History of Algorithms and Algorithmic Thinking

This section talks about the history of algorithms and the history of algorithmic thinking. Well, the topic of this course is the history of computer science, but it's not very easy to define what is the time period where we should start with the history of computer science subject. If we consider that computer science is about computing machines, then we can say that the first general purpose computational machines were Charles Babbage's differential engine and the analytical engine. They were both general purpose computational machines created by the British scientist Charles Babbage in 1800s. By the way, the first program in history was written for the analytical engine. Ada Lovelace was the first programmer in history and she wrote a program for computing the Bernoulli numbers for this analytical engine. The thing about the first program like the thing about any first program written in a new programming language or a new technology is the fact that it didn't run, of course, but not for the same reasons. First programs written in a new programming language or a new technology usually don't run when you first try to execute them because the program has a flaw, a runtime error or a compilation error. The reason that Ada's program didn't work was because the analytical engine machine was never actually built, at least in Ada Lovelace and Charles Babbage's lifetime; the same is true for the differential engine. The reason Charles Babbage tried to build this machines and at the same time his idea was that before the years 1800s, public workers in the ministries and also sailors, lawyers and workers in commerce and various other domains in United Kingdom, they all used various printed tables in order to compute all sorts of math formulae. For example, if a government representative has to compute the taxes for a specific person, he/she will probably use some complicated math formula in order to compute the amount of money that person owes to the government. In the same way, sailors on ships need to compute complicated coordinate values depending on the position of the moon and on the stars or the position of the sun to determine where to go and so on. Before 1800 everybody would use some complicated math textbooks containing all sorts of tables' values for a specific math formula. These formula textbooks were similar to what I had when i was a child in school – there were these math copybooks that had on the back the multiplication tables of numbers from 1 to 10. We, as kids, had to learn the multiplication tables by heart using those copybooks. If someone doesn't know these multiplication values by heart, he/she can look them up un the back cover of those math copybooks. It was the same in that period, so before 1800s, in the whole Europe, including Great Britain, people would have complicated textbooks with tables that contain the values of the exponential function or the values of some common polynomial functions and the person who needs to compute a specific value, would have to just look it up into in that textbook (see Fig. 1.1). This is how it was done before 1800.

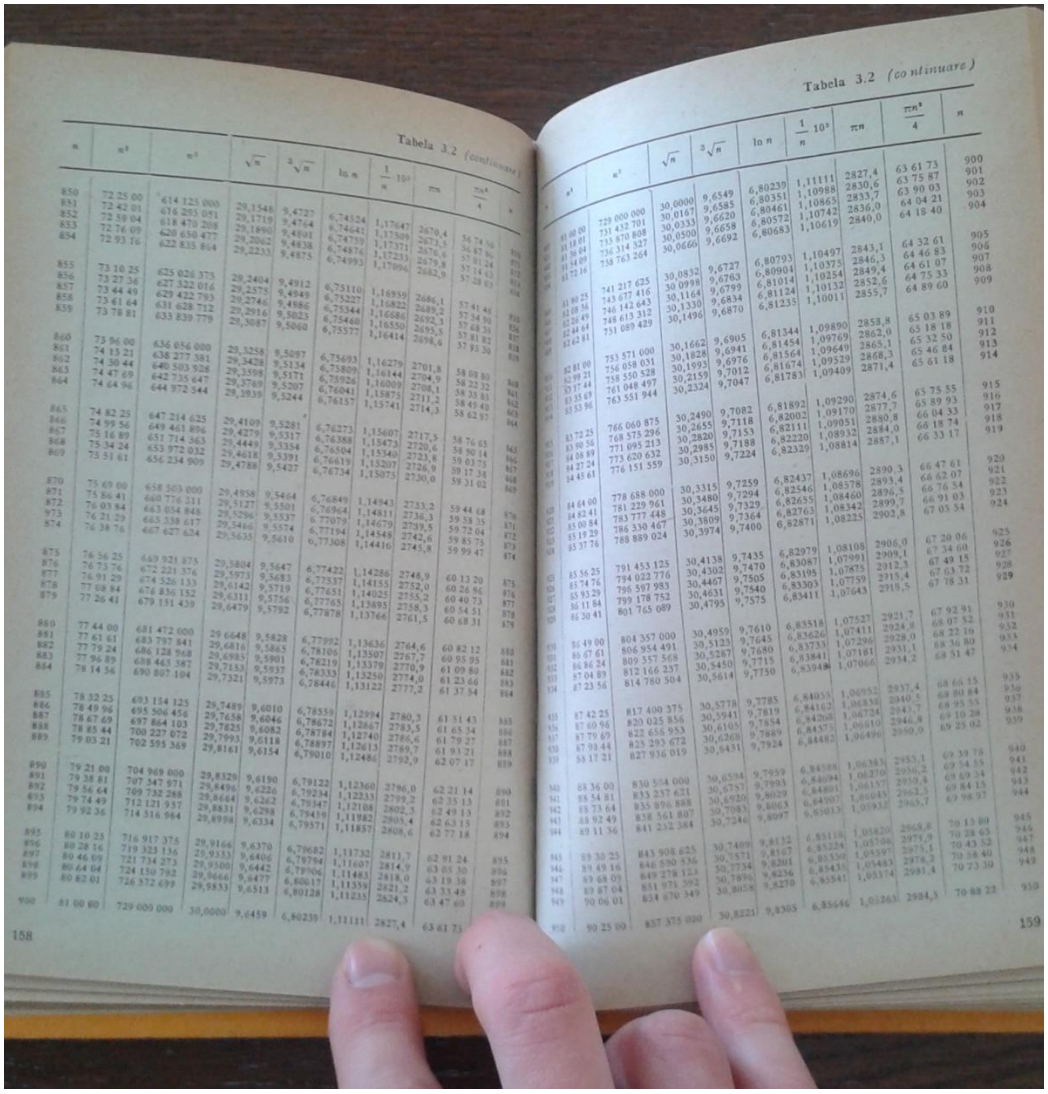


Fig 1.1 A photo of a book with functional values tables

The problem was that the value for a function you would find in one textbook would differ slightly from the value of the same function in another textbook. Charles Babbage had the idea that if he could create a mechanical computer (there were no electricity then) that would compute these values accurate and faster, such a computational machine would be quite a huge advantage for the whole government and it would allow him to compute and print textbooks with consistent values. He designed such a machine called the differential engine, he then build a small prototype (not a full scale machine), but of course he wanted funding from the British government in order to build a full scale differential engine. I am going to do a short summary right now because I will talk about these two machines, the differential engine and the analytical engine, in depth later on. So, Charles Babbage asked for funding from the British government and he said that he would need about 1000 pounds and he would create the differential machine in one or two years. The funding continued for about 17 years and he got 17000 pounds in total and the computer was still far from being ready. Actually, the differential engine was never built during Charles Babbage's lifetime, but it was built around 1990, in modern times for a museum. The machine was built for the museum according to the specifications of Charles Babbage and it actually worked. The reason that the differential engine was not built in the 1800s, during Charles Babbage's lifetime was on one hand due to Charles Babbage's personality. He was an egocentric person, he would argue with everyone, he was quite an unfriendly person, so a person that's very hard to work with. And that's one reason he couldn't put together a team of technicians to work on the differential engine. The second reason differential engine did not get completed was that the industrial equipments and tools available in that period were not very precise and they couldn't cut iron wheels with a very precise tooth size as required by Charles Babbage's design. Hence the available industrial equipments did not support building such a complex and precise machinery. So these were the reasons that the differential engine was not built during Charles Babbage lifetime. He tried to build this engine between 1822 and 1842 and when he saw that it's not getting to the end of it, he abandoned the project and imagined a new machine called the analytical engine which he still tried to build, but he never completed it either. These two machines were the first general purpose computational machines (especially the analytical engine) meaning that besides computing the basic 4 operations, addition, subtraction, multiplication and division, these machines could also evaluate more complex functions like exponential, logarithm, trigonometric and polynomial, and they even had conditional execution (IF), loops, program statements. You can see in Fig. 1.2 and Fig. 1.3 the differential engine and, respectively, the analytical engine, recreated in 1990 for museums. They are not simple copy models, they actually work.

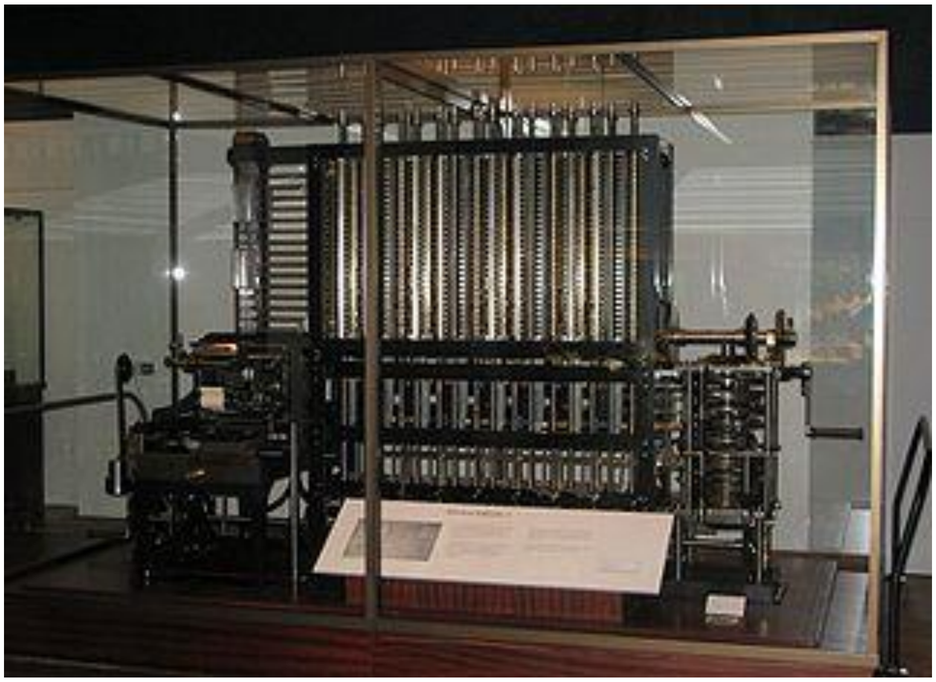


Fig. 1.2 Copy model of Differential Engine,

Charles Babbage, 1822-1842[1]

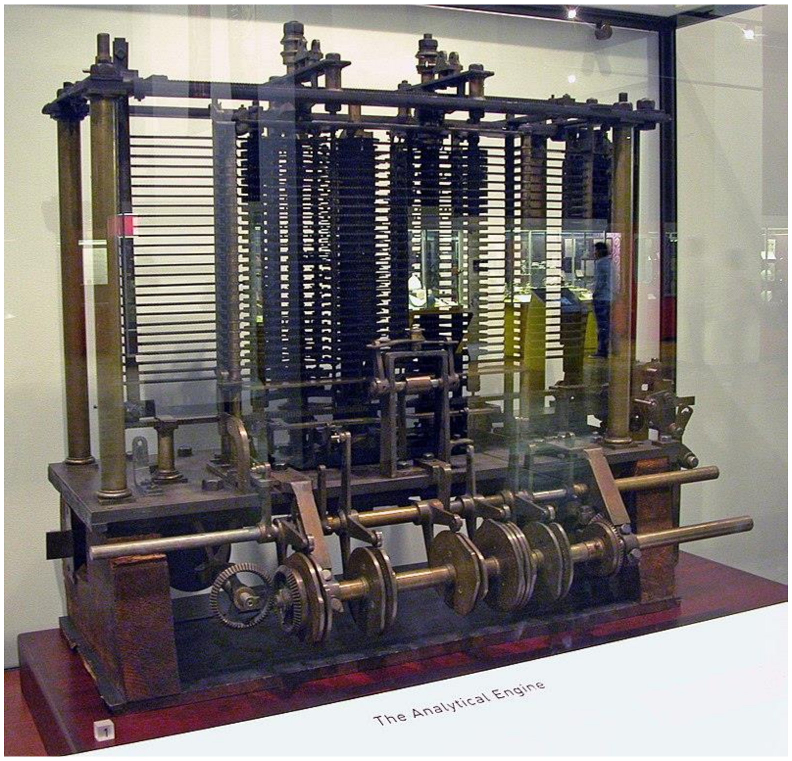


Fig. 1.3 Copy model of Analytical Engine,
Charles Babbage, 1837[2]

Before these two machines were imagined in the 1800s, there were other computational machines available in the society, but those computational machines were equivalent to the so-called pocket calculators meaning they would compute only addition, subtractions, multiplications and divisions. There were some of them that still computed some polynomials, but that's about it. The differential engine and the analytical engine were the first general purpose computational machines because they could compute addition, subtraction, multiplications and divisions, but besides these four primary operations, they could also compute/approximate functions like polynomial, trigonometrics, logarithms and also the analytical engine also had conditional statements like IF and loop cycles (i.e. it could execute the same instruction repeatedly), it had a rudimentary programming language. You could even write programs for it. Actually, Ada Lovely wrote a program for it on punch cards but we will talk about those later. So that is why I was saying that those two machines were the first general purpose computational machines in order to separate them from previous existing computational machines which were very simple and would normally compute additions, abstractions, divisions and multiplication just like a pocket calculator would do.

[1] http://www.wikipedia.org
[2] http://www.wikipedia.org

Right, so we can say that the history of computer science started with those first general purpose computational machines, but before we talk about the history of computer science, we should know what *computer science* is. The term computer science (in Romanian that would be “Informatica”) first appeared in an article in the Communications of the ACM in 1959 which is an old journal still edited today by the *Association for Computing Machinery* (ACM) which is the largest professional association in Computer Science; you can also become member of the Association for Computing Machinery if you pay an annual fee. Nowadays, *computer science* is quite a large domain. We can try to separate this large computer science domain into subdomains and we would have something like this. We have first theoretical computer science which includes data structures, algorithms, computation theory, information and coding theory and programming languages. Computer science also includes computer system theory which includes topics more related to the hardware architecture of the computer system like computer architecture and engineering, operating systems, computer networks, databases, concurrent and distributed systems, security. A third subdomain of computer science would be computer applications which includes applications that rely on the first two categories, like artificial intelligence, graphics and audio-video processing, computer vision etc. Lastly, we have the software engineering subdomain which is the science of writing programs and reusing code. The above taxonomy is just an example of a taxonomy in computer science and there are two taxonomies of computer science published by the ACM and if you look at those, you see that those taxonomies include a lot of sub-domains of computer science, not only these four sub-domains.
If, on the other hand, we look at the problem even from a higher ground, we can say that computer science is generally about *hardware* and about *software* and if we talk about hardware, we might say that the history of computer science began with those two machines, the differential engine and analytical engine. But even this is not entirely correct because there were computational engines, well not general purpose computational engines, but there were mechanical computers (equivalent to packet computers) before these two machines. Computational machines like mechanical calculators appeared first in the 1600s, but they even existed in rudimentary forms (e.g. abacus) in ancient history, 2700 BC. If we move to the software part, then programs and software and programming actually began around 1950s when the first electrical computers were built and the first programming languages were developed and the first pieces of software like the operating system software was built. But that is if we consider that software is programming, but software is also algorithmic thinking and algorithms because a program is just an algorithm expressed in a programming language. So an algorithm is very is strongly linked to the term software and actually the term *algorithm* was first invented by an arab mathematician named Muhammad ibn Musa al-Khwarizmi, the father of algebra, in a book written in 820 and translated into latin in the 12th century. *Algorithm* is derived from al-Khwarizmi’s name and refers to arithmetic techniques with hindu numerals. Hindu numerals means indian numbers, i.e. the numbers that we use today throughout the world. They are actually arabic numbers, but the arabs adopted the numbers from the Indians and that's why they are also called hindu numbers.

The modern definition of algorithm is the following: a sequence of steps necessary in order to solve a specific problem and, in addition, a set of data on which the operational steps operate. Actually this idea of decomposing a problem into several subproblems and solve them sequentially, solving a problem in steps (i.e. algorithmic thinking), it's very old in human history. It can eventually be traced back to ancient history (Euclid's algorithm – 300BC for computing the greatest common divisor, Eratostene's sieve – an algorithm for computing the prime numbers which are smaller than a specific value) and even further back in time – as part of mathematics. So, algorithmic thinking is very old in human civilization and we can encounter algorithmic thinking starting from ancient history up to today. We can consider algorithmic thinking as rudimentary computer science. The roots of algorithmic thinking in ancient history can be traced as part of the mathematical science. In ancient history, all the sciences were more or less merged together and a scientist would generally be a mathematician, an astronomer, a physicist and sometimes a doctor and a philosopher. In the remaining of this section we will talk about algorithmic roots of various civilizations in ancient history.

The Babylonian

We will start with the Babylonian civilization. They lived in Mesopotamia which is a land on which currently Turkey and Syria lies on. It was a fertile plain between two rivers, Tigris and Euphrates. Their civilization lasted between 2000BC and 600BC. They were later conquered by Sumerians and then by Akkadians. This old civilization had important advances in sciences, especially in mathematics, but they were famous for their writing which is called cuneiform writing because their letters and their digits resemble nails, i.e. they have this wedge-shaped symbols. The cuneiform writing was also later adopted by Sumerians. We know of cuneiform writing because the Babylonians wrote a lot of clay tablets like the one depicted in Fig. 1.4.

Fig. 1.4 Babylonian clay tablet[3]

[3] http://www.wikipedia.org

The Babylonians would take a piece of wet, arrange it into a flat surface, draw numbers and digits and symbols and drawing on it using some sharp object like a pencil or a needle and then they would leave this clay tablet in the sun to dry out and after it had hardened, they would use it for storing information. They would write texts on these clay tablets, but also numbers and mathematical results. I have already told you that there were many books with mathematical tables before 1800, i.e. before Charles Babbage designed the differential engine and the analytical engine, instead of those books written on paper with formulas and values, the Babylonians had mathematical tables written on clay tablets. They had advanced mathematical knowledge, they could extract the square root and the cubic root of a number, they could work with Pythagorean triplets 1200 years before Pythagoras, had a knowledge of pi (the report of which is the division of the circumference of the circle and its diameter) and possibly e (the exponential function), could solve some quadratics and even polynomials of degree 8, solved linear equations, dealt with circular measurement. In ancient history, many civilizations dealt with Pythagorean triplets and Pythagorean triplets are just three numbers a, b and c natural numbers that are in a relation of the form $a^n+b^n=c^n$ where n is a natural number larger than 1. If n=2 this gives us the Pythagoras theorem. The babylonian worked with Pythagorean triplets, but they were made famous by the greek Pythagora. They would create those clay tablets on which they would write the result of various equations or various approximation of pi etc. They had a numbering system in base 60, a sexgesimal numbering system. All the digits were nails like symbols and the numbering system is visible in Fig. 1.5. Note that computer science students are often complaining that the binary system is hard to work with, now imagine that the Babylonians used 60 digits.

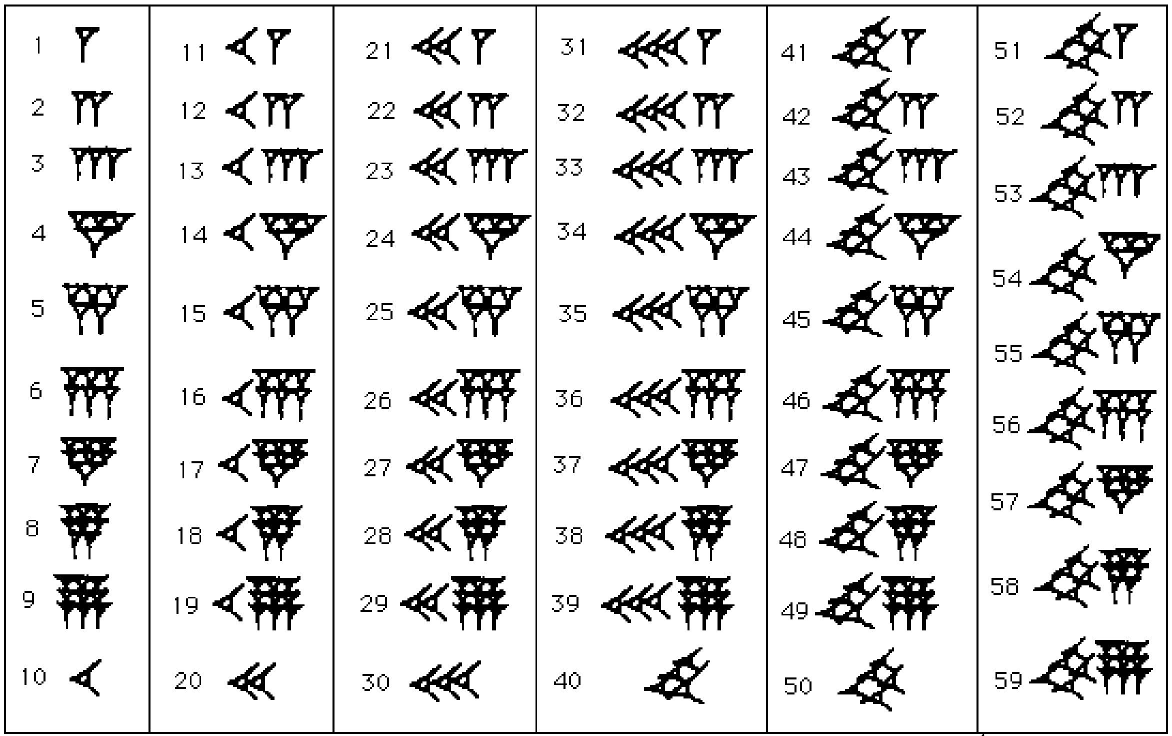


Fig. 1.5 Babylonian Sexagesimal (base 60) number system[4]

[4] http://www.wikipedia.org

We still have reminiscences, things in our lives in modern society that are derivatives from this base 60 numbering system. For example, a minute has 60 seconds, an hour has 60 minutes, there's 360 degrees in a circle which is a multiple of 60. You may also notice that there's no zero digit and you will see that many ancient civilizations, almost all of them, the Arabs, the Indians, the Greeks, the Romans, they didn't use the zero digit. Zero doesn't make any sense because one means that you have one object, e.g. one apple to trade, two would mean two apples to trade etc. (digits were used in commerce where you exchange goods), but there were no reason to talk about zero products, e.g. zero apples. Actually, the Indians invented the digit zero around the year 800AC and the digit zero was popularized in Europe that italian famous for a very famous arithmetic progression, Fibonacci. Fibonacci published a book at the beginning of 1000s called “De libero abaci” which was a commercial trading book involving math and there he wrote for the first time the digit zero. It took a lot of time until the rest of the civilization actually used zero and the number zero was not used in legal contracts until around around 1700. In the Babylonian math base 60 is quite useful because many numbers are divisors of 60 and so, a lot of fractions don't have an infinite number of fractional digits (e.g. 1/2, 1/3, 1/4, 1/5, 1/6, 1/10, 1/12, 1/15 and 1/20). Although some fractions still are infinite (e.g. 1/7, 1/13), but of course the Babylonians would just approximate those numbers. For computing these rational numbers (i.e. fractional numbers with a finite number of fractional digits), the Babylonian had many reciprocal tables printed on clay (i.e. the value of 1/a where a is a natural number). They had a division algorithm like $a/b = a \times (1/b)$ which was based on multiplication, i.e. it was based on multiplying the number with the reciprocal of this number. Then, they would use a table for reciprocals and another table for multiplications and using the aforementioned algorithm, they could compute the division of two numbers. They would also have some formulas that allowed them to compute a multiplication based on addition and subtractions:

$$ab = [(a+b)^2 - a^2 - b^2]/2$$

$$ab = [(a+b)^2 - (a-b)^2]/4$$

with the help of clay tablets which would contain squared numbers. They also had formulas for approximating the square root of a number:

$$\mathrm{sqrt}(a^2 + b) \approx a + b/2a$$

The Babylonians could solve quadratic equations like $ax^3 + bx^2 = c$ using clay tables. As already mentioned, they had tables for powers of a number. And also they had clay tables with Pythagorean triplets as you can see in Fig. 1.6.

Fig. 1.6. Clay tablet with Pythagorean triplets[5]

The Egyptian

While the Babylonian had a more arithmetic knowledge of mathematics, meaning they used numbers a lot, in contrast with them, the Egyptian had a more geometrical knowledge of mathematics and this is because the Egyptian lived on the shore of the Nile river and there was a dry season and a wet season, and in the wet season when the Nile would flood all the lands and after the flooding would pass, people needed to measure the area of the lands again to know which land belongs to each person and for this, they needed geometrical knowledge like computing the perimeter of a specific land part and also computing the area of that specific land. They also wanted to be able to measure circles, compute ratios for pyramids and so on. They also didn't have the zero digit in their numbering system. They had a numbering system in base 10 as opposed to the Babylonian. Their numbering system is mostly based on digits drawn using simple rods (see Fig. 1.7). For numbers larger than 1000, they used other odd looking symbols.

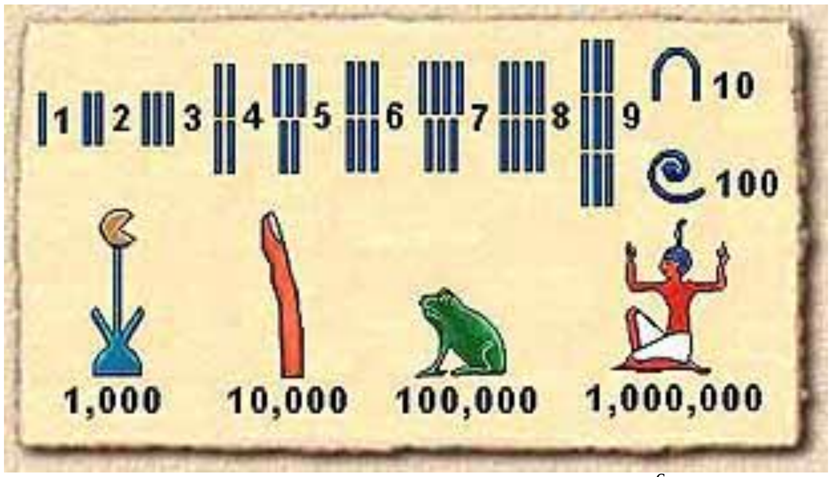
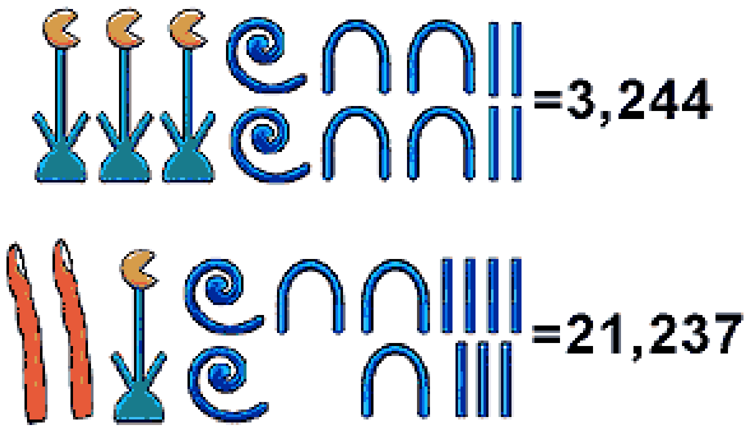


Fig. 1.7 Egyptian numbering system[6]



[5] http://www.wikipedia.org
[6] http://www.wikipedia.org

Fig. 1.8 Examples of Egyptian numbers[7]

The Chinese

The Chinese civilization was an enclosed civilization and it remained this way until I believe in 1800. This means that they usually produced a science parallel to the European science. Many concepts were encounter in Chinese science and European science under different names. The Chinese also knew about the Pythagorean numbers and they also knew Pythagoras theorem which they called it Gougu rule. Math in China was for a long time hidden from other civilizations. They had a practical approach, not axiomatic, like the European approach. They invented the gun powder and other things, but their inventions were not known outside of the Chinese civilization until later in history. The most famous book in the ancient history of Chinese mathematics was called "Nine chapters of the mathematical art" (*Jiuzhang suanshu*) it dates from around 100AD and contains about 246 practical problems from various field of mathematics:

- Chapter 1: land surveying, area problems, additions/ substractions/ multiplications/ divisions of fractions, approximations of PI
- Chapter 2: exchange of goods, proportions
- Chapter 3: proportions: direct, inverse, compound, arithmetic and geometric progressions
- Chapter 4: areas, unit fractions
- Chapter 5: problems on construction of canals, ditches, dykes; volumes
- Chapter 6: ratio and proportions; fair distribution of goods
- Chapter 7: linear equations solved by making two guesses at the solution, then computing the correct answer from the two errors
- Chapter 8: solving systems of linear equations
- Chapter 9: right angled triangles, Gougu rule, quadratic equations

The Maya

The Maya civilization lived between 250AD and 900AD in Yucatan Peninsular (Mexico). They were conquered by the Spanish conquistadors. Their civilization was centered around large cities for example the city of Tikal had 50.000 people at its peak. The Maya civilization declined after 900 and were conquered by Spanish navigators in 1500. Their legacy is largely destroyed by the Spanish conquistadors. What we still have from them is three codexes containing their culture inscripted on tree bark. The most well preserved codex of the three is the Dresden codex – you can see a part of it in Fig. 1.9.

