CS 201 Section 1 Midterm Exam Suggested Solutions

1.(a) Performance can only be properly compared by computing the total time required to execute P on each processor. This is given by the formula:

$$\text{Time for P} = \frac{\text{Cycles per Instruction} \times \text{Instructions executed for P}}{\text{Clock rate}}$$

Computing this for each machine, we get:

$$A: \quad \frac{1.5\ cycles/instr \times (3 \times 10^6)\ instr}{3 \times 10^8 cycles/sec} = 1.5 \times 10^{-2} sec$$

$$B: \quad \frac{1.1\ cycles/instr \times (2 \times 10^6)\ instr}{2 \times 10^8 cycles/sec} = 1.1 \times 10^{-2} sec$$

$$C: \quad \frac{3.5\ cycles/instr \times (6 \times 10^5)\ instr}{1 \times 10^8 cycles/sec} = 2.1 \times 10^{-2} sec$$

So machine B has the best performance on program P.

(b) Machine C, for two reasons: it requires 3.5 cycles per instruction (whereas a RISC machine is normally designed to execute at close to 1 cycle per instruction) and it requires many fewer instructions to execute P than does A or B, which suggests that its instructions are more powerful (whereas a RISC normally has less powerful instructions).

2. (Tanenbaum Ex 5.3) It is impossible. We can easily prove this by simply counting all the different bit patterns that will be needed to represent all instructions. There are 8 different choices for each register operand, so there must be $4 \times 8^3 = 2048$ patterns representing Group I instructions, $255 \times 8 = 2040$ patterns representing Group II instructions, and 16 patterns representing Group III instructions, for a total of 4104. A 12-bit word can only hold 4096 distinct patterns, so there is no way to do it.

We can also show the impossibility by attempting to work out a expanding-opcode solution. Since each operand selects among 8 registers, it requires $log_2 8 = 3$ bits. Thus we need $\log_2 4 + 3 \times 3 = 11$ bits to describe a Group I instruction, and these 11 bits are completely used, with no escape values left over. We can use the 12th bit to distinguish Group I instructions from Group II/III instructions; we then have to fit all the latter into 11 bits. Group II opcodes will fit in $\lceil \log_2 255 \rceil$ bits, with one "escape value" left over, which we can use to indicate a Group III instruction. The remaining 3 bits can be used in Group II to describe the single operand, so that works out fine. But to distinguish the Group III instructions, we need $\log_2 16 = 4$ bits, one more than we have left. Since all the encoding choices were already tight, we can conclude that no solution is possible.

3.(a) (Tanenbaum Ex. 2.12) The number of trits is given by

$$\lceil \log_3(2^{32}) \rceil = \lceil 32(\log_3 2) \rceil = \left\lceil \frac{32}{log_2 3} \right\rceil \approx \left\lceil \frac{32}{1.585} \right\rceil \approx \lceil 20.19 \rceil = 21$$

(b) You probably should not. Suppose there is room to store n bits on a chip today. The promised new technology will allow us to store n trits on a chip next year for a total data capacity of 3^n. But according to Moore's "Law," by next year the forces of binary-based progress will have managed to place about $1.6n$ bits on a chip. So the trit-based technology would only be worth investing in if $3^n > 2^{1.6n}$. Taking binary logs on both sides, this is equivalent to asking whether $n \log_2 3 > 1.6n$, i.e., whether $\log_2 3 > 1.6$. In fact, $\log_2 3 \approx 1.58$.

4. Suppose the total number of calls is n. Then the total execution times of each routine is given by $n \times$ (% of all calls) $\times$ (executution time per call).

This works out as follows:

`foo`	$50n$ μsecs
`bar`	$120n$ μsecs
`baz`	$100n$ μsecs
`bif`	$90n$ μsecs

Hence, we can do best by optimizing `bar`. The biggest improvement we can hope to make is to optimize it to the point of taking no time at all. In that case (and assuming, again optimistically, that the main program never took any time to start with) the total execution time after optimization will be $240n$ μsecs, compared to an original total execution time of $360n$ μsecs, representing a 33% improvement.

5.(a). The typist produces keystrokes at roughly

$$\frac{60 \text{ word/min} \times 5 \text{ char/word}}{60 \text{ secs/min}} = 5 \text{ char/sec}$$

which allows

$$\frac{1 \times 10^8 \text{ instrs/sec}}{5 \text{ chars/sec}} = 2 \times 10^7 \text{ instrs/char}$$

(b) In theory, the bus can handle $2 \times (8.33 \times 10^6) = 1.666 \times 10^7$ bytes per second. Since each pixel has one of 256 colors, it can be described by a single byte, so a single refresh of the whole screen requires sending $768 \times 1024 = 7.86 \times 10^5$ bytes. To keep up with the typist, the screen must be refreshed 5 times/sec, for a total data rate of 3.93×10^6 bytes/sec, well within the theoretical capacity of the bus. (In fact, as noted in class, the bus can probably handle only 4MBytes/sec, so it was ok to use that figure instead – in which case, it can still handle the refresh load, though only by a hair.)

6. The purpose of register windows is to improve the performance of procedural programs by allowing locals and arguments to be saved in (fast) on-chip registers rather than stored on the (slow) memory-based stack. By allowing only a limited number of registers to be visible in the active window, it is possible to limit the size of the register operand fields in instructions, while still increasing the number of on-chip registers available to the program as a whole.

7. Statement 1 references a few variables repeatedly, whereas statement 2 references a larger number of variables once each. On the SPARC, this allows statement 1 to be coded more compactly than statement 2, since the reused values can be kept in registers without the need for a `ld` instruction to get them from memory each time. So $B_{\text{SPARC}}(1) < B_{\text{SPARC}}(2)$. On the JVM, since there are no registers, all the variables will need to be reloaded from memory each time regardless of re-use, so $B_{\text{JVM}}(1) \approx B_{\text{JVM}}(2)$. So $R(1)$ and $R(2)$ have similar denominators, but $R(1)$ has a smaller numerator than $R(2)$; hence, $R(1) < R(2)$.