# INSTITUTO POLITÉCNICO NACIONAL

## ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA

SIMULADOR: UNIDAD LÓGICA ARITMÉTICA ASOCIATIVA

TESIS

QUE PARA OBTENER EL TÍTULO DE INGENIERO EN COMUNICACIONES Y ELECTRÓNICA

PRESENTAN

MOISÉS ALAN CHAVARRÍA RAMOS

DANIEL RAMÍREZ MARTÍNEZ

ASESORES

DRA. MARÍA ELENA ACEVEDO MOSQUEDA

DR. MARCO ANTONIO ACEVEDO MOSQUEDA

DR. JESÚS JAIME MORENO ESCOBAR

MÉXICO, D.F.  MARZO 2015

# INSTITUTO POLITÉCNICO NACIONAL

**ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA**
**UNIDAD PROFESIONAL "ADOLFO LÓPEZ MATEOS"**

# REPORTE TECNICO

**QUE PARA OBTENER EL TITULO DE** **INGENIERO EN COMUNICACIONES Y ELECTRÓNICA**

**POR LA OPCIÓN DE TITULACIÓN** **TESIS COLECTIVA Y EXAMEN ORAL INDIVIDUAL**

**DEBERA (N) DESARROLLAR** **C. MOISES ALAN CHAVARRIA RAMOS**
**C. DANIEL RAMIREZ MARTINEZ**

**"SIMULADOR: UNIDAD LÓGICA ARITMÉTICA ASOCIATIVA"**

DEMOSTRAR LA EFECTIVIDAD DE RECUPERACIÓN DE PATRONES DE LAS MEMORIA ASOCIATIVA BIDIRECCIONALES ALFA-BETA A TRAVÉS DE UN SIMULADOR DE UNA UNIDAD LÓGICA ARITMÉTICA ASOCIATIVA.

- ESTUDIAR LOS DIFERENTES MODELOS ASOCIATIVOS
- CONOCER EL ALGORITMO DE LAS MEMORIAS ASOCIATIVAS Y BIDIRECCIONALES ALFA-BETA
- IMPLEMENTAR LAS MEMORIAS ASOCIATIVAS BIDIRECCIONALES ALFA-BETA
- DISEÑAR E IMPLEMENTAR UNA UNIDAD ARITMÉTICA LÓGICA UTILIZANDO MEMORIAS ASOCIATIVAS ALFA-BETA A NIVEL SOFTWARE
- EVALUAR EL RENDIMIENTO DEL MODELO PROPUESTO.

**MÉXICO D.F. A 11 DE ENERO DE 2016**

**A S E S O R E S**

**M. EN C. MARÍA ELENA ACEVEDO MOSQUEDA**

**M. EN C. MARCO ANTONIO ACEVEDO MOSQUEDA**

**DR. JESÚS JAIME MORENO ESCOBAR**

**ING. PATRICIA LORENA RAMÍREZ RANGEL**
**JEFE DEL DEPARTAMENTO DE**
**INGENIERÍA EN COMUNICACIONES Y ELECTRONICA**

# Simulador: Unidad Lógica Aritmética Asociativa


Moisés Alan Chavarría Ramos
Daniel Ramírez Martínez

Escuela Superior de Ingeniería Mecánica y Eléctrica
Departamento de Ingeniería en Comunicaciones y Electrónica

Instituto Politécnico Nacional

TRABAJO TERMINAL
Para obtener el título de

*Ingeniero en Comunicaciones y Electrónica*

3 de Marzo de 2015

| | |
|---|---|
| **Asesor Técnico** | **Dra. María Elena Acevedo Mosqueda**<br>Academia de Computación<br>Departamento Academico de Ingeniería en Comunicaciones y Electrónica<br>Escuela Superior de Ingeniería Mecánica y Eléctrica<br>Instituto Politécnico Nacional |
| Metodológicos | |

| | |
|---|---|
| **Asesor Técnico** | **Dr. Marco Antonio Acevedo Mosqueda**<br>Maestría en Telecomunicaciones<br>Sección de Estudios de Posgrado e Investigación<br>Escuela Superior de Ingeniería Mecánica y Eléctrica<br>Instituto Politécnico Nacional |
| Metodológicos | |

| | |
|---|---|
| **Asesores Metodológicos** | **Dr. Jesús Jaime Moreno Escobar**<br>**M. en C. Genaro Zavala Mejía**<br>Academia de Computación<br>Departamento Academico de Ingeniería en Comunicaciones y Electrónica<br>Escuela Superior de Ingeniería Mecánica y Eléctrica<br>Instituto Politécnico Nacional |

| | |
|---|---|
| **Jurado Evaluador** | **Presidente: Ing. Federico Felipe Durán**<br>**Secretario: M. en C. Rosa Mercado Moreno** |
| Metodológicos | |
| | Academia de Computación<br>Departamento Academico de Ingeniería en Comunicaciones y Electrónica<br>Escuela Superior de Ingeniería Mecánica y Eléctrica<br>Instituto Politécnico Nacional |


This document was typeset by the author using LaTeX $2_\varepsilon$.

The research described in this book was carried out at Escuela Superior de Ingeniería Mecánica y Eléctrica, Instituto Politécnico National of Mexico.



Printed in Mexico

A Mis Padres y Hermanos.

# Agradecimientos

A MIS PADRES Y HERMANOS:

Por ayudarme a cruzar el camino de la superación y a conseguir uno de mis más grandes objetivos.

Porque creyeron en mí y porque me sacaron adelante, dándome ejemplos dignos de superación y entrega, porque en gran parte gracias a ustedes, hoy puedo ver alcanzada mi meta, ya que siempre estuvieron impulsándome en los momentos más difíciles de mi carrera, y porque el orgullo que sienten por mí, fue lo que me hizo ir hasta el final. Va por ustedes, por lo que valen, porque admiro su fortaleza y por lo que han hecho de mí.

A MIS ABUELOS,TÍOS Y TÍAS:

Por apoyarme en éste proceso y que siempre tuvieron una palabra de apoyo y aliento para mí durante mis estudios.

Gracias por haber fomentado en mí el deseo de superación y el anhelo de triunfo en la vida.

A MIS ASESORES:

Gracias por el apoyo, paciencia y tiempo que nos brindaron en la realización de este trabajo.

Mil palabras no bastarían para agradecerles su apoyo, su comprensión y sus consejos en los momentos difíciles. A todos, espero no defraudarlos y contar siempre con su valioso apoyo, sincero e incondicional.

*Gracias por todo.*

Moisés Alan Chavarría Ramos

# Agradecimientos

Con el gusto de agradecer el apoyo recibido a lo largo de mi formación profesional, personal y académica dedico este trabajo a mi familia.

Daniel Ramirez Martínez

# Resumen

En este trabajo se presenta un caso de estudio de las Memorias Bidireccionales Alfa - Beta para demostrar su efectividad en la recuperación de patrones, y se propone un simulador de una unidad lógica aritmética orientada a los modelos asociativos para ratificar la certeza de su funcionamiento.

El simulador aritmético lógico constará de tres memorias asociativas, cada una se encargará de realizar las operaciones aritméticas básicas de suma, resta y comparaciones, esto con el fin de comprobar las fases de aprendizaje y recuperación de este tipo de memoria asociativa.

Las memorias de suma y resta, se implementarán para realizar las operaciones aritméticas de multiplicación y división, con el fin de cubrir más operaciones dentro del modelo propuesto de ALU.

Debido a que las memorias asociativas bidireccionales $\alpha$-$\beta$ se caracterizan por una recuperación correcta, se garantiza que no habrá fallo en el resultado de recuperación de los patrones programados en la fase de aprendizaje. Esto queda comprobado con este trabajo.

# Abstract

This paper presents a case study of the alpha-beta memories to demonstrate their effectiveness in recovering patterns also, and a simulator of an arithmetic logic unit oriented associative models to ratify certain operation is proposed.

The arithmetic logic simulator consists of three associative memories, each of undertaking the basic arithmetic operations of addition, subtraction and comparisons, this in order to check learning retrieval phases of this type of associative memory.

Memories addition and subtraction, are implemented to perform the arithmetic operations of multiplication and división, in order to have more operations within the model proposed Arithmetic Logic Unit.

Since the associative memories $\alpha$-$\beta$ are characterized by a successful recovery, it is guaranteed that they won´t recover wrong programmed patterns in learning phase.

# Glosario

| | |
|---|---|
| *ALU* | *Arithmetic Logic Unit*, Unidad Aritmética Lógica. |
| $ALU_A$ | *Arithmetic Logic Unit Associative*, Unidad Aritmética Lógica Asociativa. |
| $ALU_C$ | *Arithmetic Logic Unit Classic*, Unidad Aritmética Lógica Clásica. |
| Ambigüedad | Posibilidad de que algo pueda entenderse de varios modos o de que admita distintas interpretaciones. |
| Aserción | Declaración de que una cosa es cierta. |
| *BAM* | *Bidirectional Associative Memories*,Memorias Asociativas Bidireccionales. |
| *GUI* | *Graphical User Interface*, Interfaz Gráfica de Usuario . |
| Hito | Acontecimiento muy importante y significativo en el desarrollo de un proceso. |
| Miríada | Infinidad, sinfín, sinnúmero. |
| Pionero | Se aplica a la persona que realiza los primeros descubrimientos o los primeros trabajos en una actividad determinada. |
| *UML* | *Unified Modeling Language*, Lenguaje Unificado de Modelado. |

# Contenido

**Lista de Figuras** **xiii**

**Lista de Tablas** **xv**

**1 Introducción** **1**
1.1 Organización del documento . . . . . 1
1.2 Planteamiento del problema . . . . . 1
1.3 Justificación . . . . . 2
1.4 Hipótesis . . . . . 2
1.5 Objetivos . . . . . 3
1.5.1 General . . . . . 3
1.5.2 Particulares . . . . . 3
1.6 Alcance del proyecto . . . . . 3
1.7 Tesis del Proyecto . . . . . 3

**2 Historia de las Memorias Asociativas y Trabajos Relacionados** **5**
2.1 Antecedentes . . . . . 5
2.2 Aplicaciones . . . . . 7
2.3 Memorias Asociativas . . . . . 8
2.3.1 Lernmatrix de Steinbuch . . . . . 9
2.3.2 Correlograph de Willshaw, Buneman y Longuet-Higgins . . 10
2.3.3 Linear Associator de Anderson-Kohonen . . . . . 10
2.3.4 La memoria asociativa Hopfield . . . . . 11
2.3.5 Memorias Asociativas Morfológicas . . . . . 12
2.3.5.1 Memorias Heteroasociativas Morfológicas . . . . 13
2.3.5.2 Memorias Autoasociativas Morfológicas . . . . . 13
2.3.6 Memorias Asociativas Alfa-Beta . . . . . 14
2.3.7 Memorias Asociativas Media . . . . . 15

**3 Algoritmo de las Memorias Alfa - Beta** **17**
3.1 Conceptos Básicos . . . . . 17
3.2 Descripción de las Memorias Asociativas Bidireccionales Alfa-Beta 19
3.3 Fundamento Teórico de las Etapas 1 y 3 . . . . . 24
3.4 Fundamento Teórico de las Etapas 2 y 4 . . . . . 25
3.5 Algoritmo . . . . . 26
3.5.1 Algoritmo de las Etapas 1 y 2 . . . . . 26
3.5.2 Algoritmo de las Etapas 3 y 4 . . . . . 29

3.5.3 Ejemplo ilustrativo de la Memoria Asociativa Bidireccional Alfa-Beta . . . . . 31

**4 Implementación de la Memoria en la ALU Asociativa 35**
4.1 Generación de la Memoria Asociativa Suma . . . . . 36
4.1.1 Transformación . . . . . 37
4.1.2 Transformada vectorial de expansión . . . . . 39
4.1.3 Memorias máx y min . . . . . 39
4.1.4 Linear Associator Modificado . . . . . 41
4.2 Generación de la Memoria Asociativa Resta . . . . . 42
4.3 Implementación de la Memoria Asociativa Suma . . . . . 45
4.4 Implementación de la Memoria Asociativa Resta . . . . . 48
4.5 Multiplicación . . . . . 51
4.6 División . . . . . 53
4.7 Simulador Aritmético Lógico . . . . . 55

**5 Resultados Experimentales 61**

**6 Conclusiones 65**
6.1 Conclusiones Finales . . . . . 65
6.2 Contribuciones . . . . . 65
6.3 Trabajo Futuro . . . . . 65

**A Código de implementación de la memoria suma. 67**

**B Código de implementación de la memoria resta. 73**

**C Código de implementación de la multiplicación y división 79**

**Referencias 83**

# Lista de Figuras


1.1 Diagrama a bloques de una memoria asociativa . . . . . 2

3.1 Diagrama a bloques de una memoria asociativa . . . . . 17
3.2 Esquema general de una Memoria Asociativa Bidireccional . . . . 19
3.3 Modelo de Kosko . . . . . 20
3.4 Etapas de una memoria asociativa bidireccional Alfa-Beta . . . . 20
3.5 Proceso a realizar en el sentido x $\rightarrow$ y . . . . . 22
3.6 Proceso en el sentido y $\rightarrow$ x . . . . . 23
3.7 Algoritmo de la fase de aprendizaje de las memorias $\alpha$-$\beta$ BAM . . 27
3.8 Algoritmo de la fase de recuperación de las memorias $\alpha$-$\beta$ BAM . 28
3.9 Algoritmo de la fase de aprendizaje de las memorias $\alpha$-$\beta$ BAM para la recuperación de patrones . . . . . 29
3.10 Algoritmo de la fase de recuperación de las memorias $\alpha$-$\beta$ BAM para la recuperación de patrones . . . . . 30

4.1 UML . . . . . 35
4.2 Patrones entrada y salida . . . . . 37
4.3 Clase Base Suma . . . . . 37
4.4 Patrones de entrada transformados . . . . . 38
4.5 Patrones de salida transformados . . . . . 39
4.6 Vectores one-hot y zero-hot del patrón . . . . . 39
4.7 Memoria Autoasociativa max . . . . . 40
4.8 Memoria Autoasociativa min . . . . . 41
4.9 Linear Associator Modificado . . . . . 42
4.10 Clase Base Resta. . . . . 43
4.11 Patrones de entrada transformados . . . . . 44
4.12 Patrones de salida Transformados. . . . . 44
4.13 Diagrama de flujo para la implementacion de la Memoria Suma. . 45
4.14 Diagrama de flujo para la implementacion de la Memoria resta. . 49
4.15 Diagrama de flujo del metodo multiplicacion . . . . . 52
4.16 Diagrama de flujo del metodo division . . . . . 54
4.17 Creación de un proyecto en C♯. . . . . 56
4.18 Ventana de un proyecto de C♯. . . . . 57
4.19 Ventana de Propiedades. . . . . 57
4.20 Acondicionamiento del ambiente de programación en visual C♯. . 58
4.21 Inserción de un boton al formulario . . . . . 58
4.22 Interfaz Grafica de Usuario de la $ALU_A$ . . . . . 59

# Lista de Tablas


2.1 Descripción del operador alfa . . . . . 14
2.2 Descripción del operador beta . . . . . 14

4.1 Resultados de Operaciones de Sumas. . . . . 36
4.2 Números binarios . . . . . 38
4.3 Resultados de Operaciones de Resta . . . . . 43
4.4 Resultados de Operaciones de Resta . . . . . 44

5.1 Operaciones aplicadas al Simulador de la $ALU_A$ . . . . . 61
5.2 Excluyendo Memoria mín . . . . . 62
5.3 Operaciones Memoria mín . . . . . 62
5.4 Resultados Excluyendo Memoria mín . . . . . 63
5.5 Excluyendo Memoria máx . . . . . 63
5.6 Resultados Excluyendo Memoria máx . . . . . 64

# Capítulo 1

# Introducción

## 1.1 Organización del documento

En este capítulo se presenta la identificación y justificación del problema, hipótesis planteada, los objetivos del trabajo de tesis. El resto del documento se organiza de la siguiente forma:

En el capítulo 2 se presenta el estado del arte de las memorias asociativas, así como los modelos más representativos que sirven como marco de referencia del modelo de memoria asociativa bidireccional alfa-beta.

En el capítulo 3 se describe detalladamente las memorias asociativas bidireccionales alfa-beta, en las cuales se basa el funcionamiento del modelo de ALU propuesta.

En el siguiente capítulo se muestra el desarrollo de la aplicación del simulador implementando las memorias asociativas.

Finalmente en los últimos dos capítulos se verán los resultados, conclusiones y trabajo a futuro para este trabajo

## 1.2 Planteamiento del problema

Actualmente se han implementado memorias asociativas Alfa-Beta en campos como, medicina, arquitectura de computadoras, etc; en el área de medicina ha contribuido con el trabajo "Clasification of cáncer recurrence with Alpha - Beta BAM" (1), "Associative models for storing and retrieving concept lattices" (2), se han usado para diagnosticar el Parkinson tal como lo muestra el trabajo "Associative memory approach for diagnosis of Parkinson's disease" (3), del mismo modo se ha implementado los modelos asociativos para predicción subcelular de proteínas como se ve en el trabajo " Modelos asociativos para predicción de la localización subcelular de proteínas" (4);

Se han implementado memorias asociativas ya que ofrecen un algoritmo de aprendizaje y recuperación datos muy eficiente y que se puede implementar para diversas áreas de trabajo o investigación.

En el campo de la arquitectura de las computadoras, hasta nuestros días se implementa el modelo clásico de la unidad aritmética lógica (ALU por sus siglas en inglés Arithmetic Logic Unit). Dicha unidad realiza las operaciones bit a bit, es decir, la ALU ejecuta el procesamiento de datos utilizando el sistema numérico binario, debido a que es la representación más simple para realizar operaciones aritméticas y lógicas.

Sin embargo, históricamente se ha tratado de utilizar otros sistemas numéricos, tales como el sistema numérico decimal, pero éste no perduró, debido a la simplicidad que implica el sistema binario en la ejecución de operaciones.

Para comprobar la eficiencia de la capacidad de aprendizaje y recuperación de la memoria Alfa-Beta, se propone una aplicación para realizar una ALU Decimal-Asociativa basada en memorias asociativas Alfa-Beta.

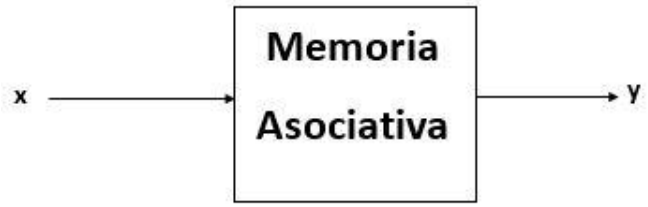



**Figura 1.1:** Diagrama a bloques de una memoria asociativa.

## 1.3 Justificación

Dada la baja capacidad de recuperación de otras memorias, en este trabajo se implementan las memorias asociativas Bidireccionales Alfa - Beta ya que estas memorias no tienen factor de olvido, en otras palabras todos los patrones de entrada con su correspondiente patrón de salida que aprenden, son recuperados al ser requeridos.

Para este trabajo sólo se usarán estas memorias en una sola dirección ya que se desea obtener el patrón asociado a la operación indicada.

## 1.4 Hipótesis

La Memoria Asociativa Alfa - Beta recupera todos lo aprendido, con lo cual el simulador de la unidad lógica aritmética será capaz de entregar el resultado de cualquier operación que se le solicite.

## 1.5 Objetivos

### 1.5.1 General

Demostrar la efectividad de recuperación de patrones de las Memoria Asociativa Bidireccionales Alfa - Beta a través de un simulador de una unidad aritmética lógica asociativa.

### 1.5.2 Particulares

- Estudiar los diferentes modelos asociativos.
- Conocer el algoritmo de las memorias asociativas y Bidireccionales Alfa-Beta.
- Implementar las memorias asociativas bidireccionales Alfa-Beta.
- Diseñar e implementar una unidad aritmética lógica asociativa utilizando memorias asociativas Alfa-Beta a nivel software.
- Evaluar el rendimiento del modelo propuesto.

## 1.6 Alcance del proyecto

Realizar una $\text{ALU}_A$, implementando Memorias Asociativas Alfa - Beta utilizando un software.

## 1.7 Tesis del Proyecto

Realizar una aplicación a nivel software de una ALU Asociativa utilizando memorias asociativas Alfa-Beta.

# 1. INTRODUCCIÓN

# Capítulo 2

# Historia de las Memorias Asociativas y Trabajos Relacionados

## 2.1 Antecedentes

Una investigación de la bibliografía que se ha generado a través de los años en relación al tema de las memorias asociativas, nos permite darnos cuenta de que antes de la década de los setenta la producción de trabajos en el área de memorias asociativas era poco, casi nulo. Importantes excepciones fueron la de Lernmatrix, surgida a inicio de la década de los sesenta.

El primer modelo matemático de memoria asociativa del cual se tiene noticia es la Lernmatrix de Steinbuch, desarrollada en 1961 por el científico alemán Karl Steinbuch, quien publicó su artículo en una revista llamada Kybernetik (5), y a pesar de la importancia de su modelo, el trabajo pasó casi inadvertido.

Ocho años después de la Lernmatrix, en 1969, tres científicos escoceses (Willshaw, Buneman and Longuet-Higgins) crearon el Correlograph, el cual es un dispositivo óptico elemental capaz de funcionar como una memoria asociativa. En palabras de los autores "el sistema es tan simple, que podría ser construido en cualquier laboratorio escolar de física elemental" (6).

En el año de 1972 se dieron grandes avances de parte de los investigadores pioneros en memorias asociativas. Para corroborar esta aserción, baste dar una vistazo a la siguiente información: apoyado por la UCLA, James A. Anderson prendió la mecha con su Interactive Memory (7); en abril, Teuvo Kohonen, a la sazón profesor de la Helsinki University of Technology, presentó ante el mundo sus Correlation Matrix Memories (8); tres meses después, Kaoru Nakano de la University of Tokyo, dio a conocer su Associatron (9); y en el ocaso del año, Shun-Ichi Amari, profesor de la University of Tokyo y uno de los más prolíficos escritores de artículos científicos hasta nuestros días, publicó un trabajo teórico donde continuaba con sus investigaciones sobre las Self-Organizing Nets of Threshold Elements (10), tópico cuyo estudio había iniciado un par de años atrás. Este trabajo de

Amari constituye una valiosa muestra del estilo teórico característico de este personaje y representa, históricamente, un importante antecedente para la creación de lo que una década más tarde se convertiría en el modelo de memoria asociativa por antonomasia: la memoria Hopfield. Los trabajos de Anderson y Kohonen, y en cierta medida el de Nakano, dieron lugar al modelo que actualmente se conoce con el nombre genérico de Linear Associator.

Si 1972 fue el año de los pioneros en el área de las memorias asociativas, 1982 fue el año del científico estadunidense John J. Hopfield. Su artículo de ese año (11), publicado por la prestigiosa y respetada National Academy of Sciences (en sus Proceedings), impactó positivamante y trajo a la palestra internacional su memoria asociativa. Dos años después, publicaría su segundo artículo, donde presentaba una extensión de su modelo original (12).

El trabajo de Hopfield ha tenido un gran impacto en las áreas de memorias asociativas.

Los trabajos de Hopfield causaron excitación en el mundo de las memorias asociativas, de tal modo que numerosos científicos, gracias a Hopfield se interesaron en estos temas y se generó una actividad importante. Así, en la segunda mitad de la década de los ochenta del siglo XX aparecieron investigadores que tomaron los modelos clásicos, los modificaron o extendieron y dieron lugar a nuevos tipos de memorias asociativas, cuya importancia radica en que son consecuencia del trabajo de los grandes hombres involucrados en la concepción y desarrollo de los modelos clásicos.

Entre la miríada de aportaciones e innovaciones en el campo de las memorias asociativas, después del espectacular éxito de la memoria Hopfied de 1982, no sucedió nada realmente trascendente hasta 1998, cuando aparecieron las memorias asociativas morfológicas.

