# CSCE312 : LAB 4

Prof. Eun Jung Kim

Spring 2024

# Groups

- Form groups of 3 (2 is also okay) and mention your team details here : GoogleDocs
- Lab 4 Deadlines:
  - **Demonstration Deadline – March 7th 2024 (Thursday)**
  - **Lab Report Deadline – March 10th 2024 (Sunday)**

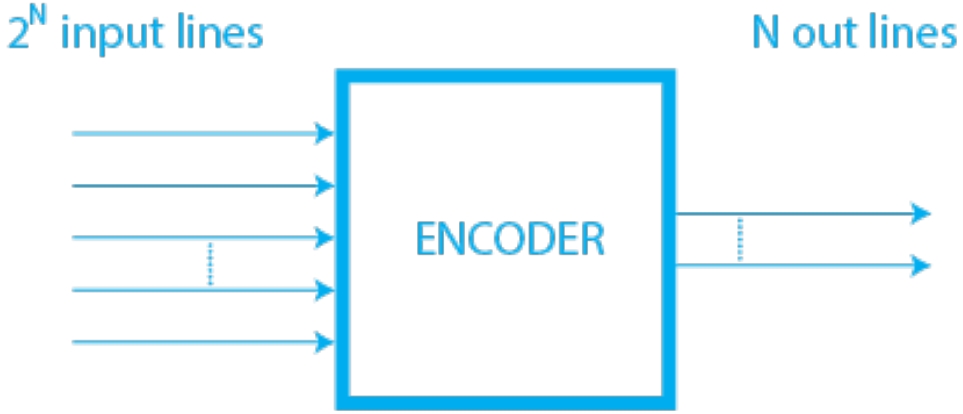
# Groups

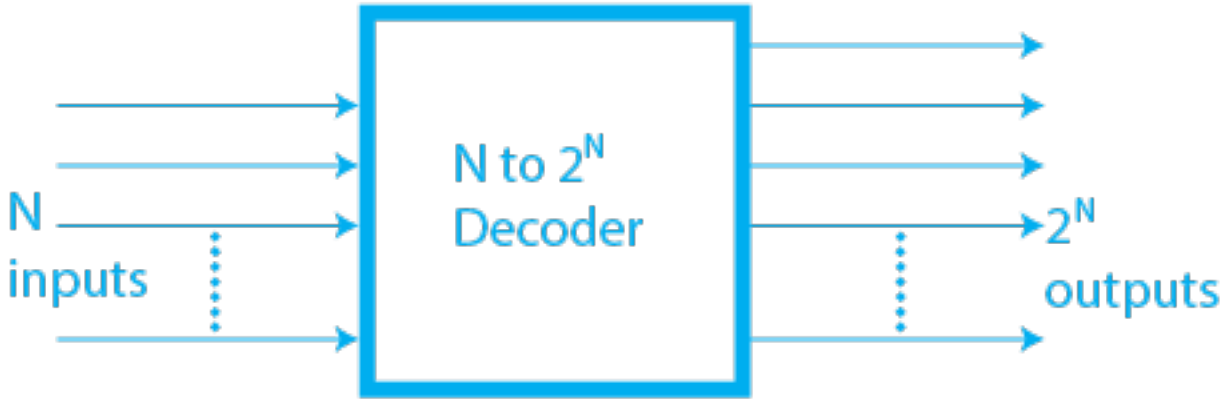
- Which section to attend?
  - If all your group members are in the same section, then attend your respective section's lab session.
  - If your group has people from different sections, decide which section you can attend as a group after going through conflicts, etc.
  - If you are having problems let the TAs know.
- How to submit as a Group?
  - One person from the group may submit the Group submission (Including group report and Logisim files)
  - Everyone must submit the individual report.

# Reminder

- You are free to use the built-in Decoders, Encoders, Multiplexers, Demultiplexers, Registers, etc. available on Logisim unless the problem explicitly mentions not to use some tool.
- Do not spend your time building these from scratch as you have already done this in previous labs.
- Break your design into smaller components and make use of subcircuits to have clean and easy to understand circuits.

