.

ENSAMBLADOR Y SIMULADOR DE ARQUITECTURA ARMv4

Daniel Gerardo Canessa Valverde
danielcanessa09@gmail.com
Felipe Alberto Mejías Loría
fmejias19@gmail.com

ABSTRACT: *A hardware simulator is a piece of software that emulates specific hardware devices and assemblers are computer programs that translate one assembly language into machine code. This paper describes an assembler that translates ARM code to machine code and a Java-based simulator for the ARM processor. The simulator was implemented with a graphical interface that allows writing assembly code and its corresponding implementation.*

PALABRAS CLAVES:
Arquitectura, ARM, Ensamblador, Simulador

1 INTRODUCCIÓN

El estudio de la arquitectura de computadores es un campo difícil debido a la alta complejidad involucrada en cualquier sistema informático. Para manejar esta complejidad, diferentes herramientas han sido desarrolladas permitiendo a las arquitecturas ser simuladas y modificadas, permitiendo ver la ejecución de programas en ensamblador en la arquitectura. Entre las herramientas desarrolladas destacan los ensambladores y los simuladores. Los simuladores son una parte integral de muchos sistemas de computación en la actualidad. Esta tecnología es muy útil, ya que hace posible que los usuarios prueben y ejecuten el software antes de que el hardware actual esté disponible, o en ausencia del hardware real. Las principales aplicaciones de simuladores consisten en estudios de arquitectura de computadores y el proceso de compilación. Los simuladores de nivel intermedio, los cuales implementan un subconjunto del set de instrucciones intentan ilustrar y enseñar dos principios generales: la arquitectura del conjunto de instrucciones y la microarquitectura. La arquitectura ARM se estudia porque es una arquitectura RISC, aplicable directamente a tecnologías y tendencias de punta como lo es el Internet de las cosas y los sistemas embebidos en general. El simulador de lenguaje ensamblador para el procesador ARM que se implementa en el proyecto tiene una interfaz visual, y además implementa casi la totalidad del conjunto de instrucciones de ensamblador ARMv4. Como afirma Louden [1] los ensambladores son un tipo de compiladores, los cuales se encargan de transformar el código ensamblador en código máquina. Los compiladores son programas que se encargan de convertir un lenguaje de alto nivel a un lenguaje de bajo nivel que logre ser interpretado por la computadora. Los programas de computadora están escritos en lenguajes de programación, sin embargo, las computadoras interpretan secuencias de instrucciones particulares, pero no programas de texto. Por lo tanto, el texto del programa se debe traducir en una secuencia de instrucciones adecuadas antes de que pueda ser procesado por un computador, por lo que el compilador convierte el código fuente en código máquina. La comprensión de las relaciones entre los lenguajes de programación y las computadoras facilitan las transiciones a nuevos lenguajes de programación y de hardware. La implementación del simulador es enriquecedora desde el punto de vista del aprendizaje, ya que hacer uso de herramientas de simulación le ayuda al estudiante a entender los conceptos difíciles de comprender. Para la educación de la arquitectura de computadores, es importante para los estudiantes que necesitan un simulador que cubra los principios de los procesadores con más detalle. Es importante para personas que están aprendiendo el set de instrucciones de ARMv4, desarrollar este tipo de proyectos, ya que combinan una serie de elementos que hacen que la parte teórica sea sólida, incentivan el pensamiento de resolución de problemas diseñando, y además fortalecen los conocimientos y habilidades en el lenguaje de programación.

2 SISTEMA DESARROLLADO

2.1 Ensamblador

Un ensamblador es un traductor para el lenguaje ensamblador de una computadora en particular. El ensamblador del sistema es un compilador de instrucciones del procesador ARM que permite el ensamblado de la instrucción a hexadecimal. Este es el modo principal de la aplicación, por lo que cuando el código está correcto, el ensamblador realiza el ensamblado del programa y además se encarga de generar un archivo out.txt, que contenga la traducción a lenguaje máquina de las instrucciones escritas en ensamblador ARMv4. El ensamblador está escrito en Java 8 y aplica las reglas de análisis léxico, sintáctico y semántico utilizadas en los compiladores para la detección de errores en la escritura del código en ensamblador.