La diferencia fundamental entre estas memorias y las memorias asociativas clásicas como el Linear Associator y la Memoria Hopfield, es que mientras éstas basan su operación en la suma y multiplicación usuales, las memorias morfológicas se basan en las operaciones morfológicas de dilatación y erosión. Estas memorias rompieron el esquema utilizado a través de los años en los modelos de memorias asociativas clásicas, que utilizan operaciones convencionales entre vectores y matrices para la fase de aprendizaje, y suma de productos para recuperar patrones.

Las memorias asociativas morfológicas usan máximos o mínimos de sumas para la fase de aprendizaje, y máximos o mínimos de sumas para la fase de recuperación (13, 14). Con este nuevo esquema, superaron claramente a las memorias asociativas clásicas.

La aparición, desarrollo, aplicaciones y consolidación de las memorias asociativas morfológicas en 1998 marcó un camino en el campo de las memorias asociativas, porque superaron en prácticamente todos los aspectos de interés, a los modelos conocidos (15, 16); con objeto de proporcionar una muestra de la magnitud de este salto cualitativo, vale la pena mencionar que al funcionar en uno de los posibles modos (autoasociativo), las memorias asociativas morfológicas tienen respuesta perfecta y capacidad infinita de aprendizaje y almacenamiento (16, 17). Un hecho que puede ser importante en el contexto de las potencialidades de la tecnología de punta a niveles de seguridad nacional, es que investigadores del Air Force Research Lab de los Estados Unidos implementaron las memorias asociati-

vas morfológicas en hardware VLSI, y actualmente se usan de manera cotidiana (18).

El contenido del párrafo anterior es la base sobre la que se forjó la motivación para llevar a cabo el trabajo de tesis (12) donde se crearon y desarrollaron las memorias asociativas Alfa-Beta: la necesidad de realizar actividades de investigación cuyo objetivo fundamental sea encontrar un modelo alternativo a las memorias asociativas morfológicas, el cual se fundamente en bases teóricas ajenas a las operaciones morfológicas de dilatación y erosión. Las nuevas memorias asociativas deben poseer, al menos, características similares a las morfológicas en cuanto a capacidad, eficiencia en respuesta e inmunidad al ruido. He ahí las expectativas.

En efecto, las herramientas matemáticas del nuevo modelo de memorias asociativas Alfa-Beta incluyen dos operaciones binarias inventadas, cuyos operadores fueron nombrados arbitrariamente con los dos primeros signos del alfabeto griego: Alfa y Beta. Las memorias asociativas Alfa-Beta constituyen el modelo con mejor rendimiento en la actualidad (19).

## 2.2 Aplicaciones

Existen diversas aplicaciones que utilizan memorias asociativas y a continuación se verán algunas de ellas.

El año 2003 marca un hito en el área de Reconocimiento de Patrones, dado que Raúl Santiago Montero, en su tesis de maestría en ciencias de la computación, introdujo un nuevo algoritmo clasificador de patrones: el Clasificador Híbrido Asociativo con Traslación (CHAT). La relevancia de este algoritmo radica en ser el fundador del Enfoque Asociativo de Reconocimiento de Patrones, al mostrar teórica y experimentalmente que las memorias asociativas pueden ser utilizadas para otras tareas de Reconocimiento de Patrones además de la recuperación (20, 21).

María Elena Acevedo Mosqueda desarrolló las Memorias Asociativas Bidireccionales Alfa-Beta (22, 23); este modelo se inspira en las Bidirectional Associative Memories (BAM) de Kosko, el cual es un famoso modelo basado en la Memoria Hopfield. El surgimiento de las BAM Alfa-Beta marca un salto cualitativo en el desarrollo de las memorias asociativas bidireccionales puesto que, mientras las BAM de Kosko sólo recupera menos del 15 porciento de los patrones, las BAM Alfa-Beta son capaces de recuperar todo el conjunto de entrenamiento completo, sin importar su dimensión ni su cardinalidad. Este modelo supera ampliamente a todos los desarrollos presentes en la literatura hasta ese momento (2006). Para mostrar sus bondades las memorias asociativas bidireccionales Alfa-Beta fueron aplicadas a un traductor Español-Inglés / Inglés-Español (24).

En 2006, se publica una extensión a las memorias asociativas Alfa-Beta que les permite operar imágenes en escala de grises (25); también, una extensión al algoritmo de recuperación de la Lernmatrix que le permite a este modelo ser aplicado a la tarea de clasificación de patrones (26). Finalmente, Mario Aldape Pérez presenta algunos avances de su trabajo en Feature Selection, al aplicar los modelos

asociativos Alfa-Beta de manera ingeniosa a este interesante campo de investigación (27, 28).

2007 fue otro año muy productivo para los modelos asociativos Alfa-Beta. Primero, continuando con el desarrollo de las BAM Alfa-Beta, se publicaron varios artículos al respecto en diversas revistas internacionales (29, 30). En dichos artículos se profundiza en el desarrollo teórico y en las aplicaciones de este modelo.

Por otro lado, Mario Aldape Pérez presenta en (31) una versión mejorada de sus trabajos anteriores, continuando con la incursión de los modelos Alfa-Beta en el área de Feature Selection. En esta ocasión introduce una memoria asociativa optimizada para la tarea de Feature Selection.

María Elena Cruz Meza introduce en su tesis de maestría una extensión a las memorias asociativas Alfa-Beta que les permite operar con imágenes en color; en particular, con aquellas imágenes codificadas en el espacio RGB (32, 33).

Mario Aldape Pérez presenta en su tesis de maestría una implementación en Hardware de las memorias asociativas Alfa-Beta. Al tratar de alimentar una base de datos conocida a su implementación con fines experimentales, desarrolla un nuevo enfoque de Feature Selection: la aplicación de los modelos Alfa-Beta a esta tarea (34, 35).

Israel Román Godínez aplica los modelos Alfa-Beta a la solución de problemas del área de Bioinformática. Durante el desarrollo de su trabajo, introduce una extensión al modelo original que le permite a las memorias heteroasociativas Alfa-Beta recuperar el conjunto fundamental completo (36). Basándose en lo anterior, desarrolla un nuevo modelo: las multimemorias Alfa-Beta, con las que resuelve su problema original de Bioinformática(37).

Otros dos modelos novedosos de reconocimiento de patrones, basados e inspirados en las memorias asociativas Alfa-Beta, son el clasificador Gama y CAINN. El primero fue desarrollado por Itzamá López Yáñez como un algoritmo general de clasificación de patrones. Su base es la operación Gama de similitud, que está basada a su vez en las operaciones Alfa y Beta; exhibe un desempeño competitivo en varias bases de datos conocidas, superando a otros clasificadores en algunas de ellas (38).

El segundo fue presentado por Amadeo José Argüelles Cruz como un nuevo modelo de red neuronal sin pesos. Al comparar su desempeño en varias bases de datos con ADAM, el modelo de red neuronal sin pesos por excelencia, éste es superado ampliamente por CAINN (39, 40).

## 2.3 Memorias Asociativas

A continuación, en esta sección se hará un breve recorrido por los modelos de memorias asociativas, con objeto de establecer el marco de referencia en el que surgieron las memorias asociativas bidireccionales.

Las memorias asociativas que se presentarán en esta sección, son los modelos más representativos que sirvieron de base para la creación de modelos matemáticos que sustentan el diseño y operación de memorias asociativas más complejas. Para cada modelo se describe su fase de aprendizaje y su fase de recuperación.

Se incluyen cuatro modelos clásicos basados en el anillo de los números racionales con las operaciones de multiplicación y adición: Lernmatrix, Correlograph, Linear Associator y Memoria Hopfield, además de tres modelos basados en paradigmas diferentes a la suma de productos, a saber: memorias asociativas Morfológicas, memorias asociativas Alfa-Beta y memorias asociativas Media.

### 2.3.1 Lernmatrix de Steinbuch

Karl Steinbuch fue uno de los primeros investigadores en desarrollar un método para codificar información en arreglos cuadriculados conocidos como crossbar (13). La importancia de la Lernmatrix (8, 14) se evidencia en una afirmación que hace Kohonen (11) en su artículo de 1972, donde apunta que las matrices de correlación, base fundamental de su innovador trabajo, vinieron a sustituir a la Lernmatrix de Steinbuch.

La Lernmatrix es una memoria heteroasociativa que puede funcionar como un clasificador de patrones binarios si se escogen adecuadamente los patrones de salida; es un sistema de entrada y salida que al operar acepta como entrada un patrón binario $\boldsymbol{X}^{\mu} \epsilon \boldsymbol{A}^{n}{}_{,A=\{0,1\}}$ y produce como salida la clase $\boldsymbol{Y}^{\mu} \epsilon \boldsymbol{A}^{P}$ que le corresponde (de entre p clases diferentes), codificada ésta con un método que en la literatura se le ha llamado one-hot (15). El método funciona así: para representar la clase k $\epsilon\{1, 2, ..., p\}$, se asignan a las componentes del vector de salida $y_j^{\mu}$ los siguientes valores: $y_k^{\mu}$ y $y_j^{\mu}$ para j = 1, 2, ..., k-1, k+1,...,p.

**Algoritmo de la Lernmatrix**

**Fase de Aprendizaje**

Se genera el esquema (crossbar) al incorporar la pareja de patrones de entrenamiento $(\boldsymbol{x}^{\mu},\boldsymbol{y}^{\mu}) \epsilon A^n$ x $A^P$. Cada uno de los componentes $m_{ij}$ de $\mathbf{M}$, la Lernmatrix de Steinbuch, tiene valor cero al inicio, y se actualiza de acuerdo con la regla $m_{ij}+ \Delta\, m_{ij}$ , donde:

$$\Delta m_{ij} = \begin{cases} +\epsilon & \text{si } x_j^{\mu} = 1 = y_i^{\mu} \\ -\epsilon & \text{si } x_j^{\mu} = 0 \text{ y } y_i^{\mu} = 1 \\ 0 & \text{en otro caso} \end{cases}$$

donde una constante positiva escogida previamente: es usual que $\varepsilon$ es igual a 1.

**Fase de Recuperación**

La i-ésima coordenada $y_i^{\omega}$ del vector de clase $\mathbf{y}^{\omega} \epsilon A^P$ se obtiene como lo indica la siguiente expresión, donde $\vee$ es el operador máximo:

$$y_i^{\omega} = \begin{cases} 1 & \text{si } \sum_{j=1}^{n} m_{ij}..x_j^{\omega} = \bigvee_{h=1}^{P} [\sum_{j=1}^{n} m_{hj}..x_j^{\omega}] \\ 0 & \text{en otro caso} \end{cases}$$

### 2.3.2 Correlograph de Willshaw, Buneman y Longuet-Higgins

El correlograph es un dispositivo óptico elemental capaz de funcionar como una memoria asociativa (9). En palabras de los autores "el sistema es tan simple, que podría ser construido en cualquier laboratorio escolar de física elemental".

**Algoritmo del Correlograph**

**Fase de Aprendizaje**

La red asociativa se genera al incorporar la pareja de patrones de entrenamiento $(\boldsymbol{x}^{\mu},\boldsymbol{y}^{\mu})\epsilon\ A^n$ x $A^m$. Cada uno de los componentes $m_{ij}$ de la red asociativa **M** tiene valor cero al inicio, y se actualiza de acuerdo con la regla:

$$m_{ij} = \begin{cases} 1 & \text{si } y_{ij} = 1 = x_j^u \\ \text{valor anterior} & \text{en otro caso} \end{cases}$$

**Fase de Recuperación**

Se le presenta a la red asociativa **M** un vector de entrada $x^{\omega}\ \epsilon\ A^n\ A^n$. Se realiza el producto de la matriz **M** por el vector $x^{\omega}$ y se ejecuta una operación de umbralizado, de acuerdo con la siguiente expresión:

$$y_i^{\omega} = \begin{cases} 1 & \text{si } \sum_{j=1}^{n} m_{ij} x_j^{\omega} \geq u \\ 0 & \text{en otro caso} \end{cases}$$

donde u es el valor de umbral. Una estimación aproximada del valor de umbral u se puede lograr con la ayuda de un número indicador mencionado en el artículo (9) de Willshaw et al. de 1969: $\log_2$n

### 2.3.3 Linear Associator de Anderson-Kohonen

El Linear Associator tiene su origen en los trabajos pioneros de 1972 publicados por Anderson y Kohonen (10, 11).

Para presentar el Linear Associator consideremos de nuevo el conjunto fundamental:

$$\{(\mathbf{x}^u,\mathbf{y}^u)\ —\mu= 1, 2, ..., \text{p}\} \text{ con A} = \{0, 1\},\ \mathbf{x}^u\ \epsilon\ A^n \text{ y } y^u\ \epsilon\ A^m$$

**Algoritmo del Linear Associator**

**Fase de Aprendizaje**

1) Para cada una de las p asociaciones $(x^{\mu}\ ,\ y^{\mu})$ se encuentra la matriz $y^{\mu} \cdot (\text{x}^{\mu})^t$ de dimensiones m x n

2) Se suman la p matrices para obtener la memoria

$$\mathbf{M}=\sum_{\mu=1}^{p} y^{\mu} \cdot (x^{\mu})^{t}=[m_{ij}]_{mxn}$$

de manera que la ij-ésima componente de la memoria M se expresa así:

$$m_{ij}=\sum_{\mu=1}^{p} y_i^{\mu}\ x_j^{\mu}$$

**Fase de Recuperación**

Esta fase consiste en presentarle a la memoria un patrón de entrada $x^{\omega}$ , donde $_{\omega\epsilon\{1,2,...,p\}}$ y realizar la operación

$$\mathrm{M}\cdot\ x^{\omega}=[\sum_{\mu=1}^{p} y^{\mu} \cdot (x^{\mu})^{t}] \cdot x^{\omega}$$

### 2.3.4 La memoria asociativa Hopfield

El artículo de John J. Hopfield de 1982, publicado por la prestigiosa y respetada National Academy of Sciences (en sus Proceedings), impactó positivamente y trajo a la palestra internacional su famosa memoria asociativa (7).

En el modelo que originalmente propuso Hopfield, cada neurona $x_i$ tiene dos posibles estados, a la manera de las neuronas de McCulloch-Pitts: $x_i = 0$ y $x_i = 1$; sin embargo, Hopfield observa que, para un nivel dado de exactitud en la recuperación de patrones, la capacidad de almacenamiento de información de la memoria se puede incrementar por un factor de 2, si se escogen como posibles estados de las neuronas los valores $x_i$ = -1 y $x_i = 1$ en lugar de los valores originales $x_i = 0$ y $x_i = 1$.

Al utilizar el conjunto {-1, 1} y el valor de umbral cero, la fase de aprendizaje para la memoria Hopfield será similar, en cierta forma, a la fase de aprendizaje del Linear Associator. La intensidad de la fuerza de conexión de la neurona $x_i$ a la neurona $x_j$ se representa por el valor de $m_{ij}$, y se considera que hay simetría, es decir, $m_{ij} = m_{ji}$. Si $x_i$ no está conectada con $x_j$ entonces $m_{ij} = 0$; en particular, no hay conexiones recurrentes de una neurona a sí misma, lo cual significa que $m_{ij} = 0$. El estado instantáneo del sistema está completamente especificado por el vector columna de dimensión n cuyas coordenadas son los valores de las n neuronas.

La memoria Hopfield es autoasociativa, simétrica, con ceros en la diagonal principal. En virtud de que la memoria es autoasociativa, el conjunto fundamental para la memoria Hopfield es $\{(x^{\mu}\ ,\ x^{\mu}\ ) \mid \mu = 1, 2, ..., \mathrm{p}\}$ con $x^{\mu} \epsilon\ A^{n}$ y A = {-1, 1}

**Algoritmo Hopfield**

**Fase de Aprendizaje**

La fase de aprendizaje para la memoria Hopfield es similar a la fase de aprendizaje del Linear Associator, con una ligera diferencia relacionada con la diagonal principal en ceros, como se muestra en la siguiente regla para obtener la ij-ésima componente de la memoria Hopfield **M**:

$$m_{ij} = \begin{cases} \sum_{\mu=1}^{p} x_i^{\mu} x_j^{\mu} & \text{si } i \neq j \\ 0 & \text{si } i = j \end{cases}$$

**Fase de Recuperación**

Si se le presenta un patrón de entrada **x** a la memoria Hopfield, ésta cambiará su estado con el tiempo, de modo que cada neurona $x_i$ ajuste su valor de acuerdo con el resultado que arroje la comparación de la cantidad $\sum_{j=1}^{n} m_{ij}\ x_j$ con un valor de umbral, el cual normalmente se coloca en cero.

Se representa el estado de la memoria Hopfield en el tiempo t por x(t); entonces $x_i$(t) representa el valor de la neurona $x_i$ en el tiempo t y $x_i$(t+1) el valor de $x_i$ en el tiempo siguiente (t+1).

Dado un vector columna de entrada $\tilde{x}$, la fase de recuperación consta de tres pasos:

1) Para t = 0, se hace $\mathbf{x}(t)= \tilde{\mathbf{x}}$ ; es decir, , $x_i(0)= \tilde{x}_{\ i}$,$\forall$ i $\epsilon$ $\{1, 2, 3, ..., n\}$
2) $\forall i \epsilon$ {1,2,3,...,n} se calcula $x_i$(t+1) de acuerdo con la condición siguiente:

$$x_i(t+1) = \begin{cases} 1 & \text{si } \sum_{j=1}^{n} m_{ij}x_j(t) > 0 \\ x_i(t) & si\ \sum_{j=1}^{n} m_{ij}x_j(t) = 0 \\ -1 & si\ \sum_{j=1}^{n} m_{ij}x_j(t) < 0 \end{cases}$$

3) Se compara $x_i$(t+1) con $x_i$(t) $\forall$ i $\epsilon$ {1, 2, 3,...,n}. Si x(t+1) = x(t) el proceso termina y el vector recuperado es x(0)=$\tilde{x}$ . De otro modo, el proceso continúa de la siguiente manera: los pasos 2 y 3 se iteran tantas veces como sea necesario hasta llegar a un valor t =$\tau$ para el cual $x_i$(t+1) = $x_i(\tau$+1) =$x_i(\tau)\forall$ i $\epsilon$ {1, 2, 3,...,n}; el proceso termina y el patrón recuperado es x($\tau$ ).

En el artículo original de 1982, Hopfield había estimado empíricamente que su memoria tenía una capacidad de recuperar 0.15n patrones, y en el trabajo de Abu-Mostafa and St. Jacques (16) se estableció formalmente que una cota superior para el número de vectores de estado arbitrarios estables en una memoria Hopfield es n.

### 2.3.5 Memorias Asociativas Morfológicas

La diferencia fundamental entre las memorias asociativas clásicas (Lernmatrix, Correlograph, Linear Associator y Memoria Asociativa Hopfield) y las memorias asociativas morfológicas radica en los fundamentos operacionales de éstas últimas, que son las operaciones morfológicas de dilatación y erosión; el nombre de las memorias asociativas morfológicas está inspirado precisamente en estas dos operaciones básicas. Estas memorias rompieron con el esquema utilizado a través de los años en los modelos de memorias asociativas clásicas, que utilizan operaciones convencionales entre vectores y matrices para la fase de aprendizaje y suma de productos para la recuperación de patrones. Las memorias asociativas morfológicas cambian los productos por sumas y las sumas por máximos o mínimos en ambas fases, tanto de aprendizaje como de recuperación (17).

Hay dos tipos de memorias asociativas morfológicas: las memorias max, simbolizadas con **M**, y las memorias min, cuyo símbolo es **W**; en cada uno de los dos tipos, las memorias pueden funcionar en ambos modos: heteroasociativo y autoasociativo.

Se definen dos nuevos productos matriciales:

El producto máximo entre **D** y **H**, denotado por $\mathbf{C} = \mathbf{D} \nabla \mathbf{H}$, es una matriz $[c_{ij}]_{mxn}$ cuya ij-ésima componente $c_{ij}$ es

$$c_{ij}=\bigvee_{k=1}^{r}(d_{ik}+h_{kj})$$

El producto mínimo de **D** y **H** denotado por $\mathbf{C} = \mathbf{D} \triangle \mathbf{H}$, es una matriz $[c_{ij}]_{mxn}$ cuya ij-ésima componente $c_{ij}$ es

$$c_{ij}=\bigwedge_{k=1}^{r}(d_{ik}+h_{kj})$$

Los productos máximo y mínimo contienen a los operadores máximo y mínimo, los cuales están íntimamente ligados con los conceptos de las dos operaciones básicas de la morfología matemática: dilatación y erosión, respectivamente.

#### 2.3.5.1 Memorias Heteroasociativas Morfológicas

**Algoritmo de las memorias morfológicas max**

**Fase de Aprendizaje**

1. Para cada una de las p asociaciones $(x^\mu , y^\mu )$ se usa el producto mínimo para crear la matriz $y^\mu \triangle(-x^\mu )^t$ de dimensiones m x n, donde el negado transpuesto del patrón de entrada $x^\mu$ se define como$(-\mathbf{X}^\mu)^t = (-x_1^\mu, -x_2^\mu, ..., -x_n^\mu)$ .

2. Se aplica el operador máximo a las p matrices para obtener la memoria **M**.

$$\mathbf{M} = \bigvee_{\mu=1}^{P}[y^\mu \Delta(-x^\mu)^t]$$

**Fase de Recuperación**

Esta fase consiste en realizar el producto mínimo $\Delta$ de la memoria **M** con el patrón de entrada $x^\omega$ , donde $\omega \epsilon$ {1, 2, ..., p}, para obtener un vector columna **y** de dimensión m:

$$\mathbf{y} = \mathbf{M} \Delta \mathbf{x}^\omega$$

Las fases de aprendizaje y de recuperación de las **memorias morfológicas min** se obtienen por dualidad.

#### 2.3.5.2 Memorias Autoasociativas Morfológicas

Para este tipo de memorias se utilizan los mismos algoritmos descritos anteriormente y que son aplicados a las memorias heteroasociativas; lo único que cambia es el conjunto fundamental. Para este caso, se considera el siguiente conjunto fundamental:

$$\{(x^\mu ,x^\mu ) \mid x^\mu \epsilon A^n, \text{ donde } \mu = 1, 2, ..., p\}$$

### 2.3.6 Memorias Asociativas Alfa-Beta

Las memorias Alfa-Beta (41) utilizan máximos y mínimos, y dos operaciones binarias originales $\alpha$ y $\beta$ de las cuales heredan el nombre.

Para la definición de las operaciones binarias $\alpha$ y $\beta$ se deben especificar los conjuntos A y B, los cuales son:

$$A = \{0, 1\} \text{ y } B = \{0, 1, 2\}$$

La operación binaria $\alpha$: A x A $\rightarrow$ B se define como en la tabla 2.1.

**Tabla 2.1:** Descripción del operador alfa

| x | y | $\alpha$ (x, y) |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 2 |
| 1 | 1 | 1 |

La operación binaria $\beta$: B x A$\rightarrow$ A se define como en la tabla 2.2.

**Tabla 2.2:** Descripción del operador beta

| x | y | $\beta$ (x, y) |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 2 | 0 | 1 |
| 2 | 1 | 1 |

Los conjuntos A y B, las operaciones binarias $\alpha$ y $\beta$ junto con los operadores $\wedge$(mínimo) y $\vee$ (máximo) usuales conforman el sistema algebraico (A, B,$\alpha$ ,$\beta$ ,$\wedge$ ,$\vee$) en el que están inmersas las memorias asociativas Alfa-Beta.
El fundamento teórico de las memorias asociativas Alfa-Beta se presenta en el siguiente capítulo de forma más completa, debido a que estas memorias son la base fundamental para la construcción del modelo de BAM utilizado en este trabajo.