# Encoders/Decoders



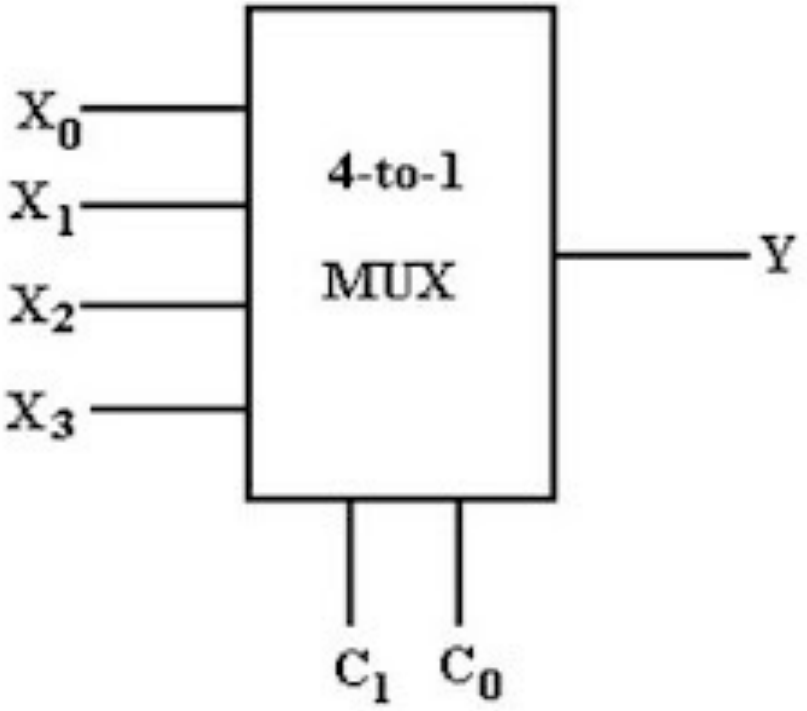

**Encoder**

- Input : $2^n$ one-bit inputs
- Output : n bits that represent the address of the line that is on in binary

**Decoder**

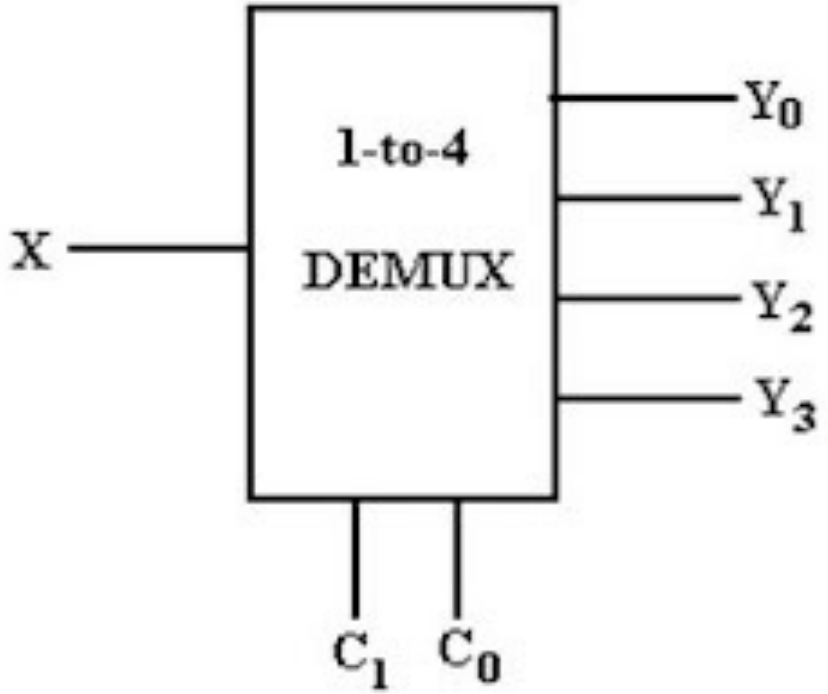
- Input : n one-bit inputs representing a binary number “b”
- Output : $2^n$ lines, where the $b^{th}$ line is on

