# Design and Implementation of a Pipelined RISC-V Processor in SystemVerilog

**Supervisor**
Prof. Joycee Mekie
Kailash Prasad

**Ruchit Chudasama**
B.Tech 3rd year
Electrical Engineering
IIT Gandhinagar

**Aryan Gupta**
B.Tech 3rd year
Electrical Engineering
IIT Gandhinagar

**nanoDC Lab**

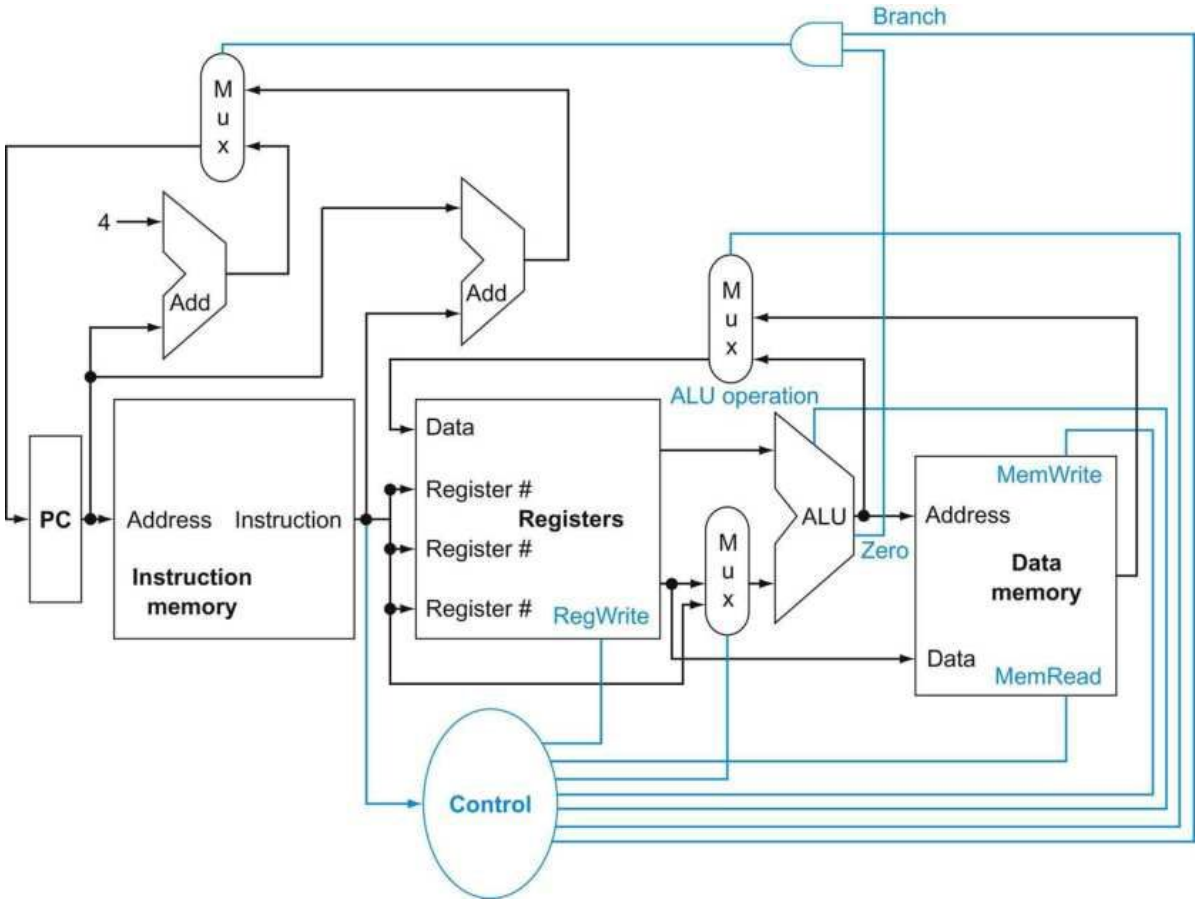
# Outline

1. Motivation
2. Design
3. Simulations
4. Challenges faced
5. The way ahead

# WHY RISC-V?

- Open Source
- Reduced Complexity
- Modular Design
- Scalable
- Community-Driven

