In this section, we will discuss various aspects of performance. Anyone buying a computer will first ask: Which machine has the best performance? Which comparable machine has the lowest price? Which one has the best price/performance ratio? But there is much more to measuring perfomance than comparing MHz specs.

This section includes the following topics:

1. Latency vs. throughput
2. Speedup
3. Amdahl's Law
4. Performance metrics of single-cycle and pipelined datapath

## 1 Two notions of performance

Consider two planes: Boeing 747 and Concorde. Their characteristics are summarized in the table below. Which plane has better performance?

| **Plane** | DC to Paris | Speed | Passengers | Throughput[1] |
|---|---|---|---|---|
| `747` | 6.5 hours | 610 mph | 470 | 286,700 |
| `Concorde` | 3 hours | 1350 mph | 132 | 178,200 |

1. measured in *pmph*, passenger-miles per hour.

The perceived performance depends on what metric is selected. We will consider two metrics, time *and* number of tasks completed in a unit of time.

**How long does a task take?** The length of a task is known as *execution time*, *response time*, or *latency.*

**How many tasks are completed in a unit of time?** The number of tasks per unit of time is called *throughput* or *bandwidth.*

We will refer to them as *latency* and *throughput* respectively.

Consider a single-cycle machine with 2ns cycle time. Its latency is 2ns, because each instruction completes in 2ns. Its throughput is $1/2ns$, because one instruction completes every 2ns. But the relationship between latency and throughput isn't as clear-cut when a machine can perform multiple tasks concurrently.

Which performance metric is more important can also depend on a perspective. For example, in a client-server system, a client is concerned with the time required to accomplish a task (i.e. latency), but the performance of the server is measured by the number of requests it can serve in a unit of time (i.e. throughput).

**Throughput** is measured in units of things per unit of time, e.g. hamburgers per hour, passenger-miles per hour. *Bigger* throughput means better performance.

**Latency** is measured in units of time (seconds, ns). Performance is inversely proportional to the latency (or execution time) of a task.

$$Performance(x) = \frac{1}{execution\_time(x)} \tag{1}$$

So, *smaller* latency means better performance.

**How to compare performance of two machines?** To compare performance of two machines X and Y, run the same program (i.e. benchmark) on both of them. This means that you can assume that they have the same throughput and only the latency (execution time) needs to be compared. In such cases a simple formula suffices:

$$N = \frac{Performance(X)}{Performance(Y)} = \frac{execution\_time(Y)}{execution\_time(X)} \tag{2}$$

With this terminology we can easily compare the performance of two planes. Concorde has better latency and Boeing has better throughput.

## 2 Amdahl's Law

Measuring performance is only a means, not an end. One use of performance measurements is determining what part of the system to optimize to maximize performance increase. Amdahl's Law measures how effective an optimization can be.

Amdahl's Law states that optimizations are limited in their effectiveness. The performance enhancement possible with a given improvement is limited by the amount that the improved feature is used.

$$Execution\_time\_after\_improvement = \frac{Time\_affected\_by\_improvement}{Amount\_of\_improvement} + Time\_unaffected\_by\_improvement \quad (3)$$

For example, doubling the speed of floating point operations sounds like a good idea. But if only 10% of the program execution time $t$ involves floating point code, then the overall performance improvement is only 5%.

$$Execution\_time\_after\_improvement = \frac{0.10t}{2} + 0.90t = 0.95t \quad (4)$$

## 3 Practice problems

1. Recall the execution time equation:

$$CPU\_time_{X,P} = Instructions\_executed_P * CPI_{X,P} * Clock\_cycle\_time_X \quad (5)$$

Consider a basic machine with the following characteristics:

| **OP Type** | Freq ($f_i$) | Cycles | $CPI_i$ |
|---|---|---|---|
| `Load` | 20% | 5 | |
| `Store` | 10% | 3 | |
| `Branch` | 20% | 2 | |
| `ALU` | 50% | 3 | |

Calculate CPI for each instruction type.

How much faster would the machine be if:

(a) we added a cache to reduce average load time to 3 cycles?

(b) we added a branch predictor to reduce branch time by 1 cycle?

(c) we could do two ALU operations in parallel?

