



25.03.20 12:57

# CSCI 150

## Introduction to Digital and Computer System Design

### Lecture 5: Registers II



Jetic Gū  
2020 Winter Semester (S1)

# Overview

- Focus: Fundamentals of Complex Digital Circuit Design
- Architecture: von Neumann
- Textbook v4: Ch7 7.6; v5: Ch6 6.6
- Core Ideas:
  1. Implementation of Register Microoperations

# Register Transfer Operations

	Operator	Example		Operator	Example
Assignment	<code>&lt;=</code>	<code>ax &lt;= 12h</code>	Bitwise AND	<code>and</code>	<code>ax <b>and</b> bx</code>
Reg. Transfer	<code>&lt;=</code>	<code>ax &lt;= bx</code>	Bitwise OR	<code>or</code>	<code>ax <b>or</b> bx</code>
Addition	<code>+</code>	<code>ax + bx</code>	Bitwise NOT	<code>not</code>	<code><b>not</b> ax</code>
Subtraction	<code>-</code>	<code>ax - bx</code>	Bitwise XOR	<code>xor</code>	<code>ax <b>xor</b> bx</code>
Shift Left	<code>sll</code>	<code>ax <b>sll</b> 2</code>	Vectors		<code>ax(3 down to 0) ax(3 down to 0)</code>
Shift Right	<code>srl</code>	<code>ax <b>srl</b> 2</code>	Concatenate	<code>&amp;</code>	<code>ax(7 down to 4) &amp;ax(3 down to 0)</code>

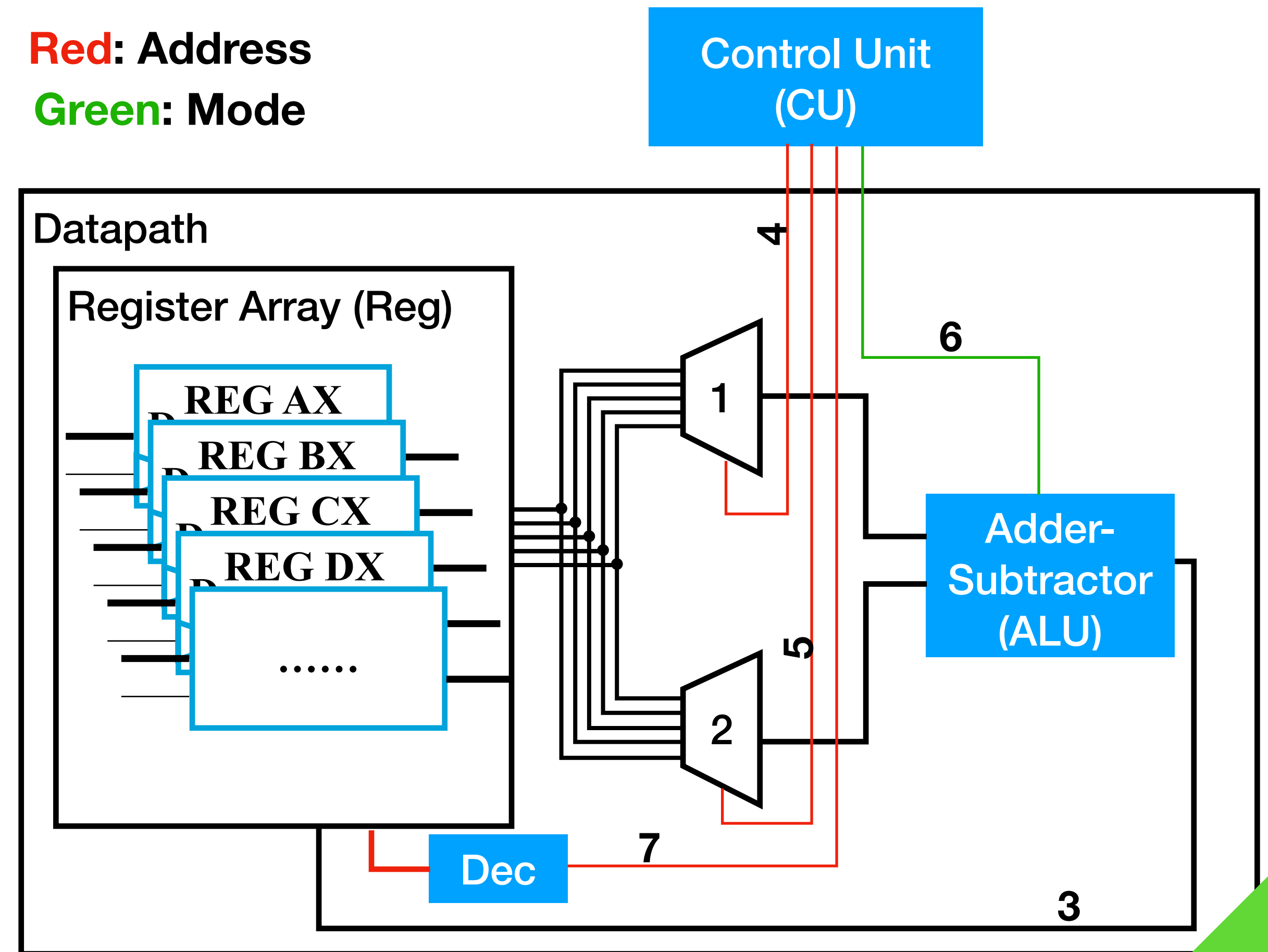
# Implementation of Datapath I

Architecture

# Datapath and Control Unit

1. ALU connected to Reg for 1st input
2. ALU connected to Reg for 2nd input
3. ALU connected to Reg to store result
4. CU tells ALU which register to take as 1st input
5. CU tells ALU which register to take as 2nd input
6. CU tells ALU which operation to do
7. CU tells which Reg to store result in  
Using decoder and EN on each Register

Red: Address  
Green: Mode



# Single Register Microoperations

- In reality, a single circuit is designed to do multiple stuff
  - a CPU can perform thousands/millions of different types of instructions
- How to design it?
- Let's start with single register Microoperations
  - Input: one of the registers; Output: another one of the registers;

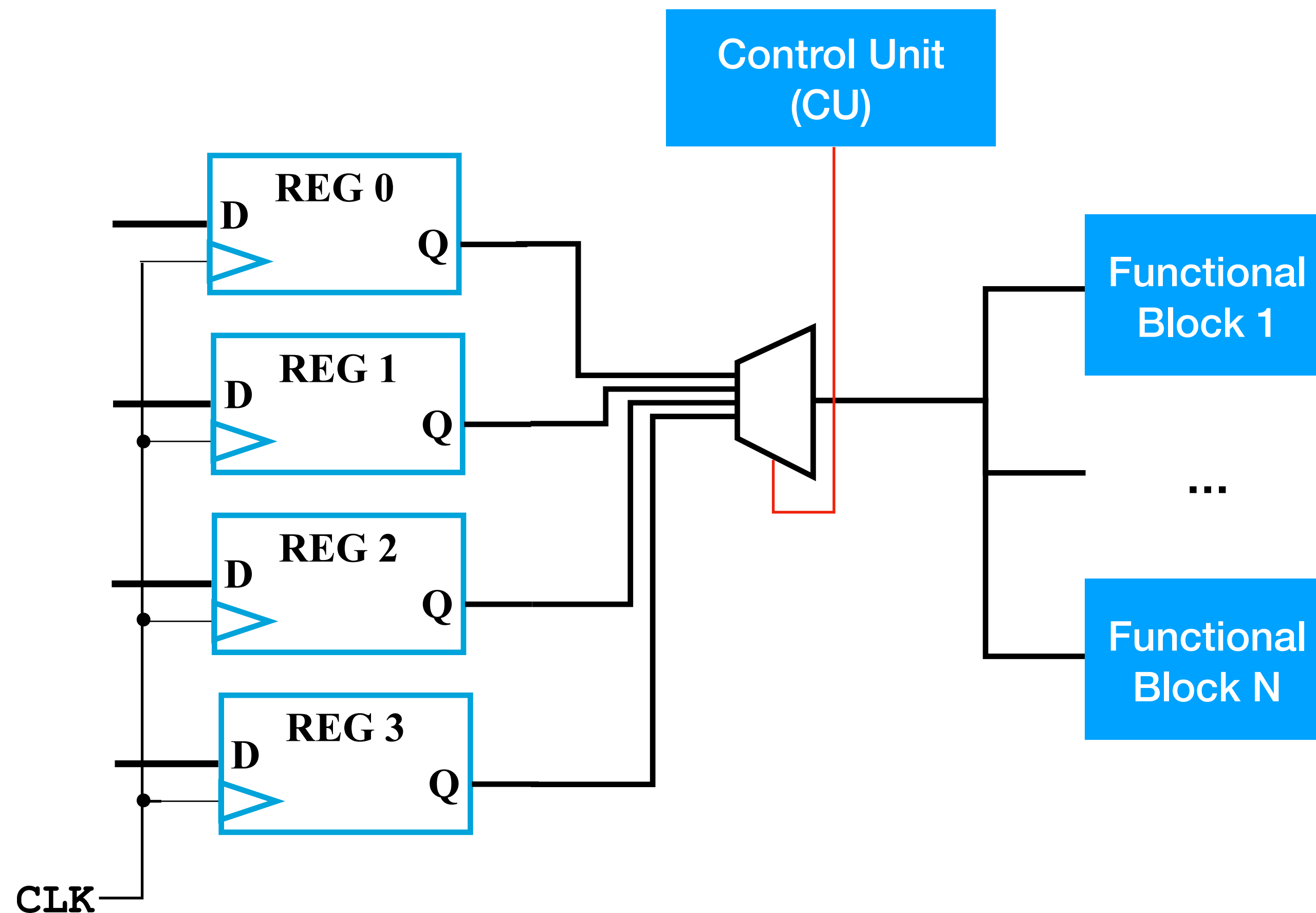
# Single Register Microoperations

	Operator	Example		Operator	Example
<b>Assignment</b>	<code>&lt;=</code>	<code>ax &lt;= 12h</code>	<b>Bitwise AND</b>	<code>and</code>	<code>ax <b>and</b> bx</code>
<b>Reg. Transfer</b>	<code>&lt;=</code>	<code>ax &lt;= bx</code>	<b>Bitwise OR</b>	<code>or</code>	<code>ax <b>or</b> bx</code>
<b>Addition</b>	<code>+</code>	<code>ax + bx</code>	<b>Bitwise NOT</b>	<code>not</code>	<code><b>not</b> ax</code>
<b>Subtraction</b>	<code>-</code>	<code>ax - bx</code>	<b>Bitwise XOR</b>	<code>xor</code>	<code>ax <b>xor</b> bx</code>
<b>Shift Left</b>	<code>sll</code>	<code>ax <b>sll</b> 2</code>	<b>Vectors</b>		<code>ax(3 down to 0) ax(3 down to 0)</code>
<b>Shift Right</b>	<code>srl</code>	<code>ax <b>srl</b> 2</code>	<b>Concatenate</b>	<code>&amp;</code>	<code>ax(7 down to 4) &amp;ax(3 down to 0)</code>

Concept

# Select Registers for Input

- Suppose we have 4 registers, we want to select one to provide input to a functional block