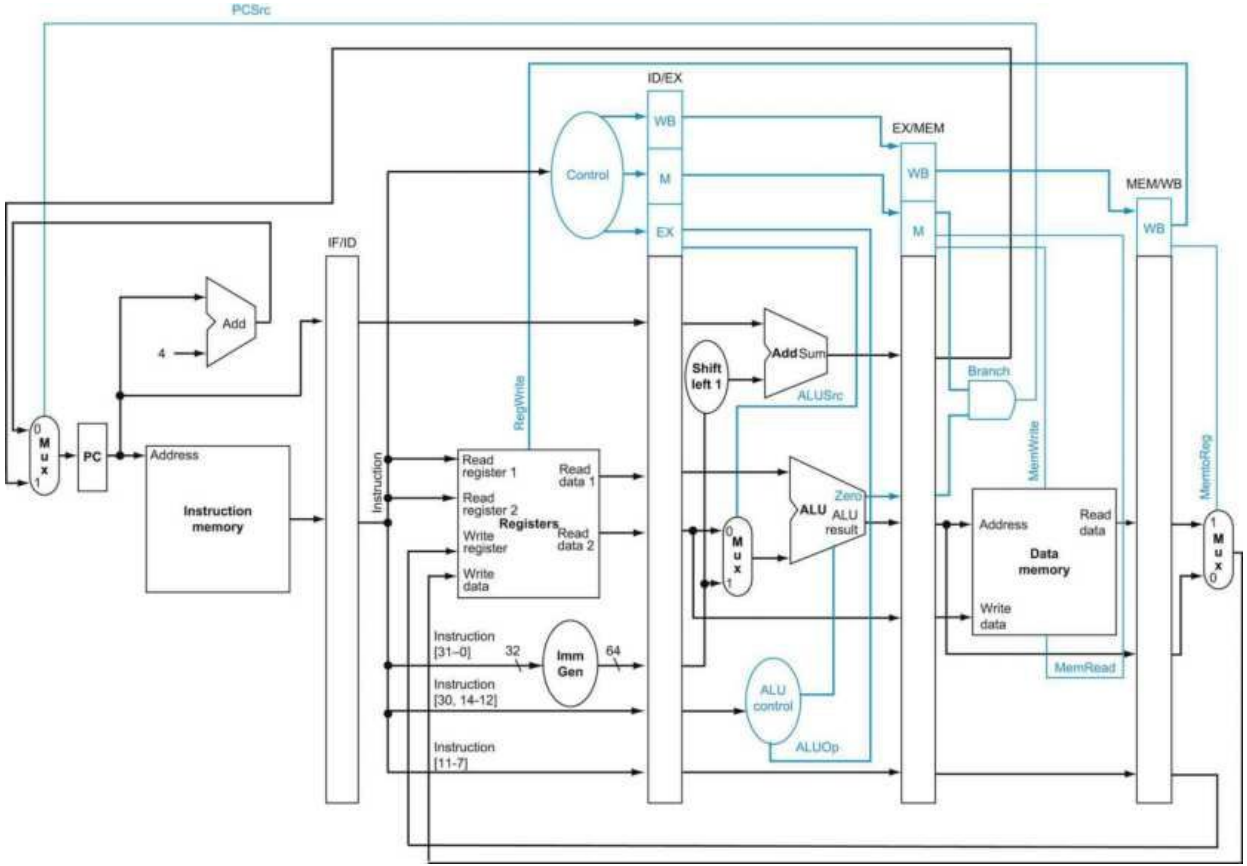
# RISC-V Datapath+Control (without pipelining)



