Nanoprocessor

“ Effortless Simplicity ”

Team : 44

Team Members

👨‍💻 Aththanayake A.M.R.N

- *Designed 2-way 3-bit Multiplexer*
- *Developed Program Counter*
- *Created 3-bit Adder*
- *Wrote Test Benches for all components*

👨‍💻 Ifaz M.I.M

- *Built 8-way 4-bit Multiplexer*
- *Designed Register and Register Bank*
- *Implemented 3-to-8 Decoder*
- *Compiled the Final Lab Report*

👨‍💻 Kumarasinghe M.P

- *Implemented the Program ROM*
- *Helped build the Program Counter and Instruction Decoder*
- *Developed Bitwise Operator*
- *Integrated the full Nano Processor*
- *Built the Instruction Decoder*

👨‍💻 Lawanya K.K.H.G

- *Designed Half Adder and Full Adder*
- *Created Slow Clock Module*
- *Built 2-way 4-bit Multiplexer*
- *Developed 4-bit Adder-Subtractor Unit*
- *Implemented Seven Segment Display*

Introduction

What is this Nano Processor?

Nano processor is a custom-designed 4-bit processor built using VHDL. It simulates the core functionalities of a basic CPU, executing instructions such as arithmetic, bitwise, and control operations.

K**ey Features:**

- Supports 4-bit operations with 8 general-purpose registers
- Executes arithmetic (ADD, NEG) and bitwise (AND, OR, XOR, NOT) instructions
- Implements program flow control using conditional jumps (JZR)
- Includes a custom instruction decoder and program ROM

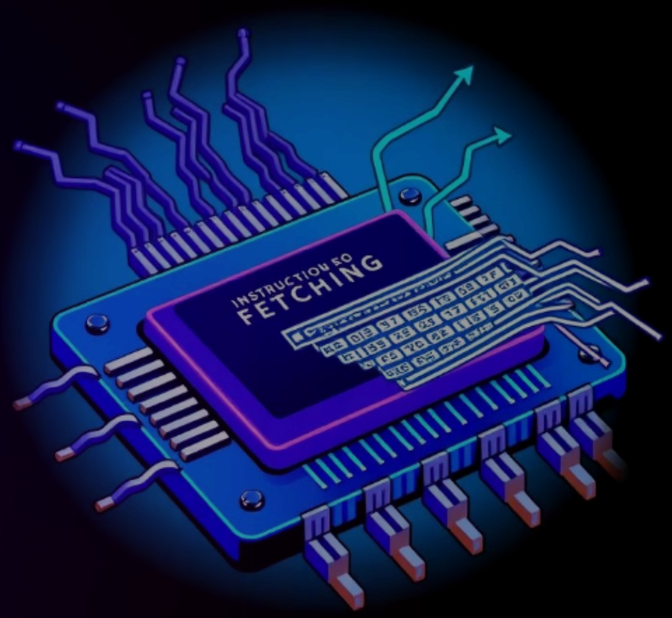
Components

1. **Program ROM**
2. **Program Counter (PC)**
3. **3-bit Adder**
4. **2-way 3-bit Multiplexer**
5. **Instruction Decoder**
6. **Register Bank**
7. **4-bit Registers**
8. **8-way 4-bit Multiplexer**
9. **2-way 4-bit Multiplexer**
10. **4-bit Add/Subtract Unit**
11. **Bitwise Operator**
12. **Half Adder and Full Adder Units**
13. **Slow Clock Generator**
14. **Seven Segment Display Decoder**

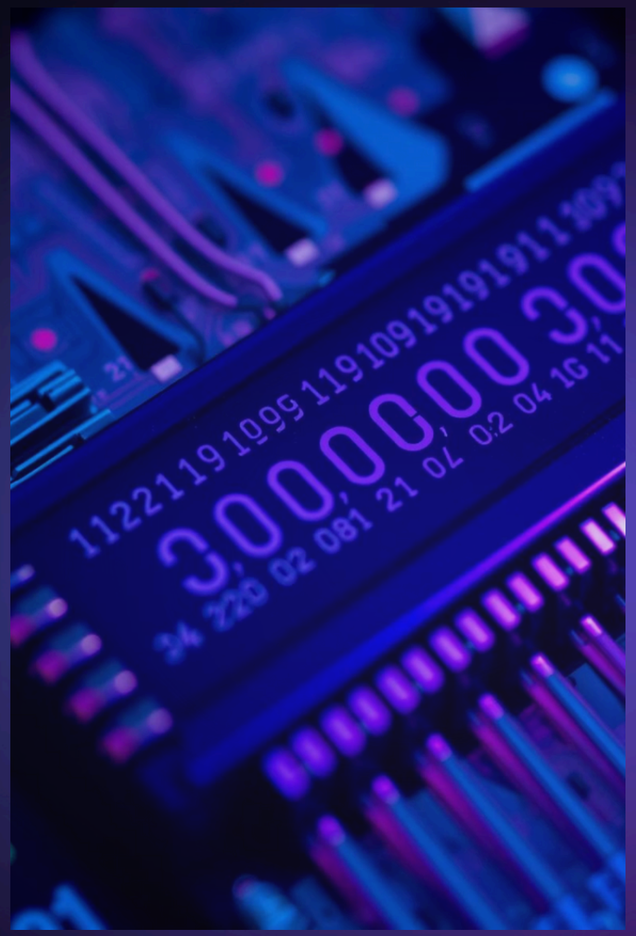
Program ROM

Key Points:

- Acts as the instruction memory of the nano processor.
- Stores up to 8 predefined 13-bit instructions.
- Accessed using a 3-bit Memory_Select address from the Program Counter.
- Outputs a specific instruction each clock cycle based on the current address.
- Instructions include MOVI, ADD, NEG, JZR, XOR, etc.
- Written using a VHDL ROM array and case logic.
- Supplies instructions directly to the Instruction Decoder for execution.
- Enables control over processor behavior by programming custom instruction sets.
- Essential for simulating a real-world instruction fetch stage.



