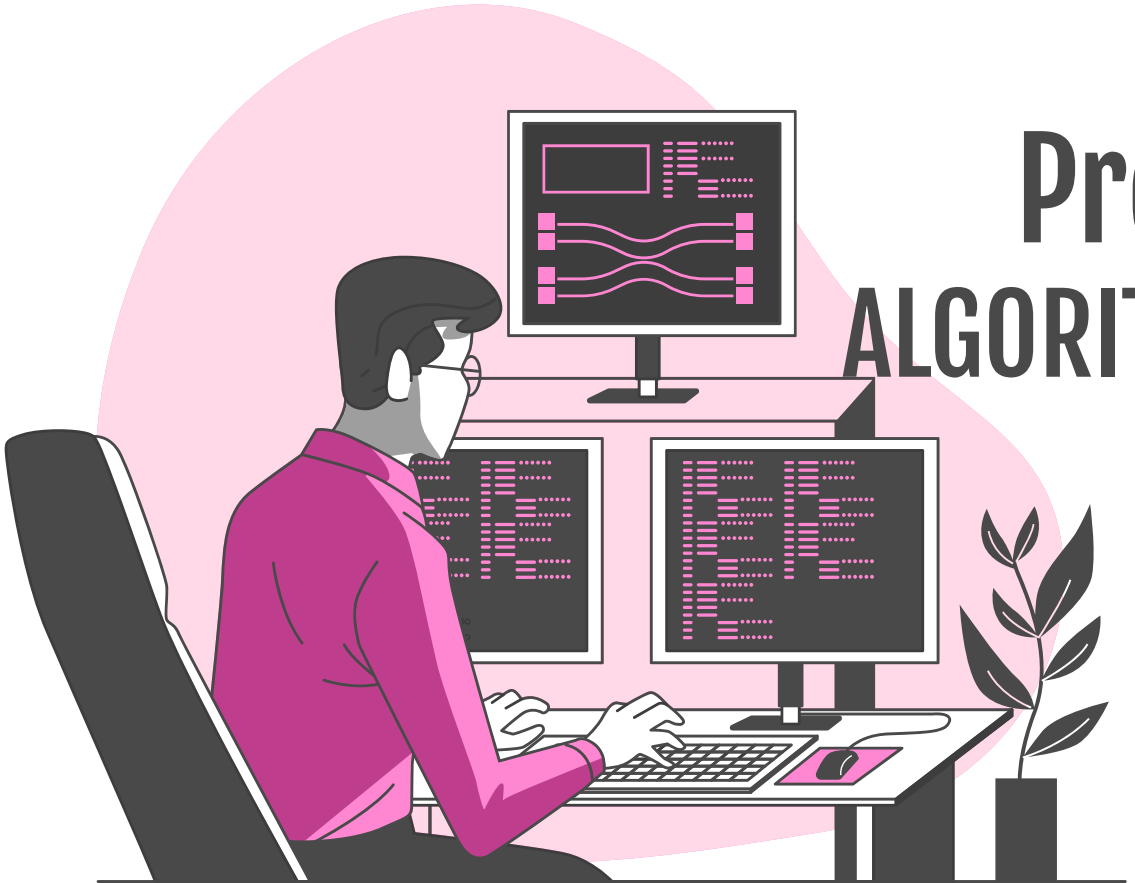


Proyecto Final
ALGORITMO BÚSQUEDA BINARIA

Arquitectura de Computadoras
ICOM

Presentado:

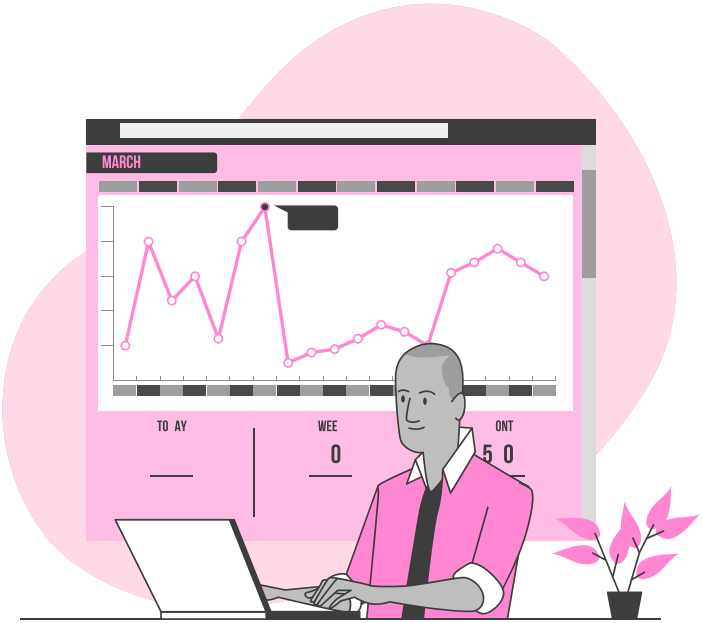
Humberto de Jesús Peña Dueñas
Karla Rebeca Hernández Elizarrarás
Elizabeth Arroyo Moreno

Mtro. Jorge Ernesto López Arce Delgado

Ingeniería en Computación

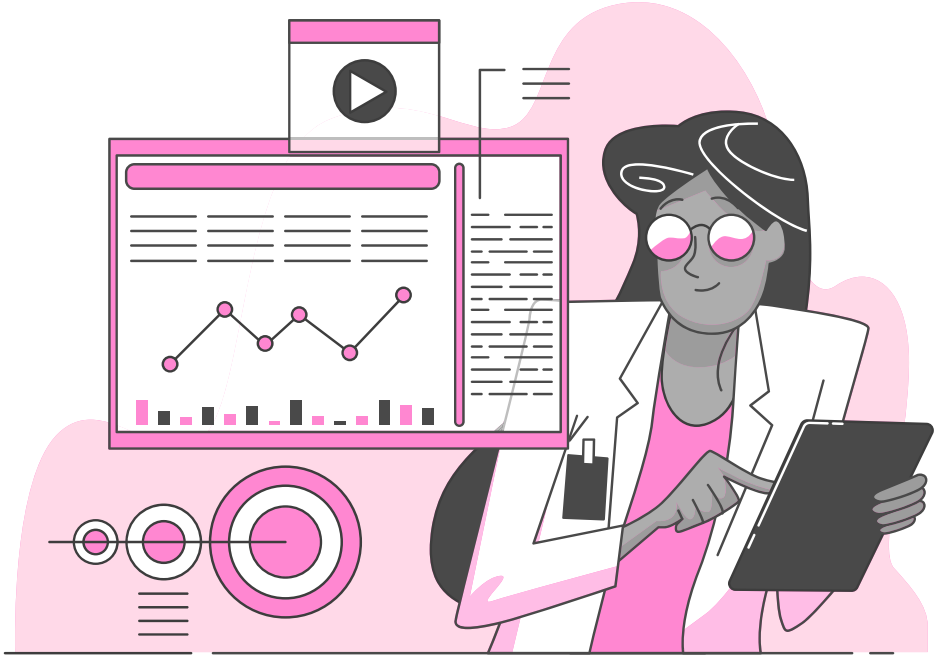
Centro Universitario de Ciencias Exactas e Ingenierías

Introducción

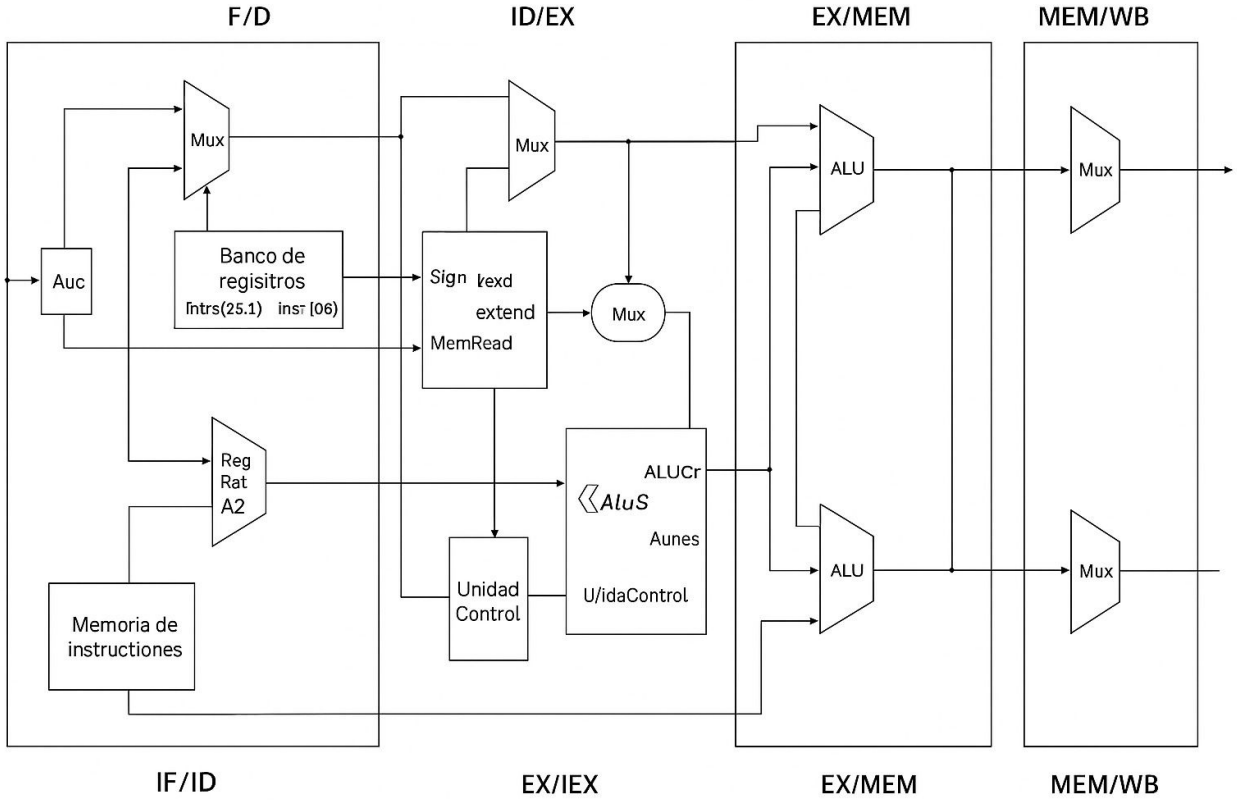


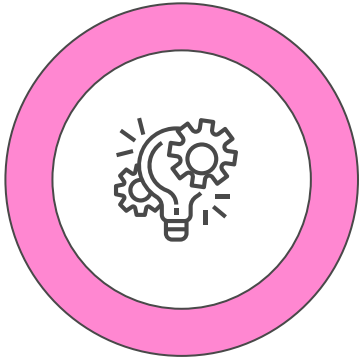
El propósito principal de este proyecto final es desarrollar e implementar un procesador de 32 bits basado en la arquitectura MIPS, capaz de interpretar y ejecutar un subconjunto específico de 28 instrucciones pertenecientes a los tres formatos clásicos definidos por esta arquitectura dejando la validación desde operaciones aritméticas hasta mecanismos de control de flujo y acceso a memoria. Para la aplicación de un algoritmo de búsqueda binaria.

Contenidos



01
Diseño


F/D
ID/EX
EX/MEM
MEM/WB
Mux
Auc
Banco de
regisitros
Sign
extend
MemRead
Mux
Mux
ALU
Mux
Reg
Rat
A2
ALUCr
AluS
Aunes
U/idaControl
Unidad
Control
Memoria de
instructiones
ALU
Mux
IF/ID
EX/IEX
EX/MEM
MEM/WB



