SUPERSCALAR RISC-V DESIGN FOR DATA ENCRYPTION

Group - 10
Parth Kulkarni, Jash Shah, Oindrila Chatterjee

TABLE OF CONTENTS

01 PARALLEL COMPUTING

02 ENCRYPTION

03 RISC-V DESIGN

04 WORKING DEMO

Parallel Computing in RISC-V (single core)

→ Task Parallelism – Using virtual threads at OS level to assign CPU cycles efficiently.

Use cases – RTOS for - IoT Devices

→ Instruction-Level Parallelism (ILP) – Using superscalar approaches, which basically involves cloning hardware computing units.

Use cases – High-performance processors and Streaming Multiprocessors

→ Pipeline Parallelism - Divides an instruction into steps that can be executed parallely.

Use cases - Video processing

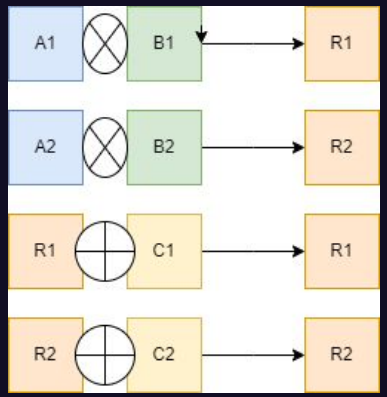
→ Thread-Level Parallelism - Allows single processor to run multiple threads based on a policy.

Use cases - CGMT and SGMT for web browsers

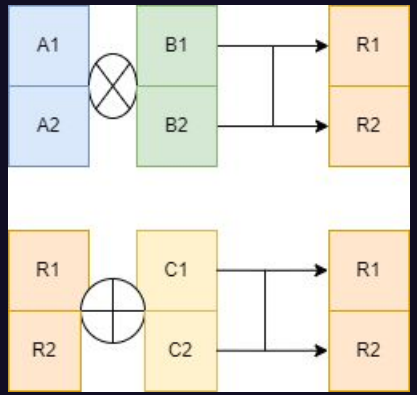
Superscalar Approach to Encryption

Username —> 8 number sequence – Public Key
Password —> 8 number sequence - Private Key
Stored Number —-–> 8 number sequence - Admin Key

Easiest form of encryption —–> Step 1 - Element-wise multiplication of the public and private keys
Step 2 - Element-wise addition of the resultant with admin key

