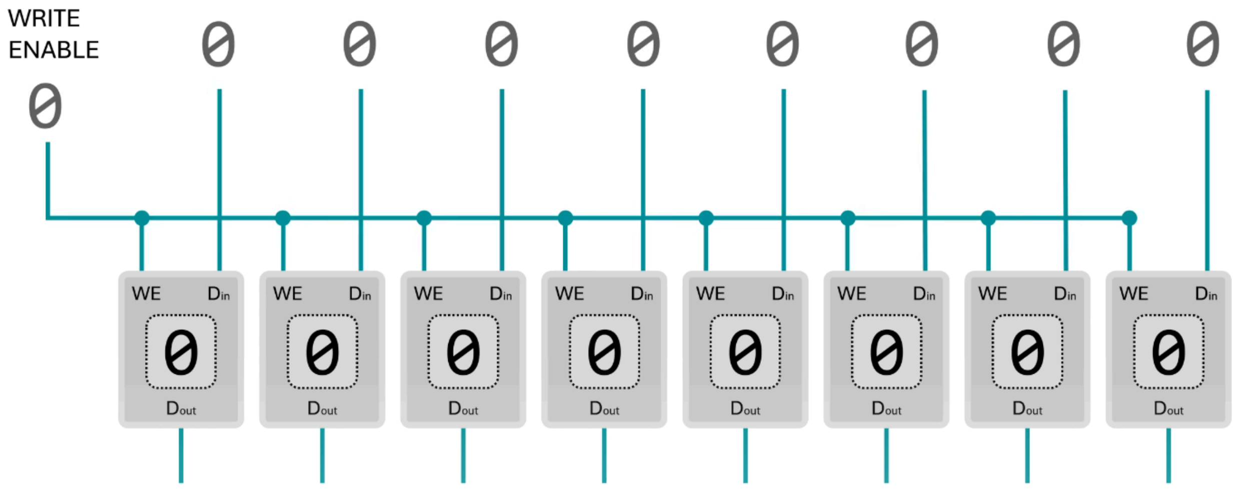
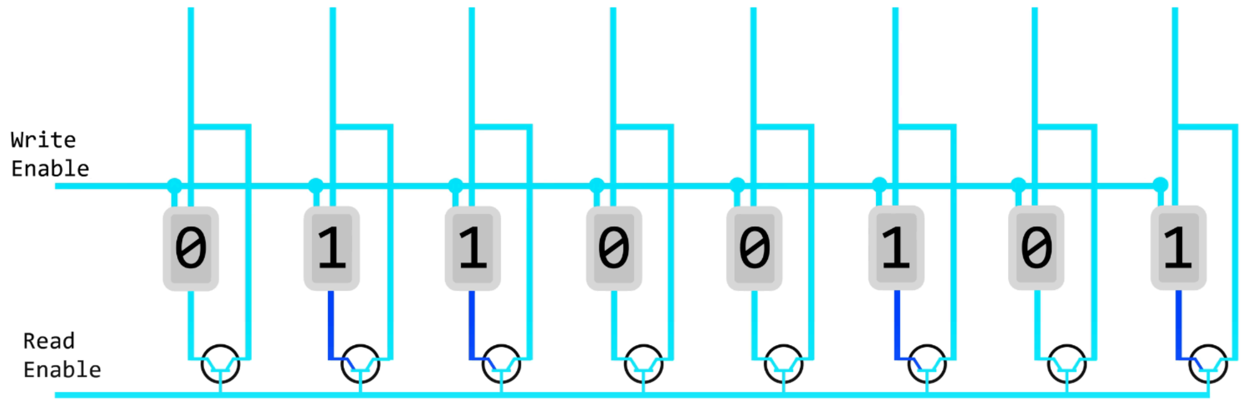
Chapter 3 - Simple CPU

