IAN McKINNELL

COURTESY OF AIR CALL

Indirect Access

Vital though indirect addressing is to computer operations, it is difficult to find examples of the technique in real life. A reasonable analogy, however, is a radio paging service. When someone wants to talk to a subscriber, he doesn't call them directly, because they could be anywhere. Instead, he calls the paging service, who then page the subscriber. This is a simple, flexible system in which the paging service provides indirect accessing (or addressing) of its subscribers

character on the screen. Most machine code programs will need to use these routines at some time. In most cases, these routines will be accessed by using a jump table, which means that routines pointed to by the jump table vectors may be changed, or relocated in memory, without changes being needed in the programs that use them. In other words, such routines are always accessed indirectly, through the appropriate pointers in the jump table. When a new version of an operating system is designed, or an updated ROM produced, it is rare for these primitive OS routines to remain in their original positions; but if the jump table remains in position, with its pointers altered so that they reflect the new OS routine addresses, any software written for the old operating system that uses the jump table will run unchanged on the new system.

A common technique used in many operating systems is to have one entry point and to make all subroutine calls to this one address. One of the CPU registers is used in addition to pass a function code that is used to determine which subroutine will be called. This function code is used as an index or offset into the appropriate vector of the jump table, and control is transferred through this pointer to the desired routine.

As an example, let us suppose that we have four Kbytes of ROM, located at $F000, the first 256 bytes of which ($F000 to $F0FF) contain a table of up to 128 addresses of subroutines stored in the ROM. The entry routine (the address by which all the OS routines are addressed) is located at $F100, and this expects a function code in the range 0 to 127 to be stored in accumulator B; this code is used by the entry routine to pass control to the appropriate subroutines and thence back to the calling program once execution is complete. The calling routine for function number 1 is:

LDB #1	put the function code in B
JSR $F100	call the entry subroutine

The entry routine itself is:

LDX $F000	start address of the jump table
LSLB	shift B one place to the left (equivalent to multiplying the contents of B by two) since each entry in the table is two bytes long. Thus the pointer appropriate to function code 1 is stored at $F002 and $F003, while the pointer for code 2 is at $F004 and $F005, and so on
BRA [B,X]	transfer control to the address found at the Bth position in the table

Note that the transfer to the routine is handled by a BRA (or JMP) rather than a BSR (or JSR); this is so that the RTS at the end of the OS routine will return control directly to the calling program instead of back to this entry routine.

Our next example shows a further possible use of indirect addressing in dealing with a memory-mapped display screen; on many micros the screen memory occupies a block of the main memory and may be accessed directly if extra speed is required. For simplicity, let us assume that the screen occupies a block of memory from $E000 to $E3FF, representing 16 lines of 64 characters. The position of the cursor is a 16-bit value in this range, and is located at $E400. The first subroutine clears the screen by writing a space character (ASCII code 32) at each character position. The second subroutine will write the