Storia dell'informatica

Il calcolatore universale:
Dalla teoria alla pratica

The Steam-Powered Turing Machine mural at the Paul G. Allen Center for Computer Science & Engineering, University of Washington, WA.

The *Entscheidungsproblem*

In continuation of his "program" with which he challenged the mathematics community in 1900, at a 1928 international conference David Hilbert asked three questions, the third of which became known as "Hilbert's Entscheidungsproblem".

A first-order statement is called "universally valid" or "logically valid" if it follows from the axioms of the first-order predicate calculus. Gödel's completeness theorem states that a statement is universally valid in this sense if and only if it is true in every interpretation of the formula in a model.

The ***Entscheidungsproblem*** (German for 'decision problem') is the challenge in symbolic logic to find a general algorithm which decides for given first-order statements whether they are universally valid or not.

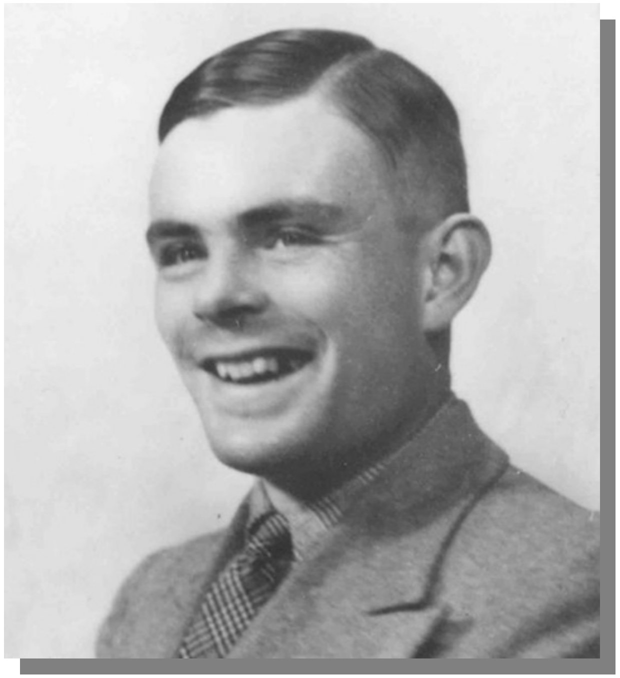
Church and Turing

In 1936, working independently, Alonzo Church and Alan Turing both showed that this is impossible. As a consequence, it is in particular impossible to decide algorithmically whether statements in arithmetic are true or false.

Before the question could be answered, the notion of "algorithm" had to be formally defined. This was done by Alonzo Church in 1936 with the concept of "effective calculability" based on his lambda calculus and by Alan Turing in the same year with his concept of Turing machines.

The two approaches are equivalent, an instance of the Church-Turing thesis.

Alan M. Turing (1912-1954)

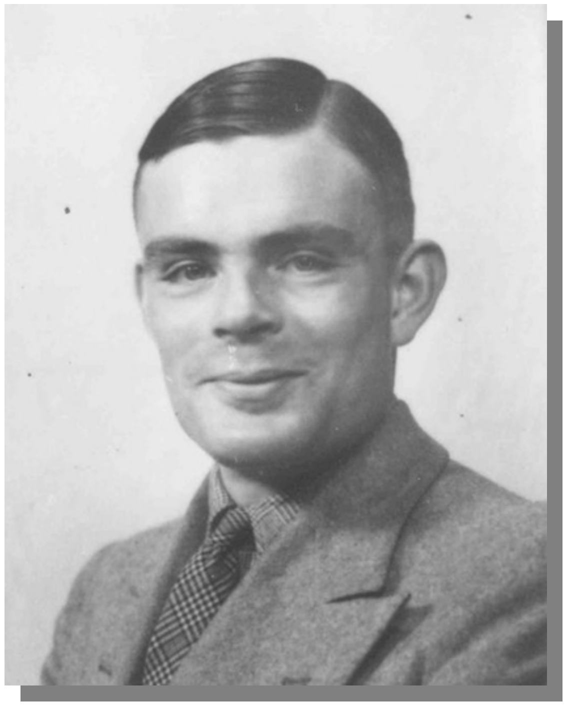


Alan Mathison Turing was a British mathematician, logician, and cryptographer. Turing is often considered to be the father of modern computer science.

During World War II, Turing worked at Bletchley Park, Britain's codebreaking centre and was for a time head of Hut 8, the section responsible for German Naval cryptanalysis. He devised a number of techniques for breaking German ciphers, including the method of the bombe, an electromechanical machine which could find settings for the Enigma machine.

In 1952, Turing was convicted of acts of gross indecency after admitting to a sexual relationship with a man in Manchester. He was placed on probation and required to undergo hormone therapy. When Alan Turing died in 1954, an inquest found that he had committed suicide by eating an apple laced with cyanide.

Turing's Contributions



Turing provided an influential formalisation of the concept of algorithm and computation with the Turing machine, formulating the now widely accepted "Turing" version of the Church–Turing thesis.

After the war, he worked at the National Physical Laboratory, creating one of the first designs for a stored-program computer, although it was never actually built. In 1947 he moved to the University of Manchester to work, largely on software, on the Manchester Mark I then emerging as one of the world's earliest true computers.

With the Turing test, Turing made a significant and characteristically provocative contribution to the debate regarding artificial intelligence.

How powerful a computer can be? Can it replace intelligence?

It came as a surprise in 1930s before the advent of digital electronic computer that Alan Turing argued that any computer, present and the ones ever will exist, is as power as the simple “Turing Machine”.

Certain problems cannot be solved by computer, no matter how powerful the computer is.

On Computable Numbers, with an Application to the *Entscheidungsproblem* (1936)

«We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions q1, q2, ..., qR which will be called "*m*-configurations". The machine is supplied with a "tape", (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the *r*-th, bearing the symbol S(*r*) which is "in the machine". We may call this square the "scanned square". The symbol on the scanned square may be called the "scanned symbol". The "scanned symbol" is the only one of which the machine is, so to speak, "directly aware". However, by altering its *m*-configuration the machine can effectively remember some of the symbols which it has "seen" (scanned) previously. The possible behaviour of the machine at any moment is determined by the *m*-configuration qn and the scanned symbol S(*r*). This pair qn, S(*r*) will be called the "configuration": thus the configuration determines the possible behaviour of the machine. In some of the configurations in which the scanned square is blank (i.e. bears no symbol) the machine writes down a new symbol on the scanned square: in other configurations it erases the scanned symbol. The machine may also change the square which is being scanned, but only by shifting it one place to right or left. In addition to any of these operations the *m*-configuration may be changed.

On Computable Numbers, with an Application to the *Entscheidungsproblem* (1936)

Some of the symbols written down will form the sequence of figures which is the decimal of the real number which is being computed. The others are just rough notes to “assist the memory”. It will only be these rough notes which will be liable to erasure.