Louden [1] explica que el ensamblador se compone de varias fases que realizan distintas operaciones lógicas, y que estas fases interactúan de manera conjunta. Las fases son las siguientes: el análisis léxico, el análisis sintáctico, el análisis semántico y la generación de código. Para la implementación del ensamblador se implementaron las siguientes fases:

.

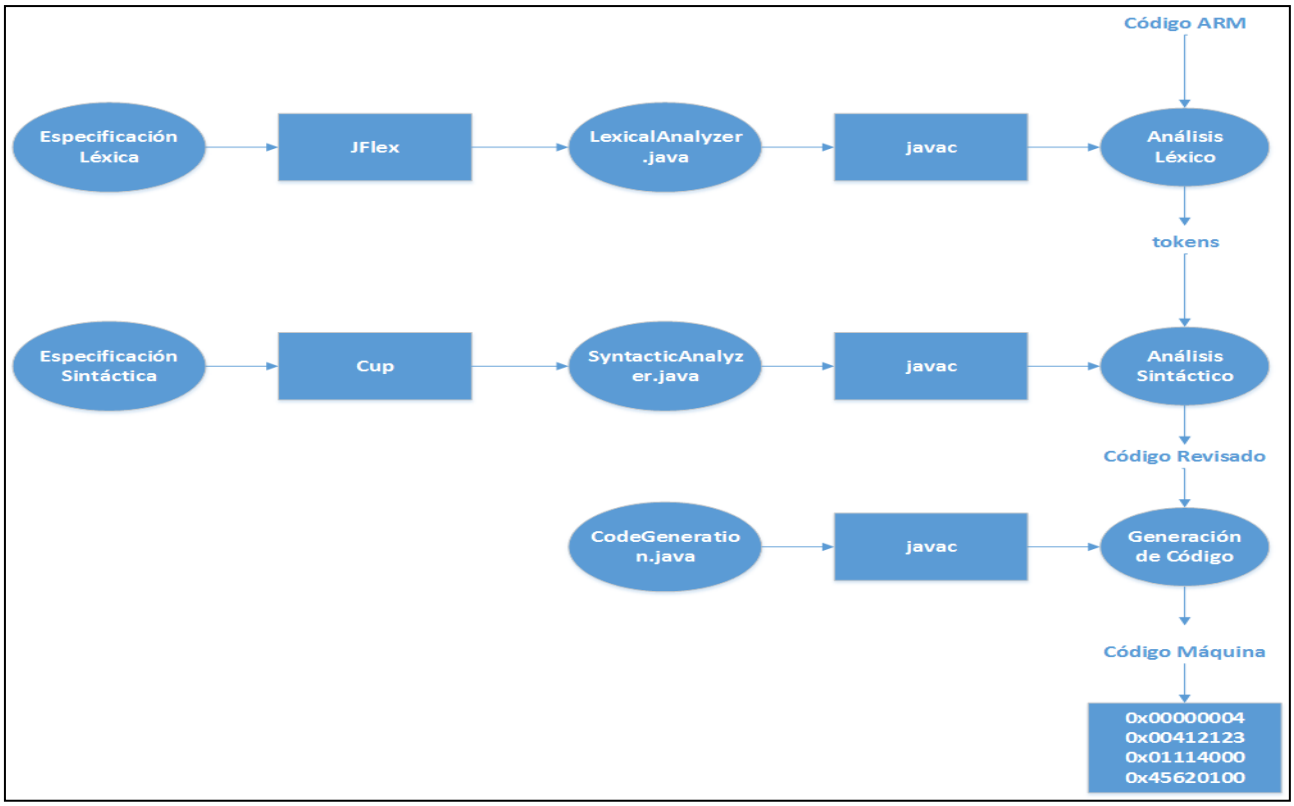

Figura 1. Diagrama de alto nivel del ensamblador

2.1.1 Análisis Léxico

En esta fase el ensamblador se encarga de efectuar la lectura del código fuente, la cual generalmente está en forma de un flujo de caracteres. El analizador se encarga de recolectar secuencias de caracteres que se definen en el archivo de especificación léxica llamados tokens. Como indican diversos ejemplos [3] para realizar el análisis léxico se utiliza un archivo de texto en lenguaje JLex que contiene la especificación léxica. Una vez que se define el archivo de especificación léxica, se ejecuta una vez para generar la clase en java del Analizador Léxico, tal y como se observa en el diagrama de la Figura 1.