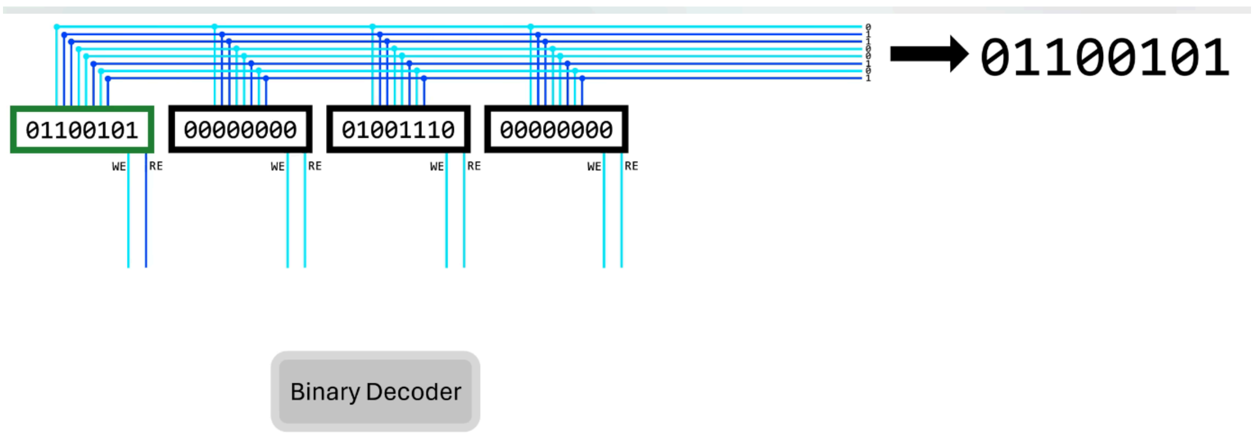
Now that we have two of the core components understood from a transistor level - the RAM and the ALU, we are well on our way to build a minimal CPU. First, let's remind ourselves of the register structure that we came up with in the last chapter.



We can modify this in the same way we did the SRAM gated latches, by combining the data in and data out lines, and adding a read enable line.



This enables us with the same advantages as SRAM - we can write to and read from individual registers by using one data line and controlling which register it reaches by using a decoder. The decoder does this by controlling the ‘write-enable’ and ‘read-enable’ lines on every register.



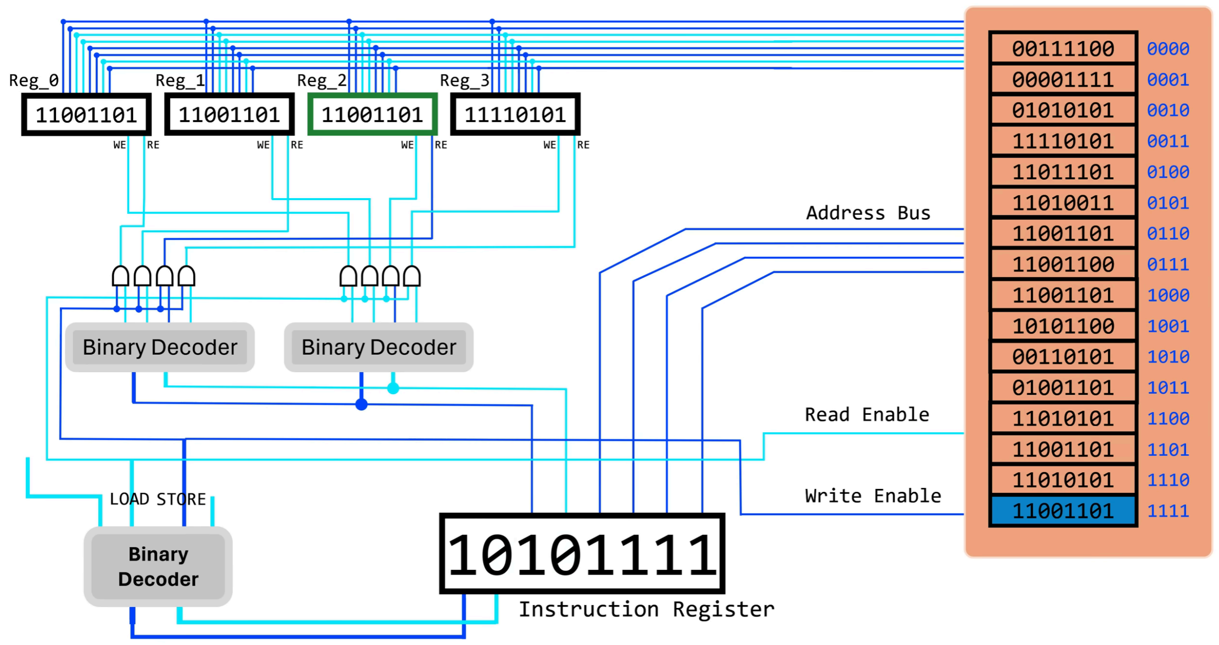
Now, let’s look at implementing some assembly instructions - in particular lets look at load and store. These two instructions are in the following format:

- Load Register_Name Address (Ex: Load R1 1101) - loads the data stored in the memory pointed by the address into the register
- Store Register_Name Address (Ex: Store R1 1101) - stores the data in the register into the memory pointed by the address

Assembly instruction is machine code in a format that is easier for us to understand. Both load and store instructions are in binary format in their simplest form. This means that we can implement these instructions easily by using decoders in the following format:

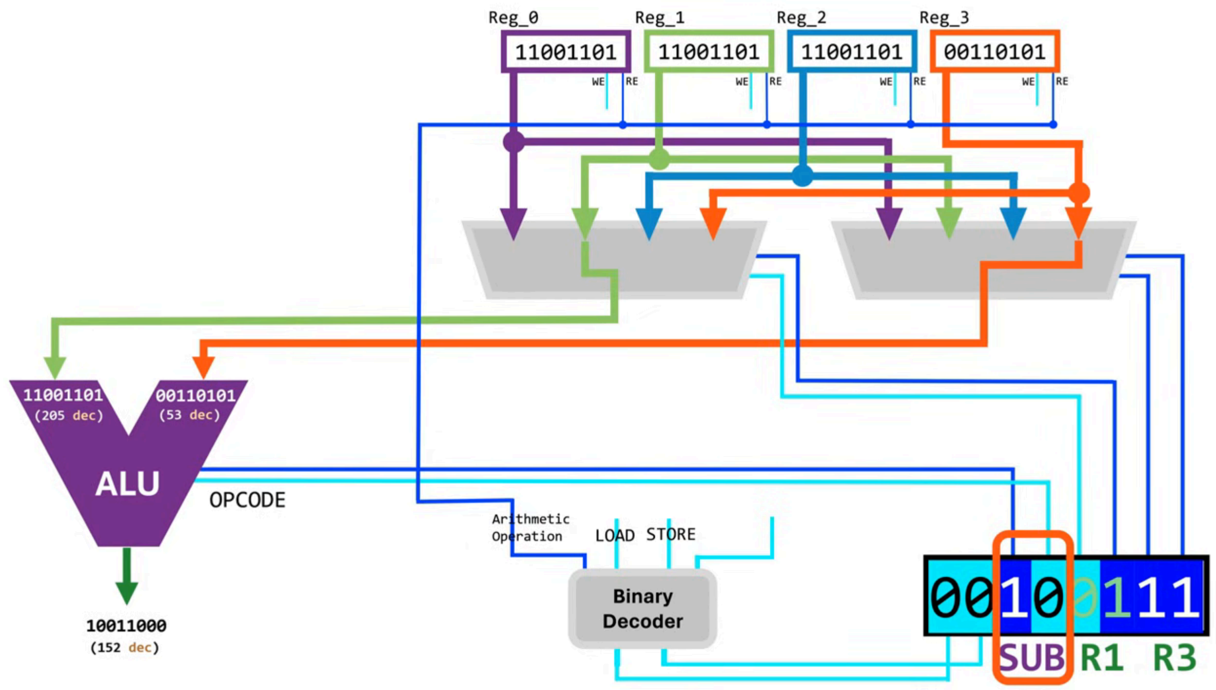
- First stage decoder would be for the load/store part. The decoder would take in this code and if it receives store, would set write enable high and if it receives load, would set read enable high.
- Second stage decoder would be to access the registers. We only want to read/write from one register at a time and this decoder would be connected to write enable and read enable lines. This is the same decoder as the one drawn above.

Both decoders can be connected in the following manner to achieve implementation of these assembly instructions.



