Actas de las Jenui, vol. 4. 2019.
Páginas: 279-286

Proyecto Tetris: aprendizaje de la programación en ensamblador por piezas

Manuel E. Acacio, Lorenzo Fernández-Maimó, Ricardo Fernández-Pascual,
Pilar González-Férez, Alberto Ros, Rubén Titos-Gil
Dpto. de Ingeniería y Tecnología de Computadores
Universidad de Murcia
30100 Murcia
{meacacio,lfmaimo,ricardof,pilargf,aros,rtitos}@um.es

Resumen

El aprendizaje del lenguaje ensamblador constituye con frecuencia uno de los objetivos formativos de alguna de las primeras asignaturas de Arquitectura de Computadores del Grado en Ingeniería Informática. Si bien el desarrollo y la depuración de programas en lenguaje ensamblador resultan esenciales para ayudar a comprender el funcionamiento básico de un procesador, son aspectos que presentan especial dificultad y/o falta de atractivo para el alumnado. En este trabajo presentamos nuestra experiencia con la enseñanza del lenguaje ensamblador MIPS a través de la codificación del videojuego Tetris. El proyecto Tetris se desarrolla en el contexto de una asignatura de primer curso y segundo cuatrimestre. Para que resulte asequible a este nivel, se proporciona al alumno una versión incompleta del programa, que habrá de completar mediante la traducción directa a ensamblador de funciones escritas en lenguaje C (también proporcionadas), y mediante la codificación del resto de la funcionalidad directamente en ensamblador. El resultado es una versión del juego plenamente operativa. El desarrollo se realiza utilizando una versión extendida del simulador MARS. Los resultados obtenidos por los alumnos muestran que este proyecto les facilita el aprendizaje del ensamblador, pues el 85.6 % de los que superan el proyecto Tetris aprueban también el examen de prácticas.

Abstract

Learning assembly language represents typically one of the formative objectives of some of the first courses related to computer architecture in Computer Engineering degrees. Although the development and debugging of programs written in assembly language are essential to help students understand the basic operation of a processor, they are also aspects that present special difficulty and/or lack of attractiveness for students. In this work we present our experience in the teaching of the MIPS assembly language through the coding of the videogame Tetris. The Tetris project is developed in the context of a first-year and second-term course. In order to be affordable at this level, the student is provided with an incomplete version of the program, which must be completed through the direct translation to the MIPS assembly language of functions written in the C language (also provided), and through the implementation of additional functionality directly in assembly language. The result is a fully operational version of the game. The development is done using an extended version of the MARS simulator. The results obtained by the students show that this project facilitates the learning of the assembly, since 85.6 % of those who pass the Tetris project also pass the practicum exam.

Palabras clave

Arquitectura de computadores, Ensamblador MIPS, Tetris, MARS, Aprendizaje Basado en Proyectos, Aprendizaje Experimental.

1. Introducción

Uno de los objetivos formativos de las primeras asignaturas de Arquitectura y Tecnología de Computadores en los planes de estudio de Grado en Ingeniería en Informática es la comprensión por parte del estudiante de la arquitectura de un procesador básico y cómo este lleva a cabo la ejecución de los programas. Para ello resulta de gran utilidad instruir al alumno en el aprendizaje del lenguaje ensamblador, habitualmente el asociado con algún repertorio de instrucciones RISC (*Reduced Instruction Set Computer*), como MIPS, ARM o más recientemente RISC-V. A través de la programación y depuración de programas escritos en lenguaje ensamblador, el estudiante comprende los distintos tipos de instrucciones que se han de incluir en el ISA

(*Instruction Set Architecture*) de un procesador moderno, los operandos con los que dichas instrucciones pueden trabajar, y en general la forma en la que las instrucciones que conforman el programa y los operandos de las mismas están almacenados en memoria (como secuencias de 0s y 1s) y cómo el procesador va accediendo a todo ello a través de las direcciones de memoria asignadas a dichos elementos.

