

# MICROPROCESADORES

## PRÁCTICA 1

### ENTORNO KEIL $\mu$ VISION 5 Y LENGUAJE DE ENSAMBLE

TITULACIONES DE GRADO DE LA  
ETSI DE TELECOMUNICACIÓN



DEPARTAMENTO DE INGENIERÍA TELEMÁTICA Y ELECTRÓNICA

UNIVERSIDAD POLITÉCNICA DE MADRID

PRIMAVERA 2020 - 2021

© 2020-21 DTE - UPM

## ÍNDICE

<b>1. INTRODUCCIÓN</b>	<b>3</b>
<b>2. ESTRUCTURA DE CARPETAS</b>	<b>5</b>
<b>3. TRABAJOS PREVIOS A LA SESIÓN PRESENCIAL</b>	<b>7</b>
<b>3.1. Instalación de la herramienta Keil <math>\mu</math>Vision 5</b>	<b>7</b>
<b>3.2. Ejercicio 1 - ciclo de trabajo con Keil <math>\mu</math>Vision 5</b>	<b>12</b>
<b>3.3. Ejercicio 2 - puntos de ruptura</b>	<b>16</b>
<b>4. TRABAJOS DURANTE LA SESIÓN PRESENCIAL</b>	<b>22</b>
<b>4.1. Ejercicio 3 - área de triángulo</b>	<b>22</b>
<b>4.2. Ejercicio 4 - operaciones aritméticas</b>	<b>22</b>
<b>4.3. Ejercicio 5 - <i>buffer</i> de datos</b>	<b>23</b>
<b>4.4. Subida de resultados a Moodle</b>	<b>24</b>

## 1. INTRODUCCIÓN

En este enunciado se describen los trabajos a realizar durante esta primera práctica. La práctica se organiza en una única sesión de laboratorio, detallándose en las siguientes secciones el trabajo que debe realizarse en ella y, también, describiéndose las actividades previas a realizar antes de esa sesión de laboratorio.

El objetivo de esta práctica es comenzar a usar el entorno de desarrollo *Keil  $\mu$ Vision 5* para microcontroladores basados en microprocesadores de la familia *Cortex-M* y, también, familiarizarse con la programación en lenguaje de ensamble.

En las siguientes prácticas de esta asignatura se trabajará ya en C/C++ (empleando el mismo entorno de *Keil*) y se abordarán técnicas de programación útiles en el desarrollo de aplicaciones para microcontroladores y sistemas empuetrados, como son las *interrupciones*, la *gestión de eventos* y las *máquinas de estados finitos controladas por eventos*.

En el ámbito de los microprocesadores, se denomina *entorno de desarrollo* al conjunto de aplicaciones que permiten la creación y la verificación del funcionamiento de los programas que ejecutará el microprocesador para gobernar un sistema electrónico construido en base a él.

Uno de los entornos de desarrollo más utilizados es el ofrecido por la firma *Keil* (ahora propiedad de arm). Los entornos de *Keil* integran varias herramientas informáticas que permiten crear, simular y depurar los programas escritos en lenguaje C/C++ o en lenguaje de ensamble para distintos microprocesadores, entre ellos casi todos los de la línea *Cortex-M* de arm.

Las herramientas de desarrollo de *Keil* permiten realizar y depurar programas para distintas familias de microcontroladores:

- MDK-Arm: para dispositivos basados en procesadores arm *Cortex-M*, es la herramienta usada en el laboratorio y soporta además dispositivos basados en ARM7, ARM9 y *Cortex-R*;
- DS-MDK: para sistemas basados en procesadores arm *Cortex-A*, con soporte también para arm *Cortex-M*;
- C166: para dispositivos basados en C166, XC166, y XC2000;
- C251: para dispositivos derivados del microcontrolador 80251;
- PK51: para dispositivos derivados del 8051.

Con excepción del entorno DS-MDK (basado en *Eclipse*), el resto de entornos de *Keil* se basan en su propia herramienta  $\mu$ Vision, actualmente en su quinta generación  $\mu$ Vision 5, particularizada en cada caso con los compiladores, ensambladores y depuradores para cada familia de procesadores. Esta será la herramienta usada para la realización de las prácticas de este laboratorio. Puede encontrar más información sobre estas herramientas en la página web de *Keil*: <http://www.keil.com/>

Habitualmente se aprende a programar realizando programas que se ejecutan sobre un ordenador personal PC; es decir se realizan programas que finalmente van a ser ejecutados por el propio microprocesador de *Intel* (actualmente familias *Core i*) que incorpora el PC en el que se han desarrollado. Los programas realizados se ejecutan bajo un sistema operativo (*MS-DOS*, *Windows*, *Linux*, *mac OS* —derivado de *UNIX*—, etc.) que ya está ejecutándose en el ordenador. El sistema operativo proporciona funciones de acceso a dispositivos de I/O (discos, puertos de comunicaciones, teclado, monitor, etc.), de manejo de la memoria, etc.

Al programar un microcontrolador, en cambio, el programa que se realiza muchas veces es el único *software* que ejecutará el microcontrolador, es decir no hay sistema operativo y todo el control y manejo del *hardware* se hace desde el propio programa que se está realizando. En estas ocasiones se utiliza también un PC como herramienta para la creación y depuración de los programas, pero la ejecución final de la aplicación será el microcontrolador (en un *hardware* diferente, por tanto) el encargado de realizarla.

Para facilitar la tarea de depuración los fabricantes de microcontroladores proporcionan *simuladores* para PC que permiten probar los programas sin necesidad de disponer del *hardware* en el que se encuentra el microcontrolador. Uno de estos simuladores es el que se empleará para esta primera práctica. Hay que hacer notar que muchas veces estos simuladores no simulan *exactamente* el comportamiento del microcontrolador, habiendo algunas facetas del mismo, por ejemplo los dispositivos de I/O que incluya, para los que la simulación puede no recoger todos los detalles del dispositivo real.

En esta práctica la fase de depuración del programa se va a realizar mediante el simulador del microcontrolador que incorpora el entorno de desarrollo. En las prácticas siguientes se ejecutarán los programas en el microcontrolador real y se comprobará el funcionamiento de dichos programas mediante un *depurador*. En ambos casos se comprobará el funcionamiento del programa realizado mediante:

- La inspección de la memoria y los registros: comprobando los valores que toman las variables y los registros del microcontrolador, modificándolos si es necesario, a medida que se ejecuta el programa.
- El control de la ejecución del programa: ejecutando cada instrucción individualmente (ejecución paso a paso), ejecutando hasta llegar a una instrucción concreta (puntos de ruptura) o bien realizando el reinicio del microprocesador (*reset*).

## 2. ESTRUCTURA DE CARPETAS

Los distintos archivos, librerías y proyectos de la herramienta *Keil* que se emplean en este laboratorio deben seguir, en el disco duro de los ordenadores, una cierta estructura para asegurar el correcto funcionamiento de todo el código que se entrega para las diferentes prácticas. En la figura 1 se muestra esta estructura de carpetas (en esta figura solo se incluyen las carpetas correspondientes a las prácticas 1 y 2). Los detalles de esta estructura son:

- Toda la estructura aparece dentro de una carpeta MICR, aunque esta carpeta puede tener cualquier otro nombre que le sea de utilidad.
- Los proyectos de *Keil* de las cuatro prácticas (P1 a P4) se encuentran dentro de las carpetas P1 a P4. Estas carpetas deben estar ubicadas directamente dentro de la carpeta MICR.
- Dentro de las carpetas P1 a P4 aparecen carpetas de nombre S? (con «?» siendo una cifra o la letra «x»). Estas carpetas contienen los materiales correspondientes a las diferentes sesiones de una misma práctica (S1, S2 o S3). Para el caso de que una cierta práctica no separe sus ejercicios en sesiones esta carpeta se llamará Sx.
- Las carpetas de nombre Previo\_S? (con «?» una cifra) indican que contienen dentro material pertinente para las actividades previas a una determinada sesión de la práctica.
- Dentro de las carpetas Previo\_S? o S? se encuentran carpetas que contienen los proyectos de *Keil*. **Debe ser muy cuidadoso con la ubicación de las carpetas de proyectos de Keil. Es imperativo que estos proyectos se encuentren en carpetas 3 niveles por debajo de la carpeta «origen» MICR, en caso contrario las**

**compilaciones de los proyectos no serán fructíferas debido a que el *linkador* no podrá localizar las librerías** que se describen sucintamente a continuación.

- Las carpetas *mbed*, *sw\_tick\_serial* y *to\_7seg* contienen librerías. *mbed* y *to\_7seg* se emplearán en las prácticas P2 a P4, mientras que *sw\_tick\_serial* solo durante la práctica P2 (y en el primer examen de laboratorio).

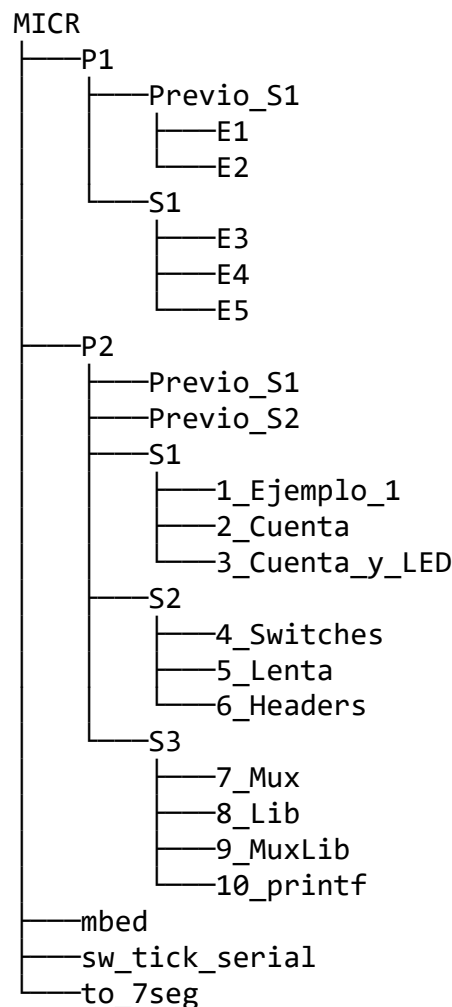


FIGURA 1: Estructura de carpetas para las dos primeras prácticas del laboratorio.

**Antes del final de cada sesión de laboratorio deberá subir a Moodle (comprimida en formato .7z. El software necesario para realizar el manejo de archivos en formato .7z es gratuito, se en-**

cuentra instalado en los ordenadores del laboratorio y puede descargarse de <https://www.7-zip.org/>) la carpeta P1 a P4 correspondiente a la práctica, lo que conformará uno de los entregables de la misma. Observará, durante la realización de las prácticas, que con cada compilación la herramienta *Keil* crea dos carpetas nuevas —dentro de la carpeta que contiene el proyecto— de nombres *~build* y *~listings* (son las únicas cuyo nombre empieza por «~»). Asegúrese de borrar dichas carpetas antes de generar el fichero .7z, de no hacerlo el tamaño del fichero comprimido generado será demasiado grande como para que *Moodle* lo admita. **Estos entregables deberán ser subidos a Moodle por todos los integrantes de cada puesto de laboratorio. Si no se subiese el entregable o se hiciese después del final de la correspondiente sesión de laboratorio, se entenderá la entrega no realizada y se calificará con 0 puntos.**

Adjunto al enunciado de esta práctica encontrará en *Moodle* un fichero comprimido que, al descomprimirlo, genera la estructura de carpetas correspondiente a la práctica 1.

### 3. TRABAJOS PREVIOS A LA SESIÓN PRESENCIAL

#### 3.1. Instalación de la herramienta *Keil μVision 5*

La herramienta *Keil μVision 5* se encuentra instalada en los PC del laboratorio. Sin embargo, le será imprescindible instalar esta herramienta en su propio PC para poder realizar muchas tareas sin necesidad de acudir al laboratorio y para las sesiones no presenciales. La herramienta solo corre en sistemas operativos del tipo *Windows*, por lo que si su PC dispone de otro sistema operativo deberá instalar alguna versión de *Windows* en una máquina virtual.

La versión del entorno que se va a emplear en el laboratorio (*Keil μVision 5.28a*) puede ser descargada gratuitamente desde la página web

## MICROPROCESADORES

de Keil (<http://www.keil.com/>, recuerde, es la herramienta MDK-Arm\*). Tenga en cuenta que deberá ejecutar el instalador como *Administrador*.

En la parte final de la instalación del entorno se abre la ventana del *Pack Installer*. Esta herramienta se conecta a los servidores de Keil para descargar las últimas versiones de las herramientas de soporte para cada microcontrolador sobre el que se desee trabajar. Pasado un pequeño tiempo, al finalizar la descarga de la base de datos de microcontroladores, la ventana del *Pack Installer* tendrá el aspecto mostrado en la figura 2.

