One might expect the context switch code on a RISC machine to start with a series of statements that each push one register:

```
push    r3
push    r4
push    r5
push    r6
push    r7
push    r8
push    r9
push    r10
push    r11
push    r12
push    lr
```

and to end with a series of statements that each pop one register:

```
pop     lr
pop     r12
pop     r11
pop     r10
pop     r9
pop     r8
pop     r7
pop     r6
pop     r5
pop     r4
pop     r3
```

Interestingly, the code does not contain such sequences. The reason is that although it is a RISC machine, the ARM processor has a single instruction that can save multiple registers. That is, an instruction can push multiple registers onto the stack or can pop multiple registers from a stack. However, instructions that operate on multiple registers take multiple cycles, just as in a CISC machine. As with the Intel processor, registers are popped in the opposite order than they are pushed.

In the code, the opcodes used to save and restore registers are *push* and *pop*. The underlying instruction takes a 16-bit mask, where each bit corresponds to one register. When a programmer codes:

```
push {r0-r11, lr}
```

the assembler generates an instruction with the bit mask set for registers 0 through 11 and the link register, r14. Thus 13 values will be pushed on the stack. The corresponding *pop* instruction must specify exactly the same set of registers. File *ctxsw.S* contains the code.