Arquitectura de Microprocesadores

Carrera de Especialización en Sistemas Embebidos Universidad de Buenos Aires

**Preguntas orientadoras**

**Mansilla Rubén Darío Co20 CESE**

**1.** Describa brevemente los diferentes perfiles de familias de microprocesadores / microcontroladores de ARM. Explique alguna de sus diferencias características.

**Cortex M**

**1.** Describa brevemente las diferencias entre las familias de procesadores Cortex M0, M3 y M4.

**2.** ¿Por qué se dice que el set de instrucciones Thumb permite mayor densidad de código? Explique.

**3.** ¿Qué entiende por arquitectura load-store? ¿Qué tipo de instrucciones no posee este tipo de arquitectura?

**4.** ¿Cómo es el mapa de memoria de la familia?

**5.** ¿Qué ventajas presenta el uso de los “shadowed pointers” del PSP y el MSP?

**6.** Describa los diferentes modos de privilegio y operación del Cortex M, sus relaciones y como se conmuta de uno al otro. Describa un ejemplo en el que se pasa del modo privilegiado a no privilegiado y nuevamente a privilegiado.

**7.** ¿Qué se entiende por modelo de registros ortogonal? Dé un ejemplo

**8.** ¿Qué ventajas presenta el uso de instrucciones de ejecución condicional (IT)? Dé un ejemplo

**9.** Describa brevemente las excepciones más prioritarias (reset, NMI, Hardfault).

**10.** Describa las funciones principales de la pila. ¿Cómo resuelve la arquitectura el llamado a funciones y su retorno?

**11.** Describa la secuencia de reset del microprocesador.

**12.** ¿Qué entiende por “core peripherals”? ¿Qué diferencia existe entre estos y el resto de los periféricos?

**13.** ¿Cómo se implementan las prioridades de las interrupciones? Dé un ejemplo

**14.** ¿Qué es el CMSIS? ¿Qué función cumple? ¿Quién lo provee? ¿Qué ventajas aporta?

**15.** Cuando ocurre una interrupción, asumiendo que está habilitada ¿Cómo opera el microprocesador para atender a la subrutina correspondiente? Explique con un ejemplo

**16.** ¿Cómo cambia la operación de stacking al utilizar la unidad de punto flotante?

**17.** Explique las características avanzadas de atención a interrupciones: tail chaining y late arrival.

**18.** ¿Qué es el systick? ¿Por qué puede afirmarse que su implementación favorece la portabilidad de los sistemas operativos embebidos?

**19.** ¿Qué funciones cumple la unidad de protección de memoria (MPU)?

**20.** ¿Cuántas regiones pueden configurarse como máximo? ¿Qué ocurre en caso de haber solapamientos de las regiones? ¿Qué ocurre con las zonas de memoria no cubiertas por las regiones definidas?

**21.** ¿Para qué se suele utilizar la excepción PendSV? ¿Cómo se relaciona su uso con el resto de las excepciones? Dé un ejemplo

**22.** ¿Para qué se suele utilizar la excepción SVC? Explíquelo dentro de un marco de un sistema operativo embebido.

**ISA**

**1.** ¿Qué son los sufijos y para qué se los utiliza? Dé un ejemplo

**2.** ¿Para qué se utiliza el sufijo ‘s’? Dé un ejemplo

**3.** ¿Qué utilidad tiene la implementación de instrucciones de aritmética saturada? Dé un ejemplo con operaciones con datos de 8 bits.

**4.** Describa brevemente la interfaz entre assembler y C ¿Cómo se reciben los argumentos de las funciones? ¿Cómo se devuelve el resultado? ¿Qué registros deben guardarse en la pila antes de ser modificados?

**5.** ¿Qué es una instrucción SIMD? ¿En qué se aplican y que ventajas reporta su uso? Dé un ejemplo.

**Desarrollo**

**Preguntas orientadoras**

**1.** La familia de microprocesadores / microcontroladores de ARM está agrupada en tres tipos:

* **Cortex A (Application):** Esta familia está conformada por microprocesadores de Aplicación, de donde toma la sigla A. Constituida por microprocesadores de alto rendimiento orientados a soportar sistemas operativos embebidos de alta performance, y alto nivel de paralelismo.

