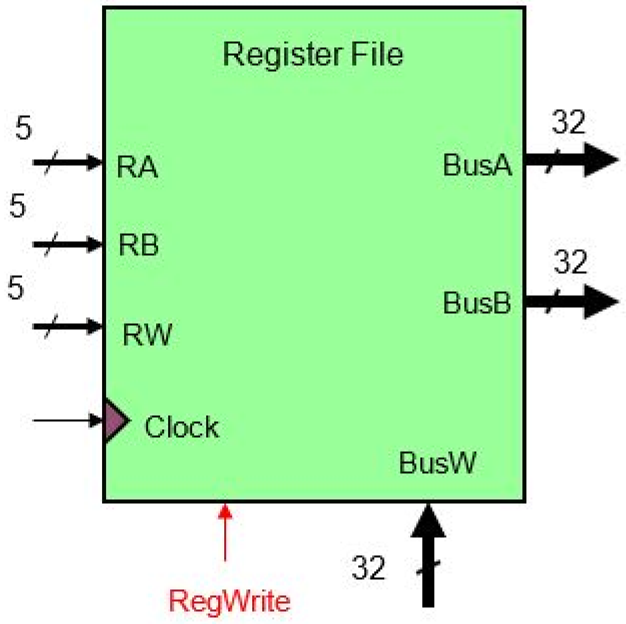
Annie, Trey, Shreyas, Zabeeh
CDA 4205

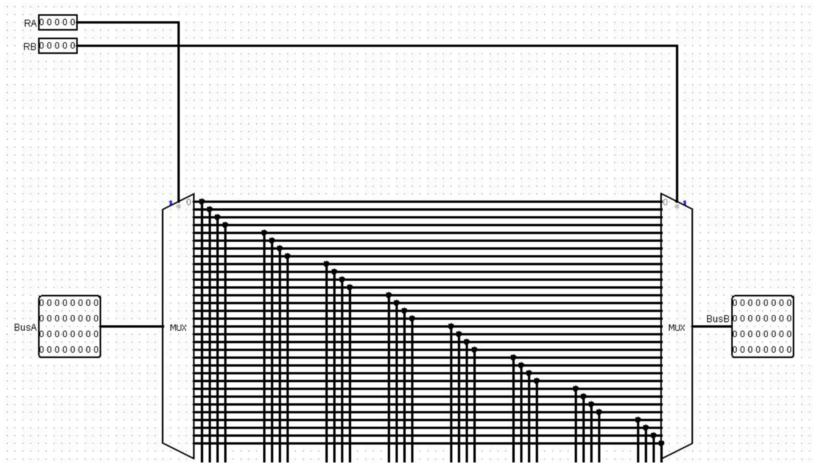
Register File and Memory

<u>TO SEE THE things in action please click the single cycle cpu we have set up implementations of the register and 4gb memory</u>

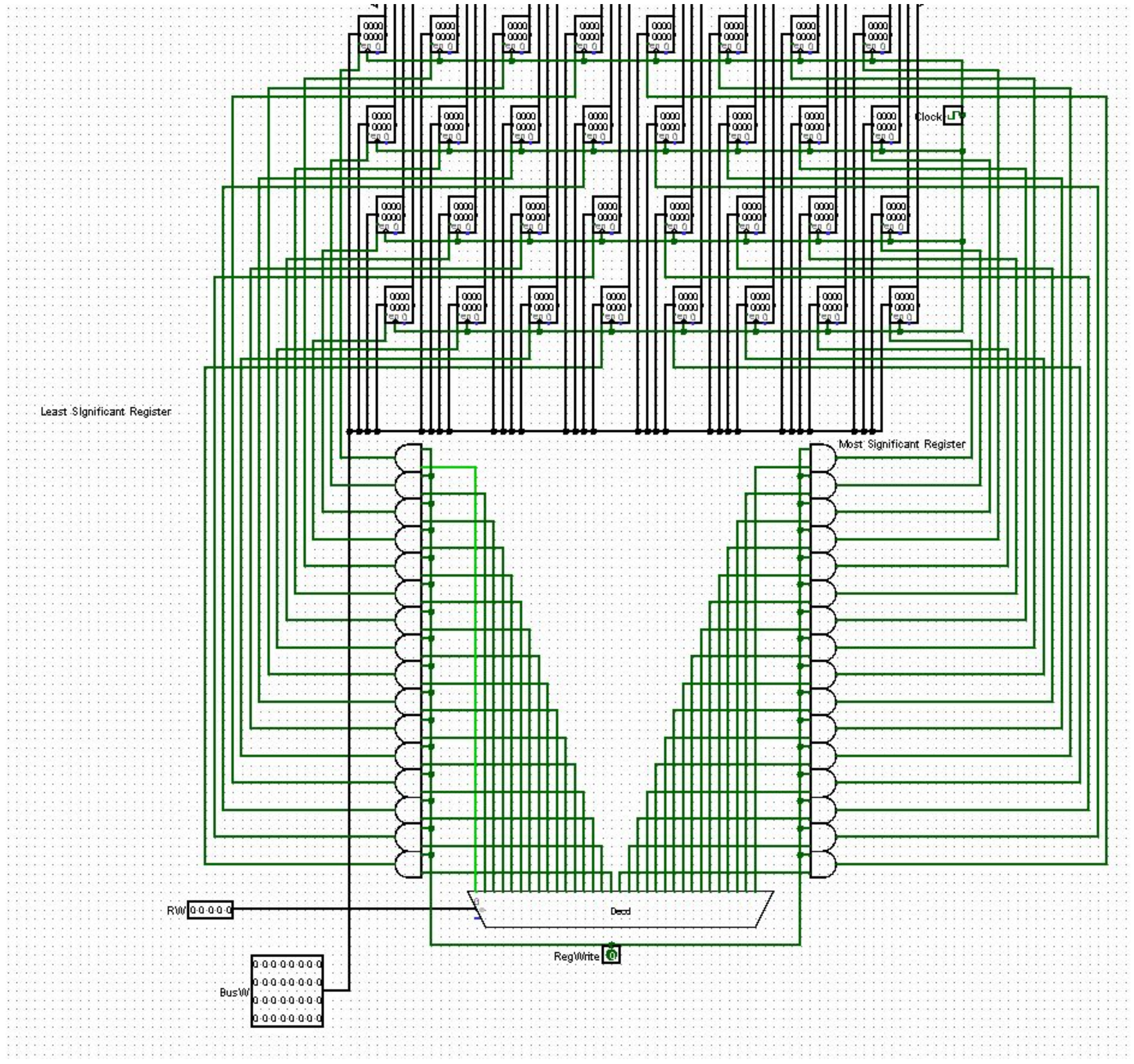
32 x 32 bit Register File



Imagine with this picture, RA uses 5 bits to select 1 out of 32 registers, which stores a 32 bit number. RA selects which register will be output into BusA
RB uses 5 bits to select another 1 out of 32 registers, and it could be the same register as RA if required. RB selects which register will be output into BusB

