL. **Definición de macro**

Definición que permite construir funciones que se especifican una sola vez con parámetros genéricos para luego ser llamadas por su nombre y los parámetros específicos que se desea operar con el fin de ahorrar la escritura repetida de segmentos similares.

Fuente: HOHL, William; HINDS, Christopher. *ARM Assembly Language Fundamentals and Techniques*. p. 73.

Desde el punto de vista del diseñador de código las macros son una alternativa muy buena para reducir la cantidad de líneas que abarca el programa y el tiempo que requiere escribirlas. El ensamblador toma las macros como una repetición de segmento, y la reducción sólo beneficia al humano y no al procesador. Este contraste evidencia que su uso será una decisión dependiente de los recursos disponibles y los objetivos del sistema.

La sintaxis de Keil uVision para una macro y cómo esta se llama en el código se representa en la figura 173.

Figura 173. **Sintaxis de una macro en Keil uVision**

```
        MACRO
$etiqueta      nombre_macro      $parámetro, $parámetro, …
$etiqueta
        ;código
        MEND

nombre_etiqueta     nombre_macro      parámetro, parámetro, …
```

Fuente: elaboración propia.

MACRO y MEND son directivas que marcan el inicio y fin de la definición mientras $etiqueta y $parámetro son espacios genéricos, en la definición que son reemplazados por valores específicos cada vez que la macro se llama en el código.

6.6.1. Ejercicio 7

Para ejemplificar el uso de macros, considerar el programa 7. El código ejecuta el promedio de cuatro números dos veces (uno almacenado en R4 y otro en R5), y en lugar de presentar una doble escritura del procedimiento recurre al uso de una macro.

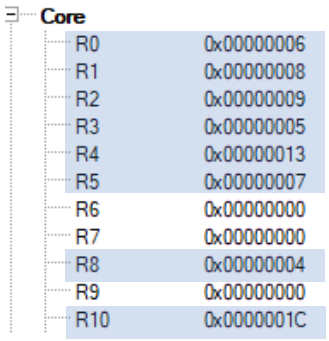
A pesar de que esto resulta ser un ejercicio muy sencillo, puede notarse la utilidad de estas estructuras para códigos en que un segmento se repite varias veces. Su contenido puede ser tan compleja o simple como se desee y operar la cantidad de parámetros necesaria para cumplirlo.

Figura 174. **Programa 7**

```
    AREA codigo, CODE, READONLY, ALIGN=2
    THUMB
    EXPORT Start

Start
    LDR R8, =4

    MACRO
; prom = (a + b + c + d)/4
$etiqueta Promedio $prom, $a, $b, $c, $d
$etiqueta
    ADD R10 , $a, $b
    ADD R10 , $c
    ADD R10 , $d
    SDIV $prom, R10, R8 ; dividir en 4
    MEND


    LDR R0, =10
    LDR R1, =20
    LDR R2, =15
    LDR R3, =31
_Prom1 Promedio R4, R0, R1, R2, R3
    LDR R0, =6
    LDR R1, =8
    LDR R2, =9
    LDR R3, =5
_Prom2 Promedio R5, R0, R1, R2, R3

Loop
    B Loop

    ALIGN
    END
```

Fuente: elaboración propia.

Figura 175. **Vista de registros y banderas programa 7**



Fuente: elaboración propia.

6.7. Punto flotante

La unidad de punto flotante es una característica opcional en la arquitectura Arm. De hecho, es una de las principales diferencias entre Cortex-M3 y Cortex-M4 (junto con las instrucciones DSP).

El set de instrucciones de las arquitecturas que contienen FPU se extiende más allá del básico para especificar mnemónicos correspondientes a este tipo de operaciones. Por lo general se ven complementados por un prefijo V y un sufijo .F32.

Aunque la mayoría de instrucciones básicas son similares a las estudiadas en las secciones anteriores, existen dos consideraciones a añadir para comprender adecuadamente el uso de este grupo:

6.7.1. El FPSCR

Como se definió en la Sección 2.3.2, las arquitecturas Arm tienen registros de estado que contienen especificaciones de cómo se ejecuta en programa (entre ellas, las banderas).

Existen varios registros de estado que se activan o dejan en reserva según el estado en que se encuentra el procesador. Para el caso del CPU en su modo principal, el APSR (registro de estado de programa de aplicación), contiene la información que sirve, por ejemplo para indicar a una instrucción compuesta BEQ que la bandera cero está levantada o no.

Cuando una instrucción de punto flotante se ejecuta (e.g. VCMP.F32), se modifica también el estado de banderas pero en este caso no será en el APSR, sino en el FPSCR (registro de estado y control de punto flotante). Cuando esto sirva para flujo condicional de programas, los sufijos no evaluarán las banderas de otro registro de estado más que del APSR; que podría resultar en un problema si es necesario operar en punto flotante hasta que se considera las instrucciones VMRS y VMSR.

Las instrucciones para movilización de banderas entre APSR y FPSCR sirven para hacer una copia de las banderas N, Z, C y V de un registro de estado a otro. Con esto es posible entonces realizar operaciones de punto flotante y utilizar condicionales que evalúen el APSR en el mismo programa. Solamente debe tenerse el cuidado de realizar la copia de banderas antes de evaluar para leer adecuadamente.

6.7.2. Conversiones

En algunas ocasiones resulta conveniente (e incluso necesario) mezclar el uso de registros de propósito general y de punto flotante en un programa. No debe en estas situaciones que la escritura y manejo de registros de punto flotante no es la misma que la de los registros de propósito general. Es decir, por ejemplo, que un hexadecimal 0x0FFF será equivalente a 4 095 en formato convencional y a $5{,}74X10^{-42}$ en formato IEEE 754.

Para el caso específico en que deba trasladarse el valor de un registro de un tipo a otro deberá considerarse entonces si lo que interesa es mantener congruencia del contenido con su valor equivalente, independientemente del formato o simplemente hacer una copia de él quizás para utilizarlo como una dirección de memoria. Con la idea de alcanzar el segundo objetivo bastará utilizar la instrucción VMOV.F32 y con la de alcanzar el primero puede utilizarse VCVT.S32.F32 o VCVTR.S32.F32 con la sintaxis que se observa en la tabla XLI.

Tabla XLI. **Instrucciones básicas UAL para VFP Cortex-M4**

Mnemónico	Instrucción	Estructura
VABS.F32	Absoluto	Sd, Sm
VADD.F32	Suma	{Sd, } Sn, Sm
VCMP.F32	Comparar	Sd, Sm Sd, #0.0
VCVT.S32.F32	Convertir de formato punto flotante a entero	Sd, Sm
VCVTR.S32.F32	Convertir de punto flotante a punto fijo con redondeo	Sd, Sm
VDIV.F32	División	{Sd, } Sn, Sm
VLDR.F32	Cargar registro	Sd, =imm32 Sd, [Rn]
VMOV.F32	Mover simple o múltiple	Sd, Sm Sd, Rt Sm, Sm1, Rt, Rt2

Continuación de la tabla XLI.

VMRS	Mover de FPSCR a registro o APSR	Rt, FPSCR APSR_nzcv, FPSCR
VMSR	Mover de registro a FPSCR	FPSCR, Rt
VMUL.F32	Multiplicación	{Sd, } Sn, Sm
VNEG.F32	Negación	Sd, Sm
VPOP	Obtener registro de pila	lista
VPUSH	Almacenar registro en pila	lista
VSQRT.F32	Raíz cuadrada	Sd, Sm
VSTR.F32	Almacenar en memoria	Sd, [Rn]
VSUB.F32	Sustracción	{Sd, } Sn, Sm

Fuente: elaboración propia.

Luego de leer apropiadamente las instrucciones se observa que en punto flotante:

- Es necesario para las instrucciones VADD.F32, VCMP.F32, VMUL.F32 y VSUB.F32 contar con dos registros para operar, a diferencia de las instrucciones básicas que soportaban otros formatos.
- Una añadidura importante a esta sección del set es VSQRT.F32 abriendo la posibilidad de calcular la raíz cuadrada de un número contenido en un registro de punto flotante.

6.7.3. Ejercicio 8

Una gran cantidad de tareas pueden llevarse a cabo utilizando las instrucciones estudiadas en las secciones anteriores. Existen ocasiones en que por eficiencia y exactitud son requeridas las instrucciones correspondientes al FPU. Un ejemplo de estos casos es el cálculo del factorial de un número mayor a cero, que ilustrará el uso de punto flotante.

Para comprender el Programa 8, es necesario recordar la definición de un factorial:

Tabla XLII. **Definición de factorial**

La función factorial de un entero positivo (*n*) se define como el producto de todos los números enteros positivos desde 1 (los números naturales) hasta *n*, ambos incluidos. La función se representa con un signo de exclamación "!" a la derecha del número que interesa (*n*!) y suele leerse como "n factorial".

Fuente: *Factorial o función factorial*. http://www.profesorenlinea.cl/matematica/Factorial.html. Consulta: 17 de noviembre de 2018.

Sabiendo la definición puede poco a poco construirse la idea de que para calcular un factorial bastaría realizar una estructura que tome el valor de *n*, disminuya su valor en uno cada ciclo y acumule el producto de cada vuelta hasta que la disminución llegue a la unidad. Para este fin se presenta el programa 8:

Figura 176. **Programa 8**

```
    AREA codigo, CODE, READONLY, ALIGN=2
    THUMB
    EXPORT Start

Start
    VLDR.F32 S0, =30     ; Número a operar (n!).
    VLDR.F32 S1, =1      ; Registro para guardar resultado.
    VLDR.F32 S30, =1     ; Constante 1.

Factorial
    ; Multiplicar el valor actual de n y el resultado acumulado.
    VMUL.F32 S1, S0
    ; Sustraer uno al valor actual de n.
    VSUB.F32 S0, S30
    ; Verificar si se ha llegado a 1 (ya no quedan números por multiplicar).
    VCMP.F32 S0, S30
    VMRS APSR_nzcv, FPSCR   ; Trasladar banderas de FPSCR a APSR.
    BNE Factorial   ; Si n no es igual a 1, seguir operando.

Loop
    B Loop

    ALIGN
    END
```

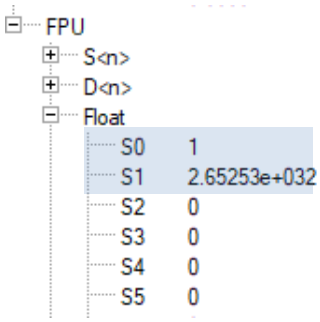
Fuente: elaboración propia.

Con el fin de comprender el funcionamiento del programa 8, se desglosan los pasos que este sigue:

- Luego del archivo Startup, el programa inicia en la etiqueta Start.
- Se cargan los valores iniciales: S0 es el número del que se desea obtener factorial (*n*), S1 es el registro que contendrá el resultado del factorial (se iguala a 1 para evitar que el producto se vuelva nulo) y S30 es una constante que servirá para el decremento de *n* y la comparación.
- Se ingresa a la subrutina Factorial.
- El producto de todos los números se acumula ciclo por ciclo en S1.
- Se disminuye *n* en uno.

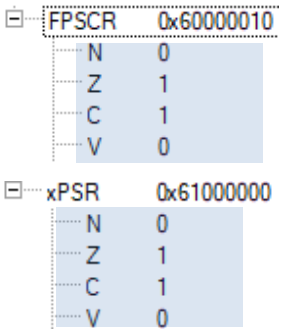
- Se compara si la disminución ha alcanzado su mínimo.
- Para poder utilizar instrucciones condicionales es necesario trasladar las banderas de punto flotante al APSR (Sección 6.7.1).
- Si la disminución no ha llegado al mínimo significa que aún quedan números por multiplicar y el programa reingresa a la subrutina Factorial. Si sí ha llegado (S0=S30), se avanza al ciclo sin fin Loop.

Figura 177. **Vista final de registros programa 8**



Fuente: elaboración propia.

Figura 178. **Vista de FPSCR y APSR de programa 8**



Fuente: elaboración propia.

Si se estudia con cuidado el programa 8 se notará que está excluido el caso del factorial de 0 (dado que en matemática 0!=1 pero en el programa esto no se contempla). La condición se cubre agregando un condicional que iguale el factorial a 1 si n es igual a cero o ejecute la subrutina Factorial si es mayor; según alguna lógica similar a la del programa 4, aunque puede no resultar necesario si *n* nunca toma valores menores a 1.

6.7.4. Uso de series matemáticas

A través de las secciones anteriores ha sido notorio que, para lenguaje de ensamblador, existen situaciones que requieren la construcción de código complejo comparado con lo disponible en el set de instrucciones *per se*.

El lenguaje de ensamblador tiene por única limitante los recursos físicos disponibles para operar, y conseguir funciones avanzadas dependerá del ingenio y experiencia del programador para trabajar la ISA.

Existen, sin embargo, herramientas dentro del razonamiento matemático que facilitan muchos procesos (situación común en todos los lenguajes de programación). Este es el caso de las series matemáticas:

Tabla XLIII. **Definición de serie matemática**

Una serie *S* es la sumatoria de una sucesión de elementos. Se comprende como la acumulación de *n* términos para evaluar cómo se comporta el conjunto según *n* tiende a un límite *L* que puede ser finito o infinito. En algunas series el resultado converge en algún valor, esto resulta útil para aproximar ciertas funciones. La notación más simple para una serie que comienza en 0 se escribe: $$S = \sum_{n=0}^{L} a_n$$ Donde *a* puede tomar la forma de cualquier expresión.

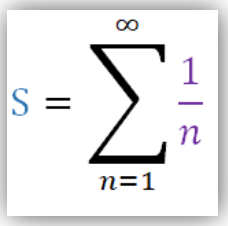
Fuente: elaboración propia.

Las series matemáticas toman muchas formas y conllevan todo un análisis fuera del campo de este texto. Interesa a esta sección es que muchas funciones disponibles en lenguaje de alto nivel pueden ser representadas en forma de series, y sirve al programador para solucionar el problema de no tenerlas en el set de instrucciones para su uso en lenguaje de ensamblador y al mismo tiempo, ayudan como una práctica para desarrollar la capacidad de escribir código a bajo nivel. Notar que algunas de estas expresiones sólo podrían calcularse correctamente en arquitecturas con FPU disponible.

6.7.5. Ejercicio 9

El primer ejercicio a trabajar en este tema es una de las formas más simples de series numéricas: la serie armónica. Esta sumatoria simplemente suma infinita cantidad de elementos a partir del momento en que *n* es uno, esto se escribe de la forma:

Figura 179. **Serie armónica**



Fuente: elaboración propia.

Por claridad debe saberse que el inicio de la serie en uno (y no en cero), se debe a que la división en cero está indefinida y que para el caso de este ejercicio el límite superior de la suma no puede ser infinito, porque requeriría tiempo infinito operar y simplemente se escoge un número relativamente grande

para sustituirlo (cosa que funciona en los casos en que las series convergen y sólo importan algunos decimales de la respuesta).

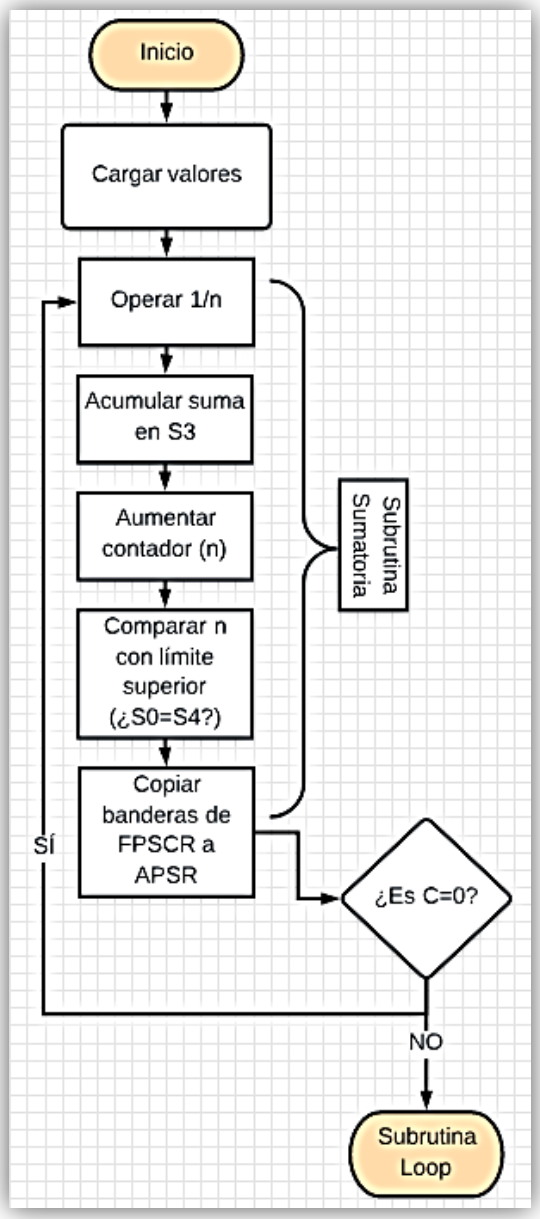
El programa derivado de la figura 179 se observa a continuación, con su diagrama de flujo para ayudar a la comprensión del mismo en la figura 181. Observar que se utiliza una variante de las instrucciones de saltos (BLO), con un sufijo que indica que se debe evaluar si el número de interés es menor que un segundo, esto se encuentra detallado en la tabla XXXI.

Figura 180. **Programa 9**

```
    AREA    codigo, CODE, READONLY,ALIGN=2
    THUMB
    EXPORT Start

Start
    VLDR.F32 S0, =1      ; Contador (n).
    VLDR.F32 S1, =1      ; Constante 1.
    VLDR.F32 S4, =50     ; Límite superior.

Sumatoria   ; Ciclo de sumatoria.
    VDIV.F32 S2, S1, S0 ; S2=1/n
    VADD.F32 S3, S2      ; S3 almacena la sumatoria.
    VADD.F32 S0, S1      ; Incrementar n.
    ; Verificar si n llegó a límite superior.
    VCMP.F32 S0, S4
    VMRS APSR_nzcv, FPSCR
    BLO Sumatoria        ; Saltar si es menor (C=0).

Loop        ; Ciclo sin fin.
    B Loop

    ALIGN
    END
```

Fuente: elaboración propia.

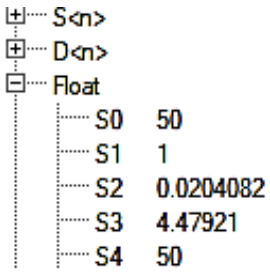
Figura 181. **Diagrama de flujo programa 9**



Fuente: elaboración propia.

El efecto del programa en los registros de punto flotante se observa en la figura 182 con el contador en S0, la constante uno en S1, el valor de cada término en S2, la sumatoria en S3 y el límite superior de la misma en S4.

Figura 182. **Vista final de registros programa 9**



Fuente: elaboración propia.

6.7.6. Ejercicio 10

Después de comprender apropiadamente cómo una serie matemática puede ser simulada en lenguaje de ensamblador puede comenzar a trabajarse con series un poco más elaboradas y con un uso más específico, como es el logaritmo natural de un número x (ln x). Esta función tiene un equivalente de rápida convergencia para valores mayores que cero, la igualdad se presenta en la figura 183.

Figura 183. **Logaritmo natural**

$$\ln x = \sum_{n=0}^{\infty} \frac{1}{2n+1}\left(\frac{x^2-1}{x^2+1}\right)^{2n+1} \quad \text{para } x > 0$$

Fuente: elaboración propia.

El logaritmo natural, junto a otras funciones como las trigonométricas, forma parte de las herramientas necesarias para el funcionamiento de ciertos sistemas (por ejemplo, filtros). El razonamiento en este ejercicio sirve como

formato para la elaboración de programas similares o incluso, mucho más complejos; y no pretende ser una guía estricta, sino el ejemplo de una forma de abordar el tema.

El código que corresponde a la aproximación de la figura 184 se muestra en el programa 10.

Figura 184. **Programa 10**

```
    AREA codigo, CODE, READONLY, ALIGN=2
    THUMB
    EXPORT Start

Start       ; Punto de inicio de programa.
    VLDR.F32 S0, =8      ; Valor de x.
    VLDR.F32 S1, =50     ; Límite superior.

    VLDR.F32 S2, =1      ; Constante 1.
    VLDR.F32 S3, =2      ; Constante 2.

    VLDR.F32 S4, =-1     ; Valor de n.

Factor_K    ; Cálculo de K en S13.
    VMUL.F32 S10, S0, S0     ; x^2
    VSUB.F32 S11, S10, S2    ; (x^2)-1
    VADD.F32 S12, S10, S2    ; (x^2)+1
    VDIV.F32 S13, S11, S12   ; K=((x^2)-1)/((x^2)+1)

Sumatoria   ; Ciclo de sumatoria.
    VADD.F32 S4, S2      ; n+1
    VMUL.F32 S20, S3, S4     ; 2*n
    VADD.F32 S20, S2     ; 2n+1
    VDIV.F32 S21, S2, S20    ; a=1/(2n+1)

    VLDR.F32 S22, =0     ; Contador subrutina Potencia.
    VLDR.F32 S23, =1     ; S23 contendrá el valor de la potencia.
    BL Potencia          ; b=K^(2n+1), BL: "branch with link".
    VMUL.F32 S24, S23, S21   ;a*b
    VADD.F32 S30, S24    ; Acumulado de sumatoria en S30.
    VCMP.F32 S4, S1      ; Verificar si n ha llegado a límite superior.
    VMRS APSR_nzcv, FPSCR
    BLO Sumatoria        ; Sufijo LO significa "menor".

Loop        ; Ciclo infinito.
    B Loop

Potencia    ; Subrutina para cálculo de potencia.
    VMUL.F32 S23, S13
    VADD.F32 S22, S2
    VCMP.F32 S22, S20    ; Verificar si contador es igual a exponente.
    VMRS APSR_nzcv, FPSCR
    BNE Potencia         ; Si no se ha terminado, continuar calculando.
    BX LR    ; Volver a línea siguiente a BL usado.

    ALIGN
    END
```

Fuente: elaboración propia.

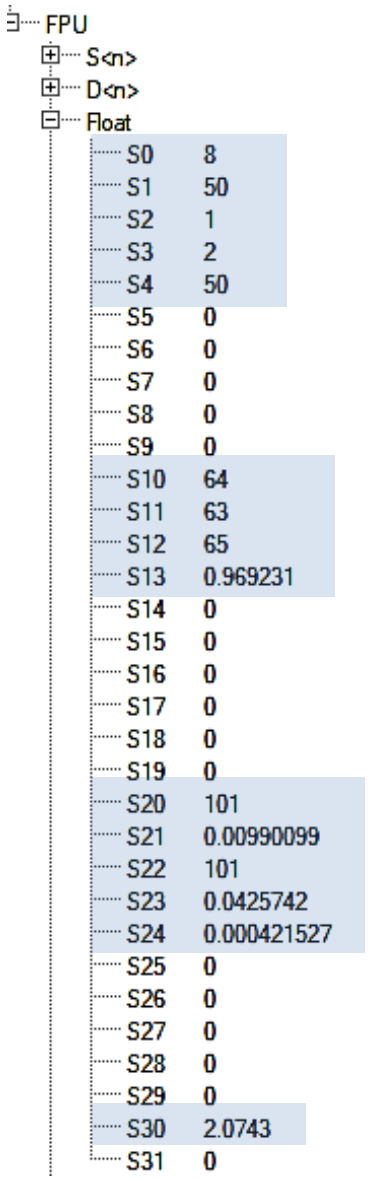
El programa utiliza varios grupos de registros de punto flotante para facilitar la comprensión del flujo:

- Luego de recorrer el archivo Startup, el programa comienza en la etiqueta Start. A partir de este punto son cargados los valores iniciales: el de x, el del límite superior y las constantes.
- La etiqueta Factor_K identifica el cálculo del factor multiplicativo en la serie que no depende de ***n***, que en la figura 183 se encuentra encerrado en paréntesis y que es constante, porque ***x*** también lo es durante toda la sumatoria. Este segmento no es un ciclo, simplemente se marca con una etiqueta para no perder de vista su función.
- La etiqueta Sumatoria identifica el ciclo más importante del programa.
- S4 se usa a partir de este punto como contador de la sumatoria (valor de n), haciendo que cada vez que se reingrese a la subrutina su valor incremente en uno con ayuda de S2.
- De forma similar al programa 9, se calcula en cada ciclo el valor de 1/(2n+1). A esto se llamará factor 'a' en los comentarios.
- Teniendo disponibles los cálculos hasta este punto y siguiendo la figura 183 puede notarse que corresponde elevar el factor K a una potencia 2n+1 que convenientemente, quedó almacenada en S20 para el cálculo de a. Una potencia es por sí misma, un procedimiento en ciclo; y por orden en el código se introduce el uso de dos instrucciones no aprovechadas hasta el momento: BL y BX (tabla XXXIX), su funcionamiento se explica apropiadamente más adelante. Simplemente para comprender el programa es necesario saber que BL dirige a una línea de código, sabiendo cuál sería la siguiente línea a ejecutar si el programa siguiera su curso normal y BX permite volver a esa línea con el uso de LR (registro de enlace).

- El programa salta a la subrutina Potencia sabiendo que la siguiente línea a ejecutar sería la multiplicación de a y b.
- Dentro de la subrutina Potencia el valor del factor K (S13), se multiplica por sí mismo hasta que su contador local (S22), alcance el valor de 2n+1. El resultado se almacena en S23 (factor b), que se reinicia cada vez que se entra al ciclo con el valor de uno para evitar nulidad del resultado por multiplicación con cero.
- Al terminar de calcular la potencia, el programa vuelve a la línea siguiente de donde saltó con el uso de BL.
- Se multiplican a y b, que corresponde al total de cada término de la sumatoria.
- Se suma el valor actual del término a todos los anteriores contenidos en S30.
- Se verifica si n llegó al valor del límite superior. Si no, se vuelve a operar el ciclo; si sí, se avanza al ciclo sin fin Loop.

El efecto del programa 10 es los registro de punto flotante es entonces:

Figura 185. **Vista de registros programa 10**



Fuente: elaboración propia.

La ca como fue mencionado anteriormente, la cantidad de registros a utilizar puede reducirse si se reutilizan algunos de ellos con valores que sólo sirvan al programa de forma temporal, o si se utiliza localidades de memoria

para almacenar en lugar de registros. Todas estas variantes dependerán de la conveniencia.