# MUX/DEMUX




**Multiplexer**

- Input : $2^n$ lines and n select bits
- Output : Based on the address represented by the n select bits, one of the input lines is chosen as the output Y




**Demultiplexer**

- Input : 1 line and n select bits
- Output : Based on the address represented by the n select bits, one of the output lines is chosen and value of X goes there

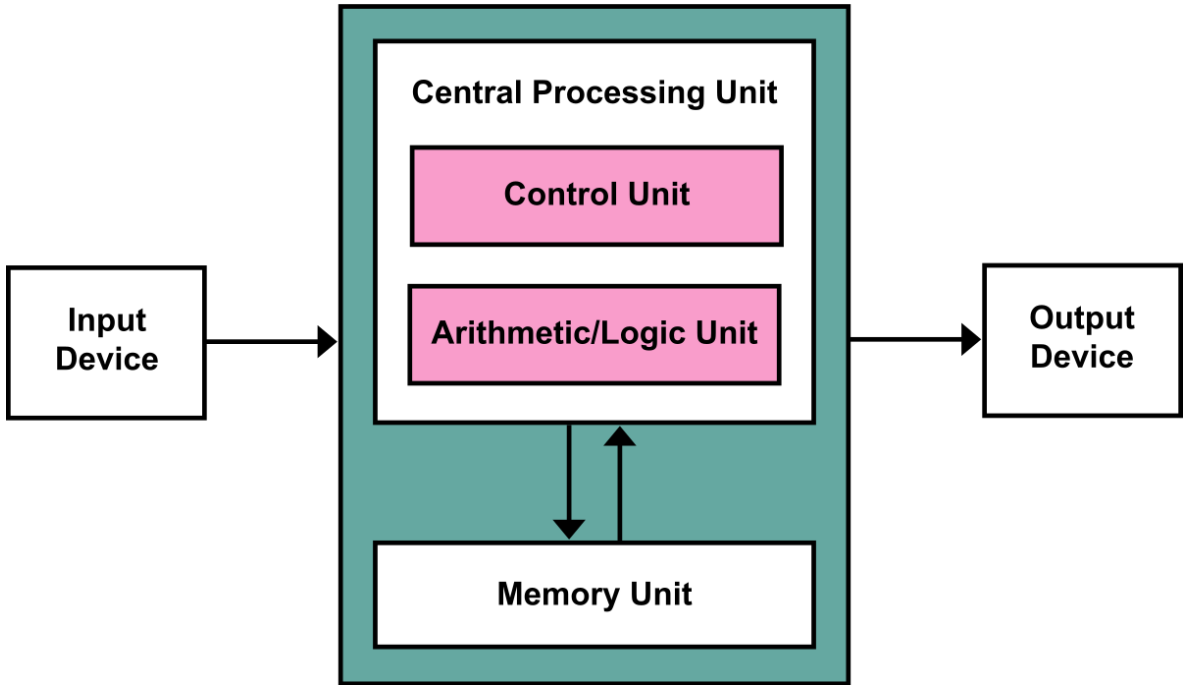
# Lab4 : Problem 1 (70 points)

- Start building a microprocessor.
- Now, you will just build the peripherals, i.e. everything apart from the CPU.
- IO interface circuitry.
- Address/Data/Control Buses
- Memory Units