### 2.3.7 Memorias Asociativas Media

Las Memorias Asociativas Media (18) utilizan los operadores A y B, definidos de la siguiente forma:

$$\text{A(x, y)} = \text{x - y}$$
$$\text{B(x, y)} = \text{x + y}$$

Las operaciones utilizadas se describen a continuación.

Sean P = $[p_{ij}]_{mxr}$ y Q=$[q_{ij}]_{rxn}$ dos matrices.
Operación $\diamondsuit_A$: $P_{mxr} \diamondsuit_A Q_{rxn} = [f^A_{ij}]_{mxn}$ donde $f^A_{ij}$= $\mathbf{med}^r_{k=1}$ A($p_{ik}$,$q_{k,j}$)
Operación $\diamondsuit_B$: $P_{mxr} \diamondsuit_B Q_{rxn} = [f^B_{ij}]_{mxn}$ donde $f^B_{ij}$= $\mathbf{med}^r_{k=1}$ B($p_{ik}$,$q_{k,j}$)

**Algoritmo Memorias Media**

**Fase de Aprendizaje**

Paso 1. Para cada $\xi$ = 1, 2, ..., p, de cada pareja ($\mathbf{x}^\xi$ , $\mathbf{y}^\xi$ ) se construye la matriz:

$$[\mathbf{y}^\xi \diamondsuit_A ((\mathbf{x})^\xi)^t]_{mxn}$$

Paso 2. Se aplica el operador media a las matrices obtenidas en el paso 1 para obtener la matriz **M**, como sigue:

$$\mathbf{M} = \mathbf{med}^p_{\xi=1}[y^\xi \diamondsuit_A (x^\xi)^t]$$

El ij-ésimo componente **M** está dado como sigue:

$$m_{ij} = \mathbf{med}^P_{\xi=1} A(y^\xi_i, x^\xi_j)$$

***Fase de Recuperación***

Se tienen dos casos:

Caso 1. Recuperación de un patrón fundamental. Un patrón $x^w$, con w $\epsilon$ {1, 2, ..., p} se le presenta a la memoria **M** y se realiza la siguiente operación:

$$\mathbf{M} \diamondsuit_B \mathbf{x}^w$$

El resultado es un vector columna de dimensión n, con la i-ésima componente dada como:

$$(\mathbf{M} \diamondsuit_B x^w)_i = \mathbf{med}^n_{j=1} \text{ B}(m_{ij}, x^w_j)$$

Caso 2. Recuperación de un patrón alterado. Un patrón $\tilde{\mathbf{x}}$ , que es una versión alterada de un patrón $x^w$, se le presenta a la memoria **M**y se realiza la siguiente operación:

$$\mathbf{M} \diamondsuit_B \tilde{\mathbf{x}}$$

De nuevo, es resultado es un vector de dimensión n, con la i-ésima componente dada como:

$$(\mathbf{M} \diamondsuit_B \tilde{\mathbf{x}})_i = \mathbf{med}^n_{j=1} \text{B}(m_{ij}, \tilde{x}_j)$$

# 2. HISTORIA DE LAS MEMORIAS ASOCIATIVAS Y TRABAJOS RELACIONADOS

# Capítulo 3

# Algoritmo de las Memorias Alfa - Beta

En este capítulo se presentan conceptos generales sobre memorias asociativas, así como el modelo de memoria asociativa bidireccional. Este último es el fundamento del modelo ALU asociativo, propuesto en este trabajo.

Entonces, se muestra el fundamento teórico sobre memorias asociativas bidireccionales Alfa-Beta, estas describen como se aplican los operadores $\alpha$ y $\beta$ para las fases de aprendizaje y recuperación descritas en este capítulo.

## 3.1 Conceptos Básicos

El propósito fundamental de una memoria asociativa es recuperar correctamente patrones completos a partir de patrones de entrada, los cuales pueden estar alterados con ruido aditivo, sustractivo o combinado.

Una ***Memoria Asociativa*** se formula como un sistema de entrada y salida, idea que se esquematizaen la fig.3.1.

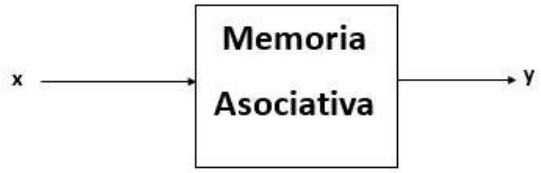



**Figura 3.1:** Diagrama a bloques de una memoria asociativa.

En este esquema, los patrones de entrada y salida están representados por vectores columna denotados por **x** y **y**, respectivamente. Cada uno de los patrones de entrada forma una asociación con el correspondiente patrón de salida, la cual es similar a la una pareja ordenada. Por ejemplo, los patrones **x** y **y** del esquema anterior forman la asociación (**x,y**).

No obstante que a lo largo del presente capítulo se respetarán las notaciones

originales de los autores de los modelos presentados aquí. A continuación se propone una notación que se usará en la descripción de los conceptos básicos sobre memorias asociativas, y en el resto de los capítulos de este trabajo.

Los patrones de entrada y salida se denotarán con las letras negrillas, $\mathbf{x}$ y $\mathbf{y}$, agregándoles números naturales como superíndices para efectos de discriminación simbólica. Por ejemplo, a un patrón de entrada $\mathbf{x}^1$ le corresponderá el patrón de salida $\mathbf{y}^1$, y ambos formarán la asociación ($\mathbf{x}^1$,$\mathbf{y}^1$); del mismo modo, para un número entero positivo $k$ específico, la asociación correspondiente será ($\mathbf{x}^k$,$\mathbf{y}^k$).

La memoria asociativa $\mathbf{M}$ se representa mediante una matriz, la cual se genera a partir de un conjunto finito de asociaciones conocidas de antemano: este es el **conjunto fundamental de aprendizaje**, o simplemente **conjunto fundamental**.

El conjunto fundamental se representa de la siguiente manera:

$$(x^{\mu}\ ,y^{\mu}\ )\ |\ \mu = 1,\ 2,\ ...,\ \mathrm{p}$$

donde p es un número entero positivo que representa la cardinalidad del conjunto fundamental.

A los patrones que conforman las asociaciones del conjunto fundamental se les llama **patrones fundamentales**. La naturaleza del conjunto fundamental proporciona un importante criterio para clasificar las memorias asociativas:

Una memoria es **Autoasociativa** si se cumple que $x^{\mu} = y^{\mu}\ \forall\ \mu\ \epsilon\ \{1,\ 2,\ ...,\ \mathrm{p}\}$, por lo que uno de los requisitos que se debe de cumplir es que n = m.

Una memoria **Heteroasociativa** es aquella en donde $\exists\ \mu\ \epsilon\ \{1,\ 2,\ ...,\ \mathrm{p}\}$ para el que se cumple que $x^{\mu} \neq y^{\mu}$ . Nótese que puede haber memorias heteroasociativas con n = m.

En los problemas donde intervienen las memorias asociativas, se consideran dos fases importantes: La fase de aprendizaje, que es donde se genera la memoria asociativa a partir de las p asociaciones del conjunto fundamental, y la fase de recuperación que es donde la memoria asociativa opera sobre un patrón de entrada, a la manera del esquema que aparece al inicio de esta sección.

A fin de especificar las componentes de los patrones, se requiere la notación para dos conjuntos a los que llamaremos arbitrariamente A y B. Las componentes de los vectores columna que representan a los patrones, tanto de entrada como de salida, serán elementos del conjunto A, y las entradas de la matriz $\mathbf{M}$ serán elementos del conjunto B.

No hay requisitos previos ni limitaciones respecto de la elección de estos dos conjuntos, por lo que no necesariamente deben ser diferentes o poseer características especiales. Esto significa que el número de posibilidades para escoger A y B es infinito.

Por convención, cada vector columna que representa a un patrón de entrada tendrá n componentes cuyos valores pertenecen al conjunto $A$, y cada vector columna que representa a un patrón de salida tendrá m componentes cuyos valores pertenecen también al conjunto$A$. Es decir:

$$\mathbf{x}^{\mu}\epsilon A^{n}\ y\ \mathbf{y}^{\mu}\epsilon A^{m}\forall\mu\epsilon\{1,2,...,p\}$$

La j-ésima componente de un vector columna se indicará con la misma letra del vector, pero sin negrilla, colocando a j como subíndice (j $\epsilon$ {1, 2, ..., n} o j $\epsilon$ {1, 2, ..., m} según corresponda). La j-ésima componente del vector columna $\mathbf{x}^\mu$ se representa por: $x_j^\mu$

Con los conceptos básicos ya descritos y con la notación anterior, es posible expresar las dos fases de una memoria asociativa:

1. **Fase de Aprendizaje** (Generación de la memoria asociativa). Encontrar los operadores adecuados y una manera de generar una matriz **M** que almacene las p asociaciones del conjunto fundamental {($\mathbf{x}^1$,$\mathbf{y}^1$), ($\mathbf{x}^2$,$\mathbf{y}^2$), ..., ($\mathbf{x}^p$,$\mathbf{y}^p$)}, donde $\mathbf{x}^\mu \epsilon A^n$ y $\mathbf{y}^\mu \epsilon A^m \; \forall \; \mu \epsilon$ {1, 2, ..., p}. Si m=n y $\mathbf{x}^\mu = \mathbf{y}^\mu \; \forall \; \mu \epsilon$ {1, 2, ..., p} tal que $\mathbf{x}^\mu = \mathbf{y}^\mu \epsilon$ {1,2,...,p}, la memoria será heteroasociativa;

2. **Fase de Recuperación** (Operación de la memoria asociativa). Hallar los operadores adecuados y las condiciones suficientes para obtener el patrón fundamental de salida $\mathbf{y}^\mu$ , cuando se opera la memoria **M** con el patrón fundamental de entrada $\mathbf{x}^\mu$ ; lo anterior para todos los elementos del conjunto fundamental y para ambos modos: autoasociativo y heteroasociativo.

Se dice que una memoria asociativa **M** exhibe **recuperación correcta** si al presentarle como entrada, en la fase de recuperación, un patrón $\mathbf{x}^\omega$ con $\omega \epsilon$ {1, 2, ..., p}, ésta responde con el correspondiente patrón fundamental de salida $\mathbf{y}^\omega$ .

Una memoria asociativa bidireccional también es un sistema de entrada y salida, solamente que el proceso es bidireccional. La dirección hacia adelante se describe de la misma forma que una memoria asociativa común: al presentarle una entrada **x**, el sistema entrega una salida **y**. La dirección hacia atrás se lleva a cabo presentándole al sistema una entrada y para recibir una salida **x**.

## 3.2 Descripción de las Memorias Asociativas Bidireccionales Alfa-Beta

En general, cualquier modelo de memoria asociativa bidireccional presente en la literatura científica actual se podría esquematizar como se muestra en la figura 3.1.


**Figura 3.2:** Esquema general de una Memoria Asociativa Bidireccional.

La BAM general es una "caja negra" que opera de la siguiente forma: dado un patrón **x** obtiene el patrón asociado **y**, y dado el patrón **y** obtiene el patrón

asociado **x**. Además, si se asume que $\tilde{x}$ y $\tilde{y}$ son las versiones ruidosas de **x** y **y**, respectivamente,la BAM recupera los patrones correspondientes, **x** y **y**, libres de ruido.

Por ejemplo, el modelo de Kosko podría esquematizarse como se muestra en la figura 3.3.

El modelo utilizado en este trabajo se comporta de la misma forma que la BAM general en lo que respecta a la operación al recuperar patrones. El modelo se ha denominado Memorias Asociativas Bidireccionales Alfa-Beta (Alpha-Beta BAM) porque las memorias asociativas Alfa-Beta, tanto max como min, juegan un papel central en el diseño de este nuevo modelo.

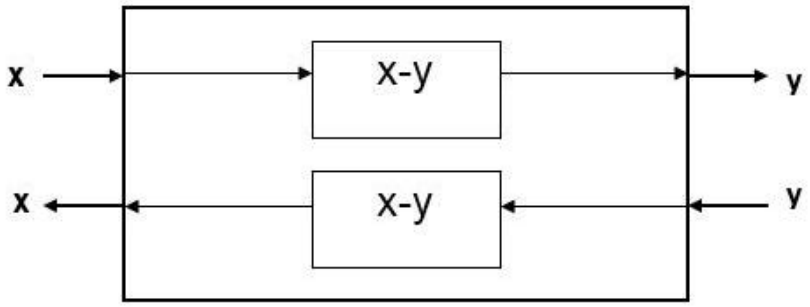



**Figura 3.3:** Modelo de Kosko según el esquema de la BAM general.

Dado que es una memoria asociativa bidireccional, la recuperación de patrones debe darse en dos direcciones opuestas. En cada una de las dos direcciones, el modelo consta de dos etapas, y las cuatro etapas se muestran en la figura 3.4.

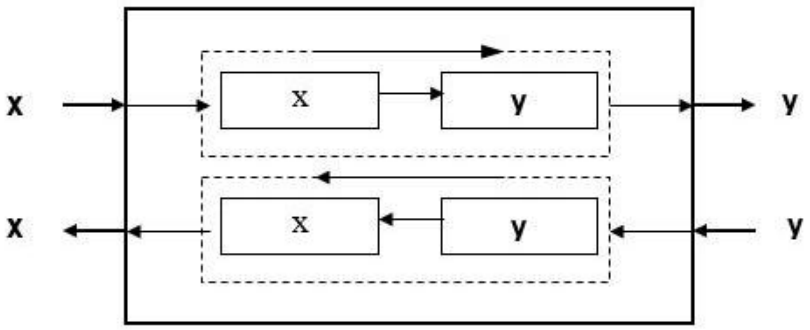



**Figura 3.4:** Esquema de las etapas de una memoria asociativa bidireccional Alfa-Beta.

En este trabajo se asume que las memorias asociativas Alfa-Beta tienen un conjunto fundamental denotado por $\{(\mathbf{x}^\mu, \mathbf{y}^\mu) \mid \mu = 1, 2, ..., p\}$ $\mathbf{x}^\mu \in A^n$ y $\mathbf{y}^\mu \in A^m$, con A = {0, 1}, n $\epsilon$ $\mathbf{Z}^+$, p $\epsilon$ $\mathbf{Z}^+$, m $\epsilon$ $\mathbf{Z}^+$ y 1 ¡ p $\leq (2^n, 2^m)$. Además, se cumple

la condición de que todos los patrones de entrada sean diferentes; es decir $\mathbf{x}^\mu = \mathbf{x}^\xi$ si y sólo si $\mu = \xi$ . Si $\forall\ \mu \in \{1, 2, ..., p\}$ se cumple que $\mathbf{x}^\mu = \mathbf{y}^\mu$ , la memoria Alfa-Beta será autoasociativa; y si la afirmación es falsa, es decir $\exists\ \mu \in \{1, 2, ..., p\}$ para el que se cumple $\mathbf{x}^\mu \neq \mathbf{y}^\mu$ , la memoria Alfa-Beta será heteroasociativa.

Antes de continuar con la descripción del proceso y el desarrollo de los fundamentos matemáticos, se definen tres tipos de vectores que se usarán intensivamente en el nuevo modelo y se definen, además, dos transformadas vectoriales originales, propias del nuevo modelo.

Definición 1 (One-Hot) Sea el conjunto A = {0, 1} y sean p $\in \mathbf{Z}^+$ , p>1, k$\in \mathbf{Z}^+$, tales que $1 \leq k \leq p$. El k-ésimo vector *one-hot* de p bits se define como el vector $\mathbf{h}^k \in A^p$ para el cual se cumple que la k-ésima componente $h^k_k$ y las demás componentes $h^k_j = 0$ , $\forall\ j \neq k$, $1 \leq j \leq p$.

Nota En esta definición se excluye el valor p = 1 porque un vector *one-hot* de dimensión 1, por su esencia misma, no tiene razón de ser.

Definición 2 (Zero-Hot) Sea el conjunto A = {0, 1} y sean p $\in \mathbf{Z}^+$, p>1, k $\in \mathbf{Z}^+$, tales que $1 \prec k \leq p$. El k-ésimo vector zero-hot de p bits se define como el vector $\mathbf{h}^{-k} \in A^p$ para el cual se cumple que la k-ésima componente $h^{-k}_k = 0$ y las demás componentes $h^{-k}_j$,$\forall\ j \neq k$, $1 \leq j \leq p$ .

Nota En esta definición se excluye el valor p = 1 porque un vector zero-hot de dimensión 1, por su esencia misma, no tiene razón de ser.

Definición 3 (Transformada vectorial de expansión dimensional) Sea el conjunto A = {0, 1} y sean n $\in \mathbf{Z}^+$,y m $\in \mathbf{Z}^+$ . Dados dos vectores cualesquiera $\mathbf{x} \in A^n$ y $\mathbf{e} \in A^m$, se define la transformada vectorial de expansión de orden m, $\tau^e : A^n \rightarrow A^{n+m}$ , como $\tau^e$ (**x, e**) = $\mathbf{X} \in A^{n+m}$, vector cuyas componentes son: $X_i = x_i$ para $1 \leq i \leq n$ y $X_i = e_i$ para $n + 1 \leq i \leq n + m$.

Definición 4 (Transformada vectorial de contracción dimensional) Sea el conjunto A = {0, 1} y sean n $\in Z^+$,$m \in$ tales que $1 \leq$ m ¡n. Dado un vector cualesquiera $\mathbf{X} \in A^{n+m}$, se define la transformada vectorial de contracción de orden m, $\tau^c : A^{n+m} \rightarrow A^m$, como $\tau^c(\mathbf{X}, m) = \mathbf{c} \in A^m$, vector cuyas componentes son: $c_i = X_{i+n}$ para $1 \leq i$ ¡ m.

Definición 5 (Vector negado) Sea el conjunto A = {0, 1} y sea un vector $\mathbf{s} \in A^n$, se define el vector negado de **s** como el vector $\bar{\mathbf{S}}$ , tal que $\bar{S}_i = -S_i$ , donde es el operador lógico de negación booleano.

Habiendo definido cinco conceptos importantes, se continúa con el proceso de la descripción del funcionamiento de las memorias asociativas bidireccionales Alfa-Beta.

En la dirección $\mathbf{x} \rightarrow \mathbf{y}$, las etapas 1 y 2 tienen como función proporcionar un $\mathbf{y}^k$ a la salida (k = 1, ..., p) dado un $x^k$ a la entrada, asumiendo que en el conjunto fundamental se incluye la asociación $(x^k, y^k)$, como se ilustra en la figura 3.4.

El modelo está diseñado de tal forma que la Etapa 2, constituida principalmente por el Linear Associator modificado, entregue como salida el vector $y^k$; si recordamos que el Linear Associator tiene recuperación correcta cuando a la entrada

se presentan vectores ortonormales, es deseable que a la salida de la Etapa 1 se tengan precisamente vectores de este tipo. En la Etapa 2: se presenta una variación original del Linear Associator que permite obtener $\mathbf{y}^k$ a partir de un vector $\mathbf{h}^k$ one-hot en su k-ésima coordenada.

Por tanto, ya se tiene resuelta la tarea que debe realizar la Etapa 2. De la figura 3.5 se observa que falta descubrir qué hace la Etapa 1.

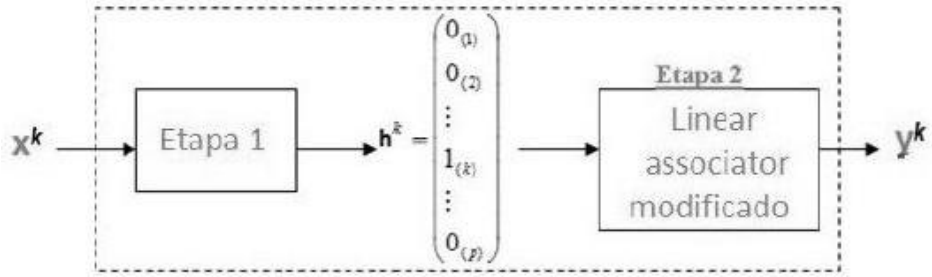



**Figura 3.5:** Esquema del proceso a realizar en el sentido x → y.

La tarea de la Etapa 1 es: dado un $\mathbf{x}^k$ o una versión ruidosa de éste ($\tilde{\mathbf{x}}^k$ ), se debe obtener sin ambigüedad y sin ninguna condición adicional, el vector *one-hot* $\mathbf{h}^k$.

La Etapa 1 juega un papel importante en la fase de aprendizaje, y realiza lo siguiente:

El primer paso de esta etapa es tomar el conjunto fundamental $\{(\mathbf{x}^\mu , \mathbf{x}^\mu ) \mid = 1, ..., p\}$, al que llamaremos Conjunto Fundamental Original de $\mathbf{x}$ ($CFO_x$). A continuación, a cada uno de los patrones $\mathbf{x}^\mu$ se le aplica la transformada vectorial de expansión dimensional del vector $\mathbf{x}^\mu$ con su correspondiente vector one-hot de p bits, esto es:

$$\tau^e \ (\mathbf{x}^\mu \ ,\mathbf{h}^\mu \ ) = \mathbf{X}^\mu$$

Se tiene el nuevo conjunto fundamental $\{(X^\mu \ ,X^\mu) \mid \mu = 1, ..., p\}$, al que denominaremos Conjunto Fundamental one-hot de $\mathbf{X}$ ($CFoh_X$). Con el $CFoh_X$ se crea la memoria autoasociativa Alfa-Beta max, de la siguiente manera:

$$\vee_x = \bigvee_{\mu=1}^{P} [\mathbf{X}^\mu \bigotimes (\mathbf{X}^\mu)^t]$$

De nuevo se toma el $CFO_x$. A cada uno de los patrones del $CFO_x$ se le aplica la transformada vectorial de expansión dimensional del vector $x^\mu$ con su correspondiente vector zero-hot de p bits, y obtenemos:

$$\tau^e(\mathbf{X}^\mu,\bar{\mathbf{h}}^\mu) = \bar{\mathbf{X}}^\mu$$

Se obtiene el nuevo conjunto fundamental $\{(\mathbf{X}^\mu\ ,\ \mathbf{X}^\mu\ )|\ \mu = 1,\ ...,\ \mathrm{p}\}$, denominado conjunto fundamental zero-hot de $\mathbf{X}$ ($CFzh_X$). Con el $CFzh_X$ se crea la memoria autoasociativa Alfa-Beta min, de la siguiente manera:

$$\wedge_x = \bigvee_{\mu=1}^{P} [\mathbf{X}^\mu \bigotimes (\mathbf{X}^\mu)^t]$$

Al presentar como entrada en la Etapa 1 el vector $\mathbf{x}^k$, y al operar las memorias $\mathbf{V}_x$ y $\wedge_x$ obtenidas en los pasos anteriores, se obtiene como resultado, de acuerdo con el algoritmo detallado en la sección 4.3, el vector one-hot $\mathbf{h}^k$ que será la entrada de la Etapa 2.

En la Etapa 2 se construye un Linear Associator con los patrones $y^\mu$ , usando un algoritmo de modificación original. Esta modificación consiste en aprovechar las características del conjunto de vectores one-hot para así obviar la fase de aprendizaje del Linear Associator, la cual se realiza implícitamente: se toman los vectores $y^\mu$ con = 1, 2, ..., p y se acomodan en una matriz $\mathbf{LAy}$ de la siguiente manera:

$$\mathbf{LAy} = \begin{bmatrix} y_1^1 & y_1^2 & \cdots & y_1^p \\ y_2^1 & y_2^2 & \cdots & y_2^p \\ \vdots & \vdots & \ddots & \vdots \\ y_n^1 & y_1^2 & \cdots & y_n^p \end{bmatrix}$$

Así, en la fase de recuperación, se presenta a la entrada de $\mathbf{LAy}$ un patrón one-hot, digamos $\mathbf{h}^k$, y se obtiene de inmediato el correspondiente $\mathbf{y}^k$ a la salida, que también es la salida de la memoria asociativa bidireccional Alfa-Beta en el sentido $\mathbf{x} \longrightarrow \mathbf{y}$.

