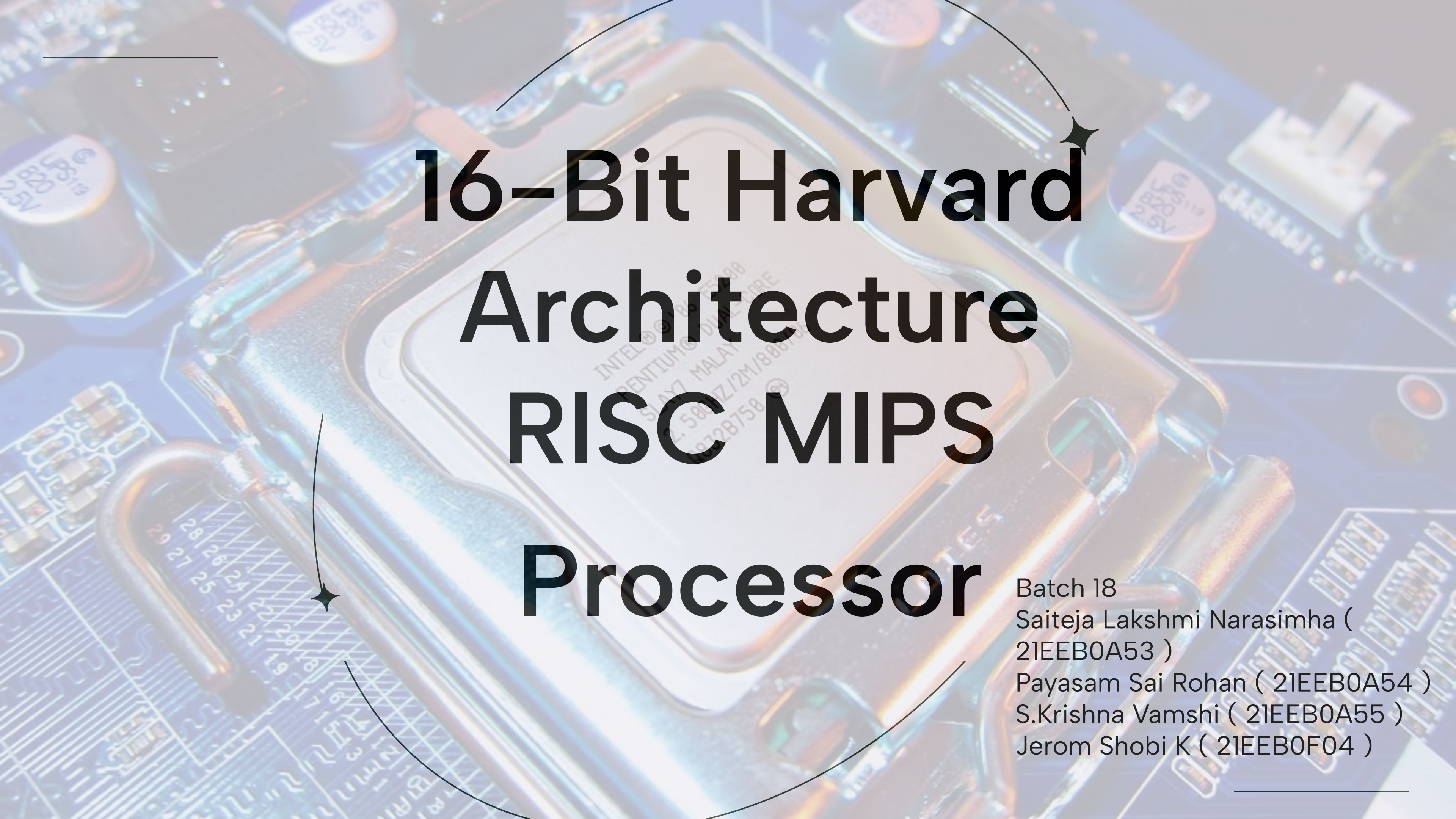


# 16-Bit Harvard Architecture RISC MIPS Processor

Batch 18
Saiteja Lakshmi Narasimha ( 21EEB0A53 )
Payasam Sai Rohan ( 21EEB0A54 )
S.Krishna Vamshi ( 21EEB0A55 )
Jerom Shobi K ( 21EEB0F04 )

# Table of contents

01 Introduction

02 Microarchitecture Design

03 RTL Design and Implementation

04 Synthesis and Results

05 Conclusion

# Abstract

This project implements a 16-bit RISC MIPS processor with **Harvard architecture,** utilizing a **non-pipelined design** to prioritize simplicity, cost, and power consumption. The compact instruction set, consisting of **ADD, SUB, OR, AND, JZ, LOAD, and STORE** operations, enables efficient arithmetic, logical, data transfer, and conditional operations.

