Introducción al entorno de desarrollo

En este laboratorio vamos a utilizar Eclipse como Entorno de Desarrollo Integrado (IDE). Es una aplicación con interfaz gráfica de usuario que nos permitirá desarrollar tanto en lenguaje ensamblador como en C, y depurar sobre un simulador de la arquitectura ARM o depurar en circuito sobre una placa con un procesador de ARM.

Eclipse nos ofrece principalmente la interfaz gráfica y la gestión de los proyectos. Para realizar el resto de tareas hace uso de otras herramientas de GNU: el ensamblador (as), el compilador (gcc), el enlazador (ld) y el depurador (gdb). Además, debemos tener en cuenta que desarrollamos sobre un PC para un entorno con procesador ARM, y por tanto necesitamos hacer compilación cruzada. Para distinguir las herramientas cruzadas de las nativas del PC suele añadírselas un prefijo que describe la arquitectura objetivo para la que se compila, en nuestro caso este prefijo es: arm-none-eabi-. Estas herramientas pueden utilizarse también directamente desde un interprete de línea de comandos (terminal en Linux o Mac OS X o cmd en Windows). En cualquier caso, debemos siempre emplear la sintaxis y reglas de programación propias de estas herramientas.

I. Creación de un proyecto en Eclipse

Para todas nuestras prácticas deberemos crear un proyecto Eclipse para compilación y depuración cruzadas utilizando el plugin GNU ARM. Para ello seguiremos los siguientes pasos:

1. Abrimos Eclipse haciendo doble click sobre el icono del escritorio.

2. Al iniciarse la aplicación nos mostrará una ventana de selección de workspace, como la que la Figura 1.6. Debemos seleccionar un workspace propio, que no es más que un directorio en el que Eclipse guardará información sobre los proyectos de todas nuestras prácticas. Para que el ordenador del laboratorio vaya lo más ágil posible, es conveniente que pongamos nuestro workspace en la carpeta C:\\hlocal. Es im- portante también que siempre guardemos una copia de nuestro workspace en un lugar seguro, porque el directorio hlocal es local al puesto y común para todos los usuarios.

3. Una vez seleccionado se abrirá la ventana principal de Eclipse. Si acabamos de crear el workspace entonces el aspecto será como el que se muestra en la Figura 1.7.

4. Debemos cerrar la pestaña Wellcome haciendo click en la cruz de la pestaña. Entonces la ventana quedará como la que muestra la Figura 1.8, es la perspectiva C/C++ de Eclipse, que está organizada de la siguiente manera:

Panel Izquierdo: el explorador de proyectos. Aparecerán todos los proyectos que tengamos en el workspace, cuando los tengamos.

Panel central: el editor . Nos permitirá editar los ficheros que contendrán el código fuente de nuestros programas.

Panel derecho. Nos permite explorar los símbolos del proyecto activo (funciones, variables, etc).

Panel inferior. Tiene varias pestañas, entre las que destacan la pestaña de errores de compilación y la consola. Desde la primera podemos ver los errores y saltar a la línea de código fuente que los causó haciendo click sobre el error. En la segunda podemos ver los comandos que ejecuta Eclipse para hacer la compilación.

5. Para crear el proyecto seleccionamos File→New→C Project, con lo que se abrirá una ventana como la de la Figura 1.9. Como indica la figura, seleccionamos las opciones ARM Cross Target Application, Empty Project y ARM GCC (Sourcery G++ Li- te). Elegimos el nombre del proyecto y pulsamos Finish. Tendremos el proyecto vacío visible en el explorador de proyectos.

6. Ahora vamos a añadirle un fichero con el código fuente de nuestro primer programa en ensamblador. Para ello seleccionamos File→New→Source File, con lo que se abrirá una ventana como la de la Figura 1.10. Para que Eclipse tome el fichero como un fichero fuente con código ensamblador le ponemos extensión .asm o .S. Una vez creado, haciendo doble click sobre el fichero en el panel izquierdo se abrirá una pestaña en editor, en la que copiamos el código del cuadro 2.

7. Antes de configurar la compilación de nuestro proyecto vamos a añadir a él un fichero más, que servirá para indicar al enlazador cómo debe construir el mapa de memoria del ejecutable final. Aprovechamos para ver otra forma de añadir un fichero al pro- yecto. En el explorador del proyecto seleccionamos el proyecto y pulsamos el botón derecho del ratón, y seleccionamos New→File. Se abrirá una ventana como la de la Figura 1.11, seleccionamos el proyecto y ponemos ld\_script.ld como nombre del fichero. Una vez creado, lo abrimos en el editor y copiamos el contenido del cuadro 3.

8. Finalmente debemos configurar el proyecto para que la compilación se realice co- rrectamente. Para ello seleccionamos el proyecto en el panel izquierdo, pulsamos el botón derecho del ratón y seleccionamos la entrada Properties en la parte inferior del desplegable. Con ello se abrirá una ventana como la que muestra la Figura 1.12. En esta ventana seleccionamos C/C++ Build→Settings y

Debemos comprobar que en Target Processor está seleccionado arm7tdmi co- mo procesador, no están marcadas ninguna de las casillas Thumb\* y está selec- cionada la opción Little Endian.

Debemos seleccionar ARM Sourcery GCC C Linker→General, y en la casilla Script file (-T) debemos escribir la ruta al fichero ld\_script.ld que hemos añadido al fichero. Podemo seleccionarlo gráficamente pulsando el botón Browse.

9. Compilamos el proyecto. Para ello seleccionamos Project→Build Project. Acaba- do este paso habremos obtenido el ejecutable final, con extensión .elf (Executable Linked Format, que se encontrará en el subdirectorio Debug, dentro del directorio de proyecto de nuestro workspace.