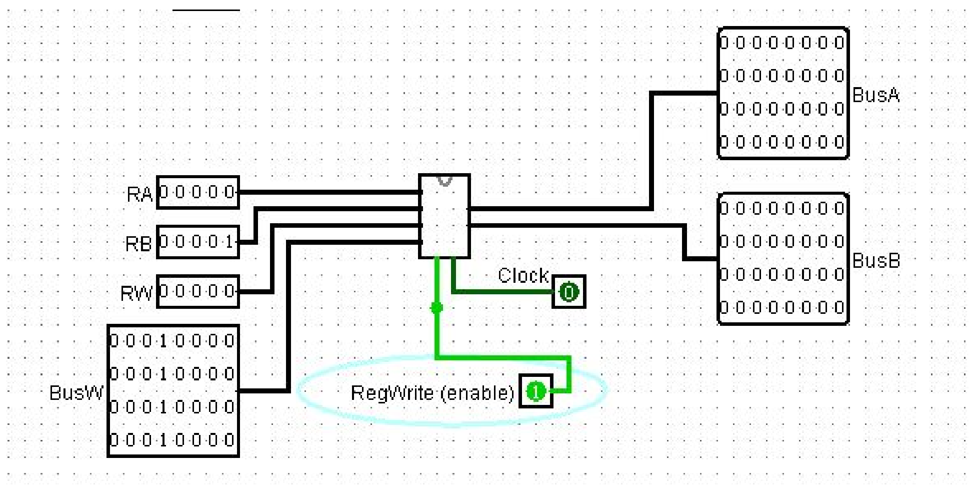


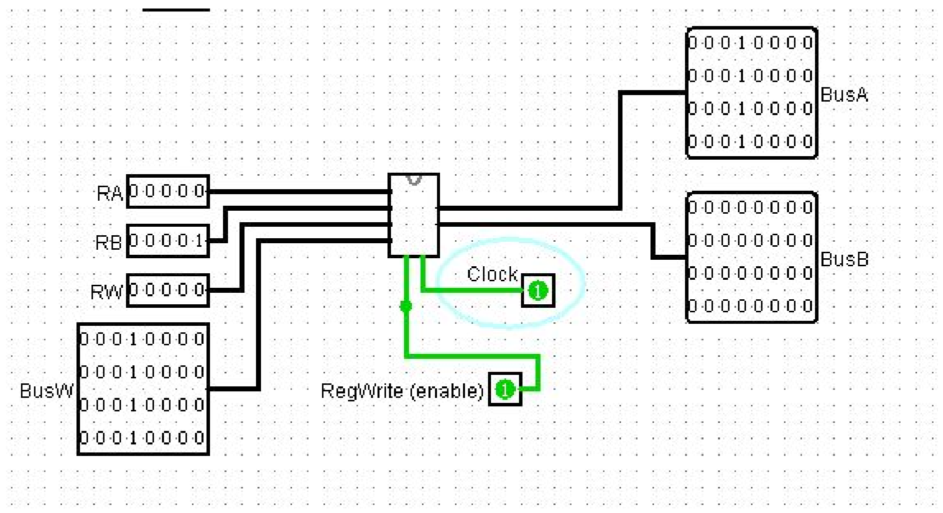
It does this by using two multiplexors, one dedicated to BusA output, and the other BusB output. And the output of the registers will always be connected to the multiplexor selection in the middle.
This covers the read/load part of the register file, and we will now go over the writing portion.
RW RegWrite and BusW all need to be working together in order to write to a register in our logisim.



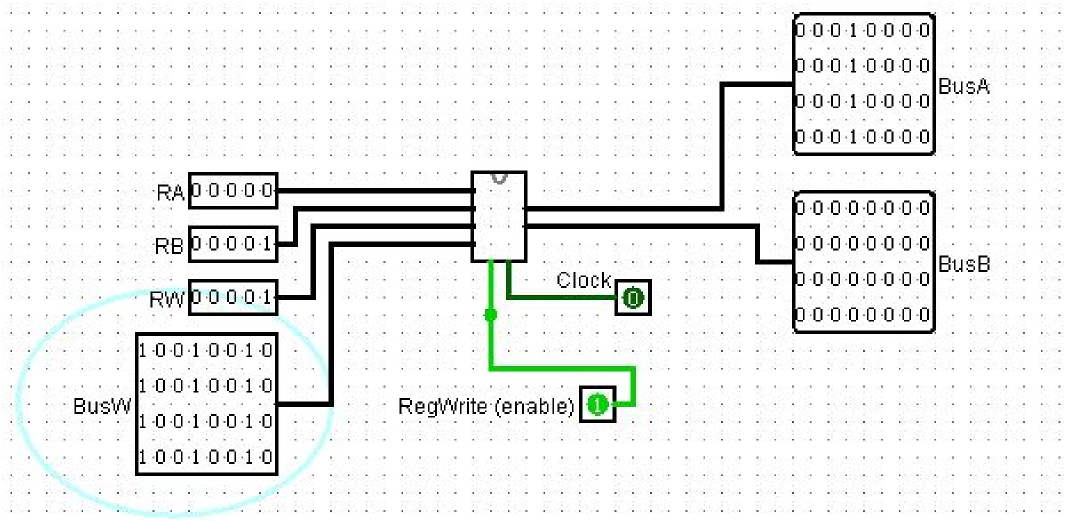
RW selects which of the 32 bits will be written to, but this time it uses a decoder. The only way a chip will written to is if both RW and RegWrite are active, as there is an and gate that needs both to be selected.
BusW is the input that will be stored in the register, and busW is connected to every register input, meaning all that is required to store a value is the regWrite, RW and a flick of the clock. Whatever value you have stored in BusW will be put into the register you select, after the clock ticks.

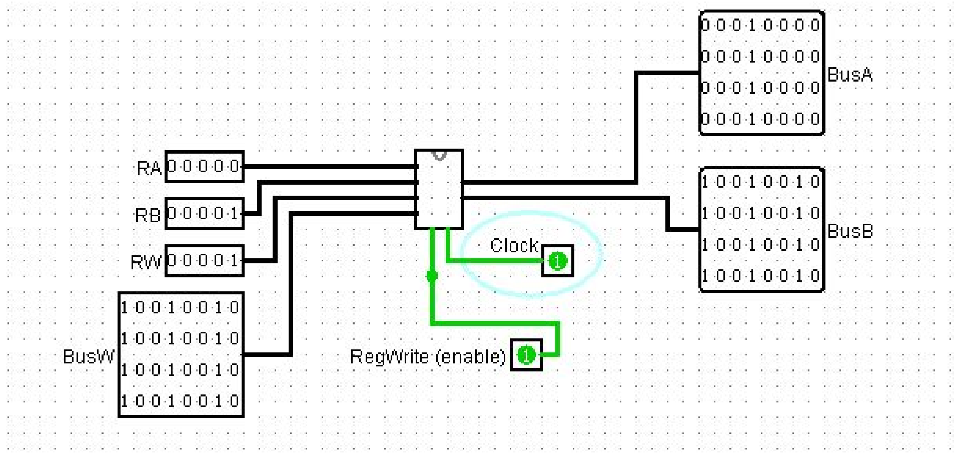
Example:


RA 0 0 0 0 0
RB 0 0 0 0 1
RW 0 0 0 0 0
BusW 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0
Clock 0
RegWrite (enable) 1
BusA 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
BusB 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0


RA 0 0 0 0 0
RB 0 0 0 0 1
RW 0 0 0 0 0
BusW 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0
Clock 1
RegWrite (enable) 1
BusA 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0
BusB 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

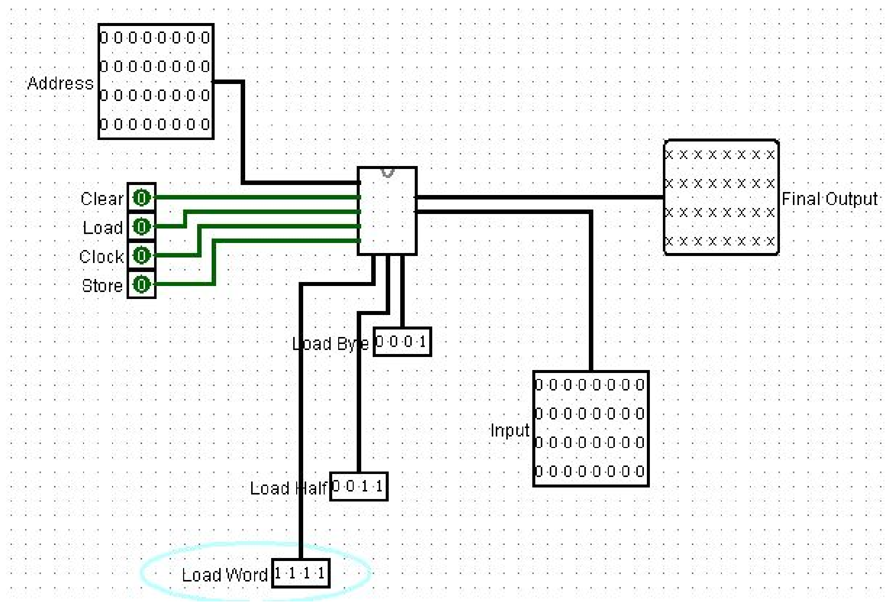
After the clock, it wrote to register 00000, and the output BusA is set to register 00000 due to RA as well.
BusB remained unchanged





Now after the clock, I wrote to register 1 instead of 0, and the resulting output is now in BusB, as that is what RB is selecting.

4GB byte-addressable memory

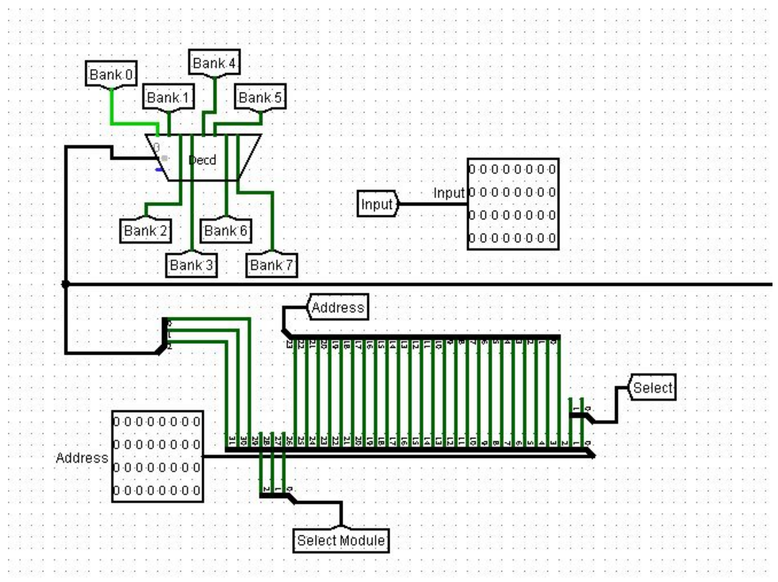


For our 4GB memory, it is a little harder to explain.
First lets look at the four single bit options to the left.
Clear sets all the bits in an individual 32 bit storage to 0 (will explain 32 bit storage soon)
Load loads all the selected bits out of the 32 bit storage based on Load byte, load half, load word
Clock just pushes in the input when store is selected, since store only activates on clock tick.
Store enables whatever is written in input to be stored in an individual 32 bit storage.

