Documentación Tecnica:
Pipelain ISA

Carlos Andrés Mata Calderón, Luis Felipe Vargas Jiménez, Ignacio Grané Rojas

Resumen

Este documento presenta una implementación detallada de un conjunto de instrucciones (ISA) diseñado para mostrar texto en una pantalla VGA. Se utiliza el algoritmo de Bresenham para calcular los puntos que forman las letras, asegurando una representación precisa y eficiente.

Introducción

La generación de gráficos y texto en pantallas VGA es una tarea fundamental en diversas aplicaciones de sistemas embebidos y dispositivos electrónicos. Este proyecto se centra en el diseño e implementación de un Application Specific Instruction Set Processor (ASIP) que utiliza un conjunto de instrucciones especializado (ISA) para representar texto en una pantalla VGA.

El ASIP está optimizado para realizar operaciones gráficas utilizando el algoritmo de Bresenham, conocido por su eficiencia en la generación de líneas y puntos. Este documento describe el proceso de diseño del ISA, la implementación del algoritmo de Bresenham para el trazado de caracteres y los detalles técnicos necesarios para mostrar texto en una pantalla VGA.

Listado de Requerimientos

Pipelining

La técnica de pipelining permite al procesador manejar múltiples instrucciones de manera simultánea al dividir su ejecución en etapas como decodificación, ejecución y escritura de resultados. Esto es crucial para optimizar el rendimiento del procesador, ya que minimiza los ciclos de inactividad entre la ejecución de instrucciones consecutivas. Por ejemplo, mientras una instrucción está en la etapa de ejecución, otra puede estar en la etapa de decodificación, logrando así un uso continuo de los recursos de la CPU.

Implementación en SystemVerilog

La implementación del procesador se realizará en SystemVerilog, un lenguaje de descripción de hardware que permite definir y simular componentes digitales. En este paso, se crearán módulos funcionales, conexiones entre ellos y se definirán los comportamientos lógicos que representen las operaciones del procesador. El objetivo es modelar el procesador con precisión, facilitando la simulación y validación antes de pasar a la implementación física.

Segmentación de Memoria y Acceso a E/S

Organizar la memoria implica definir una estructura para la RAM, y ROM, permitiendo un acceso rápido a los datos más utilizados.

Pruebas Unitarias e Integración

Las pruebas unitarias verifican individualmente la funcionalidad de cada componente, identificando fallos en módulos como la ALU o los registros. Las pruebas de integración, por otro lado, se enfocan en asegurar que estos componentes interactúan correctamente cuando se combinan. Esta doble capa de pruebas es esencial para detectar errores en etapas tempranas y corregirlos antes de avanzar en el diseño. (Revisar las Figura 6, Figura 7 y Figura 8)

Diagramas de Bloques

Los diagramas de bloques son representaciones gráficas que detallan la estructuiones de solución al problema: Para el problema planteado de- berán documentarse al menos dos opciones de solución. Cada solución deberá ser acompañada de algún tipo de diagrama. Estas opciora y la interacción de los componentes principales del procesador. Incluyen módulos como la ALU, registros, memoria y buses. Estos diagramas son fundamentales para visualizar cómo fluyen los datos y las señales de control a través del procesador, permitiendo a los ingenieros identificar posibles cuellos de botella y optimizar el diseño.

Algoritmo de Bresenham

El algoritmo de Bresenham se utiliza para trazar líneas rasterizadas, es decir, representadas mediante píxeles en una pantalla. Es un método eficiente que calcula qué píxeles deben encenderse para crear líneas lo más rectas posibles en pantallas de baja resolución. Esto resulta esencial en la representación gráfica de diagramas y figuras en una salida VGA.

Imagen Binarizada

Para simplificar la representación gráfica y optimizar el procesamiento, las imágenes se binarizarán, lo que implica reducirlas a un formato en blanco y negro de 250x250 píxeles. Al trabajar con una paleta limitada a solo dos colores, se reduce la complejidad del procesamiento y el tiempo de renderizado, haciendo más rápida la generación de gráficos.

Presentación VGA

La interfaz VGA se empleará para la visualización de gráficos en monitores estándar, permitiendo que los datos de salida del procesador se conviertan en imágenes visuales claras. Esto facilita la visualización de diagramas, gráficos y figuras generadas por el software del sistema.

Compilador

Un compilador es esencial para traducir el código ensamblador al formato binario que pueda ser ejecutado directamente por el procesador. Esto incluye interpretar instrucciones personalizadas que se implementarán en el diseño del procesador, lo que garantiza que el software generado sea compatible con el hardware específico.