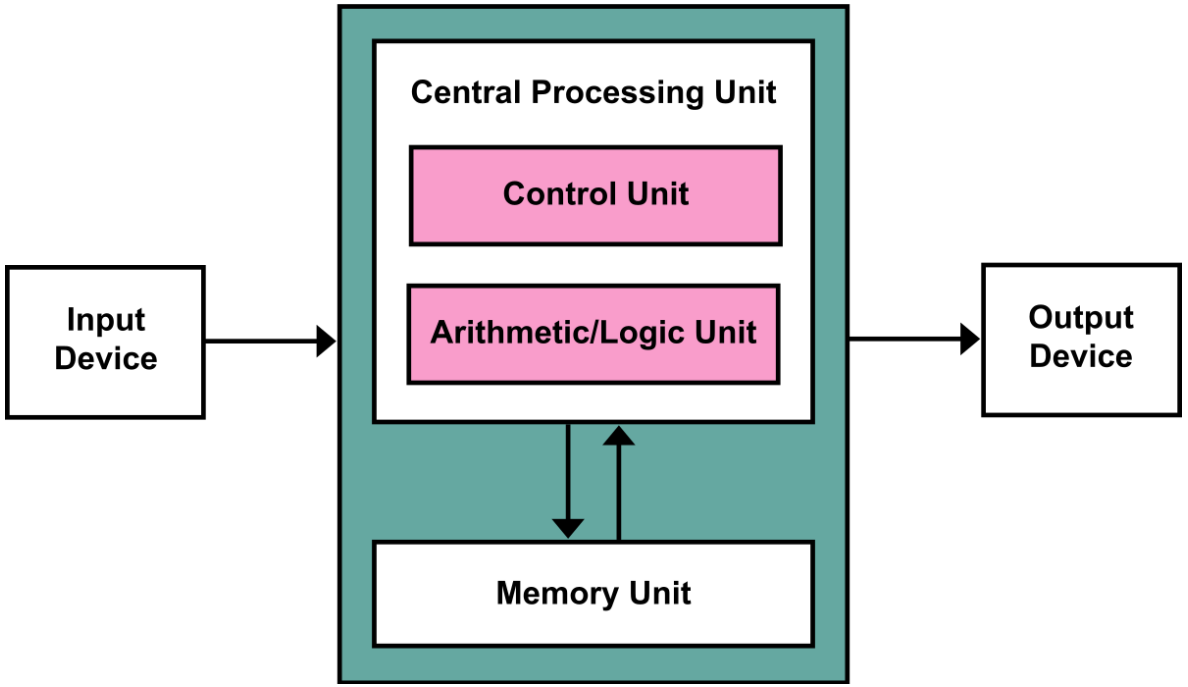
# Von Neumann Architecture

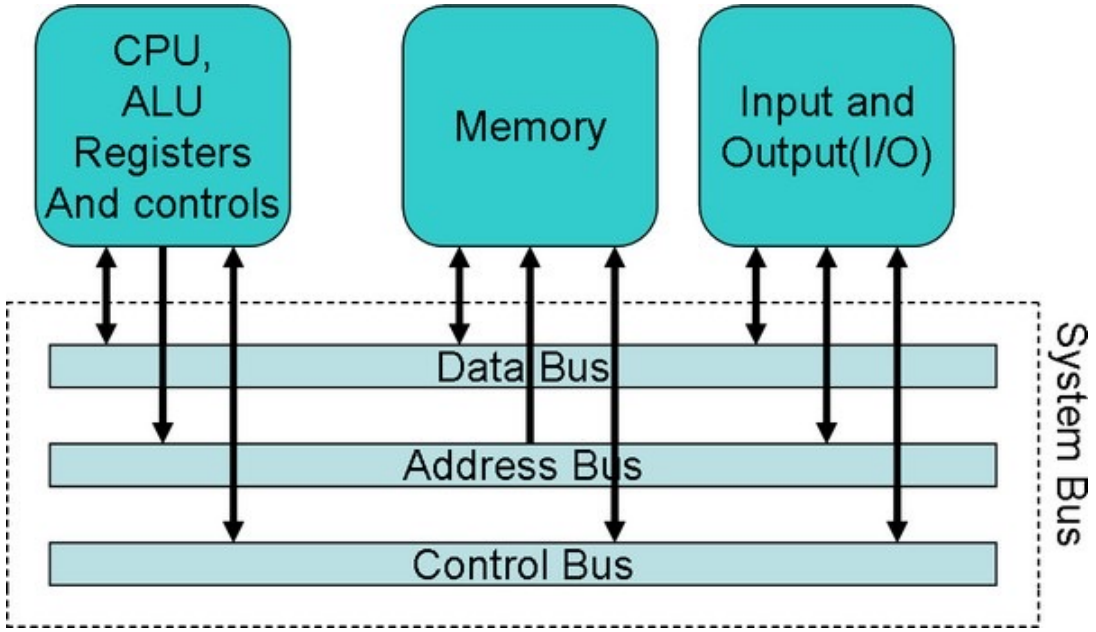
A design format for computers.