# 01 Introduction

## RISC Processor Features

- **Simple instruction set**: In a RISC machine, the instruction set contains simple, basic instructions, from which more complex instructions can be composed.

- **Same length instruction**: Each instruction is of the same length, so that it may be **fetched in a single operation**.

- **Very few addressing modes and formats**: This is unlike CISC processors, where the number of addressing modes are very high

- **Load and Store architecture**: The RISC architecture is primarily a Load and Store architecture, implying that all the memory accesses take place using Load or Store type operations.

# 02 Microarchitectural Features

- Interfaced to data memory of size **256 x 16 bits**
- **8 bit** address bus
- 16 registers, each of size 2 bytes ( RA, RB…RP)
- ALU capable of **16 bit arithmetic operations**
- Instruction memory of size **256 x 16 bits** ( ranging from 0 to 255 )
- Opcode of **constant length** ( 16 bit for all instructions )
- A **separate adder block** was used to get the next value for program counter
- Jump instructions are added in their own 8 bit jump adder
- Zero flag (ZF) given as output with result from ALU which is used in JUMP instruction

# Instruction Format

| | | |
|---|---|---|
| 4 bits<br>Opcode | 4 bits<br>XXXX | 8 bits<br>Displacement |

**Jump Instruction**

- JZ : 0100

Displacement ranges from 0 to 255 bytes

| | | | |
|---|---|---|---|
| 4 bits<br>Opcode | 4 bits<br>Destination | 4 bits<br>Source 2 | 4 bits<br>Source 1 |

**Arithmetic Instructions**

- ADD : 0000
- SUB : 0001
- AND : 0010
- OR : 0011

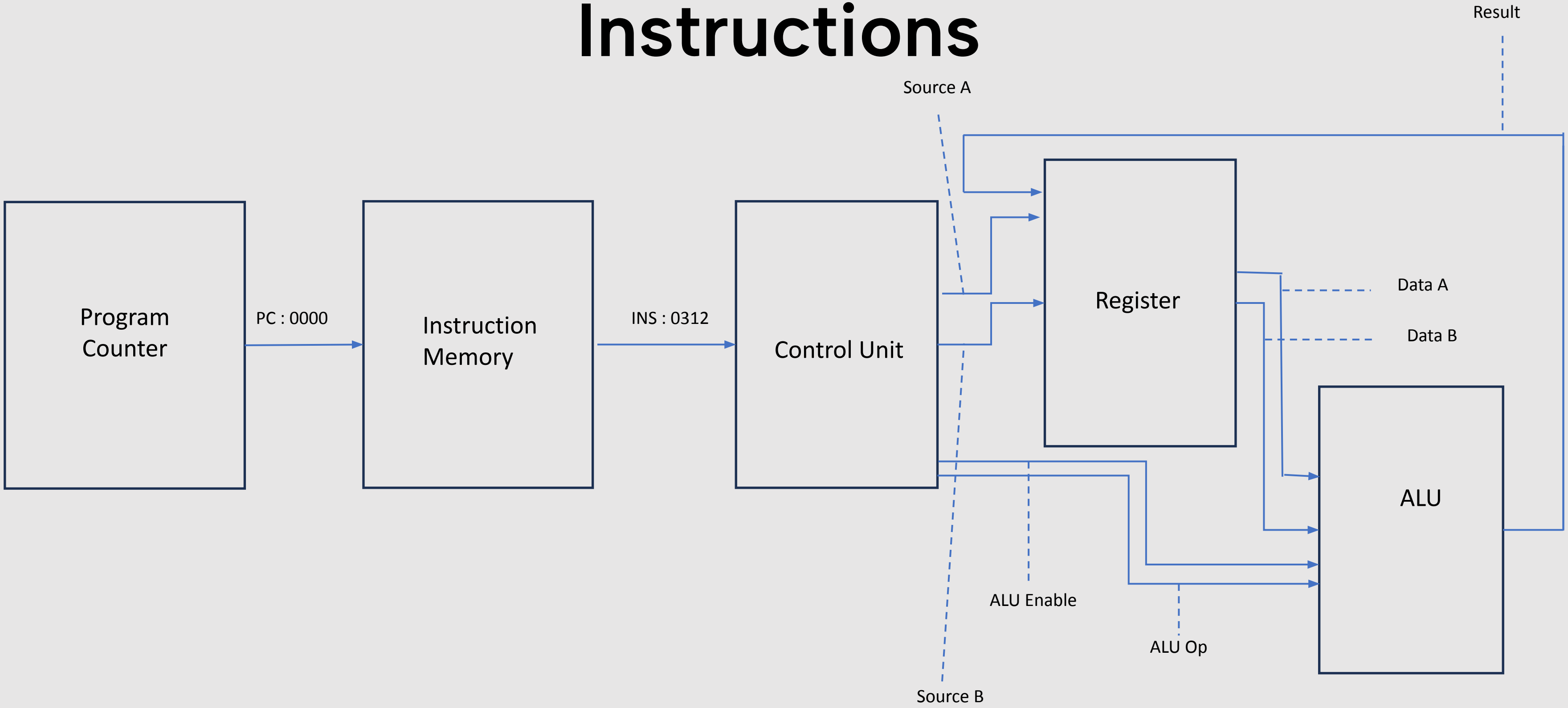
The addresses for destination, source and source  2 range from RA to RP ( 16-bit registers )