Sample Instructions in ROM



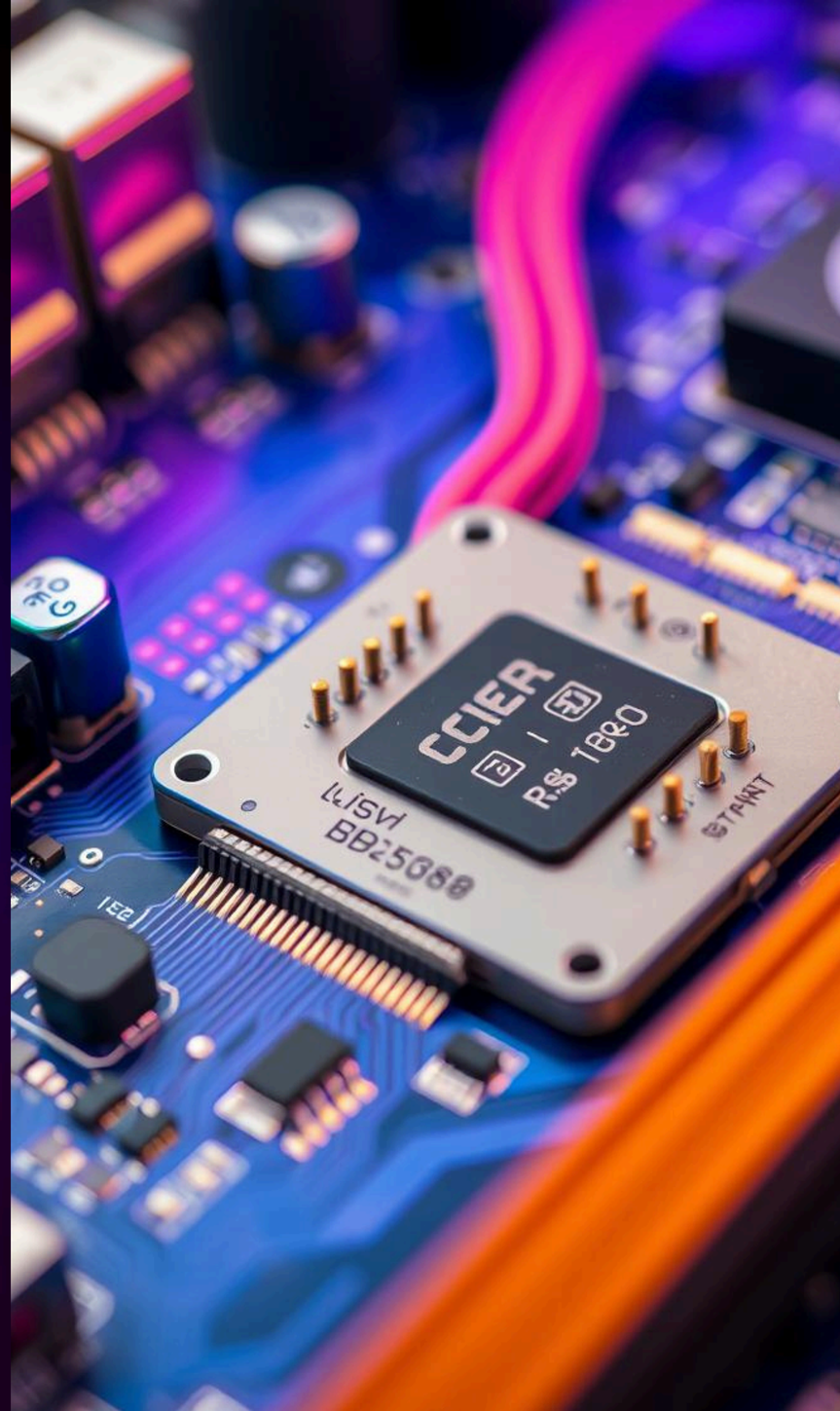
This table demonstrates data loading, arithmetic, and control flow. ROM allows the nano processor to execute a mini-program repeatedly.

000	101110000011	MOVI R7, 3	Load immediate value 3 to R7
001	010001000000	MOVI R1, 1	Load 1 into R1
002	111111001000	XOR R1, R7	Bitwise XOR between R1 and R7
003	011000000001	JZR R0, 3	Jump to address 3 if R0 is 0

Program Counter

- A Program Counter (PC) is a register in a processor that holds the memory address of the next instruction to be executed

- After each instruction, the PC increments automatically or updates based on control signals, enabling sequential program execution or jumps in control flow (e.g., loops, branches).



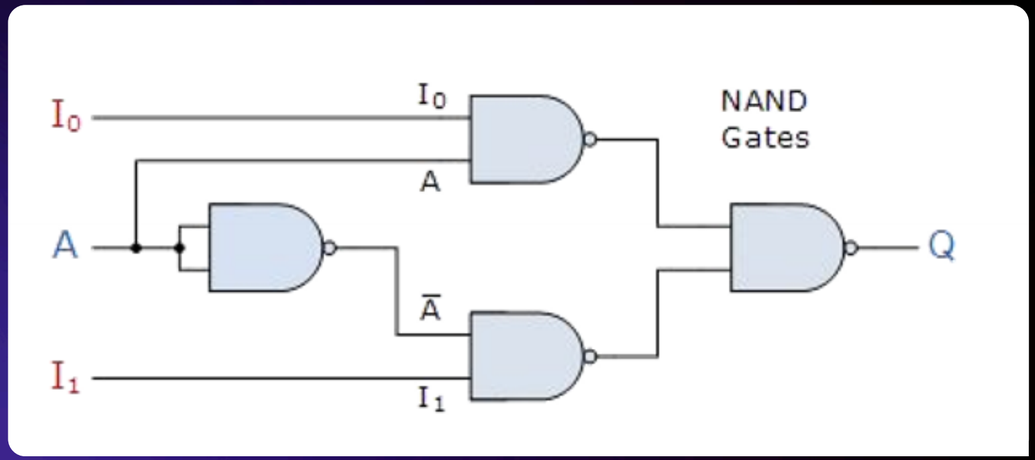
3-bit Adder

- A 3-bit adder is a digital circuit that performs binary addition on two 3-bit numbers, optionally including a carry-in from a previous stage.

- The output consists of a 3-bit sum and a carry-out, which is used for extending addition to higher bit-widths in multi-bit arithmetic operations.