[7] http://www.wikipedia.org

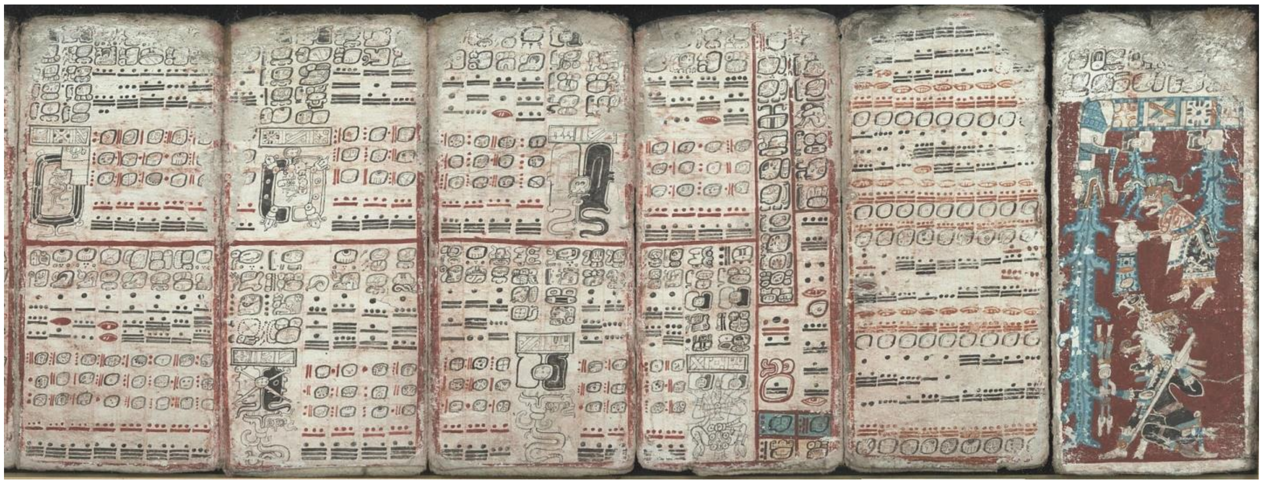


Fig. 1.9. The Dresden codex of the Maya

The Maya had various gods and when the Spanish catholic navigators arrived in South America, they saw these pagan gods, they believed there was only one god so they burned everything, leaving few things behind from the Maya culture. Their numbering system had the base 20 (see this in Fig. 1.20). What is noticeable here is that they had a symbol for zero, but actually they didn't use zero much. Their digits look to me like Morse code, with dots and dashes.

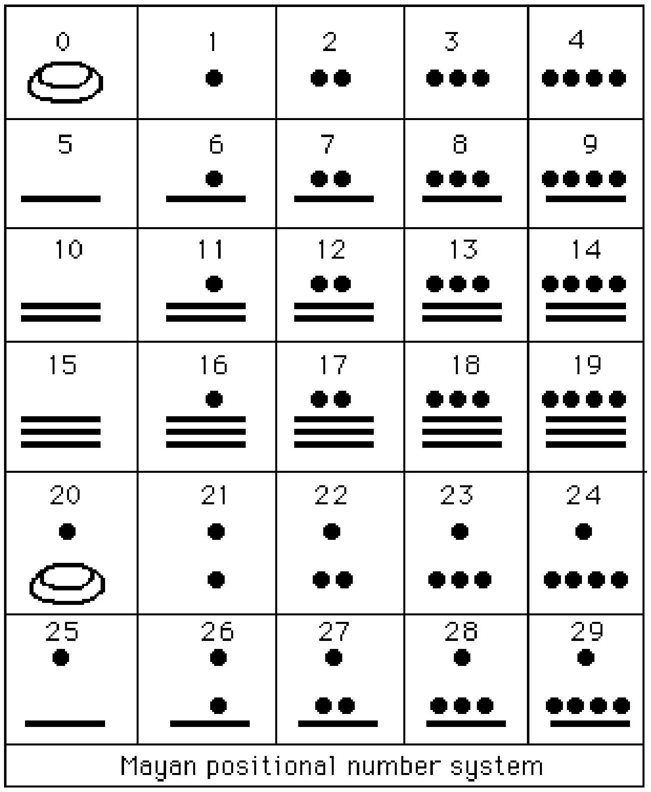


Fig. 1.20. The Maya numbering system[8]

[8] http://www.wikipedia.org

They had two calendars, a ritual/religious calendar of 260 days and a civil calendar used in agriculture in order to know when to seed the grounds and to work the land. This civil calendar had 365 days which is exactly the number of days we have in our own modern calendar (except leap years). The Maya also performed astronomical measurements. They did not use multiplications and divisions, only additions and subtractions.

The Greeks

The ancient Greek civilization is quite a surprise in the history of human civilization because of all the accomplishments they obtained in many sciences, in math and philosophy, in chemistry, in physics and so on. They had many personalities important for the human culture, but only a few of them are mentioned here, those that had important mathematical contributions. For example there's no philosopher here, only those that have had a link with mathematics, there is no Socrates or Plato or Aristotle because they didn't have important contribution in math. From 600BC to 500AD, the Greeks made important contributions to mathematics and several other sciences. Algorithmic thinking was reflected in Greek culture at least by Euclid’s algorithm for greatest common divisor and Eratostene’s sieve algorithm for finding prime numbers. Their civilization was centered around large cities like Athens and Sparta.

Thales of Millet

Starting with Thales of Millet, he lived between 634BC and 546BC in the island of Millet. He stated the theorem that bears his name about similar triangles. He predicted a solar eclipse in the 28th of May in 585BC. He also stated that the base angles of an isosceles triangle are equal. In astronomy, he believed that the earth was a flat disc floating on an infinite ocean. There are still contemporary persons who still believe this.

Pithagoras of Samos

Pithagora of Samos lived between 560BC and 480BC in the island of Samos. Everybody knows about Pythagoras through Pythagoras’ theorem, but actually Pythagoras was a good mathematician, but also a sort of priest. He founded a religion, a mystic cult called the Pythagorean cult which had the pentagram as symbol. He was the head of this mystic cult. The lives of the members of this cult involved working with numbers which they considered that are gods and govern everything. The members of the cult allocated all their time to study numbers and most particularly, to study Pythagorean numbers, i.e. those numbers that are that are grouped together by the relation: $a^2 + b^2 = c^2$. They had many unbreakable rules, like for example, they were vegetarian, they did not eat beans etc. They developed the theory of Pythagorean numbers. They proved that the square root of two is an irrational number. They also believed in reincarnation, even for animals. They invented the fundamentals of music theory that are still used today, based on numbers and proportions; they realized that dividing the length of a string (of a musical instrument) into ratios of 1/2, 1/3, 1/4, and 1/5 creates the musical intervals of an octave, a perfect fifth, a second octave, and a major third respectively. These ratios between musical intervals, i.e. the octave, the perfect fifth, the major and minor third, are used even today for a stringing a guitar and fretting the the fretboard of a guitar. If you know guitars, there's the piece of flat wood glued on top of the guitar’s neck that's called the fretboard and there are iron rods placed on specific distances on that fretboard and when the guitar player presses down a guitar string over one specific iron rod which is also called a fret, then the player divides the length of the string into specific proportions (the ones discovered by Pythagoras’ cult) and that's how he/she obtains various musical notes. These were all known to the ancient Greeks, to members of the Pythagoras cult.

Euclid, father of axiomatic Geometry

Euclid lived between 325BC and 270BC in Alexandria, Greece. He is famous for setting a rigorous grounds on Mathematics, especially the part of Mathematics we call Geometry. He published this book called “Elements” which laid the foundations of axiomatic geometry.

Elements began with postulates/axioms (common knowledge, general truth that he considered true beforehand like the axiom of the parallels which says that through a point which is outside a line we can only draw one single line that's parallel with the original line) and derived and proved all results from these postulates in a rigorous manner. He also developed the algorithm for finding the greatest common divisor that bears his name.

Eratosthenes

Eratosthenes lived 284BC and 192BC and is famous for the algorithm called Eratosthenes sieve that computes all natural prime numbers that are smaller than a given number. He built a star map with 675 stars and he measured the diameter of the Earth.

Heron of Alexandria

Heron of Alexandria lived between 10AC and 70AC in Alexandria and is famous for the formula of computing the area of a triangle using the semi-perimeter. He also built an iterative algorithm for computing the square root of a number. He also built a steam engine.

Archimedes of Syracuse

Archimedes lived in Syracuse between 287BC and 212BC. He is famous for Archimedes law which says that if you submerge a solid body into a volume of water then that solid body would be pushed from bottom to top by a force that is equal to the weight of the liquid that is displaced by that solid body. He had important contributions in mathematics and physics. There were various legends about Archimedes, one says that he discovered Archimedes law when he was baiting and then he ran naked on the streets of Syracuse and shouted “Eureka!” because he was very happy with his discovery. There is another legend about him that says he built giant mirrors and set on fire the roman ships that came on sea to conquer the Greeks. And this way, the Greeks managed to push back the roman army for a number of days until eventually the Romans conquered the Greeks. The legend says that he was killed by a roman soldier because he was drawing circles in the sand on the beach and there was this roman soldier that was near Archimedes and Archimedes complained that the roman soldier stood in the sun and cast a shadow on Archimedes circles and the roman soldier killed him. These are all legends about Archimedes's life, we don't know if they are true or not.

Klaudios Ptolemaios

Klaudios Ptolemaios lived between 85AD and 165AD in Alexandria. He approximated PI and most importantly, formulated the geocentric model of space which states that the Earth was the center of the known universe and the planets and the sun all orbit around the Earth. This model lasted a long time in human culture, until it was overthrown by Nicolaus Copernicus in around 1500s when he proposed the heliocentric model which means that the sun is the center of the known universe, i.e. the galaxy. The Copernican model was promoted by Giordano Bruno and Galileo Galilei who both suffered the opposition of the Catholic Church (Giordano Bruno was even burned alive by the Roman Inquisition).

Diophantos of Alexandria

Diophantos lived in Alexandria approximately between 201/215 AD – 285/299 AD. He is also known as the father of algebra because he wrote a collection of 13 books called “Arithmetica” and containing 130 algebraic problems. Equations in these books are also called Diophantine equations. Most of the things he wrote in those books were not actually invented by him, but they were rather a survey or a summary of all known arithmetics. Those books or those books were famous because they were used as an important source of mathematical knowledge throughout the centuries (6 of these books were known in Europe in the late 15th century). Originally written in ancient greek, they were translated into latin and arabic after the Greek civilization declined. Pierre de Fermat, the French mathematician, extended the results from these books by writing various solutions and theorems on the edge of pages from these books. After the Greek civilizations declined, they were conquered by the Macedonian empire leaded by Alexander the Great who conquered most of the Europe and parts of Asia too, arabs, Indians. The Macedonians would take books from various people that they conquered and they would translate those books into greek and they would deposit those books into libraries. One famous library is the library of Alexandria. The arithmetic books of Diophantos were also stored there. Many scientists came to the library of Alexandria to gain knowledge, including the indians who studied at the library of Alexandria and then they they brought European knowledge to India. Following, the Arabs who were a conquering people, conquered most of the Europe and also the Indian civilization and they took the knowledge from the Indians and brought it back to Europe. Diophantos Arithmetica is also famous due to Pierre de Fermat, the French mathematician who lived in the 1600s. He wrote on the edge of a page from Arithmetica that there is no $n > 2$ for which there are Pythagorean triplets meaning if $n > 2$, there are no three numbers a, b and c such that $a^n+b^n=c^n$. The relation works only for $n = 2$ which gives the theorem of Pythagoras. And he wrote on a side of Diophantos Arithmetica that there is a very nice demonstration of this (this is also called Fermat’s last theorem), but there isn’t enough space on the side of the page to write it. From 1600s until 1995, many mathematicians wondered what was the proof of Pierre de Fermat because they didn't know it and they tried to solve it for more than 300 years and eventually Andrew Wiles, a British mathematician, solved it in 1995. There are organizations that give important prices to people who solve unsolved, old mathematical problems and I think Andrew Wiles received a 1 million dollars prize because he proved this unsolved problem, Fermat’s last theorem. You will see in a following chapter, another link of computer science to unsolved mathematical problems.

The Indian civilization

Indian mathematics was influenced by the greeks, egyptians and Babylonians. The European knowledge was brought to India by Alexander the Great when he conquered India. Around 5 century AD the Indians moved from a Greek numbering system to a Babylonian-like one but in base 10. They also invented the number zero. Because they decoupled numbers from their geometric significance, they also used negative numbers. You can see in the following picture an evolution of Indian numbers. The modern digits that we use are called arabic numbers because the Arabs conquered the Indians and all Europe and they adopted this base 10 numbering system from the Indians and brought it to western Europe.

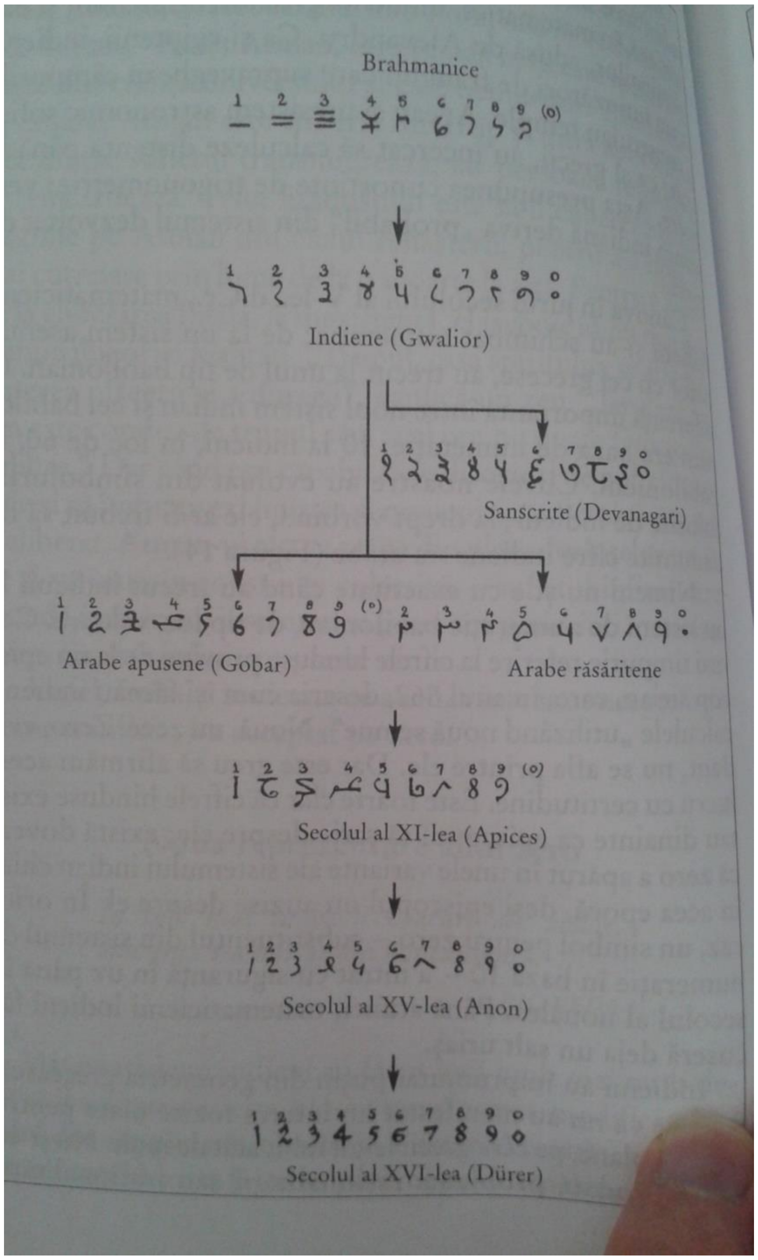


Fig. 1.21 Evolution of Indians and Arabic numbers

Aryabhata who lived between 476AD – 550AD was a famous Indian scientist who approximated PI, approximated Earth and lunar rotations and affirmed the theory of heliocentrism. Brahmagupta (598-668 AD) was another indian scientist. He solved 1st degree, 2nd degree and some quadratic degree equations. He wrote the first book to treat zero as a number. He tried to define division by zero (0/0=0). He also stated arithmetic rules for positive and negative numbers and zero.

The Arabic

The Islam rised by the year 700, occupying Egipt, Syria, Mesopotamia, Persia. Islam advanced on the west until Spain and France and conquered China and India on the east. They would absorb the wisdom of the conquered people. They took the hindu base 10 numerals from indians and brought it back in Europe. A famous Arabic mathematician who has connections to computer science is Abu Abdullah Muhammad bin Musa al-Khwarizmi.

Al-Khwarizmi lived between 780AD - 840AD. Al-Khwarizmi is considered the “father of algebra” because he wrote a book “Al-jabr w'al-muqabala”. “al-jabr” becomes “algebra” and it means “to complete a task”. He then wrote another book about hindu numbers that was translated in the 12 century to latin “Algoritmi de numero Indorum”. His name was translated into latin by “Algoritmi”. An “algorithm” would be a trick for working with hindu numbers. This is how the concept of an algorithm was born.

Theoretical foundations of Computer Science

In the previous section, I talked about what is computer science and how can we find the beginnings of computer science in human history. I have talked about the beginning of the hardware part in computer science, then we referred to the software part and we thought about algorithms and algorithmic thinking as the first roots of computer science in human history. I have said that there are references to algorithmic thinking throughout human history starting from ancient history, from the Babylonian, Egyptians and Greeks. And so, we have talked a little bit about the history of algorithmic thinking, about the fact that computer science didn't existed as a special science field, but instead roots of computer science like algorithmic thinking existed as part of mathematics. I discussed about various aspects of algorithmic thinking in the Babylonian, Greeks and Egyptians and then we moved from ancient history to the middle ages and we have reached year 1000 when we talked about the Arabic and the Indians and how they brought the mathematical knowledge back to Europe. We will now fast forward in time until we get to 1800s and we discuss about the first mechanical calculators, the two machines that were designed by Charles Babbage, the differential engine and the analytical engine. We will then go further to the beginnings of the theoretical computer science which coincides with the first electrical computers or electromechanical computers.

The first general purpose, mechanical computers were designed by the English scientist Charles Babbage, the first one is called the differential engine and the second one is called the analytical engine. By general purpose computational machines we mean that besides computing the basic 4 operations, addition, subtraction, multiplication and division, these machines could also evaluate more complex functions like exponential, logarithm, trigonometric and polynomial, and they even had conditional execution (IF), loops, program statements. In some way, Charles Babbage's differential engine is not actually a general purpose computer, but the next one the analytical engine is. But anyway, the differential engine is a more complicated mechanical computing machine because it is able to compute addition, subtractions, divisions and multiplications, but also polynomial functions up to the seventh degree and you will see that the next machine, the analytical engine, does so much more and it gets quite close to the concept of computers as we have them today, of course keeping the historical differences in mind. So Charles Babbage tried to build the differential engine between years 1822 and 1842, he didn't succeed in this time interval, he eventually moved along and tried to build an even more complex computational machine, the analytical engine, but failed to do that also. Prior to 1822, the British government with his various agencies, compartments and departments used many tables with mathematical values of several polynomial functions, logarithmic functions, trigonometric and exponentials in commerce, computing taxes, measuring land areas, sailing on see etc. There were entire books containing tables of mathematical, numeric data and due to the fact that all these values were approximations of functions' values, two different books would report different values for the same function in the same sample point. Charles Babbage thought that a machine that was able to instantly compute these values and compute them with high accuracy would be very useful for the British government and many other parts of the industry. And he built a small prototype of this machine in 1822 and with this small prototype he went to the British government, got a grant of 1700 pounds from the British government in 1823 and he promised them that he would build the differential engine in one year. After about 20 years and after the British government founded his project with more than 17.000 pounds Charles Babbage finally abandoned the project due to technological problems. The idea that the basic idea was that the industrial equipment existing at the time did not permit manufacturing very accurate and very precise wheels, rotors and rods with specific distance between the teeth of the wheel and so the precision required in order to build Charles Babbage's differential engine was not something achievable at that time. That was one reason for which the differential engine was never built completely in Charles Babbage's lifetime. The second reason was related to Charles Babbage's personality - he was a hard headed person, had strong convictions and he didn't get along with many people. The design idea was pretty great; it represented numbers in base 10, it was able to compute polynomial functions up to the seventh degree, it represented negative numbers as 10's complement. The main mathematical idea which lied behind the concept of the differential engine was the method of finite differences and the idea is that if we have a polynomial with a degree k, then the k difference is a constant always. I'll show you an example here. So let's assume we have this third degree polynomial :

$$f(x) = 2x^3 - 3x^2 + x + 1$$

and we already know the values f(0)=1, f(1) = 1, f(2)=7, f(3)=31, f(4)=85 and we want to compute the values of this polynomial in the points 5, 6, 7, … The method of the finite differences relies on computing the values from the following table:

x	**f(x)**	**D1(x)**	**D2(x)**	**D3(x)**
0	1	0	6	12
1	1	6	18	12
2	7	24	30	12
3	31	54	42	12
4	85	96		12
5	(181)?			12

On the first column we have the values of x for which we want to compute f(x). On the second column we write with black the values of f(x) we already now (we have computed them manually). On column "D1(x)" we have the first order differences: D1(0) = f(1)-f(0), D1(1) = f(2) – f(1), … On column "D2(x)" we have the first order differences: D2(0) = D1(1) - D1(0), D2(1) = D1(2) – D1(1), … On column "D3(x)" we have the first order differences: D3(0) = D2(1) - D2(0), D3(1) = D2(2) – D2(1), … The values written in black are the one we already know and the values written in green or red are the ones we are going to compute. We can see in the table that the difference of the third degree remains constant to 12. This value depends on the actual polynomial, but it is constant. When we reached the difference of the degree of the polynomial (i.e. 3rd degree for our example), in order to compute the next value of the polynomial, f(5), we can go backwards, so we first complete the D3(x) column with all those red values of 12, then we know that D3(2)=12, so we can compute D2(3) = D3(2) + D2(2) = 42. Knowing D2(3) = 42, we can now compute D1(4) = D2(3) + D3(3) = 96. Knowing D1(4), we can now compute f(5) = f(4) + D1(4) = 181. This is the method of finite differences. The advantage of this method is that there's no need for multiplication and division operations which are harder to implement in hardware, so we can compute the value of this polynomial using just additions and subtractions. The disadvantage is that we need to compute all the values of f( ) in all these points and all the differences up to the point that we are interested. Charles Babbage's idea was to use the differential engine in order to produce an accurate book with tables of values of various polynomial functions, because there were so many books with mathematical tables used throughout the British empire and they had various differences/errors. You can now see the connection between these books that contain tables with various mathematical results and the Babylonian clay tablets which contained equivalent things. I have told you that the differential engine wasn't built in Charles Babbage's lifetime, but it was later built in 1995 by the London Science Museand you can see here a video on how the differential engine worked: https://www.youtube.com/watch?v=BlbQsKpq3Ak.

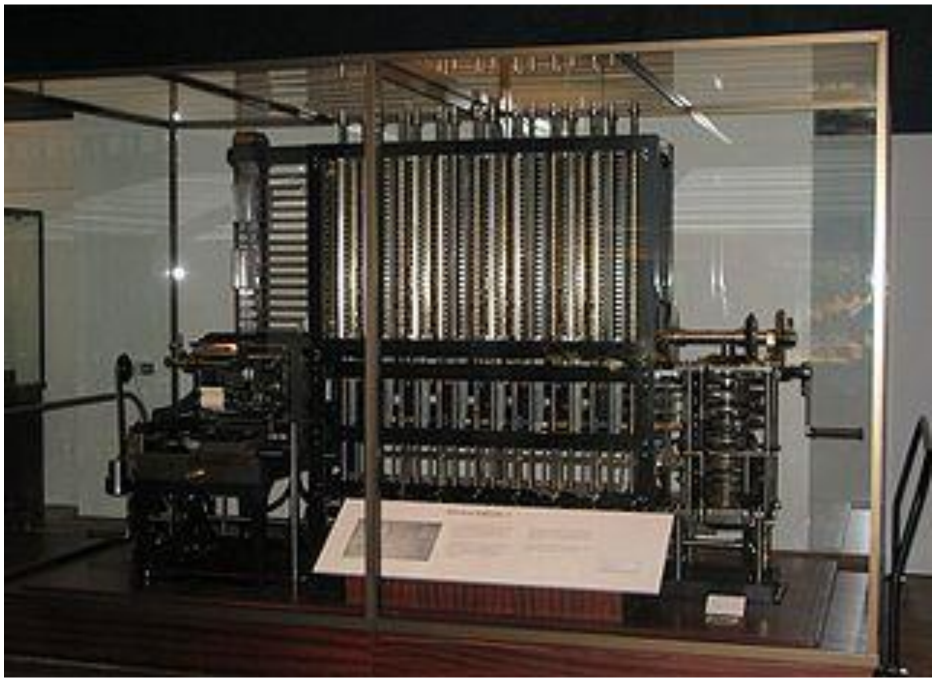


Fig. 2.1 Copy model of Differential Engine at London Science Museum, Charles Babbage, 1822-1842[9]

[9] http://www.wikipedia.org

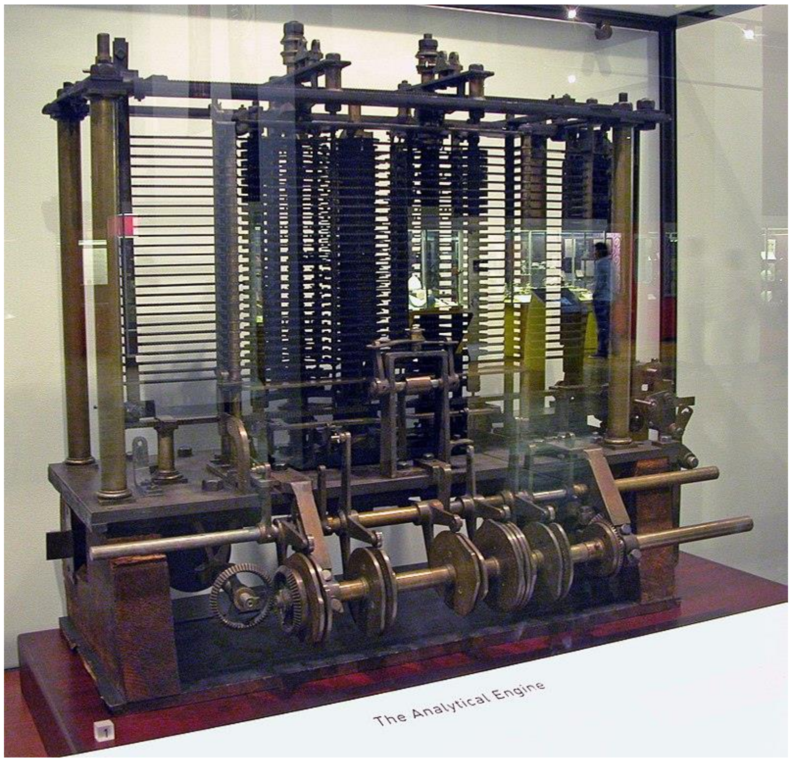


Fig. 2.2 Copy model of Analytical Engine,
Charles Babbage, 1837[10]

Now moving on to the analytical engine. Sometime along the way, Charles Babbage abandoned the project of the differential engine and he started thinking about a more complicated machine, a machine that would be as large as a locomotive and that was called the analytical engine. Actually, this was the first general purpose computational machine because it could add, subtract, divide and multiply, compute polynomial functions as the differential engine, but also it could perform conditional execution like we have today in programming languages the conditional instruction, "IF", it had loop cycles and a primitive form of parallel programming. Also, it had its own memory, it had a memory store that was able to store 1000 numbers of 40 decimal digits each, so that would be equivalent to 16.2 kilobytes of memory in today's memory terms. The input was on punch cards - you will see what punch cards are in a couple of paragraphs, and it had an arithmetical logic unit (a mill) which could perform additions, subtractions, multiplications, division and square root. And it also had a sort of procedures or functions and a primitive form of assembly language. One could actually program it, one could write a program for it. That's what Ada Lovelace did, she wrote a program specification for this machine, but of course the program never worked because the machine was never completely built, but this is what Ada did and Ada Lovelace who was the first programmer in history. There is a programming language named in honor of Ada which is is still active today. This machine was later built, in 1990s for a museum.