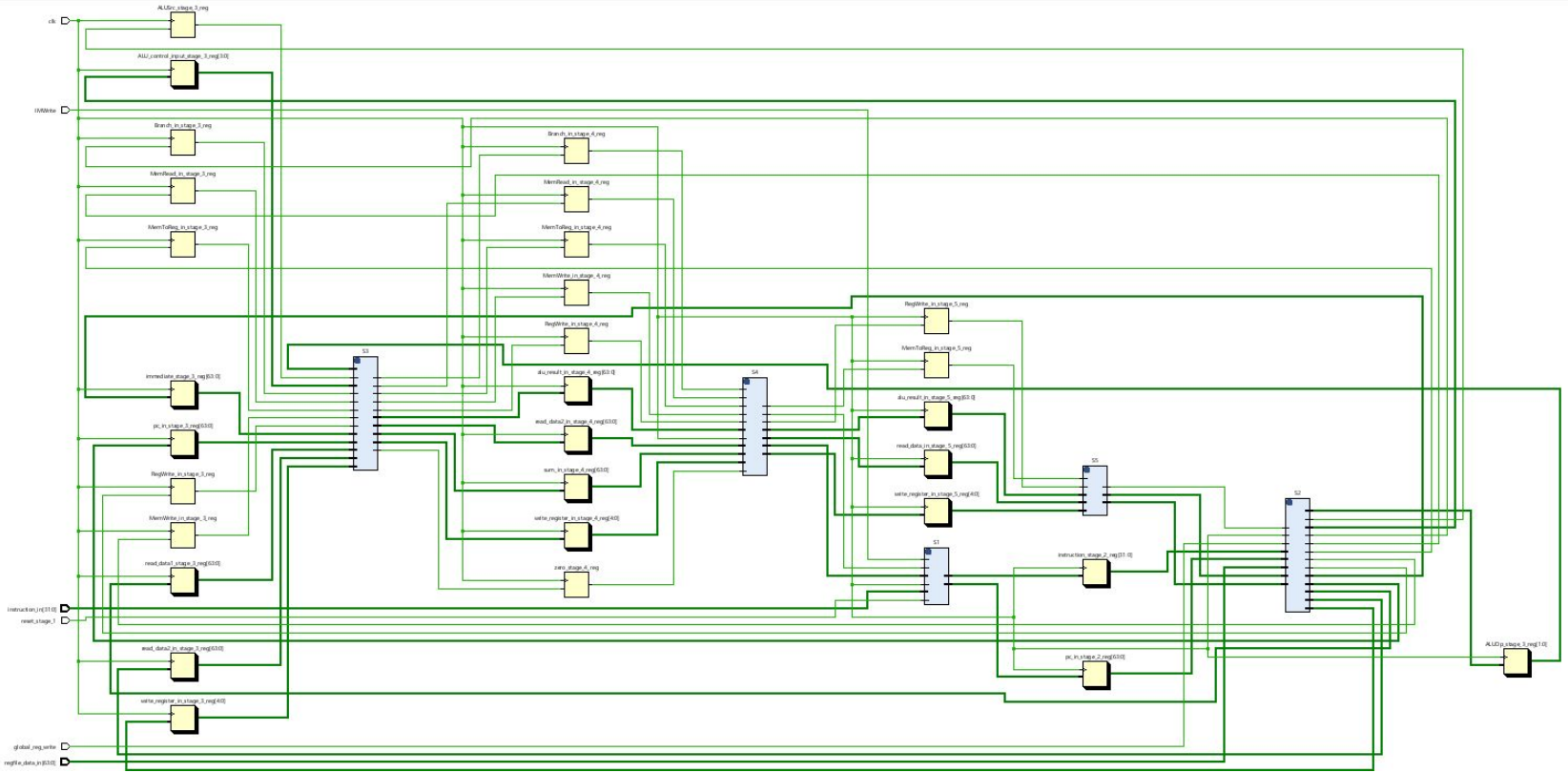
# WHY PIPELINED RISC-V?

