Appendix A Computer Generations

This appendix provides additional background on the evolution of computer hardware and software, to supplement the discussion in section 1.3. Links to several further online resources on this topic are provided at the end of this appendix.

In terms of computer **hardware**, there are at least four generations of digital computer systems. *First generation* computers were introduced shortly after World War II. Based on *vacuum tube* technology, these were large, required lots of power, generated plenty of heat, and needed constant maintenance.

Second generation computers were introduced in 1959, with the release of the IBM 7090. These used *transistors* instead of vacuum tubes, resulting in a dramatic decrease in cost, size, and power consumption as well as an increase in reliability.

Third generation computers, introduced in 1964 with the announcement of the IBM 360, were the first to use *integrated circuits* (ICs). An IC consists of a large number of electronic circuits etched onto a single semiconductor chip. This use of microchips led to more dramatic drops in cost, size, and power consumption, and to greater reliability.

By 1971, the first general purpose processor-on-a-chip or *microprocessor* had been released: this was a major breakthrough in chip architecture. Second, *large scale integration* (LSI) had been achieved, delivering a major increase in chip density (number of circuit elements contained on the one chip). Although LSI led to size reductions, its primary purpose was to speed up processing by reducing the distances that electrons had to travel. These advances introduced the *fourth generation* of computers. In 1975, very large scale integration (VLSI) was achieved, packing over 100,000 transistors onto a single chip.

Although some modern CPUs operate at frequencies in excess of 3 GHz, heat generation problems indicate that much higher frequencies may be unattainable. Instead, the main contributor to future performance boosts is likely to come from *multicore* technology, packing many processors onto a single chip. In February 2008, Intel announced its Itanium quad-core CPU (code named "Tukwila") with a record-busting two billion transistors. The year 2008 also saw the introduction of 8-core chips, and chips with much higher core counts are expected in the near future.

As regards computer **software**, there have been five generations of *computer languages*. Computers were first programmed in *machine language*. Each machine

language instruction consists of a sequence of binary digits (bits). For example, with an 8086 processor, the machine code instruction to add 7 to the AX register is: 00000101 00000111 00000000. The byte 00000101 indicates that the following 16 bit data word should be added to the AX register; you may recognize 00000111 as binary for the decimal number 7. The final 00000000 is the "high order byte" of the 16 bit representation of 7.

You can imagine how boring and error-prone it was to program in strings of 1s and 0s all the time. So humans invented a mnemonic code called *assembly language*. In an assembly language for the 8086 chip, the previous machine instruction may be written as: "ADDI AX, 7". The ADDI, or "add immediate", indicates that the number to be added is contained immediately within the instruction; so the 7 is interpreted as data instead of as an address holding the data. The AX part indicates the target variable for the addition.

Although humans can (with some difficulty!) read assembly code, the computer cannot understand anything but pure machine language. A program called an assembler translates assembly language into machine language. Although modern assembly languages are better than older ones, they are still rather painful to use. Besides being hard to understand, assembly language programs tend to be very long since each instruction corresponds to just one machine language instruction. In this sense, assembly language is very low level.

Most programmers nowadays work almost exclusively with high-level languages, such as C#, COBOL and SQL. These are easier to write in and understand because they are closer to English. Moreover, a high level program is typically much shorter than an assembly program because one high level instruction usually does the job of several machine language instructions. For example, the following Pascal instruction tells the computer to write out the square root of the value of x: "write (sqrt(x))". The equivalent machine code is quite lengthy. As with assembly language, high-level language programs must be translated into machine language before they can be executed on a computer.

Some high level languages include optimizers to generate more efficient machine code, and some include facilities that come close to matching the power of assembly languages. Since high level code can be written and maintained much more easily than assembly code, the use of assembly languages has almost died out, being relegated to rare tasks where extra hardware control or greater speed is needed (e.g. fast animation).

With respect to computer languages, machine language is *first generation*, assembly language is *second generation* and high level languages are at least *third generation*. Notice that it is wrong to define an *n*th generation language as one that runs on an *n*th generation computer. Even the later first generation computers could run the first three generations of computer languages (the first high level language, FORTRAN, was released in 1957).

