



27.07.20 13:42

# CSCI 150

## Introduction to Digital and Computer System Design

### Lecture 5: Registers III



Jetic Gū  
2020 Summer Semester (S2)

# Overview

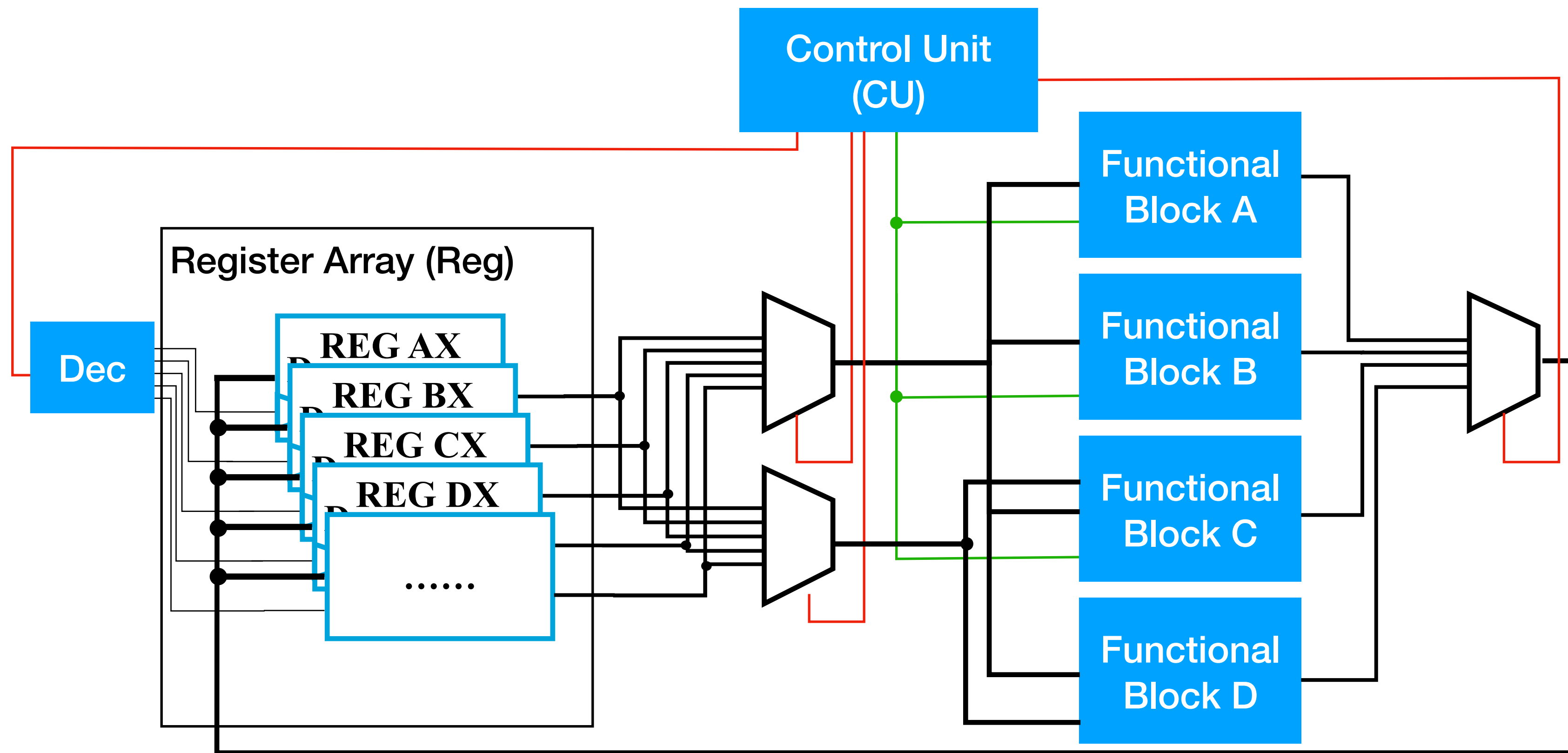
- Focus: Fundamentals of Complex Digital Circuit Design
- Architecture: von Neumann
- Textbook v4: Ch7 7.6; v5: Ch6 6.6
- Core Ideas:
  1. Implementation of Register Microoperations