Código Ensamblador

Para demostrar las capacidades del compilador y del procesador personalizado, se deben proporcionar ejemplos de código ensamblador que reflejen las instrucciones personalizadas. Esto brinda a los desarrolladores una referencia sobre cómo usar las nuevas características y permite probar y validar el rendimiento del procesador.

Elaboración de Opciones de Solución al Problema

Arquitectura de Memorias

Seleccionar adecuadamente la arquitectura de memoria del procesador es fundamental porque esta decisión influye directamente en el rendimiento y la eficiencia del sistema. La arquitectura de memoria determina cómo el procesador almacena y accede tanto a los datos como a las instrucciones, afectando la velocidad de procesamiento y la capacidad para manejar tareas simultáneamente. Una elección acertada puede minimizar los cuellos de botella, mejorar la respuesta del sistema bajo carga y optimizar el uso de recursos, lo que es crucial en aplicaciones de alto rendimiento o en entornos donde la eficiencia operativa es prioritaria.

1. Arquitectura Von Neuman: La arquitectura de Von Neumann utiliza una única memoria para almacenar tanto datos como instrucciones, permitiendo una ejecución secuencial de comandos. Este diseño simplifica la estructura del hardware y es esencial para la funcionalidad básica de las computadoras modernas. [3]

2. Arquitectura Harvard: La arquitectura Harvard se caracteriza por tener memorias separadas para datos e instrucciones, lo que permite que la CPU acceda a ambos simultáneamente. Este diseño mejora la velocidad de procesamiento y es especialmente útil en sistemas donde el rendimiento y la eficiencia en la ejecución de instrucciones son críticos. [3]

Arquitectura de Conjunto Instrucciones

1. CISC (Complex Instruction Set Computing): La arquitectura CISC está diseñada para minimizar la cantidad de instrucciones por programa sacrificando la simplicidad y el tiempo de ejecución de cada instrucción. Cada instrucción puede realizar tareas complejas, lo que reduce la cantidad de líneas de código necesarias para realizar operaciones. [2]

2. RISC (Reduced Instruction Set Computing): La arquitectura RISC está diseñada para tener un conjunto de instrucciones simple y uniforme. Cada instrucción realiza una operación básica y se ejecuta en un ciclo de reloj fijo, permitiendo un pipeline más profundo y eficiente. [1]

Cantidad de Bits de la Instrucción

1. Instrucciones de 32 bits: La arquitectura con instrucciones de 32 bits permite codificar más información en cada instrucción, lo que facilita operaciones complejas y reduce el número total de instrucciones necesarias para completar una tarea. Esta opción es adecuada para sistemas con más registros y tipos de instrucciones.

2. Instrucciones de 16 bits: La arquitectura con instrucciones de 16 bits es más compacta y eficiente en términos de memoria, pero limita la cantidad de información que puede ser codificada en cada instrucción. Esta opción es adecuada para sistemas con menos registros y operaciones más simples.