[10] http://www.wikipedia.org

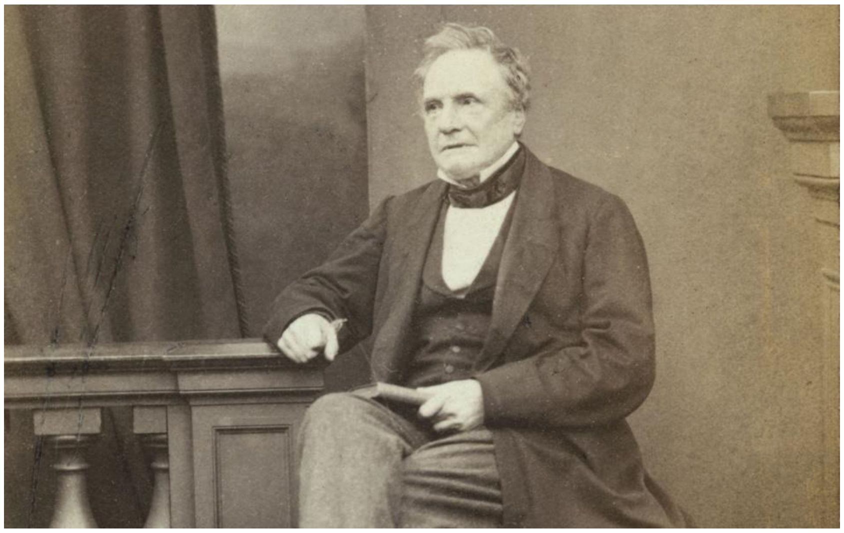


Fig. 2.3. Charles Babbage, 1791-1871

Moving on with the history of general purpose computing machines, we arrive at looms (i.e. in Romanian this is “razboi de tesut”). When I was a child, I don't know, 8 or 10 years old, my grandmother who lived in the countryside had a loom made from wood. I could watch her making carpets when I was at her house, in vacation. The loop had some pedals that you press and then you need to pass the needle which is a big (about 20 cm long) pointy shaped piece of wood, through some walls of threads. You can see pictures of loom in Fig. 2.4.

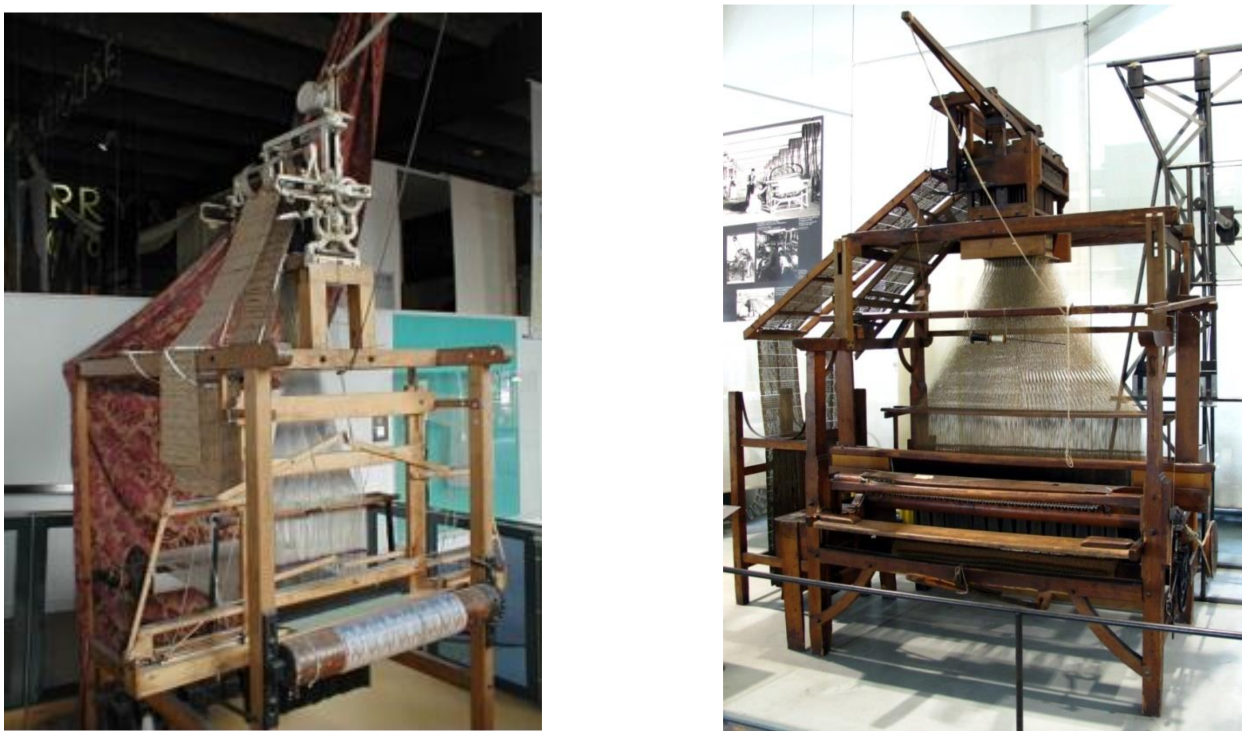


Fig. 2.4. Looms [11]

[11] http://www.wikipedia.org

A French guy, named Joseph Marie Jacquard who lived in France and had small looms company came to an idea in 1804 that instead of manually moving that wooden needle between those threads in the rolls of threads, he could write a template on a paper like the one depicted in Fig. 2.5 and the loom can "read" this template one line at the time and saw a tapestry according to this template. A hole in a row of the template means that the needle should pass below the roll of threads and a full space means that the needle should pass above the roll of threads.

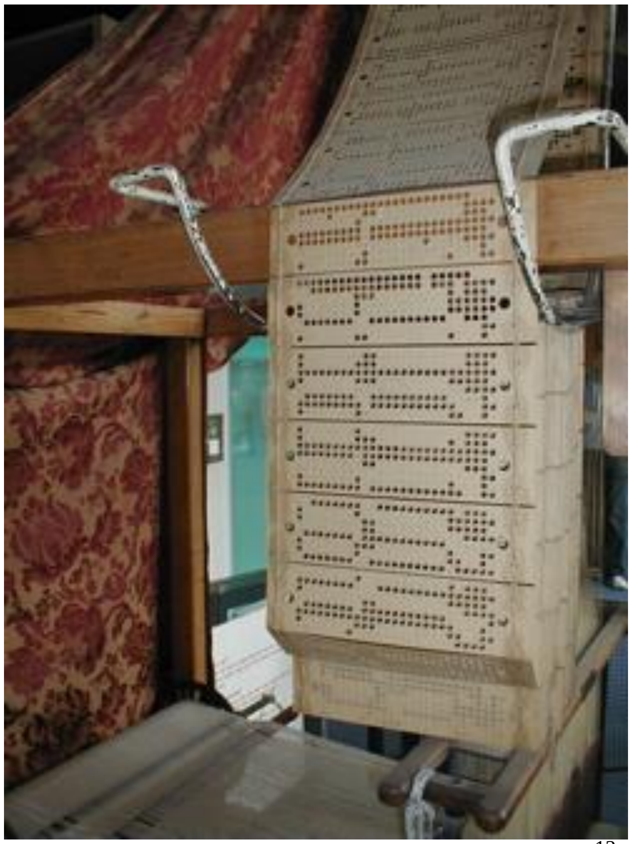


Fig. 2.5. Punched cards for loom templates[12]

So, Joseph Marie Jacquard thought he could use this kind of punch cards to describe a sewing pattern and then he could *program* those looms to build a specific tapestry according to those patterns. And this should be done automatically. Well, I don't know all the details for this, but I'm expecting that there was a human person who only needed to press some pedals and no more interaction was needed from the human person with the loom. So those punched cards patterns would tell the loom exactly what to do and when to move the needle below the threads and when to rise it above the threads. Jacquard invented this in 1804 and Charles Babbage thought of using the punched cards on the analytical engine in 1837 as a way of writing programs on punched cards. Starting with Babbage, punched cards started to be used as a medifor storing programs until about 1970 when the keyboard and the mouse were invented and also the monitor and the floppy-disk storage. While punched cards were

[12] http://www.wikipedia.org

used in the computing history up to 1970 in Western Europe and the United States , in Romania, they were used up to 1992, immediately after the anti-communist revolution in 1989. So punched cards were used in Romania a lot until 1992 and in the university students would program a computer by writing the program on each on these punch cards. They wouldn't operate the computer. There was only one computer per university and the students would not operate a computer like we do it today. Now I have my own notebook and I operate it myself, but back then, the students would operate a tabulating machine (i.e. a machine that punches holes into paper/cardboard cards). So the students would write programs on punched cards and they would come with the stack of punched cards to the operator, i.e. the computer technician and they would give that stack of punched cards representing a program to that technician and the technician would run the program written on those punched cards on the computer and then the student would come the next day to collect the output of the program which would be a listing on paper and that listing could be the result of running the program or usually, that listing would contain a compiling error or a runtime error. A compiler error means that there was one of the cards punched incorrectly. So the student would have to take this the stack of cards and then replace the incorrect card a correct one and submit the program again to the technician. This is how programming happened before 1989 in Romania and throughout the world using punched cards.

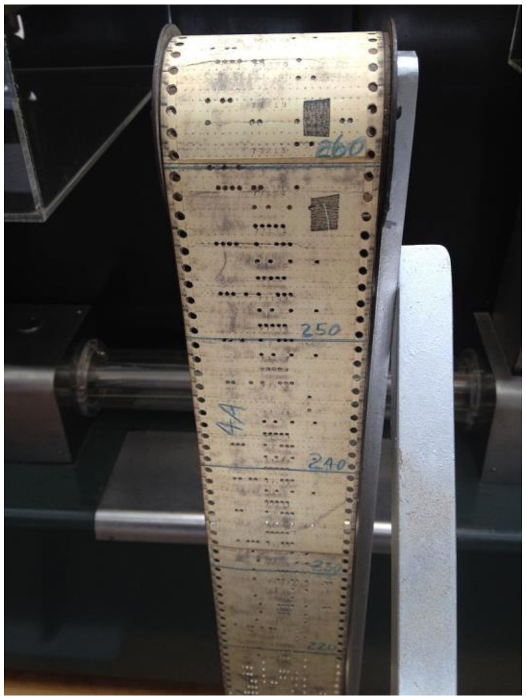


Fig. 2.6. Old computer punched card (1944)[13]

[13] http://www.wikipedia.org

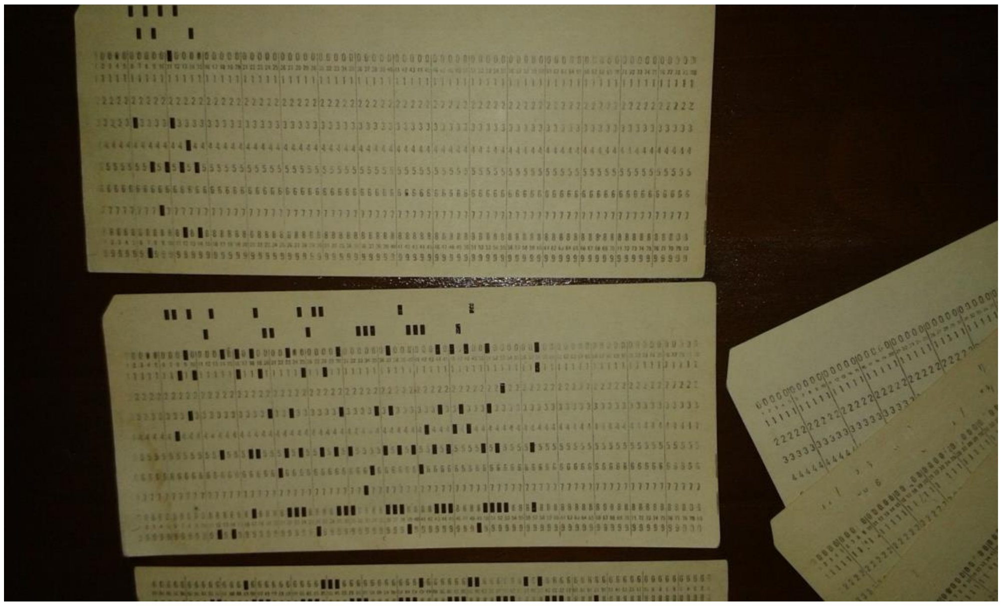


Fig. 2.7. My collection of “modern” punched cards

Any one of these punch cards contains one line of code of the program, so you can imagine that it takes a lot of cards to write a complete program. There is no standard for punched cards; different manufacturers use different formats. Most formats rely on 6-bit BCDIC code, where some bits represent a “zone” and the rest represent “an index in that zone”. A column always encodes one symbol (letter, digit etc.) from the current line of code. There are 9 rows for the “index” (1-9) and three rows for the “zone” (0, 11, 12) – 4 zones (0, 11, 12 or none punched). You can see the rows of the “index” and the “zone” in Fig. 2.8. An example of a format is at:
http://www.columbia.edu/cu/computinghistory/026.html
Depending on which row on a specific column is punched, that is the index, and depending on the zone selected using the extra 3 rows, we obtain the character that is encoded by that column. This code is just a two level code, first one hole selects the zone and depending on that zone, the other hole selects a specific symbol digital letter from a specific symbol set.

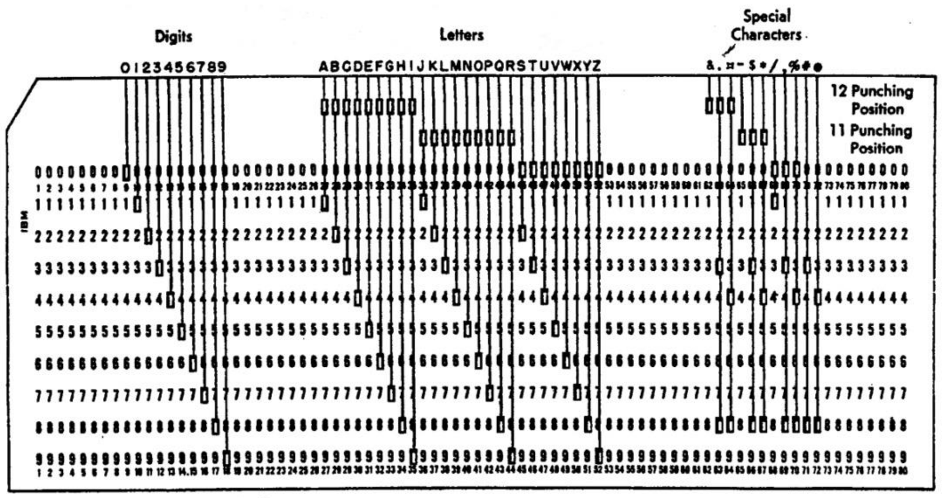


Figure 2. Punching Positions in Card

Fig. 2.8 The “index” and “zone” rows of a punched card[14]

In the United States , in 1804 Hermann Hollerith who was a businessman, an inventor and a statistician worked for the United States Census Bureau. The census in the United States was usually performed this way: a designated person from United States Census Bureau would come to each family's home and he/she would have a list of papers to fill; he/she would write on paper how many adults live in this home, how many children live in this home, what's their religious beliefs, how many are girls, how many are boys, educational level etc. Then, the person working at the Census Bureau would gather all these statistical data on paper and would bring it to some offices and there all the statistical data from the county would be reunited and transcribed and saved. Needless to say that the census was a laborious process. Herman Hollerith thought that this process would happen a lot faster if used punch cards to store all the statistical data. And he built machines that would process these punch cards automatically and produce statistics. These are called tabulating machines – you can see one in Fig. 2.9.

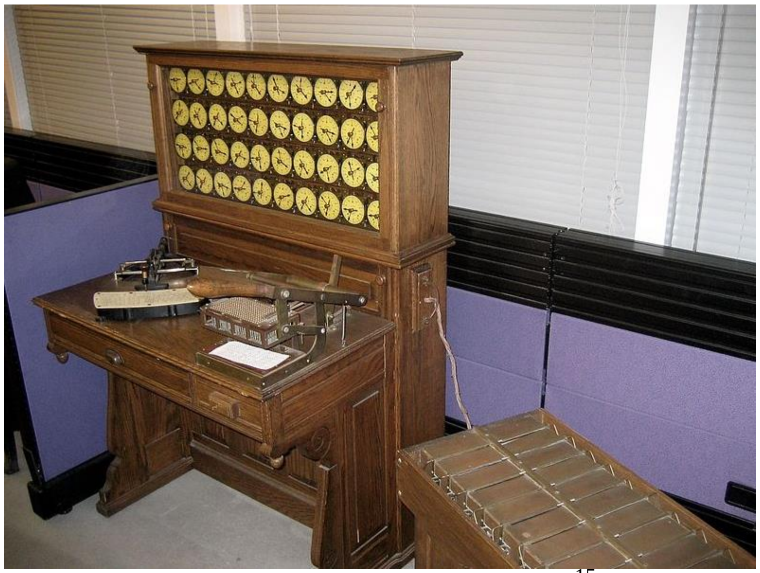


Fig. 2.9. A tabulating machine[15]

These tabulating machines that Hermann Hollerith built would tabulate data meaning they would produce holes into cards and there were also the second type of machines that would process those cards and read the holes, i.e. the data, from the punched cards. Hermann Hollerith used these tabulating machines in the 1819 US census and the census was completed two years before the estimated time, so it was quite efficient. And then he started a company producing tabulating machines and this company later merged with other three companies and they formed a large company which also exists today and is called IBM (International Business Machine). The company of Hermann Hollerith was called the Tabulating Machine Company.

[14] http://www.columbia.edu/cu/computinghistory/026-card.jpg
[15] http://www.wikipedia.org

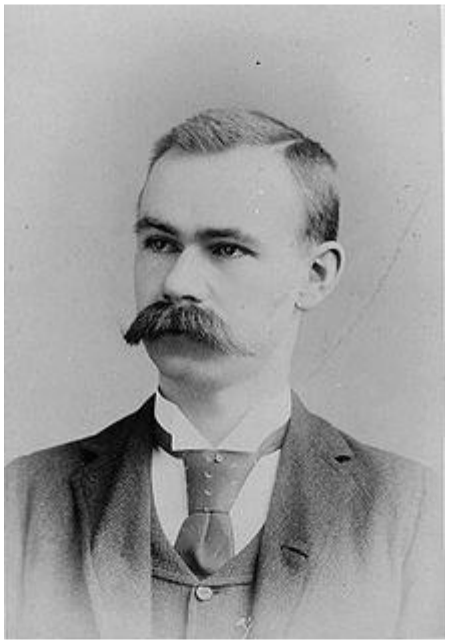


Fig. 2.10. Hermann Hollerith

Fig. 2.11. 1819 US census

Now we should actually move to the theoretical foundations of computer science. Theoretical foundations of computer science are linked to three individuals that well, are very smart individuals, geniuses, but each of them had their own peculiarities. Each of them were in their own way “special” meaning they had traits that separate them from the rest of the population besides the fact that they were really, really smart. And those three individuals are Kurt Goedel who is an Austrian mathematician, logician and philosopher, Alan Turing who is a British mathematician and John von Neumann who is a Hungarian-American mathematician, chemist, a physicist, computer scientist and many other things. John von Neumann was a lot of things and you will see that at the end of this section. All of them were really bright individuals. But the whole story starts with David Hilbert (see Fig. 2.12), a German mathematician, and his list of unsolved mathematical problems. David Hilbert proposed in 1900 in Paris at a Math conference a list of 10 initial unsolved, important problems in the field of Mathematics. These problems are unsolved problems similar to the one I have talked about in previous sections, Fermat’s last theorem about the Pythagorean numbers. I have also told you the legend surrounding that theorem that remained unsolved for about 300 years when it was finally solved by Andrew Wiles in 1995.

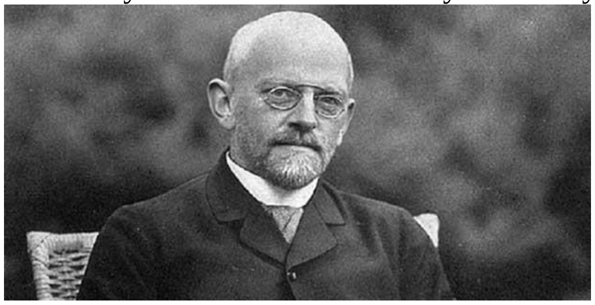


Fig. 2.12. David Hilbert

So these 10 initial problems are those kind of unsolved problems in mathematics that existed in mathematics for a long time. David Hilbert completed this list soon afterwards with other 13 more problems, resulting in a total list of 23 unsolved problems. You can see the list below[16]:

1. The continuum hypothesis (i.e., there is no set whose cardinality is strictly between that of the integers and that of the real numbers).
2. Prove that the axioms of arithmetic are consistent.
3. Given any two polyhedra of equal volume, is it always possible to cut the first into finitely many polyhedral pieces that can be reassembled to yield the second?
4. Construct all metrics where lines are geodesics.
5. Are continuous groups automatically differential groups?
6. Mathematical treatment of the axioms of physics:
   (a) axiomatic treatment of probability with limit theorems for foundation of statistical physics
   (b) the rigorous theory of limiting processes "which lead from the atomistic view to the laws of motion of continua".
7. Is a^b transcendental, for algebraic $a \neq 0,1$ and irrational algebraic b ?
8. The Riemann hypothesis ("the real part of any non-trivial zero of the Riemann zeta function is 1/2") and other prime number problems, among them Goldbach's conjecture and the twin prime conjecture.
9. Find the most general law of the reciprocity theorem in any algebraic number field.
10. Find an algorithm to determine whether a given polynomial Diophantine equation with integer coefficients has an integer solution.
11. Solving quadratic forms with algebraic numerical coefficients.
12. Extend the Kronecker–Weber theorem on Abelian extensions of the rational numbers to any base number field.
13. Solve 7th degree equation using algebraic (variant: continuous) functions of two parameters.
14. Is the ring of invariants of an algebraic group acting on a polynomial ring always finitely generated?
15. Rigorous foundation of Schubert's enumerative calculus.
16. Describe relative positions of ovals originating from a real algebraic curve and as limit cycles of a polynomial vector field on the plane.
17. Express a nonnegative rational function as quotient of sums of squares.
18. (a) Is there a polyhedron that admits only an anisohedral tiling in three dimensions?
    (b) What is the densest sphere packing?

19. Are the solutions of regular problems in the calculus of variations always necessarily analytic?
20. Do all variational problems with certain boundary conditions have solutions?
21. Proof of the existence of linear differential equations having a prescribed monodromic group.
22. Uniformization of analytic relations by means of automorphic functions.

