Full Name:

Andrew Id:

15-418/618 Spring 2021
Exercise 1

	Registered students	Waitlist students
Assigned:	Fri., Feb. 5	Fri., Feb. 5
Due:	Wed., Feb. 17, 11:00 pm	Wed., Feb.. 17, 11:00 pm

Overview

This exercise is designed to help you better understand the lecture material and be prepared for the style of questions you will get on the exams. The questions are designed to have simple answers. Any explanation you provide can be brief—at most 3 sentences. You should work on this on your own, since that's how things will be when you take an exam.

You will submit an electronic version of this assignment to Canvas as a PDF file. For those of you familiar with the LaTeX text formatter, you can download the template and configuration files at:

http://www.cs.cmu.edu/~418/exercises/ex1.zip

Instructions for how to use this template are included as comments in the file. Otherwise, you can use this PDF document as your starting point. You can either: 1) electronically modify the PDF, or 2) print it out, write your answers by hand, and scan it. In any case, we expect your solution to follow the formatting of this document.

Problem 1: Instruction-Level Parallelism

Consider the following code where each line within the function represents a single instruction.

```
typedef struct {
    float x;
    float y;
} point;

inline void innerProduct(point *a, point *b, float *result)
{
    float x1 = a->x; // Uses a load instruction
    float x2 = b->x;
    float product1 = x1*x2;
    float y1 = a->y;
    float y2 = b->y;
    float product2 = y1*y2;
    float inner = product1 + product2;
    *result = inner; // Uses a store instruction
}

void computeInnerProduct(point A[], point B[], float result[], int N)
{
    for (int i = 0; i < N; i++)
        innerProduct(&A[i], &B[i], &result[i]);
}
```

In the following questions, you can assume the following:

- N is very large ($> 10^6$).
- The machines described have modern CPUs, providing out-of-order execution, speculative execution, branch prediction, etc.
  - There are ample resources for fetching, decoding, and committing instructions. The only performance limitations are due to the number, capabilities, and latencies of the execution units.
  - The branch prediction is perfect.
- There are no cache misses.
- The overhead of updating the loop index `i` is negligible.
- The load/store units perform any necessary address arithmetic.
- The overhead due to procedure calls, as well as starting and ending loops, is negligible.

Suppose you have a machine M_1 with two load/store units that can each load or store a single value on each clock cycle, and one arithmetic unit that can perform one arithmetic operation (e.g., multiplication or addition) on each clock cycle.

A. Assume that the load/store and arithmetic units have latencies of one cycle. How many clock cycles would be required to execute `computeInnerProduct` as a function of N? Explain what limits the performance.

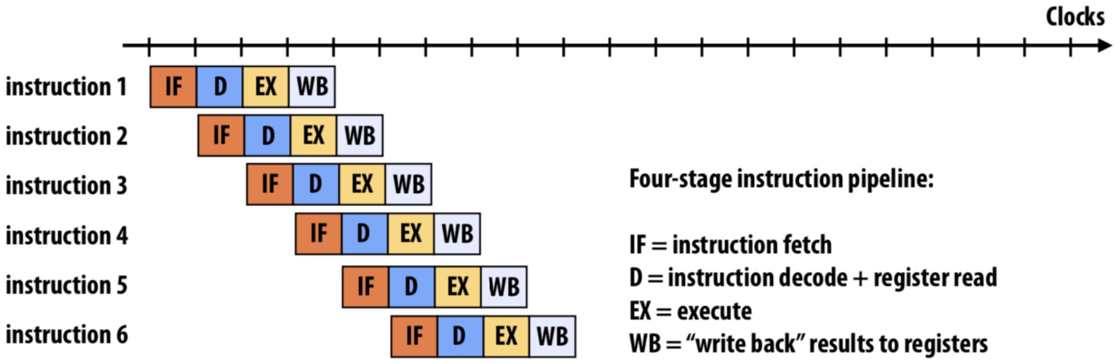
It takes $6N$ clock cycles since x1,x2 and y1,y2 loads together. Data dependencies limits the parallelism in this function.

B. Now assume that the load/store and arithmetic unit have latencies of 10 clock cycles, but they are fully pipelined, able to initiate new operations every clock cycle. How many clock cycles would be required to execute `computeInnerProduct` as a function of N? Explain how this relates to your answer to part A.

x1,x2 takes 10 cycle. Y1,y2 takes 10 cycle. Product2, inner result takes each 10 cycle. Total of 50 cycle. This time calculation of product1 is not visible to programmer.

Problem 2: A Yinzer Processor Pipeline

Yinzer Processors builds a single core, single threaded processor that executes instructions using a simple four-stage pipeline. As shown in the figure below, each unit performs its work for an instruction **in one clock**. To keep things simple, assume this is the case for all instructions in the program, including loads and stores (memory is infinitely fast).
The figure shows the execution of a program with six **independent instructions** on this processor. *However, if instruction B depends on the results of instruction A, instruction B will not begin the IF phase of execution until the clock after WB completes for A.*



A. Assuming all instructions in a program are **independent** (yes, a bit unrealistic) what is the instruction throughput of the processor?

N program takes $3 + N$ cycle.

B. Assuming all instructions in a program are **dependent** on the previous instruction, what is the instruction throughput of the processor?

N program takes $(4 + 1) * N$ cycle.

C. What is the latency of completing an instruction?

1 clock cycle because WB ␣IF

D. Imagine the EX stage is modified to improve its throughput to two instructions per clock. What is the new overall maximum instruction throughput of the processor for independent instructions?

It does not change because the critical path stays still.