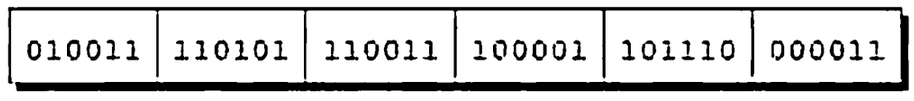
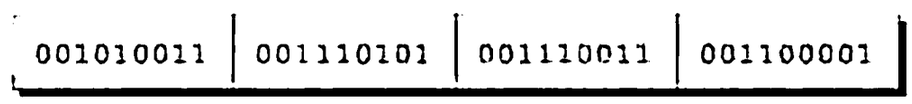
When memory is divided into words, each word has an address. An integer usually occupies one word, so a pointer to an integer can just be an address. However, a word can store more than one character. For example, a 36-bit word might store six 6-bit characters:



or four 9-bit characters:



For this reason, a pointer to a character may need to be stored in a different form than other pointers. A pointer to a character might consist of an address (the word in which the character is stored) plus a small integer (the position of the character within the word).

On some computers, pointers may be "offsets" rather than complete addresses. For example, CPUs in the Intel x86 family (used in many personal computers) can execute programs in several modes. The oldest of these, which dates back to the 8086 processor of 1978, is called ***real mode***. In this mode, addresses are sometimes represented by a single 16-bit number (an ***offset***) and sometimes by two 16-bit numbers (a ***segment:offset pair***). An offset isn't a true memory address; the CPU must combine it with a segment value stored in a special register. To support real mode, older C compilers often provide two kinds of pointers: ***near pointers*** (16-bit offsets) and ***far pointers*** (32-bit segment:offset pairs). These compilers usually reserve the words `near` and `far` as nonstandard keywords that can be used to declare pointer variables.

Q: If a pointer can point to *data* in a program, is it possible to have a pointer to program *code?

A: Yes. We'll cover pointers to functions in Section 17.7.

Q: It seems to me that there's an inconsistency between the declaration

```
int *p = &i;
```

and the statement

```
p = &i;
```

Why isn't `p` preceded by a `*` symbol in the statement, as it is in the declaration? [p. 244]

A: The source of the confusion is the fact that the `*` symbol can have different meanings in C, depending on the context in which it's used. In the declaration

```
int *p = &i;
```

the `*` symbol is *not* the indirection operator. Instead, it helps specify the type of `p`, informing the compiler that `p` is a *pointer* to an `int`. When it appears in a statement,