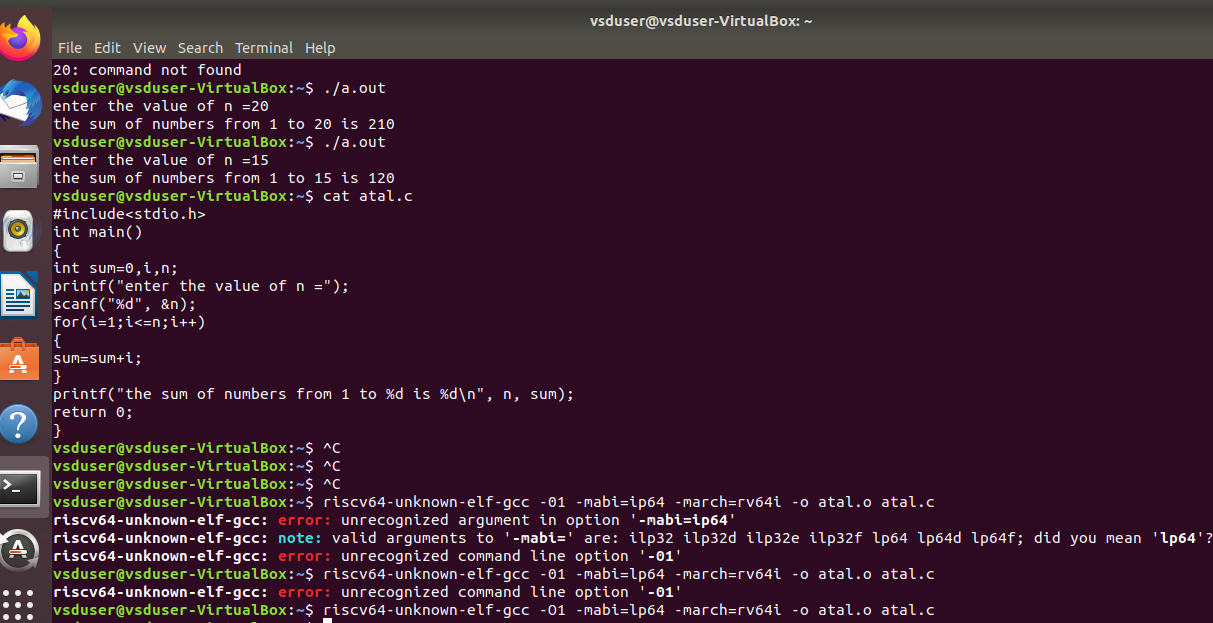
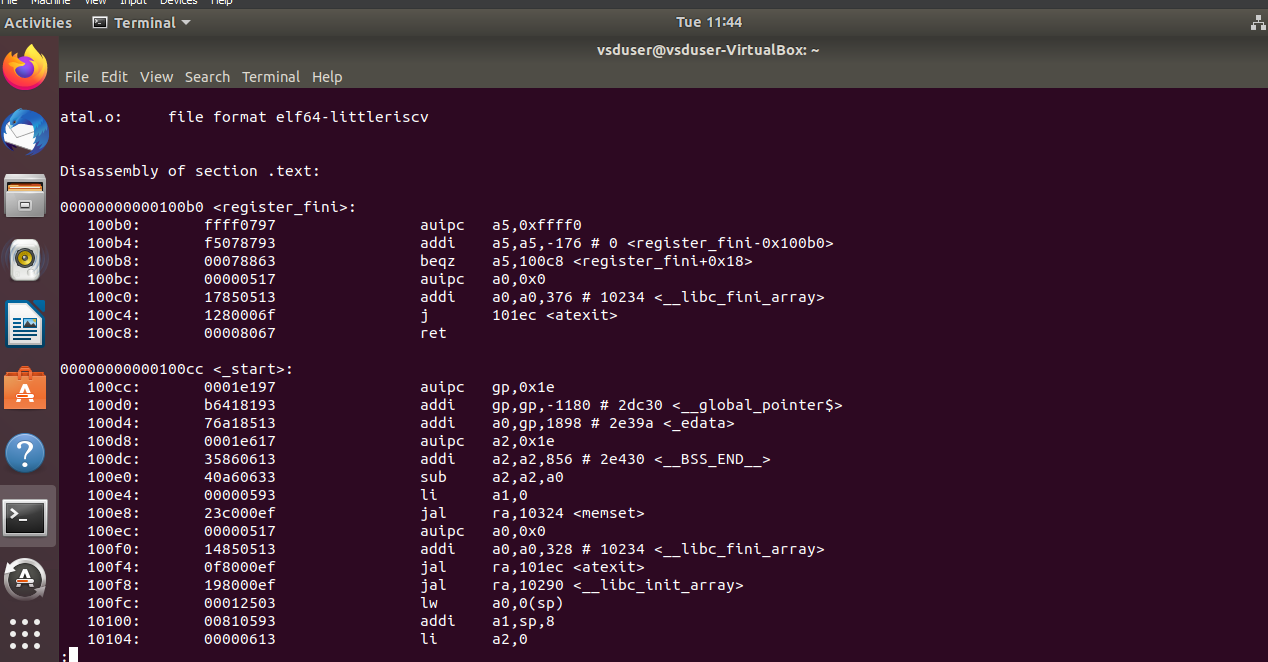


