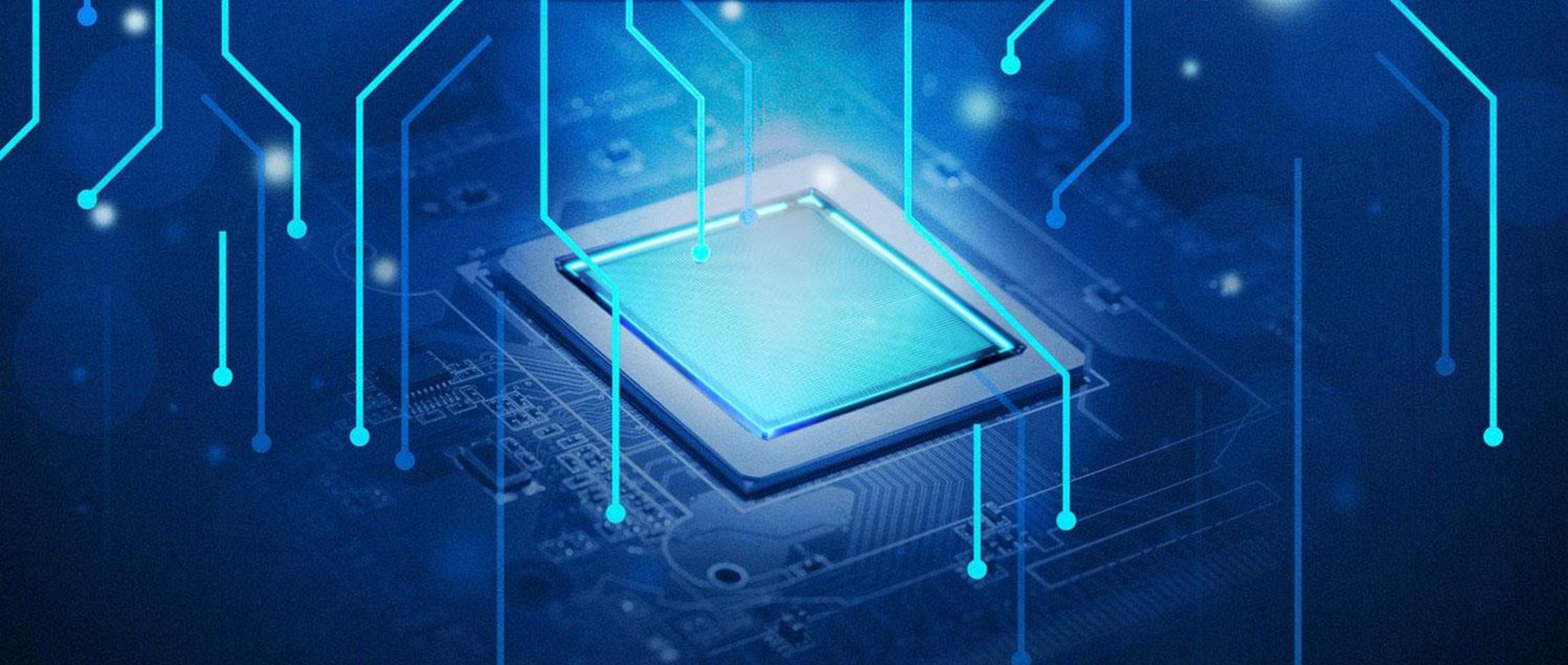
# Finite Field Processor

Foundations of Computer Architecture SP21

Sabbir Ahmed
Calley Tinsman

# Contents



# Background

# Finite Fields

## Brief Mathematical Background

- A **field** is a set on which addition, subtraction, multiplication, and division are defined.
- A **finite field** (or Galois field) is a field that contains a finite number of elements.
- A **primitive polynomial** generates all **elements** of a finite field.
- We can perform math using these elements!

## Applications

- Mathematical applications such as number theory and algebraic geometry.
- Computer science applications such as cryptography, coding theory, and error detection & correction.

# Our Example

## For the duration of this presentation, we'll be using the primitive polynomial $x^3 + x^2 + 1$.

- This "input" polynomial of GF(2) has degree 3 and generates $2^3 - 1 = 7$ elements.
- These elements are **cyclic** [2].
- We use two different representations for the elements: **element form** and **polynomial form**. Different mathematical operations require using different forms.
- The number zero is the additive identity [1].

