Patrick Woodrum
pwoodru
CPSC 3300
Homework 1
Due 11:59PM Thursday, January 28
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.

1. (30pt) A processor P has a 2.7 GHz clock rate and has a CPI of 1.2.

(a) If the processor executes a program in 6 seconds, find the number of cycles and the number of instructions.

Clock Cycles = CPU Time x Clock Rate
= 6s x 2.7GHZ
= 16.2 x 10^9 cycles

To find instruction count (IC) use:
IC = (CPU Time x Clock Cycle Time) / CPI
IC = (6s x 2.7x10^9) / 1.2
IC = (1.62x10^10) / 1.2
IC = 1.35x10^10
or IC = 13,500,000,000

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

MIPS = Clock rate / CPIx10^6
MIPS = 2.7x10^9 / 1.2x10^6
MIPS = 2250

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

Execution time = (instructions x CPI) / clock rate
Execution time(new) = 0.8 x Execution time(old)
Therefore
CPI(new)/clock rate(new) = 0.8 x CPI(old)/clock rate(old)
CPI(new) = 1.2, so
1.2/clock rate(new) = 0.8/clock rate(old)
Clock rate(new) = 1.2/0.8 x clock rate(old)
Clock rate(new) = 1.5 x clock rate(old)
Therefore the old clock rate must increase by 50%
Increasing 2.7GHZ by 50% yields <u>4.05GHZ</u>

Patrick Woodrum
pwoodru

2. (20pt) Consider two different implementation 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 (class A), 2 (class B), 3 (class C), and 5 (class D).

   Given a program with a dynamic instruction count of 25 million instructions divided into classes as follows: 15% class A, 50% class B, 25% class C, and 10% class D.

   (a) What is the global CPI?
   A = 1.5x10^5, B = 5x10^5, C = 2.5x10^5, D = 10^5
   Time(P1) = (1.5x10^5 + 5x10^5 x 2 + 2.5x10^5 x 3 + 10^5 x 4) / (2.5x10^9)
   Time(P1) = 9.2x10^-4

   CPI(P1) = (9.2x10^-4) x (2.5x10^9) / 10^6
   CPI(P1) = 2.3

   (b) Find the clock cycles required to run the program on P1.
   Clock Cycles utilize part of the equation above.
   A = 1.5x10^5, B = 5x10^5, C = 2.5x10^5, D = 10^5
   Clock Cycles(P1) = (1.5x10^5 + 5x10^5 x 2 + 2.5x10^5 x 3 + 10^5 x 4)
   Clock Cycles = 2.3x10^6

3. (20pt) Assume for a given processor the CPI of arithmetic instructions is 2, the CPI of load/store instructions is 6, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns: 240 million arithmetic instructions, 70 million load/store instructions, 100 million branch instructions.

   (a) Suppose we find a way to double the performance of the arithmetic instructions. What is the speedup of our machine?
   The old clock cycles was:
   (2 x 240) + (6 x 70) + (3 x 100) = 1200
   To double arithmetic instructions our clock cycles becomes:
   (1 x 240) + (6 x 70) + (3 x 100) = 960

   Clock cycles(new)/clock cycles(old) = 960/1200 = 0.8
   The speedup of the machine after doubling the performance of arithmetic instructions was 20%.

   (b) If in addition to the arithmetic instruction optimization we also find a way to double the performance of the load/store instructions, what is the overall speedup of our machine?
   The old clock cycle (with optimization) was:
   (1 x 240) + (6 x 70) + (3 x 100) = 960
   To double the load/store instructions:
   (1 x 240) + (3 x 70) + (3 x 100) = 750

   Clock cycles(new)/clock cycles(old) = 750/960 = 0.78125
   The speedup of the machine was a further increase of 21.875%.
   The overall speedup was 20% + 21.875% = 41.875%.