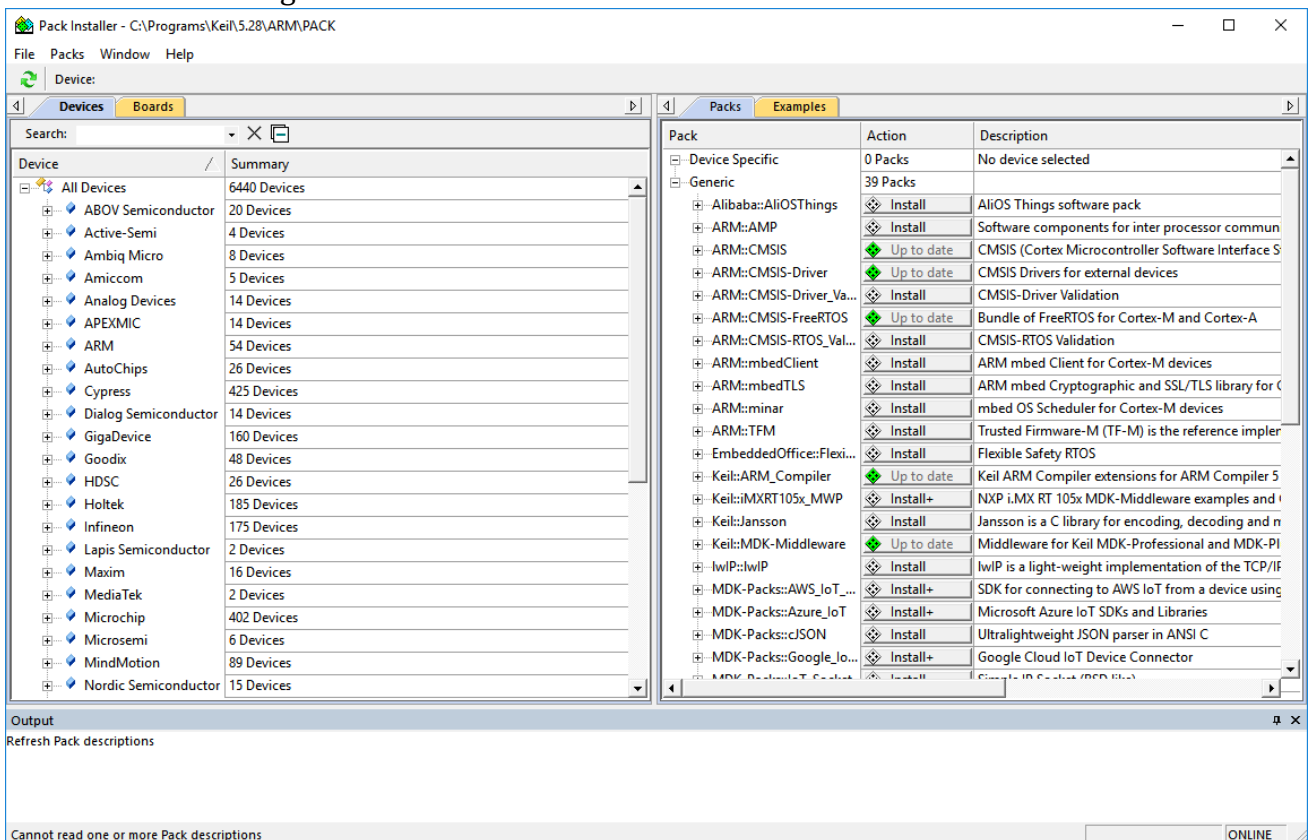


FIGURA 2: Ventana del *Pack Installer* durante la instalación de  $\mu$ Vision 5.

\* Si la versión disponible es posterior a la 5.28a también puede usarla, pero asegúrese de que las versiones de los *Packs* para los distintos procesadores son exactamente las especificadas más adelante en esta práctica.



## PRÁCTICA 1: ENTORNO Y LENGUAJE DE ENSAMBLE

Las herramientas de soporte para cada procesador ocupan espacio en el disco duro, por lo que solamente se instalan las necesarias para los procesadores con los que se va a trabajar. En este laboratorio se hará uso de los microcontroladores:

- NXP LPC1114/13FHI33/201: este es un microcontrolador de la firma NXP basado en un núcleo arm *Cortex-M0* que se empleará para las actividades en lenguaje de ensamble (esta práctica).
- NXP LPC1768: microcontrolador de la firma NXP basado en un microprocesador arm *Cortex-M3*, es el que se encuentra en las tarjetas *mbed LPC1768*.
- STM STM32L432KC: microcontrolador de la firma ST *Microelectronics* basado en un core arm *Cortex-M4* con unidad de punto flotante, es el microcontrolador empleado por las placas *Nucleo-l432kc*.

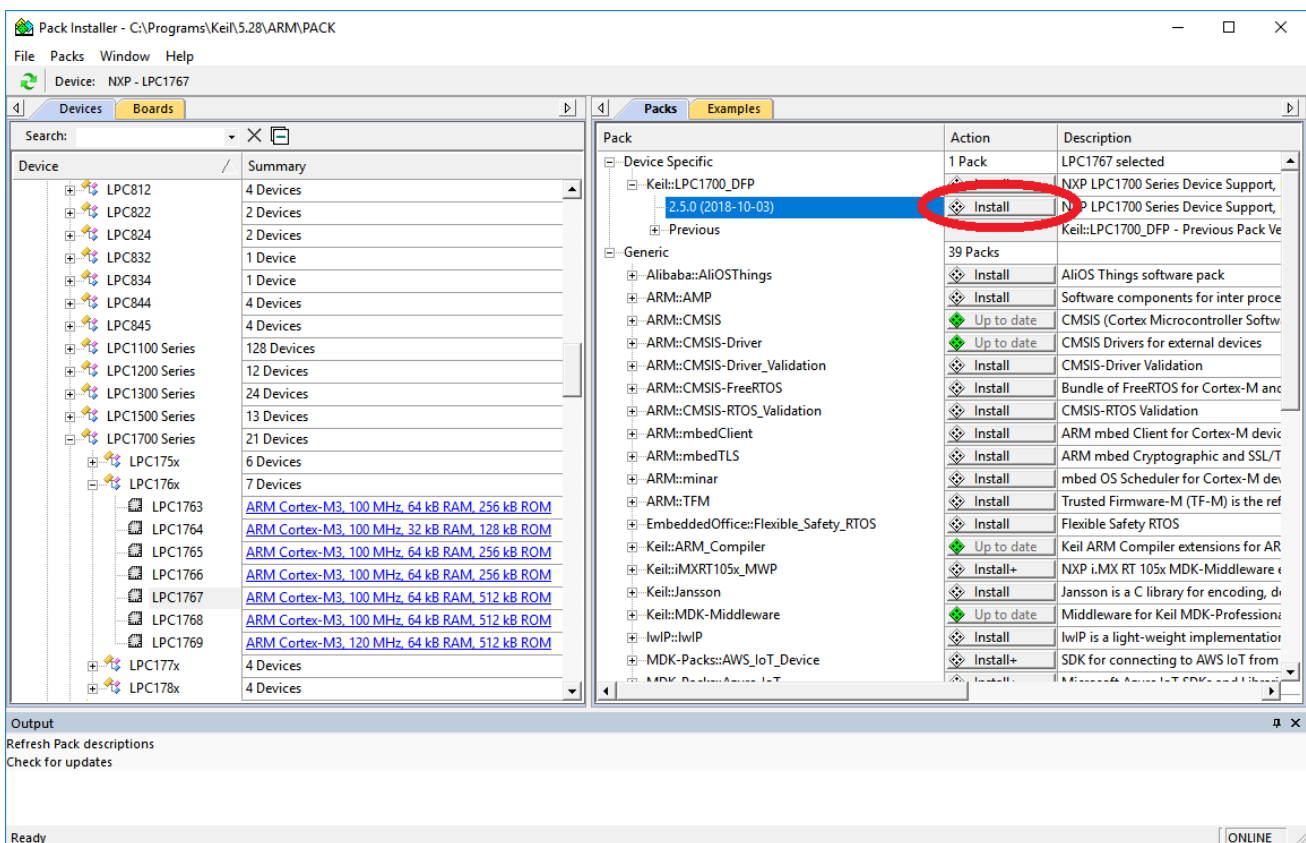


FIGURA 3: Instalación de la versión 2.5.0 de las herramientas para el microcontrolador LPC1768.

## MICROPROCESADORES

Se empezará instalando el soporte para el microcontrolador LPC1768. Para ello busque, en la ventana izquierda del *Pack Installer*, el fabricante NXP y, dentro de él (pique en «+») seleccione la serie LPC1700. Ahora, en la ventana derecha, haga *click* en el «+» de Keil::LPC1700\_DFP y, finalmente, sobre el botón *Install* de la versión 2.5.0, tal y como se ve en la figura 3 anterior. Si en el momento en que realiza usted estas operaciones existiera una nueva versión del *pack* para este procesador, busque la versión 2.5.0 dentro del «+» de Previous.

Para los restantes procesadores se actúa de forma similar, aunque las versiones a instalar son diferentes. Concretamente, para instalar el soporte para el microcontrolador STM32L432KC, busque el fabricante STMicroelectronics, serie STM32L4, e instale la versión 2.2.0, como se ve en la figura 4. Para el caso del microcontrolador LPC1114/33/201, es fabricante NXP, serie LPC1100, y versión 1.4.0.