Operaciones lógicas y aritméticas

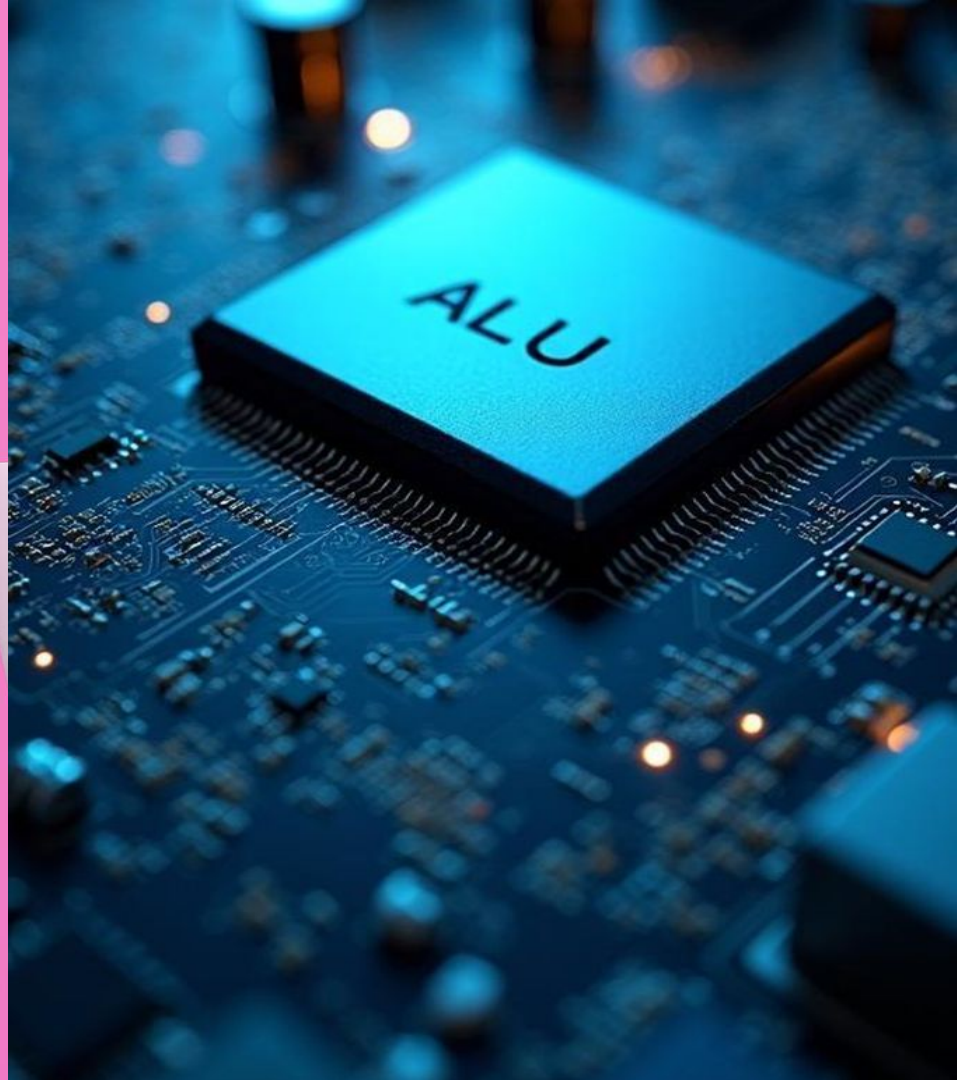
Las ALUs realizan operaciones aritméticas como sumas, restas, multiplicaciones y divisiones y las lógicas como AND, OR, NOT. Estas operaciones son básicas para el procesamiento de datos en los microprocesadores.

En la ALU se usan diferentes configuraciones para llevar a cabo estas operaciones, las cuales se ejecutan a través de instrucciones que controlan el flujo de datos entre los registros y la ALU.

02
MIPS

Arquitectura MIPS

La implementación de ALUs en la arquitectura MIPS deja hacer cálculos y operaciones lógicas mediante instrucciones. En MIPS, las operaciones aritméticas y lógicas son desglosadas en diferentes tipos de instrucciones donde la ALU recibe señales de control que determinan el tipo de operación a realizar, lo que contribuye a una ejecución eficiente.



MIPS+ALU

Análisis de sumadores, operadores lógicos y comparadores.

Introducción a instrucciones MIPS tipo R e I.

Se explica cómo la ALU procesa operaciones aritméticas y lógicas.

INSTRUCCIONES TIPO:

R

Realizan operaciones aritméticas y lógicas entre registros. Las instrucciones incluyen 'add', 'sub', 'mul' y 'and', que operan sobre los datos almacenados en los registros especificados. Estas instrucciones son fundamentales para la manipulación de datos en el procesador, permitiendo realizar cálculos complejos y operaciones de comparación.