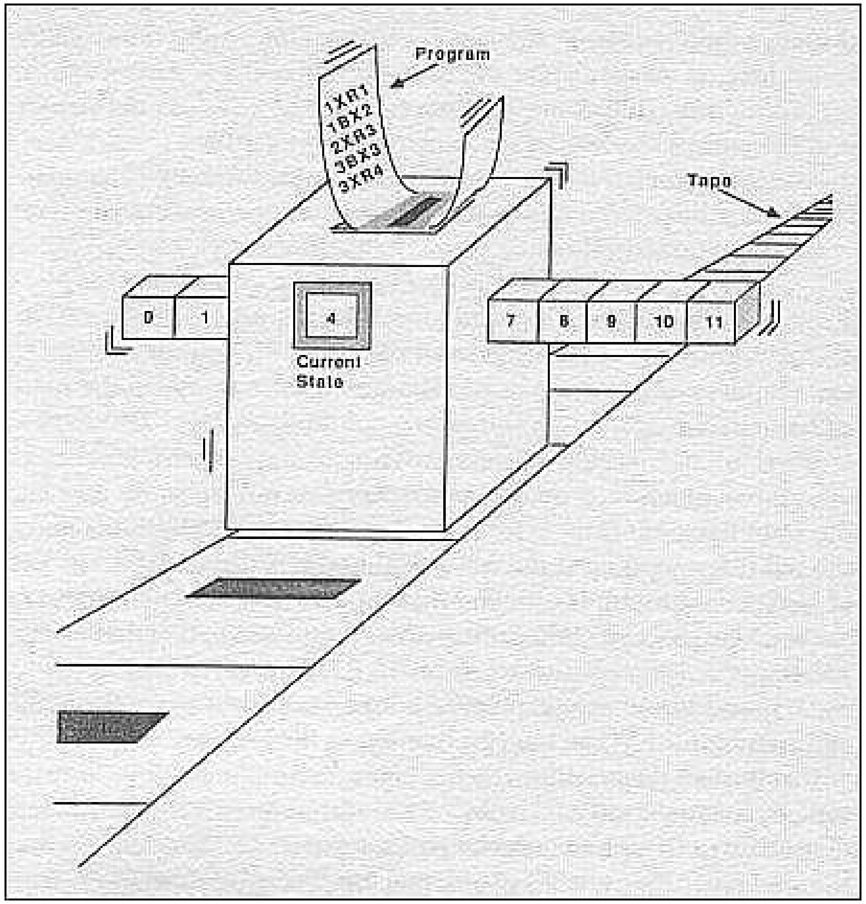
It is my contention that these operations include all those which are used in the computation of a number. The defence of this contention will be easier when the theory of the machines is familiar to the reader. In the next section I therefore proceed with the development of the theory and assume that it is understood what is meant by “machine”, “tape”, “scanned”, etc.»

The Turing Machine

The Turing machine consists of a tape, a moving head which can read and print a set of symbols, an internal state of the machine, and a “program”.

A program is a list of rules what to do next given the current reading and internal state.

It can move to the left or right, change internal state and print a symbol on the tape.



Turing Machine Details

1. A set of characters (the **alphabet**), including blank, serves as input and output symbols on the tape. E.g. {a,b,c, … }
2. A **tape** divided into a sequence of numbered cells each containing one character or a blank. The tape starts from cell 1, and extends to the right indefinitely.
3. A **tape head** that can in one step read the contents of a cell on the tape, replace it with some other character, and reposition itself to the next cell to the right or left of the one it has just read. At the start of the process, tape head always begins by reading the input in cell 1.
4. A finite set of internal **states** of the machine, including START, HALT, and other named states such as 1, 2, 3, …
5. A **program**, which is a set of rules that tell us, on the basis of the letter the tape head just read, how to change states, what to print, and where to move the tape head. We use five symbols

   (*state*, *letter read*, *new state*, *letter write*, *direction*)

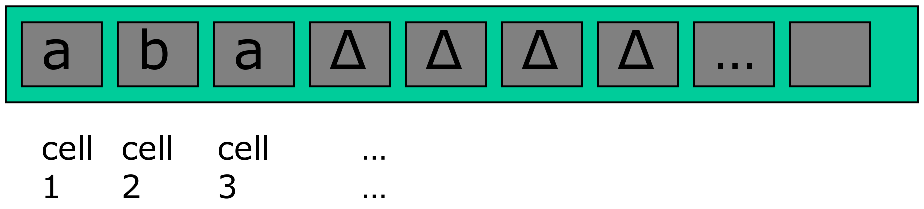
   to indicate such a rule.

Notes on Turing Machine

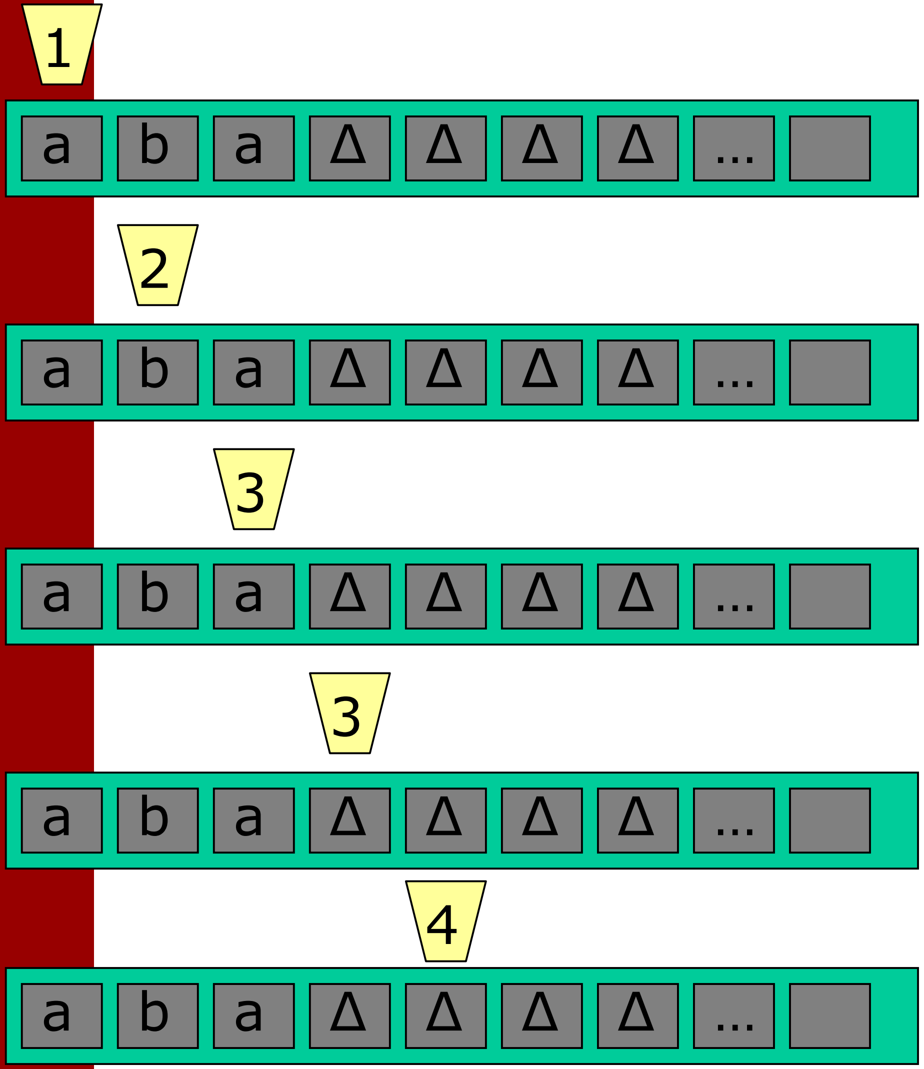
- The character set will be finite, but the number and symbols are unspecified. Same for the states.
- The machine begins from the START state and stops at the HALT state.
- Illegal action causes the machine to crash (such as move to left when it is on cell 1, or moved to state *q* but no instruction found for that state).

Turing Machine to Recognize a Symbol *b* in the Second Cell

- The symbols will be [a, b, space=Δ]
- States: 1=START, 2, 3, 4=HALT
- Program: (1,a,2,a,R)
  (1,b,2,b,R)
  (2,b,3,b,R)
  (3,a,3,a,R),(3,b,3,b,R)
  (3,Δ,4,Δ,R)
- Starting tape is



Execution of the Turing Machine Example



Start machine by position the head in cell 1, in state 1.

