**Proyecto Grupal #1**
Instituto Tecnológico de Costa Rica
Arquitectura de Computadores II - CE 4302
Prof. Luis Barboza Artavia

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

# DISEÑO E IMPLEMENTACIÓN DE UN ASIP VECTORIAL PARA LA APLICACIÓN DE FILTROS CLÁSICOS EN IMÁGENES

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) vectorial para la aplicación de tres filtros clásicos a una imagen, escala de grises, sepia y negativo**.

Figura 1. *Imagen a la cual se le ha aplicado un filtro de escala de grises*

El **paralelismo** en múltiples niveles ha sido la motivación del diseño de computadoras con el fin de mejorar el rendimiento en términos de energía y costo. Un tipo básico es el **paralelismo a nivel de datos**, donde surge porque hay muchos datos que pueden ser operados al mismo tiempo. Las arquitecturas vectoriales son los referentes en este aspecto, porque utilizan el concepto de **SIMD** (*Single Instruction Multiple Data*) para explicar el paralelismo a nivel de datos aplicado a una única instrucción a una colección de datos de manera paralela.

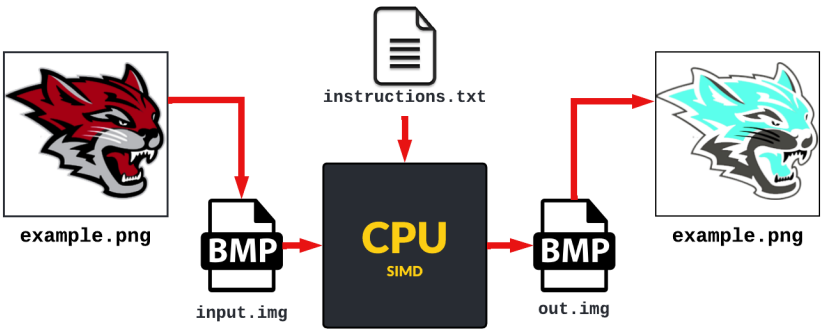



Figura 2. *Diagrama general de la ejecución del programa.*

## 2. Requerimientos del Sistema

Este proyecto pretende implementar la **arquitectura del set de instrucciones** para un **procesador vectorial**. En este proyecto se desarrollará un set de instrucciones propio y específico, se debe implementar el procesador vectorial y se debe desarrollar el software para compilar el programa y mostrar el resultado.

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:<br>● Instrucciones escalares para registros, inmediatos, memoria y condicionales.<br>● Datos inmediatos de 8 bits.<br>● Banco de registros escalares con 8 registros de 32 bits.<br>● Banco de registros vectoriales con 8 registros que almacenan un vector de 8 datos de 32 bits (256 bits en total).<br>● Instrucciones vector-vector y vector-escalar.<br>● El sistema debe poseer al menos 4 lanes.<br>● Manejo de decisiones condicionales mediante flags.<br>● Las instrucciones deben ser de 26 bits. |
| **Microarquitectura** | Memoria | El sistema debe tener capacidad de segmentación de memoria en datos e instrucciones.<br>● Se requiere de una memoria de instrucciones y datos que reciba la dirección como entrada y a su salida presente el contenido almacenado.<br>● El valor de la dirección se le realiza un corrimiento de bits para ajustarla al alineamiento de las memorias.<br>● 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.<br>El algoritmo debe seleccionarse mediante los interruptores de la FPGA. |
| | 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 ejecutar, las |

| | | |
|---|---|---|
| | | flags condicionales y otros bits como el inmediato y el load para determinar las señales de control para cada etapa del procesador.<br>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 aplicación de filtros** | Escala de Grises | El algoritmo obtiene el promedio de los tres valores de los canales RGB, se multiplican por una constante. |
| | Sepia | Para este filtro se extrae el canal RGB y se calculan nuevos valores multiplicándolos por una constante. |
| | Negativo | Por cada píxel el nuevo valor de RGB se consigue restando 255 a cada entrada. |
| **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 filtrada que se ha generado. |
| | Generación de la imagen | El sistema deberá interpretar el contenido y generar un archivo de imagen.<br>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. |