| | | | |
|---|---|---|---|
| 4 bits<br>Opcode | 4 bits<br>Data Read Register Address | 4 bits<br>Data Store Register Address | 4 bits<br>Data Memory Address |

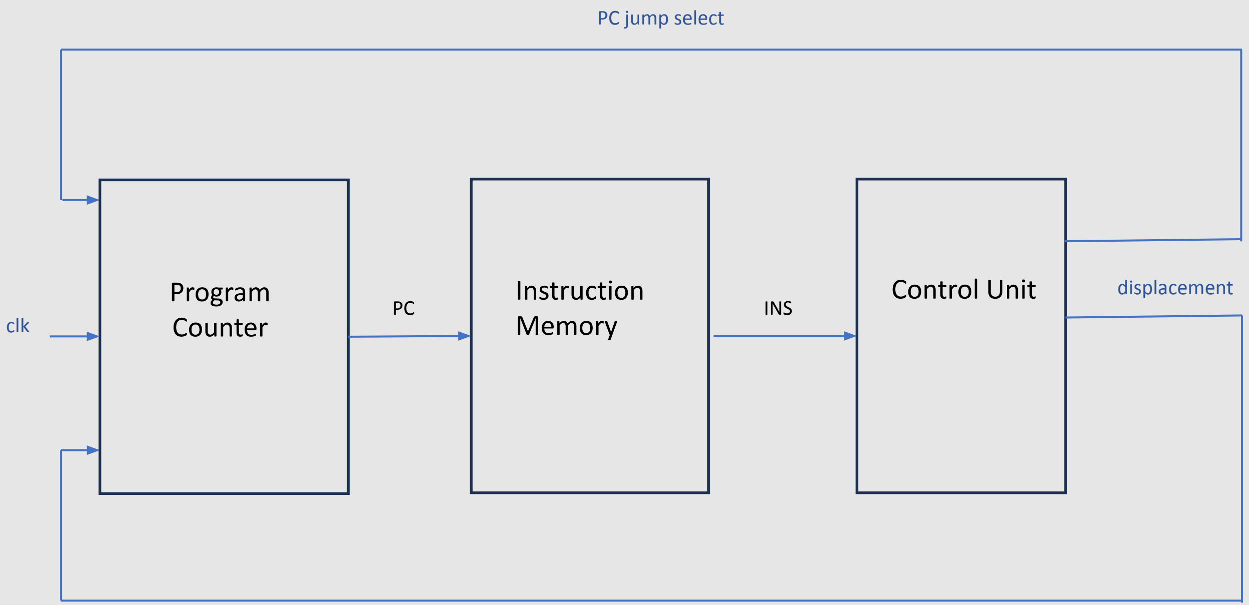
**LOAD / STORE Instructions**

- LOAD  :0101
- STORE : 0110

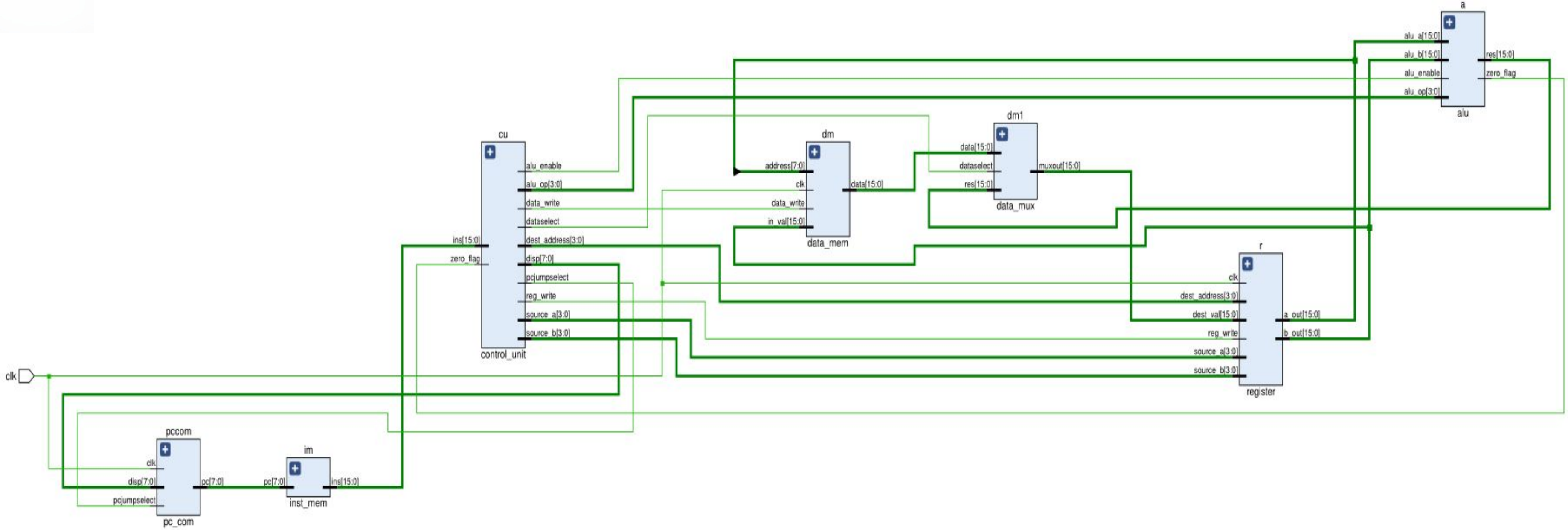
# Data Flow Path for Arithmetic Instructions



