THE IN A HURRY SERIES

The History of Computing

in a Hurry

From the abacus to the chip

Max Brocklesby

BOOKSINAHURRY.COM

Contents

The Whole Thing in One Page

The history of computing is often told as a shrinking act: abacus, brass calculator, room-sized electronic machine, desktop, phone, chip. The objects matter, but size is not the organising principle. The deeper story is a succession of bottlenecks: preserving numbers, repeating procedures, automating control, switching quickly, remembering reliably, programming flexibly, manufacturing cheaply and widening access.

An abacus stores the state of a calculation while a trained person supplies the method. Mechanical calculators automate arithmetic but still wait for a person to decide what comes next. The nineteenth-century leap was to make control portable. Punched cards could change a loom's behaviour. Charles Babbage designed an Analytical Engine whose cards would direct a general calculating mechanism. Ada Lovelace explained that such a machine could manipulate symbols under rules, not merely quantities. The Engine remained unfinished, an early warning that a design is not yet a technology.

Computation was also a profession. Governments, observatories, armies and laboratories employed people called computers to produce tables and trajectories. Herman Hollerith's census system mechanised part of that office by turning facts about millions of people into punched-card records that could be sorted and counted repeatedly. Computing was becoming administration as well as mathematics.

In the twentieth century, logic found a physical form. Claude Shannon showed that Boolean algebra could describe switching circuits. War supplied urgency and money. Konrad Zuse's Z3, Britain's Colossus and America's ENIAC crossed different thresholds, which is why the unqualified title of first computer is unhelpful. They combined automatic operation, programmability, generality and electronics in different ways.

The stored program produced the architecture that still matters most. Instructions could occupy memory as encoded information, so a machine's task could change without extensive rewiring. The Manchester Baby proved

electronic stored-program operation in 1948; EDSAC soon turned the idea into a working service. Compilers then automated part of programming itself. Software became an accumulating layer of machinery.

Electronics still had to become dependable and affordable. The transistor reduced the heat, power and fragility of vacuum-tube systems. Integrated circuits put many components and connections onto semiconductor material. IBM's System/360 showed that compatibility could matter as much as speed. The microprocessor put a general processing unit on a chip and turned the computer from a destination into a component.

Then access became scarce. Time-sharing put many users in conversation with one expensive machine. Personal computers moved control to the desk. Graphical interfaces reduced the knowledge needed to operate them. Networks connected the machines, and low-power processors pushed computation into pockets and ordinary objects.

No lone inventor made this happen. States bought weapons and censuses; universities developed memory and languages; corporations built factories, standards and service organisations; women performed much early calculating and programming work before the occupation was professionalised in ways that often reduced their opportunities. Every solved bottleneck created another one.

A computer is therefore not one machine invented once. It is a stack of ways to store state, encode procedure, switch decisions, remember instructions, reproduce components and organise access. The abacus and the chip sit at opposite ends of that stack, but both work only because people have decided what their states mean and what should happen next.

That is the book.

Why You Should Care

A computer once had a desk, a supervisor and a salary.

Before the word named an electronic machine, it named a job. Human

computers calculated astronomical tables, artillery trajectories, insurance figures and engineering results. At Langley, the American aeronautics laboratory that later became part of NASA, women worked in computing pools, reading instruments, applying formulas and plotting results. By 1946 the ranks at Langley had grown to roughly four hundred women. Electronic machines entered organisations that already knew how to divide calculation into procedures, distribute it and check it.

That reversal is more than a historical curiosity. It tells you what a computer is. The familiar box is only the visible layer. Underneath it are representations, instructions, memory, standards and institutions. A phone can act as a camera, map, bank terminal and music player because radically different activities have been translated into forms that one general machine can process. The same processor does not understand photographs, routes or money. It manipulates encoded states under rules supplied by software and connected organisations.

The general-purpose part matters more than spectacular speed. A machine designed only to multiply remains a multiplier however fast it becomes. A stored-program computer can change its role when the instructions change. Once instructions themselves are data that can be copied and modified, useful procedures become cheap to reproduce. One expensive act of writing software may direct millions of machines. That asymmetry helps explain why computing firms can scale so quickly and why errors can scale with them.

The history also gives a better test for technological claims. A prototype proves that something can work. It does not prove that it can be manufactured reliably, afforded, maintained, trusted or fitted into existing work. Babbage had a general architecture without the industrial system needed to finish it. ENIAC had electronic speed but awkward programming. The transistor had to be turned from a laboratory device into a repeatable component. The integrated circuit mattered because manufacturing could reproduce many connected devices. System/360 mattered because customers could carry software and skills from one model to another. The

microprocessor mattered because a processor could be sold as a component rather than a whole institution.

That distinction is useful far beyond old hardware. When somebody announces a technical breakthrough, ask what bottleneck has moved. Faster calculation can expose memory limits. Cheaper hardware can expose software costs. Easier programming can produce larger programs. More storage can produce larger archives. Better networks can make security, attention or energy the scarce resource. Computing rarely abolishes work. It changes which work is expensive enough to notice.

There is a political consequence too. Every layer decides who may act without asking somebody else. A mainframe centre concentrates scarce equipment and expertise. Time-sharing widens access while leaving control centralised. The personal computer shifts more decisions to the user. Networked services make the user's machine more capable by depending on remote infrastructure. Convenience and dependence can increase together. History makes that trade-off easier to see because earlier systems display their seams more openly.

The subject also cuts through the mythology of the solitary inventor. Computing advanced through mathematicians, clerks, engineers, codebreakers, operators, programmers, managers, governments, component makers and standards committees. Some people made decisive contributions, but the machine only changed society when a surrounding system could reproduce the contribution. The famous object is usually the tip of a much larger organisation.

By the end of this book, you should be able to look at any digital system and ask four questions. What is being represented? Where are the instructions? Which bottleneck was solved to make this scale possible? Who controls the layers the user does not see? Those questions turn computing from a procession of magical inventions into an intelligible history of choices.

Once that history is visible, the modern computer looks less like a miraculous object and more like accumulated infrastructure.

The Core Ideas

1. Computation Was Organised Labour

The word computer first named a person because large calculations became manageable before they became mechanical.

Astronomers needed tables of planetary positions. Navigators needed values they could carry to sea. Insurers, surveyors, engineers and artillery officers needed more arithmetic than any one expert wanted to repeat. The efficient answer was to split the job. One person established a method, others performed repeated steps, supervisors divided the work and checkers compared results. Printed tables then carried the finished computation to people who never met the original calculators.

That organisation already contains several parts of a modern computer system. There is a procedure, working memory, stored results, error checking and a division between the people who build the calculation and those who use it. The hardware was paper, ink, counters and trained hands.

An abacus shows the boundary clearly. Its beads preserve state. They let a quantity remain visible outside the operator's head and give place value a physical arrangement. The instrument does not decide which operation is appropriate. A trained user supplies the procedure. Counting rods and boards across Asia and Europe worked through related partnerships between material arrangement and learned method. They were complete technologies in their own settings, not failed electronic computers waiting for a future inventor.

Written notation moved another part of the work. Hindu-Arabic positional numerals, including zero, made paper calculation compact and systematic as the notation spread through the Islamic world and into Europe. Printed arithmetic texts could standardise methods. Logarithmic tables turned multiplication and division into easier operations by shifting difficult calculation to the people who produced the tables. Computation was already being separated into expensive preparation and cheap reuse.

Mechanical calculators then automated individual operations. Pascal's seventeenth-century machine used decimal wheels and automatic carries. Leibniz's stepped reckoner extended the ambition to multiplication and division. Later office calculators became practical tools for clerks. Yet the operator still read the problem, chose an operation, entered the numbers and decided when the answer meant anything. Mechanisation had crossed the arithmetic boundary without crossing the control boundary.

By the nineteenth and twentieth centuries, human computing had become industrial. Observatories, governments and laboratories organised teams around repeated numerical work. The occupation was often classified as clerical and increasingly assigned to women, partly because institutions could buy mathematically skilled female labour more cheaply under the discriminatory pay structures of the time. At Langley, the first female computing pool began in 1935 with five women and grew dramatically during the Second World War. Black women were recruited into a segregated West Area computing group in the 1940s. Their calculations supported aeronautical research and, later, the American space programme.

Electronic computers did not make that organisation vanish. They entered it. Human computers prepared problems, punched data, checked machine output and often became programmers. The new machine inherited a task that had already been decomposed into explicit procedures.

That is the first causal condition of the book. Automation needs legibility. Work can be transferred to a machine only after somebody has made the states, rules and stopping conditions explicit enough to repeat. The history of computing begins with people learning to turn reasoning into an organised procedure.

2. Control Becomes Portable

A calculator performs an operation chosen by its user. A programmable machine carries a sequence of choices in some external form. That difference changed the history more than another turn of mechanical speed.

The key precedent came from textiles. Joseph-Marie Jacquard's loom used punched cards to determine which warp threads were raised. The holes did not power the loom or contain a mathematical algorithm. They selected actions. Change the card sequence and the same mechanism could weave a different pattern. Control had become detachable from the machine.

Charles Babbage pushed that idea into calculation. His Difference Engine, begun in the 1820s, was designed to produce mathematical tables by a fixed numerical method and print them directly, reducing human copying error. His later Analytical Engine was more ambitious. Babbage separated a store for numbers from a mill that operated on them and proposed punched cards to specify operations. The design allowed repetition and conditional changes in sequence. Its arithmetic was decimal and its components mechanical, yet its organisation is recognisably general-purpose.

