A Very Brief, Incomplete History of Computing

Foundations of Computer Science

E.H. Turner and R.M. Turner

2017-05-24.1

[**Note:** This is a somewhat incomplete chapter at this time; sorry about that. There is a plethora of information on the Web, however, if you don't see what interests you here.]

In the beginning...

No introductory textbook would be complete without some history of the field it introduces. Here, we give a brief, incomplete, and very informal history of the field. There are better sources for more complete histories; we will provide links to them in the "Further Reading" section of this chapter.

In one sense, since computing at its root is problem solving, the history of computing is very old—in some sense, as old the time when the first person told another the first informal algorithm for accomplishing some task, perhaps tanning hide or lighting a fire. The first computing *devices* were as simple as paper and pencil (or papyrus and brush, as the case may be) and, later, the abacus.

Aside from that, however, we can trace the true beginnings of the field to mid-1800s, when Charles Babbage and Ada Lovelace were working on the *difference* and *analytical engines*. In a very real sense, Ada Lovelace was the first computer programmer; she was interested in how one would specify algorithms for the analytical engine to solve problems. In fact, she was even interested in how chess might be played. She was so influential that a programming language, Ada, is named after her.

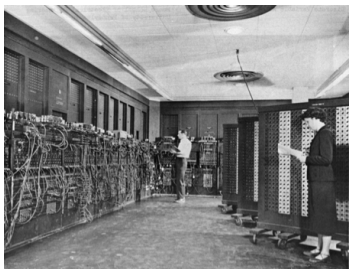

Figure 1: Programming ENIAC [US Army photograph]

Electronic digital computers

The field's formal and practical foundations began to be laid in earnest in the 1940s with seminal work by Claude Shannon, Alan Turing, John von Neumann, and others. The basics of the kinds of computers that have reigned since (*digital stored program computers*, sometimes called *von Neumann machines*) were developed during this time, and fundamental properties of information (Shannon) and computing (Turing) were explored. Alan Turing proposed a formal, abstract machine, now referred to as a *Turing machine*, that is equivalent to a computer and that can, it is thought, compute anything that is computable. (Not everything is, it turns out.)

Beginning in the 1940s, but especially in the 1950s and 1960s, practical computing machines were developed; the term "computer" came to mean these machines, whereas in the past it referred to a person who did computations. In 1946, the first general purpose computer was built, called the Electronic Numerical Integrator and Computer (ENIAC), shown in Figure 1. It was designed by John Mauchly and J. Presper Eckert. Unlike the computers that most of us have in our pockets—smartphones—ENIAC was 8 by 3 by 100 feet in size and weighed about 27 tons.

As can be seen in the photograph, programming ENIAC was not like programming computers today. It was not a stored program computer. Programs would be entered via a combination of cables to hook functional units of the computer together and "portable function tables" into which tables of numbers could be entered. Most of the programming was done by six women, who were later inducted into the Women in Technology International Hall of Fame.

Like some of its successors, ENIAC was one of a kind. Even though it was general-purpose, it was still built for a particular application (military calculations), and it was very expensive. It was followed shortly by true stored-program computers, but it wasn't until the 1950s that computers became more like products for sale, albeit still at a steep price.

One of the most successful computer companies was IBM (International Business Machines). IBM had been a long-time player in the business machines field, and in the early 1950s, entered the nascent computer business with their 700-series computers (see Fig. fig:intro:704). The first of the series, the IBM 701, was for scientific use and was introduced in 1952. Their (slightly) later computer, the IBM 702, was for business purposes. There were two, since it was thought that the needs of science and business were sufficiently different to warrant it. Scientific computing needed numerical operations, typically floating point (approximations to real numbers), while business needed to process text and to deal with decimal numbers (dollars and cents, e.g.; and decimal, since roundoff and other errors from floating point calculations could not be tolerated).

