Preguntas orientadoras

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

Cortex A:

Son procesadores de alto rendimiento optimizados para aplicaciones que emplean un sistema operativo de propósito general en sistemas embebidos de alta performance.

Su denominación "A" proviene de *Application*, se pueden encontrar en dispositivos como celulares o tables.

Se destaca la optimización para ejecutar diversas aplicaciones al mismo tiempo a costa de una disminución del tiempo de respuesta de las mismas, aspecto secundario en dispositivo de usuario.

Cortex R:

Son procesadores orientados a sistemas de tiempo real donde prima la necesidad de implementar soluciones con requerimientos temporales estrictos.

En los sistemas de tiempo real la respuesta a eventos o estímulos debe ser en un tiempo acotado y preestablecido comportándose de forma determinística.

Su denominación "R" proviene de *Real Time*, se pueden encontrar en sistemas críticos, como dispositivos médicos o sistemas de operación automovilísticos.

Cortex M:

Son procesadores orientados a dispositivos de consumo masivo y sistemas embebidos compactos. Son procesadores de uso general diseñados para alta densidad de código y con gran cantidad de periféricos. Su denominación "M" proviene de *Microcontrollers*, se pueden encontrar en dispositivos como celulares o tables.

Cortex M

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

Las principales diferencias que pueden destacarse entre las familias de procesadores Cortex M0, M3 y M4 son:

- **Arquitectura:** los procesadores Cortex M0 poseen arquitectura ARMv6-M tipo Von Neumman, mientras que los M3 tienen la arquitectura ARMv7-M y los M4 la arquitectura ARMv7E-M ambas tipo Harvard. Esto se traduce en un compromiso entre costo y capacidad de procesamiento.
- **Set de instrucciones:** los M0 manejan solamente el set de instrucciones Thumb-1 de 16 bits mientras que las familias M3 y M4 incorporan el set Thumb-2.
- **Soporte para RTOS:** las familias M3 y M4 poseen periféricos específicos como el SysTick timer o MPU(Unidad de memoria protegida) que facilitan la implementación de sistemas operativos.
- ***Hardware de procesamiento:*** los Cortex M4 incorporan unidad punto flotante (opcional), DSP y aritmética saturada, que no están presente en los M3 y M0.

En lineas generales, la familia M0 posee menor capacidad de procesamiento y performance ya que está destinada a aplicaciones de bajo costo y consumo. En cambio los procesadores M3 y M4 se a orientan sistemas donde se requiere mayor procesamiento e incorporan funcionalidades para aplicaciones diversas.

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

El set de instrucciones Thumb está compuesto por instrucciones de 16 bits con funciones específicas. Se dice que permite mayor densidad de código en comparación de un set de instrucciones de 32 bits exclusivamente. En muchas operaciones simples no resulta necesario emplear muchas instrucciones, un set de 16 bits es suficiente y permite optimizar la cantidad de código que se debe almacenar.

En la familia Corte-M3/4 incorpora el set de instrucciones Thumb-2 que permite instrucciones de 16 y 32 bits mejorado la optimización desde el punto de vista de la densidad del código, eficiencia y performance.

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

Una arquitectura *load-store* significa que para realizar una modificación de un dato almacenado debe cargarse previamente en un registro, procesarse y luego volver a escribirse en memoria utilizando una serie de operaciones separadas.

Por ejemplo, para incrementar un valor almacenado en una memoria SRAM, el procesador necesita usar una instrucción para leer el datos desde la SRAM y colocarlo en un registro interno, una segunda instrucción para incrementar el valor del registro y finalmente, una tercera instrucción para escribir el valor modificado en la posición de memoria donde se encontraba.

Dentro del set de instrucciones del dispositivo no existen instrucciones que permitan directamente modificar una valores del mapa memoria.

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

En la familia de procesadores Cortex-M el mapa de memoria esta compuesto por un ancho de palabra de 32 bits y puede contener hasta 4 Gbytes de espacio, limite impuesto por la cantidad de direcciones posibles que se pueden acceder con un 32 bits de direccionamiento ($2^{32} = 4.29 Gbits$).

El mapa de memoria se encuentra particionado en diferentes secciones donde se alojan (*"mapean"*) todos los componentes del sistema, como memoria flash, memoria SRAM, periféricos, etc. Exceptuando restricciones especificas sobre algunas zonas de memoria es posible acceder a todos los registros mediante un mismo bus de datos y set de instrucciones, característica que permite acceder fácilmente al contenido o registro del dispositivo empleando punteros en lenguaje C.

La partición del mapa de memoria está determinada por el fabricante de cada microcontrolador que contiene un procesador ARM y debe consultarse en la hoja de datos del dispositivo.