First rule says change state to 2, move to right

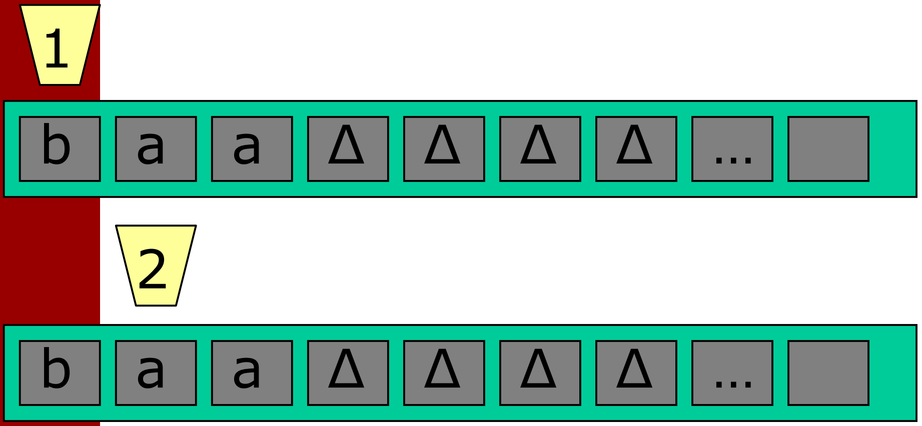
3rd rule says change state to 3, move to right

4th rule says, move to right

Last rule says change to state 4. Since we are in state 4, we must halt. Program executed successfully.

Program:
(1,a,2,a,R)
(1,b,2,b,R)
(2,b,3,b,R)
(3,a,3,a,R)
(3,b,3,b,R)
(3,Δ,4,Δ,R)
1=start,
4=halt.

Execution of the Turing Machine Example (different input)



Start machine by position the head in cell 1, in state 1.

Second rule says change state to 2, move to right.

When in state 2 and reading *a*, there is no instruction as what to do, the machine crashes (given you the answer that there is no *b* in cell 2).

Program:
(1,a,2,a,R)
(1,b,2,b,R)
(2,b,3,b,R)
(3,a,3,a,R)
(3,b,3,b,R)
(3,Δ,4,Δ,R)
1=start,
4=halt.

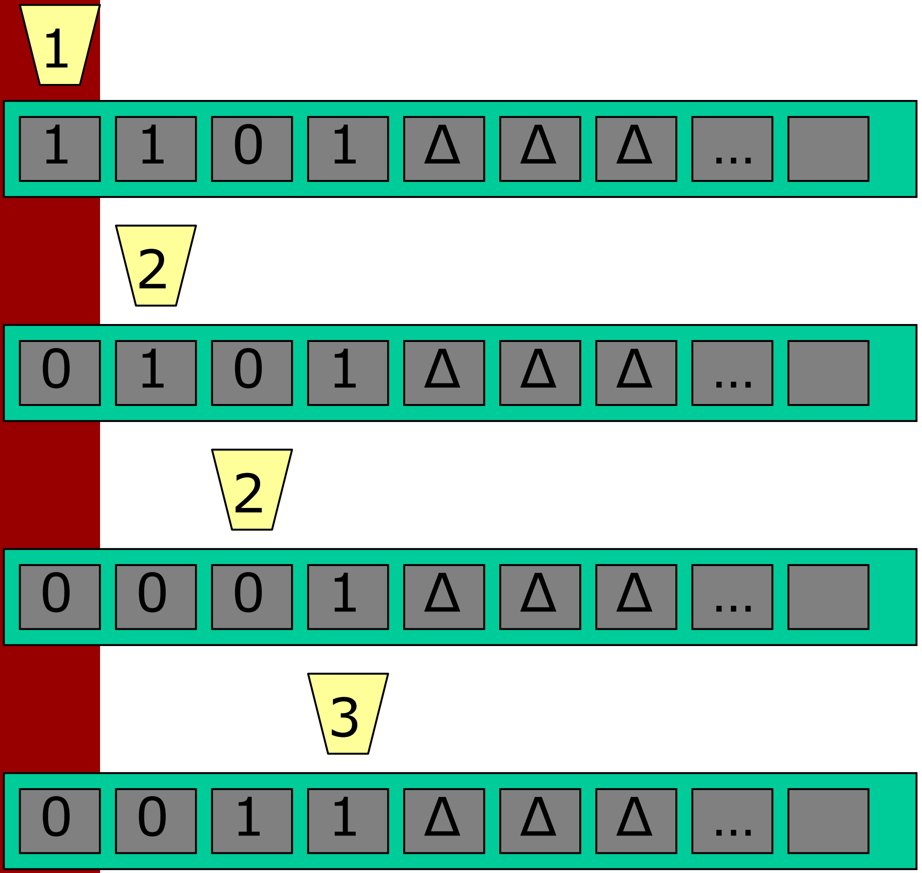
Adding 1 to a Binary Number

- The states are [1=START, 2, 3=HALT]
- Characters on tape are [0,1,space=Δ]
- The program is

(1,0,3,1,R)
(1,1,2,0,R)
(2,0,3,1,R)
(2,1,2,0,R)

Meaning of the "program": When start (state 1) if the digit under head is 0, change it to 1, and halt (e.g., 1100 + 1 = 1101). If it is 1, change it to 0 and remember the carry by putting it in state 2. In state 2, we need to add the carry, just like it is in state 1.

Running of the Adding-1-program: Add 1 to 1011



We set up the input tape with the binary number 1011, least significant digit at the leftmost.

By 2nd rule, change tape to 0, change state to 2, move to right.

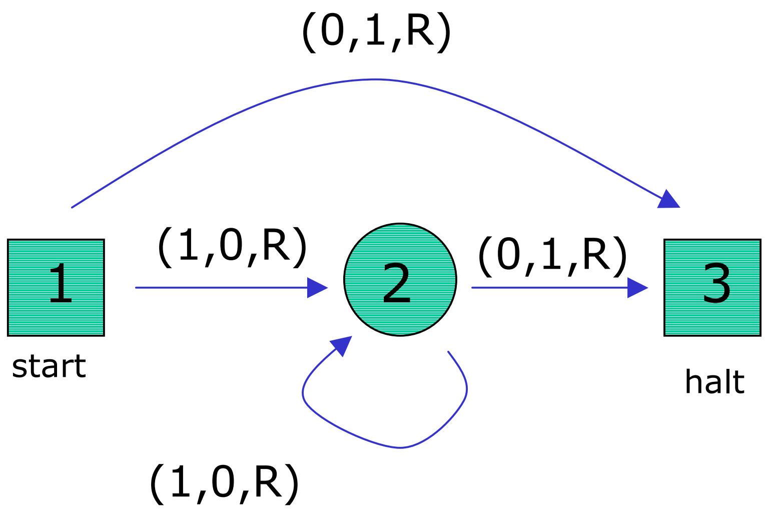
By 4th rule, change tape to 0, move to right.

By 3rd rule, change 0 to 1, and halt. The result is 1100.

(1,0,3,1,R)
(1,1,2,0,R)
(2,0,3,1,R)
(2,1,2,0,R)

First pair of numbers are current state and reading, second pair new state and writing, last symbol (R or L) is moving direction.