2-way 3-bit Multiplexer

- A 2-way 3-bit multiplexer selects one of two 3-bit input values based on a single control (select) signal.



- It outputs the selected 3-bit input, allowing controlled data flow and decision-making in digital circuits, commonly used in processors for routing data between components or executing conditional operations.

Instruction Decoder

Key Points: Instruction Decoder

- Acts as the control logic unit of the nano processor.
- Takes in a 13-bit instruction from the Program ROM.
- Decodes the instruction to generate control signals for other processor components.
- Identifies the operation type using the opcode (top 3 bits of the instruction).
- Controls:
  - Register selection (Reg_Enable)
  - MUX inputs (MUX_A_Select, MUX_B_Select)
  - Immediate loading (Load_Select, Immediate)
  - Arithmetic control (AddSub_Select)
  - Bitwise logic (Bit_Select, Mode)
  - Jump behavior (Jump_Flag, Jump_Address)
- Implements conditional logic (e.g., enabling jump if register value is zero).
- Enables a wide range of operations like MOVI, ADD, NEG, JZR, XOR, AND, OR, NOT.
- Essential for the Decode stage in the processor’s Fetch-Decode-Execute cycle.
- Built using VHDL case statements based on opcode analysis.

Register Bank

Key Points: Register Bank

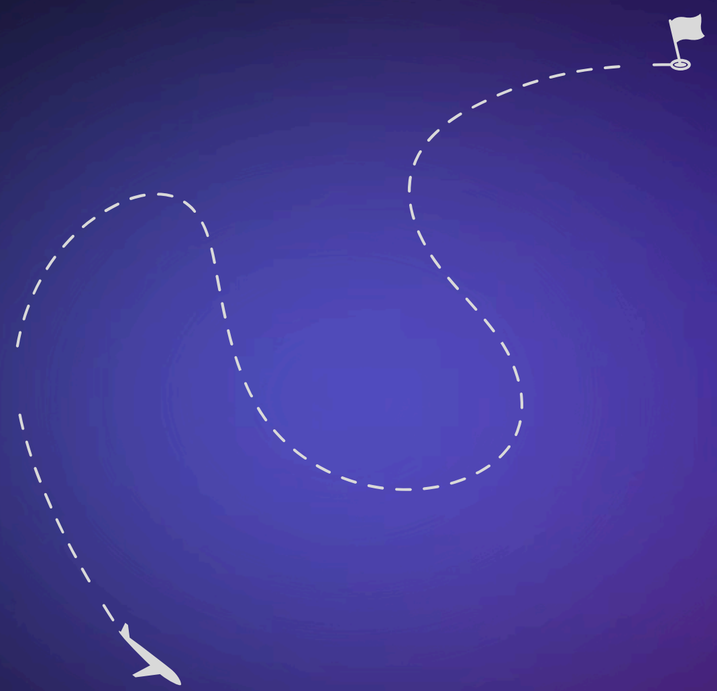
- Consists of 8 registers, each 4 bits wide (R0–R7).
- R0 is hardwired to 0000 and acts as a read-only constant zero register.
- Accepts input data via Value_In and writes to a register based on the 3-bit Reg_En signal.
- A 3-to-8 Decoder is used to convert Reg_En into one-hot reg_en_out signals.
- Only one register is enabled at a time for writing, ensuring proper data storage.
- All registers are synchronously updated on the rising edge of the clock (Clk).
- Each register supports reset functionality, clearing to 0000 when Reset is active.
- Modular design using reusable 4-bit register components (Reg_4_B).
- Output ports (R0 to R7) provide register values for use in ALU, bitwise operations, or display.
- Critical for instructions like MOVI, ADD, NEG, and bitwise logic, enabling data storage and transfer within the processor.

4-Bit Register

Key Points: 4-bit Register

- Stores a 4-bit data value for temporary or intermediate use.
- Used as a building block inside the Register Bank (R0–R7).
- Data is loaded on the rising edge of the clock, if Enable (En) is high.
- Includes a Reset input to clear the register to 0000.
- Only updates its value when both Enable and Clock conditions are met.
- Helps implement instructions that write data into specific registers (e.g., MOVI, ADD, NEG).
- Designed in VHDL using process blocks sensitive to Clk and Reset.
- Output (Q) provides the stored 4-bit value for use by the ALU or display units.

8-way 4-bit Multiplexer



Key Points: 8-way 4-bit Multiplexer

- Selects one out of 8 different 4-bit inputs (e.g., R0–R7).
- Controlled using a 3-bit select line (S) from the Instruction Decoder.
- Used to choose register values as inputs to the ALU or Bitwise Unit.
- Enables flexible data routing from the Register Bank to the execution units.
- Ensures only one register’s data is passed through based on control logic.
- Critical for instructions that operate on register pairs (e.g., ADD Ra, Rb, XOR Ra, Rb).
- Designed using a VHDL case statement that maps selection inputs to output.
- Helps simplify the processor’s operand selection mechanism.

2 way 4 bit Multiplexer

- Selects between two 4-bit inputs: S and Immediate
- Controlled by Load_Select signal
- '0' selects input S
- '1' selects input Immediate
- Output is assigned to Value_In
- Used for immediate value loading in processor operations

Timing and Fundamental Arithmetic

Clock Divider (Slow_Clk)

Converts a fast 50 MHz clock to a 0.5 Hz signal. This slows down execution for observation and testing. Essential for debugging step-by-step operations.