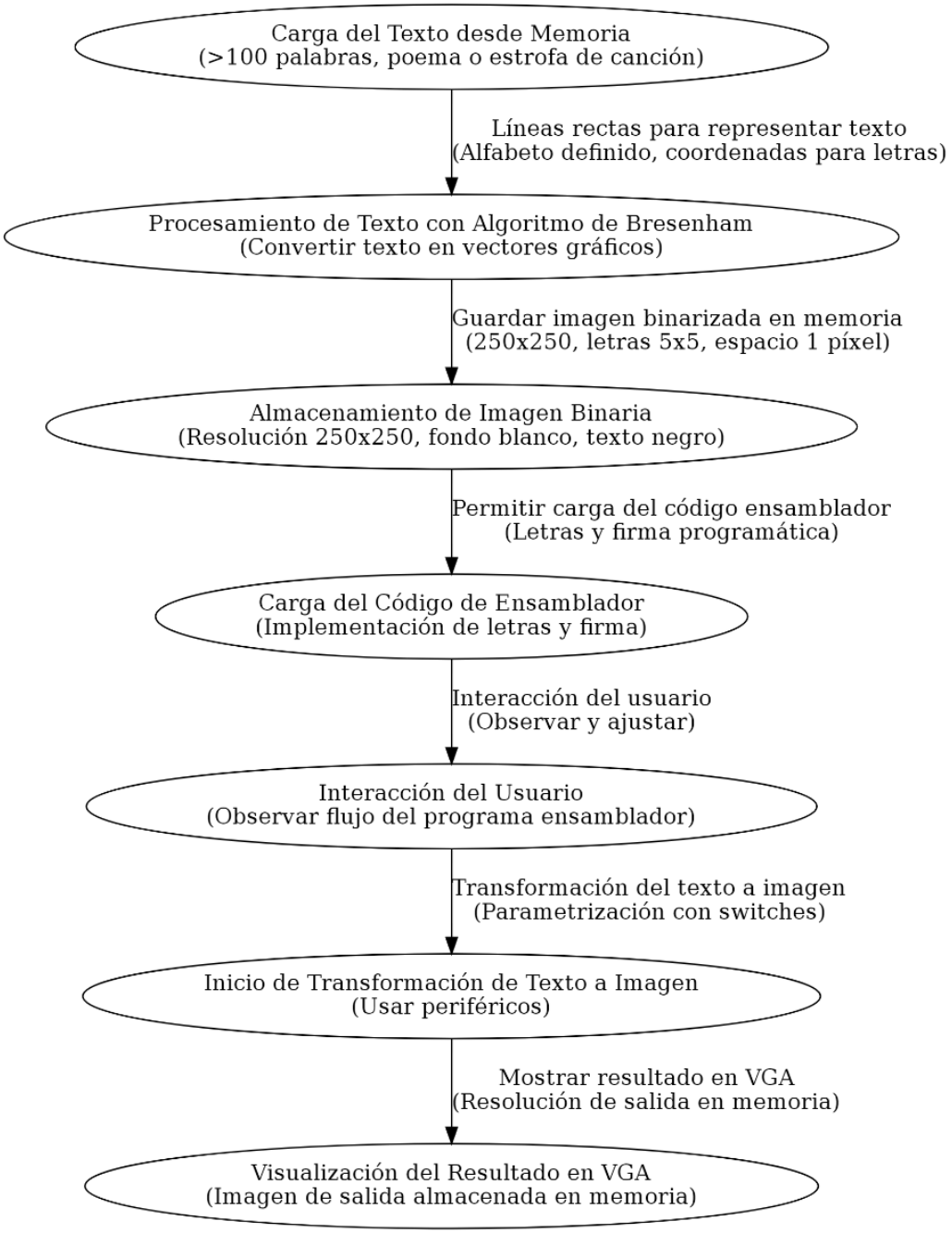

Figura 1: Proceso de generación de texto

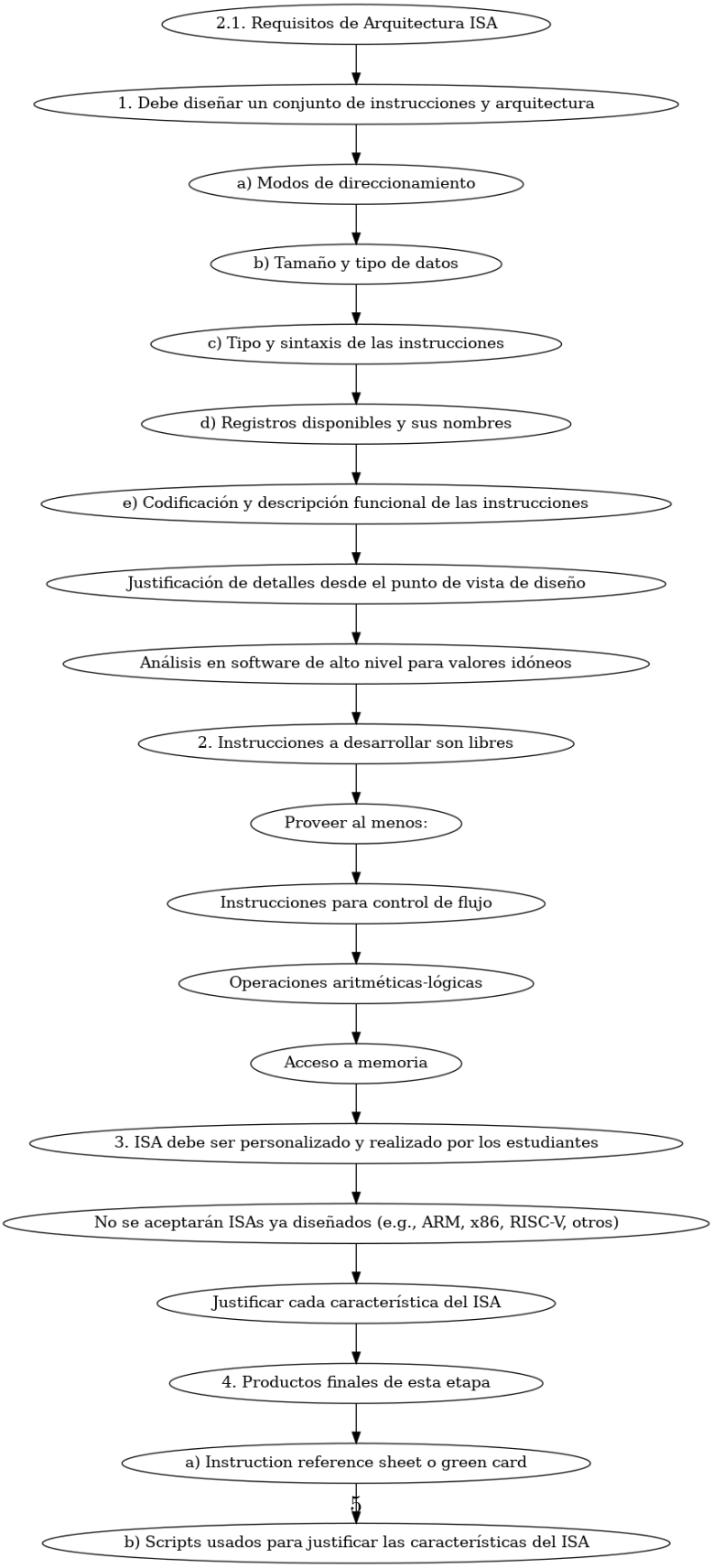

Figura 2: Requisitos del ISA

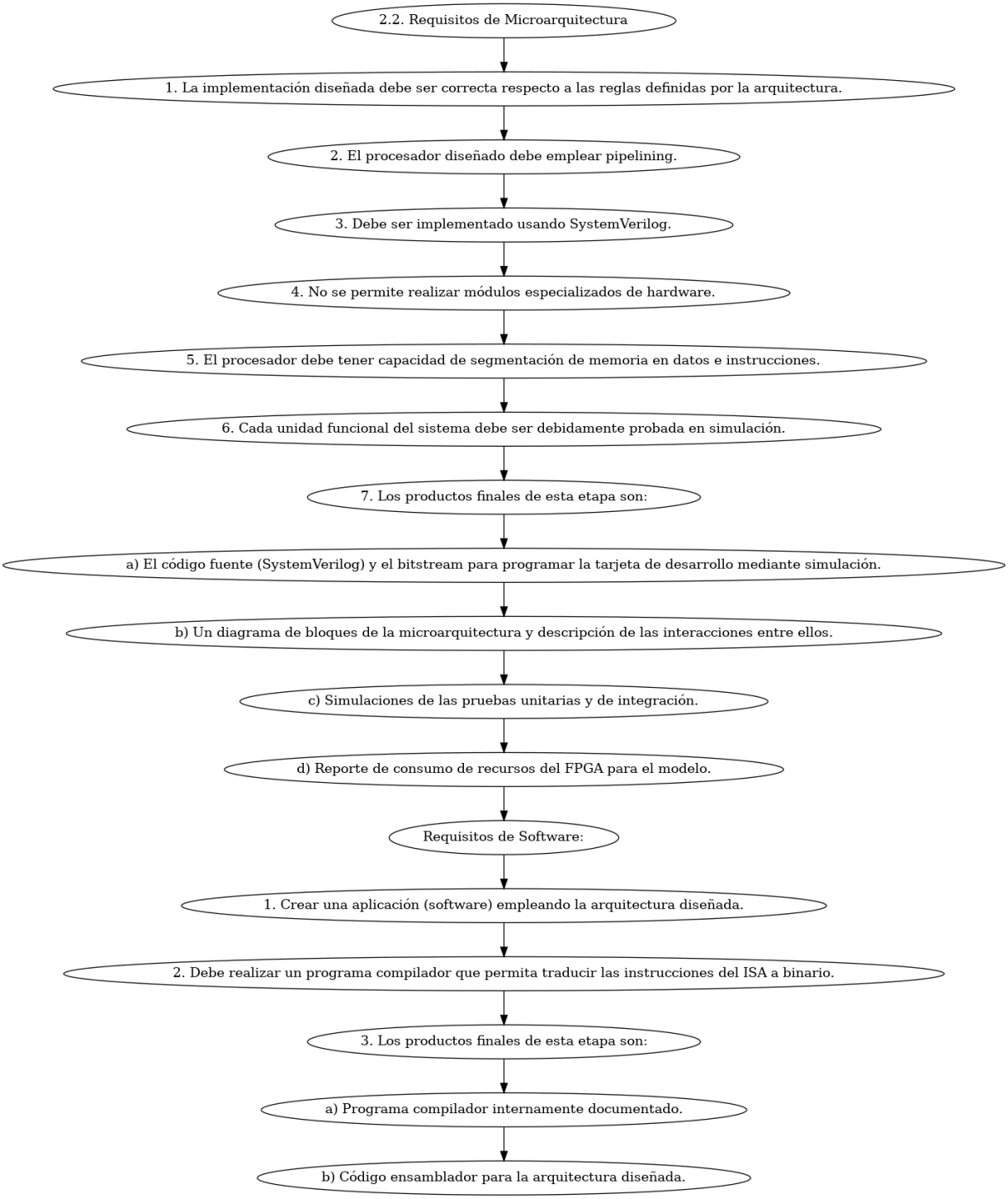

Figura 3: Requisitos de microarquitectura

Comparación de Soluciones

Arquitectura de Memorias

1. Von Neumann

   - *Ventajas*: Simplicidad de diseño, ya que en la arquitectura Von Neumann, los datos y los programas se almacenan en la misma memoria y se accede a ellos a través de un único canal de datos y direcciones. Esto simplifica el diseño y la construcción del hardware. Flexibilidad, pues al usar la misma memoria para almacenar tanto datos como instrucciones, es más fácil modificar el programa sin cambiar el hardware.
   - *Desventajas*: Cuello de botella del bus de datos, pues como los datos y las instrucciones comparten la misma ruta de memoria, solo se pueden procesar de uno en uno, lo que limita la velocidad de ejecución. Vulnerabilidades de seguridad, ya que al almacenar datos y programas en la misma memoria, se potencian los riesgos de seguridad donde programas maliciosos pueden alterar o corromper datos y otros programas fácilmente.

2. Harvard