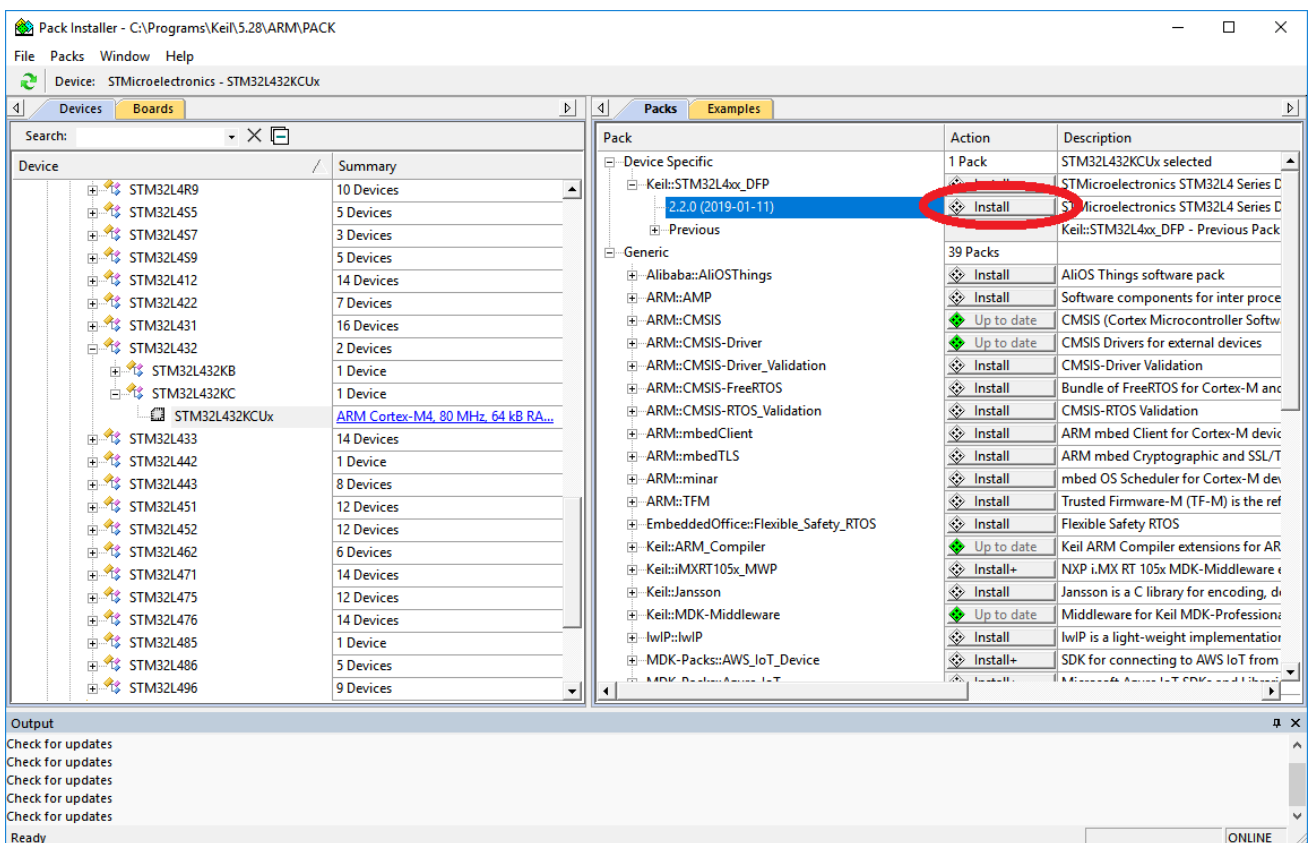



FIGURA 4: Instalación del soporte para el microcontrolador STM32L432KC.

## PRÁCTICA 1: ENTORNO Y LENGUAJE DE ENSAMBLE

Una vez instalado el soporte para estos tres microcontroladores puede cerrar el *Pack Installer*.

Para terminar la instalación de *Keil  $\mu$ Vision 5* deben configurarse algunas de sus preferencias. Para ello abra *Keil  $\mu$ Vision 5* ejecutándolo como administrador (botón derecho sobre el icono del programa y pique sobre  Ejecutar como administrador). Si no arrancase *Keil  $\mu$ Vision 5* de esta manera los cambios que realizaría no tendrían efecto tras cerrarlo. Una vez arrancado vaya a Edit → Configuration... y ajuste en la pestaña Editor todos los campos tal como se muestra en la figura 5. Asegúrese especialmente de marcar las opciones View White Space, Insert spaces for tabs (en 3 lugares) y ajuste el tamaño de las tabulaciones (Tab size) a 2 (C/C++ files), 4 (ASM files) y 2 (Other files). Una vez ajustadas todas las opciones como se ha dicho, pulse sobre OK y cierre *Keil  $\mu$ Vision 5*. Queda ya terminada la instalación y configuración de la herramienta. Si estos últimos ajustes de la configuración del editor no se hicieran, ocurriría que la *indentación* de su código podría verse modificada si edita el código en su ordenador y luego en el ordenador del laboratorio o viceversa.

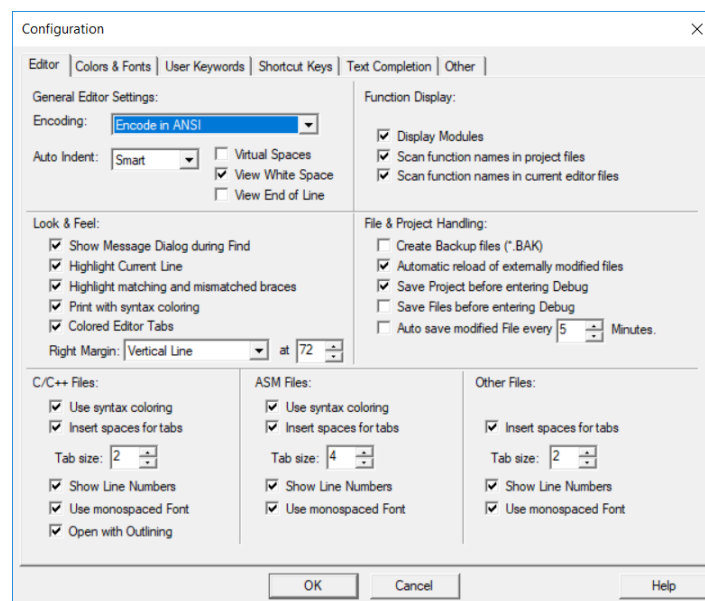


FIGURA 5: Preferencias del editor en *Keil  $\mu$ Vision 5*.

El acceso a las placas *mbed LPC1768* y *Nucleo-l432kc* desde  *$\mu$ Vision 5* (necesario para las siguientes prácticas) requiere la instalación de algún

*software* adicional (*drivers* y *firmware*), cuya instalación se abordará en la práctica 2 del laboratorio.

### 3.2. Ejercicio 1 - ciclo de trabajo con Keil $\mu$ Vision 5

En este ejercicio se presenta el manejo mínimo de la herramienta *Keil  $\mu$ Vision 5* y el uso de las herramientas básicas de depuración que permiten la ejecución paso a paso de un programa, así como visualizar —y modificar— el contenido de los registros y la memoria del microcontrolador.

Antes de comenzar a explicar la funcionalidad a implementar en esta práctica, es necesario conocer brevemente el proceso de desarrollo de aplicaciones empleando el entorno de desarrollo de *Keil  $\mu$ Vision 5*. Haga doble *click* sobre el fichero MICR\P1\Previo\_S1\E1\MICR.uvprojx (que se encuentra en el descargable —en *Moodle*— para esta práctica), se abrirá un entorno de desarrollo como el mostrado en la figura 6.