Figure 2: IBM 704 electronic data processing machine. [NASA photograph]

Stored programs

Up until this time, computers had been programmed mostly either by rewiring them (e.g., ENIAC) or, more commonly by this time, by entering binary machine language programs into the computer, either directly, with switches, or in some other manner. Programs written in binary are just strings of 1s and 0s. Consequently, they are very difficult for humans to read, and they are prone to errors when being

written or toggled into a computer via switches. Although a computer can only understand 1s and 0s, a more human-friendly method of representing programs was developed, called *assembly language* (see Chapter). Each instruction in an assembly language (which is usually specific for a given machine) corresponds to an instruction in the computers binary *machine language*, but uses mnemonics rather than binary digits. For example, whereas the machine instruction for adding two numbers together, each contained in a *register* (a storage location), might look something like:

```
0001 0000 0000 0110
```

the equivalent assembly language instruction might look like:

```
ADD R1,R2
```

Since a computer cannot directly understand assembly language, a special program, called an *assembler* must first translate the assembly language program (called the *source code*) into a machine language program (called the *object code*). This, after some additional processing to edit in links to library programs and so forth, then becomes an *executable file*, or just *executable*, that can be loaded into the computer and run.

Assembly language greatly increases the ease of programming, but programs written in assembly language are still very low-level and tailored toward the machine (i.e., the computer) rather than to a person's way of thinking about a problem. In the late 1950s, however, researchers began to develop *high-level languages*, languages much closer to human languages. In a high-level language (HLL) program, each line might represent several to several dozen assembly language lines. For example, a HLL statement such as:

```
Y = 2*X**2 + 3*X - 7
```

would set a *variable* `Y` to the value $2x^2 + 3x - 7$, where `X` is another variable. The equivalent assembly-language program would do this with statements various multiplication, addition, subtraction, and storage statements, e.g.,

```
LD R1,X    ;put X in register 1
LD R2,X    ;put X in register 2
MUL R1,R2  ;R1 <- R1 * R2
MUL R1,#2  ;multiply it by 2
MUL R2,#3  ;multiply X (in R2) by 3
ADD R1,R2  ;add the first two terms
SUB R1,#7  ;subtract 7
ST R1,Y    ;store result
```

The first HLL that saw any sort of widespread use was a scientific language, FORTRAN. The name FORTRAN stands for for FORmula TRANslation. As the name indicates, the focus was on quickly performing calculations. This language—which is still in use, by the way!—revolutionized the way people interacted with computers.

A bit later, COBOL, a high-level language for business-oriented computers, was invented by Grace Hopper and others. The name COBOL stands for COmmon Business-Oriented Language. COBOL is more concerned with moving and manipulating data than in numeric calculations, although it can do that too, of course, just as FORTRAN can manipulate text. It is an extremely verbose and English-like language, meant, in part, for non-specialists to be able to use. COBOL, too, is still around and in used; by some estimates, it is the language with the most lines of code written in it of all the programming languages in the world.

In between these two was a third language, developed not for numerical or text calculations, but for artificial intelligence: LISP (LISt Processing language), developed by John McCarthy. LISP (or Lisp, as we will write it in this book) was radically different from these other two "imperative" languages. Whereas a program in an imperative language is like a recipe, telling the computer what to do, a program in a *functional language* like Lisp instead applies functions to data and other functions. Lisp's data reflects its original purpose as an AI language: it includes symbols and lists which can simplify dealing with symbolic information (e.g., a sentence in English). Lisp, too, is still in widespread use.

Just a little later, a very influential language, ALGOL, was developed. This language, which influenced virtually all imperative and a good many object-oriented languages following it, was itself not used much. However, from it came many innovations that are still in use in a wide variety of modern languages.

Figure 3: IBM System/360 (By Jordiferrer (Own work) [CC BY-SA 3.0])

The 60s: Mainframes

About the start of the 1960s, IBM created a family of computers with the aim of unifying the two types that had come before, scientific and business-oriented. This was the IBM 360 family of machines, probably the most successful series of general-purpose computers of that era (Figure 3). The instruction set—the set of operations a computer can carry out—for the 360 family included both scientific and business-oriented operations. In addition, an innovation was that a customer could buy a computer that fit his or her needs, since there was a family of machines of varying abilities. A program developed for one computer in the family would run on any of the others, and

so a customer could upgrade his or her computer without having to rewrite code.

