Lecture 2 — Computer Basics; Intro to C++

J. Zarnett
jzarnett@uwaterloo.ca

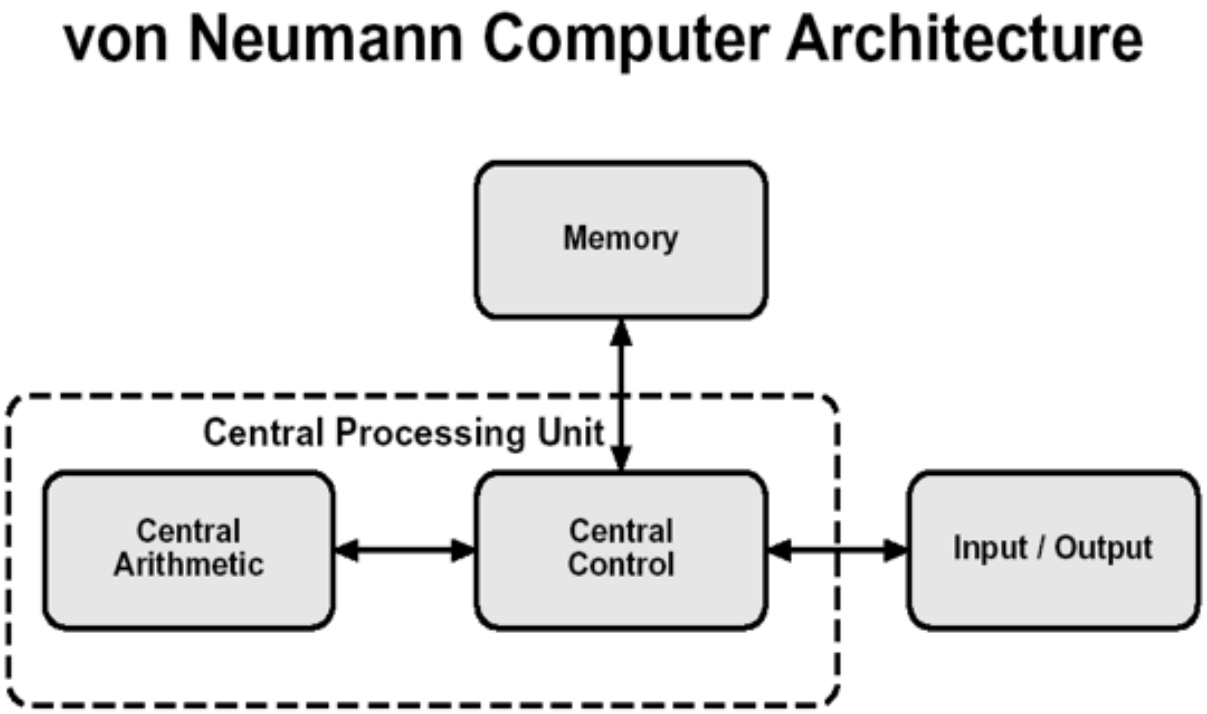
Department of Electrical and Computer Engineering
University of Waterloo

September 6, 2016

Acknowledgments: W.D. Bishop

Part I

Computer Basics



Source: J. von Neumann, First Draft of a Report on the EDVAC. Technical Report W-6700RD-492, Moore School of Electrical Engineering, University of Pennsylvania, June 1945.

Central Processing Unit

The CPU contains a central control unit and a central arithmetic unit.

CPU may also contain registers (high-speed storage locations) to store intermediate results and system state.

CPUs control computer systems using a sequence of instructions.
Instructions manipulate data and control peripheral devices.

Intel Pentium IV processor is an example of a CPU.

Central Control Unit

Central control unit performs two functions:

1. Fetches instructions from memory
2. Performs instruction sequencing

All information flow within the computer is directed by the central control unit.

Central Control Unit

Central arithmetic unit performs two functions:

1. Arithmetic operations (e.g., add, subtract, etc.)
2. Logic operations (e.g., bit-wise AND, bit-wise OR, etc.)

Data is manipulated by the central arithmetic unit.