Another Way to Represent Turing Machine Programs



A list of 5 Items:
(1,0,3,1,R)
(1,1,2,0,R)
(2,0,3,1,R)
(2,1,2,0,R)

The number in square or circle represents the current state of the machine. The arrow points to next state in a move. The triplet of symbols means (read, write, direction). Where to move is determined by what is read currently.

What Does This Program Do?

- States [1=START,2,3,4,5,6,7,8=HALT]
- Characters [a,b,Δ]
- Program:

  (1,a,2,Δ,R), (1,b,5,Δ,R), (1,Δ,8,Δ,R), (2,a,2,a,R), (2,b,2,b,R), (2,Δ,3,Δ,L), (3,a,4,Δ,L), (3,Δ,8,Δ,R), (4,a,4,a,L), (4,b,4,b,L), (4,Δ,1,Δ,R), (5,a,5,a,R), (5,b,5,b,R), (5,Δ,6,Δ,L), (6,Δ,8,Δ,R), (6,b,7,Δ,L), (7,a,7,a,L), (7,b,7,b,L), (7,Δ,1,Δ,R).

The program determines if the strings formed by *a*'s and *b*'s are palindrome. It halts if yes, crashes if no.

The Halting Problem

The **halting problem** is a decision problem which can be informally stated as follows:

> *Given a description of a program and its initial input, determine whether the program, when executed on this input, ever halts (completes). The alternative is that it runs forever without halting.*

Alan Turing proved in 1936 that a general algorithm to solve the halting problem for *all* possible program-input pairs cannot exist.

We say that the halting problem is **undecidable** over Turing machines.

The Halting Problem: Sketch of Proof / 1

The proof proceeds by reductio ad absurdum. Start by choosing a programming language, a scheme that associates every program with at least one string description. Now suppose that someone claims to have found an algorithm halt(p, i) that returns **true** if *p* describes a program that halts when given as input the string *i*, and returns **false** otherwise.

Construct another program trouble that uses halt as a subroutine:

```
function trouble(string s)
    if halt(s, s) == FALSE then return TRUE
    else loop forever
```

This program takes a string *s* as its argument and runs the algorithm halt, giving it *s* both as the description of the program to check and as the initial data to feed to that program. If halt returns **false**, then trouble returns **true**, otherwise trouble goes into an infinite loop.

Since all programs can be represented by strings, there is a string *t* that represents the program trouble. Does trouble(t) halt?

The Halting Problem: Sketch of Proof / 2

Consider both cases:

1. If trouble(t) halts, it must be because halt(t, t) returned **false**, but that would mean that trouble(t) should not have halted.
2. If trouble(t) runs forever, it is either because halt itself runs forever, or because it returned **true**. This would mean either that halt does not work for every valid input, or that trouble(t) should have halted.

Either case concludes that halt did not give a correct answer, contrary to the original claim. Since the same reasoning applies to *any* program that someone might offer as a solution to the halting problem, there can be no solution.

The Halting Problem: Sketch of Proof / 3

This classic proof is typically referred to as the **diagonalization proof**, so called because if one imagines a grid containing all the values of halt(p, i), with every possible *p* value given its own row, and every possible *i* value given its own column, then the values of halt(s, s) are arranged along the main diagonal of this grid.

The proof can be framed in the form of the question: what row of the grid corresponds to the string *t*?

The answer is that the trouble function is devised such that halt(t, i) differs from every row in the grid in at least one position: namely, the main diagonal, where *t*=*i*.

This contradicts the requirement that the grid contains a row for every possible *p* value, and therefore constitutes a proof by contradiction that the halting problem is undecidable.

Back to the *Entscheidungsproblem*

The negative answer to the *Entscheidungsproblem* was then given by Alonzo Church in 1936 and independently shortly thereafter by Alan Turing, also in 1936.

Turing reduced the halting problem for Turing machines to the Entscheidungsproblem, and his paper is generally considered to be much more influential than Church's.

The work of both authors was heavily influenced by Kurt Gödel's earlier work on his incompleteness theorem, especially by the method of assigning numbers (a Gödel numbering) to logical formulas in order to reduce logic to arithmetic.

Turing's argument follows. Suppose we had a general decision algorithm for first-order logic. The question whether a given Turing machine halts or not can be formulated as a first-order statement, which would then be susceptible to the decision algorithm. But Turing had proved earlier that no general algorithm can decide whether a given Turing machine halts.

Universal Turing Machine

- It is possible to build (the most powerful) Turing machine that can simulate the behavior of any other Turing machine (including itself).
- A Turing machine exists that take an encoded Turing machine *T* and its input data as input on its tape. The result is the same as if executing the lesser Turing machine *T*.

The Church-Turing Thesis

The thesis can be stated as follows:

> *"Every 'function which would naturally be regarded as computable' can be computed by a Turing machine."*

Due to the vagueness of the concept of a "function which would naturally be regarded as computable", the thesis cannot formally be proven. Disproof would be possible only if humanity found ways of building hypercomputers whose results should "naturally be regarded as computable".

Any computer program can be translated into a Turing machine, and any Turing machine can be translated into any general-purpose programming language, so the thesis is equivalent to saying that any general-purpose programming language is sufficient to express any algorithm.

Can Machines Think? The Turing Test

(From, A. M. Turing, *Computer Machinery and Intelligence*, 1950)

I propose to consider the question, "Can machines think?" This should begin with definitions of the meaning of the terms "machine" and "think." The definitions might be framed so as to reflect so far as possible the normal use of the words, but this attitude is dangerous, If the meaning of the words "machine" and "think" are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning and the answer to the question, "Can machines think?" is to be sought in a statistical survey such as a Gallup poll. But this is absurd. Instead of attempting such a definition I shall replace the question by another, which is closely related to it and is expressed in relatively unambiguous words.

The Imitation Game

The new form of the problem can be described in terms of a game which we call the 'imitation game." It is played with three people, a man (A), a woman (B), and an interrogator (C) who may be of either sex. The interrogator stays in a room apart front the other two. The object of the game for the interrogator is to determine which of the other two is the man and which is the woman. He knows them by labels X and Y, and at the end of the game he says either "X is A and Y is B" or "X is B and Y is A." The interrogator is allowed to put questions to A and B thus:

C: Will X please tell me the length of his or her hair?

Now suppose X is actually A, then A must answer. It is A's object in the game to try and cause C to make the wrong identification. His answer might therefore be:

"My hair is shingled, and the longest strands are about nine inches long."

In order that tones of voice may not help the interrogator the answers should be written, or better still, typewritten. The ideal arrangement is to have a teleprinter communicating between the two rooms. Alternatively the question and answers can be repeated by an intermediary. The object of the game for the third player (B) is to help the interrogator. The best strategy for her is probably to give truthful answers. She can add such things as "I am the woman, don't listen to him!" to her answers, but it will avail nothing as the man can make similar remarks.

We now ask the question, "What will happen when a machine takes the part of A in this game?" Will the interrogator decide wrongly as often when the game is played like this as he does when the game is played between a man and a woman? These questions replace our original, "Can machines think?"

The new problem has the advantage of drawing a fairly sharp line between the physical and the intellectual capacities of a man. No engineer or chemist claims to be able to produce a material which is indistinguishable from the human skin. It is possible that at some time this might be done, but even supposing this invention available we should feel there was little point in trying to make a "thinking machine" more human by dressing it up in such artificial flesh. The form in which we have set the problem reflects this fact in the condition which prevents the interrogator from seeing or touching the other competitors, or hearing -their voices. Some other advantages of the proposed criterion may be shown up by specimen questions and answers. Thus:

Q: Please write me a sonnet on the subject of the Forth Bridge.

