Enseñanza Práctica de Estructura y Organización de Computadores con Raspberry Pi

Cristóbal Camarero, Elena Zaira Suárez, Esteban Stafford, Fernando Vallejo, Carmen Martínez[1]

Resumen-- La docencia en asignaturas de Estructura y Organización de Computadores suele hacerse eligiendo la arquitectura de un procesador como ejemplo de uso. En el caso de la Universidad de Cantabria, tradicionalmente se ha enseñado la arquitectura MIPS, estando los laboratorios basados en ésta. Sin embargo, con la reciente aparición de s dispositivos de bajo coste tipo Raspberry Pi, nos planteamos la posibilidad de cambiar el modelo de procesador, con el objetivo de que los alumnos puedan acceder de manera sencilla a una plataforma para realizar las prácticas de forma más autónoma. En este artículo se resumen los avances de este proyecto docente, desde el punto de vista tanto del hardware como del software, que tendrá su implantación en el siguiente curso 2017/18. Entre estos avances se incluye el desarrollo de un depurador sobre el sistema operativo RISC OS.

Palabras clave-- Arquitectura ARM, Raspberry Pi, RISC OS, Innovación docente, Estructura y Organización de Computadores.

I. Introducción

El estudio de la Estructura y Organización de los Computadores comprende una parte importante de la formación obligatoria de un graduado en Ingeniería Informática. Además, supone los primeros pasos de toda una mención de la titulación, la mención en Ingeniería de Computadores. En esta materia se pretende desarrollar, a diferentes niveles, la competencia específica:

Capacidad de conocer, comprender y evaluar la estructura y arquitectura de los computadores, así como los componentes básicos que los conforman.

Para adquirir dicha competencia, generalmente se estudia la arquitectura de un procesador. Tradicionalmente, la mayoría de las Universidades han optado por procesadores con arquitecturas IA32 (Intel) o MIPS, sin embargo, en los últimos años se está optando por la arquitectura ARM [1]. En la Universidad de Cantabria, en el marco de un proyecto de innovación docente, se ha optado por el cambio a esta nueva arquitectura con la idea de mejorar, sobre todo, la parte de prácticas de laboratorio que debe realizar el alumno.

El objetivo que se ha planteado para este proyecto es lograr que una misma arquitectura fuese empleada en la mayor parte de los niveles de enseñanza de los fundamentos hardware de los computadores. Se pretendía abarcar desde los niveles más bajos, como es el conocimiento del procesador en aspectos como su repertorio de instrucciones, mecanismos de entrada/salida y diseño del camino de datos y la unidad de control. Además, abarcar con la misma arquitectura los aspectos avanzados del computador o del diseño de los sistemas operativos más cercanos al hardware. Inicialmente, se han elegido unos objetivos orientados hacia los niveles más bajos, sin perder de vista que posteriormente debe servir para otros niveles.

Aparte de los aspectos puramente teóricos del diseño de las asignaturas, había que centrarse en la parte práctica y el objetivo planteado se basaba en buscar una solución única para la mayoría de las asignaturas. Además, la experiencia previa ha demostrado que el alumno asimila mejor los conceptos básicos cuando se enfrenta a una plataforma real, en vez de comenzar los primeros pasos con el desarrollo basado en simulación. Por tanto, el objetivo buscado en la parte de laboratorio es encontrar una plataforma basada en la arquitectura ARM capaz de soportar la mayoría de las prácticas. Inicialmente se han elegido unos objetivos orientados hacia los niveles más bajos, sin perder de vista que posteriormente debe servir para otros niveles. De esta forma, se han seleccionado como principales objetivos los siguientes:

- La plataforma seleccionada debe poder generar, ejecutar y depurar código escrito en lenguaje ensamblador.
- Debe permitir mezclar código ensamblador y enlazarlo con código escrito en alto nivel.
- Se debe poder desarrollar y depurar código escrito para la gestión de excepciones e interrupciones con muy baja influencia del S.O.
- Se debe poder conectar dispositivos simples y se deben poder manejar a nivel de sus registros sin interferencia del S.O.

En un primer análisis se barajaron dos alternativas hardware: sistemas basados en Arduino [2] y sistemas

[1] Dpto. de Ingeniería Informática y Electrónica. Universidad de Cantabria, Spain. E-mails: cristobal.camarero@unican.es, elena-zaira.suarez@alumnos.unican.es, {esteban.stafford, fernando.vallejo, carmen.martinez}@unican.es

basados en Raspberry Pi [3]. Sin embargo, el procesador de Arduino no soporta un sistema operativo moderno. Elegimos por tanto Raspberry Pi, que cuenta como otras ventajas su amplia comunidad de usuarios y su extensa información en Internet. Además, varias Universidades Españolas ya habían optado por esta alternativa [4], lo que permitía compartir experiencias.