- Increased throughput
- Lower power consumption
- Better utilization of resources

# RISC-V Datapath+Control (with pipelining)



# RISC-V Datapath+Control (with pipelining)



# Design Specifications and Assumptions

- Instruction size: 32-bit
- 64-bit processor
- 32 registers
- Size of Instruction Memory = 128*32 bits
- Size of Data Memory = 128*64 bits

- PC is updated on posedge of clk
- Instruction memory is read on negedge of clk
- All the pipeline registers are updated on posedge clk
- Read from regfile is asynchronous
- Write to regfile happens on negedge of clk
- Data Memory RW happens on negedge of clk
- Little Endian

# Instructions

We will be examining an implementation that includes a subset of RISC-V instruction set:

R-type instructions: *add*, *sub*, *and*, *or*

I-type instructions: load doubleword (*ld*)

S-type instructions: store doubleword (*sd*)

B-type instruction: branch if equal (*beq*)

| Name (Bit position) | Fields 31:25 | 24:20 | 19:15 | 14:12 | 11:7 | 6:0 |
|---|---|---|---|---|---|---|
| (a) R-type | funct7 | rs2 | rs1 | funct3 | rd | opcode |
| (b) I-type | immediate[11:0] | | rs1 | funct3 | rd | opcode |
| (c) S-type | immed[11:5] | rs2 | rs1 | funct3 | immed[4:0] | opcode |
| (d) SB-type | immed[12,10:5] | rs2 | rs1 | funct3 | immed[4:1,11] | opcode |

# Explaining the ISA

| Instruction | Execution/address calculation stage control lines | | Memory access stage control lines | | | Write-back stage control lines | |
|---|---|---|---|---|---|---|---|
| | ALUOp | ALUSrc | Branch | Mem-Read | Mem-Write | Reg-Write | Memto-Reg |
| R-format | 10 | 0 | 0 | 0 | 0 | 1 | 0 |
| ld | 00 | 1 | 0 | 1 | 0 | 1 | 1 |
| sd | 00 | 1 | 0 | 0 | 1 | 0 | X |
| beq | 01 | 0 | 1 | 0 | 0 | 0 | X |

| Instruction | ALUOp | operation | Funct7 field | Funct3 field | Desired ALU action | ALU control input |
|---|---|---|---|---|---|---|
| ld | 00 | load doubleword | XXXXXXX | XXX | add | 0010 |
| sd | 00 | store doubleword | XXXXXXX | XXX | add | 0010 |
| beq | 01 | branch if equal | XXXXXXX | XXX | subtract | 0110 |
| R-type | 10 | add | 0000000 | 000 | add | 0010 |
| R-type | 10 | sub | 0100000 | 000 | subtract | 0110 |
| R-type | 10 | and | 0000000 | 111 | AND | 0000 |
| R-type | 10 | or | 0000000 | 110 | OR | 0001 |

| Signal name | Effect when deasserted | Effect when asserted |
|---|---|---|
| RegWrite | None. | The register on the Write register input is written with the value on the Write data input. |
| ALUSrc | The second ALU operand comes from the second register file output (Read data 2). | The second ALU operand is the sign-extended, 12 bits of the instruction. |
| PCSrc | The PC is replaced by the output of the adder that computes the value of PC + 4. | The PC is replaced by the output of the adder that computes the branch target. |
| MemRead | None. | Data memory contents designated by the address input are put on the Read data output. |
| MemWrite | None. | Data memory contents designated by the address input are replaced by the value on the Write data input. |
| MemtoReg | The value fed to the register Write data input comes from the ALU. | The value fed to the register Write data input comes from the data memory. |