Sólo resta describir el proceso que se sigue en el sentido de y→x para evidenciar la bidireccionalidad del modelo. Este proceso se esquematiza en la figura 3.6.

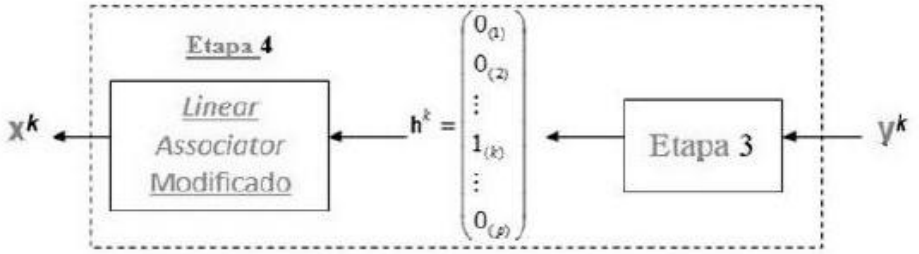



**Figura 3.6:** Esquema del proceso a realizar en el sentido y → x.

La tarea de la Etapa 3 es similar a la tarea de la Etapa 1, pero con argumentos $y^k$: dado un $y^k$ o una versión ruidosa de éste ($\tilde{y}^k$), se debe obtener sin ambigüedad y sin ninguna condición adicional, el vector one-hot $h^k$.

Como primer paso, se toma como conjunto fundamental $\{(\mathbf{y}^\mu, \mathbf{y}^\mu) \mid \mu = 1, ..., p\}$, al que llamaremos Conjunto Fundamental Original de y ($CFO_y$). A continuación, cada uno de los patrones $y^\mu$ se le aplica la transformada vectorial de expansión dimensional del vector y con su correspondiente vector one-hot de p bits, esto es:

$$\tau^e\ (y^\mu\ , h^\mu\ ) = Y^\mu$$

Ahora tenemos el nuevo conjunto fundamental $\{(Y^\mu\ , Y^\mu) |\ \mu = 1, ..., p\}$, que denominaremos Conjunto Fundamental one-hot de $\mathbf{Y}$ ($CFoh_Y$). Con el $CFoh_Y$ se crea la memoria autoasociativa Alfa-Beta max, de la siguiente manera:

$$\vee_y = \bigvee_{\mu=1}^{P} [\mathbf{Y}^\mu \bigotimes (\mathbf{Y}^\mu)^t]$$

De nuevo, se toma el $CFO_y$. A cada uno los patrones del $CFO_y$ se le aplica la transformada vectorial de expansión dimensional del vector $y^\mu$ con su correspondiente vector zero-hot de p bits, lo que resulta en:

$$\tau^e(\mathbf{Y}^\mu, \bar{\mathbf{h}}^\mu) = \bar{\mathbf{Y}}^\mu$$

Se obtiene el nuevo conjunto fundamental $\{(\bar{\mathbf{Y}}^\mu\ , \bar{\mathbf{Y}}^\mu) \mid \mu = 1, ..., p\}$, denominado conjunto fundamental zero-hot de $\mathbf{Y}$ ($CFzh_Y$). Con el $CFzh_Y$ se crea la memoria autoasociativa Alfa-Beta min, de la siguiente manera:

$$\wedge_y = \bigvee_{\mu=1}^{P} [\mathbf{Y}^\mu \bigotimes (\mathbf{Y}^\mu)^t]$$

De manera similar a la Etapa 2, en la Etapa 4 se construye un Linear Associator modificado con los patrones $\mathbf{x}^\mu$ con $\mu = 1, 2, ..., p$ y se acomodan en una matriz $\mathbf{LAx}$ de la siguiente manera:

$$\mathbf{LAx} = \begin{bmatrix} x_1^1 & x_1^2 & \cdots & x_1^p \\ x_2^1 & x_2^2 & \cdots & x_2^p \\ \vdots & \vdots & \ddots & \vdots \\ x_n^1 & x_1^2 & \cdots & x_n^p \end{bmatrix}$$

Hasta este momento, se describió el proceso para la construcción y operación de las memorias asociativas bidireccionales Alfa-Beta; es decir, se han descrito los pasos necesarios para concretar ambas fases: la de aprendizaje y la de recuperación de patrones.

En las siguientes dos secciones se presentará detalladamente el fundamento teórico que sustenta el funcionamiento de las memorias asociativas bidireccionales Alfa-Beta.

## 3.3 Fundamento Teórico de las Etapas 1 y 3

El algoritmo de la Etapa 1 y, por consiguiente, de la etapa 3 es la contribución más importante de las memorias asociativas bidireccionales alfa-beta.

## 3.4 Fundamento Teórico de las Etapas 2 y 4

En esta sección se presenta el fundamento teórico que sustenta el diseño y operación de las Etapas 2 y 4, cuyo elemento principal es una variación original del Linear Associator.

Sea $\{(\mathbf{x}^\mu\ ,\ \mathbf{y}^\mu\ )\ —\ \mu = 1,\ 2,\ ...,\ \mathrm{p}\}$ con $\mathrm{A} = \{0,\ 1\}$ , $\mathbf{x}^\mu \epsilon\ \mathbf{A}^n$ y $\mathbf{y}^\mu \epsilon\ A^m$ el conjunto fundamental del Linear Associator.

La fase de Aprendizaje consiste de dos etapas:

3) Para cada una de las p asociaciones $(\mathbf{x}^\mu\ ,\ y^\mu\ )$ se encuentra la matriz $\mathbf{y}^\mu \cdot (\mathbf{x}^\mu)^t$ de dimensiones m x n

4) Se suman la p matrices para obtener la memoria

$$M = \sum_{\mu=1}^{p} y^\mu \cdot (X^\mu)^t = [m_{ij}]_{mxn}$$

de manera que la ij-ésima componente de la memoria M se expresa así:

$$m_{ij} = \sum_{\mu=1}^{p} y_i^\mu x_j^\mu$$

La fase de recuperación consiste en presentarle a la memoria un patrón de entrada $x^\omega$ , donde $\omega \epsilon\ \{1,\ 2,\ ...,\ \mathrm{p}\}$ y realizar la operación

$$M \cdot x^\omega = [\sum_{\mu=1}^{p} y^\mu \cdot (x^\mu)^t] \cdot \mathbf{x}^\omega$$

La siguiente forma de expresión nos permite investigar las condiciones que se deben cumplir para que el método de recuperación propuesto dé como resultado salidas correctas.

$$M \cdot x^\omega = y^\omega \cdot [(x^\omega)^t \cdot x^\omega] + \sum_{\mu \neq \omega} y^\mu \cdot [(x^\mu)^t \cdot x^\omega]$$

Para que la expresión anterior arroje como resultado al patrón $y^\omega$ , es preciso que se cumplan dos igualdades:

1) $[(\mathbf{x}^\omega)^t \cdot x^\omega] = 1$

2) $[(\mathbf{x}^\omega)^t \cdot x^\omega] = 0$ siempre que $\mu \neq \omega$

Lo que nos indica que para que haya recuperación correcta, los vectores $x^\mu$ deben ser ortonormales. Si sucede este caso, entonces, para $\mu = 1,\ 2,\ ...,\ \mathrm{p}$, tenemos que:

$$y^1 \cdot (x^1)^t = \begin{pmatrix} y_1^1 \\ y_2^1 \\ \vdots \\ y_m^1 \end{pmatrix} \cdot \begin{pmatrix} x_1^1 & x_2^1 & \cdots & x_n^1 \end{pmatrix} = \begin{pmatrix} y_1^1 & 0 & 0 & \cdots & 0_{(n)} \\ y_2^1 & 0 & 0 & \cdots & 0_{(n)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_m^1 & 0 & 0 & \cdots & 0_{(n)} \end{pmatrix}$$

$$y^2 \cdot (x^2)^t = \begin{pmatrix} y_1^2 \\ y_2^2 \\ \vdots \\ y_m^2 \end{pmatrix} \cdot \begin{pmatrix} x_1^2 & x_2^2 & \cdots & x_n^2 \end{pmatrix} = \begin{pmatrix} 0 & y_1^2 & 0 & \cdots & 0_{(n)} \\ 0 & y_2^2 & 0 & \cdots & 0_{(n)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & y_m^2 & 0 & \cdots & 0_{(n)} \end{pmatrix}$$

$$y^p \cdot (x^p)^t = \begin{pmatrix} y_1^p \\ y_2^p \\ \vdots \\ y_m^p \end{pmatrix} \cdot \begin{pmatrix} x_1^p & x_2^p & \cdots & x_n^p \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 & \cdots & y_{1(n)}^p \\ 0 & 0 & 0 & \cdots & y_{2(n)}^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & y_{m(n)}^p \end{pmatrix}$$

Por lo tanto,

$$\mathbf{M} = \sum_{\mu=1}^{p} \mathbf{y}^\mu \cdot (\mathbf{x}^\mu)^t = \begin{pmatrix} y_1^1 & y_1^2 & y_1^3 & \cdots & y_{1(n)}^p \\ y_2^1 & y_2^2 & y_2^3 & \cdots & y_{2(n)}^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ y_3^1 & y_3^2 & y_3^3 & \cdots & y_{m(n)}^p \end{pmatrix}$$

Aprovechando la característica mostrada por el Linear Associator cuando los vectores de entrada x son ortonormales, y dado que, por la Definición 1, los vectores one-hot $h^k$ con k = 1, ..., p son ortonormales, se puede obviar la fase de aprendizaje al evitar hacer las operaciones vectoriales realizadas por el Linear Associator, y simplemente colocando los vectores en orden para formar el Linear Associator.

Las etapas 2 y 4 corresponden a dos Linear Associator modificados, construidos con los vectores **y** y **x**, respectivamente, del conjunto fundamental.

## 3.5 Algoritmo

En esta sección se describen paso a paso los procesos requeridos por la BAM Alfa-Beta tanto en la Fase de Aprendizaje, como en la Fase de Recuperación en el sentido de x $\rightarrow$ y, algoritmo para las Etapas 1 y 2, y en el sentido y $\rightarrow$ x, algoritmo para las Etapas 3 y 4.

Para finalizar este apartado, se presenta un ejemplo que ilustra la aplicación del algoritmo de las Etapas 1 y 2.

### 3.5.1 Algoritmo de las Etapas 1 y 2

El siguiente algoritmo describe los pasos requeridos por la memoria asociativa bidireccional Alfa-Beta para realizar la fase de aprendizaje y la fase de recuperación en el sentido de $\mathbf{x} \rightarrow \mathbf{y}$.

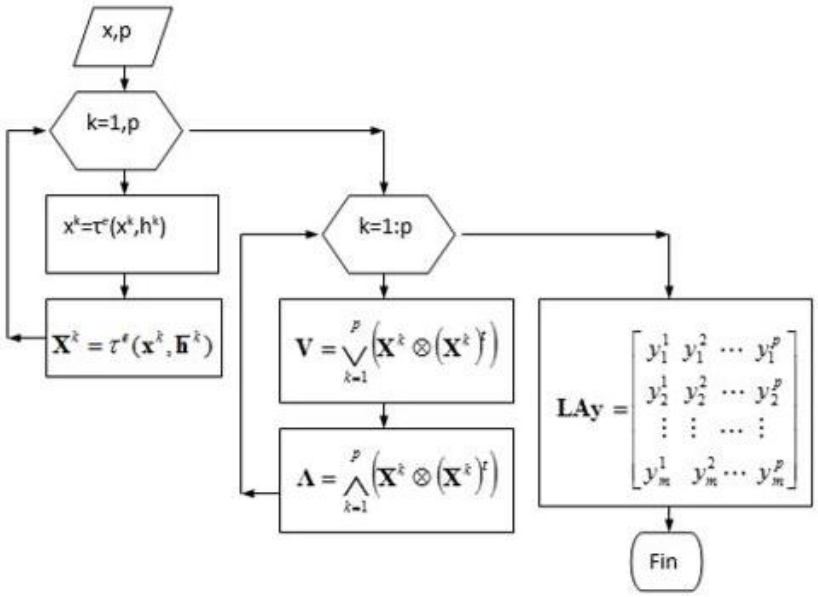



**Figura 3.7:** Diagrama de flujo del algoritmo de la fase de aprendizaje de las memorias $\alpha$-$\beta$ BAM.

Paso 1. Como datos se tienen los p patrones de entrada $\mathbf{x}$.

Paso 2. Se aplica la transformada de expansión vectorial del vector $\mathbf{x}$ para cada $\mathbf{X}^k$, utilizando como argumentos cada uno de los vectores de entrada $\mathbf{x}^k$ y cada vector one-hot $\mathbf{h}^k$.

Paso 3. Se aplica la transformada de expansión vectorial del vector $\mathbf{x}$ para cada $\bar{X}^k$ , utilizando como argumentos cada uno de los vectores de entrada $x^k$ y cada vector zero-hot $\bar{h}^k$ .

Paso 4. Se crea una memoria asociativa Alfa-Beta *max* V con el conjunto fundamental

Paso 5. Se crea una memoria asociativa Alfa-Beta *min* $\wedge$ con el conjunto fundamental

Paso 6. Se crea una matriz $\mathbf{LAy}$, que consiste de un Linear Associator modificado utilizando los patrones de salida $\mathbf{y}$.

## FASE DE RECUPERACIÓN

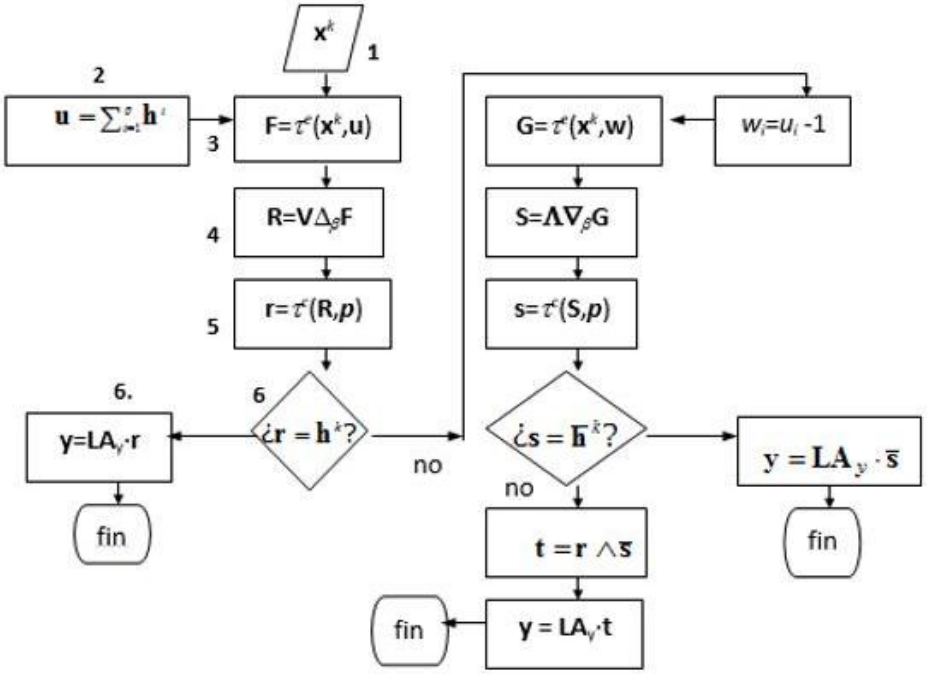



**Figura 3.8:** Diagrama de flujo del algoritmo de la fase de recuperación de las memorias $\alpha$-$\beta$ BAM.

Paso 1. Presentar, a la entrada de la etapa 1, un vector del conjunto fundamental $x^k \in A^n$ para algún índice k $\in$ {1, ..., p}

Paso 2. Construir el vector $\mathbf{u} \in A^p$

Paso 3. Aplicar la transformada vectorial de expansión del vector **x** utilizando como argumentos el vector $x^k$ y el vector **u**, para obtener **F**.

Paso 4. Operar la memoria autoasociativa Alfa-Beta max $\mathbf{V}$ con **F**, para obtener un vector **R**.

Paso 5. Aplicar la transformada de contracción del vector **R**, cuyos argumentos son el vector **R**, obtenido en el paso anterior, y p (número de patrones), para obtener el vector **r**.

Paso 6. **Si r** es un vector one-hot, entonces r es el k-ésimo vector one-hot, $h^k$, (Basado en el Teorema 4.2) **entonces**.

Paso 6.1 Se realiza la operación **LAy** * **r**, lo que resultará en el $y^k$ correspondiente. **Fin. Si no**, entonces

Paso 7. Construir el vector $\mathbf{w} \in A^p$

Paso 8. Aplicar la transformada vectorial de expansión del vector $\mathbf{x}$ utilizando como argumentos el vector $\mathbf{x}^k$ y el vector $\mathbf{w}$, para obtener $\mathbf{G}$.

Paso 9. Operar la memoria autoasociativa Alfa-Beta min $\wedge$ con $\mathbf{G}$, para obtener un vector $\mathbf{S}$.

Paso 10. Aplicar la transformada de contracción del vector $\mathbf{S}$, cuyos argumentos son el vector $\mathbf{S}$, obtenido en el paso anterior, y p (número de patrones), para obtener el vector $\mathbf{s}$.

Paso 11. **Si** $\boldsymbol{s}$ es un vector zero-hot, entonces $s$ es el k-ésimo vector zero-hot, $h^{-h}$ , (Basado en el Teorema 4.4) **entonces**.

Paso 11.1 Se realiza la operación $\mathbf{LAy}\cdot\mathbf{s}$, lo que resultará en el $\mathbf{y}^k$ correspondiente. **Fin. Si no**, entonces

Paso 12. Realizar la operación AND lógica entre r y $\bar{\mathbf{s}}$ para obtener el vector $\mathbf{t}$, el cual (Basado en el Teorema 4.5) será igual al k-ésimo vector one-hot.

Paso 13. Realizar la operación $\mathbf{LAy}$ * t, para obtener el $\mathbf{y}^k$ correspondiente. **Fin**.

### 3.5.2 Algoritmo de las Etapas 3 y 4

El siguiente algoritmo describe los pasos requeridos por la memoria asociativa bidireccional Alfa-Beta para realizar la fase de aprendizaje y la fase de recuperación en el sentido de $\mathbf{y} \rightarrow \mathbf{x}$.

**FASE DE APRENDIZAJE**

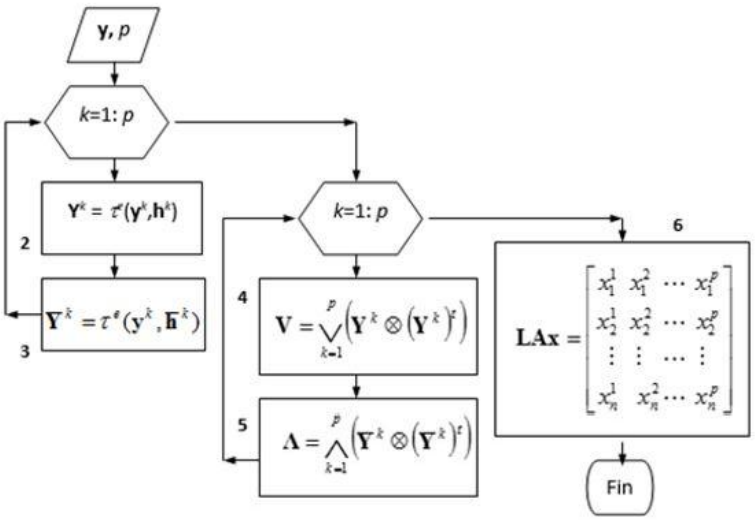



**Figura 3.9:** Diagrama de flujo del algoritmo de la fase de aprendizaje de las memorias $\alpha$-$\beta$ BAM para la recuperación de patrones.

Paso 1. Como datos se tienen los p patrones de salida $\mathbf{y}$.

Paso 2. Se aplica la transformada de expansión vectorial del vector $\mathbf{Y}$ para cada $\mathbf{Y}^k$, utilizando como argumentos cada uno de los vectores de entrada $\mathbf{y}^k$ y cada vector one-hot $h^k$.

Paso 3. Se aplica la transformada de expansión vectorial del vector $\bar{\mathbf{Y}}$ para cada $\bar{\mathbf{Y}}^k$ , utilizando como argumentos cada uno de los vectores de entrada $\mathbf{y}^k$ y cada vector zero-hot $\mathbf{h}^{-k}$.

Paso 4. Se crea una memoria asociativa Alfa-Beta $max$ $\mathbf{V}$ con el conjunto fundamental

$$\{(\mathbf{Y}^k, \mathbf{Y}^k)|k = 1, ..., p\}$$

Paso 5. Se crea una memoria asociativa Alfa-Beta $min$ $\wedge$ $\wedge$ con el conjunto fundamental

$$\{(\mathbf{Y}^{-k}, \mathbf{Y}^{-k})|k = 1, ..., p\}$$

Paso 6. Se crea una matriz $\mathbf{LAx}$, que consiste de un Linear Associator modificado utilizando los patrones de entrada $\mathbf{x}$.

**FASE DE RECUPERACIÓN**

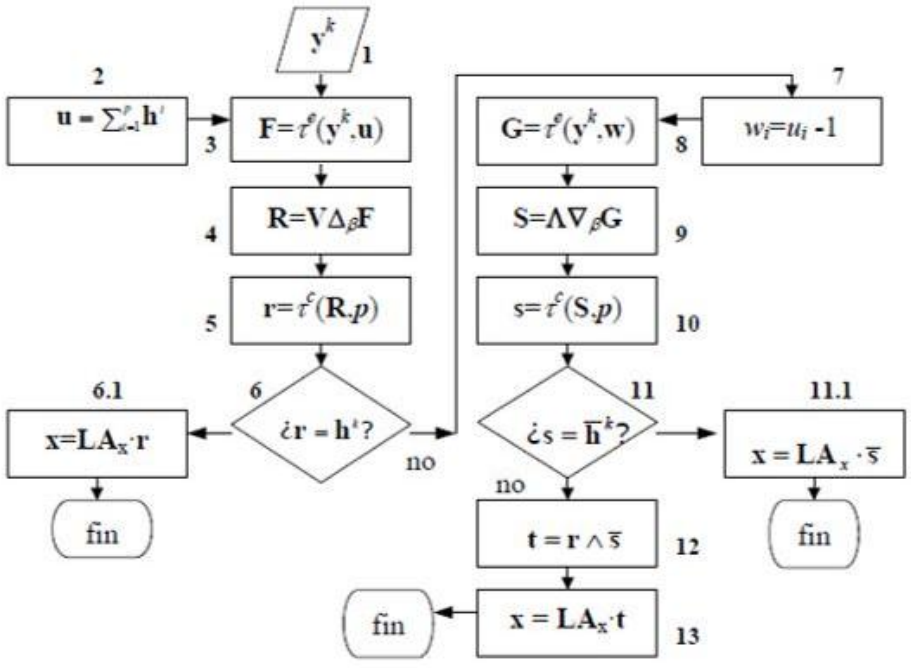



**Figura 3.10:** Diagrama de flujo del algoritmo de la fase de recuperación de las memorias $\alpha$-$\beta$ BAM para la recuperación de patrones.

Paso 1. Presentar, a la entrada de la etapa 1, un vector del conjunto fundamental $y^k \epsilon A^m$ para algún índice k $\epsilon$ {1, ..., p}

Paso 2. Construir el vector $\mathbf{u} \epsilon A^p$

Paso 3. Aplicar la transformada vectorial de expansión del vector **y** utilizando como argumentos el vector $\mathbf{y}^k$ y el vector **u**, para obtener **F**.

Paso 4. Operar la memoria autoasociativa Alfa-Beta $max\,\mathbf{V}$ con **F**, para obtener un vector **R**.

Paso 5. Aplicar la transformada de contracción del vector **R**, cuyos argumentos son el vector **R**, obtenido en el paso anterior, y p (número de patrones), para obtener el vector **r**.

Paso 6. **Si r** es un vector one-hot, entonces $r$ es el k-ésimo vector one-hot, $\mathbf{h}^k$, (Basado en el Teorema 4.2) **entonces**.
Paso 6.1 Se realiza la operación **LAx -r**, lo que resultará en el $\mathbf{x}^k$ correspondiente. **Fin. Si no**, entonces

Paso 7. Construir el vector $\mathbf{w} \epsilon A^p$

Paso 8. Aplicar la transformada vectorial de expansión del vector **y** utilizando como argumentos el vector $\mathbf{y}^k$ y el vector **w**, para obtener **G**.

Paso 9. Operar la memoria autoasociativa Alfa-Beta min $\wedge$ con **G**, para obtener un vector **S**.

Paso 10. Aplicar la transformada de contracción del vector **S**, cuyos argumentos son el vector **S**, obtenido en el paso anterior, y p (número de patrones), para obtener el vector **s**.

Paso 11. **Si s** es un vector zero-hot, entonces s es el k-ésimo vector zero-hot,$\mathbf{h^{-}}^h$, (Basado en el Teorema 4.4) **entonces**.

Paso 11.1 Se realiza la operación **LAx**$\cdot\ \bar{\mathbf{S}}$, lo que resultará en el $\mathbf{x}^k$ correspondiente. **Fin. Si no**, entonces

Paso 12. Realizar la operación AND lógica entre r y $\bar{\mathbf{s}}$ para obtener el vector t, el cual (Basado en el Teorema 4.5) será igual al k-ésimo vector one-hot.

Paso 13. Realizar la operación **LAx-t**, para obtener el $\mathbf{x}^k$ correspondiente. **Fin**.

### 3.5.3 Ejemplo ilustrativo de la Memoria Asociativa Bidireccional Alfa-Beta

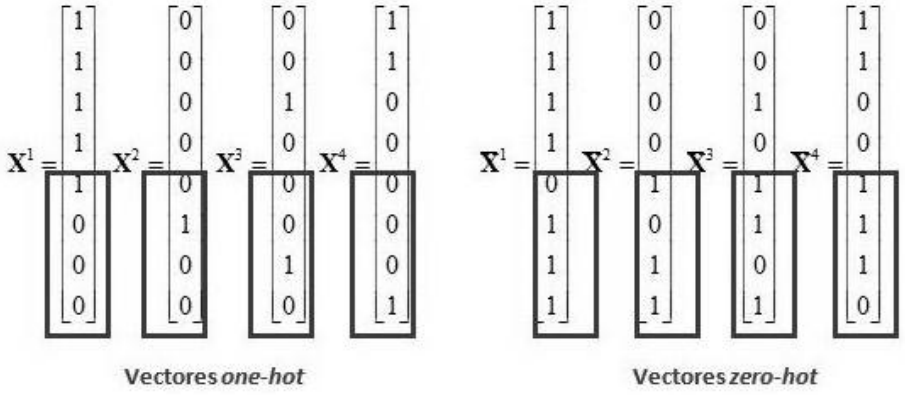
Sean los siguientes 4 pares de patrones (p=4), los patrones de entrada, **x** con dimensión 4 (n=4) y los patrones de salida, **y** con dimensión 3 (m=3).

$$x^1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, y^1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, x^2 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, y^2 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},$$

$$x^3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, y^3 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, x^4 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix}, y^4 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

## FASE DE APRENDIZAJE

Se aplica la transformada vectorial de expansión del vector x, para obtener cada uno de los vectores $\mathbf{X}^k$ y $\bar{\mathbf{X}}^k$




Se generan las memorias autoasociativas Alfa-Beta $max \vee = \bigvee_{k=1}^{4} (\mathbf{X}^k \bigotimes (\mathbf{X}^k)^t)$ y $min \wedge = \bigwedge_{k=1}^{4} (\bar{X}^k \bigotimes (\bar{X}^k)^t)$

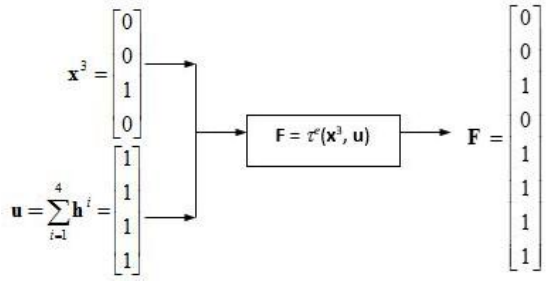
$$\mathbf{V} = \begin{bmatrix} 1&1&2&2&2&2&2&1 \\ 1&1&2&2&2&2&2&2 \\ 2&2&1&2&2&2&2&2 \\ 1&1&1&1&1&2&2&2 \\ 1&1&1&1&1&2&2&2 \\ 2&2&2&2&2&1&2&2 \\ 2&2&1&2&2&2&1&2 \\ 1&1&2&2&2&2&2&1 \end{bmatrix} \quad \text{y} \quad \mathbf{\Lambda} = \begin{bmatrix} 1&1&0&1&0&0&0&0 \\ 1&1&0&1&0&0&0&0 \\ 0&0&1&1&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 1&1&1&1&0&1&0&0 \\ 1&1&0&1&0&0&1&0 \\ 0&0&1&1&0&0&0&1 \end{bmatrix}$$

Se crea el Linear Associator Modificado utilizando los patrones de salida **y**

$$\mathbf{LAy} = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

### *FASE DE RECUPERACIÓN*

Se presenta a la entrada de la BAM Alfa-Beta el patrón $x^3$, se construye el vector $u$ y se construye la expansión

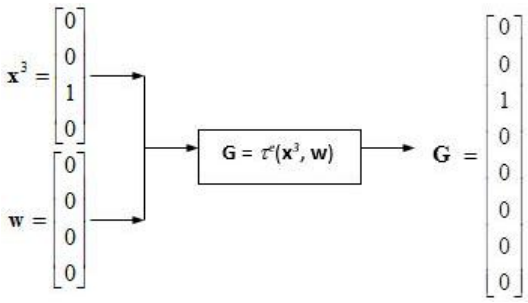



Se opera la memoria autoasociativa Alfa-Beta max V con F

$$\mathbf{R} = \mathbf{V} \Delta_\beta \mathbf{F} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Se realiza la contracción: $\mathbf{r} = \tau^c(\mathbf{R}, \mathbf{4}) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$

Debido a que **r** no es un vector *one-hot*, entonces se continúa con el algoritmo
Se construye el vector **u**y con el patrón $\mathbf{x}^3$, se construye la expansión




Se opera la memoria autoasociativa Alfa-Beta *min* $\wedge$ con **G**

$$\mathbf{S} = \Lambda\nabla_{\beta}\mathbf{G} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

Se realiza la contracción: $\mathbf{s}=\tau^{c}\ \mathbf{(S,4)} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix}$

El vector **s** no es un vector zero-hot, por lo que se continúa con el proceso.

Se realiza la operación AND entre el vector **r** y el vector negado de **s**,$\bar{S}$, para obtener el vector **t**

$$\mathbf{t} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \Lambda \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Con esta operación se obtiene el tercer vector *one-hot*

Se obtiene el vector $\mathbf{y}^3$ mediante

$$\mathbf{LAy} \cdot t = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix} \bullet \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \mathbf{y}^3$$

# Capítulo 4

# Implementación de la Memoria en la ALU Asociativa

En este capítulo se presenta la creación y aplicación de las memorias asociativas de suma y resta, así como la implementación de cada una de ellas, además el funcionamiento de la $ALU_A$ para realizar las operaciones que ofrece.

La $ALU_A$ realiza las operaciones de suma, resta, multiplicación, división y comparaciones, mediante una serie de procesos que conjuntamente logran el correcto funcionamiento de la misma. Cada etapa del proceso tiene una función fundamental.

El funcionamiento de la $ALU_A$, se basa en dos clases principalmente:

1. Clase base suma.
2. Clase base resta.

Cada una de ellas contiene la creación de sus memorias, que son la base de la recuperación de la $ALU_A$ y los métodos necesarios para el correcto funcionamiento de la misma. En la figura 4.1 se muestra el diagrama UML de la $ALU_A$ ,que indica las clases heredadas dentro del programa, y las dos clases base que son el fundamento de para realizar las operaciones.

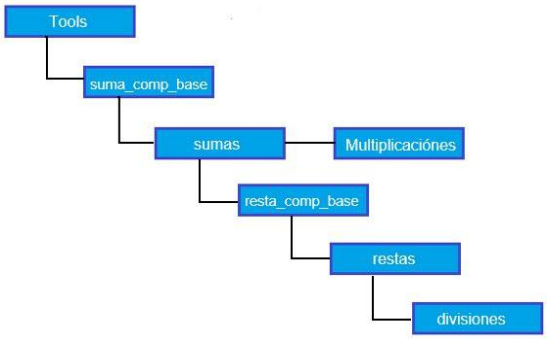



**Figura 4.1:** Diagrama UML de herencia de clases

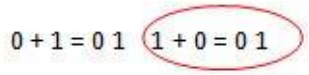
## 4.1 Generación de la Memoria Asociativa Suma

Se creó la memoria asociativa mediante la asociación de cada uno de los patrones de entrada con sus correspondientes patrones de salida.

Para establecer los patrones de entrada y salida que tiene la $ALU_A$, se realizó un análisis para descartar las operaciones aritméticas que se repitieran con los números del 0 a 9, de manera que sólo permanecieran aquellos resultados donde el primer sumando fuera mayor al segundo sumando. Esto con el propósito de descartar operaciones redundantes y no llenar las memorias *máx* y *min* con operaciones innecesarias.

Se realizó sólo con estos números ya que cualquier operación aritmética se basa en la suma o resta de los resultados de éstos.

Entonces si se tiene la operación 0 + 1= 01 y 1 + 0= 01, ambas dan como resultado 01, de modo que se descarta la primera operación ya que su primer sumando es menor al primer sumando de la segunda operación.




Basado en lo anterior, se realizó para todas las combinaciones que existen entre 0 y 9, obteniendo una serie de resultados contenidos en la tabla 4.1.

**Tabla 4.1:** Resultados de Operaciones de Sumas.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0+0=00 | 1+0=01 | 2+0=02 | 3+0=03 | 4+0=04 | 5+0=05 | 6+0=06 | 7+0=07 | 8+0=08 | 9+0=09 |
| | 1+1=02 | 2+1=03 | 3+1=04 | 4+1=05 | 5+1=06 | 6+1=07 | 7+1=08 | 8+1=09 | 9+1=10 |
| | | 2+2=04 | 3+2=05 | 4+2=06 | 5+2=07 | 6+2=08 | 7+2=09 | 8+2=10 | 9+2=11 |
| | | | 3+3=06 | 4+3=07 | 5+3=08 | 6+3=09 | 7+3=10 | 8+3=11 | 9+3=12 |
| | | | | 4+4=08 | 5+4=09 | 6+4=10 | 7+4=11 | 8+4=12 | 9+4=13 |
| | | | | | 5+5=10 | 6+5=11 | 7+5=12 | 8+5=13 | 9+5=14 |
| | | | | | | 6+6=12 | 7+6=13 | 8+6=14 | 9+6=15 |
| | | | | | | | 7+7=14 | 8+7=15 | 9+7=16 |
| | | | | | | | | 8+8=16 | 9+8=17 |
| | | | | | | | | | 9+9=18 |

Con los resultados obtenidos en la tabla 4.1 se obtuvieron los 55 patrones de entrada y salida para realizar la operación de suma en la $ALU_A$ ya que sin importar si se ingresa un sumando, de más de una cifra la suma se realiza digito por digito.

Cada patrón está compuesto de 2 elementos "1,0", es por ello que los 55 patrones de entrada y de salida quedan formados como se muestra en la figura 4.2.

| Entrada | | Salida |
|---|---|---|
| (0,0) | → | (0,0) |
| (1,0) | → | (0,1) |
| (1,1) | → | (0,1) |
| (2,0) | → | (0,2) |
| . | | . |
| . | | . |
| . | | . |
| (9,9) | → | (1,8) |

**Figura 4.2:** Patrones de entrada y salida

La clase base suma consta de varios métodos y propiedades que se observan en la figura 4.3.

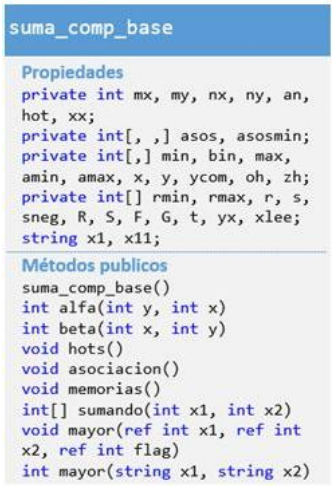



**Figura 4.3:** Clase Base Suma

Dentro de los métodos encontramos la generación de los vectores *one* y *zero hot*, la memoria asociativa de suma, el método para encontrar el valor mayor de los patrones ingresados y el método para sumar un par de patrones, y los vectores necesarios para realizar las operaciones

### 4.1.1 Transformación


Ya que los procesos manejados en la $ALU_A$ se basan en comparaciones, se requiere transformar los patrones de entrada así como los patrones de salida, para homogenizar el proceso, esto se realiza, transformando los patrones de entrada y salida a su equivalente en binario, basados en la tabla 4.2.

**Tabla 4.2:** Números Binarios del 0 a 9.

| | |
|---|---|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |

Cada patrón compuesto de 2 elementos "[0,1]", realizando la transformación, ahora se compone de 8 elementos ya que cada número está formado por 4 bits y al haber 2 elementos surgen 4 bits de cada elemento.

Entonces el patrón original "[0,1]", quedaría transformado "[0 0 0 0,0 0 0 1]", es por ello que los 55 patrones ahora quedan formados como se muestra en la figura 4.4 y la figura 4.5 respectivamente.

```
[0,0]{ 0, 0, 0, 0, 0, 0, 0, 0 }
[1,0]{ 0, 0, 0, 1, 0, 0, 0, 0 }
[1,1]{ 0, 0, 0, 1, 0, 0, 0, 1 }
[2,0]{ 0, 0, 1, 0, 0, 0, 0, 0 }
[2,1]{ 0, 0, 1, 0, 0, 0, 0, 1 }
                :
                .
[9,9]{ 1, 0, 0, 1, 1, 0, 0, 1 }
```

**Figura 4.4:** Patrones de entrada transformados

[0,0]{ 0, 0, 0, 0, 0, 0, 0, 0 }
[0,1]{ 0, 0, 0, 0, 0, 0, 0, 1 }
[1,1]{ 0, 0, 0, 1, 0, 0, 0, 1 }
[2,0]{ 0, 0, 1, 0, 0, 0, 0, 0 }
[2,1]{ 0, 0, 1, 0, 0, 0, 0, 1 }
:
.
[1,8]{ 0, 0, 0, 1, 1, 0, 0, 0 }

**Figura 4.5:** Patrones de salida transformados

El análisis anterior se realizó para presentar los patrones de entrada y salida a la memoria; se presentaron las entradas, las cuales son los números del cero al nueve de forma binaria, ya que son los números con los que se realiza cualquier combinación numérica y los patrones de salidas, es decir los posibles resultados que se pueden obtener con estos números, de esta manera, se realiza las operaciones requeridas en la $\text{ALU}_A$.

### 4.1.2 Transformada vectorial de expansión

En base a los pasos 2 y 3 de la etapa de aprendizaje en sentido x $\longrightarrow$ y, vista en el capítulo 3 se realizaron los vectores *one-hot* y *zero-hot* de los 55 patrones de entrada para generar las memorias *máx* y *min*.

La Figura 4.6 muestra los vectores *one-hot* y *zero-hot* del patron [0,0].

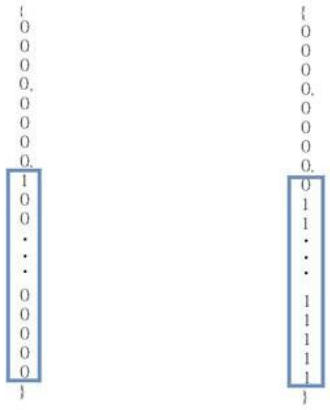


**Figura 4.6:** Vectores one-hot y zero-hot del patrón [0,0].

### 4.1.3 Memorias máx y min

Teniendo los 55 vectores *one-hot* y *zero-hot* respectivamente se generan las memorias autoasociativas Alfa-Beta *máx* y *min*, Fig.4.7 y Fig. 4.8 respectivamente.

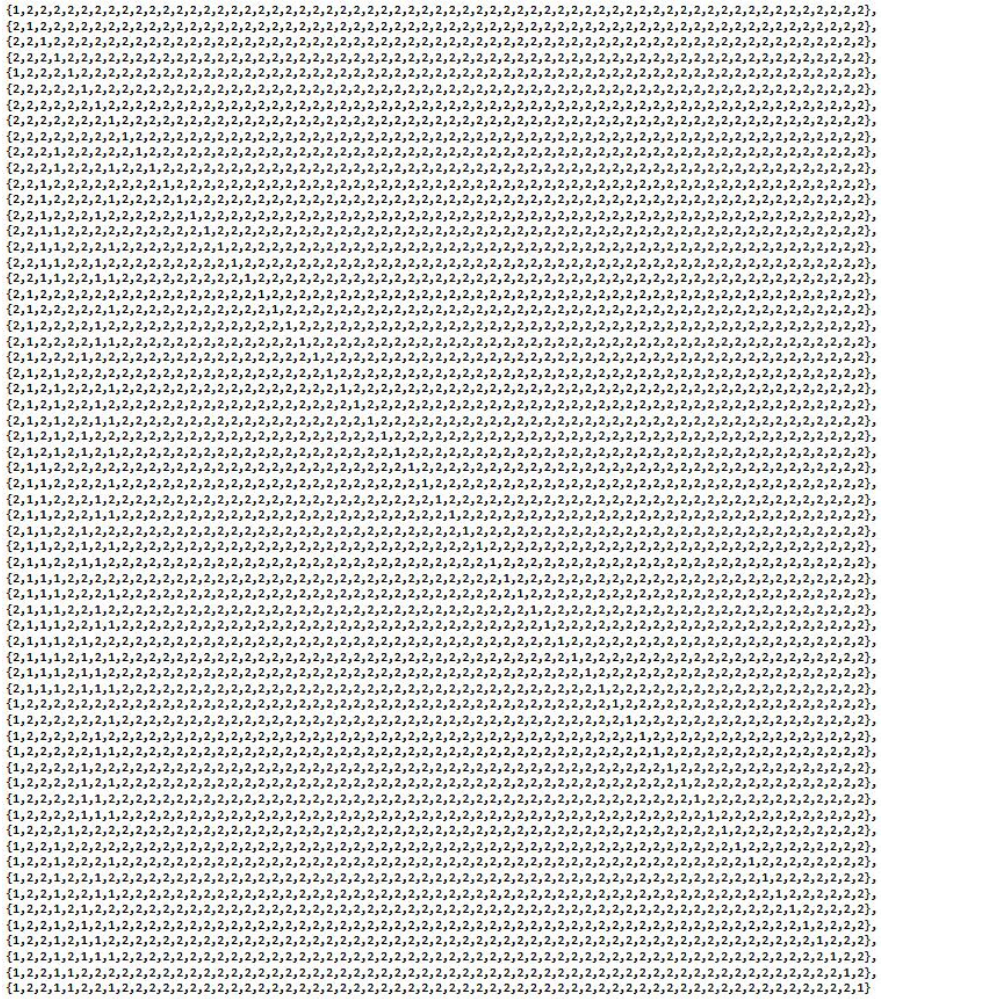


**Figura 4.7:** Memoria autoasociativa máx.

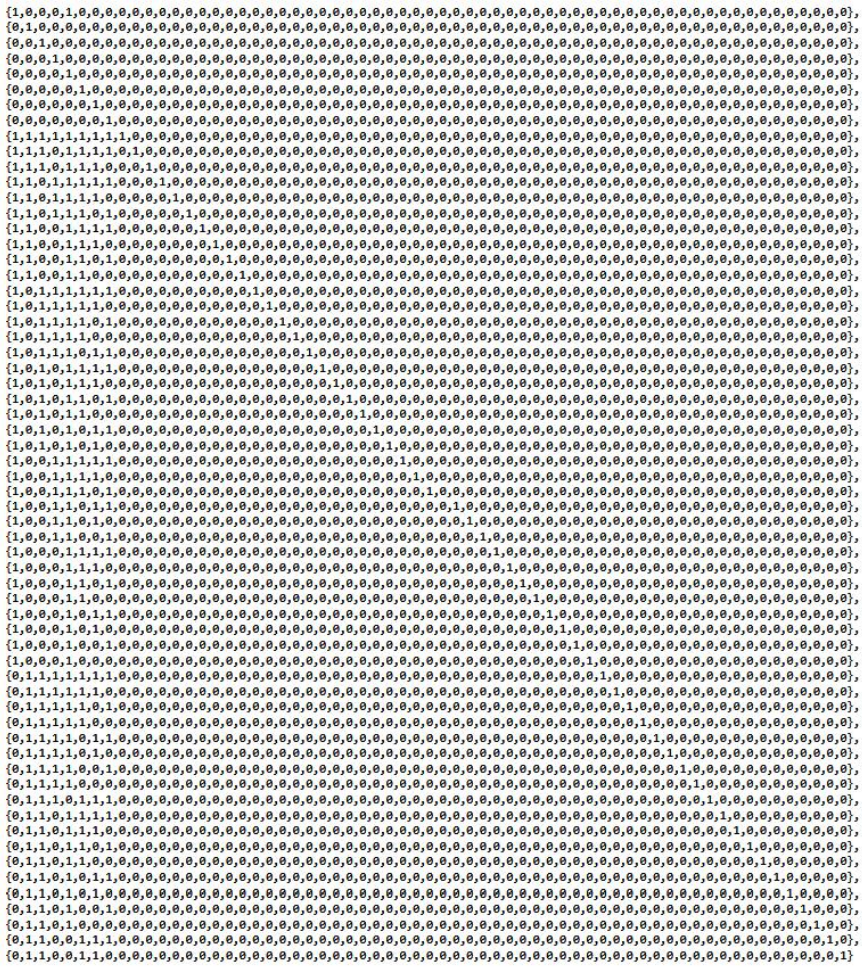
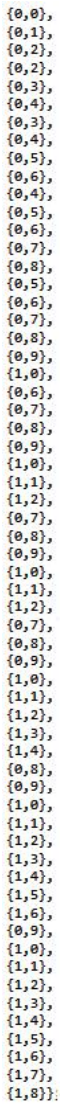


**Figura 4.8:** Memoria autoasociativa min.

### 4.1.4 Linear Associator Modificado

Se crea el Linear Associator Modificado utilizando los patrones de salida y de acuerdo al paso 6 del algoritmo de las etapas 1 y 2.

La figura 4.9 muestra los patrones de salida del *Linear Associator Modificado*



```
{0,0},
{0,1},
{0,2},
{0,2},
{0,3},
{0,4},
{0,3},
{0,4},
{0,5},
{0,6},
{0,4},
{0,5},
{0,6},
{0,7},
{0,8},
{0,5},
{0,6},
{0,7},
{0,8},
{0,9},
{1,0},
{0,6},
{0,7},
{0,8},
{0,9},
{1,0},
{1,1},
{1,2},
{0,7},
{0,8},
{0,9},
{1,0},
{1,1},
{1,2},
{0,7},
{0,8},
{0,9},
{1,0},
{1,1},
{1,2},
{1,3},
{1,4},
{0,8},
{0,9},
{1,0},
{1,1},
{1,2},
{1,3},
{1,4},
{1,5},
{1,6},
{0,9},
{1,0},
{1,1},
{1,2},
{1,3},
{1,4},
{1,5},
{1,6},
{1,7},
{1,8}};
```

**Figura 4.9:** Patrones de salida de Linear Associator Modificado.

## 4.2 Generación de la Memoria Asociativa Resta

La clase base resta al igual que la clase base suma contiene métodos y propiedades que se observan en la figura 4.10.