[16] https://en.wikipedia.org/wiki/Hilbert%27s_problems

23. Further development of the calculus of variations.

Many of those problems would shape the evolution of the mathematical science in the next century. Some of them were later disproved, some others were proved, but still there are some of them that are still unresolved to this day. Many of them require intensive mathematical knowledge like problem 8, checking out whether the real part of any non-trivial zero, i.e. any root of the Riemann Zeta function is 1/2. But others like Goldbach's conjecture on prime numbers is very simply formulated, i.e. it is very easy to understand it. Golbach's conjecture lies unsolved for more than 300 years. It is stated like this: every even number greater than two can be described as the sum of two prime numbers. Actually it was partially demonstrated using powerful computers and simulations and it was shown to be true for numbers up to 4 times 10 at the power of 18. But still it's not proven for any general even natural number. There are very large prices like one million dollars offered by various organizations for those who will solve one of these long lasting problems. So if you want to get rich and you are really good in math, you could take a swing at these problems. We are not interested here by all these problems, but rather with the second problem from the list and this problem is proving that axioms of the arithmetic are consistent. The idea of this problem is if you remember when we talked about Euclid and the beginning of axiomatic thinking and axiomatic science. When he wrote the very famous book called "The Elements" which introduced axiomatic geometry, he envisioned the following thinking for a general science or an abstract science like mathematics: each abstract science should start with a set of axioms which are common truths, very simple truths that are assumed true a priori by everybody and starting from this set of simple truths, all the knowledge in a specific area like arithmetics or geometry can be derived by logical reasoning, by syllogisms and other logical reasoning. This is also generally true even for experimental science like physics; so, you start with a set of common truths and then you put in logical reasoning and then you add external observed data and you mix them together and you come up with new theories in physics. But mostly this is true for abstract sciences like mathematics. Euclid's principle guided mathematical thinking and mathematical development throughout the years, even today we use the same principles; we start with those axioms, then build on them some other theorems derived from the axioms, then use those simple theorems in order to derive some more complex theories and so on build on them in a way we reuse software in software engineering today. In software development, we start from simple system calls that perform input/output functions, these system calls are implemented by the operating system, then we take library like the libc library in Linux or lib32 in Windows that wrap these simple system calls and add further functionality and then, we use some other libraries like the node.js interpreter which wraps over all these system libraries and build something more complex and then use node.js packages that use all these functions and build on them and do something more complex and so on. This process is similar in Mathematics and it was introduced first by Euclid in ancient Greece. Euclid formulated a set of axioms, simple common truths like the one which says it is possible to draw a straight line between any two points or we can extend the line segment to the left and to the right infinitely or we can describe a circle with the center of the circle and the radius of the circle or if we have one line we can draw through a point exterior to that line on the same plane only one single parallel line to the initial one (i.e. the axiom of parallels). So these are the axioms that were introduced by Euclid and are general truths accepted as true by most of the population and starting from those, we can build through logical reasoning more advanced theories, but it's very important that those theories are derived from the initial axioms using logical reasoning. We have the same thing for Arithmetics, we have several axiomatic systems for arithmetic, the one of the most famous being the Peano's axioms:

1. 0 is a natural number.
2. For every natural number x, x = x. That is, equality is reflexive.
3. For all natural numbers x and y, if x = y, then y = x. That is, equality is symmetric.
4. For all natural numbers x, y and z, if x = y and y = z, then x = z. That is, equality is transitive.
5. For all a and b, if b is a natural number and a = b, then a is also a natural number. That is, the natural numbers are closed under equality.
6. For every natural number n, S(n) is a natural number. That is, the natural numbers are closed under S.
7. For all natural numbers m and n, m = n if and only if S(m) = S(n). That is, S is an injection.
8. For every natural number n, S(n) = 0 is false. That is, there is no natural number whose successor is 0.
9. If K is a set such that: 0 is in K, and for every natural number n, n being in K implies that S(n) is in K, then K contains every natural number. [The Induction axiom]

Hence, we start with a set of simple axioms and we derive all the arithmetic theories. The Greeks brought their contributions to mathematics and revolutionized mathematics, then the Macedonian conquered the Greeks, under Macedonian ruling, the Greek mathematics still progressed because the Macedonians allowed Greek cities to rule themselfs and then the Roman empire conquered the Greeks. During Roman ruling the mathematical progress stopped evolving, along came the middle ages when people were more concerned with wars and how to survive and there weren't important science development during the middle ages in Western Europe. But the Indians carried on with the mathematical thinking and then, when the Arabic population conquered most of Europe, thy went to Asia, conquered the Indians, the Mongolian and so they adopted the knowledge from the Indians and brought it back to Europe. Starting with the renaissance period, the mathematical science continued to evolve, there were new theories added to mathematics, there was Pierre de Fermat in 1600, Isaac Newton and Gotfried Leibnitz in the 1500s who introduced the infinitesimal calculus, there was Leonard Euler in the 1700s and throughout these years, Mathematics became a powerful science capable to explain many of the world phenomenons. Most mathematicians assume that starting with arithmetics axioms (although this is true with respect to all areas of mathematics), we can derive from them more complicated truths and using those truths, we can derive more complicated truths and so on. They implicitly thought that if a mathematician is wise enough, he/she can prove this by deriving the mathematical sentence from those set of axioms or arriving to a contradiction starting from the initial set of axioms (which would disprove the initial statement). This means that mathematics is complete. The mathematicians considered this to be an implicit truth, but sometimes during 1800s mathematicians challenged this and tried to to logically prove this. They started asking themselves whether this can be said about any possible statement in arithmetics. So they wondered whether given a random arithmetic statement, whether we can say about this that it is true or it is false meaning we can derive this statement from the set of axioms or we can get to a contradiction starting from a set of axioms. So every mathematician assumed this is true, but nevertheless, they tried to prove this. Another way of describing this problem is to prove that the axioms of arithmetic are consistent. This is the reason this problem lies here on David Hilbert's list because it's not so easy to prove that the arithmetic system is consistent. Kurt Goedel, an austrian mathematician and philosopher who lived in Austria and then moved to Princeton US, proved the incompleteness theorem in 1931. He proved that the axioms of arithmetic are not complete. He called it “Einscheindung's problem” or the problem of decidability (i.e. deciding whether the axioms of arithmetic are complete or not). Kurt Goedel proved that there are mathematical assertions for which we cannot prove that they are either true or false starting from that set of axioms using mathematical thinking and this is not linked to the mathematical abilities of the person that tries to prove this. In 1931 he proved this result in a conference, but few mathematicians actually believed it, even fewer mathematicians actually understood the demonstrations which was quite complicated. So it took some time until Kurt Goedel's result actually got accepted by the mathematical computing. By the way, he proved this theorem by offering a counter example, meaning he offered an example of a mathematical statement which he proved that cannot be derived from the initial set of axioms and he can also not disprove that statement. Maybe you can imagine that the counter example he had given looks simple, but it was very abstract, it was defined using just symbols. Another proof of the fact that arithmetic system is not consistent was later given by Alonso Church who was Alan Turing's advisor from the United States . And yet another demonstration of the incompleteness theorem belonged to Alan Turing.

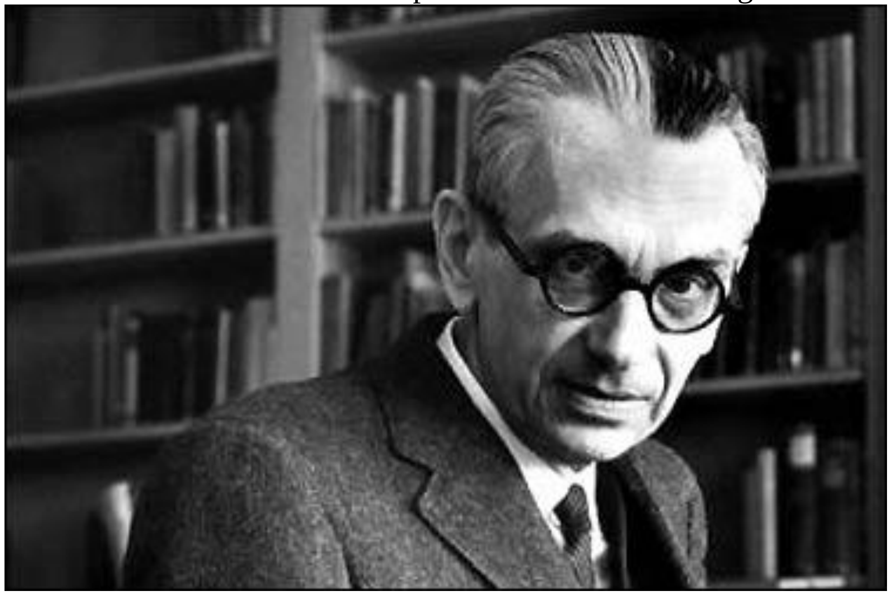


Fig. 2.13 Kurt Goedel

Kurt Goedel actually proved two incompleteness theorems the first one is that no consistent system of axioms like the aromatic system is capable of proving all available truths in the system so there are statements there are mathematical assertions or there are assertions about the natural numbers that cannot be proven correct within the arithmetic system. And the second theory is that the consistency of an axiomatic system cannot be proven from within the system, meaning there are statements that need to be proven from outside the system using something else than just the axioms of the system and this does not matter on on how many axioms you use for your system. I have told you that he lived in Vienna, in Austria, but then he moved to Princeton, United States . He was good friend with Albert Einstein.

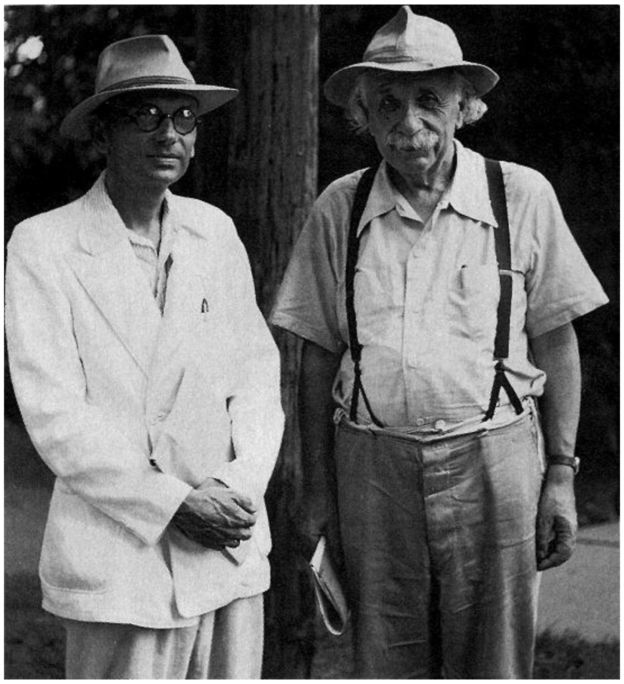


Fig. 2.14. Kurt Goedel and Albert Einstein

I have told you that besides the brains, what separates these three individuals (i.e. Kurt Goedel, Alan Turing and John von Neumann) from the rest of the world is other particularities. Goedel was a very introverted person, a very shy person. He didn't have many friends and one interesting fact about this is that he lived in Princeton and after the second world war there was the tensions between the United States and the communist Russia, there were all sorts of conspiracy theories flying around in US that the russian communists would come to United States to make the United States of America a communist nation and so on. At that time there was the beginnings of the cold war between the United States and Russia. There were spy stories all around and Kurt Goedel only ate food prepared by his wife, he only trusted the food that was prepared by his own wife because of his fears that the food might be poisoned by russian communists. So when his wife died, he refused to eat anymore and he died of starvation and when he died, he weighted about 32 kilograms.

The second man that layed the theoretical foundations of Computer Science was Alan Turing. Alan Mathison Turing lived between 1912 and 1954 in England and United States , besides the fact that he was really smart, his particularity (which was uncommon at that time) was the fact that he was homosexual and at that time it was illegal to be homosexual in England. And actually this is what partially caused his death, but I will get to that later. By the way, there is a nice movie from five years ago called “The imitation game” about the life of Alan Turing. Alan Turing was interested in science at a very young age. He liked mathematics and physics and he didn't like so much social sciences like literature, history and so on. But the focus on the public Britain schooling system at that time was on social sciences, art and literature and what we inherited from the ancient Greeks. The headmaster of the Sherborne school where Alan Turing studied when he was a a child said to his parents “If he is to be solely a Scientific Specialist, he is wasting his time at a Public School.”. Meaning he could just learn the scientific part home with some tutor in order to be an engineer or something like that because the focus of public schools was on social science sciences. He really loved school, there's one story about his first day of school which coincided with a general strike of railway and train workers and he couldn't get to school by train so he rode a bike for 97 kilometers. He started the journey one day before the school started and he spent the night at the local inn, but he got there in time when the school started. So this is to show that he really loved school. He showed stunning abilities for mathematics. By the age of 15 he could solve advanced mathematical problems without studying elementary calculus, that's the calculus which involves derivatives and the infinitesimal calculus. At the age of 16 he could understand Einstein's theories of relativity, the special one and the general relativity one and from 1931 to 1934 he studied at King's College in Cambridge. After his master dissertation in 1935 at the age of 22, he was elected as a fellow of King's College meaning he started to teach there. He proved in his dissertation the central limit theory in probability. Then he wrote a paper “On Computable Numbers, with an Application to the Entscheidungsproblem” in 1936 in which he proved Goedel’s theorem using Universal Turing Machines. The Universal Turing Machines are just abstract computers and actually the computers that we use today like my notebook and your notebooks, they all follow Turing and John von Neumann's architecture. In 1936 he moved to Princeton and in 1938 he obtained a PhD in mathematics under the supervision of Alonzo Church. Alonzo Church also invented lambda calculus and he established the roots of functional programming. John von Neumann wanted to hire Alan Turing as a postdoc assistant, but Alan Turing left for England. Now let's talk about this article “On computable numbers with an application to the einscheidungsproblem” from 1936. Turing transformed Kurt Goedel’s decidability problem into the halting problem and he offered another proof in this way of Kurt Goedel's incompleteness theorem. The decidability problem is whether we can decide/prove that a statement is true or false. Turing's halting problem is to say whether we can say or prove that any program-input data pair eventually terminates or not. The Universal Turing Machine is a mathematical model of an abstract computing machine (i.e. a computer). Is just an abstract computer that would have a brain and then will have a strip of tape and on this tape there are symbols and this tape is the input tape and also the output tape and the machine will just read one symbol at a time from this tape and depending on the internal state of the machine it will move to a different state and then advance on the input tape or it could write a symbol on the output tape. Alan Turing proved that the halting problem of the universal Turing Machine is equivalent to the decidability problem and he proved that the halting problem is undecidable. He proved this by reduction at absurdum, he assumed that it is true and he then showed that we get a contradiction.

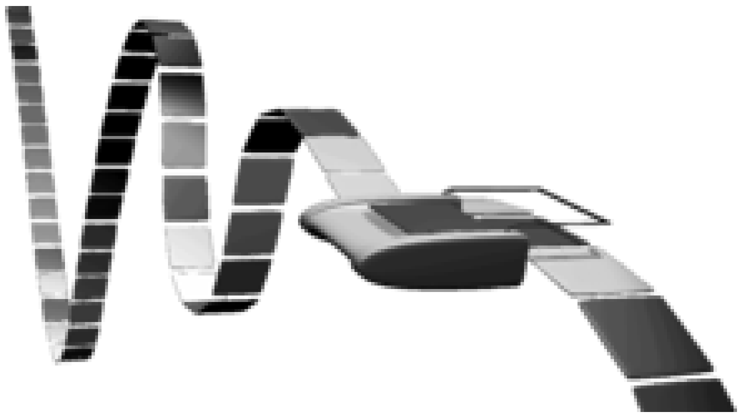


Fig. 2.15. The Universal Turing Machine graphical concept [17]

The formal definition of the Turing Machine is: A Turing machine is similar to an automaton , it is a tuple $M=(S, A, b, \Sigma, T, s_0, F)$ where:

- S is a finite, non-empty set of states
- A is a finite non-empty set of tape alphabet symbols
- $b \in A$ is the black symbol (the only symbol allowed to occur on the tape infinitely often at any step during the computation)
- $\Sigma \subset A \setminus \{b\}$ is the set of input symbols (that can appear on the input tape)
- $s_0 \in S$ is the initial state
- $F \subset S$ is the set of final/accepting states
- $T : (S \setminus F) \times A \rightarrow S \times A \times \{L,R\}$ is a transition function, L is a left shift, R is a right shift. If T is not defined on the current state and the current tape symbol, then the machine M halts.

The mathematical definition of a Universal Turing Machine is very similar to an automaton which is studied in the formal languages and compilers domain. An Universal Turing Machin would be a tuple which is formed by a set of states which define the automaton and the Turing Machine moves from one state to another one, it has an alphabet of tape symbols, it has a black symbol which is just an empty symbol, it has the set of input symbols which is just the alphabet without the black symbol (those input symbols appear on the input-output tape) and then it has a an initial state just like an automaton and a set of final states. The execution of the Turing Machine is completed by reaching one of the final states and the transition function specifies the movement from one state to another one and performs the left shift on the tape or right shift on the tape.

Turing actually proved three incompleteness theorems. He first proved a theorem that would establish an equivalence between the decidability problem and the halting problem and he said that if there were a method/algorithm to prove that the statement S is true or false, then a Turing Machine M would be able to execute this method/algorithm and would eventually terminate with the result true or false. If the statement is true, the algorithm would terminate with the result true, if it is false, the algorithm would terminate with the result false. So this is the equivalence between the decidability problem and the halting problem. Then he proved the second theorem which says that there is no Turing machine M that can say for any arbitrary (computer program – input data) pair whether this program will finish running or will continue forever. For most of the pairs computer program and input data, the Turing Machine M can say whether the execution terminates or not, but for some of them, it cannot say. Let's consider the example: if there exist a function **halt(p)** that determines whether program p terminates for any **p** (**halt(p)** returns true or false), consider the following **p**:

```
function p() {
      if (halt(p)==true) { loop_forever() }
}
```

Then the function halt(p) gives contradictory results for the above p( ) example.

In the above example, there is a function halt(p) that determines whether the program p terminates for any program p. So halt(p) returns true if the program terminates or false if it

[17] http://www.wikipedia.org

doesn't terminate. Let’s try to evaluate halt(p). Let's assume that the actual program p terminates. If p terminates, then halt(p) is of course true, but in that case there's a loop forever and halt(p) returns true although the program does not terminate because it loops forever. Now if we assume that p does not terminate, in this case, when we evaluate halt(p), it is not evaluated to true because it doesn't finish. So halt(p) should return true although we assume that p does not terminate, which gives contradictory results. And this is how, in just a very short summary, Alan Turing proved that there are statements, i.e. computer program input data pairs for which the Turing Machine cannot say whether its execution terminates or not.

Some other important facts and important accomplishments of Alan Turing were related to the second world war namely to British intelligence and decrypting german encrypted messages. During the war much less than today the atmospheric space was filled with radio messages, today even more now with the wi-fi networks and cellular networks. Today we have a lot of electromagnetic waveforms around us. Nowadays the atmospheric space is very crowded with electromagnetic waves, but during the second world war even if it was not that crowded, there was still radio communication used for the actual radio service, but also for sending messages from the army headquarters to the troups on the field. Let's say the German army headquarters in Berlin wanted to send commands and informative messages to ships in the Atlantic Ocean or to the infantry fighting in Europe and the same thing did the British and the Allied forces and of course since the whole message is sent through the air using radio waves, you may imagine that the enemy would detect the radio waves and would get the message. So it's very important to encrypt the message and the Germans used a very famous encryption system called the German Enigma machine and this is a symmetric cryptography system.

A quick side note here about encryption systems. There are two basic cryptographic systems nowadays, one of them is encryption with a symmetric key which means that you use the same secret key in order to encrypt the message and in order to decrypt the message and the other one is a cryptographic system with asymmetric keys meaning that you have one key for encrypting the message and you have a totally different key in order to decrypt the message. The first system, the symmetric key encryption system, is based on the fact that you start with the initial message which is composed of letters and then you do all sorts of permutations and shifts they are grouped usually in boxes and you do all those permutations and shifts linked together, but sometimes even in parallel and then you unite the results in order to get to a sequence of letters which are very different than the initial unencrypted data. And it was the same idea with the Enigma machine. Even if you talk about modern symmetric key cryptographic systems like DES (i.e. Data Encryption Standard) or AES (i.e. Advanced Encryption Standard) they are all based on these boxes of shifts and xors and bit rotations and they apply this sequentially a large number of times so that we start from an initial set of unencrypted data and we get an output encrypted sequence of characters which are totally different than the unencrypted sequence of characters. The advantage of the first system is that is pretty fast to encrypt the text and to decrypt it assuming that the encryption key is not extremely long, but even if it is extremely long it's pretty fast. The disadvantage is that of course if I want to pass a encrypted message to one of my friends, I would have to encrypt this message with the secret key then I would have to give this person the encrypted message, but also I have to give him the secret key because without the secret key he cannot decrypt the message and there's the problem that if I have to pass to the friend the secret key so that nobody finds out the secret key, then I could just pass him the secret message without encrypting it and decrypting it. So this is an important disadvantage, if I can pass the secret key to my friend and I suppose that nobody captures that key, then in the same way I could just pass the message unencrypted. This disadvantage is solved by the second encryption system, the asymmetric encryption system which is also called private and public key encryption system and which was invented by Rivest, Shamir and Adelman. The most famous algorithm of public and private key encryption is RSA. The idea of RSA is that you start with two very large random prime numbers with a lot of digits, more than 10 digits and based on these random numbers, you compute two keys, one is the public key and one is the private key and these two keys have an important property, they are complementary: you use one of the keys to encrypt the message and the other one to decrypt the message. And if we get back to the situation that I mentioned before, so I have a friend and I want to send an encrypted message to that friend, then my friend can generate a public and a private key and he would keep the private key to himself as a secret and he would show the public key let's say on his website for everybody to see. I can take this public key, I encrypt my message with the public key and I show my message on my website publicly to everybody to see because nobody can decrypt it if they don't have the secret key which is the private key kept by my friend. My friend can see the message and can decrypt the message using his private key. In this case there's no problem, we avoid that problem of me sending him some secret stuff like the secret key and then sending him the secret message, i.e. the encrypted message. Because we are using very large numbers, it's very hard to do a brute force attack on this public-private key system meaning that in order to check all the combinations, scientists assumed that it would take the current computing power hundreds of years to compute all the possibilities in a brute force attack. Of course, there are legends about NSA, the National Security Agency in the United States who supposedly have this powerful quantum computer and they can decrypt messages encrypted with public-private keys in days, but those are just legends. RSA is used in emails encryption, the HTTPS protocol, internet banking. So RSA is used in a lot of services today and if we wouldn't have RSA, then none of these services would be possible. An important note is that actually RSA is hard to compute, so RSA is used only in the beginning of the communication in HTTPS to establish and encrypt a symmetric key and then the symmetric key is used further because it's a lot faster to encrypt and decrypt using that one. That symmetric key is refreshed from time to time during the lifetime of that communication session.

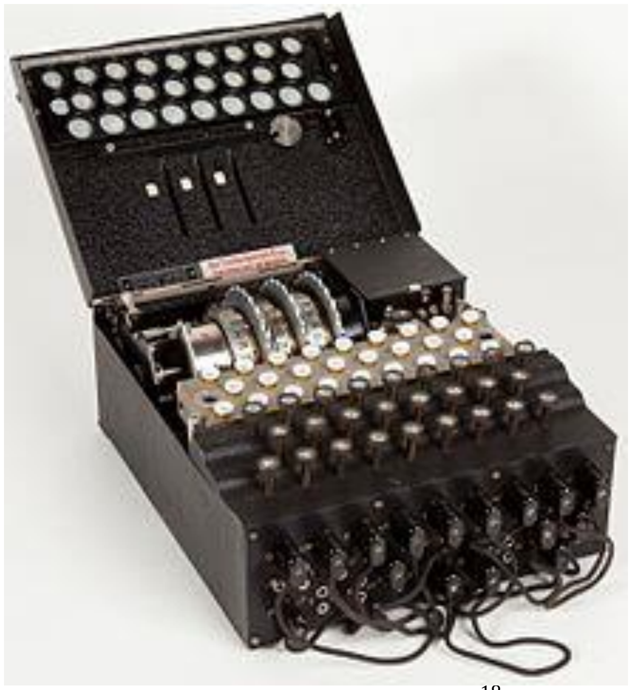


Fig. 2.16 German Enigma[18]

Enigma was a symmetrical cryptographic system and the Polands actually cracked the enigma system. The Poland spies captured a bunch of Enigma machines and managed to crack it in 1932. But then the Germans produced the second generation Enigma which was a lot harder to crack and nobody could brute force attack it. Because Poland and the British forces were allies in the second world war, the Poland shared their knowledge of Enigma with the British spies in 1939 and this is how Alan Turing got to know Enigma. What Alan Turing's teams what in Bletchley park did was to invent a electromechanical computer called the Bomb (by the way, the Poland machine that would decrypt the first Enigma was also called the bomb) that would simulate all the possible combinations of Enigma (i.e. it would perform a brute force attack) very fast and most of the combinations would get to a contradictory result, so using Turing's bomb one could simulate all Enigmas possible states and you can you can rule them out in a matter of hours and then there were still a few cases that needed to be checked by hand and that thing could be done a couple of hours by a humans. Imagine that the British forces would capture the radio messages at night and they would need a decrypted message by morning so it was more than enough time for the British to decrypt the German messages. A bombe contained 26 Enigma simulators and can execute jobs in parallel.

[18] http://www.wikipedia.org

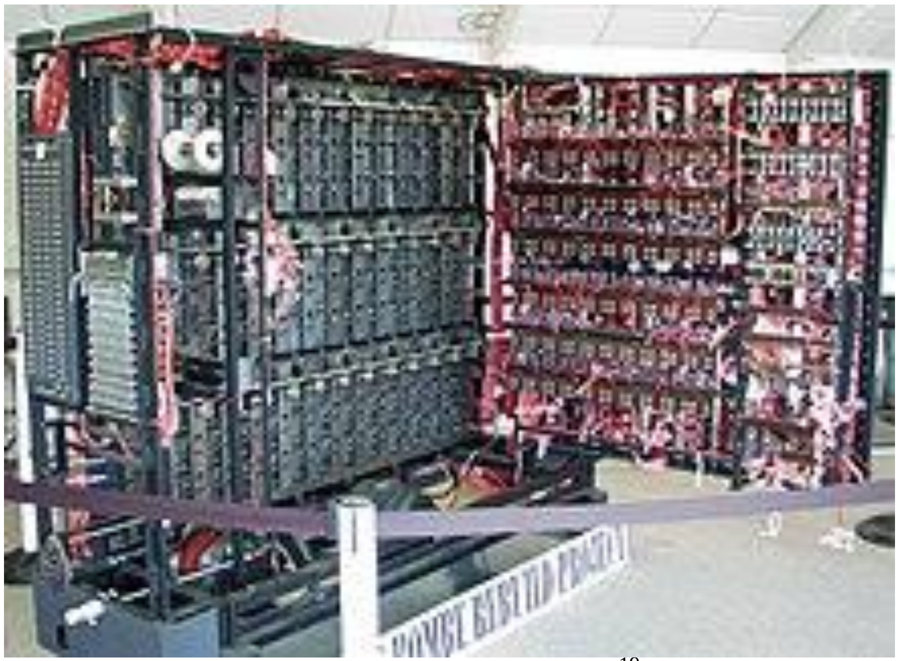


Fig. 2.17. Turing's Bombe[19]

The Enigma chiper is a German encryption machine which scrambles the 26 letters of the alphabet and it has in one variant with four rotors 10 at the power of 22 possible rotor setting meaning that you would have to brute force and test all this 10 at the power of 22 possible settings and this is what the bomb did. It would simulate all these settings but very fast. The bomb would not simulate all these states completely, meaning the bomb could rule out a lot of this possible state as impossible. When a key is pressed one or more rotors rotate and each rotor had 26 letters and it had 26 contacts which would connect to the electrical contacts of the next rotor and so on. So a key is just an electrical pathway through this system of rotors.

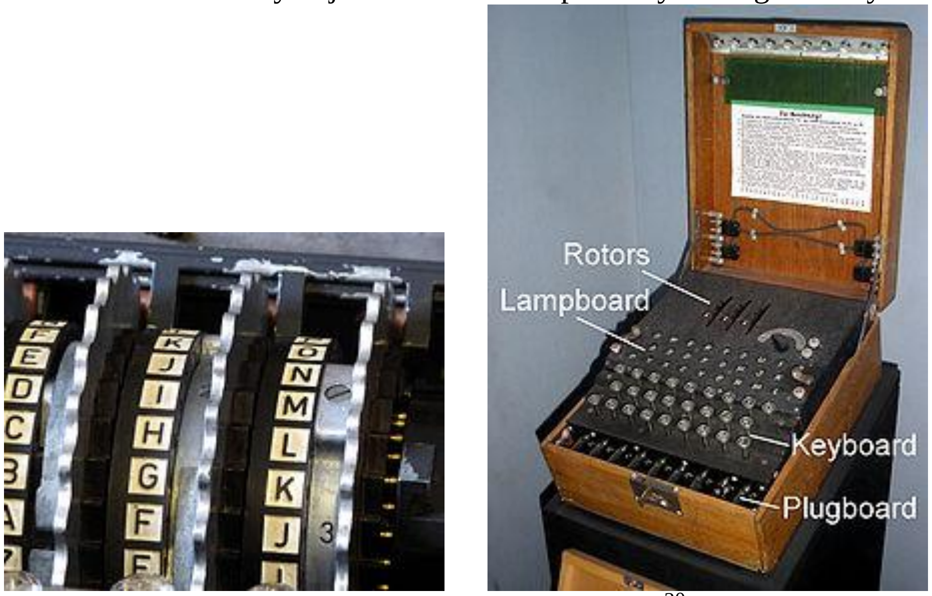


Fig. 2.18. Enigma rotor system[20]

[19] http://www.wikipedia.org

The idea of the bomb is to use a text which is already known to be present in the encrypted text as examples of such text are ‘Heil Hitler’ which would be a text that would be present in all german encryption messages or weather because usually you send weather reports to ships and to forces. Alan Turing assumed that this text will always be present in the encrypted message and using this assumptions, he was able to rule out a lot of possible combinations of Enigma. It is assumed that Turing’s bomb and the decryption work of Alan Turing in Bletchley Park has shortened the war by two years and saved thousands of lives.

