CS 152 Fall 2001 Homework #1 Solutions

1.51 Looking at the equation at the bottom of page 48, to maintain yield, defects per area must be decreased to compensate for the increase in die area.

1.52 Defects per Area = \( \frac{2}{\text{Die area}} \cdot \left( \sqrt{\frac{1}{\text{Yield}}} - 1 \right) \)

1.53 1980: Defects per Area = \( \frac{2}{0.16 \text{ cm}^2} \cdot \left( \sqrt{\frac{1}{0.48}} - 1 \right) = 5.5 \text{ defects/cm}^2 \)

1992: Defects per Area = \( \frac{2}{0.97 \text{ cm}^2} \cdot \left( \sqrt{\frac{1}{0.48}} - 1 \right) = 0.9 \text{ defects/cm}^2 \)

Defects per Area reduced by \( (5.5 - 0.9) / 5.5 = 84\% \)

2.4 M1: \( 200 \times 10^6 /10 \text{seconds} \times 3 \text{seconds} = 60 \times 10^6 \text{ instructions} \)
M2: \( 160 \times 10^6 / 5 \text{seconds} \times 4 \text{seconds} = 128 \times 10^6 \text{ instructions} \)

2.13 Compiler 1: M1 gets \( (4)(0.3) + (6)(0.5) + (8)(0.2) = 5.8 \text{ CPI} \)
\( \Rightarrow 400 \text{MHz}/5.8 \text{CPI} = 69.0 \text{ MIPS} \)
M2 gets \( (2)(0.3) + (4)(0.5) + (3)(0.2) = 3.2 \text{ CPI} \)
\( \Rightarrow 200 \text{MHz}/3.2 \text{CPI} = 62.5 \text{ MIPS} \)
\( \Rightarrow \text{M1} (69/62.5) = 1.1 \text{ times faster than M2} \)

Compiler 2: M1 gets \( (4)(0.3) + (6)(0.2) + (8)(0.5) = 6.4 \text{ CPI} \)
\( \Rightarrow 400 \text{MHz}/6.4 \text{CPI} = 62.5 \text{ MIPS} \)
M2 gets \( (2)(0.3) + (4)(0.2) + (3)(0.5) = 2.9 \text{ CPI} \)
\( \Rightarrow 200 \text{MHz}/2.9 \text{CPI} = 69.0 \text{ MIPS} \)
\( \Rightarrow \text{M2} (69/62.5) = 1.1 \text{ times faster than M1} \)

M1: 5.8 CPI for compiler 1
6.4 CPI for compiler 2
\( (4)(0.5) + (6)(0.3) + (8)(0.2) = 5.4 \text{ CPI} \)
\( \Rightarrow 400 \text{MHz}/5.4 \text{CPI} = 74.1 \text{ MIPS} \)
\( \Rightarrow \text{Use third-party compiler} \)

M2: 3.2 CPI for compiler 1
2.9 CPI for compiler 2
\( (2)(0.5) + (4)(0.3) + (3)(0.2) = 2.8 \text{ CPI} \)
\( \Rightarrow 200 \text{MHz}/2.8 \text{CPI} = 71.4 \text{ MIPS} \)
\( \Rightarrow \text{Use third-party compiler} \)

M1 is faster than M2 if the fastest compiler is used for both machines.
2.16 \(300 \times 10^6 (0.1 \times 30 + 0.15 \times 20 + 0.05 \times 50 + 0.7) = 2.76 \times 10^9\) instructions.

2.18 Mbase: \(\text{CPI} = (2)(0.4) + (3)(0.25) + (3)(0.25) + (5)(0.1) = 2.8\) CPI  
Mopt: \(\text{CPI} = (2)(0.4) + (2)(0.25) + (3)(0.25) + (4)(0.1) = 2.45\) CPI

2.24 Mopt is \((600/2.45)/(500/2.8) = 1.37\) times faster than Mbase.  
Therefore, if we implement the hardware, it would be 37\% faster than Mbase, 
while implementing the compiler only improves the machine 12\% in the same 
time.  In eight months, this would lead to \((1.37)(1.034)^2 = 1.43\) times faster than 
Mbase.  In comparison, in eight months, Mbase would be \((1.034)^8 = 1.31\) times 
faster.  However, by using both hardware and compiler improvements, Mboth 
could be 1.51 times faster than Mbase.  Therefore, both hardware and compiler 
improvements should be made.

2.32 Speedup due to faster multiply = \(\frac{1}{0.9 + 0.1/2} = 1.05\).  Not worth it, since this 
would result in a 20\% increased cycle time and only a 5\% speedup.

2.38 Program 1: \(\text{MFLOPS} = 10\), Program 2: \(\text{MFLOPS} = 100\).  This illustrates a 
problem since the execution time for the machines A and C do not reflect the 
difference in MFLOPS.  For example, computer A takes 1000 times as long for 
program 2 as program 1, which could mean program 2 has some floating point 
instructions that machine A does not, so multiple floating point instructions are 
performed for each instruction in program 2.  Likewise, program 1 might have 
floating point instructions that computer C does not, which would explain why it 
takes as much time for program 1 as it does for program 2 despite the difference 
in MFLOPS.

2.42 We want the program to execute in 33.3 seconds, therefore cutting 66.7 seconds 
from our original execution time.  Therefore, \(4/5\) of the time spent on floating 
point instructions must have been 66.7 seconds and \(83.3\%\) of the initial execution 
time must be spent on floating point instructions.

2.45 \(\text{MULT/4 + MEM + OTHER} = \text{MULT + MEM/2 + OTHER}\)  
\(\Rightarrow\) \(\text{MEM} = 1.5\) \(\text{MULT}\).  So any program that has 1.5 times as many memory 
accesses as multiplies will result in a tie.  For example, 20\% time spent on 
multiplies, 30\% on memory and 50\% on others.