A : Count me out on this one. I never could write poetry.

Q: Add 34957 to 70764.

A: (Pause about 30 seconds and then give as answer) 105621.

Q: Do you play chess?

A: Yes.

Q: I have K at my K1, and no other pieces. You have only K at K6 and R at R1. It is your move. What do you play?

A: (After a pause of 15 seconds) R-R8 mate.

ELIZA

ELIZA is a famous 1966 computer program by Joseph Weizenbaum, which parodied a Rogerian therapist, largely by rephrasing many of the patient's statements as questions and posing them to the patient.

Weizenbaum chose the context of psychotherapy to "sidestep the problem of giving the program a data base of real-world knowledge", the therapeutic situation being one of the few real human situations in which a human being can reply to a statement with a question that indicates very little specific knowledge of the topic under discussion.

Eliza worked by simple parsing and substitution of key words into canned phrases. Depending upon the initial entries by the user the illusion of a human writer could be instantly dispelled, or could continue through several interchanges.

It was sometimes so convincing that there are many anecdotes about people becoming very emotionally caught up in dealing with ELIZA for several minutes until the machine's true lack of understanding became apparent. All this was due to people's tendency to attach to words meanings which the computer never put there.

An Example (Real) Conversation with ELIZA / 1

young woman: Men are all alike.

eliza: In what way?

young woman: They're always bugging us about something specific or other.

eliza: Can you think of a specific example?

young woman: Well, my boyfriend made me come here.

eliza: Your boyfriend made you come here?

young woman: He says I'm depressed much of the time.

eliza: I am sorry to hear you are depressed.

young woman: It's true. I'm unhappy.

eliza: Do you think coming here will help you not to be unhappy?

young woman: I need some help, that much seems certain.

eliza: What would it mean to you if you got some help?

young woman: Perhaps I could learn to get along with my mother.

eliza: Tell me more about your family.

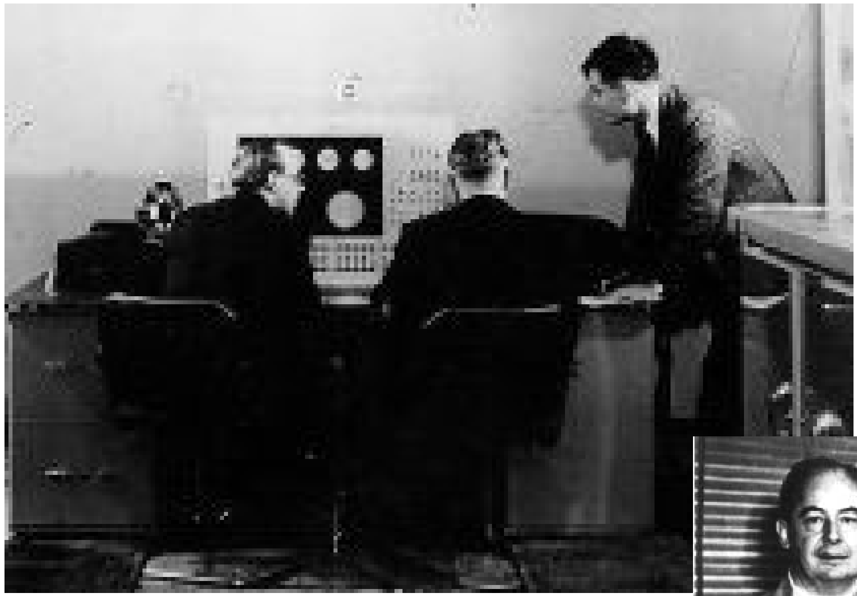
young woman: My mother takes care of me.

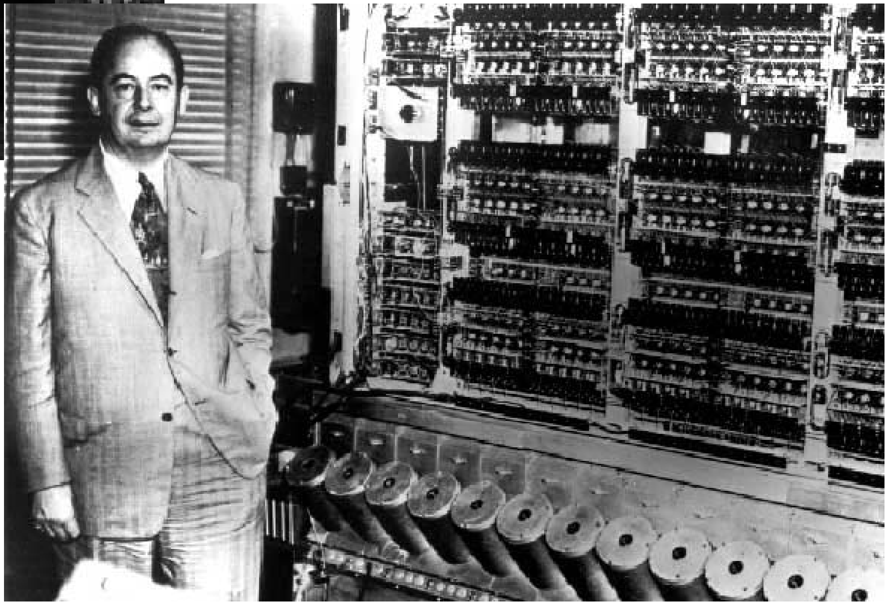
eliza: Who else in your family takes care of you?

An Example (Real) Conversation with ELIZA / 2

young woman: My father.
eliza: Your father.
young woman: You are like my father in some ways.
eliza: What resemblance do you see?
young woman: You are not very aggressive but I think you don't want me to notice that.
eliza: What makes you think I am not very aggressive?
young woman: You don't argue with me.
eliza: Why do you think I don't argue with you?
young woman: You are afraid of me.
eliza: Does it please you to believe I am afraid of you?
young woman: My father is afraid of everybody.
eliza: What else comes to mind when you think of your father?
young woman: Bullies.
eliza: Does that have anything to do with the fact that your boyfriend made you come here
….