I

Trabajan con valores inmediatos y controlan el flujo del programa, incluyen 'addi', 'load word (lw)', y 'store word (sw)', para la interacción entre la memoria y los registros. Las instrucciones de control de flujo, como 'beq' y 'bne', permiten tomar decisiones basadas en condiciones, lo que hace que el programa se ejecute de manera más dinámica.

J

Las instrucciones tipo J en la arquitectura MIPS son utilizadas para realizar saltos incondicionales a una dirección específica del programa, alterando el flujo de ejecución de manera directa. Ambas comparten una estructura de 32 bits que combina un opcode de 6 bits y una dirección de destino de 26 bits, la cual se expande a 28 bits al agregar dos ceros a la derecha y se une con los 4 bits más significativos del contador de programa (PC).

03
PYTHON

Algoritmo: Búsqueda Binaria

Programado en ensamblador MIPS con instrucciones válidas (addi, lw, beq, srl, j, etc.).

Utilizó registros $s0-$s5 y $t0-$t7.

Se convirtió a código binario usando un script en Python (deco_asm_binario.py).

Se cargó en la memoria de instrucciones para ejecución real.

```
addi $s2, $zero, 0
addi $s3, $s4, -1
loop:
slt  $t0, $s3, $s2
bne  $t0, $zero, not_found
add  $t1, $s2, $s3
srl  $t1, $t1, 1
mul  $t2, $t1, 4
add  $t3, $s0, $t2
lw   $t4, 0($t3)
beq  $t4, $s1, found
slt  $t5, $t4, $s1
bne  $t5, $zero, go_right
addi $s3, $t1, -1
j loop
go_right:
addi $s2, $t1, 1
j loop
found:
add  $s5, $zero, $t1
j end
not_found:
addi $s5, $zero, -1
end:
nop
```

Lógica ASM

Inicializa los índices:

```
addi $s2, $zero, 0      # Índice izquierdo = 0
addi $s3, $s4, -1       # Índice derecho = tamaño - 1
```

Inicio del bucle (loop):

- Termina si `$s2 > $s3`, es decir, si ya se buscó todo el arreglo sin éxito.

```
loop:
slt  $t0, $s3, $s2
bne  $t0, $zero, not_found
```

Logica ASM

Calcula el punto medio:

```
add  $t1, $s2, $s3      # Suma izquierda + derecha
srl  $t1, $t1, 1        # Divide entre 2 (desplazamiento lógico)
```

Carga el valor del medio:

```
mul  $t2, $t1, 4        # t1 * 4 (palabras de 4 bytes)
add  $t3, $s0, $t2      # Dirección del elemento medio
lw   $t4, 0($t3)        # Carga el valor
```

Compara el valor medio con el buscado:

- Si son iguales → salta a `found`
- Si el valor buscado es mayor → mueve el índice izquierdo (`$s2`)
- Si es menor → mueve el índice derecho (`$s3`)

Logica ASM

Cuando se encuentra el valor:

```
found:
add  $s5, $zero, $t1   # Guarda el índice medio (t1) en s5
j end
```

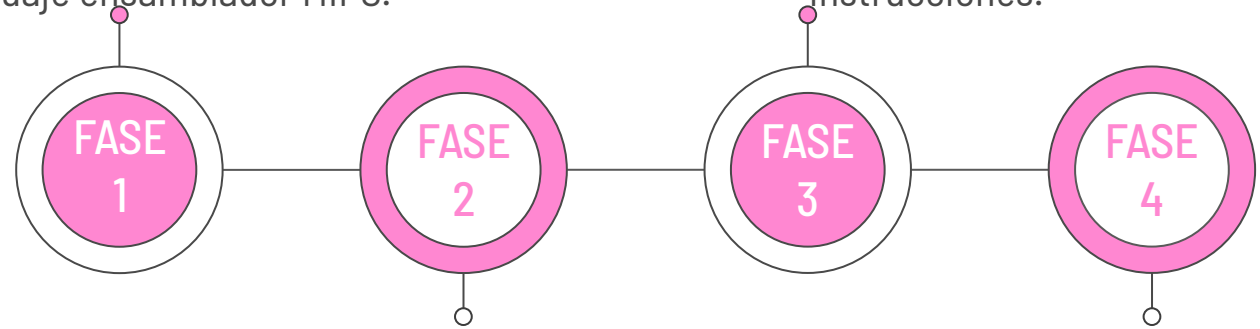
Cuando no se encuentra el valor:

```
not_found:
addi $s5, $zero, -1     # Si no se encuentra, s5 toma el valor -1
```