- CPU : performs arithmetic, logical and control operations.
- Memory : place where computer stores and retrieves data from.
- I/O devices : interface to external world.
- Buses : connections between other components that enable appropriate data transfer.



# Von Neumann Architecture



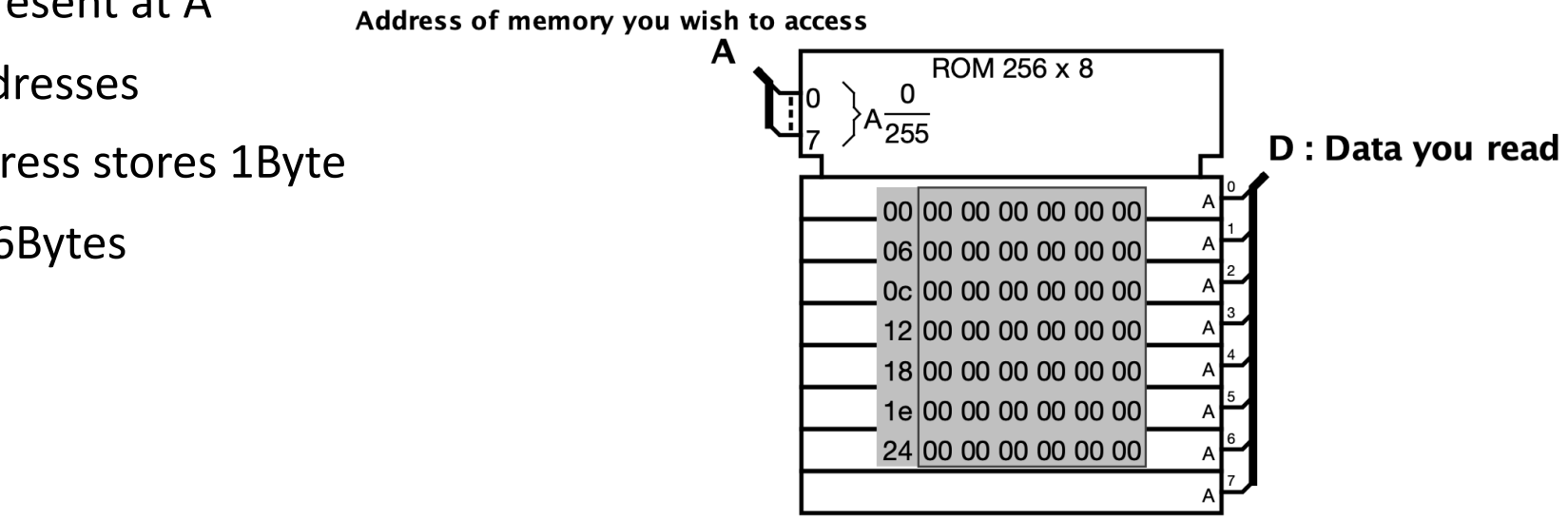



# Components

- CPU
  - performs arithmetic, logical and control operations.
  - Intel 8088, Intel Pentium, AMD Ryzen, Apple M1, etc.
- Memory
  - place where computer stores and retrieves data from.
  - SRAM, DRAM, ROM, Caches, Registers, Hard Disc, SSD, etc.
- I/O devices
  - interface to external world.
  - Mouse, Keyboard, Monitors, Speakers, etc.
- Buses
  - connections between other components that enable appropriate data transfer.
  - Universal Serial Bus (USB), UART, SPI, Ethernet, Direct Memory Access (DMA), etc.

