CS 201 Section 1 Homework 1 Suggested Solutions

1.a.

kilo (k)	10^3
giga (G)	10^9
tera (T)	10^{12}
pico (p)	10^{-12}
nano (n)	10^{-9}
mega (M)	10^6
micro (μ)	10^{-6}
peta (P)	10^{15}

Note: most of these are defined on Tanenbaum pg. 19; for the rest, as simple web search for "SI Units" would turn up a page like

`http://physics.nist.gov/cuu/Units/prefixes.html`

b. We know

$$\text{Clock frequency} = \frac{1}{\text{Cycle time}}$$

so if

$$\text{Clock Frequency} = 200 \text{ MHz} = 200 \times 10^6 \text{ Hz} = 2 \times 10^8 \text{ Hz}$$

then

$$\text{Cycle time} = (2 \times 10^8)^{-1} \text{ sec} = 0.5 \times 10^{-8} \text{ sec} = 5 \times 10^{-9} \text{ sec} = 5 \text{ nsec}$$

c.

$$\frac{32 \times 2^{20}}{32 \times 10^6} = \frac{2^{20}}{10^6} = 1.048576$$

So the error is about 4.9%.

The International Electrotechnical Commission would prefer us to use the terms "kibi" (abbreviation "Ki") and "mibi" (abbreviation "Mi") for 2^{10} and 2^{20} respectively. See `http://physics.nist.gov/cuu/Units/binary.html`

2. When we **translate** a program P_1 from one language L_1 to another language L_2, we replace each L_1 instruction in P_1 with an equivalent sequence of L_2 instructions, thus generating a new program P_2 entirely in L_2. When we use language L_2 to **interpret** language L_1, we use a fixed program I (the **interpreter**), written in L_2, which takes a description of an arbitrary L_1 program P_1 as data, examines each instruction in turn, and executes the effect of the instruction directly using a fixed sequence of L_2 code within I.

3. Moore's law predicts that the number of transistors on a chip increases by about 60% from year n to year $n+1$, i.e.,

$$\frac{\text{Count}_{n+1}}{\text{Count}_n} = 1.6$$

Assuming the size of the chip remains the same, this implies that the *area* of each transistor shrinks by the sme amount, i.e.,

$$\frac{\text{TransistorArea}_{n+1}}{\text{TransistorArea}_n} = \frac{\text{ChipArea}/\text{Count}_{n+1}}{\text{ChipArea}/\text{Count}_n} = \frac{1}{1.6} = 0.625$$

Assuming transistors are round, this implies that

$$\frac{\text{TransistorDiameter}_{n+1}}{\text{TransistorDiameter}_n} = \frac{2\sqrt{\frac{\text{TransistorArea}_{n+1}}{\pi}}}{2\sqrt{\frac{\text{TransistorArea}_n}{\pi}}} = \sqrt{\frac{\text{TransistorArea}_{n+1}}{\text{TransistorArea}_n}} = \sqrt{0.625} \approx 0.791$$

So the diameter of each transistor will shrink from 1 micron ($\equiv 10^{-6}$ meters) to about 0.8 microns.

4. In the absence of pipelining, the execution cycle for the datapath will take at least $5 + 10 + 5 = 20$ *nsec*. (In fact, it will take longer, since the machine must also fetch each instruction before executing it – but since we're asked to calculate *maximum* MIPS, we can ignore this.) If one instruction requires at least 20 $nsec = 2 \times 10^{-8} sec$, the machine can execute at most

$$\frac{1}{2 \times 10^{-8}} = 0.5 \times 10^8 = 50 \times 10^6$$

instructions per second, i.e., it is capable of at most 50 MIPS.

5. You can't really say which computer is faster, because it depends on how much work each instruction does (on average). Suppose a given program requires 10 instructions on computer 1, but 40 instructions on computer 2. Then the program will run twice as fast on computer 1, even though each individual instruction takes twice as long.

6.

a.

We want

$$\frac{\text{New time}}{\text{Old time}} = 0.5$$

where

$$\text{New time} = \frac{\text{Old multiply time}}{\text{Improvement}} + \text{Old non-multiply time}$$

I.e.,

$$\text{New time} = \frac{0.75 \times \text{Old time}}{\text{Improvement}} + 0.25 \times \text{Old time}$$

So

$$\frac{\text{New time}}{\text{Old time}} = \frac{\frac{0.75 \times \text{Old time}}{\text{Improvement}} + 0.25 \times \text{Old time}}{\text{Old time}}$$

Hence

$$\text{Improvement} = \frac{0.75 \times \text{Old time}}{0.5 \times \text{Old time} - 0.25 \times \text{Old time}} = 3$$

That is, the new computer must do multiplies three times faster.

b. No improvement to the speed of multiply can make the program run ten times faster. To see this, suppose that multiplies on the new machine were so fast that they took no time at all. Then program time on the new machine would still be 25% of the original time; i.e., the program would only run four times faster than before. (If we repeat the above sort of math, we end up with a negative value for Improvement!)