The Analytical Engine was never completed. That failure belongs inside the main story because it establishes a recurring rule. Architecture, manufacturing and institutions must mature together. Babbage changed designs, precision engineering was expensive, disputes with his engineer Joseph Clement damaged progress and British government support eventually ended. A machine can be conceptually sound and historically premature if the surrounding system cannot deliver it.

Ada Lovelace's 1843 notes on Luigi Menabrea's description of the Engine are important for a different reason. Her Note G set out the successive changes required to calculate Bernoulli numbers. It is often called the first computer program, although Babbage had written earlier unpublished program-like material and the modern category does not map neatly onto either person's notation. The safer distinction is stronger: Lovelace published a detailed worked plan for directing a machine that did not yet exist and described the generality of symbolic manipulation with unusual clarity.

She understood that numbers in the Engine need not stand only for quantities. If formal relations could be represented numerically, the machine might manipulate symbols associated with music or other domains. The

point was not that the Engine would understand those things. It was that representation lets one mechanism act on many kinds of subject. That is the conceptual move behind digital media today.

Punched cards soon travelled down a more practical route. Herman Hollerith's electromechanical system for the 1890 United States census represented attributes of each person as holes in standard card positions. Pins passing through holes closed circuits and advanced counters. Cards could be sorted and counted again for another question. The system reduced tabulation cost and time, then spread into railways, insurers, governments and large companies. Hollerith's business lineage eventually became part of IBM.

This was computation as administration. A card made information movable and machine-readable, but it also forced the world into predefined fields. A category had to exist before a hole could represent it. Machine-readable records increased organisational power by making populations and transactions easier to compare, aggregate and reorder.

The same physical idea therefore produced two different historical powers. A punched card could carry instructions for a machine or data about a person. In both cases, something that had lived in human memory or judgement acquired a standard external form. Once control and information became portable, machines could begin to act on them repeatedly and at scale.

3. Logic Becomes Switchable

Babbage had a general architecture but no suitable electronic components. The twentieth-century bridge came from an unexpected meeting between abstract logic and telephone switching.

George Boole's nineteenth-century algebra described propositions through operations corresponding to AND, OR and NOT. Boole was formalising reasoning, not designing computers. Telephone engineers were solving a separate problem: how to route signals through networks of relays whose contacts were open or closed.

Claude Shannon joined the two. In his 1937 MIT master's thesis he showed that Boolean algebra could analyse and simplify relay and switching circuits. A logical expression could be translated into a network of physical switches, and a circuit could be reasoned about algebraically. Design moved from wiring intuition towards a formal method.

That connection matters because a computer must do more than calculate quantities. It must make controlled distinctions. Is this value zero? Has a condition been met? Should execution continue here or jump elsewhere? Switching circuits give those decisions a physical mechanism.

Binary representation fitted the engineering well, though it was never a philosophical necessity. Babbage's engines were decimal. ENIAC used decimal accumulators. Analogue computers represented quantities continuously. Binary became dominant because physical devices can distinguish two broad ranges of state more reliably than many closely spaced ones. A voltage can vary somewhat while still being interpreted as a zero or a one. Noise changes the exact signal before it changes the category. Two-state components can also be combined into larger logical structures without every designer having to reason about the analogue behaviour of the whole system.

Konrad Zuse exploited that advantage in Germany. His Z3, completed in 1941, used relays for binary floating-point arithmetic and read instructions from punched film. It operated automatically through a program sequence, but the instructions were not stored in its memory and its normal program mechanism did not provide the flexible branching associated with later general-purpose machines. It nevertheless crossed a major threshold: a functioning program-controlled digital calculating machine built from switching components.

Analogue machines remained serious rivals. Vannevar Bush's differential analyser represented equations through the motion of shafts, gears and integrators. Engineers configured a physical system whose behaviour corresponded to the mathematics they wanted to solve. Such machines could be intuitive and effective for continuous problems. Digital computing

did not win because analogue work was foolish. It won because discrete states were easier to copy, store and restore, while programmability allowed one architecture to be repurposed across many tasks. Hybrid analogue-digital systems continued to matter in engineering, and specialised analogue computation survived where continuous physical relationships could be exploited efficiently. The later dominance of digital machines can make the abandoned branches look obviously inferior. They were often sensible answers to the components, costs and problems available at the time.

The deeper shift is therefore from arithmetic mechanism to logical composition. Once a designer can build reliable switching elements and combine them according to formal rules, complexity becomes modular. A small logical decision can be made into a component, then thousands and eventually billions of such decisions can be composed.

The computer begins to look modern at this point, even before electronics supply the speed.

4. War Pays for Several Electronic Breaks

The Second World War did not produce one first computer. It produced several urgent problems, several secret programmes and several machines that crossed different thresholds.

Germany's Z3 automated engineering calculations with electromechanical relays. At Bletchley Park, Britain's Colossus attacked the Lorenz cipher used for high-level German communications. Tommy Flowers and his team used large numbers of vacuum tubes, high-speed paper tape and configurable switches and plugboards to perform logical tests on intercepted data. Colossus was electronic, digital and programmable within its codebreaking role, but it was not a general machine for arbitrary numerical work. Its secrecy delayed public recognition for decades.

American artillery created another demand. Firing tables translated elevation, charge, weather and other conditions into settings gunners could use. Human computers, many of them women, calculated the trajectories

with desk calculators and differential analysers. At the University of Pennsylvania's Moore School, John Mauchly and J. Presper Eckert proposed a large electronic machine that could push through such calculations far faster.

ENIAC used roughly 18,000 vacuum tubes and occupied a room of cabinets. It demonstrated that a large electronic digital system could run useful numerical work at unprecedented speed and with workable reliability. Its original programming method exposed the next bottleneck. Programs were configured through cables, switches and function tables rather than loaded as ordinary data from a shared memory. Electronic arithmetic had become fast; changing the job remained laborious.

The people who solved that problem deserve to remain inside the machine's history. Six women recruited from the Moore School's human computing operation became ENIAC's first programmers: Kathleen McNulty, Frances Bilas, Betty Jean Jennings, Ruth Lichterman, Betty Snyder and Marlyn Wescoff. They studied diagrams, traced data paths and translated mathematical procedures into the physical configuration of a machine for which programming practice scarcely existed. Their contribution was obscured for years, partly because programming had not yet acquired the prestige later attached to it.

War accelerated development because military institutions could spend heavily for codebreaking, ballistics, weapons research and control. It also damaged the historical record. Secrecy isolated projects. National prestige encouraged singular origin stories. Patents turned definitions into legal weapons. Hardware design and programming were assigned different status even when both were necessary to make a machine work.

That is why the familiar first-computer argument has no clean answer. The Atanasoff-Berry Computer used electronic binary arithmetic but was not a general programmable machine. Z3 was automatic and program-controlled but electromechanical. Colossus was electronic and programmable but specialised. ENIAC was electronic and broadly general-purpose but originally cumbersome to reconfigure. Later stored-program machines

added another decisive property.

The useful question is therefore: first at what? Automatic operation, programmability, generality, electronics and stored instructions are separate thresholds. Early machines combined them in different ways because they were built for different institutional problems.

War paid for several breaks at once. Peace would have to turn them into a common architecture and an industry.

5. Memory Makes the General Machine Practical

Electronic speed is less impressive if a new problem requires days of rewiring. The stored program solved that mismatch by turning instructions into information that the machine could keep in memory.

Alan Turing had supplied the logical foundation before the war. His 1936 paper described abstract machines that read and write symbols according to finite rules and showed that a universal machine could simulate any other machine of the same formal kind when supplied with an adequate description. It was not an engineering plan for a 1940s computer. It established the idea that generality could reside in encoded instructions rather than in a different physical machine for every task.

During work around ENIAC and its successor EDVAC, engineers developed the architecture of an electronic stored-program machine. John von Neumann's 1945 First Draft of a Report on the EDVAC described memory, arithmetic, control and input-output in a form that circulated widely. The document was enormously influential and attached his name to the model, but the design emerged from collaborative discussions involving Eckert, Mauchly and others. History kept the convenient label and lost some of the committee.

The difficult part was memory. Engineers could draw a block marked store long before they had a fast, dependable and affordable way to build one. Mercury delay lines circulated pulses through tubes of liquid. Williams tubes stored patterns of charge on cathode-ray screens. Magnetic drums and later

magnetic cores offered other compromises among speed, capacity, cost and reliability. Where information lived, how quickly it could be reached and whether it survived long enough became central design questions.

At Manchester, Frederic Williams and Tom Kilburn built the Small-Scale Experimental Machine mainly to test Williams-tube memory. On 21 June 1948 the Baby ran a program held electronically in its own store. That made it the first working electronic digital computer to execute a program stored in electronic memory. It was a proof machine, not a useful computing service.

EDSAC at Cambridge crossed the next practical threshold. In May 1949 it ran its first programs and soon served university researchers. Maurice Wilkes's group developed reusable subroutines so programmers could call tested pieces of code instead of rebuilding common operations each time. A computer was beginning to accumulate software as well as hardware.

Compilers moved another bottleneck. Early programmers wrote numeric codes or mnemonic assembly instructions close to the hardware. Grace Hopper's compiler work and the IBM team led by John Backus pushed translation into software. FORTRAN, released in 1957, allowed scientists and engineers to express formulas in a higher-level language and rely on a compiler to produce efficient machine instructions. The compiler did not remove the need for programming. It enlarged the class of people who could write programs and the scale of programs worth attempting.

Operating systems, libraries and shared services followed for the same reason. Once instructions could call other instructions, software became an environment. The scarce resource shifted from electronic arithmetic to memory capacity, programmer time, machine scheduling and the management of increasing complexity.

The stored program is the central architectural pivot because it changes identity from a property of hardware into a temporary state. A payroll system, simulation and game can occupy the same physical machine at different times. Memory made that generality practical, and software made it cumulative.