Características principales:

* Contienen varios núcleos de alta frecuencia, del orden de los GHz.
* Tienen integrada mucha memoria RAM del orden de los GB.
* Tienen mucha memoria caché

Aplicaciones: Nintendo switch con las siguientes características:

* 4 Cortex A-57 trabajando a 1GHz
* 4 Cortex A-53 trabajando a 1,3 GHz
* 4 GB de memoria RAM LPDDR4
* **Cortex R (Realtime):** Esta familia está conformada por microprocesadores orientados a soportar sistemas de tiempo real donde se necesita implementar soluciones de baja. latencia, alta predictibilidad y elevada capacidad de cómputo. Por estas características se utilizan en sistemas críticos, como por ejemplo: sistemas de control para automóviles ( control de tracción, de frenos, ets), dispositivos médicos críticos, dispositivos industriales críticos, etc**.**
* **Cortex M (Microcontroller):** Son microcontroladores orientados a dispositivos de uso masivo y sistemas embebidos compactos. Están diseñados para soportar alta densidad de código y ser programados en C.

Los más conocidos son:

* **Cortex M0/M0+:** Pensados para una implementación mínima de bajo consumo y bajo costo.
* **Cortex M3/M4/M7:** Suman mayor performance, más funcionalidades (división por hardware), FPU (Unidad de Punto Flotante), MPU (Unidad de Protección de memoria), etc.

Ejemplos en placas de desarrollo: Imagen de la pantalla de un video juego

Descripción generada automáticamente con confianza media Un circuito electrónico

Descripción generada automáticamente con confianza media

PSOC 5LP: Cortex M3

Raspberry PI Pico: Cortex M0+

Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja Imagen de la pantalla de un computador

Descripción generada automáticamente con confianza baja

STM32-Nucleo F676ZI: Cortex M7

STM32-Nucleo F429ZI: Cortex M4

**Cortex M**

**1.** Diferencias entre las familias de procesadores de ARM:

# Cortex M0 y M0+

* Tienen arquitectura ARMv6-M
* La mayoría soporta instrucciones Thumb (16 bits)
* Algunos soportan Thumb-2 (32 bits)
* Tienen multiplicación por hardware de 32 bits
* No tienen división por hardware
* No tienen solución “saturated math”
* No contienen extensión DSP (procesamiento de señales digitales)
* No incluyen FPU (Unidad de Punto Flotante)

# Cortex M3

* Tienen arquitectura ARMv7-M
* Todos soportan instrucciones Thumb (16 bits)
* Todos soportan Thumb-2 (32 bits)
* Tienen multiplicación por hardware de 32 bits o 64 bits
* Tienen división por hardware
* Tienen solución “saturated math”
* No contienen extensión DSP (procesamiento de señales digitales)
* No incluyen FPU (Unidad de Punto Flotante)

# Cortex M4

* Tienen arquitectura ARMv7E-M
* Todos soportan instrucciones Thumb (16 bits)
* Todos soportan Thumb-2 (32 bits)
* Tienen multiplicación por hardware de 32 bits o 64 bits
* Tienen división por hardware
* Tienen solución “saturated math”
* Contienen extensión DSP (procesamiento de señales digitales)
* Opcionalmente (SP) incluyen FPU (Unidad de Punto Flotante)

Cortex M7

* Tienen arquitectura ARMv7E-M
* Todos soportan instrucciones Thumb (16 bits)
* Todos soportan Thumb-2 (32 bits)
* Tienen multiplicación por hardware de 32 bits o 64 bits
* Tienen división por hardware
* Tienen solución “saturated math”
* Contienen extensión DSP (procesamiento de señales digitales)
* Opcionalmente (SP & DP) incluyen FPU (Unidad de Punto Flotante)

**2.** El set de instrucciones Thumb permite una mayor densidad de código en comparación con el set de instrucciones ARM tradicional, debido a que las instrucciones Thumb son instrucciones de 16 bits, mientras que las instrucciones ARM son de 32 bits. Esto puede ser beneficioso en términos de ahorro de memoria y aumento de la velocidad de transferencia de las instrucciones al microprocesador, aunque puede tener limitaciones en cuanto a la complejidad de las operaciones que se pueden realizar ya que necesitaría mas instrucciones thumb para resolver algunas tareas mas complejas.

**3.** La arquitectura load-store es un tipo de arquitectura en la que solo se pueden realizar operaciones de carga (load) y almacenamiento (store) entre la memoria y los registros, mientras que las operaciones aritméticas se realizan exclusivamente entre registros.