2.1.2 Análisis Sintáctico

En esta fase el ensamblador se encarga de revisar el código fuente en forma de tokens proveniente del análisis léxico y realiza el análisis sintáctico, el cual determina la estructura del programa. El analizador se encarga de determinar los elementos estructurales del programa y sus relaciones. Como afirman diversos autores [3] para realizar el análisis sintáctico se utiliza un archivo de texto en lenguaje Cup que contiene la especificación sintáctica. Una vez que se define el archivo de especificación sintáctica, se ejecuta una vez para generar la clase en java del Analizador Sintáctico, tal y como se observa en el diagrama de la Figura 1.

2.1.3 Análisis Semántico

En esta fase el ensamblador se encarga de revisar el significado del programa, ya que se encarga de verificar los tipos y las declaraciones. Esta parte del análisis se realiza durante la generación de código, en caso de que ocurra algún error se notifica al usuario.

2.1.4 Generación de código

En esta fase el ensamblador se encarga de generar el código objetivo, en este caso genera el código hexadecimal correspondiente de cada una de las instrucciones. El ensamblador utiliza las propiedades de cada uno de los tipos de instrucciones de ARMv4 para realizar la adecuada generación del código. Para realizar la generación de código se utiliza una clase que decodifica cada uno de los tipos de instrucciones, además de que revisa si ocurren errores semánticos. Para generar el código se debe realizar el proceso que se muestra en la Figura 1. Primero debe pasar por el análisis léxico, el cual genera tokens que recibe el analizador sintáctico, cuando se llega a esta etapa el analizador revisa que el código cumpla con las reglas gramaticales y envía al generador de código máquina el código ARM. Una vez ahí se genera el código objetivo correspondiente y se escribe en el archivo de salida “out.txt” el código generado.

.

2.2 SIMULADOR

El simulador tiene como objetivo, emular el comportamiento del hardware cuando se ejecuta un código en lenguaje ensamblador ARMv4. Para realizar esto el simulador se ha desarrollado en el lenguaje de programación Java 8, apoyándose en el editor de código NetBeans 7, del cual se ha utilizado también su componente swing para el desarrollo de la interfaz gráfica. Para realizar la simulación, se ha desarrollado una aplicación que utiliza que emplea el paradigma de programación orientado a objetos, se han creado clases para representar virtualmente los componentes físicos que interactúan con el microprocesador durante la ejecución de un programa. Las clases principales son: instrucciones, memoria, banco de registro y banderas de la ALU, esto se puede ver en el siguiente diagrama.

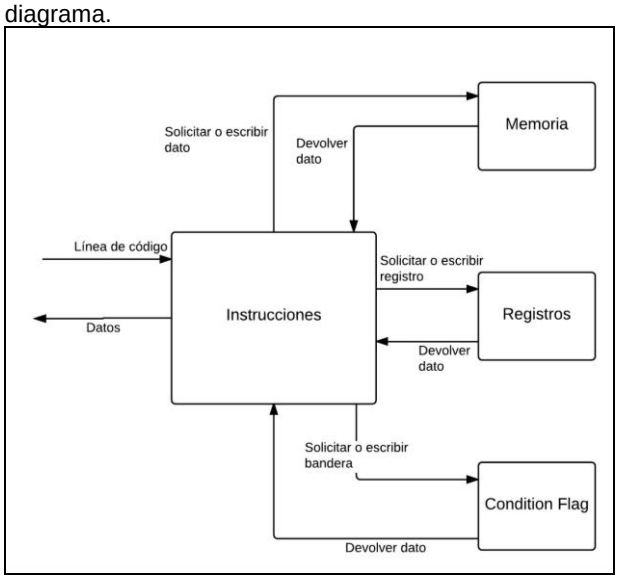

Figura 2. Diagrama de alto nivel del simulador.