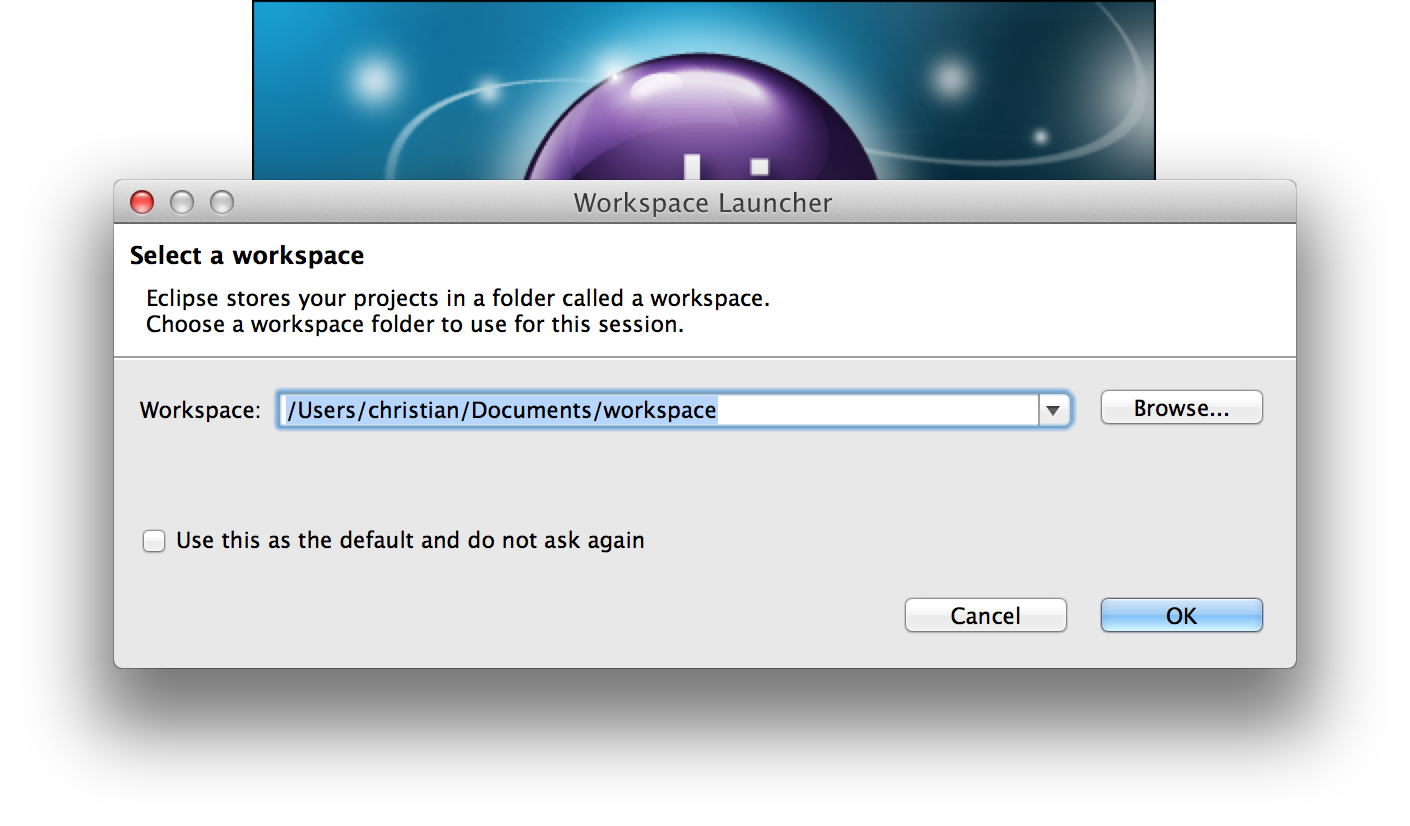


Figura 1.6: Ventana de selección de workspace.



Figura 1.7: Ventana de eclipse al abrir un workspace vacío.

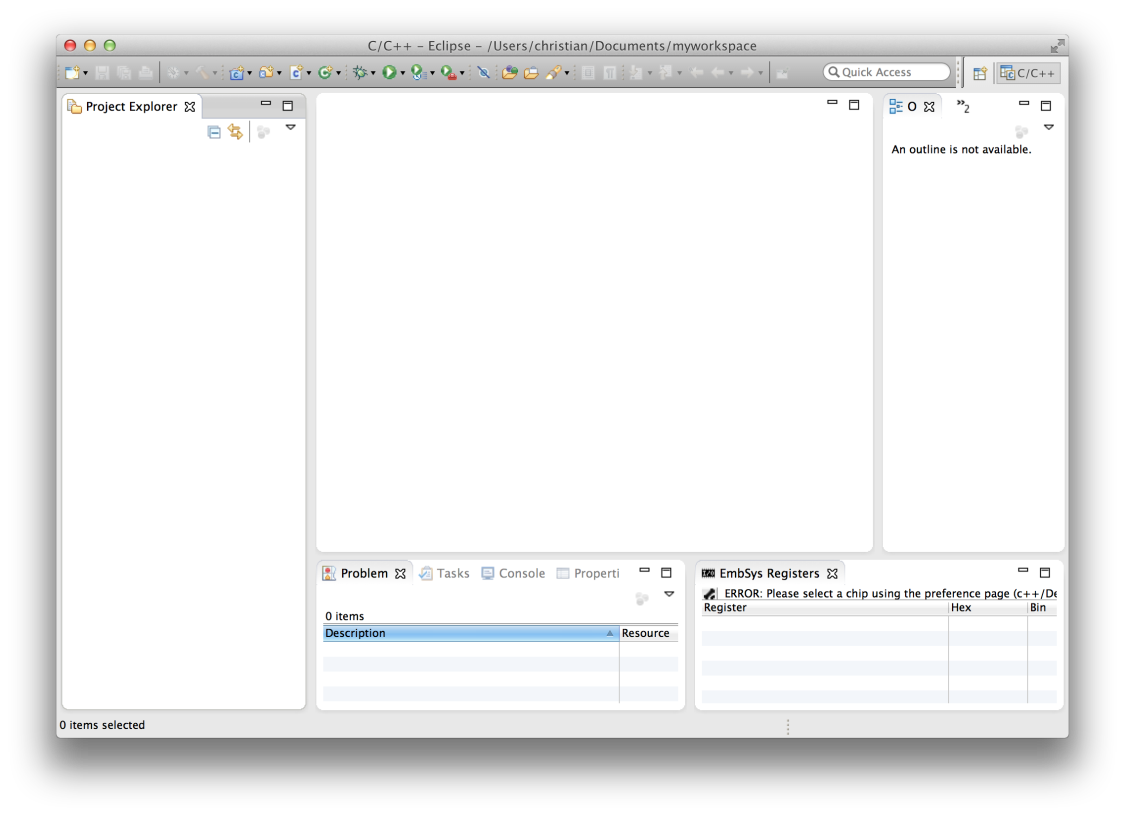


Figura 1.8: Ventana de eclipse con la perspectiva C/C++ sin proyectos.

