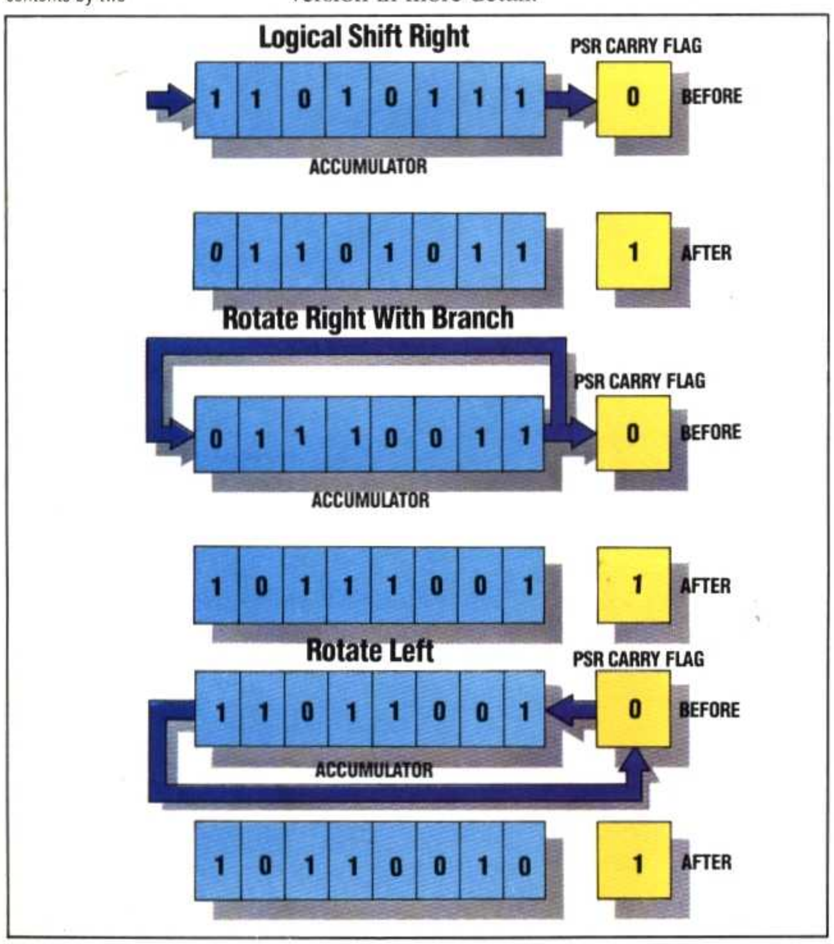
We know that if we do the lo-byte subtraction first the result is $7B, and a carry. That carry is then added by the SBC instruction to $21, making $22, which is then subtracted from $37, giving $15. The answer, $157B, can be seen to be correct by checking the decimal version.

Two-byte arithmetic on the Z80, therefore, follows this simple procedure:

1) Clear the carry flag.
2) Subtract the lo-bytes with carry.
3) Subtract the hi-bytes with carry.

The 6502 version of this sequence differs in the first particular — the carry flag must be set to permit a 'borrow' out of the lo-bytes from the hi-byte. If no borrow occurs, then the subtraction proceeds as normal, and the carry flag remains set for the subtraction of the hi-bytes, which should similarly proceed normally. If an underflow occurs in the lo-byte subtraction, however, the carry flag acts as the 'ninth bit' of the accumulator. This ensures that a correct result occurs there, and that the carry flag is then reset. When the hi-bytes are subtracted with a reset carry flag, the effect is the same as in the Z80 hi-byte subtraction with the carry flag set — the number to be subtracted is decremented before the subtraction takes place. Both methods of dealing with the subtraction borrow have their equivalent in the old-fashioned arithmetic methods of 'borrowing' here, and 'paying back' there. Let's consider the 6502 version in more detail.

Shift Work
The Shift and Rotate instructions are used primarily to examine the contents of a register bit by bit. With each shift, the top or bottom bit of the register is moved into the PSR carry flag; the state of the carry flag can then be used by a branch instruction to determine the flow of program control (as can be seen in the multiplication subroutine in this instalment). The rotate instructions can be used so that register contents are preserved, but the logical shift instructions shift zeros in as they shift bits out. A left shift, therefore, multiplies the register contents by two, and a right shift divides the contents by two



If we clear the carry flag, and subtract $E4 from $5F, the result is $7A in the accumulator, and the carry flag remains clear. We have seen from the Z80 example that a 'true' result is $7B with the carry flag indicating a negative number. $7B is the two's complement of the 'real' answer (-$85). We can see that $7A is the one's complement of this number, and that the state of the carry flag, therefore, is a kind of switch on the accumulator's mode. That is to say, it is set for two's complement, and reset for one's complement.

If we do the subtraction on the 6502 with the carry flag set, then the accumulator contains $7B, and the carry flag is reset. If this is a two-byte subtraction, putting the carry flag into reset state will ensure that the hi-byte subtraction result is decremented, thus taking care of the 'borrow' from the lo-bytes.

MULTIPLICATION

Consider the decimal multiplication sum:

174	Multiplicand
×209	Multiplier
1566	1st Partial Product
000	2nd Partial Product
+348	3rd Partial product
36366	Final Product

You don't have to understand positional notation to use this method, you just have to be able to follow simple procedures and do single-digit multiplication. The heart of the method is the writing of each partial product one place to the left of the previous product (the empty columns are left blank here for emphasis). Once the necessity for this is accepted, then forming the partial products requires only a knowledge of the multiplication tables.

The combination of shifting partial products and rote learning of tables is what makes decimal long multiplication difficult for many people. There is only one real product in binary multiplication, and that is one times one; all other single-digit products result in zero. Consider this binary long multiplication sum:

1101	=	13 decimal
1001	=	9 decimal
1101		1st Partial Product
0000		2nd Partial Product
0000		3rd Partial Product
1101		4th Partial Product
1110101	=	117 decimal

The shifting of partial products is clearly seen in this example, as is the overall simplicity of multiplication in binary. A partial product is equal to either zero or to the shifted multiplicand, depending on whether the corresponding multiplier bit is one or zero. That immediately sounds like the sort of test we've become used to as