**Table 1:** The 8 Element Vectors of $x^3 + x^2 + x^0$ in $GF(2)[x]$

| Element | Symbol | Polynomial Form | Symbol |
|---|---|---|---|
| 0 (NULL) | [1] | $0+0+0$ | 000 |
| $\beta^0$ | 000 | $0+0+\beta^0$ | 001 |
| $\beta^1$ | 001 | $0+\beta^1+0$ | 010 |
| $\beta^2$ | 010 | $\beta^2+0+0$ | 100 |
| $\beta^3$ | 011 | $\beta^2+0+\beta^0$ | 101 |
| $\beta^4$ | 100 | $\beta^2+\beta^1+\beta^0$ | 111 |
| $\beta^5$ | 101 | $0+\beta^1+\beta^0$ | 011 |
| $\beta^6$ | 110 | $\beta^2+\beta^1+0$ | 110 |
| $\beta^7$ | [2] | $0+0+\beta^0$ | 001 |

# Finite Field Arithmetic

### Addition & Subtraction

+/−

- Addition & subtraction are equivalent
- Performed by computing bitwise XOR with the operands in polynomial form
- Example:

$$\begin{aligned} \beta^4 + \beta^3 &= 111 \oplus 101 \\ &= 010 \\ &= \beta^1 \end{aligned}$$

### Multiplication

×

- Identical to the sum of the operands in element form modulus $2^n - 1$
- The modulus is required because of the cyclic nature of the finite field
- Example:

$$\begin{aligned} \beta^4 \times \beta^3 &= (100 + 011)\ \%\ 111 \\ &= 000 \\ &= \beta^0 \end{aligned}$$

