# COIS 2300H Assignment 1 Solutions ish

This assignment is worth 10% of your final grade. Bonus marks will not let you exceed 100%.

Theory Questions from the textbook **0.5 marks** for each full question (3.5 marks total)

(§ references a particular section in the book)

In the ~~un~~likely event that you discover solutions to these problems on the internet I have changed (some of) them, and marked that change by ~~crossing out~~ the original. You are to answer with the changes I made. Feel free to use the solutions to help you figure out how to solve the problem, that’s the idea.

# A

**1.4**  <§1.4> Assume a color display using ~~8~~ 10 bits for each of the primary colours (red, green, blue) per pixel and a frame size of ~~1280 × 1024~~ **3440x1440 (widescreen)**

**a.** What is the minimum size in bytes of the frame buffer to store a frame?

**b.** How long would it take, at a minimum, for the frame to be sent over a **400**

Mbit/s network?  
c. **Sri/Alaadin added question:** What bandwidth do you need (at a minimum) to transmit this data at **144** frames per second?

(note that in **a** we are ignoring the fact that data would mostly likely be packed into 32 bits per pixel regardless of colour depth).

|  |
| --- |
| **a.** 3440x1440x30(bits/pixel) = 148 608 000 bits/8 bits/byte = 18 576 000 (~18.5Mb)  **b.** 148608000bits/400000000 bits/s = 0.37152 seconds. 148 c 148608000bits/frame\*144frames/sec = 21 399 552 000 bits = ~21.4 Gb/s |

# B

**1.5** [4] <§1.6> Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a ~~3 GHz~~  4.4GHz clock rate and a CPI of 1.5. P2 has a ~~2.5 GHz~~ 3.3GHzclock rate and a CPI of 1.0. P3 has a 4.1 GHz clock rate and has a CPI of 2.2.  
**a.** Which processor has the highest performance expressed in instructions per second? **Sri: give that number**

**b.** If the processors each execute a program in 10 seconds, find the number of

cycles and the number of instructions.

**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?

|  |
| --- |
| All I did was change some numbers around,   Here are the original numbers, if they have the original numbers then they cheated, if they did the right numbers and came up with something like the right result give them full marks.    a. IPS1  IPS2  IPS3  b. instructions1  clock cycles1  instructions2  clock cycles2  instructions3  clock cycles3  c  Therefore This indicates a 71 increase in clock rate |
|  |

# C

**1.12** Section 1.10 cites as a pitfall the utilization of a subset of the performance equation as a performance metric. To illustrate this, consider the following two

processors. P1 has a clock rate of 4 GHz, average CPI of 0.9, and requires the

execution of 5.0E9 instructions. P2 has a clock rate of 3 GHz, an average CPI of

0.75, and requires the execution of 1.0E9 instructions.  
**1.12.1** [5] <§§1.6, 1.10> One usual fallacy is to consider the computer with the

largest clock rate as having the largest performance. Check if this is true for P1 and

**1.12.2** [10] <§§1.6, 1.10> Another fallacy is to consider that the processor executing

the largest number of instructions will need a larger CPU time. Considering that

processor P1 is executing a sequence of 1.0E9 instructions and that the CPI of

processors P1 and P2 do not change, determine the number of instructions that P2

can execute in the same time that P1 needs to execute 1.0E9 instructions.

**1.12.3** [10] <§§1.6, 1.10> A common fallacy is to use MIPS (millions of

instructions per second) to compare the performance of two diff erent processors,

and consider that the processor with the largest MIPS has the largest performance.

Check if this is true for P1 and P2.

**1.12.4** [10] <§1.10> Another common performance fi gure is MFLOPS (millions

of fl oating-point operations per second), defi ned as

MFLOPS = No. FP operations / (execution time × 1E6)

but this fi gure has the same problems as MIPS. Assume that 40% of the instructions

executed on both P1 and P2 are fl oating-point instructions. Find the MFLOPS

fi gures for the programs.