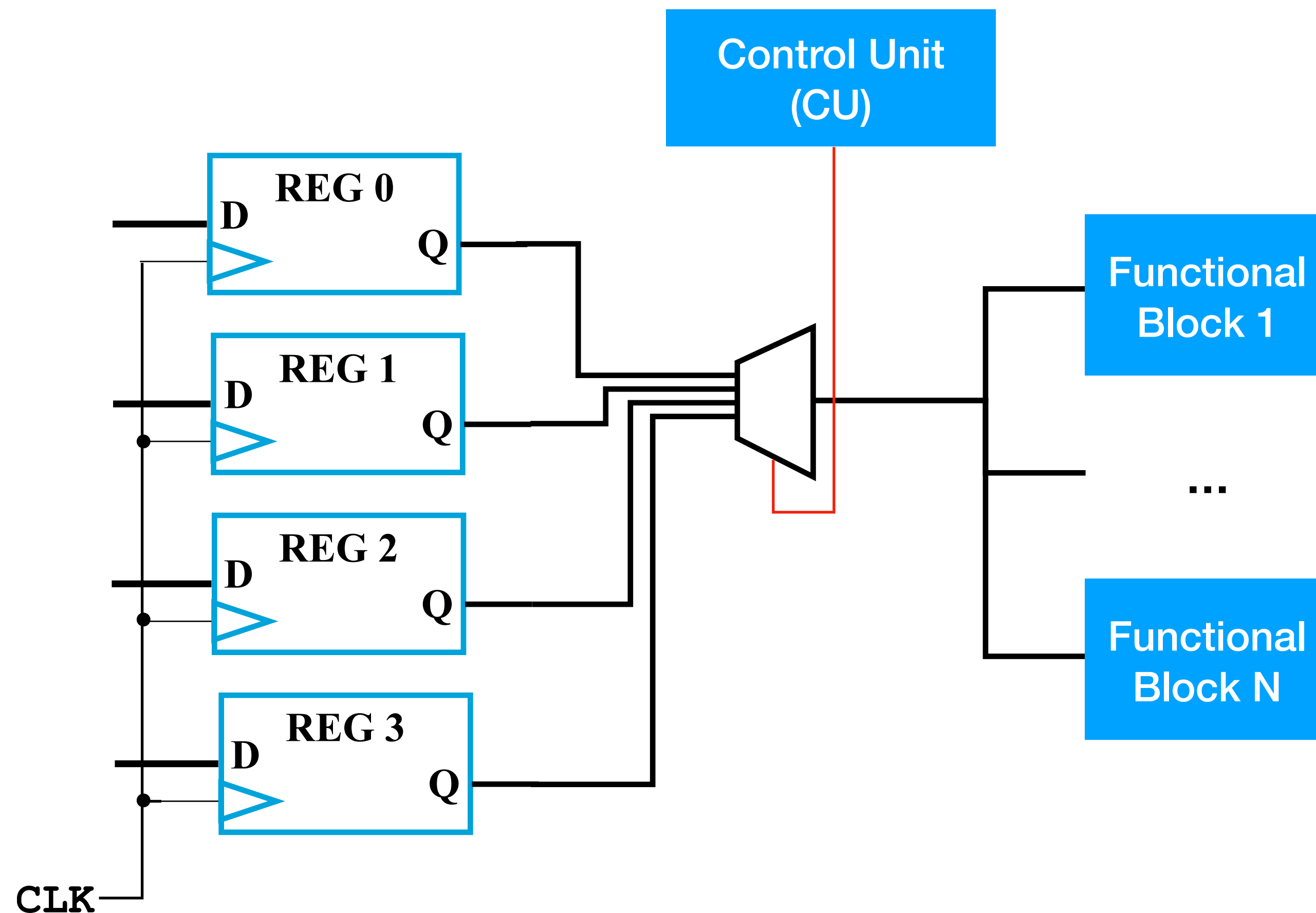




# Select Registers for Input

- Suppose we have 4 registers, we want to select one to provide input to a functional block

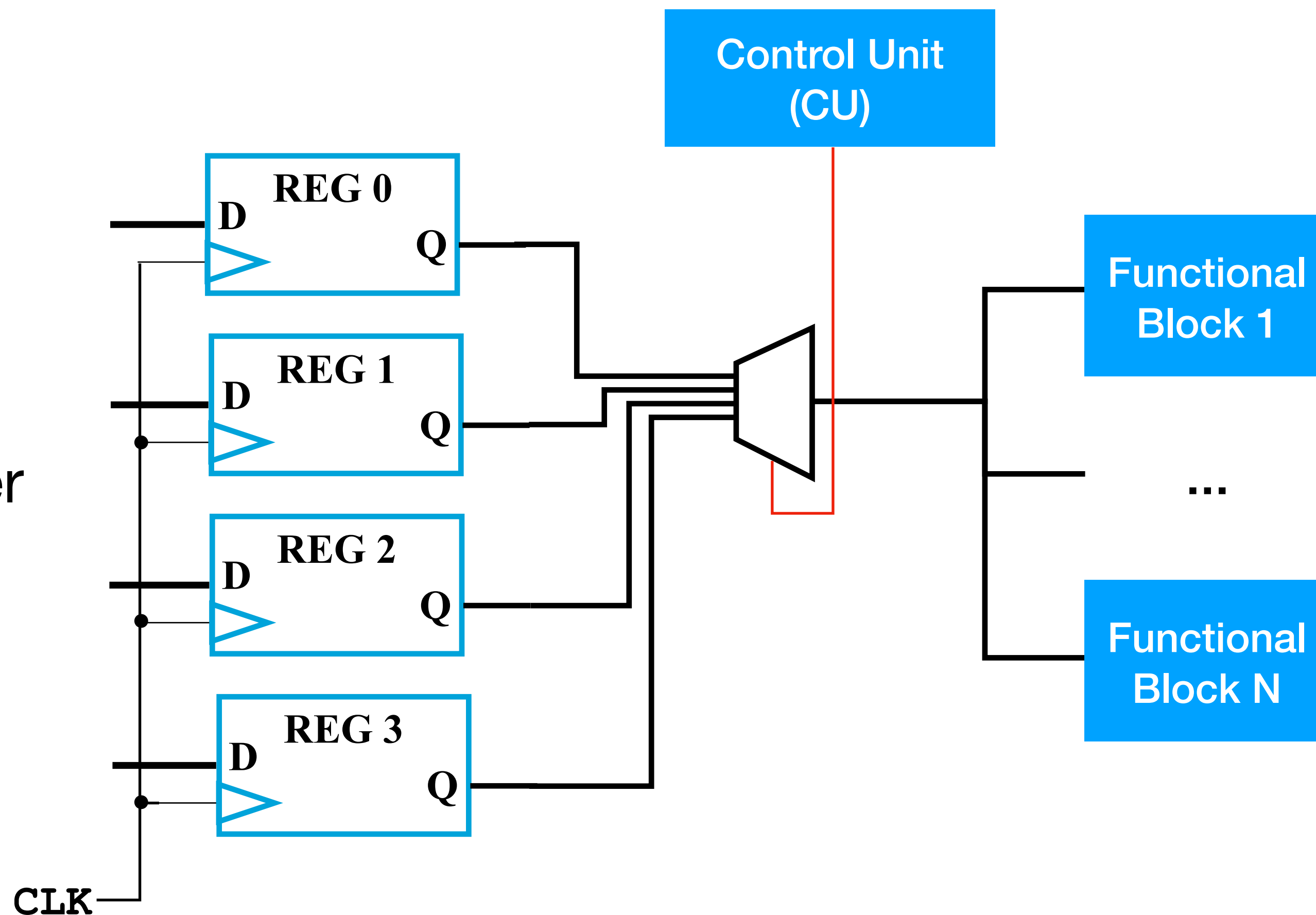
- Suppose  $2^n$  registers, each 8-bit



# Select Registers for Input

- Suppose we have 4 registers, we want to select one to provide input to a functional block

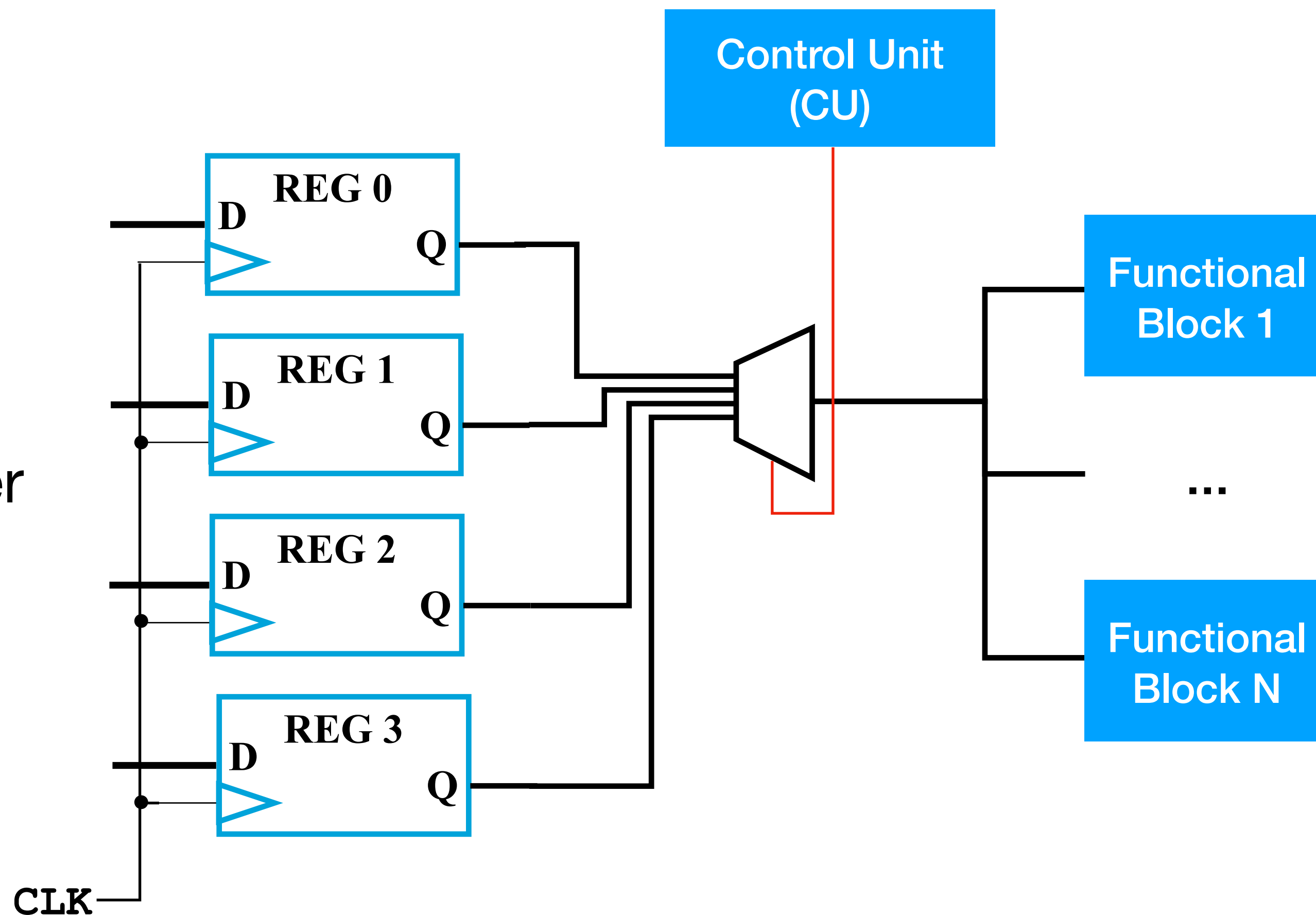
- Suppose  $2^n$  registers, each 8-bit
- What kind of multiplexer should be used?



# Select Registers for Input

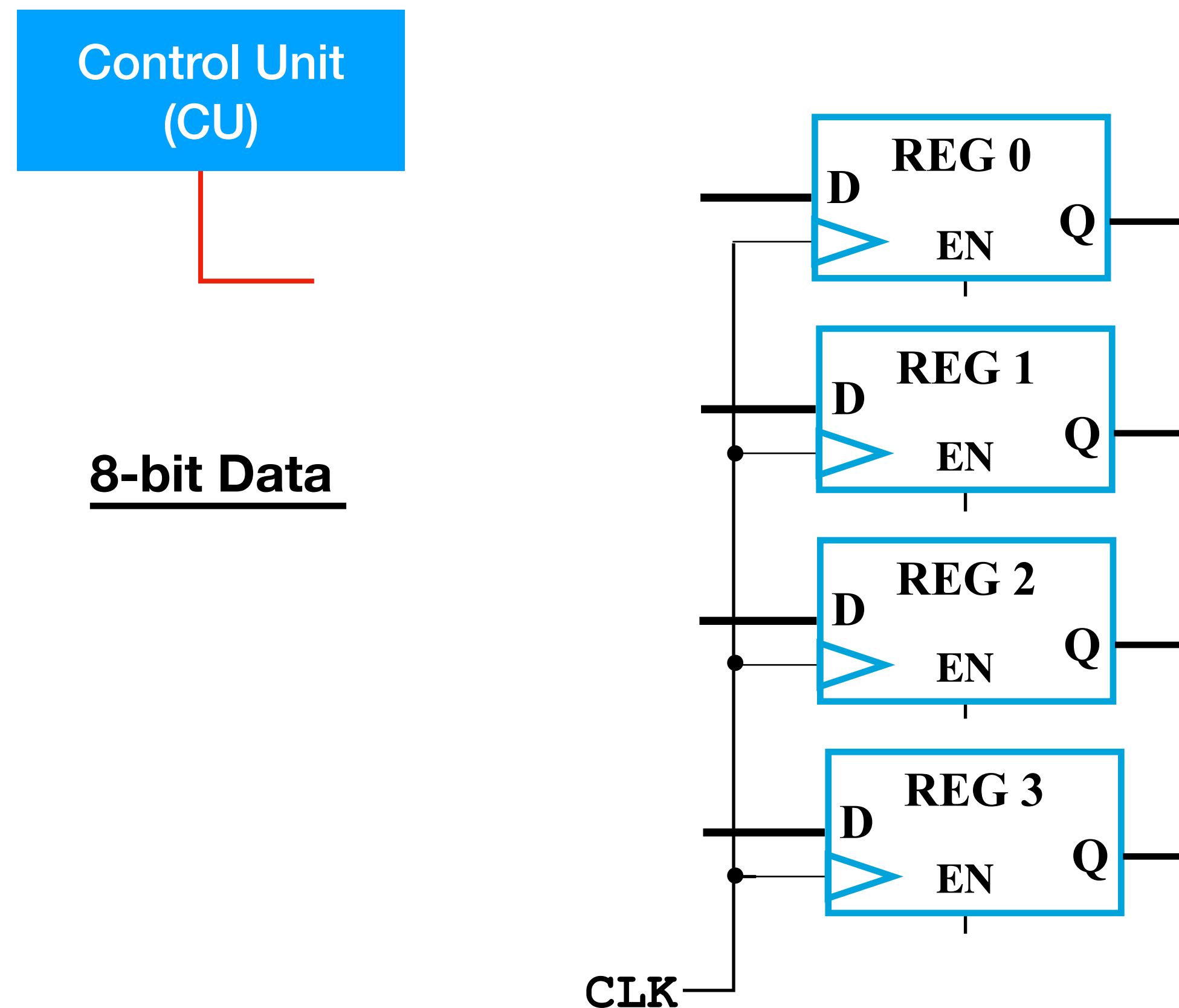
- Suppose we have 4 registers, we want to select one to provide input to a functional block

- Suppose  $2^n$  registers, each 8-bit
- What kind of multiplexer should be used?
- How many bits on the red wire?