Por último, para finalizar de definir la plataforma que debe servir de entorno hardware para la nueva arquitectura, hay que determinar el sistema operativo que se debe instalar. En principio la elección se ha realizado sobre tres escenarios. El primero es emplear el sistema operativo Raspbian [5], este sistema operativo es una distribución libre de GNU/Linux basada en Debian y optimizada para la Raspberry Pi. Esta opción ha sido desechada ya que, aunque permite trabajar de forma adecuada en lenguaje de bajo nivel, requiere conocimientos adicionales cuando se quiere trabajar con dispositivos de E/S. La segunda alternativa fue trabajar sin sistema operativo, es decir en modo Bare Metal [6]. Esta opción obliga a desarrollar código en una máquina diferente a la que debe ejecutarse (compilación cruzada) y no existen herramientas de depuración de código, por lo que obligaba a desarrollar algún tipo de depurador para este modo. Por estos motivos también se desechó. La última opción planteada ha sido el trabajar con el sistema operativo RISC OS [7]. Este sistema operativo ha sido íntegramente desarrollado para plataformas basadas en ARM, permite instalar las principales herramientas de desarrollo y tiene un acceso simple a los dispositivos de E/S. Dado que inicialmente parece cubrir los objetivos planteados y hay una versión libre para ser ejecutado en los sistemas Raspberry Pi, se ha optado por usar este sistema operativo.

Este artículo se organiza de la siguiente manera. En la Sección II se detalla el hardware elegido para el laboratorio, en la Sección III el sistema operativo y en la Sección IV el software de desarrollo. En la Sección V se describe el diseño del software de depuración CRISCdbg para depurar programas sobre RISC OS, desarrollado en este proyecto. En la Sección VI se describen tres tipos de prácticas estándar, es decir, programación en ensamblador, manejo de dispositivos periféricos y programación usando interrupciones y excepciones. Para finalizar, en la Sección VII se resumen las principales conclusiones del proyecto y líneas futuras de desarrollo.

II. La Raspberry Pi

Una vez decidido que la plataforma de prácticas iba a ser una Raspberry Pi (en adelante RPi), había que analizar las diferentes versiones y sus características [8]. Actualmente hay cuatro versiones básicas con un total de 6 modelos disponibles:

- RPi 1: Es la versión básica, basada en el System on Chip (SoC en adelante) BCM2835, dispone de un procesador ARM 11, single-core a 700 MHz. Actualmente está disponible en dos modelos A+ y B+ con pequeñas diferencias entre ambos.
- RPi 2: versión del año 2014, está basada en el SoC BCM2836, cuya principal diferencia es el procesador que cambia a un ARM quad-core Cortex A7 a 900 MHz. Actualmente solo está disponible en el modelo B.
- RPi 3: es la tercera generación de la RPi, la única diferencia con su predecesor es el cambio del SoC al modelo BCM2837, la única diferencia en este caso es la sustitución del procesador por un ARMv8 quad-core de 64-bit a 1.2GHz.
- RPi Zero: en 2015 salió al mercado esta versión, de reducido tamaño. Aparte de por su tamaño, se caracteriza por disponer de un número reducido de periféricos. Está basada en el SoC BCM2835 a 1 GHz. Hay a la venta dos versiones, la original y el modelo W que incorpora conectividad inalámbrica (WiFi y Bluetooth).

Cuando se han analizado en detalle las distintas versiones de la RPi para determinar qué modelo debía usarse en el laboratorio, hemos encontrado cierta dificultad para realizar el acceso a los periféricos en los modelos RPi 2 y 3. Esto se debe a que no se ha publicado la documentación de los SoCs de estos modelos. Aunque se puede llegar a encontrar casi toda la información en Internet, no creemos que sea la mejor forma de plantear un dispositivo docente. Además, si tenemos en cuenta el uso que se va a hacer inicialmente con estos dispositivos, no parece necesario disponer de una arquitectura multicore, aunque tampoco llegaría a ser un inconveniente.

Finalmente, se ha tomado la decisión de adoptar como plataforma el modelo RPi 1 B+, que además de cumplir con todos los requisitos deseados, es más barata que los modelos con tecnologías más actuales. También pensamos que se podrían usar los modelos de la RPi Zero, pero poseen menos puertos que la RPi 1 B+ lo que puede dificultar la conexión a un puesto de laboratorio. El modelo RPi Zero W, que se ha empezado a comercializar en febrero de 2017, posee tecnología inalámbrica, por lo que puede estudiarse este modelo como alternativa a la RPi 1 B+.