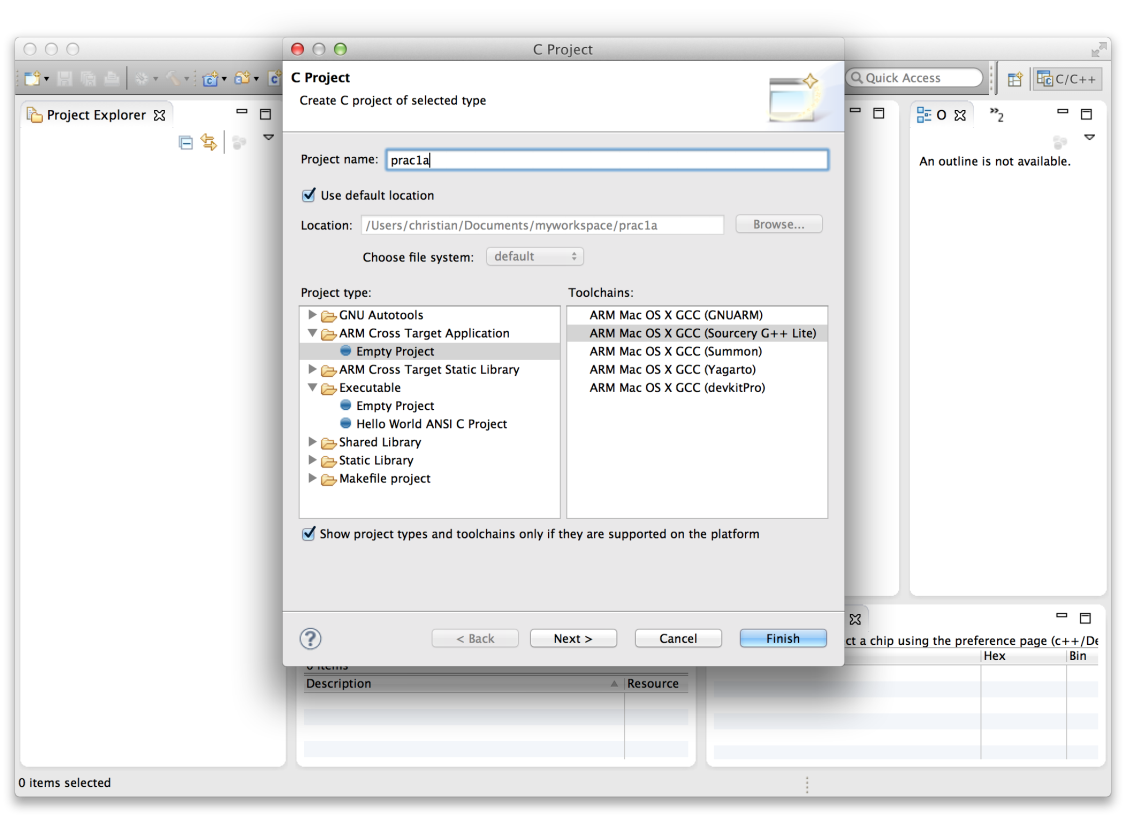


Figura 1.9: Ventana de creación de C project de tipo GNU ARM.

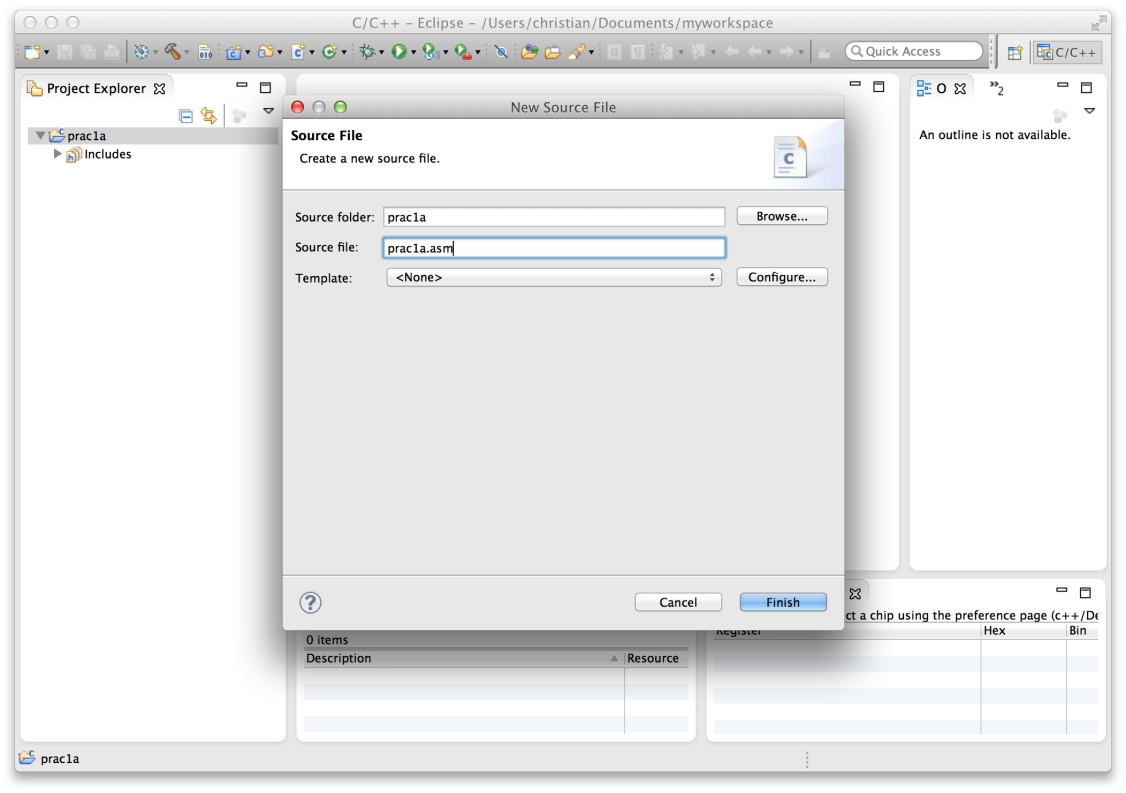


Figura 1.10: Ventana de creación de nuevo fichero fuente.