6. Scale Requires Components, Compatibility and Capital

A useful electronic computer needs thousands, then millions, then billions of parts to behave well enough that the user can ignore them. Speed alone could never produce mass computing. Reliability, manufacturing and compatibility had to become industrial disciplines.

Vacuum tubes supplied electronic switching but imposed heat, power, size and maintenance costs. At Bell Laboratories in 1947, John Bardeen and Walter Brattain demonstrated a working point-contact transistor while working in a group led by William Shockley; Shockley soon developed the junction transistor. Solid-state devices promised smaller size, less power and greater reliability, although turning laboratory physics into dependable production required years of materials work and process control.

Discrete transistors then exposed another bottleneck: connection. Every component had to be wired to others, and every joint could fail. Jack Kilby at Texas Instruments demonstrated an integrated circuit in 1958. At Fairchild, Jean Hoerni's planar process protected semiconductor structures beneath silicon dioxide, and Robert Noyce proposed using deposited metal to interconnect devices on the same wafer. The combination led to planar monolithic integrated circuits that could be manufactured in volume. The history therefore has more than one inventor because integration and scalable fabrication were related but distinct achievements.

Manufacturing altered the economics. Photographic patterns could reproduce many devices across a wafer. Better process control improved entire batches rather than one hand-built circuit. Military and space programmes bought early integrated circuits at prices consumer markets would not tolerate because size, weight and reliability mattered urgently. Higher volume then reduced costs and encouraged further integration. Gordon Moore's 1965 observation about economically attractive component density became a planning expectation that coordinated manufacturers, equipment suppliers and customers for decades. It was an industrial target, not a physical law.

Standardisation mattered at the level of whole machines. IBM's System/360, announced in 1964, offered a family of computers of different sizes around a common architecture. A customer could upgrade without discarding all software, peripherals and training. IBM had to coordinate processors, operating systems, storage, documentation, factories, sales and support. The cost and delays were severe, but compatibility reduced the risk of adopting a computer as a long-term organisational platform. A specification became an economic asset.

Digital Equipment Corporation loosened another constraint with minicomputers. Machines such as the PDP-8 were cheaper and small enough to sit in laboratories, factories and departments that could not justify a central mainframe installation. Access widened before the personal computer existed.

The microprocessor compressed central processing onto an integrated circuit. Intel's 4004 emerged in 1971 from a contract with the Japanese calculator company Busicom. Ted Hoff and Stan Mazor developed the architectural proposal, Federico Faggin led the chip implementation and Masatoshi Shima contributed detailed logic from the customer side. The result was a commercially available general-purpose microprocessor within a supporting chip family. A processor could now be purchased as a component and embedded in another product.

These developments solved different scaling problems. The transistor improved switching. Planar integrated circuits improved density and reproducibility. Compatible architectures protected investment. Minicomputers lowered the organisational price of access. Microprocessors lowered the physical and economic price of putting a processor into a system.

The computer spread because components, software and standards became reproducible enough for strangers to trust. Mass computing was a manufacturing and coordination achievement as much as an invention.

7. Access Becomes the Next Bottleneck

Once processors became smaller and cheaper, the main scarcity was no longer arithmetic. It was access: who could get time on a machine, how directly they could interact with it and how much knowledge the interface demanded.

Batch computing made expensive hardware efficient by making users wait. Programs and data arrived through cards or tape, operators scheduled runs and results returned later. Time-sharing reversed the priority. One central computer switched among several interactive users quickly enough that each terminal appeared responsive. The hardware remained scarce, but access began to feel personal.

Researchers pushed that idea towards a new style of work. Douglas Engelbart's 1968 demonstration at SRI showed a mouse, linked text, windows, remote collaboration and interactive editing. Xerox PARC's Alto in the 1970s combined graphical display, mouse, networking and individual use in a research computer. Apple later commercialised graphical interaction through the Lisa and, far more successfully, the Macintosh. Interfaces shifted another burden from users into software. Recognition could replace memorised commands.

Microprocessors widened ownership. The Altair 8800 reached hobbyists in 1975 as a kit whose incompleteness invited expansion. The Apple II, Commodore PET and TRS-80 arrived in 1977 as more complete systems. VisiCalc gave small businesses a compelling reason to buy a personal computer. IBM's Personal Computer in 1981 supplied corporate legitimacy and, through widely available components and published technical information, helped create an ecosystem of compatible clones. Users no longer had to submit work to a central computing department.

Networks complicated that independence almost immediately. Packet-switching research in Britain, France and the United States developed ways to divide messages, route them over shared links and reassemble them. ARPANET linked research machines. TCP/IP made

unlike networks interoperable. Ethernet connected local machines. The World Wide Web made remote information feel browsable rather than merely reachable. The personal computer became a gateway to other computers.

Mobile systems shifted the engineering priorities again. A desktop could spend electricity and generate heat for speed; a battery-powered device could not. ARM began at Acorn in Britain in the 1980s with a relatively economical processor design and later a licensing model that allowed many companies to incorporate ARM cores into their own chips. Low-power processing became more valuable as computing moved into portable devices. By the time smartphones combined touch interfaces, sensors, radios, operating systems and downloadable applications, the machine had become a platform hidden inside another product category.

This is where Core Idea 1 returns. Human computation began as visible organised labour: people followed procedures around tables and tools. Modern computing appears to remove that labour because the procedure is buried behind software, chips and remote services. The people did not disappear. They moved into programming, semiconductor fabrication, data-centre operation, interface design, moderation, repair, logistics and standards. Earlier decisions remain fixed in code and categories long after their authors have left.

Solving access also created dependence. A mainframe user depended on a central operator. A personal-computer user gained local control. A networked user gained far more capability while depending on remote protocols, services and providers. The direction is not from dependence to independence. It is from visible dependence to layered dependence, with more capability at each step.

The computer's final historical trick was therefore to stop looking like a computer. Once processing became cheap enough, general enough and connected enough, it could disappear into phones, cars, appliances, factories and infrastructure. The machine became ordinary precisely because the system around it became enormous.

How It Actually Works

Marks, beads and tables

The first computational technologies were ways of refusing to lose count.

A notch, pebble or token lets a quantity outlive the attention of the person counting it. An abacus goes further by assigning positions different values. A bead near one rod can mean units, another tens or another fraction, depending on the instrument and practice. The device makes place value tangible and allows a trained hand to carry, borrow and combine quantities quickly. It does not contain a fixed method. Two people can use the same frame to follow different procedures.

Counting boards, rods and bead frames took different forms across cultures. They were not crude machines waiting to become electronic. Their power lay in a convention shared between hand, surface and trained memory. The same physical counters could support different methods, which is why the instrument cannot be separated from the practice that made it work.

Written numerals changed what could be stored away from the body. The spread of Hindu-Arabic positional notation and zero through the Islamic world into Europe made calculation on paper more flexible than Roman numerals allowed. Printed arithmetic books then stabilised methods across distance. The word algorithm preserves the Latinised name of the ninth-century scholar al-Khwarizmi, though the full history of algorithms belongs elsewhere. Here its significance is material: a procedure can be written so another person can reproduce it.

Tables multiplied the effect. Once logarithms were published in the seventeenth century, multiplication could be converted into addition and division into subtraction. Astronomers and navigators could use precomputed values instead of repeating difficult derivations. The labour moved upstream into making, checking and printing the table.

Mechanising arithmetic

The dream of replacing that labour with gears arrived beside clocks and precision instruments.

Blaise Pascal built a calculator in the 1640s to help with his father's tax accounts. Its wheels represented decimal digits and carried automatically from one place to the next. The carry was the hard part: a small movement in one wheel had to release a reliable full step in another after nine. Pascal made several machines, but they were expensive, delicate and narrower in use than skilled clerks with familiar tools.

Leibniz later designed the stepped reckoner, using a cylinder with teeth of graduated length to support multiplication and division through repeated operations. He also argued for binary arithmetic, seeing philosophical elegance in reducing number to one and zero. The machine suffered its own mechanical difficulties. The history of computing is littered with devices that worked in demonstration, worked partly or could be made to work by their inventor, then failed the harsher test of routine use by strangers.

Mechanical calculators became practical businesses in the nineteenth century. Thomas de Colmar's Arithmometer and later office machines gave clerks dependable assistance with commercial arithmetic. They accelerated individual operations but left the work sequence outside the mechanism. A person still read the problem, chose an operation, entered numbers, turned a crank and copied the answer.

The cards and the unfinished engine

The Jacquard loom showed how a machine could change behaviour without changing its mechanism. A sequence of punched cards determined the pattern woven into cloth. The same loom could produce another design when given another chain.

Babbage borrowed that separation for the Analytical Engine. He described a store holding numbers, a mill carrying out operations and cards controlling the sequence. Cards could repeat a group of instructions, and the design

included conditional movement through the program. Output might be printed or punched to reduce copying. In modern language, the machine had an architecture, memory, processor, program and input-output.

Building it required thousands of precise parts, a stable design and a government willing to fund years of development. Babbage supplied none of these conditions consistently enough. Work on the earlier Difference Engine consumed public money and political patience. The engineer Joseph Clement produced components of remarkable quality, but disputes over cost, workshop ownership and design changes helped end the arrangement. Government support stopped in 1842.

Ada Lovelace entered the history through explanation rather than construction. Her 1843 notes set out how cards might direct the Engine through a Bernoulli-number calculation and argued that the machine could operate on symbols whenever their relations were expressible. She also stated its limit clearly: it could follow analysis but did not originate the relations supplied to it. The machine remained unfinished, so her program was never run on it.