   - *Ventajas*: Mayor velocidad de procesamiento, al tener memorias separadas para datos e instrucciones, con sus respectivos buses, permite la ejecución simultánea de la recuperación de datos y de instrucciones, eliminando el cuello de botella presente en la arquitectura Von Neumann. Seguridad mejorada, separar físicamente la memoria de datos de la memoria de instrucciones puede aumentar la seguridad.
   - *Desventajas*: Complejidad de diseño, el diseño separado para dos memorias y sistemas de bus hace que el hardware sea más complejo y potencialmente más costoso de diseñar y fabricar. Menos flexibilidad, la separación de memorias puede hacer que sea más difícil realizar ciertas optimizaciones de programación que son más factibles en la arquitectura Von Neumann.

Arquitectura de Conjunto Instrucciones

1. CISC

   - *Ventajas*: Menor cantidad de líneas de código debido a la complejidad de las instrucciones. Ideal para aplicaciones con instrucciones complejas y poco frecuentes.
   - *Desventajas*: Mayor consumo de energía y recursos debido a la complejidad del hardware. Pipeline más complicado y menos eficiente. Mayor dificultad en el diseño y depuración del procesador.

2. RISC

   - *Ventajas*: Alta eficiencia y rendimiento debido al pipeline profundo y a la simplicidad de las instrucciones. Facilidad en el diseño, implementación y depuración del procesador. Menor consumo de energía y recursos en comparación con CISC.