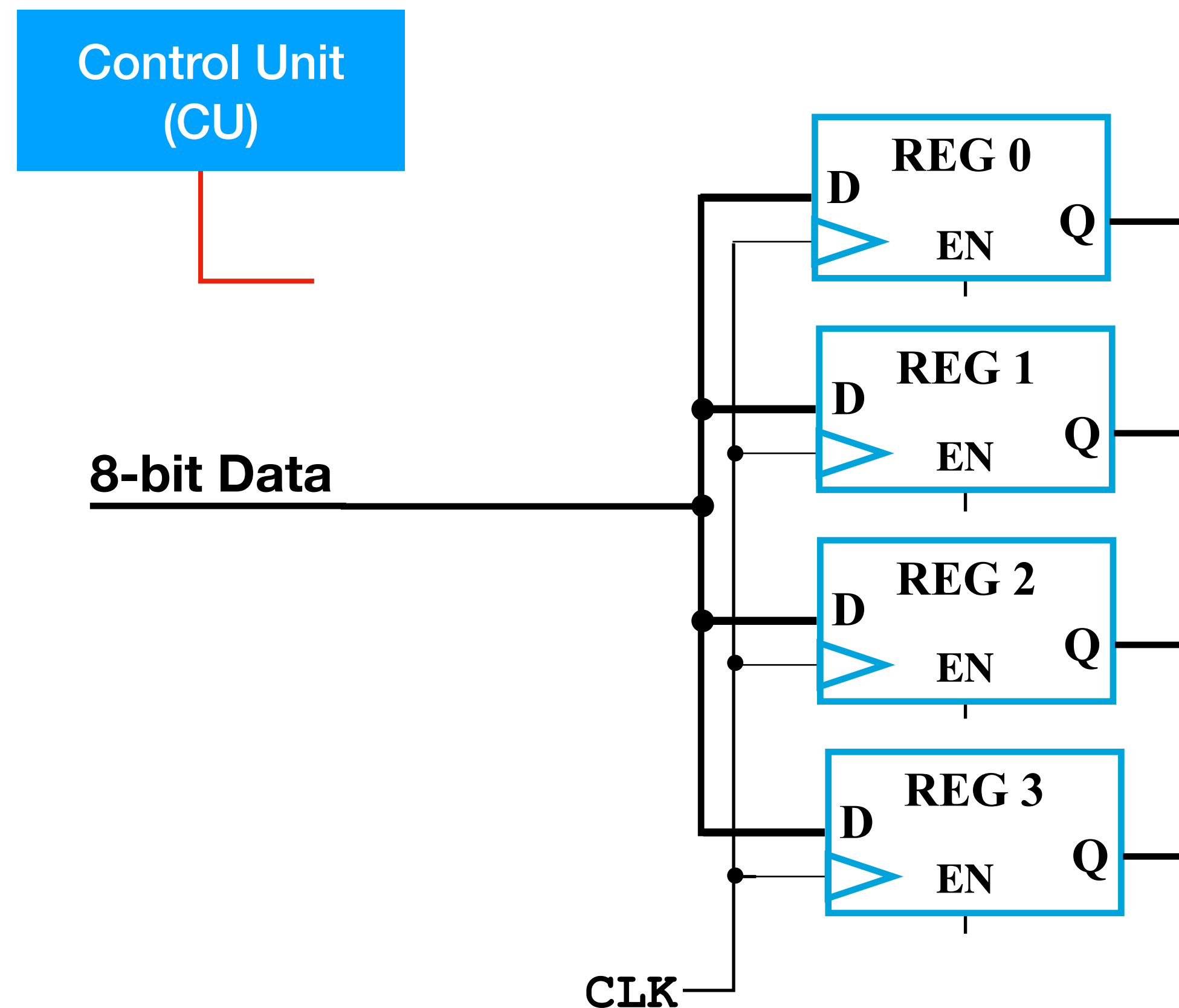
# Select Registers to Output

- Suppose we have 4 registers, we want to select one to store our output



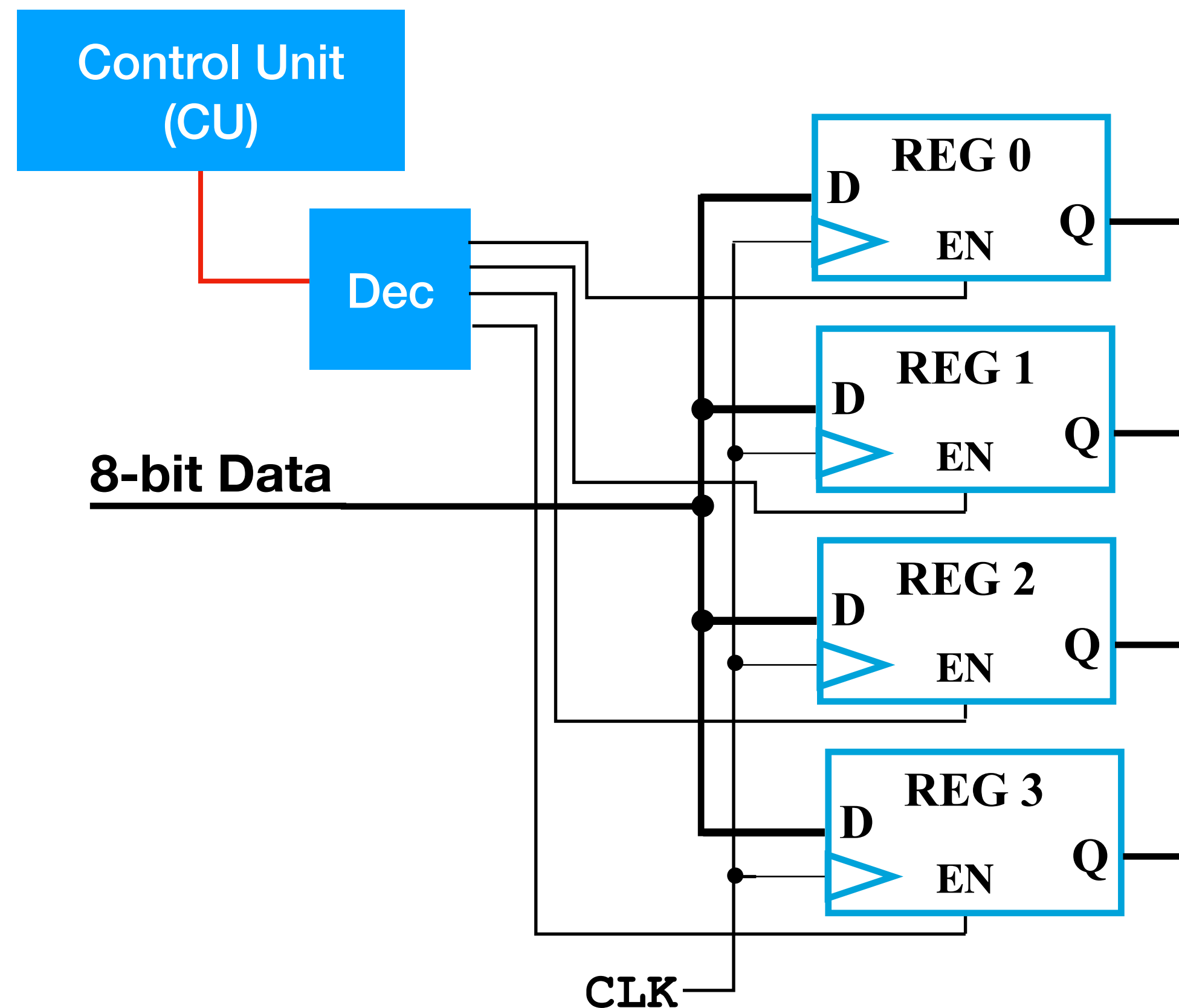
# Select Registers to Output

- Suppose we have 4 registers, we want to select one to store our output



# Select Registers to Output

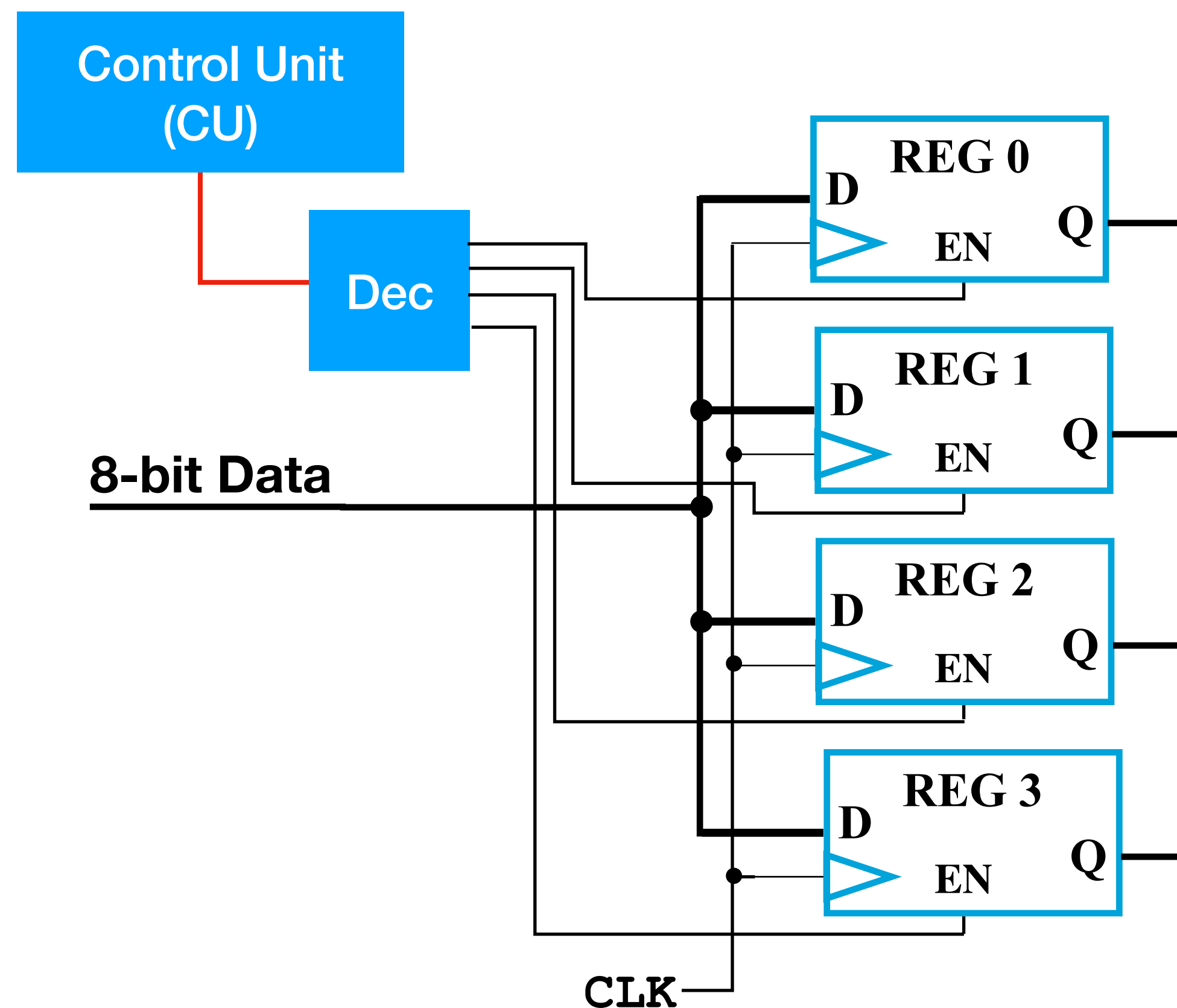
- Suppose we have 4 registers, we want to select one to store our output



