

Arquitectura MIPS

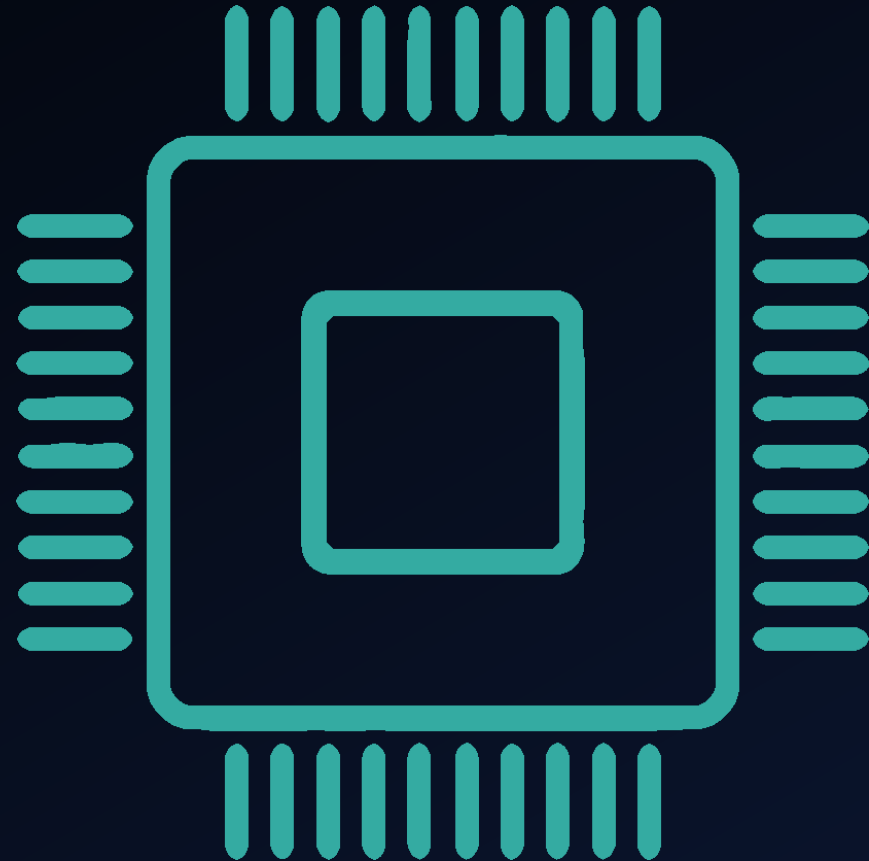
EQUIPO CONEJOS MIPS

- Arturo Daniel Agredano Gutiérrez
- Carlos Eduardo Román Bravo
- Cristian Márquez Canizales
- Alexia Gómez Rubio



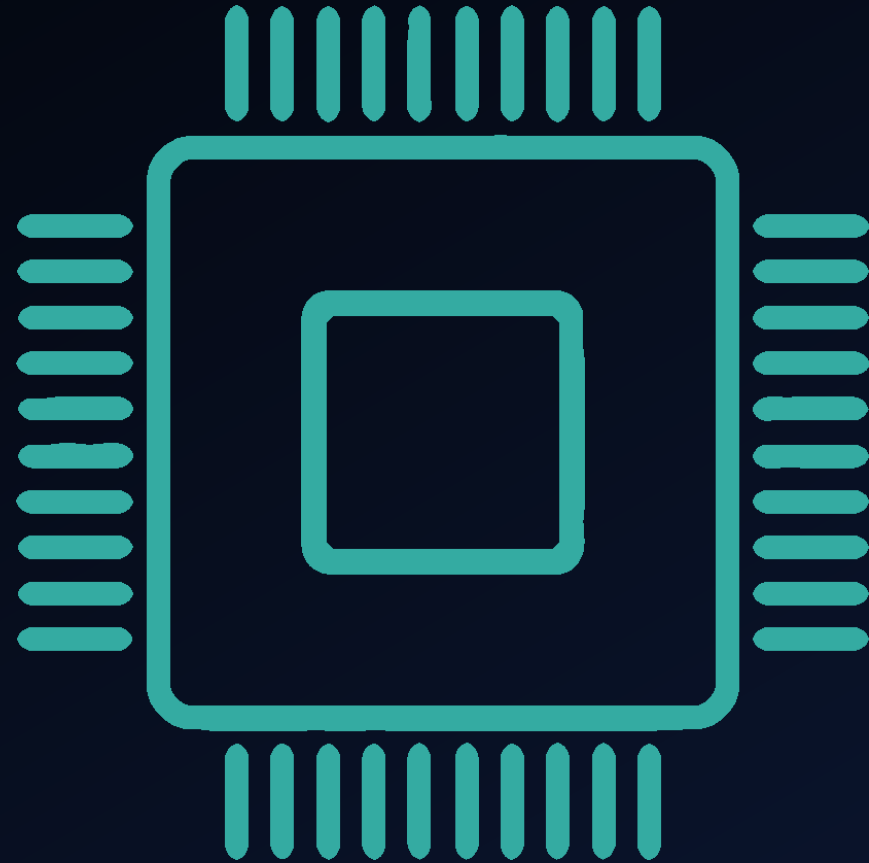
¿DE QUÉ TRATA EL PROYECTO?