En una arquitectura load-store, las instrucciones de carga (load) se utilizan para transferir datos desde la memoria a los registros, mientras que las instrucciones de almacenamiento (store) se utilizan para transferir datos desde los registros a la memoria.

Este tipo de arquitectura **no permite realizar operaciones aritméticas** directamente entre la memoria y los registros. Por lo tanto, l**as instrucciones que implican operaciones aritméticas con operandos en la memoria**, como por ejemplo "add" (suma) o "sub" (resta), n**o están disponibles** en una arquitectura load-store.

En resumen, una arquitectura load-store es aquella en la que las operaciones de carga y almacenamiento se realizan entre la memoria y los registros, mientras que las operaciones aritméticas se realizan exclusivamente entre registros.

**4.** El mapa de memoria de la familia ARM varía dependiendo del modelo específico y la implementación de la arquitectura. Sin embargo, en términos generales, la familia ARM utiliza una estructura de memoria en la que se divide el espacio de direcciones en diferentes regiones.

A modo de descripción general del mapa de memoria típico de la familia ARM, podemos considerar la siguiente subdivisión de regiones:

* Zona de código (Code Section): Esta zona se utiliza para almacenar el código ejecutable, como las instrucciones del programa y las constantes. Es una región de solo lectura (read-only) y suele estar ubicada en la parte inferior del espacio de direcciones.
* Zona de datos (Data Section): Esta zona se utiliza para almacenar los datos del programa, como variables globales y estáticas. Puede ser de lectura y escritura (read-write) y suele estar ubicada en la parte superior del espacio de direcciones.
* Pila (Stack): La pila se utiliza para almacenar las variables locales y los registros de activación de las funciones. Generalmente crece hacia abajo en la memoria y se encuentra en la parte superior del espacio de direcciones.
* Montón (Heap): El montón se utiliza para almacenar datos dinámicos, como objetos o estructuras de datos creados en tiempo de ejecución. Generalmente crece hacia arriba en la memoria y se encuentra en la parte inferior del espacio de direcciones.
* Además de estas regiones principales, también puede haber otras regiones en el mapa de memoria, como zonas reservadas para el sistema operativo, controladores de dispositivos, tablas de vectores de interrupción, entre otros.

|------------------------------------|

| Zona de código (ROM) |

|------------------------------------|

| Zona de código (RAM) |

|------------------------------------|

| Zona de datos (RAM) |

|------------------------------------|

| Pila |

|------------------------------------|

| Montón |

|------------------------------------|

| Zona de dispositivos |

|------------------------------------|

| Zona de sistema |

|------------------------------------|

**5.** El uso de "shadowed pointers" (punteros en sombra) en el PSP (Process Stack Pointer) y el MSP (Main Stack Pointer) en la arquitectura ARM presenta varias ventajas:

* Seguridad: Los "shadowed pointers" permiten tener una copia de seguridad de los punteros de pila principales (PSP y MSP). Esto es útil en caso de que se produzca un desbordamiento de pila o un error en el programa, ya que se puede acceder a la copia en sombra para recuperar la integridad del sistema.
* Protección contra overflow (desbordamiento) de pila: El uso de "shadowed pointers" puede ayudar a detectar y proteger contra desbordamientos de pila. Si se produce un overflow en el PSP o el MSP, la copia en sombra se mantendrá intacta y se podrá utilizar para detectar el overflow y tomar medidas correctivas.
* Facilidad de depuración: Al tener una copia en sombra de los punteros de pila, se facilita la depuración de errores relacionados con la pila. Se pueden utilizar herramientas de depuración para comparar los valores de los punteros en sombra con los valores actuales y detectar cualquier discrepancia o anomalía.
* Recuperación de errores: En caso de que se produzca un error crítico en el programa y se pierda la integridad de los punteros de pila principales, los "shadowed pointers" pueden utilizarse para recuperar el sistema y restaurar los valores correctos de los punteros de pila.

En resumen, el uso de "shadowed pointers" en el PSP y el MSP proporciona una capa adicional de seguridad y protección contra desbordamientos de pila, facilita la depuración de errores relacionados con la pila y permite la recuperación del sistema en caso de errores críticos.

