All exercises must be done by yourself. You may discuss questions and potential solutions with your classmates, but you may not look at their code or their solutions. If in doubt, ask the instructor.

Acknowledge all sources you found useful.

Partial credit is available, so attempt all exercises.

Submit your answers as a PDF file.

Exercise 1

Examine the processor information (/proc/cpuinfo) on a CS department machine and report back:

1. The machine name csug.cycle2
2. The CPU model name Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
3. The highest version of AVX supported 2, avx2
4. The register size for the AVX supported (see Intel documentation for this)

https://software.intel.com/en-us/articles/introduction-to-intel-

Exercise 2

Consider a program that operates in a pipeline fashion on multiple image files, performing three tasks in sequence on each image file. Each task takes roughly the same time:

Levels → Normalize → Sharpen

Assume there are many image files to be processed this way, and each image file can be processed independently of others. How would you map these tasks to processors to obtain a scalable version of the program?

Exercise 3

Although pipelines stalls and bypasses can delay with arbitrary delays and satisfying dependences/hazards, mis-speculation (i.e. an incorrect branch prediction) means that pipelines will also need to be "flushed" – all speculative instructions that have finished executed or are still being executed will need to be thrown away. Argue that deeper pipelines (i.e. those that have more stages) hurt performance on speculative processors.

guess: deeper pipeline -> more overload on each instructions -> more cost on speculative instructions ()

Exercise 4

AK p71. Consider the loop below.

```
for(i = 0; i < 7; i++) {
      a(i) = a(6-i) + 1
}
```

Answer the following questions:

1. How many distinct loop dependences can you find in the loopd? three true dependencies and three anti dependencies
2. How many distinct distance vectors does this reduce to? 6, 4, 2, -2, -4, -6 ===> 6
3. How many direction vectors does that reduce to? 3, <, =, >

Exercise 5

AK p67 (Adapted from AK) Consider the following code in C:

```
for(i = 0; i < 9; i++) {
      // increment all by 1
      for(j = 0; j < 10; j++) {
           a[j] = a[j] + 1
      }

      // swap two elements
      tmp = a[i];
      a[i] = a[i+1];
      a[i+1] = tmp;
}
```

i = 0, add(0~9), swap(0, 1)
i = 1, add(0~9), swap(1, 2)
i = 2, add(0~9), swap(2, 3)

Treat the inner loop as a single statement.

Since move swap statements to before the inner for-loop does not preserve the dependency

1. Why can't the swap statements be moved to before the inner `for`-loop in a dependence-based framework?

2. If the values of `a` are examined only after the *outer* `for`-loop, does moving the swap statements before the inner `for`-loop produce an equivalent program?

change the order of inner loop and swap does not change the dependency from outer loop

The program will be equivalent if the instruction of swap statements does not interleave with the inner for-loop

Exercise 6

Branch prediction predicts the result of a branch to enable speculation. In effect, it speculates on control dependences.

Currently, instructions must wait for their inputs to be available before then can begin execution. Outline a scheme for speculation on data dependences, and any assumptions you may need to make it work or to make it profitable.

END.