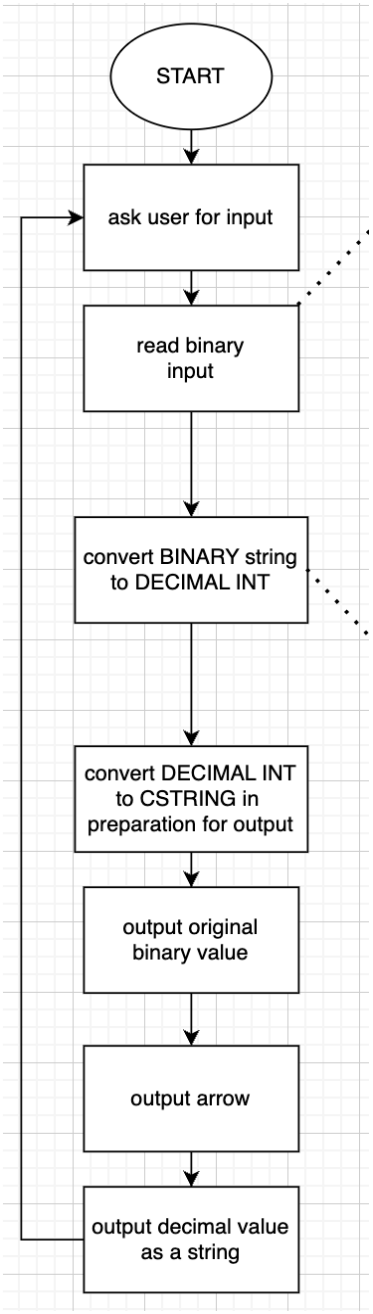


Main

The purpose of our main is to serve as an entry point into our program. It handles execution by prompting the user and calling a group of functions to perform the following tasks:

(1) : Prints out command list to the console and prompt the user for input.

(2) : Reads and parses user input.

(3) : Conditional sign extension to a 16 bit binary value.

(4) : Converts two's complement binary value to decimal equivalent.

(5) : Displays output with original binary value, followed by an arrow, and finally a signed decimal value.

