# JUST HOW LARGE IS A 64-BIT ADDRESS?\*

# Rick Simpson

This work is produced by OpenStax-CNX and licensed under the Creative Commons Attribution License  $3.0^{\dagger}$ 

#### Abstract

For decades we've used 16-bit and 32-bit computers. Now 64-bit computers are becoming affordable. This module makes clear the enormous increase in memory addressing capability of 64-bit addresses by comparing the physical sizes of familiar objects.

## 1 A bit of history

The first personal computers were **16-bit machines**. They dealt naturally with 16-bit-wide binary quantities: they had a **word size** of 16 bits. This did not affect the size of the numbers that they could compute with, but it affected **memory addressing**. Generally, memory addresses were limited to 16-bit numbers, meaning that they ranged between 0 and  $2^{16} - 1$ , or 65, 535. That's only 64Ki bytes<sup>1</sup>, where "Ki" stands for "kilobinary", or 1024. While that was enough memory for early BASIC programs, it's not nearly enough for modern programs that use today's elaborate full-color windowed user interfaces.

The 16-bit addresses of the early machines were too small even then. Hardware schemes involving segmentation registers were introduced so that more than 64K bytes of memory could be attached to the computer. The operating system would manipulate the segmentation registers so that different parts of the memory could be used by different programs, or by the same program at different times. Fundamentally, though, the programs had to deal with 16-bit memory addresses.

It took a computer hardware revolution to fix this. The new personal computers were **32-bit machines**: they used 32-bit integers and 32-bit memory addresses. 32-bit memory addresses range from 0 to  $2^{32} - 1$ , or 4, 294, 967, 295. This is 4Gi bytes, "four gigabinary bytes", a bit more than four billion bytes. Personal computers didn't actually have this much storage, of course (at least not back then), but the size of the address stopped being a programming problem. Any ordinary register in the computer could hold a memory address that points anywhere in memory. The operating system had no need to manipulate segmentation registers to make it look (at least momentarily) like a 16-bit register could address whatever data was being used.

It's difficult to express how big an impact this was on program developers. Suddenly, size of programs and data was no longer a problem. Things had to fit in the memory the computer had, but this was much more than 64Ki bytes. Programming became much simpler; programs could be larger, could incorporate more functions, and could do much bigger tasks. The benefits of the larger address were significant enough that it was worth going through **porting** difficulties in order to move operating systems and applications from 16-bit to 32-bit machines.

Today 16-bit machines are history as far as personal computers and larger computers are concerned. 16bit processors are still made (they are generally very inexpensive) but they are used for specialized purposes

<sup>\*</sup>Version 1.2: Oct 28, 2010 10:16 am -0500

 $<sup>^{\</sup>dagger}$  http://creativecommons.org/licenses/by/3.0/

 $<sup>^{1}</sup>$ "Prefixes for binary multiples" < http://cnx.org/content/m13081/latest/>

such as controlling appliances and automobiles rather than in general-purpose computers. Almost all of today's PCs and large server computers are 32-bit machines, with the exceptions being **64-bit machines**.

Sixty-four-bit machines are becoming increasingly common. Once limited to the machine rooms of large companies because of their expense and physical size, 64-bit computers can now be purchased as home PCs. They use 64-bit integers and 64-bit memory addresses. Sixty-four-bit memory addresses range from 0 to  $2^{64} - 1$ , or 18, 446, 744, 073, 709, 551, 615, or about  $1.845 \times 10^{19}$ , 18.45 exabytes. Such a large number is well beyond our everyday experience. We may think we understand how to compare 16-bit and 32-bit memory sizes, 65 thousand to 4 billion, but  $1.845 \times 10^{19}$ . is beyond our comprehension as just a number. The goal of this module is to make the size difference real to us by comparing the sizes of familiar physical objects.

## 2 Comparing surface areas

Let's assume that each individual **address** in our computer represents a small **area** such as a tiny portion of a piece of paper. The computer's entire memory is represented by the piece of paper, and each memory address "addresses" a small part of it.