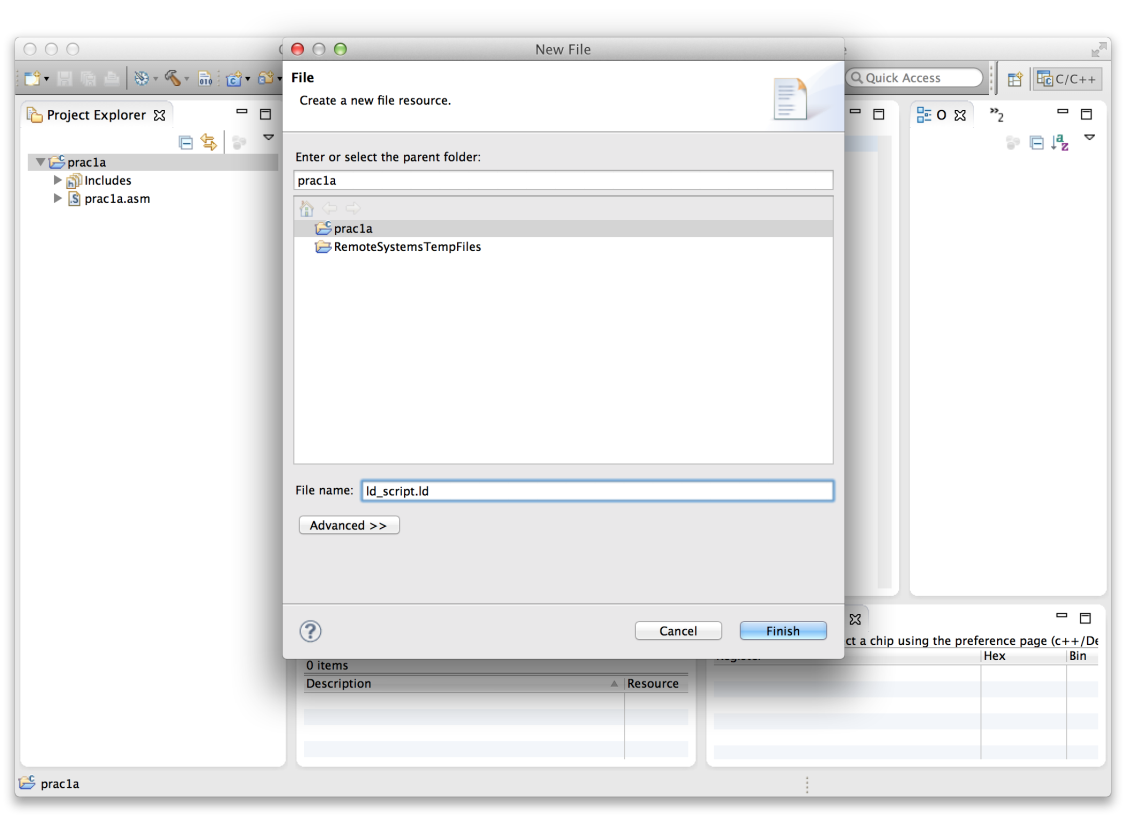


Figura 1.11: Ventana para añadir un nuevo fichero al proyecto.

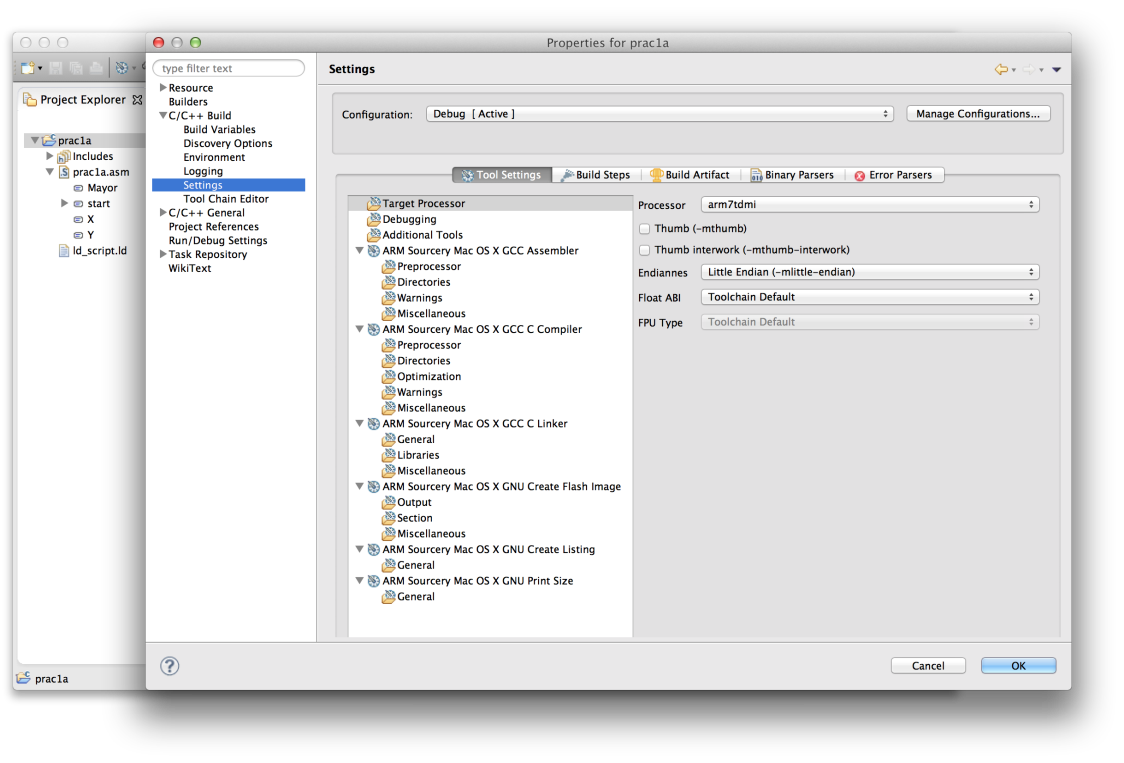


Figura 1.12: Ventana de propiedades (properties) del proyecto.

II. Depuración sobre simulador

Tras generar el ejecutable de nuestro proyecto, nos toca comprobar que funciona co- rrectamente. Para ello usaremos el depurador arm-none-eabi-gdb utilizando Eclipse como interfaz. El depurador nos permitirá ejecutar el programa instrucción a instrucción, po- ner puntos de parada (breakpoints), examinar registros, memoria, etc. En estas primeras prácticas no vamos a utilizar un procesador ARM real, configuraremos el depurador para que utilice un simulador.

Hay dos plugins de Eclipse que nos permiten hacer la depuración con GDB sobre el ARM: GDB Hardware Debugging y Zylin Embedded debug. El primero resulta más sen- cillo para la depuración en circuito, mientras que el segunda es más sencillo para utilizar con el simulador. Por ello en estas primeras prácticas usaremos Zylin.

Para depurar nuestro proyecto seguimos los siguientes pasos:

1. Abrimos la perspectiva Debug. Para ello seleccionamos Window→Open Perspective→Debug. La apariencia de la ventana de Eclipse cambiará a la mostrada en la Figura 1.13. Como podemos ver, está dividida en varias regiones, cada una de ellas con pestañas:

Superior Izquierda: información sobre el proceso de depuración lanzado. Superior Derecha: pestañas donde podemos visualizar los registros, las variables, los breakpoints, . . .

Central Izquierda: código fuente en depuración. Al principio sólo aparecen los ficheros que tenemos abiertos en la perspectiva C/C++. Se irán abriendo auto- máticamente nuevas pestañas si el programa en ejecución salta a alguna función definida en otro fichero compilado con símbolos de depuración.

Central Derecha: en principio sólo nos muestra una lista de los símbolos definidos por el programa. Es interesante añadir a este panel una pestaña con el código desensamblado. Para ello seleccionamos Window→Show View→Disassembly.

Inferior: múltiples pestañas con distinto propósito. Por ejemplo aquí podemos poner el visor de memoria, seleccionando Window→Show View→Memory.