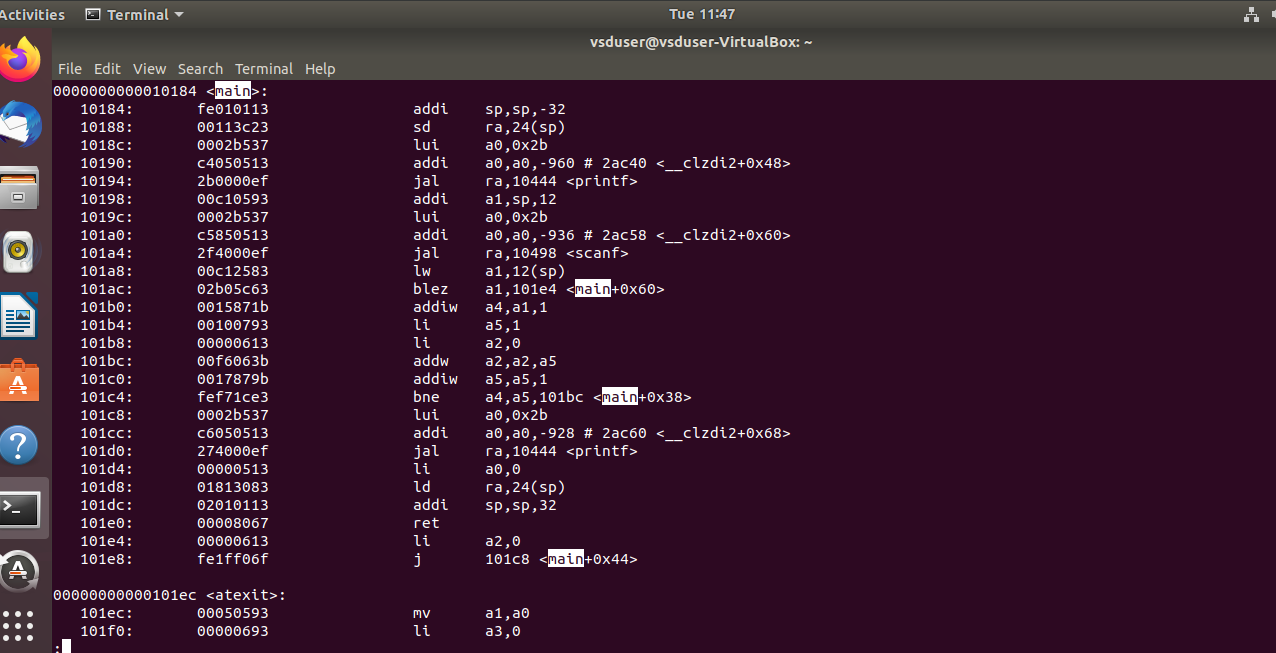
riscv64-unknown-elf-gcc -O1 -mabi=lp64 -march=rv64i -o sum\_1ton.o sum\_1ton.c

