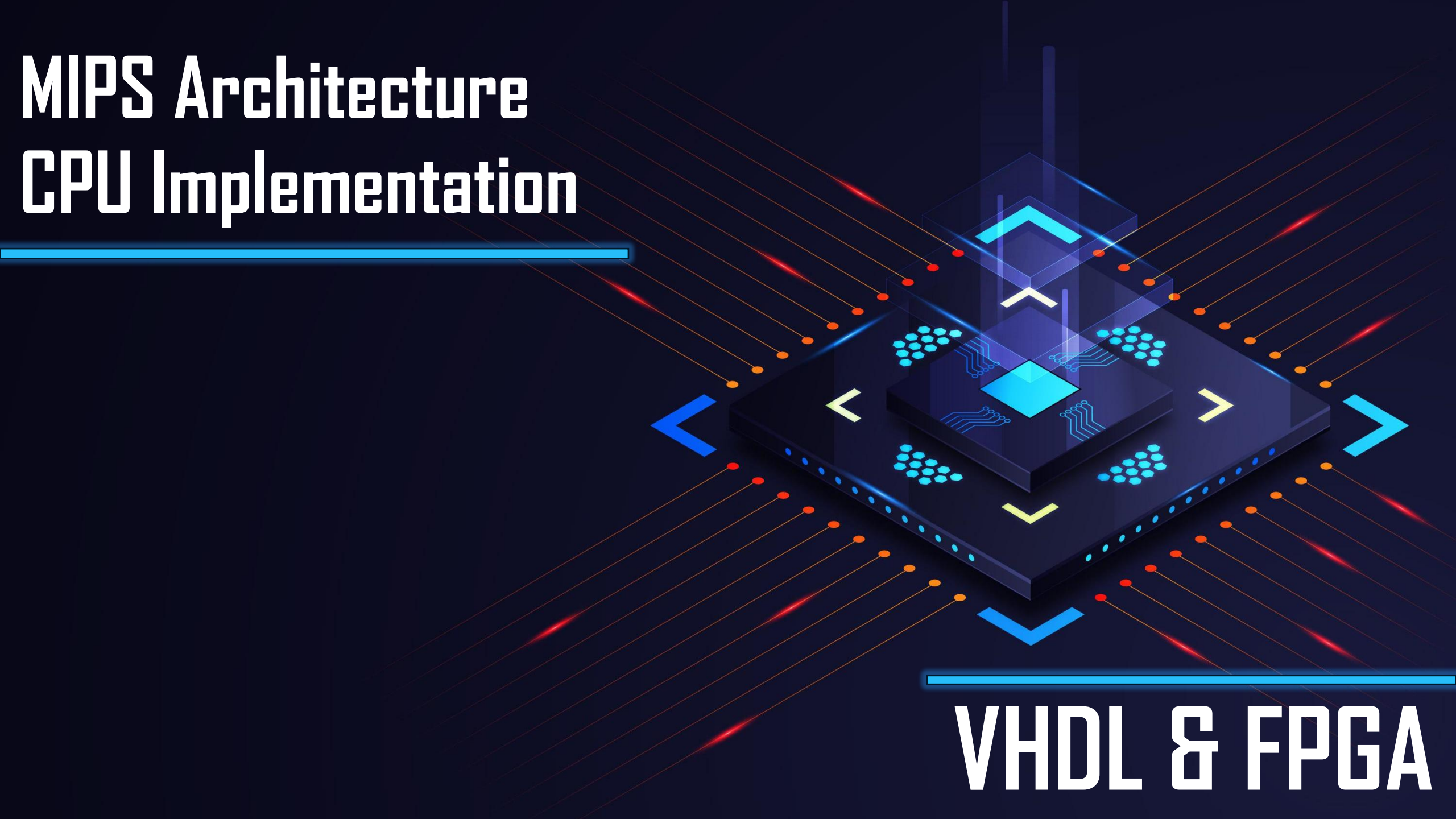
# MIPS Architecture CPU Implementation



## VHDL & FPGA

# ABOUT

## Team

- Youssef Mohamed
- Moslem Sayed
- Mina Magdy
- Mohamed Salem

# This isn't just a CPU

- it's a versatile powerhouse ready to fetch, decode, and execute any program thrown at it.