# Select Registers to Output

- Suppose we have 4 registers, we want to select one to store our output

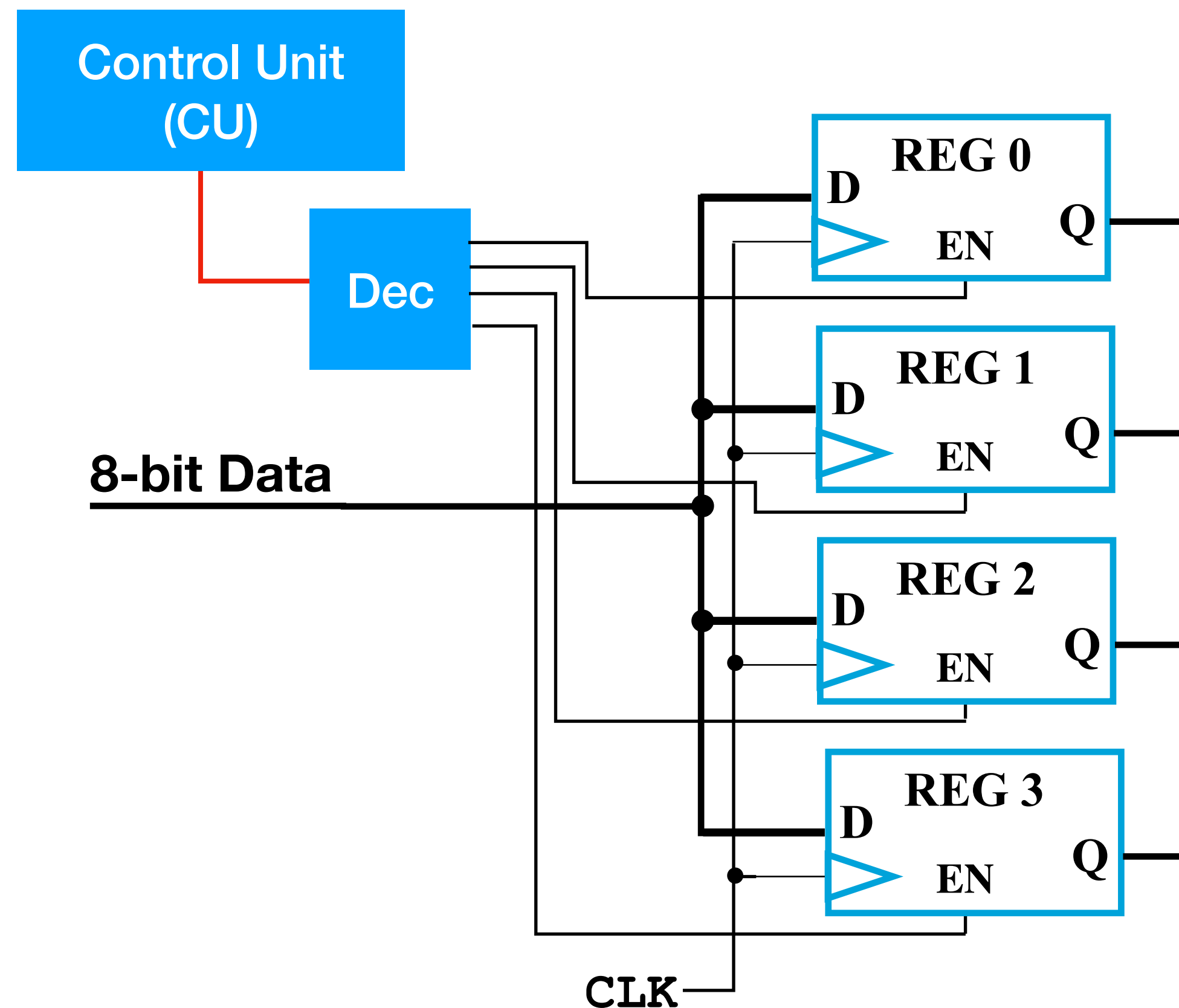
- Suppose  $2^n$  registers, each 8-bit



# Select Registers to Output

- Suppose we have 4 registers, we want to select one to store our output

- Suppose  $2^n$  registers, each 8-bit
- What kind of decoder should be used?

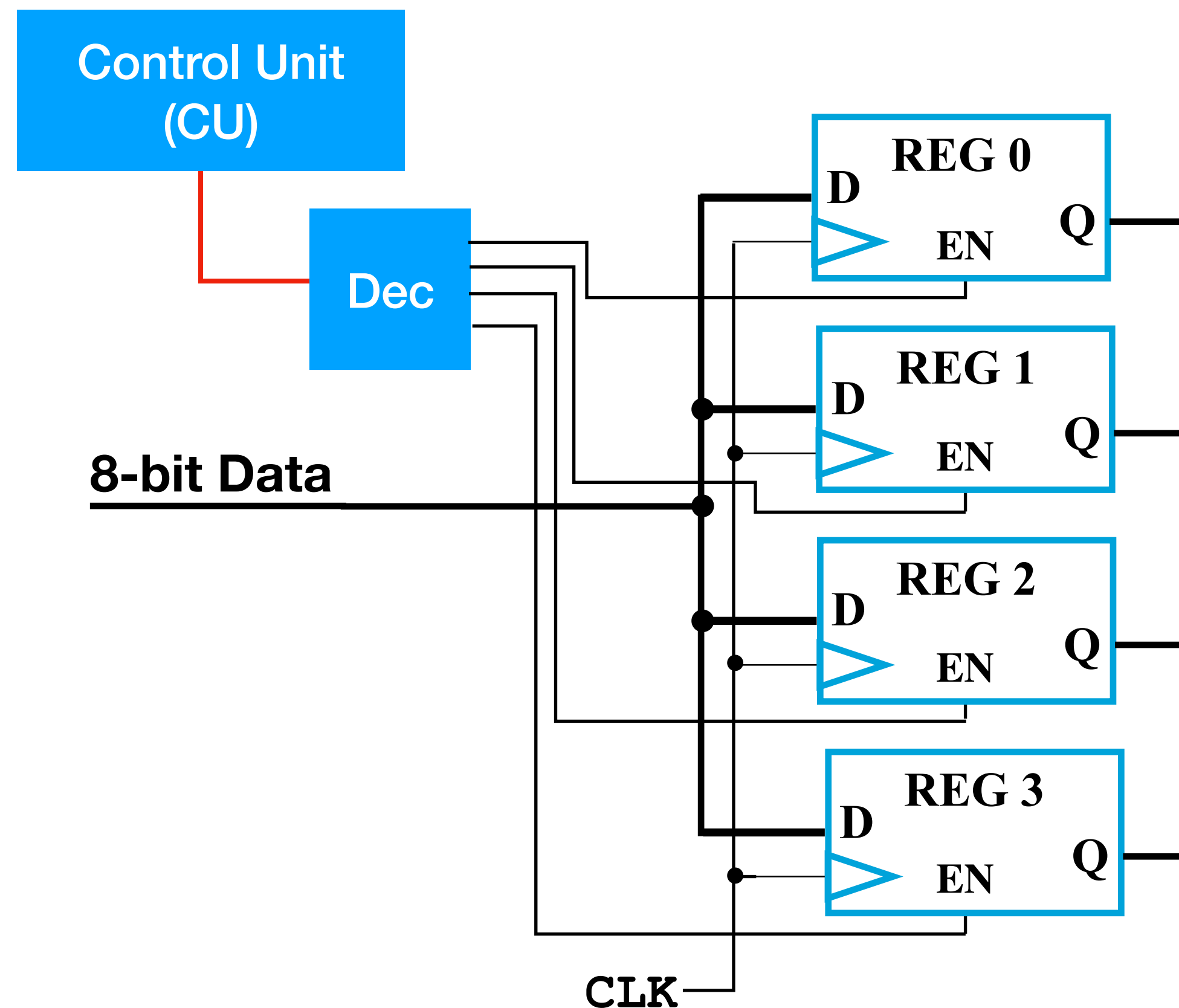




# Select Registers to Output

- Suppose we have 4 registers, we want to select one to store our output

- Suppose  $2^n$  registers, each 8-bit
- What kind of decoder should be used?
- How many bits on the red wire?



# Single Register Microoperations

- Assignment  
e.g. `ax <= 12h`
- Register Transfer  
e.g. `ax <= bx`
- Shift left by  $X < 8$   
e.g. `ax sll X`
- Shift right by  $X < 8$   
e.g. `ax srl X`
- Bitwise NOT  
e.g. `not ax`
- Vector by  
 $0 \leq X \leq Y < 8$   
e.g. `ax (Y down to X)`
- Rotate left by  $X < 8$   
e.g. `ax rll X`
- Rotate right by  $X < 8$   
e.g. `ax rrl X`

Functional  
Block 1

...

Functional  
Block N

# Single Register Microoperations

0. Assignment

e.g. `ax <= 12h`

1. Register Transfer

e.g. `ax <= bx`

2. Shift left by  $X < 8$

e.g. `ax sll X`

2. Shift right by  $X < 8$

e.g. `ax srl X`

3. Bitwise NOT

e.g. `not ax`

4. Vector by

$0 \leq X \leq Y < 8$

e.g. `ax (Y down to X)`

5. Rotate left by  $X < 8$

e.g. `ax rll X`

5. Rotate right by  $X < 8$

e.g. `ax rrl X`

Functional  
Block 1

...