Todos los modelos seleccionados están basados en el SoC de Broadcom BCM2835 [9] que integra tanto un procesador (CPU) de la serie ARM11, un procesador gráfico (GPU) VideoCore IV, memoria SRAM y varios periféricos. En concreto, el modelo RPi 1 B+ que se ha seleccionado dispone de 512 MB de SRAM, que son compartidos entre la CPU y la GPU, un terminal de tarjeta micro-SD para instalar el software. En cuanto a los principales periféricos, dispone de salida de video por un puerto HDMI, cuatro conectores USB, controlador Ethernet 10/100 y 40 terminales de GPIO donde algunos de ellos se usan para funciones especiales como comunicación serie (UART), interfaz SPI o I2E. Los terminales GPIO permitirán conectar pequeños

periféricos como pulsadores, leds, zumbadores y sensores de distinto tipo.

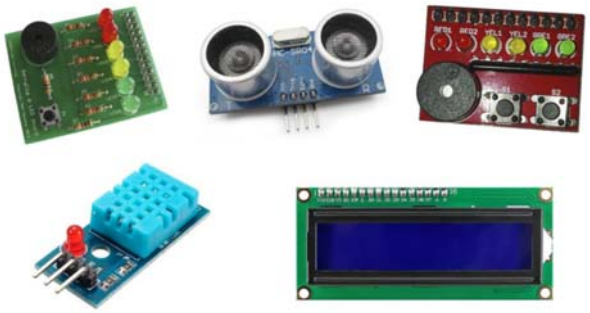

Figura 1 Diferentes periféricos para Raspberry Pi.

La RPi Zero y Zero W, disponen del mismo SoC, luego tienen la misma CPU, GPU y memoria SRAM, pero disponen de una menor gama de periféricos. Únicamente disponen de un puerto USB (OGT), la salida de video se realiza con un puerto Mini-HDMI y no dispone de puerto Ethernet. La versión Zero W dispone de conexión 802.11 b/g/n wireless LAN y Bluetooth (4.1 y BLE), además de los 40 terminales GPIO con la misma funcionalidad. Esto hace que estos modelos también puedan ser usados en este proyecto.

III. Sistemas Operativos en la Raspberry Pi

Para poder tener un entorno más amigable y didáctico se ha decido instalar el sistema operativo RISC OS. Es un sistema operativo que se desarrolló en Acorn Computers para trabajar con chips ARM [7]. Actualmente, su código fuente es mantenido mediante una licencia Opensource por RISC OS Ltd. Se trata del único sistema operativo que esta íntegramente desarrollado en código ensamblador de ARM y resulta bastante sencillo de utilizar.

Del sistema operativo necesitaremos básicamente una serie de llamadas a sistema. Por un lado, las llamadas al sistema que nos dotan de cierta funcionalidad de E/S básica (leer carácter, mostrar cadena, etc). En la Tabla 1 se detallan algunas de las llamadas al sistema que se utilizarán en el laboratorio. Por otro lado, para el trabajo con dispositivos externos y manejo de excepciones, se necesitarán varias llamadas al sistema (cambio de modo de ejecución, deshabilitar interrupciones, etc). En la Tabla 2 se resumen algunas de las necesarias. Para más información se puede consultar el sitio web de RISC OS [10].

Nombre	Código	Argumentos	Resultados	Descripción
OS_Write0	0x02	R0 = Dir. string		Mostrar string.
OS_Exit	0x11			Exit
OS_ConvertInteger4	0xDC	R0 = entero R1= buffer R2 = tamaño buffer	R0 = buffer R1 = dir. null R2 = bytes libres	Convertir un entero de tamaño word a un string.
OS_ReadLine	0x0E	R0 = buffer R1 = tamaño buffer R2 = menor caract. permitido allowed R3 = mayor character permitido	R1 = caracteres leídos, sin return.	Leer string a un buffer
OS_PrintChar	0x5d	R0=carácter		Mostrar carácter

Tabla 1 Llamadas a sistema RISC OS para E/S.

Nombre	Código	Argumentos	Resultados	Descripción
OS_EnterOS	0x16	Modo supervisor		
OS_LeaveOS	0x7C	Modo usuario		
OS_Memory	0x68	Info en []	Info en []	Varias acciones con la memoria. P.e. calcular direcciones lógicas.
OS_ClaimProcessorVector	0x69	R0 = num vector R1 = rutina		Pide un vector
OS_AddToVector	0x47	R0 = vector R1 = rutina		Asignar la rutina indicada al vector seleccionado.
OS_ClaimDeviceVector	0x4B	R0 = nº dispositivo R1 = rutina		Pide un vector de dispositivo

Tabla 2 Llamadas a sistema RISC OS para manejo excepciones y dispositivos.