**Figura 4.10:** Clase Base Resta.

Los métodos y variables utilizados en esta clase son similares a los de la clase suma, con la diferencia que se encuentra en esta clase el método restar, para efectuar dicha operación con un par de patrones.

El número de patrones de entrada para la resta son 210, por ende el tamaño de la memoria surgida es mayor tamaño comparada con la de la suma, esto debido a los "prestamos" que se realizan en una resta cuando el minuendo es menor que el sustraendo, la tabla 4.3 y 4.4 muestran los patrones para la operación resta.

**Tabla 4.3:** Resultados de Operaciones de Resta

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0-0=00 | 1-0=01 | 2-0=02 | 3-0=03 | 4-0=04 | 5-0=05 | 6-0=06 | 7-0=07 | 8-0=08 | 9-0=09 |
| | 1-1=00 | 2-1=01 | 3-1=02 | 4-1=03 | 5-1=04 | 6-1=05 | 7-1=06 | 8-1=07 | 9-1=08 |
| | | 2-2=00 | 3-2=01 | 4-2=02 | 5-2=03 | 6-2=04 | 7-2=05 | 8-2=06 | 9-2=07 |
| | | | 3-3=00 | 4-3=01 | 5-3=02 | 6-3=03 | 7-3=04 | 8-3=05 | 9-3=06 |
| | | | | 4-4=00 | 5-4=01 | 6-4=02 | 7-4=03 | 8-4=04 | 9-4=05 |
| | | | | | 5-5=00 | 6-5=01 | 7-5=02 | 8-5=03 | 9-5=04 |
| | | | | | | 6-6=00 | 7-6=01 | 8-6=02 | 9-6=03 |
| | | | | | | | 7-7=00 | 8-7=01 | 9-7=02 |
| | | | | | | | | 8-8=00 | 9-8=01 |
| | | | | | | | | | 9-9=00 |

Del mismo modo que en la suma, es necesario transformar los patrones de entrada, esto se realizó de acuerdo a lo visto en la sección 4.1.1.

La Figura 4.11 y 4.12 contienen respectivamente los patrones de entrada y salida transformados.

**Tabla 4.4:** Resultados de Operaciones de Resta

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 10-00=10 | 11-00=11 | 12-00=12 | 13-00=13 | 14-00=14 | 15-00=15 | 16-00=16 | 17-00=17 | 18-00=18 | 19-00=19 |
| 10-01=09 | 11-01=10 | 12-01=11 | 13-01=12 | 14-01=13 | 15-01=14 | 16-01=15 | 17-01=16 | 18-01=17 | 19-01=18 |
| 10-02=08 | 11-02=09 | 12-02=10 | 13-02=11 | 14-02=12 | 15-02=13 | 16-02=14 | 17-02=15 | 18-02=16 | 19-02=17 |
| 10-03=07 | 11-03=08 | 11-03=09 | 13-03=10 | 14-03=11 | 15-03=12 | 16-03=13 | 17-03=14 | 18-03=15 | 19-03=16 |
| 10-04=06 | 11-04=07 | 11-04=08 | 13-04=09 | 14-04=10 | 15-04=11 | 16-04=12 | 17-04=13 | 18-04=14 | 19-04=15 |
| 10-05=05 | 11-05=06 | 11-05=07 | 13-05=08 | 14-05=09 | 15-05=10 | 16-05=11 | 17-05=12 | 18-05=13 | 19-05=14 |
| 10-06=04 | 11-06=05 | 11-06=06 | 13-06=07 | 14-06=08 | 15-06=09 | 16-06=10 | 17-06=11 | 18-06=12 | 19-06=13 |
| 10-07=03 | 11-07=04 | 11-07=05 | 13-07=06 | 14-07=07 | 15-07=08 | 16-07=09 | 17-07=10 | 18-07=11 | 19-07=12 |
| 10-08=02 | 11-08=03 | 11-08=04 | 13-08=05 | 14-08=06 | 15-08=07 | 16-08=08 | 17-08=09 | 18-08=10 | 19-08=11 |
| 10-09=01 | 11-09=02 | 11-09=03 | 13-09=04 | 14-09=05 | 15-09=06 | 16-09=07 | 17-09=08 | 18-09=09 | 19-09=10 |
| 10-10=00 | 11-10=01 | 11-10=02 | 13-10=03 | 14-10=04 | 15-10=05 | 16-10=06 | 17-10=07 | 18-10=08 | 19-10=09 |
| | 11-11=01 | 11-11=01 | 13-11=02 | 14-11=03 | 15-11=04 | 16-11=05 | 17-11=06 | 18-11=07 | 19-11=08 |
| | | 12-12=00 | 13-12=01 | 14-12=02 | 15-12=03 | 16-12=04 | 17-12=05 | 18-12=06 | 19-12=07 |
| | | | 13-13=00 | 14-13=01 | 15-13=02 | 16-13=03 | 17-13=04 | 18-13=05 | 19-13=06 |
| | | | | 14-14=00 | 15-14=01 | 16-14=02 | 17-14=03 | 18-14=04 | 19-14=05 |
| | | | | | 15-15=00 | 16-15=01 | 17-15=02 | 18-15=03 | 19-15=04 |
| | | | | | | 16-16=00 | 17-16=01 | 18-16=02 | 19-16=03 |
| | | | | | | | 17-17=00 | 18-17=01 | 19-17=02 |
| | | | | | | | | 18-18=00 | 19-18=01 |
| | | | | | | | | | 19-19=00 |

```
[0,0]{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }
[1,0]{ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0 }
[1,1]{ 0, 0, 0, 0, 1, 0, 0, 0, 0, 1 }
[2,0]{ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0 }
[2,1]{ 0, 0, 0, 1, 0, 0, 0, 0, 0, 1 }
                :
                .
[19,19]{1, 0, 0, 1, 1, 1, 0, 0, 1, 1}
```

**Figura 4.11:** Patrones de Entrada Transformados.

```
{0,0}
{0,1}
{0,0}
{0,2}
{0,1}
  :
  .
{0,0}
```

**Figura 4.12:** Patrones de Salida Transformados.

## 4.3 Implementación de la Memoria Asociativa Suma

El proceso que realiza la $ALU_A$ para realizar la entrega de un resultado de la operación de suma es semejante al que se maneja al realizar las operaciones manualmente.

En la $ALU_A$ se realizar un proceso similar ya que se suma un par de números llamados sumandos.

En el siguiente ejemplo se realiza la suma de un par de numero donde se muestra el proceso realizado para obtener el resultado manualmente y como lo va realizando la $ALU_A$.

La figura 4.13 ilustra el proceso llevado a cabo en la $ALU_A$.

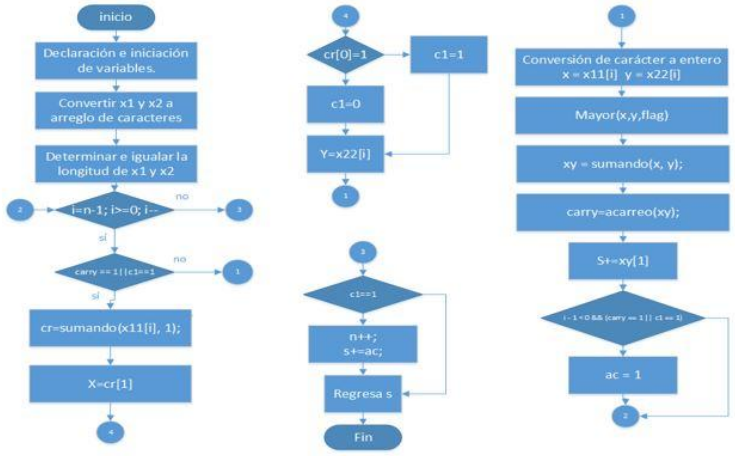



**Figura 4.13:** Diagrama de flujo para la implementación de la Memoria Suma

Cabe mencionar que para el manejo de números con punto flotante se limita a cuatro decimales y el resultado máximo que puede entregar esta en el intervalo de los valores de un entero de 32 bits con signo.

Para realizar la suma 3.7+9.24 en la $ALU_A$, se ingresan en la GUI (véase Figura 4.22) del programa los sumandos en este caso, el usuario ingresar los dos sumandos.

El primer sumando

$$X1= 3.7$$

Enseguida se ingresa el segundo sumando

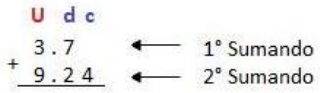
$$X2=9.24$$

Y escoge la operación a realizar, en este caso suma que esta denotada con el código "000". Véase figura 4.22.

La suma de los números 3.7 + 9.24 se ordenarían cada digito en su correspondiente columna dependiendo el peso que tenga en la suma, es decir el primer

número de derecha a izquierda representa las unidades, el segundo las decenas y continuaría sucesivamente, de existir punto decimal dentro de la suma el primer número de izquierda a derecha después del punto representa los decimales, el segundo las centésimas y continua según el peso que tenga.

De este modo la suma anterior se representa de la siguiente forma:




Donde:

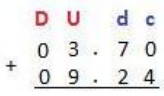
U: Representa las unidades.

d: Representa las décimas.

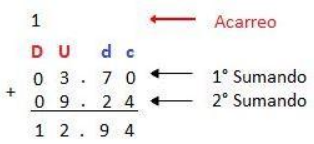
c: Representa las centésimas

Lo siguiente es convertir ambos sumandos a una cadena de caracteres para determinar e igualar la longitud de ambos sumandos, en caso de existir punto decimal se quita el punto decimal de la operación y para igualar la longitud de rellana los espacios con ceros.


Manualmente el proceso que se realiza es similar al que realiza la $ALU_A$, para igualar las longitudes de los sumando, se agregan ceros en la columna del sumando que requiera ser igualado, aunque manualmente se sigue conservando el punto.




Realizado lo anterior se comienza con la suma iniciando desde las centésimas para el ejemplo anterior.



Dentro de la $\text{ALU}_A$, se presentan los patrones de entrada para dar paso al comienzo de los ciclos para la recuperación y se comienza a trabajar los vectores con la memoria autoasociativa.


Para recuperar el resultado asociado a los patrones de entrada ingresados en la $\text{ALU}_A$, se presentan los dos patrones de entrada ingresados por el usuario y se aplica la transformada vectorial de expansión del vector x utilizando como argumentos el vector $\text{x}^k$ y el vector $u$, para obtener $F$.

Se opera la memoria autoasociativa Alfa-Beta máx. V con $F$, para obtener el vector $r$.

Si r es un vector one-hot, entonces $r$ es el k-ésimo vector one-hot, $h^k$, (Basado en el Teorema 4.2) entonces, se realiza la operación LAy * $r$, lo que resultará en el vector $\text{y}^k$ correspondiente y se entrega el resultado; si no, entonces se construye el vector $w$.

Se aplica la transformada vectorial de expansión del vector x utilizando como argumentos el vector $x^k$ y el vector $w$, para obtener $G$.

Se opera la memoria autoasociativa Alfa-Beta *min* con $G$, para obtener un vector $S$, al cual se le aplica la transformada de contracción.

Ahora se pregunta si $s$ es un vector zero-hot, de ser un vector zero-hot se entrega el resultado; si no

Se realiza la operación AND entre el vector $r$ y el vector negado de $s$, de tal modo que obtendremos un vector $t$.

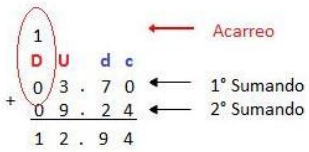
Entonces se aplica la operación AND para el Lay por el vector $t$, para obtener el patrón asociado.

Al cumplirse el ciclo, se pregunta si existe un acarreo, si no existe, se revierte el proceso y se convierte la cadena a entero para comparar cual es el número mayor entre ambos sumandos.

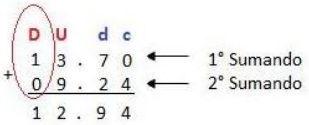
Se llama al método sumando y se activa la bandera de acarreo.

Continua el procedimiento hasta finalizar el ciclo, entonces si existe un acarreo se realiza el mismo procedimiento que al realizar manualmente la suma.

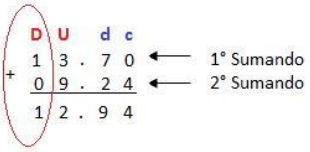
Se suma el 1 del acarreo más el 0 del primer sumando que dan un total de 1.




Enseguida se suma el 1 que resulto de sumar el acarreo más el primer sumando con el 0 del segundo sumando.




Al no existir más acarreos y no haber más sumandos damos por finalizada la operación.




Completo el proceso de suma en la $\text{ALU}_A$, se convierte el resultado en cadena y se coloca el punto decimal en su posición correspondiente.

El último paso es mostrar en la GUI el resultado.

## 4.4 Implementación de la Memoria Asociativa Resta

La resta o sustracción es la operación que representa la eliminación de objetos de una colección.

El proceso para la obtención del resultado de una resta en la $\text{ALU}_A$, tiene similitudes con los pasos para llevar a cabo la suma, la figura 4.14 muestra el proceso que se sigue para realizar la resta.

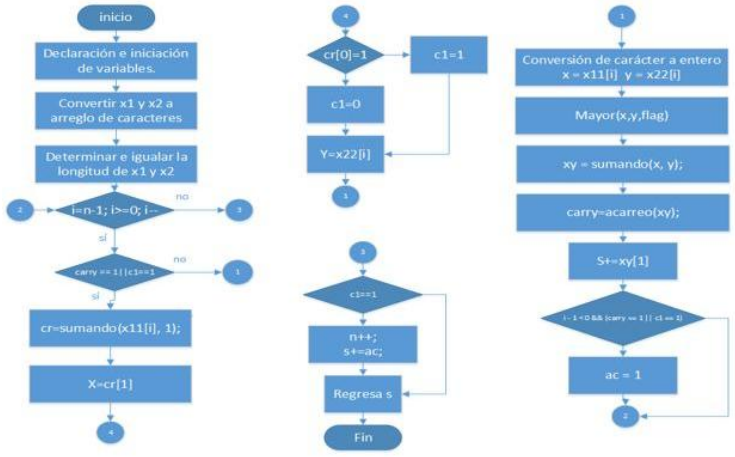



**Figura 4.14:** Diagrama de flujo para la implementación de la Memoria Resta

Para generar la memoria asociativa de resta se sigue el proceso realizado en la sección 4.1., pero esta vez realizando la operación de resta.

El número de patrones generados de entrada y salida aumentó y por ende el tamaño de la memoria aumento significativamente, por esta razón no se encuentra en el documento.

En la resta se implementan y se trabajan nuevamente los vectores para la operación de la memoria.

La forma de operar en esta clase es similar a la clase base suma, se presentan los patrones de entrada a la memoria, se convierten a un arreglo de caracteres y se iguala su longitud, entre ambos patrones.

Para realizar la resta 653-391 en la $ALU_A$, se ingresan en la GUI (véase Figura 4.22) del programa el minuendo y sustraendo de la operación. El usuario ingresar cada uno respectivamente.

El minuendo

$$X1=653$$

Enseguida se ingresa el sustraendo

$$X2=391$$

Y escoge la operación a realizar, en este caso suma que esta denotada con el código "001". Véase figura 4.22.

La resta o sustracción de los números 653 - 391 se ordenarían de la siguiente forma:

| | C | D | U | |
|---|---|---|---|---|
| | 6 | 5 | 3 | ⟵ Minuendo |
| − | 3 | 9 | 1 | ⟵ Sustraendo |

El siguiente paso es convertir el minuendo y sustraendo a una cadena de caracteres para determinar e igualar la longitud de ambos patrones de entrada si fuera necesario.

Se comienzan los ciclos para la recuperación y se comienza a trabajar los vectores con la memoria autoasociativa del mismo modo que en la suma.

Para recuperar el resultado asociado a los patrones de entrada ingresados en la $\text{ALU}_A$ del mismo modo que para la suma, se presentan los dos patrones de entrada ingresados por el usuario y se aplica la transformada vectorial de expansión del vector x utilizando como argumentos el vector $x^k$ y el vector $u$, para obtener $F$.

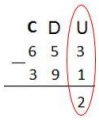
Se opera la memoria autoasociativa Alfa-Beta máx. V con $F$, para obtener el vector $r$.

Si $r$ es un vector one-hot, entonces $r$ es el k-ésimo vector one-hot, $\text{h}^k$, (Basado en el Teorema 4.2) entonces, se realiza la operación LAy * $r$, lo que resultará en el vector $\text{y}^k$ correspondiente y se entrega el resultado; si no, entonces se construye el vector $w$.

Se aplica la transformada vectorial de expansión del vector x utilizando como argumentos el vector $\text{x}^k$ y el vector w, para obtener $G$.

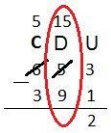
Se opera la memoria autoasociativa Alfa-Beta *min* con $G$, para obtener un vector $S$.

Manualmente, cada dígito del sustraendo se sustrae del dígito por encima de él (minuendo) comenzando de derecha a izquierda dicho de otra forma de unidades a centenas.




Efectuada la primera sustracción se pasa a restar la columna de las decenas, en este caso la siguiente operación a efectuar es "5 - 9" el minuendo es inferior al sustraendo, por lo cual se le suma 10 a este digito del minuendo y se resta uno al digito de su lado izquierdo.

Este 10 es "prestado" y este préstamo es el que activa nuestra bandera en la $\text{ALU}_A$ para conocer los valores de los siguientes patrones a restar.



| | C | D | U |
|---|---|---|---|
| − | ~~8~~ | 5 | 3 |
| | 3 | 9 | 1 |
| | | 6 | 2 |

El proceso anterior la $ALU_A$ lo efectúa comparando los dos valores a restar, y compara cual si el minuendo es mayor al sustraendo de no ser así, se realiza el "préstamo" y se activa la bandera carry y se llama al método restar.

Entonces la resta ahora quedaría con los siguientes valores después de haber realizado el préstamo y efectuada la resta de las decenas.

Continúa el procedimiento hasta finalizar el ciclo, con el método restar.

| | C | D | U |
|---|---|---|---|
| − | 5 | 5 | 3 |
| | 3 | 9 | 1 |
| | | 6 | 2 |

Al no existir más valores para efectuar la resta se da por concluida y se manda a imprimir el resultado obtenido en la GUI.

| | C | D | U |
|---|---|---|---|
| − | 5 | 5 | 3 |
| | 3 | 9 | 1 |
| | 2 | 6 | 2 |

## 4.5 Multiplicación

La multiplicación es una operación que consiste en sumar un número tantas veces como indica otro número es decir sumar el multiplicando tantas veces como indica el multiplicador.

Ejemplo

La multiplicación 8*5 tiene como resultado o producto 40, esto se logra mediante la operación de sumar 5 veces 8.

8*5=40 8+8+8+8+8=40

La $ALU_A$ sigue el mismo principio, por lo cual hace uso de la clase base suma para efectuar la operación.

La figura 4.15, muestra el esquema general del método de multiplicación, en el que se observa cómo es que trabaja y hace uso de la función suma para realizar la asociación y entregar el resultado.

Para realizar la multiplicación 7x3 en la $ALU_A$, se ingresan en la GUI (véase Figura 4.22) del programa el multiplicando y el multiplicador, en este caso, el usuario ingresa en X1 el multiplicando y en X2 el multiplicador.

El multiplicando

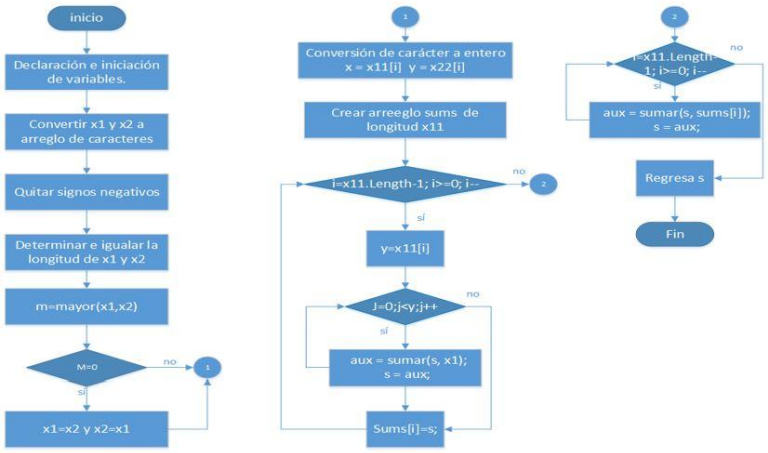



**Figura 4.15:** Diagrama de flujo del méétodo multiplicar

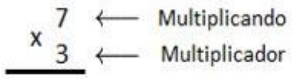
X1=7

Enseguida se ingresa el multiplicador

X2=3

Y escoge la operación a realizar, en este caso suma que esta denotada con el código "010". Véase figura 4.22.

La operación al realizarla manualmente se denotaría de la siguiente manera.




En la $ALU_A$ se realiza el proceso de convertir ambos patrones a una cadena de caracteres.

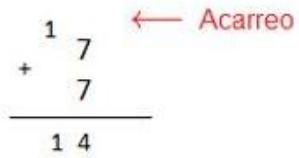
Se quitan los signos o signo negativos de los patrones si existiera, y se igualan las longitudes de las cadenas.

El siguiente paso es buscar el número mayor entre ambos patrones.

Se hace referencia a los métodos de la clase suma para efectuar la multiplicación, esto implica la generación de los vectores y comenzar con las comparaciones para obtener el resultado, esto como se vio en la implementación de la clase suma.

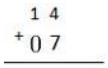
Manualmente se debe sumar 3 veces 7, llamando a la clase suma primero se sumaría 2 veces el 7.

La operación anterior nos arrojaría el siguiente resultado, donde habría un acarreo.

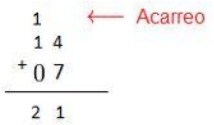



Ahora se ha sumado 2 veces ya el 7, el factor multiplicador indica que deben ser 3, entonces ahora al resultado anterior se suma 7 nuevamente.

Ahora bien la $ALU_A$ tendría ahora dos nuevos patrones de entrada para el método suma 14 y el siete que hace falta sumar, y la $ALU_A$ ahora completa las longitudes de ambos números, lo que manualmente se representaría de la forma siguiente.




Se realiza la operación y se termina el ciclo de suma ya que se completaron las iteraciones del valor del multiplicador.




Finalmente se devuelve el producto de la multiplicación el cual para el ejemplo anterior es 21 y se muestra en la GUI del programa.

## 4.6 División

La división por el contrario es una resta sucesiva que consiste en averiguar cuantas veces un número (divisor), está contenido en otro número (dividiendo), el resultado recibe el nombre de cociente.

La división 12/3 da como resultado 4, el proceso para llegar a este resultado es restar a 12 de tres en tres unidades, hasta tener como resultado o residuo 0, y el número de veces restadas u contenidas es el cociente o resultado.

12-3=9 9-3=6 6-3=3 3-3=0

La Figura 4.16 contiene el diagrama del flujo del metodo division, el cual muestra el proceso que realiza la $ALU_A$ a partir de la entrada de patrones, hasta entregar el resultado final.