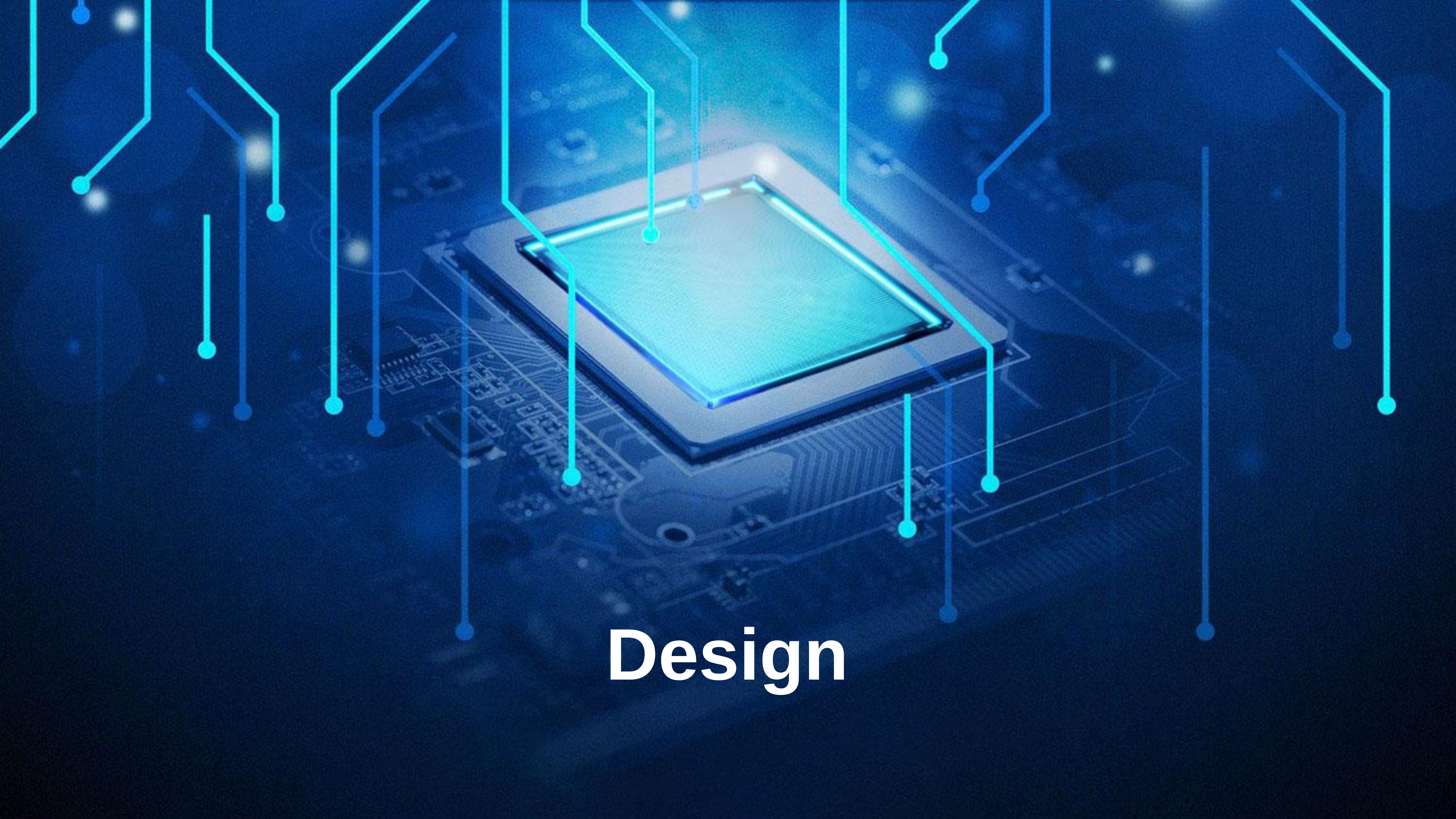
### Division

÷

- Similar to multiplication, but is the difference of the operands in element form modulus $2^n - 1$
- Example:

$$\begin{aligned} \beta^4 \div \beta^3 &= (100 - 011)\ \%\ 111 \\ &= 001 \\ &= \beta^1 \end{aligned}$$

Note: The output form will always be the same as the input form.



# Design

# Scope

## Input polynomials, $GF(p^n)$

- $p = 2$
- Assumed primitive
- Polynomial degrees: $2 \leq n \leq 9$
- Represented as a 1-indexed 9-bit vector, with the bit position representing all the terms except the 0$^{th}$
  - Example: $x^9 + x^7 + x^3 + x^2 + 1$ is represented as $1\ 0100\ 0110$

## Values generated in $GF(2^n)$

- Field order: $2^n - 1$ elements
- Field range: $[0, 2^n - 2]$
- Elements **do not** include the additive identity (-1$^{st}$ term)
- Elements include the multiplicative identity (0$^{th}$ term)
- Elements stop generating on $2^n - 1$

# Instructions

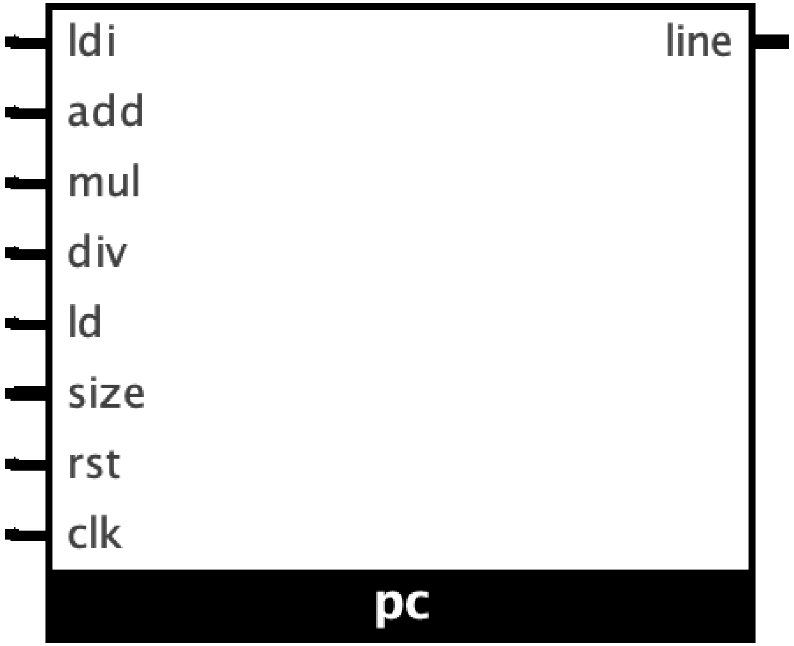
| Opcode | Instruction | Description | Cycles |
|---|---|---|---|
| 000 | ldi | Load immediate value to operand 1 | 2 |
| 001 | add | Add operand 1 to operand 2 | 2 |
| 010 | mul | Multiply operand 1 by operand 2 | 2 |
| 011 | div | Divide operand 1 by operand 2 | 2 |
| 100 | ld | Load memory value to operand 1 | 2 |
| 101 | init | Initialize polynomial constants | 2 |
| 110 | gen | Generate all $2^n$ polynomial elements | $2^n - 1$ |
| 111 | rst | Reset | 2 |