IV. Software de Desarrollo

Para completar el laboratorio es necesario buscar un conjunto de herramientas de desarrollo. El modelo que se pretende seguir es que las herramientas no formen una plataforma integrada de desarrollo de software, sino que se desean un conjunto de aplicaciones separadas para que el alumno comprenda todos los pasos necesarios para la generación de un programa: edición, ensamblado/compilado, enlazado y carga/ejecución. Además, se pretende que no se trate de una herramienta cruzada, sino que se ejecute sobre la propia RPi con el sistema operativo RISC-OS.

Aunque se barajaron otras opciones, como usar el intérprete de BBC BASIC que viene de forma nativa en el sistema operativo, se decidió que la mejor opción era instalar un paquete herramientas de GNU que bajo la denominación de GCC [11] incorpora un compilador de C, un ensamblador y un enlazador. Estas herramientas permiten generar ejecutables escritos en lenguaje de bajo nivel (ensamblador), en lenguaje de alto nivel (C y C++) y en una mezcla de ambos lenguajes. Además, el alumno debe realizar explícitamente todos los pasos para obtener el ejecutable: compilado (gcc), ensamblado (as) y enlazado (ld).

Una vez instaladas las herramientas se han generado programas simples, usando el editor ¡StrongED que por defecto viene en RISC-OS, y se han ejecutado sin problemas.

El siguiente paso es depurar el código escrito. En este punto se busca que el alumno pueda comprobar sobre la máquina el efecto de la ejecución de una instrucción, analizando cómo se han modificado los registros o alguna posición de memoria.

Para cubrir este objetivo principal, el depurador a instalar debe poder ejecutar un programa paso a paso, insertar puntos de ruptura, ver el contenido de los registros y la memoria, poder ver el código y poder modificar los valores almacenados en registros o memoria. Sería deseable que pudiese usar los símbolos empleados en los ficheros fuente, aunque este requisito no debe considerarse imprescindible, ya que también es interesante hacer ver al alumno que los símbolos escritos en un programa fuente son sustituidos por valores numéricos en los programas ejecutables.

Se han barajado varias opciones de depurador, algunas de las que se han probado han sido:

- Depurador de GNU (GDB) [12], este depurador aunque aparentemente cumple todos los requisitos deseables para este proyecto, no ha sido posible integrarlo con el resto de herramientas y ha tenido que ser desechado.
- Depurador de RISC OS [10], funciona muy bien para realizar programas con BASIC y puede usarse, con cierta limitación, en programas escritos en ensamblador y C, pero no dispone de toda la funcionalidad requerida. Puede considerarse un punto de partida para programas simples.
- Depurador DDT, se trata de un depurador desarrollado para RISC OS que se incluye en el paquete DDE [13], que en principio podría haber sido de utilidad. El problema es que se trata de un producto de pago que no ha podido ser evaluado al no disponer de una versión de prueba.
- Depurador Breakaid [14], se trata de un pequeño depurador aun en desarrollado por un usuario de las RPi que no puede considerarse un depurador cien por cien. Permite ver el contenido de la memoria y algunas direcciones insertando instrucciones tipo break directamente en el código.

Dado que no ha sido posible encontrar un depurador compatible con la funcionalidad deseada, como parte de este proyecto se ha iniciado el desarrollo de un depurador propio que cumpla los requisitos planteados. Además, se pretende que el depurador permita el análisis del código asociado a rutinas de tratamiento de las excepciones/interrupciones.

V. Desarrollo de un depurador

Ante la falta de funcionalidad de los depuradores analizados para RISC OS, se ha decidido crear un depurador propio para RISC OS que se ha llamado CRISCdbg.

Ya que estamos sobre un sistema operativo con entorno gráfico, se ha decidido que el depurador tenga una interfaz gráfica; con lo que recibe inspiración de otros depuradores gráficos como es por ejemplo OllyDbg para Windows [15].

A. Funcionalidad deseada

En esta subsección describimos la funcionalidad deseada del depurador, para más adelante detallar el estado del prototipo actual.