   - *Desventajas*: Mayor cantidad de líneas de código debido a la simplicidad de las instrucciones. Menos adecuado para aplicaciones con instrucciones complejas y frecuentes.

Aspecto	Opción 1: CISC	Opción 2: RISC
Simplicidad del Diseño	Baja	Alta
Rendimiento	Medio	Alto
Consumo de Recursos	Alto	Bajo
Facilidad de Implementación	Baja	Alta
Paralelismo y Pipeline	Bajo	Alto
Gestión de Peligros	Compleja	Sencilla
Flexibilidad	Alta	Media

Tabla 1: Comparación entre Arquitectura CISC y RISC

Cantidad de Bits de la Instrucción

1. Instrucciones de 32 bits
   - *Ventajas*: Capacidad para operaciones complejas en una sola instrucción. Menor número de instrucciones necesarias para completar una tarea. Mayor flexibilidad y capacidad de expansión.
   - *Desventajas*: Mayor consumo de memoria para almacenar instrucciones. Mayor complejidad en la decodificación de instrucciones.
2. Instrucciones de 16 bits
   - *Ventajas*: Menor consumo de memoria para almacenar instrucciones. Simplicidad en la decodificación de instrucciones. Menor complejidad en el diseño del hardware.
   - *Desventajas*: Capacidad limitada para operaciones complejas en una sola instrucción. Mayor número de instrucciones necesarias para completar una tarea.

