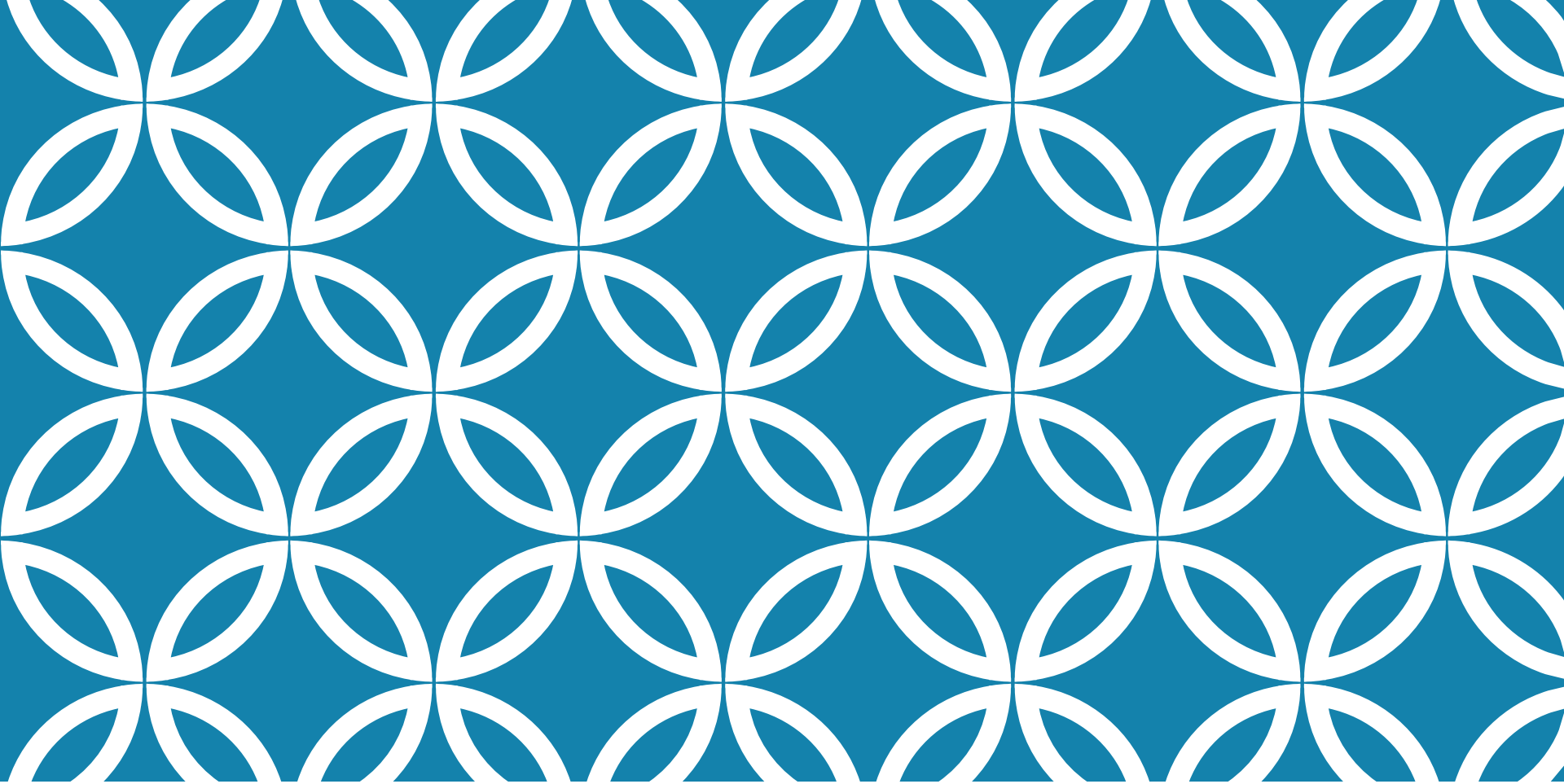
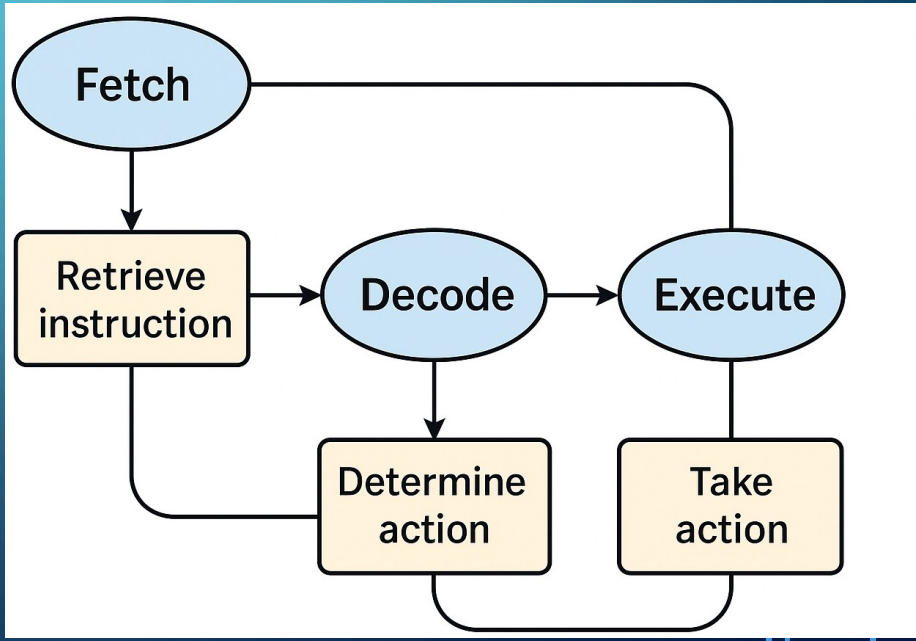