Alan Turing also devised several other electromechanical computers while he worked at Bpetchley Park after the war ended. Alan Turing also invented the famous Alan Turing artificial intelligence test which is not met by any artificial intelligence machine today. The idea is to have two rooms separated by a screen and the person from one room cannot see/interact with the person from the other room. But the first room has a terminal like a monitor and a keyboard and if a person comes in to the first room, he/she could write questions using the keyboard for the person or the machine sitting in the second room (that the first person cannot see). The person or the machine in the second room writes answers and those answers appear on the monitor in the first room. The idea is that if the first person in the first room asks questions and if in the second room there's a machine who answers the first person questions and the first person doesn't realize that it's a machine answering (and not a human), then this means that that machine has artificial intelligence. This is like a reverse captcha test if you know those captcha tests used on the web that ask the user to compute a very simple addition or just say that this is a bridge or a plane or whatever - this is a test to check whether there's a human user behind the computer. Some other notable facts about Alan Turing are that he was a very good athlete and he won several athletic contests. I already mentioned that Turing was homosexual and it was illegal in Great Britain in that time to be homosexual and some thieves broke into his house and he had to call the police and the police somehow discovered that he was homosexual, so he was sentenced to either do prison time or to accept a hormonal therapy in order to decrease the homosexual traits. He accepted the hormonal therapy, but it was making him very sick and he died in 1954. He died of cyanide poisoning, it was most probably suicide, but no one really knows for sure whether it was suicide or someone killed Alan Turing. In 2009 the British Government and the Queen issued a public pardon for the way the British Government treated Alan Turing during the war and the most famous prize in computer science has the name of Alan Turing. The Alan Turing prize is the most famous award issued by ACM (Association for Computing Machinery); it is like the Nobel prize in computer science or the Fields medal in mathematicians. Also about Turing's legacy, Turing's nephew Dermot Turing makes public presentations all over the world about Alan Turing's work and he said that Alan Turing probably committed suicide because he felt like he was losing control control over his body, control over his life and he probably committed suicide. In Fig. 2.19 you can see Dermot Touring on the scene of the Students Cultural House in Cluj-Napoca in 2017 giving a public presentation at the “IT days” conference. He will probably come again to Cluj Napoca.

[20] http://www.wikipedia.org

Fig. 2.19. Dermot Turing giving a public presentation in Cluj-Napoca, 2017

Now we move on to John von Neumann. He lived between 1903 and 1957. He was an Hungarian-American mathematician. His original name was Janos Lajos, but he changed his name to John von Neumann when he moved to the United States . He was jewish and he moved to the United States because of the Holocaust and the nazis action against jewish people. He was a mathematician, physicist, computer scientist, engineer, polymath, a very bright person. He made a large number of important contributions even if he lived only 50 years. He had contributions in foundation of mathematics, ergodic theory, functional analysis, statistics and many more, but also in physics, quantum mechanics, nuclear physics, economics game theory and computing. He was renowned for his instant intelligence, I mean he was able to instantly compute something or discover something. I have told you when Kurt Goedel presented his incompleteness theorem and the proof of the incompleteness theorem at the conference, a few mathematician actually thought it was true and few of them actually understood the demonstration. But John von Neumann was present at that conference and immediately said corollary of that incompleteness theorem. So he immediately realized something that even Kurt Goedel didn't thought of it. He published over 150 papers, he was part of “the martians” – a group of Hungarian mathematicians that fleed Europe and moved to United States to universities because of the nazi occupation. The martians group was made from: Paul Erdos, Paul Halmos, Theodore von Karman, John G. Kemeny, John von Neumann, George Polya, Leo Szilard, Edward Teller, and Eugene Wigner. Among them, there was Paul Erdos who is very famous for his large number of scientific collaborations. He created the Paul Erdos number which is a graph that links scientific authors who collaborated (in paper writing) with Paul Erdos and he defined the Paul Erdos number of a scientist as the path distance in this graph from Paul Erdos to the vertex of that scientist. Paul Erdos had the number 1. Another martian was John Kemeny who taught at Dartmouth College and laid the foundation of the first programming languages. Then there was Leo Szilard which made important contribution to mathematics, then DNA and biology. John von Neumann was born in Budapest, he was a true child prodigy. At age of six he could divide two eight digit numbers in his head and could speak ancient Greek. By the age of eight he was familiar with differential and integration integral calculus.He studied at public Lutheran Fasori Evangélikus Gimnázium, but also had private tutors. At age 15 he studied advanced calculus with the renowned Gabor Szego and on their first meeting he was brought to tears by the boy’s mathematical talent. By age 19 he published 2 major math papers, in one he gave the modern definition of ordinal numbers (which superseded Cantor’s definition). In 1923 he entered Pázmány Péter University in Budapest as PhD candidate in mathematics and ETH Zurich as chemical engineering student (to please his father because his father wanted him to be an engineer). He completed his habilitation in math in 1927 and started lecturing as privatdozent at Univ. of Berlin in 1928 (the youngest person ever elected in the history of the university on any subject). By 1929 he published 32 major papers in mathematics (at a rate of nearly 1 major paper per month in the last 2 years which is mind blowing). In 1929 he moved to Princeton University, US and in 1933 he was offered lifetime professorship. John von Neumann also participated to the Manhattan project. The Manhattan project reunited a set of preeminent american scientists, mathematicians and physicists and so on like John von Neumann Richard Feynman, Albert Einstein etc. and they suggested the American president Roosevelt to seriously consider using the atomic bomb on the Japanese because the Americans faced the threat of maybe losing the war or if not losing the war, faced the threat of having a very long war. Then president Roosevelt finally used the atomic bomb on two Japanese cities, Hiroshima and Nagasaki. The team of scientists involved in the Manhattan project performed various nuclear tests at Los Alamos and used computer simulations (ENIAC, a computer we will talk about a subsequent section) which showed that the blast effect of the atomic bomb would be larger if the atomic bomb would detonate above the ground not when it hits the ground. And that's why those two bombs, Fat John and Little boy, that were launched over Hiroshima and Nagasaki actually exploded 100 meters above the ground to produce more destruction. John von Neumann died because of the nuclear test he participated in Los Alamos because no one knew many things about nuclear radiation and he died of bone, pancreatic and prostate cancer. He died in pain in 1957 under military surveillance, 24 hours per day due to all the secrets he knew. He invented the Merge Sort algorithm in 1945. He was a consultant to build the important ENIAC and EDVAC American supercomputers. While he was consulting for EDVAC, he described the computer architecture where the program and the data are stored in the memory, both of them. He described this architecture which now we call the Turing-von Neumann architecture. This is the modern computer architecture that we have today for my notebook, for the tablet or phone etc. The computer has a memory and we store here the program and the data, has the control unit which set which establishes what should be executed next, it has an arithmetic unit which computes arithmetic operations and an input and an output (see Fig. 2.20).

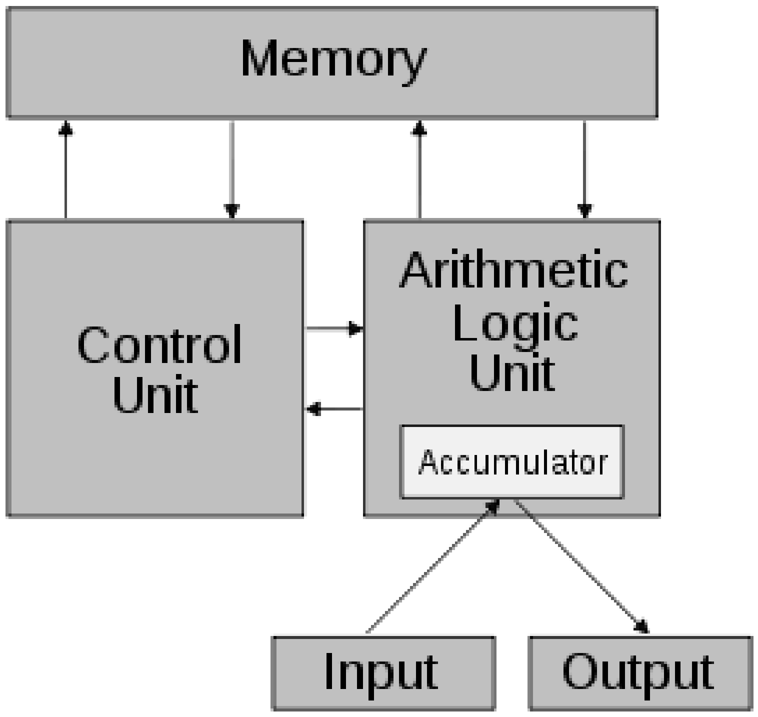


Fig. 2.20. The Turing-von Neumann computing architecture[21]

Some fun facts about von Neumann was that he could quickly memorize whole pages from the telephone book and he would entertain friends by reciting addresses. He would just look over two pages from the phone book for a couple of seconds and then he would close it and was able to recite from his head any number or any address that was present on those page books. Then he was socially active. The weirdness of John von Neumann was that he used to drink and he was also - I don't know the term in English – "a women's man". He was a very bad driver, but because he was well paid, he used to come with a new car at the beginning of each university year. He had a lot of accidents and one of the accidents was reported to the police like this: he said that well it was such a nice weather, I used to drive very peacefully and all the trees on the right side would pass in an orderly fashion one after another on the right side and suddenly one of them jumped in front of the car. So he really didn't care much about these things. He used to eat and drink, he used to throw parties at his house in Princeton every weekend - actually there were a significant number of scientists that were complaining about the loudness coming from Neumann's house, one of them being Einstein. But he liked this loud environment, he could think very well in this loud environment playing loud music, he would he was married two times, he took great caring his clothing, always wore formal suits.

[21] http://www.wikipedia.org

Fig. 2.21. John von Neumann

History of primitive computing devices

Charles Babbage designed the differential engine and the analytical engine around 1820 to 1840 and those two, especially the analytical engine were the first general purpose mechanical calculators. This means that they could compute the four basic arithmetic operations: addition, subtraction, multiplication and division, but also more complicated mathematical expressions like evaluating a polynomial, exponential and logarithmic function and also, at least the analytical machine, had the ability of performing conditional instructions, loops, had input on punched cards and had an operating language. But simple mechanical computing devices equivalent to pocket calculators (i.e. they could perform only additions, subtractions, divisions and multiplications) that existed before Charles Babbage. Actually, simple computational devices existed even in the ancient history. And this is the subject of this section. And although we will generally focus on mechanical calculators that existed before Charles Babbage, i.e. before 1800s, we will still go after 1800s until about 1960 just to see how they evolved, but still restrict our discussion to basic calculators that could compute the four basic arithmetic operations. Actually, the oldest devices were just tools that assisted humans in performing computations and some of them didn't compute any arithmetic operations, but were used to predict the positions of the moon or the sun, to predict the next solar eclipse and other astronomical events.

Let’s start with various ancient devices used for computing in ancient history. One of the oldest one is the abacus which was used in 2400 BC. The babylonian used it, the romans used it and they used it in order to perform arithmetic computation. They used it in commerce, but also in engineering, in computing taxes. It worked in base 10. They have rods of iron on which there are several balls placed and you can move them around. It's just like computing additions with the help of your ten fingers, but this time you have more than ten fingers because you have a lot of balls that you can move around from left to right in order to add numbers.

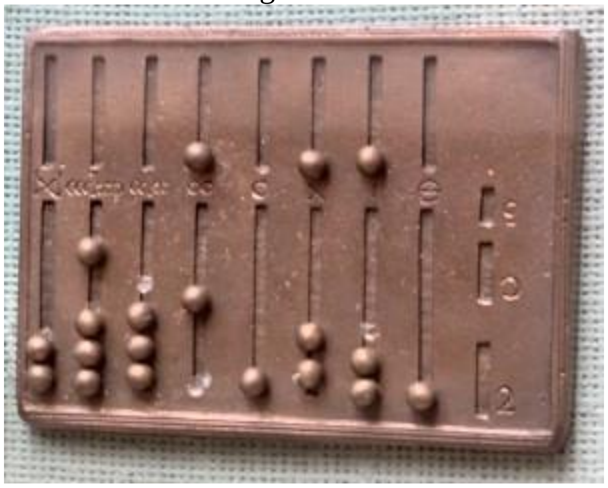


Fig. 3.1 Abacus, 2400BC

Another very old computing device is the Antikythera mechanism that was used by ancient greeks around 150BC and 100BC. Few things are known about this mechanism. It is considered the first analog computer. We don't know a lot of things about it, because it was discovered by scuba divers near some islands in Greece and they actually found a stone on which there were imprinted a couple of metal wheels, but that's all we have. We know that it predicted astronomical positions, follow moon’s movements, predict eclipses. It's a complex clockwork having 37 gear wheels, the one we know about. You can see in Fig. 3.2 a graphical representation of it although it's not an accurate representation. It was able to compute the position of the sun in a quite complicated way because it used the old, inaccurate, egocentric system where the earth was in the center of the universe and the sun and the planets rotated around the earth, which is wrong, but it was the common conception at the time. This is the first analog computer meaning it doesn't represent data using numbers, but it represents data using a wheel movement.

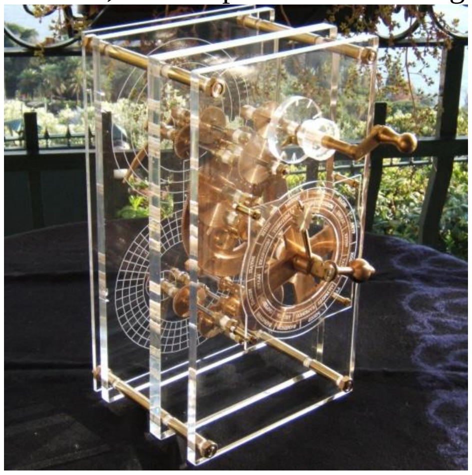


Fig. 3.2 Antikythera mechanism, 150BC - 100BC

Another very important device is Al-Jazari's castle clock. Actually, a lot of these computational devices that computed time and computed solar positions and moon positions were built in towers in castles. Al-Jazari’s castle clock was built around 1206, it works with water and it is able to display the zodiac, the solar position and the lunar orbits. It is able to compute the time and whenever there was a fixed hour there were a bunch of robotic figurines, little soldiers that would exit the tower and move around in front of the tower and then go back inside. Also, there was music during this operation. The device was programmable, the earliest programmable analog computer. You could program the length day and night every day. Al-Jazari’s castle clock had a gold zodiac dial breathtaking bronze birds on its battlements and fine group of mechanical musicians at its gate. The hours since sunrise were shown by the open doors on the clock front as well as by roundels that lit up the passage of time. The minutes were reflected by the progress of a crescent moon across the clock's freeze. At its top a large dial mimicked the movement of the moon and stars across the sky as the sun rose and set within. The clock movements were regulated by an ingenious system of reservoirs, troughs and pulleys. The clock central mechanism was a reservoir from which an accurately calibrated flow of water was allowed to seep into a float chamber and into a plate and spout as a weighted float. Descendent within the reservoir it tugged at two pulleys one that set the cart on its motion and another that slowly pulled a sphere which in turn rotated a cut-out crescent disc to shine light from candles onto the randles and shift the sun and moon dial on top of the clock. As the cart moved along its road upon the hour a vertical shaft triggered the upper and lower doors to reveal the figure of the man and the words dominion is god. When this happened the gate on a rail behind the doors opened to release two balls which traveled down chutes and into the beaks of two gold falcons which then tilted and dropped the balls into two copper vases.

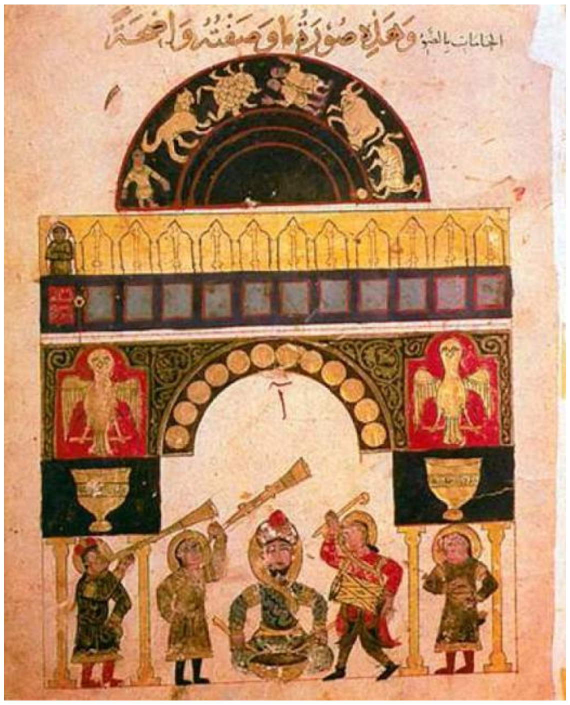


Fig. 3.3 Picture with Al-Jazari’s castle clock

We now skip through the years and we get to 1600 and talk about John Napier's calculating tables or John Napier's bones which were tables or bones actually columns or cylinders of bone material on which there were numbers imprinted and those columns or those tables could be used in order to perform additions, subtractions, multiplications and divisions. They look like Fig. 3.4.

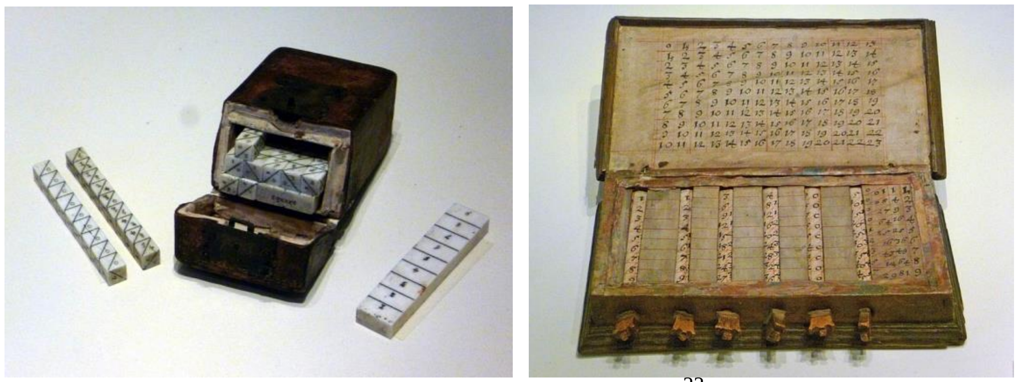


Fig. 3.4 John Napier’s bones[22]

They could perform multiplications as a sequence of addition operations and division as a sequence of subtract operations. And the way this would work is the following. I can show you an example of how multiplication was performed[23]. You would have one column with the digits from one to nine and ten columns with ten values each, each column corresponding to the digits

[22] http://www.wikipedia.org
[23] https://en.wikipedia.org/wiki/Napier%27s_bones

from one to nine. In order to compute 425 x 6 = 2550, you would do the following. Starting from the column bone with the digits and 3 other columns corresponding to digits 4, 5 and 6 (see Fig. 3.5), you would take the row of numbers corresponding to 6 and you would compute the result by adding diagonally 4 to 1, then 2 plus 3 – see Fig. 3.6. So these are John Napier's bones or John Napier’s calculating tables they were used in England in 1600s.

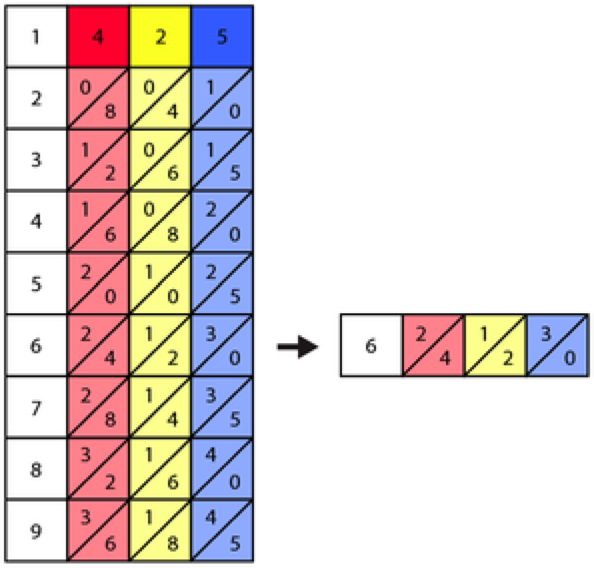


Fig. 3.5 Computing 425 x 6 using Napier’s bones[24]

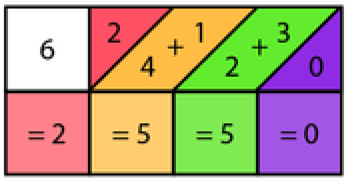


Fig. 3.5 Computing 425 x 6 using Napier’s bones[25]

The next device is the slide rule from 1620. It has real numbers represented on a row as distance on a line and it allows multiplication and division operation to be carried out faster than it was previously possible. You can compute exponents and trigonometric functions and roots and logarithms with it. You can do this by moving the mobile piece in the center. I cannot explain how it functions because I don't really know.

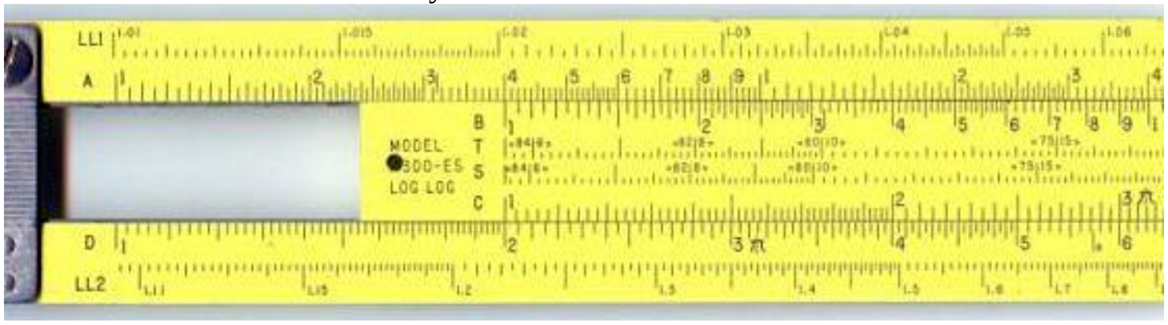


Fig. 3.6. The slide rule, 1620

[24] https://en.wikipedia.org/wiki/Napier%27s_bones
[25] https://en.wikipedia.org/wiki/Napier%27s_bones

We go to the first digital mechanical calculator in 1623. It was built by Wilhelm Schickard. It was mechanical based on wheels and rotors, but it represented data with digits the way John Napier's bones do. The machine was designed to assist in all the four basic functions of arithmetic (addition, subtraction, multiplication and division) on 2 multi-digit numbers.

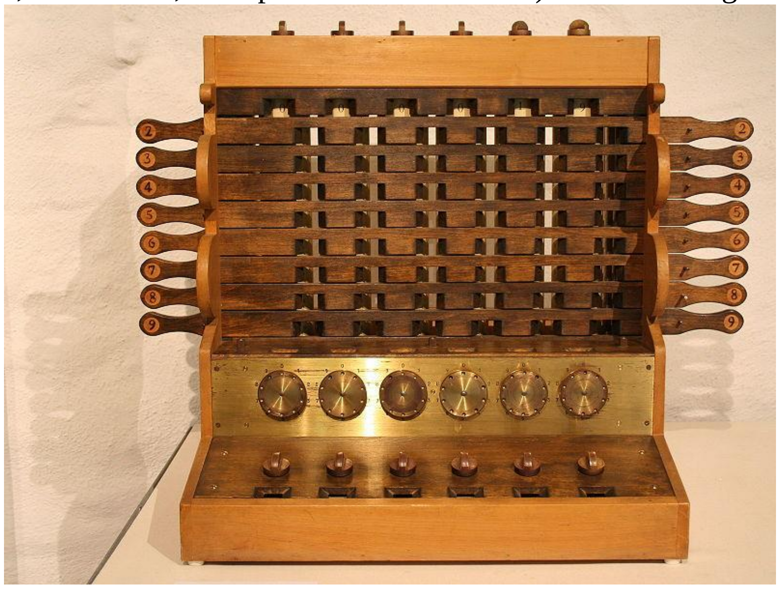


Fig. 3.7 Schickard's digital mechanical calculator, 1623

Another machine is the Pascaline built in 1642 by Blaise Pascal. It took a lot of time for Blaise Pascal to build this - about 20 years. He started working to the Pascaline in 1642 when he was 19 years old and he finally built 50 prototypes until he got to a final working version. He then built 20 more. He got the Pascaline idea as a tool to help his father compute taxes. This computing device would represent negative number as complements in different bases. It could perform additions, subtraction, multiplications and divisions through repeated additions and subtractions.

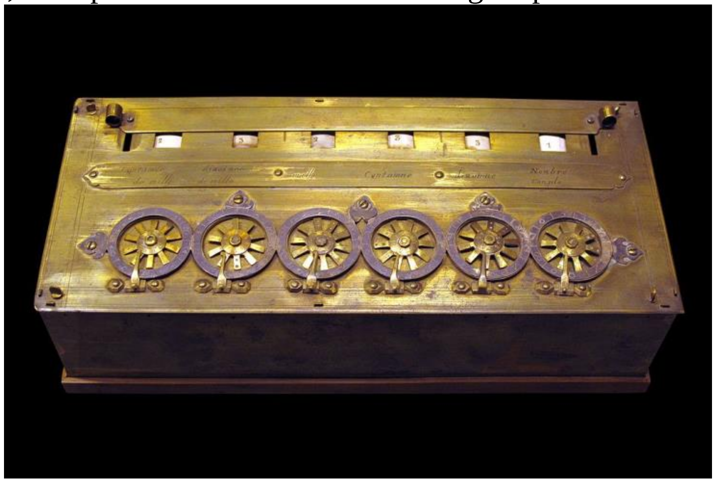


Fig. 3.8 Pascaline, 1642

Another computing device is Leibniz's Stepped Reckoner in 1672. It has a long shaft and rotors and wheels. Leibniz worked 40 years on its design and finally, he produced two working prototypes. He initially tried to add automatic multiplication to the Pascaline. Finally, the prototype could perform more complicated operations: adding or subtracting eight digit numbers to/from a 16 digit number, multiply two eight digit numbers to get a 16 digit result and divide a 16 digit number by an eight digit divisor. Multiplication and division procedure is detailed here: https://en.wikipedia.org/wiki/Stepped_reckoner

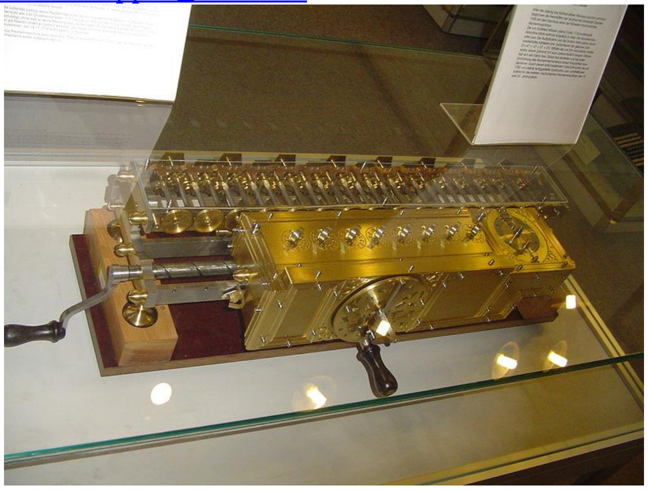


Fig. 3.9. The Stepped reckoner

The next mechanical computing device is the Thomas Arithmometer. It was built in 1820. It was the first commercial mechanical device. It was the first digital mechanical calculator reliable enough to be used on a daily basis in an office in France. A 12-digit arithmometer sold for 300 francs in 1853 which was 30 times the price of the table of logarithms book. It had various models with capacities of 10 12 16 and 20 digits. It could perform additions, subtractions, multiplications and divisions with numbers of 10 12 16 and 20 digits.