Functional  
Block N

Example

# Single Register Microoperations

0. Assignment

e.g. `ax <= 12h`

1. Register Transfer

e.g. `ax <= bx`

2. Shift left by  $X < 8$

e.g. `ax sll X`

2. Shift right by  $X < 8$

e.g. `ax srl X`

3. Bitwise NOT

e.g. `not ax`

4. Vector by

$0 \leq X \leq Y < 8$

e.g. `ax (Y down to X)`

5. Rotate left by  $X < 8$

e.g. `ax rll X`

5. Rotate right by  $X < 8$

e.g. `ax rrl X`

0. Assign.

1. Transfer

2. Shift L/R

3. NOT

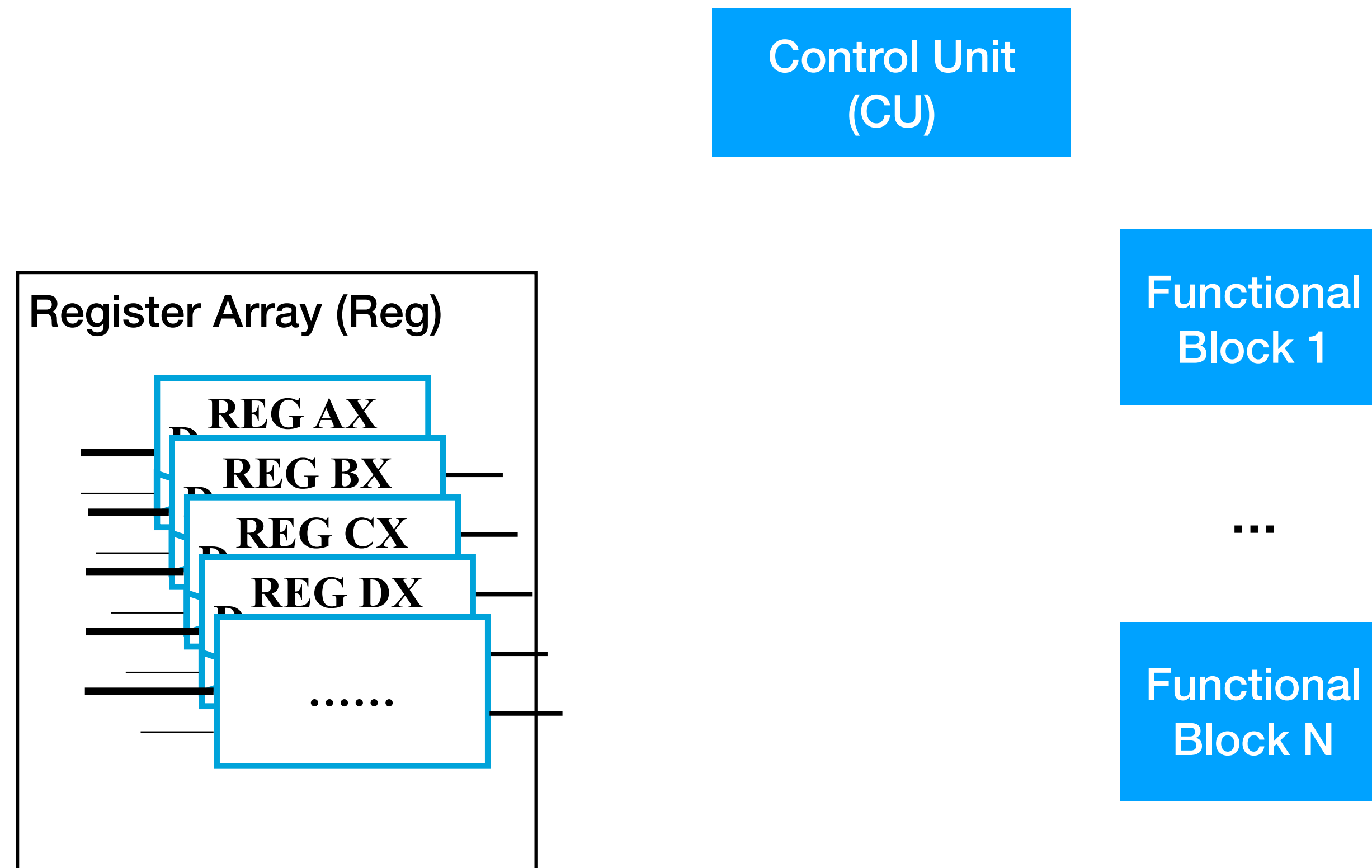
4. Vector

5. Rotate L/R

Example

# Single Register Microoperations

**Red:** Address  
**Green:** Mode

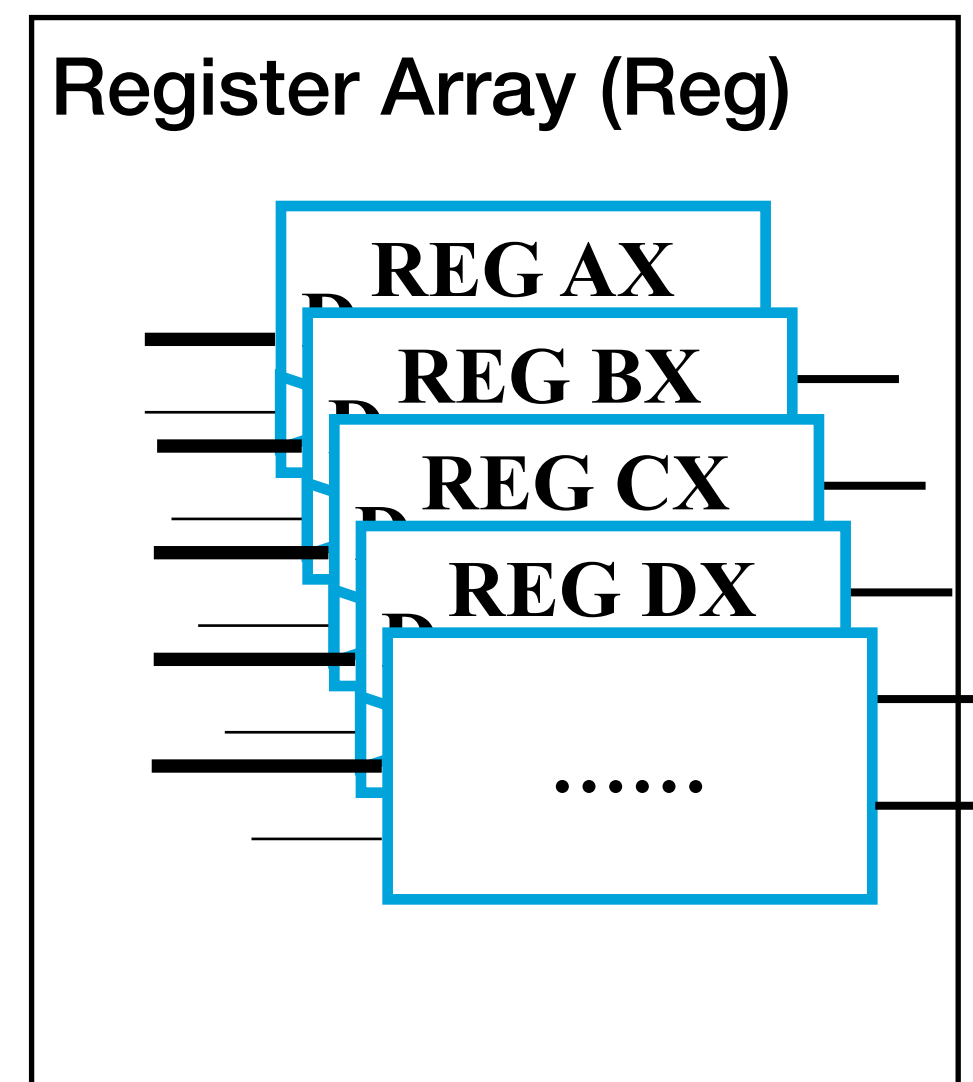


# Single Register Microoperations

**Red:** Address

**Green:** Mode

- **Select** Input REG to be fed into Processing Blocks



Control Unit  
(CU)

Functional  
Block 1

...

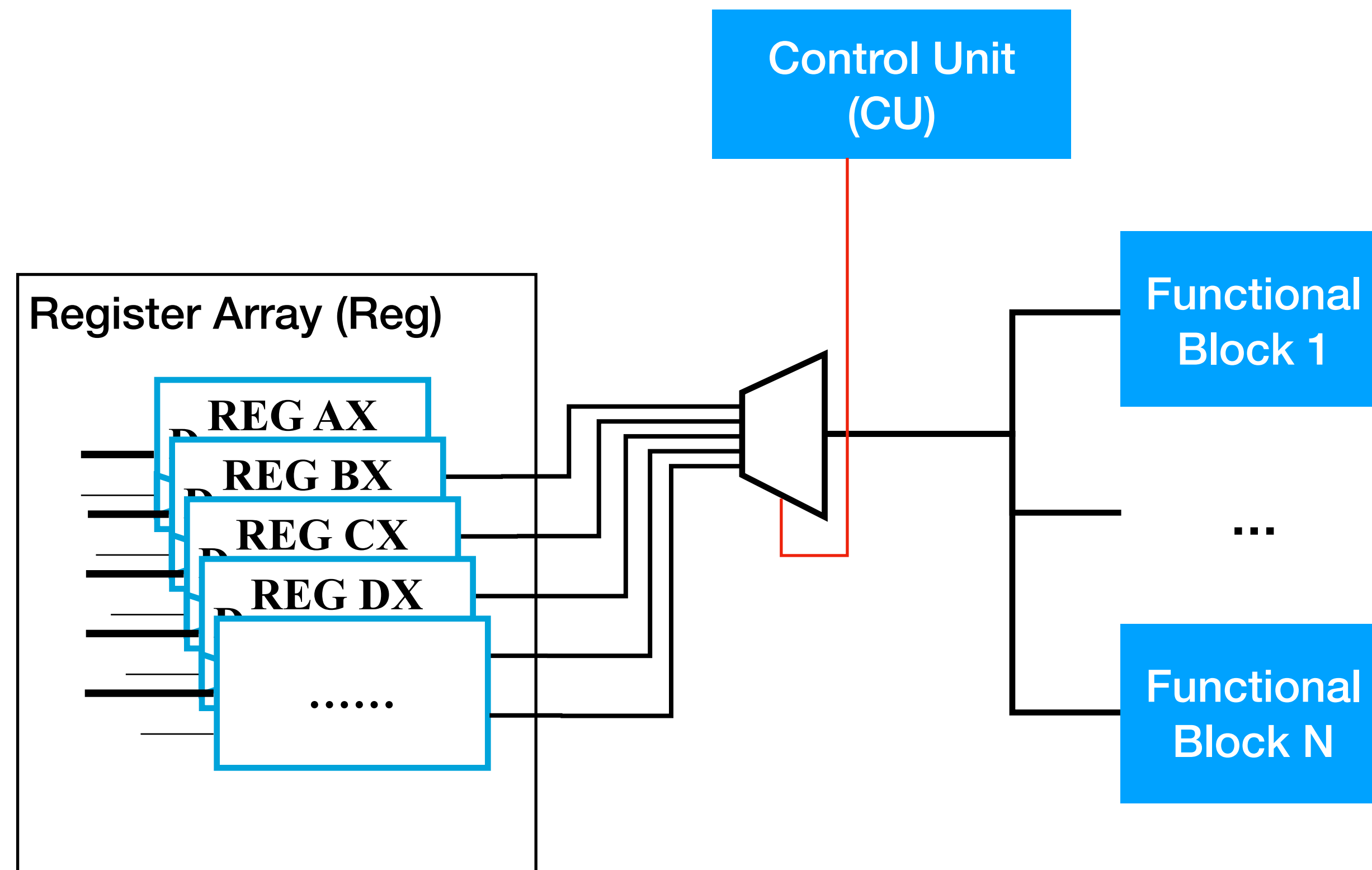
Functional  
Block N

# Single Register Microoperations

**Red:** Address

**Green:** Mode

- **Select** Input REG to be fed into Processing Blocks

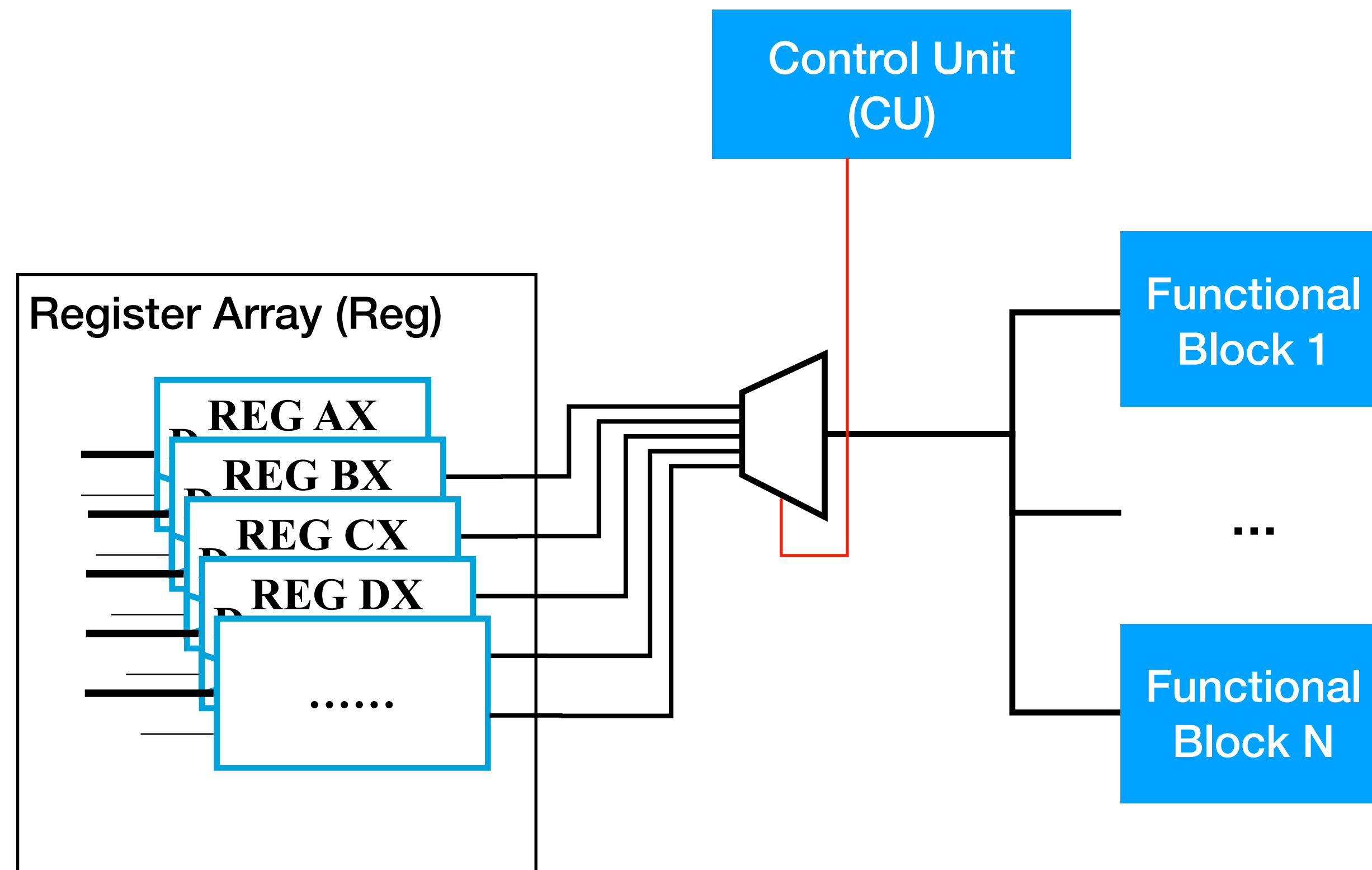


# Single Register Microoperations

Red: Address

Green: Mode

- **Select** Input REG to be fed into Processing Blocks
- Select **mode** for Processing Blocks