The central arithmetic unit is also sometimes called the arithmetic and logic unit (ALU).

Memory systems use a linearly addressable array of cells to store binary information:

- Each cell has a unique address
- Each cell stores a single binary digit (bit)
- Cells are grouped to represent a larger range of possible values

Grouping	**Number of Bits**	**Number of Values Represented**
Nybble	4 bits	16 values
Byte	8 bits	256 values
Word	n bits	2^n values
Long Word	$2n$ bits	2^{2n} values

Memory systems can store both instructions and data.

Programs keep track of which memory cells contain instructions and which memory cells contain data.

Input / output (I/O) systems allow the exchange of information between a computer system and its environment.

System Type	Source	Destination	Examples
Input	Environment	Computer	Keyboard, Mouse
Output	Computer	Environment	Printer, Monitor
I/O	Either	Either	Network Card, Modem

Basic Execution Cycle of a Computer

Computers typically perform the following steps to execute a single instruction:

1. Fetch the next instruction to be performed
2. Decode the instruction
3. Fetch operands (if necessary)
4. Perform the operation
5. Store the result (if necessary)

Computers use a program counter to store the address of the next instruction to be performed.

This counter is automatically updated by the computer as instructions are performed.

Computer System

Computer systems are comprised of hardware, software, and firmware.

Hardware is a term for the physical components of a computer system.
Examples include the CPU, network cards, etc.

Software is a term for the programs, procedures, and documentation associated with a computer system.
Examples include MS-Word, MS-Excel, etc.

Firmware is a term for programs stored within a read-only hardware device within a computer system.
An example is the BIOS (Basic I/O System).

Computer Program

A computer program is an instruction sequence to perform a particular task.

Programs can be created in machine language but this development process is very tedious.

As an example, the following machine language program adds two numbers on an Intel Pentium processor:

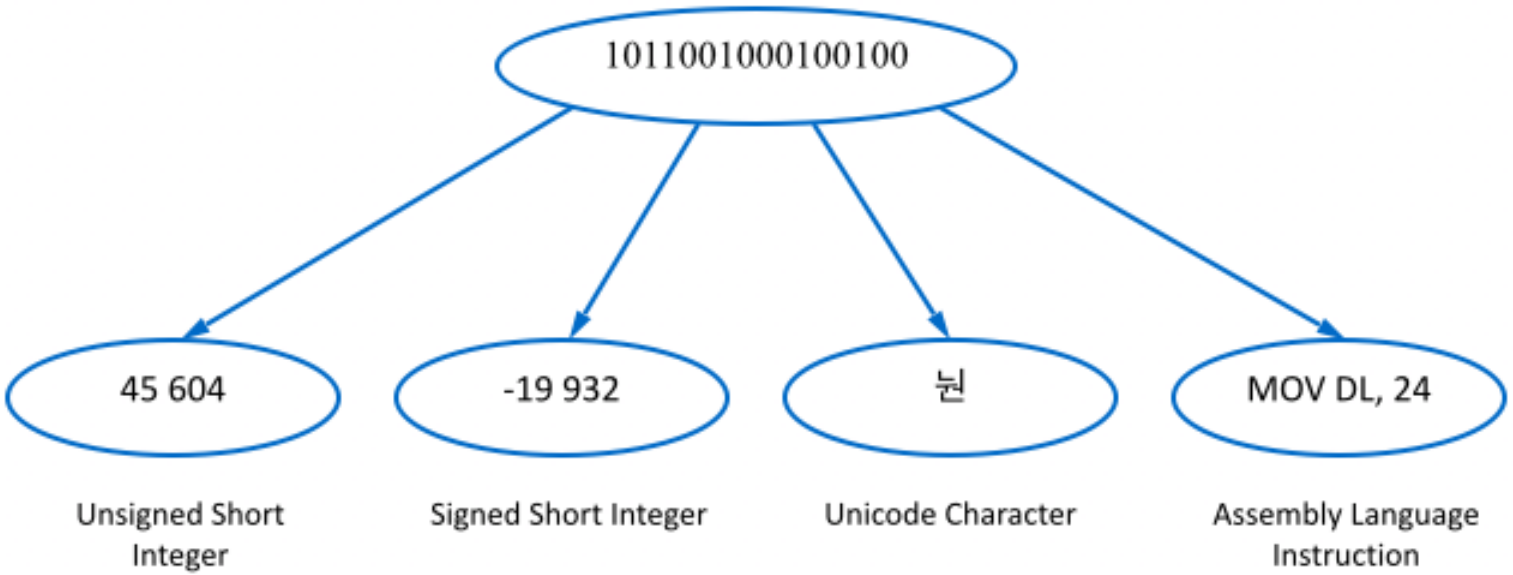
```
10100001 01101000 10111100 01000001 00000000
00000011 00000101 01101100 10111100 01000001 00000000
10100011 01110000 10111100 01000001 00000000
```

