COMENTARIO TECNICO

Microcontroladores de 32 bits ARM...
... O como no temerle al cambio!!

Por Ing. Marcelo E. Romeo[1] - Ing. Eduardo A. Martínez[2]

Parte 1

1. Un poco de historia

1.1 Arquitectura RISC

En 1975, IBM[iii] se encontró con el requerimiento de Ericson de una central telefónica que procesara 300 llamadas por segundo con un promedio de unas 20.000 instrucciones por llamada, lo cual hacia necesario un procesador que realizara 12 millones de instrucciones por segundo; en dicha época, ningún procesador de la época tenía esa potencia.

Un tiempo antes, se realizó un estudio estadístico de la frecuencia de ocurrencia de las instrucciones sobre un programa generado por un compilador de alto nivel en una computadora basada en un procesador de importantes prestaciones para la época.

Contrariamente a lo imaginado, la computadora realizaba mucho más frecuentemente operaciones sencillas y muy esporádicamente operaciones complejas. El corolario fue que el compilador no empleaba las instrucciones muy poderosas (que emplearía un programador en assembler), sino que utilizaba instrucciones sencillas.

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

Tabla 1 - Frecuencia de ocurrencia de las instrucciones

1 Universidad de Belgrano – Universidad de San Martín – UTN - FRBA
2 Universidad de Belgrano

La central solicitada no requería muchas prestaciones y los cálculos que se requerían no eran muy complejos, básicamente se necesitaba, añadir y mover datos y combinar campos.

Para resolver el requerimiento, IBM diseñó un procesador especial optimizando las instrucciones más frecuentemente utilizadas y descartando las instrucciones complejas difíciles de implementar en silicio y de uso poco frecuente. Las mismas deberían reemplazarse por varias instrucciones sencillas pero muy rápidas.

Como suele ser frecuente, el proyecto no se cristalizó en un equipo comercial, pero dejó un importante sedimento tecnológico: computadoras con una arquitectura de pocas instrucciones pero muy veloces. Nació la arquitectura RISC (*Reduced Instruction Set Computers*).

El cambio laboral del diseñador Joel Birnbaum de IBM a HP y el nacimiento dentro de HP del PA-RISC (*Precision Architecture RISC* o RISC de arquitectura precisa) conjuntamente con el apoyo de las Universidades de Berkeley y Stanford a la investigación de RISC como arquitectura por sus aplicaciones científicas y la energía puesta por HP por su comercialización en el mercado de servidores, crearían la confianza suficiente en el mercado para la eclosión del estándar.

Las principales características que comenzarían a hacer destacar a la arquitectura RISC fueron:

- Load / Store: Operaciones registro a registro separadas de los accesos a memoria.
- Repertorio de instrucciones cuidadosamente seleccionados, optimizados e implementados en memoria
- Instrucciones en formato fijo.
- Modos de direccionamiento simple
- Caches de instrucciones y datos separadas (Arquitectura Harvard)
- Paralelismo (Pipeline)

1.2 La familia de microcontroladores ARM

1.2.1 Orígenes y principios

En 1990, ARM una empresa surgida de *Acorn Computers Limited of Cambridge, England,* diseñó el primer microcontrolador comercial de 32 bits de ancho de palabra basado en arquitectura RISC[3]

El concepto tecnológico - comercial novedoso que implementaron fue la de diseñar el núcleo de la familia de microcontroladores sin fabricarlos, sino que cedieron la producción a empresas de semiconductores de primera línea (sólo como ejemplo, NXP, Atmel, Texas Instrumentes, Freescale, Analog Devices, etc)[4]. Cada fabricante respetaría el núcleo original y el repertorio de instrucciones, pero pudieron incorporarle sus propios periféricos (Conversores A/D y D/A, UART, SPI, I^2C, etc.), lo cual, por un lado, fue un estímulo al crecimiento de soluciones en hardware, pero, sin embargo, produjeron incompatibilidades entre componentes similares de distintas marcas.

[3] Acorn había hecho gran fama y fortuna al desarrollar una pequeña computadora de aplicaciones estudiantiles y hobbistas en apoyo a programas didácticos presentados en la BBC de Londres.