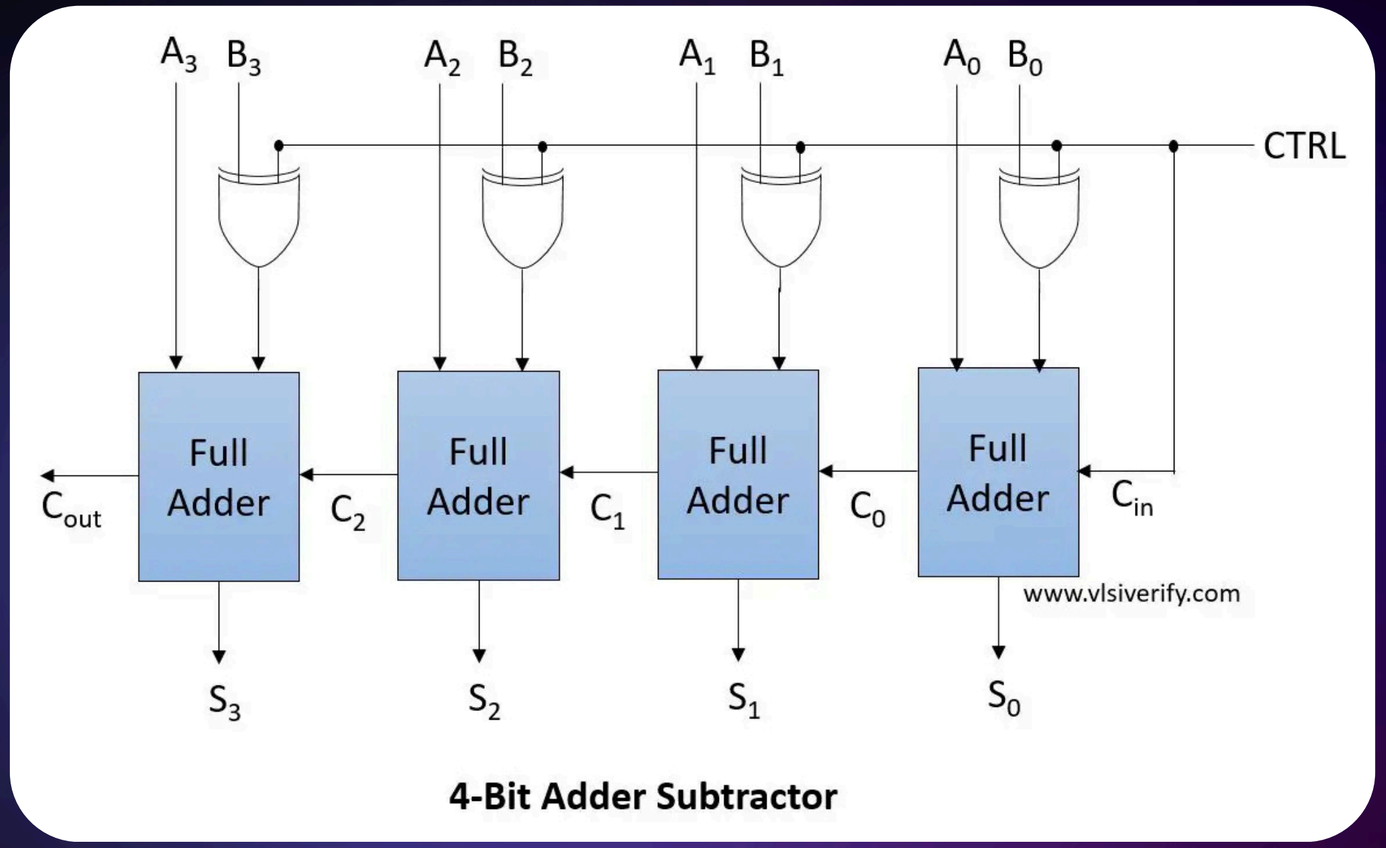
Half Adder (HA)

Computes sum and carry for two 1-bit inputs. Specifically, The Sum is the answer you'd write down.
The Carry is what you'd pass to the next column, like when adding in math.
.It forms the base for binary arithmetic.

Full Adder (FA)

Adds three 1-bit inputs: A, B, and Carry-In. Specifically, . Built from two Half Adders, it forms the core of multi-bit binary addition.

ADD/SUB Unit (ADD_SUB)