# Register Transfer Operations

	Operator	Example
Assignment	<code>&lt;=</code>	<code>ax &lt;= 12h</code>
Reg. Transfer	<code>&lt;=</code>	<code>ax &lt;= bx</code>
Addition	<code>+</code>	<code>ax + bx</code>
Subtraction	<code>-</code>	<code>ax - bx</code>
Shift Left	<code>sll</code>	<code>ax sll 2</code>
Shift Right	<code>srl</code>	<code>ax srl 2</code>

	Operator	Example
Bitwise AND	<code>and</code>	<code>ax and bx</code>
Bitwise OR	<code>or</code>	<code>ax or bx</code>
Bitwise NOT	<code>not</code>	<code>not ax</code>
Bitwise XOR	<code>xor</code>	<code>ax xor bx</code>
Vectors		<code>ax(3 down to 0)</code> <code>ax(3 down to 0)</code>
Concatenate	<code>&amp;</code>	<code>ax(7 down to 4)</code> <code>&amp;ax(3 down to 0)</code>

# Example Datapath Architecture



# Implementation of Datapath II

Individual Functional Blocks

# Single Register Microoperations

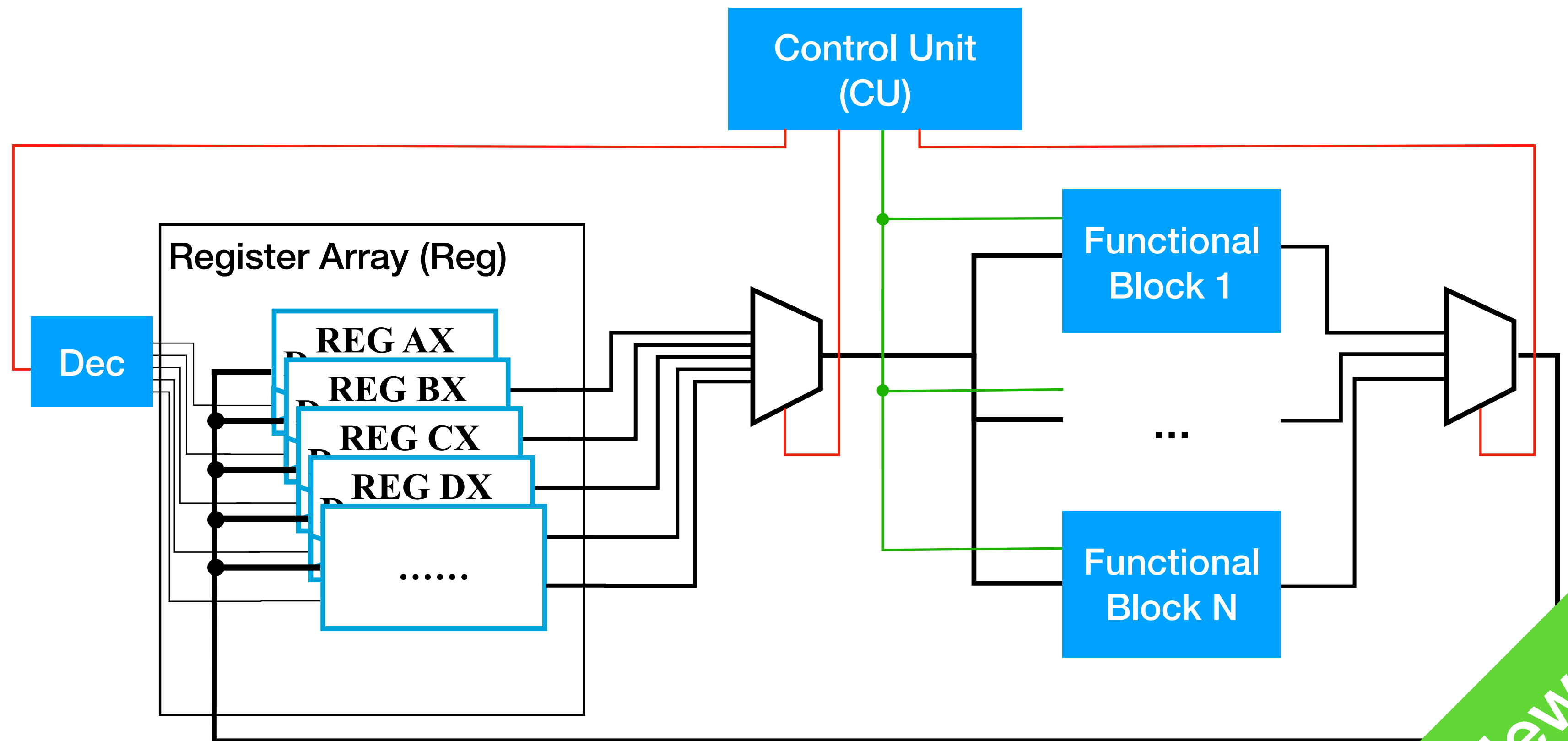
	Operator	Example		Operator	Example
<b>Assignment</b>	<code>&lt;=</code>	<code>ax &lt;= 12h</code>	<b>Bitwise AND</b>	<code>and</code>	<code>ax <b>and</b> bx</code>
<b>Reg. Transfer</b>	<code>&lt;=</code>	<code>ax &lt;= bx</code>	<b>Bitwise OR</b>	<code>or</code>	<code>ax <b>or</b> bx</code>
<b>Addition</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>

