Interfaz C Assembler Cortex - M

Agenda

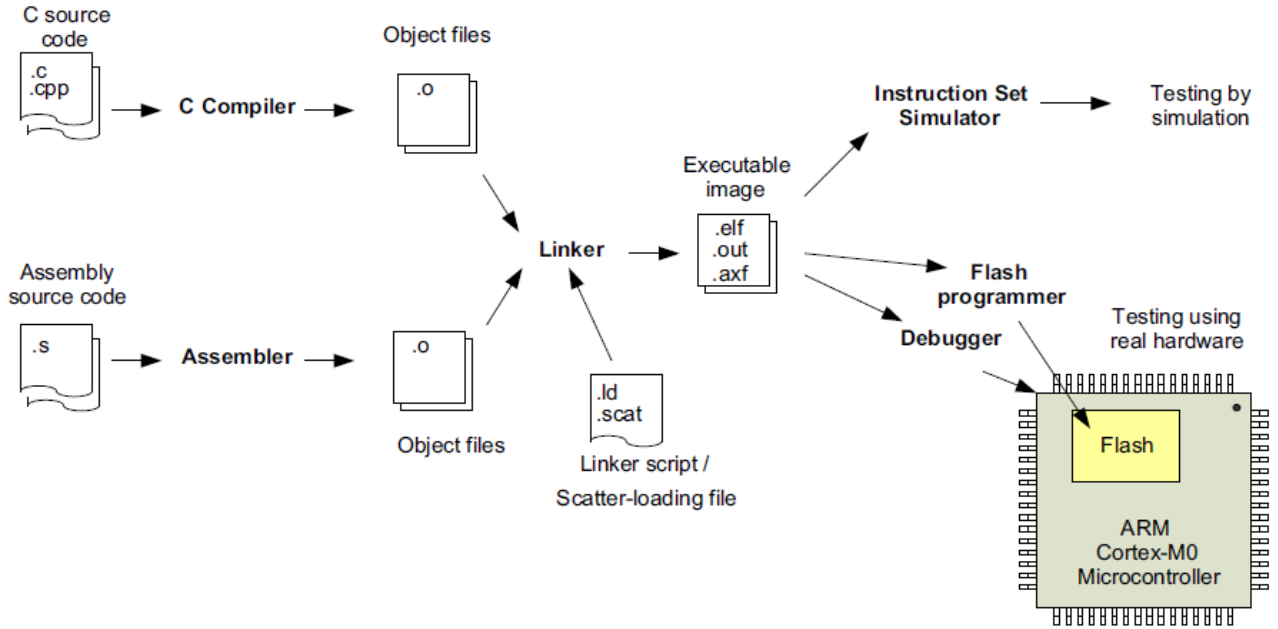
- Necesidad de la interfaz C assembler
- Llamada a función.
- Estructura del archivo de assembler.
- Pasaje de parámetros.
- Pasaje de parámetros por pila.
- Ejemplo.
- Tipos de instrucciones para Cortex-M0

Necesidad.

- En la enorme mayoría de los casos, las aplicaciones se van a desarrollar en C de punta a punta sin tener necesidad de conocer los detalles de la arquitectura o del conjunto de instrucciones.

- En algunos casos va a ser de utilidad conocer las instrucciones que tiene el procesador y su utilidad:
  - ***Debug.*** Algo no funciona y buscamos el por qué
  - ***Rendimiento.*** La mayoría de los procesadores tienen instrucciones para operaciones específicas y se necesitan para mejorar el rendimiento de una aplicación. (primero la aplicación tiene que funcionar, nunca perder de vista ☺).
  - ***Estudio.*** En la enorme mayoría de los casos no vamos a construir nuestras aplicaciones de cero (Sistemas operativos, drivers SD, USB, bibliotecas de código probadas, etc.) o tenemos que escribir una biblioteca de código para que use otra gente y es necesario entender el funcionamiento de código escrito por otras personas.
  - ***Académico.*** Para terminar de entender una arquitectura dada, estudio de la misma, lectura / escritura de material científico.