Patrick Woodrum
pwoodru

4. (30pt) On machine newton using the **perf** tool, 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. Refer to the following web page for information on how to use **perf** to count the number of instructions and cycles among other statistics: https://perf.wiki.kernel.org/index.php/Tutorial#Counting_with_perf_stat

   I: Download the **whetstone** benchmark to your home directory:
   http://www.netlib.org/benchmark/whetstone.c

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

   ```
   gcc -o whetstone whetstone.c -lm
   #link the math library with -lm
   ```

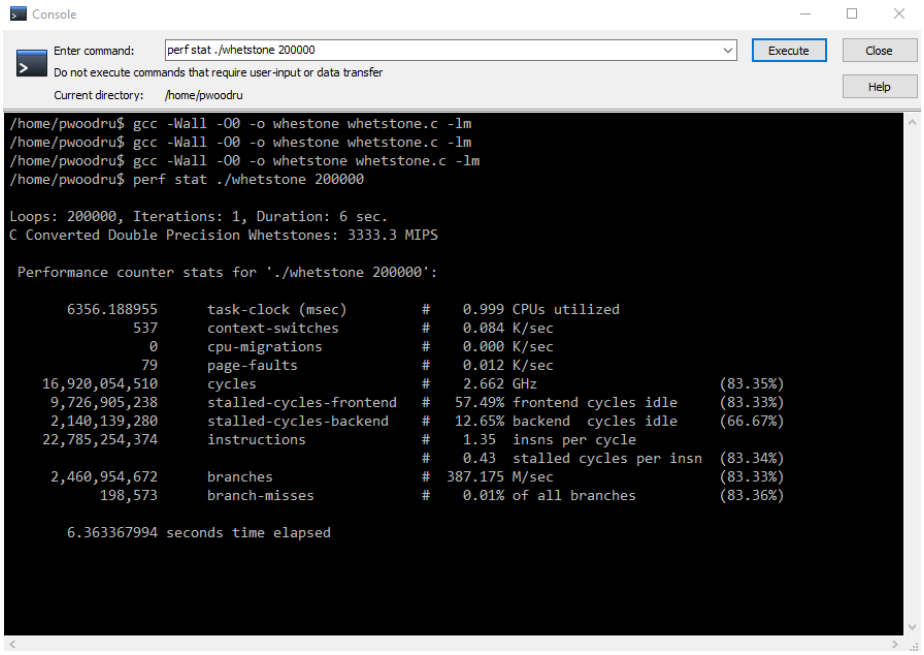
   II: Examine the performance of **whetstone** looping 200,000 times (**./whetstone 200000**) compiled with the following levels/options:
   a. -O0
   b. -O1
   c. -O2
   d. -O3
   e. -O3 -funroll-loops

   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 (ms)	Speedup
-O0	22,785,254,374	16,920,054,510	1.35	6363	0
-O1	10,957,309,469	8,819,209,982	1.24	3466	45.6%
-O2	6,056,123,284	6,141,991,202	0.99	2503	27.8%
-O3	6,021,948,416	6,127,308,031	0.98	2498	0.2%
-O3 -funroll-loops	5,015,729,716	5,442,507,576	0.92	2245	10.1%

Screenshots:

Patrick Woodrum
pwoodru




```
/home/pwoodru$ gcc -Wall -O0 -o whestone whetstone.c -lm
/home/pwoodru$ gcc -Wall -O0 -o whestone whetstone.c -lm
/home/pwoodru$ gcc -Wall -O0 -o whetstone whetstone.c -lm
/home/pwoodru$ perf stat ./whetstone 200000

Loops: 200000, Iterations: 1, Duration: 6 sec.
C Converted Double Precision Whetstones: 3333.3 MIPS

 Performance counter stats for './whetstone 200000':

      6356.188955      task-clock (msec)         #    0.999 CPUs utilized
              537      context-switches          #    0.084 K/sec
                0      cpu-migrations            #    0.000 K/sec
               79      page-faults               #    0.012 K/sec
   16,920,054,510      cycles                    #    2.662 GHz                      (83.35%)
    9,726,905,238      stalled-cycles-frontend   #   57.49% frontend cycles idle     (83.33%)
    2,140,139,280      stalled-cycles-backend    #   12.65% backend  cycles idle     (66.67%)
   22,785,254,374      instructions              #    1.35  insns per cycle
                                                #    0.43  stalled cycles per insn  (83.34%)
    2,460,954,672      branches                  #  387.175 M/sec                    (83.33%)
          198,573      branch-misses             #    0.01% of all branches          (83.36%)

      6.363367994 seconds time elapsed
```

```
/home/pwoodru$ gcc -Wall -O1 -o whetstone whetstone.c -lm
/home/pwoodru$ perf stat ./whetstone 200000

Loops: 200000, Iterations: 1, Duration: 3 sec.
C Converted Double Precision Whetstones: 6666.7 MIPS

 Performance counter stats for './whetstone 200000':

      3460.495749      task-clock (msec)         #    0.998 CPUs utilized
              288      context-switches          #    0.083 K/sec
                0      cpu-migrations            #    0.000 K/sec
               75      page-faults               #    0.022 K/sec
    8,819,209,982      cycles                    #    2.549 GHz                      (83.35%)
    4,545,662,944      stalled-cycles-frontend   #   51.54% frontend cycles idle     (83.36%)
    1,862,643,971      stalled-cycles-backend    #   21.12% backend  cycles idle     (66.74%)
   10,957,309,469      instructions              #    1.24  insns per cycle
                                                #    0.41  stalled cycles per insn  (83.36%)
    1,633,000,511      branches                  #  471.898 M/sec                    (83.36%)
           36,409      branch-misses             #    0.00% of all branches          (83.28%)

      3.465877827 seconds time elapsed
```

```
/home/pwoodru$ gcc -Wall -O2 -o whetstone whetstone.c -lm
/home/pwoodru$ perf stat ./whetstone 200000

Loops: 200000, Iterations: 1, Duration: 3 sec.
C Converted Double Precision Whetstones: 6666.7 MIPS

 Performance counter stats for './whetstone 200000':

      2498.125735      task-clock (msec)         #    0.998 CPUs utilized
              212      context-switches          #    0.085 K/sec
                0      cpu-migrations            #    0.000 K/sec
               75      page-faults               #    0.030 K/sec
    6,141,991,202      cycles                    #    2.459 GHz                      (83.34%)
    4,016,291,532      stalled-cycles-frontend   #   65.39% frontend cycles idle     (83.36%)
    1,864,407,036      stalled-cycles-backend    #   30.36% backend  cycles idle     (66.73%)
    6,056,123,284      instructions              #    0.99  insns per cycle
                                                #    0.66  stalled cycles per insn  (83.36%)
      841,009,101      branches                  #  336.656 M/sec                    (83.35%)
           28,833      branch-misses             #    0.00% of all branches          (83.32%)

      2.502805475 seconds time elapsed
```

Patrick Woodrum
pwoodru

```
/home/pwoodru$ gcc -Wall -O3 -o whetstone whetstone.c -lm
/home/pwoodru$ perf stat ./whetstone 200000

Loops: 200000, Iterations: 1, Duration: 3 sec.
C Converted Double Precision Whetstones: 6666.7 MIPS

 Performance counter stats for './whetstone 200000':

       2493.290543      task-clock (msec)         #    0.998 CPUs utilized
               210      context-switches          #    0.084 K/sec
                 0      cpu-migrations            #    0.000 K/sec
                76      page-faults               #    0.030 K/sec
     6,127,308,031      cycles                    #    2.458 GHz                      (83.38%)
     4,000,625,083      stalled-cycles-frontend   #   65.29% frontend cycles idle     (83.29%)
     1,898,448,008      stalled-cycles-backend    #   30.98% backend  cycles idle     (66.66%)
     6,021,948,416      instructions              #    0.98  insns per cycle
                                                  #    0.66  stalled cycles per insn  (83.28%)
       816,887,378      branches                  #  327.634 M/sec                    (83.38%)
            29,889      branch-misses             #    0.00% of all branches          (83.40%)

       2.497963006 seconds time elapsed
```

```
/home/pwoodru$ gcc -Wall -O3 -funroll-loops -o whetstone whetstone.c -lm
/home/pwoodru$ perf stat ./whetstone 200000

Loops: 200000, Iterations: 1, Duration: 3 sec.
C Converted Double Precision Whetstones: 6666.7 MIPS

 Performance counter stats for './whetstone 200000':

       2240.948425      task-clock (msec)         #    0.998 CPUs utilized
               190      context-switches          #    0.085 K/sec
                 0      cpu-migrations            #    0.000 K/sec
                76      page-faults               #    0.034 K/sec
     5,442,507,576      cycles                    #    2.429 GHz                      (83.24%)
     3,741,091,669      stalled-cycles-frontend   #   68.74% frontend cycles idle     (83.23%)
     2,240,581,260      stalled-cycles-backend    #   41.17% backend  cycles idle     (66.82%)
     5,015,729,716      instructions              #    0.92  insns per cycle
                                                  #    0.75  stalled cycles per insn  (83.42%)
       596,485,616      branches                  #  266.176 M/sec                    (83.41%)
            30,878      branch-misses             #    0.01% of all branches          (83.31%)

       2.245343023 seconds time elapsed
```