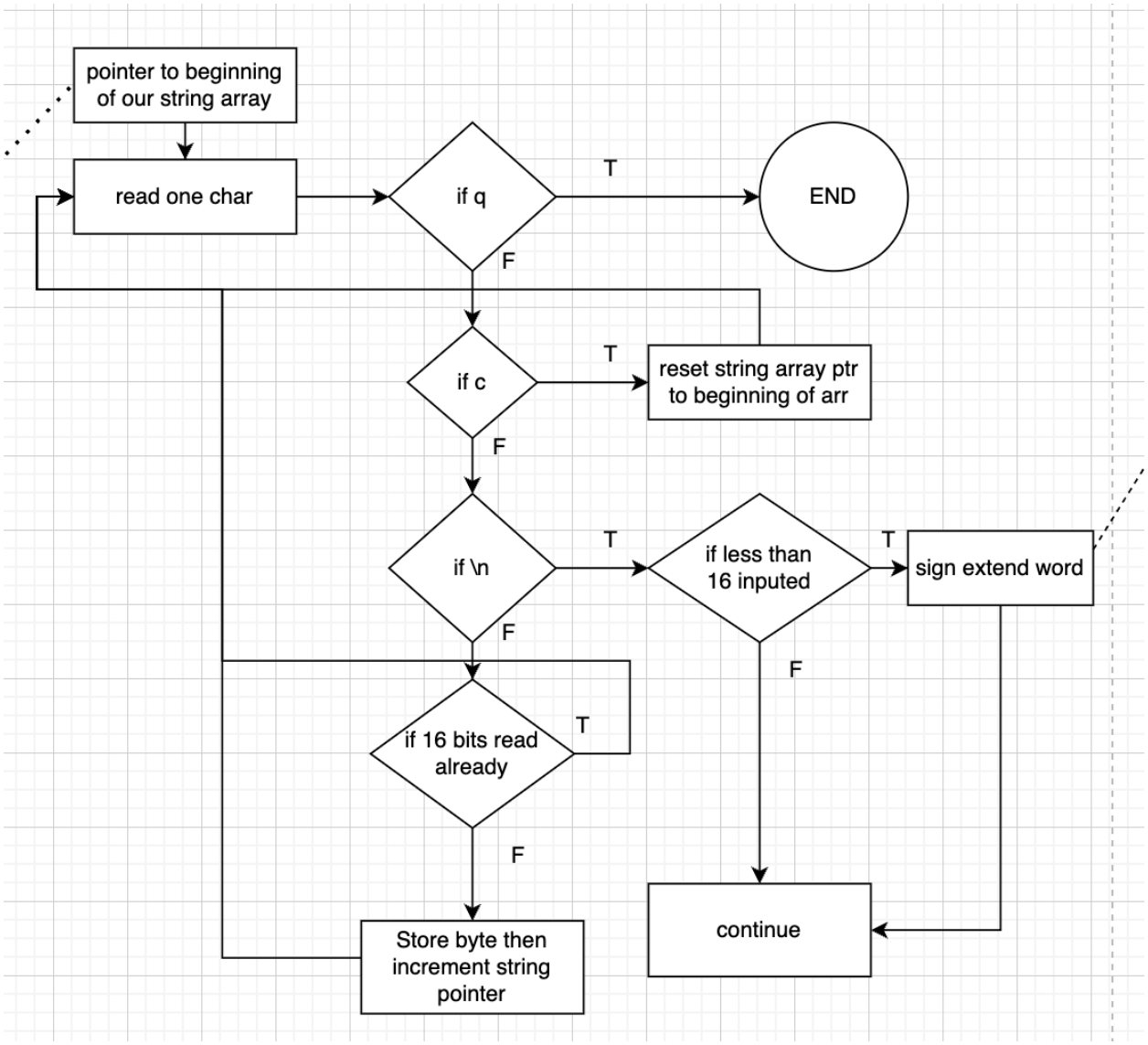
Input

The program begins with the input handling step, in which the user is given six options with which to manipulate their input (0, 1, backspace, enter, c, and q). Once the user hits enter, the program utilizes supervisor call #63 to read from the input buffer and store 64 bytes into a temporary string address used for processing. Next, we loop through each character inside the temporary string array, checking for our action commands (c, q, Newline/Enter) or ascii values 0 and 1.

Processing:

- ➔ If we encounter a character that is not any of these valid keys, we will raise a warning and ignore the key before moving on to processing the next character.
- ➔ If the current character is a 0 or 1, we will move this character from our temporary string array into the next open position in our return array, increment the amount of filled positions, and open the next position in our return array for the next 0 or 1.
- ➔ If the current character we are reading is ‘c’, we will update our command flag to -2, signaling an input clear, finish transferring/clearing the input buffer, and ignore the entire line of input. Finally, we will reset our parsed return-string array pointer to its original position and prompt the user for a new line of input.
- ➔ If the current character we are reading is ‘q’, we will update our command flag to -1, signaling program termination.
- ➔ If the current character we are reading is a newline character, or ‘\n’, this means we have reached the end of the user input. We stop reading from our temporary string array and check the length of the parsed array that has been returned.

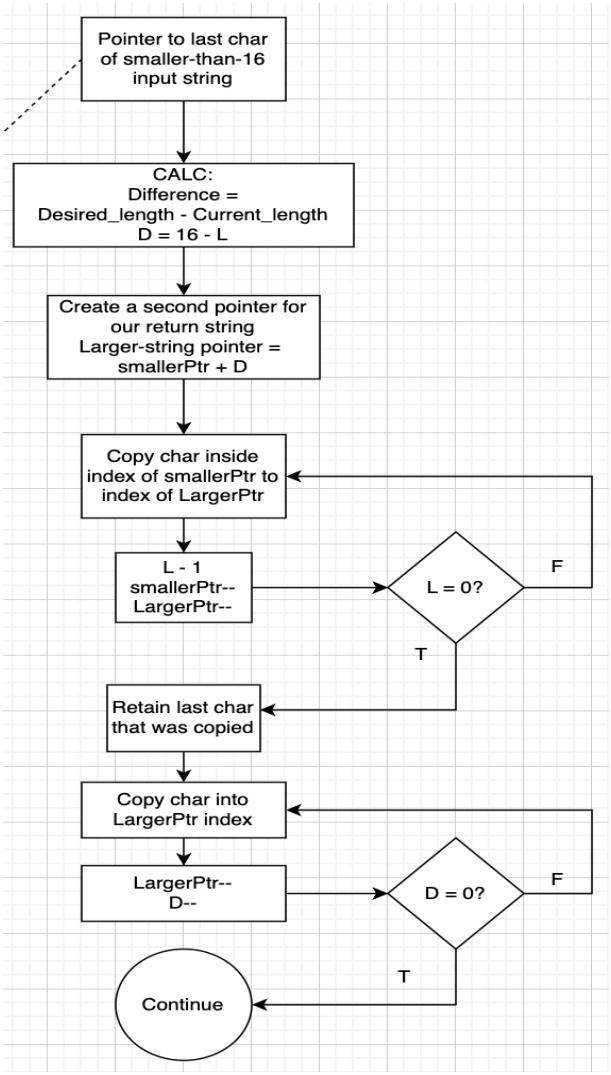
If the length of our parsed array is 16, great, we move on to conversion. If the length is less than 16, we need to sign-extend our binary input so it is 16 bits.