- we'll unravel the intricacies of MIPS.

- we’ll understand the role VHDL played in bringing this CPU to life, and appreciate its capability to handle a diverse range of programs. Without further ado, let's dive into the world of MIPS CPU design!

INTRO

# MIPS ARCHITECTURE COMPONENTS

- Program Counter (PC)
- Registers
- Instruction Memory (ROM)
- Data Memory (RAM)
- Arithmetic Logic Unit (ALU)
- Control Unit
- Flags (Zero, Carry, Overflow)
- Decoder
- Multiplexer (MUX)
- adder

MIPS CPU

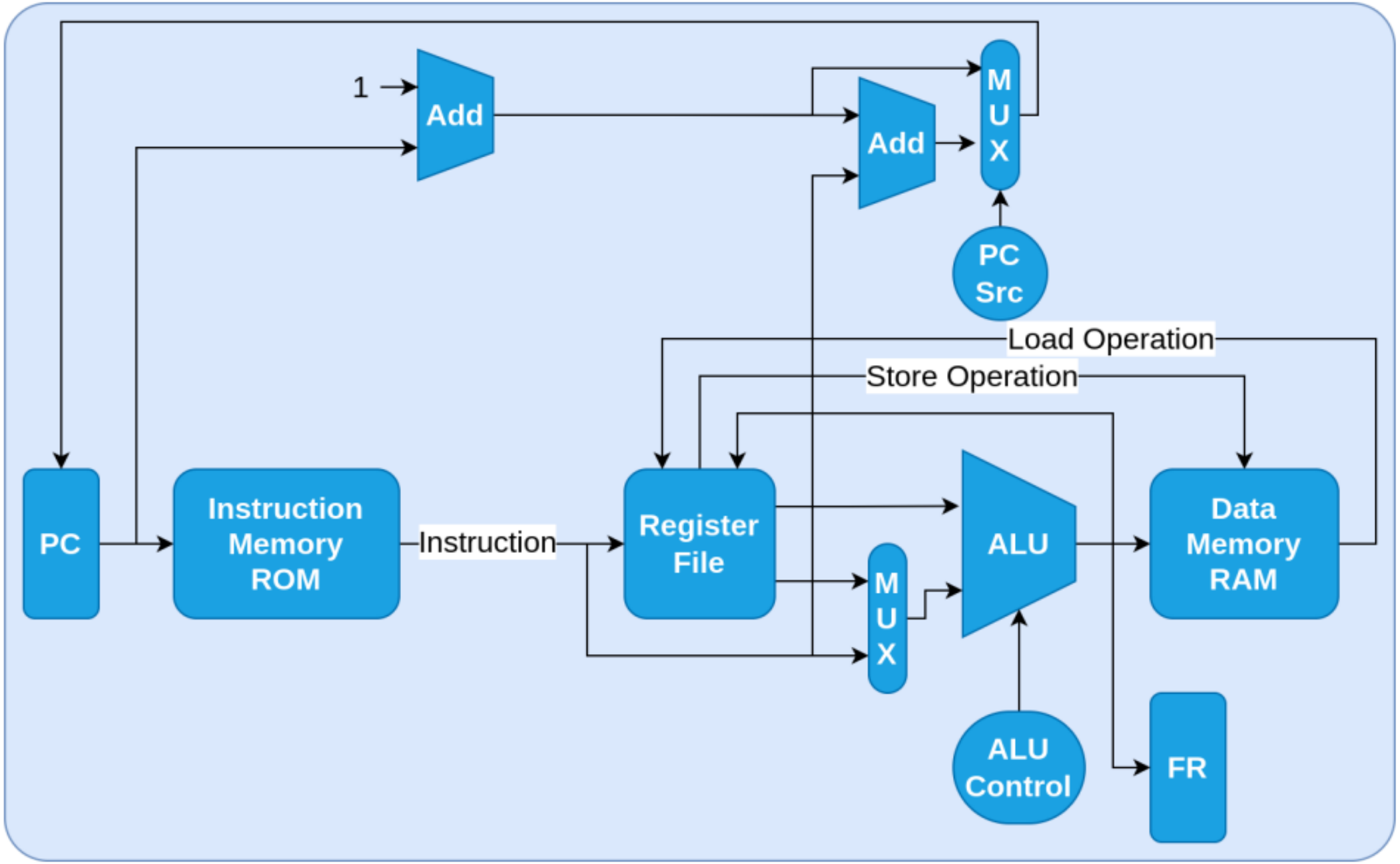



**Figure-2: MIPS Processor Data Path**

DESIGN

# VHDL IMPLEMENTATION

- The VHDL implementation of the MIPS architecture revolves around creating a digital representation of a processor that adheres to the principles of Reduced Instruction Set Computing (RISC).