**1.12.Sri/Alaadin -** Explain in ~100-200 words what this question is trying to illustrate, and what metric(s) you should use to compare CPU performance given there are obvious problems with clock rate, IPC, IPS etc.

|  |
| --- |
|  |
| 1.12. Sri: The point of the question is that different metrics are useful at different times. You need to measure performance relative to the actual workload, or at least a reasonable approximation thereof.   There are a bunch of directions they could go with this question.  If you are comparing two CPU’s of the same basic type but say one has more cores, then maybe IPS is the overall better measure (if the problem can be parallelised).  If you are working on a numerical simulation then FLOPS might make the most sense – even across disparate architectures, because the workload will be FLOP heavy. |

# D

**1.15** <§1.8> **Sri/Alaadin Note: Do this problem using a spreadsheet** When a program is adapted to run on multiple processors in a multiprocessor system, the execution time on each processor is comprised of computing time and the overhead time required for locked critical sections and/or to send data from one processor to another.

Assume a program requires t = 100 s of execution time on one processor. When run *p* processors, each processor requires t/p s, as well as an additional 4 s of overhead, irrespective of the number of processors. Compute the per-processor execution time for ~~2, 4, 8, 16, 32, 64, and 128~~  2, 4, 6, 8, 12, 16, 24, 28, 32, 56, 64, 224, and 448 processors. For each case, list the corresponding speedup relative to a single processor and the ratio between actual speedup versus ideal speedup (speedup if there was no overhead).  
  
(Note: those numbers of nodes are all real CPUs or at least multi CPU configurations you can buy for home/server/HPC systems).

|  |
| --- |
|  |
| Students should have an extra couple of rows on the table, but it just continues the trend.   Make sure they’re doing the ones in the question, that’s how we know they aren’t cheating     |  |  |  |  |  | | --- | --- | --- | --- | --- | | Processor | Execution Time/processor | Total Time | Relative Speedup | Actual Speedup  / Ideal Speedup | | 1 | 100s | 100 |  |  | | 2 | 50 | 54 | 100/54 = 1.85 | 1.85/2 = 0.93 | | 4 | 25 | 29 | 100/29 = 3.45 | 3.45/ 4 = 0.86 | | 6 | 16.7 | 20.7 | 100/20.7 = 4.83 | 4.83/6 = 0.81 | | 8 | 12.5 | 16.5 | 100/16.5 = 6.06 | 6.06/8 = 0.76 | | 12 | 8.3 | 12.3 | 100/12.3 = 8.13 | 8.13/12 = 0.68 | | 16 | 6.25 | 10.25 | 100/10.25 = 9.76 | 9.76/16 = 0.61 | | 24 | 4.16 | 8.16 | 100/8.16 = 12.25 | 12.25/24 = 0.51 | | 28 | 3.57 | 7.57 | 100/7.57 = 13.21 | 13.21/28 = 0.47 | | 32 | 3.125 | 7.125 | 100/7.125 = 14.04 | 14.04/ 32 = 0.44 | | 56 | 1.79 | 5.79 | 100/5.79 = 17.27 | 17.27/56 = 0.31 | | 64 | 1.5625 | 5.5625 | 100/5.5625 = 17.98 | 17.98/64 = 0.28 | | 224 | 0.45 | 4.45 | 100/4.45 = 22.47 | 22.47/224 = 0.10 | | 448 | 0.223 | 4.223 | 100/4.223 = 23.68 | 23.68/448 = 0.05 | |

# E

**2.7** <§2.3> Show how the value**s** ~~0xabcdef12~~ 0x0134 433C **AND** -23002020 (that’s base 10, so convert to hex first), would be arranged in memory of a little-endian and a big-endian machine. Assume the data is stored starting at address 0. – (By endianness we mean byte ordering <http://www.yolinux.com/TUTORIALS/Endian-Byte-Order.html> has a guide)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Note the trick with this question is recognising that the first bit in -23002020 needs to be a 1 for negative, and to expect that if it’s stored byte wise it’s an 8 byte arrangement. They might think of -23002020 in a 7 byte arrangement which gets them a slightly different answer but same basic idea.  Big Endian  0x0134 433C -23002020 = 0b 1000 0001 0101 1110 1111 1011 1010 0100‬ = 0x 815E FBA4‬  If they treat this as a 7 byte setup it is 09 5E FB A4   |  |  |  |  | | --- | --- | --- | --- | | Address | data | Address | Address | | 12 | 3C |  | A4 | | 8 | 43 |  | FB | | 4 | 34 |  | 5E | | 0 | 01 |  | 81 |   Little endian 3C 43 34 01 A4 FB 5E 81   |  |  |  |  | | --- | --- | --- | --- | | Address | Data | Address | Data | | 12 | 01 |  | 81 | | 8 | 34 |  | 5E | | 4 | 43 |  | FB | | 0 | 3C |  | A4 | |

