Proyecto Grupal #1
Instituto Tecnológico de Costa Rica
Arquitectura de Computadores I - CE 4301
Prof. Luis Alberto Chavarría Zamora

Estudiante: Gabriel Abarca Aguilar, Esteban Alvarado Vargas, Olman Castro Hernández

Diseño e Implementación de un ASIP para la generación de gráficos y texto

Documento de Diseño

1. Descripción

Para aplicar los conceptos de arquitectura de computadores se plantea el diseño e implementación en hardware de un *Application Specific Instruction Set Processor* (ASIP) para la generación de gráficos mediante un editor de texto.

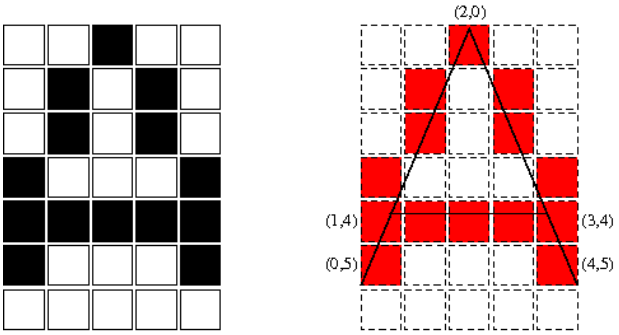


Figura 0. *Gráfico generado para la letra "A" en un bitmap (izquierda) y su vectorización (izquierda)*

Los gráficos por computadora son un área de la informática visual. A pesar de que existen procesadores especializados para realizar su generación (GPU) para efectos de este proyecto se pretende crear un ASIP que permita la generación de gráficos 2D por computador para crear una imagen a partir de estructuras geométricas simples como líneas, círculos, diagonales, entre otras. El sistema aplicará los algoritmos de Bresenham para la generación de estas figuras básicas.

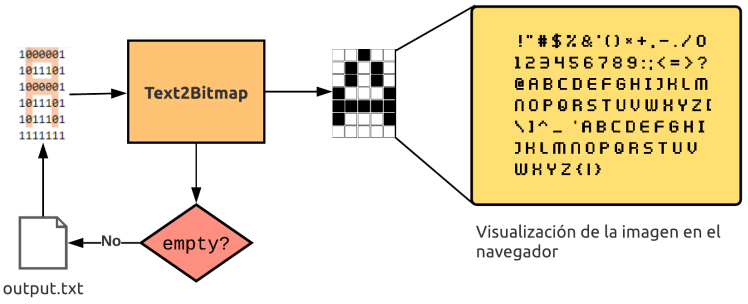


Figura 1. *Diagrama general de la ejecución del programa.*

2. Estado del Arte

Dentro del mundo de la arquitectura de procesadores existen distintos sets de instrucciones que proporcionan diferentes herramientas a los usuarios de estos sistemas. Cada una posee funcionalidades específicas, así como algunas debilidades; en el proyecto expuesto en este documento, se ha creado el propio ISA (Instruction set architecture) con el fin de desarrollar las instrucciones que más se adecuan a la resolución del problema, sin embargo existen ya arquitecturas definidas en el mercado, dentro de las tres más populares podemos encontrar ARM, RISC-V y x86.

Las ISA's mencionadas con anterioridad, fueron estudiadas con el fin de crear el set de instrucciones que más se adaptara a las necesidades del problema planteado y se explican a continuación:

RISC-V

El ISA RISC-V fue desarrollado originalmente en la división de informática en el departamento de EECS en Berkeley, Universidad de California. Esta arquitectura se encuentra basada en un tipo RISC; su principal objetivo es convertirse en un ISA universal, al ser de código abierto, otra característica que posee este ISA es la modularidad y su núcleo principal es RV32I [1].

Su set de instrucciones está compuesto por instrucciones tipo [1]:

- **Tipo R:** para operaciones entre registros.
- **Tipo I:** para inmediatos cortos y loads.
- **Tipo S:** para stores.
- **Tipo B:** para branches.
- **Tipo U:** para inmediatos largos.
- **Tipo J:** para saltos incondicionales.