16-BIT RISC PROCESSOR DESIGN

- Design • Simulation
- Debug • Synthesis

OVERVIEW

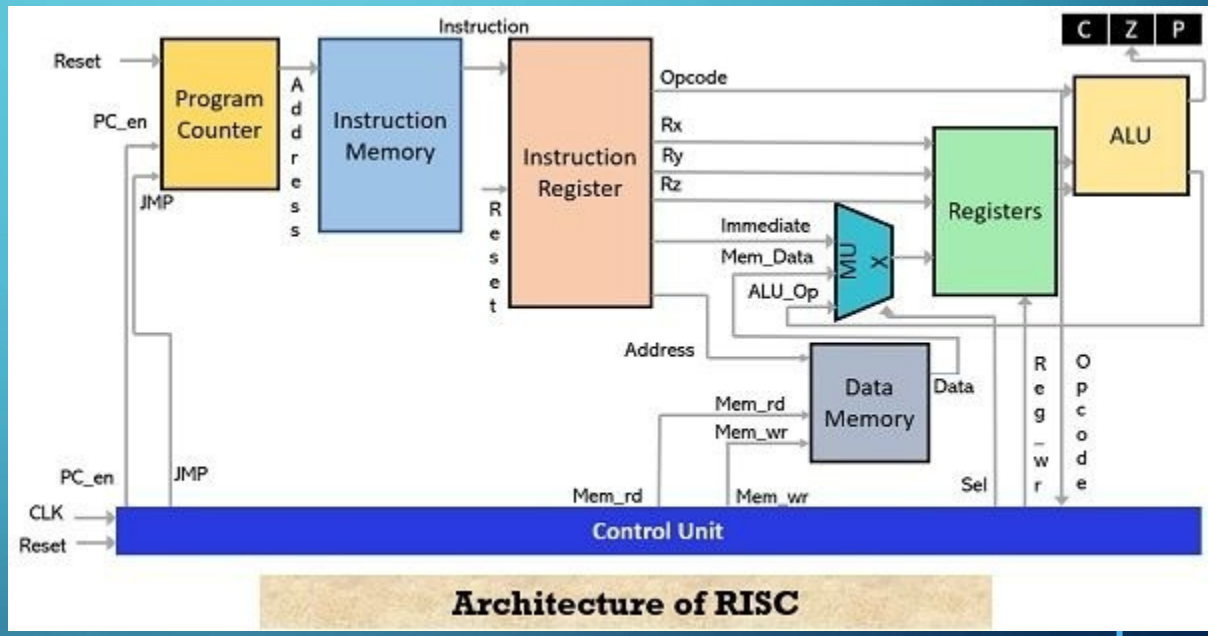
- Objective: To Build a simple 16-bit RISC processor
- Tools: Verilog, Vivado
- Pipeline: Fetch, Decode, Execute, Write-Back