[4] Una idea similar a la que tuvo Linus Torvalds con sus sistema operativo Linux

1.2.2 Fuentes de la creación de la familia ARM

De la arquitectura RISC tomaron:

- Arquitectura *load-store*: Las instrucciones que acceden a memoria son distintas de las instrucciones que procesan los datos
- Formato fijo de palabra de instrucción. Todas de 32 bits
- Máquina de tres direcciones (1er operando, 2º operando y resultado) en la que en la instrucción se trabajará con registros punteros a operandos pero no directamente con memoria

A su vez incorporaron algunos conceptos novedosos como ser:

- **Ortogonalidad:** Las instrucciones siguen un molde repetitivo. Por ejemplo la forma de encarar la suma (operandos, formato), será la misma que para la OR exclusiva. No existe registros especiales y esas operaciones pueden realizarse con todos los registros.

- **Implementación de varios modos de trabajo o jerarquías**. Aparece por primera vez en un microcontrolador el modo supervisor y el modo usuario. El primero, con plenos poderes, supervisará la actuación de los programas de usuario, impidiendo que el sistema colapse y que solo pudieran hacerlo aplicaciones de usuario que podrían ser rescatadas por el supervisor.

- **Excepciones**: Son los casos particulares de control de flujo (saltos) en los que un efecto posiblemente no deseado de la ejecución del programa lleva a una situación de falla, como un fallido acceso a memoria, un código de operación inválido, etc., llevan a que el programa se desvíe de su curso original y pase a ejecutar un tramo de programa que atienda esta situación excepcional. Las interrupciones son un caso particular de las excepciones.
- **Thumb**: Aparece la posibilidad de operar con códigos de operación comprimidos a 16 bits[5]. Si bien con ello parecería duplicarse el tamaño de la memoria de programa, la realidad es que las instrucciones son menos poderosas que las de 32 bits, de forma que se requiere más de una instrucción de 16 bits para equiparar a una de 32 bits.

- **Bajo consumo:** Se partió de la necesidad de bajo consumo, por lo que se bajó la tensión de alimentación (1,8 V para el núcleo) y un interesante manejo de la frecuencia de trabajo (ya que la potencia disipada es función directa de la frecuencia) durante el tiempo de ejecución. Se llega a un consumo típico de 0,28 mW/MHz.

- **Interrupciones:** Se dispondrá de dos interrupciones vectorizadas IRQ y FIQ las cuales tendrán habilitaciones separadas y FIQ será de mayor prioridad que IRQ. Se tomarán como casos particulares de excepciones.

- **Eficiencia en la generación de código en C:** Esta arquitectura fue concebida para trabajar en lenguaje C con una densidad de código muy superior a los microcontroladores de 8 bits previos en los que el lenguaje C apareció mucho después del diseño del microcontrolador.

- **Herramientas de depuración de bajo costo e incorporadas:** Cada microcontrolador dispone de una interfaz JTAG, originada para depurar DSPs y que permiten generar herramientas de depuración en tiempo casi-real de muy bajo costo.

[5] De la misma forma que la música en formato mp3 no mantiene la calidad de un CD original.

2. Características Básicas de la familia de procesadores ARM

2.1 Arquitectura

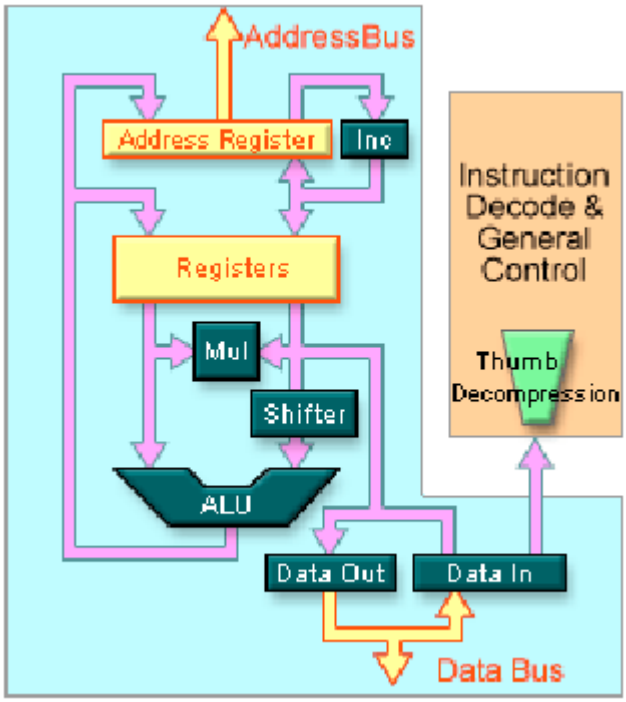

Fig 1: Arquitectura básica de un microcontrolador ARM7

En la **figura 1** vemos a nivel de bloques elementales la arquitectura de un microcontrolador de la familia ARM7. Observamos muchas coincidencias con las arquitecturas tradicionales de los microcontroladores. Sin embargo debemos destacar algunas particularidades