2. Creamos una configuración de depuración para el proyecto usando el plugin Zylin.

Para ello seleccionamos Run→Debug configurations..., y se abrirá una ventana como la mostrada en la Figura 1.14. En el panel izquierdo seleccionamos Zylin Embedded debug (Native) y pulsamos el botón que está en la parte superior iz- quierda del panel () para crear la configuración. Deberíamos tener una ventana como la mostrada en la Figura 1.15. Ahora debemos rellenar correctamente las si- guientes pestañas de la configuración:

* Debugger: en esta pestaña indicamos el depurador a utilizar. En el laborato- rio la ruta al toolchain cruzado se ha añadido al path del usuario, por tan- to basta con poner el nombre del depurador en la entrada GDB debugger: arm-none-eabi-gdb. Además de esto, en la parte superior podemos colocar un breakpoint temporal en donde nosotros queramos. Vamos a colocarlo en la dirección del símbolo start, para ello escribimos \*start en el cuadro Stop on startup at: El resultado se muestra en la Figura 1.16.
* Commands: en esta pestaña damos los comandos que gdb debe

ejecutar en la inicialización al iniciar la depuración. En el cuadro Initialize commands es- cribimos target sim. Esto indica que el objetivo de depuración es el simulador. En el cuadro Run commands escribimos:

load

run

Esto indica al depurador que cargue el fichero seleccionado la pestaña princi- pal, leyendo de éste los símbolos de depuración. La ventana resultante se muestra en la Figura 1.17.

3. Ya estamos listos para depurar, para ello hacemos click en el botón Debug. Esto guardará la configuración de depuración con el nombre seleccionado en la primera pestaña e iniciará una sesión de depuración con esta configuración. Si más adelante queremos volver a depurar este proyecto no será necesario crear una configuración de depuración, bastará con seleccionar la que acabamos de crear. La Figura 1.18 nos muestra el estado inicial que debería tener la ventana en depuración si hemos seguido todos los pasos:

En la parte izquierda del panel central debemos encontrar el código del cuadro 2, con la primera línea de código tras la etiqueta start marcada en verde, con una flecha en el marco izquierdo. Esto quiere decir que el programa a comenzado correctamente su ejecución simulada y que se ha detenido en el breakpoint que hemos puesto en la dirección de start.

En la parte derecha del panel central debemos encontrar el código desensamblado. Es el contenido de la memoria en el entorno de la dirección de la instrucción actual, reinterpretada como instrucciones.

4. Antes de simular nuestro código vamos a abrir una visor de memoria para poder evaluar el valor de nuestras variables. Para ello, lo primero que tenemos que hacer es abrir la pestaña Memory en el panel inferior, si no la tenemos abierta ya (Window→Show View→Memory). Seleccionamos esta pestaña y añadimos un nuevo monitor pinchando en el icono . Se abrirá una ventana como la de la Figura 1.19 en la que debemos escribir la dirección a partir de la que queremos monitorizar la memoria. Si ponemos la dirección de descarga, 0x0C000000, en la que hemos colocado la sección .data, seguida en orden por las secciones .bss y .text, el aspecto de la ventana resultante debe ser similar al mostrado en la Figura 1.20.

5. Ahora, para simular todo el código se pude pulsar sobre el icono de Resume o F8 y después pulsar sobre el icono de Suspend .

¿Cómo sabemos si el código se ha ejecutado correctamente? El dato mayor se ha escrito en la posición de memoria reservada y etiquetada como Mayor. Se puede comprobar su valor en el visor de memoria, que habrá quedado marcado en rojo como ilustra la Figura 1.21.

6. Sin embargo, para entender mejor el funcionamiento de cada una de las instrucciones conviene realizar una ejecución paso a paso. Además, si el resultado no es correcto tendremos que depurar el código para encontrar cuál es la instrucción incorrecta, para lo que una ejecución paso a paso nos será muy útil. Para ello:

Paramos la simulación pulsando el botón Terminate, . Se habilitará el botón Remove all terminated launches  que nos permitirá limpiar el panel Debug (a veces no se habilita este botón, entonces deberemos pinchar con el bo- tón derecho del ratón en la sesión de depuración y seleccionar Terminate and Remove).

Si necesitamos modificar el código, pasamos a la perspectiva C/C++, editamos los ficheros, los guardamos y recompilamos el proyecto. Luego volvemos a la perspectiva debug.

Podemos iniciar una sesión de depuración con la última configuración de depu- ración utilizada pulsando el botón .

Para ejecutar/simular paso a paso tenemos las siguientes opciones:

• Step Over (): ejecuta hasta la siguiente instrucción. Si es una llamada a subrutina para después de la ejecución completa de la subrutina.

• Step Into (): ejecuta hasta la siguiente instrucción. Si es una llamada a subrutina se detiene en la primera instrucción de la subrutina.

• Poner un Breakpoint (punto de parada) y reanudar la ejecución pulsando el botón Resume . Para poner un Breakpoint nos ponemos en el margen izquierdo de alguna instrucción (tanto en el panel de código fuente como en el panel de desensamblado) y hacemos un doble click. Aparecerá la marca . Podemos poner varios Breakpoints, la ejecución se detendrá en aquel que se alcance primero.

Además, nos interesará observar los cambios que va realizando nuestro programa, para ello:

• Para ver cómo va cambiando el valor de los registros a medida que ejecutamos las instrucciones usaremos el visor de registros, que se encuentra en el panel superior derecho. Conviene seleccionar para este visor, con el botón, el layout horizontal.

• En el visor de memoria veremos los cambios que nuestro programa realice sobre la memoria.

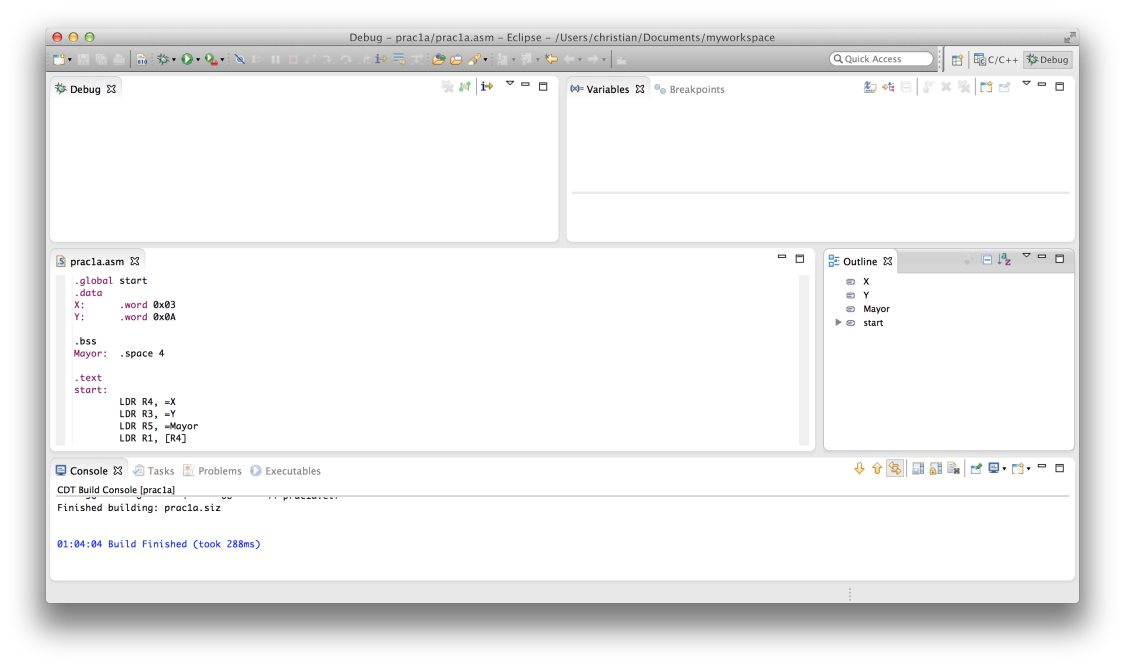


Figura 1.13: Perspectiva de depuración.