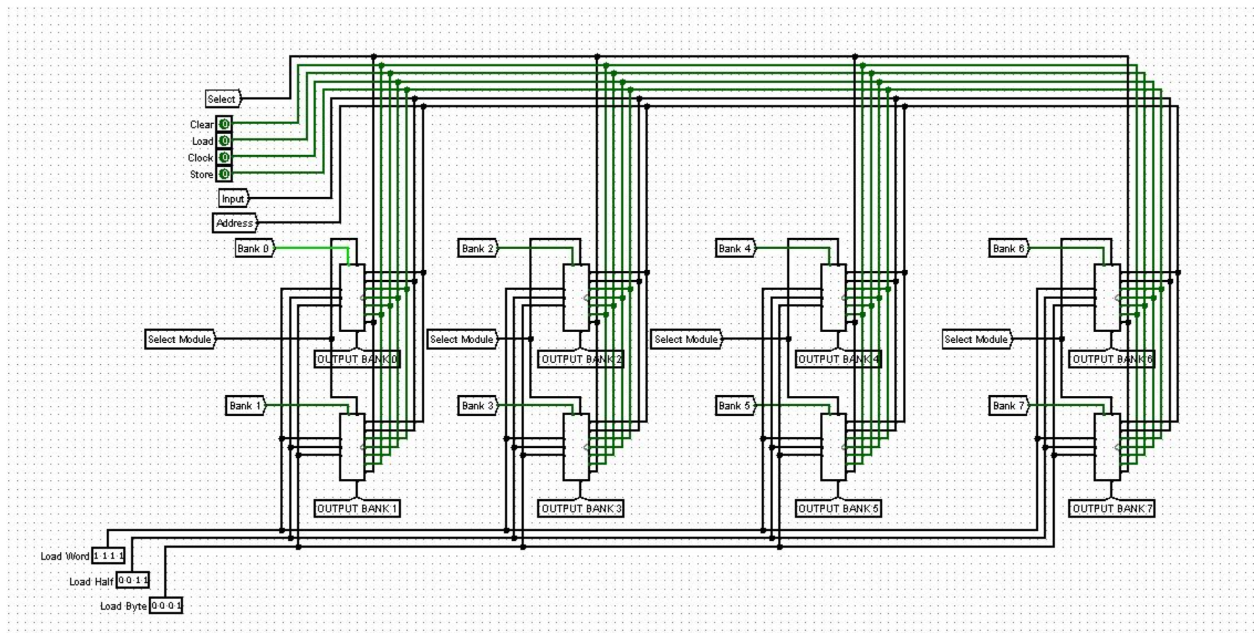
Then, Load Byte, Load Half, and Load Word are temporarily all hard coded, to select 8, 16, and 32 bits respectively all from the right side of our individual 32 bit storage.

Final Output is our 8, 16, or 32 bit number that we load from our storage.

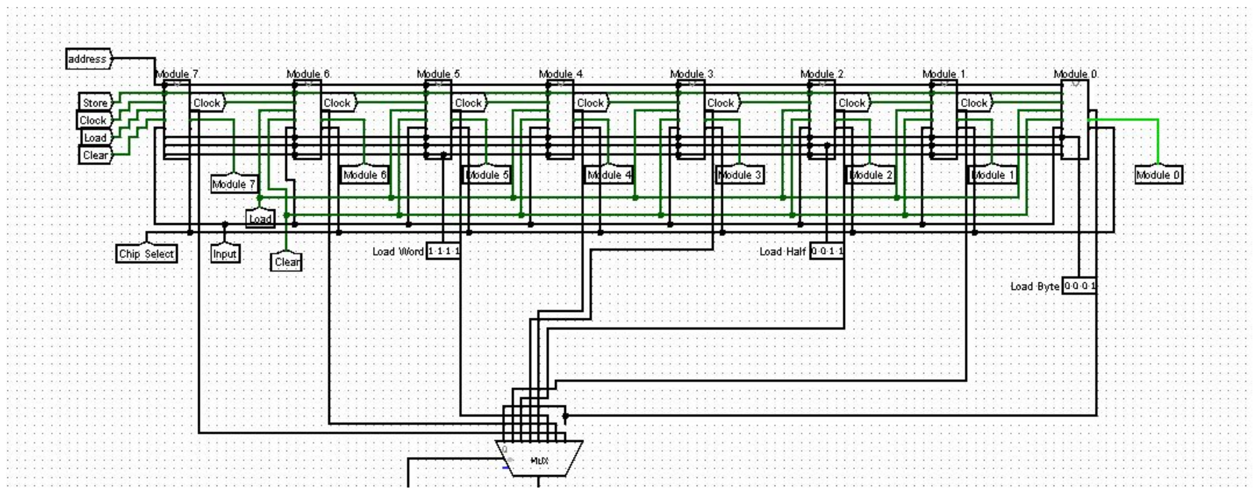
Now we get to our address.