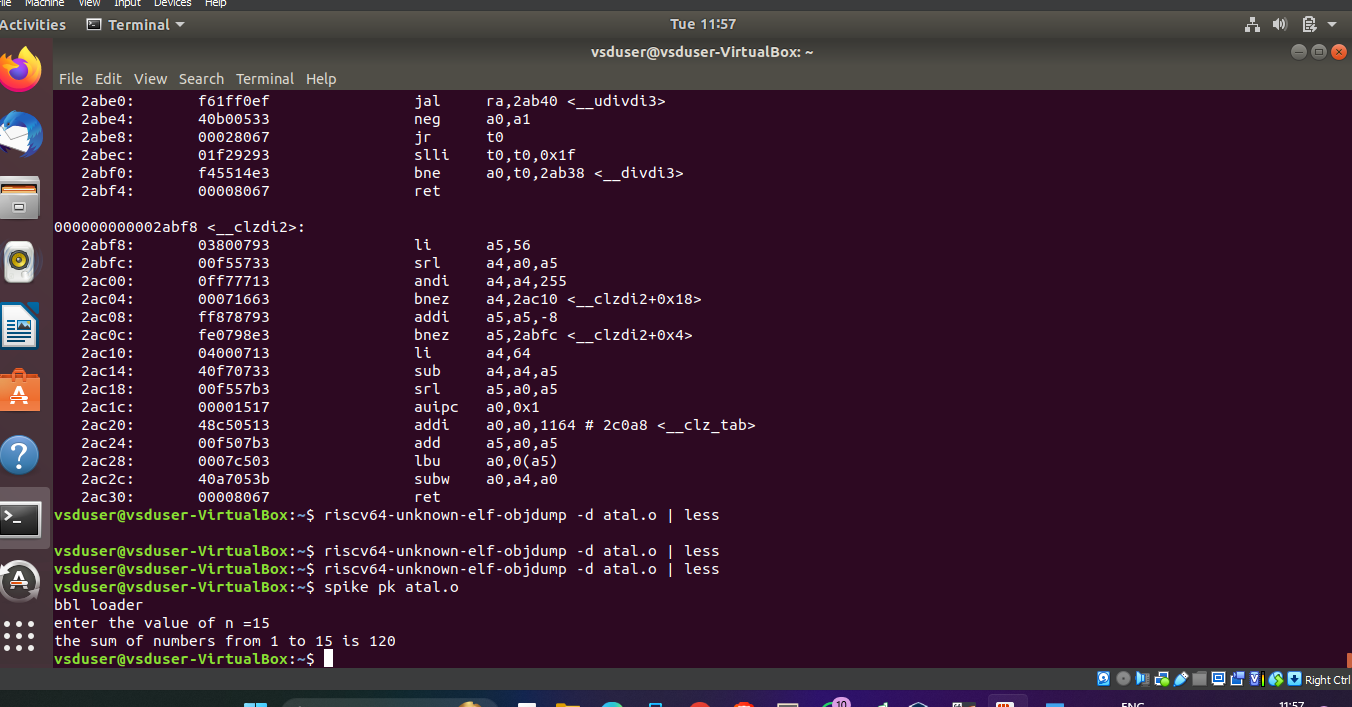
riscv64-unknown-elf-gcc -O1 -mabi=lp64 -march=rv64i -o sum\_1ton.o sum\_1ton.c