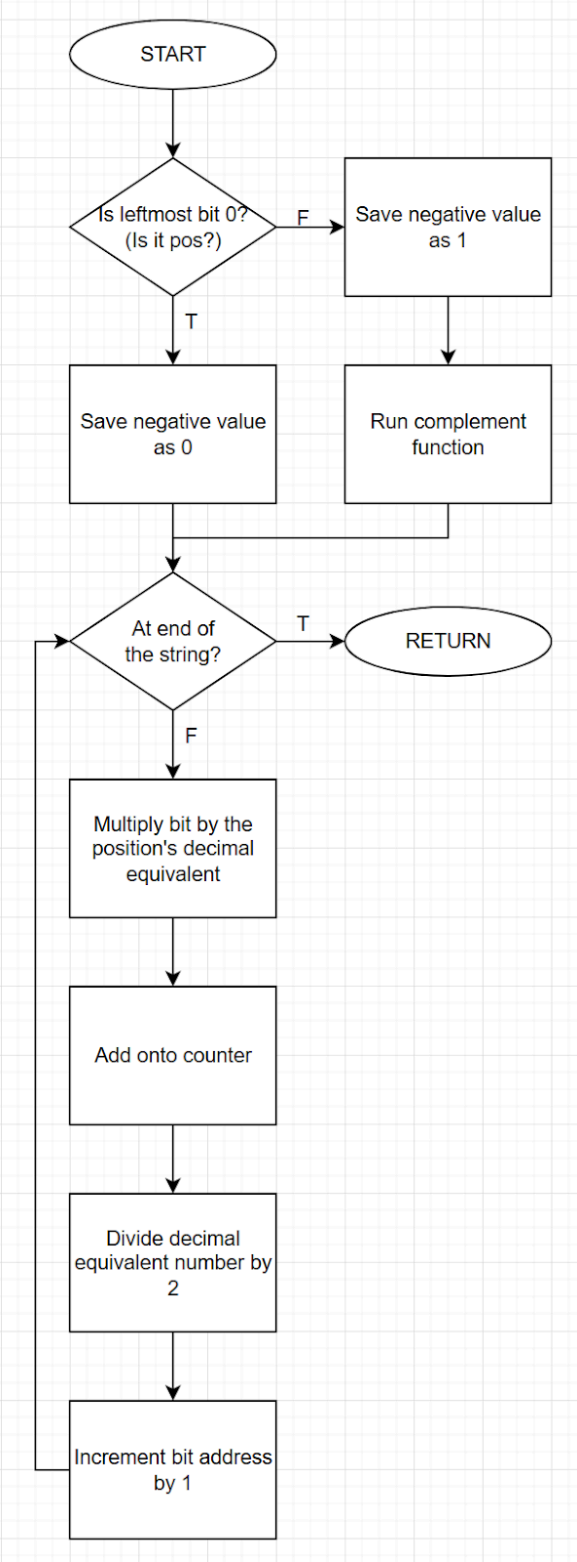
Sign Extension:

In the event that the final parsed input value is less than 16 bits we will call the prependStr function. This function looks at the difference between our desired length (16) and the length of our current string (L) and will prepend the first character by that

difference (D = 16 - L). The function uses a decrement pointer algorithm resembling a queue to store the current smaller string into a larger fitted string, which prevents misalignment issues. It starts from the last digit in our current string and copies that to the last position in our desired length string. Then it decrements the current string pointer and updates the last position of our larger string to be one before our previously copied value. This will execute L times. Once we have reached the first value of our smaller string, we retain that value, then continue copying it D times until we reach the beginning of our larger string. Return and begin conversion.