We'll start with a 16-bit computer, and assume that 16-bit addresses can span the surface of a 4-inch square Post-it (R) (Section 6: Notes) note (Figure 1 (Sixteen-bit addressing)).



Figure 1: A 16-bit-wide address can span the area of a 4-inch Post-it note (16 sq in; 103 sq cm)

Each individual memory address in our 16-bit computer then represents

$$\frac{area \ of \ note}{memory \ size} = \frac{16 \ sq \ in}{2^{16}} = \frac{1}{4096} sq \ in = .000244 sq \ in = .157 sq \ mm \tag{1}$$

Imagine the note covered with an array of tiny squares, each 1/64 inch (.4 mm) on a side. This is about the size of a dot made by a pencil with a .4 mm lead. Each memory address would then designate one of the small squares (Figure 2 (Mapping addresses to areas)).



#### Mapping addresses to areas



Let's move on to 32-bit addressing and see how much bigger computer memory can be when we **double** the width of the address. Each individual address will still represent the same amount of surface area, a square 1/64 inch (.4 mm) on a side. Now we'll have  $2^{32}$  addresses, or 4,294,967,296 rather than 65,536.  $2^{32}$  is  $(2^{16})^2$ , so our 32-bit address covers 65,536 times as much area as our 16-bit address. This is far too much area for a piece of paper, so we'll have to move to something a bit larger: a tennis court, including the surrounding out-of-bounds area (Figure 3 (Thirty-two-bit addressing)). If you had 65,536 Post-it notes, each 4 inches square, you could just about cover a tennis court with them.





3

A 32-bit address can span so much more area than a 16-bit address that it goes beyond just being bigger or even **a lot** bigger, it's different in some fundamental way. The vast increase in the amount of memory that can be addressed by a program means that not only can existing programs be adapted to deal with larger problems, but new applications that weren't possible at all on the smaller machine can now be created.

The final step, at least for now, is to a 64-bit address. Sixty-four-bit machines are still not common today, but they have become affordable even to consumers (in late 2010, Hewlett-Packard<sup>2</sup> 's Internet list price for a workstation containing an AMD 64-bit processor was US \$759). Again, the wider address spans so much more memory than in the previous generation that things are fundamentally different. To see this, we'll still let each individual address be represented by a tiny square 1/64 inch (.4 mm) on a side. 16-bit addressing required only a Post-it note to represent all the memory, while 32-bit addressing required a tennis court. 64-bit addressing spans  $2^{64}$  bytes, which is  $(2^{32})^2$ . Our 64-bit address spans more than four billion times as much memory as a 32-bit address. Even using our tiny square area to represent each memory address requires a considerable portion of the earth's surface to represent  $2^{64}$  bytes of memory: Western Europe (Figure 4 (Sixty-four-bit addressing)).







## 3 Length, distance

Let's compare lengths rather than areas. We'll assume that 16-bit addresses span only half a millimetre, the diameter of a common pencil lead. Then each individual address represents a length of

$$\frac{.5mm}{2^{16}} = .0000076mm \tag{2}$$

<sup>2</sup>http://www.hp.com

This is a tiny length, just .0076 microns. When we consider that the width of the tiny lines etched onto a modern computer chip are roughly 1 micron, then we can see how small it really is.

Now that we've established what length one individual address corresponds to, and we see what  $2^{16}$  of them correspond to (.5 mm), we can scale our length up for 32-bit and 64-bit addressing (Table 1: Comparing distances).

### **Comparing distances**

An address this wide	Spans this length	Which is approximately
16 bits	$.5 \mathrm{mm}$	Width of a pencil dot
32 bits	114.27  feet  (34.83  m)	Height of a 10-story building
64 bits	1 A.U.	Distance from the Earth to the Sun

#### Table 1

So if we start with a tiny block so short that 130 of them stacked up would be only 1 micron high, by the time we stack up  $2^{64}$  of them our stack would reach from the Earth to the Sun, about 93,000,000 miles (about 150,000,000 km).