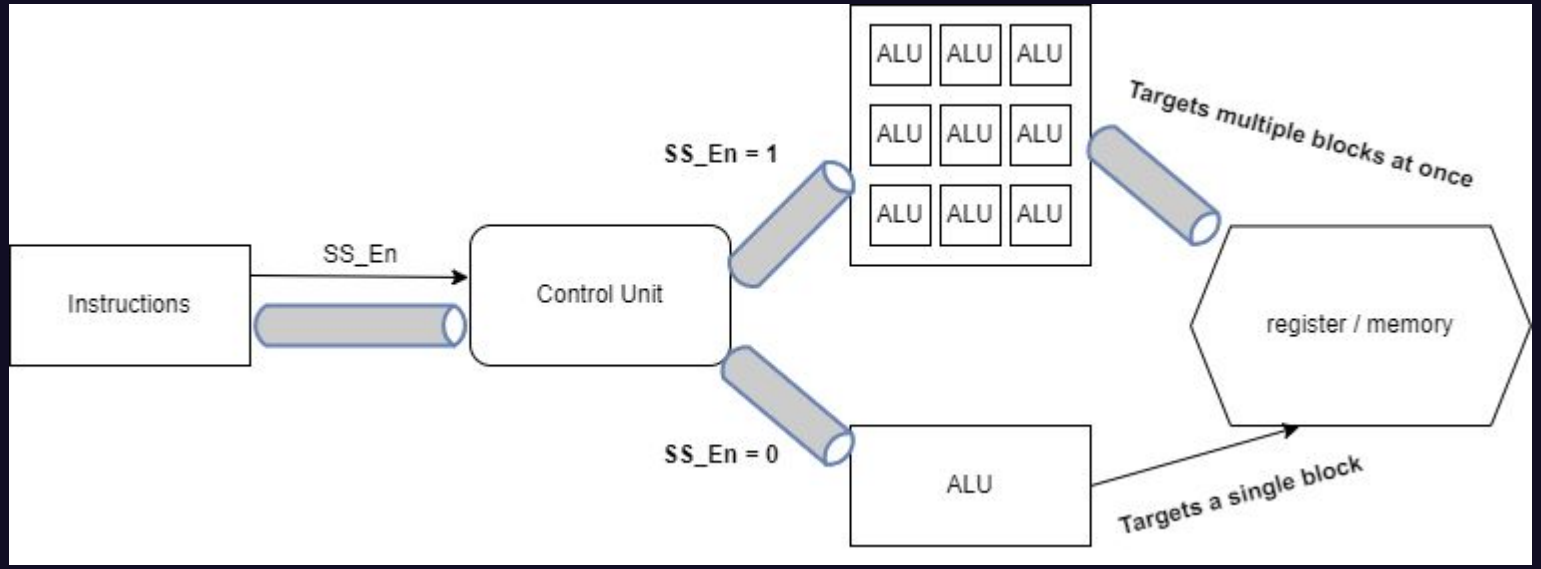


Conventional Implementation



Multi-ALU system

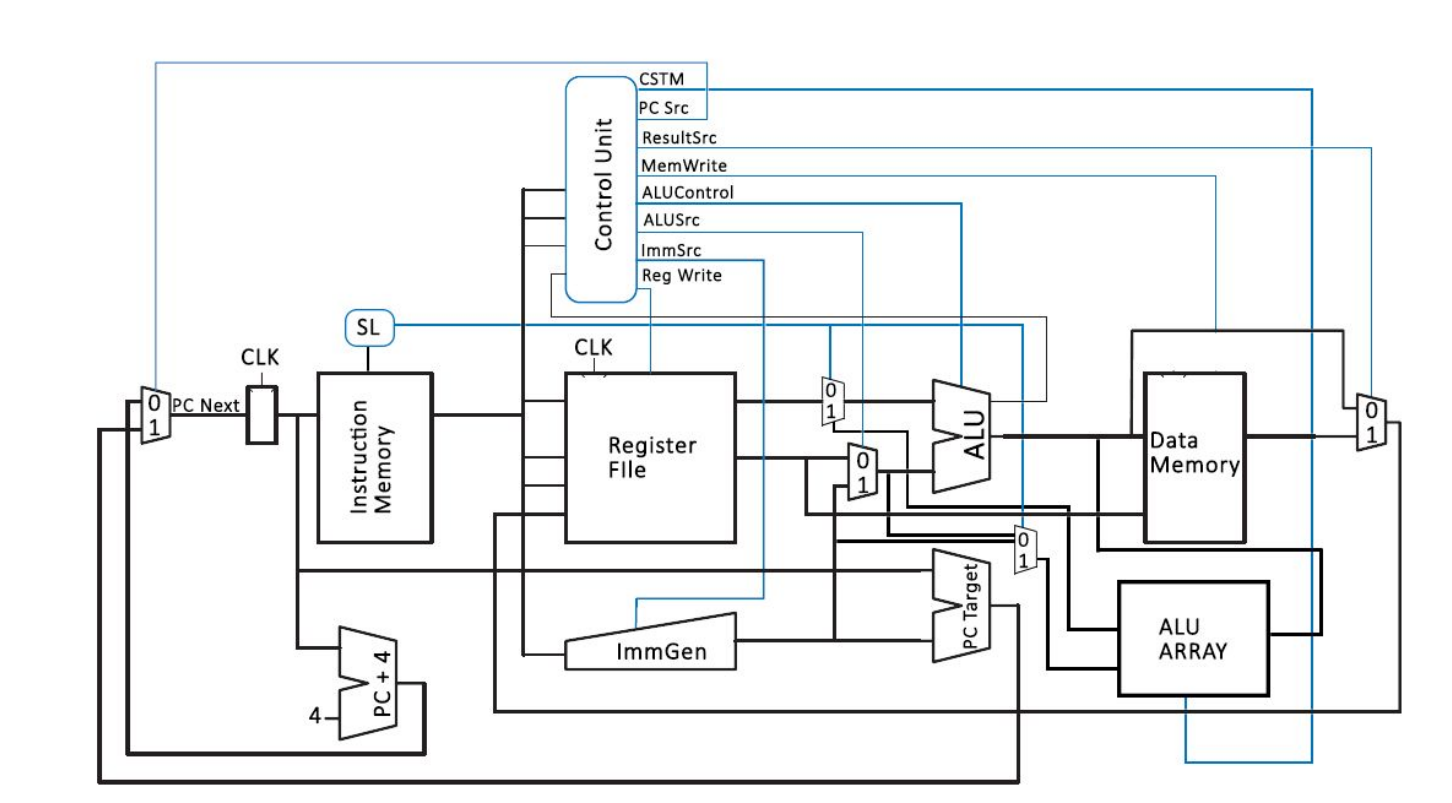
Architectural Tweaks for RISC-V Adaptation