6.7.6.1. Saltos con enlace

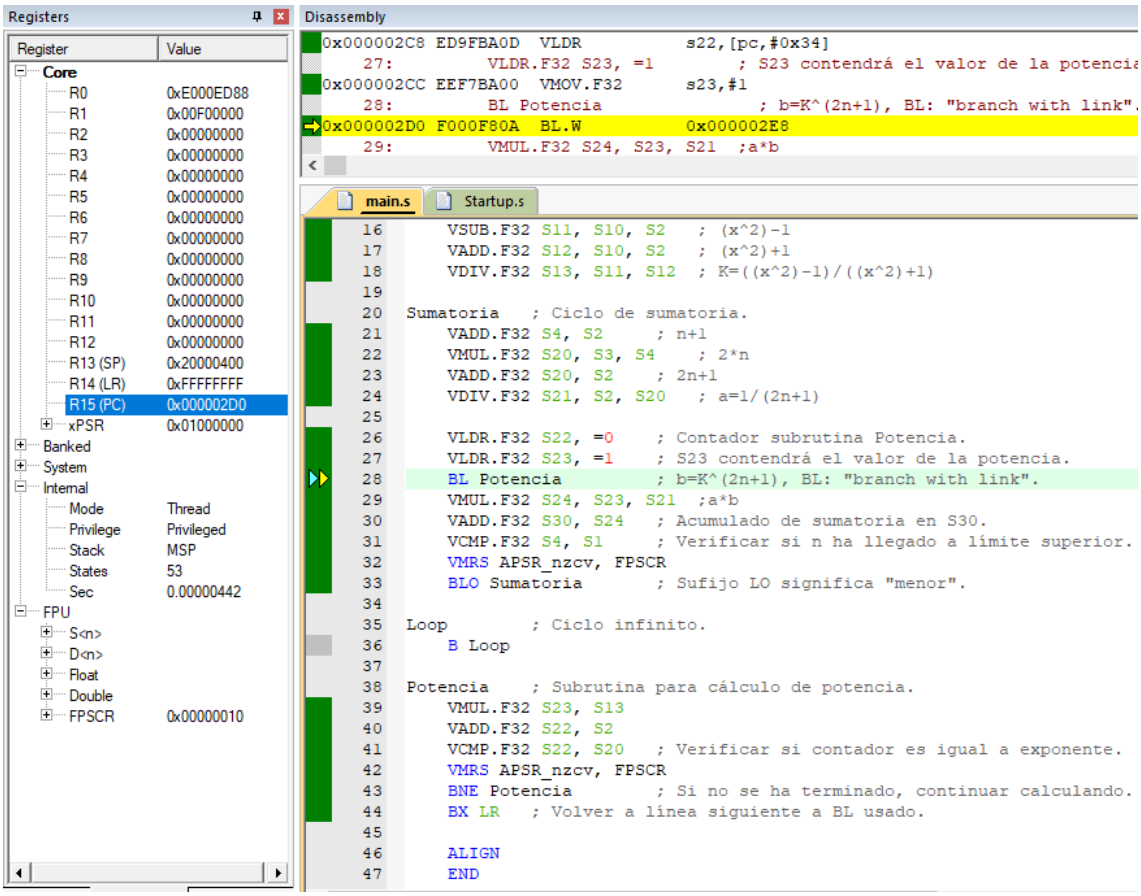
En el programa 10 fue explicado superficialmente el funcionamiento de las instrucciones BL y BX (*branch and exchange,* saltar e intercambiar). En esta subsección se ampliará haciendo uso del mismo código y la visualización de ventanas en el depurador.

Por simplicidad se considera primero la instrucción BL. Este es un mnemónico que se acompaña de una dirección (que puede ser indicada por una etiqueta), como el resto de instrucciones *branch* para indicar a dónde se dirigirá el programa.

Resulta fundamental observar el comportamiento del contador de programa (PC), que lleva el conteo de la línea de código que se ejecuta en el momento, y del registro de enlace (LR).

Para el momento en que no se ha ejecutado la instrucción BL, el LR se encuentra con valor 0xFFFFFFFF y el PC lleva el conteo natural. En el caso de estar a punto de ejecutar BL Potencia (figura 186), el PC será 0x000002D0.

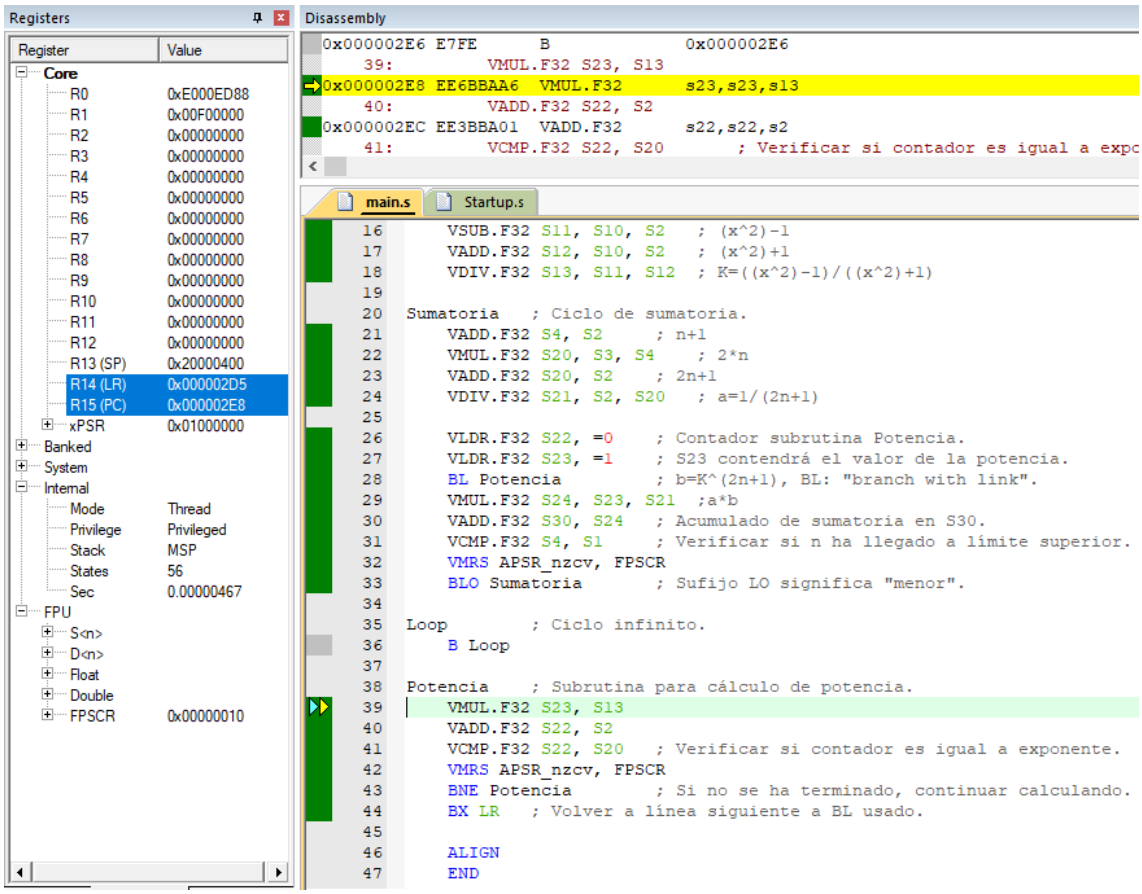
Figura 186. **Contenido de PC y LR antes de instrucción BL**



Fuente: elaboración propia.

Justo después de ejecutar el salto con enlace, el LR toma el valor del PC en el paso anterior y suma cinco bytes. Entonces LR toma para el programa 10 el valor 0x000002D5 y el PC lleva el conteo natural luego de dirigirse a la subrutina deseada.

Figura 187. **Contenido de PC y LR después de instrucción BL**

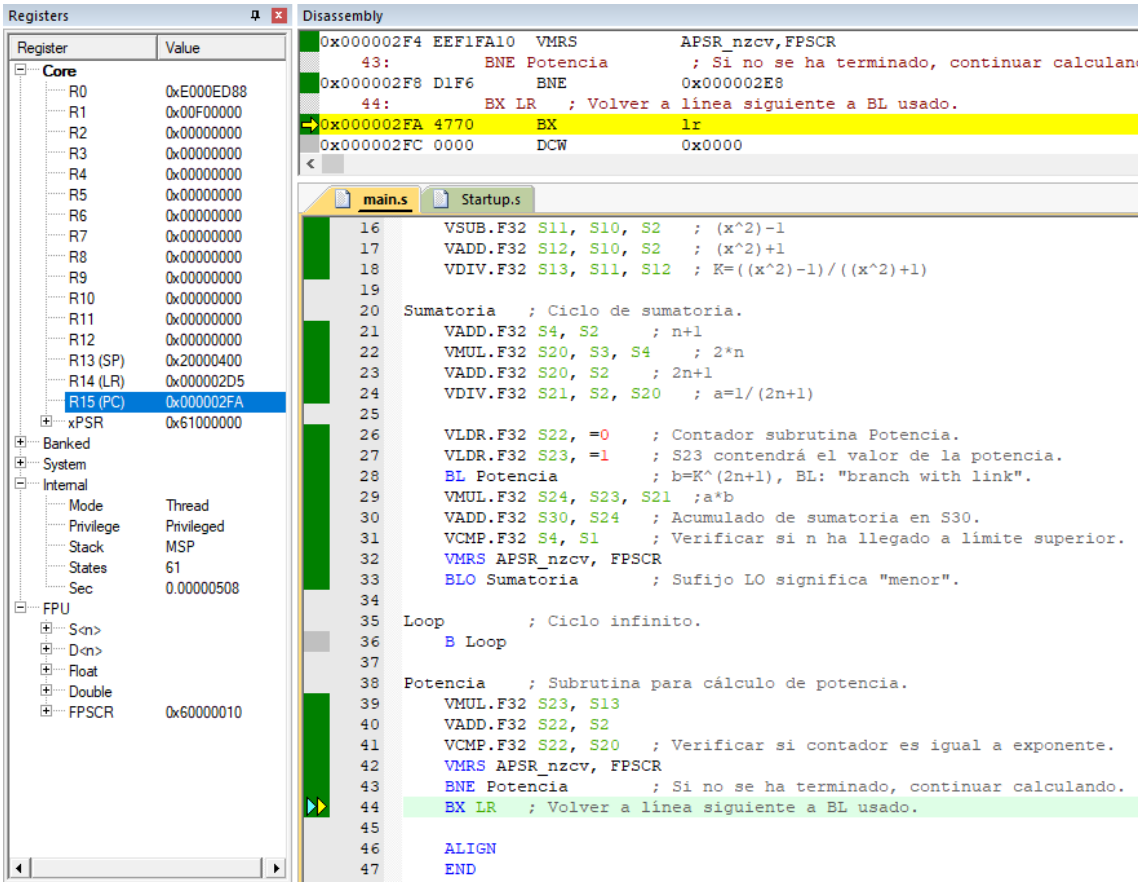


Fuente: elaboración propia.

Si se observa adecuadamente la subrutina potencia, se observa que se indica a la máquina que se reingresará al mismo ciclo siempre que el contador local (S22), no sea igual que el valor de exponente (S20), comenzando en cero por el reinicio en la línea 26 cada ciclo de Sumatoria. Cuando los registros son iguales entonces el programa pasa a la línea 44 con la instrucción BX LR.

Justo antes de que la instrucción BX se ejecute el PC tiene su valor natual de incremento en el segmento que se encuentra y LR almacena la dirección que se almacenó con BL.

Figura 188. **Contenido de PC y LR antes de instrucción BX**

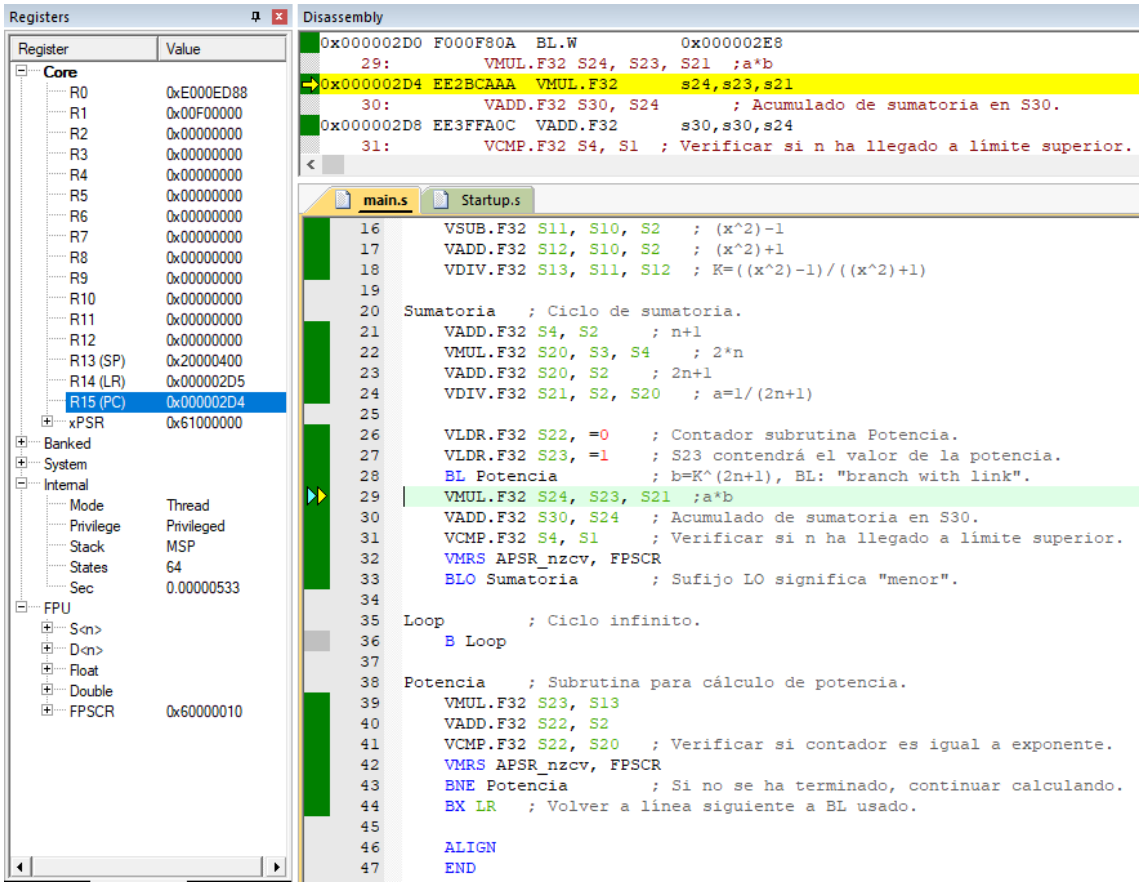


Fuente: elaboración propia.

Ejecutar BX LR quiere decir que se hará un salto hacia la dirección que indica el LR menos un byte, llenando el PC con el nuevo valor. Para el programa 10 en este momento PC tomará la dirección 0x000002D4 que para el

humano se verá igual que dirigirse a la línea 29 (inmediatamente después de la instrucción BL original).

Figura 189. **Contenido de PC y LR después de instrucción BX**



Fuente: elaboración propia.

Las precauciones en el uso de este tipo de instrucciones incluyen cuidar la sobreescritura inintencionada de LR que impidan regresar al punto de interés con BX LR.

6.8. Configuración de periféricos en tarjeta de desarrollo

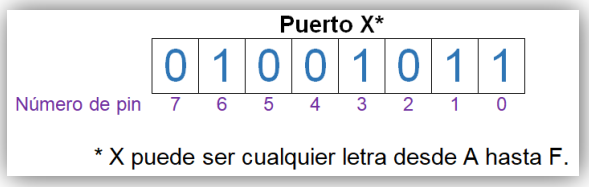
Repasando la definición de los sistemas embebidos en la sección 3.1.2 resulta evidente que estos se caracterizan por su interacción con el medio que los rodea, sea para analizarlo por medio de sensores, ofrecer una respuesta por medio de actuadores o para comunicarse con otros sistemas con el uso de módulos y protocolos. Así, hasta ahora, todos los conceptos estudiados de programación de un procesador Cortex-M4F forman la base para el siguiente gran paso: configurar los periféricos del microcontrolador.

Aunque para el estudio completo del microcontrolador se abarcaría, idealmente, todos sus periféricos, esta sección se limitará a la configuración del más simple de ellos: las entradas y salidas de propósito general (GPIO). Se procurará, dar la introducción básica necesaria en caso el programador desee comprender la configuración y uso de los otros periféricos, con ayuda de textos específicos en el área.

Los tres conceptos fundamentales para comprender todos los cálculos de direcciones y máscaras necesarias para habilitar apropiadamente los pines del microcontrolador son:

- Cada puerto (del A al F), se compone de 8 bits, que físicamente corresponde a 8 pines numerados de 7 a 0.

Figura 190. **Composición de un puerto**



Fuente: elaboración propia.

En algunas situaciones resulta conveniente utilizar la forma hexadecimal del contenido del puerto, especialmente para modificar la información en los registros de configuración o para encendido y apagado de pines. Por ejemplo, en la figura 190 se muestra que los pines de interés son 6, 3, 1 y 0 y la máscara a utilizar en el método leer-modificar-escribir es 0x4B en lugar del número binario de 8 dígitos.

- Los puertos del microcontrolador están mapeados a memoria, lo que quiere decir que tienen direcciones asignadas para almacenamiento de su información, y tienen las direcciones base de la tabla XLIV (esto se observa también en el mapa de memoria de la tabla XXXIV), y para acceder a registros específicos del puerto se suman los offset debidos.

Tabla XLIV. **Direcciones base de puertos del TM4C123GH6PM**

Puerto	Dirección base
A	0x40004000
B	0x40005000
C	0x40006000
D	0x40007000
E	0x40024000
F	0x40025000

Fuente: elaboración propia.

- Cada puerto tiene un conjunto de registros destinados a su configuración, estos pueden ser leídos y escritos para lograr cualquier habilitación posible de periféricos (idealmente con ayuda de la documentación). El resumen de los registros principales se encuentra a continuación.

Tabla XLV. **Registros para configuración de periféricos**

Nombre	Definición	Configuración	Offset
AMSEL	Selección de modo analógico, conecta los pines a ADC o comparador analógico.	Un 1 en el pin deseado lo activa.	0x528
DEN	Habilitación de modo digital.	Un 1 en el pin deseado lo activa.	0x51C
LOCK	Bloqueo de pines.	Para desbloquear se escribe la constante 0x4C4F434B en este registro*, conocida como GPIO_LOCK_KEY.	0x520
DIR	Especificación de dirección.	Un 1 en el pin deseado lo activa como salida y un 0, como entrada.	0x400
PCTL	Selección de función alternativa.	Se utiliza la especificación de la Figura 191 para elegir una de las funciones que pueden tomar los pines en el microcontrolador**.	0x52C
AFSEL	Función alternativa. Especifica si ha sido seleccionado un modo alternativo, actuando junto a la selección de PCTL.	Un 0 indica que no hay ninguna función alternativa seleccionada (modo GPIO).	0x420
DATA	Encierra las direcciones del puerto completo (8 pines).		0x3FC

Fuente: elaboración propia.

*Los únicos pines bloqueados por defecto en la tarjeta de desarrollo Tiva C son F0, C3-C0 y PD7.

** Existen varios modos en que un pin puede funcionar (I/O, UART, PWM, CAN, etc.), estos mostrados en la figura 191 con excepción de PC3-PC0 por

estar reservados para JTAG. Aunque no todos los pines pueden funcionar con todos los periféricos, por orden se asigna números del cero al 14 para activar el multiplexor de puerto e indicar al periférico que debe conectarse, y cuatro bits en este registro pueden especificar cuál desea utilizarse. Ejemplo de esto es que PB4 puede funcionar como CAN Rx, y correspondería escribir 0x01 en PCTL si se desea usar esta función o 0x00 si se desea usar como GPIO (el mismo concepto se aplica a todos los pines).

Figura 191. **Funciones de pines en microcontrolador TM4C123GH6PM**

IO	Ain	0	1	2	3	4	5	6	7	8	9	14
PA0		*Port*	*U0Rx*							CAN1Rx		
PA1		*Port*	*U0Tx*							CAN1Tx		
PA2		Port		SSI0Clk								
PA3		Port		SSI0Fss								
PA4		Port		SSI0Rx								
PA5		Port		SSI0Tx								
PA6		Port			I_2C1SCL		M1PWM2					
PA7		Port			I_2C1SDA		M1PWM3					
PB0		Port	U1Rx						T2CCP0			
PB1		Port	U1Tx						T2CCP1			
PB2		Port			I_2C0SCL				T3CCP0			
PB3		Port			I_2C0SDA				T3CCP1			
PB4	Ain10	Port		SSI2Clk		M0PWM2			T1CCP0	CAN0Rx		
PB5	Ain11	Port		SSI2Fss		M0PWM3			T1CCP1	CAN0Tx		
PB6		Port		SSI2Rx		M0PWM0			T0CCP0			
PB7		Port		SSI2Tx		M0PWM1			T0CCP1			
PC4	C1-	Port	U4Rx	U1Rx		M0PWM6		IDX1	WT0CCP0	U1RTS		
PC5	C1+	Port	U4Tx	U1Tx		M0PWM7		PhA1	WT0CCP1	U1CTS		
PC6	C0+	Port	U3Rx					PhB1	WT1CCP0	USB0epen		
PC7	C0-	Port	U3Tx						WT1CCP1	USB0pflt		
PD0	Ain7	Port	SSI3Clk	SSI1Clk	I_2C3SCL	M0PWM6	M1PWM0		WT2CCP0			
PD1	Ain6	Port	SSI3Fss	SSI1Fss	I_2C3SDA	M0PWM7	M1PWM1		WT2CCP1			
PD2	Ain5	Port	SSI3Rx	SSI1Rx		M0Fault0			WT3CCP0	USB0epen		
PD3	Ain4	Port	SSI3Tx	SSI1Tx				IDX0	WT3CCP1	USB0pflt		
PD4	*USB0DM*	*Port*	*U6Rx*						*WT4CCP0*			
PD5	*USB0DP*	*Port*	*U6Tx*						*WT4CCP1*			
PD6		Port	U2Rx			M0Fault0		PhA0	WT5CCP0			
PD7		Port	U2Tx					PhB0	WT5CCP1	NMI		
PE0	Ain3	Port	U7Rx									
PE1	Ain2	Port	U7Tx									
PE2	Ain1	Port										
PE3	Ain0	Port										
PE4	Ain9	Port	U5Rx		I_2C2SCL	M0PWM4	M1PWM2			CAN0Rx		
PE5	Ain8	Port	U5Tx		I_2C2SDA	M0PWM5	M1PWM3			CAN0Tx		
PF0		Port	U1RTS	SSI1Rx	CAN0Rx		M1PWM4	PhA0	T0CCP0	NMI	C0o	
PF1		Port	U1CTS	SSI1Tx			M1PWM5	PhB0	T0CCP1		C1o	TRD1
PF2		Port		SSI1Clk		M0Fault0	M1PWM6		T1CCP0			TRD0
PF3		Port		SSI1Fss	CAN0Tx		M1PWM7		T1CCP1			TRCLK
PF4		Port					M1Fault0	IDX0	T2CCP0	USB0epen		

Fuente: VALVANO, Jonathan; YERRABALLI, Ramesh. *Chapter 6: Parallel I/O ports*. http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C6_MicrocontrollerPorts.htm. Consulta: 24 de noviembre de 2018.

En el caso de código que sólo utiliza GPIO ha de considerarse, que el valor del PCTL siempre será 0.

Por convención, los registros de la Tabla XLV serán declarados en los programas como constantes (usando la directiva EQU), con el formato 'GPIO_PORT*x*_*registro*_R'. Por ejemplo, para la selección de función analógica en el puerto C, el nombre a leerse será 'GPIO_PORTC_AMSEL_R'.

Existe un registro más a tomar en cuenta para la configuración inicial de periféricos: el de conexión del reloj. Este se presenta como uno solo en común a todos los puertos (y no uno para cada uno), y su dirección no es un offset sino la dirección completa del registro que se desea modificar y la máscara correspondiente consistirá en un 1 en el lugar del puerto que se desee sincronizar.

Tabla XLVI. **Registro para sincronización de puertos**

Nombre	**Definición**	**Configuración**	**Dirección**
SYSCTL_RCGC2_R	Reloj. A cada puerto del A al F le corresponde un bit de este registro destinado a activar su reloj.	Bit 0: Puerto A Bit 1: Puerto B Bit 2: Puerto C Bit 3: Puerto D Bit 4: Puerto E Bit 5: Puerto F	0x400FE108

Fuente: elaboración propia.

6.8.1. Programación *bit-specific*

Dado que repetir los pasos del método leer-modificar-escribir (leer sección 6.4.2), para todos los pines en un puerto haría al código excesivamente largo, se recurre a una estrategia más de la programación amigable de Arm, explicada por Jonathan Valvano y Ramesh Yerraballi, que utiliza las direcciones base de

cada puerto y suma los *offset* correspondientes a los pines que interesa configurar para obtener una sola dirección a leer y escribir por todos los pines. Esto se utiliza para lectura y escritura de los pines, no para su configuración (ya que en la segunda basta la máscara habitual de unos y ceros en las posiciones de los bits de interés).

Como se mencionó anteriormente, según los pines de cada puerto que se desee utilizar, se sumará a la dirección base los offset necesarios. Esta constante para cada pin está dada por la expresión $4*2^b$, donde b es la posición del bit, los resultados se presentan a continuación en formato hexadecimal:

Tabla XLVII. **Constantes por número de pin en puerto**

Número de bit	Constante
7	0x0200
6	0x0100
5	0x0080
4	0x0040
3	0x0020
2	0x0010
1	0x0008
0	0x0004

Fuente: elaboración propia.

Notar que la suma de todas las constantes de la tabla XLVII tiene como resultado el número hexadecimal 0x03FC, correspondiente al offset del registro DATA en la tabla XLV. Así se demuestra que este contiene las direcciones pertinentes a todos los pines de un puerto.

6.8.2. Ritual de inicialización

Todas las consideraciones de diseño expuestas anteriormente están orientadas a la configuración adecuada de pines en la tarjeta Tiva C. Los GPIO requieren el tipo más sencillo de procedimiento, y sirven para los fines de este texto sin extender el contenido a los dominios de tutoriales específicos para el microcontrolador TM4C123GH6PM.

Por estas razones, se especifica el orden correcto en que debe accederse a los registros necesarios conformando el ritual de inicialización de puertos GPIO*:*

- Sincronizar puerto conectando reloj.
- Desbloqueo de pines (PC3-PC0, PD7, PF0), paso opcional.
- Deshabilitar función analógica.
- Limpiar bits en PCTL (ya que cero corresponde al modo de entrada y salida según Figura 191).
- Especificar dirección (entrada o salida).
- Limpiar bits en función alternativa (porque PCTL es igual a cero).
- Habilitar función digital.

A pesar de que este es el orden sugerido, los pasos pueden intercambiarse a conveniencia con excepción estricta del primer paso dado que es imposible configurar un puerto si no está sincronizado.

6.8.3. Ejercicio 11

Con el fin de responder a cualquier duda que quedara de las especificaciones para cálculos de máscaras y direcciones en la sección 6.8, se

propone el único ejercicio que está centrado en un programa directamente sino en cálculos. Está fuertemente recomendado el repaso de la sección 6.4.2 para evitar confusión en este ejercicio.

Se supondrá la situación en que se desea configurar los pines A0 y A1 como salidas digitales (por ejemplo, para encendido y apagado de un led), y el pin A6 como entrada digital (por ejemplo, un switch), en la tarjeta de desarrollo Tiva C. Por orden y fines didácticos se seguirá al pie de la letra el ritual de inicialización de la subsección 6.8.2:

Figura 192. **Bits de interés en puerto a para ejercicio 11**

Puerto A

	0	1	0	0	0	0	1	1
Número de pin	7	6	5	4	3	2	1	0

Fuente: elaboración propia.

El primer paso es la habilitación del reloj. Para esto se especificó en la Tabla XLVI que el registro SYSCTL_RCGC2_R tiene un bit correspondiente a cada puerto. Considerando que se usará solamente el puerto A, la máscara necesaria para encender el reloj con el método leer-modificar-escribir será 0x00000001.

El segundo paso es el desbloqueo, sin embargo ningún pin del puerto A se encuentra bloqueado y se salta al tercer paso con la deshabilitación de función analógica. Con este paso debe considerarse dos cosas:

- La dirección real del registro AMSEL se calcula sumando su offset (0x528, tabla XLV), y la dirección base del puerto A (0x40004000, tabla

XLIV). Entonces la localidad de memoria en que se debe leer y escribir en este paso es 0x40004528.

