Measurement and Units of Measure

Introduction

Computer Science deals with three overlapping areas of interest: hardware, software, and theory. Computer Architecture is a subject within computer science that deals with hardware, which means (among other things) that you need to be comfortable with measuring the physical properties of various hardware devices and systems. The purpose of this document is to review some key topics that your probably learned long ago, but may have forgotten.

Units of Measure and Orders of Magnitude

We always measure the physical properties of a thing using particular units. For example, the unit for measuring your body's mass is the *pound* and the unit for measuring your height is *inches.* But, of course, it's not as simple as that. If you use the metric system you measure your mass in kilograms, not pounds, and you measure your height in centimeters. And even if you measure your height in inches, you are probably mix two different units: *feet* and *inches*.

Although we can measure the mass and lengths of the devices and we deal with in computer architecture, we are typically more interested in a third property: *time* (measurement unit: the *second)* and the related properties of *period* and its reciprocal, *frequency*. Furthermore, computers manipulate physical representations of *information*, which has its own measurement unit, the *bit*.

The first issue to deal with is the enormous range of values that occur in the physical and information worlds. The term *order of magnitude* refers to how big or small something is, specifically its size rounded to the nearest power of ten. For example, 123 would round to 100, 575 rounds to 1000; you would say that 575 is one order of magnitude larger than 123 because its order of magnitude has one more decimal digit.

In the physical world of length, mass, and time, prefixes are used to indicate multiplication of the basic units (meter, gram, second) by various orders of magnitude. There are prefixes for every three orders of magnitude, as well as several intermediate ones. The multiplication prefixes commonly encountered in computer architecture are:

Multiplier	Order of Magnitude	Prefix (abbreviation)	Common Name
0.000 000 000 001	10^{-12}	pico (p)	Trillionth
0.000 000 001	10^{-9}	nano (n)	Billionth
0.000 001	10^{-6}	micro (μ)	Millionth
0.001	10^{-3}	milli (m)	Thousandth

Multiplier	Order of Magnitude	Prefix (abbreviation)	Common Name
1,000	10^{+3}	kilo (K)	Thousand
1,000,000	10^{+6}	mega (M)	Million
1,000,000,000	10^{+9}	giga (G)	Billion
1,000,000,000,000	10^{+12}	tera (T)	Trillion
1,000,000,000,000,000	10+15	peta (P)	Quadrillion

In computer architecture, small numbers are most often associated with *time* (seconds): how many milliseconds (msec) it takes a spinning disk to make one rotation, how many picoseconds (psec) it takes for a gate to change state, etc. Big numbers are most often associated with information storage capacities (how many gigabytes (GB) of main memory or terabytes (TB) of disk a computer has) or the reciprocal of time (how fast is a clock, in GHz). The remainder of this document tries to put all this terminology in order.

Time (period, frequency)

The basic unit of time is the *second*, and we use prefixes to denote fractions of a second. So one second contains 1,000 milliseconds (msec), 1,000,000 microseconds (*µ*sec), etc. Remember, the prefix names represent three orders of magnitude, so there are a thousand picoseconds in a nanosecond, a thousand nanoseconds in a microsecond, a thousand microseconds in a millisecond, and a thousand milliseconds in a second.

Skill: represent the same value using different prefixes.

The same value can be represented in a number of ways. For example, 1 nsec is the same as 1,000 psec as well as 0.001 *µ*sec. Although it is not a hard and fast rule, the normal way to represent a value is to scale it so that the integer part is a number between one and 999. The conversion factor for adjacent prefixes is either 1,000 or 0.001, depending on which way you are going. The trick is not to go the wrong way! If you know the conversion factor between feet and inches is 12, that only applies to converting feet to inches: 5'×12 give 60". To go the other way you have to use 1÷12 as the conversion factor, which is the same as dividing by 12 rather than multiplying: 60"÷12 gives 5'. Feet and inches are everyday concepts for us, so we are not likely to make silly mistakes, like multiplying inches by 12 to get feet (720' in 60"—no way!). But not so with scientific prefixes: you have to think carefully when converting something like nanoseconds to microseconds to make sure you don't end up with picoseconds by mistake. Just remember to multiply when going to smaller units and to divide when going to larger units.

- Convert 1.250 nsec to psec: Multiply, giving 1,250 psec.
- Convert 1.250 nsec to μsec: Divide, giving 0.001250 μsec.
- How many picoseconds in 3.7 μsec? Multiply by 10^{+6}, giving 3,700,000 psec.