- La totalidad de los registros son de 32 bits.
- La unidad aritmética lógica recibe uno de los operandos directamente del banco de registros mientras que el otro operando pasa por un registro de desplazamiento. Ese segundo operando podrá pasar transparentemente por ese registro (sin verse afectado) o bien ser rotado a derecha o izquierda hasta en 5 lugares, pudiendo multiplicarlo o dividirlo por 2^n. Esas operaciones se realizan en una sola instrucción y en un solo ciclo de máquina. Este pasaje por el *barrel shifter* es similar al que se encuentra en los procesadores digitales de señales (DSP) y que suele utilizarse para implementar filtros digitales. Aquí también podremos implementar filtro digitales de buenas características hasta el rango de audio.
- Decompresor Thumb. El procesador es único ya sea que se empleen códigos de operación de 16 ó 32 bits. Cuando llegan los de 16 bits, automáticamente se los expande a 32 bits (obviamente que con espacios rellenados por el propio procesador) y se los ejecuta normalmente

2.2 Modos de operación.

La familia introduce la novedad de disponer de siete modos distintos de operación.

- **User**: Modo NO privilegiado para la mayoría de las aplicaciones.
- **FIQ**: Se ingresa con una interrupción de alta prioridad (fast).
- **IRQ**: Se ingresa con una interrupción de baja prioridad (normal).
- **Supervisor**: Se ingresa en reset y cuando se ejecuta una interrupción por software que permitirá subir en nivel de jeraquías (SWI).
- **Abort**: Se emplea para gerenciar violaciones en el acceso a memoria.
- **Undef**: Se emplea para gerenciar instrucciones indefinidas.
- **System**: Modo privilegiado que emplea los mismos registros que el modo usuario
.

Esta asignación de modos de trabajo se ve acompañada con una disposición de registros que permite un rápido cambio de contexto.

Inmediatamente después del reset, el procesador se inicia en modo Supervisor (donde tiene todas las atribuciones). Realiza las inicializaciones y operaciones propias del modo y luego cambia los bits de modo (fig 3) a modo usuario. El proceso inverso obviamente no es sencillo, sino que deberá solicitar autorización al supervisor para operaciones inicialmente inhabilitadas por medio de un *software interrupt* (SWI) que temporariamente lo lleva a modo supervisor.

2.3 Registros.

La familia ARM7 dispone de 16 registros de 32 bits cada uno de ellos que se designan desde R0 a R15. En principio son todos idénticos y sólo dos tienen funciones específicas que son el R15, que se emplea como Contador de Programa (*Program Counter*), y el R14 (*Link Register*), que es utilizado para almacenar la dirección de retorno cuando se llama a una subrutina o se genera una excepción. Cabe destacar que es responsabilidad del programador resguardar las direcciones de retorno en el caso de pretender implementar subrutinas anidadas.

Si bien puede utilizarse como puntero a la pila cualquier registro, se empleará el registro R13 por omisión y para ciertas instrucciones como tal.