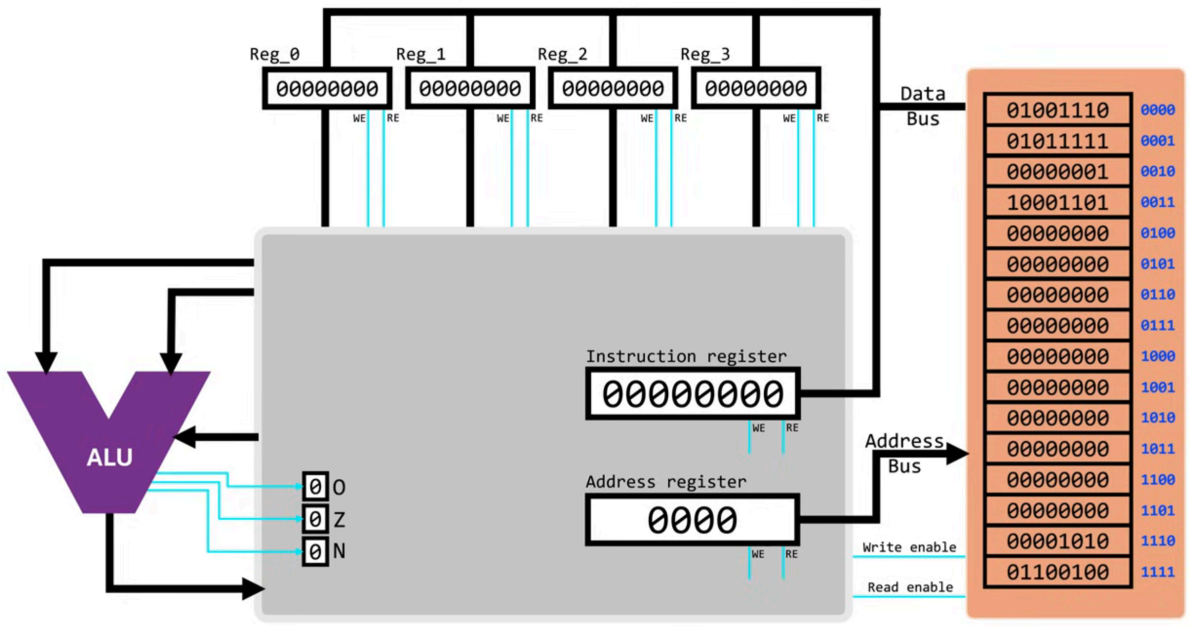
So we have load and store working now - but what about arithmetic operations? We had previously looked at the ALU and how do we connect the ALU to this structure? This follows a similar pattern.

- We can set 00 to be the starting two bits for arithmetic operation for example.
- The next two bits of the instruction would represent the type of arithmetic operation to be done and would serve as the opcode for the ALU.
- The next two sets of bits would represent the next two registers onto which the operations must be performed - the operands of the ALU. This should be the input to a multiplexer - a structure that takes in multiple inputs (all the registers in this case) and select lines (which would be the two sets of bits we are referring to), and would direct only one of those multiple inputs to the output based on the value of the select line.
- The output of the ALU would then be stored in the register required



If we combine the circuitry for this with the ones for load and store as well, we get a very messy circuit for one image - so we can put all this in a black box that we can call the control unit. The control unit is responsible for directing the circuitry to the necessary components based on the value of the instruction register.

Combining all the previous components together, we can simplify it as below:



This big grey box is the control unit - it has a lot of circuitry inside it for breaking down the instruction but we will focus on two main registers:

- Instruction register - this holds the current instruction to be executed
- Address register - this points to the memory location of the next instruction to execute.

We've heard about the fetch - decode - execute cycle of the CPU and with this structure, we can now explain it easily:

- Fetch - The read enable input of the address register is activated to get the next instruction from memory. Then the write enable input of the instruction register is set high to load this instruction into the instruction register - we have 'fetched' the instruction.
- Decode - The read enable of the instruction register is set high and serves as an input to the rest of the internal circuitry of the control unit, which interprets this instruction and sends the necessary signals to all the required components to perform this instruction. The instruction is 'decoded'.
- Execute - The components carry out the task required in the manner explained previously and store the output in the required register. The instruction is

'executed'. The address register value is incremented in this stage as well to point to the next instruction to be executed.

And there we have it - we have created a CPU that can execute instructions!