

Great Ideas in Computing

University of Toronto CSC196
Fall 2025

Class 4: September 25 (2026)

Announcements and Agenda

Announcements

- Office Hours: Mondays and Wednesdays 10:30-11:30.
I will be holding the Friday class instead of Elise. It will be part tutorial and part new discussion.
- I have posted the final question for Assignment 1.
- Tentative schedule for guest presentations:
September 30: Professor Peter Love will discuss quantum computation
October 21: Professor Thomas Ristenpart will discuss computer security
November 6: Professor Ohad Shamir will discuss some theoretical aspects of neural networks.

Today's Agenda

- Going over floating point representation and an even more simplified representation.
- The memory hierarchy and memory management.
- A brief mention of data flow, multi-core and parallel computation.
- Data type and data structures

An example for the simplified binary representation

Repeating the even more simplified version as follows:

Consider a floating point number $s \# e_{k-1} e_{k-2} \dots e_0 \# b_{\ell-1} \dots b_1 b_0$

We let e_{k-1} be the sign of the exponent so that

$x = (-1)^s \cdot 2^{E'} \cdot (1 + \sum_{i=1}^{\ell} b_{i-1} 2^{-i})$ where now $E' = (-1)^{e_{k-1}} \cdot E$.

That is we are letting the leading bit of the exponent be the sign of the exponent rather than using a biased exponent.

Let's again consider an 8 bit number with a 4 bit exponent field and a 3 bit significand.

Now the floating point word 11010111 is equal to
 $-1 \cdot 2^{-2} \cdot (1 + \frac{7}{8}) = -\frac{1}{4} \cdot \frac{15}{8} = -\frac{15}{32}$.

And the floating point word 10101111 is equal to $-2^5 \cdot \frac{15}{8} = -60$ in decimal.

An even more simplified floating point representation

Let's define a more simplified representation as follows:

Consider again, a floating point number $s \# e_{k-1} e_{k-2} \dots e_0 \# b_{\ell-1} \dots b_1 b_0$

Again let e_{k-1} be the sign of the exponent.

Now, we will also not use the implied leading 1 in the fractional part.

Let's again consider an 8 bit number with a 4 bit exponent field and a 3 bit significand.

Now the floating point word 11010111 is equal (in decimal) to

$$-1 \cdot 2^{-2} \cdot \left(\frac{7}{8}\right) = -\frac{1}{4} \cdot \frac{7}{8} = -\frac{7}{32}.$$

And the same word 11010111 if it was declared or understood to be an integer would represent the decimal integer

$-(1 + 2 + 4 + 16 + 32 + 64) = -119$ assuming again that the leftmost bit is the sign bit.

NOTE: As mentioned before, the same binary word in memory can represent a floating point (i.e. possibly a fraction), an integer, or an instruction.

Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*. The von Neuman model assumes a random access memory in which each *fixed length word* is “addressable”. We conceptually think of each word being accessed (read or written) at some unit (of time) cost.

Is this how your lap top memory is organized?

Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*. The von Neuman model assumes a random access memory in which each *fixed length word* is “addressable”. We conceptually think of each word being accessed (read or written) at some unit (of time) cost.

Is this how your lap top memory is organized?

Take a look at the specifications of your lap top and you will almost surely see different types of memory. This is called the memory hierarchy. Let's just consider a two level hierarchy, where the fast memory might be called the cache and a slower memory which we can just call the “main memory”. (There can be more than two levels in the memory hierarchy including external memory which still might be addressable.)

Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*. The von Neuman model assumes a random access memory in which each *fixed length word* is “addressable”. We conceptually think of each word being accessed (read or written) at some unit (of time) cost.

Is this how your lap top memory is organized?

Take a look at the specifications of your lap top and you will almost surely see different types of memory. This is called the memory hierarchy. Let's just consider a two level hierarchy, where the fast memory might be called the cache and a slower memory which we can just call the “main memory”. (There can be more than two levels in the memory hierarchy including external memory which still might be addressable.)

The cache is a more costly, relatively small but a fast memory and main memory is considerably larger and slower in terms of accessing time. So, in fact, different accesses can take different amounts of time. However

The memory hierarchy and caching

An effective memory management system (part of the operating system) lets us conceptually ignore the different costs associated with accesses by exploiting “locality of reference” so that it is likely that the next access is in the cache. To be more effective when we need a word, we bring in a block of words (called a **page**) at the same time.