- La máscara que se usará para modificar los bits de interés con el método de leer-modificar-escribir se calculará haciendo uso de la figura 192. Si el valor binario resultante de señalar con un uno los bits de interés se convierte al hexadecimal entonces se obtiene la máscara 0x43.

Si se escribe la secuencia de código para indicar en el registro AMSEL que debe desactivarse la función analógica será, entonces, algo similar a la figura 193:

Figura 193. **Deshabilitación de función analógica para pines A0, A1 y A6**

```
LDR R0, =40004528
LDR R1, [R0]
BIC R1, 0x43
STR R1, [R0]
```

Fuente: elaboración propia.

El cuarto paso consiste en indicar al PCTL que los pines serán utilizados como GPIO que, como se explicó anteriormente, consiste en llenar el registro con ceros en las posiciones de los pines A0, A1 y A6. La dirección real del registro será 0x4000452C por el mismo análisis que en el paso anterior y la modificación de su contenido también se hará en este caso con una instrucción BIC por interesar la escritura de ceros. Para este registro debe tenerse el cuidado de no confundir la máscara con la de los demás pasos, ya que para el PCTL cada dígito hexadecimal (cuatro bits) contiene la información de cada pin por lo que la máscara necesaria para escribir cero en los bits 0, 1 y 6 será

0x0F0000FF. El resto del método de modificación continúa igual que en los demás pasos

El quinto paso especifica dirección de los pines. En este caso no debe olvidarse que se requiere que A0 y A1 sean salidas (máscara 0x03), y A6 entrada (máscara 0x40), en el registro DIR del puerto A (dirección real 0x40004400).

El sexto y séptimo paso se realizan con la misma máscara que en el segundo con direcciones 0x40004420 para AFSEL y 0x4000451C para DEN. Los cálculos de direcciones son posibles no perdiendo de vista la tabla XLV.

Por último, para este procedimiento se recuerda al programador que la programación *bit-specific* permite leer y escribir en las localidades de memoria correspondientes a A0, A1 y A6 con una sola dirección, que resulta de la suma de las constantes de la tabla XLVII. Para este caso la suma será entonces 0x10C y la dirección total se dará tomando en cuenta la del puerto A: 0x4000410C. Esto servirá para leer el estado de las entradas o modificar el de las salidas cuando todo haya pasado el ritual de inicialización, y puede tomarse las constantes individualmente o como conjuntos.

Los cálculos se vuelven más sencillos conforme el programador se familiariza con su significado, es necesario realizar varias veces este procedimiento con distintos valores para verificar que se ha comprendido y así, avanzar a su implementación.

Tabla XLVIII. **Direcciones y máscara para pines A0, A1 y A6 en Tiva C**

Reloj		
Dirección: 0x400FE108		
Máscara para sincronización: 0x00000001=0x01		
Puerto A		
Dirección base: 0x40004000		
Registro	Offset	Dirección real
AMSEL	0x528	0x40004528
DEN	0x51C	0x4000451C
LOCK	0x520	0x40004520
DIR	0x400	0x40004400
PCTL	0x52C	0x4000452C
AFSEL	0x420	0x40004420
DATA	0x3FC	0x400043FC
Máscara para configuración de A0, A1 y A6: 0x43		
Máscara para configuración de A0 y A1: 0x03		
Máscara para configuración de A6: 0x40		
Dirección de A0, A1 y A6: 0x4000410C		
Dirección de A0, A1: 0x4000400C		
Dirección de A6: 0x40004100		

Fuente: elaboración propia.

6.8.4. Ejercicio 12

Utilizando los principios explicados en el ejercicio 11, se desarrollará el código necesario para realizar obtener un led parpadeante (*blinking led*), en la tarjeta de desarrollo Tiva C.

Para apoyar la comprensión ordenada de este ejercicio, la tabla XLIX resume las direcciones y máscaras necesarias para trabajar el pin F2. Estos se calculan de la misma forma que en la subsección anterior.

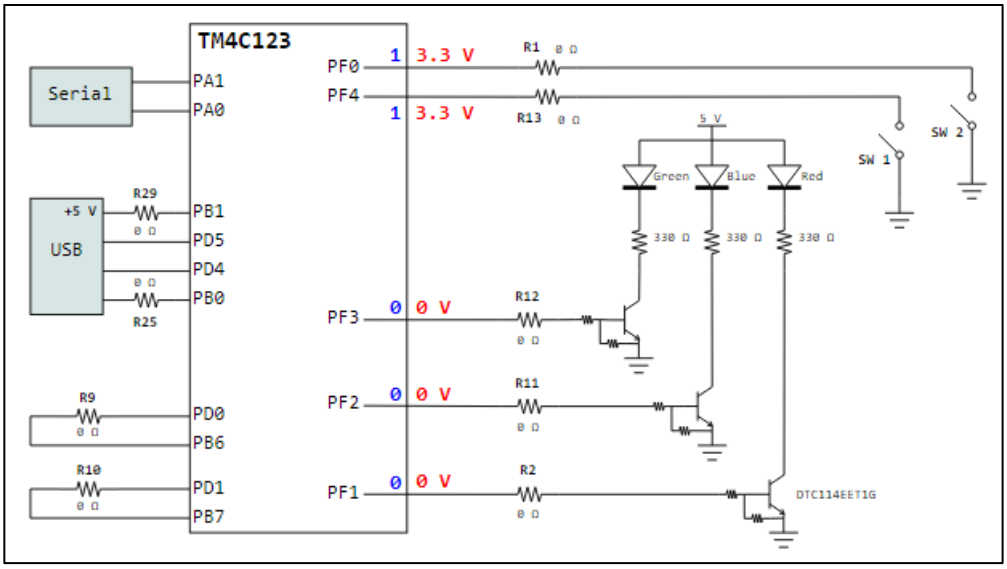
Tabla XLIX. **Direcciones y máscara para pin F2 en Tiva C**

Reloj		
Dirección: 0x400FE108		
Máscara para sincronización: 0x00100000=0x20		
Puerto F		
Dirección base: 0x40025000		
Registro	Offset	Dirección real
AMSEL	0x528	0x40025528
DEN	0x51C	0x4002551C
LOCK	0x520	0x40025520
DIR	0x400	0x40025400
PCTL	0x52C	0x4002552C
AFSEL	0x420	0x40025420
DATA	0x3FC	0x400253FC
Máscara para configuración de F2: 0x04		
Dirección de F2: 0x40025010		

Fuente: elaboración propia.

Una de las ventajas de utilizar una tarjeta de desarrollo es que suele traer elementos extra al chip que facilitan ciertas tareas o pruebas para luego elaborar hardware más completo. Para el caso de la Tiva C, algunos pines del microcontrolador TM4C123GH6PM (como se muestra en la figura 194), son conectados a puntos de interés común en muchas aplicaciones. En este ejercicio interesa la conexión a los led de la placa que, juntos, componen un RBG (*Red, Blue, Green*), en los pines F1-F3.

Figura 194. **Mapeo de pines de TM4C123GH6PM en tarjeta de desarrollo Tiva C**



Fuente: VALVANO, Jonathan; YERRABALLI, Ramesh. *Chapter 6: Parallel I/O ports*. http://users.ece.utexas.edu/~valvano/Volume1/E-Book/C6_MicrocontrollerPorts.htm. Consulta: 24 de noviembre de 2018.

Por simplicidad en este ejercicio se tomará simplemente el pin F2 (led azul), dejando abierta la posibilidad de cambiar el color de la luz sea eligiendo otro pin del led RBG o combinaciones de ellos para obtener los enlistados en la tabla L. En caso de que el programa 11 deba ser utilizado para otro color de led, bastará con calcular las máscaras adecuadas para configuración y la dirección para su escritura tomando en cuenta la tabla XLVII.

Tabla L. **Combinaciones de color posibles en LED RBG de Tiva C**

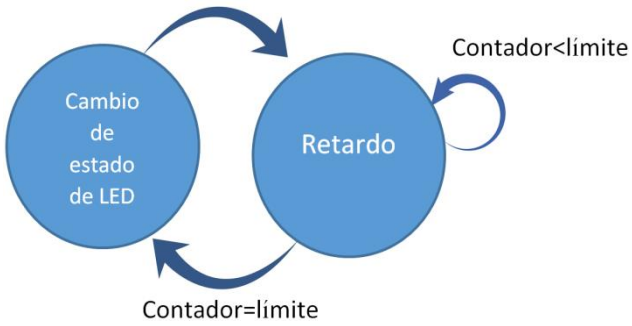
Color	Pines requeridos
Rojo	F1
Azul	F2
Verde	F3
Amarillo	F1 y F3
Magenta	F1 y F2
Cian	F2 y F3
Blanco	F1, F2 y F3

Fuente: elaboración propia.

También puede existir el caso de querer utilizar un pin de otro puerto para controlar un led convencional o algún otro actuador. De cualquier forma debe tenerse presente que todas las salidas digitales binarias se comportan igual (con estado alto y bajo), este ejercicio podría aplicar para cualquier tipo de salida siempre que se calculen adecuadamente las direcciones y máscaras.

El led parpadeante se compone de una máquina de estados de Moore muy simple, como se muestra en la figura 195. Esta consiste simplemente en cambiar de estado el pin que controla el led (si es alto, cambiar a bajo y viceversa), para luego esperar la cantidad de tiempo deseada para poder observar la luz encendiéndose y apagándose.

Figura 195. **Máquina de estados para programa 11**



Fuente: elaboración propia.

Para el retardo se utilizará la rutina del programa 6, lo que da como resultado un led parpadeante de un segundo por estado (frecuencia de 0,5 Hz).

La conmutación (cambio de estado o *toggle*), se hace con ayuda de la función lógica XOR. Cuando esta se realiza entre el estado del led y un uno lógico, el resultado es la negación del estado en el siguiente instante en el tiempo. Así, como se especifica en la tabla, una compuerta XOR intercalada con un retardo permitiría encender el led, mantenerlo en ese estado durante un tiempo, apagarlo, mantener el estado y repetir el ciclo.

Tabla LI. **Funcionamiento de XOR para conmutación**

Q_n	Operación	Q_{n+1}
0	0 XOR 1 = 1	1
1	1 XOR 1 = 0	0

Q_n: instante actual. Q_n: instante siguiente.

Fuente: elaboración propia.

El encendido y apagado del led se hace con el método leer-modificar-escribir en la dirección del pin que interesa (en este caso, F2), con la excepción de que para este caso no se utilizarían las tradicionales BIC y ORR, sino XOR para modificar el contenido del registro.

Con la práctica que el programador ha adquirido hasta este punto basta con leer ordenadamente el código de la figura 194 para comprender cómo se desarrolla en un programa de UAL el ritual de inicialización y el control de salidas. Tomar en cuenta que las direcciones importantes han sido declaradas como constantes por comodidad.

Como se ha repetido varias veces a lo largo de este capítulo y el anterior, el programa propuesto es una forma sencilla de solucionar los requerimientos pero puede, en algún momento, mejorarse con la práctica, la comodidad y el conocimiento de la ISA. Se alienta la exploración de modificaciones al programa 11 para crear código más compacto y eficiente.

Figura 196. **Programa 11**

```
GPIO_PORTF2         EQU 0x40025010
GPIO_PORTF_DIR_R    EQU 0x40025400
GPIO_PORTF_AFSEL_R  EQU 0x40025420
GPIO_PORTF_DEN_R    EQU 0x4002551C
GPIO_PORTF_AMSEL_R  EQU 0x40025528
GPIO_PORTF_PCTL_R   EQU 0x4002552C
SYSCTL_RCGCGPIO_R   EQU 0x400FE608
CONSTANTE           EQU 2000000

          AREA    codigo, CODE, READONLY,ALIGN=2
          THUMB
          EXPORT Start

Start
    ; Paso 1: activación de reloj para puerto F.
    LDR R1, =SYSCTL_RCGCGPIO_R
    LDR R0, [R1]
    ORR R0, R0, #0x20
    STR R0, [R1]
    NOP
    NOP
    ; Paso 2: desbloqueo de pines. Es este caso los pines están libres.
        ; En este programa no hay pines que necesiten desbloqueo.
    ; Paso 3: deshabilitar función analógica.
    LDR R1, =GPIO_PORTF_AMSEL_R
    LDR R0, [R1]
    BIC R0, #0x04
    STR R0, [R1]
    ; Paso 4: configurar como GPIO, PCTL=0.
    LDR R1, =GPIO_PORTF_PCTL_R
    LDR R0, [R1]
    BIC R0, R0, #0x00000F00
    STR R0, [R1]
    ; Paso 5: especificar dirección PF2.
    LDR R1, =GPIO_PORTF_DIR_R
    LDR R0, [R1]
    ORR R0, R0, #0x04
    STR R0, [R1]
    ; Paso 6: limpiar bits en función alternativa.
    LDR R1, =GPIO_PORTF_AFSEL_R
    LDR R0, [R1]
    BIC R0, R0, #0x04
    STR R0, [R1]
    ; Paso 7: habilitar como puerto digital.
    LDR R1, =GPIO_PORTF_DEN_R
    LDR R0, [R1]
    ORR R0, R0, #0x04
    STR R0, [R1]
    ;       Regreso a rutina Start.
    B Loop

Delay   ; Rutina de retardo de 1s.
    ADD R0, #1
    NOP
    NOP

    NOP
    CMP R0, R1
    BNE Delay
    BX LR

Loop    ; Ciclo para encendido y apagadao de LED.
    ; Cargar en R0 el contenido de la dirección de memoria a la que
    ; apunta R1 (GPIO_PORTF2=0x40025010)
    LDR R1, =GPIO_PORTF2
    LDR R0, [R1]
    ; Cambiar al valor del LED al negado del anterior (R0 = R0^0x04).
    EOR R0, R0, #0x04
    ; Almacenar el valor de GPIO_PORTF2 modificado.
    STR R0, [R1]
    ; Reiniciar contador para retardo e ir a Delay con enlace.
    LDR R0, =0
    LDR R1, =CONSTANTE
    BL Delay
    ;Repetir subrutina Loop.
    B Loop

    ALIGN
    END
```

Fuente: elaboración propia.

6.8.5. Ejercicio 13

Para ampliar el ejercicio anterior, se propone el código para control de dos led en la tarjeta Tiva C (F1 y F2), por medio de uno de sus botones (F0). El programa consistirá en el encendido de las luces cuando el botón se encuentre sin presionar y el apagado cuando se presione, siendo así porque el estado de este botón se encuentra negado en esta tarjeta de desarrollo. Si se observan las combinaciones de la tabla L, puede deducirse que el color de la luz será magenta; sin embargo este puede modificarse y al igual que en los otros ejemplos, los puertos y pines específicos pueden cambiarse para adaptarse a las necesidades del proyecto.

Los pines correspondientes a led se encuentran desbloqueados (como en el ejercicio de la sección 6.8.4). El pin del botón (F0), por otro lado, está bloqueado por defecto y exige una consideración especial para su uso libre. En esta situación se usará el registro LOCK como el resto de los destinados a configuración, con la excepción de que no se usará una máscara sino la llave de desbloqueo que permitirá, hacer efectivo el desbloqueo mediante un nuevo registro: *Commit Register* (CR, con offset 0x524). En este simplemente se especifica que los cambios de se harán en los bits marcados con un uno y se usará una máscara 0xFF para abarcar los 8 bits de un puerto.

Los valores calculados para este ejercicio se encuentran en la tabla LII.

Tabla LII. **Direcciones y máscaras para pines F0, F1 y F2 en Tiva C**

Reloj		
Dirección: 0x400FE108		
Máscara para sincronización: 0x00100000=0x20		
Puerto F		
Dirección base: 0x40025000		
Registro	Offset	Dirección real
AMSEL	0x528	0x40025528
DEN	0x51C	0x4002551C
LOCK	0x520	0x40025520
CR	0x524	0x40025524
DIR	0x400	0x40025400
PCTL	0x52C	0x4002552C
AFSEL	0x420	0x40025420
DATA	0x3FC	0x400253FC
Máscara para configuración de F0, F1 y F2: 0x07		
Máscara para configuración de F0: 0x01		
Máscara para configuración de F1 y F2: 0x06		
Dirección de F0: 0x40025004		
Dirección de F1 y F2: 0x40025018		

Fuente: elaboración propia.

El programa de la figura 197 se desarrolla de forma similar al programa 11. Para su comprensión adecuada debe tenerse en cuenta el flujo del código con las instrucciones de saltos, que la rutina Delay en este caso se utiliza como método antirrebote (para obtener lecturas correctas), con un retardo de 50ms y que las direcciones de registro de los pines se usan para leer el estado de las entradas (botón) y modificar el de las salidas (led).

Figura 197. **Programa 12**

```
GPIO_PORTF12        EQU 0x40025018
GPIO_PORTF0         EQU 0x40025004
GPIO_PORTF_DIR_R    EQU 0x40025400
GPIO_PORTF_AFSEL_R EQU 0x40025420
GPIO_PORTF_DEN_R    EQU 0x4002551C
GPIO_PORTF_LOCK_R   EQU 0x40025520
GPIO_PORTF_AMSEL_R EQU 0x40025528
GPIO_PORTF_PCTL_R   EQU 0x4002552C
SYSCTL_RCGCGPIO_R   EQU 0x400FE608
GPIO_LOCK_KEY       EQU 0x4C4F434B
GPIO_PORTF_CR_R     EQU 0x40025524
CONSTANTE           EQU 100000

        AREA    codigo, CODE, READONLY,ALIGN=2
        THUMB
        EXPORT Start

Start
    ; Paso 1: activación de reloj para puerto F.
    LDR R1, =SYSCTL_RCGCGPIO_R
    LDR R0, [R1]
    ORR R0, R0, #0x20
    STR R0, [R1]
    NOP
    NOP
    ; Paso 2: desbloqueo de pines.
    LDR R1, =GPIO_PORTF_LOCK_R  ; uso de llave
    LDR R0, =GPIO_LOCK_KEY
    STR R0, [R1]
    LDR R1, =GPIO_PORTF_CR_R    ; "commit"
    MOV R0, #0xFF
    STR R0, [R1]
    ; Paso 3: deshabilitar función analógica.
    LDR R1, =GPIO_PORTF_AMSEL_R
    LDR R0, [R1]
    BIC R0, #0x07
    STR R0, [R1]
    ; Paso 4: configurar como GPIO, PCTL=0.
    LDR R1, =GPIO_PORTF_PCTL_R
    LDR R0, [R1]
    BIC R0, R0, #0x000000FF
    BIC R0, R0, #0x00000F00
    STR R0, [R1]
    ; Paso 5: especificar dirección de PF1 y PF2.
    LDR R1, =GPIO_PORTF_DIR_R
    LDR R0, [R1]
    ORR R0, R0, #0x06
    STR R0, [R1]
    ; Paso 5: especificar dirección de PF0.
    LDR R1, =GPIO_PORTF_DIR_R
    LDR R0, [R1]
    BIC R0, R0, #0x01
    STR R0, [R1]
    ; Paso 6: limpiar bits en función alternativa.
    LDR R1, =GPIO_PORTF_AFSEL_R
    LDR R0, [R1]
    BIC R0, R0, #0x07
    STR R0, [R1]
    ; Paso 7: habilitar como puerto digital.
    LDR R1, =GPIO_PORTF_DEN_R
    LDR R0, [R1]
    ORR R0, R0, #0x07
    STR R0, [R1]

    LDR R3, =CONSTANTE
    B Loop
```

```
Delay   ; Rutina de retardo de 50ms.
    ADD R2, #1
    NOP
    NOP
    NOP
    NOP
    CMP R2, R3
    BNE Delay
    BX LR

Encender ; Rutina de encendido de F1 y F2.
    LDR R5, =GPIO_PORTF12
    LDR R4, =0x06
    STR R4, [R5]
    B Loop
Apagar  ; Rutina de apagado de F1 y F2.
    LDR R5, =GPIO_PORTF12
    LDR R4, =0x0
    STR R4, [R5]
    B Loop

Loop    ; Ciclo para lectura de switch.
    ; Leer el valor del switch.
    LDR R1, =GPIO_PORTF0
    LDR R0, [R1]
    ;Retardo antirrebote.
    LDR R2, =0
    BL Delay
    ; Si F0=1, encender LED. Si F0=0, apagar LED.
    CMP R0, #0x01
    BEQ Encender
    CMP R0, #0x0
    BEQ Apagar
    B Loop

    ALIGN
    END
```

Fuente: elaboración propia.

7. PROGRAMACIÓN DE PROCESADOR CORTEX-A53 UTILIZANDO EL SOC BROADCOM BCM2837B0

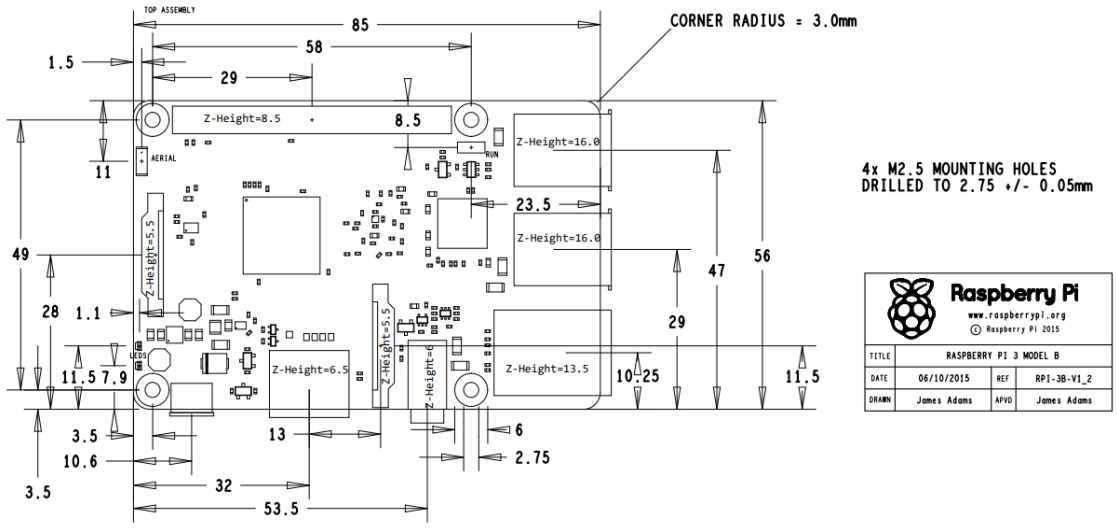
El desarrollo de toda la teoría y especificaciones acerca del uso del lenguaje de ensamblador llevó en el capítulo anterior a la puesta en práctica de los conocimientos para programar procesadores Arm del perfil de microcontroladores. En este se desarrollará la guía básica para programación de procesadores Cortex-A53 con ayuda de los temas abarcados en los capítulos 2, 4 y 5.

Ha de tomarse en cuenta que las secciones siguientes se encuentran divididas de forma similar al capítulo 6 y la explicación de los grupos fundamentales en el set de instrucciones no se repetirá (para evitar redundancia); en su lugar, se presenta una tabla de resumen en la sección 7.2 y se indicará las subsecciones donde se encuentre desarrollada la explicación de cada grupo para luego apoyar con los ejercicios de este capítulo. Así, este capítulo deberá leerse estrictamente en conjunto con el anterior debido a que los ejercicios son simplemente traducciones, para su uso en el chip mencionado. Debe tomarse en cuenta que esto es posible con facilidad debido a que se utilizará el set de instrucciones A32 (sección 4..), que corresponde a los mnemónicos disponibles en Armv7-A y Armv8-A en su estado AArch32, el set A64 se estudiará brevemente en la Sección 7… para servir de introducción al análisis más profundo en documentos como *ARMv8 Instruction Set Overview Architecture Group*. Se recomienda, al igual que en la guía anterior, la consulta del set de instrucciones para información en casos específicos.

7.1. Raspberry Pi

Raspberry Pi es llamado mini ordenador en diversos ambientes por contener las características de un computador personales en una sola placa que puede sostenerse en la palma de la mano. Se caracteriza por ser una serie de placas de bajo costo, bajo consumo y tamaño ideal para adaptarse a proyectos de todo tipo. Por lo general se utiliza sobre el hardware un sistema operativo con herramientas de *open software* (es común que su base esté en la estructura de Linux).

Figura 198. **Esquemático Raspberry Pi 3**



Fuente: *Mechanical Drawings.* https://www.raspberrypi.org/documentation/hardware/raspberrypi/mechanical. Consulta: 20 de diciembre de 2018.

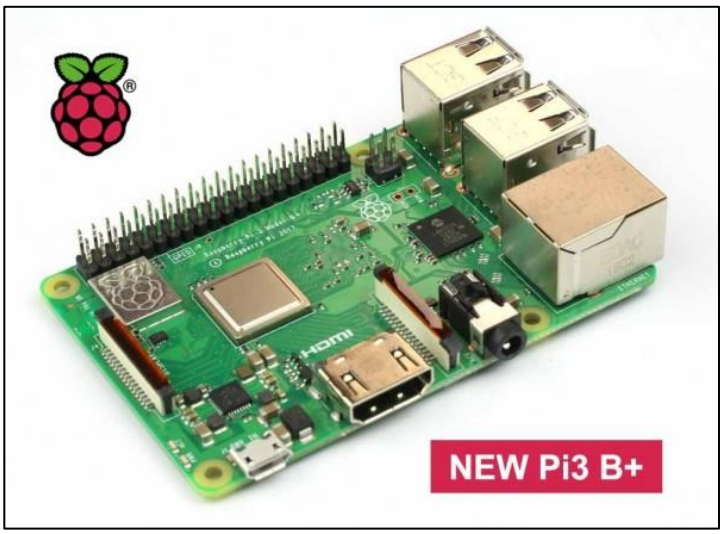
Como se mencionó, las placas Raspberry contienen periféricos que le permiten servir de dispositivo maestro como son: GPIO, UART, SPI e I2C; habilitando aplicaciones simples de control de actuadores hasta visión artificial,

IoT, robótica y mini *clusters* de computadores. La potencia de las Raspberry es mucho menor a la de una PC convencional, y sus aplicaciones también se limitan por este factor, sin embargo son una gran alternativa de bajo consumo.

El chip BCM2837B0 de la empresa Broadcom fue diseñado para su uso en el mini ordenador Raspberry Pi modelos 3B+ y 3A+. Tiene por procesador un arreglo de cuatro núcleos Cortex-A53 (quadcore), de 64 bits, el primero de Armv8-A usado en Raspberry Pi, a 1,4 GHz ello hace su rendimiento y capacidad de procesamiento en paralelo mucho mayor a la de un microcontrolador convencional.

Junto a la GPU VideoCore IV de 32 bits, RAM de 512MB, conectores RJ45 para conexión a la red, módulos WiFi y Bluetooth, puertos USB, tecnología PoE (Power over Ethernet), lector de tarjeta SD para almacenamiento no volátil y salidas de video y audio HDMI; el procesador en Raspberry Pi 3B+ es capaz de ejecutar funciones complejas, entre las que suelen ser más solicitadas situaciones en que deba funcionar como servidor para control de bases de datos, asignación de direcciones lógicas para otros dispositivos, control de sitios web, entre otros.