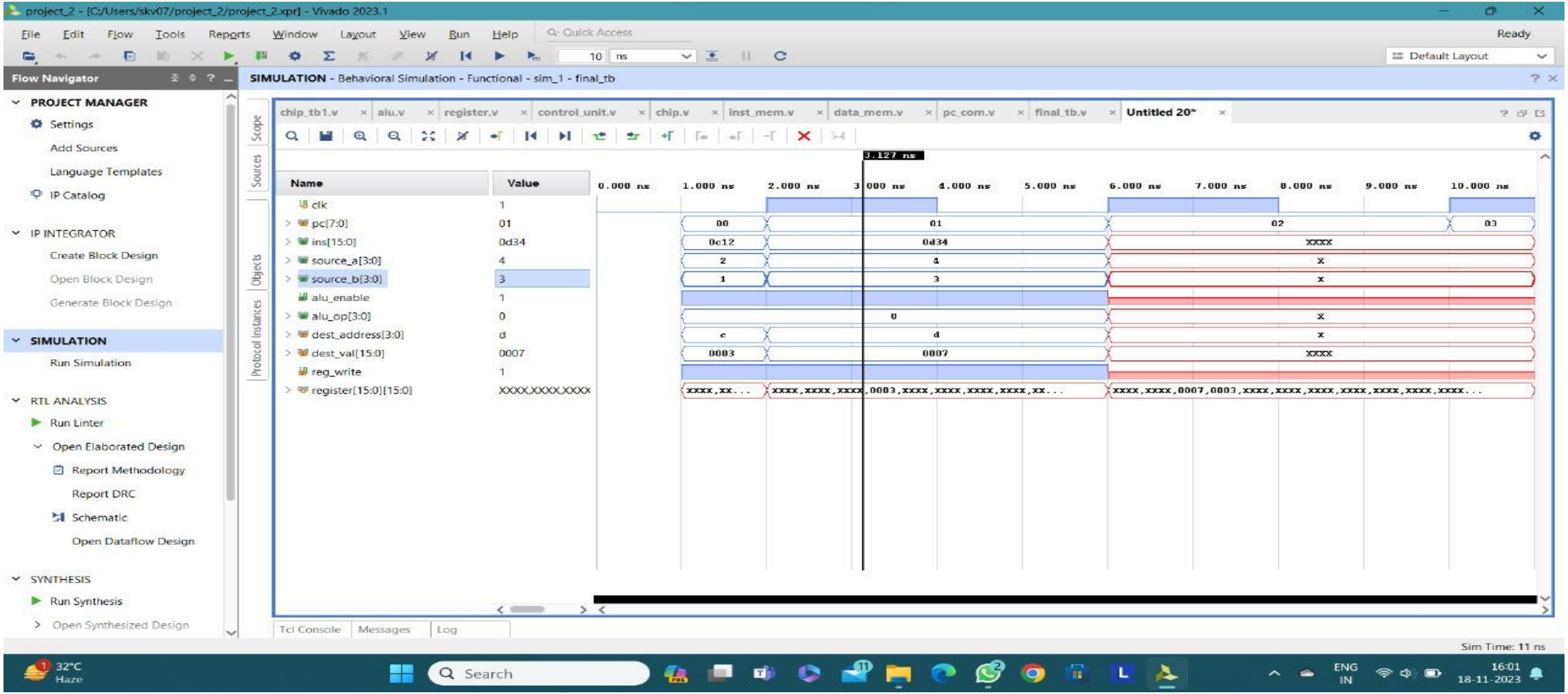
# Data Flow Path for Jump Instruction



# 03 RTL Schematic of Processor



# 04 Result



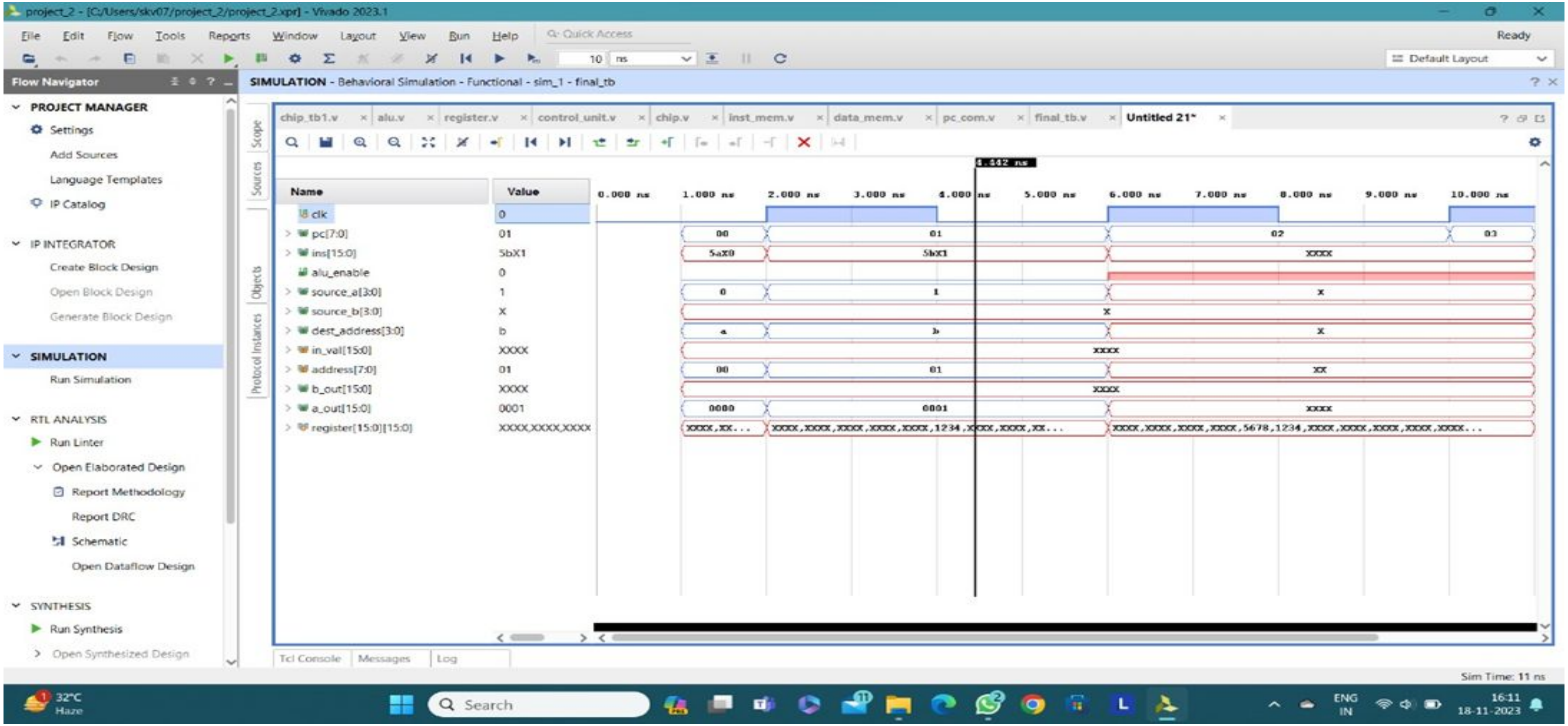
project_2 - [C:/Users/skv07/project_2/project_2.xpr] - Vivado 2023.1

SIMULATION - Behavioral Simulation - Functional - sim_1 - final_tb

| Name | Value |
|---|---|
| clk | 1 |
| pc[7:0] | 01 |
| ins[15:0] | 0d34 |
| source_a[3:0] | 4 |
| source_b[3:0] | 3 |
| alu_enable | 1 |
| alu_op[3:0] | 0 |
| dest_address[3:0] | d |
| dest_val[15:0] | 0007 |
| reg_write | 1 |
| register[15:0][15:0] | XXXX,XXXX,XXXX |