Use seconds to measure *latency*—how long it takes something to happen. Examples are how long it takes a gate to change state; how long it takes to execute and instruction (or a program); how long it takes to get a copy of some information that is in main memory or on disk, etc.

Time is also intimately related to various *rate* measures—see below.

Information

Computers are all about manipulating digital information, represented as *binary digits*; John Tukey of Bell Laboratories coined the name *bit* to mean one binary digit in 1946. But the digital age owes much of its binary underpinnings to a colleague of Tukey's at Bell Labs, Claude Shannon, who proposed the bit as the unit of measure for information in 1948. Because Bell Labs was part of the telephone system, Shannon addressed the issue of how to measure the information capacity of a communication channel, such as a telephone line. Simply stated, Shannon defined the bit as the amount of uncertainty that is reduced by answering one yes/no question. For example, if I do not know whether it is raining outside and I look out the window, my uncertainty is reduced by one bit—whether it is actually raining or not. Of course, I will undoubtedly obtain other information by looking out the window, but my uncertainty about whether or not it is raining has been reduced by one bit.

Key to understanding information measurement is the notion of the number of possibilities our uncertainty covers. If I ask you to pick a number between 1 and 100, my uncertainty spans 100 possibilities. The number of bits of uncertainty, however, is just $log_2(100)$, because I can determine which number you picked by getting the answer to that many yes/no questions. My strategy is to use each question to eliminate half of the possible outcomes. Rather than ask, "Is it 1? … Is it 2? … " until you say "yes," I might start by asking, "Is it an odd number?" Regardless of your answer, I will have eliminated half of the possible outcomes and reduced my uncertainty by one bit. Since $log_2(100)$ is approximately 6.644, my "binary search" strategy should reduce my uncertainty after just 6-7 questions rather than the 50 it would take on average if I just asked about each number individually.

The number 6.644 in the previous example raises some important points. Obviously, you can't ask a fraction of a yes/no question; either you ask the question or you don't, and if you do ask it, the answer has to be either "yes" or "no." First, let's look at a way in which the fractional number of bits actually does make sense. Then we'll look at how information is stored in a computer, where fractional bits don't exist any more than fractional yes/no questions.

To understand fractional bits of information, think in terms of the *average* number of yes/no questions you would have to ask assuming that you get to play the guessing game lots of times. Consider these two sequences of questions and answers:

Sequence 1		Sequence 2	
Question	**Answer**	**Question**	**Answer**
Bigger than 50?	No	Bigger than 50?	No
Bigger than 25?	No	Bigger than 25?	No
Bigger than 12?	No	Bigger than 12?	No
Bigger than 6?	No	Bigger than 6?	No
Bigger than 3?	No	Bigger than 3?	No
Bigger than 2?	No	Bigger than 2?	Yes

After the six questions in Sequence 1, we still don't know what the number was that we are searching for: it could be either 1 or 2, and we would have to ask a seventh question to find out which it is. But after the same six questions and after receiving all the same answers the first five time, we know that the answer is 3, which is the only number bigger than 2 that is not bigger than 3. The point is that, depending on the number you are trying to guess and on how you choose to divide the uncertainty in half, sometimes it will take six questions and sometimes it will take seven. You could try playing the game lots of times with a patient friend (or a computer programmed to act like a patient friend), and count how often it takes six questions and how often it takes seven. OK, I just played the game 1,000 times and found that I got the secret number after six guesses 356 times, but the rest of the time it took seven guesses. What is the average number of guesses? It is a *weighted average.* The two numbers being averaged are six and seven, but instead of just adding them together and dividing by two, you multiply the value six by a weight of 356 and multiply seven by the weight (1,000 - 356), then you add the two products and divide by the number of cases, 1,000. That's $((356\times6)+(644\times7))\div1{,}000 = 6.644 = \mathit{log_2(100)}$. Of course if this the game is played truly randomly, it won't always come out exactly 6.644, but if you play it enough times (a) you will get very bored and/or lose all your friends and (b) will come closer and closer to the actual fractional number of bits.

> Weighted averages show up a lot in this course. A shortcut (sometimes) is to adjust all the weights so they add up to 1.0, eliminating the need for division at the end. In the example, divide each weight by the sum of the weights (giving 0.356 and 0.644), then just multiply and add.

[integer numbers of bits]

Rates (speed, bandwidth)

Weighted Averages (CPI)

Performance (Execution Time)

Comparisons (ratios, percentages)