La estructura de cada una de estas instrucciones es posible visualizarse en la Figura 2.

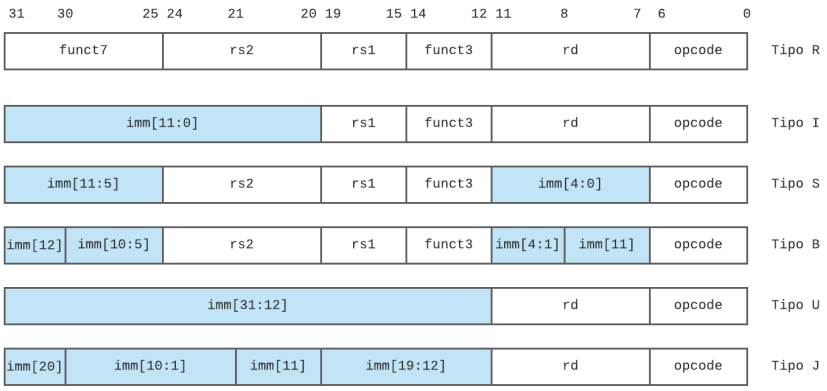


Figura 2. *Set de instrucciones RISC-V*

ARM

ARM es una arquitectura basada en RISC, creada por ARM Holdings. Esta arquitectura posee instrucciones de 32 y 64 bits, desde el 2009 el 90% de todos los procesadores RISC de 32 bits integrados en el mercado son ARM y se utilizan principalmente en dispositivos móviles y consolas de videojuegos portátiles, estos se convirtieron en unos de los procesadores más utilizados del mundo. [2]

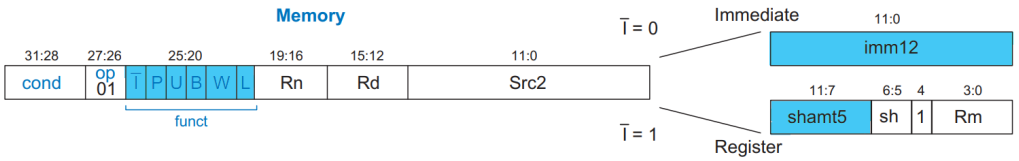


Figura 3. *Set de instrucciones ARM, tipo memoria* [3]

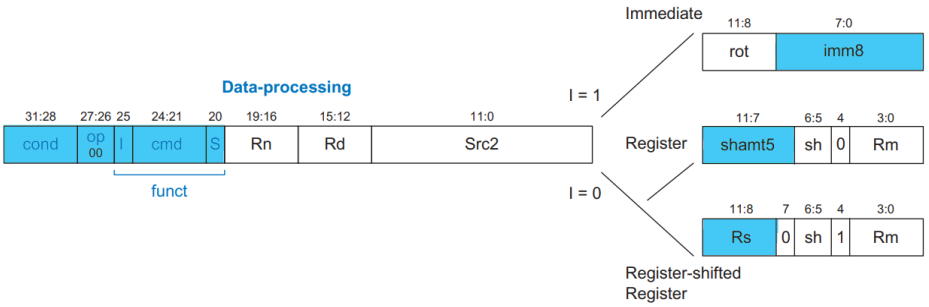


Figura 4. *Set de instrucciones ARM, tipo datos* [3]

Las Figuras 3 y 4, muestran los distintos set de instrucciones ARM, dependiendo el tipo de instrucción ya sea para acceso a memoria “D” o para manejo de datos “R e I”.

x86

x86 es una de las ISA’s que posee una de las evoluciones a través de la historia con más trayectoria, su diseño inicia a principio de los años 1976 y es considerada una de las arquitecturas CISC más robustas del mercado, brindando altas especialidades en distintos ámbitos, otorgando cientos de instrucciones para la resolución de conflictos, además es la arquitectura utilizada por la mayor cantidad de computadores del mundo [4].