12-bit instruction – opcode (bits 11-9) & operand (bits 8-0)

# Components

## Program Counter

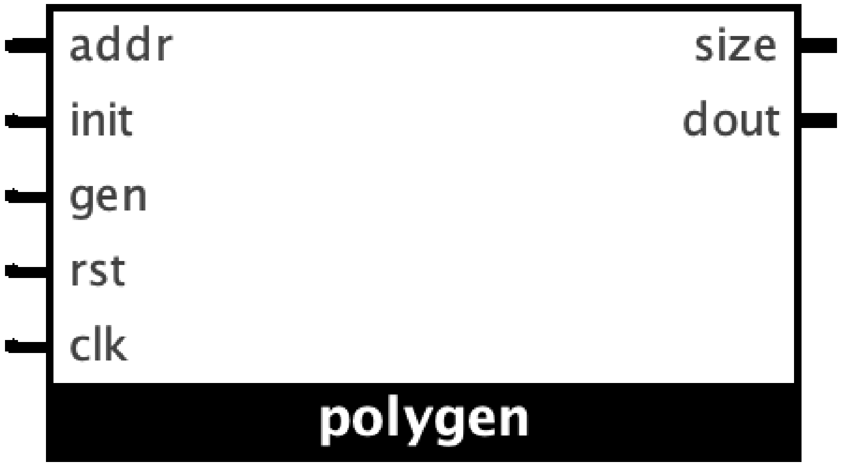
- The program counter (PC) holds the address of the next instruction to be executed.
- Because different instructions require a different number of clock cycles to complete, the PC must know the instruction being performed (the first five inputs).
- In order to properly consider the number of clock cycles required to generate the field elements, it must also know the number of elements (size).
- The output is the next instruction to execute.



# Components

## Polynomial Generator

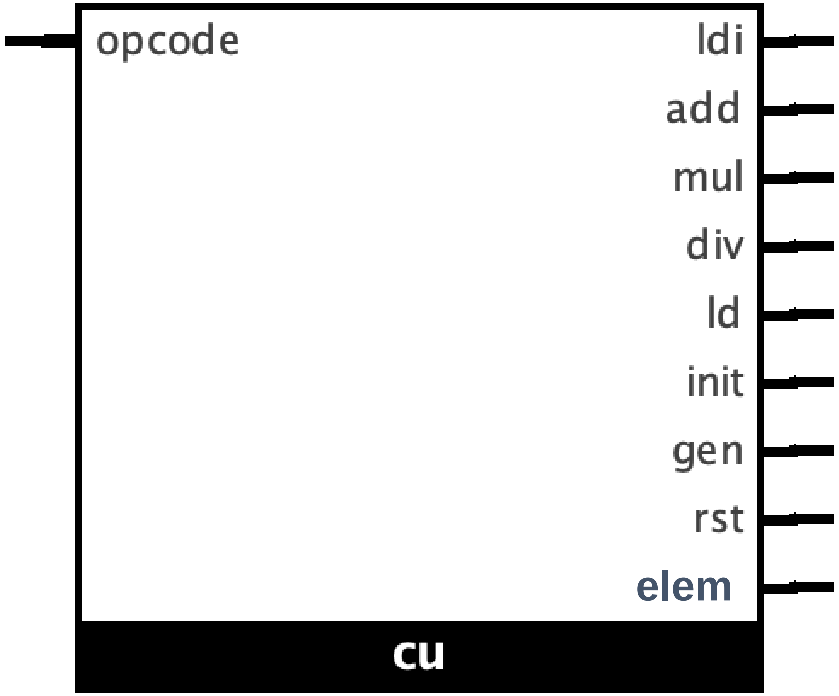
- The polynomial generator spawns all the elements generated by the input polynomial. These are the elements that we will perform mathematical operations with.
- The address (addr) is the element form of the polynomial.
- The constants in the polynomial generator are reset when init is set to high and the polynomials are populated when gen is set to high.
- The outputs are the size of the field and the polynomial form of the element.



# Components

## Control Unit

- The control unit processes the opcode to split it into individual instructions to send to the other components in the circuit.
- The elem output denotes whether to use the element or polynomial form of the element.