- The overarching idea is to model the fundamental functionalities of a MIPS CPU, including its instruction execution cycle, data storage, and control flow.

VHDL

# VHDL IMPLEMENTATION

- **Sequential Execution:**

The VHDL code employs a sequential execution model, mirroring the step-by-step nature of a real processor. This is encapsulated within a process block that triggers on rising clock edges or in response to a reset condition.

# VHDL IMPLEMENTATION

- **Instruction Fetch-Decode-Execute Cycle:**

The core of the implementation revolves around the fetch-decode-execute cycle, how a processor fetches instructions from memory, decodes them to understand the operation, and executes the corresponding action.

# VHDL IMPLEMENTATION

**Memory Representation:**

The code initializes arrays to represent critical memory components, such as ROM (Read-Only Memory) for instructions, general-purpose registers for data storage, and RAM
(Random Access Memory) for additional data storage.

VHDL

# VHDL IMPLEMENTATION

**Data and Control Flow:**

The VHDL model manages data and control flow through the CPU. It includes mechanisms for arithmetic and logical operations, branching based on conditions, and the storage and retrieval of data from memory.

**Flag Handling (3 - bits):**

•The code incorporates flag handling, where certain conditions, such as zero results, carry, or overflow, are tracked and reflected in the

| Zero flag | Overflow | Carry |
|---|---|---|

VHDL

# INSTRUCTION SET

**Table-4: MIPS CPU instruction set**

| Mne-monic | Assembly Format | Fields | | | | | Description | | Type |
|---|---|---|---|---|---|---|---|---|---|
| - | - | opcode 4 | rs 3 | rt 3 | rd 3 | function 3 | Arithmetic | | R |
| add | ADD $rs, $rt, $rd | 0001 | xxx | xxx | xxx | 000 | $rd ← $rs + $rt | addition | RRR |
| sub | SUB $rs, $rt, $rd | 0001 | xxx | xxx | xxx | 001 | $rd ← $rs - $rt | subtraction | |
| mul | MUL $rs, $rt, $rd | 0001 | xxx | xxx | xxx | 010 | $rd ← $rs * $rt | Multiplication | |
| div | DIV $rs, $rt, $rd | 0001 | xxx | xxx | xxx | 011 | $rd ← $rs / $rt | Division | |
| and | AND $rs, $rt, $rd | 0001 | xxx | xxx | xxx | 100 | $d ← $s AND $t | and | |
| or | OR $rs, $rt, $rd | 0001 | xxx | xxx | xxx | 101 | $d ← $s OR $t | or | |
| xor | XOR $rs, $rt, $rd | 0001 | xxx | xxx | xxx | 110 | $d ← $s XOR $t | xor | |
| cpl | CPL $rs, $rt, $rd | 0001 | xxx | xxx | xxx | 111 | $d ← NOT $s | complement | |

INSTRUCTION

# INSTRUCTION SET

| - | - | opcode<br>4 | rs<br>3 | rt<br>3 | address /<br>immediate<br>6 | Immediate | | I |
|---|---|---|---|---|---|---|---|---|
| addi | ADD $rs, $rt, imm | 0010 | xxx | xxx | imm | $rt ← $rs + imm | addition | RRI |
| subi | SUB $rs, $rt, imm | 0011 | xxx | xxx | imm | $rt ← $rs - imm | subtraction | |
| andi | AND $rs, $rt, imm | 0100 | xxx | xxx | imm | $rt ← $rs AND imm | and | |
| ori | OR $rs, $rt, imm | 0101 | xxx | xxx | imm | $rt ← $rs OR imm | or | |
| xori | XOR $rs, $rt, imm | 0110 | xxx | xxx | imm | $rt ← $rs XOR imm | xor | |
| - | - | opcode<br>4 | rs<br>3 | address / immediate<br>9 | | Immediate | | I |
| li | LI $rs, imm | 0111 | xxx | imm | | $rs ← imm | Load immediate | RI |
| lm | LM $rs, addr | 1000 | xxx | addr | | $rs ← M[addr] | Load from memory | |
| sm | SM $s, addr | 1001 | xxx | addr | | M[addr] ← $rs | Store to memory | |