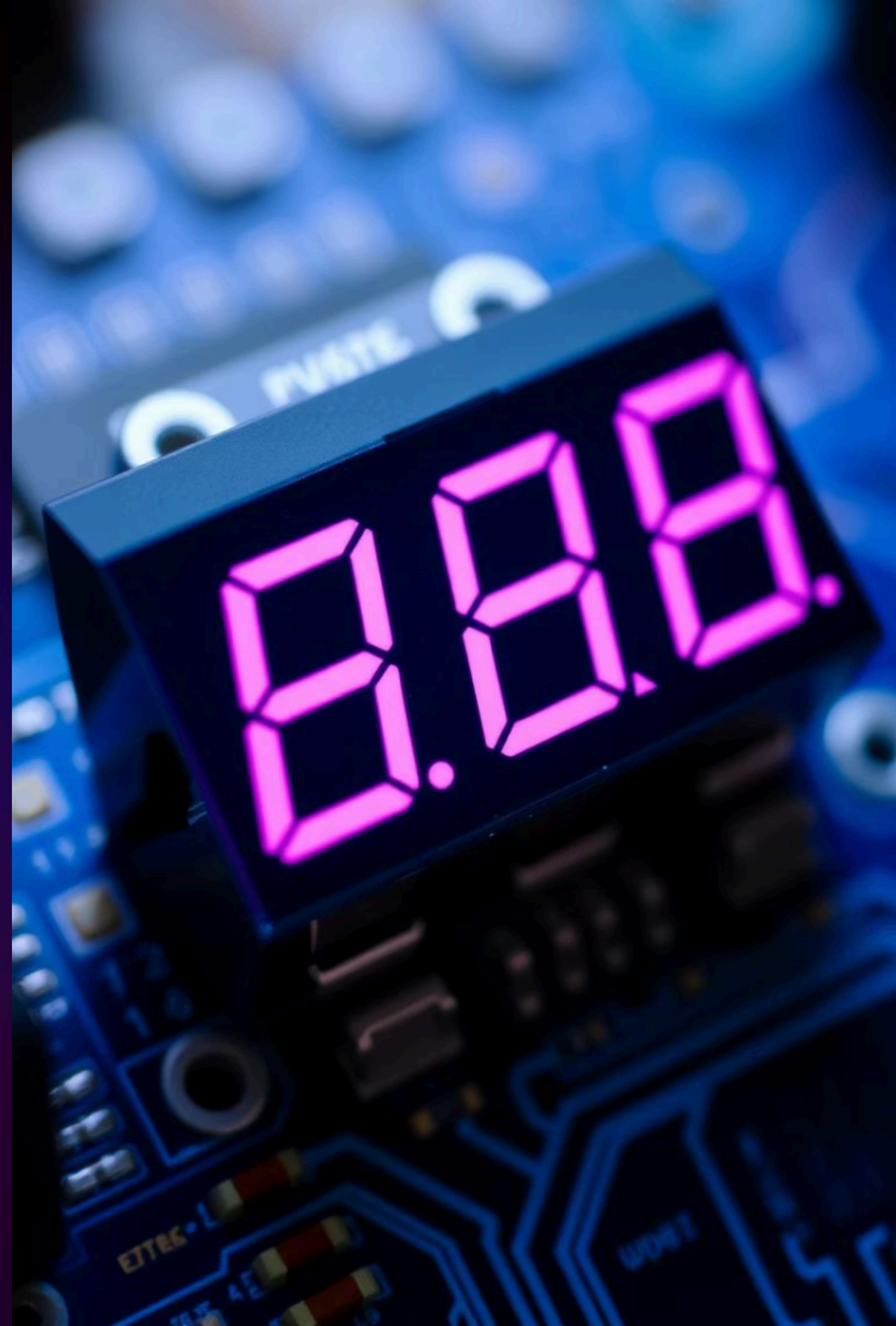
4-Bit Adder Subtractor

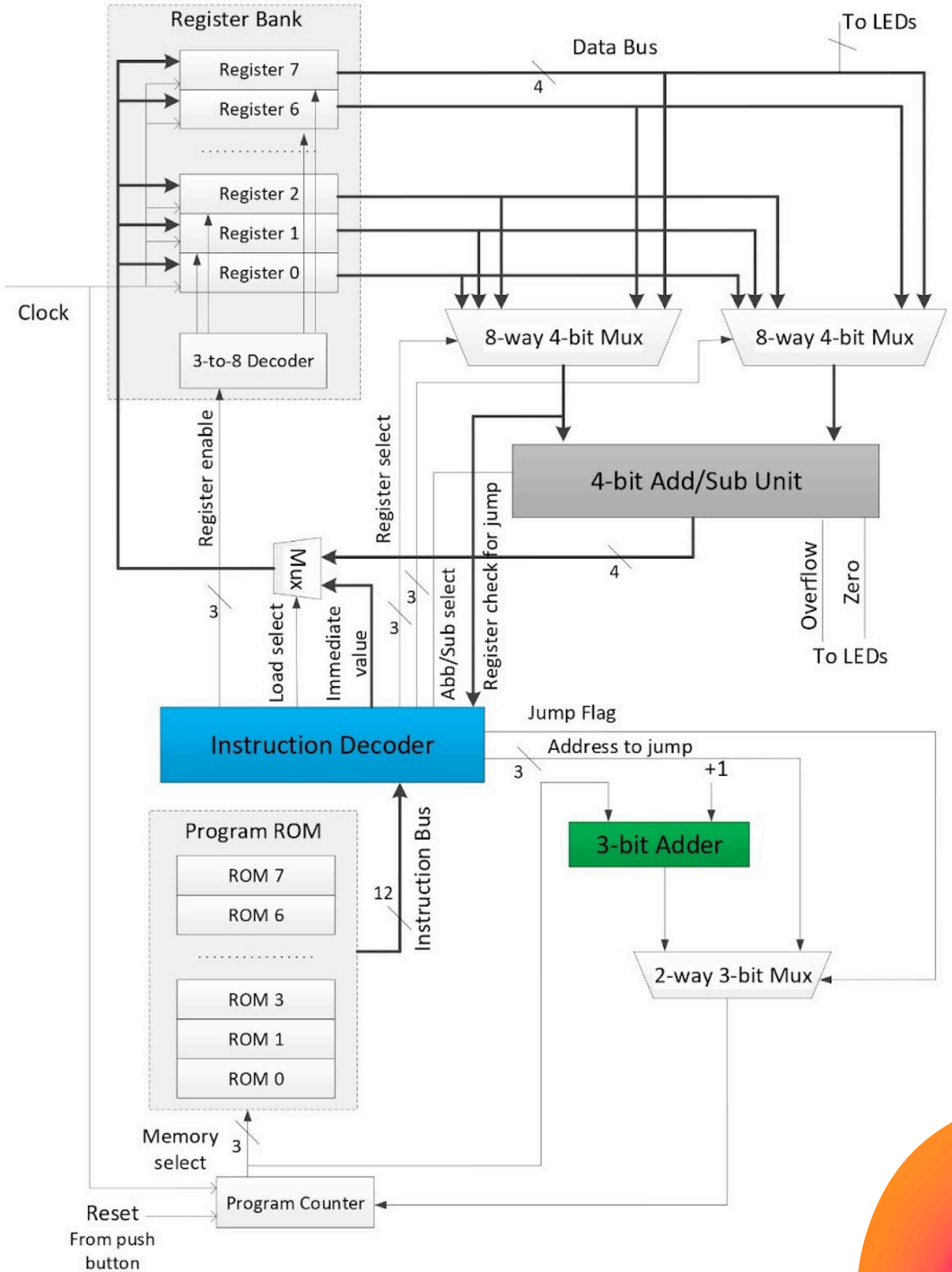
- Performs 4-bit addition or subtraction based on AddSub_Select; uses XOR logic to invert B inputs for subtraction.
- Default value for AddSub_Select in ‘0’.
- Built using cascaded full adders; provides Sum, Zero, and Overflow outputs for arithmetic and condition checks.

7-Segment Display

- Implements a 16-entry ROM lookup table that maps 4-bit binary inputs (0–15) to 7-bit outputs for a 7-segment display.

- Displays hexadecimal digits (0–F) by directly driving the appropriate segments based on the input address.