2. Use the basic machine table from question 1, but assume that the frequency column indicates the percentage of execution time spent in the corresponding instruction type. Use Amdahl's Law to answer the following: if you could decrease the cycle time of one of the instruction types by 1 cycle, which instruction type would you optimize? How would the new execution time compare to the original one?

3. Intel's Itanium (IA-64) ISA is designed to facilitate executing multiple instructions per cycle. If an Itanium processor achieves an average CPI of .3 (3 instructions per cycle), how much faster is it than a Pentium4 (which uses the x86 ISA) with an average CPI of 1?
   (a) Itanium is three times faster
   (b) Itanium is one third as fast
   (c) Not enough information

4. Assume the following delays for the main functional units of the single-cycle datapath shown below:

| **Functional Unit** | Time Delay |
|---|---|
| `Memory` | 5ns |
| `ALU` | 4ns |
| `Register File` | 3ns |

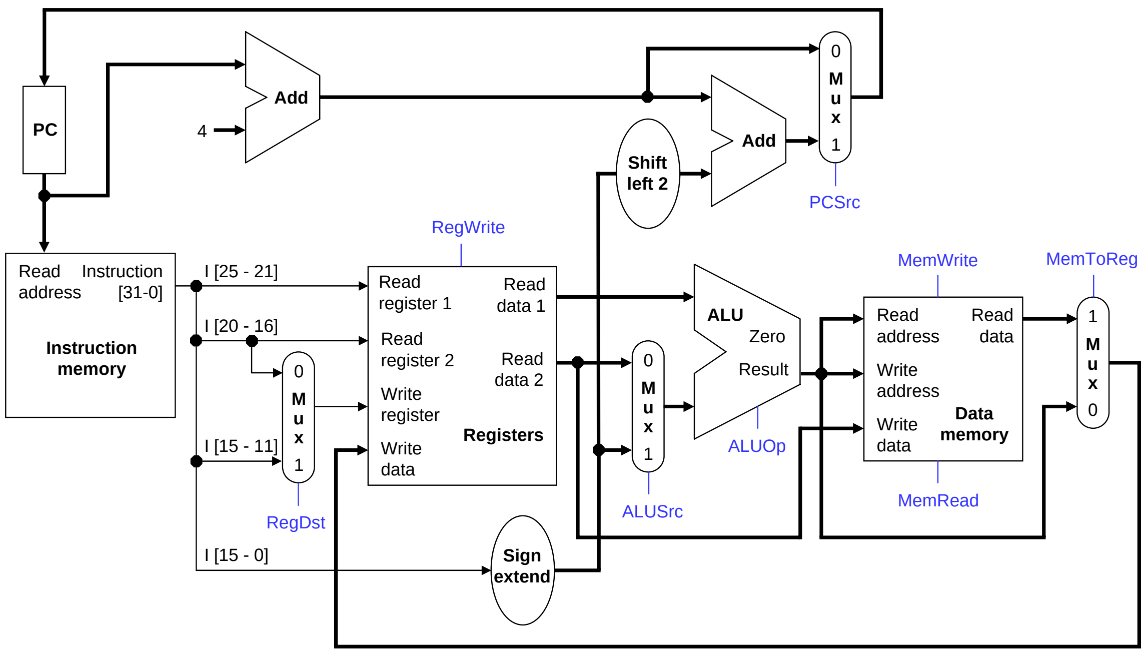
Given the following instructions: `lw`, `sw` and `add`, calculate:

(a) Minimum time to perform each instruction
(b) Time required on a single-cycle datapath

Write your answers in the table below. State any assumptions.

| **Instruction** | instruction time | instruction in Single-Cycle datapath |
|---|---|---|
| `lw` | | |
| `sw` | | |
| `add` | | |

Single-cycle datapath



5. Consider the following set of instructions:

```
add $sp, $sp, -4
sub $v0, $a0, $a1
lw  $t0, 4($sp)
or  $s0, $s1, $s2
lw  $t1, 8($sp)
```

   Assuming the instructions are executed on a **single-cycle machine** with 10ns cycle time, compute:

   (a) cycle time (hint: this is *not* a trick question)

   (b) instruction latency

   (c) instruction throughput

6. Assume that the code above is executed on a 5-stage **pipelined** machine discussed in lecture. You might first draw the pipeline diagram in the space below.

   If a pipeline stage takes 2ns, calculate:

   (a) cycle time

   (b) instruction latency

   (c) instruction throughput