Capítulo 2. Conociendo ARM

2.1 Introducción

En 1983 Acorn Computers comenzó el desarrollo de la arquitectura ARM. El propósito original para la creación de ARM era el desarrollo de un procesador avanzado, similar al MOS 6502. El primer diseño comercializado fue ARM2 que se comercializo en 1986, este contaba con un bus de 32 bits, un espacio de direcciones de 26 bits y 16 registros de 32 bits [9]. En 1990 el equipo de desarrollo de Acorn se separó, formando ARM Ltd.

A diferencia de otros fabricantes de microcontroladores, ARM no manufactura microcontroladores. El negocio de ARM está basado en licencias de propiedad intelectual **IP**. Con este modelo de negocio ARM provee el diseño de los procesadores a los fabricantes de semiconductores para que estos lo integren al resto del diseño del chip. Además de las licencias IP ARM provee herramientas de desarrollo de software tales como compiladores C y hardware para depuración y programación [10]. En la figura 2.1 se muestra una comparativa de entre diferentes microcontroladores.

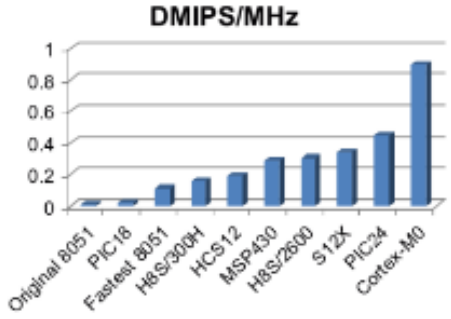

Figura 2.1 Comparativa de diferentes procesadores [10]

Una de las ventajas de utilizar ARM es la compatibilidad entre sus diferentes plataformas, por lo que un programa corriendo en un Córtex M0+ puede ser trasladado a un Córtex M3, aunque para esto se requiere cambiar más que nada la configuración de los periféricos del microcontrolador en particular. En cuanto a las diferencias entre los diferentes Córtex M es en general una mayor capacidad de procesamiento, tal que los ARM Córtex M4 cuentan con un juego de instrucciones que los hace capaces para realizar procesamiento de señales, además de contar con FPU en el caso del Córtex M4-F. A continuación, se muestra las siguientes figuras que demuestran las diferencia entre los distintos juegos de instrucciones de los ARM Córtex M.