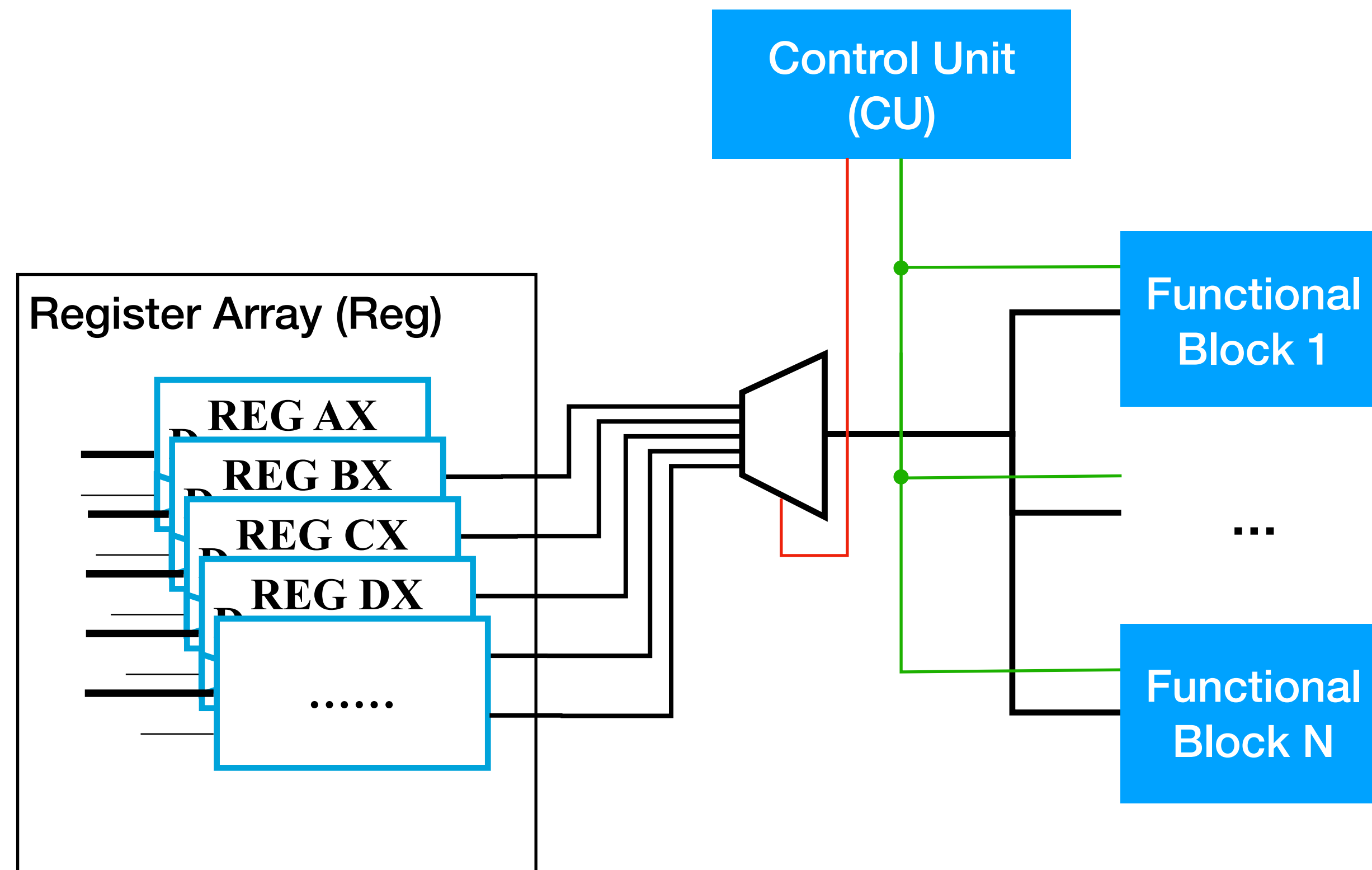


# Single Register Microoperations

Red: Address

Green: Mode

- **Select** Input REG to be fed into Processing Blocks
- Select **mode** for Processing Blocks

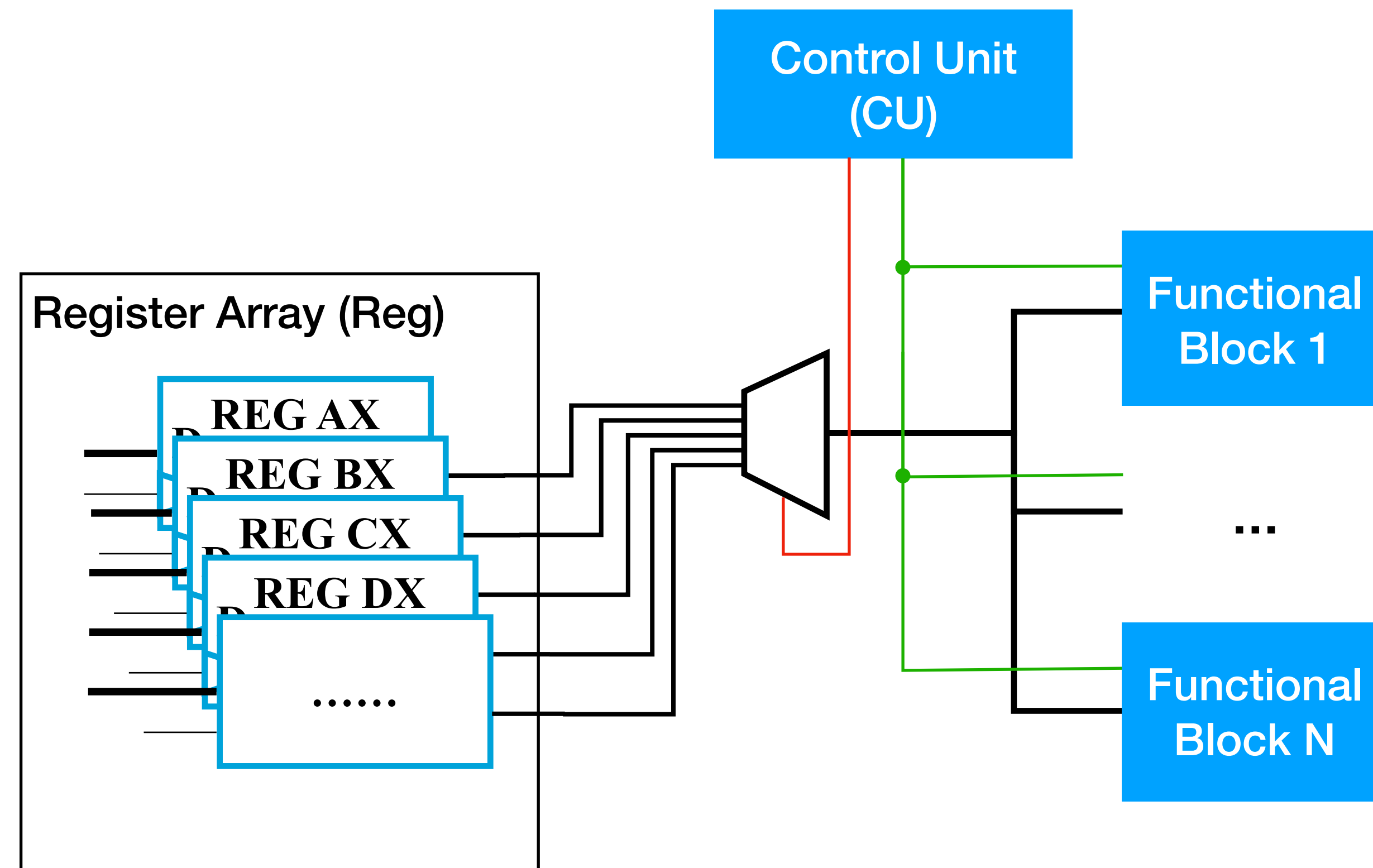


# Single Register Microoperations

**Red:** Address

**Green:** Mode

- **Select** Input REG to be fed into Processing Blocks
- Select **mode** for Processing Blocks
- **Select** Processing Block for output to REG

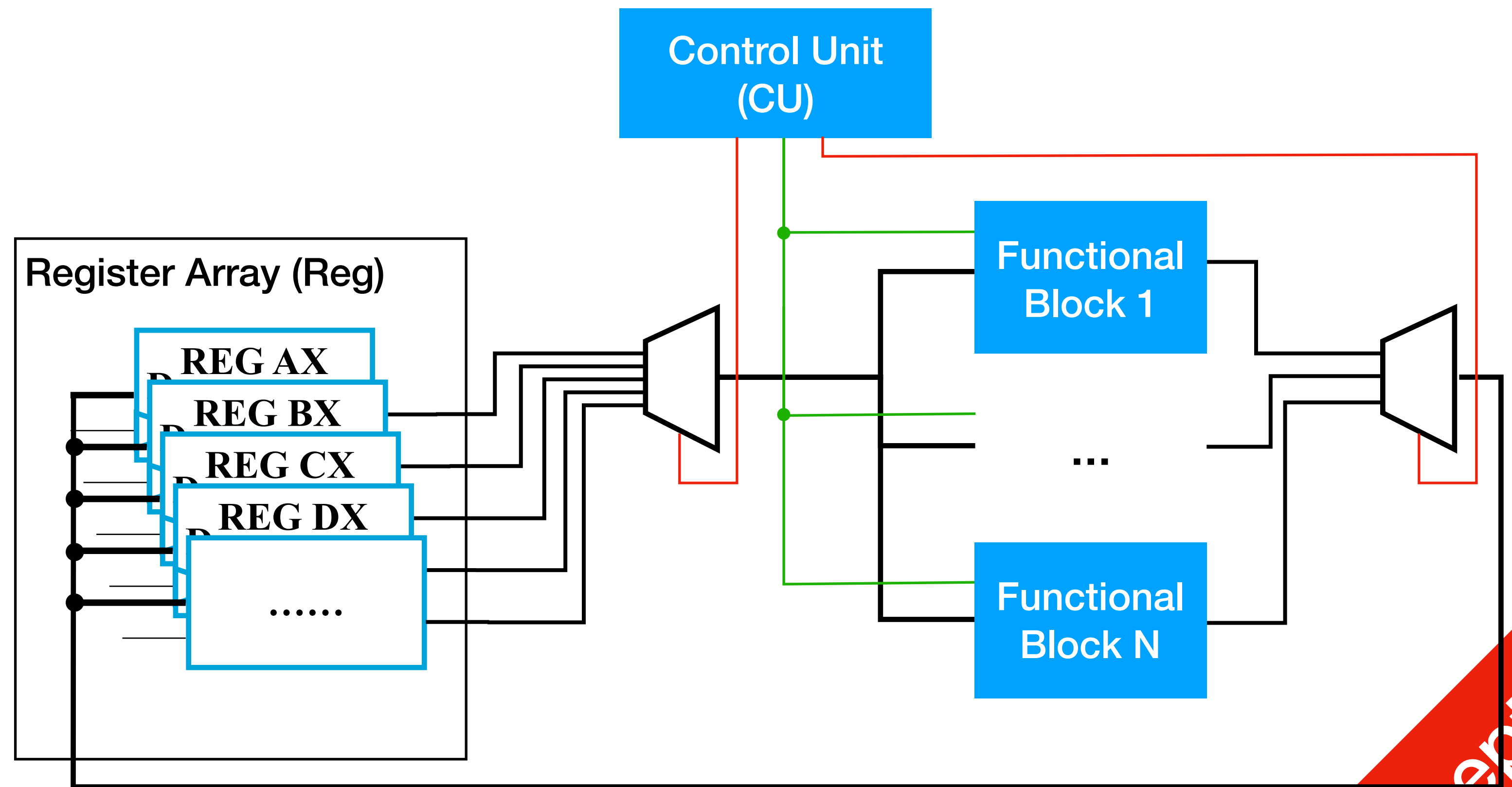


# Single Register Microoperations

Red: Address

Green: Mode

- **Select** Input REG to be fed into Processing Blocks
- Select **mode** for Processing Blocks
- **Select** Processing Block for output to REG

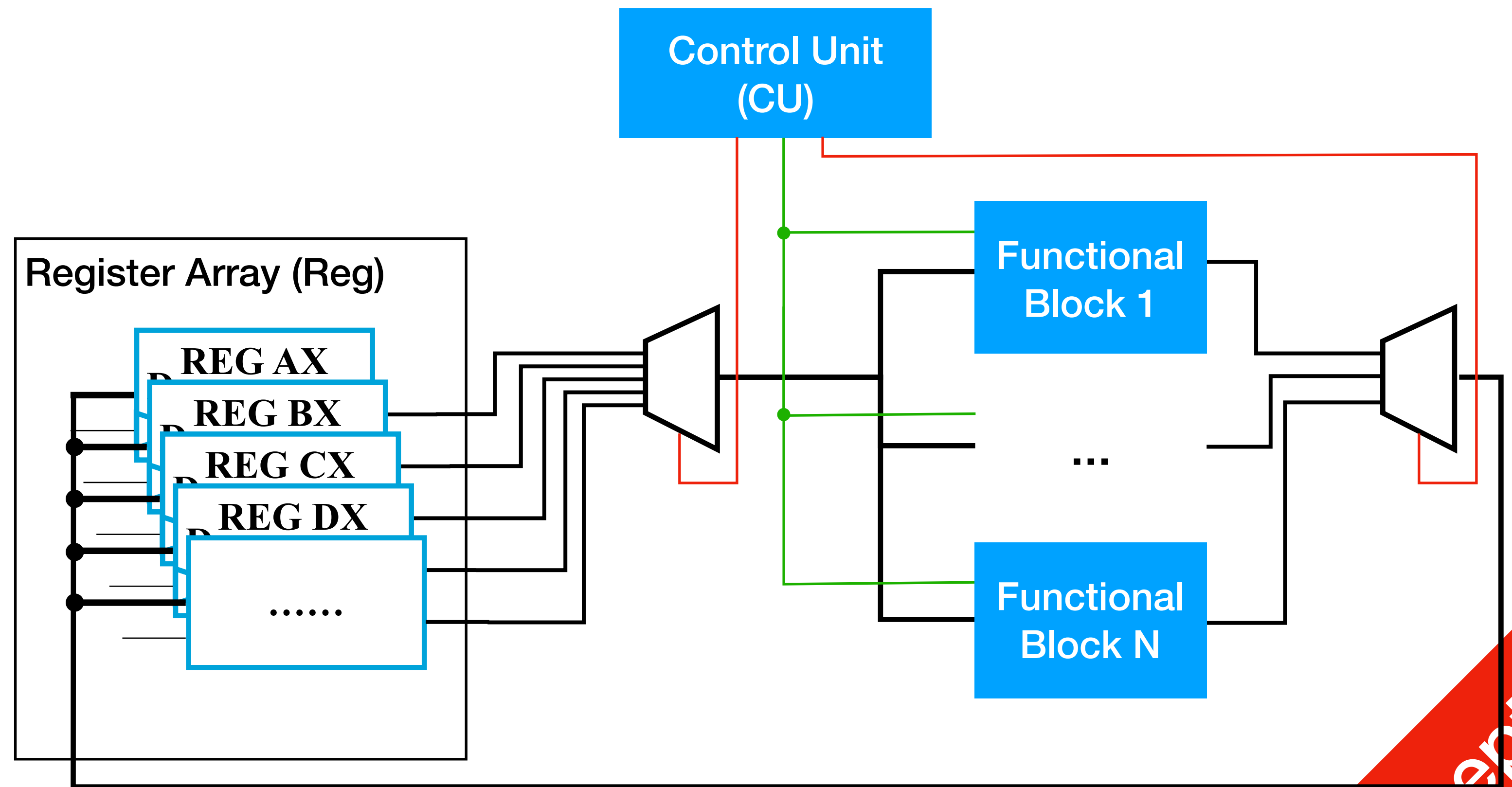


# Single Register Microoperations

Red: Address

Green: Mode

- **Select** Input REG to be fed into Processing Blocks
- Select **mode** for Processing Blocks
- **Select** Processing Block for output to REG
- **Select** Receiving REG