A pesar de las grandes ventajas que tiene la enseñanza del lenguaje ensamblador en los primeros cursos de la carrera, esta resulta con frecuencia una tarea bastante ardua. La mayoría de los estudiantes han sido introducidos a la programación a través de alguno de los lenguajes de alto nivel existentes, como por ejemplo Java, con lo que es usual que encuentren la programación en lenguaje ensamblador como algo tedioso (en el mejor de los casos) y frustrante (en el peor de los casos), con la consiguiente desmotivación que ello conlleva. Este es un hecho ampliamente reconocido y al que habitualmente se trata de poner remedio a través de la definición de proyectos prácticos que resulten de interés para el estudiante, bien por su aplicabilidad en determinados campos, bien porque se trate de desarrollos que les resultan especialmente atractivos, como pudiese ser el de un videojuego. Por citar solamente algunos ejemplos de lo primero, en [2] se propone una herramienta software que simula un robot autónomo que debe ser controlado mediante un programa externo (tan complejo como se desee) desarrollado en lenguaje ensamblador x86. De forma similar, en [3] se muestra otra herramienta software que simula un radar controlado a través de un programa externo encargado de calcular una coordenada a partir de un ángulo y una distancia. En [1] se describen las aplicaciones ARMSim y QtARMSim que proporcionan un entorno didáctico de simulación de la arquitectura ARM Thumb y tratan de explotar como factor motivador el hecho de que ARM es una arquitectura actual y ampliamente difundida, especialmente en dispositivos móviles, smartphones y tablets. En cuanto a proyectos prácticos que tienen que ver con el desarrollo de videojuegos, mencionar la experiencia descrita en [6] para cursos avanzados de lenguaje ensamblador, o más recientemente, la propuesta de desarrollo de videojuegos en máquinas con recursos limitados realizada en [5], en la que se les pide a los estudiantes el desarrollo de un videojuego para Amstrad CPC 464 (un ordenador antiguo que empleaba un procesador Zilog Z80 y disponía únicamente de 64 KiB de memoria RAM) tratando de aprovechar al máximo los escasos recursos disponibles. Por último, en [4], se describe un proyecto consistente en la implementación de Flappy Bird, un popular juego reciente de plataformas móviles, para CHIP-8, una máquina virtual poco conocida de los años 70, usando Octo, un ensamblador moderno en línea con entorno de desarrollo y emulador de dicha arquitectura.

En este trabajo presentamos nuestra experiencia con la enseñanza del lenguaje ensamblador MIPS en el contexto de una asignatura del segundo cuatrimestre del primer curso del Grado de Ingeniería en Informática. Este lenguaje ensamblador es utilizado además en la asignatura de Compiladores de segundo curso. Nuestra aproximación trata de motivar al estudiante a través del desarrollo de un videojuego ampliamente conocido, Tetris. Para que el desarrollo resulte asequible en este punto de la carrera, teniendo en cuenta que son alumnos de primer curso con muy poca práctica en programación en general, se les proporciona una versión incompleta del programa en código ensamblador de MIPS, que utilizarán como punto de partida. En una primera fase (la primera pieza del puzzle), el estudiante habrá de ampliar el código ensamblador para obtener una versión del juego con una funcionalidad básica (controlar la forma en la que las piezas van cayendo). Para ello, se le suministra también el código de una versión en C del programa que incluye dicha funcionalidad (la versión en ensamblador se ha obtenido inicialmente traduciendo la versión en C, por lo que ambas versiones están estructuradas exactamente igual). El primer trabajo del alumno consistirá en ir traduciendo de C a ensamblador MIPS las funciones que no aparecen en el código ensamblador recibido. El hecho de disponer del código C facilita mucho la labor al estudiante durante sus primeros pasos en el aprendizaje del lenguaje ensamblador. Una vez que durante la primera fase el estudiante ha adquirido cierta destreza con el desarrollo de código en ensamblador, en una segunda fase (segunda pieza) tendrá que ampliar aún más las capacidades del programa ensamblador codificando él mismo la funcionalidad de ciertos aspectos relativamente sencillos y que se les indican, como por ejemplo, la gestión de un marcador de puntuación. En una tercera fase (tercera y última pieza), aquellos estudiantes que así lo deseen, pueden incluir además características más avanzadas. Se emplean las metodologías de Aprendizaje Basado en Proyectos (ABP) y Aprendizaje Experimental a lo largo de 8 sesiones prácticas. En concreto, los alumnos se organizan en grupos de 2 personas y disponen de un boletín en el que se explica con detalle la estructura del programa y las tareas a realizar en las sesiones prácticas, contando con la supervisión directa del profesorado de la asignatura durante las mismas.

Para la realización del proyecto se ha utilizado la herramienta MARS, un simulador de MIPS escrito en Java diseñado específicamente para la docencia, y que por lo tanto, resulta muy simple de utilizar por parte de los alumnos. En concreto MARS dispone de un IDE (*Integrated Development Environment*) que permite escribir, depurar y simular programas escritos en

ensamblador de MIPS sobre una máquina idealizada sin sistema operativo pero con un conjunto de llamadas al sistema emuladas que permiten realizar entrada/salida y otras funciones simples. Eso sí, ha sido necesario ampliar y mejorar algunos aspectos de la herramienta para poder llevar a cabo el desarrollo de videojuegos sobre MARS (la herramienta se distribuye bajo licencia GPLv2). Entre otras cosas, se ha mejorado la interfaz de usuario en lo que respecta a las llamadas al sistema relacionadas con la entrada/salida, la simulación de la entrada por teclado, permitiendo la lectura de las teclas pulsadas en la terminal mediante el uso de MMIO (*Memory Mapped Input Output*) e interrupciones, y se ha incorporado un depurador de convenios de programación que detecta el uso incorrecto de los registros y genera avisos en tiempo de ejecución.