New versions of FORTRAN and COBOL continued to be in widespread use in this era of computers, but other languages were emerging, as well. Other special-purpose languages were also developed, for example, Simula (simulation), SNOBOL (string manipulation), APL (mathematics), and RPG (business). Simultaneously, general-purpose languages were arising, such as ALGOL, which laid the groundwork for many modern programming languages, and PL/I, which aimed to combine the best features of FORTRAN and COBOL (among others). BASIC, a language created for ease of use and heavily used in education (and the forerunner of many other versions, including Visual Basic), was created during this era as well.

Figure 4: IBM System/370-145, ca. 1974 (from Wikimedia Commons, user Oliver.obi, Creative Commons BY-SA 3.0 license)

In 1970, the next major family of IBM mainframes was introduced: the IBM System/370 family (Figure 4). This brought innovations such as virtual memory (see Chapter) and provided an upgrade path for users that was backward-compatible with the System/360 family and that would would continue for almost two decades.

Other mainframes were on the stage as well, including Digital Equipment Corporation's (DEC's) DECsystem-10 (also referred to as PDP-10 or DEC-10), and Control Data Corporation's CDC 6600 and Cyber computers. The DEC-10 was very common in academic computing and which was often used for time-sharing—that is, letting multiple interactive users share the machine at the same time. It was also very important in the construction of ARPANET, the forerunner of the Internet.

There was a wealth of programming languages in the 1970s. FORTRAN and COBOL continued to be major players, as did PL/I, BASIC, and Lisp (in artificial intelligence). Pascal, Niklaus Wirth's language based on ALGOL, began to be widely-used to teach programming in the mid-1970s. Dennis Ritchie created the C programming language about this time, as well, at Bell Laboratories. Not only is C still an extraordinarily important programming language, it was and is the language in which Unix and Unix-like operating systems (e.g., Linux) are written. It is also the basis of object-oriented languages such as Objective C, C++, and C#, and its syntax has influenced many other languages, including Java and JavaScript.

The number and kind of operating systems also diversified about this time. Operating systems, which we will cover in more detail in Chapters –, are programs that are responsible for managing the computer's resources, ensuring that users' programs run, providing services for those programs, and so forth. IBM continued its OS/360 operating system in this era as OS/MVS and VM/CMS, which was one of the first virtual machine operating systems in widespread use.

(A *virtual machine* OS is one that lets users have their own virtual copy of the hardware, on which they can run other operating system.) DEC had their own operating systems for including TOPS-10 and TOPS-20 (Timesharing/Total Operating System) for their mainframe computers, the DEC-10 and DEC-20.

Mid-60s–70s: Minicomputers

The *minicomputer* era began in the mid-1960s, ushering in cheaper computers that, while not inexpensive enough for home use, made dedicated laboratory and computer science department computers feasible. The most successful of these early machines was DEC's *PDP-8* minicomputer. It was about the size of a large storage cabinet or refrigerator, which was small for computers at that time. There were more PDP-8 computers sold than any other computer up to that time.. They were considered inexpensive compared to mainframes, costing $18,500 initially (equivalent to more than $140,000 at the time of writing).

Originally, there was no operating system for the PDP-8. Instead, an initial program could be entered using the toggle switches on its front panel, one 12-bit word at a time, in binary. This "bootstrap program" would be sufficient to allow other programs to be read into the computer via a keyboard or paper tape.[1] Later, simple operating systems were developed for PDP-8 models.

DEC followed their success with the PDP-8 with the *PDP-11* minicomputer (Figure fi:history:pdp-11), which was possibly the best-selling minicomputer ever. It ran a variety of operating systems, including DEC's version of Unix, called *Ultrix*. The PDP-11 was followed up by DEC's VAX-11 (Virtual Access Extension) minicomputer in the mid-1970s. The VAX minicomputers ran DEC's own VMS operating system, but versions of Unix were ported to it as well.