- El depurador debe ser capaz de cargar ficheros binarios ELF [16] en memoria para su posterior ejecución. Además, debe mostrar información por varias ventanas: código, registros, datos, pila, comandos. Las ventanas podrían usarse de forma interactiva para, por ejemplo, fijar puntos de ruptura, cambiar valores de memoria o cualquier otra acción de las que se fijen por comando.
- Debe poder ejecutar el código, según el modo de ejecución: hasta el siguiente breakpoint, paso a paso, hasta la siguiente SWI, etc. Además, debe guardar y mostrar el estado de los registros de la aplicación, incluyendo el CPSR y los registros de punto flotante.
- Debe poder añadir y eliminar puntos de ruptura, tanto mediante la interfaz como mediante la ventana de comandos.

- Debe permitir que una lista de variables se ancle a la ventana de datos, es decir, que las variables estén siempre visibles.
- Debe ser capaz de editar los valores de los registros, de la memoria, de ensamblar instrucciones, etc.
- Debe capturar cada posible excepción: por ejemplo, si una aplicación provoca un acceso no alineado indicarlo y pausar la ejecución.
- El depurador debe poder analizar el código dentro de las excepciones gestionadas por el programa que se está depurando.

Hay funcionalidades más avanzadas que podrían incluirse de forma opcional:

- Ser capaz de cargar un programa en la dirección que indique el propio archivo ELF; como puede coincidir la dirección con la del propio CRISCdbg habría que cambiar la MMU cada vez que se fuese a entrar y salir de ejecutar.
- Análisis de código: se podría mostrar una marca desde el comienzo de un bucle hasta el final, así como el punto de entrada a las funciones, tal como hace OllyDbg.

B. Esquema del Depurador

El código del depurador presenta dos partes claramente definidas, la primera parte es un código principal que se encarga de la comunicación con el usuario y la segunda parte es el código que permite ejecutar el programa que debe depurarse. El código principal se encargará de la gestión del contenido de las distintas ventanas que forman el depurador, tendrá un intérprete de comandos y dará paso al proceso de depuración a petición del usuario.

El proceso de depuración se basa en la introducción en el código a depurar de instrucciones tipo break. Antes de comenzar a ejecutar el código, en función del comando seleccionado, se insertarán estas instrucciones break en los puntos donde se desea que se detenga la ejecución del código, se cambiará el contexto de ejecución y se saltará al programa bajo depuración. El programa se puede detener al llegar a una de las instrucciones introducidas, un error de programa o por la finalización normal. En cualquier caso, hay que salvar el contexto de ejecución, mostrar el mensaje de lo que ha ocurrido y volver al interfaz de usuario.

Un caso especial a tener en cuenta es la ejecución paso a paso donde debe introducirse la instrucción break justo después de la instrucción que se desea ejecutar. En este caso hay una situación especial que se produce cuando la instrucción que se va a ejecutar produce una bifurcación del código de forma condicional. La arquitectura ARM presenta varias instrucciones de este tipo al permitir la ejecución condicional de instrucciones, luego en este caso hay que analizar la instrucción que se va a ejecutar y comprobar si puede producir un salto, en este caso se debe calcular la posible dirección de destino y colocar también en esa dirección una instrucción break. Algunas de las instrucciones que deben analizarse son las instrucciones tipo branch (B y BL), instrucciones de movimiento con destino el PC (MOV, LDR, LDM) y alguna operación aritmética con destino el PC (SUBS).

El depurador es un programa que usa el entorno WIMP de RISC OS, así que tiene que satisfacer los requisitos de cooperative scheduling y llamar a Wimp_Poll periódicamente. Hacer el depurador cooperativo es más seguro, ya que de otra forma el SO podría expulsar la tarea de depuración después de haber cambiado el manejador de una excepción.

C. Estado Actual

Actualmente el depurador se encuentra en una fase inicial de su desarrollo, pero ya se ha demostrado su viabilidad y puede ser utilizado para depurar pequeños programas, con las siguientes limitaciones:

- El código que se desea depurar se carga por defecto al iniciarse el proceso de depuración, debe llamarse test y tiene un tamaño limitado a 64 KB incluyendo código y datos. La limitación de tamaño viene dada por la implementación que se ha usado, donde la estructura completa del ejecutable se carga sobre un espacio del área de datos del depurador. Además, esta estructura fija las direcciones que deben usarse en el proceso de enlazado del código.
- Toda la información se muestra únicamente sobre dos ventanas. La primera se encarga de mostrar en un solo bloque las áreas de código y datos. Para cada posición, ya sea código o datos, se muestra la dirección de memoria, su valor en hexadecimal, en ASCII y el desensamblado. Se ha optado por resaltar la siguiente instrucción que debe ejecutarse. En la ventana de registros, actualmente, se muestran únicamente los 16 registros de propósito general, aunque en breve se añadirá el CPSR y los registros de punto flotante.
- No se ha desarrollado el intérprete de comandos y únicamente se detecta la pulsación de cualquier tecla. Una vez detectada la pulsación de una tecla se llevan a cabo dos posibles acciones, si se trata de la tecla ‘r’ (run) se ejecuta el programa sin introducir puntos de ruptura, esto es hasta que finaliza normalmente. Si se pulsa cualquier otra tecla se ejecuta paso a paso detectando las instrucciones de bifurcación únicamente en las instrucciones B, BL y MOV desde registro.