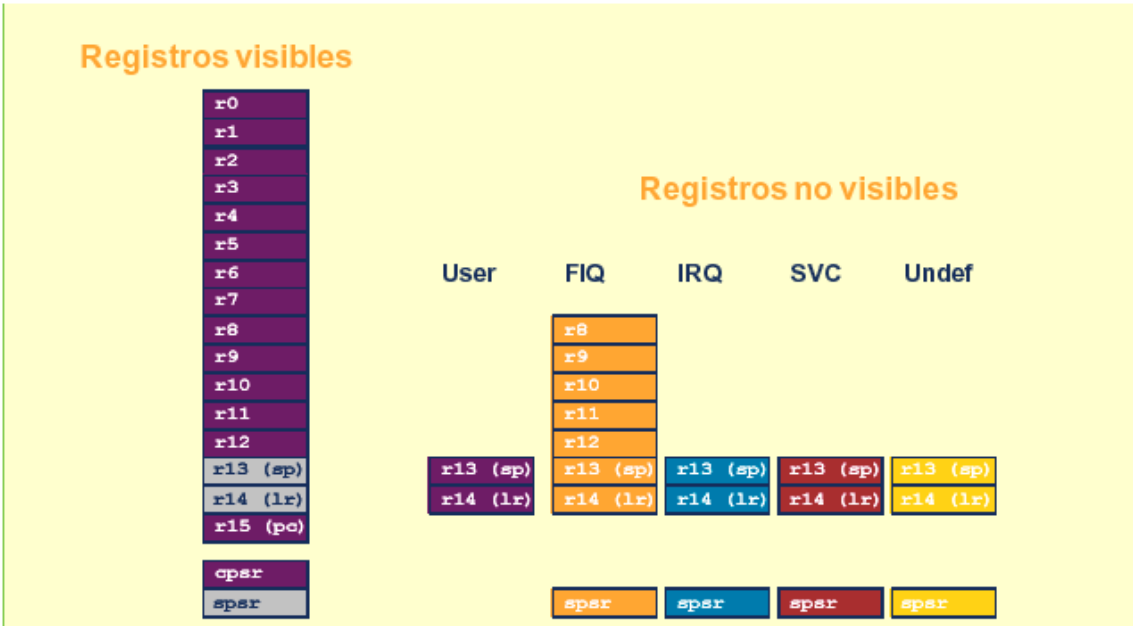

Fig 2. Registros del ARM7

En la **figura 2** vemos los registros del procesador. En todo momento se tendrán 16 registros disponibles más dos registros de estado que almacenarán los tradicionales *flags* de acarreo, cero, signo, etc. y los bits que indican el modo de trabajo y las máscaras de interrupción.

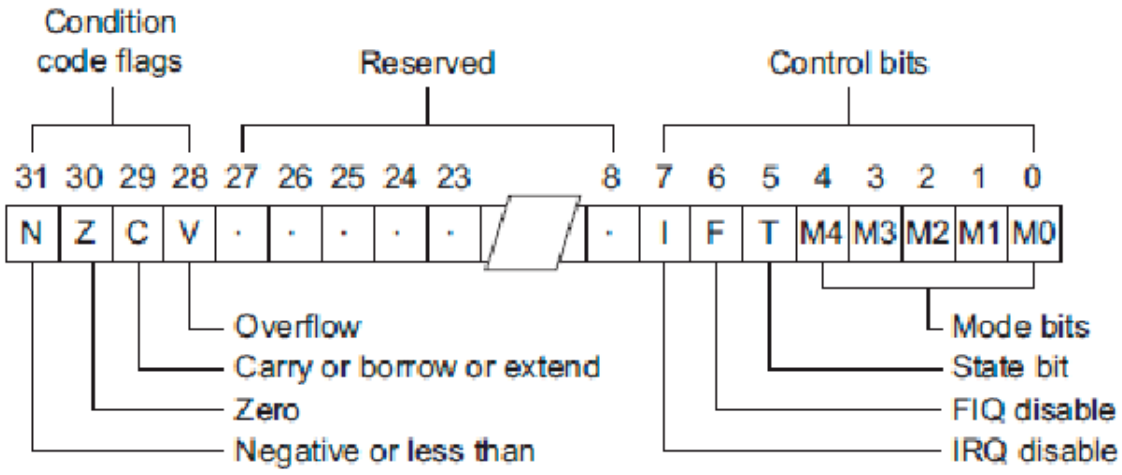

Fig 3. Registro de estado del programa (CPSR)

Para tratar de entender el porqué de esta distribución de registros imaginemos que se está ejecutando un programa en modo usuario; si aparece un pedido de interrupción FIQ y el mismo está desenmascarado (habilitado, o sea que el bit F de la figura 3 vale 0), el procesador pasará a atender ese pedido de interrupción.

Para ello migrará su modo de trabajo de **user a fiq.** Ello implicará:

- Que se reemplacen los registros R8_usr a R14_usr por los registros R8_fiq a R14_fiq.
- El R15 (Contador de Programa) se almacene en el R14_fiq (*Link Register*) para tener una idea de donde se debe retornar (ver 2.4 Pipeline) y el estado actual del procesador (CPSR) se copiará en el SPSR_fiq.
- El modo de operación se cambiará modificando los cinco bits menos significativos del CPSR de la figura 3.
- El contador de programa se forzará a un valor comprendido entre 00 y 0x1C dependiendo de la razón de cambio de modo de trabajo, según se ve en la figura 4.
- Se deshabilitan las interrupciones IRQ y en este caso, también FIQ.