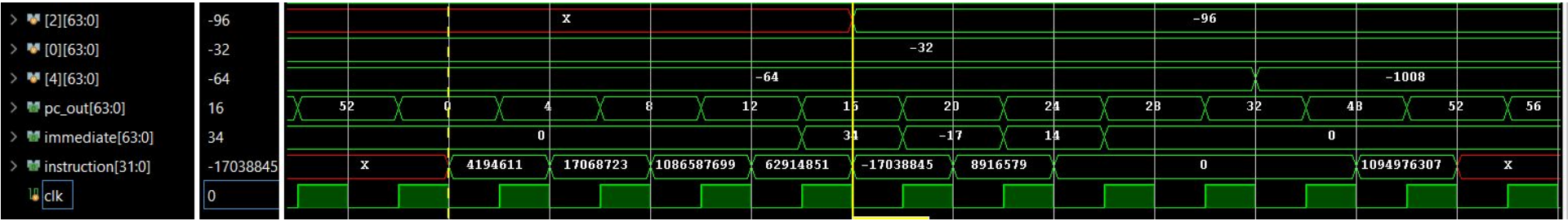
# Initialisation of Instruction Memory and Registers

| |
|---|
| R2 = R0 +R4 |
| R5 = R8 & R16 |
| R6 = R8 - R12 |
| mem[R0+ imm] = R28 |
| R4 = mem[R24 + imm] |
| BEQ R8, R16 |
| NOP |
| NOP |
| NOP |
| NOP |
| R2 = R0 + R8 |
| NOP |
| R6 = R8 - R20 |

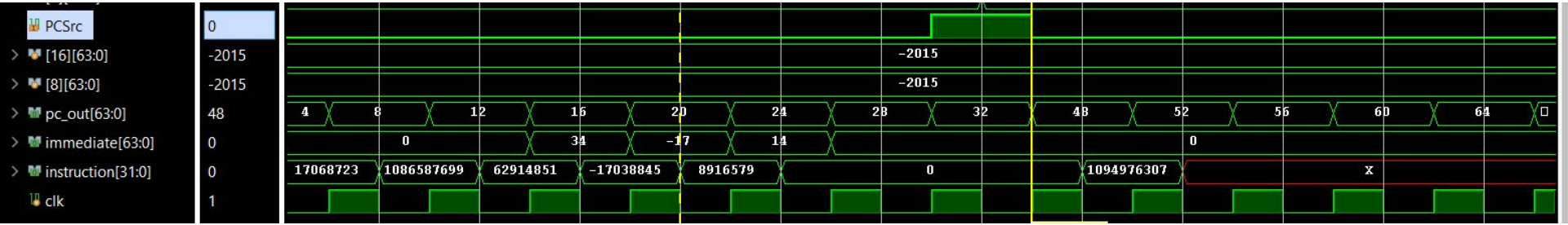
| | |
|---|---|
| R0 | -32 |
| R4 | -64 |
| R8 | -2015 |
| R12 | -4020 |
| R16 | -2015 |
| R20 | 19 |
| R24 | 19 |
| R28 | -1008 |

# Simulation for R-type and B-type instruction

Add Instruction ( R2 = R0 +R4 )

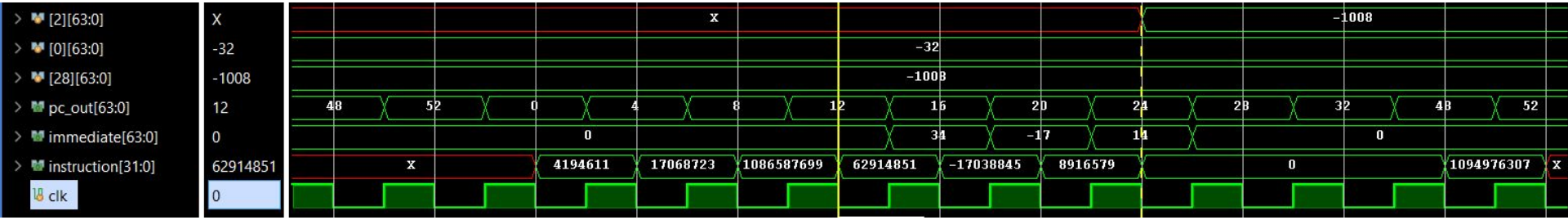



Branch Instruction ( BEQ R8, R16 )

