CSCI-21 Final programming project. Due before the start of the final exam.

You may turn off branch and load delays for this program, if you wish. It will make it considerably simpler. Be sure to turn on memory-mapped I/O. Make sure you fully understand the memory mapped I/O example program (`memmapIO.asm`) from my Web site before you start designing or programming this.

You may not use syscall for input or output, because the program would then block on the input. Instead, use an interrupt-driven I/O routine to handle all input and output. **<u>You must do ALL input and output through the interrupt-driven memory-mapped input/output services, NOT through syscall.</u>** **<u>You may NOT poll for I/O.</u>** The only syscall you may use for this program is a syscall with code 10 to end the program.

Write a program that starts with an arrays of characters (at least 60 characters in length), labeled '`source`' (holding an `.asciiz`) and a `.space` buffer labeled '`display`', bigger than the `source` string. Initially, make the `source` array contain a c-string with a '\n' before the terminating NUL. Include upper- and lower-case letters, digits, and other characters (e.g., punctuation and spaces) in the string. Start your program by copying the source array into the display buffer. Use a subroutine (using the "real-world" calling convention) to do this (passing in the bases of the buffers on the stack).

After copying the string, enable interrupts, and then loop, examining a variable (which may be changed by code inside the interrupt handler) until the user tells the program to quit. Whenever an output ready (transmitter) interrupt occurs, the interrupt handler will print the next <u>single</u> character from the `display` array, wrapping back to the beginning after it prints the '\n'. Whenever the input (receiver) interrupt occurs, the interrupt handler will extract the user's input (<u>one character</u>) and do one of the following tasks depending on the user's input:

's': sort the display array using any easy sort routine (bubble or ripple is fine). Do not sort the '\n'.

't': toggle the case of every alphabetic character (for example, 'T' becomes 't', 't' becomes 'T' and all non-alphabetic characters stay unchanged).

'a': replace the `display` array elements with the `source` elements. Do not use the subroutine you used in the main routine for this.

'r': reverse the elements in the `display` array (again, excluding the '\n').

'q': quit – set the variable being polled by the main loop, thus having the main code end the program.

Ignore any other character in input. Handle the commands from the user inside the interrupt handler, not inside the main program.

The 'q' command should just set the variable being queried (repeatedly) by the main program, which will cause the main program to then exit with a syscall with code 10. **<u>You MAY NOT use registers to send information between the program and the interrupt handler.</u>**

Note: a keyboard interrupt could happen in the middle of displaying the string, so the characters being displayed could change mid-line. This program is obviously designed to build on previous assignments, so you should reuse code wherever possible. The primary new feature is the interrupt-driven input and output. You will read and process each character as it arrives. Again, you MAY NOT use syscall for any I/O. This is about three or four pages of code in total.