INSTRUCTION

# INSTRUCTION SET

| - | - | opcode<br>4 | address<br>12 | | | Jump Type | | J |
|---|---|---|---|---|---|---|---|---|
| beq | BEQ $rs, $rt | 1010 | xxx | xxx | addr | if($rs == $rt)<br>PC ← Addr<br>else PC ← PC + 1 | Branch if equal | |
| bgt | BGT $rs, $rt | 1011 | xxx | xxx | addr | if($rs > $rt)<br>PC ← Addr<br>else PC ← PC + 1 | Branch if greater than | |
| blt | BLT $rs, $rt | 1100 | xxx | xxx | addr | if($rs < $rt)<br>PC ← Addr<br>else PC ← PC + 1 | Branch if less than | |
| bc | BC $rs, $rt | 1101 | xxx | xxx | addr | if(CF == 1)<br>PC ← Addr<br>else PC ← PC + 1 | Branch on carry | |
| bz | BZ $rs, $rt | 1110 | xxx | xxx | addr | if(ZF == 1)<br>PC ← Addr<br>else PC ← PC + 1 | Branch on zero | |
| br | BR addr_imm | 1111 | addr | | | PC ← Addr | Unconditional Branch | |
| nop | NOP | 0000 | ---------- | | | No operation | No operation | other |

INSTRUCTION

# Required Program

- Calculate the number of ones in the binary representation of a decimal number

# Instruction set

```
"0111110000000000", -- load zero to final result register
 "0111000000011101", -- load number
 "0111001000000010", -- load 2 for division
 "0100000010000001", -- and with one
 "1110000000000110", -- check the result of and if zf =1 jump for address 6
 "0010110110000001", -- increment the final result register
 "0001000001000011", -- divide 2
 "1110000000000000", -- if number is equal to zero start again
 "1111000000000011", -- if number not equal to zero then loop again
```

[Source Code]