The Science Museum later built Babbage's Difference Engine No. 2 from his plans and demonstrated that the design could work. That success does not retroactively finish the Analytical Engine. It shows the difference between conceptual feasibility and a nineteenth-century project capable of delivery.

Counting populations and governing by card

Punched cards became useful first through data, not general-purpose programming.

The United States census faced a growing administrative crisis. The 1880 count took years to tabulate, while population growth threatened to make the next one slower still. Herman Hollerith proposed recording each person's attributes as holes in a card. A clerk placed the card in a press. Pins descending through holes touched mercury cups, closed circuits and advanced the appropriate counters. Sorters grouped cards so the same population could be counted in different ways.

The 1890 system turned a person into a standard record and a question into a machine-readable position. It shortened key stages of tabulation and made repeated classification practical. Hollerith built a business around the equipment. His company later joined the 1911 merger that became the Computing-Tabulating-Recording Company, renamed International Business Machines in 1924.

Punched-card systems spread through governments, railways, insurers and large corporations. Their importance lay less in spectacular arithmetic than in routine control. Payroll, inventory, billing and accounts could be broken into card flows. Wiring panels set the sequence for tabulators. Rooms filled with keypunch operators, card sorters and machines whose speed mattered because organisations now processed whole populations of transactions.

The card imposed discipline. A field had to be defined before it could be punched. Information that did not fit the schema was omitted or forced into a category. Computing expanded administrative power by making records movable, comparable and countable.

Logic, relays and analogue rivals

By the 1930s there was no single agreed road to the computer.

Analogue machines represented a problem through a physical system whose changing quantities behaved like the thing being studied. Vannevar Bush's differential analyser at MIT used rotating shafts, gears and wheel-and-disc integrators to solve differential equations approximately. Engineers connected the machine differently for each problem, then watched curves emerge. It was large, oily and useful. Copies helped calculate artillery tables.

Digital machines represented quantities as discrete values. Telephone engineering supplied relays and a culture of switching circuits. Shannon's Boolean treatment gave designers a method for reducing logical expressions and translating them into networks. At Bell Labs, George Stibitz built relay calculators that could be controlled remotely. At Harvard, Howard Aiken and IBM produced the Automatic Sequence Controlled Calculator,

known as Mark I, completed in 1944. A fifty-foot shaft synchronised its electromechanical parts; instructions arrived on punched tape.

Analogue and digital systems coexisted for decades. Analogue machines could model continuous systems in parallel and give engineers immediate interaction. Early digital machines were expensive and often used in batches. Digital systems won the larger market because discrete information could be copied, stored and processed with error margins, and because one programmable architecture could attack many problems. The victory was gradual rather than ordained.

Several roads through war

Konrad Zuse built machines in Germany with little knowledge of work elsewhere. His relay-based Z3 ran in 1941, using binary floating-point arithmetic and punched film for instructions. It could execute a sequence automatically, though not store the program in memory. Zuse's work received limited wartime backing and the original machine was destroyed.

Britain's codebreaking effort followed intercepted communications. Tommy Flowers, an engineer from the General Post Office, argued that vacuum tubes could be reliable if kept running rather than repeatedly switched on and off. Colossus began operating at Bletchley Park in 1944. It read punched paper tape at high speed and applied programmable logical tests to help analyse Lorenz traffic. Ten machines were built by the war's end. Most were later destroyed and the project stayed secret, leaving public histories to grow around machines that could be discussed.

ENIAC arose from the backlog in ballistic calculations. Mauchly brought experience with electronic counting; Eckert supplied formidable engineering control. Construction began in 1943. The finished machine used decimal accumulators, electronic pulses and plugboard programming. It was completed too late to transform wartime firing-table production, but it soon worked on problems including atomic research, weather and numerical mathematics.

Its programmers had no programming manual in the modern sense. They

traced data paths through diagrams, decomposed equations and decided how pulses should move among units. Reconfiguration could take days. In 1948 ENIAC was modified so a form of program could be read from its function tables, sacrificing some parallel speed for easier control. Even a famous machine changed identity during its working life.

The stored-program race

The plan for EDVAC pushed instructions into electronic memory. Von Neumann's First Draft of a Report on the EDVAC described the logical arrangement clearly enough to circulate among research groups. The document helped standardise thinking and helped turn a team's developing ideas into the so-called von Neumann architecture.

Memory technologies competed. Mercury delay lines circulated pulses through tubes of liquid and returned them after a fixed interval. Williams tubes used charged spots on cathode-ray screens. Magnetic drums stored patterns on rotating surfaces. Each forced programmers to care about physical timing and location.

The Manchester Baby existed mainly to test the Williams tube. On 21 June 1948 it ran a stored program that searched for a factor of a number. The program was small and the machine austere, but instructions and data occupied the same electronic store. The Manchester Mark I developed from it and led to Ferranti's commercial machine.

At Cambridge, Wilkes chose delay-line memory for EDSAC. On 6 May 1949 it ran a program calculating squares and another producing a table of primes. Researchers soon used it as a service rather than an experiment. EDSAC also developed reusable subroutines, stored pieces of program that could be called by other programs. Software began to form a library.

Turing's Automatic Computing Engine design at the National Physical Laboratory aimed for high speed and compact instruction encoding. Bureaucratic delay frustrated the full project, though Pilot ACE ran in 1950. There was no clean baton passing from theory to one machine. The stored-program computer emerged from overlapping work on logic,

memory, electronics and institutional need.

Selling a system, not a box

Early commercial forecasts often underestimated demand because they imagined computers as rare scientific instruments. The market expanded when businesses found repeatable administrative uses.

The US Census Bureau received the first UNIVAC I in 1951. The machine used magnetic tape and gained public attention by forecasting the 1952 presidential election from an early sample. IBM entered electronic computing from a strong base in punched cards, customer relationships and service. Its 1401, announced in 1959, became a widely used business computer because it fitted existing card and tape operations.

Hardware alone could not make these machines useful. Programmers wrote routines for input, sorting, payroll, scientific calculation and device control. Hopper's A-0 system linked symbolic instructions to stored routines. Her later FLOW-MATIC work influenced COBOL, a common business language developed through an industry and government committee. Backus's FORTRAN team at IBM built a compiler capable of translating mathematical expressions into code efficient enough for the costly IBM 704. As jobs multiplied, software also had to schedule processors, manage memory and coordinate input-output. Operating systems grew from the practical need to stop expensive hardware sitting idle while every program reinvented the machinery around it.

Programming shifted from wiring a machine to managing layers of symbolic instructions. The shift opened the field while creating a shortage of experienced workers. Employers struggled to define the occupation, test aptitude and control software projects whose difficulty was hard to measure. The term software engineering expressed an ambition for discipline before it described a settled method.

System/360 raised the commercial stakes. Announced in 1964, the family shared an instruction architecture across machines of different sizes. IBM had to develop processors, peripherals, software, factories and training

together. Delays and cost overruns threatened the company, but compatibility gave customers a path upward and gave software a larger market. A computer was now a long-term platform.

The processor becomes a component

Transistors replaced tubes gradually through the 1950s and 1960s. They reduced power and size, but wiring thousands of separate devices remained expensive. Integrated circuits compressed components and connections into repeated patterns on semiconductor wafers.

The first customers were willing to pay for weight and reliability. Missile guidance and the Apollo programme bought early integrated circuits. As production improved, costs fell. Semiconductor memory began replacing magnetic cores. The logic of manufacturing rewarded volume: a good design could be reproduced across many chips, while each new generation of fabrication equipment demanded enormous capital.

The microprocessor gathered the central control and arithmetic functions onto one integrated circuit. Intel's 4004 appeared in 1971 as part of a chip set for Busicom calculators. It was modest by later standards, but its general design allowed one processor to serve many control tasks when paired with memory and input-output. Intel's later 8008 and 8080, Motorola's 6800 and MOS Technology's 6502 widened the field.

DEC's minicomputers had already shown that smaller general machines could sit near the work. Microprocessors drove the price low enough for industrial controllers, instruments, games and hobby systems. The computer no longer needed to be sold as a complete central installation. It could disappear inside another machine.

A computer for one person

The January 1975 cover of Popular Electronics presented the Altair 8800 kit. A buyer entered programs through front-panel switches and read results from lights unless extra equipment was added. The machine became influential because its bus allowed other firms to make compatible boards and because hobbyists treated incompleteness as an invitation. It was not the first small computer and it was barely usable out of the box. Its importance was social and architectural: it gathered an expanding market of component makers, programmers and owners around a machine ordinary enthusiasts could imagine controlling themselves.

Bill Gates and Paul Allen supplied a BASIC interpreter. The Homebrew Computer Club in California became a meeting place for designs, parts and arguments about ownership. Steve Wozniak showed the Apple I there, then designed the more complete Apple II with keyboard, colour graphics, expansion slots and BASIC in memory. The floppy drive and VisiCalc spreadsheet turned it from an enthusiast's machine into a business tool.

Other systems, including the Commodore PET and Radio Shack TRS-80, built a mass market. IBM's 5150 PC entered in August 1981 using an Intel processor, an operating system supplied by Microsoft and an architecture assembled largely from available components. IBM's name reassured corporate buyers. The open documentation and purchasable parts allowed other companies to build compatible machines, shifting power from one product to an ecosystem of clones.

Interface changed alongside ownership. Engelbart's SRI system had shown interactive text and a mouse in 1968. Xerox's Alto combined graphical documents, networking and personal use in the 1970s. Apple's Lisa brought those ideas to a commercial personal computer at an unaffordable price; the Macintosh reached a wider audience in 1984. Pointing at visible objects reduced the amount of command language a user had to remember. The computer became easier to approach and harder to inspect.