ARCHITECTURE

Components:

- ALU, ALU Control Unit
- Register File
- Instruction Memory
- Data Memory
- Control Unit
- Data Path Unit
- Pipeline Stages

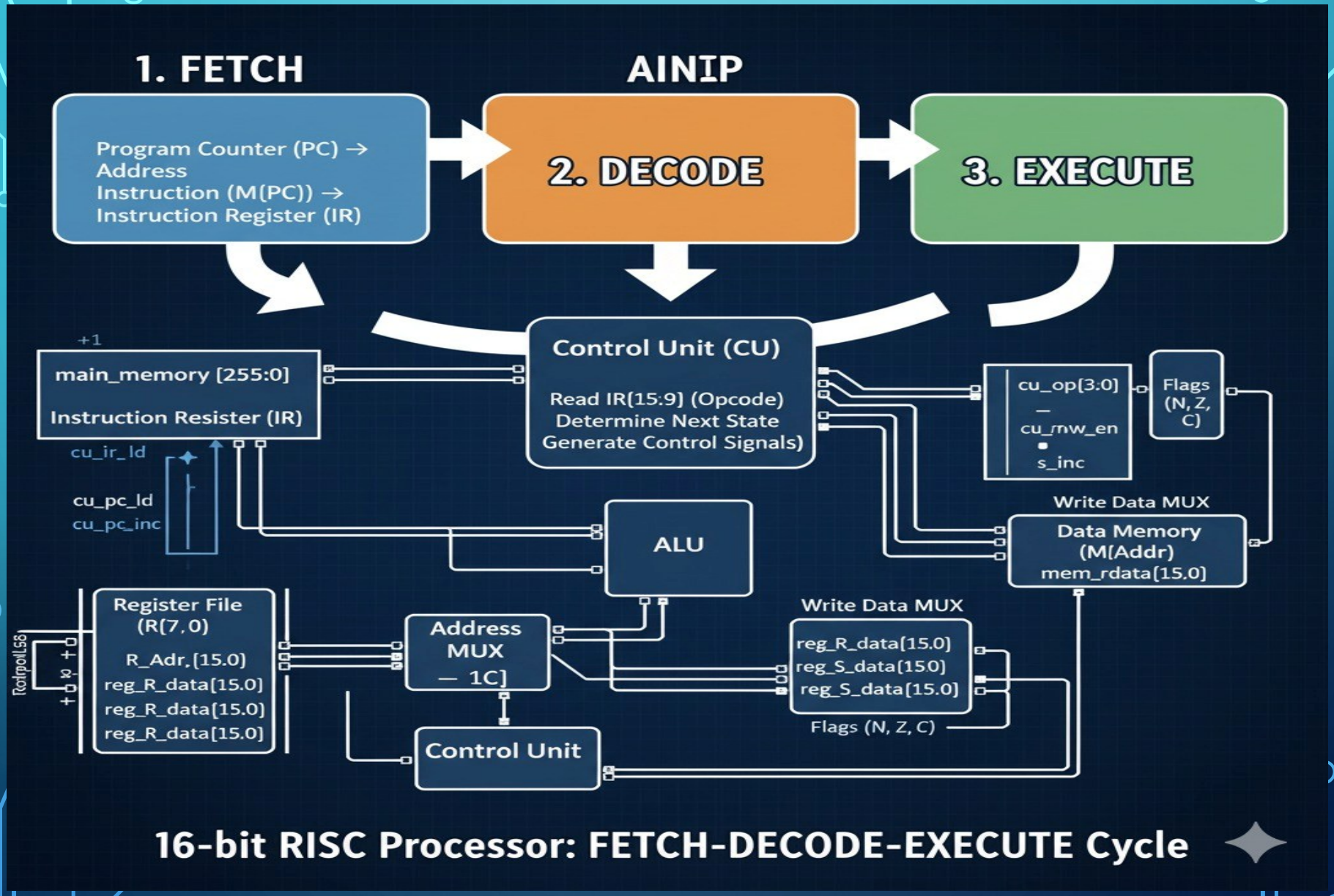


Architecture of RISC

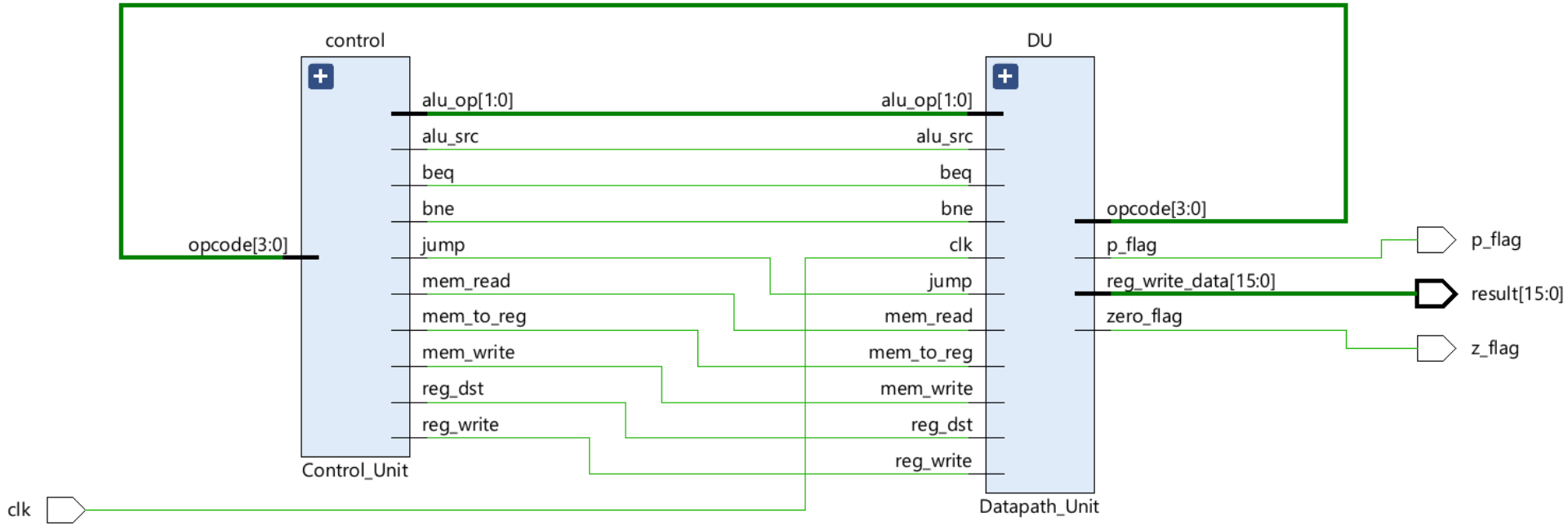
- 16-bit RISC ISA, single-cycle datapath
- Top-level: Risc_16_bit instantiates Control_Unit + Datapath_Unit
- Memory init via $readmemb; simulation in Vivado

PIPELINE STAGES

- 1. Fetch – Instruction read
- 2. Decode – Register file read
- 3. Execute – ALU operations
- 4. Memory Access – Accesses data memory
- 5. Write-Back – Store results