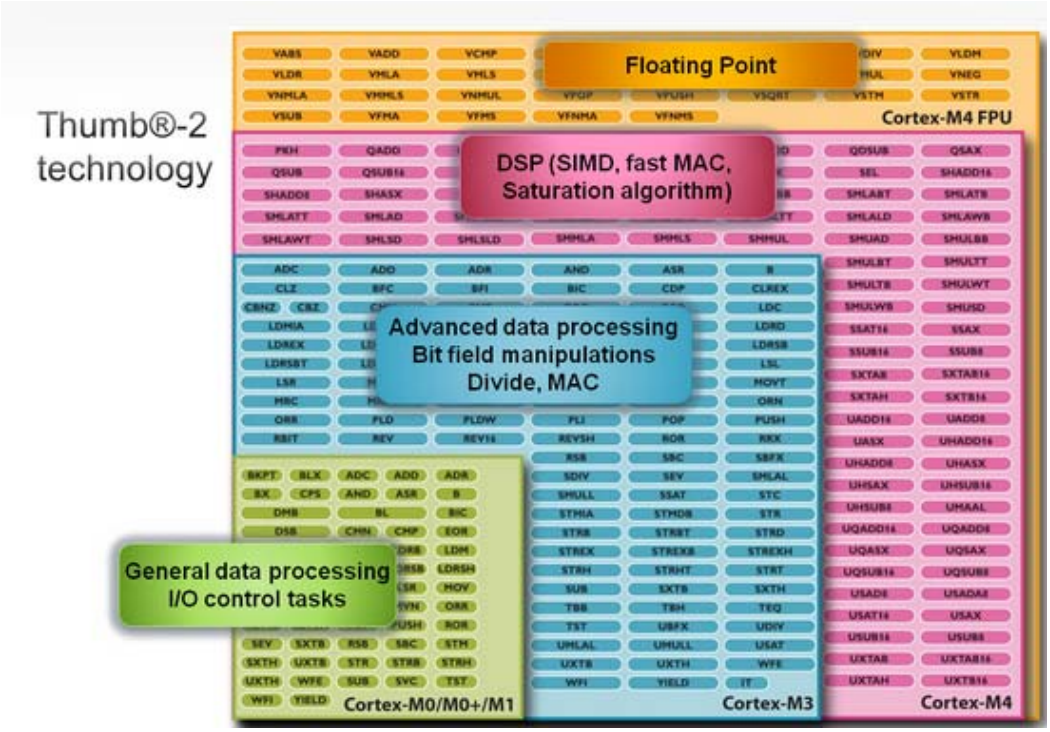

Figura 2.2 Juego de instrucciones Córtex M [11]

2.2 Características de la tarjeta

El desarrollo del presente trabajo se realizó en la tarjeta FRDM KL25Z de Freescale (figura 2.3), esta tarjeta tiene un microcontrolador KL25Z128 ARM Córtex M0+, este microcontrolador está diseñado para bajo consumo de energía y está basado en la arquitectura ARMv6-M. Entre las características más importantes de esta tarjeta podemos citar las siguientes: 128 KB flash

- 16 KB SRAM
- USB OTG
- Interfaz OpenSDA
- Acelerómetro
- LED RGB
- Slider touch capacitivo
- DMA de 4 canales
- Soporte del compilador online mbed

El uso de DMA se realiza desde pasadas revisiones de la arquitectura ARM, este mecanismo otorga la capacidad de transferir registros a ciertos módulos o periféricos del microcontrolador sin que el CPU se encargue de realizarlo.

Figura 2.3 Tarjeta FRDM KL25Z [12]

2.3 ARM Córtex M0+

2.3.1 Revisión General

El ARM Córtex ARM0+ es un microcontrolador RISC con arquitectura Von Neumann, pues consta de un solo bus para datos e instrucciones. Este microcontrolador utiliza un conjunto de instrucciones llamado THUMB, dicho juego de instrucciones es en su mayoría de 16 bits, aunque existen instrucciones de 32 bits para ciertas acciones. El ARM Córtex M0+ tiene un conjunto de 56 instrucciones base, por lo que una instrucción puede tener más de una forma [10].

El ARM Córtex M0+ consta de 2 niveles de *pipeline,* 16 registros de 32 bits, el NVIC (*Nested Vectored Interrupt Controller*) y un multiplicador de 32 bits que varía según el fabricante [13]. En el caso de la tarjeta empleada, Freescale estableció que sus MCU's cuentan con un multiplicador de un solo ciclo.

Además de esto, el Córtex M0+ cuenta con lo siguiente:

- 32 interrupciones y estas a su vez tienen cuatro niveles de prioridad
- 1 NMI (Nonmaskable Interrupt)
- Mapa de memoria definido para portabilidad de software, el espacio de memoria es lineal
- Systick Timer para el uso de RTOS
- Instrucción SVC para uso de RTOS
- Excepción Fault Handling para detectar errores en el sistema.

- Soporte de formato de memoria little endian y big endian

Una vez explicado lo anterior, podemos empezar con lo que es conocido como el modelo del programador. En el modelo del programador el Córtex M0+ puede operar en dos estados, tal y como se muestra en la figura 2.3 en [10].

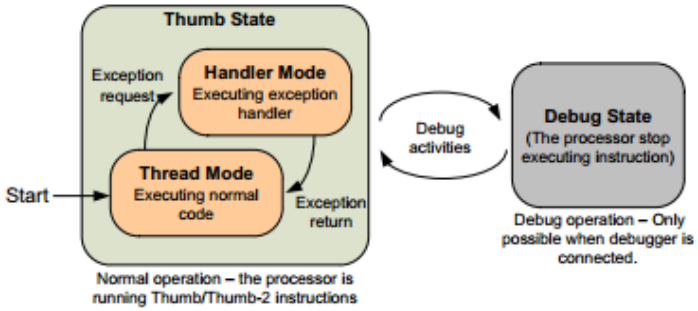

Figura 2.3 Estados de operación Córtex M0

Cuando el microcontrolador está corriendo un programa está en el estado Thumb, dentro del cual puede estar en Thread Mode o Handler Mode, con lo que se puede estar usando un registro stack pointer diferente entre estos modos, el uso de diferentes stack pointers será visto más adelante.