VI. Desarrollo de Prácticas de Laboratorio

En esta sección se describe la realización de tres prácticas tipo de las diferentes asignaturas involucradas en el proyecto. En primer caso se considera una práctica de programación en ensamblador, típica de primer curso.

Seguidamente, se dan las bases para la realización de las prácticas que involucran E/S. Finalmente, se describe cómo se afrontaría una práctica de manejo de interrupciones.

A. Prácticas de ensamblador

Para la realización de una práctica típica de ensamblador donde el alumno diseña, codifica y depura un pequeño programa. Todo el desarrollo debe hacerse sobre la plataforma RPi mencionada en la Sección II. En este apartado se va a describir el proceso de depuración tanto si se lleva a cabo en el entorno de depuración de RISC OS, como usando el depurador CRISCdbg que se está desarrollando en el grupo, tal y como se encuentra en la actualidad y ha descrito en el Sección V.

En el enunciado de la práctica se pide al alumno que realice un programa que efectúe la suma de dos vectores de enteros de 32 bits en un tercer vector. Además, dicho programa debe mostrar un mensaje al finalizar.

Se pedirá a los alumnos que creen un directorio de trabajo Practicas, con dos subdirectorios "s" y "o". Para ensamblar y cargar los alumnos tendrán que hacer:

```
#as –o pratica1.o practica1.s
#ld –o practica1 practica1.o
```

Para ejecutar en el modo depuración, usando el depurador de RISC OS:

```
#load practica1 8000
#go 8088
```

De esta manera, se puede ver la memoria de instrucciones, la memoria de datos, el contenido de los registros y poner puntos de ruptura. Por ejemplo:

```
#breakset 80c4
#showregs
```

Nos mostrará el contenido de los registros a la salida del lazo que recorre los vectores. Sin embargo, el depurador de RISC OS no permite la ejecución instrucción a instrucción de forma cómoda.

Para depurar con CRISCdbg, habrá que enlazar practica1 en la dirección 98000, es decir:

```
#ld –Ttext 98088 –o practica1 practica1.o
```

Seguidamente, habrá que ejecutar el depurador, haciendo doble clic sobre el ejecutable CRISCdbg. Automáticamente aparecen dos ventanas, una con el contenido de la memoria (datos e instrucciones) y otra con el contenido de los registros. Un ejemplo de la vista se encuentra en la Figura 2. La depuración se podrá realizar ejecutando instrucción a instrucción, pulsando cualquier tecla, excepto la "r", que se encarga de la ejecución hasta el final como se ha comentado anteriormente.