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

Las principales ventajas de la utilización de los "shadowed pointers" que puede traducirse como punteros ocultos o sombreados se dan en sistemas embebidos que utilicen sistemas operativos o RTOS. En estos sistemas el manejador de excepciones utiliza el MSP con un stack principal, mientras que las tareas emplean el PSP con un espacio de memoria asignado funcionando como stack para cada una. Cuando se realiza un cambio de contexto se utiliza el PSP actualizando el stack de las tareas, sin alterar el MSP y stack del sistema operativo. Esta forma de operación se traduce en un sistema más robusto y confiable. Si una tarea produce un error que genera la corrupción de los datos de su stack, el sistema operativo puede recuperar su estado empleando el MSP al conservar el stack principal sin alteraciones.

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

La familia de procesadores Cortex-M3/4 poseen dos modos de operación, modo privilegiado y no privilegiado, también llamado modo usuario.

El modo de funcionamiento privilegiado se caracteriza porque permite el acceso a todo el mapa de memoria por parte de la/s aplicación/es del usuario. En contraposición, en el modo de operación de usuario existen áreas determinadas de memoria protegidas que no son posibles acceder durante la ejecución del programa.

Una aplicación típica del uso de los modos de usuario, se da al emplear un sistema operativo. Es deseable que las tareas o aplicación del usuario no accedan a los componentes del sistema operativo ubicándolo en un área protegida de memoria solo accesible en el modo de privilegiado.

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

Un modelo de registro ortogonales permite que cualquier instrucción que utilice registros como argumento puede operar con cualquier registro independientemente del uso específico de la instrucción. En este tipo de modelo no existen instrucciones para operar con registros particulares, por ejemplo, una instrucción que lea un registro de configuración del sistema, para realizar este tipo de operación se emplea una instrucción de lectura que permite acceder cualquier registro o posición de memoria.

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

La principal ventaja de las instrucciones condicionales IT es que no se destruye el proceso de *pipeline* comparado con un salto condicional mediante una instrucción *branch*, esto se traduce en una mejora de la performance controlando el flujo del programa de forma sencilla y eficiente.

Un bloque de instrucciones IT puede encadenar hasta 4 instrucciones condicionales subsecuentes, permitiendo verificar diferentes condiciones mediante el uso de sufijos como EQ, NE, etc.

Ejemplo:

```
...
ITTE EQ
moveq r0,10      @Se ejecuta si Z = 1
addseq   r0,1    @Se ejecuta si Z = 1
movne    r0,r1   @Se ejecuta si Z = 0
...
```

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

Las excepciones reset, NMI y Hardfault son las de mayor prioridad del sistema. Todas tienen prioridad negativa y no configurable por el usuario siendo la de menor valor la más prioritaria.

Reset: es la excepción de mayor prioridad (-3) se dispara cuando una de los tres tipos de reinicio del sistema se ejecuta (Power On Reset, System Reset, Processor Reset).

NMI (Non-Maskable Interrupt): posee prioridad -2, puede ser generada por periféricos internos que monitorean la integridad del sistema, por ejemplo *watchdogs* o *Brown-Out Detector (BOD)*.

Hardfault: posee prioridad -1, es disparada cuando existe un evento de falla*(fault event)*, como ejecución de una instrucción indefinida, acceso erróneo al bus del sistema o a una posición de memoria.

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

Después de una señal de reset y antes de la ejecución del programa principal se ejecuta la secuencia de reset del microcontrolador.

En los procesadores Cortex-M se leen las primeras dos palabras del mapa de memoria (direcciones 0x00000000 y 0x00000004), esta posiciones corresponden a la tabla de vectores(vector table) y contienen el valor inicial del puntero a la pila principal (*Main Stack Pointer*-MSP) y del contador de programa(*Program Counter*-PC).

Luego que estas dos palabras son leídas, el procesador configura el MSP y el PC con dichos valores. La configuración inicial del MSP es necesaria por si ocasionalmente se produce alguna excepción, como NMI o HardFault justo luego del reset.

La mayoría de los entornos de desarrollo en C realizan la actualización del MSP antes de ejecutar el programa *main()* en la función llamada *startup*. La inicialización de la pila en dos pasos permite utilizar una memoria externa para tal fin.

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

Se entiende como *"core peripherals"* aquellos componentes que forman parte del procesador pero no son el núcleo en si mismo. Su objetivo principal es brindar soporte y ampliar las capacidades del procesador generalmente implementando funcionalidades específicas. Algunos ejemplos son:

- NVIC (Nested Vector Interrupt Controller).
- SysTick (System timer).
- System Control Block (SCB)
- MPU (Memory Protection Unit).