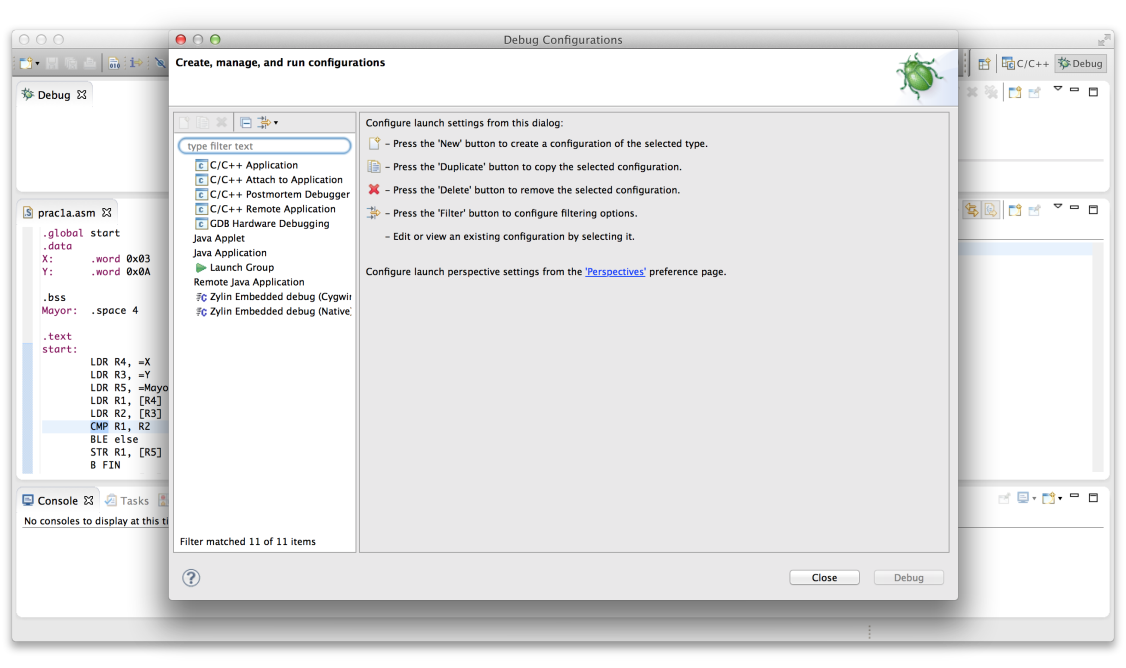


Figura 1.14: Ventana de configuraciones de depuración.

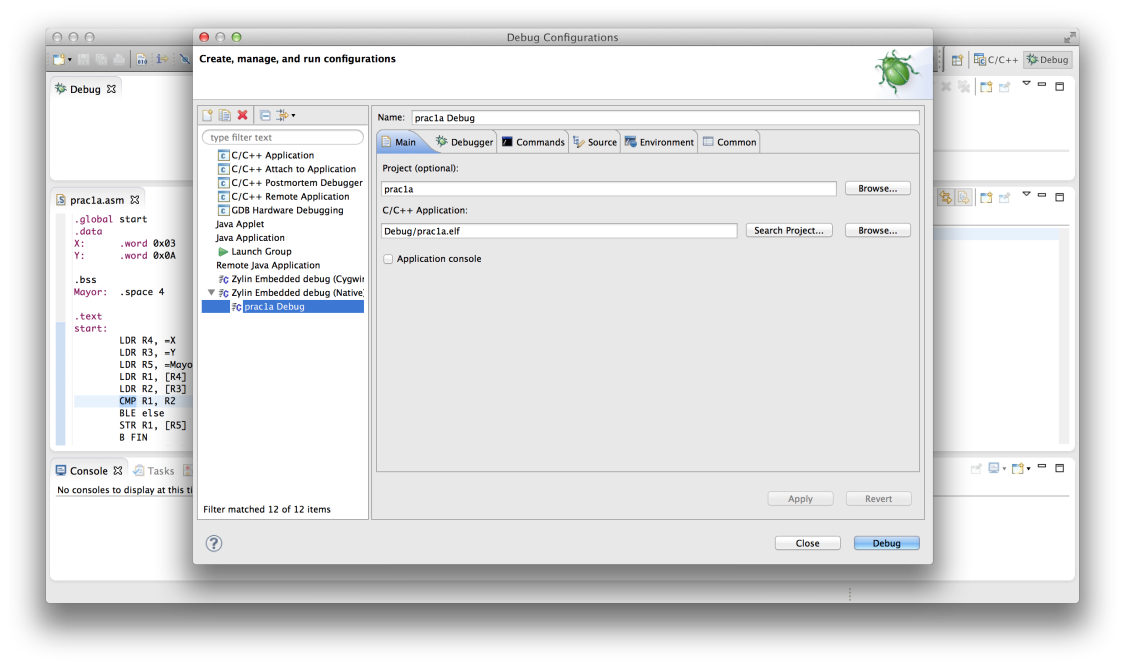


Figura 1.15: Ventana de creación de una nueva configuración Zylin nativa para el proyecto.