Sim Time: 11 ns

*Waveform for ADD Instruction*



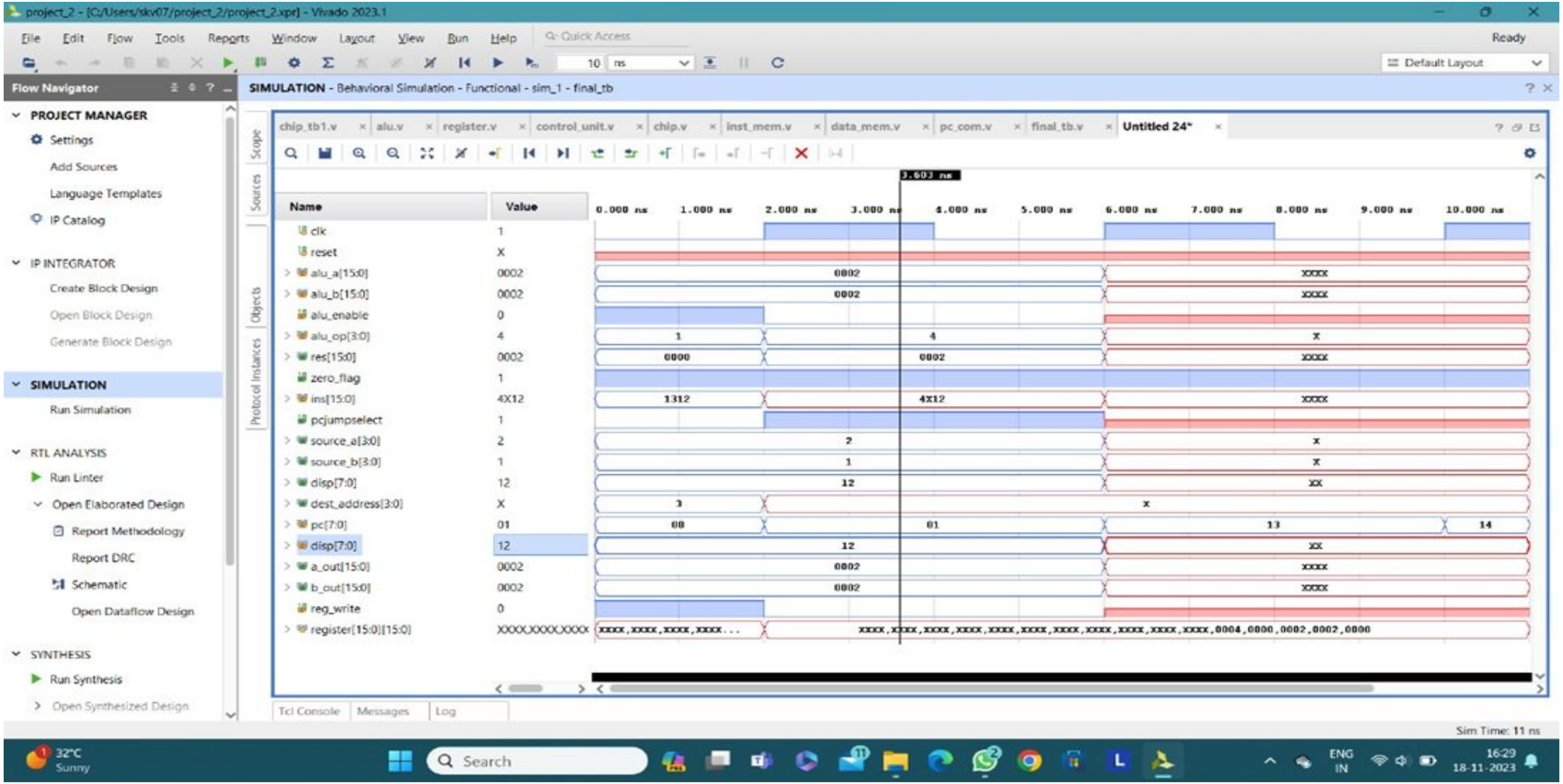
project_2 - [C:/Users/skv07/project_2/project_2.xpr] - Vivado 2023.1

SIMULATION - Behavioral Simulation - Functional - sim_1 - final_tb

| Name | Value |
|---|---|
| clk | 0 |
| pc[7:0] | 01 |
| ins[15:0] | 5bX1 |
| alu_enable | 0 |
| source_a[3:0] | 1 |
| source_b[3:0] | X |
| dest_address[3:0] | b |
| in_val[15:0] | XXXX |
| address[7:0] | 01 |
| b_out[15:0] | XXXX |
| a_out[15:0] | 0001 |
| register[15:0][15:0] | XXXX,XXXX,XXXX |

Sim Time: 11 ns

*Waveform for LOAD Instruction*



| Name | Value |
|---|---|
| clk | 1 |
| reset | X |
| alu_a[15:0] | 0002 |
| alu_b[15:0] | 0002 |
| alu_enable | 0 |
| alu_op[3:0] | 4 |
| res[15:0] | 0002 |
| zero_flag | 1 |
| ins[15:0] | 4X12 |
| pcjumpselect | 1 |
| source_a[3:0] | 2 |
| source_b[3:0] | 1 |
| disp[7:0] | 12 |
| dest_address[3:0] | X |
| pc[7:0] | 01 |
| disp[7:0] | 12 |
| a_out[15:0] | 0002 |
| b_out[15:0] | 0002 |
| reg_write | 0 |
| register[15:0][15:0] | XXXX,XXXX,XXXX |

*Waveform for JUMP Instruction*

# 05 Conclusion

- Simplified Design for Cost-Effectiveness

Compact Instruction Set for Efficient Execution

- Applications in Embedded Systems