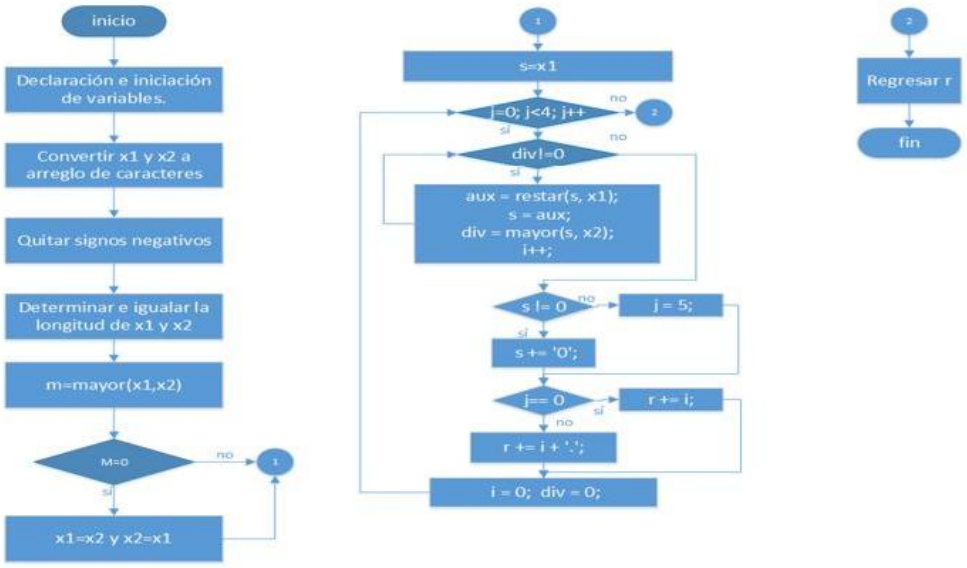



**Figura 4.16:** Diagrama de flujo del método división

Para el caso de la división $8 \div 4$ en la $ALU_A$, se ingresan en la GUI (véase Figura 4.22) del programa el dividiendo y el divisor, en este caso, el usuario ingresa en X1 el dividiendo y en X2 el divisor.

El dividiendo

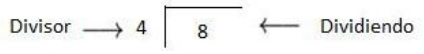
X1=8

Se ingresa el divisor

X2=4

Y escoge la operación a realizar, en este caso suma que esta denotada con el código "011". Véase figura 4.22.

La operación al realizarla manualmente se denotaría de la siguiente manera.




Del mismo modo que para realizar la resta, se convierte ambos patrones a una cadena de caracteres.

Para realizar la división por restas sucesivas, es necesario eliminar el punto si existe en alguno de los patrones para realizar la operación.

De igual forma se eliminan los signos o signo negativos de los patrones si existiera, y se igualan las longitudes de las cadenas de ambos patrones.

De nueva cuenta se busca el número mayor entre ambos patrones, y se llaman los

métodos de la clase basa resta para efectuar la división.
En este caso en la $ALU_A$ se comienza a restar el dividiendo menos el divisor.

$$\begin{array}{r} 8 \\ -\ 4 \\ \hline \end{array}$$

Al igual que en la resta se verifica si es mayor ahora el minuendo, en este caso lo es, por ende no existe necesidad de pedir prestado.

$$\begin{array}{r} 8 \\ -\ 4 \\ \hline 4 \end{array}$$

El resultado anterior se compara para saber si es igual a 0, de no ser así se continúan con los ciclos de recuperación, y se llama al método resta nuevamente.
Por lo cual la siguiente resta quedaría como se muestra.

$$\begin{array}{r} 4 \\ -\ 4 \\ \hline 0 \end{array}$$

Ahora el resultado es 0, por consiguiente el proceso de división ha finalizado.
El cociente o resultado de la operación es el número de veces que se realizó la resta, en este caso se realizaron 2 restas para llegar a obtener como residuo 0, por lo tanto en la GUI del programa se muestra el número 2.

## 4.7 Simulador Aritmético Lógico

Para desarrollar la aplicación de interfaz gráfica, se seleccionó un compilador y un lenguaje de programación de acuerdo a las necesidades que demanda la aplicación y el conocimiento de las herramientas del lenguaje de programación.

Para efectuar la GUI de la aplicación se ha seleccionado el lenguaje de programación C♯, porque cuenta con las herramientas, el poder de procesamiento y operación necesarias en el proceso, además de ser una herramienta de programación orientada a objetos.

Por otro lado se podría basar el programa en C++, pero existe una polémica entre C++ y C♯, pues C++ es una herramienta que podría estar un escalón arriba de C♯, puesto que la mayoría de las aplicaciones de control e interfaz de usuario están basadas en código C++, pero el surgimiento de C♯ se basa en co rregir los errores que presenta C++, como son la extracción de librerías y la forma

de depurar la información.

Además realizar una aplicación basada en C♯ permite ampliar los conocimientos de una tecnología poderosa que va en ascenso en el desarrollo de las tecnologías de la información.

Para trabajar en una aplicación con ventanas y controles de visual C♯, hay que seguir las siguientes instrucciones:

1. Archivo.

2. Nuevo Proyecto.

3. Plantillas.

4. Visual C♯

5. Aplicación para Windows Forms.

La figura 4.17 muestra la interfaz de creación de un proyecto para Windows Forms en C♯.

Antes de aceptar el tipo de plantilla, se debe asignar un nombre al proyecto en el ítem "Nombre" y se da click en aceptar.

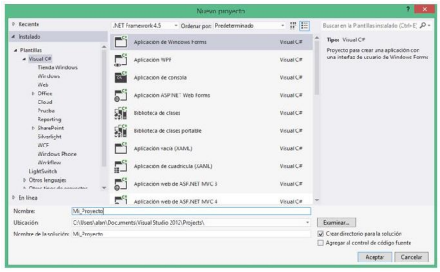


**Figura 4.17:** Creación de un proyecto en C♯.

La figura 4.18, muestra la ventana principal del programa que ejecuta las acciones de este proyecto.

Para trabajar con la ventana y las herramientas de visual estudios, se puede acondicionar el ambiente de trabajo.

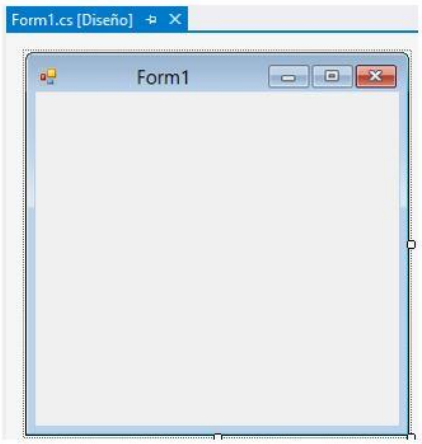



**Figura 4.18:** Ventana de un proyecto de C♯.

Se modificas las propiedades: "Name","Text", "Font", de la ventana principal usando las ventana de propiedades de visual C♯.

En la figura 4.19, se muestra la ventana de propiedades de la ventana principal, como se observa, existe un campo llamado (Name), este campo permite declarar cual será el nombre la ventana principal.

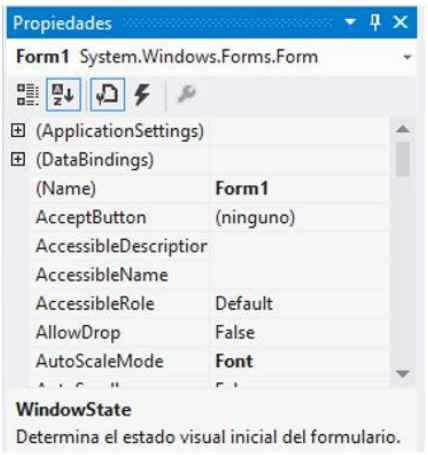



**Figura 4.19:** Ventana de Propiedades.

En la figura 4.20 se muestra como queda el ambiente visual de desarrollo donde se realizó la GUI.

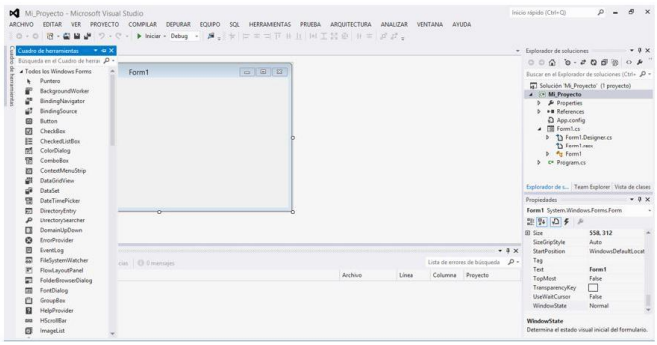


**Figura 4.20:** Acondicionamiento del ambiente de programación en visual C♯.

Ahora, se insertan una serie de botones y cajas de texto que sirvan para elegir la operación a realizar en la $ALU_A$, y para recibir los patrones de entrada respectivamente.

La figura 4.21 muestra la inserción de un botón al formulario.

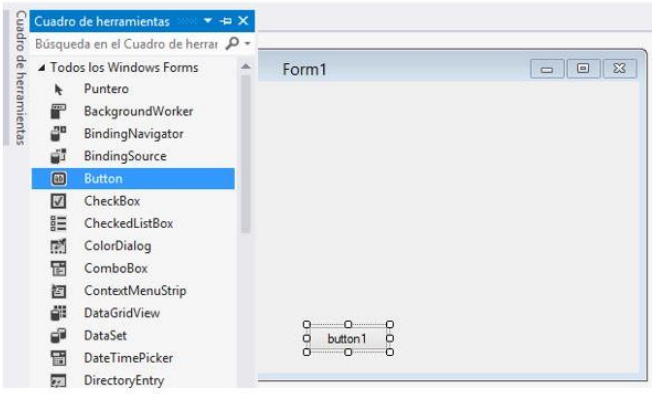



**Figura 4.21:** Inserción de un botón al formulario

Después de insertar botones, cajas de texto, y adecuarlos a nuestras necesidades para la aplicación, finalmente la GUI de la $ALU_A$ se muestra en la figura 4.22.

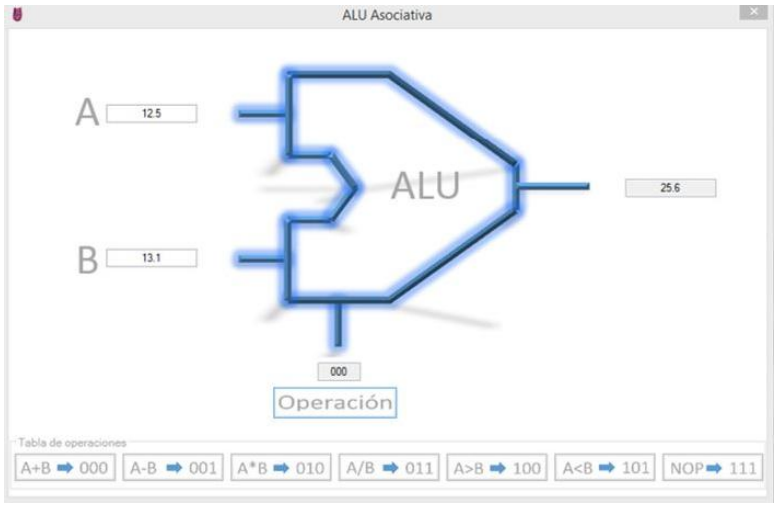



**Figura 4.22:** Interfaz Gráfica de Usuario de la $ALU_A$

La GUI cuenta con dos cajas de texto donde se ingresan los patrones de entrada para efectuar una de las operaciones que el usuario puede elegir en la parte inferior de la ventana. Con el botón "Operación" se efectúa la acción para entregar finalmente la asociación a los patrones de entrada ingresados o dicho de otra forma el resultado esperado en una caja de texto.

# 4. IMPLEMENTACIÓN DE LA MEMORIA EN LA ALU ASOCIATIVA

# Capítulo 5

# Resultados Experimentales

Para realizar las pruebas necesarias del simulador de la $ALU_A$ se inició con las pruebas elementales, estas pruebas consistieron en ingresar una serie de operaciones para comprobar el funcionamiento de aprendizaje y recuperación de las memorias para obtener el resultado requerido siempre manteniéndose el intervalo de funcionamiento de los valores de un entero de 32 bits con signo, la tabla 5.1 muestra las operaciones realizadas con su respectivo resultado.

**Tabla 5.1:** Operaciones aplicadas al Simulador de la $ALU_A$

| Suma | Resta | Multiplicación | División |
|---|---|---|---|
| 11 + 2=13 | 11 - 2=09 | 11 * 2=22 | 11 / 2=5.5 |
| 102 + 67=169 | 102 - 67=35 | 102 * 67=6834 | 102 / 67=1.5223 |
| 99 + 10=109 | 99 - 10=89 | 99 * 10=990 | 99 / 10=9.9 |
| 98 + 54=152 | 98 - 54=44 | 54 * 98= 5292 | 98 / 54=1.8148 |
| 33 + 9=42 | 33 - 9 = 24 | 33 * 9=297 | 33 / 9=3.66 |
| 374.437 + 43.24=417.677 | 374.437 - 43.24=331.197 | 374.437 * 43.24=16190.65 | 374.437 / 43.24=8.659 |
| 253 + 153.53=406.53 | 253 - 153.53= 94.47 | 253 * 153.53=38843.09 | 253 / 153.53=1.647 |
| 34 + 3= 37 | 34 - 3=31 | 34 * 3=102 | 34 / 3=11.33 |
| 12 + 7=19 | 12 - 7=05 | 12 * 7=84 | 12 / 7=1.71 |
| 13.6 + 8.9= 22.5 | 13.6 - 8.9= 4.7 | 13.6 * 8.9= 121.04 | 13.6 / 8.9=1.528 |
| 1.1 + 0.5=1.6 | 1.1 - 0.5=0.6 | 1.1 * 0.5=0.55 | 1.1 / 0.5=2.2 |
| 235.4 + 35.2=270.6 | 235.4 - 35.2=200.2 | 235.4 * 35.2=8286.08 | 235.4 / 35.2=6.687 |
| 34.2 + 23.51=57.71 | 34.2 - 23.51 =10.69 | 34.2 * 23.51=804.042 | 34.2 / 23.51=1.452 |
| 4.52 + 4.1=8.62 | 4.52 - 4.1=0.42 | 4.52 * 4.1=18.532 | 4.52 / 4.1=1.1024 |
| 12.4 + 4.594=16.994 | 12.4 - 4.594=7.806 | 12.4 * 4.594=56.965 | 12.4 / 4.594=2.699 |
| 23.04 + 3.423=26.463 | 23.04 - 3.423=19.617 | 23.04 * 3.423=78.865 | 23.04 / 3.423=6.730 |
| 1343 + 6.43=1349.43 | 1343 - 6.43= 1336.57 | 1343 * 6.43=8635.49 | 1343 / 6.43=208.864 |
| 341 + 45.5=386.5 | 341 - 45.5=295.5 | 341 * 45.5=15515.5 | 341/ 45.5=7.494 |
| 32 + 32=64 | 32 - 32=00 | 32 * 32=1024 | 32 / 32=01 |
| 04 + 04=08 | 04 - 04=00 | 04*04=16 | 04/04=01 |

Las pruebas anteriores que se le realizaron al simulador de la $ALU_A$ fueron aprobadas con un 100 porciento de efectividad obteniendo siempre el resultado deseado.

Las pruebas de mayor que "A>B" y menor que "A<B" así como la NOP, arrojaron la misma efectividad del 100 porciento.

Realizado lo anterior se realizó una prueba más para probar la dependencia si es que existiera de las memorias máx y mín, para la operación de simulador.

Para ello se procedió a quitar primero la memoria mín, y se realizaron las mismas pruebas sin esta memoria, para esta prueba solo se marcó como correcta "√" si arrojaba el resultado correcto e incorrecta " x " en caso contrario.

Entonces de las 80 operaciones realizadas obtuvimos los siguientes resultados.

**Tabla 5.2:** Excluyendo Memoria mín

| Suma | Resta | Multiplicación | División |
|---|---|---|---|
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | √ | x | √ |
| x | √ | x | √ |

Observando la tabla anterior se vio que para las operación que tenía el mismo elemento en el minuendo y sustraendo, dividiendo y divisor para la operaciones de resta y división respectivamente el resultado fue correcto, obteniedo unicamente un 5 por ciento de recuperación.

Luego entonces se procedió a realizar una serie de operaciones para verificar el resultado anterior y se obtuvieron los resultados de la tabla 5.3.

**Tabla 5.3:** Operaciones Excluyendo Memoria mín

| Suma | Resta | Multiplicación | División |
|---|---|---|---|
| 5+5=10 | 5-5=00 | 5*5=25 | 5/5=01 |
| 2+2=04 | 2-2=00 | 2*2=4 | 2/2=01 |
| 3+3=06 | 3-3=00 | 3*3=09 | 3/3=01 |
| 1+1=02 | 1-1=00 | 1*1=01 | 1/1=01 |
| 7+7=14 | 7-7=00 | 7*7=49 | 7/7=01 |

De nueva cuenta se marcó como correcta "$\surd$" si arrojaba el resultado correcto e incorrecta " x " en caso contrario.Y obtuvimos los resultados de la tabla 5.4.

**Tabla 5.4:** Resultados Excluyendo Memoria mín

| Suma | Resta | Multiplicación | División |
|---|---|---|---|
| x | $\surd$ | x | $\surd$ |
| x | $\surd$ | x | $\surd$ |
| x | $\surd$ | x | $\surd$ |
| x | $\surd$ | x | $\surd$ |
| x | $\surd$ | x | $\surd$ |

De acuerdo a la tabla a la tabla 5.4 se observa que ahora con estas 20 operaciones obtuvimos el 50 por ciento de recuperación trabajando únicamente con la memoria *máx* para estas operaciones.

El proceso ahora se realizó quitando la memoria *máx* y trabajando únicamente con la memoria *min*. Para realizar esta prueba se tomaron las operaciones de la tabla 5.3 y nuevamente se marcó como correcta "$\surd$" si arrojaba el resultado correcto e incorrecta " x " en caso contrario.

**Tabla 5.5:** Excluyendo Memoria máx

| Suma | Resta | Multiplicación | División |
|---|---|---|---|
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| x | x | x | x |
| $\surd$ | $\surd$ | $\surd$ | $\surd$ |
| $\surd$ | $\surd$ | $\surd$ | $\surd$ |

Ahora se notó que trabajando únicamente con la memoria mín para la operación que tenía el mismo elemento a operar se obtenían correctamente los resultados, entonces de las 80 operaciones que se realizaron se obtuvo un 10 por ciento de recuperación.

Entonces se trabajó con la tabla 5.3 para obtener los resultados de correcto o incorrecto pero trabajando con la memoria *mín* y obtuvimos los resultados que se muestran en la tabla 5.6.

**Tabla 5.6:** Resultados Excluyendo Memoria máx

| Suma | Resta | Multiplicación | División |
|---|---|---|---|
| √ | √ | √ | √ |
| √ | √ | √ | √ |
| √ | √ | √ | √ |
| √ | √ | √ | √ |
| √ | √ | √ | √ |

Ahora en la tabla 5.6 se observa que al realizar dichas operaciones se obtuvo el 100 por ciento de recuperación trabajando únicamente con la memoria *mín* para estas 20 operaciones.

# Capítulo 6

# Conclusiones

## 6.1 Conclusiones Finales

- Se estudiarón los diferentes modelos asociativos.
- Se conoció el algoritmo de las memorias asociativas Alfa-Beta y Bidireccionales Alfa-Beta.
- Se implementarón las memorias asociativas bidireccionales Alfa-Beta.
- Se diseñó e implementó una unidad aritmética lógica asociativa utilizando memorias asociativas Alfa-Beta a nivel software utilizando Visual Estudio 2013.
- Las memorias alfa y Beta juegan un papel indispensable en este trabajo y están relacionadas una con otra, ya que si falta una de ellas no se arrojarían los resultados correctos, como quedo expuesto en el capítulo 5.
- Finalmente se comprobó el caso de estudio de las memorias asociativas, verificando el rendimiento de dicha memorias, la cual se pudo justificar que este tipo de memoria tiene un factor de olvido igual a cero; todo lo aprendido se recupera satisfactoriamente.

## 6.2 Contribuciones

El presente trabajo da pauta para la investigación e implementación de las memorias asociativas en la aérea de arquitectura de computadoras, en la cual no se habían implementado antes.

## 6.3 Trabajo Futuro

En este trabajo se realizó el desarrollo y la implementación a nivel software de una $\text{ALU}_A$, para la cual el siguiente paso sería implementarla en un FPGA.

Del mismo modo se puede trabajar en la resolución de la $\text{ALU}_A$, para aumentar

la capacidad de entrega de resultados, así mismo se aumentara el número de decimales que trabaje.

Finalmente queda el presente trabajo para que se puedan hacer las pruebas necesarias de rendimiento con una ALU tradicional.