For ease of current implementation, we assume that in our encryption application, each of the 8 numbers in “keys” are less than 10(decimal). So, they can be expressed in a single digit hex. This helps us account for entire sequence of the key in a 32bit value.

For example - ‘h ABCDDCBA corresponds to a public key – [10, 11, 12, 13, 13, 12, 11, 10]

System Architecture



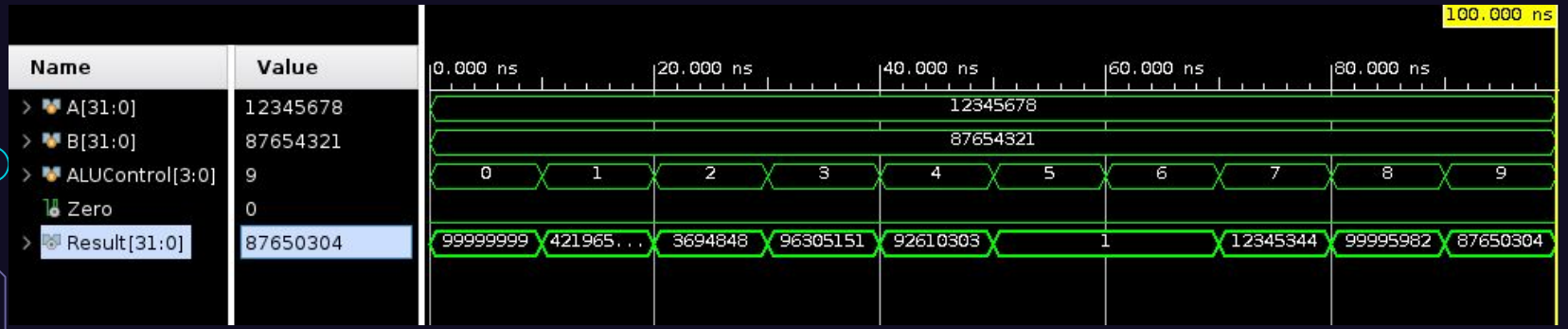
Instructions Implemented

Type	Instructions
R	ADD, SUB, SLT, OR, AND
I	LW, ADDI
S	SW
B	BEQ

Basic instructions

```
ALU ALU_tb_inst (.A(A), .B(B), .ALUControl(ALUControl), .Zero(Zero), .Result(Result));

initial begin
    A = 32'd12345678;
    B = 32'd87654321;
    ALUControl = 4'b0000; // test add operation
    #10;
    ALUControl = 4'b0001; // test sub operation
    #10;
    ALUControl = 4'b0010; // test and operation
    #10;
    ALUControl = 4'b0011; // test or operation
    #10;
    ALUControl = 4'b0100; // test xor operation
    #10;
    ALUControl = 4'b0101; // test slt operation
    #10;
    ALUControl = 4'b0110; // test sltu operation
    #10;
```



1. 00500113
Assembly = addi x2, x0, 5
Binary = 0000 0000 0101 0000 0000 0001 0001 0011

02. 00C00193
Assembly = addi x3, x0, 12
Binary = 0000 0000 1100 0000 0000 0001 1001 0011