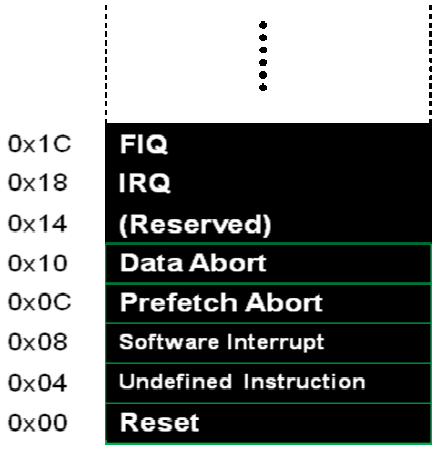

Fig 4. Tabla de vectores

Lo trascendente de esta en que en un mínimo tiempo[iii] se ha cambiado el contexto de ejecución del programa ya que se han salvado los registros R8 a R14 y el registro de *flags* para comenzar a procesar la interrupción. Pensemos el tiempo que se demoraría en salvaguardar en la pila tantos registros (¡De 32 bits!) en un procesador tradicional.

Debemos destacar que el el espacio reservado a cada vector (4 bytes) solo cabe una instrucción por lo que deberá colocarse un salto al *handler* de la excepción.

En los demás modos de operación se salvaguardarán, por omisión, sólo el *Link Register* y el *Stack Pointer*, de lo que podemos concluir que cada modo de operación podrá tener su propia pila en la que podrá almacenar variables locales y de uso exclusivo.

2.4 Pipeline.

Cuando mencionamos algunas de las características heredadas por ARM mencionamos el paralelismo que se corporiza en una tubería o *pipeline* que se muestra en la **Figura 5.**

Por la misma, y respetando el ciclo perpetuo que realiza un microprocesador, el proceso de ejecución de una instrucción se divide en tres etapas:

- Búsqueda de código de operación (en forma binaria, qué debe hacer)
- Decodificación (interpretar que debe hacer)
- Ejecución propiamente dicha

ARM postuló realizar las tres sub-operaciones en forma simultánea pero para distintas instrucciones. Es decir, buscar el código de operación de la instrucción 3, mientras se decodifica el código de operación leído anteriormente de la instrucción 2 y se ejecuta la instrucción 1 leída y decodificada previamente.

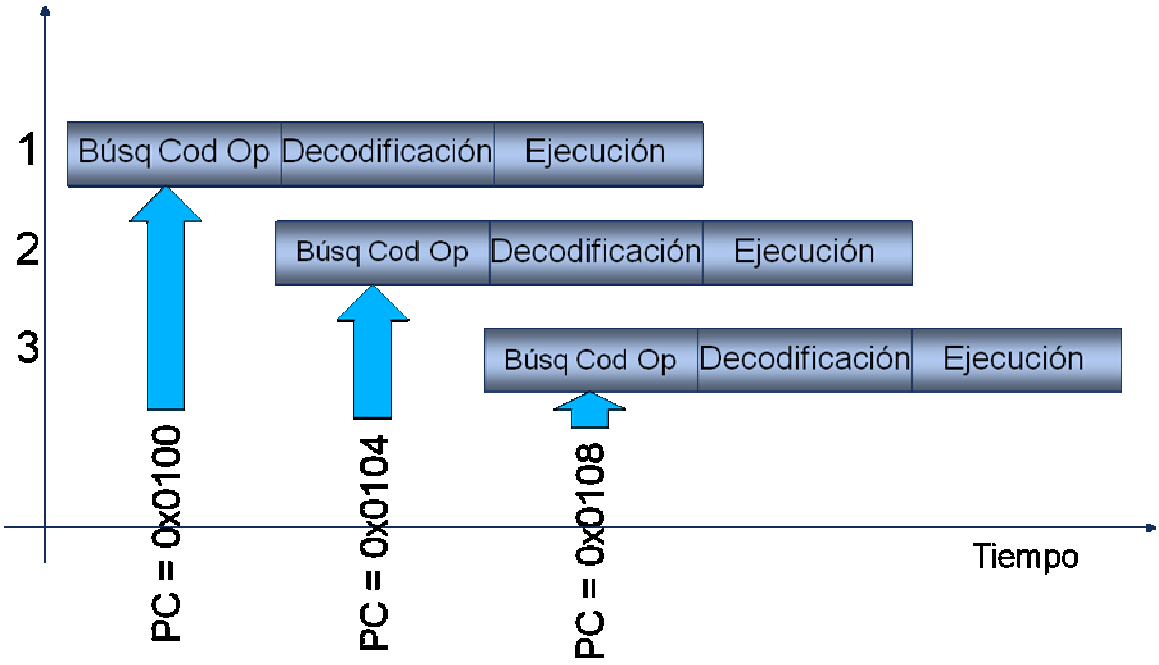

Fig 5 – Pipeline

Si bien este paralelismo parece ser una panacea, presenta algunas dificultades.