2.3.2 Registros

El ARM Córtex M0+ cuenta con 16 registros de 32 bits y algunos registros especiales, a continuación, se da una descripción de dichos registros.

- **R0-R12**: Son registros de uso general, estos están divididos en registros altos y

bajos debido a que algunas instrucciones Thumb solo acceden del R0 al R7 [10].

- **R13-Stack Pointer**: Este registro es "bancado" mediante el bit #1 del registro CONTROL el cual indica que stack pointer utilizar [14]:

  0: **MSP** **Main Stack Pointer** Es el stack pointer después de reset y es utilizado durante las ISR

  1: **PSP** **Process Stack Pointer** Este es utilizado en el modo thread

- **R14-Link Register:** Almacena un valor de retorno para las rutinas, funciones e ISRs.

- **R15-PC:** Contiene la dirección actual del programa, al reset este se carga con 0x00000004.

- **XPSR:** Es un registro "combinado" del estado del programa, el cual se puede descomponer en:
  - **Application PSR**: contiene las banderas de la ALU
  - **Interrupt PSR**: contiene el número de la interrupción que está siendo atendida.
  - **Execution PSR**: Indica que el microcontrolador está en el estado Thumb si T=1.

En la figura 2.4 se puede observar cómo están constituidos estos registros, mientras que la tabla 2.1 da una descripción más clara sobre cada uno de los campos de APSR.

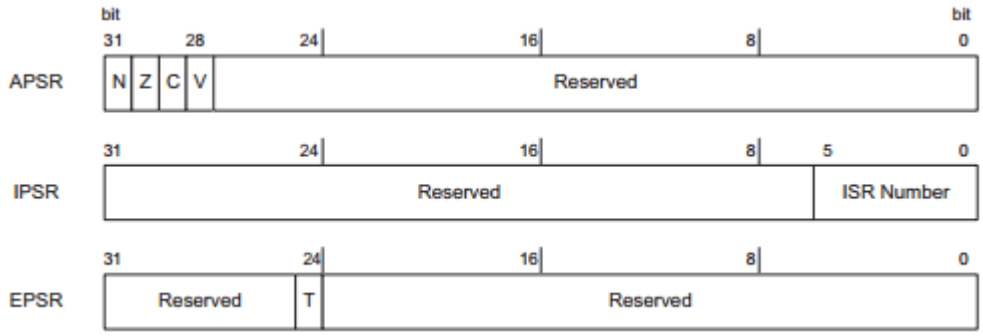

Figura 2.4 Registros PSR [14]

Bits	Name	Function
[31]	N	Negative flag
[30]	Z	Zero flag
[29]	C	Carry or borrow flag
[28]	V	Overflow flag
[27:0]	-	Reserved

Tabla 2.1 Campos del registro APSR [14]

- PRIMASK- Sirve para bloquear todas las interrupciones que se pueden configurar su prioridad.
- CONTROL: Controla el stack utilizado y el nivel de privilegio para la ejecución del código. A continuación, se muestra su asignación de bits.

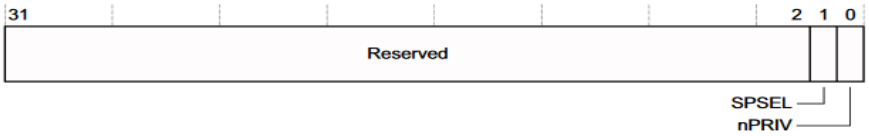

Table 2-8 CONTROL register bit assignments

Bits	Name	Function
[31:2]	-	Reserved
[1]	SPSEL	Defines the current stack: 0 = MSP is the current stack pointer 1 = PSP is the current stack pointer. In Handler mode this bit reads as zero and ignores writes.
[0]	nPRIV[a]	Defines the Thread mode privilege level: 0 = Privileged 1 = Unprivileged.

