A more sophisticated *getreg* function would also consider the subsequent uses of x and the commutativity of the operator *op* in determining the register to hold the value of x. We leave such extensions of *getreg* as exercises.

Example 9.5. The assignment `d := (a-b) + (a-c) + (a-c)` might be translated into the following three-address code sequence

```
t := a - b
u := a - c
v := t + u
d := v + u
```

with d live at the end. The code-generation algorithm given above would produce the code sequence shown in Fig. 9.10 for this three-address statement sequence. Shown alongside are the values of the register and address descriptors as code generation progresses. Not shown in the address descriptor is the fact that a, b, and c are always in memory. We also assume that t, u and v, being temporaries, are not in memory unless we explicitly store their values with a MOV instruction.

STATEMENTS	CODE GENERATED	REGISTER DESCRIPTOR	ADDRESS DESCRIPTOR
		registers empty	
t := a - b	MOV a, R0	R0 contains t	t in R0
	SUB b, R0		
u := a - c	MOV a, R1	R0 contains t	t in R0
	SUB c, R1	R1 contains u	u in R1
v := t + u	ADD R1, R0	R0 contains v	u in R1
		R1 contains u	v in R0
d := v + u	ADD R1, R0	R0 contains d	d in R0
	MOV R0, d		d in R0 and memory

Fig. 9.10. Code sequence.

The first call of *getreg* returns R0 as the location in which to compute t. Since a is not in R0, we generate instructions MOV a, R0 and SUB b, R0. We now update the register descriptor to indicate that R0 contains t.

Code generation proceeds in this manner until the last three-address statement d := v + u has been processed. Note that R1 becomes empty because u has no next use. We then generate MOV R0, d to store the live variable d at the end of the block.

The cost of the code generated in Fig. 9.10 is 12. We could reduce this to