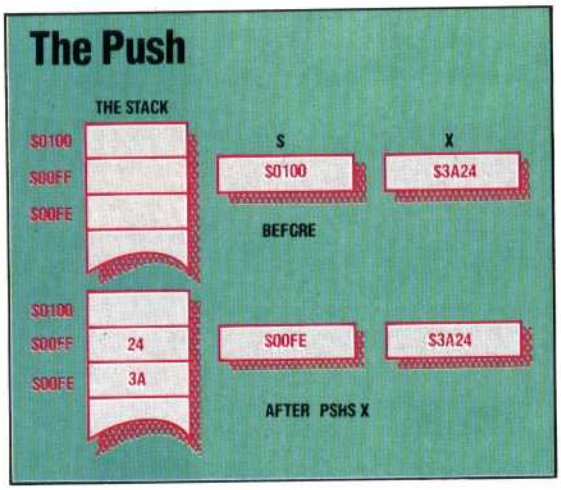
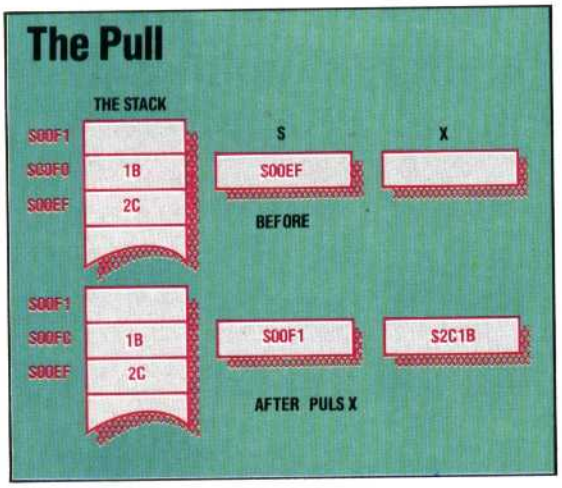
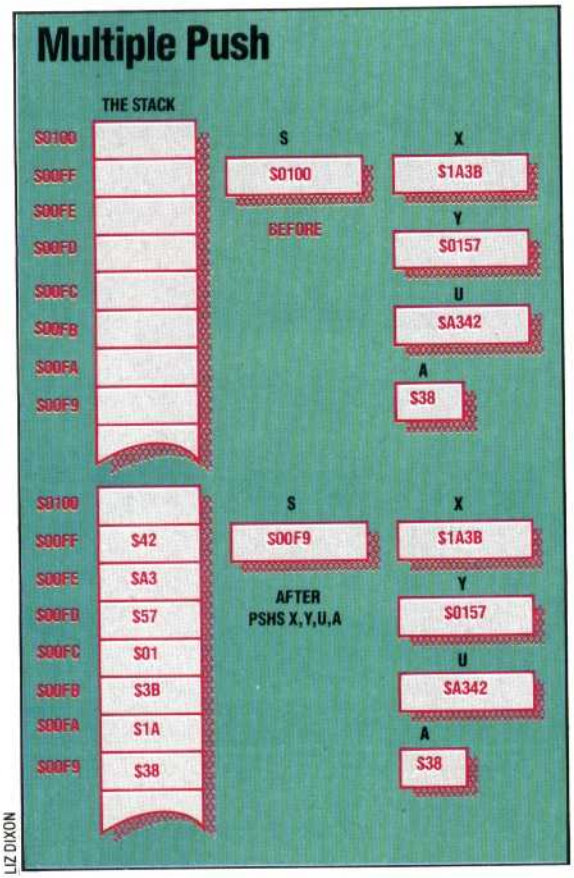
Push Off
The 6809 stack pointer always addresses the 'top' of the stack — that is, the byte most recently written to. When a PSHS X is executed, therefore, S is decremented by two, so that it points to the new stacktop, and the contents of X (a two-byte register) are then written at that address in hi-lo format. Notice that the stack 'rises to zero' — the stack pointer points to lower locations in memory as the stack grows



Pull Together
When PULS X is executed, the contents of the two bytes at the current stack pointer address are copied to X, and S is then incremented by two to point to the new stacktop



Push Off Together
When a multiple-register stack operation is executed, the registers involved are accessed in a pre-determined order — PC,U or S,Y,X,DP,B,A,CC. When PSHS X,Y,U,A is executed, therefore, the contents of U are stacked first, followed by Y, X and A



LIZ DIXON

of plates reached the ceiling, and no more could be added to it.

Stacks in computers work in much the same way. The two operations of adding and removing items are known as *pushing* and *pulling* (or *popping*), respectively. The two extreme situations we have just mentioned are referred to as *underflow* and *overflow.*

Stacks can be implemented in a number of ways (using arrays in a BASIC program, for example), but the method that we are considering requires a block of available memory and a register that we can designate the *stack pointer.* This pointer is necessary to keep track of the current location of the listhead. Unlike a stack of plates, a memory stack cannot be assessed by inspection since there is nothing to distinguish a memory location containing an item of stack data from the next location, which may not be part of the stack. It's worthwhile pointing out that, just as data is not really 'loaded' from memory into a register but only copied, so similarly items are not really 'pulled' off a stack — only the pointer to the top of the stack is changed.

The stack pointer, therefore, contains the address of the current top of the stack. There are two variations possible here: the stack pointer can give either the address of the next free location where data can be stored, or it can give the address of the last item of data stored in the stack. This latter is the convention used by the 6809 processor, although there is no particular advantage in this over the former method — other processors use that technique just as readily.

A significant difference of organisation between a memory stack and a stack of plates in a canteen is that the former grows downwards in the 6809 system: as more items are pushed onto the stack, the stack pointer address gets lower and lower—it is said to 'rise towards zero'.

STACK OPERATIONS

The two 6809 stack operations are represented by the instructions PSH, to push data onto the stack, and PUL, to pull it off. These operations can be applied to either of the two pointers, S and U, so we have PSHS, PULS, PSHU and PULU. The data that is operated on must come from, or go to, a register, although a number of registers can be pushed or pulled in one instruction.

The instruction PSHS X will have the effect of *first* decrementing S, the stack pointer, by two (or one if an eight-bit register is pushed) to give the address of the next free stack location, and *second* storing the contents of X at that address. The first diagram illustrates this procedure. Notice again the 6809 hi-lo addressing convention: the hi-byte ($3A) of X is stored at $00FE , a lower position in memory than the lo-byte ($24), which is stored at $00FF. If you use an assembler, these details of whether stack pointers increment or decrement are irrelevant — the assembler does all the memory management necessary.

The instruction PULS X has the opposite effect: