CESCA Keyboard Interface:

Contents of the keyboard controller:

Invalid input: No key has been pressed since the last check, or it's still being processed. It can be easily detected using CMP-IN 0xFF followed by a JZ (see macros in the main document).

Input register (8 bits)							
0	0	0	0	0	0	0	0

ASCII character: The pressed key corresponds to an ASCII character. If the input expected a number, you can subtract 48 (ASCII code for '0') to the input. Note the following:

- The NULL character is interpreted as the invalid input.
- The Enter key is considered a special key instead of an ASCII code for CR or LF.

Input register (8 bits)	
0	ASCII code (7 bits)

Special keys: If a program only expects to receive the arrow keys and other special keys, this format allows it to easily read them using an immediate mask (CMP-IN):

Input register (8 bits)							
1	Custom	Enter	Del	□	□	□	□

Del = Backspace key (delete)
Custom = Assign custom key by pressing **Ctrl+Alt+(custom key)** when that key is pressed, it will be treated as a special key instead of ASCII. Press **Ctrl+Alt** or reset the Arduino to remove it.
The following keys can't be mapped as a custom key: Ctrl, Alt, Enter, Del, arrow keys.

Keypress buffer: The Arduino keyboard controller has an input buffer, so an input can be accepted even if the previous one is still being processed. By default (every time the Arduino gets reset), the size of this buffer is 8 inputs (1 being processed and 7 waiting), and once it's full no more keypresses will be accepted.

- The buffer can be **wiped** by pressing **Ctrl+Backspace**. The waiting keypresses will be deleted, but the one being processed won't be wiped to avoid errors.
- The buffer can be **resized** by pressing **Ctrl+(number key from 0 to 9)**. The buffer will be resized to the corresponding power of 2 (0 disables the buffer, 1 resizes it to 2 inputs, 9 resizes it to 512 inputs). If the new size is smaller than the previous one, all the most recent inputs inputs that don't fit will be deleted.

Generic input controller:

The computer can accept input from any device, as long as its controller meets these requirements:

- **Must appear as a register:** The computer expects an 8 bit “Input register”, with outputs connected directly to the bus and in a high impedance state by default.

- **Invalid input:** If Input register contains 0x00 as its value, the input is considered invalid. This means the user hasn’t entered any input yet, or that the controller is still processing it. The controller can decide how to represent its data using 8 bits, as long as it’s not with a 0x00.

- **Takes 2 control signals:**
    - **Inp (Input):** Output enable signal for the Input register. The contents of the register are published to the bus for as long as Inp is high.
    - **Ack (Acknowledge):** When Ack is high, the computer is indicating that it has finished processing the current input. The contents of the Input register must get set to 0x00 and the controller can start accepting a new input, or displaying the next input in its buffer.

  Note that Ack will only be active if the current contents of the Input register aren’t 0x00, and Ack and Inp will never be active at the same time.

- **Timing:** The controller must follow these timing requirements:
    - **Enable output:** The contents of the bus will be read half a clock cycle after the Inp signal gets set to high.
    - **Disable output:** After the Inp signal gets set to low, the computer waits one clock cycle before writing anything to the bus to avoid bus contention.
    - **Ack hold time:** The Ack signal stays high for exactly one clock cycle. The check frequency of the controller needs to be higher than the computer clock to avoid missing it. Also, the controller should make sure it only interprets the Ack once (by waiting until Ack is low again) to avoid skipping an input.
    - **Ack to Inp:** When Ack changes from high to low, the Inp signal will stay low for at least two clock cycles. By then, the contents of the Input register must be either 0x00 or the next input.

  With the slow clock speeds of such a simple architecture, I expect the controller to be orders of magnitude faster than the computer, so there shouldn’t be any timing problem. Of course, using interrupts would be a much more robust solution.

All those requirements can be accomplished using a real register and some logic ICs (for simple devices), or with an Arduino faking to be a register (for complex devices such as a keyboard).