Este proyecto se centra principalmente en la construcción de un procesador.



¿DE QUÉ TRATA EL PROYECTO?

Iniciamos con la construcción del “Single Datapath” que será el encargado de implementar/decodificar instrucciones básicas tipo R.





LOS OBJETIVOS DE ESTA SEMANA

Crear un DataPath capaz de ejecutar
las instrucciones ADD, SUB, AND,
OR, y SLT.

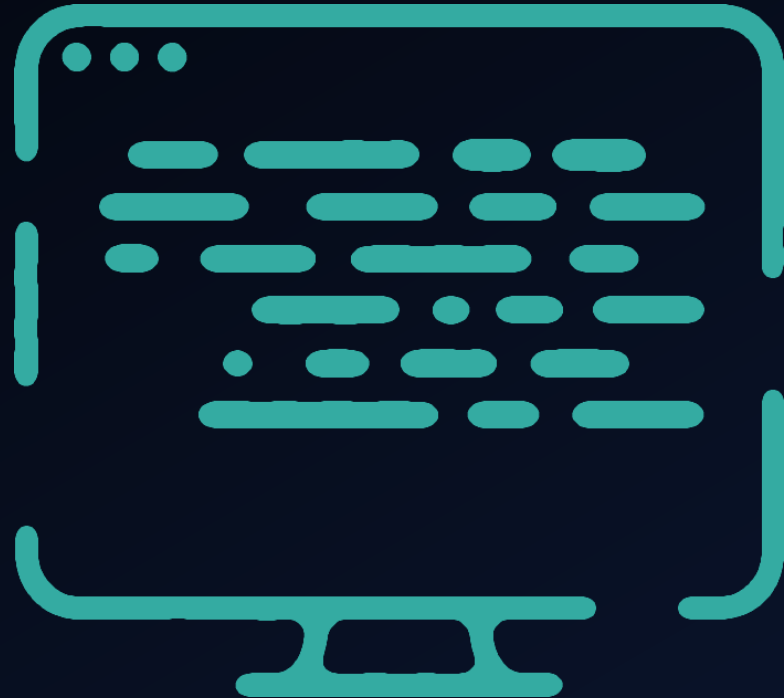


LOS OBJETIVOS DE ESTA SEMANA

Esto integrando los módulos del
BAM (ALU, Banco de Registros,
Memoria de datos) y añadiendo una
Unidad de control y una ALU control.

LOS OBJETIVOS DE ESTA SEMANA

Implementar un programa en python para crear un archivo .txt con instrucciones en lenguaje máquina.