Este proyecto ya se encuentra configurado para trabajar con el microcontrolador LPC1114/33/201. Este es un microcontrolador de la firma NXP basado en un procesador arm *Cortex-M0*, su frecuencia máxima de reloj es de 50 MHz y dispone de 24 KiB de memoria *Flash* (a partir de la dirección 0x0000 0000), 6 KiB de memoria SRAM (dirección 0x1000 0000) y 2 KiB de memoria EEPROM, todo ello (junto con varios periféricos genéricos: *timers*, convertidores, GPIO, etc.) en un encapsulado de 33 pines. Se emplea para esta práctica ya que el lenguaje de ensamble de los procesadores basados en *Cortex-M0* es mucho más simple que el de las familias *Cortex-M3* y *Cortex-M4* de las placas NXP y STM de las siguientes prácticas de laboratorio.

Este proyecto ya se encuentra configurado para poder trabajar con el simulador. En prácticas posteriores se empleará la tarjeta para ejecutar los programas, en cuyo caso la depuración se hará con el *debugger*, en vez del simulador. En caso de querer empezar el desarrollo de un proyecto desde cero sería necesario realizar varias configuraciones del entorno de desarrollo.

En la parte izquierda se encuentran los ficheros que componen el proyecto, en este caso sólo aparece el fichero asm.s. Analice el código de la aplicación y comprobará que se trata de un sencillo programa que realiza la función AND de los datos almacenados en dos posiciones de

## PRÁCTICA 1: ENTORNO Y LENGUAJE DE ENSAMBLE

memoria y guarda el resultado en otra posición. Una vez realizada esta funcionalidad el programa se queda en un bucle infinito.

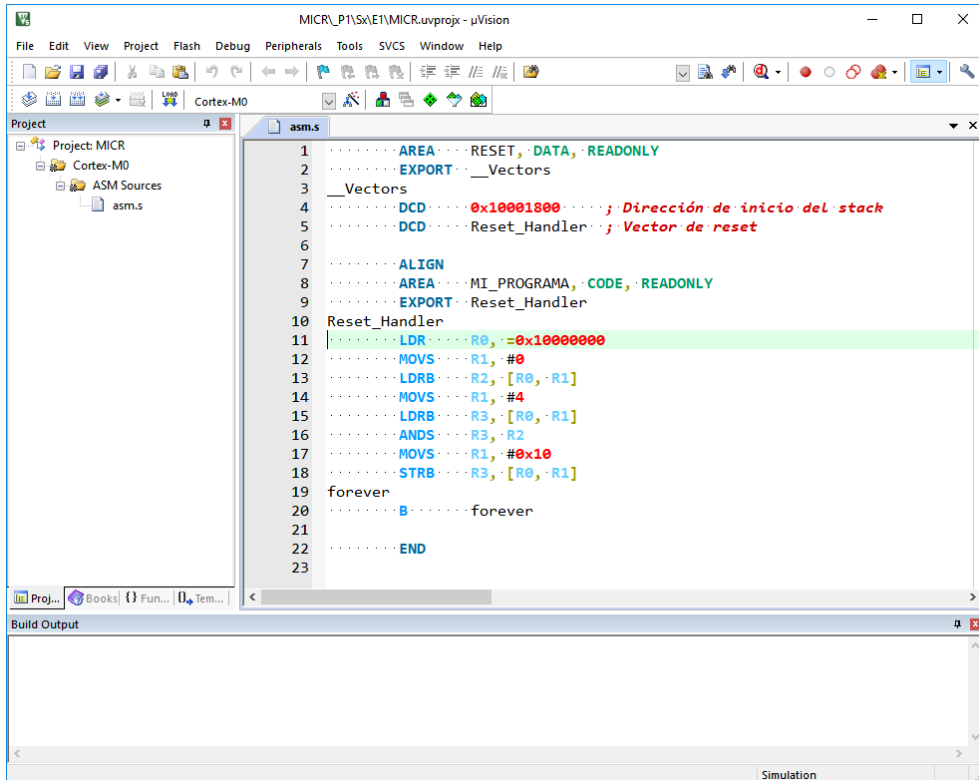





FIGURA 6: Entorno de desarrollo Keil  $\mu$ Vision 5.

Adicionalmente, el programa comienza con una serie de directivas del ensamblador que indican las posiciones de memoria en la que se almacena la pila (`0x1000 1800`) y el código (`Reset_Handler`). Ambas son las primeras direcciones de la tabla de vectores (`__Vectors`). Más detalles sobre el funcionamiento de esta tabla se explicarán en las clases de teoría. También se define un área de código (`MI_PROGRAMA`) en la que se incluye la etiqueta `Reset_Handler`, que es donde se iniciará la ejecución.

Una vez analizado (o editado) el programa es necesario ensamblarlo para detectar posibles errores. Para ello debe emplear estos tres iconos que se encuentran en la parte superior izquierda   . El primero ensambla (compila, para el caso de que el código fuente sea C/C++) sólo el fichero activo, el segundo ensambla (o compila) y enlaza (*linka*) los ficheros que hayan cambiado desde el último ensamblado (compilación) y el tercero ensambla (compila) y *linka* todo de nuevo. Normal-

## MICROPROCESADORES

mente sólo es necesario emplear el segundo. Como resultado del proceso, y si no hay errores en el programa, se genera un fichero ejecutable que puede correr en el microcontrolador. Para comprobar si todo el proceso se ha realizado correctamente se debe consultar la ventana inferior (Build Output), en la que aparecerán los mensajes de salida de ensamblador, preprocesador, compilador y *linkador*, incluyendo errores y *warnings*.

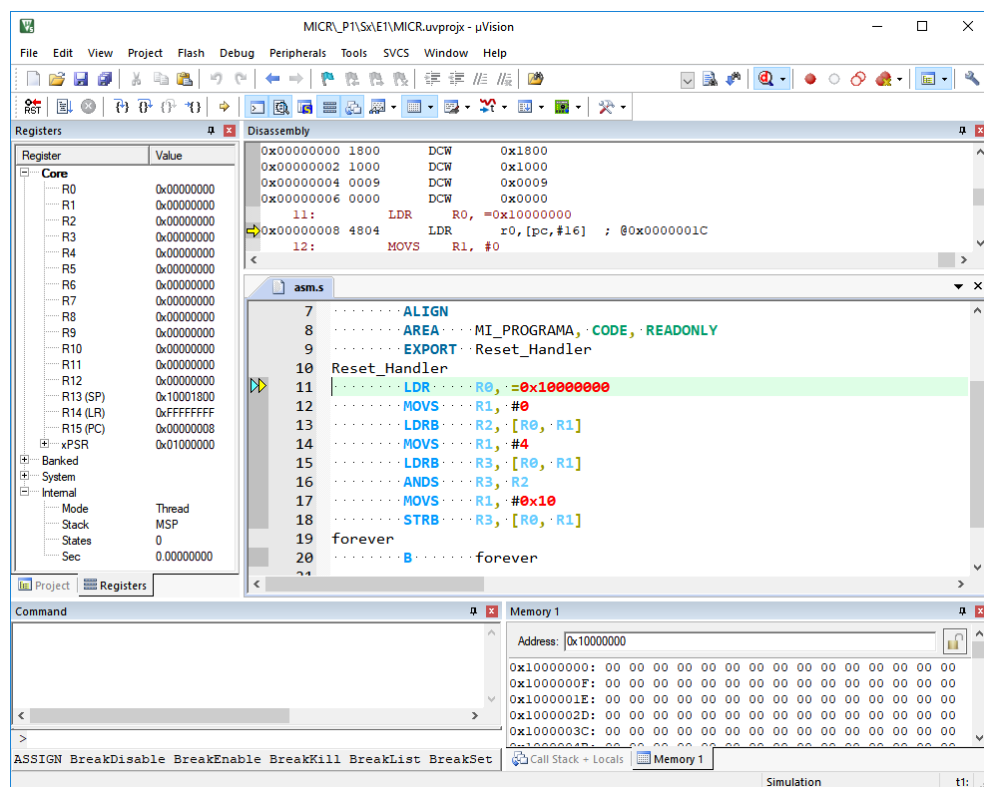















FIGURA 7: Keil µVision 5 en modo de depuración.