From stand-alone machine to networked pocket

Research networks made distant computers share resources before personal machines became common. Packet switching divided messages into units that could take routes through a network and be reassembled. Work at the National Physical Laboratory, ARPA and elsewhere developed the method. ARPANET's first links went live in 1969. TCP/IP later supplied rules for connecting different networks, with ARPANET's major transition completed on 1 January 1983.

Ethernet connected machines locally. Modems and online services brought homes into communication. The World Wide Web, developed by Tim Berners-Lee at CERN from 1989, made networked information easier to publish and follow through links. The Internet's full operation belongs to another book; its effect here was to change the personal computer from a self-contained tool into a terminal for distributed systems.

Mobile computing required another optimisation. Desktop processors could spend power to gain speed. A battery-powered device had to ration energy and heat. The ARM architecture, developed at Acorn in Britain from 1985 and later licensed widely, succeeded through efficient design and a business model that let other companies incorporate processor cores into their own systems. Phones became programmable computers long before the smartphone made the fact obvious.

The 2007 iPhone joined a multitouch interface, mobile network, sensors and a desktop-derived operating system in one object. A native third-party app platform followed in 2008. Its significance was architectural rather than magical: decades of semiconductor integration, graphical interaction, radio infrastructure, operating systems and software distribution had become one consumer system. The route from the abacus ended, for the moment, with a computer whose user rarely saw a number being calculated.

How we know

Computing leaves abundant evidence and unusually treacherous labels. Machines survive in museums, but a preserved cabinet may not show how it changed during use. Programs vanish when media decay, formats become unreadable or source code was never kept. Corporate archives emphasise products that firms chose to celebrate. Military secrecy hid Colossus for decades. Patent cases encouraged participants to sharpen claims about invention.

Reconstruction helps. Babbage's plans can be tested by building mechanisms from them; Zuse's Z3 and early British machines have been reconstructed; software preservation projects run old code in emulators. Oral histories recover working practice, though memory is selective and status shapes who was interviewed.

The largest trap is the word first. Historians can support several answers by changing the adjectives attached to computer. The sound method is to identify the property at issue, use contemporary documents and examine what later systems inherited. Influence, routine use and reproducibility often explain more than priority by a few months.

What People Get Wrong

“One genius invented the computer”

The myth survives because invention stories prefer a face, a date and a finished object. Computing supplies none of them. Babbage described a programmable architecture but did not build it. Turing established the logic of a universal machine but did not turn his 1936 abstraction into the standard hardware design. Zuse, Flowers, Mauchly, Eckert, the ENIAC programmers, Williams, Kilburn, Wilkes and many others crossed different thresholds.

The computer is a bundle of properties assembled over time: automatic control, programmability, electronic speed, logical branching, stored

instructions, reliable memory, mass manufacture and usable software. These arose in different projects because different institutions were solving different problems. Even the celebrated individual usually stood inside a team, laboratory, procurement system and manufacturing base. Archives then magnify the people whose papers were preserved, whose employers promoted them and whose work escaped secrecy. Public memory inherits those inequalities.

The correction matters because it changes how innovation is recognised. A concept, a working prototype, a manufacturable device and a widely adopted system are separate achievements. Rewarding only the person nearest the patent or public launch hides the work that made the invention usable. It also teaches the wrong managerial lesson: breakthroughs are often failures of coordination before they are failures of imagination.

“Ada Lovelace wrote the first computer programme”

Lovelace's 1843 Note G contains a table describing how Babbage's Analytical Engine could calculate Bernoulli numbers. It is the first widely known published program for that unbuilt machine and a remarkable piece of explanation. The absolute title is less secure. Babbage had prepared earlier unpublished program-like notations, and the modern category of a computer program had not yet stabilised.

Reducing Lovelace to a priority claim also weakens her strongest contribution. She saw the Analytical Engine as a general symbolic machine rather than an elaborate calculator. If relations could be expressed formally, she argued, the engine might operate on symbols representing things such as music. She also denied that it could originate its own truths.

The honest version is better than the slogan. Lovelace helped articulate what programmability could mean before a general-purpose machine existed. Her notes linked an operational plan to a theory of the machine's wider possibilities, a rare combination even among people close to Babbage. Whether she was the first programmer depends on definitions and

surviving documents; her importance does not. A title contest asks who stood first in a line. Her notes ask the more important question of what kind of line it was.

“ENIAC was the first computer”

ENIAC was a decisive machine: electronic, digital, broadly programmable and powerful enough to prove that thousands of vacuum tubes could work together usefully. It was not the uncontested first computer because computer can describe several different combinations of features.

Zuse's Z3 ran automatically from a program in 1941 but used relays and kept instructions outside memory. Colossus was electronic and programmable by 1944 but built for codebreaking rather than general calculation. The Atanasoff-Berry Computer used electronic binary arithmetic earlier but was specialised and not program-controlled in the later sense. Manchester's Baby ran a program from electronic memory in 1948. EDSAC became an early practical stored-program service in 1949.

ENIAC's public unveiling, physical drama and post-war availability gave it an advantage in memory that secret or destroyed machines lacked. A later American patent case added another incentive to define the origin around particular machines. Calling ENIAC first without an adjective turns a useful history into a semantic contest. State the criterion and the dispute largely disappears. What remains is a richer account of several teams learning, under different constraints, how to make automatic calculation programmable, electronic and reusable.

“Computers use binary because they think in ones and zeros”

Computers do not think in a native numerical language. Engineers choose representations, and early machines used decimal, binary and other arrangements.

Binary became dominant because two-state components are easier to distinguish reliably. A circuit need not preserve one exact voltage for zero

and another exact voltage for one. It can treat broad ranges as low or high, leaving room for noise and variation. Boolean logic also maps cleanly onto networks of such switches. The output of one circuit can control the next, letting complex operations grow from repeatable units.

The ones and zeros are therefore a convention interpreted through physical systems. At lower levels there are voltages, charges, magnetic orientations or other states; at higher levels the same bit patterns may represent numbers, letters, colours or instructions. ENIAC's decimal design is enough to disprove inevitability. Binary won because it combined well with evolving components and logic. The correction prevents a category mistake: binary is an engineering representation, not evidence that the machine understands a hidden language.

“The computer revolution began in a garage”

Garages are real and overburdened. Apple used the Jobs family garage, hobby clubs mattered and small firms moved personal computing into forms large companies had ignored. None of them began from bare wood and inspiration.

The Altair depended on a commercially available Intel processor, electronic component suppliers, magazines, mail order and decades of publicly funded research. Apple depended on semiconductor manufacturing, university and corporate engineering cultures, venture capital and a market trained by calculators and kits. Graphical computing drew on ARPA-funded work at SRI and corporate research at Xerox PARC. Networks grew from state-backed projects and standards communities.

The image became persuasive because personal computing did mark a cultural break. Machines reached enthusiasts who could own, modify and sell them without a mainframe organisation's permission. That freedom was new; its technical foundations were not. The garage story mistakes the last assembly point for the origin. Small teams can combine existing technologies brilliantly, but the available parts, knowledge and customers were produced by institutions with long budgets. The correction matters

whenever public investment disappears from a story later told as private genius. It changes what societies think they should fund before a market exists.

“Programming was always a male profession”

Early programming inherited the status of calculation and clerical work. Women formed large parts of human computing pools, programmed ENIAC, worked on Colossus, wrote code for science and business, and helped develop compilers and languages. Employers often assumed programming was detailed routine work suitable for women, even while the work demanded mathematical judgement and invention.

As software grew in commercial importance, the occupation acquired prestige, career ladders and a masculine professional identity. Hiring tests, job classifications, management structures and cultural stereotypes changed who was recruited and promoted. The language shifted too: low-status coding could be treated as routine, while systems analysis and software engineering attracted authority. Women did not leave because programming naturally became harder. Institutions redefined the work and its status.

This is more than recovery of overlooked names. It shows that the apparent character of a technical occupation can be produced by pay, classification and authority. Today's gender pattern cannot be read backwards as evidence about who built the field. Nor can present prestige be used to infer the status attached to the same tasks when they were first performed.

“Moore's law is a law of nature”

Gordon Moore observed in 1965 that the number of components placed economically on leading integrated circuits had been rising rapidly and projected the trend for ten years. His early rate was roughly annual; in 1975 he revised the expectation towards a doubling about every two years.

Nothing in physics requires an industry to meet that schedule. Moore was discussing economically attractive component density, not promising that

every program would run twice as fast on a calendar. Progress depended on lithography, materials, design tools, factory investment, demand and coordinated expectations across suppliers. The forecast became a target. Firms planned products and research around it, while customers came to expect cheaper performance.

The pattern held impressively for decades, then became harder and costlier as dimensions shrank and power, heat and manufacturing complexity pressed back. Designers responded through parallelism, specialised accelerators, larger systems and new packaging as well as smaller features. Moore's law is best understood as a historical compact between engineering and economics. Calling it natural hides the institutions that made it true for so long.

Use It

Ask what is being represented

Computers act on representations, not on the world directly.

A census card turns a person into holes in predefined positions. A digital photograph turns light measurements into numbers. A bank balance turns an institutional claim into a record. The machine can process each efficiently because somebody has decided which features matter and how they will be encoded.

When a digital system feels objective, inspect that conversion first. What entered the representation? What was omitted? Which categories were available? Can two different real situations collapse into the same code? Many failures attributed to computation begin earlier, when a messy subject is forced into a neat data structure. Hollerith's card is useful here because its physical holes make the choice visible. Modern databases hide it better.

Follow the boundary between person and machine

Automation is rarely a clean replacement. It is a transfer of specific steps.