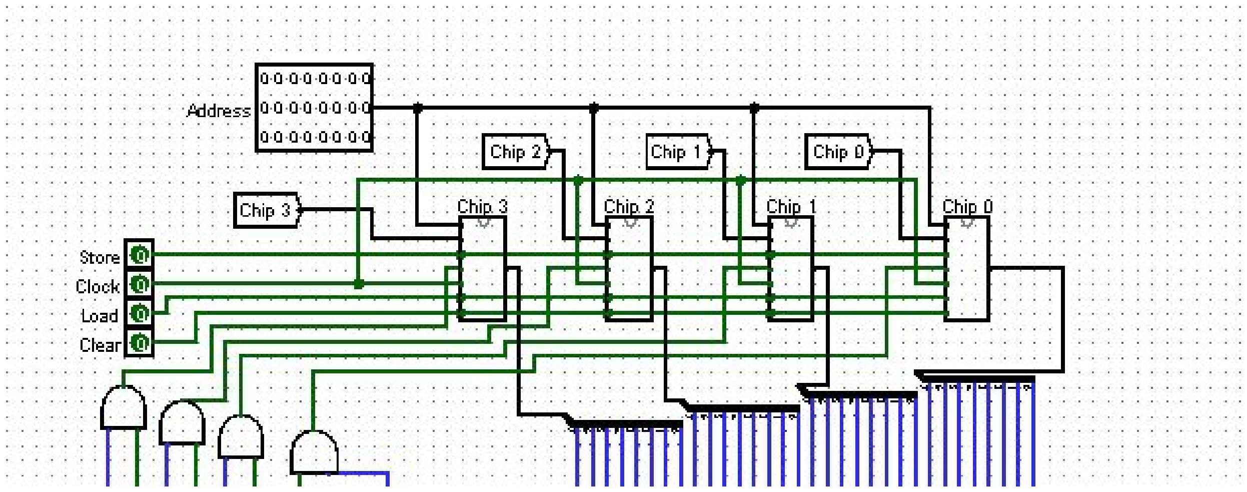
Our address is broken down into 32 bits. Bit 31, 30, and 29 select the bank of storage we currently look at. Bits 28, 27, and 26 select which module inside of the bank we select, and the last two bits 0, and 1 select how many bytes we are writing to in memory, 8, 16, or 32.
So for the 32 bit address 0000 0000 0000 0000 0000 0000 0000 0000:
It would select bank 0, module 0, and chip 0 to write too and would only be able to store/load 8 bits.
For 0010 0100 0000 0000 0000 0000 0000 0001:
It would select bank 1, module 1, and chip 0 and 1, and would store/load 16 bits.
For 1001 0000 0000 0000 0000 0000 0000 0010:
It would select bank 4, module 4, and chips 0, 1, 2, 3. Which would be able to store/load 32 bits.