2.2.1 Memoria

La memoria físicamente tiene como objetivo almacenar datos en forma de bits, en diferentes posiciones. La memoria virtual que se ha desarrollado tiene una longitud de 2KB y es representada por la estructura de datos, arreglo. De la posición 0x00 a la posición 0x3FF se destina para almacenar el programa, de la posición 0x400 a la 0x7FF se destina para almacenar datos. Cada slot de memoria puede almacenar 1Byte. Esta clase interactúa con la clase de instrucciones, la cual le puede solicitar un dato dada una posición de memoria o le puede solicitar que almacene un dato dada también una posición de memoria.

2.2.2 Banco de registros

Es común en las arquitecturas RISC contar con gran cantidad de registros de uso general. El banco de registros virtual implementado es representado por una clase que contiene 16 variables del tipo de dato “string”, con los nombres de r0 a r15. El tipo de dato es “string” por el motivo de que cada una de estas variables va a contener un dato en hexadecimal, en el lenguaje de programación java los datos de este tipo deben ser almacenados en variables de tipo “string”. Esta clase interactúa con la clase de instrucciones, la cual le puede solicitar el dato de un registro o le puede solicitar que almacene un dato en un registro dado.

2.2.3 Banderas ALU

Cuando se agrega una “s” al final del nemónico de una instrucción lógica o aritmética o esta instrucción es específicamente “cmp” o “cmn” el estado de las banderas de la ALU cambia. Las banderas de la ALU se encuentran contenidas en una clase. Son representadas por variables de tipo booleano y reciben los nombres de: “zero”, “carry”, “negative” y “overflow”. En el sistema desarrollado estas variables cambian solo cuando se da la ejecución de la instrucción “cmn” o “cmp”. La clase de instrucciones es la que interactúa con esta clase, esta clase puede solicitar que cambie el estado de alguna bandera específica o que se retorne el estado de una bandera específica.

2.2.4 Instrucciones

El objetivo de esta clase es ejecutar una instrucción de entrada y devolver los datos que se deben desplegar en la interfaz gráfica, específicamente el estado de la memoria, banco de registros y banderas de la ALU, emulando el comportamiento del hardware en caso de que hubiese ejecutado el código cargado. En esta clase primero se verifica que el archivo “out.txt” se haya generado satisfactoriamente, con esto se descartan posibles errores léxicos o sintácticos en las instrucciones a ejecutar. Luego se procede a decodificar la instrucción y a ejecutarla según corresponde, las clases de la memoria, el banco de registros y banderas de la ALU se pueden ver involucradas en la ejecución de la instrucción.

3 RESULTADOS

Los resultados que se obtuvieron en el modo de ensamblado fueron los siguientes:

```
0x0     0xe3a02e42
0x4     0xe3a0600f
0x8     0xe5c26000
0xc     0xe3a030f0
0x10    0xe5c23001
0x14    0xe3a070ff
0x18    0xe5c27002
0x1c    0xe5d27002
0x20    0xe35700ff
0x24    0xaa000000
0x28    0xeafffffb
0x2c    0xe5d23000
0x30    0xe3a09004
```

Figura 3. Archivo con el código ensamblado

.