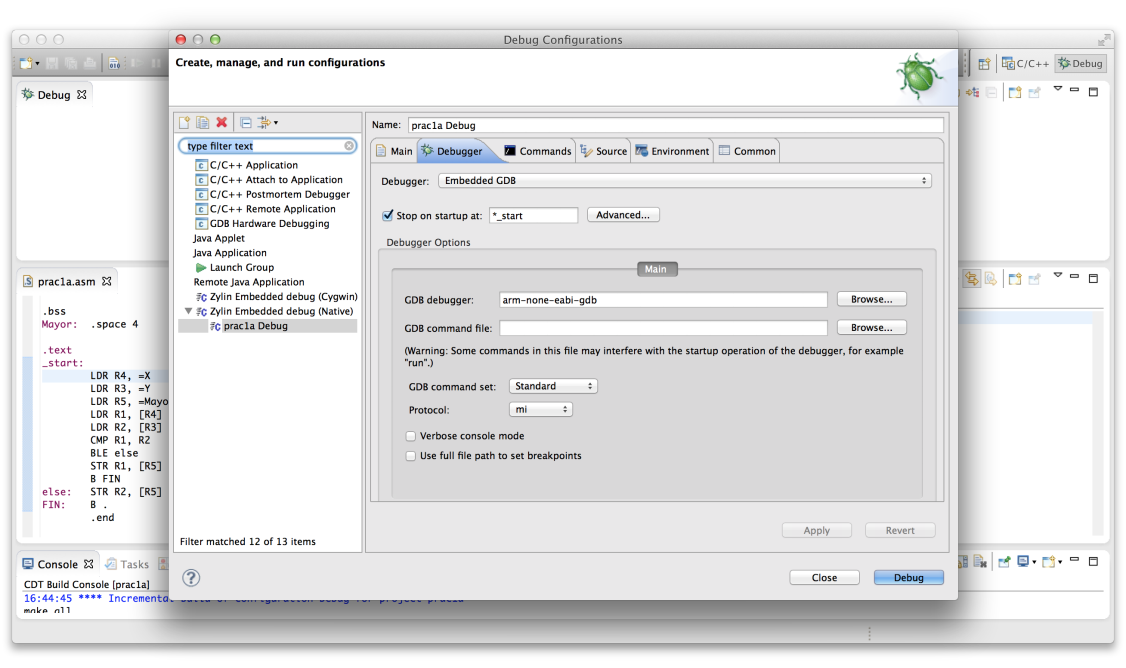


Figura 1.16: Pestaña debugger de la configuración de depuración Zylin.

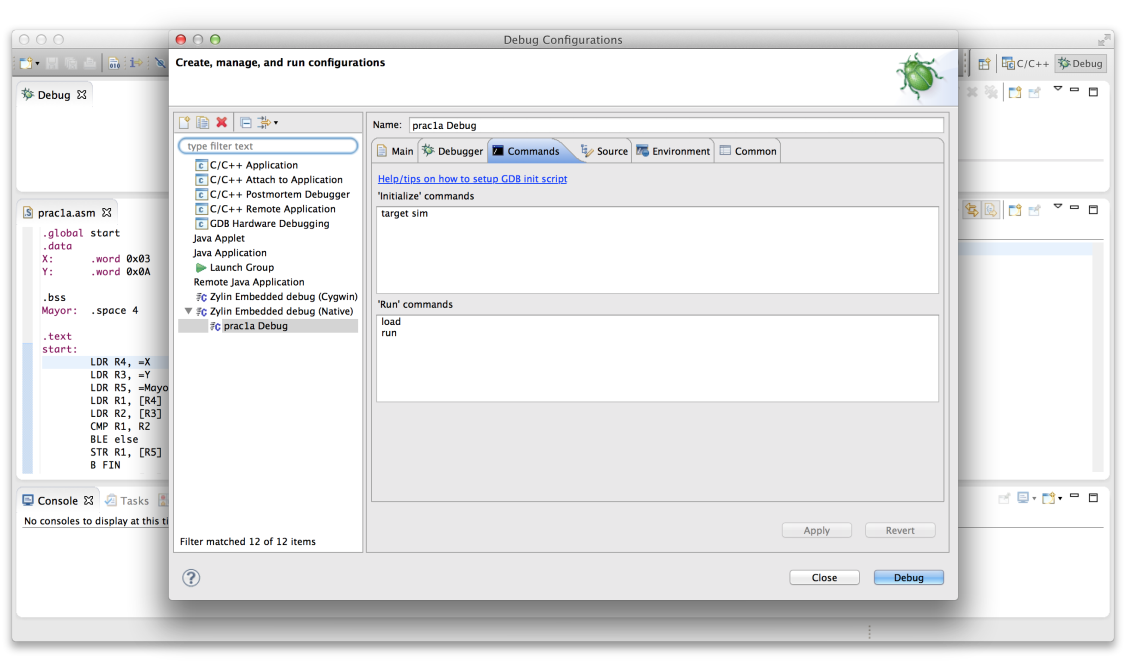


Figura 1.17: Pestaña commands de la configuración de depuración Zylin.

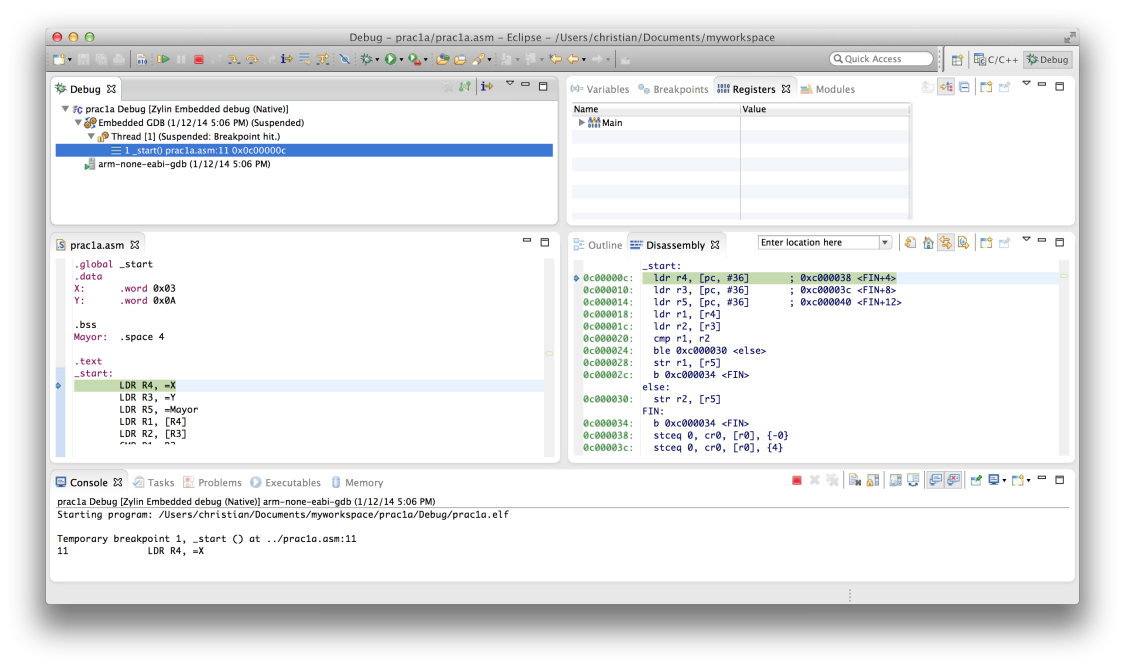


Figura 1.18: Aspecto inicial de la ventana de depuración.