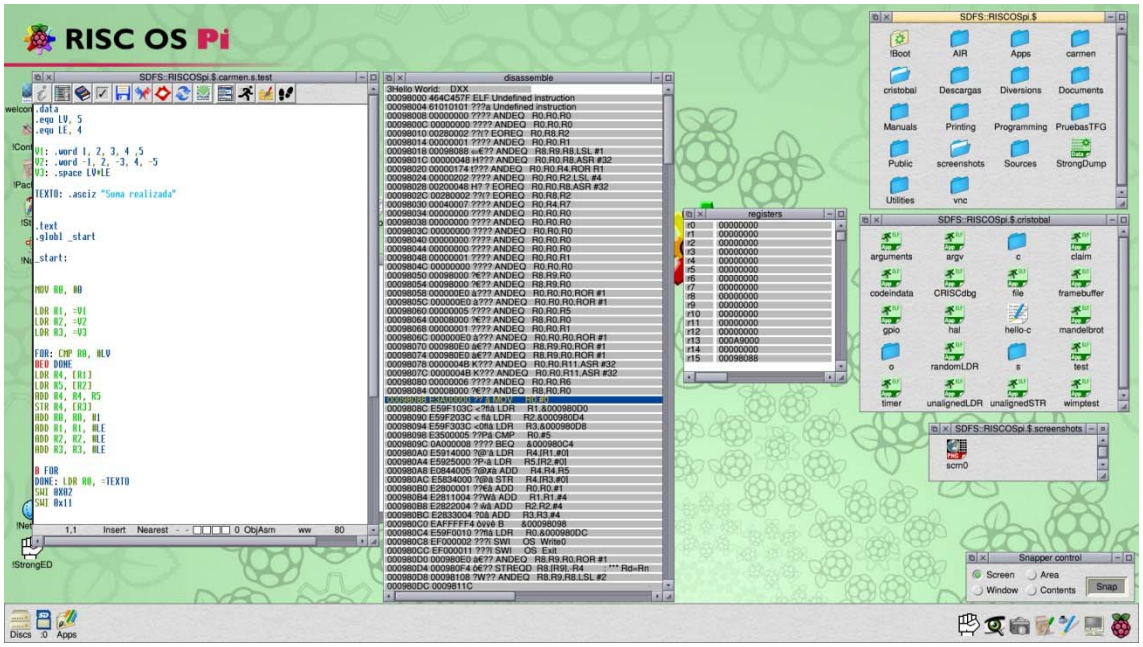

Figura 2 Vista de las ventanas del depurador CRISCdbg.

B. Prácticas de E/S

En la asignatura de Estructura de Computadores, la RPi se usará para las prácticas de E/S. En esta subsección, para realizar un ejemplo de programación básico de E/S en el que no sea necesario manejar interrupciones, vamos a considerar un pequeño dispositivo, que dispone de 6 leds, 2 pulsadores y un zumbador, ya que es el más sencillo para programar y hay información detallada de sobre él [4]. La dirección base de la GPIO en la RPi 1 B+ es 0x20200000.

Para modificar cualquiera de los registros del dispositivo hay que utilizar primeramente dos llamadas a sistema. La primera, OS_Memory, para obtener la dirección lógica que le asocia RISC OS a la GPIO. Seguidamente, la llamada a sistema OS_EnterOS, para cambiar el modo de ejecución a supervisor. De esta manera, establecer un led para una operación de salida será activar cierto bit en el registro GPFSEL0, por ejemplo. A continuación, encenderlo o apagarlo supone una escritura en los registros GPSET0 y GPCLR0, respectivamente.

C. Gestión de excepciones/interrupciones

Finalmente veremos los pasos para realizar una práctica de gestión de excepciones y otra de interrupciones. En estos casos RISC OS posee un gestor que se encarga de determinar la fuente de la excepción/interrupción y llamar a la rutina de tratamiento correspondiente realizada por el alumno.

Un ejemplo de práctica del primer caso, el tratamiento de una excepción, puede ser plantear al alumno la realización de determinadas acciones cuando se encuentre con un acceso a memoria en una dirección no permitida, esto generará una excepción de tipo "data abort". En el caso el alumno deberá realizar una llamada al sistema denominada "OS_ClaimProcessorVector", pasando como argumento la dirección de memoria de la rutina. Hay que tener en cuenta que, en la RPi seleccionada, si se pretende trabajar con excepciones por problemas de alineamiento, se debe configurar el procesador para que produzca este tipo de excepciones. Un ejemplo de código para las excepciones de alineamiento sería:

```
;En R0 en los 8 primeros bits numero vector a tratar
;Address exception y el bit 9 un 1 para habilitar
MOV  R0, #0x104
LDR  R1, =rutina_atencion
MOV  R2, #0
SWI  "OS_ClaimProcessorVector"
```

En la práctica de manejo de interrupciones, se plantea al alumno la gestión de una interrupción generada por un dispositivo externo simple, por ejemplo un timer. En este caso la llamada al sistema a utilizar para la ubicación de la rutina será "OS_ClaimDeviceVector" y al programar la rutina del timer se hará algo de forma similar al ejemplo anterior.

```
MOV  R0, R4              ; Timer device IRQ number
ADR  R1, timerhandler    ; Device driver routine
MOV  R2, R12             ; Workspace pointer
SWI  "OS_ClaimDeviceVector"
```

Donde timerhandler es un manejador de las interrupciones del timer desarrollado por el alumno. Además, existen llamadas a sistema específicas para activa/desactivar interrupciones en general.

VII. CONCLUSIONES Y TRABAJO FUTURO

La docencia de las asignaturas de Estructura y Organización de Computadores se suele fundamentar en laboratorios basados en simulación o en hardware real. En la Universidad de Cantabria hasta el momento se combinaban estas dos opciones, con un laboratorio basado en la arquitectura MIPS. Sin embargo, existe un problema en cuanto a que los alumnos no pueden disponer del hardware real para trabajar de forma autónoma, siendo posible sólo realizar las prácticas en el laboratorio de la Facultad.