## 3. Diseño de la solución

La microarquitectura es uno de los elementos que ya poseen un fundamento teórico de cómo debía realizarse. A pesar de esto, como equipo de desarrollo debimos tomar algunas decisiones de diseño. Se presentan a continuación las propuestas:

### Pipeline

Para el pipeline del procesador se considera una primera propuesta en la que se tiene **un único camino de datos con todos sus componentes preparados para una tratar datos vectoriales**. Un módulo de control sería el encargado de manejar el modo de operación de los componentes y decidir si se efectúan las acciones escalares o vectoriales. La arquitectura contiene una **ALU vectorial de 8 *lanes***, en el caso de necesitarse una operación escalar, el control selecciona solamente un *lane* para esta ejecución.

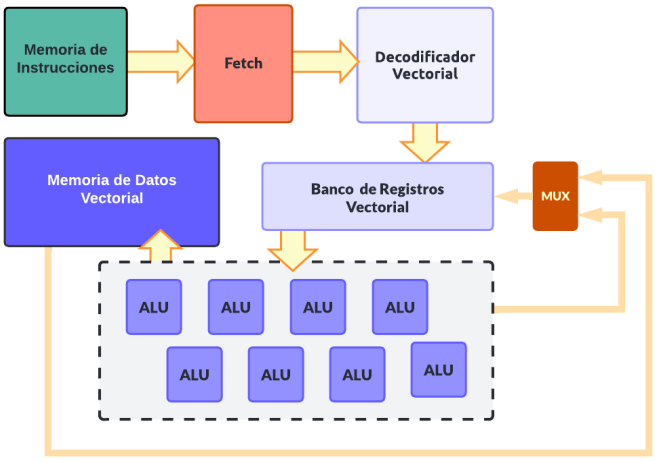



Figura 3. *Pipeline vectorial completo.*

La segunda propuesta utiliza **dos caminos de datos** para las etapas de decodificación, ejecución, memoria y writeback, una para las operaciones escalares y otra para las operaciones vectoriales. **La unidad de control se encarga de determinar a cuál pipeline dirigir el flujo.** Este diseño posee 2 ALU, una escalar y una vectorial. Un aspecto a importante es la

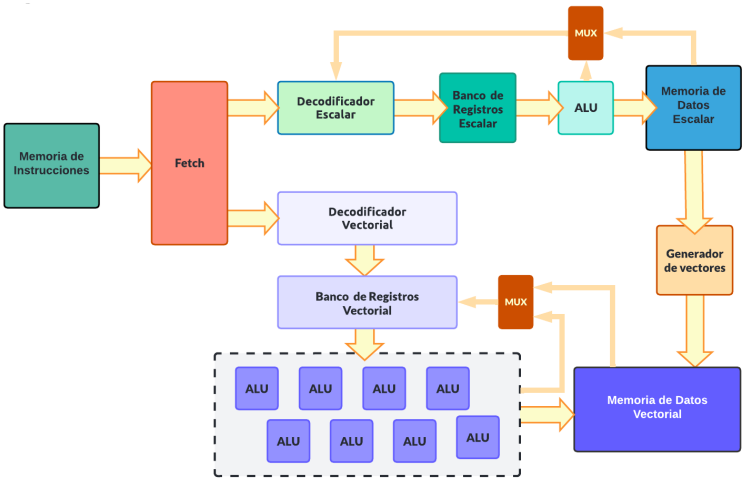



Figura 4. *Pipeline vectorial combinado.*

utilización de **dos memorias de datos**, una escalar y una dedicada a vectores. El módulo **convertidor vectorial** se encarga de cargar los vectores de la memoria principal de datos a la memoria vectorial.

## 4. Comparativa entre propuestas de solución

En la sección anterior se presentaron las propuestas de solución para el diseño de la **microarquitectura para el procesador vectorial**, en esta se realizará una comparación entre cada uno de los diseños propuestos.

