Really Dumb Terminal Emulator

Communicating with a front panel computer using sense switches and LED's can get old fast. In the late 1970's, Lear Siegler came to our rescue with a modestly priced "interactive display terminal', the ADM-3A. In kit form it sold for $995. We called it a dumb terminal because it did little beyond echoing to a screen what was type on the keyboard. Graphics were non-existent and, being a serial device, speed was limited to 19,200 kilobits per second. Nevertheless it was sooooo much better than switches and LED's that it became a staple item for many PC pioneers.

I wanted to give users of the WhippleWay Front Panel Emulator a chance to experience keyboard-video I/O similar to the ADM-3A. The catch is that I did not emulate it in any great detail. If fact, my emulation is several degrees dumber than the basic dumbness of the ADM-3A. However, it should suffice to provide you with the basic idea of how such a terminal interfaced to front panel computer like an Altair 8800.

Below are the operating instructions needed to use the RDT (Really Dumb Terminal) along with a sample keyboard to screen echo program.

- The RDT functions within a text area bordered in red. You must click your cursor with the text area any time you want to use the terminal.
- When a key is pressed, bit 7 (the most significant bit) of a terminal status word changes from one to zero. The terminal status word can be loaded into the A register using the "IN 1" instruction. At this point the key struck is available as an ASCII character on input port 2. The "IN 2" instruction loads the character into the A register.
- For outputting, bit 6 of the terminal status word is one if the processor is "busy" outputting a previously output character. When the bit is zero, an "OUT 2" instruction outputs the character stored in the A register to the text area at the position of the cursor as indicated by the underscore "_" character.
- The text area will accommodate sixteen lines with the next (seventeenth) line scrolling up from the bottom. The first line is lost.
- Only three control characters are implemented:
    - Cntl Y(ASCII 25) – Clears the text area
    - Cntl Z (ASCII 26) – Backspaces on character
    - CR (ASCII 13) – Performs a new line.
      All other control characters are ignored but may produce effects as short cut keystroke s used by the browser or Windows.
- Since the data stored in the text area is not available, each keystroke must be captured byh your program and handled appropriately including echoed to the text area.
- The program "Simple_I-O.asm" demonstrates how a basic input – output echo would work. It works this way:
    - The clear text area is moved immediate to the C register and execution branches to the output portion of the program.
    - The output busy bit is checked by AND'ing the terminal status word (IN 1) with 0x40 in which all bits are zero except bit 6. The result of the AND'ing will be a one if output is busy, in which case, jump on not zero (JNZ) branches back to input the terminal status word again. The program loops here until the output is no longer busy and the anding

yields a zero, at which time the character is moved from the C register to the A register (MOV A,C) and outputted (OUT 2).
    - An unconditional branch (JMP) back to the inputting portion of the program places processing in a loop exactly like the output except the wait now is for bit 7 of the terminal status word to go to zero indicating a keystroke has occurred.
    - The IN 2 instruction loads the keyboard character into the A register and from there it is moved to the C register for preservation. Note that if left in the A register, it will be overwritten in the outputting loop and lost.
    - The program falls into the outputting loop where, when output is not busy, the C register will be output and the process begins again.
- Of course, much more programming is required to handle keyboard input. In a command line interpreter, for instance, keystrokes are saved in a memory buffer and then processed when a CR (carriage return) is detected. This would include detecting and handling backspacing and screen clearing within the buffer.

That's all folks!