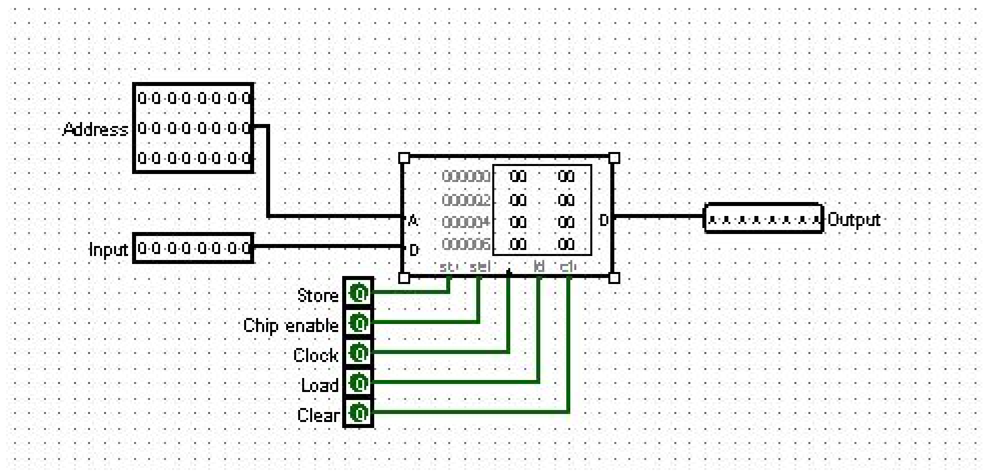
Here are our 8 banks



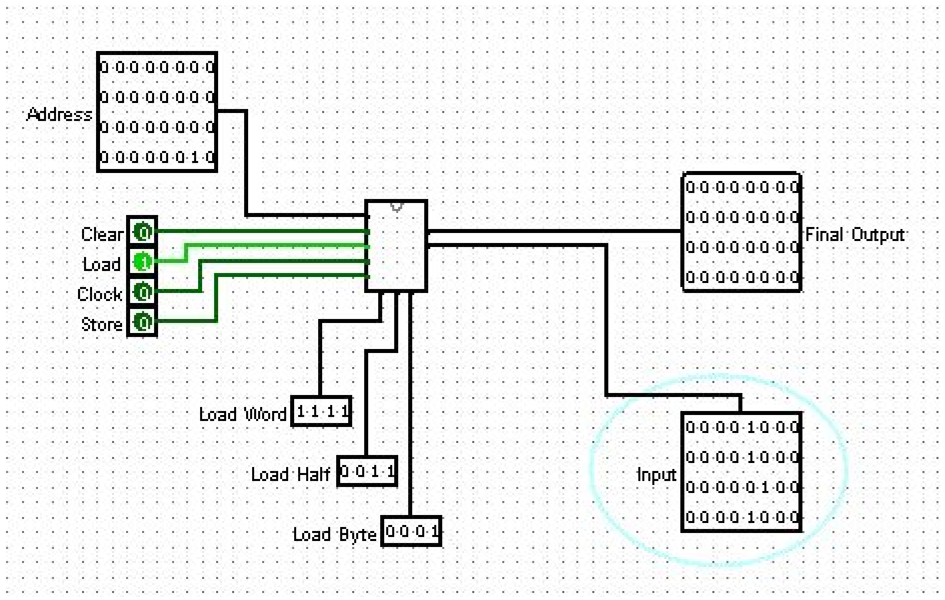
Inside each bank are 8 modules.



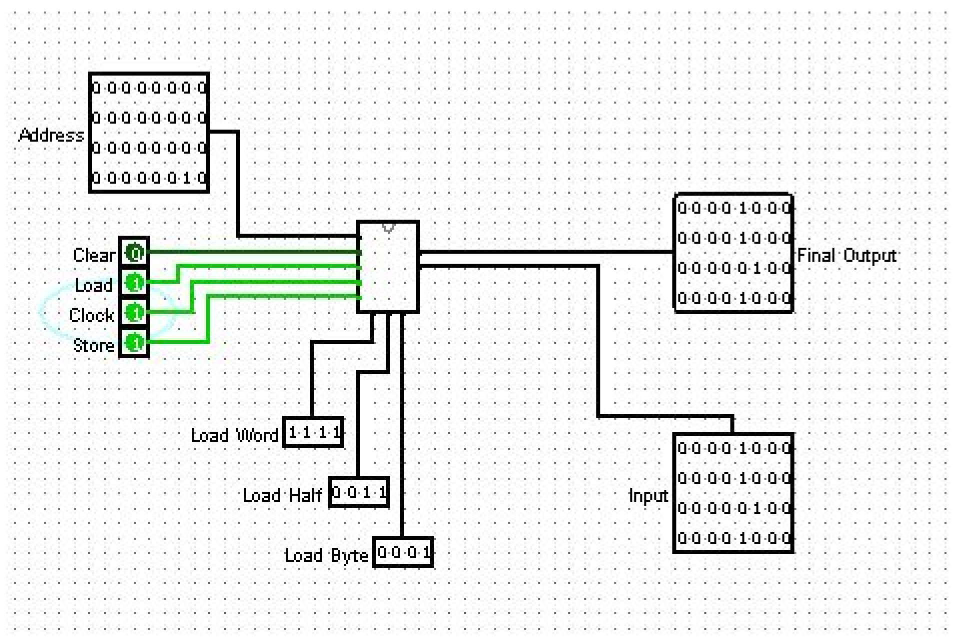
Inside each module are 4 chips.



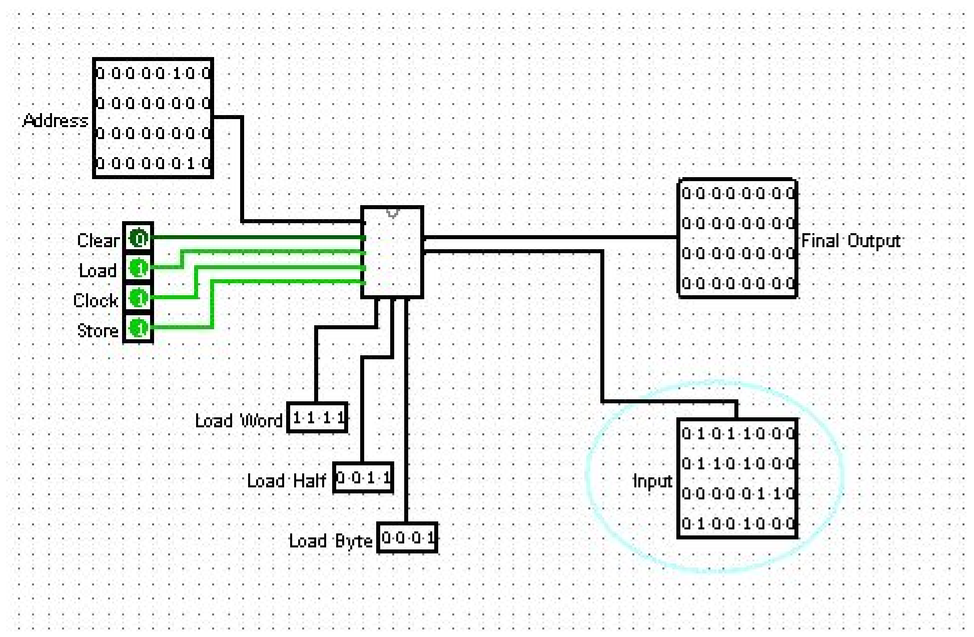
And each chip stores the information we want.