# Apéndice A

# Código de implementación de la memoria suma.

En el apéndice A se encuentra el código que realiza la implementación de la memoria suma, así como algunas de sus propiedades, métodos para la obtención de los vectores F, R, la generación de la memoria y el método para obtener el número mayor.

```
            this.ny = 2;//numero de elementos de cada clase
            this.my = 55;//numero de clases totales
            this.mx = 55;// numero de patrones totales
            hot = 63;//tamaño de los vectores one hot y zero hot 55 por las clases y 8 por patrones
            this.nx = 8;//numero de elementos de cada patron
            xlee = new int[nx];//patron leido
            asos = new int[hot, my, hot];//autoasociacion de los vectores one hot
            asosmin = new int[hot, my, hot]; //autoasociacion de los vectores zero hot
            min = new int[hot, hot];//memoria min
            max = new int[hot, hot];//memoria max
            amin = new int[hot, hot];//memoria min beta G
            amax = new int[hot, hot];//memoria max beta F
            zh = new int[mx, hot];//vectores zero hot
            oh = new int[mx, hot];//vectores one hot
            yx = new int[ny];
            rmax = new int[ny];
            rmin = new int[ny];
            r = new int[mx];
            s = new int[mx];
            sneg = new int[mx];
            t = new int[mx];
            R = new int[hot];
            S = new int[hot];
            F = new int[hot];
            G = new int[hot];
public int alfa(int y, int x)
        {
            if ((y == 0 && x == 0) || (y == 1 && x == 1)) return (1);
            if (y == 0 && x == 1) return (0);
            if (y == 1 && x == 0) return (2);
            return (3);
        }
        public int beta(int x, int y)
```

```
        {
            if (((y == 0) && (x == 0)) || ((y == 0) && (x == 1)) || ((y == 1) && (x == 0))) return (0);
            if (((y == 1) && (x == 1)) || ((y == 2) && (x == 1)) || ((y == 2) && (x == 0))) return (1);
            else return (0);
        }
        public void hots()
        {
            for (int i = 0; i < mx; i++)
            {
                for (int j = 0; j < hot; j++)
                {
                    if (j < nx) oh[i, j] = zh[i, j] = x[i, j];
                    else
                    {
                        if (i == (j - nx)) Zh[i, j] = 0;
                        else Zh[i, j] = 1;
                        if (i == (j - nx)) Oh[i, j] = 1;
                        else Oh[i, j] = 0;
                    }
                }
            }
        }
        public void asociacion()
        {
            for (int i = 0; i < my; i++)
                for (int j = 0; j < hot; j++)
                    for (int k = 0; k < hot; k++)
                    {
                        asos[j, i, k] = alfa(oh[i, j], oh[i, k]);
                        asosmin[j, i, k] = alfa(zh[i, j], zh[i, k]);
                    }
        }
        public void memorias()//genera memorias max y min
        {
            int maxa, mina;
            maxa = asos[0, 0, 0];
            mina = asosmin[0, 0, 0];
            for (int i = 0; i < hot; i++)
            {
                for (int j = 0; j < hot; j++)
                {
                    for (int k = 0; k < mx; k++)
                    {
                        if (asos[i, k, j] > maxa) maxa = asos[i, k, j];
                        if (asosmin[i, k, j] < mina) mina = asosmin[i, k, j];
                    }
                    min[i, j] = mina;
                    max[i, j] = maxa;
                    maxa = asos[j, 0, j];
                    mina = asosmin[j, 0, j];
                }
            }

        }

public int[] sumando(int x1, int x2)
        {

            int maxa, mina;
            for (int i = 0; i < 4; i++)
            {
                xlee[i] = bin[x1, i];
```

```
    xlee[i + 4] = bin[x2, i];
}
for (int i = 0; i < hot; i++)
{
    if (i < nx) G[i] = F[i] = xlee[i];
    else
    {
        G[i] = 0;
        F[i] = 1;
    }
}
//obtencion de R
for (int i = 0; i < hot; i++)
{
    for (int j = 0; j < hot; j++)
    {
        amax[i, j] = beta(F[j], max[i, j]);
    }
}
for (int i = 0; i < hot; i++)
{
    mina = amax[i, i];
    for (int j = 0; j < hot; j++)
    {
        if (amax[i, j] < mina) mina = amax[i, j];
    }
    R[i] = mina;
}

//obtencion de S

for (int i = 0; i < hot; i++)
{
    for (int j = 0; j < hot; j++)
    {
        amin[i, j] = beta(G[j], min[i, j]);
    }
}

for (int i = 0; i < hot; i++)
{
    maxa = amin[i, i];
    for (int j = 0; j < hot; j++)
    {
        if (amin[i, j] > maxa) maxa = amin[i, j];
    }
    S[i] = maxa;
}
//colas r y s
for (int i = nx; i < hot; i++)
{
    s[i - nx] = S[i];
    r[i - nx] = R[i];
    if (s[i - nx] == 0) sneg[i - nx] = 1;
    else sneg[i - nx] = 0;
    t[i - nx] = sneg[i - nx] & r[i - nx];

}

for (int i = 0; i < ny; i++)
{
    yx[i] = 0;
```

```
            for (int j = 0; j < mx; j++)
            {
                yx[i] += y[j, i] * t[j];
            }
        }
        return yx;
    }

public void mayor(ref int x1, ref int x2, ref int flag)
    {
        xx = x2;
        int maxa, mina;
        for (int i = 0; i < 4; i++)
        {
            xlee[i] = bin[x1, i];
            xlee[i + 4] = bin[x2, i];
        }
        for (int i = 0; i < hot; i++)
        {
            if (i < nx) G[i] = F[i] = xlee[i];
            else
            {
                G[i] = 0;
                F[i] = 1;
            }
        }
        //obtencion de R
        for (int i = 0; i < hot; i++)
        {
            for (int j = 0; j < hot; j++)
            {
                amax[i, j] = beta(F[j], max[i, j]);
            }
        }
        for (int i = 0; i < hot; i++)
        {
            mina = amax[i, i];
            for (int j = 0; j < hot; j++)
            {
                if (amax[i, j] < mina) mina = amax[i, j];
            }
            R[i] = mina;
        }

        //obtencion de S

        for (int i = 0; i < hot; i++)
        {
            for (int j = 0; j < hot; j++)
            {
                amin[i, j] = beta(G[j], min[i, j]);
            }
        }

        for (int i = 0; i < hot; i++)
        {
            maxa = amin[i, i];
            for (int j = 0; j < hot; j++)
            {
                if (amin[i, j] > maxa) maxa = amin[i, j];
            }
            S[i] = maxa;
```

```
        }
        //colas r y s
        for (int i = nx; i < hot; i++)
        {
            s[i - nx] = S[i];
            r[i - nx] = R[i];
            if (s[i - nx] == 0) sneg[i - nx] = 1;
            else sneg[i - nx] = 0;
            t[i - nx] = sneg[i - nx] & r[i - nx];

        }
        x11 = "";
        for (int i = 0; i < ny; i++)
        {
            yx[i] = 0;
            for (int j = 0; j < mx; j++)
            {
                yx[i] += ycom[j, i] * t[j];
            }
        }
        if (!Convert.ToBoolean(yx[0]) && !Convert.ToBoolean(yx[1]))
        {
            x2 = x1;
            x1 = xx;
            flag = 1;
        }
        else flag = 0;
    } //flag=1 cuando y>x y flag=0 cuando x>y
    public int mayor(string x1, string x2)
    {
        int x, y, aux = 0, carry = 0;
        char[] a, b;
        int m = 0, n;
        a = x1.ToCharArray();
        b = x2.ToCharArray();
        x = a.Length;
        y = b.Length;
        n = longitud(ref a, ref b, x1, x2);//sin signos negativos, se obtiene la longitud de las cadenas
        mayor(ref y, ref x, ref carry);
        if (carry == 1) m = 1;
        else
            for (int i = 0; i < n; i++)
            {
                x = Int32.Parse(a[i].ToString());
                y = Int32.Parse(b[i].ToString());
                if (x == y) aux++;
                mayor(ref x, ref y, ref carry);
                if (carry == 1)
                {
                    m = 0;
                    i = n;
                }
                else m = 1;
            }
        if (aux == n) m = 1;
        return m;
    }

}
```

# A. CÓDIGO DE IMPLEMENTACIÓN DE LA MEMORIA SUMA.

# Apéndice B

# Código de implementación de la memoria resta.

En el apéndice B se encuentra el código que realiza la implementación de la memoria resta, así como algunas de sus propiedades, métodos para la obtención de los vectores hot, la generación de las memorias alfa y beta, y el método para obtener la resta de dos números.

```
public int alfa(int y, int x)
        {
            if ((y == 0 && x == 0) || (y == 1 && x == 1)) return (1);
            if (y == 0 && x == 1) return (0);
            if (y == 1 && x == 0) return (2);
            return (3);
        }
        public int beta(int x, int y)
        {
            if (((y == 0) && (x == 0)) || ((y == 0) && (x == 1)) || ((y == 1) && (x == 0))) return (0);
            if (((y == 1) && (x == 1)) || ((y == 2) && (x == 1)) || ((y == 2) && (x == 0))) return (1);
            else return (0);
        }
        public void hots()
        {
            for (int i = 0; i < mx; i++)
            {
                for (int j = 0; j < hot; j++)
                {
                    if (j < nx) oh[i, j] = zh[i, j] = x[i, j];
                    else
                    {
                        if (i == (j - nx)) Zh[i, j] = 0;
                        else Zh[i, j] = 1;
                        if (i == (j - nx)) Oh[i, j] = 1;
                        else Oh[i, j] = 0;
                    }
                }
            }
        }
        public void asociacion()
        {
            for (int i = 0; i < my; i++)
                for (int j = 0; j < hot; j++)
```

```
            for (int k = 0; k < hot; k++)
            {
                asos[j, i, k] = alfa(oh[i, j], oh[i, k]);
                asosmin[j, i, k] = alfa(zh[i, j], zh[i, k]);
            }
}
public void memorias()//genera memorias max y min
{
    int maxa, mina;
    maxa = asos[0, 0, 0];
    mina = asosmin[0, 0, 0];
    for (int i = 0; i < hot; i++)
    {
        for (int j = 0; j < hot; j++)
        {
            for (int k = 0; k < mx; k++)
            {
                if (asos[i, k, j] > maxa) maxa = asos[i, k, j];
                if (asosmin[i, k, j] < mina) mina = asosmin[i, k, j];
            }
            min[i, j] = mina;
            max[i, j] = maxa;
            maxa = asos[j, 0, j];
            mina = asosmin[j, 0, j];
        }
    }

}
public int[] restar(int x1, int x2)
{
    int maxa, mina;
    for (int i = 0; i < 5; i++)
    {
        xlee[i] = bin[x1, i];
        xlee[i + 5] = bin[x2, i];
    }
    for (int i = 0; i < hot; i++)
    {
        if (i < nx) G[i] = F[i] = xlee[i];
        else
        {
            G[i] = 0;
            F[i] = 1;
        }
    }
    //obtencion de R
    for (int i = 0; i < hot; i++)
    {
        for (int j = 0; j < hot; j++)
        {
            amax[i, j] = beta(F[j], max[i, j]);
        }
    }
    for (int i = 0; i < hot; i++)
    {
        mina = amax[i, i];
        for (int j = 0; j < hot; j++)
        {
            if (amax[i, j] < mina) mina = amax[i, j];
        }
        R[i] = mina;
    }
```

```
        //obtencion de S
        for (int i = 0; i < hot; i++)
        {
            for (int j = 0; j < hot; j++)
            {
                amin[i, j] = beta(G[j], min[i, j]);
            }
        }
        for (int i = 0; i < hot; i++)
        {
            maxa = amin[i, i];
            for (int j = 0; j < hot; j++)
            {
                if (amin[i, j] > maxa) maxa = amin[i, j];
            }
            S[i] = maxa;
        }
        //colas r y s
        for (int i = nx; i < hot; i++)
        {
            s[i - nx] = S[i];
            r[i - nx] = R[i];
            if (s[i - nx] == 0) sneg[i - nx] = 1;
            else sneg[i - nx] = 0;
            t[i - nx] = sneg[i - nx] & r[i - nx];

        }
        for (int i = 0; i < ny; i++)
        {
            yx[i] = 0;
            for (int j = 0; j < mx; j++)
            {
                yx[i] += y[j, i] * t[j];
            }
        }
        return yx;
    }

}
class restas : restas_Base
{
    protected string x1, x2, s = "";
    protected char[] x11, x22, cs;
    protected int x, y, n, m;
    public int restar(string x1, string x2)
    {
        int r;
        string sig="";
        s = "";
        this.x1 = x1;
        this.x2 = x2;
        x11 = x1.ToCharArray();
        x22 = x2.ToCharArray();
        if (x11[0] == '-' && x22[0] == '-')//si ambos números son  negativos
        {
            quitar_signo(ref x11, ref x1);
            quitar_signo(ref x22, ref x2);
            r = sumar(x1, x2);
            s += r.ToString();
            s += '-';
        }
        if (x11[0] != '-' && x22[0] == '-')//si el segundo número es negativo
```

```
    {
        quitar_signo(ref x22, ref x2);
        r = sumar(x1, x2);
        s += r.ToString();
    }
    if ((x11[0] != '-' && x22[0] != '-') || (x11[0] == '-' && x22[0] != '-'))
    {
        if (x11[0] == '-' && x22[0] != '-')
        {
            sig="-";
            quitar_signo(ref x11, ref x1);
        }
        m = mayor(x1, x2);
        if (m == 0)
        {
            sig = "-";
            s = resta(x2, x1);
        }
        else
        {
            s = resta(x1, x2);
        }

    }
    s += sig;
    x11 = s.ToCharArray();
    s = "";
    for (int i = x11.Length - 1; i >= 0; i--)
    {
        s += x11[i].ToString();
    }

    return Convert.ToInt32(s);
}//método que determina como se efectuará la resta
public string resta(string x1, string x2)
{
    int j = 1, aux, carry = 0,ca=0;
    int[] xy, cr;
    this.x1 = x1;
    this.x2 = x2;
    x11 = x1.ToCharArray();
    x22 = x2.ToCharArray();
    s = "";
    n = longitud(ref x11, ref x22, x1, x2);
    for(int i=n-1; i>=0; i--)
    {
        if (carry == 1||ca==1)
        {
            if (x11[i] == '0')
            {
                x11[i] = '9';
                ca = 1;
            }
            else
            {
                cr = restar(Int32.Parse(x11[i].ToString()), 1);
                x11[i] = Convert.ToChar(cr[1].ToString());
                ca = 0;
            }
        }
            x = Int32.Parse(x11[i].ToString());
            y = Int32.Parse(x22[i].ToString());
```

```
            aux = sumar(x11[i].ToString(), x22[i].ToString());
            if (aux == 0)
            {
                carry = 0;
            }
            else mayor(ref x, ref y, ref carry);
            if (carry == 1)
            {
                x = Int32.Parse(x11[i].ToString());
                y = Int32.Parse(x22[i].ToString());
                x = sumar("10", x.ToString());
            }
        xy = restar(x, y);
        s += xy[1].ToString();
    }
        return s;
    }

}
```

# B. CÓDIGO DE IMPLEMENTACIÓN DE LA MEMORIA RESTA.

# Apéndice C

# Código de implementación de la multiplicación y división

En el apéndice C se encuentra el código de la clase multiplicación y división, estas clases son heredadas de las clases base suma y resta respectivamente ya que como se mencionó anteriormente son las bases de ambas operaciones.

```
class multiplicaciones : sumas
    {
        protected int n, m;
        protected string x1, x2;
        protected char[] x11, x22;
        public string multiplicar(string x1, string x2)
        {
            string s = "0", signo = "";
            string[] sums;
            int x, y, aux;
            this.x1 = x1;
            this.x2 = x2;
            x11 = x1.ToCharArray();
            x22 = x2.ToCharArray();
            if (x11[0] == '-' && x22[0] == '-')//si ambos números son  negativos
            {
                signo = "-";
                quitar_signo(ref x11, ref x1);
                quitar_signo(ref x22, ref x2);
            }
            if (x11[0] == '-' && x22[0] != '-')//si el primer número es negativo
            {
                signo = "-";
                quitar_signo(ref x11, ref x1);
            }
            if (x11[0] != '-' && x22[0] == '-')//si el segundo número es negativo
            {
                quitar_signo(ref x22, ref x2);
            }
            n = longitud(ref x11, ref x22, x1, x2);
            m = mayor(x1, x2);
            if (m == 0)
            {
                s = x1;
                x1 = x2;
```

```
            x2 = s;
            s = "";
        }
        x = Int32.Parse(x1);
        y = Int32.Parse(x2);
        x11 = x2.ToCharArray();
        sums = new string[x11.Length];
        for (int i = x11.Length-1,k=0; i >= 0; i--,k++)
        {
            y = Int32.Parse(x11[i].ToString());
            s = "";
            for (int j = 0; j < y; j++)
            {
                aux = sumar(s, x1);
                s = aux.ToString();
            }
            for (int l = 0; l < k; l++) s += '0';
            sums[i] = s;
        }
        s = sums[0];
        for (int i= 1; i < x11.Length; i++)
        {
            aux = sumar(s, sums[i]);
            s = aux.ToString();
        }
        s = signo + s;
        return(s);
    }
}
class divisiones : restas
{
    protected int n, m;
    protected string x1, x2;
    protected char[] x11, x22;
    public string dividir(string x1, string x2)
    {
        string s = "0", signo = "", r = "";
        int x, y, aux, i = 0, div = 1;
        this.x1 = x1;
        this.x2 = x2;
        x11 = x1.ToCharArray();
        x22 = x2.ToCharArray();
        if (x11[0] == '-' && x22[0] == '-')//si ambos números son  negativos
        {
            signo = "-";
            quitar_signo(ref x11, ref x1);
            quitar_signo(ref x22, ref x2);
        }
        if (x11[0] == '-' && x22[0] != '-')//si el primer número es negativo
        {
            signo = "-";
            quitar_signo(ref x11, ref x1);
        }
        if (x11[0] != '-' && x22[0] == '-')//si el segundo número es negativo
        {
            quitar_signo(ref x22, ref x2);
        }
        n = longitud(ref x11, ref x22, x1, x2);
        m = mayor(x1, x2);
        if (m == 0)
        {
            div = 0;
```

```
        }

            s = x1;
            for (int j = 0; j < 4; j++)
            {
                while (div != 0)
                {
                    aux = restar(s, x2);
                    s = aux.ToString();
                    div = mayor(s, x2);
                    i++;
                }
                if (s != "0") s += '0';
                else j = 5;
                if (j == 0) r += i.ToString() + '.';
                else r += i.ToString();
                i = 0;
                div = 1;
           }


        return(r);
    }
}
```

# C. CÓDIGO DE IMPLEMENTACIÓN DE LA MULTIPLICACIÓN Y DIVISIÓN

# Referencias


[1] Federico Felipe María Elena Aceveda, Marco Antonio Acevedo. Clasification of cancer recurrence with alpha - beta bam. *Mathematical Problems in Engineering*, 2009:1–14, 2009. 1

[2] Marco Antonio Acevedo María Elena Acevedo, Cornelio Yañez-Márquez. Associative momodel for storing and retrieving conceptlattices. *Mathematical Problems in Engineering*, 2010:1–27, 2010. 1

[3] Federico Felipe Elena Acevedo, Antonio Acevedo. Associative memory approach for the diagnosis of parkinson´s disease. *Pattern Recognition, Third Mexican Conference MCPR 2011*, LNCS 6718:103–117, 2011. 1

[4] Calderón Sambarinon MJ Acevedo Mosqueda ME, Acevedo Mosqueda MA. Modelos asociativos para la predicción de la localización siubcelular de proteínas. *Revista Mexicana de Ingeniería Biomédica*, XXXIII:17– 28, Junio 2012. 1

[5] M. H. Hassoun. Associative neural memories. *Oxford University Press*, 1993. 5

[6] T Kohonen. Self-organization and associative memory. *Springer-Verlag*, 1989. 5

[7] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982. 5, 11

[8] M. & Papert S. Minsky. Perceptrons, mit press. *Cambridge*, 1969. 5, 9

[9] Buneman O. & Longuet-Higgins H. Willshaw, D. Non-holographic associative memory. *Nature*, (222):960–962., 1969. 5, 10

[10] J. A. Anderson. A simple neural network generating an interactive memory. *Mathematical Biosciences*, 14:197–220., 1972. 5, 10

[11] T. Kohonen. Correlation matrix memories. *IEEE Transactions on Computers ,C-21*, 4:353–359., 1972. 6, 9, 10

[12] C. Yáñez-Márquez. Memorias asociativas basadas en relaciones de orden y operadores binarios. *Tesis doctoral, CIC-IPN*, 2002. 6, 7

[13] P. K. Simpson. Artificial neural systems. *Pergamon Press*, 1990. 6, 9

[14] H. Steinbuch, K. & Frank. Nichtdigitale lernmatrizen als perzeptoren. *Kybernetik*, 1(3):117–124., 1961. 6, 9

[15] Kakarountas A. Sklavos N. & Goutis C. E. Papadomanolakis, K. A fast johnson-mobius encoding scheme for fault secure binary counters. *Proceedings of Design, Autamtions and Test in Europe*, pages 1–7., 2002. 6, 9

[16] J. Abu-Mostafa, Y. & St. Jacques. Information capacity of the hopfield model. *IEEE Transactions on Information Theory, IT-31*, (4):461–464., 1985. 6, 12

[17] Sussner P. & Diaz-de-Leon J. L. Ritter, G. X. Morphological associative memories. *IEEE Transactions on Neural Networks*, 9:281–293., 1998. 6, 12

[18] Barrón R. & Vázquez R. Sossa, H. New associative memories to recall real-valued patterns. *CIARP, LNCS 3287*, pages 195–202., 2004. 7, 15

[19] J.L Yáñez Márquez, C. & Díaz-de-León Santiago. Memorias asociativas basadas en relaciones de orden y operaciones binarias, computación y sistemas. *Computación y Sistemas (Revista Iberoamericana de Computación incluida en el Índice de CONACyT)*, 6(4):. 300–311, 2003. 7

[20] R. Santiago Montero. Clasificador hibrido de patrones basado en la leaar-matriz de steinbuch y el linear associator de anderson-kohonen. *Tesis de Maestría, IPN Centro de Investigación en Computación*, 2003. 7

[21] Díaz-de-León Santiago J.L. & Yáñez Márquez C. Santiago Montero, R. Clasificador asociativo de patrones: avances teóricos. *Proc. del XII Congreso Internacional de Computación, CIC 2003, organizado por el Centro de Investigación en Computación del Instituto Politécnico Nacional.*, pages (RP10) 1–11., 2003. 7

[22] M.A. Acevedo Mosqueda. Memorias asociativas bidireccionales alfa-beta. *Tesis de Doctorado, IPN Centro de Investigación en Computación*, 2006. 7

[23] Yáñez-Márquez-Cornelio & López-Yáñez I. Acevedo-Mosqueda, M.E. A new model of bam: Alpha-beta bidirectional associative memories. *Springer-Verlag Berlin Heidelberg*, pages 286–295., 2006. 7

[24] Yáñez-Márquez-Cornelio & López-Yáñez I. Acevedo-Mosqueda, M.E. Alpha - beta bidirectional associative memories based translator. *IJCSNS International Journal of Computer Science and Network Security*, 6(5A):190–194., 2006. 7

[25] Sánchez-Fernández-L. P. & López-Yáñez I. Yáñez-Márquez, Cornelio. Alpha-beta associative memories for gray level patterns. *Springer-Verlag Berlin Heidelberg*, 818-823., 2006. 7

[26] López-Yáñez-I. & Yáñez-Márquez-Cornelio Román-Godínez, I. A new classifier based on associative memories. *IEEE Computer Society*, 55-59, 2006. 7

[27] Yáñez-Márquez-Cornelio & López Leyva-L.O. Aldape-Pérez, M. Feature selection using a hybrid associative classifier with masking technique. *IEEE Computer Society, Proc. Fifth Mexican International Conference on Artificial Intelligence, MICAI 2006*, pages 151–160., 2006. 8

[28] Yáñez-Márquez-Cornelio & López Leyva-L.O. Aldape-Pérez, M. Optimized implementation of a pattern classifier using feature set reduction, research in computing science. *Special issue: Control, Virtual Instrumentation and Digital Systems.*, 24:11–20, 2006. 8

[29] Yáñez-Márquez-Cornelio & López-Yáñez-I. Acevedo-Mosqueda, M.E. Alpha-beta bidirectional associative memories: Theory and applications, neural processing letters (revista isi-jcr). *Springer-Verlag Berlin Heidelberg*, 26(1):1–40, 2007. 8

[30] Yáñez-Márquez-Cornelio & López-Yáñez-I. Acevedo-Mosqueda, M.E. A new model of bam: Alpha-beta bidirectional associative memories. *Journal of Computers, Academy Publisher*, 2(4):49–56., 2007. 8

[31] Yáñez-Márquez-Cornelio & Argüelles-Cruz-A.J. Aldape-Pérez, M. Optimized associative memories for feature selection. *Lecture Notes in Computer Science (ISI Proceedings), LNCS 4477, Springer-Verlag Berlin Heidelberg,*, pages 435–442., 2007. 8

[32] M.A. Cruz Meza. Aprendizaje y recuperación de imágenes en color mediante memorias asociativas alfa-beta. *Tesis de Maestría, IPN Centro de Investigación en Computación*, 2007. 8

[33] Cruz-Meza-M.E. Sánchez-Garfias-F.A.-López-Yáñez I. f Yáñez-Márquez, Cornelio. Using alpha-beta associative memories to learn and recall rgb images. *Lecture Notes in Computer Science (ISI Proceedings), LNCS 4493, Springer-Verlag Berlin Heidelberg*, pages 828–833., 2007. 8

[34] M. Aldape Pérez. Implementación de los modelos alfa-beta con lógica reconfigurable. *Tesis de Maestría, IPN Centro de Investigación en Computación*, 2007. 8

[35] Yáñez-Márquez-Cornelio & Argüelles-Cruz-A.J. Aldape-Pérez, M. Fpga implementation of alfa-beta & associative memories, research in computing science. *Special issue: Computer Engineeering, IPN México*, 30:27–36, 2007. 8

[36] Cornelio . Román-Godínez, I. & Yáñez-Márquez. Complete recall on alpha-beta heteroassociative memory. *Springer-Verlag Berlin Heidelberg*, pages 193–202., 2007. 8

[37] I. Román Godínez. Aplicación de los modelos asociativos alfa-beta a la bioinformatica. *Tesis de Maestría, IPN Centro de Investigación en Computación*, 2007. 8

[38] I. López Yáñez. Clasificador automático de alto desempeño. *Tesis de Maestría, IPN Centro de Investigación en Computación*, 2007. 8

[39] A.J. Argüelles Cruz. Redes neuronales alfa-beta sin pesos: teoría y factibilidad de implementación. *Tesis de Doctorado, IPN Centro de Investigación en Computación*, 2007. 8

[40] Argüelles-Cruz-A.J.: Alarcón-Paredes, A. Cainn - weightless alpha-beta neural network. *Proc. Electronics, Robotics and Automotive Mechanics Conference CERMA 2008 IEEE Commputer Society.*, 2008. 8

[41] C. Yánez. Memorias asociativas basadas en relaciones de orden y operadores binarios, tesis de doctorado. *Centro de Investigación en Computación*, 2002. 14