Selección Propuestas Final

Arquitectura Harvard

En nuestro proyecto de arquitectura de computadores, optamos por implementar la arquitectura Harvard en lugar de la arquitectura de von Neumann. La decisión de utilizar la arquitectura Harvard se basó en su capacidad para separar físicamente la memoria de instrucciones (ROM) y la memoria de datos (RAM). Esta separación permite que las instrucciones y los datos se manejen simultáneamente, lo que resulta en una mayor eficiencia y velocidad de procesamiento. En particular, la ROM se utiliza exclusivamente para almacenar las instrucciones del programa, garantizando así que las instrucciones no se modifiquen accidentalmente durante la ejecución del programa. Por otro lado, la RAM se dedica al manejo dinámico de los datos del programa, proporcionando una memoria de acceso rápido que facilita la lectura y escritura de datos durante la operación del procesador. Esta dualidad en el manejo de la memoria permite una ejecución más rápida y eficiente de los programas, evitando los cuellos de botella típicos de la arquitectura von Neumann, donde tanto las instrucciones como los datos comparten el mismo bus y memoria. El diagrama de bloques desarrollado para tales propósitos se adjunta en la Figura 4, en donde es posible observar la conexión entre el procesador con las respectivas memorias ROM de instrucciones y RAM para la escritura de datos.

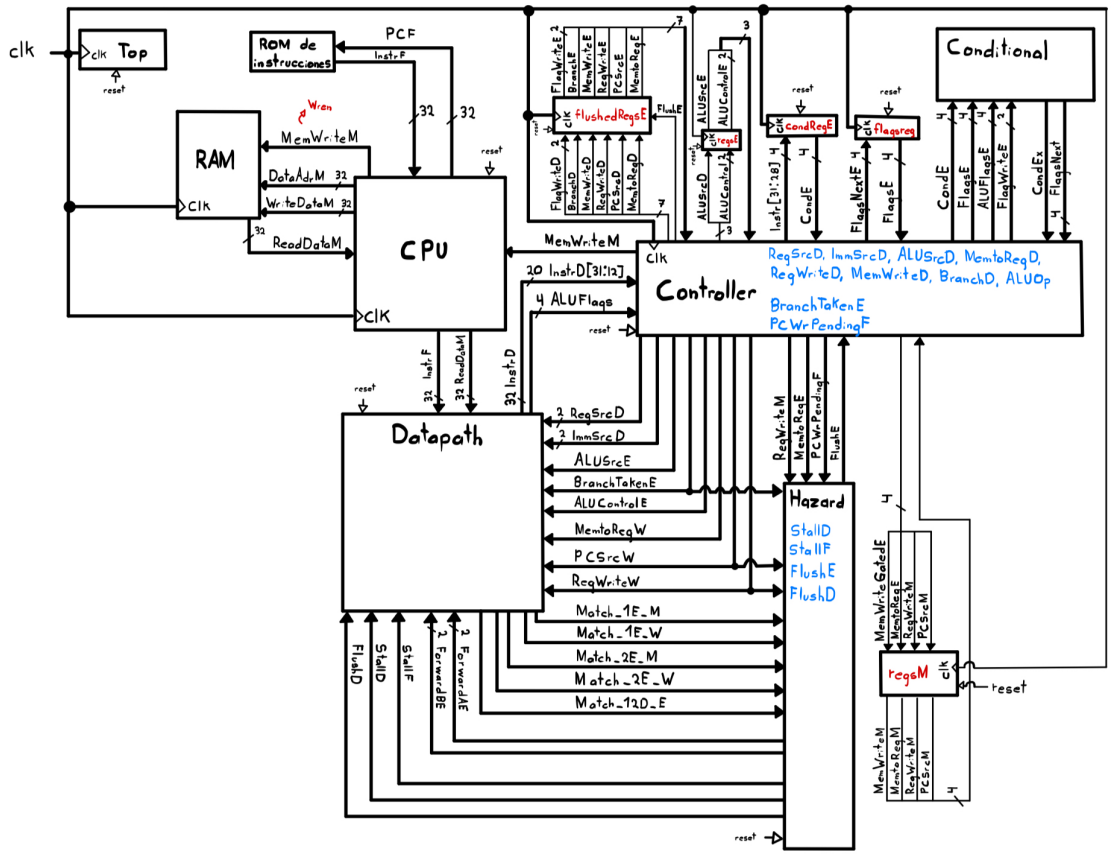

Figura 4: Diagrama de bloques de la arquitectura desarrollada

Arquitectura RISC

Para este proyecto, la Opción 2: Arquitectura RISC es la más adecuada. La simplicidad en el diseño, implementación y depuración, junto con la alta eficiencia y rendimiento del pipeline profundo, hacen que la arquitectura RISC sea ideal para la implementación en un FPGA con restricciones de área y recursos. Aunque la arquitectura CISC ofrece ventajas en términos de menor cantidad de líneas de código y mayor flexibilidad para instrucciones complejas, su mayor consumo de recursos y complejidad hacen que sea menos viable para este proyecto.