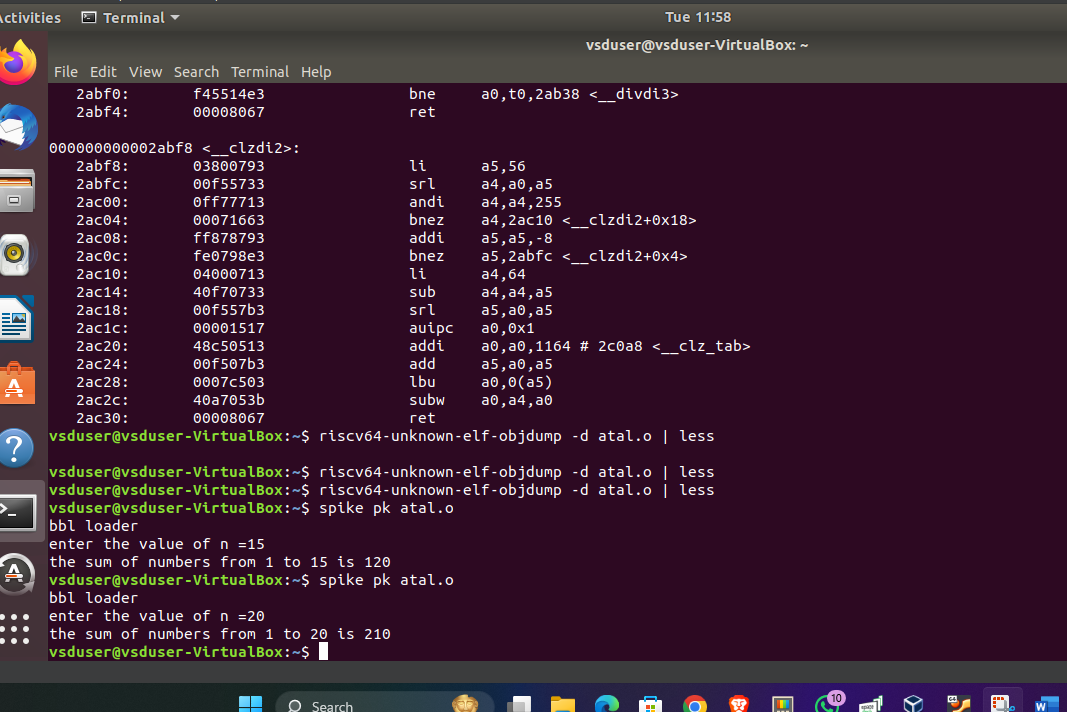
riscv64-unknown-elf-objdump -d sum\_1ton.o