There are hundreds of high-level programming languages. Most of these are *procedural*, emphasizing the algorithmic side of programming (the procedures showing how to carry out the task) and are generally classified as third generation. This is true even of later procedural languages such as Modula–3 and C#.

When applied to languages, the term *fourth generation* typically refers to high-level database languages used for querying databases or building associated user interfaces such as screen forms. Fourth generation languages are primarily *declarative* in nature rather than procedural. That is, the programmer essentially declares *what* has to be done rather

than *how* to do it. Fourth generation languages (4GLs) are also highly interactive, supporting an ongoing dialogue between the human and the system.

Note that 4GLs represent a quantum leap beyond 3GLs for work with large, complex databases. Suppose we wish to extract some particular information from a database, and no program is on hand for our particular query. In a 3GL like C# or Java we would typically have to write pages of code, then compile and debug this until finally it could be run to yield the required results. In contrast, such an ad hoc query could typically be formulated easily in a single statement in SQL and the answer obtained immediately.

It has been estimated that approximately 80 percent of computer software applications fit into the information systems basket, with the emphasis on the data rather than the algorithm. Given the higher productivity of 4GLs in this area, many programming tasks previously performed in languages like COBOL are now handled by languages like SQL. However there is still a place for 3GLs like Java or C#. Sometimes an application is best coded partly in a 4GL and partly in a 3GL. Some programming problems cannot be handled efficiently by 4GLs. Examples of this include computer aided learning, theorem provers, advanced mathematical work, hardware control, and compiler construction. For most database applications however, the data-centered, set-oriented approach of languages like SQL is more appropriate.

The first four generations of computers are based on the von Neumann architecture, which includes a CPU (central processing unit), main and secondary memory, and input and output devices. The CPU is the heart of the computer: it controls the actions performed by the system and computes the values required for arithmetic and logical operations. The main memory stores information that can be "immediately" accessed by the CPU, including the program (or program segment) currently being executed.

The secondary memory stores (usually on disk or tape) information not currently being processed. An input device (e.g. keyboard) enables data to be sent from the outside world to the CPU, and an output device (e.g. monitor or printer) enables results to be communicated to the outside world.

The versatility of this model derives from the fact that the user can input the program to be processed by the CPU (this "stored program" concept is usually attributed, perhaps mistakenly, to John von Neumann) as well as the data to be operated on. This model is based on the notion of serial processing with at most one instruction being executed at any time.

Today's "*fifth generation*" systems typically incorporate large scale *parallel processing*, (many instructions being processed at once), different memory organizations, and new hardware operations specifically designed for symbol manipulation (not just number-crunching). Instead of one central processor, there may be thousands of processors working simultaneously on different aspects of the problem being processed. This new hardware organization is often coupled with software that focuses on *knowledge representation*.

Although programmers working with such "fifth generation" machines use high level languages, such as Prolog 2, it is more appropriate to reserve the term *fifth generation language* (5GL) for the language in which most users will communicate with such systems. What is this language? The human-machine interfaces are being designed to allow significant use of *natural language* and images. In this sense, the fifth will be the last of the language generations (at least from the human verbal interface point of view).

There a re many online resources on the history of computing (electronic, mechanical, or manual). Here are just a few:

http://en.wikipedia.org/wiki/History_of_computing_hardware
http://en.wikipedia.org/wiki/Computer
http://en.wikipedia.org/wiki/Timeline_of_computing.
http://en.wikipedia.org/wiki/History_of_computing
http://trillian.randomstuff.org.uk/~stephen//history/timelines.html
http://vmoc.museophile.com/

Exercise A1

1. For each of these pairs, which item best describes 4th generation database languages.

A.	low-level	B.	high-level
C.	declarative	D.	procedural
E.	algorithm-centered	F.	data-centred
G.	set-oriented	H.	record-oriented
I.	associate by name	J.	must specify access paths

2. "An *n*th generation computing language is one that runs on *n*th generation computer hardware". Is this definition correct?

3. As a computer trivia question, the IBM 7090 was originally named the 709T, where the "T" denoted the transistorized version of the old 709 model. Why was the name changed?