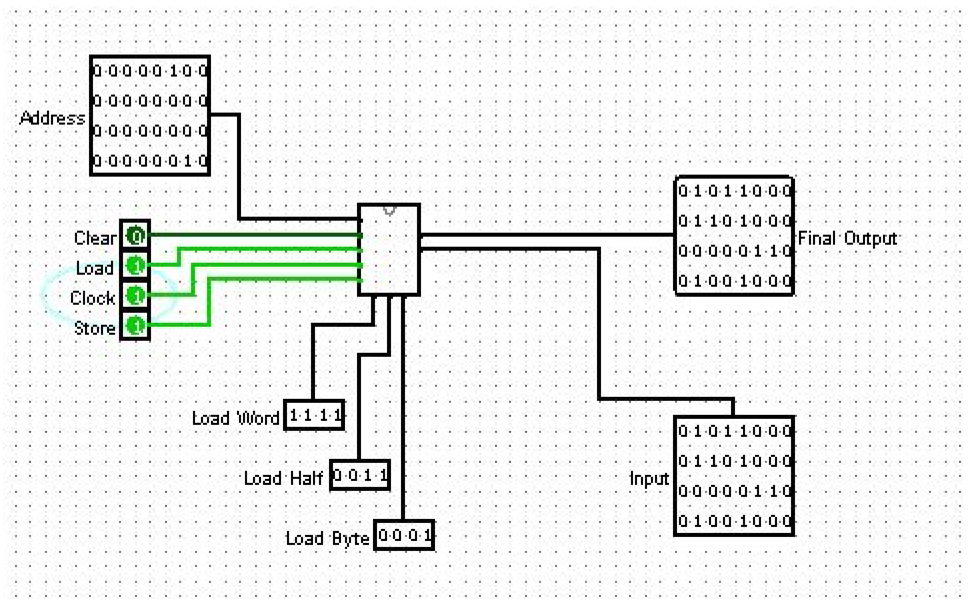
In this picture, I am currently loading all 32 bits from bank 0, module 0, and chip 0-3 and I am about to store the value from our input.



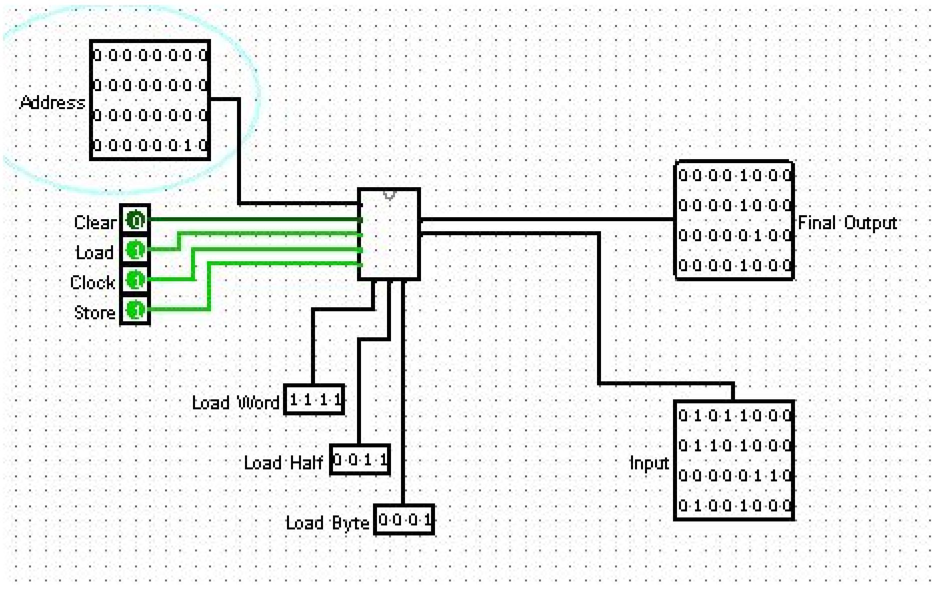
Here we see it did store to the chips, because we see the output is now our input.



Now I am preparing to store a different value inside of bank 0, module 1, chip 0-3.



After flicking the clock, it successfully wrote the data, as the output is now the input.
But is our information on bank 0 module 0 chip 0-3 still there?



Yes, we can flick the 26th bit back and forth to change from module 0 to module 1, and both of the inputs we previously entered will return back to us in our output 32 bit block.