**6.** El procesador Cortex-M de ARM utiliza un modelo de seguridad basado en modos de privilegio y operación para garantizar la protección y el aislamiento entre diferentes niveles de software. Los modos de privilegio y operación en el Cortex-M son los siguientes:

* Modo Handler (Privilegiado): Este modo es el más alto en términos de privilegios y se utiliza para ejecutar rutinas de manejo de excepciones y eventos, como interrupciones y excepciones de sistema. En este modo, se tiene acceso completo a todas las instrucciones y registros del procesador.
* Modo Thread (Privilegiado): Este modo se utiliza para la ejecución de tareas normales del programa. En este modo, se tiene acceso completo a todas las instrucciones y registros del procesador.
* Modo Unprivileged (No privilegiado): Este modo es el de menor privilegio y se utiliza para ejecutar tareas con restricciones de seguridad. En este modo, se restringe el acceso a ciertas instrucciones y registros del procesador, lo que ayuda a proteger el sistema operativo y las aplicaciones de software de posibles amenazas.

La conmutación entre los diferentes modos de privilegio y operación se realiza mediante el cambio de bits específicos en el registro de estado (PSR) del procesador. El bit de Control (CONTROL) en el PSR se utiliza para habilitar o deshabilitar el cambio de modo privilegiado a no privilegiado y viceversa.

A continuación, se describe un ejemplo de cómo se pasa del modo privilegiado al no privilegiado y nuevamente al modo privilegiado:

* Inicialmente, el procesador se encuentra en el modo privilegiado (Modo Handler o Modo Thread).
* Se ejecuta una instrucción que cambia el bit de Control en el PSR para habilitar el modo no privilegiado.
* A partir de ese momento, el procesador se encuentra en el modo no privilegiado, donde se restringe el acceso a ciertas instrucciones y registros.
* Durante la ejecución en el modo no privilegiado, se produce una excepción o interrupción.
* El procesador cambia automáticamente al modo privilegiado (Modo Handler) para manejar la excepción o interrupción.
* Una vez finalizada la rutina de manejo de excepciones, se ejecuta una instrucción que cambia nuevamente el bit de Control en el PSR para habilitar el modo privilegiado.
* A partir de ese momento, el procesador se encuentra nuevamente en el modo privilegiado (Modo Handler o Modo Thread).

No se puede regresar directamente del modo no privilegiado al modo privilegiado. Si se desea hacerlo, se debe disparar una interrupción o excepción que saque al sistema del modo no privilegiado y lo lleve al modo Handler (privilegiado); sólo estando en un modo privilegiado se puede cambiar el bit de control de PSR para habilitar el modo privilegiado.

**7.** Se dice que la arquitectura ARM cuenta con un modelo de registros ortogonal porque la configuración de cada registro no interfiere en la del otro. El contenido de cada registro no afecta al contenido del otro.

Por ejemplo, cuando se utiliza un sistema operativo embebido, el registro CONTROL podría re programarse en cada cambio de contexto para permitir que algunas tareas de la aplicación se ejecuten con un nivel de acceso privilegiado y otras se ejecuten con un nivel de acceso sin privilegios.

Decimos que las configuraciones de nPRIV y SPSEL son ortogonales, dado que son posibles cuatro combinaciones diferentes de nPRIV y SPSEL, aunque sólo tres de ellas se usan comúnmente en aplicaciones del mundo real.

En la mayoría de las aplicaciones simples sin un sistema operativo embebido, no es necesario cambiar el valor del registro CONTROL. Toda la aplicación puede ejecutarse en un nivel de acceso privilegiado y utilizar únicamente el MSP.

**8.** Las instrucciones de ejecución condicional (IT, por sus siglas en inglés) son una característica de la arquitectura ARM que permite ejecutar instrucciones condicionalmente en función del estado de las banderas o flags del procesador. Estas instrucciones ofrecen varias ventajas, entre las que se incluyen:

* Reducción del tamaño del código: Las instrucciones condicionales permiten ejecutar diferentes instrucciones en función de una condición, lo que puede eliminar la necesidad de escribir múltiples bloques de código separados. Esto reduce el tamaño del código y facilita su mantenimiento.
* Mejora del rendimiento: Al ejecutar instrucciones condicionalmente, se pueden evitar saltos o desvíos innecesarios, lo que puede mejorar el rendimiento del programa al reducir la cantidad de ciclos de reloj necesarios para ejecutarlo.
* Mayor flexibilidad: Las instrucciones condicionales permiten adaptar el flujo de ejecución del programa según diferentes condiciones, lo que brinda una mayor flexibilidad en el diseño del software. Esto puede ser especialmente útil en casos donde se necesitan tomar decisiones basadas en múltiples condiciones.
* Optimización del código: Al utilizar instrucciones condicionales, el compilador puede realizar diferentes optimizaciones, como la eliminación de código muerto o la reorganización de instrucciones para aprovechar al máximo el pipeline del procesador.