Para poder comprobar el funcionamiento de este programa se debe pasar la herramienta al modo de depuración (ver figura 7 anterior) pulsando el icono . Esto hará que el código se pueda simular (sin necesidad de tener la placa) o transfiera a la placa para comenzar la ejecución (depende de la configuración). El proyecto está configurado para *simular* y para que la ejecución se detenga justo en el gestor del *reset* (etiqueta *Reset\_Handler*). A partir de este momento se puede:

- poner *breakpoints* (puntos de ruptura, se estudiarán en el siguiente apartado) mediante el botón  (o picando a la izquierda del número de línea);

- ejecutar el programa directamente empleando el botón .
- ejecutar paso a paso utilizando los botones    —el primero de estos tres hace que, si lo que se trata de ejecutar es una función, se pase a ejecutar paso a paso las instrucciones dentro de la función; el segundo hace que, si se trata de ejecutar una función, esta se ejecute como si fuera una única instrucción; finalmente el tercero de los botones hace que se ejecuten todas las instrucciones hasta salir de la función en la que se encuentra el programa— (las diferencias entre ellos son más útiles en C/C++, aquí use );
- detener la ejecución en curso mediante el botón .
- resetear el procesador mediante el botón .
- Según se ejecuta cada paso, la flecha que aparece a la izquierda del código ( o ) se irá desplazando línea a línea. Tenga en cuenta que se puede ejecutar paso a paso en la ventana inferior (se emplea , que puede contener C/C++ o lenguaje de ensamble, en esta práctica es lenguaje de ensamble) o en la ventana superior (se emplea , siempre lenguaje de ensamble, aun cuando el código fuente sea C/C++). Para identificar qué código se está ejecutando paso a paso debe identificar cuál de las dos ventanas tiene el foco.




Antes de comenzar a ejecutar el programa es necesario tener en cuenta la información que se encuentra almacenada en las direcciones (de byte) de memoria implicadas (0x1000 0000 y 0x1000 0004). Por defecto, y como se muestra en la ventana que aparece en la parte inferior derecha de la figura (Memory 1), las posiciones implicadas están a 0. Por tanto antes de ejecutar la aplicación hay que modificar estos valores. Si no aparece la ventana de memoria porque se haya cerrado previamente, basta con ir al menú de View → Memory Window → Memory1. La dirección de inicio del visualizador de memoria se puede fijar escribiéndola en el campo Address. Es posible visualizar la memoria como bytes (por defecto) y también como datos de otros tamaños, codificados con o sin signo, en decimal, hexadecimal o ASCII. Use el botón derecho sobre el visualizador de memoria para cambiar esto.

Para poder realizar una prueba se propone que en la posición 0x1000 0000 se almacene el dato 0xFA y en la 0x1000 0004 el dato 0x6E. Para modificar el valor de una posición de memoria basta con realizar un doble *click* sobre la posición de memoria y escribirlo.

Cuando el programa está parado es posible consultar y modificar el contenido de la memoria y también los registros del procesador. El

valor de los registros se muestra en la parte izquierda de la pantalla. Inicialmente se aprecia que todos los registros de propósito general (de R0 a R12) se encuentran inicializados a 0.

Una vez que todo esté configurado, es momento de comenzar a ejecutar el programa paso a paso, analizando la evolución del contenido de los registros y la memoria. Hay que recordar que el objetivo de este ejercicio es conocer cómo se depura una aplicación paso a paso, y no tanto analizar la funcionalidad del programa.

Seguidamente conviene conocer la funcionalidad de otros dos botones que aparecen en la vista de depuración. El primero permite reiniciar la ejecución del programa mediante el icono  RST, verá que la aplicación vuelve al comienzo y que los valores de los registros se inicializan. El segundo permite parar la ejecución de un programa que se está ejecutando de forma continua debido a que se haya pulsado sobre el botón , se trata del botón . Para comprobar su funcionalidad realice un *reset* con el botón anteriormente explicado, ejecute el programa de forma continua y pulse el botón de parar, comprobará que el programa se encuentra en el bucle infinito que hay al final del código.

Para finalizar con este ejercicio modifique el programa para que en lugar de emplear datos de 8 bits se utilicen datos de 32 bits y que en lugar de hacer la operación AND se realice la suma aritmética. Emplee como datos en memoria los números 0x9AC3 34FE (en la dirección 0x1000 0000) y 0x33A5 432D (en la dirección 0x1000 0004). **Tenga en cuenta para ello que el procesador *Cortex-M0* de esta práctica trabaja en modo *little-endian*.**

**Deberá enseñar al docente, en el laboratorio al principio de la sesión, su programa funcionando con las modificaciones pedidas y los datos en memoria propuestos.**

### 3.3. Ejercicio 2 - puntos de ruptura

Este ejercicio tiene como objetivo la edición de un programa en lenguaje de ensamble y su depuración —empleando *breakpoints*— para comprobar su funcionamiento, así como observar el contenido final de algunas posiciones de memoria y valores almacenados en los registros.

Para poder realizar este apartado haga una copia del proyecto de *Keil μVision 5* del apartado anterior (es decir, todo lo que se encuentre en



## PRÁCTICA 1: ENTORNO Y LENGUAJE DE ENSAMBLE

MICR\P1\Previo\_S1\E1, excepto las carpetas cuyo nombre comience por «~») en la carpeta MICR\P1\Previo\_S1\E2 y trabaje sobre esta copia. Sustituya el código, a partir de la etiqueta `Reset_Handler`, por:

```
Reset_Handler
    LDR    R0, =0x10000008
    MOVS   R1, #8
    EORS   R2, R2

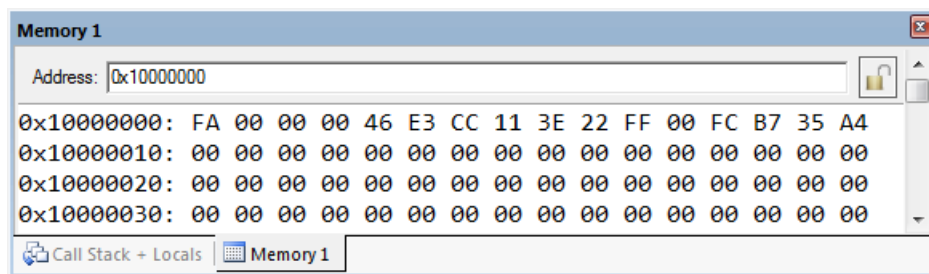
loop
    SUBS   R1, #1
    BMI    exit
    LDRB   R3, [R0, R1]
    LSRS   R3, #1
    BCC    loop
    ADDS   R2, #1
    B      loop

exit
    MOVS   R1, #0x08
    STRB   R2, [R0, R1]

forever
    B      forever

END
```

Edite el programa, ensámblelo, corrija los errores que puedan aparecer y pase al modo de depuración. Adicionalmente, modifique el contenido de las posiciones de memoria que emplea el programa para que contengan los siguientes valores:



Hay que recordar que, por defecto, se emplea el simulador para ejecutar el programa. Ejecute paso a paso el programa tratando de inter-


## MICROPROCESADORES

prestar la funcionalidad. Rellene las siguientes tablas, indicando cómo quedará la memoria y los registros del microprocesador tras la ejecución de todo el programa.