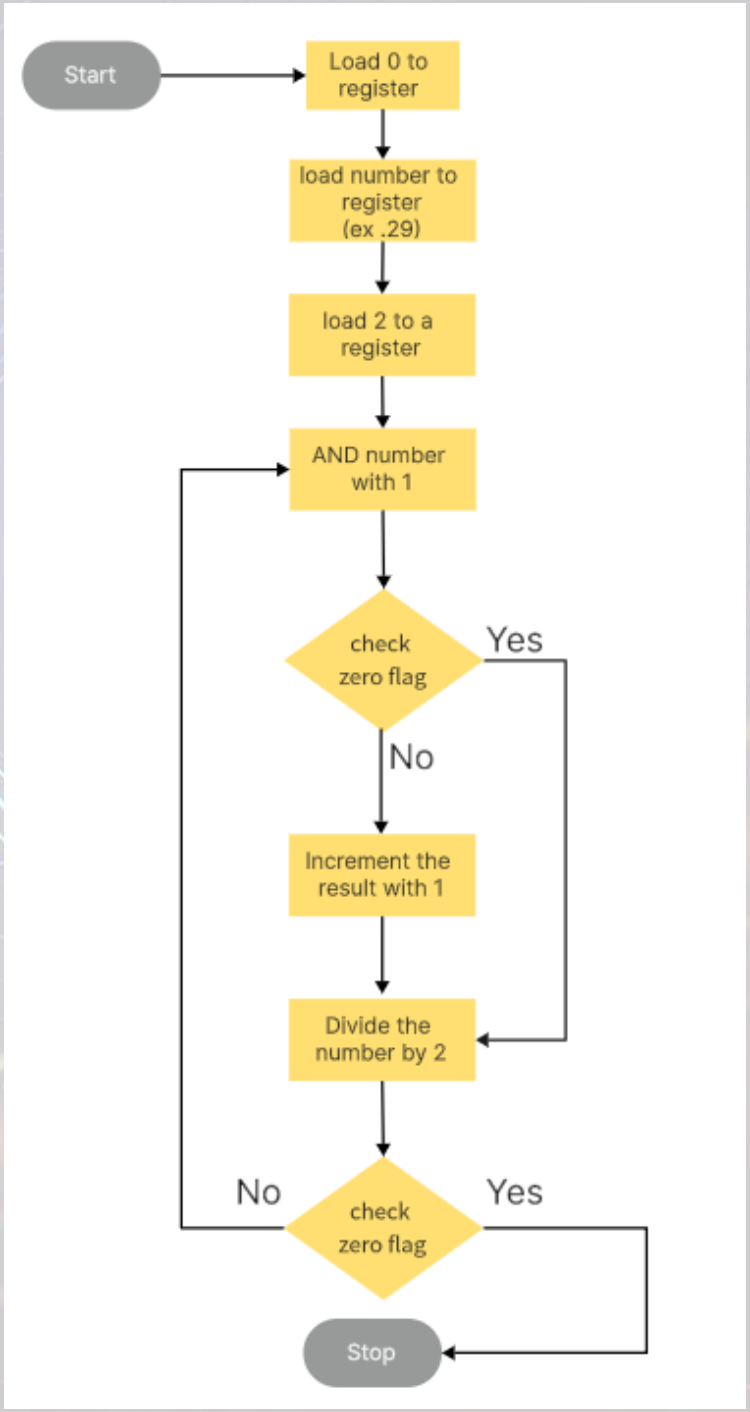
INSTRUCTION

# Flow Chart for the required program

## Instruction set

```
"0111110000000000",
"0111000000011101",
"0111001000000010",
"0100000010000001",
"1110000000000110",
"0010110110000001",
"0001000001000011",
"1110000000000000",
"1111000000000011",
```

[Source Code]




FLOWCHART

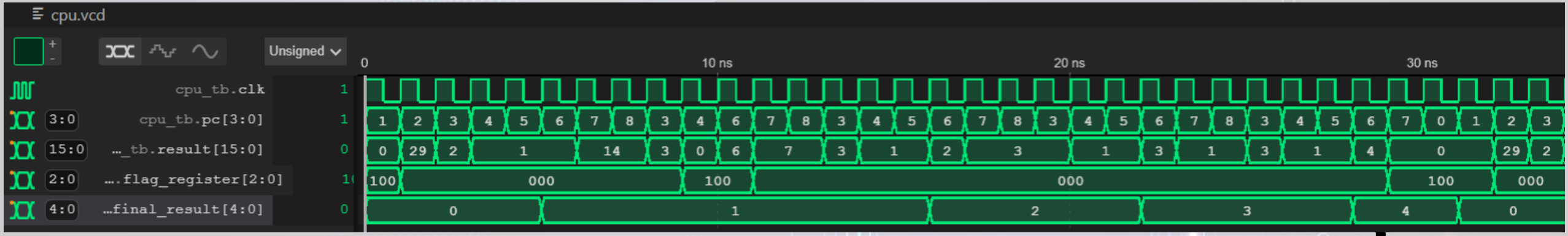
# Simulation and Verification

- VHDL is not just a description language but also facilitates simulation and verification. The code allows for testing and verifying the functionality of the simulated MIPS CPU under different scenarios and inputs.

SIMULATION

Source Code

# Simulation and Verification




- Final result is the result of the required program that is executed by the cpu.
- These program counts the number of ones in 29 (11101)
- After the number of ones in calculated and the answer is 4 it continues to complete the rest of the instructions but I make the new instruction is the repeat of the program

SIM.

[Source Code]

# FPGA & Real time

- FPGA or Field-Programmable Gate Array, is a type of programmable logic device that can be configured and reconfigured by a user or a designer after manufacturing. Unlike traditional application-specific integrated circuits (ASICs) that are custom-designed for specific applications and have a fixed function, FPGAs provide a flexible platform that allows users to implement digital circuits and functions according to their specific needs.

[Source Code]

FPGA

# FPGA & Real time


[Source Code]

FPGA

# Contact


- youssef.mohamed.34@h-eng.helwan.edu.eg
- moslem.sayed.shehata@h-eng.helwan.edu.eg
- mina-magdy88@h-eng.helwan.edu.eg
- mohamed-salem@h-eng.helwan.edu.eg


Source Code

# Thank You