El proyecto de programación Tetris empezó a aplicarse en el curso 2015–2016, y lo largo de este tiempo hemos constatado una mayor motivación de los estudiantes por la realización de las prácticas y una mejora en el aprendizaje del lenguaje ensamblador, que se pone de manifiesto en el mayor porcentaje de alumnos que superan el examen final de ensamblador al que se les somete (80.7 %). La calificación obtenida en el proyecto Tetris es un indicador claro acerca de las posibilidades de éxito del alumno en dicho examen final de prácticas. Así, el 40 % de los alumnos que no aprueba el proyecto Tetris (nota menor que 5) tampoco logran superar el examen de prácticas. Este porcentaje cae al 22 %, 16 % y 7 % para los alumnos que obtienen en el proyecto una calificación de Aprobado, Notable o Sobresaliente, respectivamente. Globalmente, el 85.6 % de los alumnos que superan el proyecto Tetris consiguen aprobar también el examen final de ensamblador.

2. Aspectos metodológicos

El principal objetivo de la parte práctica de la asignatura en la que se enmarca el proyecto Tetris es que el alumno conozca el repertorio de instrucciones de un procesador RISC (MIPS en este caso) y que sea capaz de realizar programas en ensamblador para dicho ISA. De esta forma, la evaluación del aprendizaje de la parte práctica de la asignatura se concentra sobre la capacidad del alumno para desarrollar código en ensamblador de MIPS, siguiendo de forma adecuada los convenios de programación definidos (uso de registros, paso de parámetros a funciones y devolución de valores y gestión de la pila, entre otras cosas).

Tres son los elementos a través de los cuales se mide el grado de adquisición de las competencias prácticas por parte del alumnado. El más importante es la realización de una prueba de programación en ensamblador (examen práctico), individual y delante del ordenador, que determina el 70 % de la nota final de prácticas y en la que los alumnos han de obtener una calificación mínima de 5 puntos sobre 10. De cara a asegurar una adecuada preparación por parte del alumno de dicha prueba de programación, se organizan durante el curso 12 sesiones de prácticas de programación en ensamblador. Durante las primeras tres sesiones se realizan sendos boletines básicos de prácticas en los que se explica (1) el funcionamiento del simulador MARS y un primer programa en ensamblador sencillo, (2) la traducción de estructuras de datos y control de C a ensamblador y (3) los convenios de programación en ensamblador, paso de parámetros a funciones y uso de la pila. La entrega de estos boletines, opcional pero recomendada, constituye el 5 % de la nota final de prácticas. El 25 % restante se obtiene a través de la realización del proyecto Tetris, al cual se dedican las 9 sesiones de prácticas restantes y para el que no se exige nota mínima (su entrega es también opcional). Tanto los boletines como el proyecto Tetris se llevan a cabo en grupos de 2 alumnos. El estudiante supera la parte práctica de la asignatura (cuyo peso en la nota final representa el 50 %) cuando la nota media ponderada de prácticas, considerando los tres elementos anteriores, es mayor o igual a 5 puntos (habiendo obtenido al menos 5 puntos en la nota del examen de prácticas).

La realización del proyecto Tetris se basa en las metodologías de Aprendizaje Basado en Proyectos y Aprendizaje Experimental. Tal y como se ha comentado, se planifican un total de 9 sesiones de prácticas para su desarrollo. Durante la primera sesión, el equipo docente de la asignatura explica aspectos generales acerca de la estructura y organización del proyecto y da una serie de recomendaciones a la hora de abordar el desarrollo del mismo (se les insiste en que deben probar la nueva funcionalidad conforme se va incluyendo, y se explica la forma de hacerlo). En las 8 sesiones restantes, cada grupo de 2 alumnos debe completar la realización del proyecto. Disponen para ello de un boletín en donde se explica con detalle la estructura del programa y las tareas a realizar en estas sesiones prácticas, que contarán con la supervisión directa del profesorado de la asignatura durante las mismas.

El estudiante percibe el proyecto Tetris como el mecanismo que debe utilizar para poder preparar adecuadamente la prueba práctica a la que será sometido al final del cuatrimestre. Esta percepción, que el equipo docente de la asignatura transmite al alumnado desde el primer día, evita las típicas situaciones de copia entre los grupos y minimiza el desequilibrio entre el trabajo realizado por los miembros del grupo, al mismo tiempo que motiva al alumno para resolver tantos ejercicios como su disponibilidad horaria le permita.

Figura 1: Piezas del Tetris.

3. El proyecto Tetris

A través del proyecto Tetris pretendemos conseguir que el alumno aprenda a desarrollar programas en el lenguaje ensamblador de una manera divertida, y de forma gradual. Para ello, usamos una versión del famoso juego Tetris [8], que se lanzó al mercado en el año 1984 y que se convirtió en un fenómeno mundial, siendo el número 1 en la lista de videojuegos más vendidos de todos los tiempos y considerado por muchos como el mejor videojuego creado hasta la fecha.