## 4 Time

If we have a disk that will hold  $2^{64}$  bytes of data and we use it to hold one long video recording, how long could that recording be?

We'll use round numbers to make things easy. Let's assume that each **frame** recorded by our video camera consists of 1 megabyte. That would be a moderate size image in full color, with no compression. Like many video cameras, ours records 30 frames each second. Thus, each second of video will require 30M bytes of space.

$$\frac{\text{total disk bytes}}{\text{bytes per sec}} = \frac{2^{64}}{30 \times 10^6} = 6.149 \times 10^{11} \text{sec} = 19,484 \text{years}$$
(3)

With an 18.45 exabyte disk and a large enough battery, we could have set up a video camera 19,500 years ago, during the last ice age, and let it record continually day and night (Figure 5 (A video camera recording for 19,500 years)). It would just be filling up the disk now.



Figure 5: A video camera recording for the past 19,500 years would just be filling its 18.45 exabyte disk now

## **5** Conclusion

By equating various widths of computer memory addresses to familiar objects and then comparing the sizes of those objects, we gain some insight into just how large a 64-bit memory address is. The cold, hard number 18,446,744,073,709,551,616 has no real meaning for us, but seeing that that many almost infinitesimal boxes, each less than 1/100 micron high, would make a stack equal in height to the distance between the Earth and the Sun convinces us that  $2^{64}$  is **really big**. We can see how computer programs can treat memory as a completely different kind of resource on a 64-bit computer, compared to memory on a 32-bit or 16-bit computer.

So now we know what "big" means, at least until we see our first 128-bit computer....

## 6 Notes

- AMD Opteron is a trademark of Advanced Micro Devices<sup>3</sup>.
- Intel is a registered trademark of Intel Corporation<sup>4</sup>.
- **Post-it** is a registered trademark of 3M Company<sup>5</sup>.

## Glossary

## **Definition 1: 16-bit machine** A computer with 16-bit-wide internal registers, and an address width of 16 bits.

## Definition 2: 32-bit machine

A computer with 32-bit-wide internal registers, and an address width of 32 bits.

## Definition 3: 64-bit machine

A computer with 64-bit-wide internal registers, and an address width of 64 bits.

<sup>&</sup>lt;sup>3</sup>http://www.amd.com

<sup>&</sup>lt;sup>4</sup> http://www.intel.com

<sup>&</sup>lt;sup>5</sup>http://www.3m.com

## **Definition 4: Word size**

Originally, this was the "natural" data width of the machine: the amount of data that could be loaded from memory or stored to memory in one operation. It was generally the width of the machine's registers. As computer to memory interfaces have gotten more elaborate, the word size has less meaning. 32-bit computers based on Intel (Section 6: Notes) processors still have a "word size" of 16 bits, even though they load and store 32-bit quantities. This is because the definition of "word" has not been changed for Intel's processors since the days of 16-bit machines. Much technical documentation refers to 16-bit words and 32-bit doublewords. Registers on an Intel 32-bit processor can each hold a doubleword.

### **Definition 5: Memory addressing**

Computers refer to memory locations by **address**, where an address is just a number. Memory addresses typically start at 0 and run to a number that's the total size of memory minus 1 (minus 1 because the first memory location is 0 rather than 1). The amount of memory that a computer can have is determined by the **width** of the memory address, because the width determines how large a number can be used as a memory address.

## **Definition 6: Porting**

Moving an existing program from one machine to another. If the machines are identical or almost so, this may mean just copying the program from one place to another. If the machines have different instruction sets, it may mean recompiling the program to generate an executable file for the new machine. If some other aspect of the machine is different, in particular the width of the machine's memory address, it may involve making modifications to the program before recompiling. The effort required to port a program may range from trivial to extremely difficult.