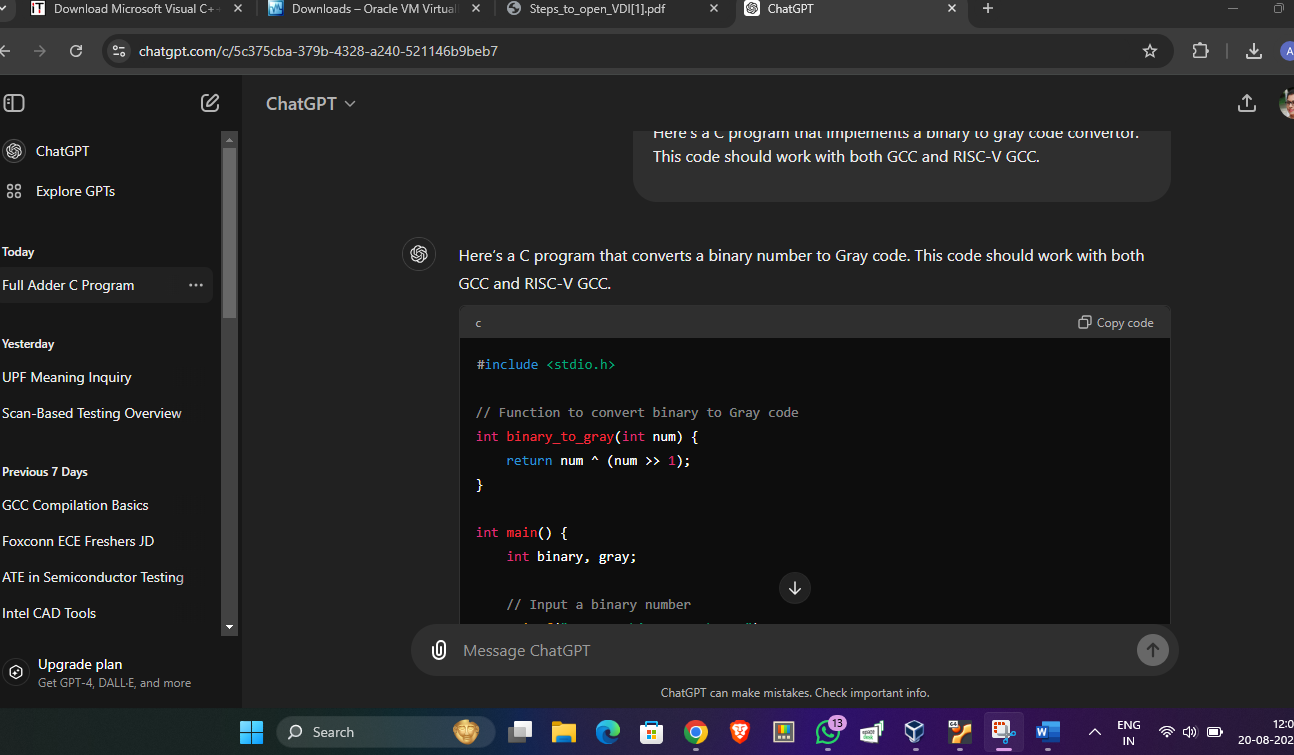












#include <stdio.h>

// Function to convert binary to Gray code

int binary\_to\_gray(int num) {

return num ^ (num >> 1);

}

int main() {

int binary, gray;

// Input a binary number

printf("Enter a binary number: ");

scanf("%d", &binary);

// Convert binary to Gray code

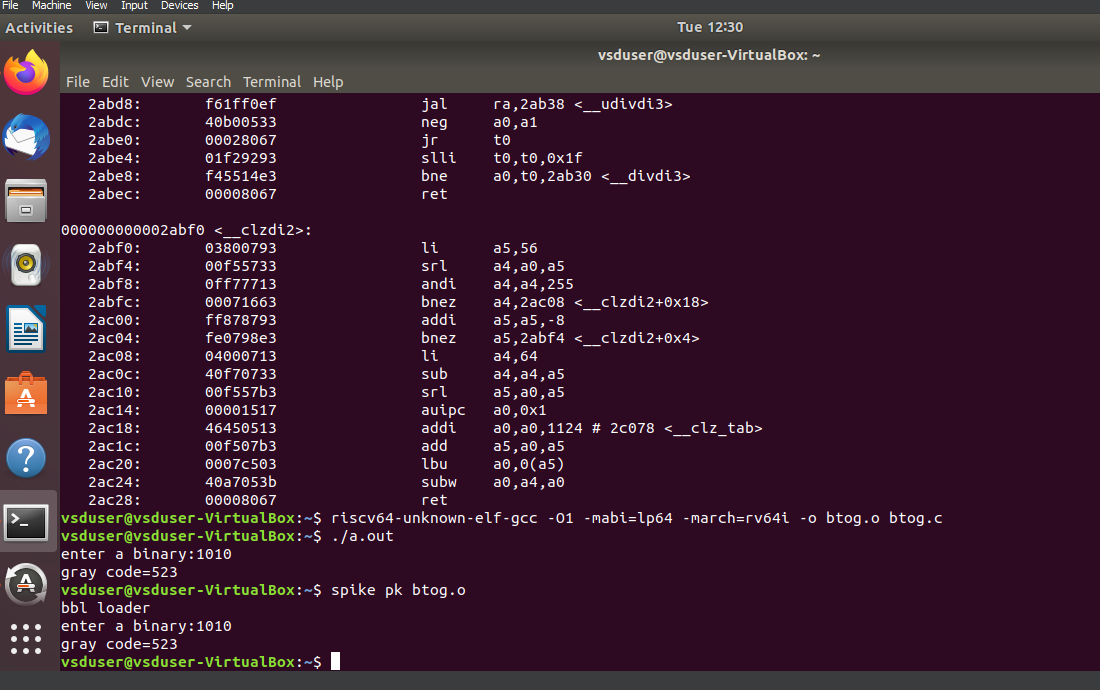
gray = binary\_to\_gray(binary);