El juego está basado en el concepto matemático de los «tetrónimos», que son un conjunto de fichas que se pueden construir acoplando 4 cuadrados por sus lados. Este concepto, a su vez, es una simplificación de los «pentónimos», que han sido utilizados para plantear diversos puzles combinatorios. En total existen 7 «tetrominós» que forman las piezas del juego del Tetris y que se muestran en la Figura 1.

Para conseguir un aprendizaje gradual e incremental, dividimos el proyecto en tres fases con distintos objetivos que se enumeran a continuación:

- La primera fase, de *traducción*, tiene como principal objetivo el acercamiento al ensamblador del alumno, usando para ello la traducción de un lenguaje de alto nivel a ensamblador. Una vez acabada, el alumno debe de ser capaz de traducir con soltura una serie de estructuras básicas de los lenguajes de programación que incluyen los bucles (`for` y `while`), estructuras condicionales (`if/else`), accesos a variables globales y locales y acceso a arrays y estructuras de datos.
- El objetivo principal de la segunda fase, llamada de *implementación*, es que el alumno sea capaz de entender el diseño de un programa sencillo en ensamblador y realizar modificaciones a ese programa de forma autónoma. Las modificaciones las debe realizar directamente en ensamblador.
- La tercera fase, de *ampliación*, persigue que el alumno afine sus conocimientos en ensamblador, desarrollando funcionalidades más avanzadas.

Para conseguir estos objetivos, el alumno dispone de dos programas que usará durante todo el desarrollo del proyecto. Uno de estos programas es una implementación en lenguaje C «completa», aunque simplificada, del juego. Esta versión en C permite jugar, aunque de forma limitada ya que no elimina las filas completas ni mantiene ningún tipo de puntuación. El otro programa es una versión en ensamblador del código C a la que se le han quitado algunos procedimientos, por lo que no funciona inicialmente. Esta segunda versión será sobre la que se realice la práctica.

La versión en C sirve al alumno de guía a la hora de entender la versión en ensamblador a lo largo de la práctica y además sirve de especificación para realizar la primera fase del proyecto. Así mismo, le permite ver el aspecto final que debe tener su programa al ejecutarse tras esa primera fase de traducción.

En el esqueleto del programa ensamblador proporcionado hay ya implementados una serie de procedimientos y funciones con aspectos básicos del programa. Su propósito es servir de base y de muestra a la hora de implementar el resto.

La tarea a realizar por el alumno durante la primera fase consiste en completar de forma incremental y guiada, la versión simplificada en ensamblador del juego. Para ello, el alumno solo tiene que traducir a ensamblador los procedimientos seleccionados cuya implementación se ha proporcionado en C pero no en lenguaje ensamblador. Al finalizar esta fase, el alumno consigue una implementación que le permite jugar con exactamente la misma funcionalidad que el programa original en C. Durante esta fase, además de aprender a traducir código de un lenguaje de alto nivel como es C, a lenguaje ensamblador, el alumno se familiariza con la estructura del programa proporcionado, lo que será útil durante las siguientes fases.

En total el alumno tiene que traducir 9 procedimientos con diferente grado de dificultad. Estos procedimientos tienen de media 6 líneas de código. A pesar de ser cortos, incluyen todos los detalles de programación que el alumno tiene que aprender como bucles y bucles anidados, estructuras condicionales, llamadas a funciones y paso de parámetros, accesos a variables globales y locales y acceso a arrays y estructuras de datos. Así, uno de los procedimientos de 6 líneas en lenguaje C se traduce en un procedimiento con más de 40 líneas en lenguaje ensamblador.

Un aspecto que el alumno ha de tener en cuenta durante esta fase de traducción, es la necesidad de diseñar una batería de pruebas para asegurar la corrección de cada procedimiento traducido y, en su caso, proceder a su depuración. A modo de ejemplo, se les facilita código que les permite probar varios de los procedimientos implementados. El alumno solo tiene que ampliar este código para probar el resto. Sin embargo, pese a las indicaciones dadas, en ocasiones el alumno decide ignorar este paso y comprobar directamente la corrección de sus procedimientos traducidos integrándolos en el resto del proyecto proporcionado, a menudo para descubrir que resulta mucho más complejo depurar los errores en un programa de mayor tamaño. De una u otra forma, se trata de que el alumno alcance un objetivo de aprendizaje transversal como es la necesi-

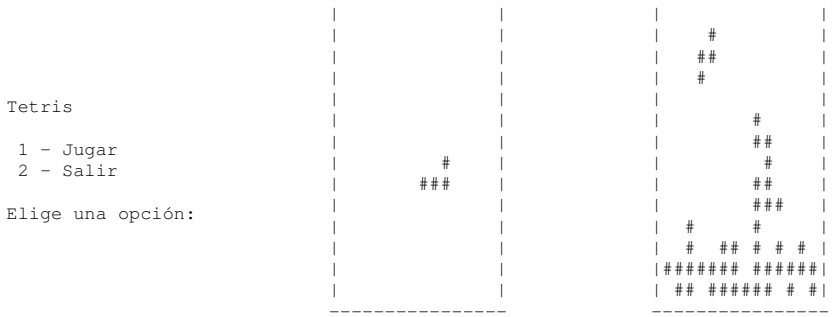

Figura 2: Ejemplos del aspecto de la salida del programa en tres momentos diferentes del juego.

dad de probar exhaustivamente cada nuevo módulo de código antes de ser integrado en un proyecto de software mayor.