- En los programas típicos para computadoras, es muy frecuente que el resultado de una instrucción sirva de operando para la siguiente instrucción. Cuando sucede esto, se rompe la secuencia de *pipeline* mostrada en la figura anterior, ya que el resultado de la instrucción 1 no está disponible en el momento en que la instrucción 2 reúne sus operandos. Entonces, la instrucción 2 debe detenerse hasta que el resultado esté disponible[iv]. En ese caso, la instrucción 2 se debe demorar ("estirar") hasta que todos sus operandos estén disponibles. En su defecto pueden intercalarse otra instrucción entre la 1 y la 2 para dar tiempo a que la 1 pueda proveer del operando que necesita la 2.

- Supongamos que mientras se está realizando la ejecución de la instrucción 1, aparece un pedido de interrupción. La instrucción 1 obligatoriamente se termina (no así la 2 y la 3). El valor del PC es 0x108 y ese valor se almacenará en el *Link Register* R14 para regresar cuando finalice la rutina de atención de interrupción. El problema radica en que la primera instrucción a ejecutar deberá ser la que se encuentra en la dirección 0x104, por lo que el programador deberá ajustar la dirección de retorno al fin de la rutina de atención de la interrupción, descontando 4 del valor del LR antes de regresar al programa original.

- El valor del PC y el CPSR se deben restaurar en forma atómica (es decir en una sola operación). Recordemos que esos registros están salvados en los registros R14 y SPSR del modo de la interrupción, por lo que si se restaura el SPSR al CPSR, se cambiará el modo y el LR_irq quedará inaccesible.

  Por tal motivo se recomienda ejecutar una instrucción del tipo

  SUBS pc, r14, #4

Nótese que NO existe una instrucción de retorno de subrutina ya que la arquitectura RISC inhibe estas instrucciones fuera de lo común.
En una sola instrucción se realizará atómicamente la resta de 4 del link Register para llevarlo al PC y simultáneamente restaurar el registro de estado.

La 's' después como modificador de código de operación representa la forma especial de la instrucción cuando el registro de destino es el PC y permite realizar la actualización atómica.

El ajuste de la dirección de retorno depende de la fuente de la excepción

Exception	Saved LR value		Recommended return instruction	Return point
	ARM	Thumb		
Reset	-	-	-	After Reset, r14_svc value is Unpredictable.
Data Abort	PC + 8	PC + 8	SUBS PC, R14_abt, #8	Returns to aborted instruction.
FIQ	PC + 4	PC + 4	SUBS PC, R14_fiq, #4	Returns to interrupted instruction.
IRQ	PC + 4	PC + 4	SUBS PC, R14_irq, #4	Returns to interrupted instruction.
Prefetch Abort	PC + 4	PC + 4	SUBS PC, R14_abt, #4	Returns to aborted instruction.
Undefined instruction	PC + 4	PC + 2	MOVS PC, R14_und	Returns to instruction after Undefined instruction.
SWI instruction	PC + 4	PC + 2	MOVS PC, R14_svc	Returns to instruction after SWI instruction.

Tabla 2 – Ajuste de la dirección de retorno dependiendo de la excepción

Nota de Radacción: El lector puede descargar este artículo y artículos anteriores desde la sección ***"Artículos Técnicos"*** en el sitio web de **EduDevices (www.edudevices.com.ar)**

Continuará en la Parte 2: Microcontroladores de 32 bits ARM – Modelos de programación, aplicaciones y ejemplos.

[ii] http://www.faq-mac.com/noticias/5777/breve-historia-risc-alejandro-pena
[iii] Exception and Interrupt Handling in ARM - **Ahmed Fathy Mohammed Abdelrazek – Universität Stuttgart**
[iv] ARM system-on-chip architecture – Second edition -Furber – Addison Wesley 2000 – ISBN 0-201-67519-6