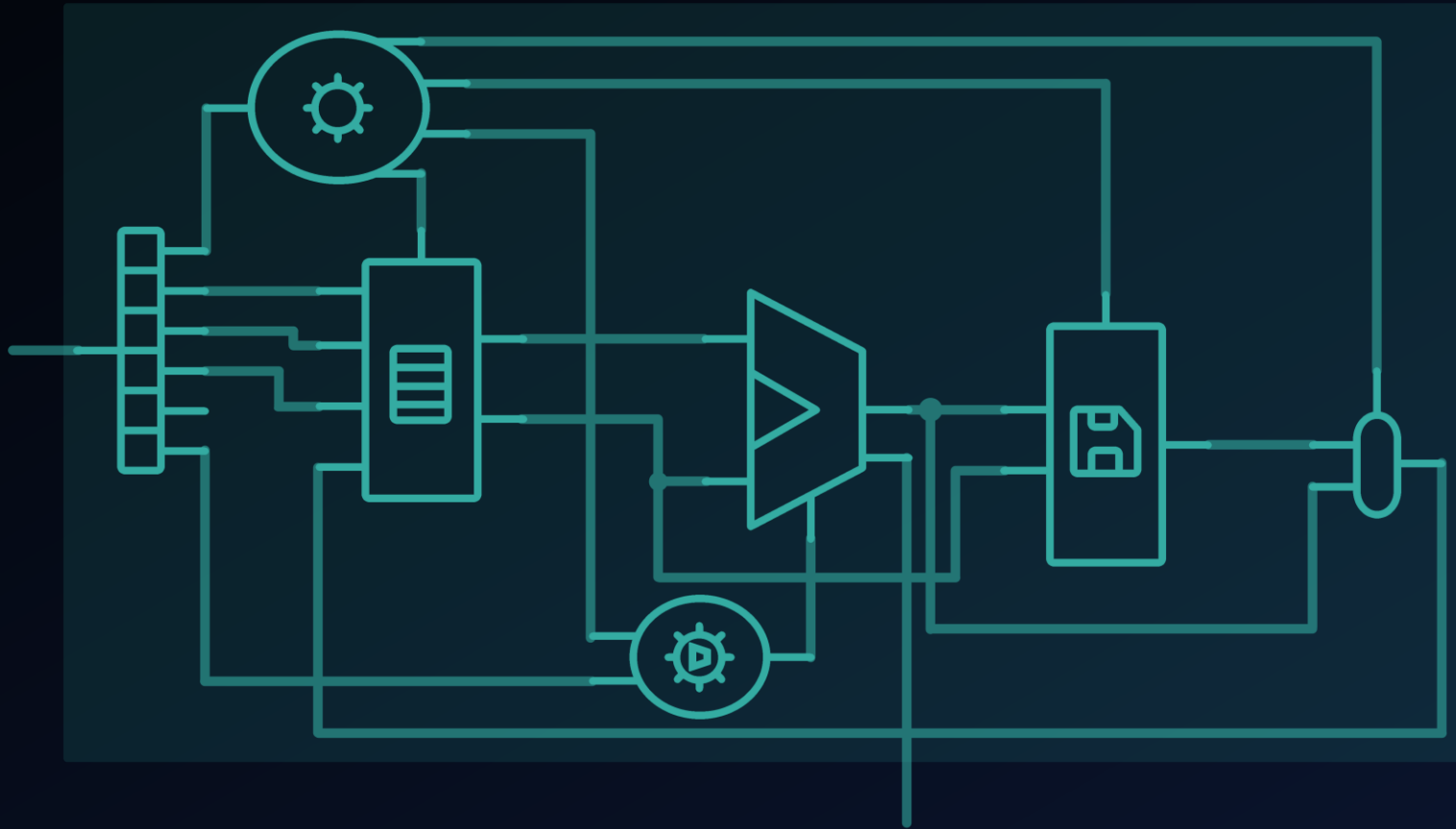


LOS OBJETIVOS DE ESTA SEMANA

Este debe visualizar el contenido del archivo, realizar una conversión de instrucciones tipo R en ensamblador a máquina y crear un nuevo archivo.txt con las instrucciones en máquina



Diseño del módulo de Verilog



Banco de Registros



Banco		
Nombre	Tipo	Longitud
ra1	Input	05 bits
ra2	Input	05 bits
di	Input	32 bits
dir	Input	05 bits
reg_write	Input	01 bits
dr1	Output Reg	32 bits
dr2	Output Reg	32 bits