Figura 2.5 Registro CONTROL [14]

2.3.3 Memoria

El Córtex M0+ tiene un espacio de direcciones de 4GB. El espacio de memoria está definido como un numero de regiones, con el fin de ayudar a la portabilidad entre diferentes dispositivos. El procesador Córtex M0+ contiene un número de componentes construidos como el NVIC y componentes para depuración que tienen una ubicación de memoria fija en el mapa de memoria, dando como resultado que todos los dispositivos con Córtex M0+ tengan el mismo modelo de programación para el control de interrupciones y debug [10].

A continuación, se muestra el mapa de memoria del ARM Córtex M0+ y se dará una breve explicación de cada una de estas regiones:

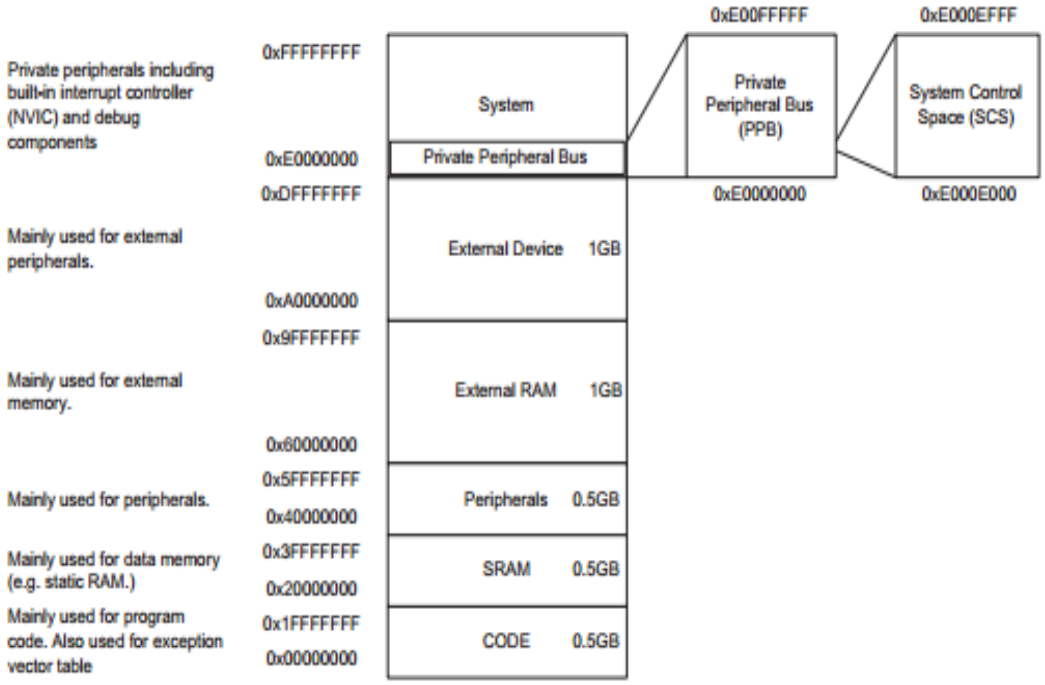

Figura 2.6 Mapa de memoria Córtex ARM [10]

Región de Programa (0x00000000-0x1FFFFFFF): El tamaño de esta región es de 512MB, es usada para almacenar el código del programa y para el vector de interrupciones.

Región SRAM (0x20000000-0x3FFFFFFF): Es una región de 512MB y es usada para almacenar datos, incluyendo el stack.

Región de Periféricos (0x40000000-0x5FFFFFFF): Esta región es de 512MB y es usada para periféricos. Los periféricos conectados a esta memoria pueden estar en los periféricos del bus AHB-Lite o en los periféricos del bus APB, estos tipos de bus se verán más adelante en este capítulo.

RAM Externa (0x60000000-0x9FFFFFFF): Esta región de memoria es dividida en dos bloques de 512 MB y es utilizada para almacenar datos, sin embargo, entre estos dos bloques existen diferentes atributos que podrían causar problemas con memorias cache.