### Pipeline

La primera propuesta, del **pipeline vectorial completo**, presenta **menor cantidad de hardware** al poseer un único camino de datos. Sin embargo, esta sencillez en el hardware se ve opacada por la **complejidad de la unidad de control**, que debe manejar el estado de todos los componentes en todo momento para determinar si están trabajando un vector o un escalar.
Por otra parte, la propuesta del **pipeline vectorial combinado**, presenta una **ruta de datos completa adicional**, pero el **hardware para controlar el flujo de operaciones resulta más sencillo**, incluso reduce la cantidad de riesgos de datos cuando exista una combinación de instrucciones escalares y vectoriales.

## 5. Propuesta Final

Luego de analizar las propuestas de solución, se elige i**mplementar la arquitectura de la propuesta del pipeline vectorial combinado**, debido a que el procesador será mucho más funcional si cuenta con un pipeline escalar, para las ejecuciones que requieran estas operaciones combinadas.

## 6. Implementación del diseño

### Compilador

El aspecto a tratar es el **compilador**. El proyecto implementa un flujo completo de compilación con etapas de análisis sintáctico y léxico, que nos permite tener una mayor seguridad de que las instrucciones iban a estar libres de errores. La Figura 5 (derecha) presenta el proceso de esta solución.

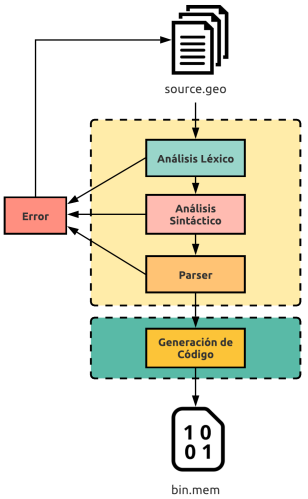



Figura 5. *Diagrama de etapas del compilador*

### Visualizador de imagen de salida

Se implementó una aplicación web que permite seleccionar un archivo `*.img` y lo reconstruye en una imagen con formato `jpg`. Luego muestra la imagen en el navegador.

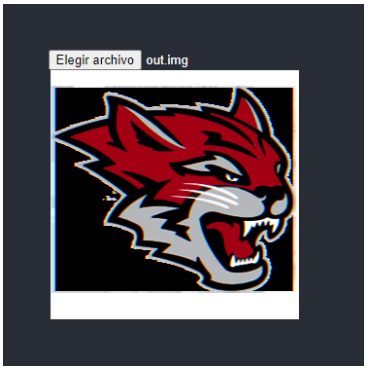



Figura 6. *Programa image-visualizer*

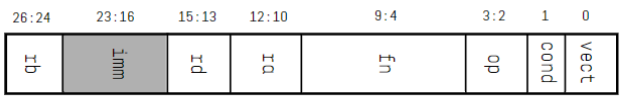
### Set de Instrucciones

El set de instrucciones desarrollado es el siguiente:

**Tabla 2. Listado de instrucciones**

| Instrucción | Operación | Descripción |
|---|---|---|
| **add <dest>,<a>,<b>** | `R[rd] = R[ra]+R[rb]` | Suma de dos valores escalares. |
| **sub <dest>,<a>,<b>** | `R[rd] = R[ra]-R[rb]` | Resta de dos valores escalares. |
| **div <dest>,<a>,<b>** | `R[rd] = R[ra]/R[rb]` | División de dos valores escalares. |
| **addv <dest>,<a>,<b>** | `V[rd] = V[ra]+V[rb]` | Suma de dos vectores. |
| **subv <dest>,<a>,<b>** | `V[rd] = V[ra]-V[rb]` | Resta de dos vectores. |
| **divv <dest>,<a>,<b>** | `V[rd] = V[ra]/V[rb]` | División de dos vectores. |
| **mulv <dest>,<a>,<b>** | `V[rd] = V[rd]*V[rb]` | Multiplicación de dos vectores. |
| **str <dest>,<src>,<imm>** | `M[rd] = R[rs+off]` | Almacena un dato escalar de la memoria al registro. |
| **ldr <dest>,<src>,<imm>** | `R[rd] = M[rs+off]` | Carga un dato escalar de un registro a una posición de memoria. |
| **strv <dest>,<src>,<imm>** | `M[rd] = V[rs+off]` | Almacena un dato vectorial de la memoria al registro. |
| **ldrv <dest>,<src>,<imm>** | `V[rd] = M[rs+off]` | Carga un dato vectorial de un registro a la memoria. |
| **bge <tag>** | `pc = pc+4, pc = tag` | Salto condicional a una etiqueta dentro del programa. |

