This instruction contains a *forward reference*—that is, a reference to a label (RETADR) that is defined later in the program. If we attempt to translate the program line by line, we will be unable to process this statement because we do not know the address that will be assigned to RETADR. Because of this, most assemblers make two passes over the source program. The first pass does little more than scan the source program for label definitions and assign addresses (such as those in the Loc column in Fig. 2.2). The second pass performs most of the actual translation previously described.

In addition to translating the instructions of the source program, the assembler must process statements called *assembler directives* (or *pseudo-instructions*). These statements are not translated into machine instructions (although they may have an effect on the object program). Instead, they provide instructions to the assembler itself. Examples of assembler directives are statements like BYTE and WORD, which direct the assembler to generate constants as part of the object program, and RESB and RESW, which instruct the assembler to reserve memory locations without generating data values. The other assembler directives in our sample program are START, which specifies the starting memory address for the object program, and END, which marks the end of the program.

Finally, the assembler must write the generated object code onto some output device. This *object program* will later be loaded into memory for execution. The simple object program format we use contains three types of records: Header, Text, and End. The Header record contains the program name, starting address, and length. Text records contain the translated (i.e., machine code) instructions and data of the program, together with an indication of the addresses where these are to be loaded. The End record marks the end of the object program and specifies the address in the program where execution is to begin. (This is taken from the operand of the program's END statement. If no operand is specified, the address of the first executable instruction is used.)

The formats we use for these records are as follows. The details of the formats (column numbers, etc.) are arbitrary; however, the information contained in these records must be present (in some form) in the object program.

Header record:

Col. 1	H
Col. 2–7	Program name
Col. 8–13	Starting address of object program (hexadecimal)
Col. 14–19	Length of object program in bytes (hexadecimal)

Text record:

Col. 1	T
Col. 2–7	Starting address for object code in this record (hexadecimal)
Col. 8–9	Length of object code in this record in bytes (hexadecimal)
Col. 10–69	Object code, represented in hexadecimal (2 columns per byte of object code)

End record:

Col. 1	E
Col. 2–7	Address of first executable instruction in object program (hexadecimal)

To avoid confusion, we have used the term *column* rather than *byte* to refer to positions within object program records. This is not meant to imply the use of any particular medium for the object program.

Figure 2.3 shows the object program corresponding to Fig. 2.2, using this format. In this figure, and in the other object programs we display, the symbol ^ is used to separate fields visually. Of course, such symbols are not present in the actual object program. Note that there is no object code corresponding to addresses 1033–2038. This storage is simply reserved by the loader for use by the program during execution. (Chapter 3 contains a detailed discussion of the operation of the loader.)

We can now give a general description of the functions of the two passes of our simple assembler.

Pass 1 (define symbols):

1. Assign addresses to all statements in the program.
2. Save the values (addresses) assigned to all labels for use in Pass 2.

```
H^COPY  ^001000^00107A
T^001000^1E^141033^482039^001036^281030^301015^482061^3C1003^00102A^0C1039^00102D
T^00101E^15^0C1036^482061^081033^4C0000^454F46^000003^000000
T^002039^1E^041030^001030^E0205D^30203F^D8205D^281030^302057^549039^2C205E^38203F
T^002057^1C^101036^4C0000^F1^001000^041030^E02079^302064^509039^DC2079^2C1036
T^002073^07^382064^4C0000^05
E^001000
```

Figure 2.3 Object program corresponding to Fig. 2.2.