The following **Least Recently Used (LRU)** algorithm is considered a good method for caching. When the cache is full and a new page is needed, we delete the page in the cache that was least recently used.

The memory hierarchy and caching

An effective memory management system (part of the operating system) lets us conceptually ignore the different costs associated with accesses by exploiting “locality of reference” so that it is likely that the next access is in the cache. To be more effective when we need a word, we bring in a block of words (called a **page**) at the same time.

The following **Least Recently Used (LRU)** algorithm is considered a good method for caching. When the cache is full and a new page is needed, we delete the page in the cache that was least recently used.

Why not just have fast memory?

The memory hierarchy and caching

An effective memory management system (part of the operating system) lets us conceptually ignore the different costs associated with accesses by exploiting “locality of reference” so that it is likely that the next access is in the cache. To be more effective when we need a word, we bring in a block of words (called a **page**) at the same time.

The following **Least Recently Used (LRU)** algorithm is considered a good method for caching. When the cache is full and a new page is needed, we delete the page in the cache that was least recently used.

Why not just have fast memory?

In computing (and life) there are almost inevitably tradeoffs to be made.

The memory hierarchy and caching

An effective memory management system (part of the operating system) lets us conceptually ignore the different costs associated with accesses by exploiting “locality of reference” so that it is likely that the next access is in the cache. To be more effective when we need a word, we bring in a block of words (called a **page**) at the same time.

The following **Least Recently Used (LRU)** algorithm is considered a good method for caching. When the cache is full and a new page is needed, we delete the page in the cache that was least recently used.

Why not just have fast memory?

In computing (and life) there are almost inevitably tradeoffs to be made.

The important thing is that you (the algorithm designer) usually do not need to know how this memory hierarchy is managed; that is, when data is brought into the cache and when it is ejected from the cache. This is referred to as caching or sometimes as paging.

Memory management also makes it possible to have several processes sharing the main memory.

A little more about memory management

Another aspect of memory management is allocating memory to arrays and other data structures. For example, when we declare (in a programming language) that A is an array of some fixed size, for efficiency we want to allocate consecutive blocks of memory words. (This will lead us to discuss the idea of data types and data structures.)

A False Economy

An algorithm designer may think they know their application and can better optimize storage management for their application.

Why is this a false economy?

This is similar to the false economy of programming an algorithm in the machine code (of a specific machine).

I think virtual memory (and other forms of “virtualization”) is a great idea.

Machine instructions

Algorithms consist of individual instructions that say what "basic operations" to perform on data and also to indicate what instruction to do next.

Instructions can be represented by strings of symbols (indeed by strings of bits)! So instructions are also be stored in the memory, say for example one instruction per word!

It is usually considered good practice to keep the instructions of a program separate from the data that the program is accessing and not allow the program to change its instructions.

Machine instructions

Algorithms consist of individual instructions that say what "basic operations" to perform on data and also to indicate what instruction to do next.

Instructions can be represented by strings of symbols (indeed by strings of bits)! So instructions are also be stored in the memory, say for example one instruction per word!

It is usually considered good practice to keep the instructions of a program separate from the data that the program is accessing and not allow the program to change its instructions.

Why is the von Neumann model such a great idea?

Machine instructions

Algorithms consist of individual instructions that say what "basic operations" to perform on data and also to indicate what instruction to do next.

Instructions can be represented by strings of symbols (indeed by strings of bits)! So instructions are also be stored in the memory, say for example one instruction per word!

It is usually considered good practice to keep the instructions of a program separate from the data that the program is accessing and not allow the program to change its instructions.

Why is the von Neumann model such a great idea?

Are we stuck in a "von Neumann tarpit?"

The benefits of a well agreed upon abstract model of computation

One of the main reasons to consider the von Neumann model a great idea is that by being a well agreed upon model, coordination amongst different people is minimized. That is,

- A computer architect doesn't need to know which programming languages will be run on their specific architecture. The abstract von Neumann model doesn't specify the instruction set, the memory management, how interrupts are handled, etc. This is what the architects do.
- A compiler writer for a programming language $\mathcal{L}$ doesn't have to know what algorithms will be implemented using the language $\mathcal{L}$.
- Without complete knowledge of the architecture, and the compiler, the algorithm designer can make a rough approximation for the memory and time requirements of their algorithm.