Figura 199. **Raspberry Pi en su modelo 3B+**

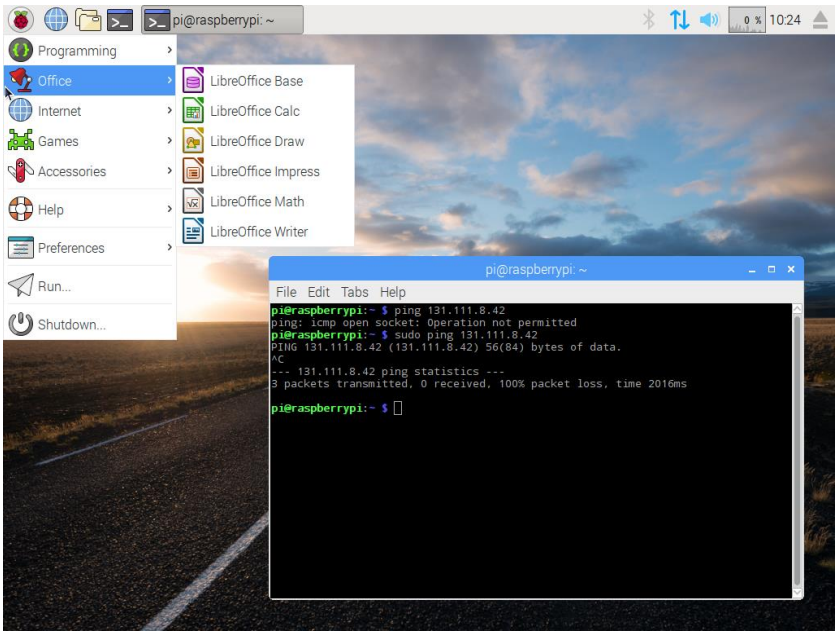


Fuente: *Raspberry. PI 3 modèle B+ 1Gb.* https://yadom.fr/raspberry-pi-3-modele-bp-1gb.html. Consulta: 20 de diciembre de 2018.

Por estar basado el procesador en una arquitectura RISC, los sistemas operativos que se utilizan sobre él no son compatibles con los habituales en procesadores Intel o AMD, por ejemplo. Existen distribuciones alternativas como Debian, Firefox OS, Kali Linux, Pidora, Raspbian, FreeBSD y NetBSD, entre muchos otros.

Varios de ellos ofrecen una versión de terminal de comandos y una gráfica mucho más amigable al usuario, pudiendo acceder directamente a ambos modos por HDMI o remotamente con SSH (Secure Shell), y servidores VNC (Virtual Network Computing), respectivamente. En este capítulo los ejercicios serán ejecutados sobre el modo gráfico de uno de los sistemas operativos más utilizados y con mayor documentación: Raspbian, para facilidad en la comprensión del uso del dispositivo en caso el programador no esté muy familiarizado con él.

Figura 200. **Escritorio PIXEL de Raspbian**



Fuente: Raspberry. *Pi's Linux-based PIXEL desktop now available for PC and Mac*. https://betanews.com/2016/12/21/raspberry-pis-pixel-desktop-now-available-for-pc-and-mac/. Consulta: 20 de diciembre de 2018.

7.1.1. Memoria

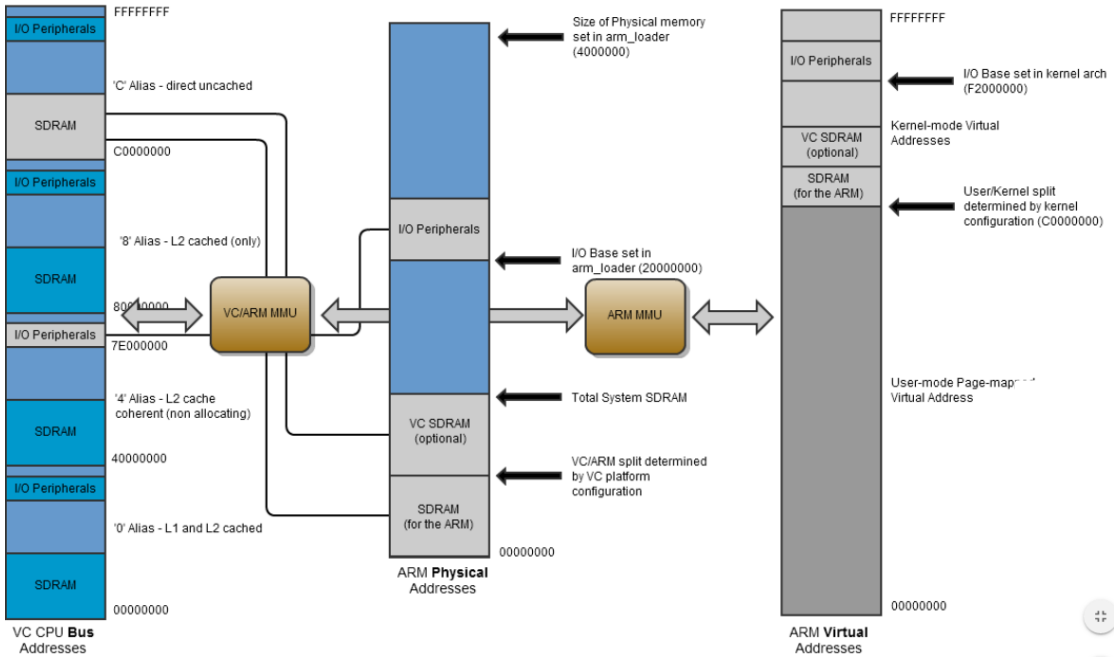
Para la consideración de direcciones de memoria en el chip Broadcom BCM2837B0 es imperativo recordar que este es un SoC, y todos sus componentes tienen direcciones de memoria asignadas.

Para este chip en específico, la organización de Raspberry Pi presenta la documentación de BCM2835 (un antecesor), como equivalente. Se definen tres grupos de direcciones para tener presente en el momento de diseñar: direcciones virtuales (que están ligadas al funcionamiento del sistema operativo), direcciones físicas de Arm y direcciones mapeadas a buses del SoC.

Estos grupos se usan con la finalidad de ubicar los espacios de memoria adecuados en RAM, y sus equivalentes se traducen por medio de una MMU de Arm (unidad de gestión de memoria).

La figura 201 expresa de forma gráfica los tres grupos de direcciones de memoria, con la especificación de que entre cada uno se encuentra el MMU traduciendo sus espacios por medio de tecnología AMBA AXI.

Figura 201. **Mapa de memoria para SoC BCM2835**



Fuente: *Broadcom Corporation. BCM2835 ARM Peripherals*. p. 4.

En el diagrama resulta importante distinguir segmentos específicos de interés para ciertas aplicaciones. Entre las consideraciones más importantes están:

- Las direcciones físicas de Arm comienzan en 0x00000000 para la RAM (columna central en figura 201).
- Existe una sección opcional de VideoCore destinada al uso en sistemas con periféricos de GPU. Esta se mapea a las direcciones de buses de la RAM en direcciones de buses comenzando en 0xC0000000.
- En la sección Arm los periféricos corresponden al rango entre 0x20000000 y 0x20FFFFFF con correspondencia directa al rango entre 0x7E000000 y 0x7EFFFFFF, para los direcciones de buses (columna izquierda en figura 201).

Por las especificaciones anteriores se aclara que el software que accede a la RAM directamente debe utilizar las direcciones físicas (con base 0x00000000), y el que accede a la RAM usando motores DMA debe utilizar las direcciones de buses (con base 0xC0000000).

7.1.2. Code::Blocks

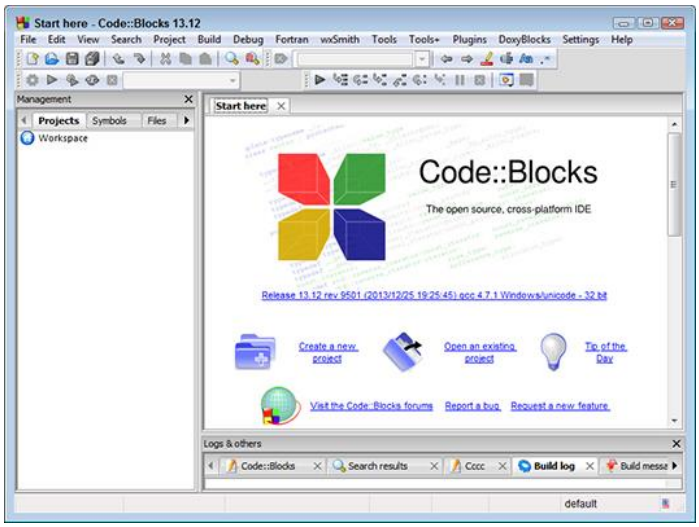
Es un entorno de desarrollo integrado gratuito principalmente orientado a programas de lenguajes C, C++, Fortran y de ensamblador. Su razón de ser se encuentra en la configurabilidad mediante complementos opcionales según el programador decida agregar dentro de una interfaz sencilla y consistente. Se comprende entonces a esta herramienta como un núcleo configurable para propósitos de programación.

Code::Blocks se caracteriza por ser libre para usarse en distintos sistemas operativos, con una licencia pública general de GNU (de interés especial para este capítulo).

La interfaz buscará en el sistema compiladores para trabajar, siendo algunos de los más comunes Borland C++ Compiler, DigitalMars Compiler, Intel C++ Compiler y GCC en sus versiones para Windows y GNU/Linux. La compilación de los programas se suma a características que hacen del proceso de diseño de código una tarea ordenada y facilitada con opciones como:

- Espacios de trabajo para diversos proyectos.
- Navegador de proyectos.
- Control y señalización de sintaxis por autocomplemento configurable, coloreo y tabulación.
- Asistente de generación de clases.
- Sistema de construcción de ejecutable rápido.
- Compilación en paralelo.
- Gestión de *breakpoints.*
- Visualización de pila y registros.
- Desensamblador.

Figura 202. **Ventana inicial de Code::Blocks**



Fuente: How to create a new Code::Blocks project that includes a header. https://www.dummies.com/programming/cpp/how-to-create-a-new-codeblocks-project-that-includes-a-header/. Consulta: 20 de diciembre de 2018.

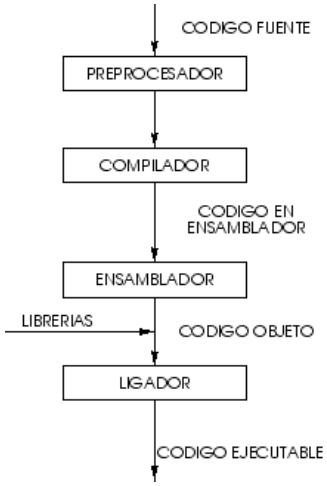
7.1.3. Uso de GCC

GCC (*GNU Compiler Collection*) es un paquete de compilación libre orientado al uso en sistemas con base en Linux para lenguajes como C, C++, Fortran, Do, Ada, entre otros. GCC es ejecutado con ayuda de diversos comandos para compilación específica y personalizada; estos se ejecutan por medio de una interfaz en caso de que se utilice un entorno como Code::Blocks.

Para el programador que aún tenga dudas acerca de cómo es posible que un compilador maneje programas en lenguaje de ensamblador, es útil recordar que el ensamblaje es parte de la compilación junto con otros pasos como se muestra en el diagrama de la figura 203. Así en caso de que se indique al

compilador que se quiere ejecutar un archivo de UAL (por ejemplo), este saltará los pasos anteriores y simplemente seguirá con el proceso.

Figura 203. **Proceso de un compilador**



Fuente: *Estructura de un Programa y Directivas del preprocesador en Lenguaje C [Tutorial 2].* https://adolfredobelizario.wordpress.com/2012/06/04/tutorial-2-lenguaje-c/. Consulta: 20 de diciembre de 2018.

7.1.3.1. Directivas

El uso de un entorno como mediador entre el programador y el compilador ahorrará el tiempo de aprendizaje de ciertos comandos específicos, porque estos se darán automáticamente por medio de botones y ventanas en el modo gráfico. Sin embargo es inherente al compilador las directivas utilizadas para construir el código de ensamblador adecuadamente y darle al sistema, las indicaciones para tratar las palabras escritas en el archivo.

Por esta razón las directivas para GCC serán propias para este compilador, así como en la Sección_ se especificaban diferencias entre CCS y

Keil. A pesar de esto, las equivalencias son sencillas dado que la mayoría de veces es cuestión de sintaxis y pueden relacionarse fácilmente a las utilizadas en el capítulo 6 por medio de la siguiente tabla de comparación:

Tabla LIII. **Comparación de directivas GCC y Keil**

Directiva	Keil	GCC
Espacio de código	TEXT	.text
Espacio de RAM	DATA	.data
Exportar etiqueta	EXPORT	.global
Alinear sección	ALIGN	.align
Declarar constant	EQU	.equ
Terminar programa	END	.end
Especificación de modo Thumb	THUMB	.thumb, .code16
Especificación de modo Arm	ARM	.code32

Fuente: elaboración propia.

Otras variantes en la sintaxis para el ensamblador de GCC incluyen:

- Los comentarios se señalan con “@”, “/**/” o “//” en lugar de “;”.
- Los operandos inmediatos se señalan con “#” o “$”.
- Las etiquetas deben estar seguidas por “:”.
- El enlazador de GCC espera una etiqueta “main” como punto de partida en el programa (equivalente a la etiqueta Start del capítulo 6).
- El programa idealmente se comenzará con la directiva .text y se terminará con .end; sin embargo, el compilador no exige estas condiciones.

7.2. Resumen de instrucciones

Como se mencionó al inicio de este capítulo, las instrucciones que pueden ejecutarse en un procesador Armv8 en estado AArch32 (es decir, el set A32),

es prácticamente el mismo que el utilizado con Armv7 y a pesar de que Cortex-A es un perfil usualmente más complejo que Cortex-M, los grupos principales de instrucciones que es necesario comprender y aprender a manejar como programador iniciante son los mismos explicados a través del capítulo 6. Por esta razón, la tabla LIII se presenta como una referencia rápida de instrucciones que se usarán en este capítulo junto con la especificación del apartado donde se encuentran explicadas. Tener en cuenta siempre que algunas instrucciones pueden no estar disponibles en ambos perfiles (no aparecerán en la tabla), y al estudiar el set completo para Cortex-A resulta una colección mucho más extensa por sus características especiales (multinúcleos, NEON, entre otros), pero estas ya no forman parte de una guía introductoria, sino requerirían de un análisis mucho más detallado del set y hardware involucrado según el área de interés.

Las instrucciones destinadas al uso en el chip Broadcom BCM2837B0 y sugeridas para comprensión inicial de los grupos son:

Tabla LIV. **Resumen de instrucciones básicas UAL Cortex-A53**

Mnemónico	Instrucción	Estructura A32
Carga y Almacenamiento (descripción en Sección 6.2)		
LDM	Cargar múltiples registros	Rn, lista
LDR	Cargar registro	Wt, [Wn{, #offset}] Wt, =imm32
MOV	Mover	Rd, Op2 Rd, #imm8
MVN	Mover NOT	Rd, Op2
STM	Almacenar múltiples registros	Rn, lista
STR	Almacenar registro de palabra	Wt, [Wn{, #offset}]
Interacción con pila (descripción en Sección 6.3)		
POP	Obtener registros de pila	{lista de registros}
PUSH	Almacenar registros en pila	{lista de registros}

Continuación de la tabla LIV.

Aritmética y lógica (descripción en sección 6.4)		
ADD	Suma	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
MUL	Multiplicación con resultado de 32 bits	{Rd,} Rn, Rm
SUB	Sustracción	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
UMULL/ SMULL	Multiplicación sin/con signo y resultado de 64 bits	RLo, RHi, Rn, Rm
AND	Compuerta AND lógica	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
BIC	Limpieza de bits	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
EOR	Compuerta OR exclusiva (XOR)	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
ORR	Compuerta OR lógica	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
ORN	Compuerta NOR lógica	{Rd,} Rn, Rm {Rd,} Rn, Op2 {Rd,} Rn, #imm12
ROR	Rotación a la derecha	Rd, Rn, <Rs|#n>
Condicionamiento (descripción en sección 6.5)		
B	Salto	etiqueta
BL	Salto con enlace	etiqueta
BLX	Salto indirecto con enlace	Rm
BX	Salto indirecto	Rm
CMN	Comparar negativo	Rn, Op2
CMP	Comparar	Rn, Op2
NOP	No operar	
VFP (descripción en sección 6.7)		
VABS.F32	Absoluto	Sd, Sm
VADD.F32	Suma	{Sd, } Sn, Sm
VCMP.F32	Comparar	Sd, Sm Sd, #0.0
VCVT.S32.F32	Convertir de formato punto flotante a entero	Sd, Sm
VCVTR.S32.F32	Convertir de punto flotante a punto fijo con redondeo	Sd, Sm
VDIV.F32	División	{Sd, } Sn, Sm
VLDR.F32	Cargar registro	Sd, =imm32 Sd, [Rn]

Continuación de la tabla LIV.

VMOV.F32	Mover simple o múltiple	Sd, Sm Sd, Rt Sm, Sm1, Rt, Rt2
VMRS	Mover de FPSCR a registro o APSR	Rt, FPSCR APSR_nzcv, FPSCR
VMSR	Mover de registro a FPSCR	FPSCR, Rt
VMUL.F32	Multiplicación	{Sd, } Sn, Sm
VNEG.F32	Negación	Sd, Sm
VPOP	Obtener registro de pila	lista
VPUSH	Almacenar registro en pila	lista
VSQRT.F32	Raíz cuadrada	Sd, Sm
VSTR.F32	Almacenar en memoria	Sd, [Rn]
VSUB.F32	Sustracción	{Sd, } Sn, Sm

Fuente: elaboración propia.

7.3. Carga y almacenamiento

La sección 6.2 explicó el primer grupo fundamental de instrucciones UAL: aquellas que se tratan de acceso a memoria. Aunque en ese caso con el programa 1 se presentó para uso en Cortex-M, es posible también usar algunas de las instrucciones en Cortex-A a pesar de que la lista se ve considerablemente reducida en especial porque MOV se encuentra limitada a segundos operandos de 8 bits o arreglos de bits rotados cierta cantidad de espacios. MOV32, MOVT y MOVW desaparecen en este perfil; y la escritura de valores mayores a 8 bits se recomienda hacerla por medio de la pseudoinstrucción *LDR Rn, =constante*.

7.3.1. Ejercicio 13

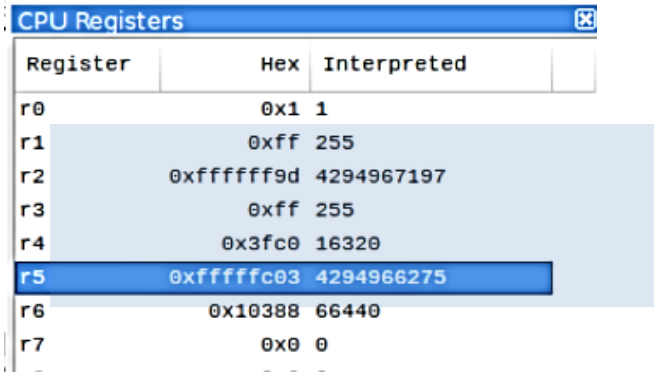
El ejercicio a continuación muestra, entonces, una traducción del código escrito para el programa 1 de forma que pueda ejecutarse en el dispositivo que interesa a este capítulo.

Figura 204. **Traducción de programa 1 para Cortex-A53 en GCC**

```
    .text
    .global main

main:
// Para enteros de hasta 8 bits o de la forma de segundo operador flexible.
    MOV R1, #255
    MOV R2, #-99
    MOV R3, #0XFF
    MOV R4, #0x3FC0
    MVN R5, #0X3FC


    .align
    .end                                // Final del programa.
```

Fuente: elaboración propia.

Figura 205. **Vista de registros para traducción de programa 1 en Cortex-A53**



Fuente: elaboración propia.

7.3.2. Ejercicio 14

A diferencia de las instrucciones para mover valores; LDR, STR y sus variantes para cargas y almacenamientos múltiples se mantienen con el mismo funcionamiento que en el ejercicio de la sección 6.2.2.

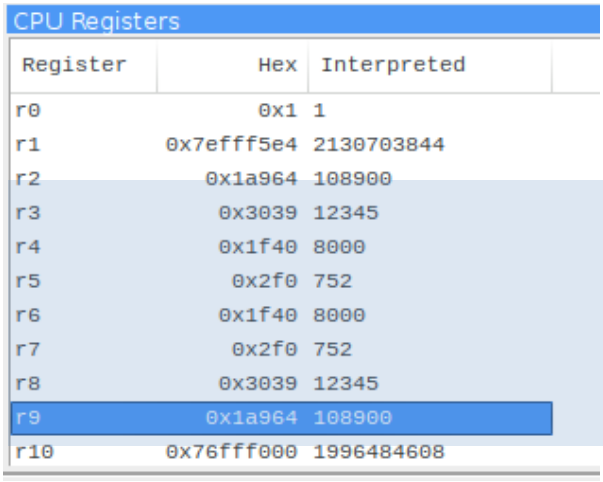
En este caso, se propone el uso de R13 (puntero de pila), para escritura y lectura de valores. Las principales variantes en el código de la figura 206 implicarían el uso de las siglas SP en lugar de R13 para indicar el uso del puntero o el uso de las instrucciones POP y PUSH (sección 6.3).

Figura 206. **Traducción de programa 2 para Cortex-A53 en GCC**

```
    .text
    .global main


main:   // Punto de entrada a partir del archivo Startup.

    // Uso de pseudoinstrucción alternativa LDR Rn, =constante.
    LDR R2, = 108900
    LDR R3, = 12345
    LDR R4, = 8000
    LDR R5, = 752

    /* Almacenamiento de valores a RAM individual con aumento
     de 4 bytes en dirección cada vez.*/
    STR R2, [R13], #4
    STR R3, [R13], #4
    // Almacenamiento colectivo de R4 y R5 en RAM.
    STM R13, {R4, R5}

    // Carga colectiva desde RAM a registros R6, R7 y R8.
    LDM R13!,{R6,R7,R8}
    // Carga individual desde RAM a R9 con la última dirección.
    LDR R9, [R13]

    .align
    .end                            //Final del programa.
```

Fuente: elaboración propia.

Figura 207. **Vista de registros para traducción de programa 2 en Cortex-A53**



Fuente: elaboración propia.

7.4. Aritmética y lógica

El grupo de aritmética y lógica es, por lo general, introducido como el primer paso para comprender cómo manipular la información contenida en el procesador. Su ventaja principal es la correspondencia con la mayoría de operaciones matemáticas y lógicas básicas.

Para Cortex-A53 estas se mantienen con el mismo comportamiento que como se estudió para el perfil de microcontroladores, con excepción de la división (SDIV y UDIV), esta se encuentra solo en formato de punto flotante.

7.4.1. Ejercicio 15

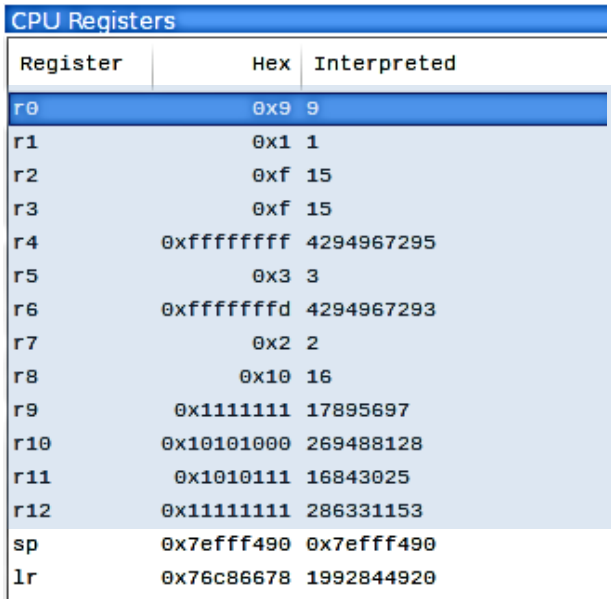
Se recomienda para el ejercicio de la figura 208 la lectura de la sección 6.4, donde se explora no sólo la lista de instrucciones, sino los fundamentos de lo conocido como programación amigable con su mantra *do not harm*.

Figura 208. **Traducción de programa 3 para Cortex-A53 en GCC**

```
    .text
    .global main

main:   // Punto de entrada a partir del archivo Startup.

// Para operaciones aritméticas con resultado de 32 bits.
    LDR     R0, =9
    LDR     R1, =2

    ADD     R2, R0, #6  // R2=R0+6
    SUB     R1, #1      // R1=R1-3
    MUL     R3. R2. R1  // R3=R2*R1
// Para multiplicación con resultado de 64 bits.
    LDR R4, =0xFFFFFFFF
    LDR R5, =0x3
    UMULL   R6, R7, R5, R4  //(R6, R7)=R5*R4

// Para operaciones lógicas.
    LDR R8, =#0x10101010
    LDR R9, =#0x11111101

    AND     R10, R8, R9 // R10=R8&R9
    EOR     R11, R8, R9 // R11=R8^R9
    ORR     R12, R8, R9 // R12=R8|R9
    BIC     R8, R9      // Limpiar en R8 los bits señalado
    ROR     R9, #8      // Rotar R9 8 bits a la derecha.

    .align
    .end
```

Fuente: elaboración propia.

Figura 209. **Vista de registros para traducción de programa 3 en Cortex-A53**



CPU Registers

Register	Hex	Interpreted
r0	0x9	9
r1	0x1	1
r2	0xf	15
r3	0xf	15
r4	0xffffffff	4294967295
r5	0x3	3
r6	0xfffffffd	4294967293
r7	0x2	2
r8	0x10	16
r9	0x1111111	17895697
r10	0x10101000	269488128
r11	0x1010111	16843025
r12	0x11111111	286331153
sp	0x7efff490	0x7efff490
lr	0x76c86678	1992844920

Fuente: elaboración propia.

7.5. Condicionales y subrutinas

Para aprender las bases de la programación por condiciones y saltos entre localidades de memoria es imperativo el estudio de teoría básica para comprender las instrucciones que se enlistan en la tabla LIII. Es necesario recurrir primero a la sección 5.4.13 para el estudio de las banderas de condición en arquitecturas Arm y luego a la Sección 6.5, para explorar cómo estas ayudan a dirigir el flujo de un programa con ayuda de las instrucciones de saltos (*branch*).

7.5.1. Ejercicio 16

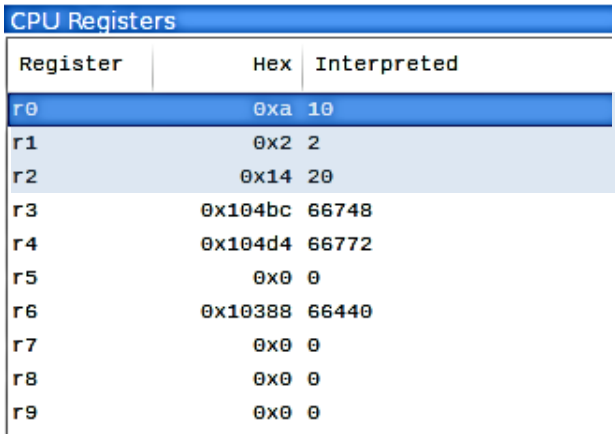
En este ejercicio se evidencia principalmente en comparación con el de la sección 6.5.2 que fue cambiada la instrucción de división por una multiplicación por no estar disponible la primera en el CPU del Cortex-A53. Además, desaparecen las instrucciones CBZ y CBNZ en este perfil y los condicionamientos deben hacerse con combinaciones del resto de operaciones en este grupo y los sufijos.

Figura 210. **Traducción de programa 4 para Cortex-A53 en GCC**

```
    .text
    .global main

main:   // Punto de entrada a partir del archivo Startup.
    // Cargar valores.
    LDR R0, =10
    LDR R1, =2
    // Comparar el contenido de R1 con 0. Si son iguales entonces Z=1
    CMP R1, #0
    BEQ Loop // Saltar a Loop si R1=0.
    MUL R2, R0, R1 // Sólo se llega a esta línea si R1!=0,


Loop:
    B Loop // Ciclo sin salida.

    .align
    .end
```

Fuente: elaboración propia.