En la segunda fase, cuando el alumno ya está familiarizado con el lenguaje ensamblador y con el programa Tetris, se le pide que realice varias mejoras al mismo para añadirle funcionalidades partiendo de su especificación. En concreto, el alumno debe, de forma autónoma, modificar los procedimientos existentes en el programa y desarrollar otros nuevos para implementar funcionalidades más avanzadas que no están incluidas en la versión en C, así como mejoras opcionales. Los ejercicios incluidos son: añadir un marcador, detectar que se ha completado una línea, eliminar las líneas completas, controlar el final de la partida y mostrar la pieza siguiente. En esta fase se consigue darle un aspecto más vistoso al juego.

En la tercera fase, cuando el alumno ya tiene soltura con el ensamblador, se le pide que añada dos mejoras que, aunque no son necesarias para el juego, permiten demostrar sus conocimientos con este tipo de programación. En concreto, que incorpore la posibilidad de realizar una configuración del juego (cambiando por ejemplo el tamaño del campo o las teclas de movimiento), y que modifique el ritmo de caída de las piezas según la puntuación alcanzada durante el juego.

El código implementado en las fases dos y tres se prueba directamente jugando y viendo si el resultado obtenido es el deseado.

La Figura 2 muestra tres ejemplos de la ejecución del programa. Una vez elegida la opción de jugar, dibuja el campo, elige una pieza aleatoriamente y la muestra en la parte superior. Entonces entra en un bucle donde se realizan las siguientes acciones: 1) Mediante consulta de estado (*polling*), comprueba si se ha pulsado alguna tecla y actúa en consecuencia, por ejemplo rotando la pieza. 2) Usando la hora actual, comprueba si ha trascurrido el tiempo necesario, y en su caso, baja automáticamente la pieza actual. Cuando la pieza no pueda bajar más, genera una nueva pieza.

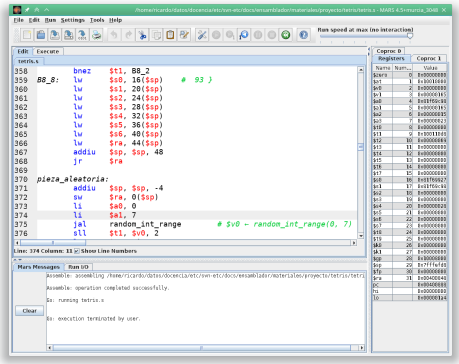

Figura 3: MARS editando un programa MIPS.

4. Extensiones a MARS

MARS [9] es un simulador de MIPS diseñado específicamente para la docencia, pensando en ser utilizado junto con el conocido libro de Patterson y Hennessy [7]. Ofrece un IDE que permite escribir y simular programas en ensamblador de MIPS sobre una máquina idealizada sin sistema operativo pero con un conjunto de llamadas al sistema emuladas que permiten realizar entrada/salida y otras funciones simples. Estas llamadas incluyen las ofrecidas por SPIM, otro simulador de MIPS más antiguo muy utilizado en docencia y citado en [7] (y que era la herramienta que utilizábamos previamente). MARS fue creado por profesores de la universidad de Missouri State y Otterbein College como una alternativa más moderna y portable a SPIM.

MARS incluye documentación de las instrucciones MIPS y de las llamadas al sistema ofrecidas integrada en el editor. Permite controlar la velocidad de simulación o ejecutar el programa paso a paso, pudiendo observar y modificar en todo momento el estado de la memoria y los registros del procesador. En la Figura 3 se muestra el editor de MARS y en la Figura 4 la ejecución de un programa MIPS usado en las prácticas de la asignatura en la que se enmarca el proyecto Tetris.

Se trata de un programa escrito en Java, por lo que los alumnos lo pueden ejecutar en casi cualquier sis-

tema operativo[1]. Se distribuye bajo licencia GPLv2, lo que ha permitido, aparte de que lo puedan instalar los alumnos libremente en sus ordenadores, que lo hayamos podido mejorar y adaptar a las necesidades específicas de nuestras prácticas y del proyecto.

Se han realizado varios cambios al programa MARS a lo largo del tiempo. Algunos de estos cambios son mejoras genéricas del programa y otros han sido motivados por las características de los programas que deseábamos utilizar para nuestras prácticas.

