

CESC16 Keyboard Interface:

Getting input on the CESC16 Computer:

The computer can access 4 peripherals, mapped at memory addresses 0xFF00 - 0xFF03. By default, address 0xFF00 is a PS/2 keyboard (using an Arduino controller) and user code can access it by performing `lw/sw` instructions to that address. However, it's recommended to avoid accessing it directly, and instead rely on system calls for I/O.

The full list of syscalls can be found on [DOCS/syscalls.md](#); here are the calls needed for getting input:

syscall INPUT.Get If the user has pressed a key since last check, `v0` returns it (see Keyboard interface below). Otherwise, `v0` contains all zeros (0x0000).

syscall INPUT.GetFull If the user has pressed or released a key since last check, `v0` returns it (see Keyboard interface). Otherwise, `v0` contains all zeros (0x0000).






syscall INPUT.Wait Halts program execution until the user presses a key, then `v0` returns it (see Keyboard interface below).

syscall INPUT.WaitFull Halts program execution until the user presses or releases a key, then `v0` returns it (see Keyboard interface below).

Note that none of these syscalls take any arguments and temporary registers are preserved (only `v0` is modified).

Keyboard interface:

After performing one of the syscalls above, `v0` will contain the following:

Input (16 bits)									
0	BR							0	ASCII code (7 bits)

- **Bits 7..0** contain the ASCII code of the pressed/released key (if the key doesn't have one it's all zeros). On the default terminal the higher 8 bits get truncated, so you can send the keyboard input directly to the terminal output and only the ASCII will get displayed.
- **Bits 13..8** are associated with special keys (commonly used in games). The bit is set to 1 if that key is pressed/released, and 0 otherwise. This allows fast detection by using a mask.
 - Note that *Enter* and *Backspace* will return an ASCII code and a mask bit simultaneously.
- **Bit 14** (BR) indicates Key Break:
 - It's set to 1 if the key has been released (break), and 0 if it has been pressed down (make).
 - When using **INPUT.Get** or **INPUT.Wait**, this bit will always be set to 0.
 - When using **INPUT.GetFull** or **INPUT.WaitFull**, keystrokes will be received twice: when they're pressed down (bit 14 is 0) and when they're lifted up (bit 14 is 1).

Remarks:

No input: After calling `INPUT.Get` or `INPUT.GetFull`, `v0` can contain 0x0000 instead of a valid input (meaning no new key has been pressed). This can be easily detected with `test v0` followed by a `jz`.

Phantom inputs: `INPUT.Get` and `INPUT.Wait` do not remove the "lift up" version of their returned keystroke! When calling `INPUT.GetFull` or `INPUT.WaitFull`, the first detected input will probably be the remnants of the last call to `INPUT.Get` or `INPUT.Wait`.

Typematic Repeat: If a key is kept down, the keyboard will continue sending keystrokes at regular intervals. The key break will always be sent only once.

Acknowledge signal: The keyboard controller won't start processing the next input until the current one has been acknowledged. An input is acknowledged by attempting to write (any value) to the input register (performing a sw). The syscalls already take care of this process, user code doesn't have to.

Keypress buffer: The Arduino keyboard controller has an input buffer, so an input can be accepted even if previous ones are still being processed. The input buffer can be wiped by restarting the Arduino.

Generic input controller:

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

- **Control signals** (both will never be active at the same time):
 - /OE: Output Enable for Input Register (active low)
 - /Ack: Acknowledge, asynchronous clear for Input Register (active low).
- **Outputs:**
 - 16 bit Data Bus: If /OE is low, outputs Input Register. If /OE is high, it's in High-Z state.
- **No input:** If the user hasn't entered any input yet, or the controller is still processing it, the Input register must contain 0x0000. The controller can decide how to represent its data, as long as it's not with 0x0000.

Example: Implementation of the PS/2 keyboard controller