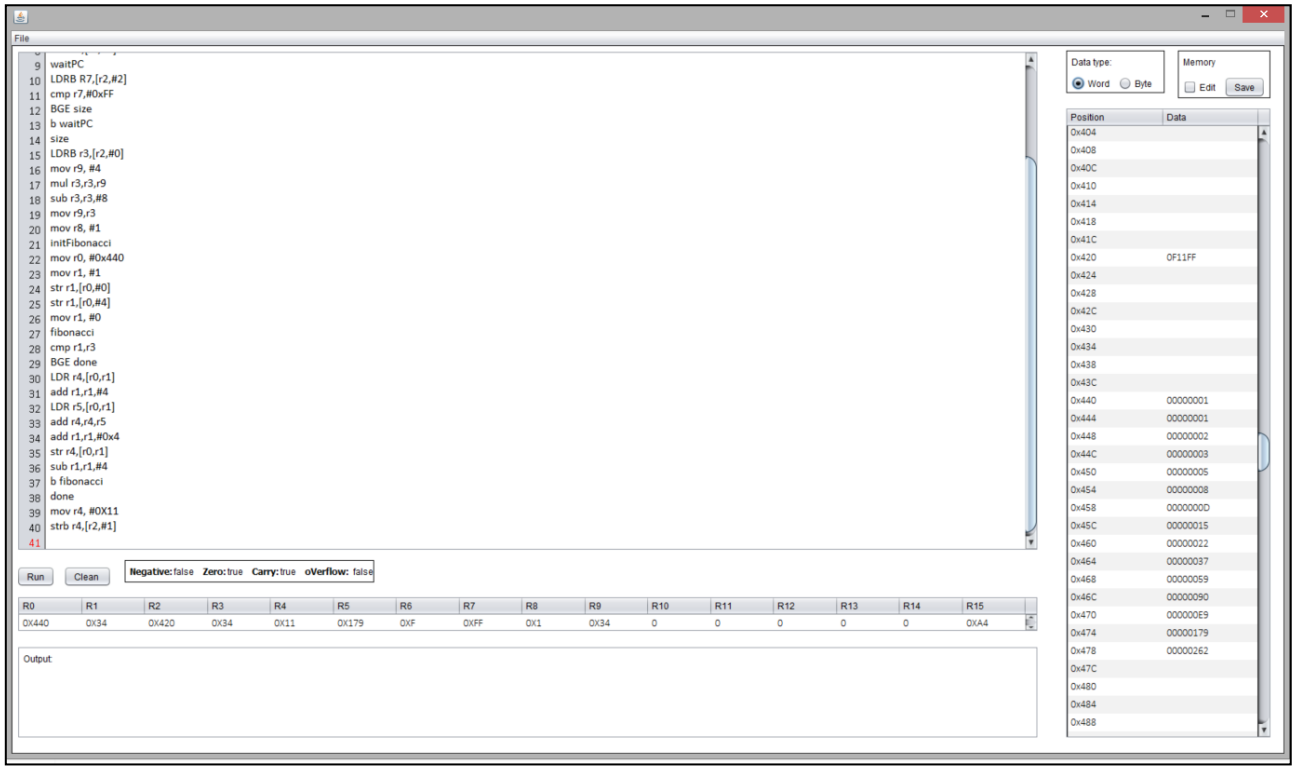

Figura 4. Simulador de ARM.

Los resultados que se obtuvieron en el modo de simulador son los que se observan en la Figura 4.

4 CONCLUSIONES

- El lenguaje ensamblador es una representación simbólica del lenguaje máquina.
- El ensamblador es un tipo de compilador que solo traduce lenguaje ensamblador.
- A diferencia de un compilador, el ensamblador se encarga de generar código máquina.
- Java ofrece gran cantidad de bibliotecas libres para realizar análisis de compilación.
- Programar un simulador de código ARM, obliga un entendimiento total de la arquitectura y como trabaja esta con el hardware.
- Una clara separación de clases independientes en el paradigma de programación orientado a objetos, permite un rápido entendimiento del código y una fácil modificación.
- El componente swing de NetBeans permite un desarrollo intuitivo y rápido de una interfaz gráfica.

5 REFERENCIAS

[1] K. Louden, Compiler construction. Boston: PWS Pub. Co., 1997.

[2] S. Harris and D. Harris, *Digital design and computer architecture: Arm Edition*.

[3] S. Gálvez and M. Mora, Java a tope: Compiladores. Traductores y compiladores con LEX/YACC, JFLEX/CUP y JAVACC. Málaga: Universidad de Málaga, 2005.