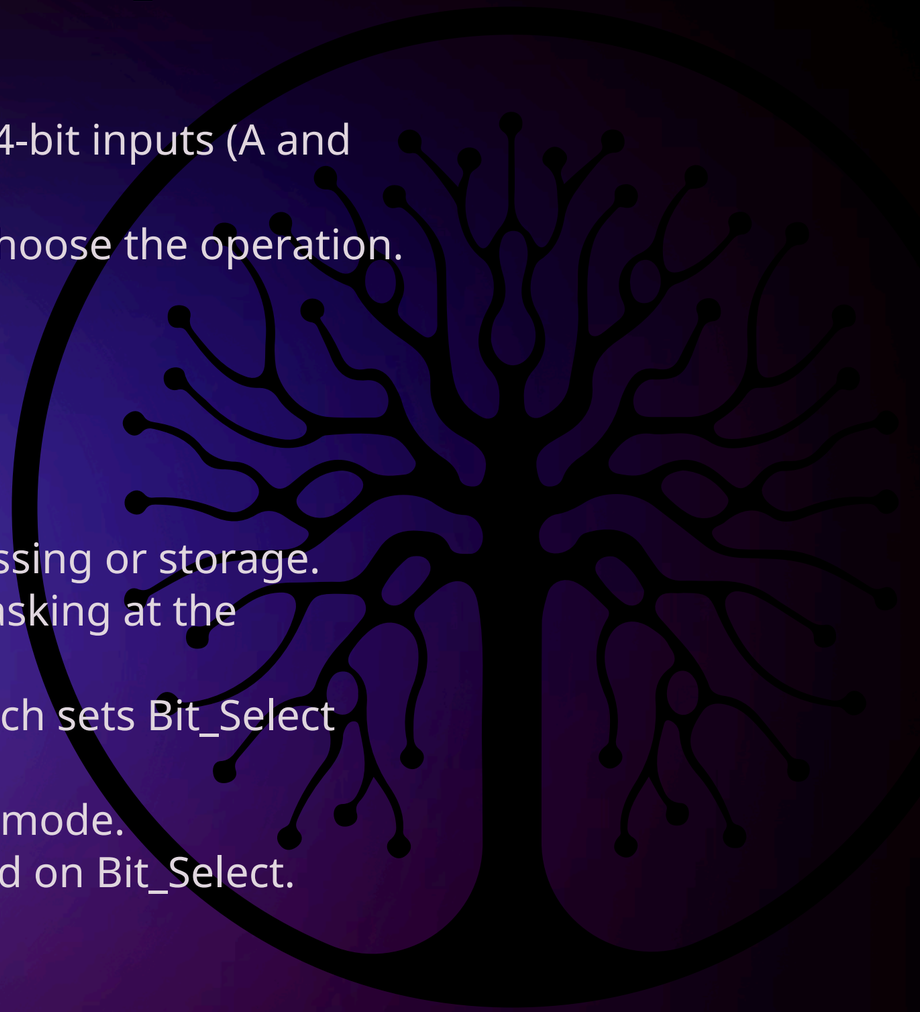

Register Bank
Register 7
Register 6
Register 2
Register 1
Register 0
3-to-8 Decoder
Data Bus
4
To LEDs
Clock
8-way 4-bit Mux
8-way 4-bit Mux
4-bit Add/Sub Unit
Register enable
Register select
Register check for jump
Mux
3
Load select
Immediate value
3
Abb/Sub select
3
4
Overflow
Zero
To LEDs
Instruction Decoder
Jump Flag
Address to jump
3
+1
3-bit Adder
Program ROM
ROM 7
ROM 6
ROM 3
ROM 1
ROM 0
12
Instruction Bus
2-way 3-bit Mux
Memory select
3
Program Counter
Reset
From push button