Herramientas



- Obviamente la interfaz entre el código en C y el código en assembler es totalmente dependiente de la herramienta utilizada. Para los ejemplos utilizados en adelante se considerará el uso de las herramientas GNU (gcc-arm-none-eabi) https://developer.arm.com/tools-and-software/open-source-software/developer-tools/gnu-toolchain/gnu-rm/downloads

Interfaz

- La interfaz entre C y assembler puede darse de dos manera diferentes:
  - Assembler en línea. Esto es básicamente agregar líneas de assembler en medio del código en C. Ver http://www.ethernut.de/en/documents/arm-inline-asm.html
  - Llamado a rutinas de assembler por medio de la interfaz de funciones en C (motivo de esta presentación).

Llamado a función en Cortex-M

- Cuando hacemos un llamado a función la instrucción utilizada es:
  - `BL foo //Copia el PC actual en LR y pone el valor foo en PC`
- Para retornar de la función:
  - `BX LR //Salta a la dirección apuntada por LR`

Llamado a función (2)

Llamada

```
FuncionDummy();
bl 0x800014c <FuncionDummy>
```

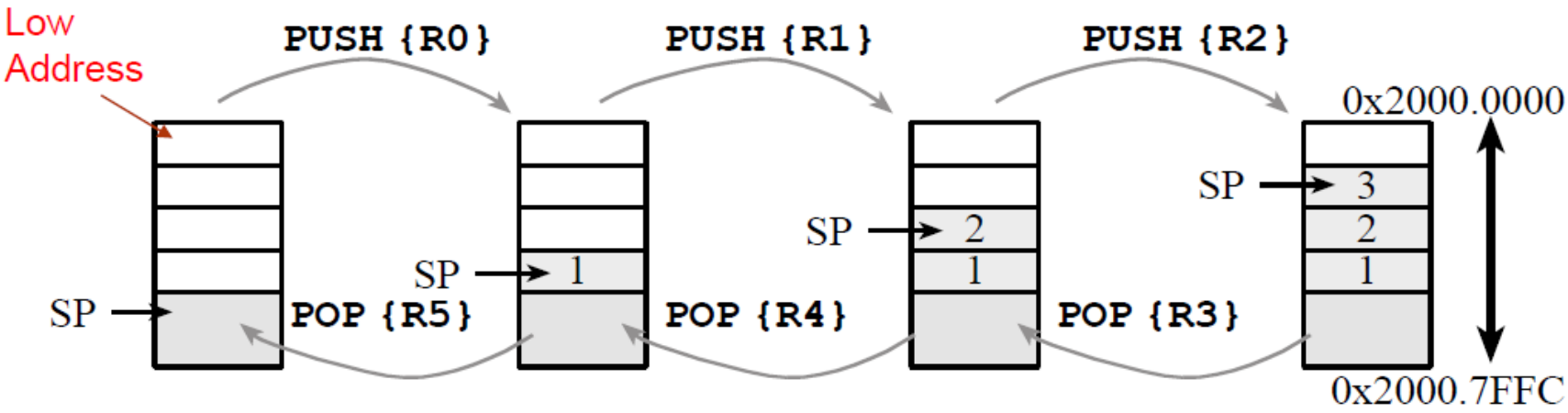
Función

```
.global FuncionDummy
.type FuncionDummy, %function
FuncionDummy:
PUSH{R4}      //R4 a pila
MOV R4, #1    //R4 = 1
POP {R4}      //restauro R4
BX LR         //Retorna
```

La pila

- La pila es una estructura de datos de tipo LIFO (last-in-first-out) con datos de 32 bits.

- El puntero a la pila SP (R13) apunta al tope de la pila.
  - SP se ***decrementa antes*** de hacer un ***PUSH*** (poner un dato nuevo en la pila)***.***
  - SP se ***incrementa después*** de hacer un ***POP*** (sacar un dato de la pila).



Interfaz entre C y ASM

- Como la interfaz entre el assembler va a ser por medio de funciones, el pasaje de datos será a través de los parámetros de la misma.

- Para ARM la interfaz entre C y ASM está definida por el documento “ARM Architecture Procedure Call Standard” (AAPCS).

Pasaje de parámetros.

- Como la interfaz entre el assembler va a ser por medio de funciones, el pasaje de datos será a través de los parámetros de la misma.