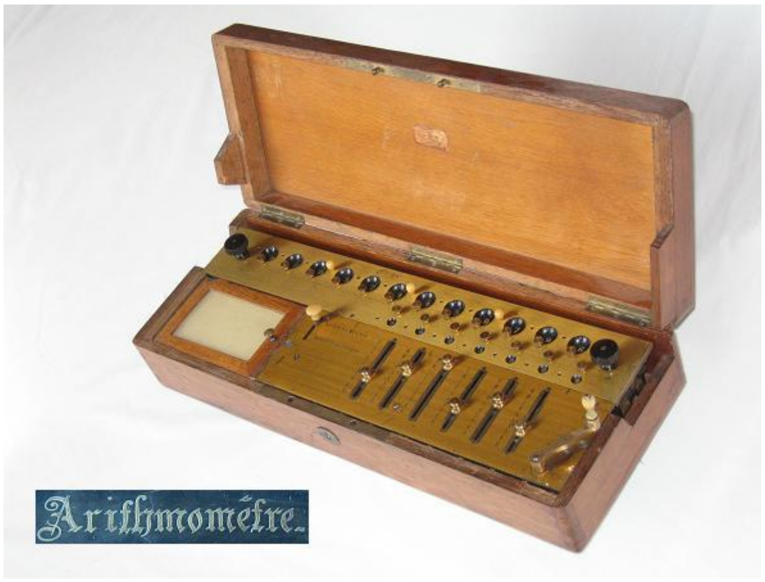


Fig. 3.10. Thomas Arithmometer, 1820

Another post 1800 mechanical calculator is the Addiator used in Berlin between 1920-1982. It was made by Addiator Gesellschaft, Berlin. It could perform naturally additions and substractions. There were also procedures for multiplications and divisions through severall additions and substractions. It looked more modern then previous calculators.

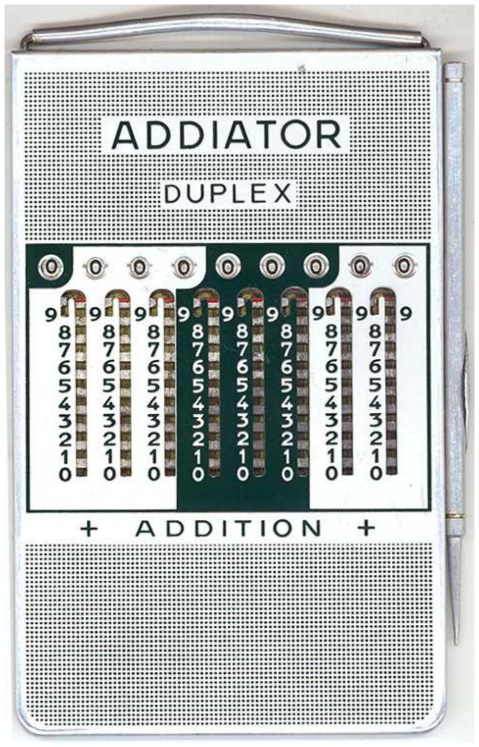


Fig. 3.11. Addiator, Berlin, 1920

Another one which looks more modern it's the Comptometer used in the United States between 1887-1970.

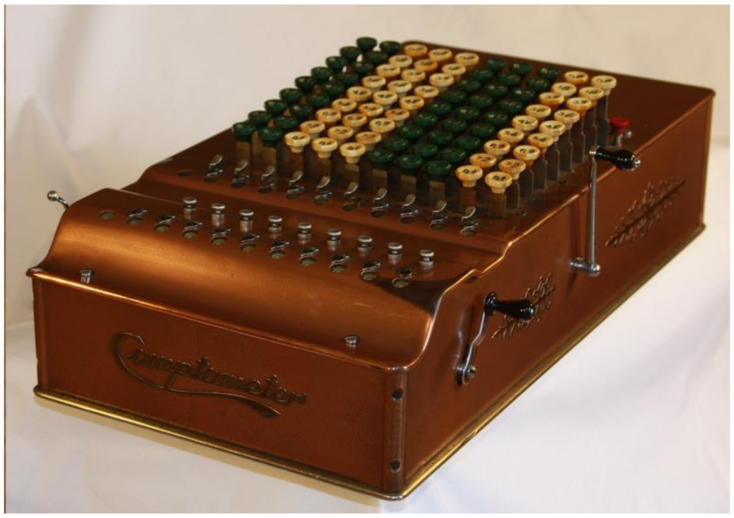


Fig. 3.12. Comptometer, US, 1887-1970

There was also the Monroe machine used in the United States between 1912-1970.

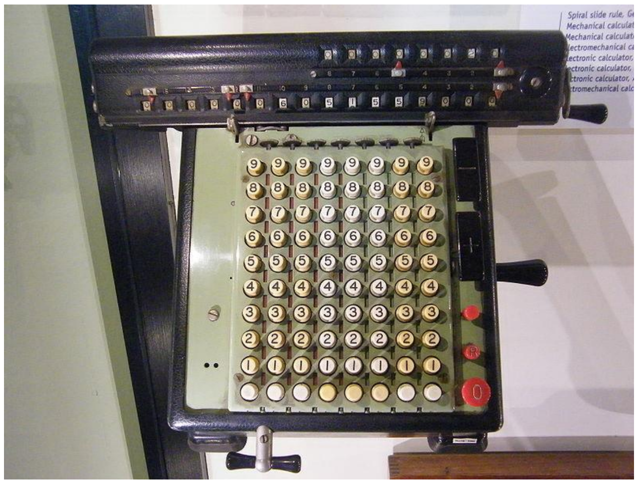


Fig. 3.13. Monroe machine, US, 1912-1970

Another device was the Curta device used in Austria between 1930-1970.

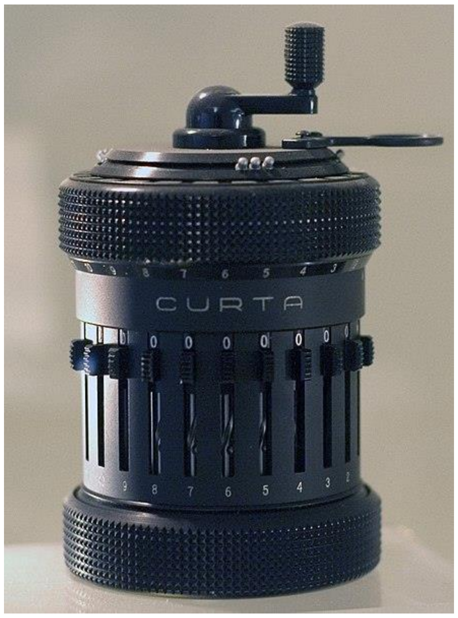


Fig. 3.14. Curta, Austria, 1930-1970

First all electronic desktop calculator was ANITA MarkVIII used in the United Kingdom starting with 1961.

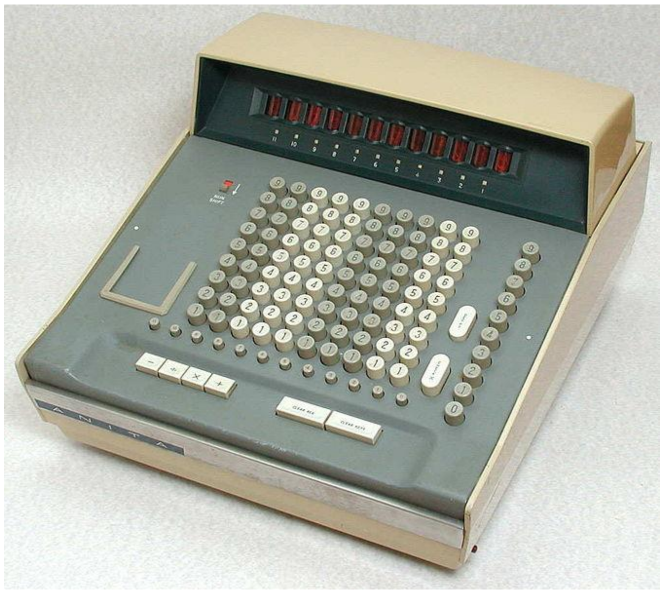


Fig. 3.15. ANITA MarkVIII, UK, 1961

Generations of Electrical Computers

I have talk about primitive mechanical computing devices that existed before 1900 in a previous section. Those were mechanical devices built using rotors and cylinders and wheels and performed mostly the four basic arithmetic operation. After 1900 general purpose electrical computers started to appear (they worked based on electricity). Today all the general purpose computers are electrical computers, but in the beginning of the 20th century the first general purpose computers also contained mechanical parts. So they were mix of mechanical and electrical parts and slowly moving towards only electrical parts. So we will start this section with mechanical-electrical computers that appeared starting from 1900 and then move to all electric computers and you will see that there are four generations of electrical computers: a) the first generation were based electrical components built using vacuum tubes, b) then the second generation was based on the transistor which replaced the vacuum tubes, c) the third generation was based on integrated circuits which are just more complex and miniaturized circuits based on a large number of transistors on the same board and d) the fourth generation is based on microprocessors - the one that started the miniaturization process where everything shrank and it led to the development of desktop computers, IBM personal computers. The fourth generation actually continues today when everything is shrinking we still have some desktop computers today, but they are slowly being replaced by notebooks which are even smaller and tablets and phones which are even smaller. This whole miniaturization process and this whole process of commercialization of computers to the general population started in 1972 once the Intel corporation invented the microprocessor.

First mechanical-electrical computers (before generation 1)

Before the first generation of computers there were mechanical-electrical computers built using mechanical parts and electrical relays. Electrical relays are still used today although quite seldomly. They are the equivalent of an electrical switch, so you can turn it on and let the current pass through it or you can turn it off and stop the current from passing through it. One of the first mechanical-electrical computers that were developed was the series of computers named Z1, Z2, Z3 and Z4 invented by the german scientist Konrad Zuse between 1938-1949. The first computer invented by Konrad Zuse is Z1. Zuse invented Z1, a mechanical computer in 1938 in his parent’s flat and using his own money. The computer never worked due to insufficient mechanical precision (similar to Charles Babbage’s problems with the differential engine and the analytical engine, nearly a century ago). It used 22-bit floating point binary numbers, it read instructions from a tape perforated 35 mm film. The computer had limited programmability, 9 instructions in total; 1 instruction took 1-20 cycles. An electrical motor gave the clock frequency of 1Hz. It weighted 1000 kg and had mechanical metal parts and a few electrical relays. The design documents were destroyed in 1944 by British bombing.

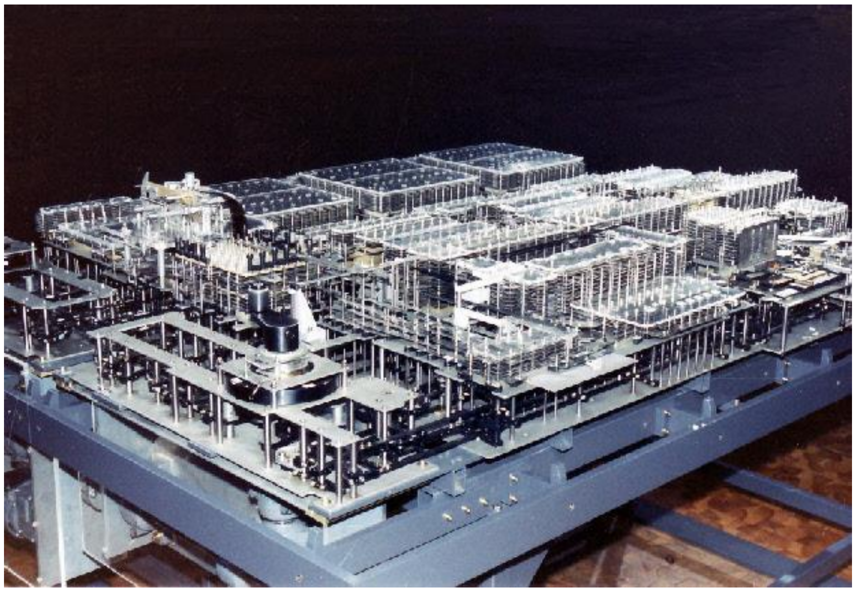


Fig. 4.1 Z1 (1938)

The second computer build by Zuse is Z2 in 1939. It is similar to Z1, but it implemented the arithmetic and control unit using electrical relays. It had the same mechanical memory with 64 words. It used 16-bit floating point arithmetic. The clock frequency was 5Hz. An addition operation took 0.8 seconds and it weighted 300 kg and consumed 1000 watts.

Fig. 4.2 Z2 with Konrad Zuse (1939)

The third mechanical-electrical computer in the series in Z3 built in 1941. Z3 was the world's first working, programmable digital computer. It used 2000 electrical relays and had a clock frequency of 5.3Hz. The Arithmetic unit could add, subtract, multiply, divide, extract square root of 22-bit floating point numbers. The program and data were stored on punched tape film. It was Turing-complete, it had loop instructions, but no conditional instruction. The addition operation took 0.8 seconds and the multiplication took 3 seconds. It weighted 1000 kg and consumed 4000 Watts.

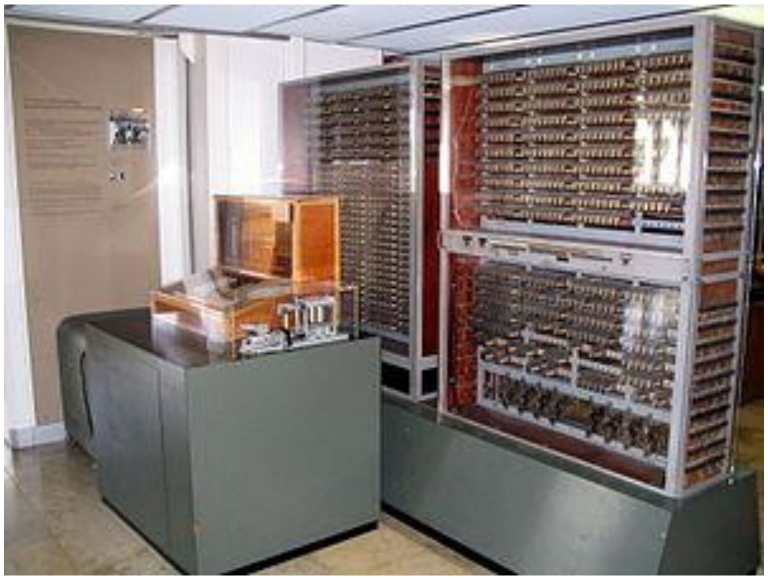


Fig. 4.3 Z3 (1941)

The fourth computer in the series is Z4 built in 1945. The computer had a frequency of 40Hz. An addition operation took 400 ms and a multiplication took 3 seconds. Input and output was provided on punch tape and programs were written on 35mm punched film. It used 32-bit floating points. It had 2.500 electrical relays and consumed 4KWatts of power. Zuse realized it is difficult to program in machine code, so he invented a programming language called **Plankalkül.**

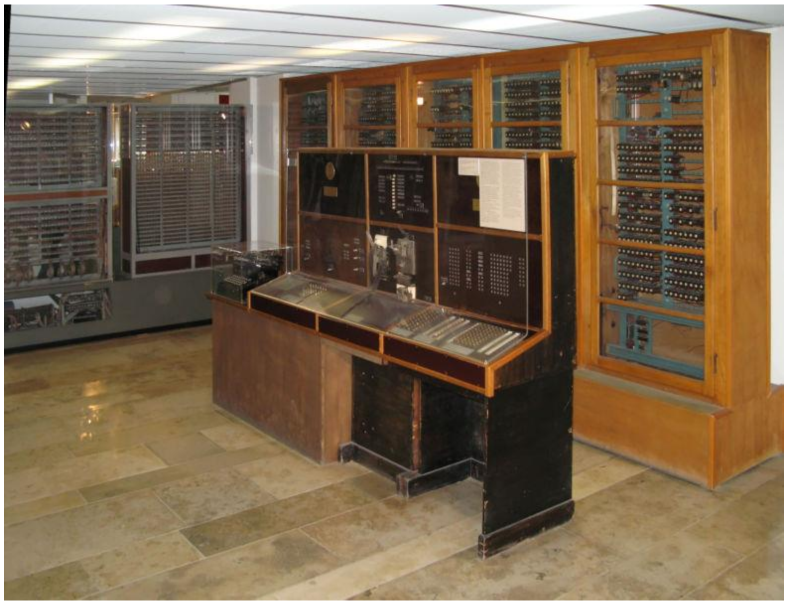


Fig. 4.4 Z4 (1945)

Since we will be talking a lot about clock frequency and computing cycle and you will see that as we advance through human history, there are better and better computers built that were faster and faster and they achieved this mostly through an increased clock frequency also through reducing the clock cycles required by one instruction, but mostly through clock frequency increase, it's very important in the beginning to actually really know what we are talking about when we say clock frequency and computing cycle. The clock frequency is just the number of beats, the number of clock signals per second. 1 Hz means that a clock or a device emits a time signal one time per second in each real second. There are electrical components called clock generators which are built inside a lot of electrical devices. You have at least one of them built in the notebook, in a phone, in a computer, in a tablet, but also in non-computer devices like a radio machine device or a television set. So a clock generator circuit just measure times meaning it sends periodic signals, electrical signals on a circuit on a board. The clock signal generators just measure time, but not the actual real time, but rather a virtual time. Clock generators that generate 8 thousand clock signals per second have a clock frequency of 8000 Hz or 8KHz. Why is there a need for a clock generator inside a computer system ? Almost every relatively complex electronic device, not to mention computer systems need such a clock generator circuit because the execution of instructions inside the computing system must be synchronized. The execution of an instruction must be synchronous, i.e. you need to have a lower, but most specifically an upper limit to the time it takes for an instruction to be executed in the computer system. Time limits are needed for the execution of every operation that happens in a computer system. Every operation in a computer system needs to be programmed precisely. You cannot have a working computer system if you have parts of it that function randomly. Let me give you an example. Every component in the computer system should know in advance how long it takes for every single operation in the computer. We have the CPU which is the microprocessor in the computer and the CPU is linked with the memory, actually usually you'd have a memory control chip that discuss directly with the memory and the CPU only talks with this memory controlling chip, but let's ignore this for a fact. Let's say that the CPU discusses this directly with the memory. And let's say we want to increment a value from a variable from the computer memory. You already know this from the assembly language courses in the first year that when I want to increment a variable in the memory actually this requires several operations not only one. First, we need to get the value from the memory and copy it onto one register from the CPU. This is accomplished with the *move* operation if you remember from the assembly language. So, first the CPU must copy using the *move* instruction the value from the memory on a CPU register, then it could increment the value from that CPU register (like the accumulator register AX, EAX) and then it should move back into the memory the value incremented that resides on the CPU register. So the first thing that the CPU needs to do is to get the value from the memory. In order to do this, it would send a command on the control bus that links the CPU with the memory to the memory which includes something like this: give me the double word (i.e. four bytes) from the memory address X. So the CPU sends this command to the memory and then it listens on the data bus that links the CPU with the memory to receive the actual 32 bits of that double word from the memory. And now the question arises: how much time should the CPU wait for the data to arrive from the memory to the CPU ? It could wait one microsecond, but what happens if not all those 32 bits arrived at the CPU ? The data is incomplete. It could wait a longer time interval instead of two seconds or three seconds. But this long wait would not be very efficient because if the data arrived at the CPU in the first millisecond and the CPU just waits for two seconds for nothing, that's going to lead to inefficient execution. So that's why there should be a bound, a time limit on each operation that happens in a computer system. And this time limit could be given in seconds, but that's quite a low granularity time scale, so it should be lower than seconds. But there's no reason for the time in the computer system to match exactly the real time that exists in the real world. That's why we could just have a timer, a clock generator circuit on the computer system that generates clock signals periodically. They don't need to match the real time meaning they don't need to send one signal in each second or in each millisecond, but it should have a clock frequency, so they should be periodically periodical and they should have a clock frequency. If we have a clock frequency throughout the computer system then you could say that every operation of getting data from the memory to the CPU takes two cycles for example, where a cycle is the interval between two clock signals; the computing cycle is just the interval between two consecutive clock signals. For Z1 for example, it says that one instruction takes 1 to 20 cycles. Let's say it takes 20 cycles because the clock frequency is 1 Hz, so a computing cycle takes 1 second; it says here that the clock frequency is 1 Hz meaning that one clock frequency is generated per second. 20 cycles would be equivalent to 20 seconds in the real time. So one instruction in Z1 would take 20 seconds in the real time. In the same way, in each computer every operation including the operation of getting data from the memory or sending data through the memory is predefined to take a fixed number of computing cycles. Actually normally you would have in a computer several clock generators not only one and if the CPU works with a clock frequency of let's say 10 Hz, but the memory is cheaper or it's weaker and it works with a clock frequency of only 1 Hz, then there is a negotiation and actually the discussion between the CPU and the memory will always follow the lowest frequency, that is 1 Hz, not 10 Hz. Please note that the clock frequency of a computer is a measure of its performance. If you have a high clock frequency, then you execute computation a lot faster than a computer with a low clock frequency. Let's take an example. Let's say we have one CPU that has a clock frequency of 10 Hz, that is there are 10 clock signals generated in one second - this is CPU1 and let's say we have a CPU2 with a clock frequency of 100 Hz. Usually an instruction takes one or two or 4 cycles. Remember that the computing cycle is just the time interval between two consecutive clock signals. CPU1 would have 10 computing cycles per second, because there are 10 clock signals per second (10 Hz) and CPU2 would have 100 computing cycles per second. Let's assume that an addition operation takes one cycle which means that CPU1 can perform 10 additions per second since CPU1 has 10 cycles per second. But CPU2 can perform 100 additions per second. So we can say that CPU2 is 10 times faster than CPU1 because it can perform 10 times more additions than CPU1 per second. For a long time the clock frequency this one the clock frequency was the driving force or that or the driving principle of CPU development and companies like Intel or Advanced Micro Devices kept increasing this clock frequency. Now we skip a little bit to the modern era and continue a little bit with the discussion about the clock frequency. So the clock frequency was the driving force of CPU development from the beginning of the 20th century to about 2004 or 2005. If a CPU has a larger clock frequency, then it performs a lot better it computes a lot faster than another CPU. But this comes with a problem. Because in order to increase the clock frequency of the CPU you need to put a lot more transistors on that CPU and also you need to consume a lot more electrical power. If you consume more electrical current, you also generate more heat during that consumption. You generate more heat which needs to be dissipated. If you have some advanced cooling system like based on liquid nitrogen then that's no problem, but usually in commercial personal computer systems like notebooks and tablets and phones and so on, even desktop computers you'd have a cooling system based on a fan that spins on top of the CPU and takes the heat and dissipates the heat in the environment. And in order to dissipate more heat with the fan, you need to consume more electrical power with the fan to spin a lot faster and to dissipate more heat. This also means more electrical consumption which also generates heat and this is a vicious circle that leads nowhere. So in order to make the CPU run faster you would increase the clock frequency you have to increase the electrical power consumption because of this there's more heat that needs to be dissipated because of this heat that needs to be dissipated there's more the fan should spin a lot faster which also means that the fan would consume more electrical current and this is a vicious circle and cannot get out of it. Actually the way out of it was taken by the Intel corporation in 2004 or 2005 when they switched to some other driving force of CPU development and CPU improvement. From 2004 to 2005 the clock frequency improvement is no longer the driving force in CPU industry. They thought we cannot increase the clock frequency without increasing the electrical consumption and generating more heat, so what we can do is just keep the clock frequency at a level and increment the number of brains on the CPU, so move to the multi-core architecture. And for from 2004 and 2005 the whole industry shifted towards multi-core computer architecture where you have several cores on the same CPU and that completes this whole parenthesis, i.e. this whole side discussion about the clock frequency and the performance of the computer system.

First generation computers (based on vacuum tubes): 1940-1960

We move to the first generation of electrical computers, those based on vacuum tubes. They were developed between 1940 and 1960. The vacuum tubes are made of a glass chamber and the chamber contains nothing, it's vacuum or it may contain some gas; inside this sealed chamber, you would find a cathode through which a hot filament passes and then a plate which is the anode. The anode has the opposite electrical charge and is actually a diode. There are more complicated vacuum tubes. They are also called valves by the British or thermionic valves. And they could also have several layers, several grids inside them. When the electrical current passes through this filament, it heats and it gets hot and because of this electrons are released from the cathode and they are attracted by the anode which means that when the filament heats, electrical current passes from the cathode to the anode although they have no physical contact. The vacuum tubes could be used to amplify electrical current, if the filament gets more heat, then more electrical current would pass from the cathode to the anode. Or if you have another grid here you can you can also charge it electrically this grid which then filters the passing of the current from the anode from the cathode to the anode and it works like a switch like a sort of an IF. So it allows the current to pass sometimes depending on how much current passes through this grid. It allows the current to pass from this cathode to anode or it stops the current from passing from this cathode to the anode. So they are useful in electrical devices and in computers as switches. If you remember, a bit is just a sequence of imaginary information that has values zero or one. A bit doesn't exist in the real world, it exists only in our minds. But it also has a physical equivalent called the bistable circuit which is just a circuit with a resistance and with a switch and when that switch is closed then the electrical current passes through it and we say that the bit corresponding to that circuit has the software value one and when the switch is open the current does not pass through the circuit and we say that the bit corresponding to that circuit has the software value zero. And the vacuum tubes are used exactly for the switches because they can be programmed by sending current through them or not. They can be programmed to close the circuit close the switch or release the switch they were later replaced by transistors which were a lot more reliable and perform the same function. Transistors are ubiquitous, a CPU is made of thousands and billions of transistors squashed together on a silicon plate. Ttransistors were invented in 1948 by John Bardeen, William Shockley and Walter Brattain. The transistors were a lot more reliable than vacuum tubes and the vacuum tubes were not so reliable because they heated and they would crash. And when the transistors came along, they were a lot cheaper to produce, they wouldn't dissipate heat (they also generate heat, but not that much), so they replaced pretty much all vacuum tubes in all electronic devices except as far as I know devices in the musical industry like amplifiers from musical instruments. The ones that are more expensive and used in concerts still use vacuum tubes because vacuum tubes produce the sort of distortion that it's also called warm distortion and is very sought by singers in the musical industry.

Fig. 4.5 Vacuum tubes

The one of the first computers built by the United Kingdom's army was the Colossus. Built between 1943 and 1945, it was developed by the British code breakers from Bletchley Park and used in the cryptographic field. It was used to break the Lorenz german chipper. Just like Alan Turing built the bomb or in similar way it was Colossus built and used. It used a lot of vacuum tubes, it was programmed by switches and plugs and not by a store program which means that when I want to add let's say if I want to compute the s 1+2 I would have to operate a bunch of switches. Using some switches I would load the two values in the memory then I would use some other switch in order to choose between additions or subtractions and I would use some other switch to choose between multiplication and division. So you do all this programming using manual switches - that's what it means that it was programmed by switches and plugs and not by a stored program. Everything is based on simple circuits that have a switch, even my phone; if the switch is on the electric current passes through it, if it's not it doesn't pass. In the beginning, all those electrical circuits with a resistor and a switch would have to be turned on and off manually. Nowadays we write programs with instructions and those instructions are executed by the CPU and the CPU has components that are called transistors that allow us to automatically close and open circuits corresponding to individual bits so we now close and open electrical circuits using programs using instruction using software. But the idea is the same. Back then you wouldn't have a program, but you have to do this manually. It had no memory, input on paper tape and for output there was an electric type writer.