En este proyecto nos planteamos la posibilidad de basar la docencia de las asignaturas de Estructura y Organización de Computadores en la arquitectura ARM. De esta manera, los laboratorio podrían dotarse con las placas Raspberry Pi, que suponen hardware real de esta arquitectura y a un costo lo suficientemente bajo para que los alumnos puedan disponer de uno en casa. Así, se han estudiado los diferentes modelos de RPi, hasta seleccionar el adecuado para nuestro propósito. También, se ha evaluado la posibilidad de utilizar el sistema operativo RISC OS, para el apoyo de estas prácticas. Además, se ha considerado todo el software necesario para el desarrollo.

Como conclusiones, hemos visto que es perfectamente posible trabajar las competencias de las materias a consideración con esta plataforma. Hemos seleccionado la RPi 1 B+ como la más adecuada. Hemos visto que RISC OS es lo suficientemente versátil como para poder trabajar la E/S a bajo nivel, pero con el apoyo de un sistema operativo. Finalmente, se ha elegido GCC como el entorno adecuado para compilar, ensamblar y enlazar. Sin embargo, hemos visto que no es posible utilizar ningún entorno de depuración de los que existen actualmente, por lo que hemos decido desarrollar uno propio. En el artículo hemos detallado el estado actual del entorno de depuración CRISCdbg.

Como trabajo futuro consideramos que, principalmente, el desarrollo del depurador para alcanzar la funcionalidad deseada es uno de los aspectos clave para el éxito del proyecto. Consideramos también importante el estudio pormenorizado de los diferentes dispositivos E/S para RPi disponibles en el mercado. Finalmente, consideramos interesante explorar las posibilidades de esta plataforma y el sistema operativo RISC OS en cuanto a la docencia de las asignaturas de Sistemas Operativos o Multiprocesadores.

VIII. AGRADECIMIENTOS

Este proyecto se ha financiado con la III Convocatoria de Proyectos de Innovación Docente, del Vicerrectorado de Ordenación Académica y Profesorado de la Universidad de Cantabria.

IX. BIBLIOGRAFÍA

[1] D. Seal, ARM Architecture Reference Manual (2nd Edition), Addison-Wesley Professional, 2001.

[2] Arduino, «ARDUINO HOME,» [En línea]. Available: https://www.arduino.cc/. [Último acceso: 05 2017].

[3] Raspberry Pi Foundation, «Raspberry Pi,» [En línea]. Available: https://www.raspberrypi.org/. [Último acceso: 05 2017].

[4] A. J. Villena Godoy, R. Asenjo Plaza y F. J. Corbera Peña, «Prácticas de ensamblador basadas en Raspberry Pi,» 2015. [En línea]. Available: http://riuma.uma.es/xmlui/handle/10630/10214. [Último acceso: 05 2017].

[5] Raspberry Pi Foundation, «RASPBIAN,» [En línea]. Available: https://www.raspberrypi.org/documentation/raspbian/. [Último acceso: 05 2017].

[6] «Bare Metal Forum,» [En línea]. Available: https://www.raspberrypi.org/forums/viewforum.php?f=72. [Último acceso: 05 2017].

[7] RISC OS Open, «RISC OS for the Raspberry Pi,» [En línea]. Available: https://www.riscosopen.org/content/downloads/raspberry-pi. [Último acceso: 05 2017].

[8] Raspberry Pi Foundation, «Raspberry Pi Products,» [En línea]. Available: https://www.raspberrypi.org/products/. [Último acceso: 05 2017].

[9] B. Corporation, «BCM2835 ARM Peripherals,» 2012. [En línea]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf. [Último acceso: 05 2017].

[10] B. Smith, Raspberry Pi, Assembly Language, Risc Os Beginners, ISBN-13: 978-099239162, 2014.

[11] Free Software Foundation, Inc., «GCC, the GNU Compiler Collection,» [En línea]. Available: http://gcc.gnu.org/. [Último acceso: 05 2017].

[12] I. Free Software Foundation, «GDB: The GNU Project Debugger,» [En línea]. Available: https://www.gnu.org/software/gdb/. [Último acceso: 05 2017].

[13] RISC OS Open, «Desktop Development Environment,» [En línea]. Available: https://www.riscosopen.org/content/sales/dde. [Último acceso: 05 2017].

[14] R. Murray, «BreakAid v0.01,» [En línea]. Available: http://www.heyrick.co.uk/software/breakaid/. [Último acceso: 05 2017].

[15] «OllyDbg,» [En línea]. Available: http://www.ollydbg.de/. [Último acceso: 05 2017].

[16] Wikipedia, «Executable and Linkable Format,» [En línea]. Available: https://es.wikipedia.org/wiki/Executable_and_Linkable_Format. [Último acceso: 05 2017].