La integración de estos periféricos favorece a la portabilidad de software dentro de la familia de procesadores de ARM. La principal diferencia con el resto de los periféricos es que están orientados a interactuar con el procesador en cambio los periféricos genéricos se orientan a interactuar con el entorno implementando funcionalidades específicas como UART, ADCs, SPI, GIOP, etc.

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

Después de un proceso de reset o inicio del sistema, todas las interrupciones se encuentras deshabilitadas y un nivel de prioridad con el valor 0. Antes de ser usadas, opcionalmente pueden programarse los niveles de prioridad.

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

El CMSIS *(Cortex Microcontroller Software Interface Standard)* es una librería de *software* desarrollada por ARM que funciona como una capa de abstracción del hardware para los procesadores de la familia Cortex-M.

Es una herramienta que permite a los fabricantes de microntroladores, herramientas de software (como depuradores o compiladores), sistemas operativos y al desarrollador de firmware final tener un estándar en común sobre el cual basar sus desarrollos asegurando cierto grado de la interoperabilidad.

La principal ventaja que aporta el uso de CMSIS es lograr una interfaz simple y consistente con el procesador y los periféricos, lo que redunda en reutilización del software, acorta el proceso de aprendizaje para los desarrolladores de microcontroladores y software final, brindando robustez al sistema al utilizar plataformas probadas.

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

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

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

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

El SysTick es un temporizador decremental de 24 bits que puede operar a la frecuencia del reloj del sistema. Es incorporado como periférico interno en los procesadores Cortex-M, integrado como parte del módulo NVIC y puede generar una excepción del sistema de forma periódica.

En dispositivos que incorporan un sistema operativo se utiliza para realizar una interrupción periódica que permita la ejecución de diferentes tareas implentando múltiples cambios de contexto en modo de ejecución de usuario.

La incorporación del periférico en el procesador favorece la portabilidad del software entre diferentes dispositivos Cortex-M3/M4. Utilizar un temporizador externo para un sistema operativo embebido, implica una mayor cantidad de requerimientos o configuraciones que debería definir e implementar el fabricante del microntrolador y posteriormente el diseñador del software.

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

La unidad *Memory Protection Unit*(MPU) consiste de un área de memoria que integra mecanismos de protección de forma que no sea accesible en todos los modos de ejecución del código de usuario.

La inclusión de la unidad MPU es una característica opcional en la familia de procesadores Cortex-M3 y M4, el fabricante del microcontrolador decide si está disponible o no. En el caso de contar con la MPU, el desarrollador posee la capacidad de dividir el espacio de memoria en diferentes regiones y definir permisos de acceso para cada una. Cuando se produce un intento de acceso prohibido se genera una excepción (fault exception) que puede disparar la ejecución de acciones de análisis, advertencia o correctivas.

Un escenario de uso muy común resulta en sistemas embebidos que implementan un sistemas operativo, usualmente se aloja en un área de memoria con privilegios que no pueda ser accedida por las otras tareas del

usuario. Otro escenario de uso es la creación de regiones de memoria de solo lectura, que permitan prevenir borrados accidentales de memoria SRAM o sobreescritura de código de instrucciones.

19. ¿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?

La unidad MPU en los procesadores Cortex M3 y M4 soporta hasta ocho regiones programables de memoria, cada una con dirección de inicio, tamaño y configuraciones programables.

Las regiones de memoria de la MPU pueden superponerse. Si una ubicación de memoria se encuentra dentro de dos regiones programadas, los atributos y permisos del acceso a la memoria se tomarán de la región con el número más alto.

Por ejemplo, si una dirección de transferencia está dentro del rango de direcciones definido para la región 1 y la región 4, se usará la configuración de la región 4.

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

La excepción PendSV (Pended Service Call) posee nivel de prioridad configurable y es ejecutada por software. Suele utilizarse en sistemas operativos de tiempo real configurada como interrupción de baja prioridad para realizar tareas largas cuando el procesador se encuentra disponible y no haya otra interrupción que atender.

21. ¿Para qué se suele utilizar la excepción SVC? Expliquelo dentro de un marco de un sistema operativo embebido.

La excepción SVC*(Supervisor Call)* es de tipo 11 y su nivel de prioridad puede ser programado. Puede dispararse por la instrucción de software SVC y posee un comportamiento levemente distinto a las interrupciones de software del módulo NVIC(por ejemplo, *Software Trigger Interrupt Register,STIR*). Las interrupciones pueden ser imprecisas debido a que se ejecutan algunas instrucciones antes de su atención, por ejemplo verificación de la existencia de otra interrupción pendiente. La excepción SVC es precisa y se ejecuta inmediatamente luego de ser llamada, salvo que haya otra excepción de mayor prioridad.

En muchos sistemas embebido que poseen un sistema operativo, este mecanismo se suele utilizar para que las tareas de usuario puedan acceder a los recursos del sistema, manteniendo la seguridad y robustez.