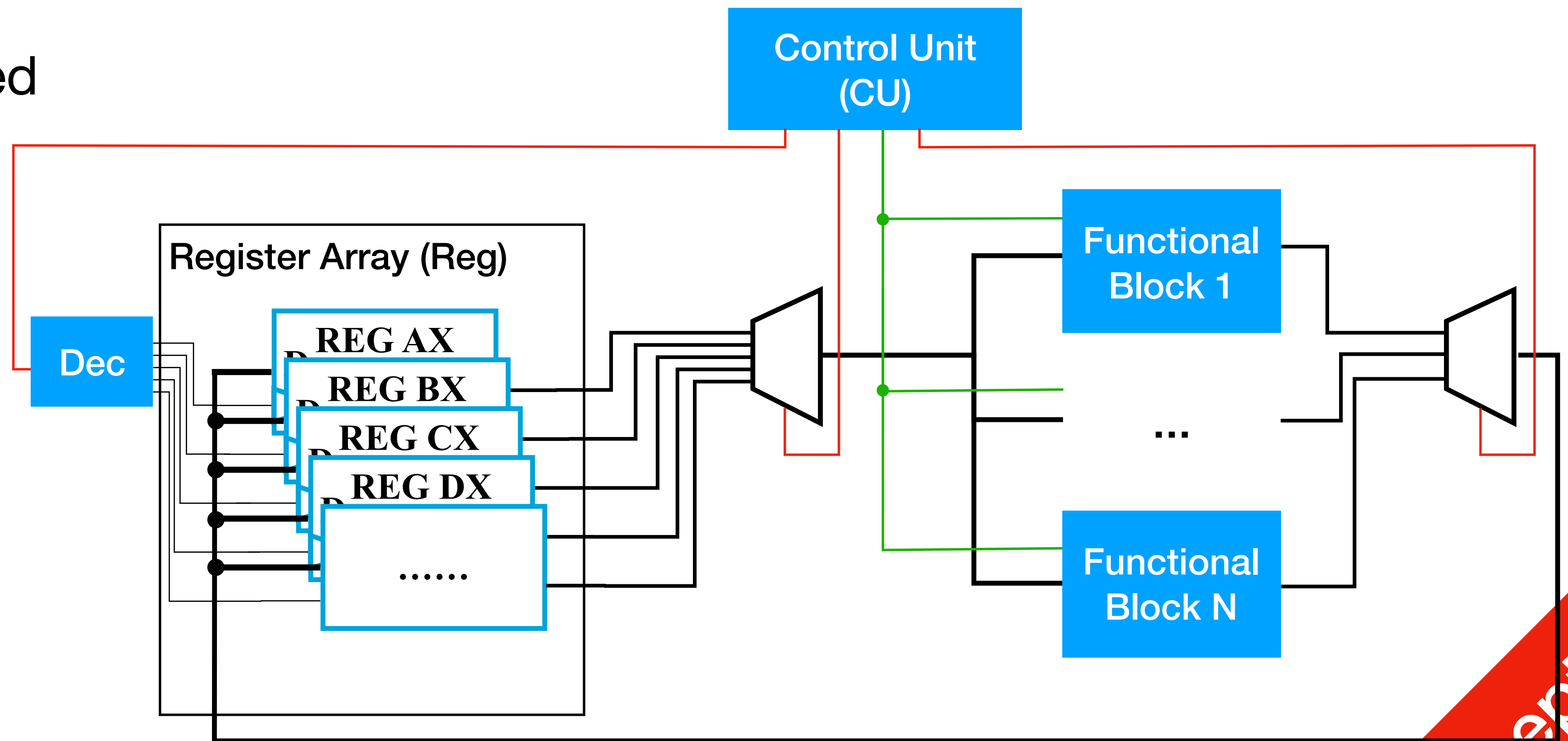


# Single Register Microoperations

Red: Address

Green: Mode

- **Select** Input REG to be fed into Processing Blocks
- Select **mode** for Processing Blocks
- **Select** Processing Block for output to REG
- **Select** Receiving REG



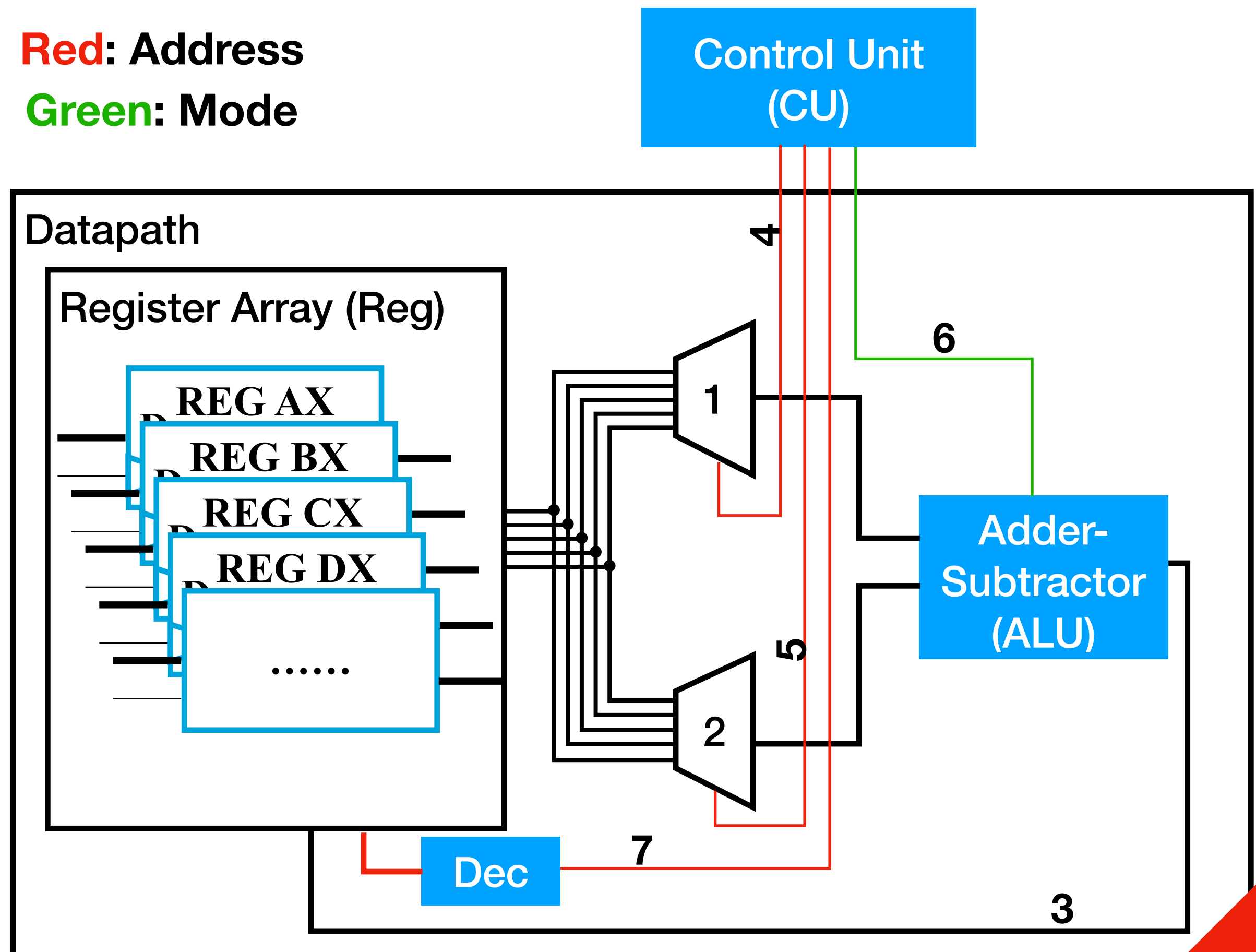
# Multiple Register Microoperations

	Operator	Example		Operator	Example
Assignment	<code>&lt;=</code>	<code>ax &lt;= 12h</code>	Bitwise AND	<code>and</code>	<code>ax and bx</code>
Reg. Transfer	<code>&lt;=</code>	<code>ax &lt;= bx</code>	Bitwise OR	<code>or</code>	<code>ax or bx</code>
Addition	<code>+</code>	<code>ax + bx</code>	Bitwise NOT	<code>not</code>	<code>not ax</code>
Subtraction	<code>-</code>	<code>ax - bx</code>	Bitwise XOR	<code>xor</code>	<code>ax xor bx</code>
Shift Left	<code>sll</code>	<code>ax sll 2</code>	Vectors		<code>ax(3 down to 0)</code> <code>ax(3 down to 0)</code>
Shift Right	<code>srl</code>	<code>ax srl 2</code>	Concatenate	<code>&amp;</code>	<code>ax(7 down to 4)</code> <code>&amp;ax(3 down to 0)</code>

Concept

# Multiple Register Microoperations

- Extension: how to incorporate more multiple register functional blocks?
- Extension: how to incorporate more single register functional blocks?

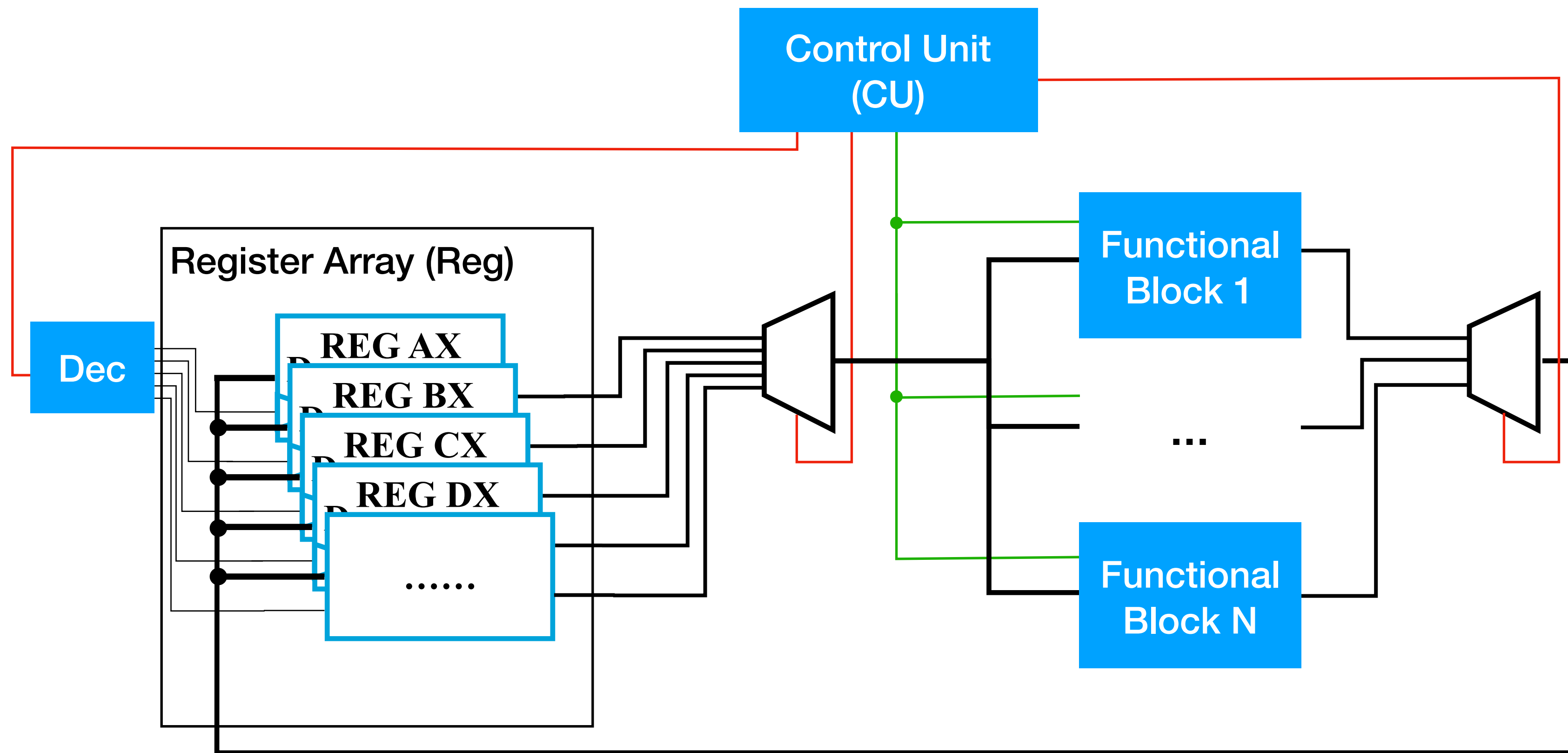


# Example Datapath Question 1

- The control unit selects between 4 operations A, B, C, D.
- A, B, C, D each takes 1/1/2/2 register(s) as input and outputs to 1 register
- A, B, C, D doesn't have additional mode selections
- Specify the control unit interface to the datapath
- Draw the circuit diagram of such datapath with a register array