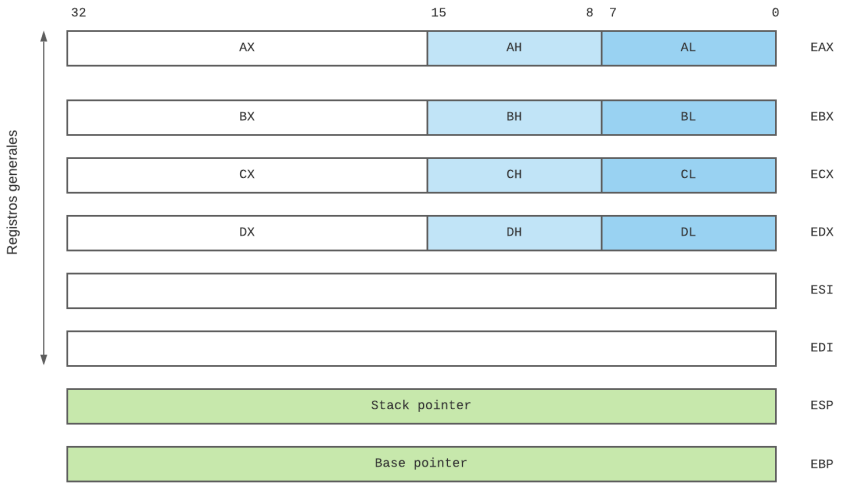


Figura 5. *Set de instrucciones x86*

La Figura 5, muestra la estructura de los registros de las instrucciones para un ISA x86 y como estas puede estar dividida en instrucciones de 8, 16 y 64 bits, según las necesidades del problema.

Los procesadores pipeline son una poderosa forma de aumentar el rendimiento de los sistemas digitales, dividiendo el proceso de un procesador uniciclo en cinco etapas,

proporcionando cambios nulos en la latencia de cada instrucción, pero aumentando el rendimiento de cinco veces [3].
El proceso manejado en estas cinco etapas corresponde a leer una instrucción de la memoria de instrucciones, seguidamente se lee el archivo de registro y se decodifica una instrucción, luego se procesa en la ALU (Arithmetic logic unit), se lee o guarda en memoría para finalmente escribir el resultado en el archivo de registros [3]. La Figura 6, muestra el comportamiento de un procesador uniciclo ante uno pipeline.