Generalmente, el acceso a los recursos de hardware es gestionado por el sistema operativo en modo no privilegiado usando la MPU. Las tareas de la aplicación (que se ejecutan en modo no privilegiado) podrían acceder a los recursos protegidos mediante un servicio del sistema operativo vía la SVC. Con este mecanismo el acceso es más robusto y seguro impidiendo acceso no autorizados a recursos críticos.

Otra ventaja de este mecanismo de acceso es que el programador de las tareas de usuario puede abstraerse de los detalles del acceso al hardware ya que el sistema operativo es quién provee el servicio necesario para la excepción SVC(mediante el *driver* del periférico) facilitando el desarrollo de las aplicación.

ISA - *Instruction Set Architecture*

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

Los sufijos son opciones de configuración que se anexan a las instrucciones modificando aspectos de la ejecución de la misma. Existen de diferentes tipos, por ejemplo sufijos de tamaño que modifican el tamaño del operador con el que trabaja la instrucción o sufijos de condición (ver pregunta 2).

Ejemplo:

```
...
ldrh r1          @ Carga a un dato de 16 bits(halfword) sin signo
...              @ empleando el sufijo h
...
strsh r1,[r0]    @ Almacena un dato de 16 bits son signo en la direccion r0
...
```

2. ¿Para qué se utiliza el sufijo 's'? Dé un ejemplo

Normalmente, las operaciones de procesamiento de datos no afectan los campos de condición N,Z,C y V (exeptuando las comparaciones dónde esto es el único efecto). La incorporación del sufijo 's' permite actualizar los campos de condiciones luego de ejecutar la instrucción correspondiente, naturalmente esta capacidad constituye un ahorro de instrucciones necesarias, por ejemplo al que implementar un bloque de ejecución condicional.

```
        ...
.label:
        mov r1,4                @ carga el el registro r1 con el numero 4
        ...
        subs r1, 1              @ decrementa el registro actualizando los flags
        bne .label              @ verifica si el flag es Z=1
        ...
```

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

Las instrucciones de aritmética saturada tienen la función de mantener el valor cuando se produce un desbordamiento *(overflow)* del dato con el cual operan, manteniendo el valor máximo posible. Suelen utilizarse en procesamiento de señales de audio donde es preferible que el dato mantenga el valor máximo en vez de reiniciarse al producirse un desbordamiento.

Ejemplo de saturación a 8 bits:

```
        ...
        mov r0 , 60             @coloca el 60 en r0
        mov r1 , 8              @coloca el 8 en r1
        mul r0 , r1             @multiplica r0 por r1 y lo almacena en r0
        usat r0 , #8 ,r0        @satura el valor de r0 a 8 bits r0 = 255
        ...
```

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?

El set de instrucciones ISA de los procesadores Cortex-M posee funcionalidades que facilitan la interoperabilidad entre los lenguaje de programación C y assembler. En cuanto a la interfaz entre funciones implementadas en assembler se definen en el documento *Procedure Call Standard for the Arm® Architecture* para poder ejecutarse desde un un código en C.

Cuando se llama una función elaborada en assembler desde un programa en C, los argumentos escritos en C se reciben en los registros r0, r1, r2 y r3 dentro de la función. El resultado se devuelve en los mismo registros dependiendo del tipo de dato a retornar la cantidad necesaria. Si la precisión del tipo de dato es menor o igual a 32 bits, se utiliza el registro r0. Si el tipo de dato es de 64 bits, el resultado se devuelve en los registros r0 y r1. Y si el tipo de dato es de 128 bits, se emplean los registros r0, r1, r2 y r3.

En el caso que la función reciba más de 4 parámetros, estos se deben colocar en la pila (stacking) y desde la función en assenmbler se deberán acceder con la instrucción *Push*.

Si se realizan operaciones con los registros r4 a r11, el contenido de estos debe colocarse en la pila con *Push* (stacking) y al final de la función se deben restaurar mediante *Pop* (unstacking). Esta operación se realiza de forma implicita para los registros de r0 a r3.

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

Las SIMD *(Single Instruction Multiple Data)* son un tipo de instrucción que permiten realizar múltiples operaciones de datos en paralelo dividiendo el largo total de la palabra del processador(32 bits) en palabras de menor tamaño (8 o 16 bits).

La aplicaciones mas comunes de este tipo de instrucciones es en procesamiento de audio realizando cálculos del canal derecho e izquierdo al mismo tiempo. Otra aplicación típica es en el procesamiento de imágenes, donde

los colores RGB de cada pixel de una imagen pueden representarse con 8 bits, la utilización de instrucciones SIMD permiten procesarse en paralelo.