ALU



ALU		
Nombre	Tipo	Longitud
op1	Input	32 bits
op2	Input	32 bits
sel	Input	04 bits
res	Output Reg	32 bits
zf	Output Reg	01 bits

Memoria



Memoria		
Nombre	Tipo	Longitud
dato	Input	32 bits
direccion	Input	32 bits
sel	Input	01 bits
salida	Output Reg	32 bits

Unidad de control



Unidad de Control		
Nombre	Tipo	Longitud
op	Input	06 bits
MemToReg	Output Reg	01 bits
MemToWrite	Output Reg	01 bits
AluOp	Output Reg	03 bits
RegWrite	Output Reg	01 bits

ALU control



ALU Control		
Nombre	Tipo	Longitud
funct	Input	06 bits
sel	Input	03 bits
AluOp	Output Reg	04 bits

ALU control



Las funciones de la Unidad de Control de ALU son:

Instrucción	Function Code	ALU Code
ADD	100000	0010
SUB	100010	0110
AND	100100	0000
OR	100101	0001
NOR	100111	1100
SLT	101010	0111

Multiplexor



Multiplexor		
Nombre	Tipo	Longitud
a	Input	32 bits
b	Input	32 bits
sel	Input	01 bits
salida	Output Reg	32 bits

La entrada de datos

- La manera en la que se envían los bits “seleccionados” es la mostrada en la tabla siguiente:

Set de bits	A dónde se envían	Función
31:26	Unidad de Control	Identificar el tipo de instrucción (R, J o I) y habilitar escritura en BR y MD
25:21	Banco de Registros	Es la dirección del primer Registro del BR
20:16	Banco de Registros	Es la dirección del segundo Registro del BR
15:11	Banco de Registros	Es la dirección donde se guardan datos en el Registro del BR
10:6	Banco de Registros	Shamt (no hace nada aún)
5:0	ALU control	Indica a la ALU qué operación realizar

Comprobando el funcionamiento

- Para comprobar que el código en verilog funciona realizamos la siguiente tabla con instrucciones en ensamblador y en máquina para mandarlo a la entrada del módulo.

Instrucción en ensamblador	instrucción en binario	Dato en \$RS	Dato en \$RT	Dato esperado en \$RD
add \$0 \$1 \$2	000000 00001 00010 00000 00000 100000	15	15	30
sub \$3 \$4 \$5	000000 00100 00101 00011 00000 100010	24	56	-32
slt \$6 \$7 \$8	000000 00111 01000 00110 00000 101010	89	2	0
and \$9 \$10 \$11	000000 01010 01011 01001 00000 100100	1001010 = 74	1010110 = 86	1000010 = 66
or \$12 \$13 \$14	000000 01101 01110 01100 00000 100101	0101011 = 43	1001010 = 74	1101011 = 107

Comprobando el funcionamiento

- El banco de registros fue precargado con los siguientes datos:

Registro	Dato en BIN	Dato en DEC	Dato en HEX
1	00000000 00000000 00000000 00001111	15	F
2	00000000 00000000 00000000 00001111	15	F
4	00000000 00000000 00000000 00011000	24	18
5	00000000 00000000 00000000 00111000	56	38
7	00000000 00000000 00000000 01011001	89	59
8	00000000 00000000 00000000 00000010	2	2
10	00000000 00000000 00000000 01001010	74	4A
11	00000000 00000000 00000000 01010110	86	56
13	00000000 00000000 00000000 00101011	43	2B
14	00000000 00000000 00000000 01001010	74	4A

Comprobando el funcionamiento

- Datos esperados en los registros de destino

Registro	Dato en BIN	Dato en DEC	Dato en HEX
0	00000000 00000000 00000000 00011110	30	1E
3	11111111 11111111 11111111 11100000	-32	FFFFFFE0
6	00000000 00000000 00000000 00000000	0	0
9	00000000 00000000 00000000 01000010	66	42
12	00000000 00000000 00000000 01101011	107	6B


Comprobando el funcionamiento

- Se llevó a cabo la simulación del módulo “Todo_TB” en el cual se esperaba que el banco de registros fuera modificado de acuerdo a las instrucciones plasmadas en las tablas previamente mostradas:

Dato en DEC	Dato en HEX
30	1E
-32	FFFFFFE0
0	0
66	42
107	6B

Memory Data - /Todo_TB/t1/ba/banco - Default			
0	30	15	15
3	-32	24	56
6	0	89	2
9	66	74	86
12	107	43	74
15	0	0	0
18	0	0	0
21	0	0	0
24	0	0	0
27	0	0	0
30	0	0	
33			

Interfaz de usuario para la generación de código máquina

A stylized, abstract graphic in the background. It features a dark teal background with a large, light teal leaf-like shape in the center. A dark teal branch or stem extends from the bottom left towards the center, passing behind the leaf. There are also some circular shapes in the background, one of which is a solid dark teal circle near the top left.

- Para realizar la GUI que permita editar un archivo asm se usó la librería Tkinter de Python
- Importamos la librería como tk

Interfaz de usuario para la generación de código máquina



- `tk.TK()`
- Crea un objeto que corresponde a una ventana donde podremos organizar elementos interactivables.

Interfaz de usuario para la generación de código máquina



- `tk.Label()`
- Una etiqueta para mostrar texto.
- Recibe:
 - La ventana en la que se va a mostrar.
 - El texto que va a mostrar.

Interfaz de usuario para la generación de código máquina



- `tk.Text()`
- Una cuadro de texto editable.
- Recibe:
 - La ventana en la que se va a mostrar.
 - El método de manejo de desborde.
 - Dimensiones.

Interfaz de usuario para la generación de código máquina



- `tk.Button()`
- Un botón que ejecuta una función.
- Recibe:
 - La ventana en la que se va a mostrar.
 - El texto que mostrará.
 - La función a ejecutar.

Interfaz de usuario para la generación de código máquina



- `tk.Pack()`
- Coloca el elemento en la ventana.
- Recibe:
 - Entre varios elementos el padding que tendrá el elemento.

Interfaz de usuario para la generación de código máquina



- `tk.delete()`
- Borra el texto de un elemento.
- `tk.insert()`
- Introduce una cadena de texto a un elemento.

Interfaz de usuario para la generación de código máquina



- Gracias a estos elementos nuestra ventana queda ordenada de la siguiente manera:

Interfaz de usuario para la generación de código máquina



- Donde los elementos son:
 - Un Label “ASM” .
 - Un cuadro de texto.
 - Contenido el archivo asm.
 - Un label “TXT”.
 - Contenido del archivo txt.
 - Botón para abrir archivo.
 - Botón para guardar cambios.
 - Botón para crear archivo binario.

Interfaz de usuario para la generación de código máquina



- `open()`
- Abre el archivo indicado.
- Recibe:
 - La dirección del archivo.
 - Un carácter que indica si se va a leer o escribir.

Interfaz de usuario para la generación de código máquina



- `filedialog.askopenfilename()`
- Abre la ventana para buscar archivos.
- Recibe:
 - El nombre del tipo de archivo.
 - La extensión que se busca.

Interfaz de usuario para la generación de código máquina



- `<file>.write()`
- Sobreescribe en el archivo indicado.
- Recibe:
 - La cadena de texto que se desea escribir.

Para convertir las instrucciones

add	\$0	\$1	\$2
sub	\$3	\$4	\$5
slt	\$6	\$7	\$8
and	\$9	\$10	\$11
or	\$12	\$13	\$14

- `<string>.split()`
 - Separa una cadena en el carácter indicado.
- `<string>.splitlines()`
 - Separa una cadena por líneas.
- Al final tenemos un arreglo bidimensional de palabras separadas.