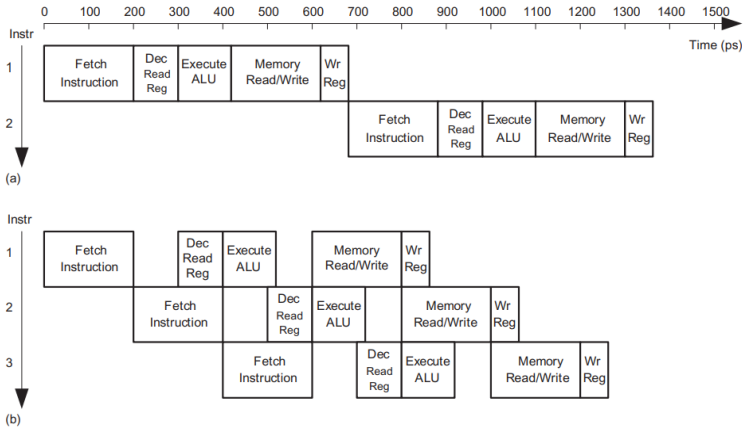
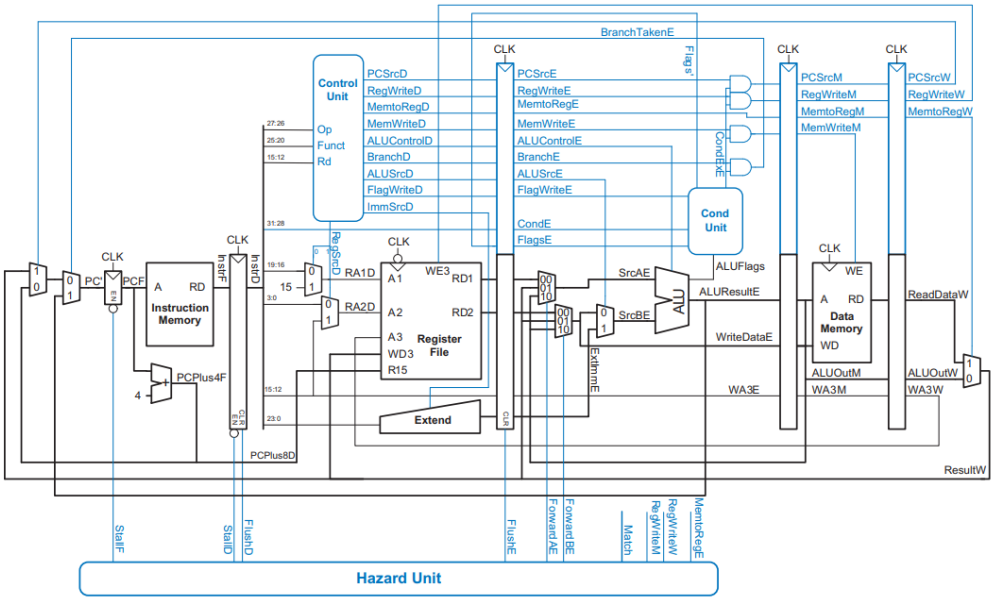


Figura 6. *Diagrama de procesos: a. Procesador uniciclo. b. Procesador Pipeline* [3]

En el caso del diseño de la microarquitectura, el proyecto ha sido basado en la microarquitectura de un procesador pipeline ARM. Ver Figura 7. Una vez puesta en marcha esta arquitectura, se han hecho las modificaciones necesarias para que exista un acople al ISA y esta sea eficiente; en la sección 4 este tema se trabaja a una mayor profundidad.Figura



7. *Microarquitectura de un procesador pipeline ARM* [3]

En el caso de los procesadores pipeline, estos poseen una unidad de detección de riesgos tal como muestra la Figura 7, esta es encargada de controlar el ciclo de procesos, adelantar o detener instrucciones en caso de que pueda existir algún tipo de conflicto entre estas [3].

3. Requerimientos del Sistema

Los *requerimientos* del sistema se presentan en la siguiente tabla:

Tabla 1. Listado de Requerimientos del Sistema

Requerimiento	Funcionalidad	Descripción
ISA	Set de Instrucciones	Se debe diseñar un conjunto de instrucciones y arquitectura que permita implementar la solución al problema.
	Características del set	El set debe contar con: ● Instrucciones para registros, inmediatos, memoria y condicionales. ● Datos inmediatos de 16 bits. ● Banco de registros con 16 registros de 32 bits. ● Manejo de decisiones condicionales mediante flags. ● Las instrucciones deben ser de 32 bits.
Microarquitectura	Memoria	El sistema debe tener capacidad de segmentación de memoria en datos e instrucciones. ● Se requiere de una memoria de instrucciones y datos que reciba la dirección como entrada y a su salida presente el contenido almacenado. ● El valor de la dirección se le realiza un corrimiento de bits para ajustarla al alineamiento de las memorias. ● La memoria de instrucciones debe ser de un tamaño mayor o igual a 1k. ● La memoria de datos debe tener la capacidad de almacenar la imagen resultante de 250x250 px, por lo que debe ser de 64k como mínimo.
	Dispositivos I/O	El sistema debe ser capaz de acceder a los dispositivos de entrada y salida. La ejecución del sistema debe comenzar cuando se active una señal de entrada. La salida del sistema se debe reflejar en pines GPIO que escribirán el contenido en un archivo de texto.
	Unidad de Control	Debe implementarse una unidad que recibe en su entrada la instrucción y toma el tipo de operación, la instrucción que va a realizar, las flags condicionales y otros bits como el inmediato y el load para determinar las señales de control para cada etapa del