Un ejemplo de uso de instrucciones de ejecución condicional en ARM sería:

```

CMP R0, #0 ; Compara el valor de R0 con 0

ITE GT ; Ejecuta las siguientes dos instrucciones si R0 es mayor que 0

MOVGT R1, #1 ; Si R0 > 0, mueve el valor 1 a R1

MOVLE R1, #0 ; Si R0 <= 0, mueve el valor 0 a R1

```

En este ejemplo, se compara el valor de R0 con 0 y, dependiendo del resultado de la comparación, se ejecutan diferentes instrucciones. Si R0 es mayor que 0, se mueve el valor 1 a R1. Si R0 es menor o igual a 0, se mueve el valor 0 a R1. Esto permite tomar diferentes acciones en función de una condición específica.

**9.** Describa brevemente las excepciones más prioritarias (reset, NMI, Hardfault):

* El **Reset** es un tipo especial de excepción. Cuando el procesador sale de un reset, ejecuta el handler de reinicio en modo Thread (en lugar del modo Handler como en otras excepciones). Además, el número de excepción en IPSR se lee como cero.

Un reset pone al PC apuntando a la dirección 00000H. Tenemos dos maneras de hacer esto:

1. Hard Reset: Desenergizar el micro y arrancar de nuevo.
2. Soft Reset: Poner el PC al valor 00000H.

Es el tipo de excepción de mas alto nivel de prioridad: -3

* **NMI** Interrupción No Enmascarable. Es la segunda excepción de mas alta prioridad después de reset: -2

Como su nombre lo indica, no puede ser deshabilitada por software. Suele ser implementada por el fabricante. Un ejemplo es el watchdog que genera un NMI que provoca un reset.

Reservada para el control del sistema: como fallas en el sistema de clock, inicialización de la ﬂash, etc.

* **Hard Fault:** Todas las clases de fallas, cuando el Handler de fallas correspondiente no se puede activar porque actualmente está deshabilitado o enmascarado por un enmascaramiento de excepciones.

Varios tipos de excepciones en los procesadores Cortex-M3 y Cortex-M4 son excepciones de manejo de fallas. Las excepciones de falla se activan cuando el procesador detecta un error, como la ejecución de una instrucción no definida, o cuando el sistema de bus devuelve una respuesta de error a un acceso a la memoria. El mecanismo de excepción de fallas permite que los errores se detecten rápidamente y potencialmente permite que el software lleve a cabo acciones correctivas

De forma predeterminada, la falla de bus, la falla de uso y la falla de administración de memoria están deshabilitadas y todos los eventos de falla desencadenan la excepción *HardFault*. Sin embargo, las configuraciones son programables y puede habilitar las tres excepciones de falla programables individualmente para manejar diferentes tipos de fallas. La excepción *HardFault* siempre está habilitada.

Uso típico para Uso indebido de memoria dinámica, incorrecta inicialización de periféricos.

**10.** Describa las funciones principales de la pila. ¿Cómo resuelve la arquitectura el llamado a funciones y su retorno?

Como en casi todas las arquitecturas de procesador, los procesadores Cortex-M necesitan memoria de Stack (pila) para funcionar y tienen punteros de pila (R13). La pila es un tipo de mecanismo de uso de memoria que permite que una parte de esta se utilice como búfer de almacenamiento de datos de tipo LIFO (último en entrar - primero en salir). Los procesadores ARM utilizan la memoria principal del sistema para operaciones de memoria de pila y tienen la instrucción PUSH para almacenar datos en la pila y la instrucción POP para recuperar datos de la pila. El puntero de pila seleccionado actualmente se ajusta automáticamente para cada operación PUSH y POP.

La pila se puede utilizar para:

• Almacenamiento temporal de datos originales cuando una función que se está ejecutando necesita utilizar registros (en el banco de registros) para el procesamiento de datos. Los valores se pueden restaurar al final de la función para que el programa que llamó a la función no pierda sus datos.

• Paso de información a funciones o subrutinas.

• Para almacenar variables locales.

• Para mantener el estado del procesador y registrar valores en caso de excepciones como una interrupción.

