an Assembly language control structure. To perform binary multiplication, examine each bit of the multiplier in turn, and add zero (if the bit is zero) or the shifted multiplicand (if the bit is one) to the total. How, then, do we examine a single bit of the multiplier, and how do we shift the multiplicand?

Testing the state of a particular bit in a byte can be done using the BIT instruction on both the Z80 and 6502 microprocessors. On the Z80, this instruction takes an address and a bit number as its operands, and the zero flag is set if that particular bit is zero and reset if the bit is one. On the 6502, the operand is an address. The contents of this address are ANDed with the accumulator, and the zero flag is set or reset, depending on whether the result is false or true.

These instructions permit subtle programming, but neither method is particularly convenient here. It would be much more convenient if the bit in question could be made to act as the carry or zero flag, so that program flow would branch automatically according to the state of each bit in turn. Needless to say, both processors' instruction sets make that possible through the use of the *shift* instructions. As the name implies, these will also solve the problem of shifting the multiplicand.

There are a variety of shift and *rotate* instructions in both instruction sets, although the Z80's are more complex than those of the 6502. In general, their effect is to shift each bit in a register one position to the right or to the left. They differ in detail in their treatment of the end bits of the register — a bit must be shifted out of the register at one end while another bit is shifted in at the other end. If bit 7 is shifted out of the register and put immediately back into bit 0, then the operation is a *rotate left.* If bit 0 is shifted into bit 7 the operation is a *rotate right.* If this is done, then the contents of the register change in order, no new values are introduced, and after eight such rotations the register will be restored to its original state.

If rotation is not employed, then a destination for the shifted-out bit is necessary, and a source must be found for the shifted-in bit. Both are most often supplied by the various condition flags of the processor status register (PSR), and in particular the carry flag. In constructing a multiplication subroutine to multiply two single-byte numbers, we need to shift the multiplicand left and the multiplier right. The multiplicand bits must be shifted out into the hi-byte of the multiplicand while zeros are shifted into the unoccupied bits. The multiplier bits have to be shifted through a PSR flag for testing, but their destination, and the state of the shifted-in multiplier bits, is unimportant unless we need to preserve the contents of the multiplier. All that concerns us about the multiplier during multiplication is whether the shifted-out bit is one or zero.

Given, therefore, that the multiplier is stored at address MPR, the multiplicand at MPDLO, and the product at PRODLO and PRODHI, we can write these subroutines as follows:

EIGHT-BIT MULTIPLICATION

6502			Z80		
	ORG	$C100		ORG	$D000
START	LDA	#$00	START	LD	BC,(MPR)
	STA	PRODLO		LD	B,$08
	STA	PRODHI		LD	DE,(MPDLO)
	STA	MPDHI		LD	D,$00
	LDX	#8		LD	HL,$00
	CLC		LOOP0	SRL	C
LOOP0	ROR	MPR		JR	NC,CONT0
	BCC	CONT0		CALL	ADDIT
	JSR	ADDIT	CONT0	SLA	E
CONT0	ASL	MPDLO	ENDLP0	DJNZ	LOOP0
	ROL	MPDHI		LD	PRODLO
	DEX			RTS	
ENDLP0	BNE	LOOP0	MPR	DB	$E2
	RTS		MPDLO	DB	$7A
MPR	DB	$E2	MPDHI	DB	$00
MPDLO	DB	$7A	PRODLO	DW	$0000
MPDHI	DB	$00	ADDIT	ADD	HL,DE
PRODLO	DB	$00		RET	
PRODHI	DB	$00			
ADDIT	CLC				
	LDA	PRODLO			
	ADC	MPDLO			
	STA	PRODLO			
	LDA	PRODHI			
	ADC	MPDHI			
	STA	PRODHI			
	RTS				

As can be seen from this example, programming the Z80 is made much easier by its 16-bit registers and associated instructions. In particular, compare the ADDIT subroutine in the two programs. The 6502 version uses ROR to rotate the multiplier rightwards through the carry, and ASL and ROL to shift the multiplicand leftwards out of MPDLO into MPDHI through the carry. The loop is controlled by the X register as a counter.

The Z80 version uses SRL to shift the multiplier rightwards through the carry, and SLA and RL to shift the multiplicand leftwards in DE via the carry. The loop is controlled by register B as a counter. Notice that the ADD instruction not only supports 16-bit register arithmetic, but also is not affected by the carry flag — unlike ADC.

In the next instalment of the course, we will discuss methods of division, and consider various ways of controlling the screen display. This will complete the tutorial element of the course, and will be followed by 6502 and Z80 exercises and examples in future instalments.

Exercise 15

1) Write a multiplication subroutine using a 16-bit multiplicand and an eight-bit multiplier of your choice.

2) Multiplication is merely repeated addition: write an eight-bit by eight-bit multiplication subroutine that does not use the shift or rotate instructions.