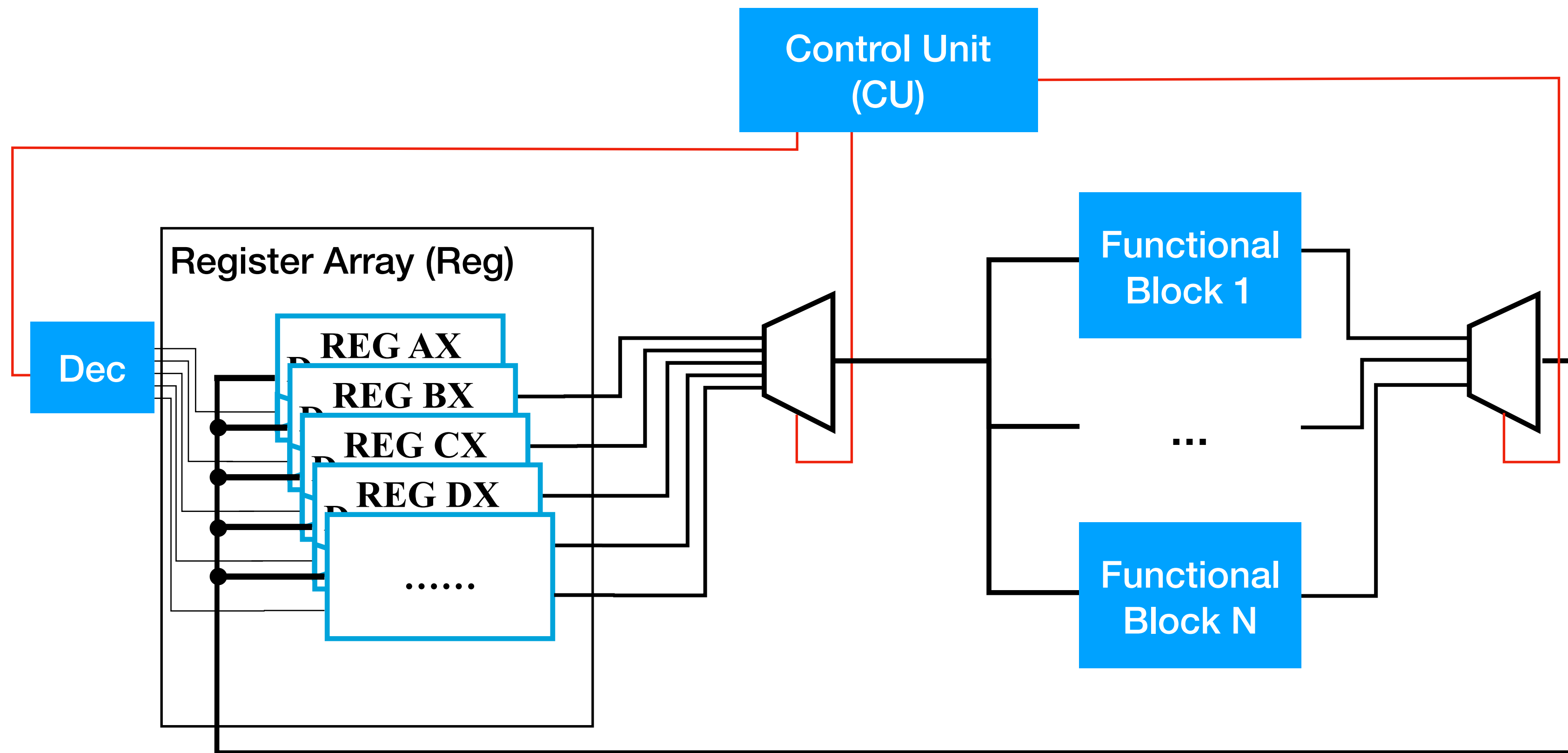


# Example Datapath Question 1

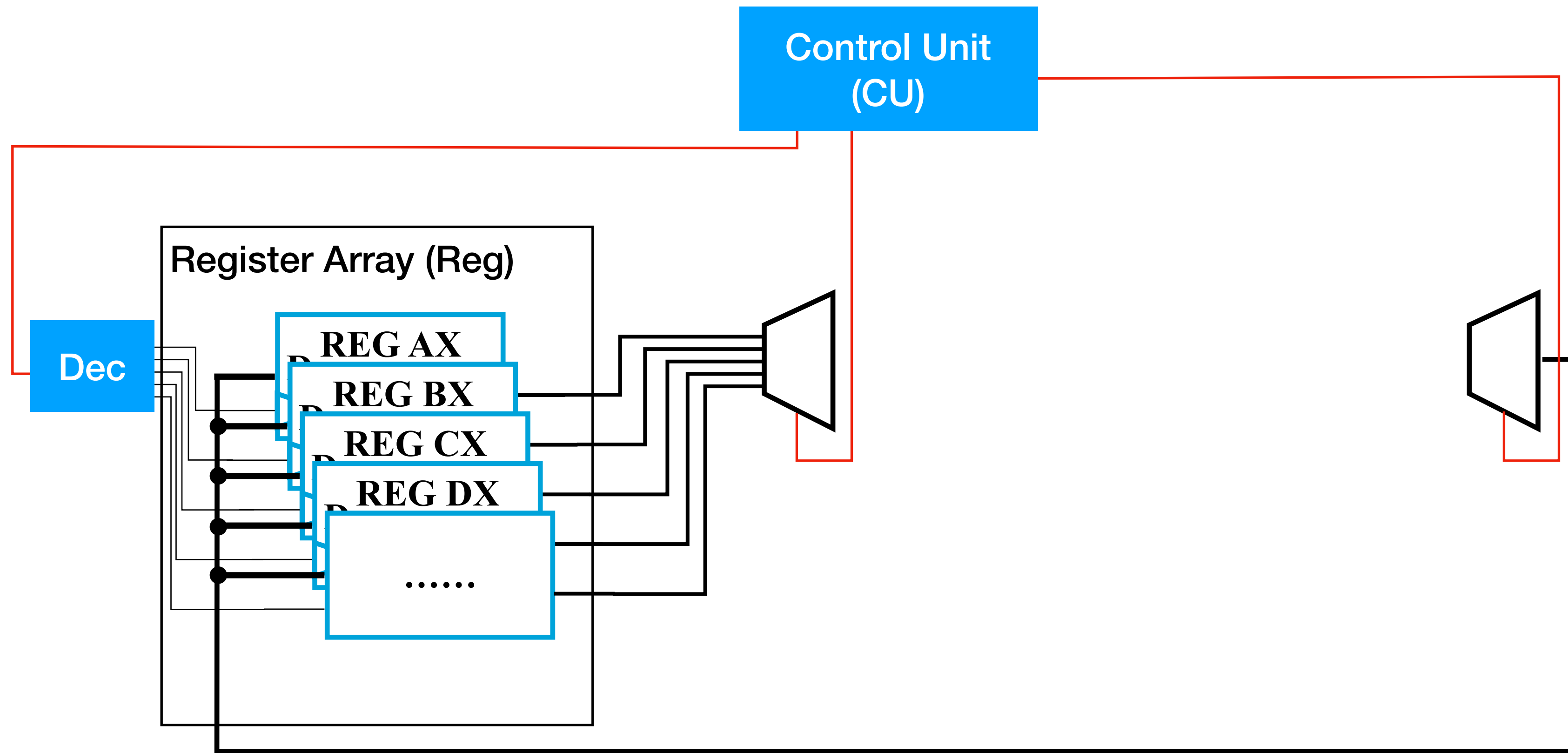


Exercise

# Example Datapath Question 1

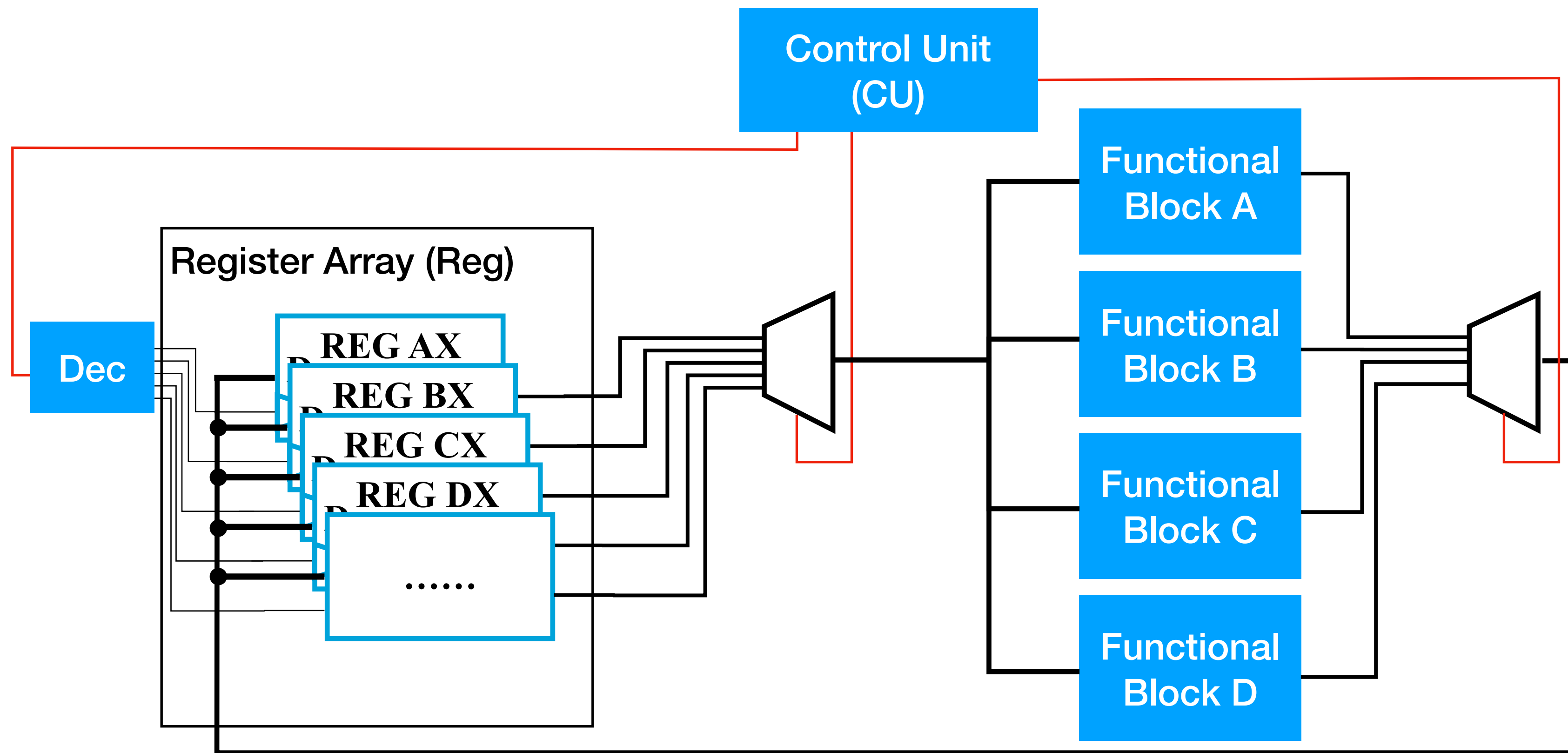


# Example Datapath Question 1



Exercise

# Example Datapath Question 1



Exercise

# Example Datapath Question 1

