Madison Chamberlain

913629196

1/27/2021

The AND Gate Turing Machine

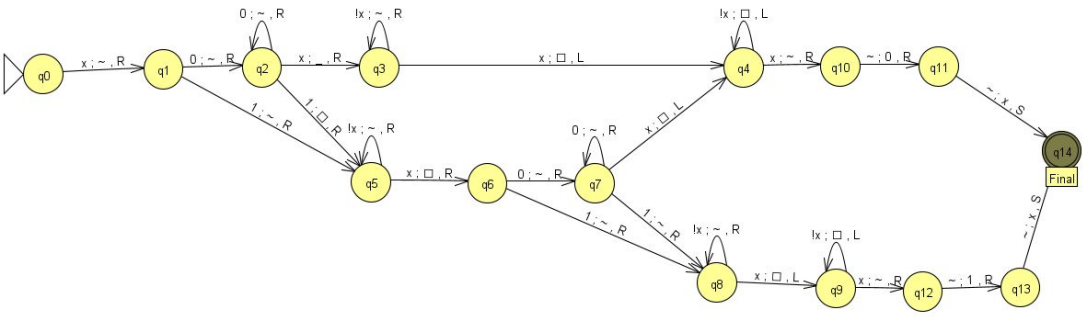
I chose to build a logical AND gate as the CPU component I was to build using a turing machine. I chose to build the AND gate because in my opinion, it is one of the most useful components needed to build a CPU.

Last quarter, I took ECS154A, and we built an entire CPU out of wires and gates. I used dozens of AND gates in my design, as ANDing input is incredibly useful and necessary. ANDing is used everywhere from higher level programming languages to within the architecture of the computer, because it is an extremely useful tool to use when you want to execute a command, but you only want to execute that command sometimes. For example, when designing a computer, you may want the computer to execute a command given by the user. However, you would not want to try to execute a command from the user if the user has not given the computer any input or if the computer is not ready to process a new command. So, if the computer wanted to avoid taking new data until both of those conditions were fulfilled, there should be a statement as follows: if the ready bit is on (meaning the computer is done doing the tasks it had done before), AND the input received bit is on (meaning that the computer user has interacted with the computer in some way to give it a new instruction) then the computer can begin to execute that instruction. If those conditions are both not true, the computer will know to

wait until they are. The logical AND gate is needed for this step. This is very important and crucial to computing, which is why I chose to build a turing machine that can do this.

In the case of logical bit inputs, if two binary numbers are given, if neither of them are 0, then the output should be true, or 1. Otherwise, so if one or both statement are equal to 0, then the gate should return false or 0. In the case of the turing machine, these two binary number inputs are given in the form of one string, starting and ending with an x, and with an x inbetween the two binary numbers to indicate where one number ends and the next starts. The way I made my turing machine return 0 if one or more binary number from the string was a 0 was basically by scanning through the first number, and determining whether it contained any 1s. If it did not, I would essentially change the string to x0x, and if not I would check the second binary number. If the second number contained only 0s again, I would change the string to x0x, or else I would change the string to x1x and halt.

The way I built a turing machine which executes this function is with a 14 state machine. As seen below.



The initial state, q0, simply detects the first x and moves one character to the right. The next state q1, determines whether the first character in the first binary number is a 0 or a 1. If it

is a 0, it goes to q2, and if it is a 1 it goes to q5. In q2, the machine will check the contents of the next character, if it is a 0, it will move right and stay in q2, and if it is a 1, it will move to q5. Once an x is hit in state q2, we will know that the first binary string is a 0, and that the AND machine must return a 0 to indicate so. We will do this by moving to state q3, and changing the x between the two binary numbers to a blank space to avoid confusion later. In q3 we will simply continue to check the next character until we come to an x indixating the end of the string. When an x is spotted, it will be changed to a blank space, and we will move into state q4. Q4 will continue left, changing each character in the string to a blank space until it reaches the initial x indicating the start of the string. When it reaches the initial x, it will go into state q10. State q10 will change the next character, a space into a 0 and move right, going into state q11. Q11 will change the next space to an x, and stop moving, ending in qhault. If however we did not have a 0 as our first binary number, we will be back in state q5. Q5 will iterate through all characters in the first string until it hits the x, it now knows that the first number was not a 0, and must check the second binary number. Upon hitting an x, the x will be changed to a blank space to avoid later confusion, and the machine will move to state q6. Q6, like q1 will determine whether or not the first character of the second binary number is a 0 or a 1. If it is a 0, it will go to state q7, which much like state q2, will keep in q7 if the rest of the characters are 0, or else it will go to state q8. If q8 is entered, it is now known that neither of the binary numbers are 0, so it will go all the way to the final x, and then move left, changing each character to a blank space until the initial x is hit, then it will move right, change that space to a 1, move right again changin that space to an x, and then move to q14 for halt. If q7 finds that all of the characters in the second binary digit are 0, it will move to q4, repeating the process from above, and returning an x0x.

An example, walked through of how this works can be seen if we input the string x10x00x into our machine. Upon getting an x in q0 (x10x00x), we will move to the next character to the right and enter q1. Now in q1 (x10x00x), we check the contents of the character, which is a 1. This will take us to state q5, and we will move right on our way to q5 (x10x00x). In q5 we will check the next character. Since it is a 0, we will move to the next right most character, staying in q5 (x10x00x). We will again check this character, since it is now an x, we will move into state q6, changing the x to a blank space and moving one to the right (x10 00x). Now we are in state q6 and we have a 0. This will cause us to go to q7, moving right along the way (x10 00x). In q7, we will check the contents of the character. It is a 0, so we stay in q7 and move one character to the right(x10 00x). The next character to the right is an x, so we move from state q7 to q4, changing that x to a blank space, and moving to the left (x10 00). In q4 we will check the next character, since it is a 0, not an x, we will stay in q4 and change the 0 to a blank space, moving left (x10 0). Again we have a 0 in q4 so we change it to a blank and move left (x10_). Now we have a blank space in q4 so we change it to a blank space and move left (x10). Now we have a 0 in q4 so we change it to a blank space and move left (x1). Now we have a 1 so we change it to a blank space and move left (x). Now we have the initial x, so we move to the right and enter q10 (x_). From q10, we change the next character to a 0 and move right and enter q11 (x0_). We are now in q11, we have a blank character, we change it to a x, stop moving and enter the final state (x0x). Our machine has changed the string to the output we wanted, x0x, and halted.

Below, I will also show the progression the turing machine will make to change x101x11x to x1x, and x00x1x to x0x, but I will not explain in detail what is going on.

Example 1:

(Input: x101x11x, output: x0x)

(x101x11x) -> (x101x11x) -> (x101x11x) -> (x101x11x) -> (x101x11x) -> (x101 11x) -> (x101 11x) -> (x101 11x) -> (x101 11) -> (x101 1) -> (x101_) -> (x101) -> (x10) -> (x1) -> (x) -> (x_) -> (x0_) -> (x0x) -> hault

Example 2:

(Input: x00x1x, output: x0x)

(x00x1x) -> (x00x1x) -> (x00x1x) -> (x00x1x) -> (x00 1x) -> (x00 1x) -> (x00 1) -> (x00_) -> (x00) -> (x0) -> (x) -> (x_) -> (x0_) -> (x0x) -> halt

As seen above, my logical AND turing machine works as it should, and it was built because it executes a highly important element in computation.