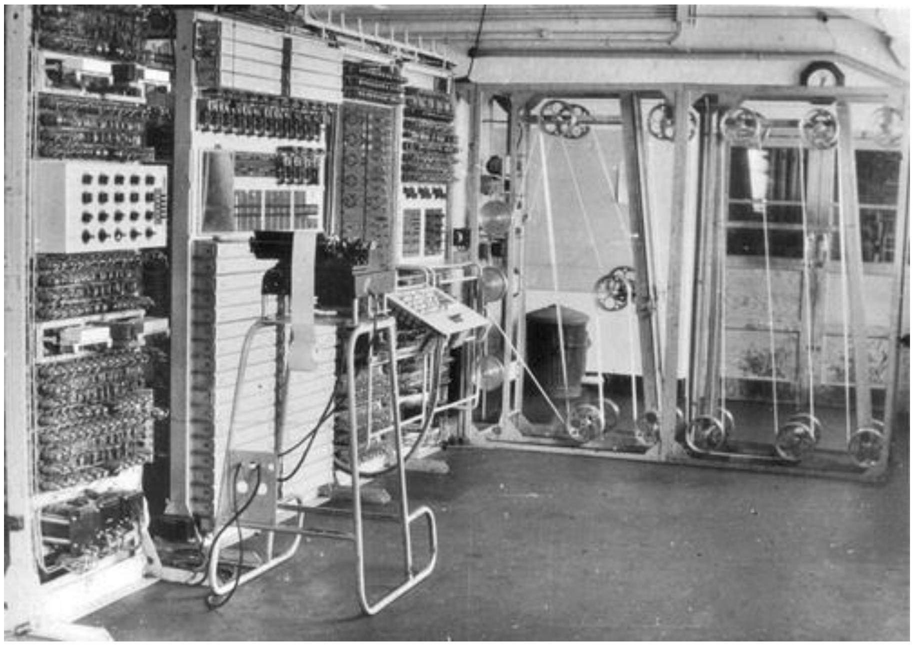


Fig. 4.6 Colosus, UK

The next ones are two powerful computers, one is ENIAC built by the United States Army 1946 and the other one is EDVAC. The ENIAC was a Turing-complete electrical general purpose computer. The name ENIAC comes from Electronic Numerical Integrator and Computer. It was built by John Mauchly and J. Presper Eckert from Univ. Pensylvannia for US Army. It could calculate a bomb trajectory in 30 seconds much faster than a human who would do it in 20 hours. It costed about 487.000 $ which would be equivalent to 6.887.000 $ in today’s money. It contained 20,000 vacuum tubes, 7200 crystal diodes, 1500 relays, 70,000 resistors, 10,000 capacitors and approximately 5,000,000 hand-soldered joints. It was big, it weighted 27 tons, occupied 167 squared meters and consumed 150 KWatts of electricity. The input and output was on IBM punched cards. It had a 100-word magnetic memory. It could do 385 multiplications per second, 40 divisions per second and 3 square root per second. It had a 200 micro second cycles (100KHz). It could perform complex sequences of operations, including loops, branches, and subroutines but not with stored programs, but with manual switches. It wav very unreliable, vacuum tubes would break daily – longest runtime approx. 5 days. Because it was so large and it consumed so much power 150 kilowatts, there were various legends that when the computer was turned on, the lights in homes in the state of Philadelphia dimmed because it consumed so much power. There are still two or three factories worldwide but that produce vacuum tubes for the musical industry one of them is in China one of them is in Russia and I think one of them in the Czech Republic.

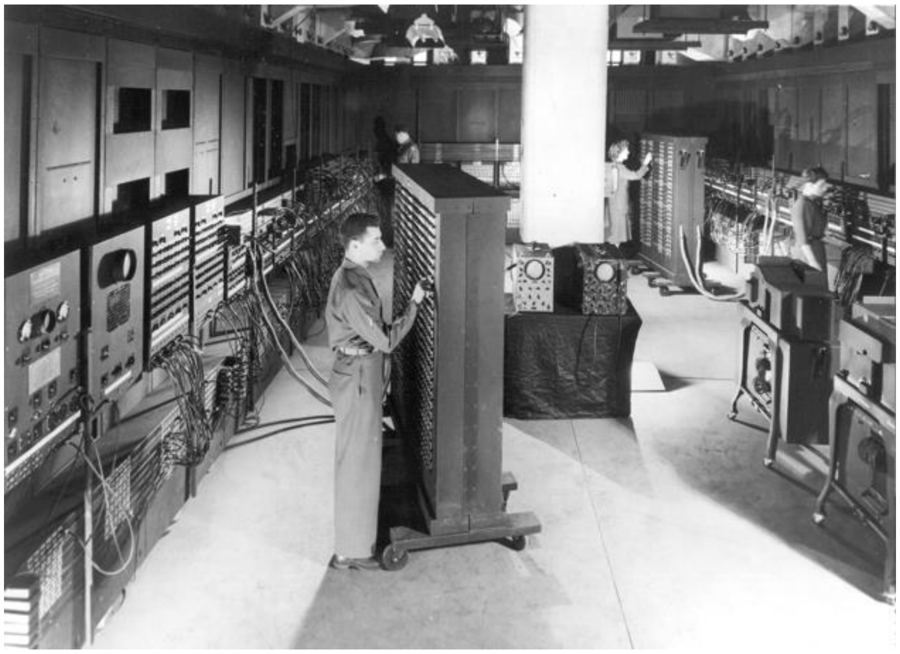


Fig. 4.7 ENIAC, 1946

Let's move on to EDVAC which was the next supercomputer built by US Army in 1949. built by the same people John Mauchly and J. Presper Eckert with John von Neumann as consultant for the US Ballistics Research Laboratory. EDVAC comes from Electronic Discrete Variable Automatic Computer. The cost of EDVAC was similar to ENIAC, 500.000 $. EDVAC used a binary system (not decimal like ENIAC) and stored programs in memory. This was the idea of John von Neumann to store the program into the memory and don't have manual switches for everything, but just run the program from the memory. It could do an addition in 867 microseconds and a multiplication in 2,9 milliseconds. It had a memory of approximately 5.5 KB. It had almost 6,000 vacuum tubes and 12,000 diodes, and consumed 56 kW of power;. It occupied 45.5 sq. meters, weighted 7,8 tons. It used to ran 20 hours per day.

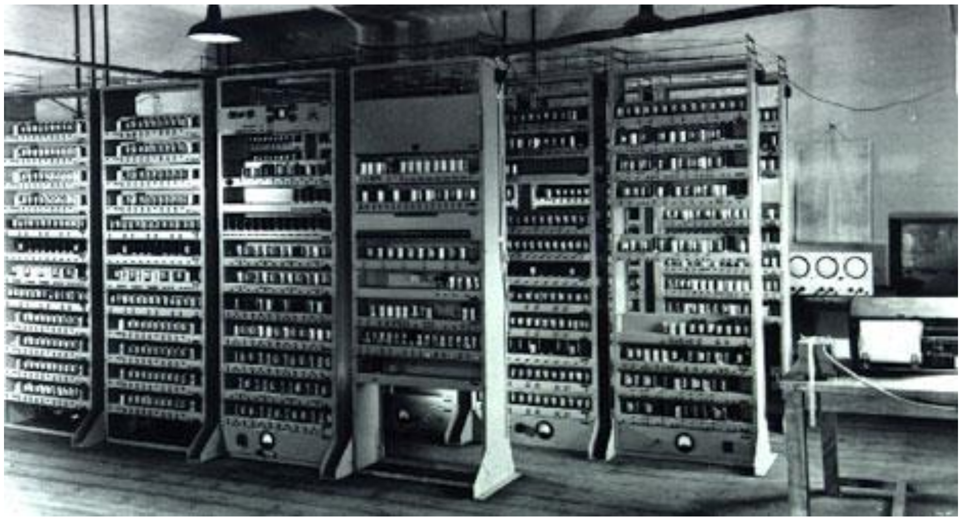


Fig. 4.8 EDVAC

Some other computers from the first generation were a series of 4 computers, Harvard Mark I-IV built in the United States between 1944 – 1952. The computers were IBM Automatic Sequence Controlled Calculator (ASCC). It was built from switches, relays, rotating shafts, and clutches. It used 765,000 electromechanical components and hundreds of miles of wire. It comprised a volume of 23 m^3 (16 m x 2.4 m x 0.61 m). It weighed about 4,500 kg. It could do 3 additions or subtractions in a second. A multiplication took 6 seconds, a division took 15.3 seconds, and a logarithm or a trigonometric function took over one minute.

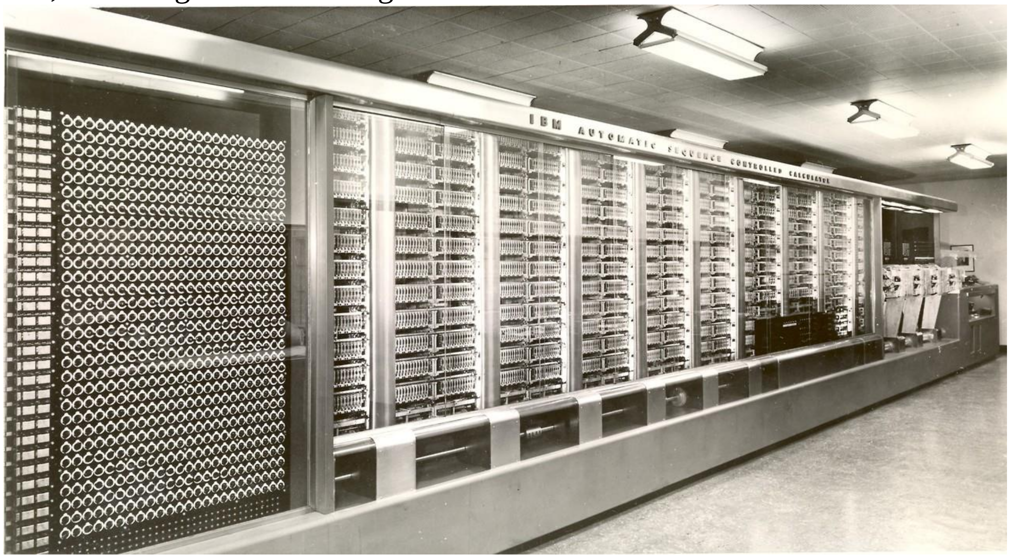


Fig. 4.9. Harvard Mark I

There's another computer called Manchester Mark I built in the UK in 1948. It was developed by the University of Manchester, consumed 25 kilowatts and it used words on 40 bits.

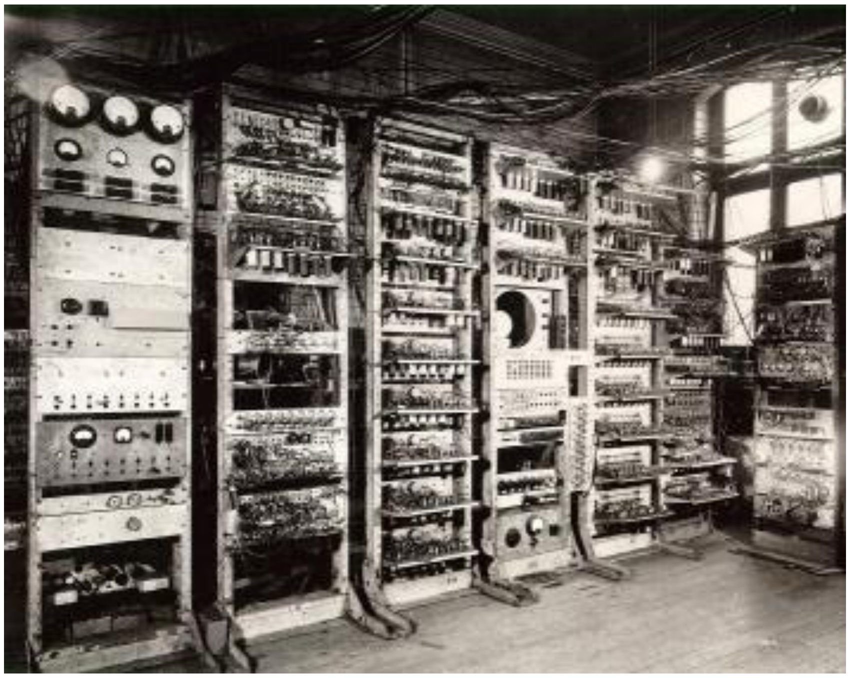


Fig. 4.10 Manchester Mark I

Before we move to the second generation of computers based on transistors, I just want to briefly pause and have a global view on computer history and to look a little bit at the evolution of computers from 1950 to the present day. So in 1950 we have a supercomputer like ENIAC, then after the transistor was invented everything began to shrink more and more and then we go to the microchip which is just a set of transistors soldered together on the same board and this allowed for even smaller computers and then starting with the invention of the first microprocessor from the Intel company, this allowed everything to become even smaller and actually it company companies to build computers for the home users because the previous ones were only for the army and powerful institutions like universities. But the miniaturization of the microprocessor allowed to build small computers like IBM-PC for the end users and this miniaturization process can continue even more around the years 2000s. Notebooks were invented a little bit before 2000. Computer notebooks were invented which shrank things even more, then tablets and smartphones which shrank things even more so everything is shrinking. But actually if we look at a Google data center or Facebook data center which is just a large building with a lot of computers. The computer is just a small part of this large data center. So if you come to think about it, this whole process starting with miniaturization from 1950 getting to the microprocessor in 1972 where things shrink a lot more and it allowed us to have personal computers, desktop computers back home and then it shrink more based on the microprocessor and the integrated circuit it allowed us to develop even smaller computer devices which are notebooks and phones and tablets, but nowadays the whole thing moves backwards to increasing things and having this large data centers and actually you may not even notice this, but I think this is the direction where we are going; we have all the computations usually done in a super computer like a data center and we would in the future I think we will only use the laptop and the tablet and the phone the smartphone we will only use them as terminals like screen monitors for sending computing tasks to the cloud. This already happens like when watching youtube, I just have a small player based on javascript on my local terminal, but everything happens and everything is stored in a data center in a youtube data center right so everything is sent from there it is automatic, it is selected there, converted there from one format to another one and it is sent to me where this small javascript player just displays the video. So it's ironic that from 1950 the history of computer devices started this long journey of shrinking everything to smaller and smaller devices going to personal computers and notebooks and then phones and tablets but now it's starting to enlarge everything again just to have data centers and I think in the future we will have this notebooks and tablets that are just basic terminals used in order to interact with the actual computer that's inside the data center.

Second generation computers (based on transistors): 1950-1970

The second generation of computers of computers is based on transistors and built using transistors. They were built between 1950 and 1970 and the transistors are just an improved vacuum tube meaning that it's just an electrical switch, it allows current to pass or it stops current to pass between the source and the drain, depending on the inner state of the transistor. Transistors were invented by John Bardeen, William Shockley and Walter Brattain at Bell Labs in 1948. It can be used as an electric amplifier or electronic switch. Using transistors we can build logic gates and using logic gates, we can build an Adder (i.e.

circuit that performs the addition of two numbers and using an adder we can create an ALU (Arithmetic and Logic Unit) which is an essential component in a CPU.

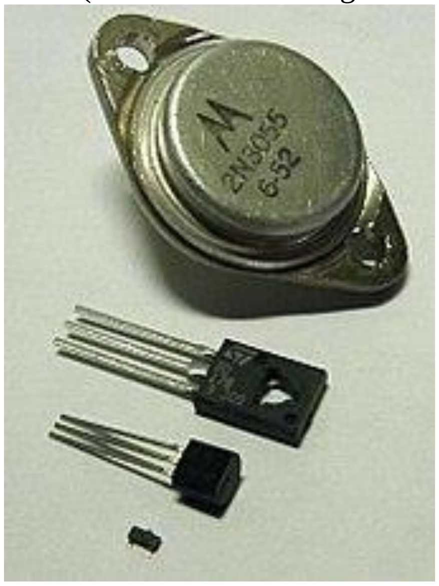


Fig. 4.11 Transistors

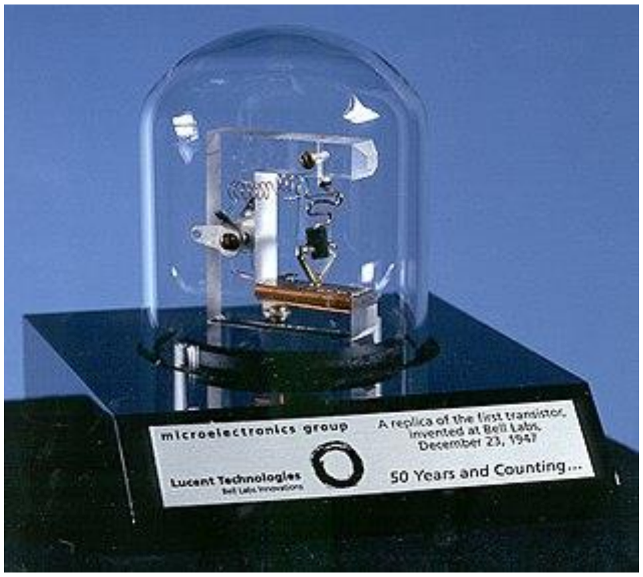


Fig. 4.12 Replica of the first working transistor

Transistors have replaced the vacuum tubes because they were more reliable, they wouldn't generate heat and then they wouldn't break so often as the vacuum tubes and also it's a lot cheaper to produce transistors. They pretty much function the same way as the vacuum tubes meaning they function as electrical electronic switches and this is very important in the computing world. It's very important to be able to turn on and off a switch that closes an electrical circuit because everything nowadays and for a lot of time, everything in computing is based on bits of information. The concept of a bit is the smallest quantity of information that can be stored in a binary computer. Pretty much all computers are binary systems. You can store on one bit the value zero or one, but as you as you know or imagine the bit is a software concept, it doesn't exist in the real world, it only exists in our mind. But the bit has an equivalent in the real world which is a bistable circuit which is just a simple circuit with a wire and a resistance and a switch on it. And when that switch is closed then the current passes through the circuit and we consider that from a software point of view that circuit has the software value one. And when the switch is open there's no current passing through the circuit and we consider that is equivalent to the software value zero. Everything that's done in computing is set bits to zero or one that's all a cpu does it just sets values to one and zero we how does it set well it depends on what we are trying to do. If we are trying to do an addition then it sets the zeros and one in the result depending on the operation and the actual data that we want to add and actually in the first version of computers, you would program them manually by switches. If you want to set the value 2 in binary in the memory you would have to use at least two switches and you have to turn on the first switch corresponding to one and turn off the second switch corresponding to zero so 10 is 2 in binary. If you have a larger value you would have to use more switches. And then you have to use a switch in order to load values into the memory. But later on using transistors and specialized transistors or specialized integrated circuits or microchips called mosfets, those were able to dictate / control the switch on some other circuit. So one circuit, one transistor would be able to control a circuit corresponding to a bit, would be able to close it and open the circuit depending on some state of the transistor. So we have electrical circuits that control other electrical circuits.

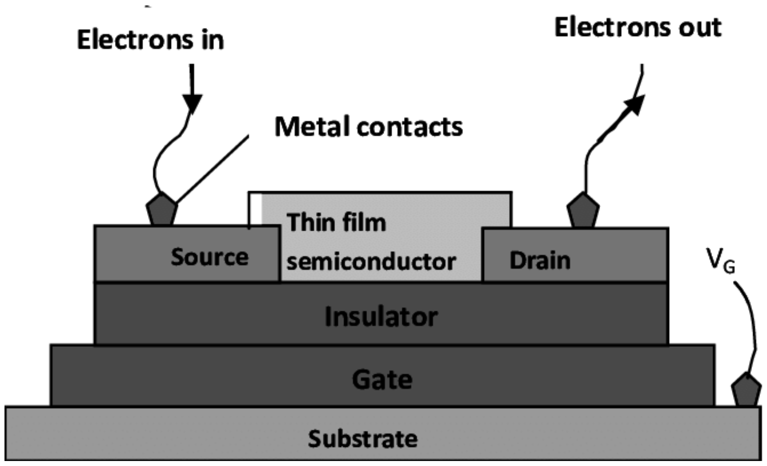


Fig. 4.13 Transistor inner parts

Depending on the actual electrical current that is received on this gate which is one of the leg of the transistor, it would allow current to pass between the source which is one of the tiny legs of the transistor to the other tiny leg of the transistor which is called the drain. Basically this means that it functions like an electrical switch: you allow the current to pass from one pin to the other or you just shut it off depending on the current that's being sent through this middle pin of the transistor. And using this functionality you're able to build logic gates which are just simple circuits that implement the logical operations and, or exclusive or and not. In the following figure, you can see an AND logic gate implemented with 2 transistors.

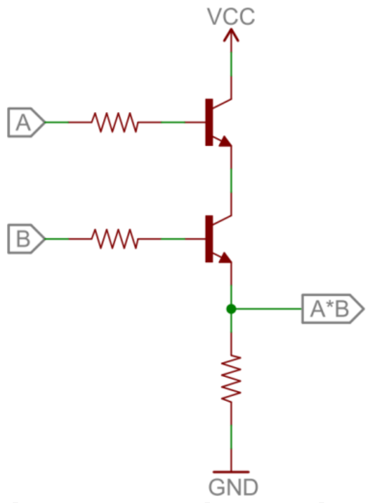


Fig. 4.14 AND logic gate implemented with 2 transistors

And using those logic gates you can build a component that would perform an addition for example which is called an adder or you can build a component that would perform the subtraction operation which is called a subtractor or another component that performs multiplication and another component that performs division and using these four components you can build an arithmetic and logic unit which is an important part of the CPU. The logic gates signs are represented in the following figure.

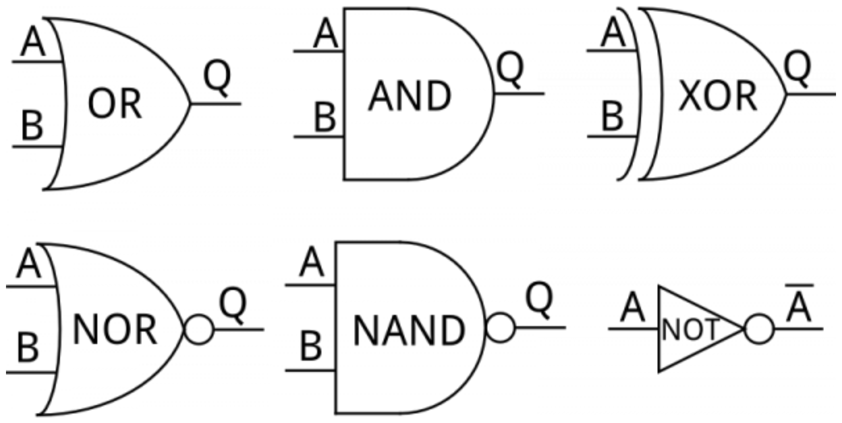


Fig. 4.15 Logic gates

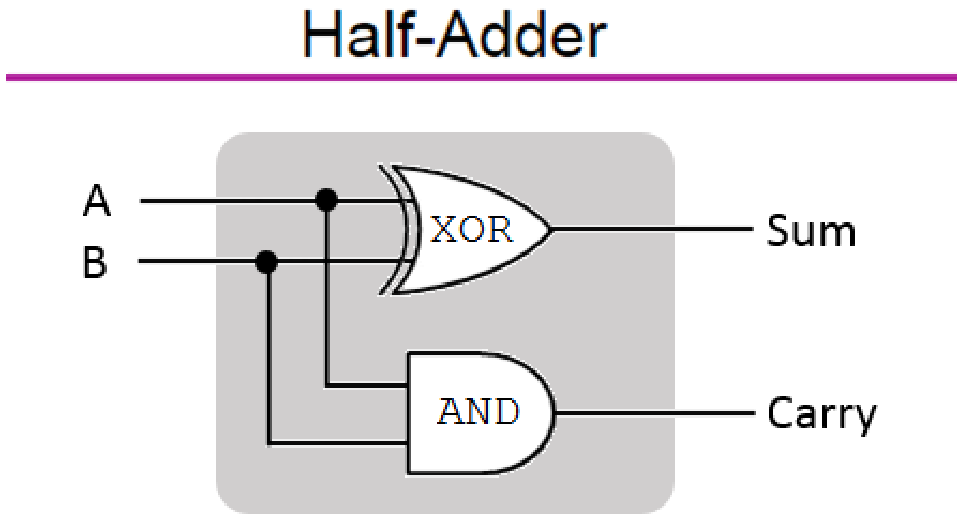


Fig. 4.16 Half-Adder using XOR and AND gates

In a half adder the carry bit is not used later on, but using this simple component we can add numbers that have eight bits each. It's very important to add two bits and then add the next two bits and also to that result add the carry from this result and just using several of these components in sequence we can build a full adder of eight bit numbers. For example and in similar ways we can build the subtractors and dividers and multiplicators.

Some examples of computers from the second generation that use the transistor technology are the following. One of them is IBM 306 in 1957. It had more than 3000 germanium transistors. It was programmable using an electric control panel. It could store 40 nine-digit numbers in the memory (magnetic core). It could perform 4,500 additions per second, it could multiply 2 nine-digit numbers, yielding an 18-digit result in 11 milliseconds, it could divide an 18-digit number by a nine-digit number to produce the nine-digit quotient in 13 milliseconds. It was supplied with a type 535 card reader/punch which had its own control plugboard.

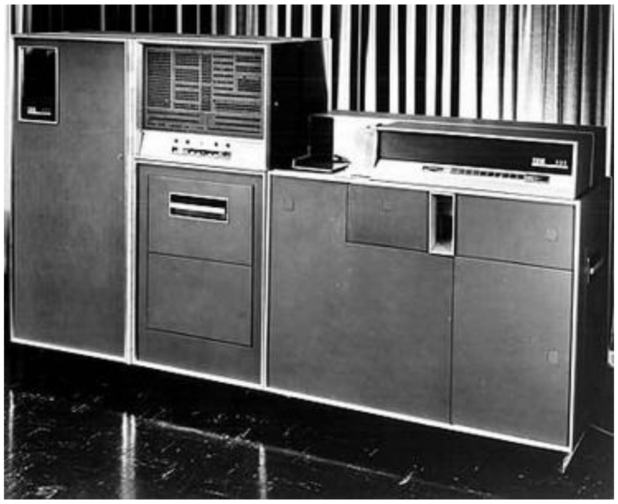


Fig. 4.17 IBM 306, 1957

Another computer was Olivetti Elea 9003 built in Italy in 1959. Elea 9003 was able to run 8-10000 instructions per second. It could store 160.000 words in the memory. Mass storage was on magnetic tape. The input was on punched paper tape or punched cards. The output was on line printer.

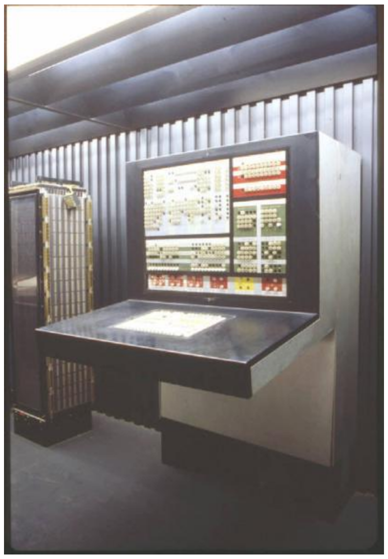


Fig. 4.18 Olivetti Elea 9003, 1959

Another 2nd generation computer was IBM 7007 built in 1958. It was IBM's first stored-program computer. The CPU speed was 27 KIPS. It costed $813.000. Disk storage had a capacity of 6 million digits.

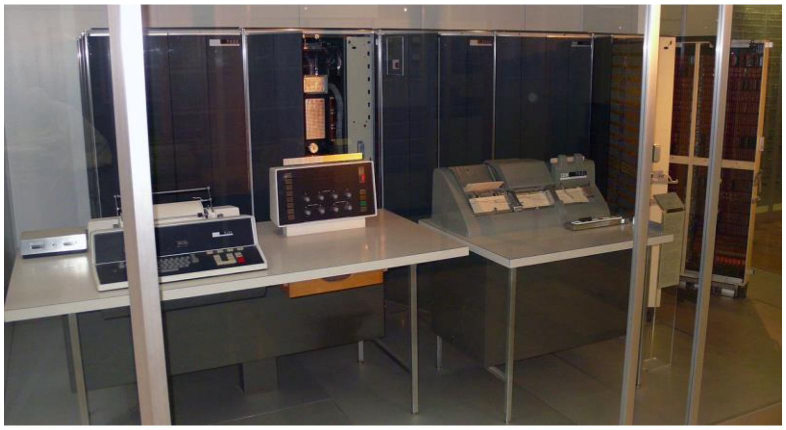


Fig. 4.19 IBM 7007, 1958

Another computer was the Honeywell 200 built in US in 1960. It used 6-bit characters. It could store 524.288 characters in the main storage. The addition would take 12 useconds. It had COBOL and Fortran compilers. Fortran is actually the first high-level programming languages that was invented in 1950s and Cobol was another programming language from the early programming languages.