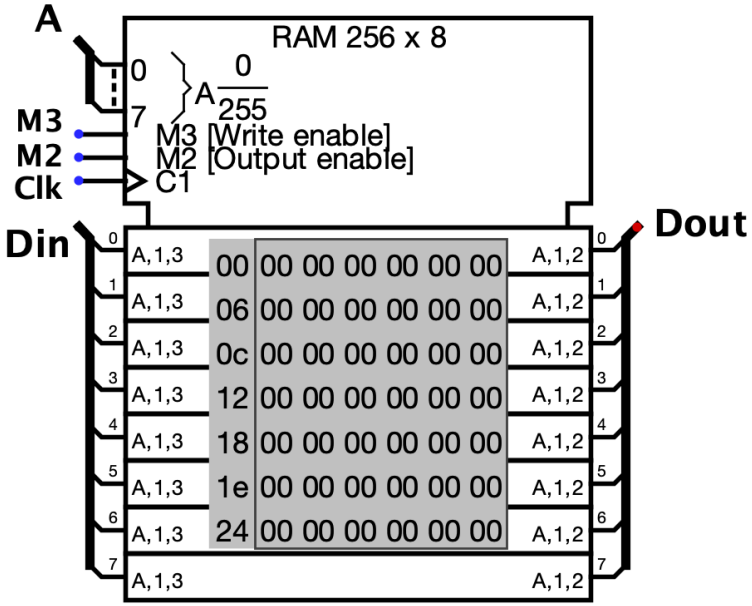
# Memory - ROM

- Read Only Memory
- Non-Volatile memory (means it is not lost when the computer switches off)
- **A[0:7]** Address you want to read
- **D[0:7]** Data you retrieve present at A
- 8 bits for address => $2^8$ addresses
- 8 bits for data => each address stores 1Byte
- Size of ROM = $2^8$ x 1B = 256Bytes



# Memory - RAM

- Random Access Memory
- Generally volatile memory (means it is lost when the computer switches off)
- **A[0:7]** Address you want to read/write
- **M3[1 bit]** Enabled if you wish to write
- **M2[1 bit]** Enabled if you wish to read
- **Clk** Clock input
- **Din[0:7]** Data you send when writing at A
- **Dout[0:7]** Data you receive when reading from A
- 8 bits for address => $2^8$ addresses
- 8 bits for data => each address stores 1Byte
- Size of ROM = $2^8$ x 1B = 256Bytes

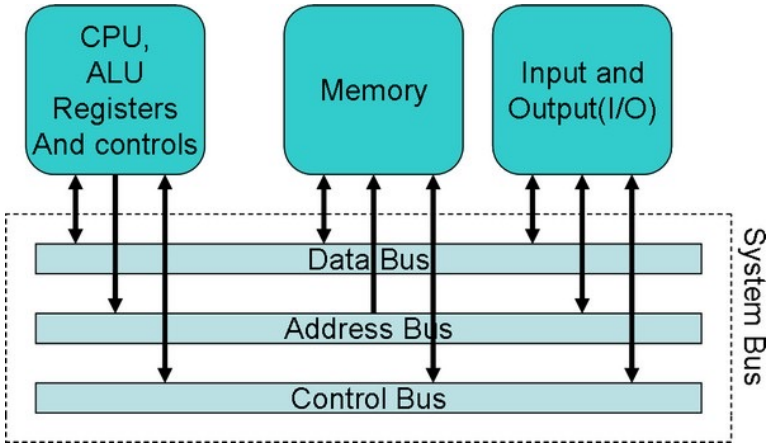


# Buses

CONTROL BUS: The common signals that are transferred from CPU to devices are memory read, memory writes, I/O read, I/O write. (what to do?)

ADDRESS BUS: The address bus is used by the CPU to send the address of the memory location or the input/output port that is to be accessed at the instant. (where to get data from/make changes?)