- Los parámetros van a ser pasados de la siguiente manera:
  - R0 el primero
  - R1 el segundo
  - R2 el tercero
  - R3 el cuarto
  - Por la pila el resto

Register	Input Parameter	Return Value
R0	First input parameter	Function return value
R1	Second input parameter	-, or return value (64-bit result)
R2	Third input parameter	-
R3	Fourth input parameter	-

Uso de los registros según AAPCS

Register	Function Call Behavior
R0-R3, R12, S0-S15	Caller Saved Register – Contents in these registers can be changed by a function. Assembly code calling a function might need to save the values in these registers if they are required for operations in later stages.
R4-R11, S16-S31	Callee Saved Register – Contents in these registers must be retained by a function. If a function needs to use these registers for processing, they need to be saved to the stack and restored before function return.
R14 (LR)	The value in the Link Register needs to be saved to the stack if the function contains a “BL” or ”BLX” instruction (calling another function) because the value in LR will be overwritten when “BL” or ”BLX” is executed.
R13 (SP), R15 (PC)	Should not be used for normal processing

Ejemplos.

- El lenguaje C no tiene operadores para rotar bits (si los tiene para hacer desplazamientos) pero el procesador tiene una instrucción RORS que los hace en un único ciclo de reloj. Se generarán las funciones:
  - `uint32_t` **`RotarDerecha(uint32_t val, uint32_t shift);`**
  - `uint32_t` **`RotarIzquierda(uint32_t val, uint32_t shift);`**

Ejemplos.

```
.global      RotarDerecha
.type        RotarDerecha, %function
RotarDerecha:
      RORS   R0, R0, R1    //Rota a la derecha R1 bits
      BX     LR            //Retorno de la funcion

.global      RotarIzquierda
.type        RotarIzquierda, %function
RotarIzquierda:
      PUSH   {R2,LR}       //Envio R2 y LR a la pila
      RSBS   R2,R1,#32     //32-R1 a la derecha es lo
      RORS   R0,R0,R2      //mismo que R1 a la izquierda
      POP    {R2,PC}       //Restaura R2 y vuelve
```

Ejemplos. Más de 4 parámetros

- Cuando una función usa más de cuatro parámetros se genera la llamada de la siguiente manera:

```
MOVS    R3, #6                  //R3 = 6 y SP = 0x20004FE8
STR     R3, [SP, #4]            //*(SP+4)=R3 y no cambia SP!!
MOVS    R3, #5                  //R3 = 5
STR     R3, [SP, #0]            //*SP=R3 y SP sin cambiar
MOVS    R3, #4                  //Los primeros cuatro a regs
MOVS    R2, #3                  // y luego llama a la funcion
MOVS    R1, #2
MOVS    R0, #1
BL      0x8000164               // <ParametrosPorPila>
//dummy = ParametrosPorPila(1, 2, 3, 4, 5, 6);
```

Ejemplos. Más de 4 parámetros

- La rutina en assembler tiene que rescatar los parámetros que están por encima de SP, y luego se puede usar la pila.

```
.global         ParametrosPorPila
.type           ParametrosPorPila, %function
ParametrosPorPila:
        LDR     R12, [SP, #4]  //Rescato el sexto parámetro
        ADDS    R0, R0, R12    //Sumo el primero contra el sexto
        LDR     R12, [SP, #0]  //Rescato el quinto parámetro
        ADCS    R0, R0, R12    //Sumo el parcial con el quinto
        PUSH    {LR}           //Ahora puedo usar la pila.
        ADCS    R0, R0, R1     //Sumo el resto
        ADCS    R0, R0, R2
        ADCS    R0, R0, R3
        POP     {PC}           //Retorno de la función
```

Tipos de Instrucciones ARM Cortex-M0

- De salto.
- De procesamiento de datos
- De manejo de memoria (ldr y str)
- De acceso al registro de estado (xPSR)
- Otras.