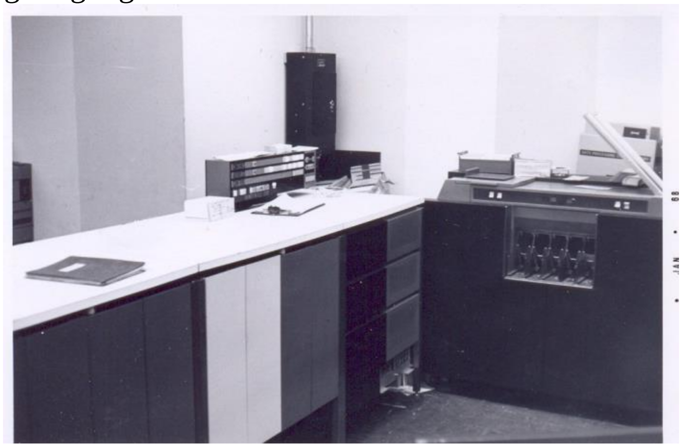


Fig. 4.20 Honeywell 200, 1960

Another computer was the PDP-8 built in 1965 in US.

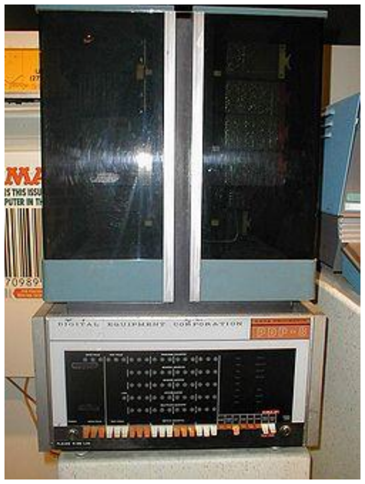


Fig. 4.21 PDP-8, 1965

Another computer was IBM M44/44X, built in 1965 in the United States. This one was built for research experimentally it explored memory paging so dividing the memory into pages also it simulated virtual machines back then in 1965.

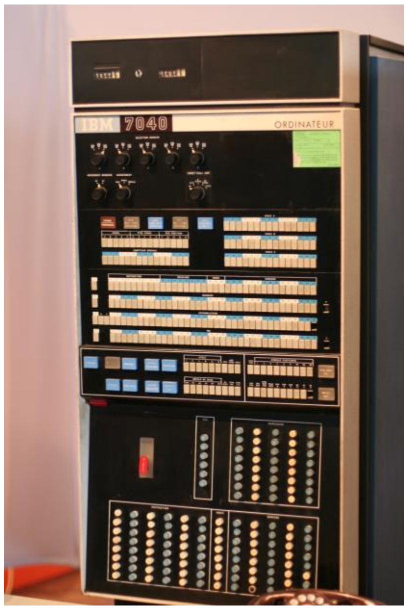


Fig. 4.22 IBM M44/44X, 1965

Third generation computers (based on the microchip): 1960-present

The third generation of computers is based on the microchip and the fourth generation is based on the microprocessor, but the microprocessor is just a specialized microchip, so we may say that the fourth generation is kind of included in the third generation. That's why we say that the computers of the third generation was were produced from 1962 till the present but still the microprocessor-based ones are more specialized generation of computers than the third generation of computers based on the microchip. The integrated circuit (i.e. microchip) was invented independently by Jack Clair Kilby and Robert Noyce. It is a set of electronic circuits on a small flat piece of semiconductor material, silicon. It integrated many tiny transistors as a one component, instead of linking discrete transistors with wires. As of 2016, typical chip areas range from a few square millimeters to around 600 mm^2, with up to 25 million transistors per mm^2. Related to the number of transistors on a microchip there's the Moore law from Gordon Moore which is the co-founder together with Robert Noyce of the Intel corporation that says that the number of transistors in the integrated circuit doubles every two years. The number of transistors employed on a chip in 2016 is 10 billion transistors.

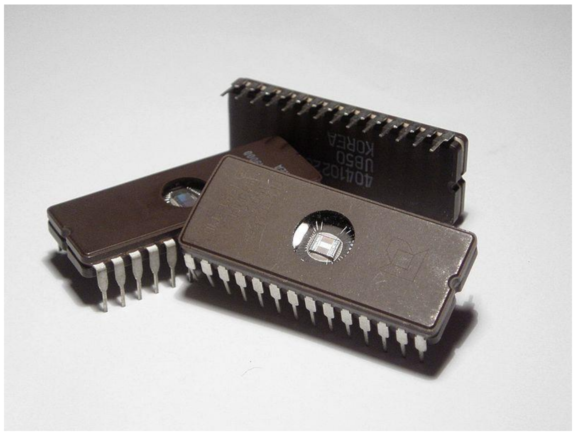


Fig. 4.23. Microchips

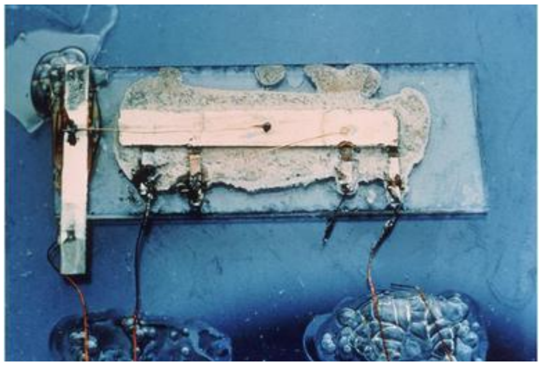


Fig. 4.24. Jack Kilby's original prototype

Nowadays pretty much all microchips are built using silicon hence the name silicon valley in United States, in San Francisco and the whole silicon revolution. Silicon is a semiconductor material – is between an electrical conductor and an insulator. If it is *doped* (i.e. treated with chemical substances) or electrical field is applied on it its electrical conductivity can be modified. It is the second most frequent element on earth (after oxygen). It is used as the base, substrate for transistors, MOSFETs, integrated circuits. By applying electrical current to the semiconductor to the silicon you can modify the electrical conductivity of the material and a microchip is built using mosfets. In a Mosfet you have this plate, this body displayed which is a semiconductor material well actually it has several layers and then you have a lot of components, on top of it which has the source and then the drains the current passes through this depending on the gate this is the gate so the current passes from the source to the drain. The Mosfet is one of the most famous transistors used in in integrated circuits. Integrated circuits are just transistors a bunch of mosfets components built on a plate of silicon semiconductor material which connects them together. Actually this semiconductor material, this silicon has several layers in order to work but it allows current to pass between them depending on the current applied to the to various parts, various areas of the semiconductor plate.

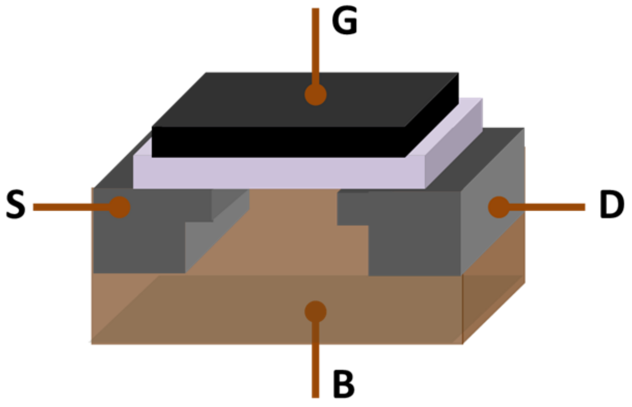


Fig. 2.25 The Mosfet

where S = source terminal, D = drain terminal, G = gate (metal), B = body (the semiconductor made of several layers of silicon); under the gate G is an insulator plate.

And actually the microprocessor that we have in a notebook is actually a specialized form of a microchip or integrated circuit, but a specialized one for very fast computations additions abstractions multiplications and division. There are at least two types of microchips: *microcontrollers* and *microprocessors*. A microcontroller (e.g. Atmel Atmega 328p used in Arduino board) is an embedded system on a chip having functions like the ones provided by the CPU, but also including memory, clock component, pins and controllers for external components. A microprocessor (CPU, e.g. Intel 4004, 8008, 8086) is a highly specialized microchip having functions like addition, subtraction, multiplication and division of integer numbers and logical bitwise operations. Usually the cpu does not contain all the components of a microcontroller, it doesn't contain memory on it I mean it contains small memory the registers but not large memory it doesn't contain a clock in it it doesn't contain ports on it for for controlling external devices so it's a more specialized component.

Some computers from the third generation built on the microchip are: IBM-360 series, Honeywell-6000 series, IBM-370/168, TDC-316, Nova. IBM 360 series, model 85 was built in 1968. It had a storage from 500KB to 4MB. The major machine-cycle time was 80 ns. Main storage data flow was 16 bytes. Main storage cycle time was either 960 or 1,040 nanoseconds. It would operate with floating point numbers with 28 fractional digits.

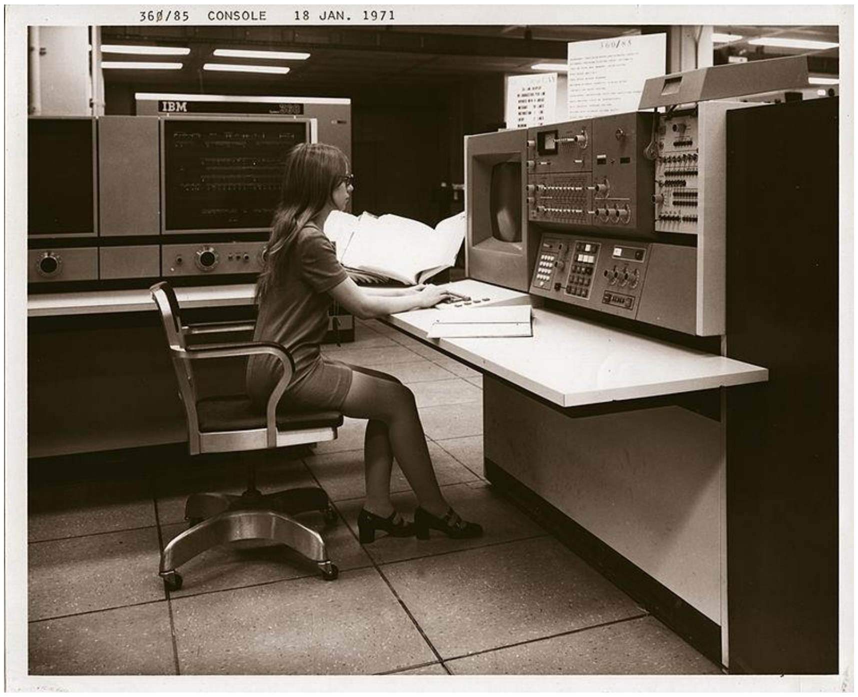


Fig. 2.26. IBM 360 series, model 85 (1968)

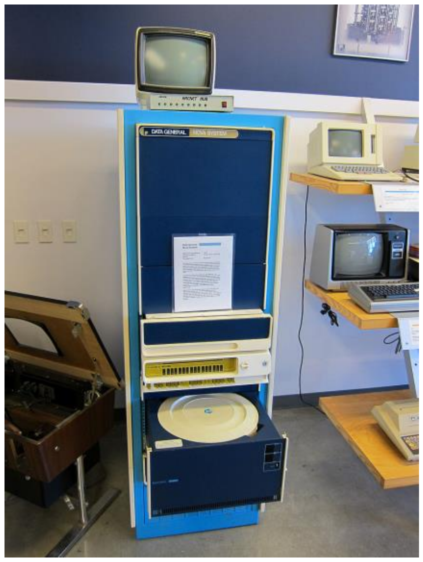


Fig. 2.27. Data General Nova, 1969 (costed 8000$)

Fourth generation computers (based on the microprocessor): 1971-present

And the fourth generation of computers are the ones built on the microprocessor which is a specialized kind of microchip, a specialized integrated circuit. The microprocessor was invented by Intel in 1971. It incorporates CPU functions in a single integrated circuit. The microprocessor is a multipurpose, clock driven, register based, digital-integrated circuit which accepts binary data as input, processes it according to instructions stored in its memory, and provides results as output. It operates with binary numbers at a clock frequency of 740 KHz. The instruction cycle took 10.8 us.

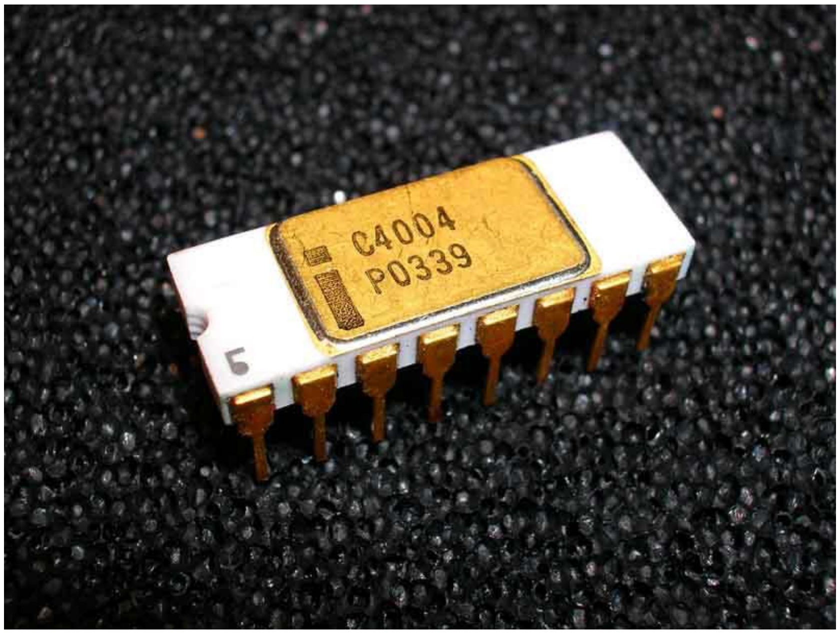


Fig. 2.28. Intel 4004, the first microprocessor; invented by Intel in 1971

The first Intel CPUs were:

- Intel 4004, 1971, first CPU, 4 bits registers
- Intel 8008, 1972, 8 bit regs, double the power of 4004
- Intel 8080, 1974, 8 bit regs, used for many microcomputers like Altair 8800 (sold thousands in a few months)
- Intel 8088, 1978, 8 bit regs, used in IBM-PC
- Intel 8086, 1978 – introduced the 16 bit x86 architecture
- Intel 80286, 1982, 16 bit regs
- Intel 80386, 1985 – introduced the 32-bit comp. architecture (IA-32)
- Intel 80486, 1989
- Intel Pentium, 1993
- Intel Xeon, Pentium 4, 2004 – introduced Intel 64 (x86-64), 64 bit architecture

Microcomputers, Personal Computers

Supported by the development of the microprocessor, computers became smaller and smaller and were accessible to end consumer. The most famous one is IBM-PC 5150 built in 1981. IBM-PC 5150 was an important milestone in the evolution of microcomputers for home consumers. Most computers produced in the following decades for home users were IBM-PC compatible. Modern desktop computers are IBM-PC descendants. It costed $1565. It used Intel 8088 at 4.7MHz. It had a memory 16KB – 640KB. It used PC DOS 1.0 OS. The storage was on 5.25'' floppy disks or optional hard disk. It had IBM 5153color display.

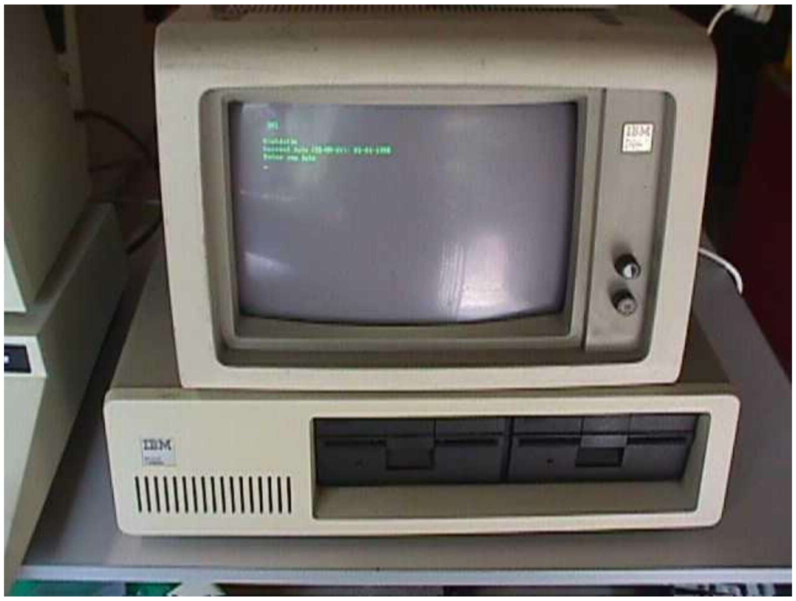


Fig. 2.29 IBM-PC 5150, 1981

Another microcomputer from which IBM-PC 5150 was inspired is Altair 8800 commercialized in 1974. It was sold for $395 as a kit and $650 assembled. It used Intel 8080 at 2MHz. It had a memory of 256 bytes, max 64KB. The Operating System was CP/M. It had storage on paper tape, floppy disk, cassette. Links demonstrating functionality:
http://oldcomputers.net/altair-8800.html
https://www.youtube.com/watch?v=suyiMfzmZKs&ab_channel=deramp5113

https://www.youtube.com/watch?v=EV1ki6LiEmg&ab_channel=deramp5113
https://medium.com/dev-genius/how-to-write-a-program-for-altair-8800-computer-3a4583fe601e

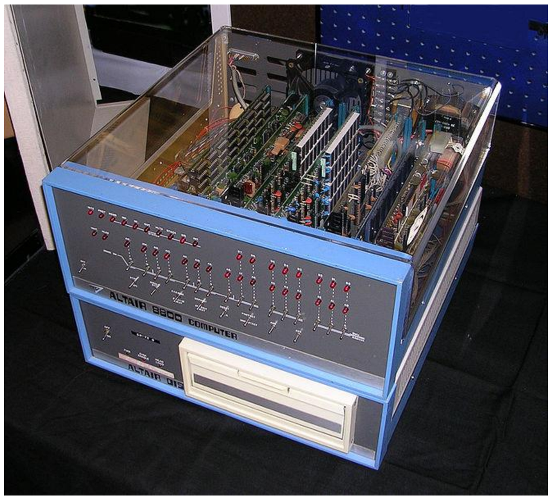


Fig. 2.30 Altair 8800, 1974

Another microcomputer from which IBM-PC 5150 was inspired is IMSAI 8080 developed in 1975. It was sold for $599 as a kit and $931 assembled. It used Intel 8080 at 2MHz. It supported max 64KB memory. The storage was on cassette or floppy disks. The OS was CP/M. The machine cycle is 0.5 usec. Links: http://rwebs.net/micros/Imsai/intro.htm

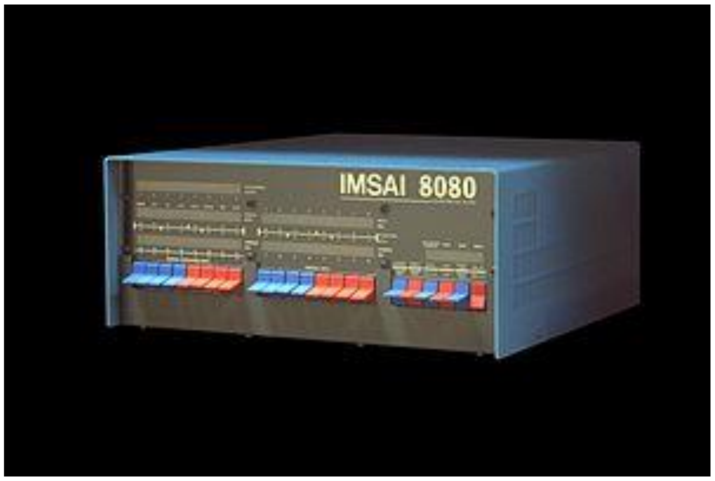


Fig. 2.31. IMSAI 8080, 1975

Finally, another microcomputer from which IBM-PC 5150 was inspired is Apple I commercialized by Steve Jobs and Steve Wozniak in 1976. It costed $666.66 . About 200 were produced. The CPU was a MOS 6502, 1.0 MHz. The memory was 4K - 64K. It had monochrome display with resolutions: 280 X 192, 40 X 24 text. Keyboard was not included. The operating system was Apple BASIC on cassette.

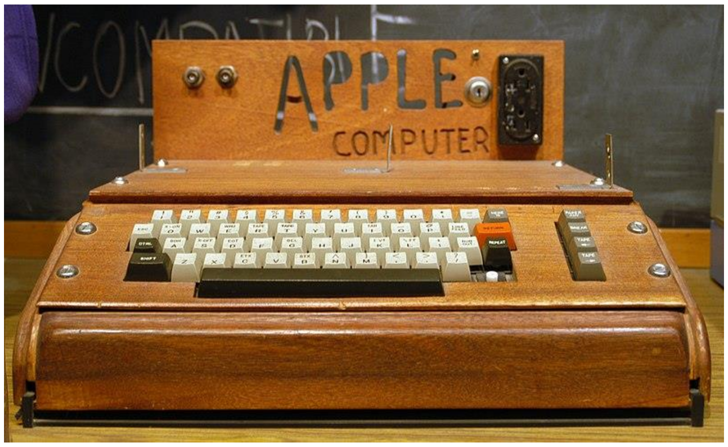


Fig. 2.32 Apple 1, 1976

The history of the Internet

In 1958, DARPA (Defense Advanced Research Projects Agency) was created by the DoD (Department of Defense) in US. In 1962, J.C.R. Licklider published a paper about a "Galactic Network". He was the first director of DARPA. He then influenced his successors at DARPA about the importance of networking concept. In 1961, Leonard Kleinrock, professor at UCLA published a paper about packet-switching being more efficient than circuit-switching communication paradigm. In 1965, Thomas Merrill & Lawrence Roberts connected 2 computers using a low-speed dialup telephone line to prove that Kleinrock was right. In 1966, Lawrence Roberts went to DARPA and planned the "ARPANET" and presented it at a conference (L. Roberts, "Multiple Computer Networks and Intercomputer Communication," ACM Gatlinburg Conference, October 1967). At this conference, Lawrence Roberts met with Roger Scantlebury who told him about two other similar, packet-switched network ideas, from UK: one created by Donald Davies and Roger Scantlebury at NLP (they also had a paper about this at the same conference) and another one created by Paul Baran and others at RAND. Roberts adopted the term "packet" from the NLP paper and the network speed of ARPANET was set to 50kbps. He refined the plans for the packet-switched network at DARPA until 1968 and DARPA released an RFQ (Request for Quote/Quotation) for the packet switches of ARPANET (called Interface Message Processor - IMP). In 1968, a group lead by Frank Heart from BBN (Bolt Beranek and Newman) won the DARPA contract and created the IMP with a team including Robert (Bob) Kahn who created ARPANET architecture; another team of the group leaded by Lawrence Roberts and Howard Frank worked on the network topology and economics; and L. Kleinrock's team from UCLA worked on the network measurement system. In september 1969 first 2 nodes were connected to ARPANET (Stanford-UCLA), IMPs were installed at both ends; "LOgin" is the first text sent over the ARPANET; 2 additional computers were connected by the end of 1969(Univ. Utah, UC Santa Barbara). Additional computers were added to ARPANET and in 1970 the Host-to-Host protocol (NCP - Network Control Protocol) was created by S. Crocker - predecesor of TCP/IP. In 1972, large successful presentation of ARPANET at a conference by Robert Kahn at the International Computer Communication Conference (ICCC). Ray Tomlinson from BBN invented the hot application, email. In 1972-1973 Robert Kahn introduced the open-network architecture (several networks managed independently, but interconnected together) at DARPA and then saw that NCP was not good for this architecture (NCP could nod address machines past the next IMP), so Robert Kahn and Vinton Cerf (from Standord) produced the TCP/IP protocols. They wanted to have only secure communication in a virtual-circuit pattern, but tests with voice packets showed them that for some class of applications, the application should decide what to do after packet losses, not the end host. So they have split initial TCP intru reliable TCP (the one we have today) and unreliable communication provided by UDP. DARPA signed 3 contracts with Stanford (a group lead by Cerf), BBN (a group lead by Ray Tomlinson) and UCL (a group lead by Peter Kirstein) to implement TCP/IP (it was simply called TCP in the Cerf/Kahn paper and it included both protocols). In 1973 Ethernet technology was developed by Bob Metcalfe at Xerox PARC. In 1976 Kleinrock published a book on ARPANET which was instrumental in the wide spread of the network. The number of independently managed networks increased, so Paul Mockapetris of USC/ISI invented the Domain Name System (DNS) which allowed scaling the distributed system. Initially, only one routing protocol was used in the ARPANET, but as the number of independently managed network increased, Interior Gateway

Protocol (IGP) was invented in order to be used inside a local network and Exterior Gateway Protocol (EGP) was used to interconnect the independent local networks. DARPA assigned UC Berkeley to implement TCP/IP in Unix, and UC Berkeley rewrote the BBN code into Unix BSD - which contributed a lot to the success of the network; In January 1, 1983 it was the "flag-day", carefully prepared several years ago, when all computers on the ARPANET should synchronously change from the NCP host protocol to TCP/IP. By 1985 there were a lot of networks inspired from the Internet technology of ARPANET: MFENet of Dept. of Energy, HEPNet for High Energy Physics, MilNet for military, CSNet for academic research in computer science, .. In 1985, NSF started the NSFNet program which reunited several of existing networks for academic research. In 1988, a National Research Council committee, chaired by Kleinrock and with Kahn and Clark as members, produced a NFS report "Towards a National Research Network", which was influential on then Senator Al Gore and laid the foundation for the future information superhighway. In 1994, a National Research Council report, again chaired by Kleinrock (and with Kahn and Clark as members again), Entitled "Realizing The Information Future: The Internet and Beyond" was released.

Fig. 1 Leonard Kleinrock, creator of packet-switched communication

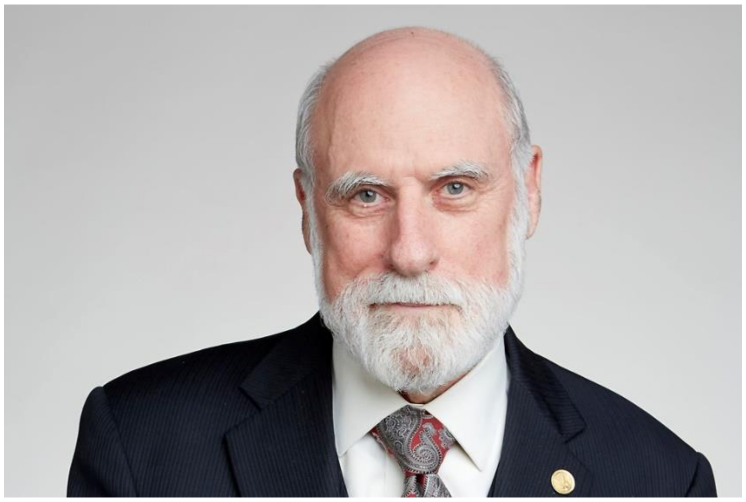


Fig. 2 Vinton Cerf, creator of the Internet

Fig. 3. Bob Kahn, creator of the Internet

Fig. 4. Johnathan Postel, the first RFC editor