The 70s–80s: Workstations

Another term for high-end computers meant primarily for single-person use is *workstation*. Early workstations were generally powerful minicomputers for scientific uses, such as Sun Microsystems Sun and SPARCstation workstations (see Figure 7). This term continues to be used for powerful microprocessor-based computers.

Some early workstations were special-purpose machines. For example, Evans & Sutherland's PS 300 (see Figure 8) was one of the first computer graphics workstations, and Lisp machines were produced by several companies (e.g., Texas Instruments, Symbolics, and Xerox) for artificial intelligence and other research (see Figure 9).

Figure 5: A PDP-8: serial #85, delivered to the University of Iowa in 1966. [Douglas W. Jones, via Wikimedia Commons]

[1] The initial program for a computer is called the "bootstrap program" because it allows the computer to "pull itself up by its bootstraps", as the old saying goes. This is the source of the terms "boot program", "booting the computer", etc.

Figure 6: A PDP-11/70 minicomputer. [Joe Mabel, via Wikimedia Commons, CC-BY-SA 3.0]

The 70s–present: Microcomputers

The 1970s also saw the first *microcomputers*. These are small computers whose *central processing unit* (*CPU*) is a single chip, the *microprocessor*, as opposed to the multiple components used in minicomputer and mainframe CPUs. The first microprocessors were not very powerful; most were 8 bits, and some were only 4 bits, and they only operated at a rate of a few hundred thousand operations per second. (Compare that to modern microprocessors, which can operate at the rate of *billions* of instructions per second.) However, they were cheap enough to be popular with hobbyists.

Microcomputers are not just a historical curiosity. Today's personal computers are microcomputers, though the term has fallen out of favor. The Apple II line of computers began in the late 1970s microcomputer era and were some of the earliest "home computers" as well as some of the first microcomputers to see widespread use in business and (especially) education. Other home computers, such as the PET and the TRS-80 from Radio Shack, were also in widespread use. In 1981, IBM introduced their take on the microcomputer/home computer, called the Personal Computer (PC). The PC inspired many "clones", resulting in widespread availability of PCs (in the generic sense) and a dominance of the microcomputer market for Microsoft's operating systems. The rest, as they say, is history.

Operating systems on microcomputers varied widely, including CP/M, MS-DOS (Microsoft's disk-operating system for PCs, loosely derived from Unix), and Apple DOS. Later PCs were able to run versions of Unix (including Apple's A/UX, IBM's AIX, and, eventually, Linux) as well as the two major divisions of PC (in the modern sense) operating system, Microsoft Windows and Apple MacOS.

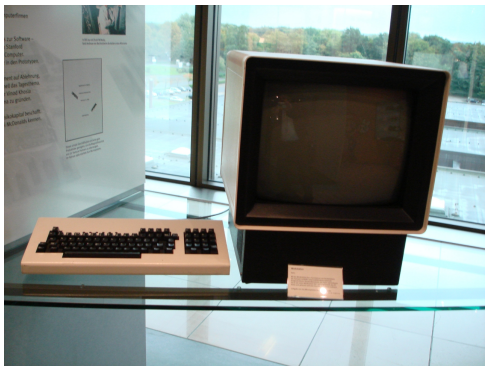

Figure 7: A Sun-100 workstation. [Peter Dieth, via Wikimedia Commons, CC-BY-SA 3.0]

Figure 8: An Evans & Sutherland PS 300 graphics workstation. [From a tweet by @ES$_{\text{Digistar}}$ (Evans & Sutherland Twitter account), December 31, 2015]