Saltos

- B (Branch)
  - Actualiza el contenido de PC
  - Puede ser absoluta BX (salto al contenido de un registro). El bit 0 del registro debe ser 1 siempre para que el procesador quede en modo Thumb.
  - Puede ser relativa al valor actual de PC.
    - +/- 256 bytes para saltos condicionales
    - +/- 1MB para saltos incondicionales (ARM v6)
- BL (Branch and Link).
  - Actualiza el contenido de PC y LR. Se usa para llamados a subrutinas.
  - BLX salta al contenido de un registro. El bit 0 debe permanecer en 1 para Cortex-Mx
  - Es relativa al valor actual de PC
    - +/- 16MB relativo al valor de PC

Procesamiento de datos

- Comunes
  - ADD, ADC, SUB, SBC, RSB
  - AND, ORR, EOR, BIC
  - MOV, MVN
  - TST, CMP, CMN
  - ADR (Pseudo Instrucción)
- Desplazamientos y rotaciones
  - ASR
  - LSL, LSR
  - ROR
- Multiplicación
  - MUL
- Extensiones con y sin signo
  - SXTB, SXTH, UXTB, UXTH
- Otras
  - REV, REV16, REVSH

Examples

```
SUBS r0,#1        (r0 ← r0 - 1)
ORRS r0,r1        (r0 ← r0 | r1)
MOVS r0,#1        (r0 ← r0 + 1)
CMP  r0,r1
RSBS r0,r1,#0     (r0 ← -r1)
ADR  r0, Start    (r0 ← [Start])



ASRS r0,r1,#7     (r0 ← r1 >> 7)
LSLS r0,r1,#3     (r0 ← r1 << 3)
RORS r0,r1        (r0 ← r0 >> r1)

MULS r0,r1,r0     (r0 ← r1 * r0)

UXTB r0,r1        (r0 ← r1[7:0])

REV r0,r1         reverse byte order in word
```

Manejo de memoria

- Lecturas y escrituras sin extensión de signo
  - LDR/STR
  - LDRH/STRH
  - LDRB/STRB
- Lecturas con extensión de signo
  - LDRSH
  - LDRSB

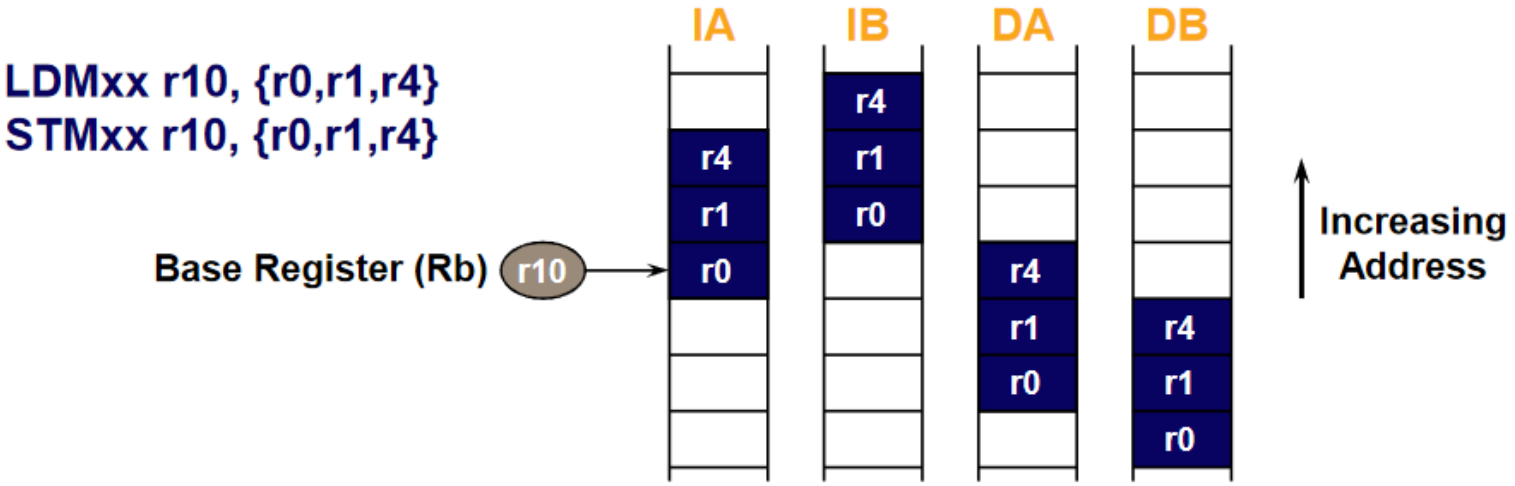
Examples

```
LDR r0,[r1]        (r0 ← [r1])

STM r0,{r1,r2}  (r1 → [r0])  (r2 → [r0+4])
LDM r0,{r1,r2}  (r1 ← [r0])  (r2 ← [r0+4])

PUSH {r1,r2}       (r1 → [SP], r2 → [SP+4])
POP {r1,r2}        (r1 ← [SP], r2 ← [SP-4])
```