Para convertir las instrucciones

add	\$0	\$1	\$2
sub	\$3	\$4	\$5
slt	\$6	\$7	\$8
and	\$9	\$10	\$11
or	\$12	\$13	\$14

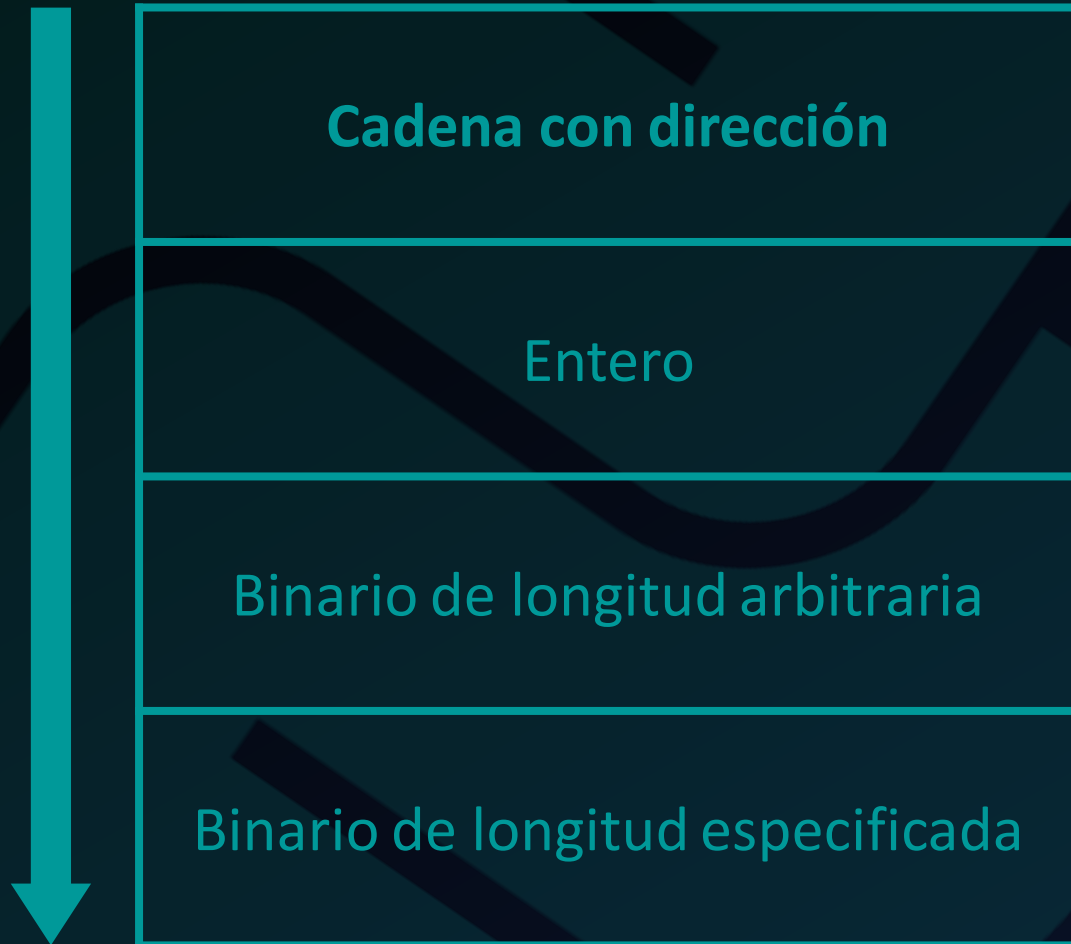
- Por medio de un diccionario identificamos el tipo de instrucción (R).
- Esto gracias a la primera columna.

Para convertir las instrucciones

```
instructions ={  
    "add" : ["R","100000"],  
    "sub" : ["R","100010"],  
    "and" : ["R","100100"],  
    "or"  : ["R","100101"],  
    "nor" : ["R","100111"],  
    "slt" : ["R","101010"]  
}
```

- En nuestro diccionario encontramos el tipo de instrucción que encontramos y su código de operación (opcode o funct).

Para convertir las instrucciones



- Para las direcciones llamamos a la una función que llamamos `IntStr_Bin()`
- Convierte la cadena que contiene la dirección a una cadena en binario de longitud especificada.

Para convertir las instrucciones

"000000"
IntStr_Bin(Elemento 2)
IntStr_Bin(Elemento 3)
IntStr_Bin(Elemento 1)
"000000"
Resultado del diccionario
Nueva línea

- Apilamos los elementos de la instrucción en una cadena.
- Al final la agregamos a la cadena general y se manda tanto al archivo como al cuadro de texto.