Hardware support for system calls

Your friend is designing a new processor instruction set and asks for advice regarding the `syscall` instruction. He hesitates between two designs:

Design 1: The `syscall` instruction takes 1 argument: the address of the kernel-level function to be called (for example, the address of the `open` function). When the instruction is executed, the processor switches from user mode to kernel mode, saves the return address and jumps to the address provided as argument.

Design 2: The `syscall` instruction takes 1 argument: an integer that corresponds to the unique identifier of the function (for example, by convention, `open` may be associated with id 2). When the instruction is executed, the processor switches from user mode to kernel mode, saves the return address and jumps to the address of a syscall handler. The syscall handler is in charge of analyzing the syscall id and jumping to the corresponding function. The address of the syscall handler is configured via a privileged CPU register.

For both designs, the other arguments of the syscall (for example, in the case of `open`, the name of the file to be opened and the access flags) are passed via the CPU registers or the stack.
Which of the two designs seems better to you? For which reason(s)?