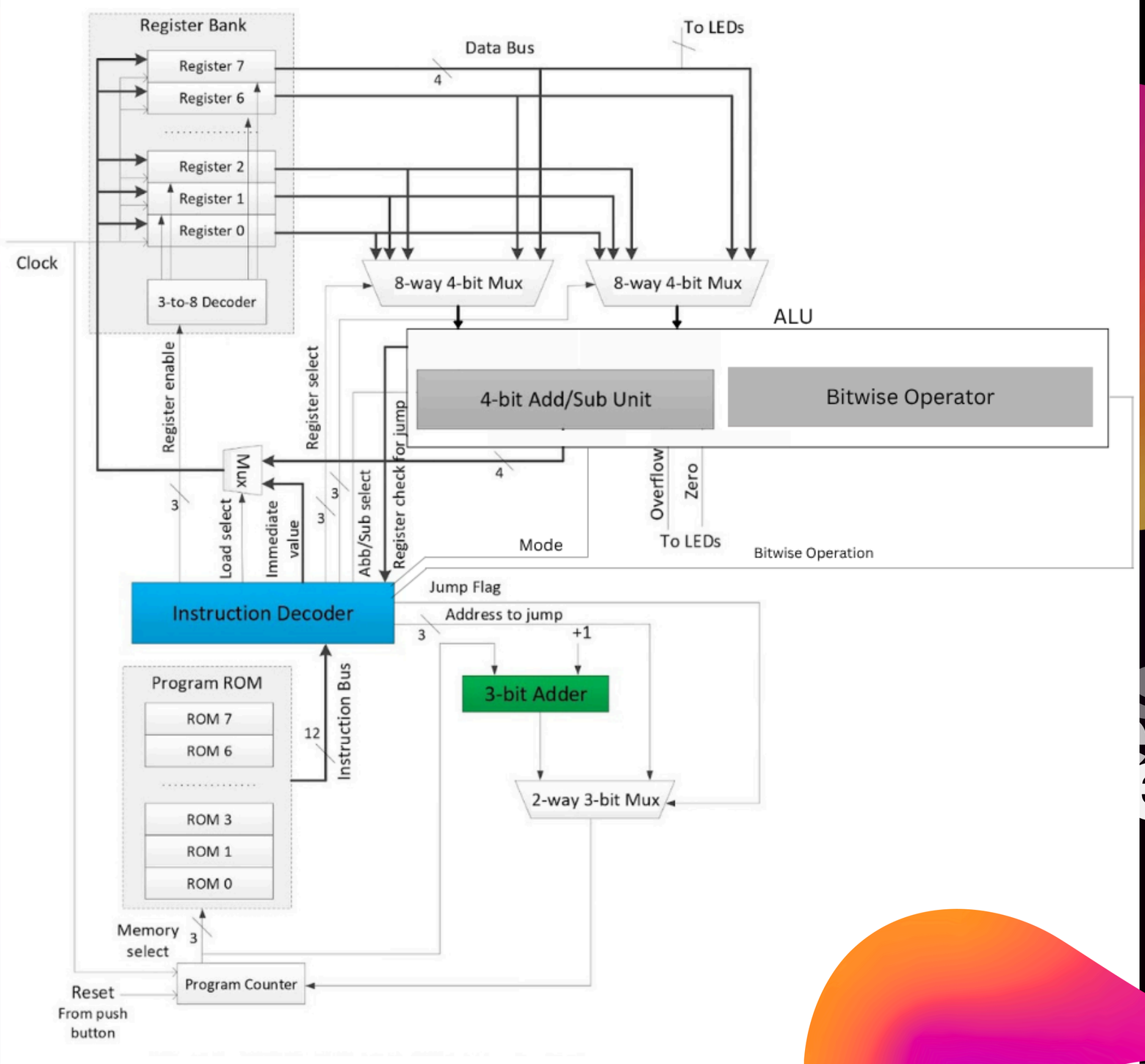
What we did for improvements

Bitwise Operator

Key Points: Bitwise Operator

- Performs bit-level logical operations on two 4-bit inputs (A and B).
- Controlled using a 2-bit Bit_Select signal to choose the operation.
- Supports four operations:
  - "00" → NOT B (bitwise complement)
  - "01" → A AND B
  - "10" → A OR B
  - "11" → A XOR B
- Output is a 4-bit result used in further processing or storage.
- Enables logical decision-making and data masking at the hardware level.
- Integrated with the Instruction Decoder, which sets Bit_Select and mode.
- Plays a vital role in the ALU's logic operation mode.
- Designed using a VHDL case statement based on Bit_Select.




Register Bank
Register 7
Register 6
Register 2
Register 1
Register 0
3-to-8 Decoder
Data Bus
4
To LEDs
Clock
8-way 4-bit Mux
8-way 4-bit Mux
ALU
4-bit Add/Sub Unit
Bitwise Operator
Register enable
Register select
Register check for jump
Mux
3
Load select
Immediate value
3
Abb/Sub select
4
Overflow
Zero
To LEDs
Mode
Bitwise Operation
Jump Flag
Instruction Decoder
Address to jump
3
+1
3-bit Adder
Program ROM
ROM 7
ROM 6
ROM 3
ROM 1
ROM 0
12
Instruction Bus
2-way 3-bit Mux
Memory select
3
Program Counter
Reset
From push button

How it works?? (XOR)

Instruction Fetch

- The Program Counter sends the current address to the Program ROM.
- The ROM outputs a 13-bit instruction, e.g., 111 RRR RRR 0000 (XOR Ra, Rb).

Instruction Decode

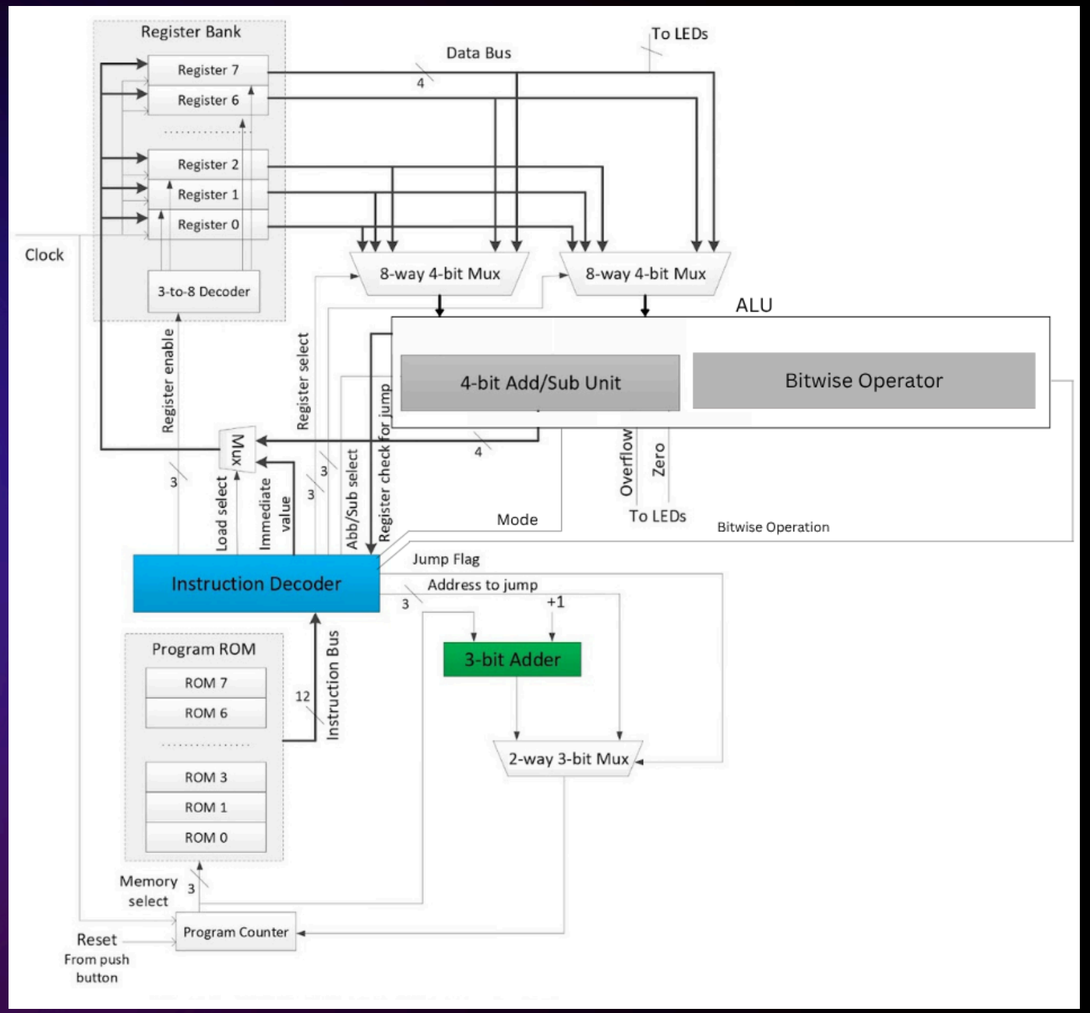
- The Instruction Decoder reads the opcode 111, which indicates a bitwise XOR operation.
- It sets:
  - Reg_Enable = destination register (Ra)
  - MUX_A_Select = Ra
  - MUX_B_Select = Rb
  - Mode = 1 (to activate the Bitwise Operator)
  - Bit_Select = "11" (to select XOR operation)

Operand Fetch

- The 8-way 4-bit MUX selects values from Ra and Rb based on the MUX_A_Select and MUX_B_Select signals.
- These values are fed as inputs A and B to the Bitwise Operator.

