CS M151B Winter 2015 Midterm 1 Solutions

1. This was an open-ended question; our answers are below, but other answers could potentially be justified too. The justification was the important part.

- Hardware Complexity COULD INCREASE because the decoding unit has to check the length of each instruction in the variable length instructions.

- Cycle Time COULD INCREASE because the hardware complexity of the decoding unit could increase, and this in turn could increase the delay of the critical path.

- Instruction Count COULD INCREASE if the variable length is used inefficiently. It COULD also DECREASE since variable-length instructions can have wider immediate fields.

- The weighted average CPI COULD INCREASE or DECREASE depending on the mix of instructions the compiler uses with the new variable-length instructions. On the other hand, the CPI of each instruction type MUST STAY THE SAME because decoding in either the fixed-length or variable-length case takes exactly one cycle.

2. The CPI for each instruction type was different across the four versions of the test. Below are the formulas that you would use to compute each of the requested quantities. Knowing what was the right thing to compute was the most important part.

- a: CPI = 0.3*CPI(load) + 0.1*CPI(store) + 0.1*CPI(branch) + 0.1*CPI(alu)
- b: ET = IC * CPI / Rate
  - Equivalently: IC * CPI * CT, where CT = (1 s / 4B cycles)
  - Expression: (4B) * (CPI) / (4B cycles / s)
- c: The easiest thing to do is first to compute the new # of instructions per type, and then the new weighted average CPI:
  - New # of L, S, B, A:
    - L = 4B * 0.3 * 0.75 = 900M
    - S = 4B * 0.1 * 0.5 = 200M
    - B = 4B * 0.1 * 1 = 400M
    - A = 4B * 0.5 * 1.1 = 2.2B
    - New total = 3.7B instructions
  - New CPI = sum over i = {L,S,B,A} of ((new # for i) * (CPI for i) / (new total))
    - = (9*CPI(load) + 2*CPI(store) + 4*CPI(branch) + 22*CPI(alu)) / 37
- d: New ET = New IC * New CPI / New Rate
  = (3.7B) * New CPI / (4B cycles / 1.1 s)
  = 1.02778 * New CPI

3. Again, the delays were different across the four versions of the test. Below are the formulas that you would use to compute each of the requested delays. T# is the time for an #-input gate, and Tmux is the time for a MUX.

- T(G0) = T2
- T(P0) = T2
- T(G3) = T2
- T(P3) = T2
- T(C3) = 2*T4 + max(T2, T(C0)) = T2 + 2*T4
- T(S3) = T2 + T(C3) = 2*T2 + 2*T4
    - *S3 = XOR(XOR(A3, B3), C3)*
- T(C4) = 2*T5 + max(T2, T(C0)) = T2 + 2*T5
- T(Ga) = T2 + 2*T4
    - *Ga = ... + G0*P1*P2*P3*
- T(Pa) = T2 + T4
- T(C16) = T4 + max(T4 + T(C4), T3 + T(Ga))
= T2 + 2*T4 + 2*T5
    - *C16 = ... + Ga*Pb*Pg + C4*Pa*Pb*Pg*
- T(S19) = 2*T2 + 4*T4 + 2*T5
- T(C20) = T2 + 4*T5
- T(C24) = T(C4) = T2 + 2*T5
- T(C28) = T2 + 4*T5
- T(X-C32) = T(C28) + T5 + T5 = T2 + 6*T5
- T(Y-C32) = T(X-C32) = T2 + 6*T5
- T(C32 after) = Tmux + T2 + 6*T5
- T(S31 before) = T(S3 in block 28 … 31)
= T2 + max(T2, T(C28)) + 2*T4
= 2*T2 + 2*T4 + 4*T5
- T(S31 after) = Tmux + T(S31 before)
- T(max delay) = Tmux + max(**T2 + 6*T5**, 2*T2 + 2*T4 + 4*T5)