# F

**2.8** [5] <§2.4> Translate ~~0xabcdef12~~ 0x134433C into decimal. **Sri note: Show your steps, by hand**

|  |
| --- |
| 20202300 |

# G

**2.46** 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: ~~500~~ 700 million

arithmetic instructions, ~~300~~ 200 million load/store instructions, ~~100~~ 80 million branch

instructions.

**2.46.1** [5] <§2.19> Suppose that new, more powerful arithmetic instructions are

added to the instruction set. On average, through the use of these more powerful

arithmetic instructions, we can reduce the number of arithmetic instructions

needed to execute a program by 25%, and the cost of increasing the clock cycle

time by only 10%. Is this a good design choice? Why?

1. **2.46.2** [5] <§2.19> Suppose that we find a way to double the performance of

arithmetic instructions. What is the overall speedup of our machine? What if we

find a way to improve the performance of arithmetic instructions by 10 times?

|  |
| --- |
| I tinkered with the numbers but it’s the same basic equations  2.46.2 is of course wrong because I changed the numbers.   base case: 700\*1+200\*10+80\*3 = 2940 cycles \* clock cycle time  a) ((0.75\*700\*1)+(200\*10)+(80\*3))\*1.1 = 2765 \* 1.1 = 3041 cycles, so not actually an improvement   1. (700\*.5) +(2000) + (240) = 2590   (700+0.1) + (2240) = 2310.  Speedup is 2940/2590 and 2940/2310, whatever those work out to (13% and 27% I guess). |

# H Big Picture theory question

<http://www.anandtech.com/show/10337/the-intel-broadwell-e-review-core-i7-6950x-6900k-6850k-and-6800k-tested-up-to-10-cores/8>

The problem with those benchmarks is that processors which are sometimes many times faster than each other (e.g. the same architecture and frequency but twice as many cores etc.) don’t seem that much faster in real applications, and trying to compare different architectures is quite challenging.

Describe a way for an end user to compare CPU’s and decide which one to buy. Some possible things to consider: Performance for certain workloads, value, cost, likely future performance or whatever else you think might be important.

(When talking about cost you can consider just the CPU itself, or overall system cost. AMD Ryzen parts have very different performance with different speed RAM for example, which makes this even more complicated).

|  |
| --- |
| The link itself actual has an answer but it boils down to different numbers of instructions per clock, different levels of parallelization and single/multi core dependencies, how well different programs context switch on different processors etc. The reason pretty much all CPU’s end up the same is also that all CPU’s can do all of the calculations requested with time to spare, while the GPU is the one that is frame limiting. |

# I Programming Problems 2 Marks Each

If you really don’t like Mars (or can’t get it to work) use SPIM or something else. Submit your source code, and documentation on how you tested it.

**Program 1:** Write a MIPS program that asks the user for two number inputs (we’ll call them *a* and *b*), calculate and print to screen: , if *b* is not 0, also calculate . Then check if a + b is in the range 10 … 20. Stick to integers, but allow positive or negative values (don’t check input types, and don’t test edge cases like very large integers)

**Program 2:** Write a MIPS program that will ask the user for a negative integer (we will call this *n* ), check if it is negative, if not throw an error. Iteratively calculate the factorial of n squared (). Assume n is between 0 and -5, there are some issues with factorial calculations with big numbers which we will discuss in class… don’t sign yourself up for those). Recursion is a problem for future assignments, so do this iteratively.

|  |
| --- |
| For both sets of programming questions it’s up to the student to show code and testing. |