- Lecturas y escrituras múltiples.
  - LDM, LDMIA/LDMFD (incremento registro base)
  - STM, STMIA/STMEA (incremento registro base)
- Manejo de Pila
  - PUSH (decrementa SP y copia)
  - POP (copia y luego incrementa SP)

LDM / STM. Funcionamiento

- Hay 4 modos de direccionamiento (sólo 2 soportados por Cortex – M0)
  - `LDMIA / STMIA`. Incrementa luego de copiar
  - `LDMIB / STMIB`. Incrementa antes de copiar
  - `LDMDA / STMDA`. Decrementa luego de copiar
  - `LDMDB / STMDB`. Decrementa antes de copiar



Acceso a registro de estado

- MRS / MSR.
  - MRS (Registro ← Registros de estado)
  - MSR (Registros de estado ← Registros

Examples

```
MRS  r0, IPSR      (r0 ← IPSR )
MSR  APSR, r0      (APSR ← r0)
```

- CPS (Cambiar el estado del procesador)

Examples

```
CPSIE     i        (CPS Interrupt Enable)
CPSID     i        (CPS Interrupt Disable (except NMI and Hard Fault))
```

Ejecución condicional

- Los flags del APSR (NZCV) se los utiliza para decidir si las instrucciones se ejecutan o no.
  - Normalmente las instrucciones previas a las condicionales alteran los flags y en función de estos la instrucción condicional se ejecuta si se cumple la condición, sino se ejecuta como NOP.
  - Cuando la instrucción termina con ‘S’ actualiza los flags. Para Cortex-M0 es obligatorio para la mayoría de las instrucciones terminarlas en S.

- La única instrucción condicional soportada por Cortex-M0 es el salto.
  - **`B<c> addr`**

Condicionales

Condition Code	Interpretation	Status Flag State
EQ	Equal / equals zero	Z set
NE	Not equal	Z clear
CS / HS	Carry set / unsigned higher or same	C set
CC / LO	Carry clear / unsigned lower	C clear
MI	Minus / negative	N set
PL	Plus / positive or zero	N clear
VS	Overflow	V set
VC	No overflow	V clear
HI	Unsigned higher	C set and Z clear
LS	Unsigned lower or same	C clear or Z set
GE	Signed greater than or equal	N equals V
LT	Signed less than	N is not equal to V
GT	Signed greater than	Z clear and N equals V
LE	Signed less than or equal	Z set or N is not equal to V
AL	Always (optional)	Any

Herramientas

- Se utilizó el Cortex-M3 STM32F103C8T6 montado en una placa “bluepill”
- https://stm32-base.org/boards/STM32F103C8T6-Blue-Pill.html

- El software utilizado fue el STM32CubeIDE 1.9.0
- https://www.st.com/en/development-tools/stm32cubeide.html
- Ejemplos. https://gitlab.com/asm_cortex_m3

Ejercicio de Interfaz C – ASM.

- Implementar en assembler la función sumar_todo con la interfaz:
  - `int32_t sumar_todo(int32_t* x, uint32_t len);`
- La función debe sumar todos los elementos del vector x y len es la cantidad de elementos que tiene el vector y puede ser un valor nulo. No hay que tener en cuenta la condición de desborde de la suma.

Bibliografía.

- The Definitive Guide to the ARM Cortex-M3, Second Edition – Joseph Yiu – Newnes – 2009.
- The Definitive Guide to the ARM Cortex-M0. Joseph Yiu. Elsevier. 2011.
- ARM®v7-M Architecture Reference Manual.
- Cortex™-M3 Revision: r1p1 Technical Reference Manual.
- ARM and Thumb-2 Instruction Set Quick Reference Card. https://developer.arm.com/documentation/qrc0001/m/