Dirección	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0x1000 0000																
0x1000 0010																

R0	
R1	
R2	
R3	

Como habrá comprobado, el bucle se ejecuta 8 veces, lo que lleva a tener que realizar muchos pasos antes de llegar a la última instrucción del programa. Esto podría ser aún peor si, en lugar de 8 iteraciones, se realizaran cientos, miles o millones. Para poder abordar la depuración de estos programas es necesario poner *puntos de ruptura*. Un punto de ruptura (en inglés *breakpoint*) hace que la ejecución se pare cuando el código llega a una determinada instrucción, para poder evaluar en ella el valor de los registros o las posiciones de memoria.

Para poner un punto de ruptura se debe hacer *click* en la parte izquierda de cada instrucción, en la zona gris a la izquierda del número de línea (ver figura **Error! Reference source not found.**). Este *click* hará que se marque la instrucción con un círculo rojo .

## PRÁCTICA 1: ENTORNO Y LENGUAJE DE ENSAMBLE

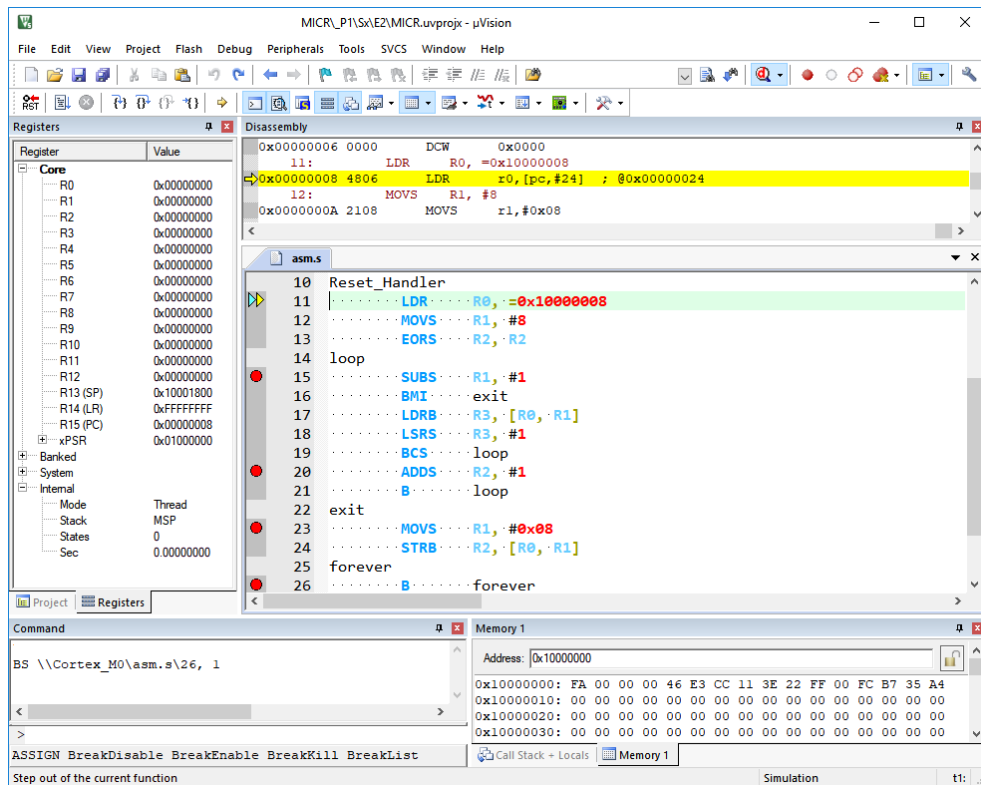





FIGURA 8: Puntos de ruptura —breakpoints—.

A continuación reinicialice el programa con  (cuidado, *no* vuelva al modo de edición con , ya que al hacerlo se pierden todos los valores que haya introducido manualmente en la ventana de memoria) y coloque cuatro puntos de ruptura: el primero en la instrucción asociada a la etiqueta **loop**; el segundo inmediatamente después de la instrucción de salto condicional a **loop**; el tercero en la instrucción asociada a la etiqueta **exit**; y el cuarto en la instrucción asociada a la etiqueta **forever**. En la figura **Error! Reference source not found.** se aprecian las instrucciones en las que se deben colocar los puntos de ruptura.

Una vez establecidos los puntos de ruptura (conviene que, si ha ejecutado el programa previamente, ponga a cero el contenido de la dirección 0x1000 0010) ejecute el programa empleando el botón . Verá que la ejecución se detiene en alguno de los puntos de ruptura. A partir de ese punto puede ejecutarse paso a paso de nuevo o lanzar de nuevo la ejecución continua hasta que se llegue al siguiente punto de ruptura.

Aprovechando los puntos de ruptura que se han definido trate de averiguar si la funcionalidad que infirió anteriormente es correcta. Para

ello es posible que deba modificar el contenido de algunas posiciones de memoria implicadas en el algoritmo.

En relación con los puntos de ruptura, conviene indicar que hay otras funcionalidades asociadas a ellos. Para analizarlo abra la ventana Debug → Breakpoints..., que mostrará lo visualizado en la Figura 9.

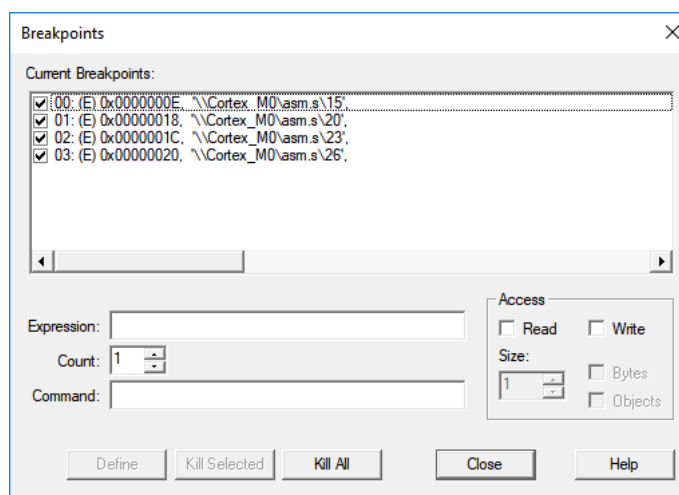


FIGURA 9: Gestión de *breakpoints*.