Figura 211. **Vista de registros para traducción de programa 4 en Cortex-A53**



Fuente: elaboración propia.

7.5.2. Ejercicio 17

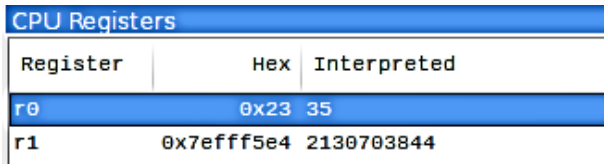
Como se mencionó en el ejercicio anterior, con excepción de CBZ y CBNZ las instrucciones de saltos se mantienen de la misma manera que en el perfil de microcontroladores. El ejercicio de la sección 6.5.3 se mantiene exactamente igual para su ejecución en el perfil de aplicaciones si las directivas son escritas apropiadamente.

Figura 212. **Traducción de programa 5 para Cortex-A53 en GCC**

```
    .text
    .global main

main:   // Punto de entrada a partir del archivo Startup.
    LDR R0, =0

Sumar:
    ADD R0, #5 // Sumar 5 al valor anterior de R0.
    CMP R0, #30 // Comparar R0 con constante 30.
    BGT Loop // Si R0>30 salir del ciclo.
    B Sumar // Se llegará a esta línea sólo si R0<=30
Loop:
    B Loop // Ciclo sin salida

    .align
    .end
```

Fuente: elaboración propia.

Figura 213. **Vista de registros para traducción de programa 5 en Cortex-A53**



Fuente: elaboración propia.

7.5.3. Ejercicio 18

El programa 6 (sección 6.5.4) mostró en el capítulo anterior una forma sencilla de inducir en el programa un retardo. En él fue explicada la razón de calcular una constante para ocupar al procesador cierta cantidad de ciclos

según interese el tiempo; con esta consideración puede saberse que independientemente de la constante que se elija, el funcionamiento será exactamente el mismo y por utilizar instrucciones comunes en las arquitecturas Arm fue necesario solo traducir las directivas para su ejecución en GCC como se muestra en la figura 214.

Figura 214. **Traducción de programa 6 para Cortex-A53 en GCC**

```
    .text
    .global main

main: // Punto de entrada a partir del archivo Startup.
    LDR R0, =0
    LDR R1, =20

Delay:
    ADD R0, #1 // Sumar 1 al valor anterior de R0.
    NOP
    NOP
    NOP
    NOP

    CMP R0, R1 // Comparar R0 con constante.
    BNE Delay // Si R0!=2000000 repetir subrutina Delay.

Loop:
    B Loop // Ciclo sin salida

    .align
    .end
```

Fuente: elaboración propia.

Figura 215. **Vista de registros para traducción de programa 6 en Cortex-A53**

CPU Registers

Register	Hex	Interpreted
r0	0x14	20
r1	0x14	20
r2	0x7efff5ec	2130703852
r3	0x104bc	66748
r4	0x104e4	66788

Fuente: elaboración propia.

Para cálculo correcto de ciclos necesarios para la Raspberry Pi 3B+ (dispositivo que contiene en SoC Broadcom BCM2837), es fundamental recordar que su reloj funciona con frecuencia de 1,4 GHz, que es mucho mayor que los 16 MHz de la Tiva C y para inducir un retardo de un segundo haría falta ocupar al procesador por 1 400 millones de ciclos. Esta es la razón por la que un retardo como el de la figura 214 es una técnica muy poco eficiente y se recomienda sin lugar a duda el uso de periféricos temporizadores; sin embargo, con fines didácticos en el aprendizaje inicial de lenguaje ensamblador es una buena práctica para ejercitar el uso de ciclos.

7.6. Macros

La sección 6.6 describió las macros y sus consideraciones como una ventaja para el programador. El uso de las mismas se mantiene en esta sección con la única excepción de que por ser definidas con directivas, GCC tiene su propia sintaxis para macros y cómo estas se invocan en el código. La forma adecuada se representa a continuación.

Figura 216. **Sintaxis de una macro en GCC**

```
.macro nombre_macro parámetro, parámetro…
//código
.endm
nombre_macro parámetro, parámetro
```

Fuente: elaboración propia.

7.7. Punto flotante

Los ejercicios de esta sección se encuentran con las mismas instrucciones que en la sección 6.7. Es importante considerar que para Cortex-A este grupo es el mismo destinado para uso con la extensión NEON, simplemente se añaden sufijos y prefijos específicos para el funcionamiento sobre vectores y escalares y la exploración de punto flotante sirve de introducción a este tema.

Para los ejercicios 20, 21 y 22 es necesario tener claro el uso del FPSCR y las instrucciones necesarias para el traslado de banderas entre registros de estado y se hace necesaria la consulta de la sección 6.7.1.

7.7.1. Ejercicio 19

El programa 7 en el capítulo anterior fue escrito como el cálculo de dos promedios de valores contenidos en registros del CPU. En este caso, con el fin de mantener el ejemplo, se decidió realizar nuevamente una macro de promedio; el perfil de aplicaciones solo cuenta con división para registros de punto flotante haciendo necesario reescribir el programa con instrucciones VFP (leer sección 6.7), usando las enlistadas en la tabla LIII.

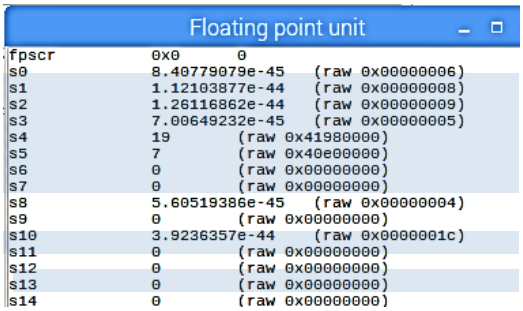
Figura 217. **Traducción de programa 7 para Cortex-A53 en GCC**

```
    .text
    .global main
    .fpu neon-fp-armv8

main:   // Punto de entrada a partir del archivo Startup.
    VLDR.F32 S8, =4

    .macro Promedio  prom, a, b, c, d
// prom = (a + b + c + d)/4
    VADD.F32    S10 , \a, \b
    VADD.F32    S10 , \c
    VADD.F32    S10 , \d
    VDIV.F32 \prom, S10, S8 // dividir en 4
    .endm
    VLDR.F32 S0, =10
    VLDR.F32 S1, =20
    VLDR.F32 S2, =15
    VLDR.F32 S3, =31

Promedio S4, S0, S1, S2, S3

    VLDR.F32 S0, =6
    VLDR.F32 S1, =8
    VLDR.F32 S2, =9
    VLDR.F32 S3, =5
Promedio S5, S0, S1, S2, S3

Loop:
    B Loop

    .align
    .end
```

Fuente: elaboración propia.

Haciendo comparación con la figura 174 puede notarse que es necesario para GCC especificar una nueva directiva: .fpu neon-fp-armv8. Con esto se indica qué hardware de punto flotante contiene el dispositivo objetivo (en este caso Broadcom BCM2837). El resto del programa se desarrolla de forma similar al estudiado en secciones anteriores.

Figura 218. **Vista de registros para traducción de programa 7 en Cortex-A53**



Fuente: elaboración propia.

7.7.2. Ejercicio 20

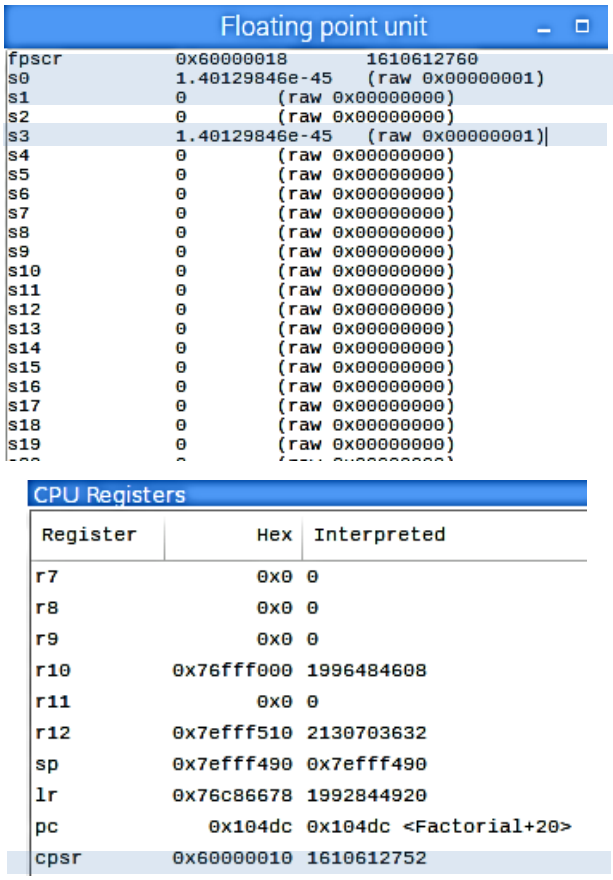
En este ejercicio se evidencia también las observaciones necesarias que el programador debe hacer al trabajar con punto flotante y condicionamiento. El factorial de un numero es calculado con el uso de subrutinas que actúan según el estado de las banderas del FPSCR.

Figura 219. **Traducción de programa 8 para Cortex-A53 en GCC**

```
    .text
    .global main
    .fpu    neon-fp-armv8

main:   // Punto de entrada a partir del archivo Startup.
    VLDR.F32 S0, =30    // Número a operar (n!).
    VLDR.F32 S1, =1     // Registro para guardar resultado.
    VLDR.F32 S30, =1    // Constante 1.

Factorial:
    // Multiplicar el valor actual de n y el resultado acumulado.
    VMUL.F32 S1, S0
    // Sustraer uno al valor actual de n.
    VSUB.F32 S0, S30
    // Verificar si se ha llegado a 1 (ya no quedan números por multiplicar
    VCMP.F32 S0, S30
    VMRS APSR_nzcv, FPSCR   // Trasladar banderas de FPSCR a APSR.
    BNE Factorial   // Si n no es igual a 1, seguir operando.

Loop:
    B Loop

    .align
    .end
```

Fuente: elaboración propia.

Figura 220. **Vista de registros para traducción de programa 8 en Cortex-A53**



Fuente: elaboración propia.

7.7.3. Ejercicio 21

El ejercicio de esta sección se usará para volver a hacer énfasis en la importancia de aprender a imitar el comportamiento de series matemáticas en lenguajes de programación. Especialmente para el de ensamblador constituye una herramienta fundamental para lograr ciertas funciones con el uso de las disponibles en el set de instrucciones de la arquitectura. Para el desarrollo más

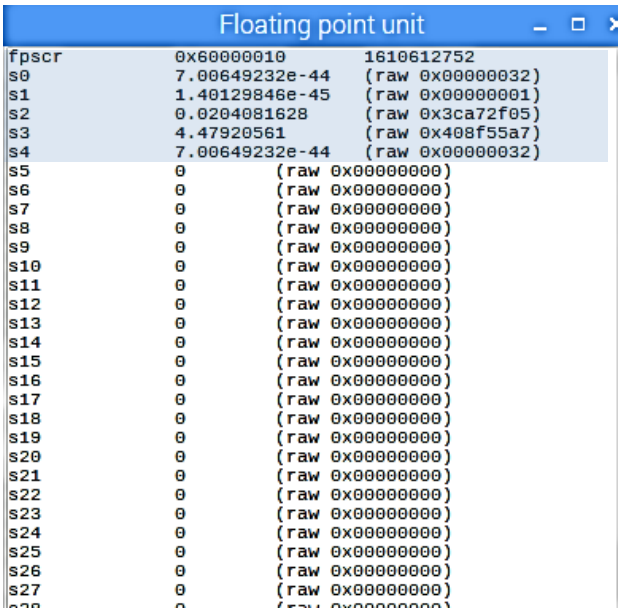
detallado de este tema y del funcionamiento del código en la figura 221 se deberá repasar la sección 6.7.4.

Figura 221. **Traducción de programa 9 para Cortex-A53 en GCC**

```
    .text
    .global main
    .fpu    neon-fp-armv8
    .syntax unified

main:   // Punto de entrada a partir del archivo Startup.
    VLDR.F32 S0, =1     // Contador (n).
    VLDR.F32 S1, =1     // Constante 1.
    VLDR.F32 S4, =50    // Límite superior.

Sumatoria:  // Ciclo de sumatoria.
    VDIV.F32 S2, S1, S0 // S2=1/n
    VADD.F32 S3, S2     // S3 almacena la sumatoria.
    VADD.F32 S0, S1     // Incrementar n.
    // Verificar si n llegó a límite superior.
    VCMP.F32 S0, S4
    VMRS APSR_nzcv, FPSCR
    BLO Sumatoria       // Saltar si es menor (C=0).

    B Loop
Loop:       // Ciclo sin fin.
    B Loop

    .align
    .end
```

Fuente: elaboración propia.

Figura 222. **Vista de registros para traducción de programa 9 en Cortex-A53**



Fuente: elaboración propia.

Resulta interesante comparar el resultado obtenido en S3 para este ejercicio y el contenido en el mismo registro del ejercicio 9 (sección 6.7.5), así se verificará que el objetivo final del programa se alcanza independientemente de la arquitectura en que se ejecute, si la reescritura se hace de forma adecuada y los dispositivos cuentan con los mismos recursos.

7.7.4. Ejercicio 22

Para este ejercicio se considera el programa desarrollado en la sección 6.7.6 con una serie matemática que aproxima el valor de un logaritmo natural. El uso de las instrucciones de salto con enlace (BL y BX), debe apoyarse en el análisis de la sección 6.7.6.1.

Figura 223. **Traducción de programa 10 para Cortex-A53 en GCC**

```
    .text
    .global main
    .fpu    neon-fp-armv8

main:   // Punto de entrada a partir del archivo Startup.
    VLDR.F32 S0, =8     // Valor de x.
    VLDR.F32 S1, =50    // Límite superior.

    VLDR.F32 S2, =1     // Constante 1.
    VLDR.F32 S3, =2     // Constante 2.

    VLDR.F32 S4, =-1    // Valor de n.

Factor_K:   // Cálculo de K en S13.
    VMUL.F32 S10, S0, S0    // x^2
    VSUB.F32 S11, S10, S2   // (x^2)-1
    VADD.F32 S12, S10, S2   // (x^2)+1
    VDIV.F32 S13, S11, S12  // K=((x^2)-1)/((x^2)+1)

Sumatoria:  // Ciclo de sumatoria.
    VADD.F32 S4, S2     // n+1
    VMUL.F32 S20, S3, S4    // 2*n
    VADD.F32 S20, S2    // 2n+1
    VDIV.F32 S21, S2, S20   // a=1/(2n+1)
    VLDR.F32 S22, =0    // Contador subrutina Potencia.
    VLDR.F32 S23, =1    // S23 contendrá el valor de la potencia.
    BL Potencia         // b=K^(2n+1), BL: "branch with link".
    VMUL.F32 S24, S23, S21  //a*b
    VADD.F32 S30, S24   // Acumulado de sumatoria en S30.
    VCMP.F32 S4, S1     // Verificar si n ha llegado a límite
    VMRS APSR_nzcv, FPSCR
    BLO Sumatoria       // Sufijo LO significa "menor".

Loop:       // Ciclo infinito.
    B Loop

Potencia:   // Subrutina para cálculo de potencia.
    VMUL.F32 S23, S13
    VADD.F32 S22, S2
    VCMP.F32 S22, S20   // Verificar si contador es igual a exp.
    VMRS APSR_nzcv, FPSCR
    BNE Potencia        // Si no se ha terminado, continuar
    BX LR   // Volver a línea siguiente a BL usado.

    .align
    .end
```

Fuente: elaboración propia.

7.8. Los SoC, C y el lenguaje de ensamblador

Por lo general, los SoC son diseñados e implementados con el fin de ser lo más eficientes posible dentro de las limitaciones propias de cada uno (tamaño, consumo de potencia, frecuencia de reloj). Según esta premisa y los beneficios que implica el uso de lenguaje ensamblador como se estudió en el capítulo 5, sería intuitivo suponer que la mejor forma de escribir código para ejecutarse en estos chips es siempre en lenguaje de ensamblador; sin embargo, esta no es la ruta óptima en todos los casos.

El tiempo que el programador tarde en realizar un programa y la simplicidad con que esto pueda lograrse también es un factor a tomar en cuenta, en especial para situaciones en que no sólo esté en proceso el diseño de software, sino también de hardware. Esta es la razón de que sea ampliamente sugerido el uso de lenguajes de nivel un poco más elevado al de ensamblador con el fin de que sea suficientemente sencillo para el humano y así la programación no se vuelva una tarea excesivamente tediosa, siendo el caso de lenguajes como C y C++ que cuentan con compiladores que buscan optimizar el código (opción que es ampliamente sugerida por los mismos fabricantes de dispositivos). Esto no excluye la importancia de los lenguajes de ensamblador: existen situaciones en que es sumamente necesario escribir secciones de un programa al nivel más bajo posible.

La ventaja en el uso del proceso general de compilación (figura 203), es que el programador tiene acceso a sus distintas etapas, con esto en mente el procedimiento sugerido para generación de código eficiente se resume en la figura 224 a continuación.

Figura 224. **Proceso sugerido de programación óptima**

Fuente: elaboración propia.

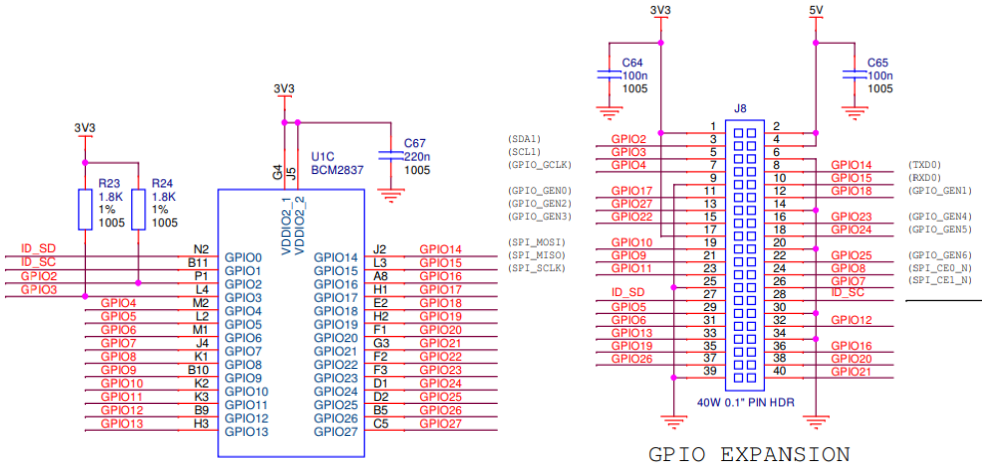
Notar que en el primer rectángulo se propone la escritura en un lenguaje de nivel medio, esta notación sugiere que debe utilizarse un lenguaje suficientemente alto para ser comprendido con facilidad por el humano pero sin llegar a encontrarse en el grupo de los de muy alto nivel (el caso de Python, C#, Matlab y Perl), como para que su nivel de abstracción sea tan alto que dificulte a la máquina crear una salida eficiente en ensamblador. Esta es la razón de que el lenguaje C sea el predilecto en el diseño de sistemas embebidos,

extendiéndose sus beneficios hasta el campo de descripción de hardware con la llamada síntesis de alto nivel (HLS).

7.8.1. Wiring Pi

Uno de los intereses más grandes en el aprendizaje de sistemas embebidos (como se evidenció en el capítulo 6), es el control de sus periféricos. A pesar de que esto es un tema muy específico a la arquitectura de la tarjeta de desarrollo que se utilice, la comprensión básica de cómo funcionan los principios facilita el estudio con ayuda de documentación para el chip objetivo. Esta tarea usualmente se vuelve mucho más sencilla con el uso de librerías escritas con definición de funciones de uso común (encendido de pines, declaración de dirección, conexión a periféricos, sincronización).

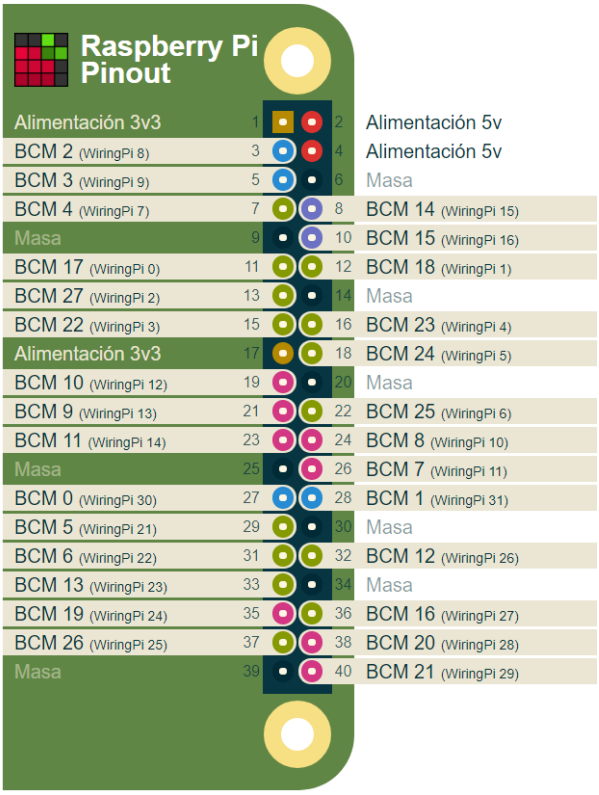
Figura 225. **Distribución de pines en tarjeta Raspberry Pi 3B+**



Fuente: Schematics. https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/rpi_SCH_3b_1p2_reduced.pdf. Consulta: 20 de diciembre de 2018.

Como forma sencilla de programación de pines de propósito general, la librería Wiring Pi fue escrita en lenguaje C para implementación en tarjetas de la familia Raspberry Pi. Wiring Pi tiene su propio sistema de numeración de los pines en la tarjeta especificado entre paréntesis en figura 226. Esto se especifica en el código con la directiva #define indicando la conexión entre el nombre que se dará en el programa al pin y el número según el orden de la librería. La facilidad con que esto se logra es la razón principal de que esta librería se volviera muy solicitada para el diseño de sistemas embebidos sencillos (especialmente para principiantes y aficionados).

Figura 226. **Numeración de pines según Wiring Pi**



Fuente: *Raspberry Pi Wiring Pi.* https://es.pinout.xyz/pinout/wiringpi. Consulta: 20 de diciembre de 2018.

Sin adentrarse demasiado en el campo de la programación en C (fuera del alcance de este texto), la figura 227 propone un ejemplo sencillo de encendido y apagado de un pin en la Raspberry Pi 3B+ a 0,5 Hz. El programa luego de la descarga de la librería al sistema de interés, se desarrolla en pasos:

- Importación de librerías
- Definición de nombre led para el pin 0 de Wiring Pi (figura 226)
- Especificación de led como salida
- Apagado de pin
- Retardo de mil milésimas de segundo
- Encendido de pin
- Retardo de mil milésimas de segundo
- Repetición de los últimos cuatro pasos en un ciclo infinito

Por el proceso sugerido en la figura 224 correspondería luego de escribir el programa, afinar los detalles accediendo al código generado por la máquina en lenguaje de ensamblador.

Figura 227. **Programa de control de periféricos en Raspberry Pi por lenguaje C**

```
#include <wiringPi.h>
#include <stdio.h>

#define LED 0

int main(void) {
	pinMode(LED, OUTPUT);
	while(1) {
		digitalWrite(LED, LOW);   // Apagar pin.
		delay(1000);              // Esperar 1 segundo.
		digitalWrite(LED, HIGH);  // Encender pin.
		delay(1000);              // Esperar 1 segundo.
	}
	return 0;
}
```

Fuente: elaboración propia.

7.9. El set de instrucciones A64

En el capítulo 4 fue expuesta la composición y características de los procesadores Cortex-A Armv8, siendo una de las más importantes que estos soportan procesamiento de 64 bits. Para su mayor aprovechamiento, una arquitectura de este tipo se utiliza con el set de instrucciones adecuado: el A64 para el estado AArch64.

Como se habrá notado hasta este punto, el aprendizaje de cualquier lenguaje de ensamblador está íntimamente ligado a las especificaciones del set de instrucciones que libere el diseñador de la arquitectura de interés. En el caso de Arm, se especifican los lineamientos iniciales de comparación con A32 para aquel que desee aventurarse en el campo de A64:

- Los registros de 64 bits se especifican con Xn donde n es un valor de 0 a 30. En contraste con la R de Armv7.
- Los registros de 32 bits se especifican con una W.
- Los registros de FPU se especifican con una V.
- Las banderas de condición se procesan sobre los 32 bits más bajos de un resultado en caso se seleccione un registro en forma de 32 bits.
- El puntero de pila es llamado WSP.
- Los sufijos para condicionamiento son separados del mnemónico por un punto (e.g. B.EQ, B.NE).

En SIMD algunos detalles son dados para el uso de escalares en la forma Bn, Hn, Sn, Dn, Qn y vectores Vn.8B, Vn.2S, Vn.1D, entre otros. donde n es algún número en el rango de 0 a 31 y las letras representan grupos de 8, 16, 32, 64 y 128 bits.

El uso de A64 provee, en general para un procesador en estado AArch64, acceso a datos, punteros y registros de 64 bits de ancho; esto con la facilidad de conmutar fácilmente al set Arm oiriginal (A32), o Thumb (T32). Se hace énfasis por último, en la recomendación de uso de documentos liberados por Arm para la comprensión de Armv8, sus estados e ISA como son *ARMv8 Instruction Set Overview Architecture Group* y The A64 instruction set version 1.0.

CONCLUSIONES

1. La presencia extendida de tecnología Arm y el legado construido hasta ahora son factores de peso para que los profesionales en área de electrónica se dediquen al estudio profundo de sus especificaciones, y las de los dispositivos que se derivan de esta arquitectura.

2. Los perfiles Cortex-M y Cortex-A son ramas de la arquitectura Arm con ventajas y objetivos específicos que resultan igual de importantes para la tecnología como se conoce en la actualidad. Por esta razón es ideal la situación en que un estudiante de pregrado en el área se adentre por igual en ambos análisis.

3. El perfil Cortex-M resulta conveniente como tema de introducción al estudio de procesadores Arm por su simplicidad frente a otros perfiles y la accesibilidad económica a los estudiantes.

4. El perfil Cortex-A tiene especificaciones de alto rendimiento que lo hacen óptimo para múltiples aplicaciones móviles actuales en diversos campos. El conocimiento de estos procesadores abre la puerta a un sinfín de sistemas mucho más sofisticados que los basados en microcontroladores, y su estudio beneficiaría a los profesionales en electrónica en innumerables escenarios.

5. A pesar del aparente desuso del lenguaje de ensamblador, son múltiples las aplicaciones actuales en que es necesario recurrir a este para la creación de sistemas eficientes y a pesar de su relativa complejidad

respecto a lenguajes de alto nivel, los procesadores Arm comparten el núcleo de sus sets de instrucciones, el aprendizaje de un set específico solo servirá de ambientación para cualquier otro.

6. Es imperativo el estudio de una arquitectura de procesador para el uso correcto de su set de instrucciones en lenguaje de ensamblador debido a que este sólo está un nivel sobre el de máquina.

7. Al momento de aprender lenguaje de ensamblador dirigido a programación de procesadores Arm, el perfil Cortex-M sirve como introducción a Cortex-A.