Prívate Peripheral Bus (0xE0000000-0xE00FFFFF): El espacio de memoria para el *internal PPB* es alojado para periféricos dentro del procesador, tales como el NVIC y componentes para debug. Este espacio de memoria es de 1MB y dentro de esta memoria está definido el SCS (System Control State), el cual contiene los registros de control de interrupción, los registros de control del sistema, los registros de control de debug y el systick timer.

Sistema (0xE0100000-0xFFFFFFFF): Esta memoria es para uso específico del fabricante.

Asimismo, los microcontroladores y procesadores ARM cuentan con un sistema de buses, tal y como se muestra en la siguiente figura:

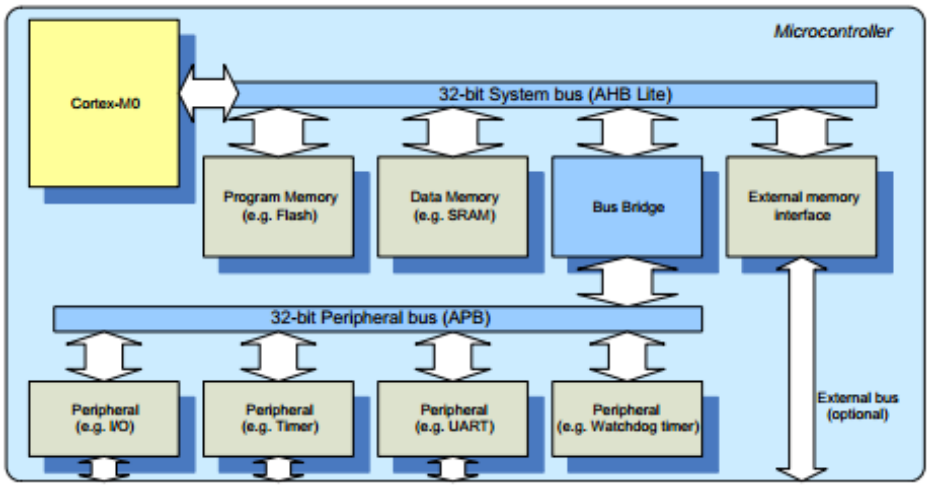

Figura 2.7 Diagrama a bloques ARM Córtex M0 [10]

La existencia de diferentes buses en el procesador o microcontrolador es que algunos periféricos son lentos, por lo que puede ser que estos dos buses tengan diferentes frecuencias de operación. En [15] podemos observar que AHB se trata de un protocolo en el que se recibe y se manda información por medio de direcciones y señales de control, además, los datos recibidos y transmitidos van por diferentes caminos, por lo que el dividir la memoria en secciones cobra mucho sentido, pues los distintos registros de un periférico pueden ser mapeados a una región de memoria para realizar escritura o lectura de los mismos. Además, puede realizar lectura y escritura en modo burst, lo cual ayudaría a leer o escribir en varias locaciones d memoria continuación, se muestra la figura 2.8 que muestra la arquitectura de AHB.