An abacus stores state while the operator supplies the method. A mechanical calculator performs arithmetic while the person chooses the sequence. ENIAC's electronics calculate quickly while programmers configure the data path. A compiler translates symbolic instructions while the programmer decides the intended result. A network service may answer instantly while depending on engineers, operators and exception handlers elsewhere.

Ask which part crossed the boundary. Which judgements remained human? Which work moved to another occupation or organisation? What checking appeared because the automated step can now fail at larger scale? The question is more informative than asking whether a machine took a job, because history shows that visible labour can shrink while surrounding labour grows.

Find the bottleneck that moved

Computing history advances by making one constraint cheap enough for another to dominate.

Human calculation was limited by labour and copying. Mechanical arithmetic shifted pressure towards control and precision engineering. Electronic switching made arithmetic fast enough that memory and programming became painful. Stored programs made reconfiguration easier, so programmer time, scheduling and software complexity mattered more. Integrated circuits made hardware cheaper and smaller, so compatibility, applications and access became decisive. Networks made remote resources abundant enough that security, bandwidth, energy and attention could become new constraints.

When evaluating a new technology, ask what complaint becomes louder if it succeeds. A tenfold improvement in one component rarely makes the whole system ten times better. Progress changes the location of scarcity.

Separate a breakthrough from a system

A prototype establishes possibility. A system makes the possibility repeatable for people who were not present at the demonstration.

Babbage's architecture could not overcome unfinished manufacturing and project organisation. The first transistor had to become a repeatable semiconductor process. An integrated circuit needed planar fabrication, testing and volume production. System/360 needed compatible hardware, software, peripherals, training and service. A personal computer needed useful applications, distribution and standards other companies could build around.

When a demonstration is presented as a revolution, look for the missing system. Can it be made repeatedly? Who pays for it? How is failure diagnosed? Can users keep previous investment? Are suppliers available? Does somebody else have permission to build compatible parts or software? History gives disproportionate attention to the first working object because it is easy to photograph. Adoption happens in the less glamorous layers.

Measure access, not only performance

The fastest machine in a locked room may matter less to daily life than a slower one people can reach.

Batch mainframes maximised expensive hardware use but made users wait. Time-sharing gave several people interactive access. Minicomputers put machines inside departments. Personal computers moved control to desks. Graphical interfaces lowered the knowledge required to act. Networks connected users to remote resources, then smartphones made that access continuous.

This lens changes how to compare technologies. Ask who can use the system, without whose permission, at what cost and with how much specialised knowledge. Then ask what new dependency arrives with the access. A cloud service may give a small organisation capabilities once reserved for a data centre while making that organisation dependent on a

remote provider. Capability and control are separate measures. Access also has a temporal dimension: a system may be cheap to enter and expensive to leave if files, software or skills become tied to its standards. The history of time-sharing and personal computing also warns against equating ownership with autonomy. A machine on your desk may still depend on proprietary software, compatible peripherals, remote updates or a network account. Access is a bundle of permissions, costs and dependencies, not a yes-or-no property.

Expect abundance to create more computing

Cheap computation does not satisfy a fixed quantity of demand.

Faster census tabulation made more classifications affordable. Electronic computers encouraged numerical modelling instead of completing a finite backlog of calculations. Compilers allowed larger software projects. Cheap storage produced larger collections. Personal computers generated demand for networks, support and applications. Faster networks encouraged heavier services.

The useful forecasting rule is therefore to avoid holding the workload constant. When an activity becomes cheaper, new users arrive, quality expectations rise and applications appear that previously made no economic sense. Efficiency can reduce the cost of one task while expanding the total system around it. This is why a forecast based only on today's uses is usually conservative: lower cost changes which uses are worth inventing.

The limits

This book follows the route that produced programmable digital computing. That selection necessarily compresses analogue computing, specialised calculators, non-Western counting traditions and many national industries. The early history is geographically broad, but the electronic chronology concentrates on Britain, Germany and the United States because the projects that became most influential in the later digital lineage were heavily concentrated there. Japan appears through Busicom and semiconductor markets; Britain reappears through ARM; production and use became far more global than this one-hour path can represent.

The book also stops at neighbouring boundaries. Algorithms owns computational methods in depth. Coding owns implementation practice. Chips owns semiconductor fabrication. The Internet owns network operation. Artificial intelligence is one important use of computing rather than the inevitable destination of its history.

No single model explains every change. Procedure, representation and bottlenecks clarify much, but materials, war, corporate strategy, state procurement, labour markets, consumer taste and accident also redirect the story. A clean technical genealogy is especially dangerous because successful standards make alternatives disappear from view.

The one thing to keep

Keep the moving bottleneck.

At every stage, ask what had to become cheap, reliable or legible before the next kind of computer could exist. The abacus helps because memory is scarce. Tables help because difficult calculation is scarce. Punched cards help because repeatable control and sortable records are scarce. Electronics helps because switching speed is scarce. Stored programs help because reconfiguration is scarce. Integrated circuits help because reliable components and connections are scarce. Personal and mobile systems help because access is scarce.

Then notice what becomes scarce next.

That habit protects against the most persistent mistake in technology history: treating the newest component as the whole system. A processor can be astonishing while software is unusable. A network can be fast while trust is weak. An automated service can be cheap while human review becomes the limiting resource. Solving one problem does not finish computation. It moves the frontier. That is a better way to read current claims about computing too: ask which old constraint has fallen, which new one is now visible and whether the surrounding system can absorb the change.

The abacus and the chip therefore belong in the same history for a deeper reason than chronology. Both are arrangements of physical states that become useful only inside a human system of meaning and procedure. What changed over thousands of years was how much of that procedure could be made explicit, stored, switched, copied and hidden.

The modern computer is the accumulated result of those transfers. Its apparent simplicity is evidence of how many bottlenecks have been buried underneath it.

Terms

Abacus. A frame or surface on which counters represent quantities by position. It stores the changing state of a calculation while a trained operator supplies the procedure and interprets the result.

Algorithm. A finite, reproducible procedure for solving a class of problems. The word derives from the Latinised name of al-Khwarizmi and predates electronic computers by centuries, preserving the older link between calculation and written method.

Analogue computer. A machine that represents a problem through continuously varying physical quantities, such as rotation or voltage. Analogue systems model relationships directly, usually produce approximate results and can simulate several changing quantities at once.

Binary. A number system using two digits, zero and one. It became dominant in digital electronics because two broad physical states can be distinguished, restored and combined reliably despite noise.

Bit. A binary digit, the smallest conventional unit of digital information. A bit records one choice between two states but gains meaning only through an agreed representation.

Boolean algebra. George Boole's formal system for combining true and false values through operations such as AND, OR and NOT. Shannon showed how it could describe switching circuits.

Byte. A fixed group of bits treated as one unit, commonly eight since the System/360 era. Bytes became basic units for characters, memory capacity and data transfer.

Computer. Originally a person employed to calculate. During the mid-twentieth century, the occupational name shifted to machines that performed growing parts of the same organised procedure.

Compiler. Software that translates instructions written in a higher-level language into a form a machine can execute. Compilers moved translation effort from programmers into reusable programs.

Central processing unit. The part of a computer that interprets instructions and performs arithmetic and control operations. The CPU occupied cabinets before the microprocessor compressed it onto a chip.

Digital computer. A machine that represents information using discrete values. Discreteness allows states to be copied and restored within tolerances, supporting reliable storage and logical control.

Difference Engine. Babbage's special-purpose mechanical design for calculating and printing mathematical tables by finite differences. It clarifies the distinction between automated calculation and general programmability.

General-purpose computer. A machine whose behaviour can be changed through instructions so it can address many classes of problem. Generality

resides in the combination of architecture and program.

Graphical user interface. An interface using visual objects, windows and pointing devices rather than relying mainly on typed commands. It lowers memory demands while hiding more of the system.

Integrated circuit. A circuit whose components and connections are formed together on semiconductor material. Integration reduced size, power, wiring failures and the cost of reproducing complex logic.

Machine code. The numerical instructions executed directly by a processor. Machine code depends on a particular architecture, making it efficient for hardware and difficult for humans to write at scale.

Mainframe. A large, centrally managed computer built for high-volume organisational work, reliability and many users. Mainframes became systems of processors, storage, peripherals, software and service.

Memory. The mechanisms that retain data or instructions for later access. Early computers used delay lines, cathode-ray tubes, drums and magnetic cores before semiconductor memory became dominant.

Microprocessor. A central processing unit implemented on one integrated circuit, or a small set in early designs. It made general computation cheap enough to embed in other products.

Minicomputer. A smaller, cheaper general-purpose computer sold from the 1960s for laboratories, factories and departments. Minicomputers brought users closer to machines previously controlled by central operations groups.

Operating system. Software that manages hardware resources and supplies common services to programs. It turns a physical machine into a controlled environment in which many programs can run.

Personal computer. A general-purpose computer designed primarily for direct use by one person. The label covers disputed early machines but became a mass-market category during the late 1970s, changing ownership and expectations of immediate access.

Programme. A sequence of instructions expressed in a form that directs a computer. British usage retains programme for this general sense, while program is also standard in technical contexts.

Punched card. A stiff card whose hole positions encode data or instructions. Cards controlled looms, recorded census information and carried programs into computers well into the twentieth century, linking textile control, bureaucracy and software history.

Relay. An electrically controlled switch with moving contacts. Relays made automatic logical and arithmetic circuits possible but were slower and more mechanical than later electronic switching devices.

Semiconductor. A material whose electrical behaviour can be controlled between conducting and insulating states. Purified semiconductor materials made transistors, integrated circuits and modern digital electronics possible.