// Output the Gray code

printf("Gray code = %d\n", gray);

return 0;

}



**Explanation:**

1. **Opcode (7 bits - 0110011):** Indicates that this is an R-type instruction.
2. **rd (5 bits - 01011):** Specifies that the result will be stored in a1 (x11).
3. **funct3 (3 bits - 100):** Specifies that this is a bitwise XOR operation.
4. **rs1 (5 bits - 01011):** Specifies that the first operand is a1 (x11).
5. **rs2 (5 bits - 01111):** Specifies that the second operand is a5 (x15).
6. **funct7 (7 bits - 0000000):** Differentiates the XOR operation.

This instruction tells the processor to perform a bitwise XOR between the values in a1 (register x11) and a5 (register x15), and then store the result back in a1 (register x11).

Xor a1,a1,a5

**Explanation:**

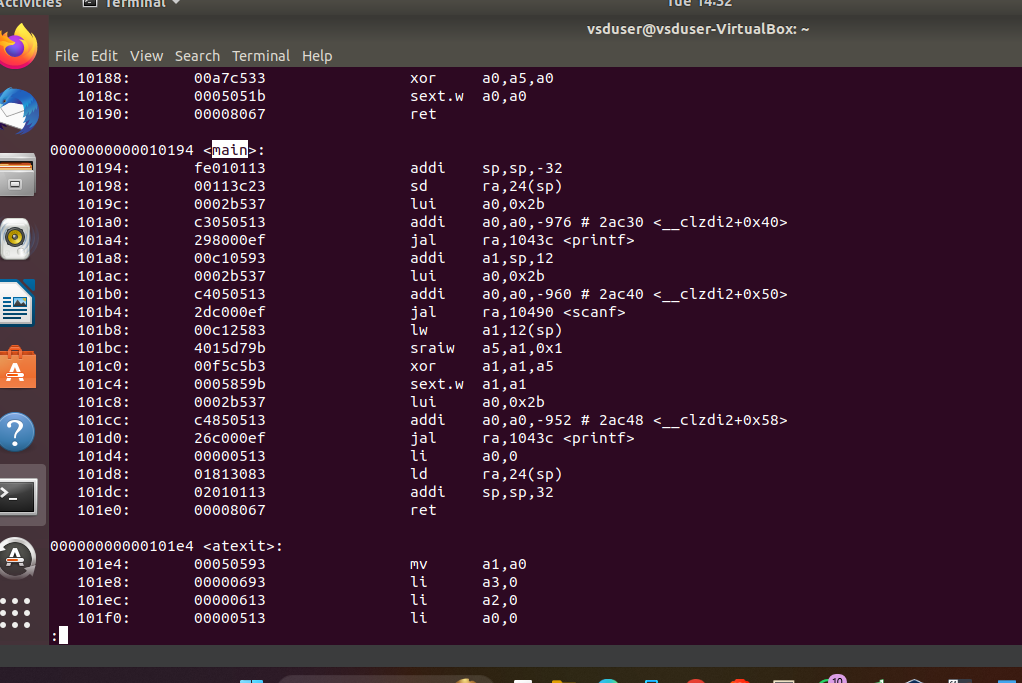
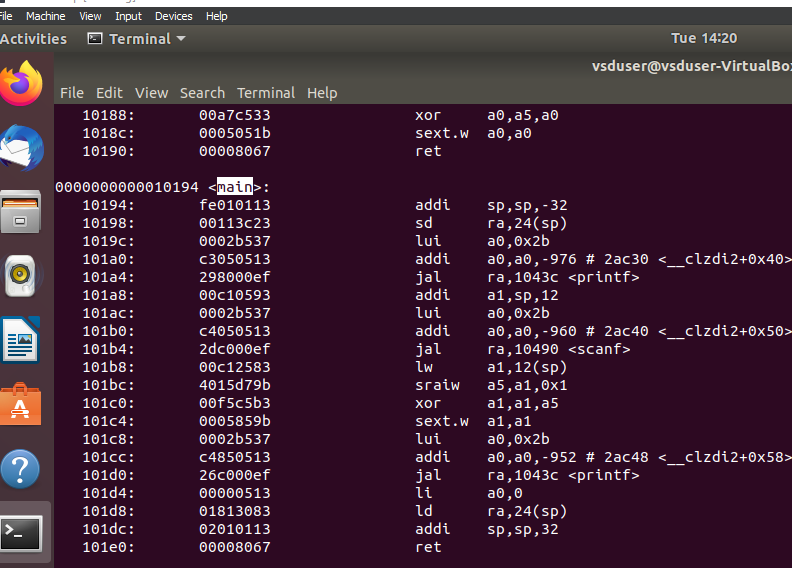
* **Opcode (0110011):** Specifies that this is an R-type instruction.
* **funct7 (0000000):** Indicates a logical operation (and differentiates XOR from similar instructions).
* **rs2 (01111):** Specifies the second source register, which is x15 (a5).
* **rs1 (01011):** Specifies the first source register, which is x11 (a1).
* **funct3 (100):** Indicates the XOR operation.
* **rd (01011):** Specifies the destination register, which is x11 (a1).

