THE GREAT DIVIDE

We conclude this series of machine code tutorials with a brief study of unsigned binary division and the use of operating system ROM routines in Assembly language screen display programming. In a summary of this introductory section of the course, we review the major themes and topics — from BASIC to branching, from arrays to assemblers.

Just as we used the manual long multiplication method as an algorithm for binary multiplication (see page 298), so the manual long division method is a model for binary division. Consider this binary long division:

```
     00001110 r00        quotient
1011)10011010            dividend
   -1011                 subtract divisor
    10000
   -1011                 subtract divisor
     1011
   -1011                 subtract divisor
      00                 no remainder
```

The essence of the method is the repeated subtraction of the divisor from the high order bits of the dividend. Depending on the result of this subtraction, a zero or a one is shifted into the quotient. The remainder is the result of the last subtraction of a divisor.

The various ways in which this algorithm may be implemented in Assembly language are not as apparent as they were for multiplication. However, as before, the Z80 version uses the power and flexibility of its 16-bit registers, while the 6502 must fetch and carry eight bits at a time. The divisor is in the address labelled DIVSR, the dividend in DVDND, the quotient in QUOT, and the remainder in RMNDR. The program in Z80 and 6502 Assembly language is given.

Notice in both cases that when the divisor is subtracted from the partial dividend with a negative result, the dividend must be restored by adding the divisor back in again. The 6502 version is noteworthy for its treatment of the processor status register after the divisor subtraction: the carry flag must be rotated into the quotient, but its state must also be preserved to indicate the result of the subtraction. Consequently, the PSR is pushed onto the stack before the rotation, and pulled off it afterwards, thus restoring the carry to its immediate post-subtraction state.

We have now considered the four rules of arithmetic — this is plainly worth doing as a programming exercise for the insight it brings to machine processes, but inventing all the various combinations of single- and multiple-byte arithmetic is unnecessary, given that programmers have been writing these routines in textbooks and magazines for years. When the need arises for variations of the routines that we have developed, they will be supplied or set as exercises.

16-BIT BY 8-BIT DIVISION

Z80			6502		
START	LD	A,(DIVSR)	START	LDA	#$00
	LD	D,A		STA	QUOT
	LD	E,$00		STA	RMNDR
	LD	HL,(DVDND)		LDX	#$08
	LD	B,$08		LDA	DVDHI
LOOP0	AND	A		SEC	
	SBC	HL,DE		SBC	DIVSR
	INC	HL	LOOP0	PHP	
	JP	P,POSRES		ROL	QUOT
NEGRES	ADD	HL,DE		ASL	DVDLO
	DEC	HL		ROL	A
POSRES	ADD	HL,HL		PLP	
	DJNZ	LOOP0		BCC	CONT1
	LD	(QUOT),HL		SBC	DIVSR
	RET			JMP	CONT2
DIVSR	DB	$F9	CONT1	ADC	DIVSR
DVDND	DW	$FDE8	CONT2	DEX	
QUOT	DB	$00		BNE	LOOP0
RMNDR	DB	$00		BCS	EXIT
				ADC	DIVSR
				CLC	
			EXIT	ROL	QUOT
				STA	RMNDR
				RTS	
			DIVSR	DB	$F9
			DVDLO	DB	$E8
			DVDHI	DB	$FD
			QUOT	DB	$00
			RMNDR	DB	$00

SCREEN OUTPUT

So far in the course we have used RAM memory and the CPU as a calculating system, and left the results of our efforts somewhere in RAM to be inspected manually using a monitor program. This is obviously unsatisfactory, but until arithmetic and subroutine calls had been studied there was simply no point in considering the screen output from machine code.

Most micros have a memory-mapped display. This means that an area of RAM is dedicated to holding an image of the screen. The screen display is composed of dots, or pixels, which are either on or off. These can, therefore, be represented by