is conventional to talk about the next free byte in the stack as the *top* of the stack, and to imagine the stack growing upwards. In both the Z80 and the 6502, however, the stack pointer is decremented by a push, so that the stack top is actually at the lower memory address than the stack bottom. This is less confusing if we describe the stack as 'rising towards zero'.

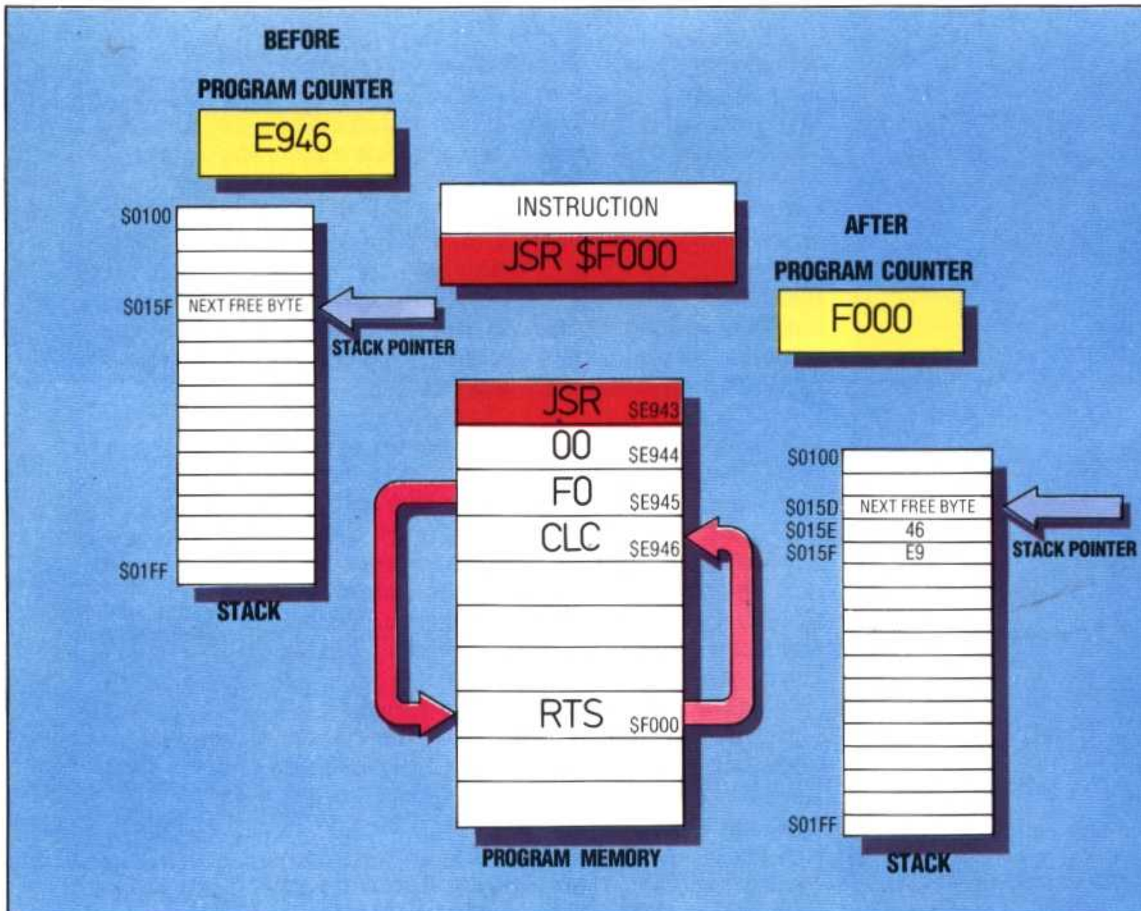
The first program fragment is also typical of programs using the stack in that the number of push instructions is exactly counterbalanced by the number of pops. This is not essential, but failure to observe this harmony of opposites when writing subroutines may result in an incorrect return from the subroutine and consequent program failure. This is one of the commonest bugs in Assembly language programs, but can be fairly easily traced by comparing the number of pop and push instructions in a program.

The Z80 version of the routine differs noticeably from the 6502 in one major respect: the 6502 pushes only single-byte registers onto the stack, while the Z80 always pushes a two-byte register. When you push or pop the Z80 accumulator, you also push or pop the contents of the processor status register, because the CPU treats these two single-byte registers as one two-byte register called the AF (accumulator-flag) register. The power of the Z80 derives greatly from its ability to handle two-byte registers.

It is a good programming habit to start subroutines by pushing the contents of all CPU registers onto the stack, and popping them off the stack immediately before returning from the subroutine. This ensures that the CPU after the subroutine call is in exactly the same state as it was before it, and means that any of the registers can be used in the subroutine with no fear of corrupting data essential to the main program. For example, consider this program subroutine:

	6502		Z80	
	LDA	LOC1	LD	A,LOC1
SUM	ADC	#$6C	ADC	A,$6C
GSUB	JSR	SUBRO	CALL	SUBRO
TEST	BNE	SUM	JR	NZ,SUM
EXIT	RTS		RET	
SUBRO	PHP		PUSH	AF
	PHA		PUSH	HL
	TXA		PUSH	DE
	PHA		PUSH	BC
	TYA		PUSH	IX
SUBR1	PHA		PUSH	IY
SUBR2	STA	LOC2	LD	(LOC2),A
	LDA	#$00	LD	A,$00
SUBR3	PLA		POP	IY
	TAY		POP	IX
	PLA		POP	DE
	TAX		POP	BC
	PLA		POP	HL
SUBR4	PLP		POP	AF
	RTS		RET	

Here, the effect of the instructions between SUBRO and SUBR1 is to push the current register



The Stack

The Stack is essential in handling subroutine calls (such as the 6502's JSR). When the 'jump to subroutine' instruction is first loaded, the program counter contains the address of the next instruction in the program ($E946 here, the address of the CLC instruction); this address is 'pushed' onto the stack causing the stack pointer to be changed to point to the next free byte of stack space ($015D here), and the operand of the jump instruction ($F000 here) is loaded into the program counter as the address of the next instruction, thus causing a branch in program flow

KEVIN JONES