1. FETCH
AINIP
Program Counter (PC) →
Address
Instruction (M(PC)) →
Instruction Register (IR)
2. DECODE
3. EXECUTE
+1
main_memory [255:0]
Instruction Resister (IR)
cu_ir_ld
cu_pc_ld
cu_pc_inc
Control Unit (CU)
Read IR(15:9) (Opcode)
Determine Next State
Generate Control Signals)
cu_op[3:0]
cu_rnw_en
s_inc
Flags
(N, Z,
C)
Write Data MUX
Data Memory
(M(Addr)
mem_rdata[15.0]
ALU
Register File
(R(7,0)
R_Adr,[15.0)
reg_R_data[15.0]
reg_R_data[15.0]
reg_R_data[15.0]
Address
MUX
— 1C]
Write Data MUX
reg_R_data[15.0]
reg_S_data[15.0]
reg_S_data[15.0]
Flags (N, Z, C)
Control Unit
16-bit RISC Processor: FETCH-DECODE-EXECUTE Cycle

SCHEMATIC


control
DU
alu_op[1:0]
alu_src
beq
bne
jump
mem_read
mem_to_reg
mem_write
reg_dst
reg_write
opcode[3:0]
Control_Unit
alu_op[1:0]
alu_src
beq
bne
clk
jump
mem_read
mem_to_reg
mem_write
reg_dst
reg_write
opcode[3:0]
p_flag
reg_write_data[15:0]
zero_flag
Datapath_Unit
p_flag
result[15:0]
z_flag
clk

INSTRUCTION SET

Operations:

- ADD, SUB, AND, OR, NOT
- Right Shift, Left Shift
- LOAD WORD, STORE WORD
- BRANCH instructions

The instruction set of the RISC processor:

A. Memory Access Instructions

1. Load word LW ws, offset(rs1) ws:=Mem16[rs1 + offset]
2. Store Word: SW rs2, offset(rs1) Mem16[rs1 + offset]=rs2

B. Data Processing Instructions

1. Add: ADD ws, rs1, rs2 ws:=rs1 + rs2
2. Subtract: SUB ws, rs1, rs2 ws:=rs1 – rs2
3. Invert (1's complement): INV ws, rs1 ws:=!rs1
4. Logical Shift Left: LSL ws, rs1, rs2 ws:=rs1 << rs2
5. Logical Shift Right: LSR ws, rs1, rs2 ws:=rs1 >> rs2
6. Bitwise AND: AND ws, rs1, rs2 ws:=rs1 • rs2
7. Bitwise OR: OR ws, rs1, rs2 ws:=rs1 | rs2
8. Set on Less Than:

SLT ws, rs1, rs2 ws:=1 if rs1 < rs2; ws:=0 if rs1 ≥ rs2

C. Control Flow Instructions

1. Branch on Equal: BEQ rs1, rs2, offset

   Branch to (PC + 2 + (offset << 1)) when rs1 = rs2

2. Branch on Not Equal: BNE rs1, rs2, offset

   Branch to (PC + 2 + (offset << 1)) when rs1 != rs2

3. Jump: JMP offset Jump to {PC [15:13], (offset << 1)}

INSTRUCTION MEMORY

The Instruction Memory stores the program that the processor executes. It outputs the 16-bit instruction corresponding to the current **Program Counter (PC)** value.

📌 **Key Features**

- **16-bit wide ROM**
- Stores **up to 16 instructions** (memory[0] to memory[15])
- **Program loaded from external file** (test.prog)
- **Combinational access** → instruction available immediately when PC changes
- **Addressing uses PC bits [4:1]** to index instruction memory

- Implements the Fetch stage of the processor pipeline
- Allows easy program changes through the .prog file
- Provides a clean interface for the Control Unit and Datapath to decode and execute instructions
- Helps understand instruction flow inside a simple RISC CPU

INSTRUCTIONS USED (TEST.PROG)