Execution (Bitwise Operator)

- The Bitwise Operator checks Bit_Select = "11" and performs A XOR B.
- This means each bit of A is XOR'ed with the corresponding bit of B.



Result Write-Back

- The output of the Bitwise Operator is routed to the destination register (Ra).
- The Reg_Enable signal activates Ra for writing.
- On the next rising clock edge, Ra is updated with the XOR result.

Operand Fetch

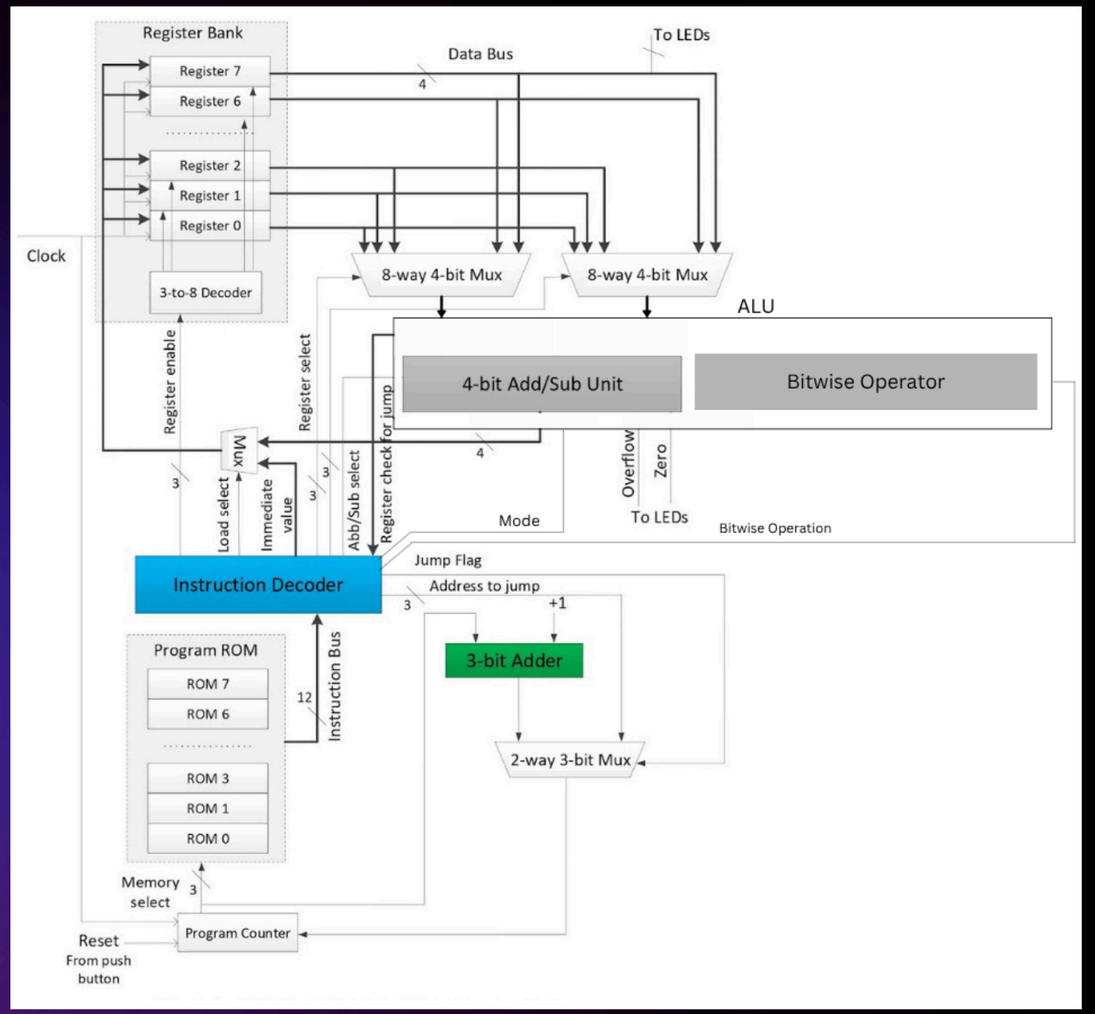
- The 8-way 4-bit MUX selects values from Ra and Rb based on the MUX_A_Select and MUX_B_Select signals.
- These values are fed as inputs A and B to the Bitwise Operator.

Execution (Bitwise Operator)

- The Bitwise Operator checks Bit_Select = "11" and performs A XOR B.
- This means each bit of A is XOR’ed with the corresponding bit of B.

Result Write-Back

- The output of the Bitwise Operator is routed to the destination register (Ra).
- The Reg_Enable signal activates Ra for writing.
- On the next rising clock edge, Ra is updated with the XOR result.



Practical Applications of the Nano Processor

1. LED Pattern Controller
   - Use the processor to control LED blink sequences or chaser patterns by writing different instruction routines.
   - For example: loop through outputs, increment a counter, or perform bitwise operations to generate effects.

2. 7-Segment Display Driver
   - Output register values directly to a 7-segment display decoder.
   - Display numbers, hex values, or computation results from operations like ADD, MOVI, etc.

3. Motor Control Logic
   - Use the instruction set to simulate basic DC motor control: start, stop, or change direction based on register values or conditions (JZR as decision logic).
   - Ideal for simple automation tasks.

5. Temperature or Sensor Threshold Alert (Simulated)
   - Simulate reading a sensor value into a register, then compare it.
   - If the value is zero or below a set point, jump to a "trigger" instruction (e.g., turning on an alert LED).

“ Effortless Simplicity ”