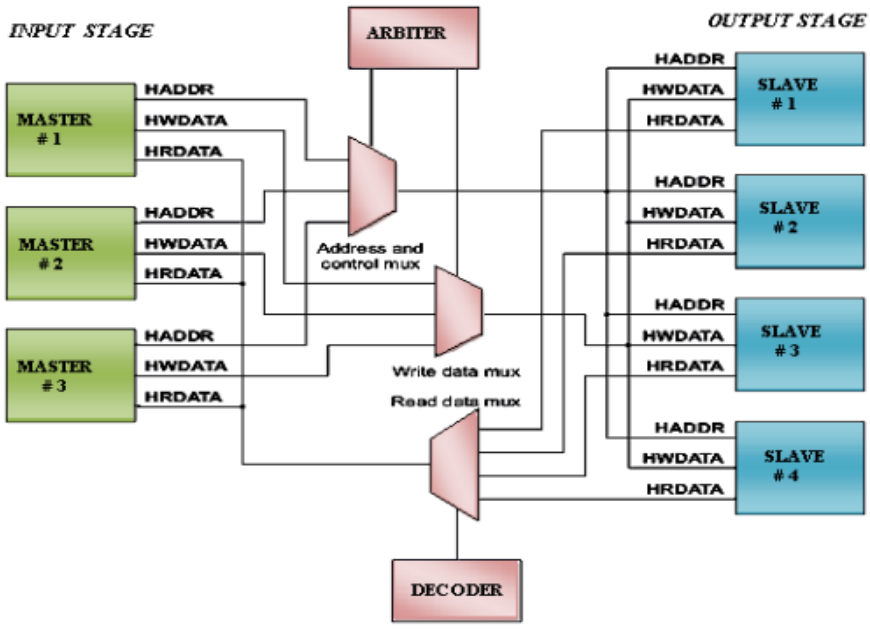

Figura 2.8 Arquitectura de AHB [15]

Por otro lado, tenemos el APB *Advanced Peripheral Bus*, el cual es solo utilizado para

acceso a dispositivos con bajo ancho de banda, no tiene pipeline y cualquier operación requiere como mínimo dos ciclos de reloj [16].

2.3.4 NVIC, interrupciones y el stack

El microcontrolador Córtex M0+ tiene construido un NVIC, el cual soporta hasta 32 interrupciones y un NMI [10]. Dentro de las funciones a cargo del NVIC podemos decir que se encuentran:

- El manejo de interrupciones, habilitación, rehabilitación y configuración de prioridades.
- El soporte de nested interrupts o interrupciones anidadas por medio de hardware
- Enmascaramiento de interrupciones
- Entrada al vector de interrupción

Para que el NVIC pueda trabajar de manera adecuada cuenta con los siguientes registros:

- **NVIC_ISER** habilita las interrupciones
- **NVIC_ICER** deshabilita la interrupción
- **NVIC_ISPR** pone el estado pendiente a una interrupción
- **NVIC_ICPR** es utilizado para limpiar el estado pendiente desde la interrupción
- **NVIC_IRP0-IRP7** son utilizados para configurar las prioridades de las

interrupciones

Asimismo, cada excepción en el M0+ tiene un número único. Las excepciones que son generadas dentro del procesador ocupan del 1 al 15, aunque no todos los números son ocupados. A continuación, se muestra la siguiente tabla que muestra las excepciones del Córtex M0+

Exception Number	Exception Type	Priority	Descriptions
1	Reset	−3 (Highest)	Reset
2	NMI	−2	Nonmaskable interrupt
3	Hard fault	−1	Fault handling exception
4−10	Reserved	NA	—
11	SVC	Programmable	Supervisor call via SVC instruction
12−13	Reserved	NA	—
14	PendSV	Programmable	Pendable request for system service
15	SysTick	Programmable	System tick timer
16	Interrupt #0	Programmable	External interrupt #0
17	Interrupt #1	Programmable	External interrupt #1
...	...	...	...
47	Interrupt #31	Programmable	External interrupt #31

Tabla 2.2 Listado de excepciones del Córtex M0+ [10]

Además de esto, cada excepción tiene una prioridad y un estado. Algunas excepciones pueden tener prioridades configurables, por ejemplo, las excepciones que no son del sistema, donde 3 es la prioridad más baja y 1 es la prioridad más alta configurable [10]. Entre las excepciones más importantes tenemos las siguientes:

Reset: Es la excepción más alta y se presenta cuando se alimenta el microcontrolador.

NMI: Es la excepción más alta después de reset y esta permanente habilitada. Puede ser

disparada por periféricos o por software.

Hard Fault: Ocurre debido a un error, ya sea de alineamiento del PC, cuando se trata de ejecutar instrucciones desconocidas o se intenta a acceder a un registro no habilitado.
SVCall: Un Supervisor call es disparada por la instrucción SVC. Es utilizada en RTOS.

PendSV: Es utilizado por RTOS para el cambio de contexto cuando no hay interrupciones activas.

Systick: Es utilizada para generación de ticks en RTOS.