Conversión ASM a Binario

Se desarrolló un script que recibe como entrada un archivo .asm con el algoritmo de búsqueda binaria escrito en lenguaje ensamblador MIPS.

El script genera un archivo de texto (binario_prueba.txt) con las instrucciones en binario, listas para ser cargadas en la memoria de instrucciones.



Cada instrucción es convertida a su correspondiente representación en código máquina (binario de 32 bits).

El archivo binario se carga en el módulo de memoria del procesador durante la simulación en ModelSim.

Conclusión

Este proyecto representó una experiencia integral en el diseño e implementación de un procesador MIPS de 32 bits, permitiéndonos aplicar de manera práctica los conocimientos adquiridos en arquitectura de computadoras, lógica digital y programación en Verilog. A lo largo de las fases, desarrollamos un datapath funcional con soporte para instrucciones tipo R, I y J, implementamos buffers para simular un pipeline real y ejecutamos un algoritmo completo, como la búsqueda binaria, utilizando un entorno de simulación profesional en ModelSim.

Además, el desarrollo de un script en Python para convertir instrucciones en ensamblador a código binario permitió automatizar la carga de programas, conectando el diseño de hardware con el software de forma eficiente.

GRACIAS POR SU ATENCIÓN

FASE 1

OBJETIVO

Comprender el funcionamiento de diferentes tipos de sumadores implementando dos sumadores en Verilog: Uno de 4 bits y otro de 8 bits con salida de 9 bits.

Evaluando sus entradas y salidas mediante testbench.

DESARROLLO

El comienzo de la implementación en Verilog de sumadores de 4 y 8 bits. Pruebas con testbench: 5 casos de suma con un enfoque en claridad y reusabilidad del diseño.

FASE 2

OBJETIVO

Extender el datapath para ejecutar instrucciones tipo integrando señales de control y extensión de signo, incluyendo soporte para instrucciones de acceso a memoria (lw, sw).
Verificando su funcionamiento con los simulaciones.

DESARROLLO

Contiene:

- PC (Contador de Programa): Dirección de siguiente instrucción.
- Memoria de Instrucciones: Código máquina.
- Banco de Registros: Lectura/escritura simultánea.
- ALU + ALU Control: Ejecución de operaciones.
- Memoria de Datos: Apoyo para lw, sw.
- Unidad de Control: Señales según opcode.
- Buffers IF/ID, ID/EX, EX/MEM, MEM/WB: Pipeline.

FASE 3

OBJETIVO

Integrar soporte completo para instrucciones tipo J (jump).
Implementar buffers entre etapas para simular un pipeline real.
Ejecutar un algoritmo funcional en ensamblador MIPS: búsqueda binaria.
Validar el comportamiento del procesador mediante simulaciones en ModelSim.

DESARROLLO

Desarrollo por módulos
Unidad de Control
- Se agregó la señal Jump.
- Se detecta el opcode 000010 (instrucción j).
- Calcula la dirección de salto y la envía al PC si está activa la señal.

Program Counter (PC)
- Recibe direcciones secuenciales (PC + 4) o direcciones de salto.
- Se modificó para aceptar la nueva entrada de control
- Unidad de Control.

Banco de Registros
- Permite lecturas y escrituras simultáneas.
- Registra resultados de la ALU y datos de memoria.
- ALU y ALU Control
- Ejecuta operaciones aritméticas, lógicas y de comparación.
- El módulo ALU_Control interpreta la operación a partir del funct y las señales de control.

Memorias
Memoria de instrucciones: precargada con binario traducido desde ASM.
Memoria de datos: contiene el arreglo ordenado para la búsqueda binaria.