RECOMENDACIONES

1. El lenguaje de ensamblador por lo general requiere más tiempo de aprendizaje que otros lenguajes; sin embargo el programador iniciante debe considerar su esfuerzo como una inversión de tiempo porque los principios del lenguaje se mantienen a pesar de que el set de instrucciones cambie. Es decir el tiempo de aprendizaje nunca será pérdida sino la base para otros temas.

2. A pesar de que la importancia del lenguaje de ensamblador se recalca repetidas veces en este texto, no debe olvidarse la importancia de otros lenguajes de programación para el diseño de sistemas eficientes.

3. Cortex-M es el perfil adecuado para comenzar el estudio si se desea aprender acerca de la arquitectura Arm. Su simplicidad relativa a nivel de set de instrucciones e implementación facilita reconocer los elementos esenciales de cada uno. Luego de dominar este perfil, se puede pasar a Cortex-A con facilidad.

4. Debe darse especial atención a la comprensión del hardware que compone una arquitectura con el fin de comprender adecuadamente qué efecto tiene el lenguaje de ensamblador sobre él y el manejo mucho más natural.

5. Por facilidad en el hallazgo de soporte es recomendado el uso prioritario de herramientas desarrolladas por o específicamente para Arm sobre

otras opciones, aunque el programador es libre de buscar su comodidad o familiaridad con ciertas líneas.

6. Es siempre importante practicar la búsqueda de información específica en documentación oficial liberada por el fabricante del dispositivo que se trabaja o el diseñador de la arquitectura, para asegurar que las fuentes son confiables y la información precisa.

7. Para el programador que maneja a cierto nivel de entendimiento el idioma inglés resulta relativamente sencillo consultar documentación oficial de fabricantes y diseñadores (dado que este es su idioma principal de publicación). Por otro lado se aconseja al programador que tenga desconocimiento del vocabulario básico en el área, que dedique tiempo a este aspecto para beneficiarse de la misma manera.

8. Los desarrolladores de software y hardware que trabajen de alguna forma con tecnología Arm son fuertemente alentados a publicar sus hallazgos, resultados, experiencia y enseñanzas en su idioma materno (en este caso, el español), para contribuir a la formación de una comunidad informada en todas las regiones geográficas.

BIBLIOGRAFÍA

1. ABDALLAH Nizar. *FPGA Technology & FPGA Design. Advanced School on Programmable System-on-Chip for Scientific Instrumentation*. Italia: Escuela de invierno llevada a cabo en International Centre for Theoretical Physics de Trieste, 2017. 258 p.

2. _____. *The Era of SoC FPGAs. Advanced School on Programmable System-on-Chip for Scientific Instrumentation*. Italia: Escuela de invierno llevada a cabo en International Centre for Theoretical Physics de Trieste, 2017. 345 p.

3. ARM Community. *A Brief History of ARM: Part 1*. [en línea]. <https:// community.arm.com / processors / b / blog / posts / a-brief-history-of-arm-part-1>. [Consulta: 28 de febrero de 2018].

4. _____. *A Brief History of ARM: Part 2*. [en línea]. <https:// community.arm.com / processors / b / blog / posts / a-brief-history-of-arm-part-2>. [Consulta: 28 de febrero de 2018].

5. _____. *ARM Cortex-A17 / Cortex-A12 processor update.* [en línea]. <https://community.arm.com/processors/b/blog/posts/arm-cortex-a17-cortex-a12-processor-update>. [Consulta: 11 de mayo de 2018].

6. Arm Ltd. *Application processors for embedded applications*. Cambridge, Inglaterra: Arm Ltd, 2013. 229 p.

7. _____. *Architecture Reference Manual ARMv7-A and ARMv7-R edition Errata markup*. Cambridge, Inglaterra: Arm Ltd, 2011. 2158 p.

8. _____. *ARM Architecture Reference Manual ARMv8, for ARMv8-A architecture profile*. Cambridge, Inglaterra: Arm Ltd, 2017. 6666 p.

9. _____. *ARM Cortex-A Series Programmer's Guide for ARMv8-A Version 1.0*. Cambridge, Inglaterra: Arm Ltd, 2015. 296 p.

10. _____. *ARM Cortex-A Series Programmer's Guide Version 4.0*. Cambridge, Inglaterra: Arm Ltd, 2013. 421 p.

11. _____. *Cortex-M0 Devices Generic User Guide*. Cambridge, Inglaterra: Arm Ltd, 2009. 140 p.

12. _____. *Cortex-M0+ Devices Generic User Guide*. Cambridge, Inglaterra: Arm Ltd, 2012. 113 p.

13. _____. *Cortex-M1 Technical Reference Manual Revision: r0p1*. Cambridge, Inglaterra: Arm Ltd, 2008. 234 p.

14. _____. Cortex-M3 Devices Generic User Guide. Cambridge, Inglaterra: Arm Ltd, 2010. 180 p.

15. _____. *Cortex-M4 Devices Generic User Guide*. Cambridge, Inglaterra: Arm Ltd, 2010. 276 p.

16. _____. *Cortex-M7 Devices Generic User Guide*. Cambridge, Inglaterra: Arm Ltd, 2015. 315 p.

17. _____. *Cortex-M23 Devices Generic User Guide Revision: r1p0*. Cambridge, Inglaterra: Arm Ltd, 2016. 70 p.

18. _____. *Cortex-M33 Devices Generic User Guide Revision: r0p3*. Cambridge, Inglaterra: Arm Lt., 2017. 339 p.

19. _____. *Cortex-A5 Revision:r0p1 Technical Reference Manual*. Cambridge, Inglaterra: Arm Lt., 2016. 247 p.

20. _____. *Cortex-A7 MPCore Revision:r0p5 Technical Reference Manual*. Cambridge, Inglaterra: Arm Lt., 2013. 269 p.

21. _____. *Cortex-A8 Revision:r3p2 Technical Reference Manual*. Cambridge, Inglaterra: Arm Ltd, 2010. 580 p.

22. _____. *Cortex-A9 Revision:r4p1 Technical Reference Manual*. Cambridge, Inglaterra: Arm Ltd, 2016. 217 p.

23. _____. *Cortex-A15 MPCore Revision:r4p0 Technical Reference Manual*. Cambridge, Inglaterra: Arm Ltd, 2013. 392 p.

24. _____. *Cortex-A17 MPCore Revision:r1p1 Technical Reference Manual*. Cambridge, Inglaterra: Arm Ltd, 2014. 321 p.

25. _____. *ARM Cortex-A32 Processor Revision:r0p1 Technical Reference Manual*. Cambridge, Inglaterra: Arm Ltd, 2017. 642 p.

26. _____. *ARM Cortex-A35 Processor Revision:r0p2 Technical Reference Manual*. Cambridge, Inglaterra: Arm Ltd, 2017. 858 p.

27. _____. *Cortex-A53 MPCore Revision:r0p4 Technical Reference Manual*. Cambridge, Inglaterra: Arm Ltd, 2016. 620 p.

28. _____. *ARM Cortex-A55 Processor Revision:r1p0 Technical Reference Manual*. Cambridge, Inglaterra: Arm Ltd, 2017. 768 p.

29. _____. *Cortex-A57 MPCore Revision:r1p3 Technical Reference Manual*. Cambridge, Inglaterra: Arm Ltd, 2016. 577 p.

30. _____. *Cortex-A72 MPCore Revision:r0p3 Technical Reference Manual*. Cambridge, Inglaterra: Arm Ltd, 2016. 575 p.

31. _____. *Cortex-A73 MPCore Revision:r0p2 Technical Reference Manual*. Cambridge, Inglaterra: Arm Ltd, 2016. 596 p.

32. _____. *ARM Cortex-A75 Processor Revision:r3p0 Technical Reference Manual*. Cambridge, Inglaterra: Arm Ltd, 2018. 800 p.

33. _____. *Arm Compiler Version 6.10 armasm User Guide*. Cambridge, Inglaterra: Arm Ltd, 2018. 1790 p.

34. _____. *Arm Compiler Version 5.06 armasm User Guide*. Cambridge, Inglaterra: Arm Ltd, 2016. 852 p.

35. _____. *The A64 instruction set version 1.0*. Cambridge, Inglaterra: Arm Ltd, 2017. 335 p.

36. _____. *ARMv8 Instruction Set Overview Architecture Group*. Cambridge, Inglaterra: Arm Ltd, 2011. 112 p.

37. _____. *Arm Cortex-M Series Processors*. [en línea]. <https:// www.arm.com/products/processors/cortex-m>. [Consulta: 25 de marzo de 2018].

38. _____. *Cortex-M0*. [en línea]. <https://developer.arm.com/ products/processors/cortex-m/cortex-m0>. [Consulta: 25 de marzo de 2018].

39. _____. *Cortex-M0+*. [en línea]. <https://developer.arm.com/ products/processors/cortex-m/cortex-m0+>. [Consulta: 25 de marzo de 2018].

40. _____. Cortex-M3. [en línea]. <https://developer.arm.com/ products/processors/cortex-m/cortex-m3>. [Consulta: 25 de marzo de 2018].

41. _____. *Cortex-M4*. [en línea]. <https://developer.arm.com/ products/processors/cortex-m/cortex-m4>. [Consulta: 25 de marzo de 2018].

42. _____. Cortex-M7. [en línea]. <https://developer.arm.com /products/processors/cortex-m/cortex-m7>. [Consulta: 25 de marzo de 2018].

43. _____. *Cortex-M23*. [en línea]. <https://developer.arm.com/products/processors/cortex-m/cortex-m23>. [Consulta: 25 de marzo de 2018].

44. _____. *Cortex-M33*. [en línea]. <https://developer.arm.com/products/processors/cortex-m/cortex-m33>. [Consulta: 25 de marzo de 2018].

45. _____. *Cortex-M35P*. [en línea]. <https://developer.arm.com/products/processors/cortex-m/cortex-m35p>. [Consulta: 14 de septiembre de 2018].

46. _____. *Security on Arm: TrustZone*. [en línea]. <https://www.arm.com/products/security-on-arm/trustzone>. [Consulta: 8 de abril de 2018].

47. _____. Security on Arm: TrustZone for Armv8-M. [en línea]. <https://community.arm.com/processors/trustzone-for-armv8-m/>. [Consulta: 8 de abril de 2018].

48. _____. *Processor Cortex-A Series*. [en línea]. <https://www.arm.com/products/processors/cortex-a>. [Consulta: 14 de abril de 2018].

49. _____. *Arm Cortex-A Series Processors*. [en línea]. <https://developer.arm.com/products/processors/cortex-a>. [Consulta: 14 de abril de 2018].

50. _____. *Arm Architecture – Armv8.2-A evolution and delivery*. [en línea]. <https://community.arm.com/processors/b/blog/posts/arm-architecture-armv8-2-a-evolution-and-delivery>. [Consulta: 14 de abril de 2018].

51. _____. *Cortex-A5.* [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a5>. [Consulta: 15 de junio de 2018].

52. _____. *Cortex-A7.* [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a7>. [Consulta: 15 de junio de 2018].

53. _____. *Cortex-A8.* [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a8>. [Consulta: 15 de junio de 2018].

54. _____. *Cortex-A9.* [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a9>. [Consulta: 15 de junio de 2018].

55. _____. *Cortex-A15.* [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a15>. [Consulta: 15 de junio de 2018].

56. _____. *Cortex-A17*. [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a17>. [Consulta: 15 de junio de 2018].

57. _____. *Cortex-A32*. [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a32>. [Consulta: 15 de junio de 2018].

58. _____. *Cortex-A35*. [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a35>. [Consulta: 15 de junio de 2018].

59. _____. *Cortex-A53*. [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a53>. [Consulta: 15 de junio de 2018].

60. _____. *Cortex-A55*. [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a55>. [Consulta: 15 de junio de 2018].

61. _____. *Cortex-A57*. [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a537>. [Consulta: 15 de junio de 2018].

62. _____. Cortex-A65AE. [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a65ae>. [Consulta: 20 de diciembre de 2018].

63. _____. *Cortex-A76AE*. [en línea]. <https://www.arm.com/products/silicon-ip-cpu/cortex-a/cortex-a76ae>. [Consulta: 20 de diciembre de 2018].

64. _____. *Cortex-A72*. [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a572>. [Consulta: 15 de junio de 2018].

65. _____. *Cortex-A73*. [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a73>. [Consulta: 15 de junio de 2018].

66. _____. *Cortex-A75*. [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a75>. [Consulta: 15 de junio de 2018].

67. _____. *Cortex-A76*. [en línea]. <https://developer.arm.com/products/processors/cortex-a/cortex-a76>. [Consulta: 15 de junio de 2018].

68. _____. *Arm DynamIQ technology*. [en línea]. <https://developer.arm.com/technologies/dynamiq>. [Consulta: 15 de junio de 2018].

69. BARRIENTOS ROJAS, David Josué. *Desarrollo del curso Introducción al diseño de sistemas embebidos, utilizando el controlador TM4C123GH6PM como actualización del laboratorio de microcontroladores*. Trabajo de graduación de Ing. Electrónica. Universidad de San Carlos de Guatemala. Facultad de Ingeniería. 2017. 128 p.

70. CARDOSO GUIMARÃES, Célio. *GNU ARM Assembler Quick Reference*. [en línea]. <http://www.ic.unicamp.br/~celio/mc404-

2014/docs/gnu-arm-directives.pdf>. [Consulta: 11 de diciembre de 2018].

71. Cleverism. *Assembly Language*. [en línea]. <https://www.cleverism.com/skills-and-tools/assembly-language/>. [Consulta: 8 de junio de 2018].

72. CROCKETT LOUISE, Elliot Ross; ENDERWITZ MARTIN, Stewart Robert. *The Zynq Book Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC*. Escocia, Reino Unido: Department of Electronic and Electrical Engineering University of Strathclyde Glasgow, 2014. 484 p.

73. EcuRed. *Code Blocks*. [en línea]. <https://www.ecured.cu/Code_Blocks>. [Consulta: 20 de diciembre de 2018].

74. _____. *Lenguaje ensamblador*. [en línea]. <https://www.ecured.cu/Lenguaje_ensamblador>. [Consulta: 8 de junio de 2018].

75. ELPROCUS. ¿*Why ARM Is Most Popular*? [en línea]. <https://www.elprocus.com/arm-architecture/>. [Consulta: 5 de marzo de 2018].

76. FERRARIO, Nicolás. *Organización de computadoras: Clase 7*. Buenos Aires, Argentina: Facultad de Informática, Universidad Nacional de la Plata, 2018. 50 p.

77. FURBER, Stephen. *ARM System-on-Chip Architecture*. Harlow, Inglaterra: Addison-Wesley, 2000. 1156 p.

78. GONZÁLEZ, Ramón. *Lenguaje ensamblador*. [en línea]. <http://ramon-gzz.blogspot.com/2012/04/lenguaje-ensamblador.html>. [Consulta: 8 de junio de 2018].

79. GOODACRE, John. *White Paper Technology Preview: The ARMv8 Architecture*. Cambridge, Inglaterra: Arm Ltd, 2011. 310 p.

80. Groupe Figaro, CCM Benchmark. *¿Qué es un bus informático*?. [en línea]. <http://es.ccm.net/contents/364-que-es-un-bus-informatico>. [Consulta: 11 de octubre de 2018].

81. HOHL, William; HINDS, Christopher. *ARM Assembly Language Fundamentals and Techniques.* 2a ed. Florida, Estados Unidos: CRC Press Taylor & Francis Group, 2015. 448 p.

82. HUERTA, Rodrigo. *Lectura 4: representación de números en punto fijo y punto flotante*. Valparaíso, Chile: Universidad Técnica Federico Santa María, 2003. 248 p.

83. LLAMAS, Luis. *¿Qué es Raspberry Pi?*. [en línea]. <https://www.luisllamas.es/que-es-raspberry-pi/>. [Consulta: 11 de diciembre de 2018].

84. IoTAgenda. *Definition: Embedded System*. [en línea]. <https://internetofthingsagenda.techtarget.com/definition/embedded-system>. [Consulta: 8 de abril de 2018].

85. MAURER, Peter. *Chapter 5: Component-Level Programming*. Michigan, Estados Unidos: Prentice Hall, 2003. 415 p.

86. Microcontrollers Tips. *What are compilers, translators, interpreters and assemblers?*. [en línea]. <https://www.microcontrollertips.com/compilers-translators-interpreters-assemblers/>. [Consulta: 20 de septiembre de 2018].

87. MILLMAN, Jacob. *Microelectronics: digital and analog circuits and systems*. Michigan, Estados Unidos: McGraw-Hill. 1979. 501 p.

88. MORÓN, José. *Señales y sistemas*. Maracaibo, Venezuela: Universidad Rafael Urdaneta, Vereda del Lago, 2011. 318 p.

89. MuyComputer. *qARM presenta el procesador más eficiente energéticamente del mundo Cortex-M0+*. [en línea]. <https://www.muycomputer.com/2012/03/13/arm-presenta-el-procesador-mas-eficiente-energeticamente-del-mundo-cortex-m0/>. [Consulta: 12 de marzo de 2018].

90. NOERIGA, Sergio. *Apuntes de clases: representación de números binarios en punto fijo y punto flotante*. Buenos Aires, Argentina: Universidad Nacional de La Plata, 2003. 188 p.

91. O mundo em um clique. *A História dos Microcontroladores*. [en línea]. <http://oincrivelmundonerd.blogspot.com/2014/03/a-historia-dos-microcontroladores.html>. [Consulta: 12 de marzo de 2018].

92. PLANTZ, Robert G. *Introduction to Computer Organization: ARM Assembly Language Using the Raspberry Pi*. [en línea]. <https://bob.cs.sonoma.edu/IntroCompOrg-RPi/frontmatter-1.html>. [Consulta: 11 de diciembre de 2018].

93. RAMÍREZ MILIÁN, Oscar Rolando. *Introducción a los sistemas físico-cibernéticos como actualización en las prácticas de laboratorio del curso de sistemas de control*. Trabajo de graduación de Ing. Electrónica. Universidad de San Carlos de Guatemala, Facultad de Ingeniería. 2016. 546 p.

94. Raspberry Pi Foundation. *BCM2837B0*. [en línea]. <https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837b0/README.md>. [Consulta: 11 de diciembre de 2018].

95. Raspberry Pi Foundation. *GPIO*. [en línea]. <https://www.raspberrypi.org/documentation/usage/gpio/README.md>. [Consulta: 11 de diciembre de 2018].

96. Re Ejected. *GNU AS ARM Reference V2*. [en línea]. <http://re-eject.gbadev.org/files/GasARMRef.pd>f. [Consulta: 11 de diciembre de 2018].

97. RUSLING David. *ARMv7A Architecture Overview*. Cambridge, Inglaterra: Arm Ltd, 2010. 327 p.

98. RYZHYK, Leonid. *The ARM Architecture*. [en línea]. <https://www.researchgate.net/publication/228392292_The_ARM_Architecture>. [Consulta: 5 de marzo de 2018].

99. SEAL, David. *ARM Architecture Reference Manual*. Cambridge, Inglaterra: Addison-Wesley, 2005. 1138 p.

100. _____. *ARMv7-M Architecture Reference Manual*. Cambridge, Inglaterra: Addison-Wesley, 2010. 716 p.

101. _____. *ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition*. Cambridge, Inglaterra: Addison-Wesley, 2011. 2158 p.

102. Semiwiki.com. *A Brief History of ARM Holdings*. [en línea]. <https://www.semiwiki.com/forum/content/2791-brief-history-arm-holdings.html>. [Consulta: 3 de marzo de 2018].

103. SHORE, Chris. *ARMv8-A CPU Architecture Overview*. Londres, Inglaterra: Arm Ltd., 2015. 302 p.

104. Techlandia. *La historia del microcontrolador*. [en línea]. <https://techlandia.com/historia-del-microcontrolador-info_516984/>. [Consulta: 12 de marzo de 2018].

105. The Telegraph Media Group Limited. *History of ARM: from Acorn to Apple*. [en línea]. <https://www.telegraph.co.uk/finance/newsbysector/epic/arm/8243162/History-of-ARM-from-Acorn-to-Apple.html>. [Consulta: 28 de febrero de 2018].

106. Texas Instruments. *ARM Cortex-M4F Based MCU TM4C123G LaunchPad Evaluation Kit*. [en línea]. <http://www.ti.com/tool/EK-TM4C123GXL>. [Consulta: 11 de diciembre de 2018].

107. Tutorialspoint. *Embedded Systems - Overview*. [en línea]. <https://www.tutorialspoint.com/embedded_systems/es_overview.htm>. [Consulta: 8 de abril de 2018].

108. Universidad de Tarapacá. *Conceptos básicos de Señales y Sistemas*. [en línea]. <http://chitita.uta.cl/cursos/2012-2/0000435/recursos/r-1.pdf>. [Consulta: 20 de septiembre de 2018].

109. Universidad Internacional de Valencia. *Conociendo el lenguaje de máquina*. [en línea]. <https://www.universidadviu.com/conociendo-lenguaje-maquina/>. [Consulta: 8 de junio de 2018].

110. Universitat Politècnica de València. *Raspberry Pi*. [en línea]. <http://histinf.blogs.upv.es/2013/12/18/raspberry-pi/>. [Consulta: 11 de diciembre de 2018].

111. VALVANO, Jonathan. *Introduction to ARM Cortex-M Microcontrollers Volume 1*. 5a ed. Texas, Estados Unidos: 2014. 594 p.

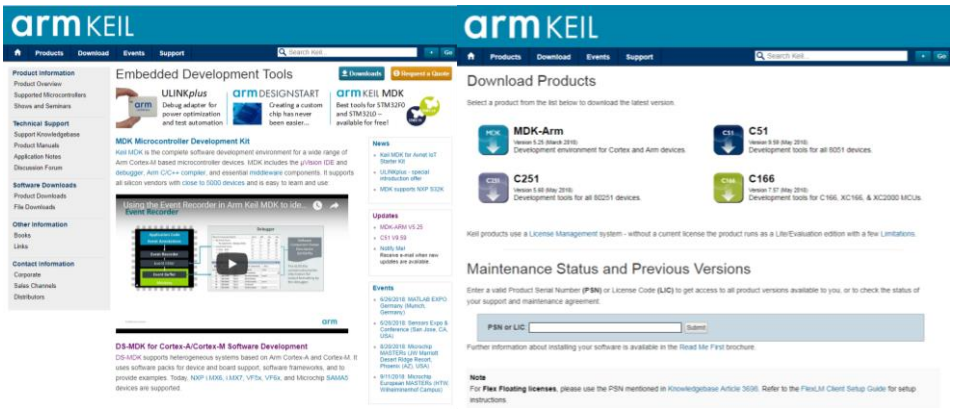
112. YIU, Joseph. *White Paper: ARM Cortex-M for Beginners. An overview of the ARM Cortex-M processor family and comparison*. Cambridge, Inglaterra: Arm Ltd, 2017. 289 p.

ANEXOS

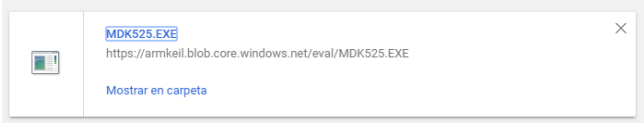
Anexo 1. **Instalación de Keil uVision 5**

La siguiente es una guía rápida en la instalación del kit necesario para desarrollar el capítulo 6.

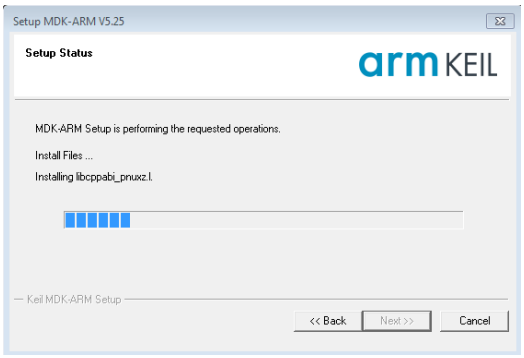
El primer paso consiste en obtener el software de la página de Arm Keil (www.keil.com). Aquí se buscará la versión más reciente del MDK (*Microcontroller Development Kit*), en este caso la 5.



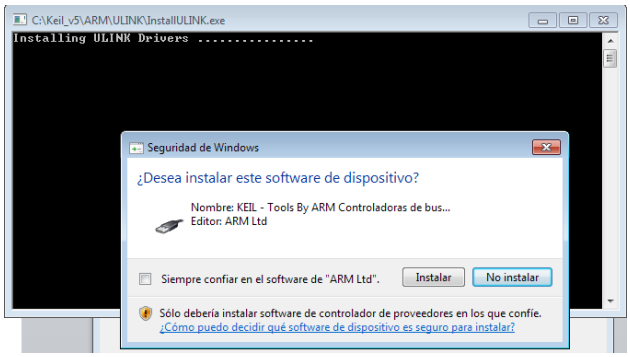
Dentro de la página de descargas se encontrará el instalador ejecutable, al tenerlo en el equipo de interés el proceso será bastante simple, porque consistirá simplemente en seguir el asistente hasta completar el proceso.



Continuación del anexo 1.

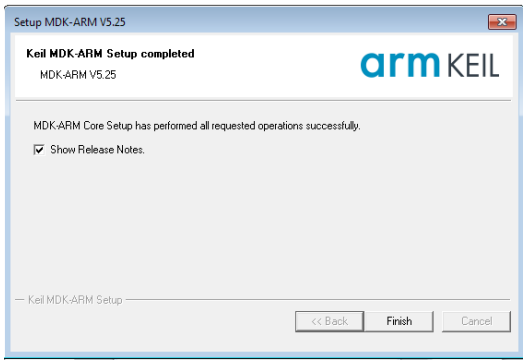


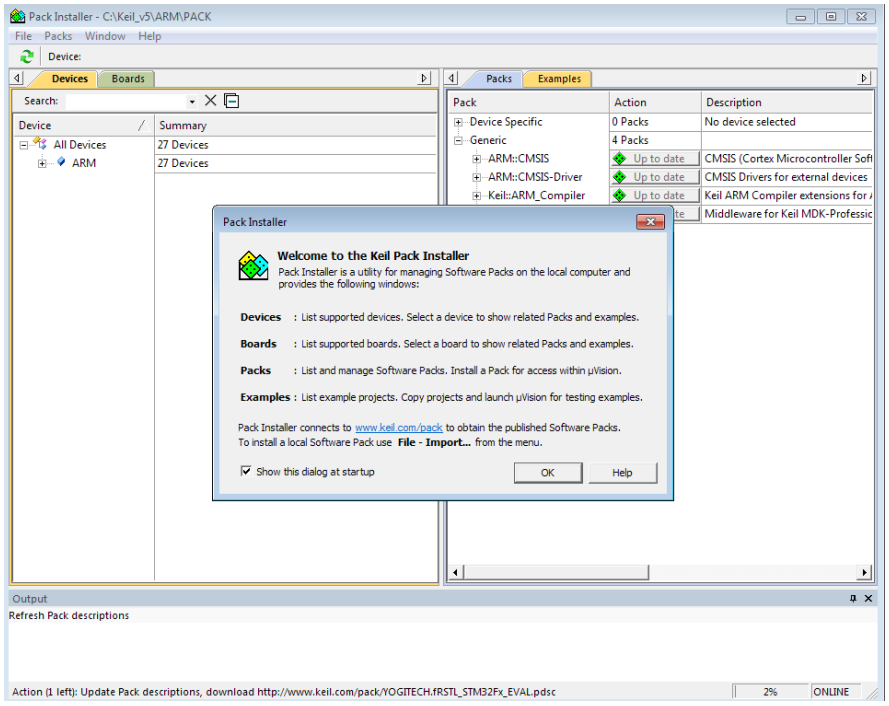
Algunos drivers se instalarán durante el proceso del asistente. Estos son necesarios para el funcionamiento correcto del kit y debe autorizarse su introducción al equipo.