Los procesadores Cortex-M utilizan un modelo de memoria de pila llamado “full-descending stack” (pila descendente completa). Cuando se inicia el procesador, el SP se configura al final del espacio de memoria reservado para la memoria de la pila. Para cada operación PUSH, el procesador primero disminuye el SP y luego almacena el valor en la ubicación de memoria señalada por el SP. Durante las operaciones, el SP apunta a la ubicación de la memoria donde se enviaron los últimos datos a la pila.

En una operación POP, se lee el valor de la ubicación de memoria señalada por SP y luego el valor de SP se incrementa automáticamente.

Los usos más comunes de las instrucciones PUSH y POP son saber el contenido de los bancos de registros cuando se realiza una llamada a una función o subrutina. Al comienzo de la llamada a la función, el contenido de algunos de los registros se puede guardar en la pila usando la instrucción PUSH y luego restaurar a sus valores originales al final de la función usando la instrucción POP. Tenga en cuenta que para cada operación PUSH (almacenar en la memoria), debe haber un POP (lectura de la memoria) correspondiente y la dirección del POP debe coincidir con la de la operación PUSH.

Cada instrucción PUSH y POP puede transferir múltiples datos hacia/desde la memoria de la pila. Dado que los registros en el banco de registros son de 32 bits, cada transferencia de memoria se genera mediante transferencias de stack PUSH y stack POP y transfiere al menos 1 palabra (4 bytes) de datos y las direcciones siempre están alineadas con límites de 4 bytes. Los dos bits más bajos del SP siempre son cero.

**11.** Describa la secuencia de reset del microprocesador.

Lo primero que se hace el micro es inicializar el Stack Pointer y luego el Program Counter (PC) apunta al vector de Reset que se encuentra en la dirección 00000H. En la dirección del vector de Reset se encuentra la dirección a la primera instrucción del programa, es decir carga el PC con la dirección de inicio en memoria de código y comienza la ejecución del programa.

**12.** ¿Qué entiende por “core peripherals”? ¿Qué diferencia existe entre estos y el resto de los periféricos?

Los “core peripherals” serían los periféricos integrados en el mismo core del micro, como por ejemplo: el NVIC (Contorlador de Vectores de Interrupciones Anidadas), La FPU (Unidad de Punto Flotante), la MPU (Unidad e Protección de Memoria), el Systick Timer (Timer del sistema).

**13.** ¿Cómo se implementan las prioridades de las interrupciones? Dé un ejemplo

La arquitectura ARM Cortex-M cuenta con un controlador de interrupciones llamado NVIC (Controlador de interrupciones vectorizado anidado) que admite hasta 240 solicitudes de interrupción y de 8 a 256 niveles de prioridad de interrupción (dependiendo de la implementación real del dispositivo).

Básicamente se le asigna un nivel de prioridad a las interrupciones del sistema que la otorgan un nivel de privilegio de acceso a los recursos el micro, de tal manera que las de mayor prioridad tomarán el control de la CPU, mientras que las de menor prioridad deberán esperar a que las de mayor privilegio desocupen la misma. Los niveles de privilegio tienen números, siendo los de menor valor numérico los de mas alta prioridad, siendo las mas privilegiadas , de valores negativos.

Por ejemplo, nivel -3 mas alta prioridad, luego nivel -2… nivel 0… nivel 5 y así sucesivamente descendiendo en su nivel de prioridad a media que aumenta su valor numérico.

Vimos ejemplo de las mas prioritarias: Reset, NMI y Hard Fault.

**14.** ¿Qué es el CMSIS? ¿Qué función cumple? ¿Quién lo provee? ¿Qué ventajas aporta?

El CMSIS (Cortex Microcontroller Software Interface Standard) es una capa de software que proporciona una interfaz estándar para el desarrollo de sistemas embebidos basados en la arquitectura ARM Cortex-M. Fue desarrollado por ARM Holdings para facilitar el desarrollo de software y la portabilidad entre diferentes microcontroladores Cortex-M.

La función principal del CMSIS es servir de base para el desarrollo de una capa de abstracción de hardware (HAL) que permite a los desarrolladores acceder de manera uniforme a los periféricos y funciones del microcontrolador, independientemente del fabricante o modelo específico del chip. Esto simplifica el desarrollo de software y facilita la portabilidad del código entre diferentes microcontroladores Cortex-M.