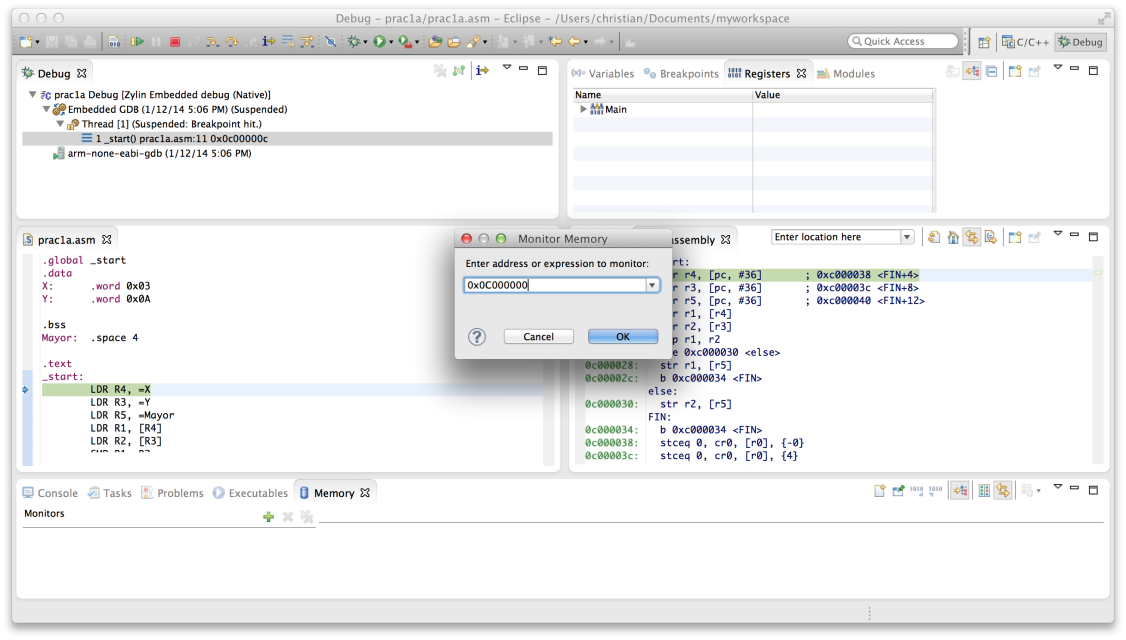


Figura 1.19: Ventana para indicar la dirección del monitor de memoria.

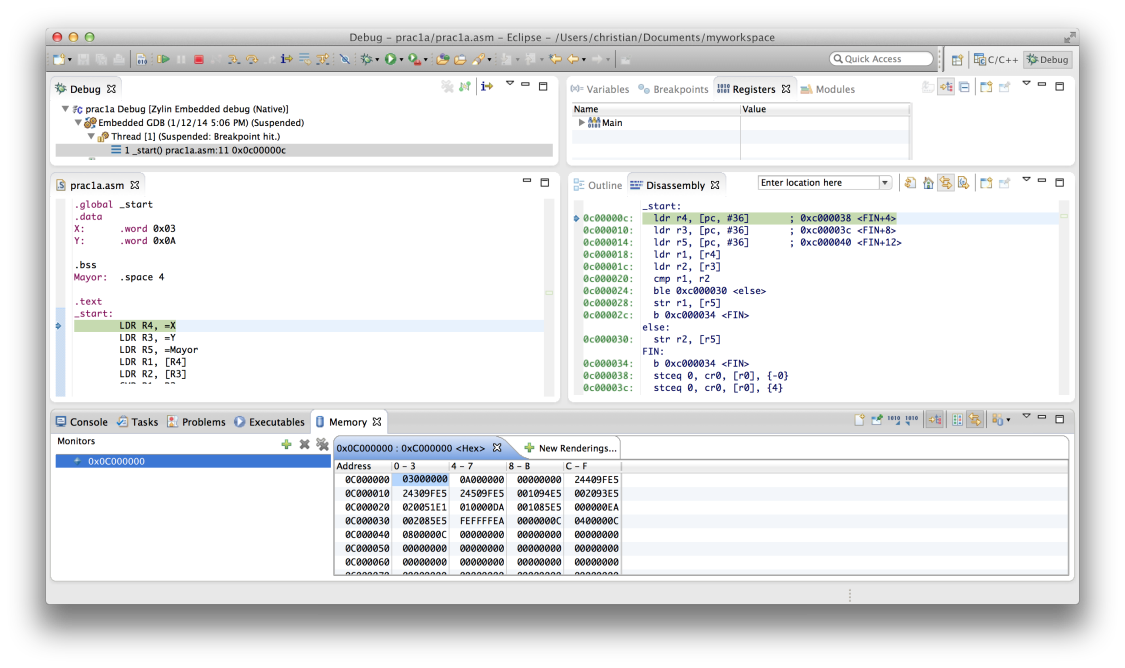


Figura 1.20: Ventana con el aspecto del monitor de memoria en la dirección 0x0C000000.

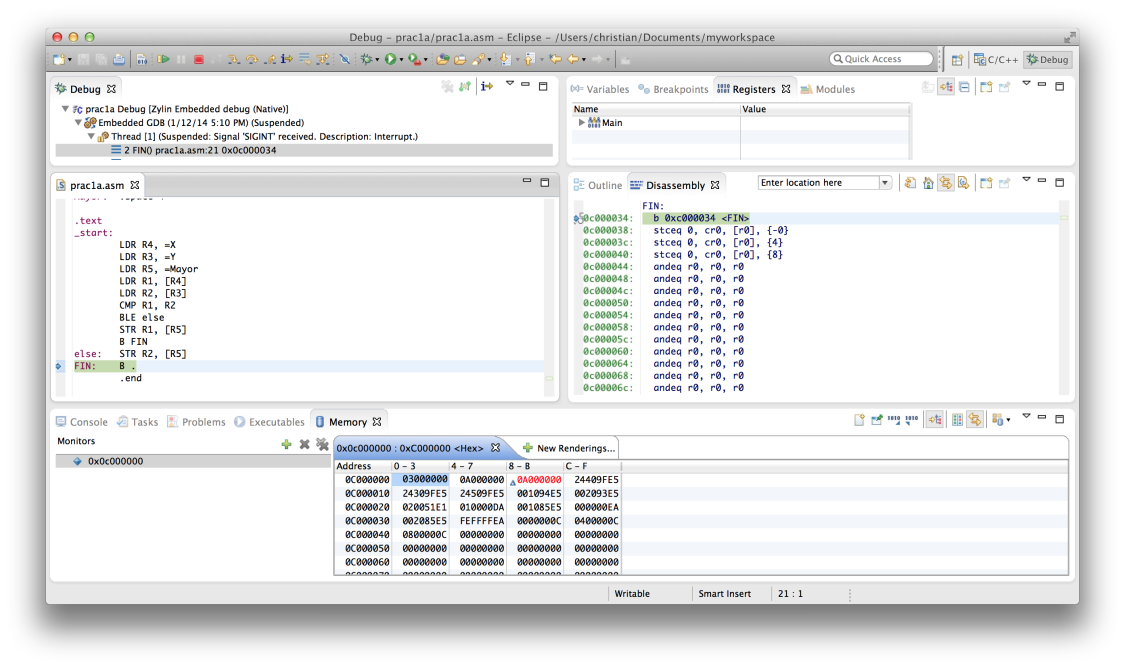


Figura 1.21: Ventana con el aspecto del monitor de memoria tras la simulación completa del programa.