- 0000_000_001_000000 // LW R0 <-Mem(R0 + 0)
- 0000_000_001_000001 // LW R1 <-Mem(R0 + 1)
- 0010_000_001_010_000 // ADD R2 <-R0 + R1
- 0110_000_001_010_000 // logical shift right R2 <-R0>>R1
- 0000_000_011_000010 // LW R3 <-Mem(R0 + 2)
- 0100_011_000_100_000 // NOT R4 <-~R3
- 1000_000_001_110_000 // OR R6 <-R0 | R1
- 1001_000_001_111_100 // Selecton less thanR7 <-(R0 < R1)

DATAPATH

Flow:

- PC → Instruction Memory
- Register Read
- ALU Execute
- Memory Access
- Write-Back to Register File

DATA MEMORY

- The Data Memory block stores and retrieves 16-bit data during program execution.
  It supports both **read** and **write** operations and is used by load/store instructions in the RISC processor.

Key Features

- **8 × 16-bit memory array** (memory[0] to memory[7])
- **Synchronous write** on the rising edge of clock
- **Combinational read** when mem_read = 1
- **Address mapped using lower 3 bits** (mem_access_addr[2:0])
- **Memory contents initialized from external file** (test.data)
- **Live monitoring of memory values** using $fmonitor for debug/logging

CONTROL UNIT

Generates control signals:

- ALU Op, RegWrite, MemWrite
- MemRead, MemToReg

Processor Control Unit Design:

	Control signals								
Instruction	Reg Dst	ALUSrc	Memto Reg	Reg Write	MemRead	Mem Write	Branch	ALUOp	Jump
Data-processing	1	0	0	1	0	0	0	00	0
LW	0	1	1	1	1	0	0	10	0
SW	0	1	0	0	0	1	0	10	0
BEQ,BNE	0	0	0	0	0	0	1	01	0
J	0	0	0	0	0	0	0	00	1

ALU

```
`timescale 1ns / 1ps
module ALU(
 input  [15:0] a,b,  //src1 AND  //src2
 input  [2:0] alu_control, //function sel
 output reg [15:0] result,  //result
 output  zero, Negative, Parity   );
always @(*)
begin
 case(alu_control)
 3'b000: result = a + b; // add
 3'b001: result = a - b; // sub
 3'b010: result = ~a;
 3'b011: result = a<<b;
 3'b100: result = a>>b;
 3'b101: result = a & b; // and
 3'b110: result = a | b; // or
 3'b111: begin if (a<b) result = 16'd1;
   else result = 16'd0;
   end
 default:result = a + b; // add
 endcase
end
assign zero = (result==16'd0) ? 1'b1: 1'b0;
assign Negative = result[15];  // MSB indicates sign
assign Parity   = ^(result);   // XOR of all bits → odd parity = 1, even = 0
 endmodule
```

ALU CONTROL UNIT

ALUOp	Opcode(hex)	ALUcnt	ALU Operation	Instruction
10	xxxx	000	ADD	LW,SW
01	xxxx	001	SUB	BEQ,BNE
00	0002	000	ADD	D-type: ADD
00	0003	001	SUB	D-type: SUB
00	0004	010	INVERT	D-type: INVERT
00	0005	011	LSL	D-type: LSL
00	0006	100	LSR	D-type: LSR
00	0007	101	AND	D-type: AND
00	0008	110	OR	D-type: OR
00	0009	111	SLT	D-type: SLT