La estructura de una instrucción es la que se muestra en la siguiente figura:




**vect:** vector/escalar
**cond:** Flag de condición
**op:** ver tabla OPS*

**fn:** I[4], P[5], U[6], B[7],W[8], L[9]

**Operación:**
[1] rd = ra ? rb
[2] rd = ra ? imm

**imm:** 8 bits de valor inmediato

Figura 7. *Estructura de una instrucción del ISA GEO.*

## Procesador Vectorial

En el diagrama de primer nivel del procesador desarrollado se presenta que el CPU presenta una arquitectura de Harvard, con dos memorias, una para instrucciones y otra para los datos.

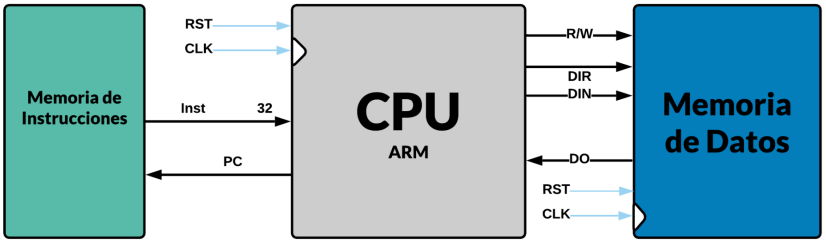



Figura 8. *Diagrama de primer nivel de procesador.*

Profundizando en el módulo del CPU, observamos que posee un módulo para el control, otro para la detección de riesgos y las rutas de datos. Esto se aprecia en el diagrama de segundo nivel de la Figura 8.

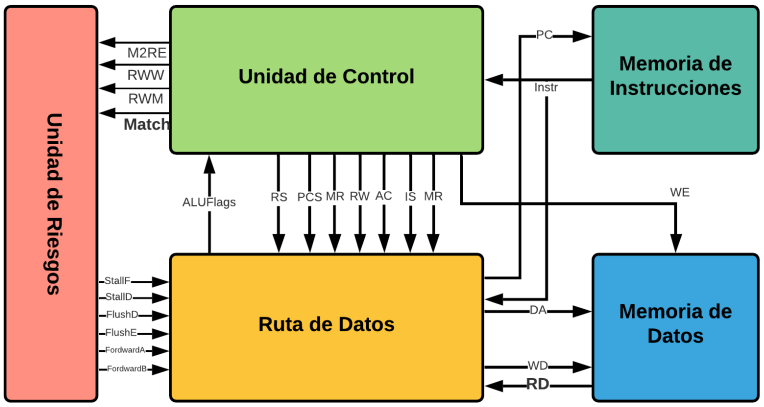



Figura 8. *Diagrama de segundo nivel de procesador.*

El diagrama de tercer nivel amplía la ruta de datos y presenta los módulos de los pipelines. La Figura 4 presenta las etapas de ambos pipelines. Se puede observar que la etapa de fetch es común y a partir de allí, se bifurcan los pipelines. En la etapa vectorial se cuenta con una unidad de registros vectoriales que permite manipular vectores de 8 datos de 32 bits y estos se envían a la ALU vectorial, que realiza una operación sobre el vector, un elemento en cada lane y en su salida presenta otro vector resultante.