Interrupciones: son excepciones generadas por un periférico o por una fuente externa a través de un pin I/O. Las interrupciones son asíncronas y necesitan ser habilitadas antes de ser usadas [10].

Una vez explicado el NVIC y las excepciones es tiempo de hablar del stack. El manejo del stack se realiza por medio de las funciones PUSH y POP. El stack es full descending, empezando su primer elemento en la dirección 0x3FFFFFFC, además el tamaño del stack frame debe alinearse al tamaño de una palabra doble. A continuación, se muestra el stack frame que se sigue al entrar en una rutina de servicio a interrupción:

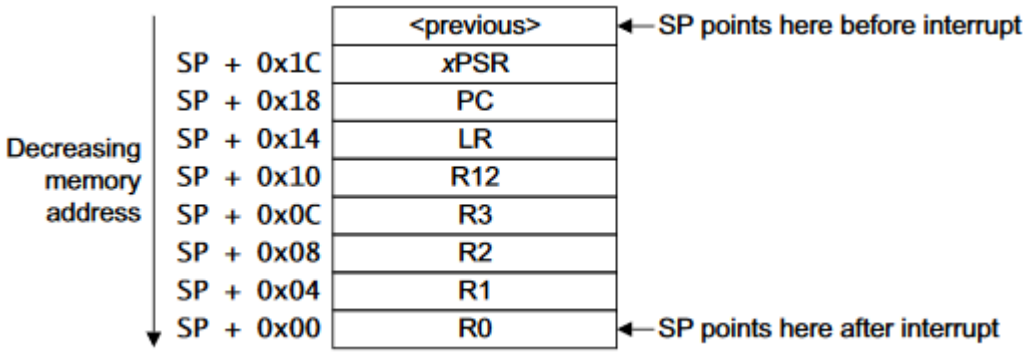

Figura 2.9 Stack Frame en ARM [14].

Como se puede observar, los registros R0 al R3 son ingresados a la pila, esto es debido a una norma implementada por ARM, pues estos registros son llamados "caller saved registers". Una vez realizado el manejo de la interrupción, el ISR ejecuta el retorno usando un valor especial llamado EXEC_RETURN, que ya fue cargado en LR. Este valor permite determinar cómo regresar de la interrupción, pues nos indica si se trata de una interrupción anidada o una interrupción simple y nos dice el stack a utilizar para el regreso.

2.3.5 Instrucciones

El juego de instrucciones para el Córtex M0+ consta de 56 instrucciones, las cuales pueden tener más de una forma, tal y como se muestra en las siguientes tablas mediante el uso de LDR, LDRH y LDRB.

Instruction	LDR/LDRH/LDRB
Function	Read single memory data into register
Syntax	LDR <Rt>, [<Rn>, <Rm>] ; Word read
	LDRH <Rt>, [<Rn>, <Rm>] ; Half Word read
	LDRB <Rt>, [<Rn>, <Rm>] ; Byte read
Note	Rt = memory[Rn + Rm]
	Rt, Rn and Rm are low registers

Tabla 2.3 Uso de instrucciones LDR, LDRH y LDRB [10]

Instruction	LDR/LDRH/LDRB
Function	Read single memory data into register
Syntax	LDR <Rt>, [<Rn>, #immed5] ; Word read
	LDRH <Rt>, [<Rn>, #immed5] ; Half Word read
	LDRB <Rt>, [<Rn>, #immed5] ; Byte read
Note	Rt = memory[Rn + ZeroExtend (#immed5 << 2)] ; Word
	Rt = memory[Rn + ZeroExtend(#immed5 << 1)] ; Half word
	Rt = memory[Rn + ZeroExtend(#immed5)] ; Byte
	Rt and Rn are low registers

Tabla 2.4 Uso de instrucciones LDR, LDRH y LDRB [10]

Estas instrucciones permiten el uso de tres operando, además cuentan con una particularidad, estas instrucciones pueden utilizar sufijos, a modo de condicional o actualización de banderas, por lo que no es necesario cargar el resultado de una comparación dentro de un registro y después comprobar si este es 0 o uno para realizar una acción, pues basta con verificar el estado de las banderas. A continuación, se muestra la tabla de sufijos.