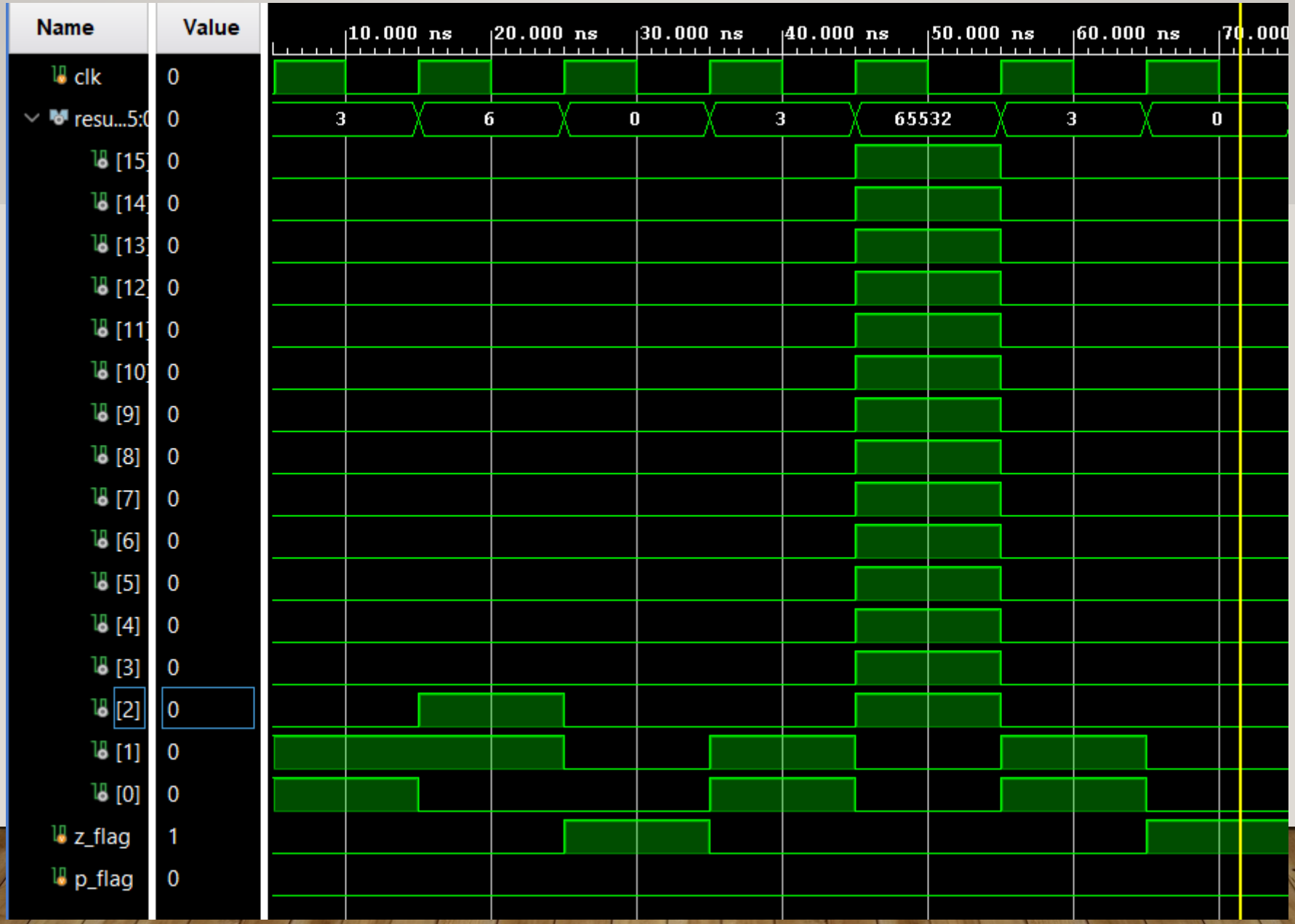
ALU_CONTROL VERILOG CODE

```
`TIMESCALE 1NS / 1PS
MODULE ALU_CONTROL( ALU_CNT, ALUOP, OPCODE);
 OUTPUT REG[2:0] ALU_CNT;
 INPUT [1:0] ALUOP;
 INPUT [3:0] OPCODE;
 WIRE [5:0] ALUCONTROLIN;
 ASSIGN ALUCONTROLIN = {ALUOP,OPCODE};
 ALWAYS @(ALUCONTROLIN)
 CASEX (ALUCONTROLIN)
   6'B10XXXX: ALU_CNT=3'B000;
   6'B01XXXX: ALU_CNT=3'B001;
   6'B000010: ALU_CNT=3'B000;
   6'B000011: ALU_CNT=3'B001;
   6'B000100: ALU_CNT=3'B010;
   6'B000101: ALU_CNT=3'B011;
   6'B000110: ALU_CNT=3'B100;
   6'B000111: ALU_CNT=3'B101;
   6'B001000: ALU_CNT=3'B110;
   6'B001001: ALU_CNT=3'B111;
  DEFAULT: ALU_CNT=3'B000;
  ENDCASE
ENDMODULE
```