En la Figura 5 se aprecia el consumo de recursos para el diseño sintetizable del procesador junto a su conjunto de instrucciones RISC. En dicha imagen es posible observar el poco uso de recursos que este conlleva y se demuestra así una de sus principales caracterísitcas como arquitectura RISC.

	Resource	Usage
1	Estimate of Logic utilization (ALMs needed)	997
2		
3	▼ Combinational ALUT usage for logic	1287
1	-- 7 input functions	27
2	-- 6 input functions	585
3	-- 5 input functions	252
4	-- 4 input functions	91
5	-- <=3 input functions	332
4		
5	Dedicated logic registers	1052
6		
7	I/O pins	97
8	Total MLAB memory bits	0
9	Total block memory bits	3145728
10		
11	Total DSP Blocks	0

Figura 5: Recursos utilizados durante el desarrollo del proyecto

32 Bits por Instrucción

Para este proyecto, la Opción 1: Instrucciones de 32 bits es la más adecuada. La capacidad de manejar operaciones complejas y la flexibilidad proporcionada por las instrucciones de 32 bits son esenciales para implementar un ASIP eficiente para la generación de gráficos y texto. Aunque la opción de 16 bits ofrece ventajas en términos de simplicidad y consumo de memoria, su capacidad limitada para operaciones complejas y el mayor número de instrucciones necesarias la hacen menos viable para este proyecto. Revisar el Anexo (6), para mayores detalles de las caracteristicas del ASIP.

Anexo

Registros para Argumentos y Variables Temporales (v0 a v3)

Estos cuatro registros están optimizados para el manejo de argumentos de funciones y para almacenar valores temporales durante cálculos intermedios. Esta característica es esencial en el diseño de ISAs porque permite a las funciones pasar y recibir valores de manera eficiente sin recurrir constantemente a la memoria, lo que sería más lento.

Registros Guardados (v4 a v11)

Estos ocho registros son esenciales para operaciones donde es necesario preservar el estado de las variables a lo largo de diferentes llamadas a funciones. Esto es particularmente útil en aplicaciones que involucran procesos iterativos o recursivos donde el estado anterior de las variables debe mantenerse intacto después de las llamadas a funciones.

Registros Especiales

Registro Temporal Adicional (v12)

Este registro se ofrece como un espacio adicional para variables temporales, lo cual es útil en operaciones complejas que requieren más de los cuatro registros básicos para variables temporales.

Registros de Control de Flujo y Sistema (sk, lk, pc)

- **r13 (sp):** El puntero de pila es crucial para la gestión de memoria dinámica en tiempo de ejecución, especialmente en lenguajes que utilizan mucho la recursividad y las llamadas a funciones.
- **r14 (lr):** El registro de enlace guarda la próxima dirección de retorno cuando una función es llamada. Esto facilita el regreso seguro a la posición correcta en el código después de una llamada a función.
- **r15 (pc):** El contador de programas es el registro más crítico para la ejecución secuencial de instrucciones, manteniendo la dirección de la siguiente instrucción a ejecutar.

Resumen del Conjunto de Instrucciones

- **sum**: Suma dos registros o un registro con un inmediato, almacenando el resultado en un registro.
- **com**: Compara dos registros o un registro con un inmediato.
- **set**: Asigna un valor inmediato o de registro a un registro de destino.
- **dec**: Resta dos registros o un registro con un inmediato.
- **divi**: Divide potencias de 2 mediante desplazamiento aritmético.
- **xor**: XOR lógico entre dos registros o con un inmediato.
- **aset**: Asigna el valor inverso de un registro o inmediato.
- **stw**: Almacena una palabra en la memoria.

- **ldw**: Carga una palabra desde la memoria.
- **savepix**: Guarda un píxel en la memoria.
- **letter**: Carga el byte de un carácter.
- **end**: Finaliza el programa.
- **nop**: No realiza ninguna operación.

Formatos de Instrucción

Data Processing Register (DPR)

Para operaciones de procesamiento de datos entre registros.

Data Processing Immediate (DPI)

Permite procesamiento de datos con valores inmediatos.

Memory Register (MR)

Accede a memoria usando registros.

Memory Immediate Offset (MIO)

Accede a memoria con un desplazamiento inmediato.

Branch (B)

Instrucciones de salto condicional e incondicional.

Special (S)

Instrucciones especiales como terminación y operaciones nulas.

Registro y Banderas

Condiciones

Describe los códigos de condición y su uso en instrucciones condicionales.

