CPSC 3300
Homework 1
Due 11:59PM, Sept 3rd
Submit your answers to canvas

Please provide sufficient space on your homework solutions so that your calculations and answers are easily readable and so that grading will be easier. Furthermore, except for the simplest questions, giving only the answer without showing your work is not acceptable. For the best chance at partial credit, show the generic equation you are starting with and any derivations needed to handle the information as given in the question, then plug in the values from the question. You may, of course, use a calculator for the homework. (Unlike the exams, the values in the homework questions are not necessarily chosen for ease of hand calculation.)

1. Moore's law

Read the following three articles. (You may use other published sources to answer the questions; please cite those sources if you do.)

- "Exponential Growth," Wikipedia: The Free Encyclopedia, accessed Aug. 22, 2015. [Online] en.wikipedia.org/wiki/Exponential_growth

- "Moore's law," Wikipedia: The free Encyclopedia. [Online] https://en.wikipedia.org/wiki/Moore%27s_law

- Chris Mack, "The Multiple Lives of Moore's Law". [Online] www.quora.com/

http://spectrum.ieee.org/semiconductors/processors/the-multiple-lives-of-moores-law

(5 points each of the following subquestions)

(a) Define exponential growth.

Exponential growth is a process that increases quantity over time. It's most easily interpreted as rapid growth. A cell splits into 2 then 4, then 8, then 16. That is exponential growth

(b) What did the original Moore's Law observe and project?

Moore's law is the observation that the number of transistors in a computer system doubles about every two years

(c) In your opinion, why has Moore's prediction been accurate over the years?

Moore's law was actually used as a project target for electronic devices so it almost became a self fulfilling prophecy. The need for smaller,faster and better computers is always present which allowed Moore's law to survive as long as it did.

(d) What is Dennard Scaling and why is it important in processor technology evolution.

Scaling law which states roughly that, as transistors get smaller, their power density stays constant. Meaning as transistor count doubles, power usage can stay relatively the same. Computers can get more powerful without necessarily using more energy

(e) According to Dr. Mack, scaling down or miniaturization marked the Moore's Law 2.0 era. Scaling down reduces the size of transistors.
List the feature sizes over the years to today.

10 µm 1971

1.5 µm 1981

600 nm 1990

130 nm 2001

22 nm 2012

5 nm 2020

(f) In your opinion, why are people discussing whether Moore's law is dead or not?

Because Moore's law has been around for around 50+ years and proven itself very accurate. If Moore's law is coming to an end then that means a potential revolution in computation is soon to come. Since the constant need for faster,smaller, better will never go away

2. (30pt)A processor P has a 4.0 GHz clock rate and has a CPI of 2.2.

(a) If the processor executes a program in 20 seconds, find the number of cycles and the number of instructions.

80 billion cycles and (80/2.2) 36.36 billion instructions

(b) What is the MIPS rate for the processor?

MIPS = Instruction Count / Execution time * 10^6
= 36.36 x 10^9 / 20 * 10^6
1818

(c) We are trying to reduce the execution time by 30% but this leads to an increase of 20% in the CPI. What clock rate should we have to get this time reduction?

Execution time of 14 seconds = (20 - (20*.3))
New CPI of 2.64 = (2.2 + (2.2*.2))

= 2.64 (Figure out how many more clock cycles)
= 4.8ghz for 36.30B instructions in 20 seconds
= 6.84ghz = ((x * 14 / 2.64) = 36.3)

3. (20pt)Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (class A, B, C, and D). P1 has a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 3.

Given a program with a dynamic instruction count of 1.0E6 instructions divided into classes as follows: 20% class A, 10% class B, 50% class C, and 20% class D.

(a) What is the global CPI?

Global CPI = (CPU-Time x Clock Rate)/IC)

CPU TIME = (10^5 +2 × 10^5 +5 × 10^5 × 3+2 × 10^5)/(2.5 × 10^9) = × 10.4 x 10^-4 sec

= 10.4 × 10^-4 × 2.5 × 10-9/10-6 = 2.6

(b) Find the clock cycles required to run the program on P1.

clock cycles (P1) = (10^5 × 1) + (2 × 10^ 5 × 2) × (5 × 10^5 × 3) + (2 * 10^5 × 3) = 26 * 10^5

4. (20pt)Assume for a given processor the CPI of arithmetic instructions is 1, the CPI of load/store instructions is 10,

and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns: 600 million arithmetic instructions, 250 million load/store instructions, 150 million branch instructions.

(a) Suppose we find a way to double the performance of the arithmetic instructions. What is the overall speedup of our machine?

1*600 + 10*150 + 3*250) = 2850
1*(.5 *600) + 10*150 + 3*250) = 2550
2850 / 2550 = 1.117
The new machine would be 1.117x faster to compared to the old one

(b) If we find a way to improve the performance of the arithmetic instructions by 10 times, what is the overall speedup of our machine?