En primer lugar, partiendo de la versión 3.7 de MARS, se mejoró el interfaz de usuario en lo que respecta a las llamadas al sistema relacionadas con la entrada/salida, de forma que fuera posible emular la interacción a través de una terminal (integrada en el interfaz gráfico de MARS), ya que inicialmente MARS solo permitía la entrada de datos a través de cuadros de diálogo. Estos cambios se integraron en la versión 4.0 principal de MARS.

Posteriormente, se mejoró la simulación de la entrada por teclado, permitiendo la lectura de las teclas pulsadas en la terminal mediante el uso de MMIO e interrupciones. Esto permite escribir en el simulador una función para ver de forma no bloqueante si el usuario ha pulsado una tecla, lo que hace posible implementar pequeños videojuegos tipo arcade. También se añadió una llamada al sistema para limpiar la pantalla.

Una mejora de especial interés que hemos añadido a MARS es un depurador de convenios de programación que detecta el uso incorrecto de los registros y genera avisos en tiempo de ejecución que permiten identificar estos problemas fácilmente en el código del programa. Esta funcionalidad permite que los alumnos se puedan dar cuenta rápidamente cuando cometen este tipo de fallos, facilitándoles mucho el aprendizaje de los convenios de programación, que es uno de los aspectos de la programación en ensamblador que causa problemas a muchos alumnos. Además, la herramienta también simplifica enormemente la evaluación de los programas de los alumnos. Esta herramienta funciona analizando la ejecución de cada instrucción, detectando las llamadas a procedimiento y llevando la cuenta de qué registros se guardan en la pila y qué registros se modifican. Es capaz de detectar las siguientes violaciones de los convenios de programación: escritura en registros preservados no guardados, lectura de registros no preservados y no inicializados, preservación incorrecta de cualquier registro preservado (incluyendo el puntero de pila) y el exceso de instrucciones de retorno. Cuando se detecta un error, se muestra su descripción y la línea donde se ha producido, además de la pila de llamadas realizadas hasta el momento para facilitar la depuración.

Adicionalmente, se han realizado algunos cambios cosméticos y de menor importancia, como cambiar las opciones por defecto del programa para facilitar a los alumnos la realización de nuestras prácticas y corregir pequeños defectos del programa. Todos los cambios realizados han sido comunicados a los autores originales de MARS para su inclusión, si así lo consideran oportuno, en futuras versiones del programa. Sin embargo, la última versión de MARS, la 4.5, fue publicada en agosto de 2014, por lo que los cambios no han sido incorporados a día de hoy.

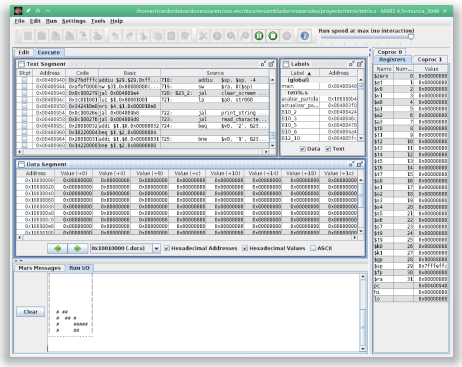

Figura 4: MARS ejecutando el proyecto Tetris.

Además de las mejoras introducidas sobre la herramienta MARS, con el fin de facilitar el prototipado de prácticas y ejercicios de ensamblador, se ha desarrollado también una herramienta que permite la compilación de programas en C sencillos a la versión de ensamblador MIPS ejecutable por MARS. Se trata de un programa que procesa programas en ensamblador de MIPS32 generados utilizando el compilador Clang y realiza algunas transformaciones sobre ellos, entre las que se incluyen deshacer el uso de huecos de retardo para los saltos, eliminar secciones y directivas innecesarias (principalmente relacionadas con la depuración), simplificar el uso de otras directivas, eliminar etiquetas no usadas, reordenar las secciones, reescribir el acceso a variables globales y añadir una pequeña biblioteca de llamadas al sistema. Algunas de las transformaciones son necesarias para que el código resultante sea entendido por MARS y otras mejoran la legibilidad del programa. La versión inicial del programa `tetris.s` se generó usando esta herramienta a partir de `tetris.c`. Una vez que la estructura del programa en C estaba decidida y probada, y después de modificar manualmente el código ensamblador generado para hacerlo más legible y de eliminar aquellos procedimientos que tienen que realizar los alumnos durante la práctica, el resultado es el fichero `tetris-esqueleto.s` que se les entrega junto con `tetris.c`. Obsérvese que no es necesario limpiar el código de los procedimientos que no aparecen

[1]Las prácticas en los laboratorios se realizan bajo Linux.

en la versión de los alumnos.