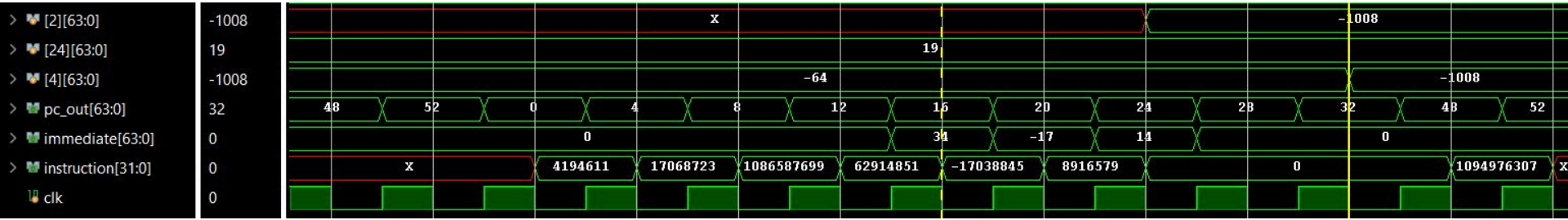


# Simulation for Load-Store Instructions

Store Instruction ( mem[R0+ imm] = R28 )



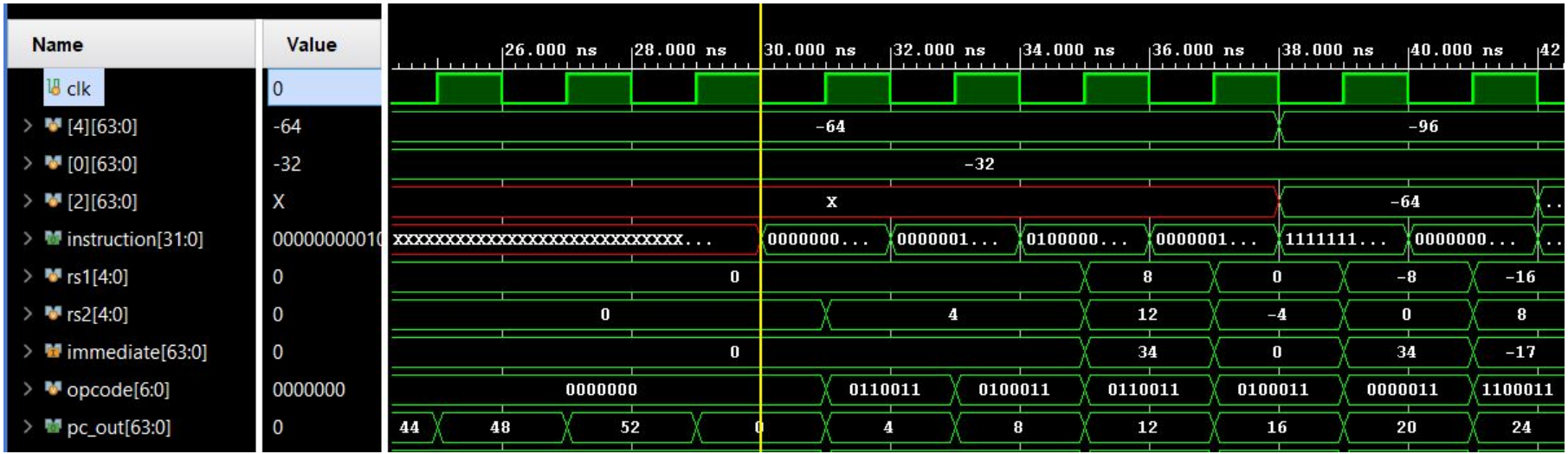

Load Instruction ( R4 = mem[R24 + imm] )



# Simulation RAW pipelining hazard

R4 = R0 + R4

mem[R0+imm] = R4



# Pre-Synthesis with netlist of individual modules

- During our attempts to synthesize the entire RISC-V design using Genus, it caused the optimization of the design to remove essential logic, resulting in NULL output.
- Consequently, we opted to synthesize each module's netlist separately and create a separate module for the pipeline registers in the top module.
- By doing so, we ensured that the top module had no logic.
- Subsequently, we executed pre-synthesis using these individual netlists and the top_processor.sv module, resulting in correct output.