Al finalizar la instalación del software básico, se abrirá el instalador de paquetes (donde se especifica qué dispositivos se espera que el programa soporte).

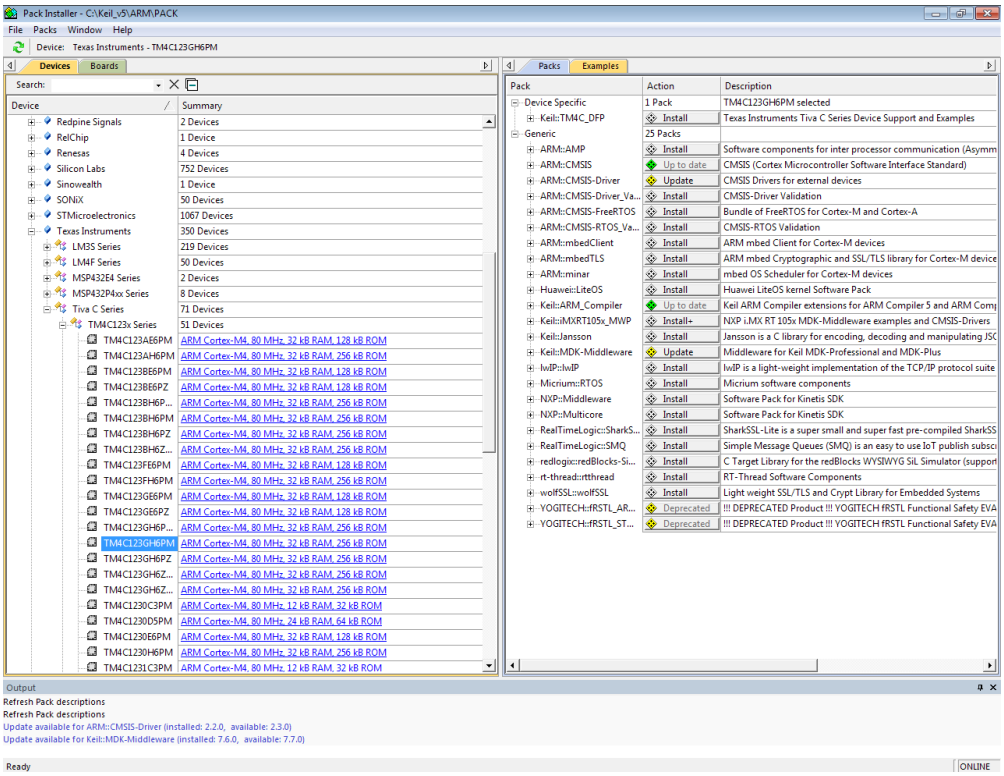
Continuación del anexo 1.





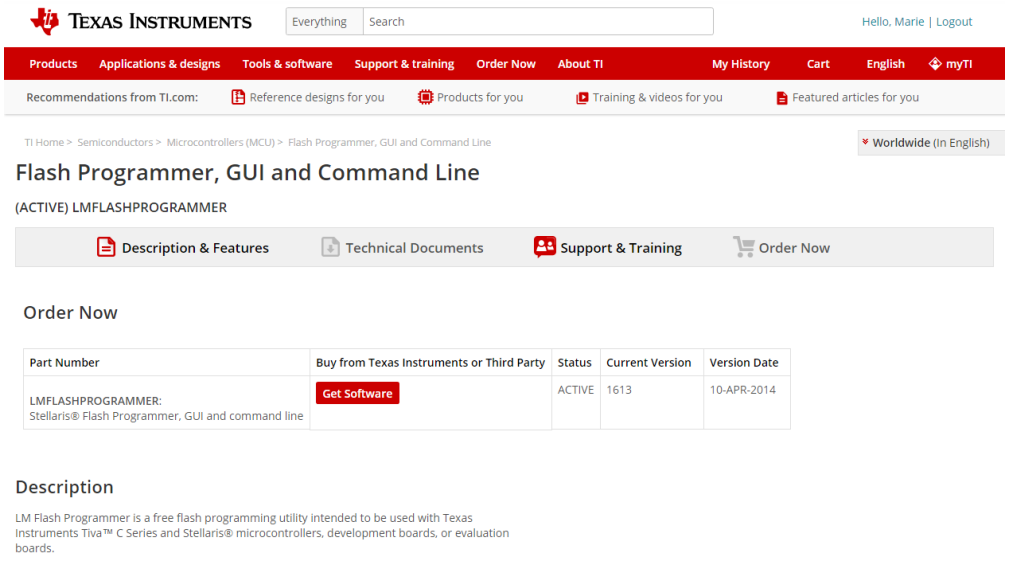
En la pestaña *Devices* dentro del listado ARM se encontrará el sublistado de Texas Instruments para seleccionar el microcontrolador específico a utilizar (en este caso, TM4C123GH6PM). En el lado derecho de la ventana, verificar que las extensiones de interés se marcan como instaladas.

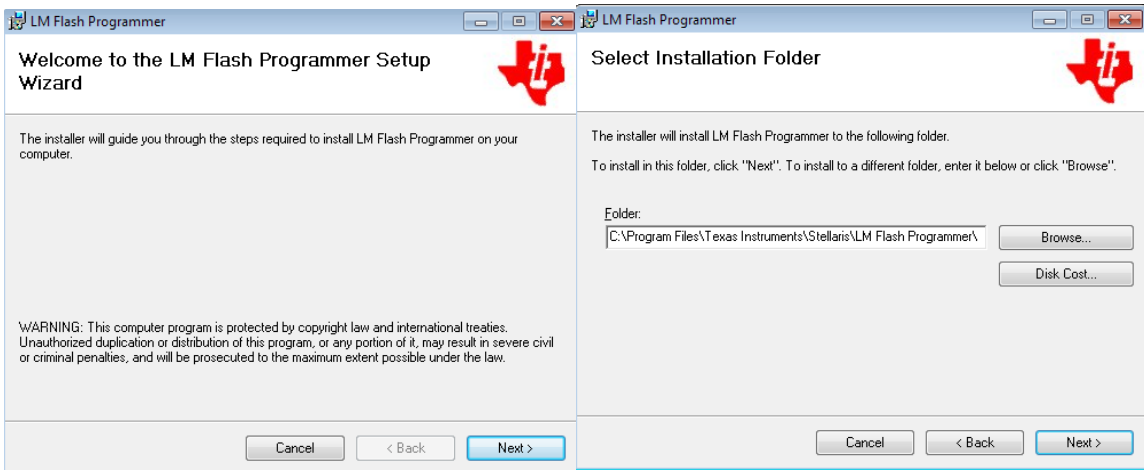
Continuación del anexo 1.

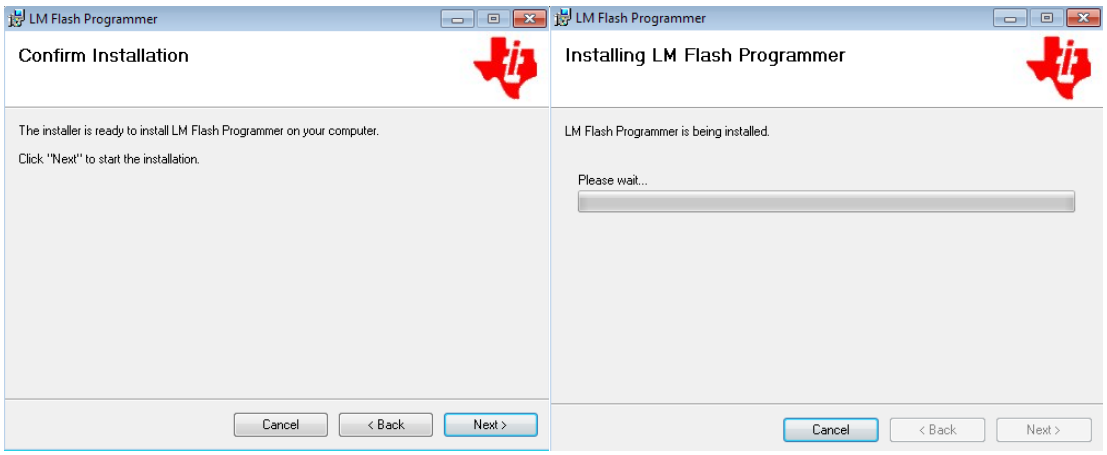


Para completar el proceso, si interesa utilizar la tarjeta de desarrollo Tiva C es necesario ingresar a la página de Texas Instruments (www.ti.com): acá se buscará primero el asistente de instalación descargable del LM Flash Programmer, que resulta bastante intuitivo.

Continuación del anexo 1.

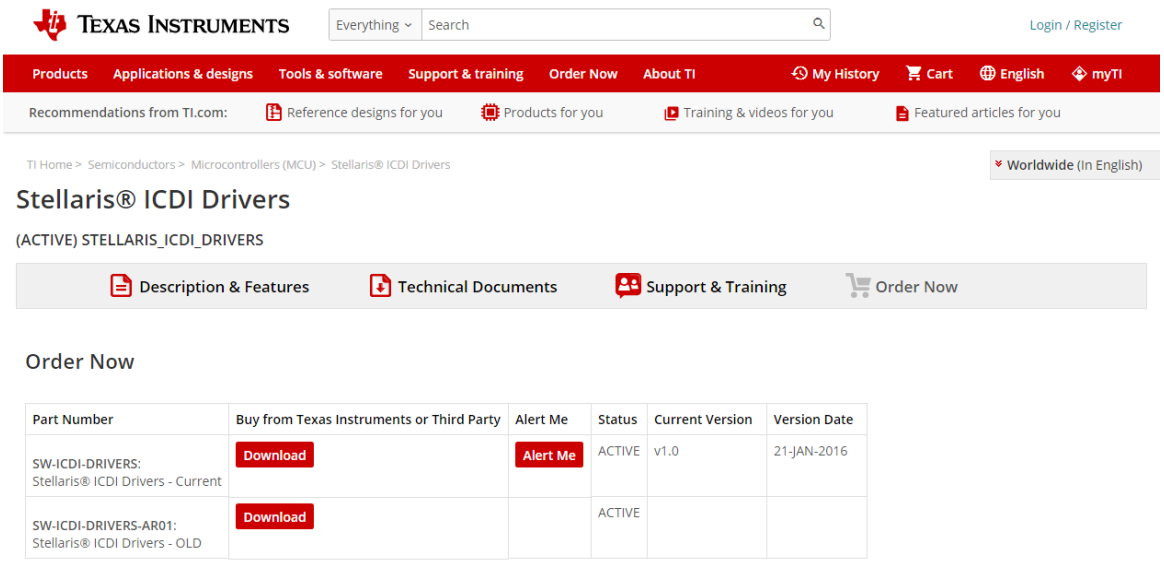






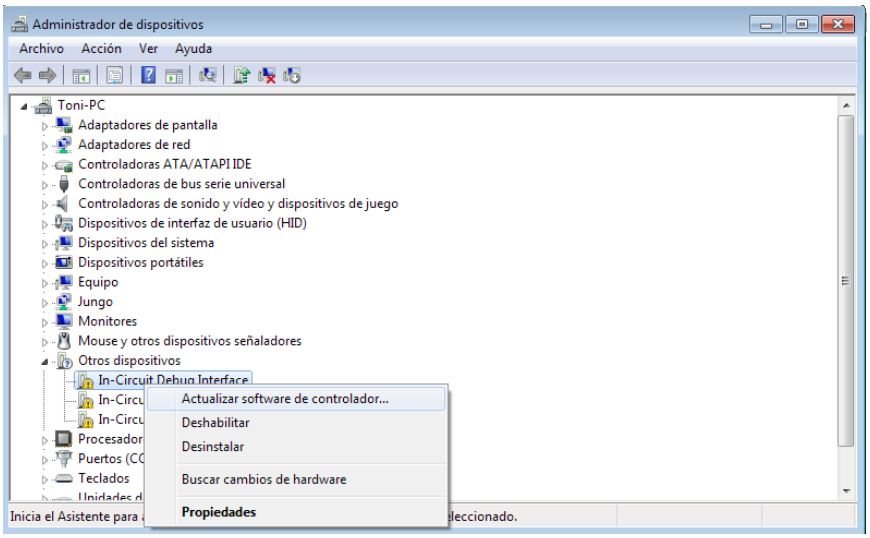
Continuación del anexo 1.

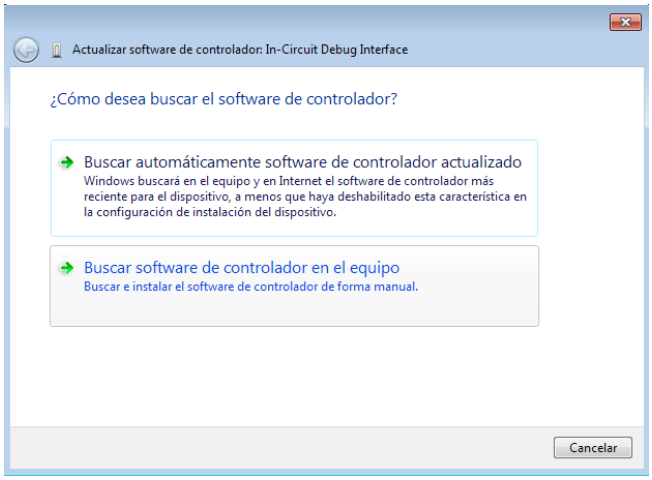
La segunda búsqueda dentro de la página de Texas Instruments serán los *Stellaris ICDI Drivers*. Este paquete hará posible reconocer y conectar apropiadamente la tarjeta de desarrollo por medio del cable USB en uno de los puertos del equipo.

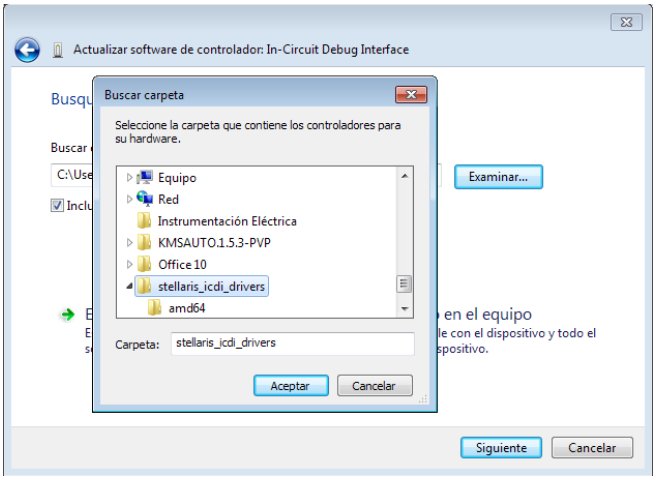


Su instalación se hace desde el administrador de dispositivos, especificando que los drivers deben buscarse en un folder específico.

Continuación del anexo 1.

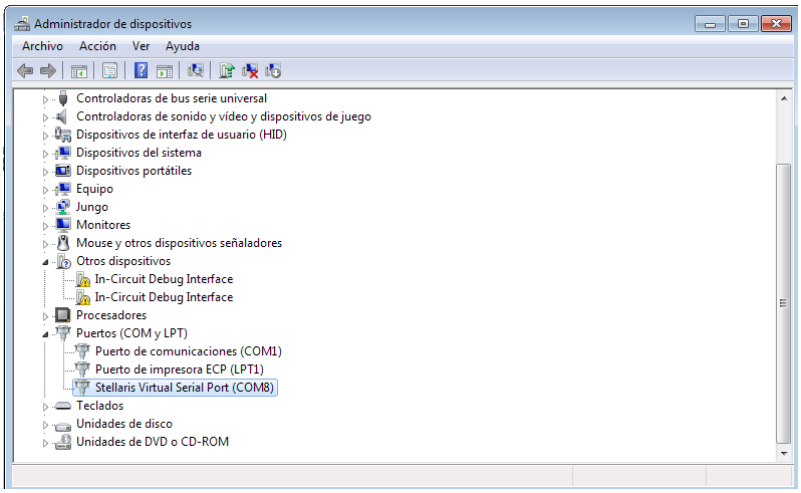


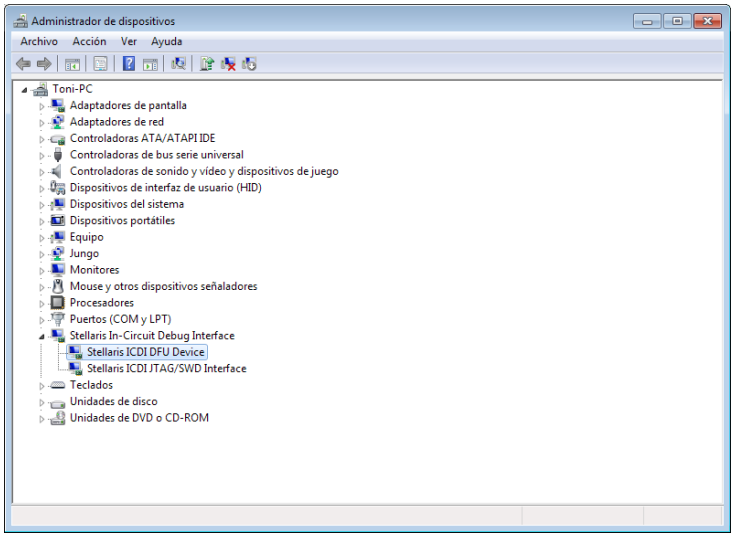




Continuación del anexo 1.

Al terminar el proceso, el puerto en que está conectada la tarjeta dejará de identificarse como interfaz de depuración en circuito. En cambio, aparecerá en la ventana de dispositivos como puerto Stellaris.

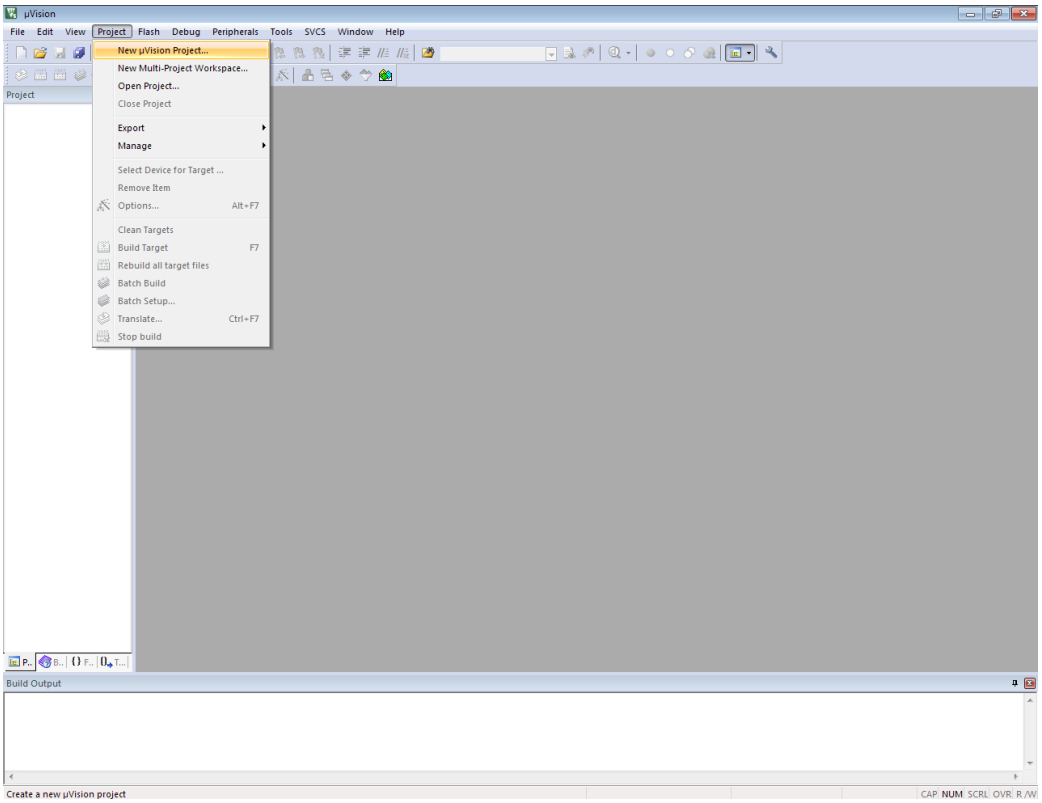




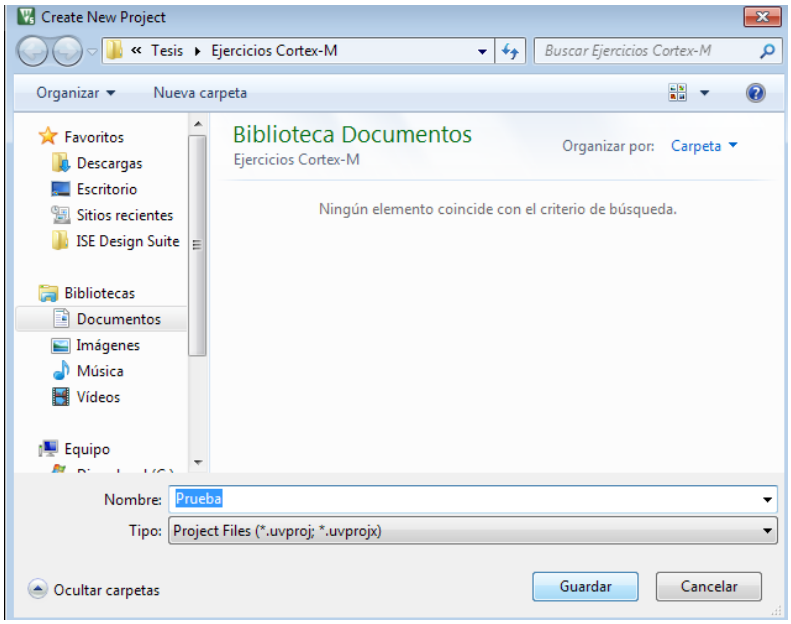
Continuación del anexo 1.

Creación de un nuevo proyecto

Crear un nuevo proyecto en Keil uVision es sencillo dado que cada paso se apoya en asistentes. El primer paso será seleccionar la opción “nuevo proyecto de uVision” en el menú *Project.* Esta opción abrirá una ventana para indicar el espacio en que se desee guardar el proyecto y su nombre.

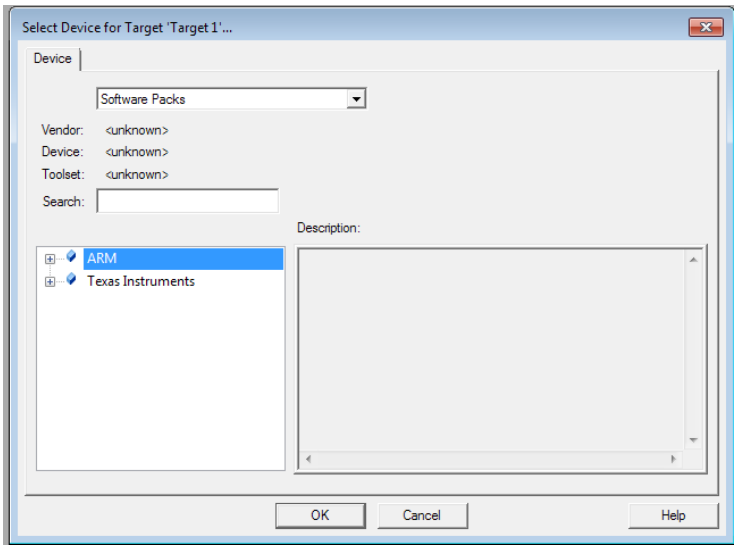


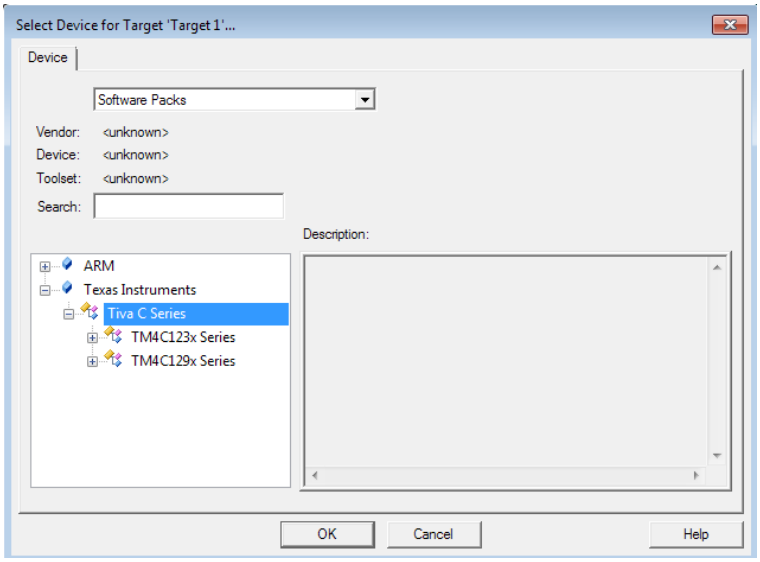
Continuación del anexo 1.



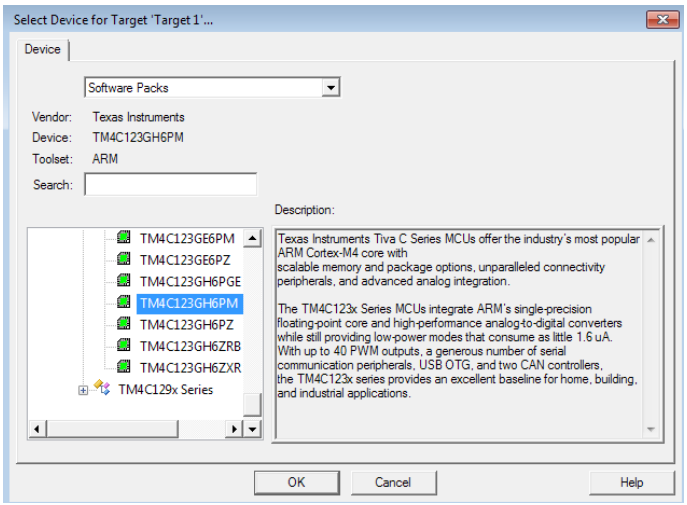
Luego de haber dado un nombre al proyecto se abrirá una ventana para seleccionar el dispositivo objetivo para el que se trabajará. Aquí debe seleccionarse en el listado de Texas Instruments, la subcategoría Tiva C Series para buscar dentro de ella el código del microcontrolador de interés (en este caso, TM4C123GH6PM).

Continuación del anexo 1.