Progress in parallel computation had been relatively slow but there is now some common approaches (e.g., MapReduce for large scale parallel computation).

Dataflow architecture

Direct from Wikipedia:

Dataflow architecture is a computer architecture that directly contrasts the traditional von Neumann architecture or control flow architecture. Dataflow architectures do not have a program counter (in concept): the executability and execution of instructions is solely determined based on the availability of input arguments to the instructions,[1] so that the order of instruction execution is unpredictable, i.e. behavior is *nondeterministic*. (My emphasis to indicate that event are not following some deterministic or well understood random distribution.)

Although no commercially successful general-purpose computer hardware has used a dataflow architecture, it has been successfully implemented in specialized hardware such as in digital signal processing, network routing, graphics processing, telemetry, and more recently in data warehousing. It is also very relevant in many software architectures today including database engine designs and parallel computing frameworks.

More on data flow architecture

From J. Paul Morrison's Flow-Based Programming text

The von Neumann machine is perfectly adapted to the kind of mathematical or algorithmic needs for which it was developed: tide tables, ballistics calculations, etc., but business applications are rather different in nature. . . .

Business programming works with data and concentrates on how this data is transformed, combined, and separated. . . . Broadly speaking, whereas the conventional approaches to programming (referred to as “control flow”) start with a process and view data as secondary, business applications are usually designed starting with data and viewing processes as secondary—processes are just the way data is created, manipulated, and destroyed. We often call this approach “data flow.” (21)

Multicore and Parallel Computation

- As you may already know, computers today are often multicore machines meaning that some “small” constant number of processes can be running simultaneously.
- When people refer to large scale parallelism they have in mind that the number of processes running in parallel can depend (at least conceptually) on the computation.
- The von Neumann architecture is an abstract model for *sequential computation*.
- In contrast to the well accepted von Neumann model for sequential computation, the situation for *parallel computation* is more nuanced.
- There is the issue of a constant number of parallel processes vs a number of processes that depends on the size of the data and/or the computation as it evolves.
- Do the processes run synchronously (i.e. according to some global clock) or asynchronously?
- Do the processes communicate mainly through a shared memory or via some communication bus?
- How do we maintain consistency of the information being shared?

Data Types

Let's give an informal definition of what we mean by a data type. Namely, a data type is a collection of items (data) and the allowable operations, relations and queries involving those items. So as an example we can have a data type called Float where the data is numbers represented in floating point representation, the operations are the standard arithmetic operations $+$, $-$, $*$, $\div$, exponentiation and perhaps logarithms. We also have the relations $<$, $=$, $>$.

You can check how Wikipedia states what is a data type.

We will next introduce the Dictionary data type.

Looking up a record

Suppose with every person in an organization we have information stored in some “device”. It could be an old printed telephone directory, a folder in a physical cabinet, or a file in a cell phone or computer.

Lets think about how it could be stored in a von Neuman architecture type computer in analogy to a file cabinet. Namely, think about one folder placed after the other. And for simplicity, say we have the same amount of information on each person.

In a computer one way to do this is to think of each person taking up some p consecutive words in memory (i.e., an array), one word for the name of the person, and $p - 1$ words for the information. The information about a person can be called a *record*. If there are n people in the organization then we would be taking up $n \cdot p$ words of memory if we stored this information in an array.

Instead of the name of the person we could have some other (unique) identifier (e.g., their social insurance number).

Dictionaries

What are the most basic operations we want to associate with such a collection of information?

Dictionaries

What are the most basic operations we want to associate with such a collection of information?

- *Search*: Look up if someone is in the organization and if so retrieve the information for this person.
- *Update content*: Change the information regarding an item
- *Insert*: Add a new person to the organization
- *Delete*: Remove a person from the organization.

Sets of objects with these operations are referred to as a *Dictionary* data type. In a static dictionary, we only want to look up and possibly modify records; in a dynamic dictionary, we also want to add and delete records. We can use different *data structures* to implement such a data type.

There can be many more operations that we want to perform on collections of data. More generally how one maintains and operates on data is known as the field of *data bases*. Analyzing data and extracting new (often statistical) information from collections of data is now called *data science* or *data analytics*. More ambitious learning of new information from data can be called *machine learning*.