Waveform 1 - SimVision

File Edit View Explore Format Simulation Windows Help

cadence

Search Names: Signal | Search Times: Value | Send To:

TimeA = 0 ps | 108,299,350ps + 0 | Time: 28,336ps : 44,336ps

Baseline = 0
Cursor-Baseline = 0

| Name | Cursor |
|---|---|
| _control_input_stage_2 | 'h 0 |
| Branch_out_stage_3 | 0 |
| Branch_stage_2 | 0 |
| IMWrite | 0 |
| MemRead_out_stage_3 | 0 |
| MemRead_stage_2 | 0 |
| MemToReg_out_stage_3 | 0 |
| MemToReg_out_stage_4 | 0 |
| MemToReg_stage_2 | 0 |
| MemWrite_out_stage_3 | 0 |
| MemWrite_stage_2 | 0 |
| PCSrc_stage_4 | 0 |
| RegWrite_out_stage_3 | 0 |
| RegWrite_out_stage_4 | 0 |
| RegWrite_out_stage_5 | 0 |
| RegWrite_stage_2 | 0 |
| alu_result_out_stage_4 | 'd 0 |
| alu_result_stage_3 | 'd 0 |
| clk | 0 |
| global_reg_write | 0 |
| immediate_stage_2 | 'd 0 |
| instruction_in[31:0] | 'h xxxxxxxx |
| instruction_stage_1 | 'h 00000000 |
| pc_out_stage_1 | 'd 0 |
| pc_out_stage_2 | 'h 0000000 |
| read_data1_stage_2 | 'd 0 |
| read_data2_out_stage_3 | 'd 0 |
| read_data2_stage_2 | 'd 0 |

00ps | 30,000ps | 32,000ps | 34,000ps | 36,000ps | 38,000ps | 40,000ps | 42,000ps | 44

10,000,000 | 20,000,000 | 30,000,000 | 40,000,000 | 50,000,000 | 60,000,000 | 70,000,000 | 80,000,000 | 90,000,000 | 108,299,350ps