# 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



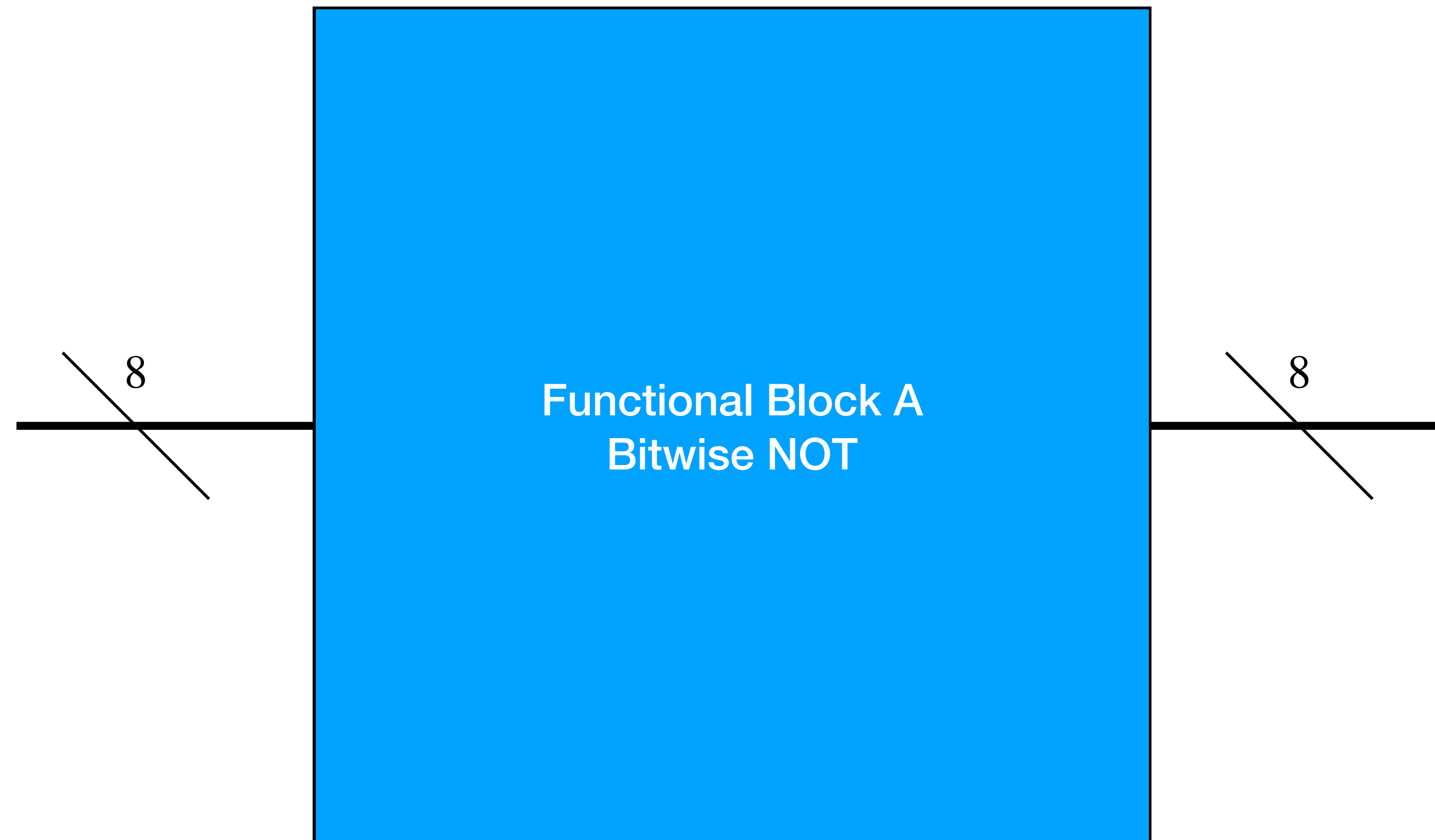
# Single Register Microoperations

- Each individual functional block design identical to the functional blocks we've covered in Lecture 3
- For operations completed in a single CLK tick, use the 5 step design procedures



