SUSPENDING OPERATIONS

We briefly introduced the concept of 'interrupt handling' when we reviewed the Toshiba HX-10 (see page 669). These are messages that interrupt the task that a processor is currently performing, in order to convey important information to it. Here, we explore the interrupt mechanism in detail.

One common application of interrupts is when we are dealing with input from the keyboard. If a program directly accesses the keyboard — usually via the operating system — to obtain the next input character, then any key that is pressed while the program is doing something else will be lost. Even when the processor is fully engaged in processing keyboard input, it is still possible for it to lose a character, especially one that follows a character that needs extra processing, such as a carriage return.

The solution is for the keyboard to interrupt the processor whenever a key is pressed, so that the processor stops what it is doing and performs an 'interrupt service routine'. This takes the character that has just been input and places it in a section of memory reserved as a *keyboard buffer*. The processor can then return to whatever it was doing and carry on as though nothing has happened.

Whenever the operating system keyboard input routine is called, it does not look at the keyboard directly but takes the next character out of the buffer instead (waiting for one to appear if the buffer is empty). This mechanism enables the user to 'type ahead' of what actually appears on the screen, and should ensure that no characters are lost.

There are, however, two possible problems. The user may type so quickly that the buffer fills up faster than the program can deal with the input, thus causing the buffer to overflow. The solution to this requires a compromise between allowing sufficient memory for an adequate-sized buffer and not wasting too much valuable memory space. The second problem arises with those users who feel uncomfortable when a character does not appear on the screen immediately a key is pressed. They may keep pressing the key, and thereby generate dozens of characters that then go into (and again may overflow) the buffer. This problem is usually solved by familiarity with the computer.

Another useful application for interrupts occurs when output is sent to a printer — which is often one of the most time-consuming operations a micro needs to perform. During printing, the processor may be required to work for 100 microseconds while it sends a character to the printer, and then wait thousands of microseconds for the printer to process that character. A *spooling* system is one answer to this: it places the files to be printed in a queue, and part of the first file in the queue is loaded into another buffer area of memory. The port that serves the printer will interrupt the processor whenever the printer is ready for another character to be sent. The interrupt service routine will then send the next character from the buffer, or (if the buffer is empty) load the next section from the file at the head of the queue into the buffer. In this way, printing can be going on in the background, while the processor is free to get on with something else.

TYPES OF INTERRUPT

There are some operations that the processor performs — such as accessing disks — where being interrupted can cause data loss or some other catastrophe. There must be, therefore, a mechanism for *masking* interrupts so that the processor can ignore any that occur during a particularly sensitive operation. If this is the case, it is preferable that a note is made to indicate that the interrupt has occurred, so that it can be dealt with later.

Conversely, if we are dealing with a disk interface that is interrupt-driven, then its

Interrupt One
When an interrupt occurs, the processor completes execution of the current instruction, and stacks the current contents of the program counter. The interrupt vector address appropriate to the interrupt is then loaded into the program counter, and control passes to that address — usually in ROM. This address points to another address — usually in RAM — where a JMP instruction directs control to the actual interrupt service routine. This is terminated by the RTI instruction, which passes control back to the main program via the return address on the stack. Since the JMP instruction is stored in RAM, it can be found and changed by the programmer so that on an interrupt, control passes first to a special-purpose user routine and then to the normal service routine