TESTBENCH

```
`timescale 1ns / 1ps
`include "Parameter.v"
module tb;
reg clk ;
wire [15:0] result;
 initial
  begin
   clk <=0;
   `simulation_time;
   $finish;
  end
 always
  begin
    #5 clk = ~clk;
  end
  Risc_16_bit uut(.clk(clk), .result(result),.z_flag(z_flag),.p_flag( p_flag));
endmodule
```

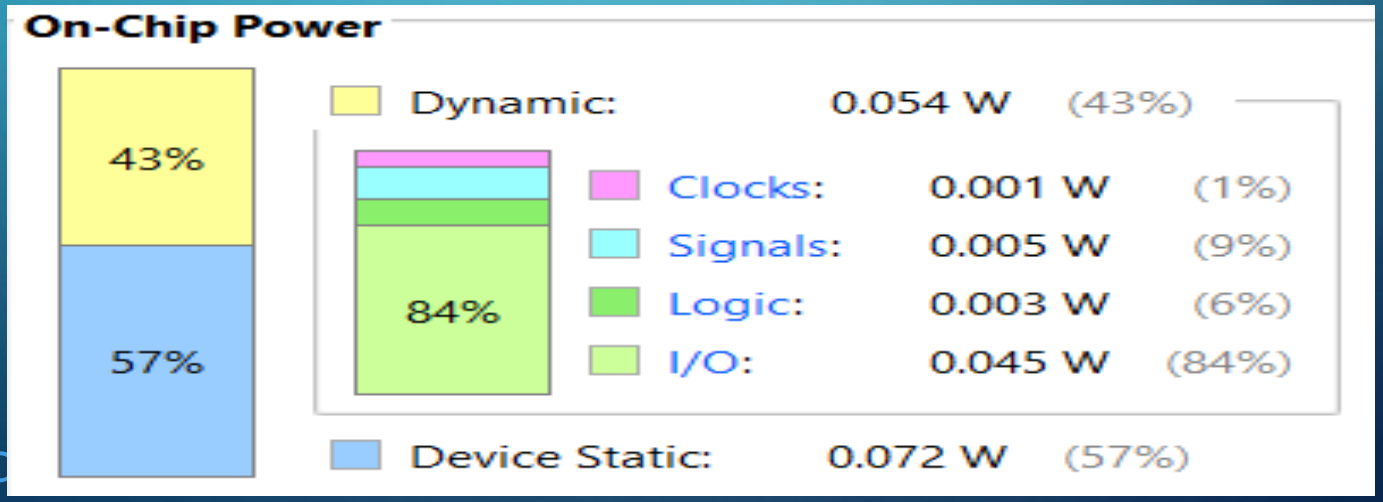
VIVADO WAVEFORM



SYNTHESIS AND IMPLEMENTATION

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0.176 ns	Worst Hold Slack (WHS):	0.376 ns	Worst Pulse Width Slack (WPWS):	3.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	244	Total Number of Endpoints:	244	Total Number of Endpoints:	69



HAZARDS HANDLING

- Structural hazards avoided
- No data hazards due to simplified design
- Control hazards handled by PC update logic

OUTCOME

- Working 16-bit processor implemented
- Reinforced understanding of CPU architecture
- Hands-on experience with RTL, simulation & synthesis
- Foundation for advanced VLSI/CPU projects

REFERENCES

- Source: isroset.org
  https://share.google/KBrFRzuKuID7NFgjf
- M. Morris Mano

THANKYOU

SUBMITTED TO:

Dr Nitesh Kashyap

Assistant Professor (Grade-I)

ECE, NIT Jalandhar

SUBMITTED BY:

Sathwik 25204132

Aditya 25904138

Vishnu 25904150

M.Tech VLSI Design

1st Semester