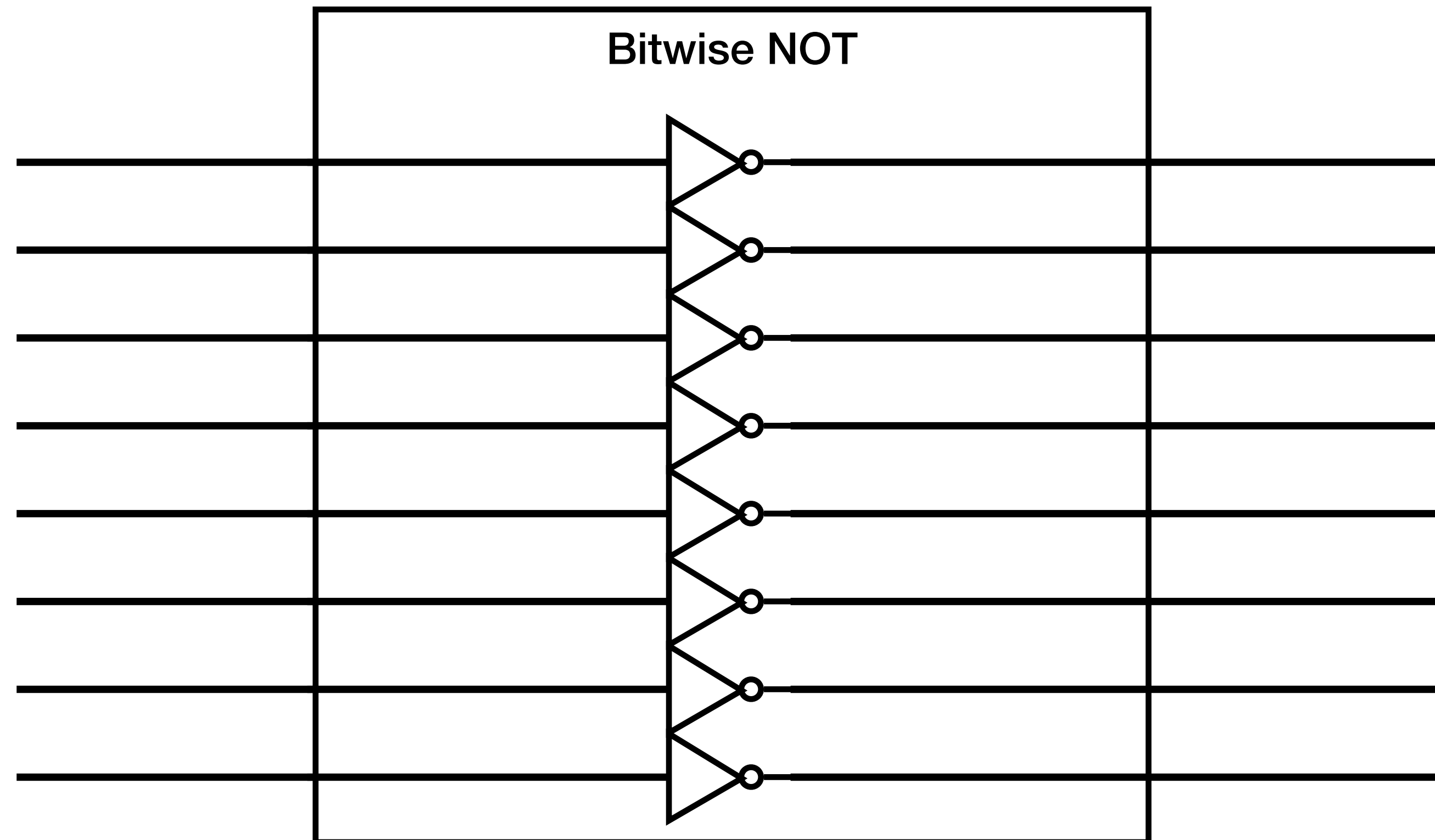
# Single Register Microoperations

- Bitwise NOT



# Single Register Microoperations