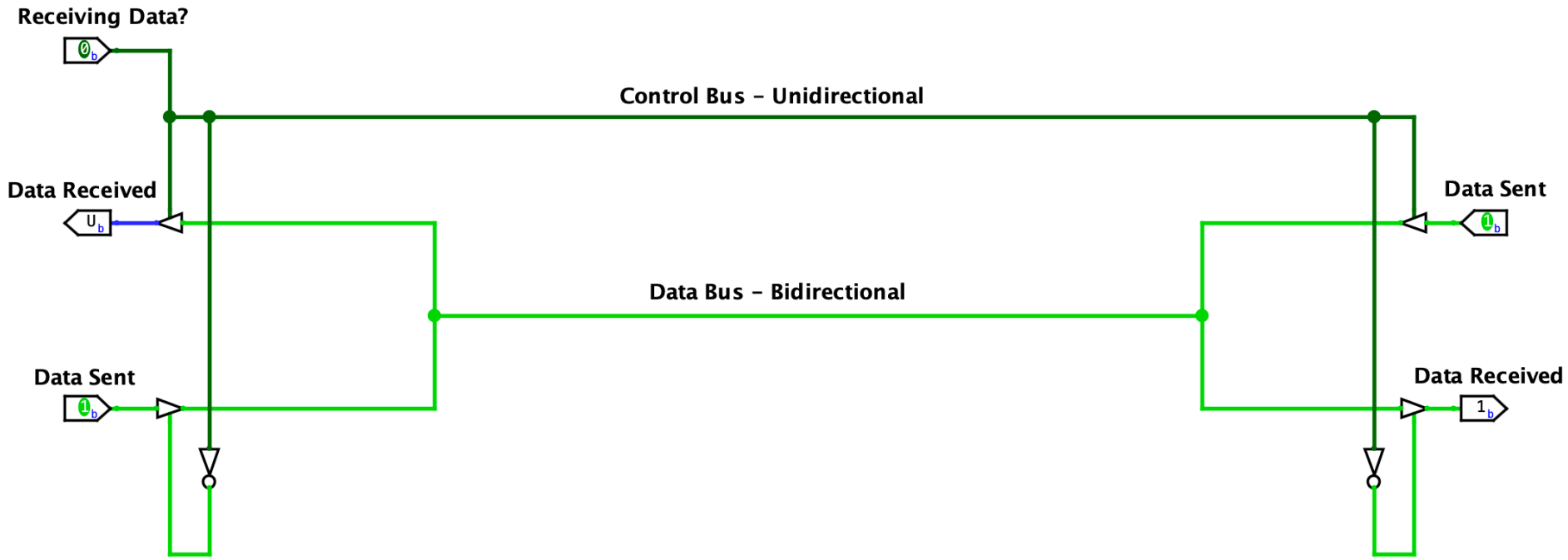
DATA BUS: A data bus is used to carry the data and instructions from the CPU to memory and peripheral devices and vice versa. (what to send/get?)



# Buses

Unidirectional Buses : Data flows in only one way

Bidirectional Buses : Based on some condition, the same bus can be used to either send or receive data. Data may flow in any of the two directions.



# Lab 4 : Problem 1

1. Design a circuit following the Von Neumann Architecture that can do the following:

- read from ROM
- read from RAM
- write to RAM
- read from I/O device
- write to I/O device

2. Store the data value transferred via data bus in the data buffer to see how values are getting transferred.

2. Prepare timing diagrams for 10 different different configurations, testing the working of the previous parts. (2 tests per part)

# Lab 4 : Problem 1

How to build the circuit?

- Build the CPU (not now, in the project), so assume that you have the appropriate signals available.
- Build the Memory Unit (consisting of one ROM and one RAM)
- Build the I/O Unit
- Use the control signals to figure out which of the 5 operations you need to do. (Build a Decoder)
- Connect them all using the appropriate buses
- Depending on the output of decoder, send read/write enable, address and data(if required) to the Memory Unit or I/O Unit and get data from the unit back to the data bus (if required)
- Whenever you try to write data or read data, you need to store that data in a Register (called the Data Buffer) so that you can use this stored data again if you need it in the next operation.

