Instruction ROM Design – 16 × 8

1 Introduction

Objective
Design, implement and simulate a **16-word × 8-bit Read-Only Memory (ROM)** to serve as the program / lookup memory for a soft microcontroller core.

Why it's important
Every CPU fetches instructions from non-volatile memory. A small on-chip ROM is perfect for boot code, fixed lookup tables, or microcode in resource-constrained FPGA/ASIC designs.

2 Functional Specifications

Signal	Width	Dir.	Description
`in_address`	4	Input	Address from Program Counter (0 – 15)
`out_instructions`	8	Output	Instruction / data stored at `in_address`

Parameter	Purpose	Default
Depth	Number of locations	16
Width	Bits per location	8

3 Theory of Operation

1. **Random-access read-only array** – contents fixed at synthesis.
2. Address (`in_address`) acts as an index; data is returned combinationally.
3. In a real microcontroller, the Program Counter increments each cycle, feeding sequential addresses to the ROM.
4. Because the ROM is pure combinational logic (no clock), access latency = one propagation delay.

4 Example Walk-Through

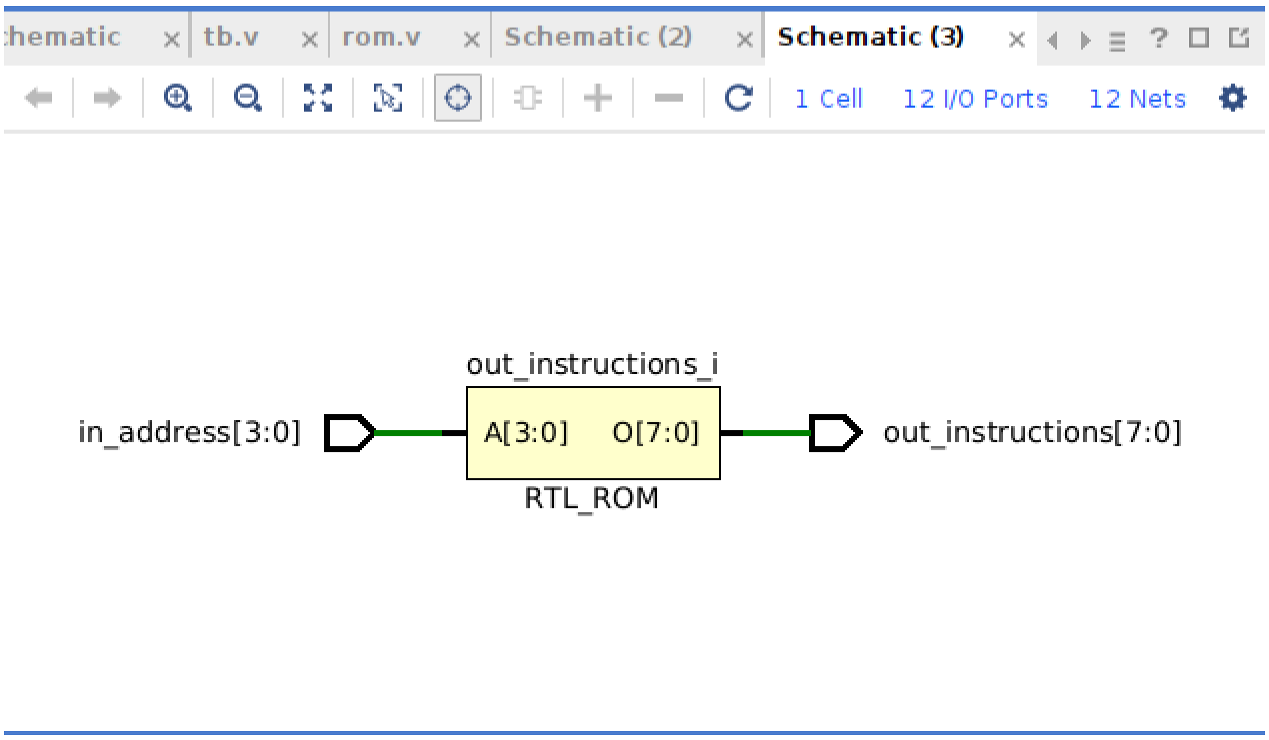
`in_address` (PC)	Stored Instruction (`out_instructions`)	Mnemonic*
`0000`	`0001 0010`	`LOAD R1, #2`
`0001`	`0010 0101`	`ADD R1, R5`
`0010`	`0011 0110`	`SUB R1, R6`
`0100`	`0101 0001`	`OR R0, R1`
`0111`	`1000 0010`	`JMP 2`