Software developers use assemblers, interpreters, and compilers to build computer programs.

Understanding Binary Quantities

A sequence of 0s and 1s can have many meanings

Consider the quantity `1011001000100100`:



Conclusion: Context is very important.

Assemblers

Assembler: a program to translate simple instructions named mnemonics from a human readable form to a machine readable form.

Assemblers translate assembly language into machine language.

Assembly language uses mnemonics to represent bit patterns.

Assembly language mnemonics (usually) have a 1:1 correspondence with machine language instructions.

Interpreter

An interpreter is a program to translate a high-level description of a program into a machine readable form at execution time.

One statement in a high-level programming language may correspond with many machine language instructions.

Interpreters may be capable of performing code optimization at execution time.

Compiler

A compiler is a program to translate a high-level description of a program into a machine readable form prior to execution.

One statement in a high-level programming language may correspond with many machine language instructions.

Compilers can perform code optimization at compile time.

Free compilers exist for all popular high-level programming languages including C, C++, Java, and C#.

Just-In-Time (JIT) Compilers

A just-in-time compiler is a special class of compiler.

Like interpreters, just-in-time compilers translate high-level descriptions of programs at execution time.

However, unlike interpreters, just-in-time compilers use a two stage compilation process.

Just-In-Time (JIT) Compilers

This allows code optimization at both compile and execution time.

Programs are first compiled into an intermediate language that is independent of the machine language of the computer.

At execution time, the intermediate language is translated to machine language by the JIT compiler.

High-Level Programming Languages

High-level programming languages make it easier to develop, test, maintain, and reuse software.

Two programming paradigms are popular:

- Structured Programming (SP): A program is designed using a top-down approach based on the use of instruction sequences called functions
- Object-Oriented Programming (OOP): A program is designed using a set of objects that encapsulate data and the functions performed on the data

Other programming paradigms exist but are rarely used.

Syntax

Syntax refers to the way in which language constructs are written to form a computer program.

Forgetting a semicolon at the end of a line is a syntax error.

Missing a closing quote on a string is an example of a common syntax error.

Syntax errors prevent compilation.

Context-sensitive text editors can help reduce the frequency of syntax errors prior to compilation.

Semantics

Semantics refers to the underlying meaning of a language construct.

Using the incorrect loop construct for a particular problem is an example of an error in semantics.

Errors in semantics do not prevent compilation but do result in a program behaving unexpectedly.

Errors in semantics can only be avoided by thoroughly understanding the programming language.

Types of Software

Application software solves a problem or provides a service in a particular application domain.

Examples: word processors, spreadsheets, and engineering software.

Types of Software

System software supports the development and use of application software.

Examples: operating systems, software development kits, and compilers.

There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.

- C. A. R. Hoare

The Software Design Process

Software design is an iterative process.

The steps are:

1. Define the problem
2. Develop an algorithm to solve the problem
   - Determine the sequence of steps required to solve a problem
   - Several algorithms may exist to solve a specific problem
   - Algorithm development may involve evaluating >1 option
3. Convert the algorithm to a specification in a suitable high-level programming language
4. Debug the computer program
5. Verify the operation of the computer program on a carefully chosen set of inputs

Software Documentation

Software documentation describes the behaviour of software using text and/or graphics.

Documentation is an essential part of software development.

