Llamado de subrutinas en ensamblador desde rutinas de escritas en lenguaje C

En muchos casos se utiliza el lenguaje ensamblador para escribir código de aplicaciones, firmware para sistemas embebidos. Esto sin embargo implica un tiempo de desarrollo muy superior al de desarrollar con lenguajes de programación de más alto nivel.

El lenguaje ensamblador cobra fuerza en general en los siguientes casos:

- **Sistemas de control de bajo nivel**
- **Rutinas de tiempos críticos**
- **Optimización de código (espacio en memoria)**
- **Instrucciones que simplemente no se pueden implementar en C, tales como manejo del STACK**

Una opción para armonizar las ventajas de trabajar en lenguajes de programación de alto nivel con la optimización de código que implica trabajar en bajo nivel es trabajar con un lenguaje de programación de alto nivel y desarrollar la tareas críticas en lenguaje ensamblador. En nuestro caso, consideraremos el lenguaje C.

Existen 2 formas de llamar rutinas en lenguaje ensamblador desde C:

- Inline ASM
- Escribir rutinas completas en lenguaje ensamblador y luego llamarlas desde C

Interface con subrutinas

Uno de los puntos fundamentales que surge es cómo se realizará la interface con la subrutina, dicho de otro modo, cómo podemos pasar los parámetros que sean necesarios.

•Una opción es utilizar punteros, lo cual lo tenemos sumamente visto en varias asignaturas que tocan el tema.

```
unsigned int volatile * const FIO0DIR = (unsigned int*) 0x2009c000;

*FIO0DIR|=(1<<22);
```

•La segunda opción es valernos del estándar AAPCS, el cual establece una serie de reglas que el compilador debe cumplir cuando de subrutinas se trata.

ARM Architecture Procedure Call Standard (AAPCS)

El estándar AAPCS define la manera en que rutinas compiladas por un lado y rutinas ensambladas por otro pueden trabajar juntas. Es una interface externa a las 2. en la siguiente tabla tenemos las funciones que cumple cada registro durante una llamada a subrutina

Register	Synonym	Special	Role in the procedure call standard
r15		PC	The Program Counter.
r14		LR	The Link Register.
r13		SP	The Stack Pointer.
r12		IP	The Intra-Procedure-call scratch register.
r11	v8		Variable-register 8.
r10	v7		Variable-register 7.
r9		v6 SB TR	Platform register. The meaning of this register is defined by the platform standard.
r8	v5		Variable-register 5.
r7	v4		Variable register 4.
r6	v3		Variable register 3.
r5	v2		Variable register 2.
r4	v1		Variable register 1.
r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.

Table 2, Core registers and AAPCS usage

ARM Architecture Procedure Call Standard (AAPCS)

Result Return/Parameters Passing

El estándar especifica que si creamos una función a la cual le pasamos 4 números de 32bits (enteros) en el momento en que se llama a la subrutina el primer parámetro se cargará en R0, el segundo en R1, etc.

A su vez, si desde la subrutina retornamos un valor entero, este retornará a travéz de R0, si de 64 bits en R0-R1, si 128bits en R0-R3.

Esto resulta sumamente importante dado que en casos en que solo se necesiten pocos parámetros puede resolverse sin necesidad de buscar un dato en memoria.

The-Definitive-Guide-to-Arm-Cortex-M3-and-Cortex-M4-Processors-Third-Edition

r3	a4		Argument / scratch register 4.
r2	a3		Argument / scratch register 3.
r1	a2		Argument / result / scratch register 2.
r0	a1		Argument / result / scratch register 1.

Table 2, Core registers and AAPCS usage

ARM Architecture Procedure Call Standard (AAPCS)

Estructura

20.3 Structure of an assembly function

An assembly function can be very simple. For example, a function to add two input parameters can be as simple as:

```
My_Add ADDS R0, R0, R1 ; Add R0 and R1, result store in R0
       BX   LR         ; Return
```

```
My_Add FUNCTION
       ADDS R0, R0, R1 ; Add R0 and R1, result store in R0
       BX   LR         ; Return
       ENDFUNC
```

ARM Architecture Procedure Call Standard (AAPCS)

Estructura

In the GNU toolchain:

```
        .text                   /* text section */
        .syntax unified         /* Unified Assembly Syntax - UAL */
        .thumb                  /* Thumb instruction set */
        .type My_Add, %function
        .global My_Add          /* Make My_Add visible from outside */
My_Add
        ADDS R0, R0, R1         /* Add R0 and R1, result store in R0 */
        BX   LR                  /* Return */
        .end                    /* End of file */
```

ARM Architecture Procedure Call Standard (AAPCS)

Estructura

En subrutinas más complejas son más los pasos a seguir. En general la estructura de una subrutina se puede dividir en los siguientes pasos:

- Prologo: guardado de los registros en el stack de la memoria, si es necesario.
- Generar espacio en el stack para variables locales, de ser necesario.
- Realizar los cálculos propios de la subrutina
- Almacenar el resultado en Ro para que esté disponible al retornar de la subrutina
- Liberar el espacio del stack que usamos para variables locales
- Epilogo: Recuperar los registros que guardamos en la pila.
- Retornar

ARM Architecture Procedure Call Standard (AAPCS)

- Si en la Subrutina no se afectan los registros de R4-R11 el prólogo y el epílogo no son necesarios

- Si no es necesario tener variables locales en la subrutina (suponiendo que podemos resolver todos los cálculos con los registros R0-R3), no es necesario generar espacio en la pila y por tanto liberarlo.