Resurgence of mainframes

The need for mainframes as shared programming platforms decreased over time as workstations and personal computers gained widespread adoption, with some of the smaller computers rivaling (or even exceeding) the raw processing power of mainframes. However, mainframes excel in handling large amounts of data, and so there has always been a demand for them, especially in business and other applications needing fast access to large databases. Mainframes align well with the recent burgeoning interest in data mining and machine learning. Some offer specialized hardware and optimizations for various purposes. For example, the newest (at the time of writing) IBM mainframe, the IBM z14 (Figure 10) has: hardware-assisted encryption; special instructions to help with decimal-based calculations

Figure 9: Symbolics 3640 Lisp machine. [Michael L. Umbricht and Carl R. Friend (Retro-Computing Society of RI) via Wikimedia Commons, CC-BY$_{\text{SA}}$ 3.0]

(e.g., for currency-related transactions in COBOL or PL/I); virtualization performance enhancements for running Linux virtual machines; garbage collection hardware enhancements to support Java programs; a compression co-processor for database indices and data; dedicated I/O processors (traditional in mainframes); and the ability to have up to 32 terabytes (TB) of main memory.

Figure 10: IBM z14 mainframe (from www.ibm.com/us-en/marketplace/z14)

Servers

The idea of a *server* has a long history in computer science. Servers are computers that do just that: provide services to other computers or users. A file server, for example, stores files for use by a group of computers and users, a compute server provides the service of running programs or parts of programs for other computers, Web servers provide Web pages, and so forth. Special-purpose servers, especially with the advent of the Web, have become a very large segment of the computer market.

Although a server can in principle be any kind of computer, most servers currently are special-purpose computers designed to be rack-mounted in large quantities. For example, Figure 11 shows a large number of servers in use by the Wikimedia Foundation to store and serve their data, including Wikipedia.

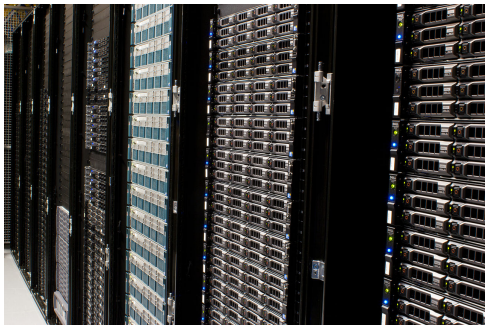

Figure 11: The Wikimedia Foundation servers. [Victor Grigas via Wikimedia Commons, CC-BY$_{SA}$ 3.0]

Supercomputers

A discussion of special-purpose computers would be incomplete without mentioning supercomputers. The term *supercomputer* is somewhat vague, but in general it means a very powerful computer, much more powerful than commodity computers, even very powerful mainframes. Very often, a supercomputer is a highly-parallel computer, i.e., one that can do many operations simultaneously. Sometimes those operations proceed in lock-step with each other, with a single operation being carried out by many processing nodes on many different pieces of data. This is called SIMD, for single instruction, multiple datapath. The other type of supercomputer is one that can perform many different instructions at the same time on many different pieces of data: MIMD (multiple instruction, multiple datapath).

Figure 12: The Cray XC50 supercomputer. [from Cray, Inc. (cray.com)]

Figure 12 shows the current (at the time of writing) flagship supercomputer from Cray, a venerable manufacturer of supercomputers. As can be seen, these supercomputers are a very large, expensive machines that are only affordable by sizable companies, research centers, and the government.

To give you an idea of the power of these machines, let's look at

the successor to the one shown (a Cray XC50) which is currently called Shasta. An initial early production model of Shasta is expected to have a performance of 8 petaflops, with the ultimate peak performance of 180 petaflops. The term *FLOPS* means "floating-point operations per second", and it is used when measuring performance of computers. A *petaflop* is 10^{15} floating-point operations per second. The current generation of high-end microprocessors used in desktops and laptops have a performance under 100 gigaflops. This means that the Shasta is estimated ultimately to be almost 2 million times faster than the fastest desktop or workstation computer. For comparison, the XC50 has a peak performance of about 25 petaflops.

