Computing Systems Architecture

Summer 2020 NYU ECE 6913 INET

Please fill in your name: ________________________________

Final, August 17th 2020

Maximum time: 140 minutes : **10:30 PM -1:00 PM** [140 minutes + 10 minutes to assemble PDF and upload]

Open Book, Open Notes,

Calculators allowed.

Must show your work in steps – to get any credit

This is NOT a group project You may NOT discuss, share your Quiz solutions with anyone else.

You must stay logged in to Zoom throughout the Quiz

Instructor available online if you have questions on the Quiz, during the Quiz – enter question in Zoom chat box at any time during Quiz

This Test has 4 problems. Please attempt all of them. Please show **all work**. Please write **legibly**

1. Please be sure to have 5-10 sheets of white or ruled paper & a Pencil, Eraser
2. Write down your solutions on 8.5 x 11 sheets of white paper, single sided with your name printed in top right corner of each sheet and with **Page Number and Problem number identified clearly on each sheet**
3. **Stop working on your Quiz at 12:50 PM** – you have 10 minutes to scan/take pictures of each sheet and upload them as completed PDF assignment to NYU Classes – you may use any of several smartphone apps to integrate your scans/pictures of sheets into a PDF file
4. Take pictures of each sheet and **upload** the PDF of all sheets after checking you have all sheets in the right order **by 1:00 PM latest.**
5. You may use iPAD to write down your solutions directly rather than on paper
6. Portal **will close at 1:00 PM** not allowing upload of your quiz after 1:00 PM

Answer any 5 of the following 6 questions. Answering all 6 will earn you extra credit. ***Note that 1 (ii) is also extra credit***

1. Students M, O & P are building custom hardware and compilers for their project

The following observations are known. CPI = 1

• Student O has built his design and it has a known execution time for a new program of 600 seconds.

• Student M proposes a single cycle data path, and plans to use a compiler that will generate 300 Billion dynamic instructions per execution.

• Student P proposes a pipelined data path, with a clock period of 1.2 ns. His compiler will generate 400 Billion instructions. Of these 400 Billion instructions, 25% are loads, and 20% are branches. 40% of all loads cause a 2-cycle load-use stall. The penalty for a mis-predicted branch is 5 cycles. The processor has full bypassing, and does not suffer from any other stalls.

(i) How fast does M need to make his clock cycle in order to make the program run faster than O's?

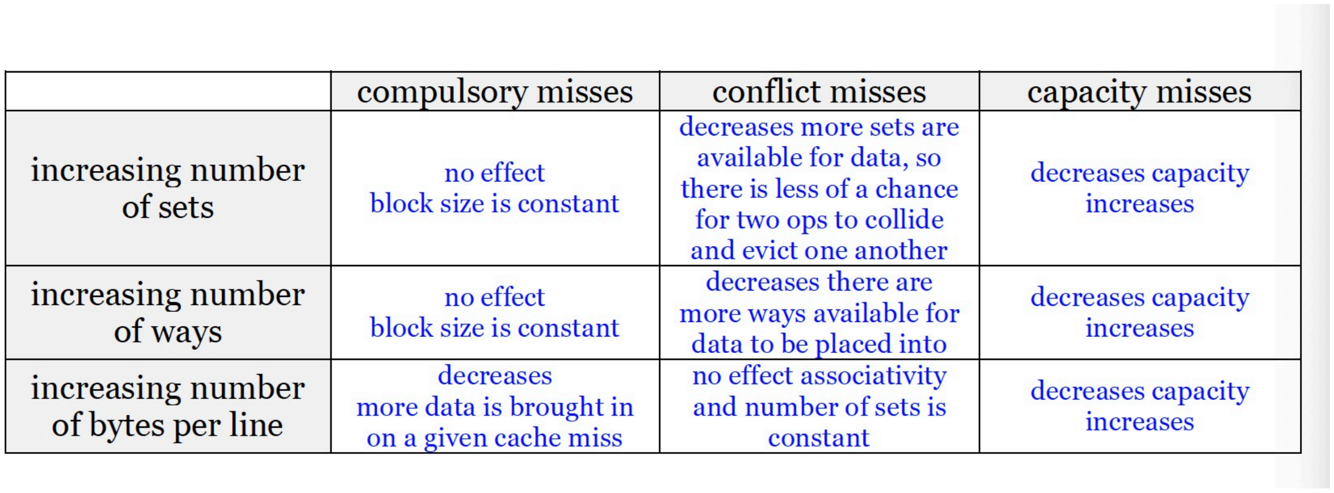
(ii) [*Optional - Extra Credit*] How accurate does P's branch predictor need to be in order to make the program run faster than O's?

2. Assume that you have a computer with 1 clock cycle per instruction (CPI=1) when all accesses to memory are in cache. The only accesses to data come from load and store instructions. Those accesses account for 25 % of the total number of instructions. Miss penalty is 50 clock cycles and miss rate is 5 %. Determine the speedup obtained when there is no cache miss compared to the case when there are cache misses

3. Suppose that when Program A is run, the user CPU time is 3 seconds, the elapsed wall clock time is 4 seconds, and the system performance is 10 MFLOP/sec. Assume that there are no other processes taking any significant amount of time, and the computer is either doing calculations in the CPU, or doing I/O, but it can't do both at the same time. We now replace the processor with one that runs six times faster, but doesn't affect the I/O speed. What will the user CPU time, the wall clock time, and the MFLOP/sec performance be now?

4. Mark whether the following modifications to cache parameters will cause each of the categories to increase, decrease, or whether the modification will have no effect. You can assume the baseline cache is set associative. Explain your reasoning. Assume that in each case the other cache parameters (number of sets, number of ways, number of bytes/line) and the rest of the machine design remain the same

	compulsory misses	conflict misses	capacity misses
increasing number of sets			
increasing number of ways			
increasing number of bytes per line			



	compulsory misses	conflict misses	capacity misses
increasing number of sets	no effect block size is constant	decreases more sets are available for data, so there is less of a chance for two ops to collide and evict one another	decreases capacity increases
increasing number of ways	no effect block size is constant	decreases there are more ways available for data to be placed into	decreases capacity increases
increasing number of bytes per line	decreases more data is brought in on a given cache miss	no effect associativity and number of sets is constant	decreases capacity increases

5. Assume the following register contents:

```
x5 = 0x00000000CCCCCCCC, x6 = 0x1122334455667788
```

For the register values shown above, what is the value of x7 for the following sequence of instructions?

```
slli x7, x5, 4
orx7, x7, x6
```

0x1234567ababefef8

6. Consider a proposed new instruction named `rpt`. This instruction combines a loop's condition check and counter decrement into a single instruction.
For example, `rpt x29, loop`
would do the following:

```
if (x29 > 0) {
x29 = x29 -1;
goto loop
}
```

a. If this instruction were to be added to the RISC-V instruction set, what is the most appropriate instruction format? Assume either of 2 cases could occur: "`loop`" could be within a few hundred lines of code from the `goto` statement or "`loop`" could be several thousand lines from the `goto` statement

b. What is the shortest sequence of RISC-V instructions that performs the same operation?

The UJ instruction format would be most appropriate because it would allow the maximum number of bits possible for the "`loop`" parameter, thereby maximizing the utility of the instruction.

It can be done in three instructions:

```
loop:
    addi  x29, x29, -1   // Subtract 1 from x29
    bgt   x29, x0, loop  // Continue if x29 not
                            negative
    addi  x29, x29, 1    // Add back 1 that shouldn't
                            have been subtracted.
```