		procesador. Sus salidas controlan los mux, la escritura en registro y memoria, junto con la operación de la ALU.
	Camino de datos	El procesador debe implementar un pipeline, por lo que requiere de un módulo que permita el paso de los datos a través de las distintas etapas del procesador: Fetch. Decode, Execution, Memory y Writeback.
	Unidad de Riesgos	Se debe implementar una unidad de detección y control de riesgos que sea capaz de detectar y realizar adelantamientos, colocar stalls y borrar instrucciones incorrectas cuando un branch es tomado, mediante flushes a las primeras etapas del procesador.
	Pruebas unitarias	Cada unidad funcional del sistema debe ser debidamente probada para verificar su correcto funcionamiento.
Compilador	Generación de código	Permite traducir las instrucciones del ISA a binario, con la finalidad de ejecutarlo en el procesador.
Programa de Generación de Gráficos	Identificación de las letras	El algoritmo debe tomar un dato de memoria e identificar si corresponde a una de las letras permitidas y ejecutar su algoritmo graficador, en caso contrario dibuja un espacio en blanco.
	Generación de texto mediante gráficos	Para cada letra se deben implementar los comandos para dibujar la letra, utilizando los métodos de Bresenham implementados para dibujar las líneas básicas.
	Algoritmo de Bresenham	Se deben implementar los métodos del algoritmo para dibujar líneas y círculos.
Render de imagen de salida	Lectura del archivo generado	El sistema deberá leer el archivo generado a la salida del procesador que contiene la imagen binarizada que se ha generado.
	Generación de la imagen	El sistema deberá interpretar el contenido y generar un archivo de imagen, donde los 0 son color negro y los 1 son blanco. La imagen debe tener una resolución de 250x250 px.
	Presentación de la imagen generada	El sistema debe presentar al usuario la imagen generada en una pantalla.

4. Diseño de la solución

Para el diseño de la solución del problema planteado se han discutido varias propuestas, tomando en cuenta aspectos del set de instrucciones, la microarquitectura y los programas de software. Se presentan a continuación las discusiones para cada etapa. Las propuestas

ISA

La **primera propuesta** de arquitectura contemplaba contener tres tipos de instrucciones: `registros, inmediatos y condicionales`. Las instrucciones de *registros* e *inmediatos* se utilizarían para las instrucciones lógicas, aritméticas y de memoria. En esta propuesta se busca tener una cantidad de instrucciones lógicas y aritméticas más amplia, con división, módulo, xor, not y complemento a dos. Una característica planteada desde el inicio fue implementar una instrucción de `retorno`.

En una revisión posterior se desarrolló una **segunda propuesta** que resultaba mejor para el problema planteado. Las instrucciones se dividirán en cuatro tipos: `registros, inmediatos, memoria y condicionales.` Las instrucciones aritméticas se redujeron a *suma, resta* y *multiplicación*, y las lógicas solamente *and* y *or*. Se agregó la instrucción `MOV` al set de instrucciones pues se descubrió que sería de uso recurrente en el programa. Finalmente se *eliminó* la instrucción de `retorno` de la arquitectura pues agrega complejidad innecesaria a la microarquitectura y el programa se podía realizar sin esta instrucción.[1]