03. FF718393
Assembly = addi x7, x3, -9
Binary = 1111 1111 0111 0001 1000 0011 1001 0011

04. 0023E233
Assembly = or x4, x7, x2
Binary = 0000 0000 0010 0011 1110 0010 0011 0011

05. 0041F2B3
Assembly = and x5, x3, x4
Binary = 0000 0000 0100 0001 1111 0010 1011 0011

06. 004282B3
Assembly = add x5, x5, x4
Binary = 0000 0000 0100 0010 1000 0010 1011 0011

07. 02728863
Assembly = beq x5, x7, 48
Binary = 0000 0010 0111 0010 1000 1000 0110 0011

08. 0041A233
Assembly = slt x4, x3, x4
Binary = 0000 0000 0100 0001 1010 0010 0011 0011

09. 00020463
Assembly = beq x4, x0, 8
Binary = 0000 0000 0000 0010 0000 0100 0110 0011

10. 00000293
Assembly = addi x5, x0, 0
Binary = 0000 0000 0000 0000 0000 0010 1001 0011

11. 0023A233
Assembly = slt x4, x7, x2
Binary = 0000 0000 0010 0011 1010 0010 0011 0011

12. 005203B3
Assembly = add x7, x4, x5
Binary = 0000 0000 0101 0010 0000 0011 1011 0011

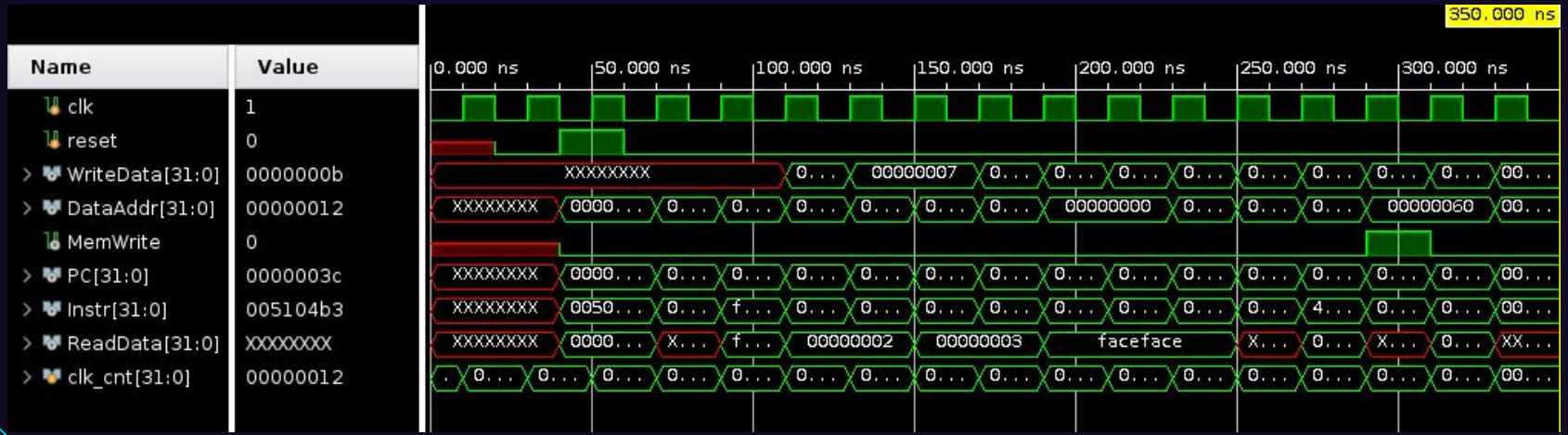
13. 402383B3
Assembly = sub x7, x7, x2
Binary = 0100 0000 0010 0011 1000 0011 1011 0011

14. 0471AA23
Assembly = lw x7, 84(x3)
Binary = 0000 0100 0111 0001 1010 1010 0010 0011

15. 06002103
Assembly = sw x2, 96(x0)
Binary = 0000 0110 0000 0000 0010 0001 0000 0011

16. 005104B3
Assembly = add x9, x2, x5
Binary = 0000 0000 0101 0001 0000 0100 1011 0011

Output for basic instructions



THANK YOU!

WIP - Integrating the superscalar architecture to pipelined architecture for also being able to show CG-multithreading approaches.