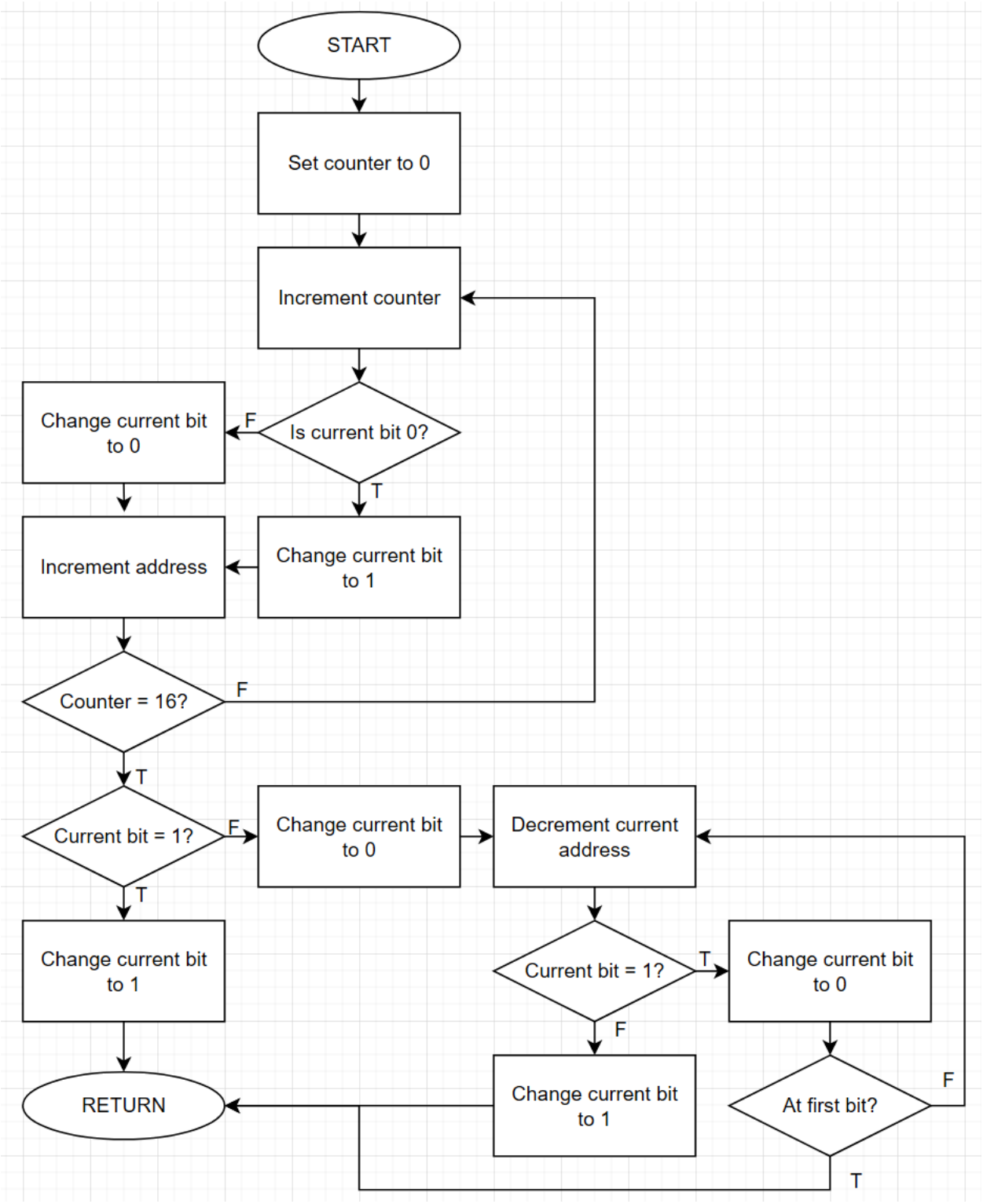


Convert Function



The Convert function is responsible for taking in the input of binary numbers as a string and converting it to a proper integer for outputting. Convert takes in the address of the null terminated string as an input, and it outputs both the decimal equivalent number and a boolean representing if the number is a positive or a negative. The program starts first by looking at the leftmost bit. Because the number is a signed int, it would be a 0 for a positive number or 1 for a negative number. If Convert finds that the binary number is a positive, nothing has to be done, as the boolean variable is already set to 0. If Convert finds that the binary number is a negative, it sets the boolean variable to 1 and runs the complement function to get 2s complement of the number. We set the decimal equivalent number to 32768 to get the decimal equivalent value of the first bit to process and we reset the total decimal number to 0. Then, we enter the loop. We first check if the current position of the address is a null character. If it is, then we exit the loop, as we have reached the end of the string. Otherwise, we subtract ASCII 0 from the current value to get a simple 0 or 1 integer for multiplying the decimal equivalent number. After multiplying, we add the result to the decimal total. We then divide the decimal equivalent number by 2, increment the current bit address by 1, and then restart the loop. Once the loop is finished, we end up with the decimal equivalent number.

Complement Function



The Complement function is a helper function for Convert to get 2s complement of the inputted binary number in the event of a negative number. As an input, we get the address of the string. Complement outputs the 2s complement of the string. First, Complement goes through every bit in the string, swapping the values of the bits. Then, Complement swaps the value of the last bit again. If the last bit was swapped to a 1 again, then there is no carry over to the other bits and we can return the function. If the last bit was swapped to a 0 again, then there is a carry, and prior bits need to be swapped. Complement then works its way backwards checking every bit along the way. If the current bit is a 1, then it is swapped to a 0 and we continue backwards unless we have hit the first bit. If the current bit is a 0, then it is swapped to a 1 and the carry is gone, in which case we can return the function.

Output

We now have our converted signed decimal value.

- ➔ Output our original 16 bit binary value.
- ➔ Followed by an arrow (-->).
  - ◆ Check if our conversion function returned a flag for negative value.
- ➔ If no negative flag was raised, simply output the converted decimal value.
- ➔ If a negative flag was raised, output a minus sign (-), then output our decimal value.