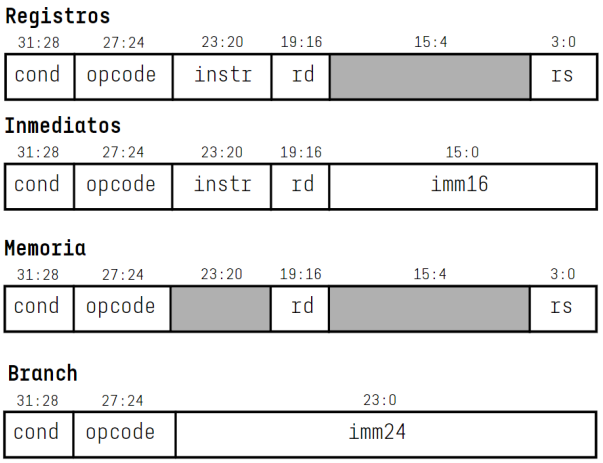


Figura 8. *Formato de instrucciones de la arquitectura diseñada seleccionada*

Microarquitectura

La microarquitectura es uno de los elementos que ya poseían un fundamento teórico de cómo debía realizarse. A pesar de esto, como equipo de desarrollo debimos tomar algunas decisiones de diseño.

En una primera instancia se diseñó una solución que aplica la *arquitectura de Harvard modificada*, es decir, que comparte una misma memoria física pero que internamente se encuentra segmentada en memoria de instrucciones y memoria de datos. Luego de valorar, se propuso una solución que implementa la arquitectura de *Harvard tradicional*.

[1] Puede observar el set de instrucciones completo aquí.

Esta decisión se tomó debido a que reducía los tamaños de memoria, elimina la necesidad de lógica de control que determina con cuál memoria se está trabajando y offset en las direcciones de memoria, lo que permite realizar un diseño más simple y sencillo de entender. La Figura 9 muestra los diagramas de primer nivel de ambas propuestas.

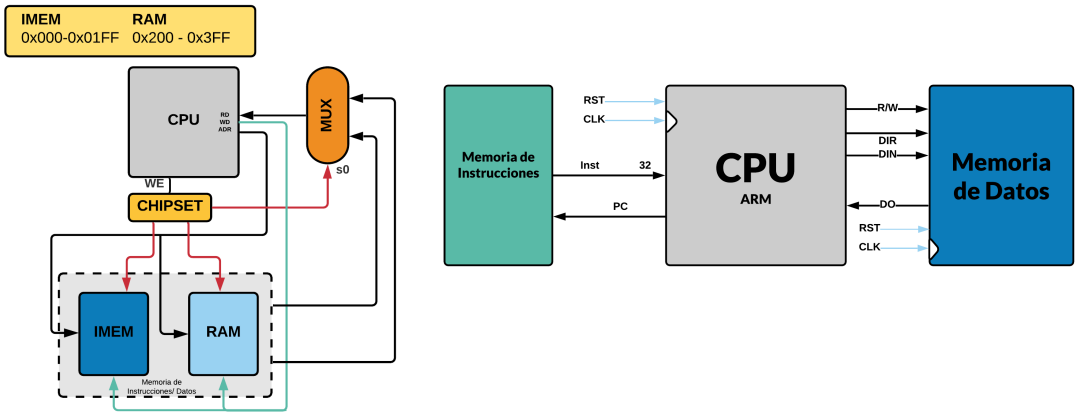


Figura 9. *Diagramas de primer nivel para Harvard (derecha) y Harvard modificada (izquierda).*

Una vez tomada esta decisión de diseño se procedió a realizar el diseño del microprocesador, que iba a constar de una **unidad de control**, una **unidad para el manejo de los riesgos** y un **camino de datos**, este diagrama se presenta en la Figura 10. En el primer diseño realizado, la unidad de control solo tomaría la instrucción y en su salida estarían las señales de control que entraban a la etapa de ejecución, el camino de los datos y las señales de control sería el mismo.