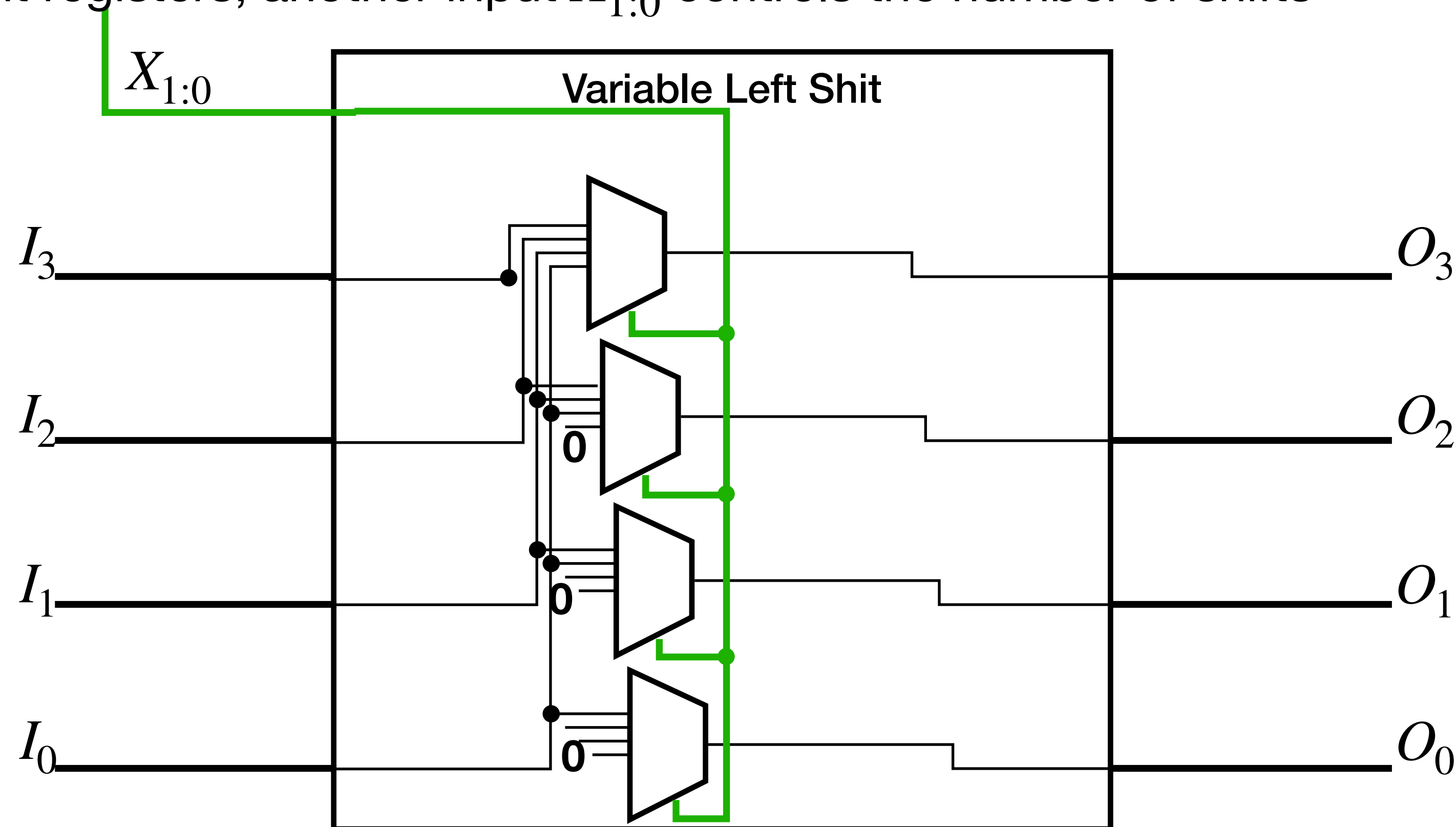
- Bitwise NOT



# Single Register Microoperations

- Left Shifter: variable shifts