Stored programme. An arrangement in which instructions reside in memory in machine-readable form, often beside data. It allows rapid reprogramming and makes one physical machine serve many purposes.

Transistor. A semiconductor device that can amplify or switch signals. Transistors replaced many vacuum-tube functions with lower power, smaller size and greater potential reliability and integration.

Turing machine. Alan Turing's abstract model of a rule-following machine reading and writing symbols on a tape. A universal version established the logical possibility of general programmable computation.

von Neumann architecture. The widely adopted stored-program arrangement described in von Neumann's 1945 EDVAC report: memory, arithmetic, control and input-output. The name understates its collaborative development and the influence gained through the report's wide circulation.

Go Deeper

The modern overview

Thomas Haigh and Paul E. Ceruzzi, *A New History of Modern Computing* (MIT Press, 2021). This is the best next step for the whole field. It begins with the electronic computer and follows changing uses through mainframes, personal machines, networks, games, smartphones and embedded systems. Its strength is refusing to treat hardware improvement as the whole story: users, software, business models and institutions keep changing what a computer is. At more than five hundred pages, it is a serious history rather than a quick survey, but its organisation by successive computing cultures makes the scale manageable. Keep a pencil nearby: the argument is stronger than the anecdote count.

The primary voice

Charles Babbage, *Passages from the Life of a Philosopher* (Longman, Green, Longman, Roberts, & Green, 1864). Read this for Babbage in his own defence: ambitious, funny, combative and incapable of making a short grievance. The chapters on the calculating engines show how he understood their purpose, the government relationship and the practical obstacles. Treat it as testimony, not verdict. Babbage is an interested witness to his own failures and gives Ada Lovelace less space than a modern reader may expect. The book is freely available in scanned editions and rewards selective reading, especially beside images of the surviving engine parts.

The labour history

Nathan L. Ensmenger, *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise* (MIT Press, 2010). Hardware histories often let programmers appear only when a machine needs code. Ensmenger reverses the view and follows programming as an occupation: recruitment, gender, status, professional claims, management anxiety and the repeated attempt to turn uncertain software work into engineering. It is centred on the United States and the post-war period, which is a strength rather than a concealed limit. Read it to understand why software difficulty is organisational as well as technical, and why occupational prestige changed who counted as an expert.

The wider interpretation

James Gleick, *The Information: A History, a Theory, a Flood* (Pantheon, 2011). Gleick ranges beyond computers through writing, telegraphy, codes, logic, information theory, biology and the modern flood of messages. It is the most inviting of these four and the least narrowly focused on hardware. Read it after the basic chronology, when Shannon's work and the idea of representing unlike things as information have begun to connect. The sweep sometimes outruns the institutional detail, but that is the bargain: it shows why computing belongs inside a larger history of storing, transmitting and transforming symbols. It is best read slowly, since its connections matter more than its chronology.

Notes and Sources

The notes follow the order of the book. Dates and priority claims were checked against primary documents, institutional histories and specialist scholarship. Web sources were last verified on 10 August 2026.

The Whole Thing in One Page and Why You Should Care

Computer as an occupation. The older occupational meaning of *computer* and the organisation of calculation through human teams are documented in the NASA history of Langley's computer pools and NASA JPL's account, When Computers Were Human. Langley's first central pool began with five women in 1935; by 1946 the organisation had trained about 400 women and distributed them across the laboratory. The manuscript uses this history to establish organised procedure as the predecessor of machine computation, not to claim that all human computers worked under one model.

No single first computer. The book separates automatic, program-controlled, electronic, general-purpose and stored-program thresholds. This follows the method used in Thomas Haigh, Mark Priestley and Crispin Rope's *ENIAC in Action* and the Computer History Museum's discussion of the never-ending quest for firsts. A priority claim is retained only with its qualifying property.

Notes on the central model

Computation Was Organised Labour

Tables, instruments and human calculation. The account draws on the long history of mathematical tables, observatory work, navigation, insurance, ballistics and scientific laboratories. The recurring system of workers, printed instructions, intermediate records, checking and publication is treated as an organisational structure rather than an early electronic architecture. NASA's Langley and JPL histories support the twentieth-century examples and the movement of women from hand calculation into programming.

Abacus and mechanical calculators. The abacus is described as a device for representing and transforming the state of a calculation under a trained operator's control. Pascal's machine and Leibniz's stepped reckoner are used to distinguish automatic arithmetic operations from programmable

control. Michael R. Williams's *A History of Computing Technology* supplied the main modern synthesis for this pre-electronic line, checked against museum collections and primary material. The manuscript avoids claiming a single origin for the abacus or treating the European mechanical line as the whole history of counting.

Control Becomes Portable

Jacquard cards. The Computer History Museum's Jacquard account documents the use of punched-card chains to control loom patterns and the influence of this removable control medium on Babbage. The cards selected actions while power, material and mechanical operation remained elsewhere in the loom.

Difference Engine and Analytical Engine. The distinction between Babbage's fixed-purpose table engine and his proposed general-purpose Analytical Engine follows the Computer History Museum's Babbage history, Charles Babbage's *Passages from the Life of a Philosopher* and the Science Museum's Babbage archive. Store, mill, operation cards, variable cards, repetition and conditional control are described in modern terms to clarify the design, without implying that a completed nineteenth-century machine existed.

Babbage's failure to deliver. The manuscript attributes the failed project to a combination of precision-engineering cost, changing designs, disputes with Joseph Clement, government fatigue and weak project control. This is narrower than a judgement that Victorian engineering was incapable of building any part of the design. The Science Museum's account of Difference Engine No. 2 records the modern construction from Babbage's plans, completed with its printing mechanism in 2002. That reconstruction supports mechanical feasibility of the later Difference Engine design, not completion of the Analytical Engine.

Ada Lovelace. Lovelace's 1843 translation and notes are read in the original publication and against the Computer History Museum's assessment. Note G contains the first widely known published stepwise plan

for the proposed Engine's Bernoulli-number calculation. Babbage's earlier unpublished notations make the absolute title of first programmer definition-dependent. Lovelace's clearer contribution is her account of a rule-governed symbolic machine whose numbers might represent material beyond quantity.

Hollerith and the census. The card reader, mercury cups, counters and sorting process follow the United States Census Bureau's description of the Hollerith machine. The book says that the system accelerated key stages and made repeated classification practical, rather than repeating the loose claim that the 1890 census was entirely completed in a few weeks. Hollerith's company history is traced through the 1911 Computing-Tabulating-Recording merger and the adoption of the IBM name in 1924.

Logic Becomes Switchable

Boole and Shannon. George Boole's algebra supplied a formal treatment of propositions through operations corresponding to AND, OR and NOT. Claude Shannon's 1937 MIT thesis, published as A Symbolic Analysis of Relay and Switching Circuits, showed how Boolean methods could analyse and simplify relay circuits and how circuits could embody logical relations. The book does not claim that Shannon alone invented digital logic; it identifies the thesis as the decisive bridge between a formal algebra and switching design.

Binary. Binary is presented as an engineering choice strengthened by the tolerance of two-state components and the composability of switching logic. Babbage's decimal engines and ENIAC's decimal accumulators demonstrate that digital computation need not be binary. The explanation concerns logical state restoration and noise margins, not a claim that every physical computer contains literal ones and zeros.

Z3. The Computer History Museum's 1941 timeline records the Z3's 2,300 relays, floating-point binary arithmetic and punched-film control. The manuscript calls it automatic and program-controlled while noting that its

program was external to memory and that its practical branching facilities did not match later general-purpose stored-program machines.

War Pays for Several Electronic Breaks

Colossus. Bletchley Park's material on Colossus and its contextual histories support the dates, purpose and construction of ten machines by the war's end. Colossus was electronic and programmable through switches and plugs, but it was designed for a specialised cryptanalytic task and did not store a general program in memory. Wartime secrecy and post-war destruction explain much of its delayed place in public history.

ENIAC. The institutional account from Penn Today, the Computer History Museum ENIAC exhibit and *ENIAC in Action* support the machine's scale, electronic decimal design, artillery purpose and programming practice. The six original programmers came from the Army's human-computing workforce and learned the machine through diagrams, panels, cables and switches. ENIAC is described as a decisive general-purpose programmable electronic machine, not as the uncontested first computer.

Memory Makes the General Machine Practical

Turing's universal machine. Alan Turing's 1936 paper, published in 1937, established an abstract account of effective procedure and a universal machine able to imitate any machine in the model when supplied with an encoded description. This is a logical result about computability, not a blueprint for the standard electronic computer.

EDVAC report and collaborative credit. John von Neumann's 1945 *First Draft of a Report on the EDVAC* circulated a stored-program design widely. The manuscript retains the conventional term *von Neumann architecture* while stating that the report condensed collaborative work involving the Moore School group, including J. Presper Eckert and John Mauchly, and that the naming overstates individual authorship.

Manchester Baby. The University of Manchester's account records the Small-Scale Experimental Machine's first run on 21 June 1948 and its role in

testing Williams-tube memory. It is identified as the first machine to run a program electronically stored in its memory, with the qualifier that it was an experimental machine rather than a regular computing service.

EDSAC. Cambridge's Computer Laboratory history records the first logged EDSAC program on 6 May 1949, regular service to researchers and the early development of subroutines and a program library. EDSAC is presented as an early practical stored-program service, not as the first machine to cross every stored-program threshold.

ACE. The National Physical Laboratory's Turing history supports Turing's 1946 ACE plan and the Pilot ACE's later operation. The account distinguishes an influential, compact design from the bureaucratic and engineering process needed to build it.