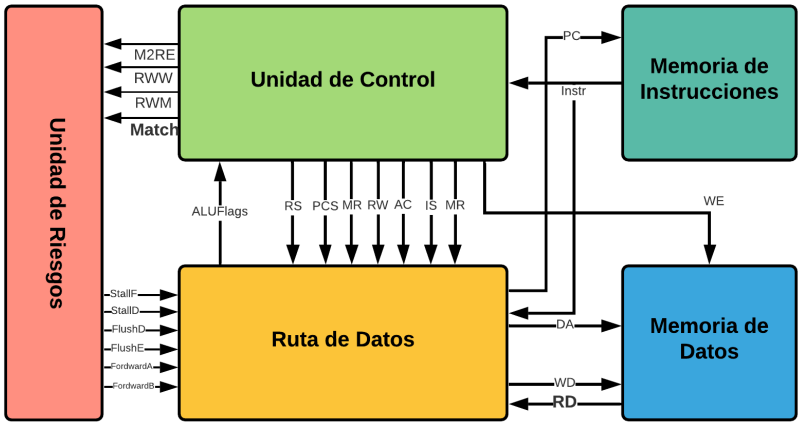


Figura 10. *Diagrama de segundo nivel de la microarquitectura diseñada.*

Esta solución fue descartada debido a que se generó un gran desorden en la estructura y era muy propensa a errores difíciles de encontrar. Para corregir esto se introdujeron registros módulo de control para que tuviera su propio pipeline con *las señales de control* y de esta manera se encontraría *aislado del pipeline de datos*.

El diagrama definitivo de la microarquitectura diseñada para correr las instrucciones del set de instrucciones GEO se presenta en la Figura 11.

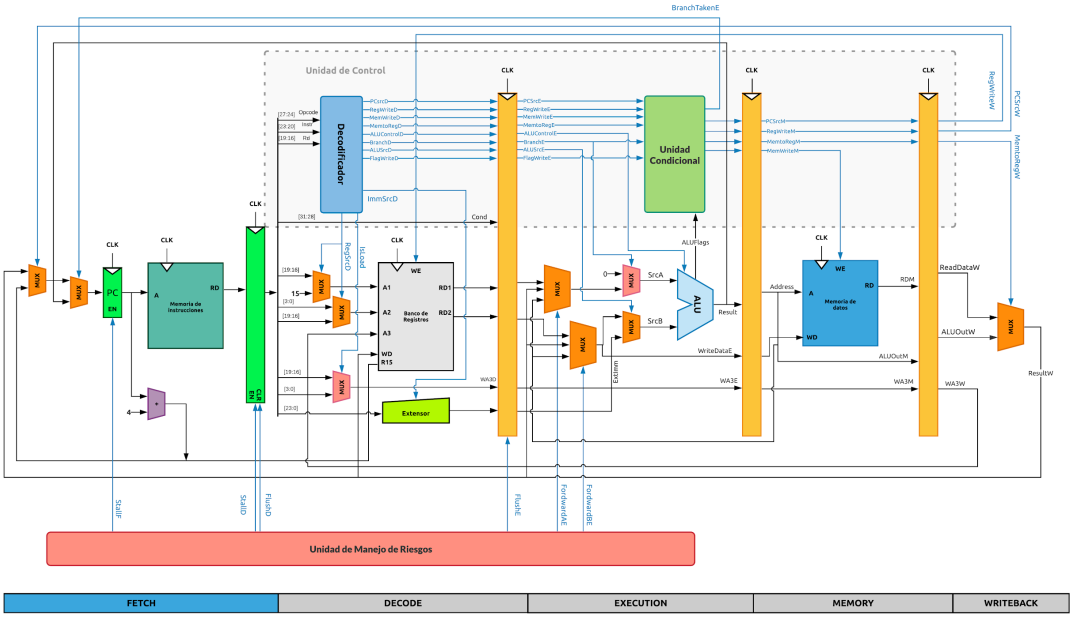


Figura 11. *Diagrama completo de la microarquitectura diseñada.*

Software

El aspecto a tratar es el compilador. El proyecto podría haberse realizado con una propuesta sencilla, desarrollando un programa que leyera los tokens y los tradujera (de acuerdo a unas reglas definidas) al binario del ISA diseñado, el diagrama de la Figura 12 (izquierda) muestra las etapas de esta solución. Sin embargo, analizamos que implementar un flujo completo de compilación con etapas de análisis sintáctico y léxico nos permitirían tener una mayor seguridad de que las instrucciones iban a estar libres de errores. La Figura 12 (derecha) presenta el proceso de esta solución.