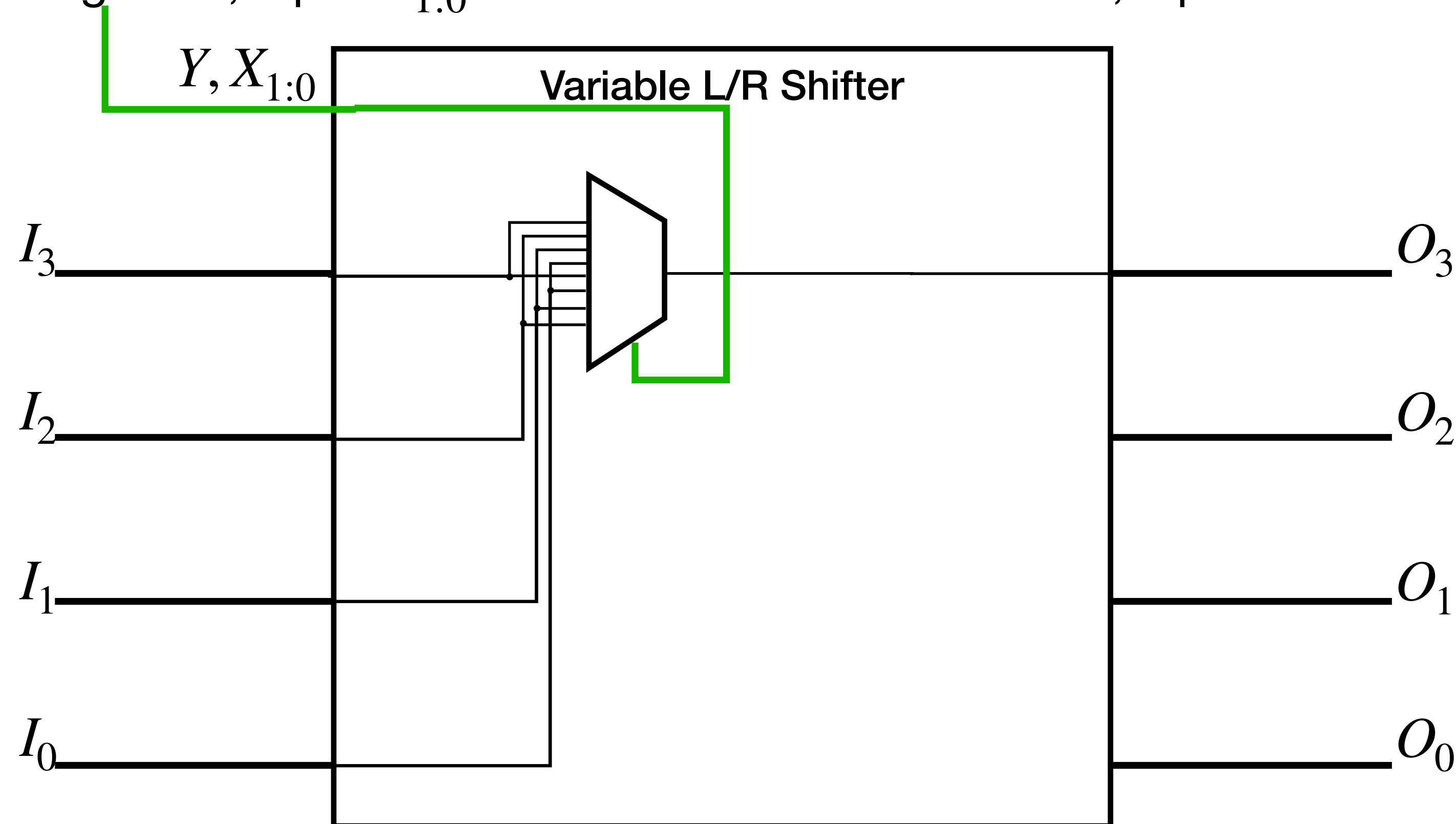
Assuming 4-bit registers, another input  $X_{1:0}$  controls the number of shifts