En esa ventana se pueden definir puntos de ruptura *condicionales* (*conditional breakpoints*), que solo detengan la ejecución en una determinada línea si, al alcanzar esa línea, se verifica además alguna condición, como por ejemplo que un registro tenga un determinado valor o que la línea haya sido alcanzada, previamente, un determinado número de veces. Es posible incluso definir puntos de ruptura sobre direcciones de memoria, de modo que la ejecución del programa se detenga cuando se accede a una determinada dirección de memoria. Por ejemplo, en la figura 10 aparecen definidos tres puntos de ruptura:

- 00. Punto de ruptura de *ejecución* (E) en la línea 15 del fichero asm.s.  
Se corresponde con la instrucción de la etiqueta **loop**. Dicha instrucción está almacenada en la dirección 0x0000 000E del mapa de memoria del procesador. El programa se detendrá justo antes de la ejecución de esa instrucción.
- 01. Punto de ruptura de *ejecución* (E) en la línea 20 del fichero asm.s.  
Se corresponde con la instrucción **ADDS R2, #1**. Este *breakpoint* detendrá el programa justo antes de ejecutar esa instrucción por segunda (o sucesiva) vez (count=2). Dicha instrucción está ubicada en la dirección 0x0000 0018.

## PRÁCTICA 1: ENTORNO Y LENGUAJE DE ENSAMBLE

03. *Breakpoint* de *acceso* (A) que detendrá el programa cuando se *acceda* en escritura (write), por tercera o sucesiva vez (count=3) a la dirección 0x1000 0010.

Para más información sobre el uso de *breakpoints* en *µVision 5* consulte la ayuda del programa.

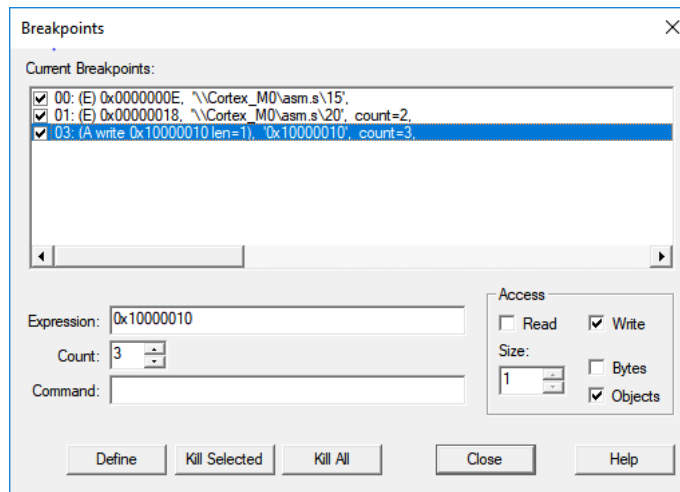


FIGURA 10: Puntos de ruptura avanzados.

**Deberá explicar al docente, en el laboratorio al principio de la sesión, la utilidad del código de este ejercicio indicando, en particular, la relación existente entre los datos inicialmente en memoria y el dato guardado en memoria por el programa. Describa aquí abajo, en una sola frase, dicha utilidad.**

Terminan aquí los trabajos previos a la sesión presencial de esta práctica.

#### 4. TRABAJOS DURANTE LA SESIÓN PRESENCIAL

##### 4.1. Ejercicio 3 - área de triángulo

Realice un programa que calcule el área de un triángulo. La longitud de la base expresada en cm estará disponible en la dirección de memoria `0x1000 0000`. La altura, también en cm, se leerá de la posición `0x1000 0001`. El resultado del cálculo, expresado en  $\text{cm}^2$ , se almacenará en la posición `0x1000 0008`. El valor de los datos de entrada será como máximo de 255 cm. Emplee el tamaño adecuado en *bytes* para almacenar el resultado. Copie el proyecto del apartado 3.2 en la carpeta `MICR\P1\S1\E3` y trabaje sobre esta copia. Es interesante intentar minimizar el número de registros del procesador que se emplean. Este ejercicio se puede realizar fácilmente empleando solo 4 registros del procesador.

##### 4.2. Ejercicio 4 - operaciones aritméticas

En las direcciones `0x1000 0000` y `0x1000 0004` se almacenan dos datos (A y B) codificados en complemento a 2 con 16 bits. Realice un programa en lenguaje de ensamble que calcule las operaciones aritméticas  $(A - B)/2$  y  $AB/4$  y almacene el resultado en las direcciones `0x1000 0010` y `0x1000 0014` respectivamente, empleando en ambos casos el número de bits que sean necesarios.

Copie el proyecto del apartado 3.2 en la carpeta `MICR\P1\S1\E4` y trabaje sobre esta copia. Emplee durante la simulación los siguientes datos en memoria y complete la tabla con los contenidos en memoria al terminar la ejecución. Inicialmente todas las direcciones de memoria a partir de la `0x1000 0008` estarán a cero.

## PRÁCTICA 1: ENTORNO Y LENGUAJE DE ENSAMBLE

Dirección	0	1	2	3	4	5	6	7
0x1000 0000	39	CF	35	14	67	DC	85	E4
0x1000 0010								

Justifique aquí los resultados verificando manualmente las operaciones en decimal.

**Deberá enseñar al docente, en el laboratorio, su programa funcionando.**

### 4.3. Ejercicio 5 - *buffer* de datos

Realice un programa que compruebe si un *buffer* alojado en memoria ha sido almacenado correctamente. El *buffer* estará almacenado a partir de la posición 0x1000 0000.

Los dos primeros bytes del *buffer* (ver figura 11) indican el número de datos (de un byte cada dato) que contiene el mismo (en el ejemplo de la figura: 0x0003 —*little-endian*—). El tercer byte tiene el resultado de realizar la OR-exclusiva de todos los datos del *buffer* (en el ejemplo: 0xA5), desde el cuarto byte hasta el final del *buffer* (en este caso es la XOR de: 0xC6, 0x77 y 0x14). A continuación aparecen los datos (cada dato es de un byte) contenidos en el *buffer* (0xC6, 0x77 y 0x14).

Si el *buffer* se ha almacenado correctamente (la XOR calculada es la misma que la información almacenada en la posición 2: 0xA5 en este caso) se cargará en el registro R0 el valor 0. En cambio, si existiese un

error en el almacenamiento, entonces se cargará el registro R0 con el valor 0xFFFF FFFF.

Realice un programa que implemente esta aplicación y compruebe su funcionamiento depurando su ejecución paso a paso. Copie el proyecto del apartado 3.2 en la carpeta MICR\P1\S1\E5 y trabaje sobre esta copia. Observe que el *buffer* de ejemplo dado a continuación es, efectivamente, un ejemplo, su programa debe ser capaz de trabajar con cualquier *buffer* de cualquier longitud posible (con hasta al menos 32 767 datos en él), lo que le obliga a codificar algún tipo de bucle.

Byte	1º	2º	3º	4º	5º	6º
Función	# de datos		XOR	1º dato	2º dato	3º dato
Contenido	0x03	0x00	0xA5	0xC6	0x77	0x14

FIGURA 11: *Buffer* de ejemplo para el ejercicio 5.

Como referencia debe saber que es posible codificar este programa con 12 instrucciones y empleando 4 registros del procesador.

**Deberá enseñar al docente, en el laboratorio, su programa funcionando.**

#### 4.4. Subida de resultados a Moodle

Elimine todas las carpetas de nombre ~build y ~listings de las carpetas de los ejercicios E1 a E6. Comprima entonces la carpeta P1 completa (en formato .7z) y súbala a *Moodle* en el enlace correspondiente (una entrega por cada estudiante, ambos miembros de la pareja deberán subir el mismo fichero). Al emplear el programa 7-*Zip* para la compresión emplee la opción Añadir al archivo... y, en la ventana que aparece seleccione en Nivel de Compresión la opción Ultra. Emplee siempre este mecanismo para generar los ficheros comprimidos que subirá a *Moodle*.

Terminan aquí las tareas a realizar para esta práctica.