Suffix	Descriptions
S	Update APSR (flags); for example, `ADDS R0, R1; this ADD operation will update APSR`
EQ, NE, CS, CC, MI, PL, VS, VC, HI, LS, GE, LT, GT, LE	Conditional execution. EQ = Equal, NE = Not Equal, LT = Less Than, GT = Greater Than, etc. On the Cortex-M0 processor, these conditions can only be applied to conditional branches. For example, `BEQ label; branch to label if equal`

Tabla 2.5 Sufijos de instrucciones [10]

Por ejemplo, la instrucción ADD con actualización de banderas:

ADD r0, r0, r1 Sin sufijo

ADDS r0, r0, r1 Con sufijo

A continuación, se muestra el juego de instrucciones para el Córtex M0 y Córtex M0+.

16-Bit Thumb Instructions Supported on Cortex-M0									
ADC	ADD	ADR	AND	ASR	B	BIC	BLX	BKPT	BX
CMN	CMP	CPS	EOR	LDM	LDR	LDRH	LDRSH	LDRB	LDRSB
LSL	LSR	MOV	MVN	MUL	NOP	ORR	POP	PUSH	REV
REV16	REVSH	ROR	RSB	SBC	SEV	STM	STR	STRH	STRB
SUB	SVC	SXTB	SXTH	TST	UXTB	UXTH	WFE	WFI	YIELD

Tabla 2.6 Instrucciones Thumb de 16 bits del Córtex M0 y M0+ [10]

32-Bit Thumb Instructions Supported on Cortex-M0					
BL	DSB	DMB	ISB	MRS	MSR

Tabla 2.7 Instrucciones Thumb de 32 bits del Córtex M0 y M0+ [10]

Como se puede observar, el juego de instrucciones no es muy grande, pero con el uso de los sufijos se puede llegar a creer que se tiene un juego de instrucciones mucho más grande. Además, algunas instrucciones permiten operandos de 8 bits, por lo que se pueden mover constantes menores o iguales a 255.

2.4 CMSIS

Debido a la creciente complejidad de los sistemas embebidos, la compatibilidad y la reutilización de código sea tomada en cuenta, pues reduce el tiempo de aprendizaje y desarrollo, es por eso que nació CMSIS. CMSIS (Cortex Microcontroller Software Interface Standart) es una capa de abstracción de hardware independiente del fabricante, la cual provee soporte para el NVIC, el bloque de control del sistema y la configuración del Systick entre otras cosas para los ARM Córtex-M [10]. En un principio CMSIS contemplaba lo siguiente:

- Funciones de acceso para el NVIC, Systick, SCB y control de interrupciones.
- Definiciones de registros estandarizadas para el NVIC, SCB y el Systick Timer.
- Funciones de acceso especial para instrucciones que no se pueden realizar en código C de manera sencilla
- Funciones de inicialización estandarizadas
- Variables de información del reloj del sistema

Ahora podemos decir que CMSIS se ha desarrollado bastante, pues cuenta con los siguientes componentes:

- **CMSIS-CORE**: API para procesadores y microcontroladores Córtex. Provee una interface estandarizada a través de los diferentes ARM Córtex.
- **CMSIS-Driver**: define drivers genéricos para interfaces de periféricos para middleware, haciéndolo reusable entre diferentes plataformas. La API es

independiente de RTOS y ayuda a comunicar los periféricos con middleware utilizando stack de comunicación, sistemas de archivo e interfaces graficas de usuario.

- **CMSIS-DSP**: Es una librería que implementa más de 60 funciones para diferentes tipos de datos, tales como q7, q15 y q31 que son de punto fijo y punto flotante de 32 bits. La librería es disponible para todas las plataformas ARM Córtex.
- **CMSIS-RTOS API**: API común para sistemas en tiempo real. Provee una interface de programación que es portable a muchos RTOS.
- **CMSIS-DAP**: Debug Access Port. Es un firmware estandarizado para la unidad de depuración, este componente es usualmente provisto en tarjetas de desarrollo [17].