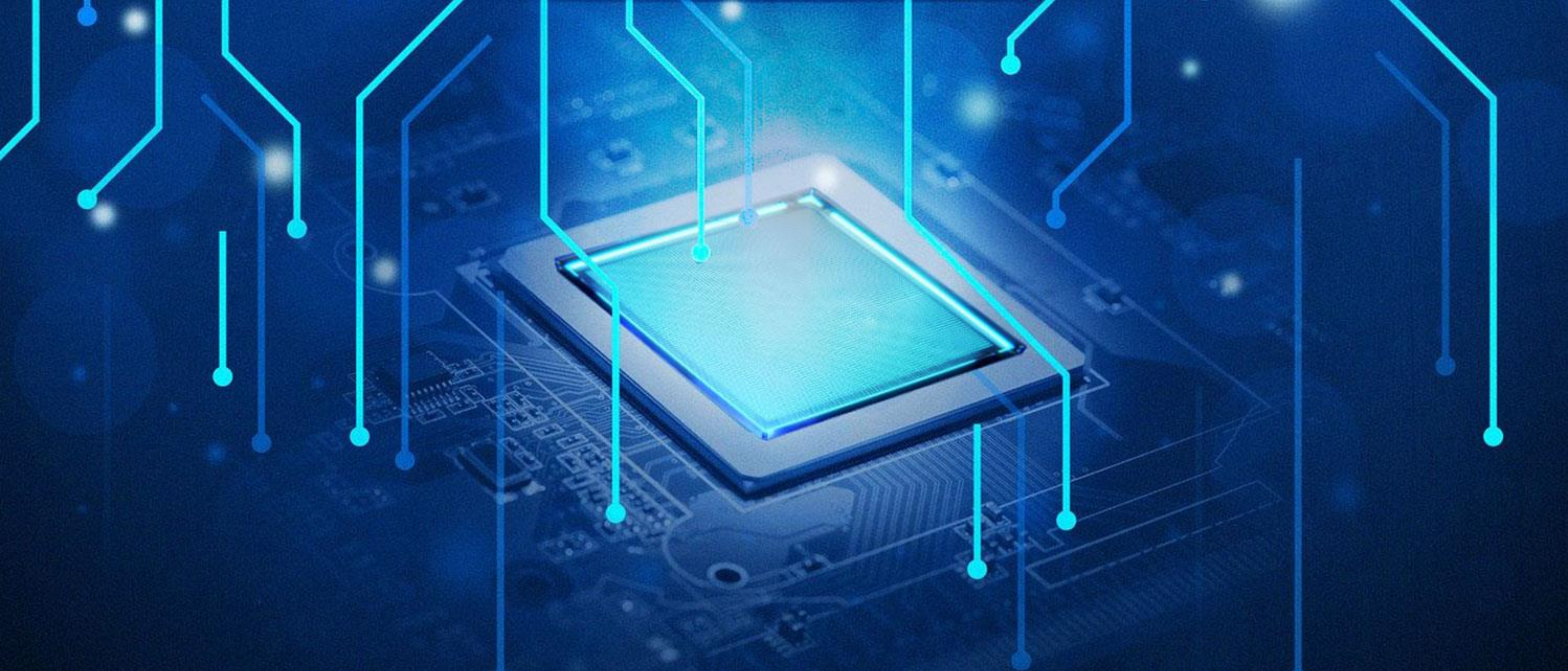


# Components

## Arithmetic Logic Unit

- The arithmetic logic unit (ALU) is responsible for performing the mathematical operations.
- The inputs include the operands, the field size, opcode, and an enable bit.
- It will perform addition/subtraction, multiplication, and division.


Live Demo



# Conclusion

# Future Work

- Improve latency of instructions:
  - Instructions without memory lookups, besides gen (ldi, add, init, rst), can be reduced to single-cycle operations by modifying the program counter
  - Memory read instructions (mul, div, ld) can be reduced to single-cycle operations by modifying the memory interface

- Add to the CPU:
  - capability to look up ALU output in memory
  - support for arithmetic with the additive identity (0)
  - support for checking irreducibility of input polynomials
  - support for checking primitivity of irreducible polynomials

- Investigate methods of optimizing generation of elements

# THANK YOU

Questions?

# References


- https://mathworld.wolfram.com/FiniteField.html
- https://en.wikipedia.org/wiki/Finite_field