El CMSIS es proporcionado por ARM Holdings, la empresa que desarrolla la arquitectura ARM. que ofrece el CMSIS como una biblioteca de software de código abierto que los desarrolladores pueden utilizar de forma gratuita.

Las ventajas del CMSIS incluyen:

* 1. Portabilidad: Permite que el código escrito para un microcontrolador Cortex-M se pueda reutilizar fácilmente en otros microcontroladores Cortex-M, lo que facilita la migración entre diferentes chips y fabricantes.
* 2. Eficiencia: Proporciona una capa de abstracción de hardware optimizada que permite un acceso eficiente a los periféricos del microcontrolador, maximizando el rendimiento y la eficiencia del sistema.
* 3. Estándares: Define una interfaz estándar para el desarrollo de software en microcontroladores Cortex-M, lo que facilita la colaboración entre diferentes desarrolladores y empresas.
* 4. Soporte de herramientas: El CMSIS es compatible con una amplia variedad de herramientas de desarrollo, como compiladores, depuradores y entornos de desarrollo integrados (IDE), lo que facilita la integración y el flujo de trabajo en el desarrollo de software.

En resumen, el CMSIS es una capa de software que proporciona una interfaz estándar para el desarrollo de sistemas embebidos basados en la arquitectura ARM Cortex-M. Facilita el desarrollo de software, la portabilidad del código y mejora la eficiencia del sistema. Es proporcionado por ARM Holdings como una biblioteca de software de código abierto.

**15.** Cuando ocurre una interrupción, asumiendo que está habilitada ¿Cómo opera el microprocesador para atender a la subrutina correspondiente? Explique con un ejemplo

Cuando ocurre una interrupción y esta es atendida la CPU realiza un cambio de contexto. Para hacer esto se produce lo que se denomina un stacking, que consiste en salvar los contenidos de determinados registros de la CDU en el stack, como ser: los registros r0 a r3, r12, ), el LR, Return address, xPSR (por defecto), el registro de estado o APSR (Application Program Status Register), que contienen la info necesaria para regresar al contexto anterior luego de atender la interrupción. También se pueden salvar registros adicionales, si se considera necesario, pero esto, debe hacerse en forma manual por parte del usuario.

**16.** ¿Cómo cambia la operación de stacking al utilizar la unidad de punto flotante?

En este caso se salvan mas registros durante el stacking: aparte de los mencionados en el punto anterior se slavan so registros s0 a s15, el FPSCR, y opcionalmente otros que el usuario considere necesario salvar.

**17.** Explique las características avanzadas de atención a interrupciones: tail chaining y late arrival.

# Interrupción de tipo tail chaining

Cuando se produce una excepción pero el procesador está manejando otra excepción de la misma o mayor prioridad, la excepción entrará en el estado pendiente. Cuando el procesador termina de ejecutar el controlador de excepciones actual, puede proceder a procesar la solicitud de excepción/interrupción pendiente. En lugar de restaurar los registros de la pila (desapilar) y luego empujarlos nuevamente a la pila (apilar), el procesador omite los pasos de desapilado y apilamiento e ingresa al controlador de excepciones de la excepción pendiente lo antes posible. De esta manera, la brecha de tiempo entre los dos manejadores de excepciones se reduce considerablemente. Para un sistema de memoria sin estado de espera, la latencia de la cadena de cola es de sólo seis ciclos de reloj.

La optimización del encadenamiento de cola también hace que el sistema sea más eficiente energéticamente porque reduce la cantidad de accesos a la memoria de la pila y cada transferencia de memoria consume energía.

# Interrupción tipo late arrival

Cuando se produce una excepción, el procesador acepta la solicitud de excepción e inicia la operación de apilamiento. Si durante esta operación de apilamiento se produce otra excepción de mayor prioridad, la excepción de llegada tardía de mayor prioridad será atendida primero.

Por ejemplo, si la Excepción n.° 1 (prioridad más baja) tiene lugar unos ciclos antes de la Excepción n.° 2 (prioridad más alta), el procesador se comportará como se muestra en la Figura 8.13, de modo que el Controlador n.° 2 se ejecute tan pronto como se complete el stacking.

**18.** ¿Qué es el systick? ¿Por qué puede afirmarse que su implementación favorece la portabilidad de los sistemas operativos embebidos?

**19.** ¿Qué funciones cumple la unidad de protección de memoria (MPU)?