CHAPTER 10: INPUT/OUTPUT ORGANIZATION

A computer must be able to interact with the outside world or it cannot perform useful work. A computer may use asynchronous data transfer to perform this interaction, with either the source or destination initiating the data transfer, with or without handshaking. The computer may also use programmed I/O, either memory mapped or isolated, to interact with I/O devices. In any of these forms, the CPU must include the logic needed to access I/O devices and to process any I/O instructions in its instruction set.

There are four types of asynchronous data transfers, distinguished by whether the source or destination initiates the transfer, and by whether or not handshaking is used. Source-initiated data transfers without handshaking are where the source outputs its data, and then strobes a control signal for a set amount of time. The destination device reads in the data during this time. The source device next deasserts the strobe and stops outputting data. Destination-initiated data transfers without handshaking are where the destination device transmits a data strobe signal to the source device which, after a brief delay, makes data available. After a set delay to ensure that valid data is ready, the destination device reads in this data and deasserts the data strobe. This in turn causes the source to stop transmitting valid data. This is analogous to the classroom situation in which a student (the destination) asks the professor (the source) a question. The professor then gives the answer (the data) and then continues lecturing, not waiting for confirmation that the answer was heard by the student.

For the next two types of asynchronous data transfers, we need to know what handshaking is. Handshaking is an automated process of negotiation that dynamically sets parameters of a communications channel established between two entities before normal communication over the channel begins. Handshaking uses an additional control signal to indicate that data is ready or has been read in. Now, the third type of asynchronous data transfer is source-initiated data transfer with handshaking. The source sets the data request signal high and then makes valid data available to the destination device. After the requisite delay to allow the data to stabilize, the destination device reads in the data. In practice, this is a minimum delay; the actual delay depends on how quickly the destination becomes ready to accept data. Once the destination device has read the data, it sends a data acknowledgement signal to the source. This tells the source that the destination has read in and no longer needs this data. The source sets its data request line low and stops sending data. The destination then resets its data acknowledgement signal. Going with the previous classroom example, the professor (source) says she will write something on the blackboard that the students (destination) should know for the upcoming test, and that they should copy it into their notes. She then writes the information (data) on the blackboard, and leaves it there until the students say they have written it down (data acknowledgement). She then erases the blackboard and continues her lecture.

The final type of asynchronous data transfer is destination-initiated data transfer with handshaking. It is similar to source-initiated data transfer using handshaking, except that the data acknowledgement signal is replaced by a data-ready signal. In yet another classroom analogy, the students ask the professor a question; she writes an answer on the blackboard and the students copy it down. After the students tell the professor that they are done, she erases the board and continues her lecture.

A method used in smaller systems to alleviate the problem of I/O devices with variable delays is called polling. In polling, the CPU sends a request to transfer data to an I/O device. The I/O device processes the request and sets a device-ready signal when it is ready to transfer data. The CPU reads in this signal via another I/O address and checks the value. If the signal is set, it performs the data transfer. If not, it loops back, continually reading and testing the value of the device ready signal. While polling is relatively straightforward in design and programming, and is often used where the CPU is not fully loaded, it is not appropriate for systems in which CPU time is at a premium. One way of resolving this is to use wait states. When wait states are used, the processor requests data from (or permission to send data to) an I/O device, which then asserts a wait signal that is sent to the CPU via the control bus. As long as the signal is asserted, the CPU stays in the wait state, still outputting the address of the I/O device and the values of the control signals needed to access the device, but not doing anything else. The I/O device continues to assert this wait signal until it is ready to send or receive data. Once it is ready, the I/O device deasserts its wait signal and the CPU completes the data transfer.

To make the most of the wasted CPU time while it waits in polling or wait states, interrupts were developed. Interrupts are an efficient method used by computers to interact with I/O devices. Instead of spending time polling I/O devices, the device issues an interrupt when it is ready to transfer data, allowing the CPU to perform other useful work while waiting for the I/O device. When processing an interrupt, the CPU first finishes executing its current instruction, then saves its current state, and finally invokes the interrupt’s handler routine. A system can have more than one interrupt; multiple interrupts can be prioritized using daisy chaining or parallel priority hardware. In addition to external interrupts, a CPU may also support internal interrupts and software interrupts.

Direct memory access is used to speed up data transfers between memory and I/O devices. The CPU first sends information to a DMA controller, including the number of bytes to be transferred, the starting address in memory, and the direction of the data transfer, and then sends a start signal. The DMA controller obtains control of the system buses from the CPU and coordinates the transfer of data directly between memory and the I/O device. Bypassing the CPU greatly speeds up the data transfer. DMA controllers may transfer data in burst mode, cycle stealing mode, or transparent mode. I/O processors often use DMA to speed up their data transfers.

Serial communication is used to transmit data between a computer and an external device one bit at a time. Asynchronous devices, such as modems, often use UARTs to coordinate the parallel data transfer within the computer and the serial data transfers with the outside world. Synchronous serial communications can reduce the overhead associated with asynchronous communication. Standards have been developed for both asynchronous and synchronous communications, including RS-232-C and USB.