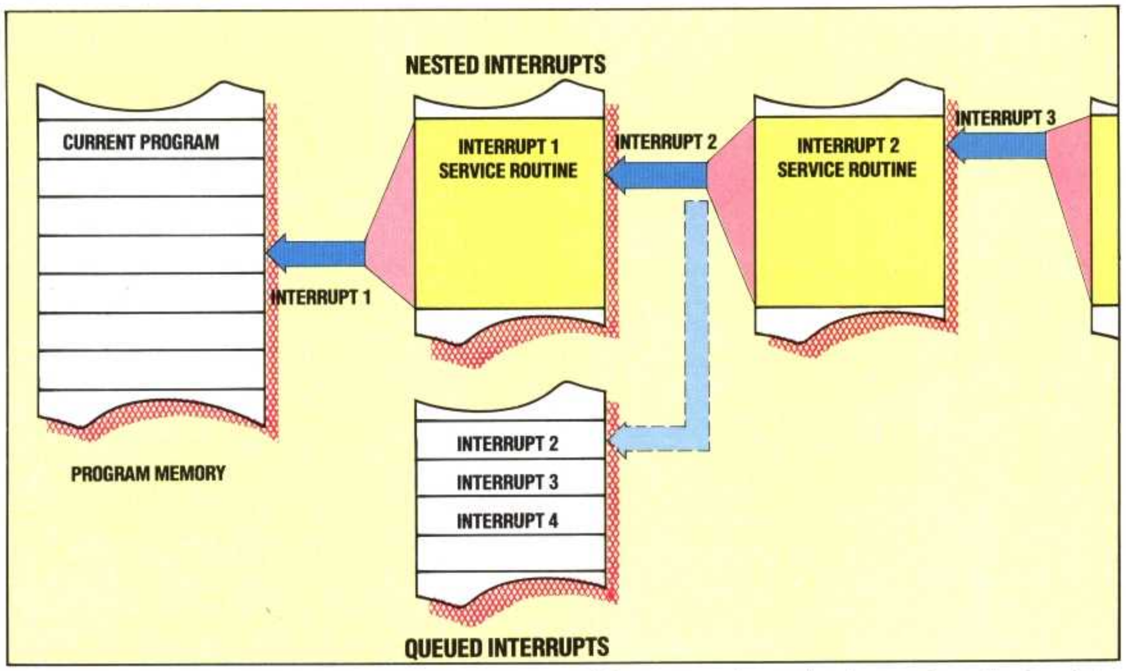

Interrupted Interrupts
If an interrupt occurs during an interrupt, one possible solution is for the processor to 'nest' the interrupts: whenever an interrupt occurs, PC is stacked and the new interrupt is handled immediately and control returns to the stacked address. The limits on this nesting are the capacity of the stacks, and the ability of the interrupt-generating devices to withstand delays in processing their interrupts.

An alternative is for the processor to stack the details of any interrupt on an interrupt queue. When the first interrupt is complete, the processor inspects the queue, processes any interrupt it finds there until the queue is empty, and eventually passes control back to the program

interrupts would be given priority and under no circumstances masked — thus giving rise to the concept of a *non-maskable* interrupt. Such an interrupt might come from a circuit that detects a drop in the mains supply voltage: its service routine would immediately start saving the current task while power remained.

When interrupts can occur from more than one source, we must consider the possibility of *nested* interrupts. If an interrupt occurs while the processor is in the middle of servicing another interrupt, there are two possible strategies for handling it. First of all, the new interrupt could be ignored until the current one is completed. Secondly, interrupts could be ranked on a scale of urgency, so that a high-priority interrupt could override the handling of one with a lower priority. In this case, the operating system would have to be able to deal with the nesting of interrupt service routines.

SOFTWARE INTERRUPTS

The SWI instruction, which we briefly mentioned on page 577, can be used in a program as a convenient way of returning to the operating system by generating its own interrupt — called the 'software interrupt' (as distinct from the hardware-generated interrupts we have been discussing so far). SWI instructions can also be used to act as breakpoints in a machine code program to aid in debugging; this facility is provided by many ROM-based machine code monitors, as well as debugging packages. The user chooses points in the code where program execution is to pause, and the instructions at these locations are replaced with SWIs. When the program is run, the interrupt service routine then allows the programmer to inspect and possibly alter the contents of registers and memory locations, and see exactly what the program is doing. When execution is resumed, the monitor/debugger replaces the instruction displayed by the breakpoint SWI, and continues with the program from that point.

The 6809 has three separate interrupt mechanisms: $\overline{\text{IRQ}}$ (Interrupt ReQuest), $\overline{\text{FIRQ}}$ (First Interrupt ReQuest) and $\overline{\text{NMI}}$ (Non-Maskable Interrupt). These are all activated by the appropriate signal being received on three pins on the processor chip. The bar above the name (in $\overline{\text{IRQ}}$, for example) indicates that they are activated by a zero signal at the processor, rather than a one. These three pins are connected to the main bus so that peripheral chips like the 6820 and 6850 can have their interrupt request output pins connected to the same bus lines. When the chips are programmed, the interrupts can be enabled and then the appropriate signals will automatically be sent.

There are also three software interrupts caused by the SWI, SW12 and SW13 instructions.

When an interrupt occurs, control passes to the *vector* address contained in a specific location at the top of memory. These vector addresses are usually found in ROM, so control will always pass from there to the same fixed address. However, this address will normally be in RAM and will contain a JMP instruction, so that the final destination can be changed to the user's own service routine. The memory locations are:

Interrupt Type	Vector
$\overline{\text{NMI}}$	$FFFC
SWI	$FFFA
$\overline{\text{IRQ}}$	$FFF8
$\overline{\text{FIRQ}}$	$FFF6
SW12	$FFF4
SW13	$FFF2

It is also worth noting that the top two bytes of memory — $FFFE and $FFFF — contain the *reset*