Scale Requires Components, Compatibility and Capital

Transistor. Nokia Bell Labs' history of the first working transistor dates the point-contact device to December 1947 and identifies John Bardeen, Walter Brattain and William Shockley. The book treats invention as the beginning of a manufacturing transition. Vacuum tubes remained in use while materials, production and circuit design matured.

Integrated circuit. Jack Kilby's 1958 demonstration and Robert Noyce's 1959 monolithic concept are separated. The Computer History Museum's Silicon Engine and its account of Fairchild's planar process support the distinction: Kilby demonstrated integration; Jean Hoerni's planar process and Noyce's deposited interconnections supplied a practical route to monolithic mass manufacture.

Early demand. Military systems and the Apollo programme bought early integrated circuits because weight and reliability justified high prices. Gordon Moore's original 1965 article also described integrated electronics as established in military systems and increasingly attractive in computers and instruments. The manuscript does not imply that public procurement was the sole cause of semiconductor progress.

Compilers and programming languages. Grace Hopper's A-0 system and FLOW-MATIC work are used to show translation and reusable routines moving into software. John Backus's FORTRAN team is supported by IBM's histories of FORTRAN and John Backus. COBOL is credited to an industry and United States government committee rather than to one inventor.

System/360. IBM's System/360 history supports the 1964 launch, common architecture, compatibility across a family, 8-bit byte and scale of the corporate commitment. IBM's own celebratory account is balanced against Haigh and Ceruzzi's wider treatment of software, migration, delays and customer systems.

Access Becomes the Next Bottleneck

Microprocessor. Intel's history of the 4004 supports the Busicom contract, the four-chip family and the contributions of Ted Hoff, Federico Faggin, Stan Mazor and Masatoshi Shima. The manuscript calls the 4004 the first commercially available general-purpose microprocessor in its historical setting, while avoiding a lone-inventor account.

Minicomputers and personal machines. The Computer History Museum computer timeline supports the PDP-8's commercial importance and the sequence from minicomputers to microprocessor systems. The personal-computer narrative is based on Haigh and Ceruzzi, with the Altair, Homebrew Computer Club, Apple II, VisiCalc, Commodore PET, TRS-80 and IBM PC selected because each changed access, use or compatibility. They are not presented as an exhaustive list of early personal computers.

Interactive interface. SRI's history of the 1968 Engelbart demonstration supports the mouse, linked text, collaborative editing, windows and interactive computing shown there. Xerox Alto, Lisa and Macintosh are treated as successive research and commercial systems rather than a simple line of one company copying another.

Networks and the Web. DARPA's ARPANET history supports the development of TCP/IP and the January 1983 transition. Packet switching also had independent British work at the National Physical Laboratory, so

ARPANET is not described as its sole origin. CERN's history of the birth of the Web supports Berners-Lee's March 1989 proposal and the browser, server and first site developed by the end of 1990. Network operation remains proportionate because *The Internet in a Hurry* owns the deeper treatment.

ARM and mobile computing. Arm's official history and its architecture histories support the Acorn origins, Sophie Wilson and Steve Furber's ARM1 design, first silicon in 1985, the 1990 joint venture and the later licensing model. The book uses ARM to explain low-power processor design and licensed cores without claiming that one architecture alone created mobile computing. Apple's original iPhone announcement supports the 2007 interface and communications claims; Apple's App Store history supports the launch of native third-party distribution in July 2008.

Moore's law. Gordon Moore's 1965 article described the rising complexity that minimised component cost and projected the trend forward. Intel's Moore's law archive records the original roughly annual rate and Moore's 1975 revision towards about two years. The manuscript treats the pattern as an empirical forecast that became an industry target, not a physical law or a promise that all software performance doubles on schedule.

Notes on the seven corrections

The seven corrections draw on the same sources but perform different jobs. The first-computer correction separates criteria; the Lovelace correction distinguishes priority from interpretation; the ENIAC correction explains why public visibility and secrecy distorted memory; the binary correction distinguishes representation from understanding; the garage correction restores public laboratories, corporate research, supply chains and standards; the gender correction follows the changing status and classification of programming; the Moore correction returns a supposed natural law to engineering and economics.

For programming labour and professionalisation, the main modern sources are Nathan Ensmenger's *The Computer Boys Take Over*, Janet Abbate's

Recoding Gender and Mar Hicks's *Programmed Inequality*. Hicks supplies the strongest British case, while Abbate and Ensmenger help separate changing occupational status from any universal national sequence. NASA, Penn and Computer History Museum archives supply the earlier human-computing and ENIAC evidence. The book therefore does not claim a single path by which programming became male; labour markets, state systems and institutions differed.

Use It and Terms

The lenses in *Use It* are interpretive deductions from the history rather than claims that a named scholar proposed a six-part method. They are constrained by the cases in the book: task displacement, general-purpose architecture, moving bottlenecks, system-building, interface layers and induced demand. The limits section states the selection explicitly. The route privileges the line leading to programmable digital systems and cannot give equal treatment to analogue computing, all counting traditions, every national industry or semiconductor fabrication.

The glossary uses contemporary conventional meanings while preserving historically important distinctions. *Programme* follows British general usage; *program* is retained where it is part of a technical term or source title. *Von Neumann architecture* is defined because readers will encounter it, with the credit limitation attached rather than hidden.

Bibliography

Primary and contemporary sources

Babbage, Charles. *Passages from the Life of a Philosopher*. London: Longman, Green, Longman, Roberts, & Green, 1864.

Boole, George. *An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities*. London: Walton and Maberly, 1854.

Menabrea, Luigi Federico. “Sketch of the Analytical Engine Invented by

Charles Babbage." Translated by Ada Augusta, Countess of Lovelace, with notes. In *Scientific Memoirs*, vol. 3, edited by Richard Taylor. London: Richard and John E. Taylor, 1843.

Moore, Gordon E. "Cramming More Components onto Integrated Circuits." *Electronics* 38, no. 8, 19 April 1965: 114-117.

Shannon, Claude E. "A Symbolic Analysis of Relay and Switching Circuits." *Transactions of the American Institute of Electrical Engineers* 57, no. 12, 1938: 713-723.

Turing, Alan M. "On Computable Numbers, with an Application to the Entscheidungsproblem." *Proceedings of the London Mathematical Society*, series 2, 42, 1937: 230-265.

von Neumann, John. *First Draft of a Report on the EDVAC*. Philadelphia: Moore School of Electrical Engineering, University of Pennsylvania, 1945.

Modern works

Abbate, Janet. *Recoding Gender: Women's Changing Participation in Computing*. Cambridge, MA: MIT Press, 2012.

Ensmenger, Nathan L. *The Computer Boys Take Over: Computers, Programmers, and the Politics of Technical Expertise*. Cambridge, MA: MIT Press, 2010.

Gleick, James. *The Information: A History, a Theory, a Flood*. New York: Pantheon Books, 2011.

Haigh, Thomas, and Paul E. Ceruzzi. *A New History of Modern Computing*. Cambridge, MA: MIT Press, 2021.

Haigh, Thomas, Mark Priestley, and Crispin Rope. *ENIAC in Action: Making and Remaking the Modern Computer*. Cambridge, MA: MIT Press, 2016.

Hicks, Mar. *Programmed Inequality: How Britain Discarded Women Technologists and Lost Its Edge in Computing*. Cambridge, MA: MIT Press, 2017.

Williams, Michael R. *A History of Computing Technology*. 2nd ed. Los Alamitos, CA: IEEE Computer Society Press, 1997.

Institutional, archival and technical sources

Arm. “A Brief History of Arm” and “The Official History of Arm.” Arm Developer and Arm Newsroom. Accessed 10 August 2026.

Apple. Original iPhone and App Store launch histories. Apple Newsroom. Accessed 10 August 2026.

Bletchley Park. “75 Years Since Colossus Arrived at Bletchley” and Colossus historical materials. Accessed 10 August 2026.

CERN. “The Birth of the World Wide Web.” CERN historical timeline. Accessed 10 August 2026.

Computer History Museum. *Babbage Engine* exhibit; *Revolution: The First 2000 Years of Computing*; *The Silicon Engine*; *Timeline of Computer History*; oral histories and collections relating to ENIAC and Jean Jennings Bartik. Accessed 10 August 2026.

DARPA. “ARPANET.” Accessed 10 August 2026.

IBM. Histories of FORTRAN, John Backus, System/360, the IBM Personal Computer and women in computing. Accessed 10 August 2026.

Intel. Histories of the 4004 and Moore's law; digital facsimile of Gordon Moore's 1965 article. Accessed 10 August 2026.

NASA. “When the Computer Wore a Skirt: Langley's Computers, 1935-1970.” NASA History. Accessed 10 August 2026.

NASA Jet Propulsion Laboratory. “When Computers Were Human” and Women at JPL historical materials. Accessed 10 August 2026.

National Physical Laboratory. Alan Turing and Automatic Computing Engine historical materials. Accessed 10 August 2026.

Nokia Bell Labs. “1956 Nobel Prize in Physics,” including the history of the

first working transistor. Accessed 10 August 2026.

Penn Today and Penn Engineering. ENIAC historical materials and accounts of its original programmers. Accessed 10 August 2026.

Science Museum, London. Babbage Papers and “Charles Babbage's Difference Engines and the Science Museum.” Accessed 10 August 2026.

SRI International. “1968 ‘Mother of All Demos’ Forecasted Much of the Technology We Use Every Day.” Accessed 10 August 2026.

United States Census Bureau. “The Hollerith Machine.” Accessed 10 August 2026.

University of Cambridge Computer Laboratory. EDSAC histories, chronology, program records and anniversary archive. Accessed 10 August 2026.

University of Manchester. Manchester Baby historical materials. Accessed 10 August 2026.