Who Invented the First Computer? ACE vs EDVAC





A Step Back: The ENIAC

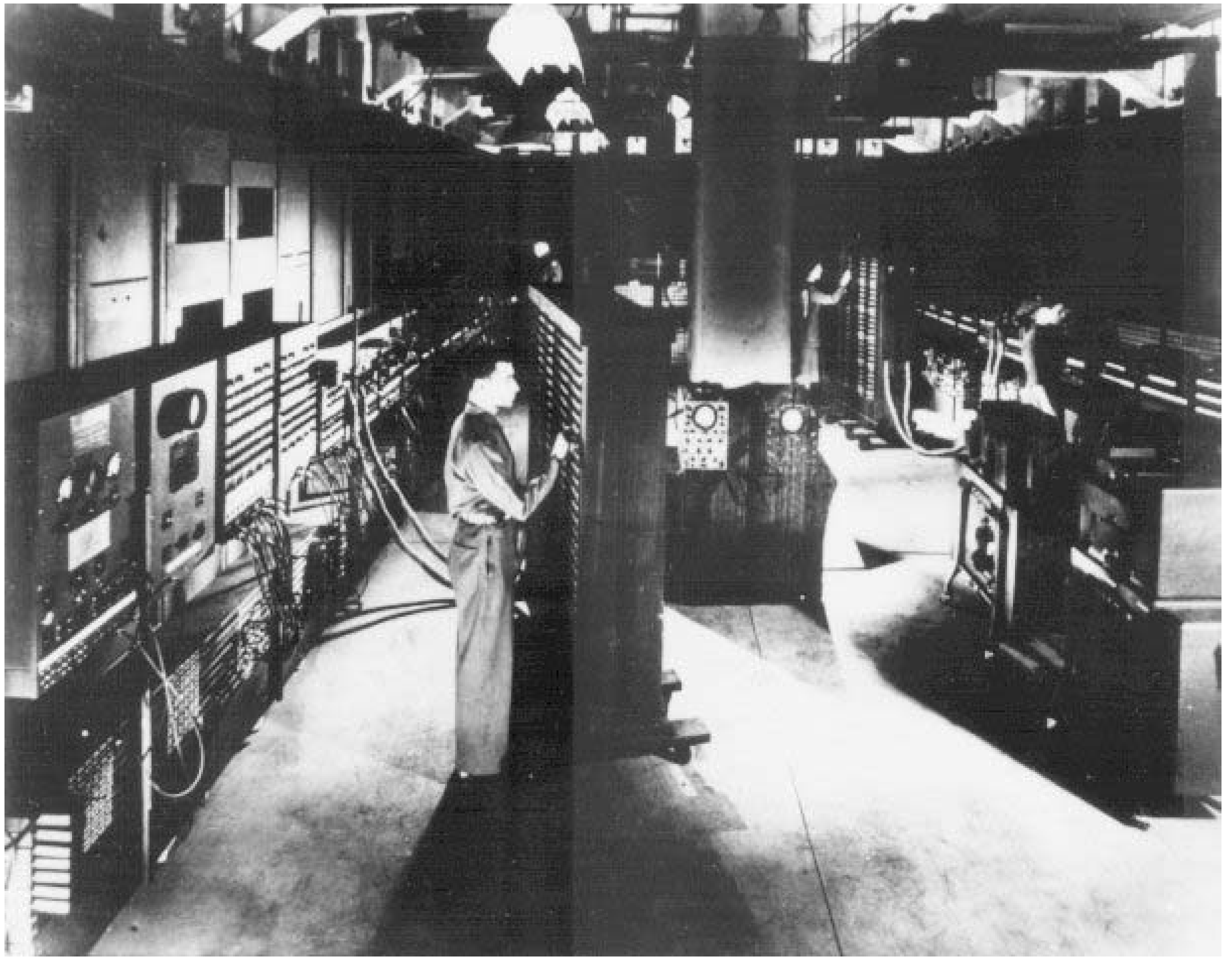
ENIAC, short for **Electronic Numerical Integrator and Computer**, was the first large-scale, electronic, digital computer capable of being reprogrammed to solve a full range of computing problems, although earlier computers had been built with some of these properties.

ENIAC was developed and built by the U.S. Army for their Ballistics Research Laboratory in 1942, with the purpose of calculating ballistic firing tables.

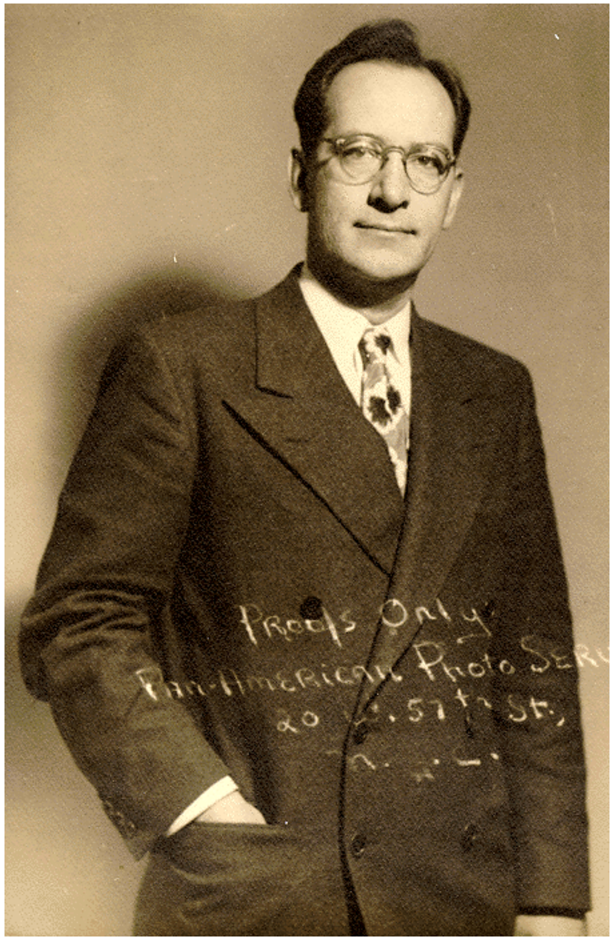
The computer was commissioned on May 17, 1943 as *Project PX* and constructed at **Penn's Moore School of Electrical Engineering** from mid-1944. It was unveiled on February 14, 1946 at the University of Pennsylvania, having cost almost $500,000.

ENIAC was shut down on November 9, 1946 for a refurbishment and a memory upgrade, and was transferred to the Aberdeen Proving Ground, Maryland in 1947. There, on July 29 of that year, it was turned on and would be in continuous operation until 11:45 p.m. on Oct. 2, 1955.

The ENIAC



John Presper Eckert Jr. and John Mauchly



Physically, ENIAC was a monster. It contained 17,468 vacuum tubes, 7,200 crystal diodes, 1,500 relays, 70,000 resistors, 10,000 capacitors and around 5 million hand-soldered joints.

It weighed 30 short tons (27 t), was roughly 8 feet (2.4 m) by 3 feet (0.9 m) by 100 feet (30 m), took up 1800 square feet (167 m^2), and consumed 150 kW of power.

Input was possible from an IBM card reader, while an IBM card punch was used for output. These cards could be used to produce printed output offline using an IBM accounting machine, probably the IBM 405.

ENIAC

Some *electronics* experts predicted that tube failures would occur so frequently that the machine would never be useful. This prediction turned out to be partially correct: several tubes burned out almost every day, leaving it nonfunctional about half the time. (According to a 1989 interview with Eckert the continuously failing tubes story was a myth: "We had a tube fail about every two days and we could locate the problem within 15 minutes.") Special high-reliability tubes were not available until 1948.

Most of these failures, however, occurred during the warm-up and cool-down periods, when the tube heaters and cathodes were under the most thermal stress. By the simple (if expensive) expedient of *never turning the machine off*, the engineers reduced ENIAC's tube failures to the more acceptable rate of one tube every two days.

In 1954, the longest continuous period of operation without a failure was 116 hours (close to five days). Given the technology available at the time, this failure rate was remarkably low, and stands as a tribute to the precise engineering of ENIAC

EDVAC

- **EDVAC** (***E****lectronic* ***D****iscrete* ***V****ariable* ***A****utomatic* ***C****omputer*) was one of the earliest electronic computers. Unlike the ENIAC, it was binary rather than decimal.
- The design for the EDVAC was developed before the ENIAC was even operational. It was intended to resolve many of the problems created by the ENIAC's design. Like the ENIAC, the EDVAC was built for the U.S. Army's Ballistics Research Laboratory at the Aberdeen Proving Ground by the University of Pennsylvania. The ENIAC designers Eckert & Mauchly were joined by John von Neumann and some others and the new design was based on von Neumann's 1945 report.
- A contract to build the new computer was signed in April 1946 with an initial budget of US$100,000 and the contract named the device the Electronic Discrete Variable Automatic *Calculator*. A major concern in construction was to balance reliability and economy. The final cost of EDVAC, however, ended up similar to the ENIAC's at just under $500,000; five-fold the initial estimate

John von Neumann (1903-1957)

John von Neumann was a Hungarian mathematician and polymath of Jewish ancestry who made important contributions in quantum physics, functional analysis, set theory, economics, computer science, numerical analysis, hydrodynamics (of explosions), statistics and many other mathematical fields.

