This system of paged memory is convenient because we can look at the address of any byte, and split it into two parts — the digits from the hundred column leftwards are the page number of the byte, and the digits from the tens column to the right are the number of bytes counted from the bottom of that page. In the example above we have actually split the address 3518 into two numbers: page number 35, and byte number 18 on that page. We call 18 an *offset*, or *page offset*, because it is the number by which you must offset (or increase) the address of the bottom byte on the page in order to reach the byte in question.

The computer, however, doesn't count in decimal, as we do — it counts in binary. The paging system depends upon being able to find the page and the offset by simply inspecting the address of the byte. The decimal address 99 is represented by 01100011 in binary, and 100 decimal is 01100100 binary; decimal 199 is 11000111, and decimal 200 is 11001000 binary. We can see, from these examples, that there's no simple way of looking at the binary numbers and telling page from page, as we can so easily do with the decimal equivalents. The reason for this is the choice of page size.

We chose 100 as the page size precisely because it's a meaningful number in the decimal system (it's a power of 10). If we are to count in binary, however, then we must choose a page size to suit *that* system. The page size used by our computers is 256, so that page0 starts with byte0 and continues to byte255; page1 starts with byte256 and continues to byte511, and so on. To see why this is convenient we must write these addresses in binary:

```
Page0: byte00000000  — byte11111111
Page1: byte100000000 — byte111111111
```

As you can see, we can count in binary from 0 to 255 in an eight-bit number; the next number — 256 — requires nine bits, and with nine bits we can count up to 511. The next number — 512 — requires ten bits, and with ten bits we can count up to 1023; and so on. We now see that if the page size is 256 and we count in binary, then the offset is the rightmost eight bits, and the page number is given by the bits from bit8 leftwards.

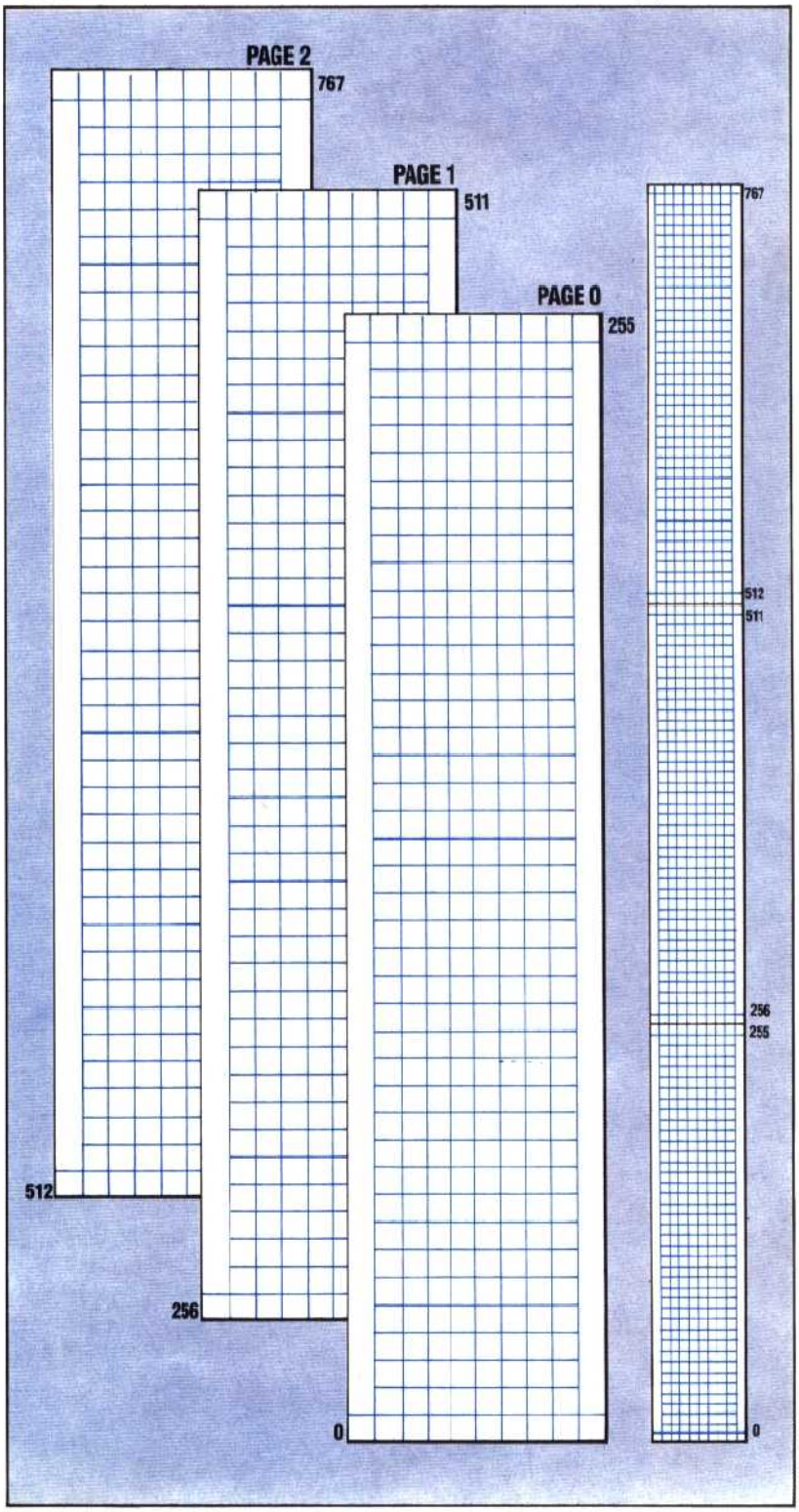
This may be puzzling since we have already stated that the CPU can handle only single bytes, and a byte contains only eight bits. Therefore, you may ask, what good is it to talk about nine- and 10-bit numbers? The answer is that all addresses in memory are treated as two-byte numbers, and the CPU deals with them one byte at a time. If we rewrite the page boundaries as two-byte numbers this system becomes more clear:

```
Page  0 starts at00000000 00000000
        ends  at00000000 11111111
Page  1 starts at00000001 00000000
        ends  at00000001 11111111
Page 10 starts at00000010 00000000
        ends  at00000010 11111111
Page 11 starts at00000011 00000000
        ends  at00000011 11111111
```

and so on.

Now we can see that when the CPU fetches information from, or puts information into, a byte in memory, that byte will be identified by a two-byte address. The first, or leftmost, byte of the two gives the page number, while the second, or rightmost, byte gives the offset.

On page 38, we provide programs that convert from decimal into binary, as well as hexadecimal numbers. The latter are used extensively in machine code, and will be fully discussed later in the course.



Paged Addressing
Paged addressing divides memory into imaginary blocks or pages of 256 bytes. All addresses are then expressed as two-byte numbers: one byte gives the page number the other gives the offset from the start of that page