1/16/2021  
  
I may need to change this design a bit, I need to find something that will work with the current interrupt module. I am thinking that maybe a wait instruction should interrupt from its state, and that the interrupt should be generated by some other ram usage circuit.

I am going to upload a truth table I created for the finite state control that determines what processor to give ram access to.

1/14/2021

I neglected to consider the case in which access is granted to a lower ranking processor, and then a higher-ranking processor requests access. I will be making a truth table for this circuit shortly to sniff out any other edge cases I haven’t considered.

1/5/2021  
  
The new system is pretty similar to that described below, but the main difference is that there is an eeprom which can decide what processor to award ram access in a collision scenario

1/5/2021

I have decided to implement memory sharing in the computer so that I can implement a screen. The ram availability checker looks to see if the ram module is currently being used and what the current microinstruction is for the cpu in question which wishes to share the ram module. If the ram module is available, and if the current microinstruction is one in which nothing occurs, the mode control logic will send a signal to the main control that indicates it is an appropriate time to access the ram module. This system should prevent cases in which authorization is granted at a micro instruction other than 0000, resulting in an incomplete execution of said instruction.

I have yet to implement a system where the main processor has priority over peripheral processors in the situation that both processors wish to access the ram module at the same time, when it is available

11/23/2020

I am not sure of what I will use the rom module for at the moment as I will not be making a proper os for this computer. Maybe I could use the rom as a code save-point, something where I can transfer code into the computer after writing it in a high level language. This would allow me to write more code as I will be more accustomed with the interface, and additionally the code would be saved. However, one problem I had with the 8 bit computer was that every time I removed its eeproms to re-write their contents, the pins would be damaged during removal and insertion into the breadboard.  
  
Possible solutions:  
  
-better chip interface – use rom or eeproms programmer  
-use wires to connect a raspberry pi to module, no removal needed   
 - needs pi to be programmed -> extra work  
 - can be tailored to needs better  
 -coding experience

The ram module should have 65536 addresses to work with. Since I intend on keeping the address bus 16 bits long, this means the rom module will waste some of the ram addresses. I am thinking about doing the first 10 bits for the ram module, meaning that 1024 addresses will be persistent and read only in the computer environment. This is not reflected in the rom modules chip design, I will only be using the first 10 bits of the rom module.

-in order to do this, I will need to have two rom modules connected together and only allow the first two bits of the second module to be used.

- I am starting to think that I may have more time and motivation to complete this project if I use an abbreviated form of multisim design.

* Have a single line representing connections, and the range of bits they represent:  
  i.e input 0-7

11/24/2020  
1:37 pm

Separate wires will be connected to the rom chip’s input, vcc, ground and enable pins for raspi connections to be made. When programming the rom chip, the computer should be off so that signal interference wont occur.