Most notably, von Neumann was a pioneer of the modern digital computer and the application of operator theory to quantum mechanics, a member of the Manhattan Project Team, and creator of game theory and the concept of cellular automata.

Along with Edward Teller and Stanislaw Ulam, von Neumann worked out key steps in the nuclear physics involved in thermonuclear reactions and the hydrogen bomb.

Eckert was delighted that von Neumann was so keenly interested in the logical problems surrounding the new idea, and these meetings were scenes of greatest intellectual activity.
This work on the logical plan for the new machine was exactly to von Neumann’s liking and precisely where his previous work on formal logics came to play a decisive role. Prior to his appearance on the scene, the group at the Moore School concentrated primarily on the *technological* problems, which were very great; after his arrival he took over leadership on the *logical* problems.

H. Goldstine

During the latter part of 1944, and continuing to the present time, Dr. John von Neumann . . . has fortunately been available for consultation. He has contributed to many discussions on the logical controls of the EDVAC, has prepared certain instruction codes, and has tested these proposed systems by writing out the coded instructions for specific problems. Dr. von Neumann has also written a preliminary report in which most of the results of earlier discussions are summarized. . . . In his report, the physical structures and devices . . . are replaced by idealized elements to avoid raising engineering problems which might distract attention from the logical considerations under discussion.

John Presper Eckert Jr. e John Mauchly

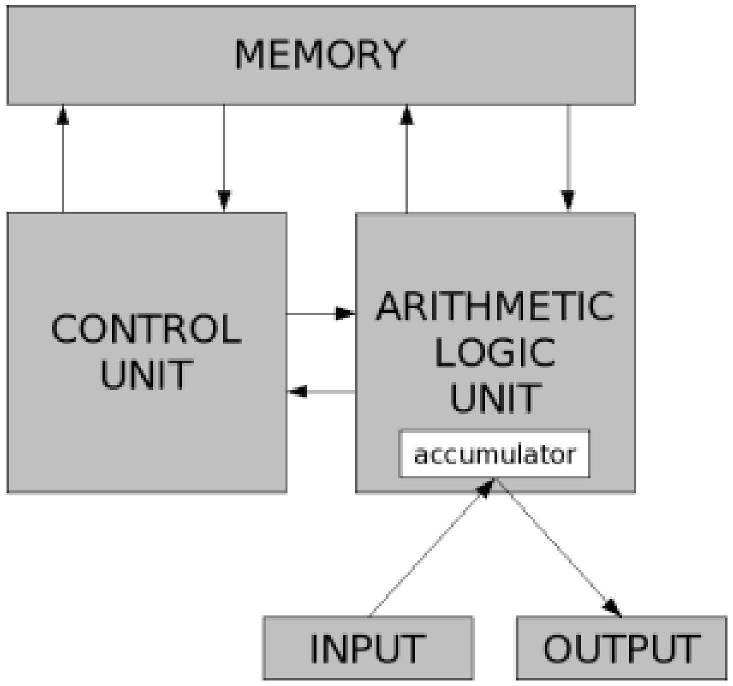
General-purpose Computers

In order to test the general applicability of the EDVAC, von Neumann wrote his first serious program, not for the kind of number crunching application for which the machine was mainly developed, but rather to simply sort data efficiently. The success of this program helped to convince von Neumann that

``… it is legitimate to conclude already on the basis of the now available evidence, that the EDVAC is very nearly an `all purpose' machine, and that the present principles for the logical controls are sound.''

John von Neumann

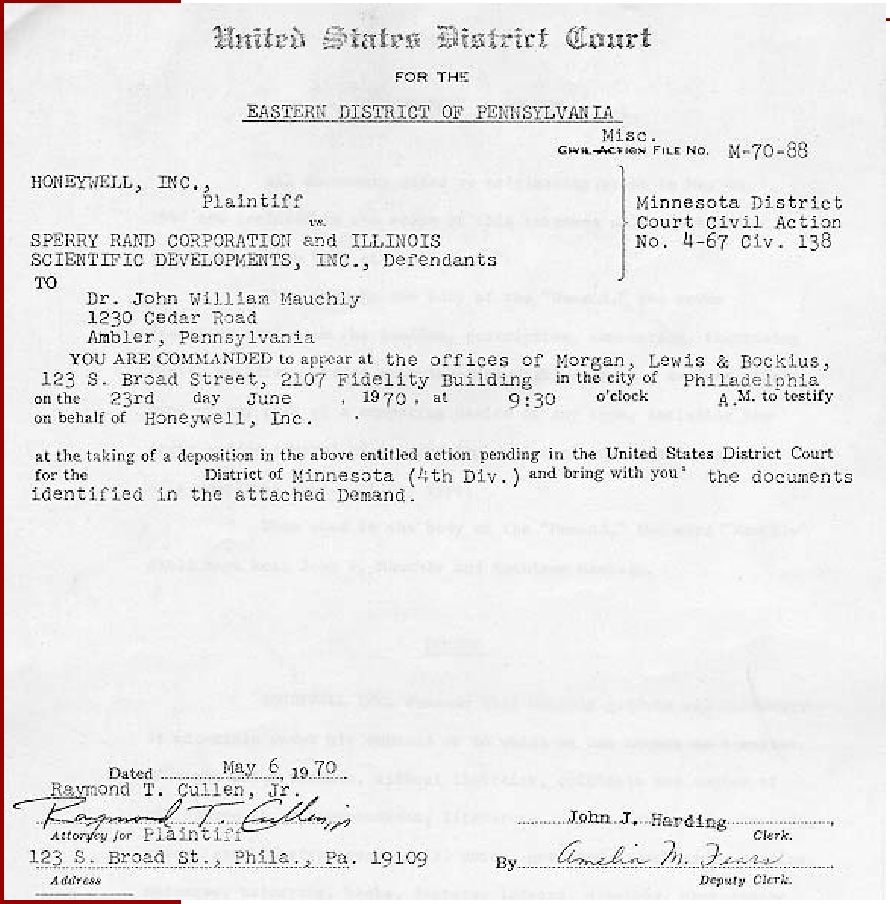
The von Neumann Architecture



The term **von Neumann architecture** refers to a computer design model that uses a single storage structure to hold both instructions and data. The term von Neumann machine can be used to describe such a computer, and remember that it implements a Turing machine.

The separation of storage from the processing unit is implicit in the von Neumann architecture.

The term "stored-program computer" is generally used to mean a computer of this design.



United States District Court

FOR THE

EASTERN DISTRICT OF PENNSYLVANIA

Misc.
~~CIVIL ACTION~~ FILE NO. M-70-88

HONEYWELL, INC.,
Plaintiff
vs.
SPERRY RAND CORPORATION and ILLINOIS
SCIENTIFIC DEVELOPMENTS, INC., Defendants

Minnesota District
Court Civil Action
No. 4-67 Civ. 138

TO
Dr. John William Mauchly
1230 Cedar Road
Ambler, Pennsylvania

YOU ARE COMMANDED to appear at the offices of Morgan, Lewis & Bockius, 123 S. Broad Street, 2107 Fidelity Building in the city of Philadelphia on the 23rd day June , 1970 , at 9:30 o'clock A.M. to testify on behalf of Honeywell, Inc.

at the taking of a deposition in the above entitled action pending in the United States District Court for the District of Minnesota (4th Div.) and bring with you the documents identified in the attached Demand.