Though the highest-performing supercomputers traditionally were built using special-purpose hardware, newer ones, such as the Cray XC50 shown, combine special purpose hardware with common off-the-shelf (COTS) hardware, in this case, the microprocessors in the Intel Xeon family.[2] At least low-end supercomputer performance can be achieved also by combining commodity computers, for example, desktops, in a *cluster*. A *Beowulf cluster* is one such cluster computer that uses desktop computers, a Unix-like operating system, and a local-area network to achieve very high parallel performance compared to its components.

[2] This is not to say that COTS = inexpensive. Xeon processors can cost up to $10,000 each at the current time.

Before leaving the topic of supercomputers, we should note that yesterday's supercomputer performance is today's commodity-level computer performance. Today's gaming systems, for example, all have performances in the low teraflop (10^{12} flops) range, which means that they are roughly equivalent to the ASCI Red supercomputerr, the fastest supercomputer in the world in 2000. And the Apple iPhone 7, which has a performance of between 12 and 87 gigaflops (depending on the benchmark), is roughly as fast as the fastest supercomputer in the world in 1990..

Mobile devices

Which brings us to the currently ubiquitous computers, mobile devices such as smartphones and tablets. Smart phones have exploded onto the scene relatively recently, propelled primarily by the Apple iPhone in 2007, followed by its successors from Apple and a host of other smartphone manufacturers. Smartphones have a touchscreen and a high-resolution display, and they tend to run either Apple's iOS or the Android operating system (developed by Google), although there are a few others with a (very) minority share of the market, such as Microsoft Windows phones. The market for tablets was jump-started with the Apple iPad, first released in 2010. The tablet market is currently dominated by iPad and Android tablets.

Low-cost, single-board computers

We would be remiss without mentioning one other category of computers, inexpensive computers for hobbyists and low-cost applications. A dominant example of this is the Raspberry Pi, which is a very small, very cheap (~$30) computer-on-a-board that includes everything needed to run (e.g.) a variant of the Linux operating system (Raspbian). Figure 13 shows the current generation of this board. Note that the board includes ports for Ethernet, USB, and connection via HDMI to a monitor.

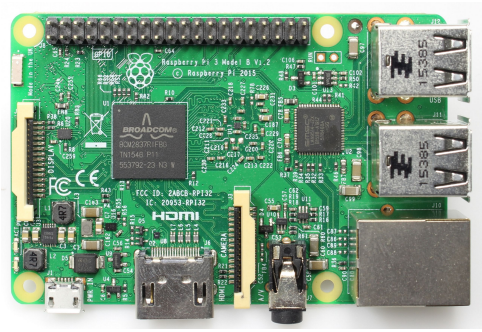

Figure 13: The Raspberry Pi 3 model B. (By Mike Magazine - Own work, CC BY-SA 4.0).

Summary

In this chapter, we have only skimmed the surface of the rich history of computing over the short time digital computers have been in existence. Unfortunately, we have given short shrift to the equally rich history of computer science itself: the development of the theory, ideas, and algorithms underlying all of computing, and the many brilliant men and women who have contributed to the field. We hope that this brief overview will whet the reader's appetite to read more about the field's history.

Review

Further reading

Exercises

1. We have left out so much of computing history, not to mention the history of computer science itself. Pick a topic of interest that is not covered here and write a brief (< 5 page) report about it.
2. The book *Hackers*, by Steven Levy, talks about the hacker culture in computer science, in particular, at MIT during the early 1960s. What did "hacker" mean then? What does the term mean now? Are there still hackers in the original sense of the term? Do you agree how the meaning of the word has changed over time?
3. Write a report about Richard Stallman and the Free Software Foundation.
4. Write a report on the Electronic Frontier Foundation.
5. Like most fields, computer science has a rich history of individuals who have contributed enormously. Choose one of the following and write a brief report about them and their contributions to computer science:
   (a) Ada Lovelace
   (b) John von Neumann

(c) Alan Turing
(d) Niklaus Wirth
(e) Grace Hopper
(f) John McCarthy
(g) Herb Simon
(h) Marvin Minksy
(i) Donald Knuth
(j) Edsgar Dijkstra