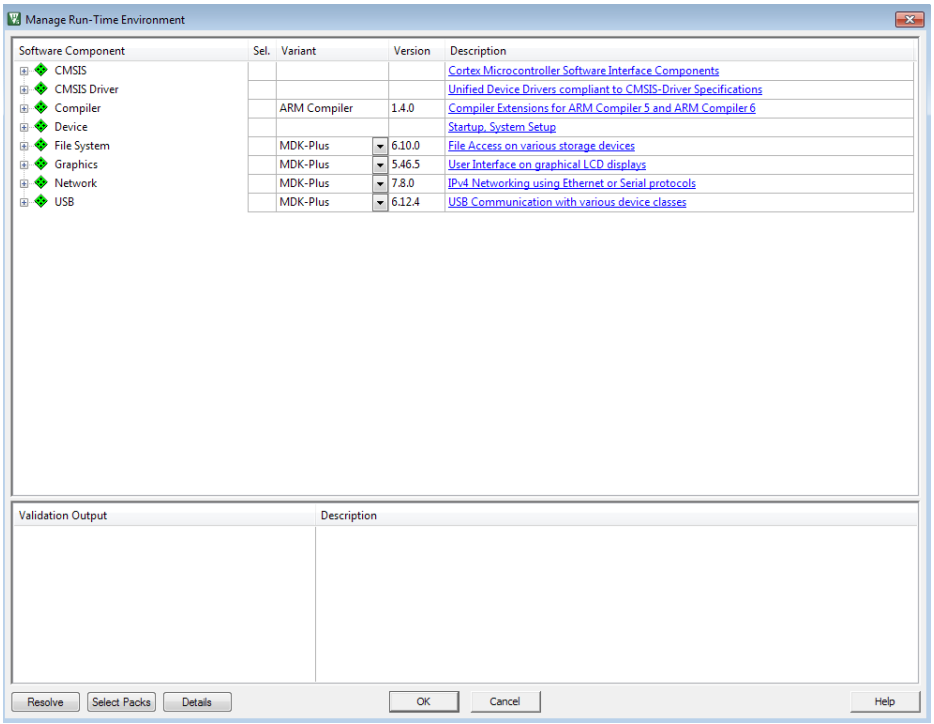




Continuación del anexo 1.

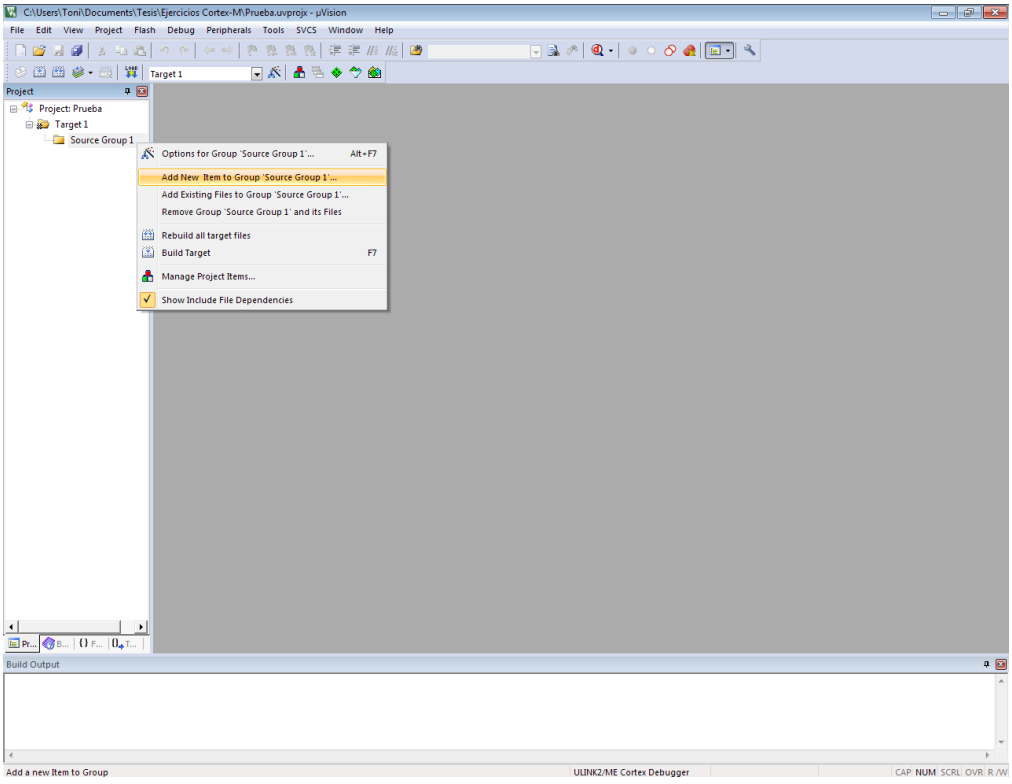


Al aceptar la selección de dispositivo, se abrirá una ventana indicando la selección de configuraciones para usos específicos. Para los fines del capítulo 6 no es necesaria ninguna, que se presiona 'OK' de nuevo.



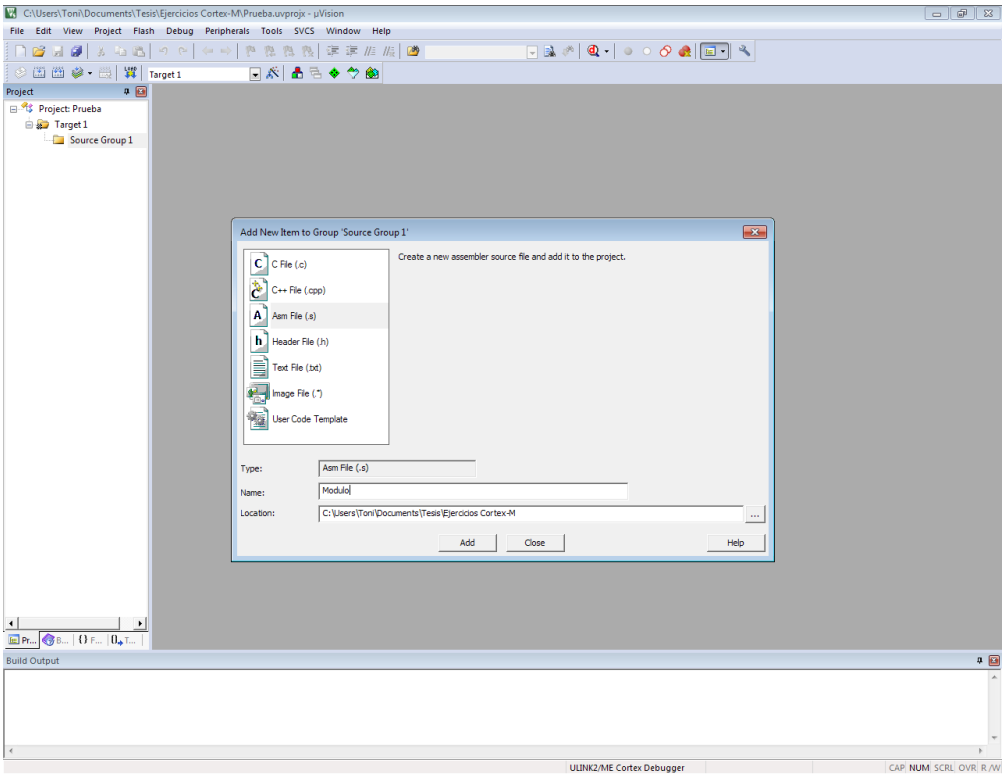
Continuación del anexo 1.

Para añadir módulos de código fuente al proyecto basta con seleccionar con un click derecho en la carpeta *Source Group* la opción 'añadir ítem al grupo'.

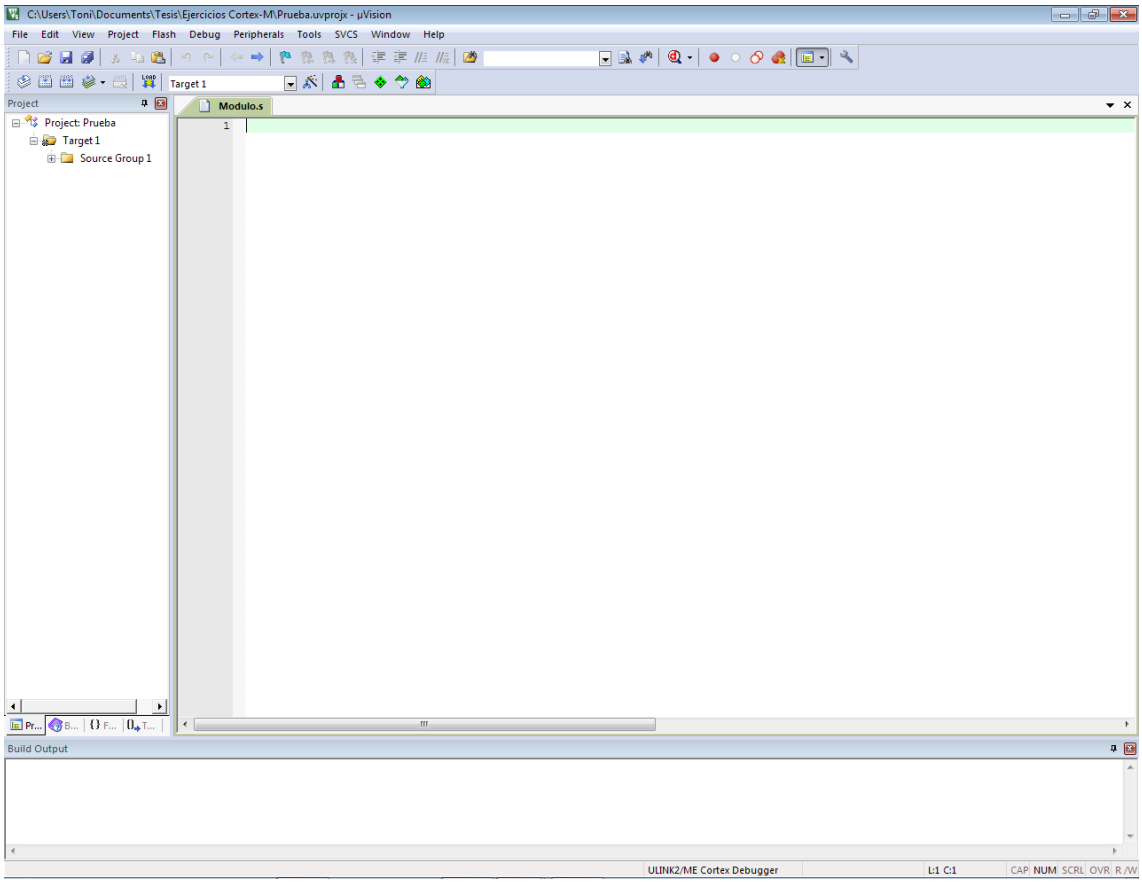


Podrá elegirse la creación de módulos de distintos tipos (siendo los más utilizados los archivos de C, C++ y ensamblador). La tercera opción generará un archivo con sufijo '.s' para el ensamblador, será la opción a elegir para todos los ejercicios del capítulo 6.

Continuación del anexo 1.

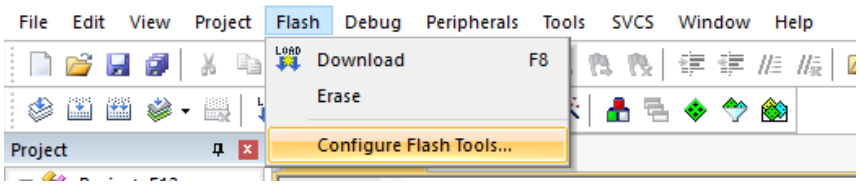


La ventana de un archivo '.s' abierto listo para escribir código en Keil uVision se verá así:



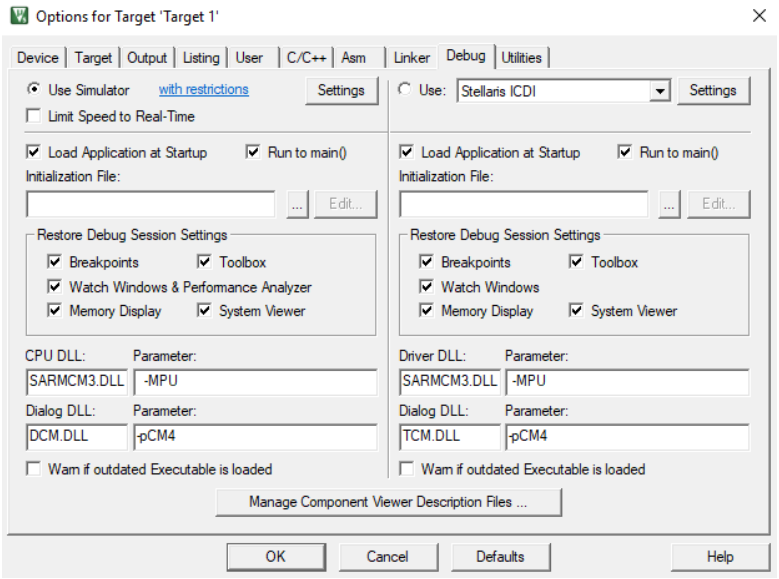
Depuración por simulación

Para observar cómo un programa se comporta es útil simular su resultado antes de cargarlo en el chip de interés. El menú *Flash* ofrece la opción *Configure Flash Tools* para especificar este modo de depuración.



Continuación del anexo 1.

Al seleccionar la opción se abrirá una venta con dos columnas principales. Para el fin de simulación debe seleccionarse la primera: *Use Simulator*.



A partir de esta configuración basta simplemente seguir los siguientes pasos para acceder a la depuración:

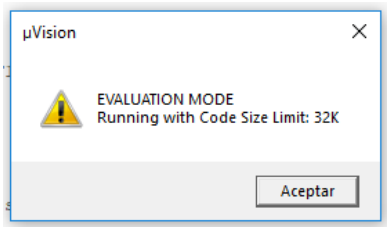
- Todos los archivos deben haber sido guardados.

- El proyecto se construye con la opción *Build*.

Continuación del anexo 1.

- Se inicia el depurador.

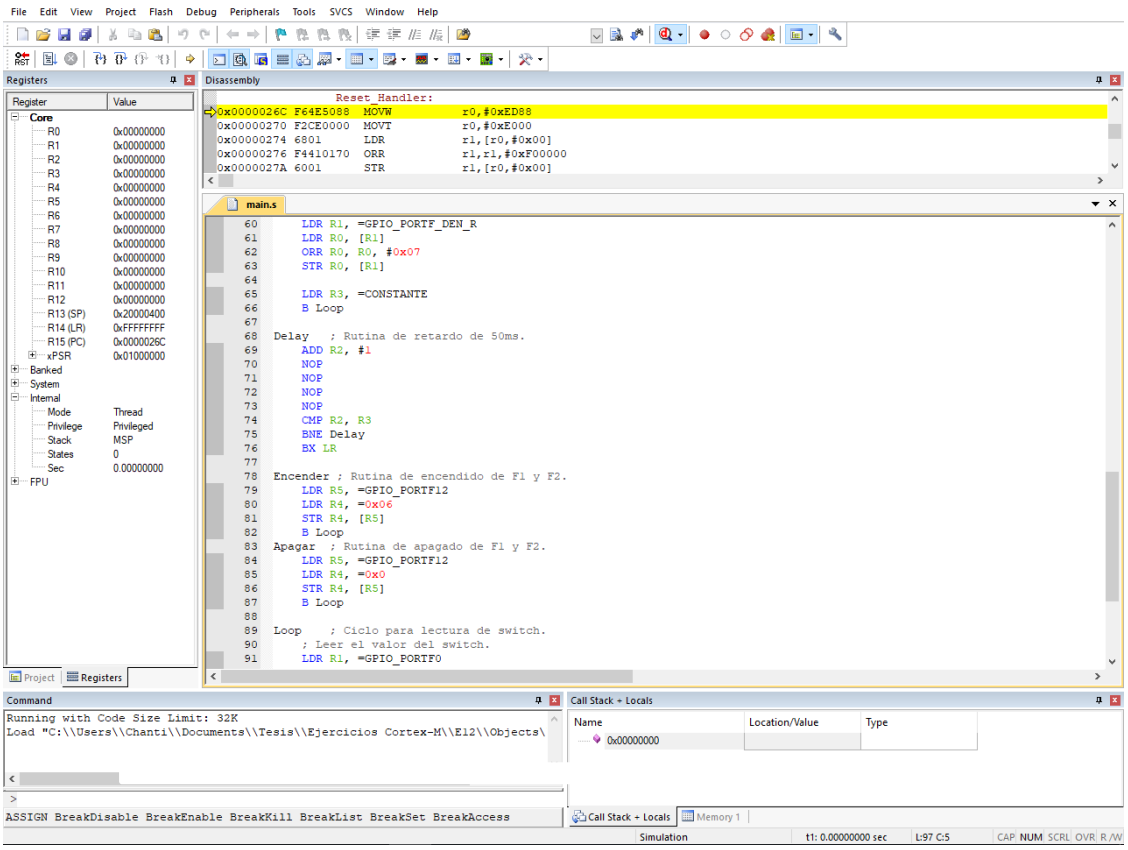
Se abrirá una pequeña advertencia luego de iniciar el depurador para indicar que el entorno cambiará. Simplemente hace falta aceptar para que el depurador comience a trabajar.



Nota: es fundamental que cada vez que un cambio se haga en el código, el proyecto vuelva a guardarse y construirse (incisos 1 y 2). Si estos pasos se omiten, el depurador seguirá evaluando la última opción construida y los cambios no se verán.

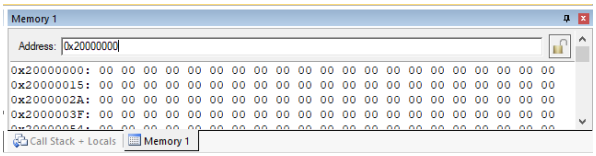
Continuación del anexo 1.

La ventana de depuración se verá así:



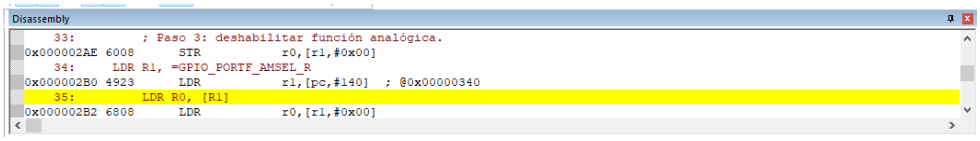
Notar que las secciones principales del código son:

La ventana de memoria y pila, donde se puede especificar la dirección que interesa.

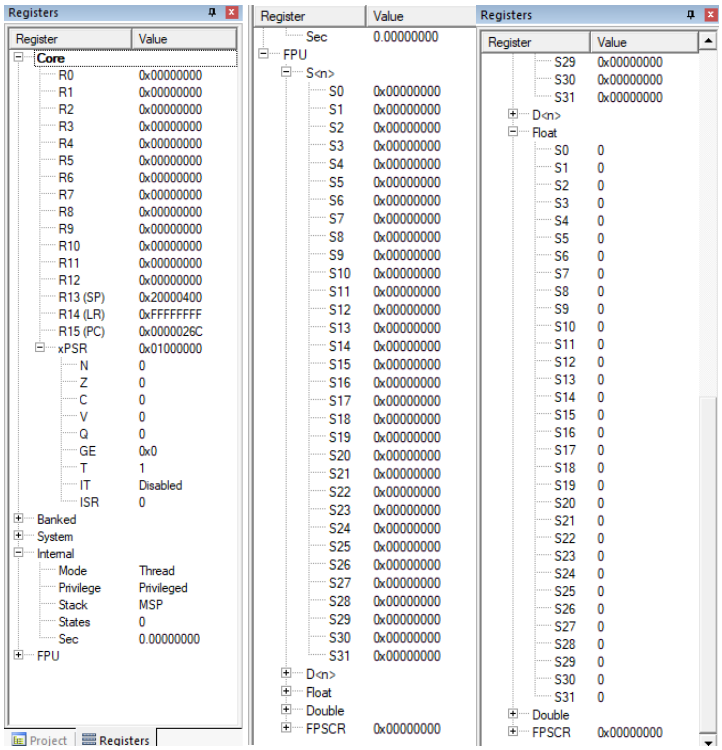


Continuación del anexo 1.

El desensamblador.

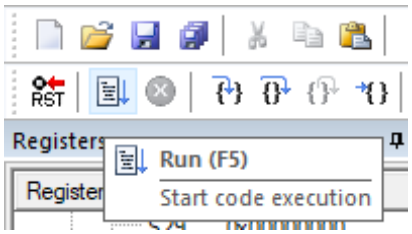


La ventana de registros (con vista del grupo del núcleo, el de FPU y los de estado).



Para controlar la ejecución del programa basta con presionar la tecla F11 para avanzar instrucción por instrucción o utilizar el botón Run que ejecuta todo el código.

Continuación del anexo 1.



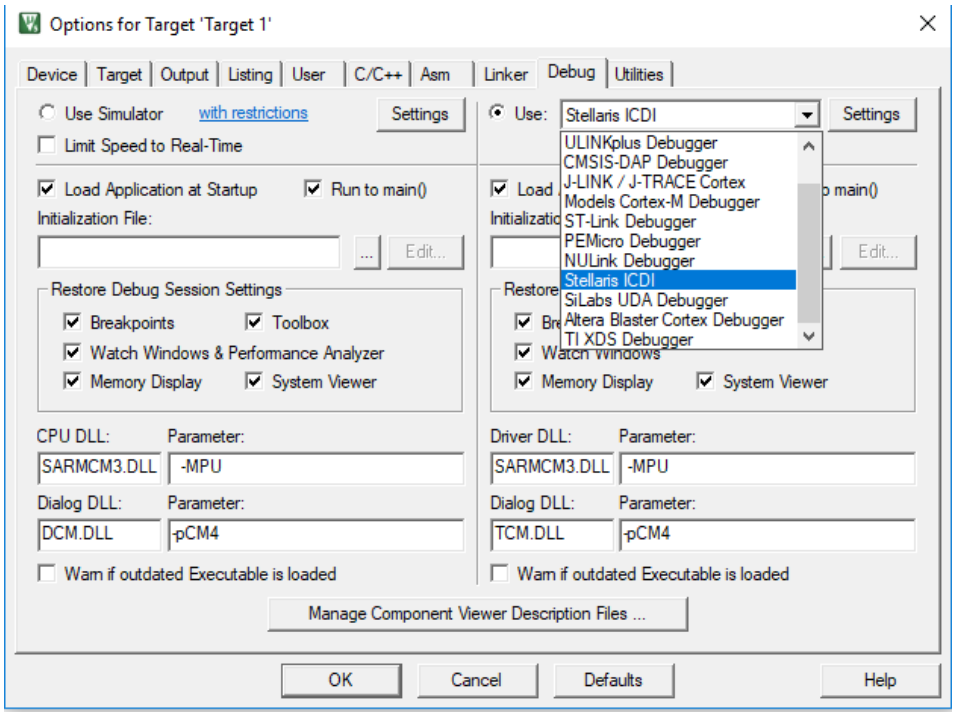
Carga de proyecto en el microcontrolador

La alternativa a la simulación es cargar el archivo ejecutable en la memoria del dispositivo. Para esto simplemente basta con volver al menú *Flash* con la opción *Configure Flash Tool.*

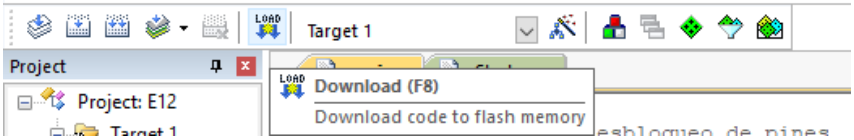
De nuevo se abrirá la ventana de dos columnas. En este caso se elegirá la segunda.

Luego de cambiar la selección a la columna de la derecha, debe asegurarse que esté seleccionada la opción Stellaris ICDI si la tarjeta de desarrollo en uso es la Tiva C.

Continuación del anexo 1.



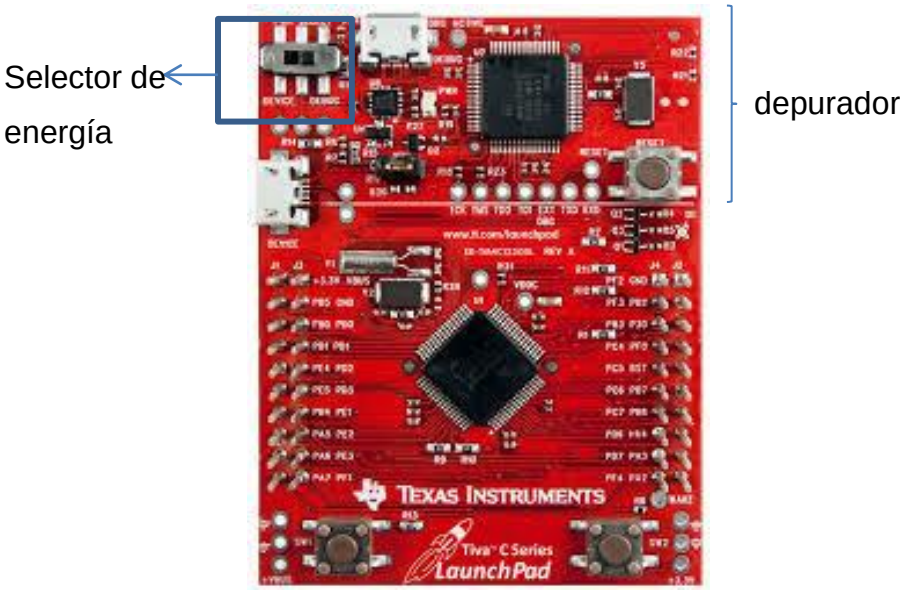
Luego de esta configuración, confirmar que la tarjeta esté conectada por medio del cable, guardar todos los archivos, construir el proyecto y descargarlo con ayuda del botón *Download*.



A partir de este momento, el microcontrolador estará listo para ejecutar el programa (quizás haga falta en algunas ocasiones presionar el botón de reinicio en la tarjeta cuando se ejecuta por primera vez).

Continuación del anexo 1.

También puede ejecutarse la ventana de depuración usando la tarjeta de desarrollo. Si se decide utilizar esta característica debe verificarse que esté seleccionado el modo de depuración en la Tiva C por medio del interruptor dedicado específicamente a dar energía a esta sección.

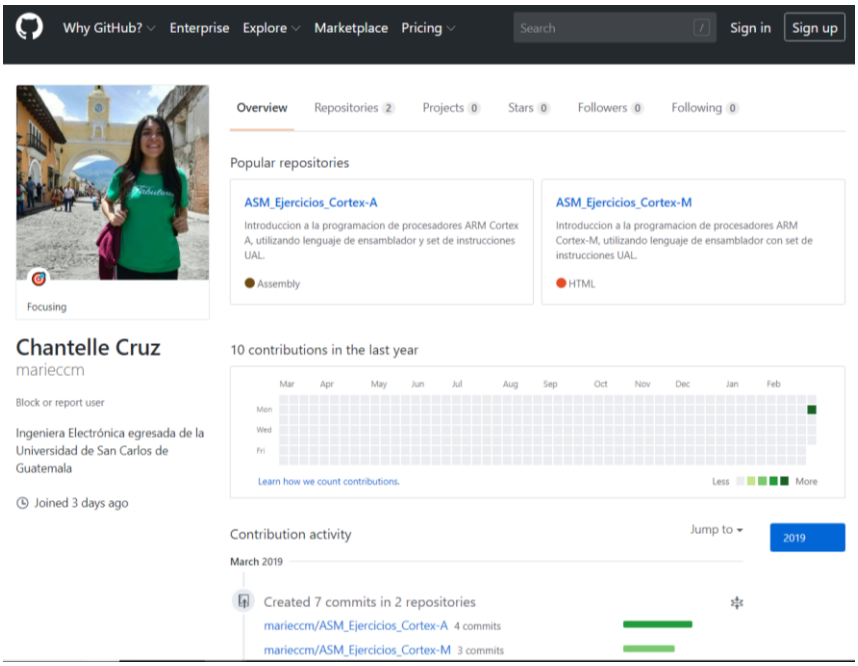


.

Continuación del anexo 1.

Repositorio de ejercicios

Los proyectos generados junto al desarrollo del contenido en capítulos 6 y 7 para introducir la programación de procesadores Cortex-M y Cortex-A se encuentran en el enlace www.github.com/marieccm.



Fuente: CRUZ MEDINA, Marie Chantelle. *Github*. www.github.com/marieccm. Consulta: 4 de marzo de 2019.