# CPU

- You do not need to build this now, assume that you have all the signals from CPU available.
- So you already have these signals from the CPU:
  - (output of CPU) Control Signal (8 bits)

| | |
|---|---|
| read from ROM | 00000001 |
| read from RAM | 00000010 |
| write to RAM | 00000011 |
| read from Input Device | 00000100 |
| write to Output Device | 00000101 |

  - (output of CPU) Data _w (8 bits) : Data to be written to memory/output device
  - (input of CPU) Data_r (8 bits) : Data to be read from memory/input device
  - (output of CPU) Address Value (8 bits) : Address of ROM/RAM byte or address of I/O device
  - (input for CPU/the whole microprocessor) Enable : If it is 1, then the whole system works, else the whole system does not work.

# Memory Unit

Build a unit that has one RAM and one ROM inside it and that has these signals:

- Clock
- Control Signal (to determine if you do something related to memory, and if yes do you read from ROM? Do you read from RAM? Or do you write to RAM?)
- Address value (where to read/write from)
- Data_in (what to write if you are writing something)
- Data_out (the data you read from ROM/RAM)

# I/O Unit

- The I/O Unit has:
  - 4 input devices taking 8-bit values and numbered from 00000001 to 00000100
  - 4 output devices sending 8-bit values and numbered from 00000101 to 00001000
- You take 8-bit input data using the input devices and you give back 8 bit data output using the output devices.
- How to implement?
  - You take four 8-bit inputs from the input devices and send it to the CPU
  - You have four 8-bit output devices (Lab 3 Problem 3) which take data from your CPU
  - If your operation is to read from I/O then you choose the correct Input device using the address and read data from that input and send it to the data buffer.
  - If your operation is to write to the I/O then you find the correct Output Device using the address and write to it the data you obtain from the CPU.

# Connections In Buses + Data Buffer

- If it is a **READ**, send address of location to the appropriate unit and get back read data from that unit.
- If it is a **WRITE**, send address and data to the appropriate unit.
- Make sure every time you try to read or write some data, you store it in the **data buffer** that stores whatever data gets passed via the data bus. If no data gets passed (example enable is off), this buffer should store the previous value that was sent via this bus.

# Data Buffer

- Implement using Register (8 bits) in Logisim.
- If current operation requires you to write, then first input the data to this buffer and then use the output of this buffer to send data through data bus to the units.
- If current operation requires you to read, then finish the read operation from unit and then input the data you just acquired into the data buffer.

# Lab 4 : Problem 1

Timing Diagrams:

- Build timing diagrams which test your built design for 10 different configurations.
- Each requirement (read ROM, read RAM, write RAM, read I/O, write I/O) needs to be tested twice.
- These cases should work completely on your design with appropriate values on all the units, inputs/outputs, data buffer, etc.
- Build timing diagrams for your processor that plots the change in all the different inputs/outputs of your circuit for the 10 different configurations.

# Lab 4 : Problem 2 (30 points)

- For 8-bit inputs:
  - Design an adder (2 input) that keeps track of overflow bit.
  - Design a subtracter – Using 2's complement (2 input)
  - Magnitude Comparator (2 input)
  - Left Barrel Shifter

- Refer to Lecture Slides to understand how to implement each of these.

# Lab 4 : Problem 3 [Bonus – 20 points]

- There is a single row of 25 LEDs and one input.
- If input is 0: The LEDS blink from Right to Left in sequence.
- If input is 1: The LEDS blink from Left to Right in sequence.
- Each LED should be ON for 0.25 seconds before it turns OFF and next LED turns ON.

- How to implement?
- Finite State Machine with 25 states.
- Rather than writing a very large truth table for each state transition and finding the complex Boolean logic, you can use a counter to increment state values one by one. You increment in case of input 0 and decrement in case of 1.
- After implementing, change the decoder to ROM and implement the same thing

# Lab 4 : Problem 3 [Bonus – 20 points]

You may use the built-in adder/subtractor/counter/decoder modules if you wish, or you could build the adder/subtractor from basic gates as required by problem 2 first and then re-use them to build your counter.

You may use the built-in register module.