the calculation. This form requires an offset field in the instruction large enough to hold an address, of course, so it is less efficient than doing it the other way; however, it is nevertheless frequently the best way.

5.4.7 Based-Indexed Addressing

Some machines have an addressing mode in which the memory address is computed by adding up two registers plus an (optional) offset. Sometimes this mode is called **based-indexed addressing**. One of the registers is the base and the other is the index. Such a mode would have been useful here. Outside the loop we could have put the address of *A* in R5 and the address of *B* in R6. Then we could have replaced the instruction at *LOOP* and its successor with

```
LOOP:   MOV R4,(R2+R5)
        AND R4,(R2+R6)
```

If there were an addressing mode for indirecting through the sum of two registers with no offset, that would be ideal. Alternatively, even an instruction with an 8-bit offset would have been an improvement over the original code since we could set both offsets to 0. If, however, the offsets are always 32 bits, then we have not gained anything by using this mode. In practice, however, machines that have this mode usually have a form with an 8-bit or 16-bit offset.

5.4.8 Stack Addressing

We have already noted that making machine instructions as short as possible is highly desirable. The ultimate limit in reducing address lengths is having no addresses at all. As we saw in Chap. 4, zero-address instructions, such as IADD, are possible in conjunction with a stack. In this section we will look at stack addressing more closely.

Reverse Polish Notation

It is an ancient tradition in mathematics to put the operator between the operands, as in $x + y$, rather than after the operands, as in $x\ y\ +$. The form with the operator "in" between the operands is called **infix** notation. The form with the operator after the operands is called **postfix** or **reverse Polish notation**, after the Polish logician J. Lukasiewicz (1958), who studied the properties of this notation.

Reverse Polish notation has a number of advantages over infix for expressing algebraic formulas. First, any formula can be expressed without parentheses. Second, it is convenient for evaluating formulas on computers with stacks. Third, infix operators have precedence, which is arbitrary and undesirable. For example, we know that $a \times b + c$ means $(a \times b) + c$ and not $a \times (b + c)$ because multiplication has been arbitrarily defined to have precedence over addition. But does left

shift have precedence over Boolean AND? Who knows? Reverse Polish notation eliminates this nuisance.

Several algorithms for converting infix formulas into reverse Polish notation exist. The one given below is an adaptation of an idea due to E. W. Dijkstra. Assume that a formula is composed of the following symbols: variables, the dyadic (two-operand) operators + − * /, and left and right parentheses. To mark the ends of a formula, we will insert the symbol ⊥ after the last symbol and before the first symbol.

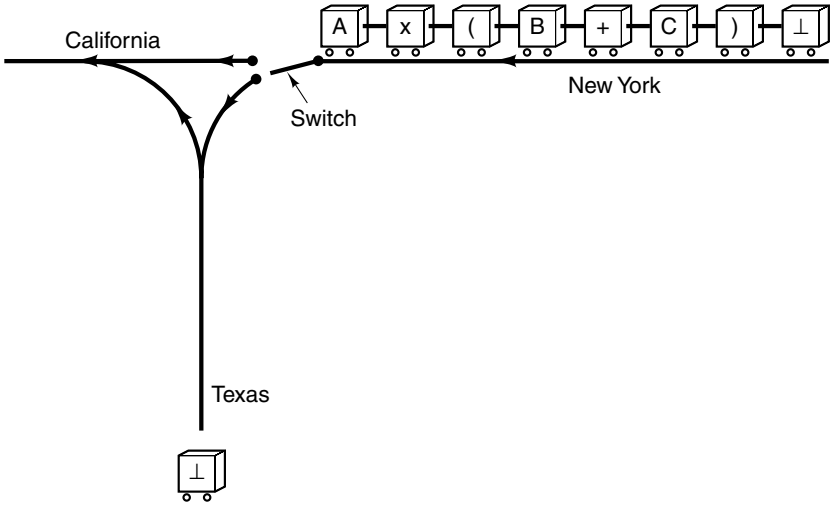

Figure 5-21. Each railroad car represents one symbol in the formula to be converted from infix to reverse Polish notation.

Figure 5-21 shows a railroad track from New York to California, with a spur in the middle that heads off in the direction of Texas. Each symbol in the formula is represented by one railroad car. The train moves westward (to the left). When each car arrives at the switch, it must stop just before it and ask if it should go to California directly or take a side trip to Texas. Cars containing variables always go directly to California and never to Texas. Cars containing all other symbols must inquire about the contents of the nearest car on the Texas line before entering the switch.

The table of Fig. 5-22 shows what happens, depending on the contents of the nearest car on the Texas line and the car poised at the switch. The first ⊥ always goes to Texas. The numbers refer to the following situations:

1. The car at the switch heads toward Texas.
2. The most recent car on the Texas line turns and goes to California.
3. Both the car at the switch and the most recent car on the Texas line are diverted and disappear (i.e., both are deleted).

4. Stop. The symbols now in California represent the reverse Polish notation formula when read from left to right.

5. Stop. An error has occurred. The original formula was not correctly balanced.

Most recently arrived car on the Texas line \ Car at the switch	⊥	+	−	x	/	(	)
⊥	4	1	1	1	1	1	5
+	2	2	2	1	1	1	2
−	2	2	2	1	1	1	2
x	2	2	2	2	2	1	2
/	2	2	2	2	2	1	2
(	5	1	1	1	1	1	3

Figure 5-22. Decision table used by the infix-to-reverse Polish notation algorithm

After each action is taken, a new comparison is made between the car currently at the switch, which may be the same car as in the previous comparison or may be the next car, and the car that is now the last one on the Texas line. The process continues until step 4 is reached. Notice that the Texas line is being used as a stack, with routing a car to Texas being a push operation, and turning a car already on the Texas line around and sending it to California being a pop operation.

Infix	Reverse Polish notation
A + B × C	A B C × +
A × B + C	A B × C +
A × B + C × D	A B × C D × +
(A + B) / (C − D)	A B + C D − /
A × B / C	A B × C /
((A + B) × C + D)/(E + F + G)	A B + C × D + E F + G + /

Figure 5-23. Some examples of infix expressions and their reverse Polish notation equivalents.

The order of the variables is the same in infix and in reverse Polish notation. The order of the operators, however, is not always the same. Operators appear in reverse Polish notation in the order they will actually be executed during the evaluation of the expression. Figure 5-23 gives several examples of infix formulas and their reverse Polish notation equivalents.