I asked the program to display 40 bytes starting at address 8048000, which precedes the address of the `main` function. Note the 7F byte followed by bytes representing the letters E, L, and F. These four bytes identify the format (ELF) in which the executable file was stored. ELF (Executable and Linking Format) is widely used by UNIX systems, including Linux. 8048000 is the default address at which ELF executables are loaded on x86 platforms.

Let's run the program again, this time displaying a block of bytes that starts at the address of the addr variable:

```
Address of main function: 804847c
Address of addr variable: bfec5484

Enter a (hex) address: bfec5484
Enter number of bytes to view: 64

 Address                 Bytes                    Characters
 -------  -------------------------------------  ----------
BFEC5484  84 54 EC BF B0 54 EC BF F4 6F  .T...T...o
BFEC548E  68 00 34 55 EC BF C0 54 EC BF  h.4U...T..
BFEC5498  08 55 EC BF E3 3D 57 00 00 00  .U...=W...
BFEC54A2  00 00 A0 BC 55 00 08 55 EC BF  ....U..U..
BFEC54AC  E3 3D 57 00 01 00 00 00 34 55  .=W.....4U
BFEC54B6  EC BF 3C 55 EC BF 56 11 55 00  ..<U..V.U.
BFEC54C0  F4 6F 68 00                    .oh.
```

None of the data stored in this region of memory is in character form, so it's a bit hard to follow. However, we do know one thing: the `addr` variable occupies the first four bytes of this region. When reversed, these bytes form the number BFEC5484, the address entered by the user. Why the reversal? Because x86 processors store data in little-endian order, as we saw earlier in this section.

The `volatile` Type Qualifier

On some computers, certain memory locations are "volatile"; the value stored at such a location can change as a program is running, even though the program itself isn't storing new values there. For example, some memory locations might hold data coming directly from input devices.

The `volatile` type qualifier allows us to inform the compiler if any of the data used in a program is volatile. `volatile` typically appears in the declaration of a pointer variable that will point to a volatile memory location:

```
volatile BYTE *p;   /* p will point to a volatile byte */
```

To see why `volatile` is needed, suppose that `p` points to a memory location that contains the most recent character typed at the user's keyboard. This location is volatile: its value changes each time the user enters a character. We might use the following loop to obtain characters from the keyboard and store them in a buffer array: