



# RISC Helpsession



## Overview

- Moon1 vs. RISC
- Buses & Tri-states
- Using RAMs and ROMs
- Jumps
- Question and Answer



## Moon1 vs. RISC What's the Same?

- Single cycle
- Stores data and program in different parts
- Uses a control ROM
- Uses registers to store data
- Subtracts
- Compares
- Jumps



## Moon1 vs. RISC What's the Same?

- Bottom line: there's a lot of similarities!
- Make sure you understand Moon1 before working on RISC
- This will make your life much easier
- Of course there are a few differences...



## Moon1 vs. RISC So What's Different?

	<b>Moon 1</b>	<b>RISC</b>
<b>Registers</b>	1 Register (Accumulator)	4 Registers
<b>RAM</b>	Uses custom RAM component	Uses built-in RAM component*
<b>Use of Multiplexers</b>	Extensive	Limited
<b>Uses Buses</b>	No	Yes
<b>Jumps</b>	Absolute	Relative to PC
<b>Does Addition</b>	No	Yes

\* the RAM has the same control scheme as the RAM you built for HW 3



## Registers in RISC

- 4 registers – r0, r1, r2, r3 (00,01,10,11)
- Design changes with 4 registers:
  - Could use a bunch of multiplexers...
  - But this is confusing and difficult to control
  - Be lazy! Use buses instead



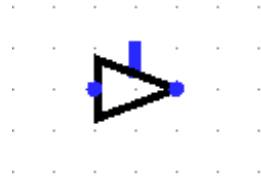
## Tri-States/Controlled Buffers

- **Parts**
  - Input data
  - Output data
  - Impedance control
- **High Impedance**
  - When impedance control is low (off), output data does not go through
  - When impedance control is high (on), output data does go through
- **Uses**
  - Making Buses



## Tri-States in Logisim

- Tri-States are called Controlled Buffers in Logisim. They look like this:



- The top input is the control
- If the control is high, value on each of the inputs on the left will be allowed through to the corresponding outputs on the right
- This might seem kind of weird, but it turns out to be really useful in RISC

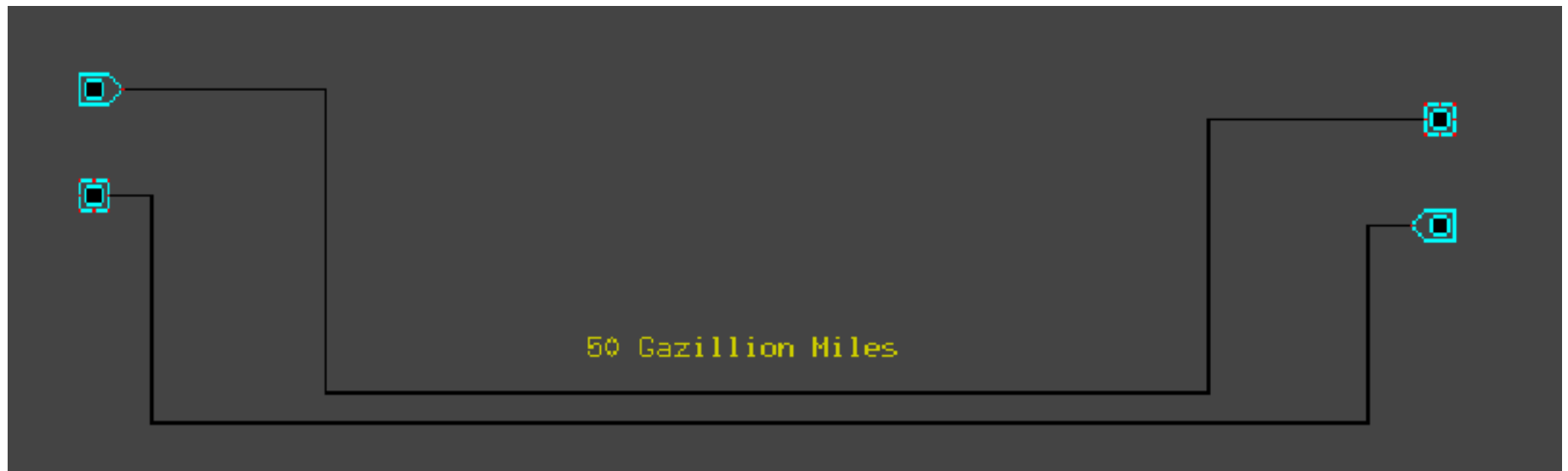


## Buses

- **What is a bus?**
  - A set of data wires that can get input from and send output to multiple sources and destinations.
- **How to make a bus?**
  - Multiple sets of data go through tri-states and then merge into one data wire.
  - Can only have 1 of the tri-states connected to the wire high at a time. Otherwise a **race condition** will occur.
- **When to use a bus?**
  - Whenever choosing between more than 2 sets of data.
  - Whenever input and output need to be on the same wires.