Well-documented software is easier to understand and maintain.

Often, three software documentation approaches are used:

- Pseudocode in the early stages of algorithm design
- Unified Modeling Language (UML) in the latter stages of design
- Source code comments during the construction of designs

Part II

Elements of a Program

Elements of a Program

At a very high level, a program consists of just data and instructions.

In memory, they look the same: a sequence of 1s and 0s.

We build a program by writing instructions to use & manipulate data.

Thus, instructions are the building blocks of programming.

Elements of a Program: Instructions

Our programs will be built using the following elements:

- Variable declarations define storage for data
- Assignment statements assign a value to variable
- Loop statements perform repetitive tasks
- Selection statements choose between 2+ paths of execution
- Function declarations define a reusable set of statements
- Function invocations use a function to perform a task
- Input statements retrieve data from a computing device
- Output statements send data to a computing device

As we go through the course, we'll learn about each of these.

Elements of a Program: Instructions

Each instruction has a specific format.

The format is defined by the programming language standard.

Failure to follow the format may be a syntax or semantic error.

An instruction is built of one or more words or symbols.

Keywords, Identifiers, and Literals

Keywords are language-defined reserved words.
- Keywords denote language constructs.
- Keywords may be reserved always or reserved in certain contexts.

Identifiers are user-defined reserved words.
- Identifiers denote classes, variables, functions, and constants.
- Identifiers must start with a letter.
- Identifiers are case-sensitive.
- Keywords may not be used as identifiers.

Literals are constant values that can be assigned to variables.
- Literals are associated with a particular data type
- The data type of a literal is determined by:
  - its usage in the program and the way it is formatted.

Keywords

In any modern programming text editor, keywords will be highlighted.

This will help you spot them in the program, and also help you recognize that you cannot use one as an identifier name.

Don’t attempt to memorize the list of keywords. If you practice programming, you will become familiar with the important ones.

Program Structure

All programs consist of many lines of code.

For a very simple program, this will all be in one file.

Complex programs will use multiple files & folders of files...

User Interface

Programs almost always have a user interface: the part of the software that interacts with humans.

In this course, we'll work with Console Applications that interact with the user just via text on the screen and keyboard input.

Most of the pieces of software you are familiar with have Graphical User Interfaces (GUIs) with pictures (icons) and buttons.

GUIs may look really neat, but they are complex and focus on them would be a barrier to learning the programming fundamentals.

Part III

Introduction to C++

The C++ Programming Language

C++ ("See Plus Plus") is general purpose computing language.

It is based on the C programming language.

It is compiled, and can be imperative or object-oriented.

It is also a high level language:

One keyword may equal many machine instructions.

C++ Reserved Keywords

```
alignas          alignof          and              and_eq           asm
atomic_cancel    atomic_commit    atomic_noexcept  auto             bitand
bitor            bool             break            case             catch
char             char16_t         char32_t         class            compl
concept          const            constexpr        const_cast       continue
decltype         default          delete           do               double
dynamic_cast     else             enum             explicit         export
extern           false            float            for              friend
goto             if               inline           int              import
long             module           mutable          namespace        new
noexcept         not              not_eq           nullptr          operator
or               or_eq            private          protected        public
register         reinterpret_cast requires         return           short
signed           sizeof           static           static_assert    static_cast
struct           switch           synchronized     template         this
thread_local     throw            true             try              typedef
typeid           typename         union            unsigned         using
virtual          void             volatile         wchar_t          while
xor              xor_eq
```

Don't try to memorize all of these; as we go through the course we will use some, but not all, of them.

Source: `http://en.cppreference.com/w/cpp/keyword`

C++ Contextual Keywords

There are a few more keywords that are reserved in certain contexts.

You can use one as an identifier, but it's very bad practice:

```
override   final   transaction_safe   transaction_safe_dynamic
```

Source: `http://en.cppreference.com/w/cpp/keyword`

To create and run a C++ program, the flow looks something like this:

1. Write the program in an editor
2. Compile the program
3. Launch the program
4. The program executes

In the next lecture we'll create our very first C++ Program:
Hello World.