Programming Languages

Aniel Nieves-González

Institute of Statistics

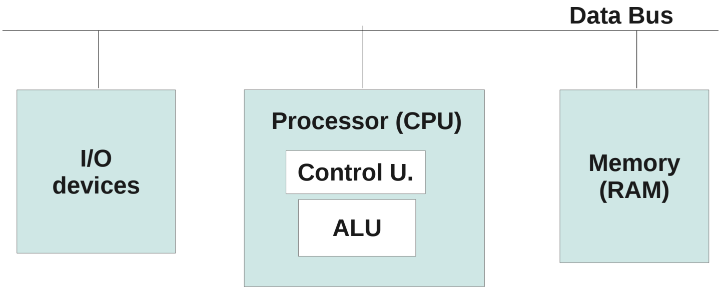
Spring 2014

"About 1,000 instructions is a resonable upper limit for the complexity of problems now envisioned."

—Herman Goldstine and John Von Neumann (1946)

Von Neumann Architecture

Modern computers with the exception of quantum computers and parallel computer systems (“supercomputers”) follow a computer architecture developed by Von Neumann in the 1940’s. The Von Neumann architecture can be illustrated as:



Von Neumann Architecture

The model proposed that not only the data, but also the instructions to be executed by the machine were stored in memory. Von Neumann invented programming as is known today.

Von Neumann Architecture

The Von Neumann architecture is a design that comprises:

- A *memory* unit that stores data and instructions to be executed. It consists of the random-access memory (RAM), which can be thought as an array of cells each with a unique address and with an access time that is uniform for all cells.
- *Input/Output* (I/O) unit: it consists of devices such as display, keyboard, mass data storage (e.g., hard drive), etc.
- The *processor* consists of registers (local storage) and two subunits:
  - Control unit (CU): it directs the operations of the processor, i.e., it directs the execution of the instructions to be executed (program).
  - Arithmetic logic unit (ALU): it performs arithmetic and logical operations.

Von Neumann Architecture: the instruction cycle

The processing unit executes the following cycle:

1. Fetch an instruction from memory. The instruction is loaded into a register from a memory address currently stored in a program counter. The program counter is updated to the address of the next instruction.
2. Decode the instruction. Here the CU determines what instruction is going to be executed. Note that each CPU or CPU family has a finite set of instructions, and such set is not unique (recall discussion on Turing Machines).
3. Read effective address. Any data needed to execute the instruction is loaded from memory.
4. Execute the instruction. The ALU might be involved in this part
5. Go to step 1.

Programming languages

- In computer systems, data and instructions are represented in numeric form.
- Recall that the computer system has a *finite* amount of resources (memory, etc). Thus, even storing information, can introduce errors.
  - Sound: sound waves are sampled (discretized) and amplitude and frequency information is stored (e.g. wav, mp3, etc.).
  - Image:
  - Characters are encoded in numeric form, e.g., ASCII, and UNICODE.
  - Integers: a finite numbers of integers can be represented. For example, signed and unsigned representation.
  - Real numbers: a finite numbers of real numbers can be represented using fixed point or floating point representation.

Programming languages

- In terms of hardware a computer system is a collection of electronic devices which are composed of electronic circuitry.
- The binary number system is used to represent data and instructions.
    - In this system all numbers are represented with $\{0, 1\}$.
    - The usage of the binary system brings reliability to the computer systems because in that case the circuitry that comprises the electronic devices only represents two states: on and off ($\{0, 1\}$).

Programming languages

A *a programming language* is a formal language design to communicate instructions to a computer. Programming languages are used to write programs, which express (implement) algorithms.

1. A *high-level language* provides strong abstraction from the details of a computing system. For example: FORTRAN, Cobol, C, Java, SQL, MATLAB, R, etc.
2. A *low-level language* provides little abstraction from the inner workings of a computer system. For example: Any assembly language.

Ultimately, any programming language is translated into machine language (machine instructions), which are in binary form. *Loosely* speaking that translation process is called compilation or interpretation.

Programming languages: brief history I

This is not a comprehensive list:

1. *Short Code* in 1949.
2. *Fortran* (mid 1950's). The name stands for formula translation. It was originally developed to tackle mathematical problems and is still used today in the scientific community.
3. *LISP* (1958). The name stands for LISt Processing. It is still used today in certain filds of computer science.
4. *COBOL* (1959). The name stands for COmmon Business-Oriented Language. It is still used in business, and administrative systems in companies, in particular in legacy applications.

Programming languages: brief history II

5. *ALGOL 60* (late 1950's). The name stands for ALGOrithmic Language. An effort to standarized programming languages.
6. *C* (1969-1973). Still used today to program applications, but it was originally intended to be a system programming languages, i.e., to write operating system software.
7. Many of the ideas observed in modern programming languages originated in the 1970's.
8. *C++* (1980). C with classes.
9. *MATLAB* (1984). The name stands for MAtrix LABoratory. It is a language and computing environment intended for numerical computing (in contrast of Maple which is focus on symbolic computing). The source of the MATLAB environment is written in C.

Programming languages: brief history III

10. *Python* (1991). It is a language design to improve code readability. In contrast to MATLAB or R and similar to the others it is a general purpose programming language.
11. *Visual Basic* (1991). Derived from BASIC by Microsoft. It was intended to be a language easy to learn and use.
12. *R* (1993). Based on language S. It is a language and computing environment intended for statistical computing and graphics. The source of the R environment is written primarily in C and Fortran.
13. *Java* (1995). It is a general purpose programming language, developed, as many languages of the 1990's, having Internet applications in mind. The sintax is similar to C, but the programs written in it are not *compiled*, but *interpreted*.

References I

Donald D. Knuth.
The art of computer programming. Volume 1: Fundamental Algorithms.
Third edition. Addison-Wesley, 1997.

G. Michael Schneider and Judith Gersting.
An Invitation to Computer Science: C++ Version.
4th Edition. Thomson: Course Technology, 2006

James L. Hein
Dicrete Structures, Logic, and Computability
Second edition. 2002.