## Bus Example – The Problem



This is a simple telegraph system used for sending Morse code across a great distance. Notice that you have to run **two** 50 gazillion mile cables between the two locations. Given that the two locations always alternate sending messages (i.e. they never send messages at the same time), how can we reduce the amount of cabling?



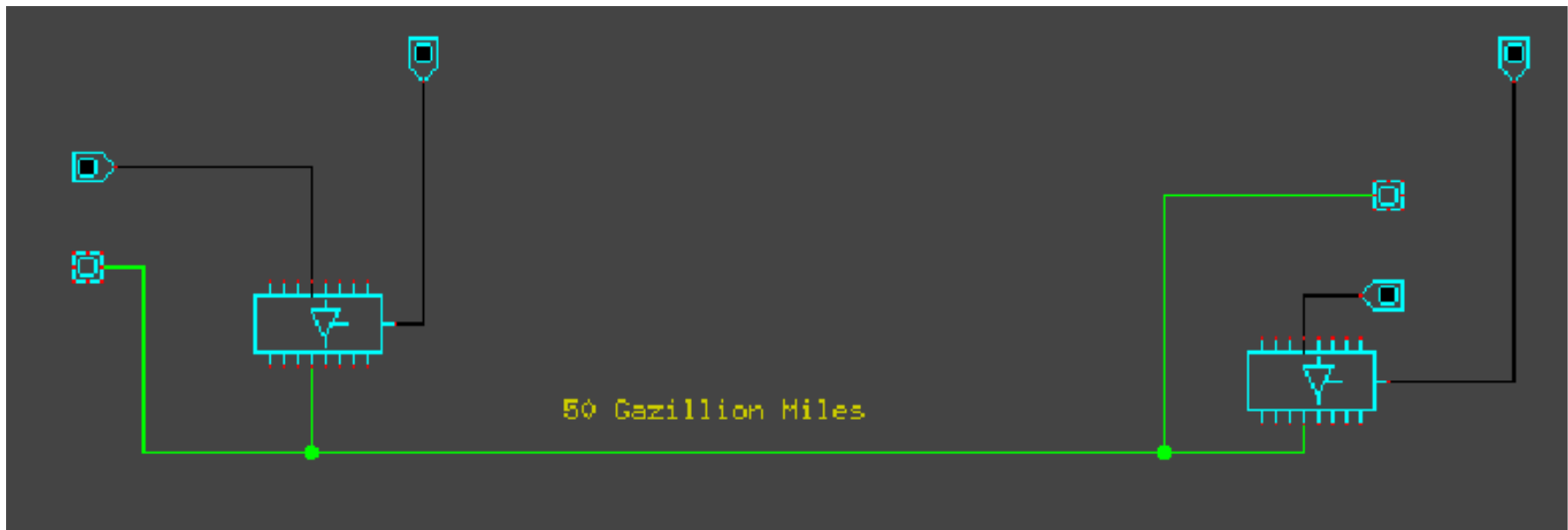
## Bus Example – Attempted Fix



This looks like it might work, but it won't. Suppose the switch on the right is on, but the one on the left is off. What is the value on the wire? One switch says it's high, and the other says it's low. So they fight, we have a race condition, and nobody's happy



## Bus Example – Real Fix



The solution is to create a bus. Put a tri-state between each of the switches and the main wire. In this case, we've chosen to control the tri-states with two additional switches, which in this problem would be analogous to “transmit” switches. Note that the two slides still have to make sure that they aren't both transmitting at the same time...



## Using RAMs and ROMs

- You'll be using the built-in RAM component for the RAM.
- And the built-in ROM for your ROM.
- Note that a ROM is just a slightly re-wired RAM; nothing special.



## Using RAMs and ROMs

- Oops, how does the RAM work again?
  - Same as specs for your RAM in HW3:

Action	sel	ld
Read	1	1
Write on rising edge of clock	1	0
Disconnect (High-Impedance)	0	x

- Where do you use them?
  - 1 or 2 control ROMs
  - 1 Program ROM
  - 1 RAM for Storage



## Dealing with ROM Data

- You can edit RAM/ROM directly with the Poke tool.
- Right-click on a RAM and click “Edit Contents...” to Save and Load from a file.
  - The ROM files are just text-based, with a simple format.
- Useful (but not mandatory) tip: you can write your own assembler for RISC Assembly Language to save yourself time creating programs.
  - And you can write a simple simulator to make sure your RISC programs are actually correct.
  - This will help a lot in debugging your circuitry.



## Jumps

- In RISC, jumps are relative to program counter.
  - Need to add the jump value to the program counter.
- It jumps if value in src register is greater than 0.
  - How to check if a number is  $>0$  with gates?



Questions?



Good Luck!