Registros

Detalles de registros de propósito específico y general.

Testbench

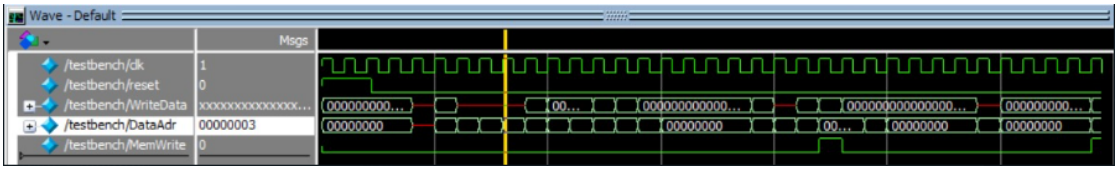

Figura 6: Testbench 1: verficación de ejecucion de instrucciones en el pipeline

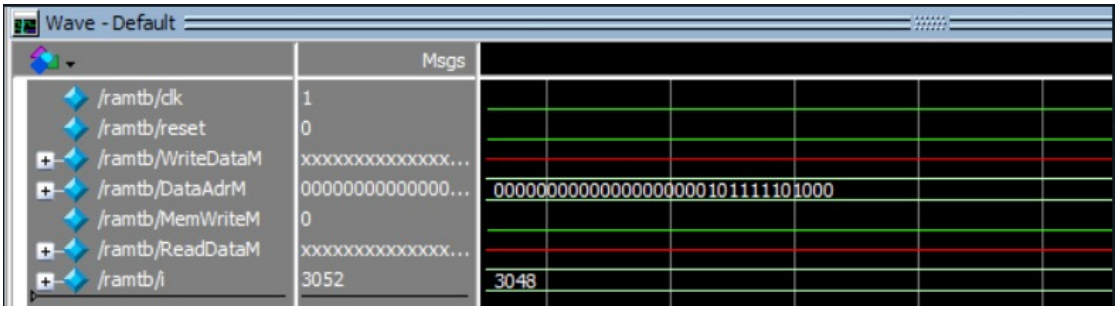

Figura 7: Testbench 2: escritura de memoria

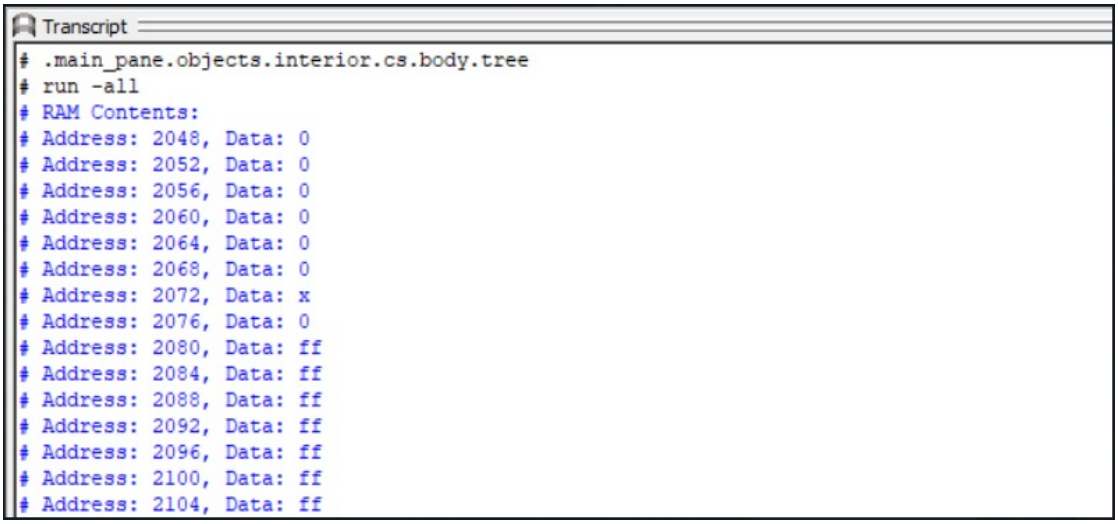

Figura 8: Direcciones de memoria (RAM) en donde se escriben las salidas del algoritmo de Bresenham

Referencias

[1] Harris, S. L., and Harris, D. M. (2015). *Digital Design and Computer Architecture: ARM Edition*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 978-0-12-800056-4.

[2] Hennessy, J., and Patterson, D. (2012). *Computer Architecture: A Quantitative Approach* (5th ed.). Elsevier – Morgan Kaufmann.

[3] González, J., and García, R. (2019). Notas de clase de los profesores: Jeferson González y Ronald García.