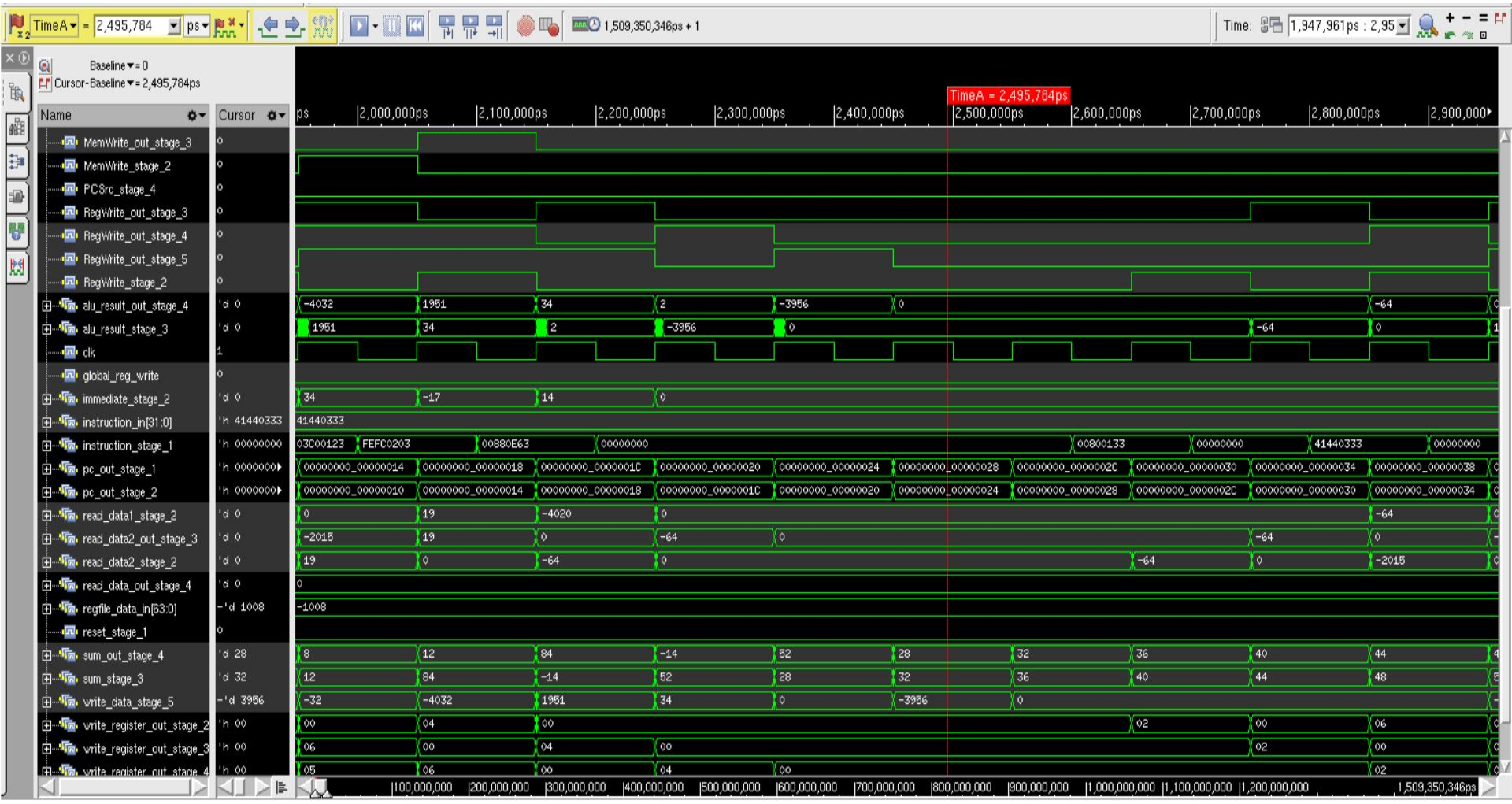
0

1 object selected

nanoDC Lab

# Post-Synthesis with netlist of individual modules

- During the post-synthesis phase, we faced a hold violation error caused by the presence of scan flops.
- To address this issue, we utilized the "set_dont_use" command with the specific name of the scan flop, resulting in the removal of all scan flops in the individual netlists.
- However, this approach led to the emergence of setup violations in our design, ultimately causing it to malfunction.
- Despite attempting to resolve the issue by increasing the clock period to 100 ns, we were unable to eliminate the timing violations.


TimeA = 2,495,784ps
Baseline = 0
Cursor-Baseline = 2,495,784ps
Name
Cursor
MemWrite_out_stage_3
MemWrite_stage_2
PCSrc_stage_4
RegWrite_out_stage_3
RegWrite_out_stage_4
RegWrite_out_stage_5
RegWrite_stage_2
alu_result_out_stage_4
alu_result_stage_3
clk
global_reg_write
immediate_stage_2
instruction_in[31:0]
instruction_stage_1
pc_out_stage_1
pc_out_stage_2
read_data1_stage_2
read_data2_out_stage_3
read_data2_stage_2
read_data_out_stage_4
regfile_data_in[63:0]
reset_stage_1
sum_out_stage_4
sum_stage_3
write_data_stage_5
write_register_out_stage_2
write_register_out_stage_3
write_register_out_stage_4

nanoDC Lab

# References

Computer Organisation and Design: The Hardware/Software interface (RISC-V Edition) By David A. Patterson and John L. Hennessy