**Operation:**

This instruction performs a bitwise XOR between the contents of the a1 and a5 registers (i.e., x11 and x15), and the result is stored back into the a1 register (i.e., x11).

**Example:**

* **Before Execution:**
  + a1 = 0b1010 (decimal 10)
  + a5 = 0b1100 (decimal 12)
* **After Execution:**
  + a1 = 0b0110 (decimal 6), which is the result of 0b1010 XOR 0b1100.



Xpr a1,a1,a5

Lui a0,0x2b

module riscv\_processor (

input wire clk,

input wire reset,

// Other inputs/outputs

);

// Register file

reg [31:0] registers [0:31];

// Instruction register

reg [31:0] instruction;

// Control signals

wire [6:0] opcode;

wire [4:0] rs1, rs2, rd;

wire [2:0] funct3;

wire [6:0] funct7;

// Extract instruction fields

assign {funct7, rs2, rs1, funct3, rd, opcode} = instruction;

// Opcode for XOR instruction

localparam OPCODE\_R\_TYPE = 7'b0110011;

localparam FUNCT3\_XOR = 3'b100;

localparam FUNCT7\_XOR = 7'b0000000;

// Main execution block

always @(posedge clk or posedge reset) begin

if (reset) begin

// Initialize registers or handle reset logic

end else begin

// Instruction decoding and execution

if (opcode == OPCODE\_R\_TYPE) begin

case (funct3)

FUNCT3\_XOR: begin

if (funct7 == FUNCT7\_XOR) begin

// Perform XOR operation

registers[rd] <= registers[rs1] ^ registers[rs2];

end

end

// Other R-type instructions can be added here

endcase

end

end

end

// Instruction fetch, decode, and other stages should be added here

endmodule