5 Verilog RTL Code

```
module rom (
   input  [3:0] in_address,      // 4-bit address (0-15)
   output [7:0] out_instructions // 8-bit instruction
);

   // 16 × 8 ROM
   reg [7:0] mem [15:0];

   // Hard-coded contents (example program)
   initial begin
      mem[0]  = 8'b0001_0010; // LOAD  R1, #2
      mem[1]  = 8'b0010_0101; // ADD   R1, R5
```

```
        mem[2]  = 8'b0011_0110; // SUB   R1, R6
        mem[3]  = 8'b0100_0011; // AND   R0, R3
        mem[4]  = 8'b0101_0001; // OR    R0, R1
        mem[5]  = 8'b0110_0010; // XOR   R0, R2
        mem[6]  = 8'b0111_0000; // NOT   R0
        mem[7]  = 8'b1000_0010; // JMP   2
        mem[8]  = 8'b1001_0001; // JZ    1
        mem[9]  = 8'b1010_0000; // STORE R0, MEM
        mem[10] = 8'b1011_0011; // LOAD  R3, MEM
        mem[11] = 8'b1100_0000; // NOP
        mem[12] = 8'b1101_0000; // HALT
        mem[13] = 8'b0000_0000; // Reserved
        mem[14] = 8'b0000_0000; // Reserved
        mem[15] = 8'b0000_0000; // Reserved
    end

    // Combinational read
    assign out_instructions = mem[in_address];

endmodule
```

RTL:-



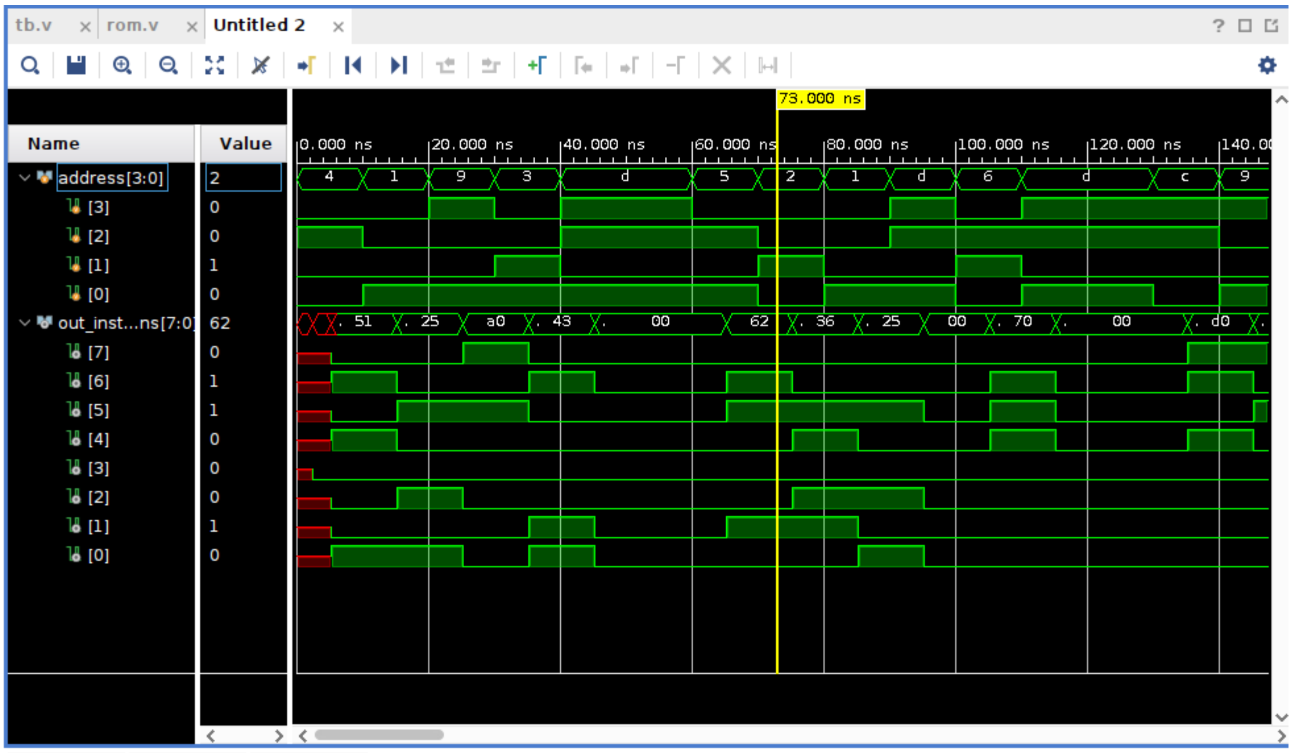
-

TESTBENCH:-

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 07/05/2025 06:55:00 PM
// Design Name:
// Module Name: tb
// Project Name:
```

```
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module tb;
reg [3:0]address;
wire [7:0]out_instructions;

rom uut(
.in_address(address),
.out_instructions(out_instructions));

initial begin
repeat (20)  begin
address = $random %16;
#10;
end
end
endmodule
```

7 Simulation Results



-

8 Applications

- **Instruction ROM** in a home-built 8-bit microcontroller
- **Lookup tables**: sine/cosine, CRC polynomials, font glyphs
- **Microcode storage** for complex CISC instructions
- **Boot ROM / reset vectors** in embedded systems

9 Conclusion

- Created and verified a **16 × 8 combinational ROM**, the first permanent memory block in the MCU project.
- Demonstrated correct address-to-data mapping in simulation.
- Next step: hook this ROM to a Program Counter and build the Instruction Decoder.