5. Resultados de evaluación

La metodología de organización y evaluación de la parte práctica explicada en la Sección 2 se lleva aplicando durante los últimos tres cursos académicos (2015–2016, 2016–2017, 2017–2018). A lo largo de este tiempo, el equipo docente de la asignatura ha evaluado a 1117 alumnos matriculados, con lo que se cuenta con una muestra bastante importante de cara a poder extraer conclusiones significativas. Puede llamar la atención el bajo porcentaje de alumnos que presentan el proyecto Tetris (58.73 %). Esto es debido fundamentalmente a que la asignatura en la que se enmarca el proyecto suele ser la que el alumno percibe como la más difícil de primer curso, y a que la fecha de entrega del mismo sea al final del cuatrimestre, antes del comienzo de los exámenes, una vez algunos alumnos han decidido ya abandonar la asignatura.

Con respecto a los resultados más importantes observados durante este tiempo, destacar el hecho de que tan solo 30 alumnos (2.7 %) se han presentado al examen de prácticas sin haber entregado el proyecto Tetris, con lo que se pone de manifiesto que el estudiante percibe el proyecto Tetris como un medio adecuado para preparar el examen de prácticas. Adicionalmente, la Tabla 1 muestra algunos datos más sobre estos tres años de evaluación de las prácticas. Aunque el 87.7 % de aprobados en la entrega del proyecto sobre presentados podría estar dentro de lo habitual en cualquier proyecto de programación, cabe destacar el alto número de aprobados en el examen de prácticas con respecto a los presentados (80.7 %). Este número no es usual en exámenes de programación, lo que demuestra que el proyecto de programación Tetris sirve en buena medida para la preparación del examen. Por último, el número de alumnos que entregan el proyecto Tetris y hacen el examen de prácticas es algo superior a la mitad de los matriculados. De estos, el 82.5 % superan las prácticas.

Además, la Figura 5 muestra la forma en la que se relacionan las notas obtenidas en el proyecto Tetris y las notas del examen de prácticas para los alumnos que realizan ambas actividades. En concreto, en el eje de abscisas se muestra las notas obtenidas en el proyecto Tetris agrupadas en 4 rangos númericos que se corresponderían con las categorías Suspenso, Aprobado, Notable y Sobresaliente. De igual forma, cada una de las barras aparece dividida en los mismos 4 rangos numéricos. En el eje de ordenadas se muestra el número de alumnos.

En primer lugar, se puede apreciar en la figura que existe una gran cantidad de alumnos con nota superior o igual a 9 en el proyecto Tetris (36.1 % sobre el total),

Número de matriculados	1117	
Proyecto Tetris		
Número de presentados	656	58.7 %
Número de aprobados	575	87.7 %
Nota media del proyecto	7.56	
Examen práctico		
Número de presentados	623	55.8 %
Número de aprobados	503	80.7 %
Nota media del examen	6.81	
Proyecto Tetris & Examen práctico		
Número de presentados	578	51.7 %
Número de aprobados	477	82.5 %
Nota media del examen	6.82	

Cuadro 1: Resumen de estadísticas.

lo que demuestra el interés de gran parte del alumnado en completar el proyecto y que lo consideran entretenido (esto también lo manifiestan en las encuestas de opinión que desde la Facultad se les pasa al final del cuatrimestre).

En segundo lugar, podemos ver cómo el porcentaje de alumnos que no supera el examen de programación se va reduciendo conforme aumenta la nota obtenida en el proyecto de Tetris. Así, el 40 % de los alumnos que no aprueban el proyecto Tetris (nota mayor o igual que cero y menor que 5) tampoco logran superar el examen de programación. Este porcentaje cae al 22 %, 16 % y 7 % para los alumnos que obtienen en el proyecto Tetris una calificación de Aprobado, Notable o Sobresaliente, respectivamente. Es importante recordar que el proyecto se realiza y califica en grupos de 2 alumnos y no siempre los dos componentes del grupo realizan la misma cantidad de trabajo, lo que justifica que existan alumnos que consigan superar el proyecto pero no aprueben el examen de prácticas.

Es interesante notar también que el 60 % de los alumnos que no consiguen superar el proyecto Tetris sí aprueban sin embargo el examen de prácticas (incluso un 14 % consiguen una calificación mayor o igual a 9 en el examen). En la mayor parte de los casos se trata de alumnos que por circunstancias diversas no han logrado alcanzar el 5 en el proyecto Tetris (generalmente han obtenido una nota superior al 4), pero a los que su realización ha ayudado a dominar la programación en lenguaje ensamblador. El porcentaje de alumnos que logran una calificación de Sobresaliente en el examen de prácticas también crece con la nota obtenida en el proyecto Tetris, llegando hasta el 48 % para aquellos que obtienen una calificación entre 9 y 10 en el proyecto Tetris.