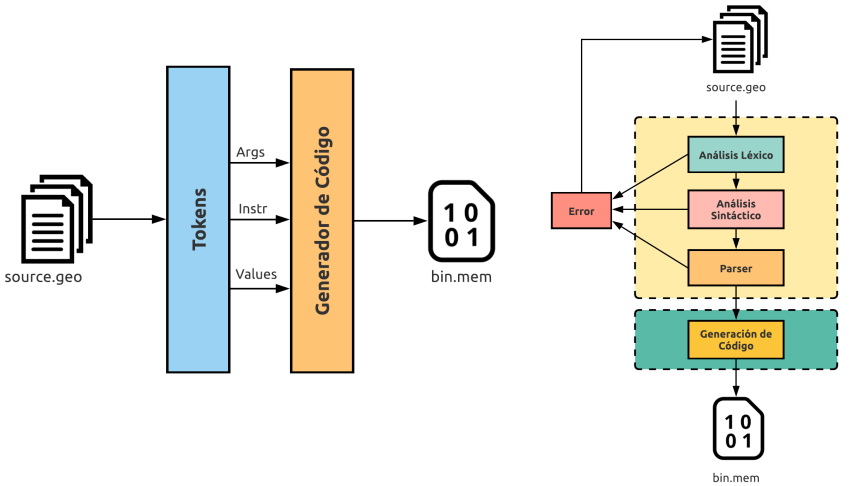


Figura 12. *Diagrama de etapas para las propuestas del compilador*

5. Referencias

[1] E. Zaira, “RISC-V UN ISA DE CÓDIGO ABIERTO” M. S. Thesis, Universidad de Cantabria, Santander, CA, 2019.

[2] S. Agudelo and V. Maldodano, “Supercomputación basada en la arquitectura ARM”, *edu*, 2017. [Online]: http://wiki.sc3.uis.edu.co/images/e/ed/GR11.pdf. [Accessed: 06-jul- 2017].

[3] S. Harris and D. Money, Digital Design and Computer Architecture, Waltham, MA: Elsevier Inc. 2016

[4] B. Shilpi, “Formal Verification of Application and System Programs Based on a Validated x86 ISA Model”, Dr. thesis. The University of Texas at Austin, Austin, TX, 2016.

6. Anexos

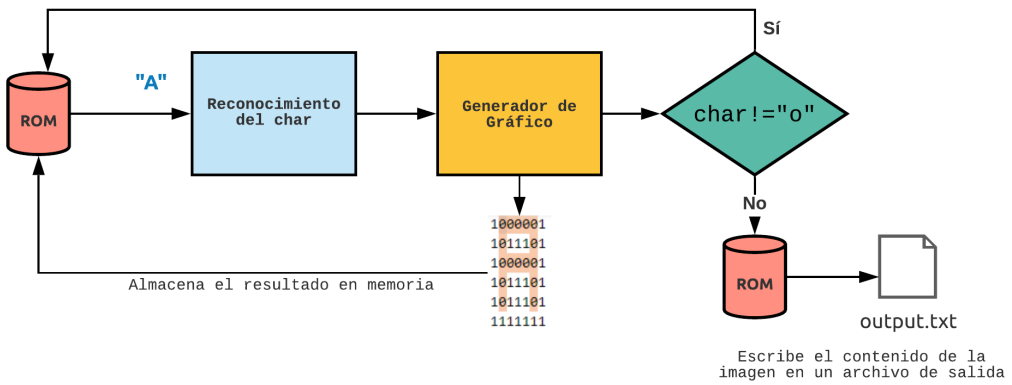


Figura 13. *Diagrama de flujo del algoritmo implementado en ensamblador*