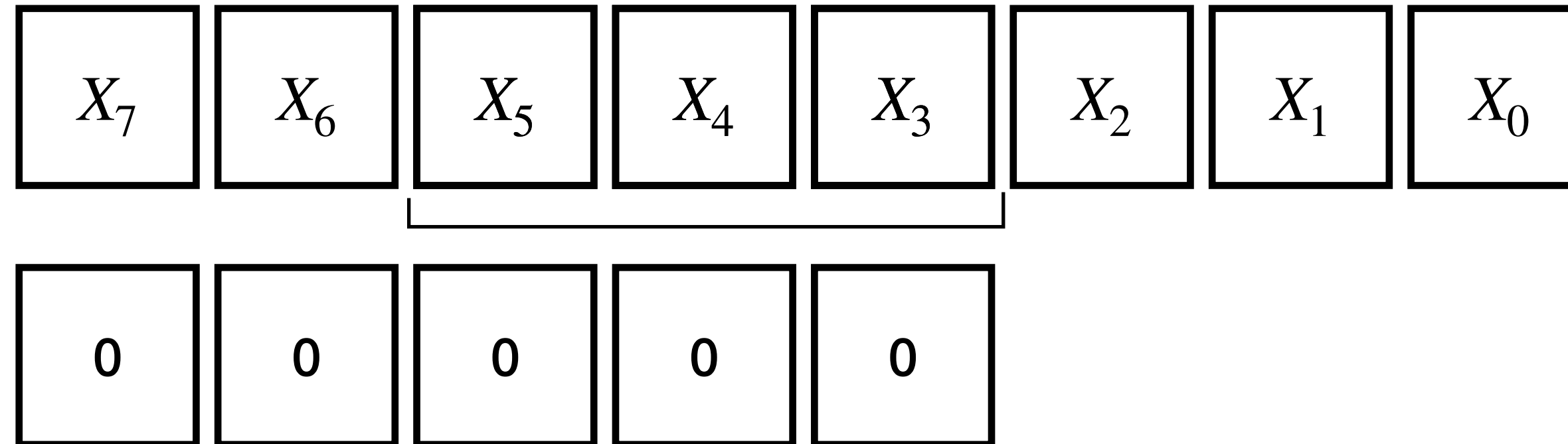
Exercise

# Single Register Microoperations

- Left/Right Shifter: variable shifts  
Assuming 4-bit registers, input  $X_{1:0}$  controls the number of shifts, input  $Y$  for direction (0:L; 1:R)



# Single Register Microoperations



- Vector:  $I$  down to  $J$  (e.g.  $\text{ax}(5 \text{ down to } 3) : I = 5, J = 3$ )
- Decompose the problem
- $J$ : the number of right shifts to perform
- $I$ : masking out everything before position  $I$  to 0

# Multiple Register Microoperations

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

Concept