Por último y de forma global, el 85.6 % de los alumos que superan el proyecto Tetris consiguen una calificación igual o superior al Aprobado en el examen final de prácticas. A la vista de los resultados, creemos

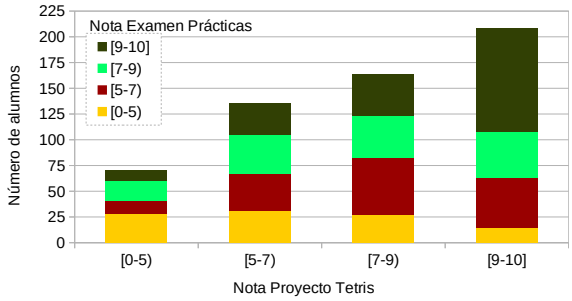

Figura 5: Relación entre las notas del proyecto Tetris y las notas del examen práctico.

que, tal y como esperábamos, el proyecto Tetris resulta un vehículo adecuado para asegurar el aprendizaje del lenguaje ensamblador por parte del estudiante.

6. Conclusiones y trabajo futuro

En este trabajo hemos presentado nuestra experiencia con la enseñanza del lenguaje ensamblador MIPS en el contexto de una asignatura del segundo cuatrimestre del primer curso del Grado de Ingeniería en Informática. Nuestra aproximación trata de motivar al estudiante a través del desarrollo del videojuego Tetris, presentando una organización del trabajo por fases con distinto grado de complejidad. La primera fase consiste en la traducción de código C a código ensamblador, y sirve para asegurar un aterrizaje suave del alumno al mundo del lenguaje ensamblador. La segunda fase implica el desarrollo de código ensamblador directamente, para incorporar ciertas características sencillas al videojuego. La última fase aborda la realización de funcionalidad más avanzada.

Para la realización del proyecto se ha utilizado la herramienta MARS, un simulador de MIPS escrito en Java diseñado específicamente para la docencia, y que previamente hemos tenido que ampliar y mejorar para que se adaptase a las necesidades del proyecto.

Los resultados obtenidos a lo largo de los 3 últimos cursos académicos ponen de manifiesto que el alumno percibe el proyecto de programación como un vehículo adecuado y ameno para el aprendizaje del lenguaje ensamblador. Además, el hecho de que el 85.6 % de los alumnos que superan el proyecto Tetris consiguen aprobar también el examen final de prácticas que deben realizar, nos sirve para concluir que, efectivamente, a través del proyecto Tetris los alumnos consiguen aprender a desarrollar programas en ensamblador de MIPS de una forma más sencilla y efectiva de lo que venían haciéndolo con anterioridad.

Todo el material que empleamos en el proyecto Tetris está disponible en el siguiente enlace: `http://ditec.um.es/proyecto-tetris`

Como línea de trabajo futuro, queremos adaptar los contenidos de toda la asignatura a RISC-V, un nuevo ISA de código abierto y libre de royalties que comenzó en 2010 en la Universidad de California, Berkeley, y que está recibiendo gran atención mundial por parte de la comunidad de Arquitectura de Computadores.

Referencias

[1] S. Barrachina, G. Fabregat, J.C. Fernández, y G. León. ARMSim y QtARMSim: Simulador de ARM para Docencia. En *XXI Jornadas de Enseñanza Universitaria de la Informática (JENUI 2015)*, Julio 2015.

[2] P.A. Castillo, A. Cañas, A. Prieto, y J.J. Castillo. Simulador de un Robot Autónomo para las Prácticas de Ensamblador de la Asignatura de Estructura de Computadores. En *X Jornadas de Enseñanza Universitaria de la Informática (JENUI 2004)*, Julio 2004.

[3] P.A. Castillo, M. García, M.G. Arenas, G. Romero, y A. Prieto. Aritmética en Coma Flotante y Programación en Ensamblador en las Prácticas de Estructura de Computadores: Control de un Simulador de Radar. En *XI Jornadas de Enseñanza Universitaria de la Informática (JENUI 2005)*, Julio 2005.

[4] N.C. Cruz, J.L. Redondo, J.D. Álvarez, y P.M. Ortigosa. Programación de un Juego en Ensamblador CHIP-8 como Actividad Complementaria en la Asignatura Arquitectura de Computadores. En *XXIX Jornadas de Paralelismo (JP 2018)*, Septiembre 2018.

[5] F.J. Gallego-Durán, R. Satorre-Cuerda, P. Compañ-Rosique, y C. Villagrá-Arnedo. Explicando el Bajo Nivel de Programación de los Estudiantes. En *XXIII Jornadas de Enseñanza Universitaria de la Informática (JENUI 2017)*, Julio 2017.

[6] J. Kawash y R. Collier. Using Video Game Development to Engage Undergraduate Students of Assembly Language Programming. En *14th Annual ACM Conference on Information Technology Education (SIGITE 2013)*, Octubre 2013.

[7] D.A. Patterson y J.L. Hennessy. *Computer Organization and Design. The Hardware/Software Interface*. Morgan Kaufmann, 5th edition, 2014.

[8] A. Pázhitnov. Tetris. https://es.wikipedia.org/wiki/Tetris, 1984. Último acceso: 29-01-2019.

[9] K. Vollmar y P. Sanderson. MARS: An Education-Oriented MIPS Assembly Language Simulator. En *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, pag. 239–243, New York, NY, USA, 2006. ACM.