1*600 + 10*150 + 3*250) = 2850
*100(.1 *600) + 10*150 + 3*250) = 2350
2850 / 2350 = 1.212
The new machine would be 1.212x faster to compared to the old one

5. (50pt) Use perf and time tool to profile program execution. Run all experiments on one of the school linux machines.

download the whetstone http://www.netlib.org/benchmark/whetstone.c benchmark to your home directory.

compile whetstone. You may need to explicitly specify the math lib folder and link to it, e.g.,

```
gcc -o whetstone whetstone.c -lm

#link the math with -lm
```

On the same machine (one in the lab), examine how compiler optimization levels and options change the number of instructions for the program whetstone and the number of CPU cycles to execute the program. Use gcc to compile your program.

The higher the level of optimization applied, the lower amount of instructions and total CPI

(a) use perf to profile the execution of whetstone. For information
about perf usage, type command

perf

you will see the commands that perf supports. You are encouraged to find online articles on perf and read them.

(b) Use utility time to profile the execution of whetstone that loops 200,000 times

time ./whetstone 200000

Explain the timing output and the definitions.
If the timings from perf and time are different, explain the cause.

Time:

Real = 2.959 sec Total time elapsed

User = 2.957 sec Time used by system overhead

Sys = 0.000 sec Time used by utility

Perf:

Real = 2.9438

User = 2.9391

Sys = .0039

The difference is very subtle but perf is from what I read a much more accurate source for timings. The difference could be caused from the CPU maybe being a little bit more busy when calling time ./whetstone compared to when perf was called. But there are many factors that could have affected the CPU's available resources at the time of execution

(c) Examine the following levels/options:

a. -O0
b. -O1
c. -O2
d. -O3
e. -O3 -funroll-loops

Use a table to show the instruction count, #cycles, IPC, and time for each of the experiments, and calculate the speedup based on the execution time with -O0. Paste your screen shot at the end.

	IC	#Cycles	IPC	Time	Speedup
-O0	23,234,212,938	11,717,294,701	1.98	2.9438	0%
-O1	9,675,019,068	6,901,229,090	1.401	1.7390	171% or 1.71x
-O2	3,470,240,350	3,603,976,577	0.962	0.9124	322% or 3.22x
-O3	3,366,622,186	3,501,547,261	0.961	0.8840	333% or 3.33x
-O3 -funroll-loops	2,646,951,884	3,348,567,785	.790	0.8475	347% or 3.473

-O0

```
Performance counter stats for './a.out 200000':

          2,943.24 msec task-clock                #    0.999 CPUs utilized

                 7      context-switches          #    0.002 K/sec

                 0      cpu-migrations            #    0.000 K/sec

                82      page-faults               #    0.028 K/sec

    11,719,080,630      cycles                    #    3.982 GHz

    23,234,173,247      instructions              #    1.98  insn per cycle

     2,335,979,051      branches                  #  793.676 M/sec

            48,528      branch-misses             #    0.00% of all branches


       2.945523784 seconds time elapsed

       2.943627000 seconds user
       0.000000000 seconds sys
```

-O1

```
Performance counter stats for './a.out 200000':

          1,738.24 msec task-clock                #    0.999 CPUs utilized

                 4      context-switches          #    0.002 K/sec

                 0      cpu-migrations            #    0.000 K/sec

                76      page-faults               #    0.044 K/sec

     6,912,030,245      cycles                    #    3.976 GHz

     9,674,877,066      instructions              #    1.40  insn per cycle

     1,514,121,592      branches                  #  871.068 M/sec

            19,117      branch-misses             #    0.00% of all branches


       1.740211426 seconds time elapsed

       1.738683000 seconds user
       0.000000000 seconds sys
```

-O2

```
Performance counter stats for './a.out 200000':

          909.98 msec task-clock                #    0.998 CPUs utilized

               5      context-switches          #    0.005 K/sec

               0      cpu-migrations            #    0.000 K/sec

              74      page-faults               #    0.081 K/sec

   3,603,553,526      cycles                    #    3.960 GHz

   3,470,184,067      instructions              #    0.96  insn per cycle

     541,769,636      branches                  #  595.364 M/sec

          14,627      branch-misses             #    0.00% of all branches


     0.911398149 seconds time elapsed

     0.910345000 seconds user
     0.000000000 seconds sys
```

-O3

```
Performance counter stats for './a.out 200000':

          882.19 msec task-clock                #    0.998 CPUs utilized

               4      context-switches          #    0.005 K/sec

               0      cpu-migrations            #    0.000 K/sec

              77      page-faults               #    0.087 K/sec

   3,501,471,389      cycles                    #    3.969 GHz

   3,366,616,423      instructions              #    0.96  insn per cycle

     519,377,857      branches                  #  588.734 M/sec

          15,249      branch-misses             #    0.00% of all branches


     0.883670700 seconds time elapsed

     0.882582000 seconds user
     0.000000000 seconds sys
```

-O3 -funroll-loops

```
Performance counter stats for './whetstone 200000':

          846.10 msec task-clock                #    0.998 CPUs utilized

               3      context-switches          #    0.004 K/sec

               0      cpu-migrations            #    0.000 K/sec

              78      page-faults               #    0.092 K/sec

   3,348,567,785      cycles                    #    3.958 GHz

   2,646,951,884      instructions              #    0.79  insn per cycle

     395,815,715      branches                  #  467.811 M/sec

          14,656      branch-misses             #    0.00% of all branches


     0.847589148 seconds time elapsed

     0.842500000 seconds user
     0.003992000 seconds sys
```