Dated May 6, 1970

Raymond T. Cullen, Jr.
Raymond T. Cullen, Jr.
Attorney for Plaintiff
123 S. Broad St., Phila., Pa. 19109
Address

John J. Harding
Clerk.

By [illegible]
Deputy Clerk.

In October 1973 Judge Earl Larson of the U.S. District Court in Minnesota rendered a decision invalidating the ENIAC patent. But rather than being a clear judgement as to who invented the electronic computer, this decision and the law suit, Honeywell v. Sperry-Rand, have done more to polarize opinions with respect to the varied contributions of many different individuals. In fact, this decision points to some of the limitations of the U.S. patent system with respect to complex technologies. Namely, the U.S. patent system sets forth certain pressures to name a sole inventor when invention itself is a often a highly collaborative process. We hope that this exhibition reveals something of the complexities involved in the process of invention. We hope also that in approaching the fifty-year mark of modern computing, we can recognize the diverse contributions of individuals, regardless of what we individually consider to be its origins.

The ACE

The **ACE (Automatic Computing Engine)** was the first computer designed in Britain; it was designed by Alan Turing in 1946.

On February 19, 1946 Turing presented a paper to the National Physical Laboratory (NPL) Executive Committee, giving the first complete design of a stored-program computer.

Unlike most other early computers, it owed nothing to EDVAC; it was a completely independent design which was contemporaneous with EDVAC.

The ACE had a 48-bit word. It used delay line main memory, and contained about 7000 vacuum tubes. Its multiplication time was about 448 microseconds.

Due to various difficulties, the first version of the ACE actually built was the Pilot ACE, a smaller version of Turing's original complete design.

The full-scale version was constructed later, in the late 1950s; it was working by late 1957, but was already obsolete, due to its reliance on delay-line main memory.

[It] is … very contrary to the line of development here, and much more in the American tradition of solving one's difficulties by means of much equipment rather than by thought.

… Furthermore certain operations which we regard as more fundamental than addition and multiplication have been omitted.

Alan Turing

By the end of 1945, Turing had produced his remarkable ACE (Automatic Computing Engine) Report. One detailed comparison of the ACE Report with von Neumann's EDVAC Report, notes that whereas the latter ``is a draft and is unfinished … more important … is incomplete …" the ACE Report ``is a complete description of a computer, right down to the logical circuit diagrams" and even including ``a cost estimate of £11,200."

La macchina ACE di Turing era molto diversa dall'EDVAC di von Neumann's (come lo erano I due matematici!).

Sebbene von Neumann volesse una macchina "all purpose" era molto piu' portato a pensare a una grossa calcolatrice (*number crunching*).

L'idea di Turing era di costruire una macchina piu' semplice e piu' indipendente dalle possibli applicazioni.

[…] An opposing paradigm, the so-called RISC (reduced instruction set computing) architecture, adopted by a number of computer manufacturers, uses a minimal instruction set on the chip, with needed functionality supplied by programming, again very much in line with the ACE philosophy.

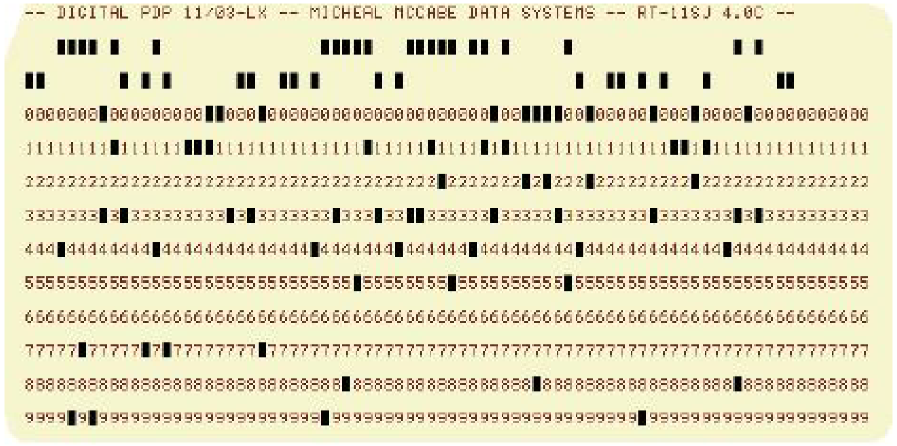
In discussions of this period, the new computers that were being developed are usually referred to as embodying "the stored program concept" because for the first time the programs to be executed were being stored within the computer. Unfortunately this terminology has served to obscure the fact that what was really revolutionary about these machines was their universal all-purpose character, while the stored program aspect was only a means to an end. The point of view of Turing and von Neumann is conceptually so simple and has so much become part of our intellectual climate, that it is diffcult to understand how radically new it was.

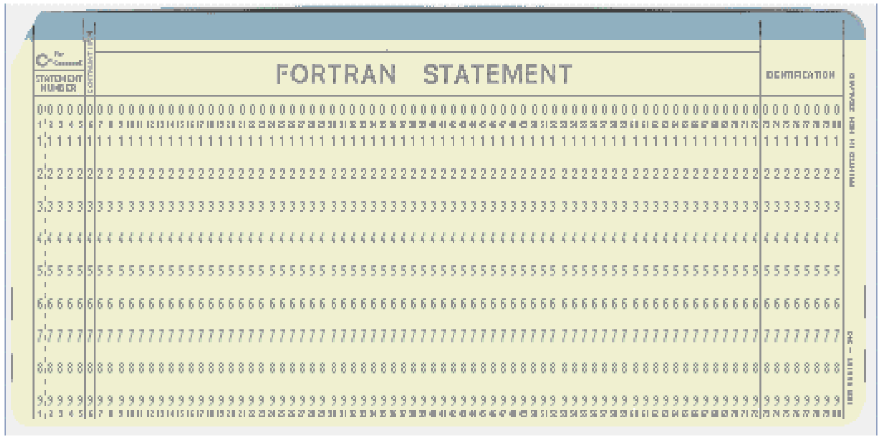
ENIAC (e UNIVAC): l'impresa ingegneristica (e commerciale)

Remington Rand presents
THE ELECTRONIC ERA FOR BUSINESS WITH UNIVAC*
FACT-TROLLER
...THE FIRST UNIVERSAL ELECTRONIC SYSTEM DESIGNED FOR BOTH MANAGEMENT AND SCIENCE
* A DEVELOPMENT OF THE ECKERT-MAUCHLY COMPUTER CORP.

IBM 701 around 1954

Punched Cards


-- DIGITAL PDP 11/03-LX -- MICHEAL MCCABE DATA SYSTEMS -- RT-11SJ 4.0C --



Programs in the 1950s to 1970s are coded on a piece of paper card with punched holes.

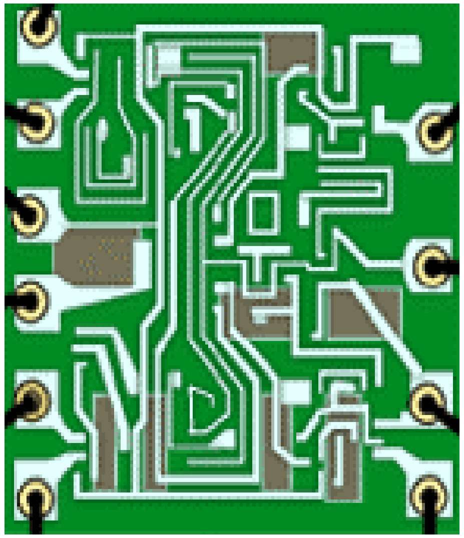
They are read by electromechanical or optical reader into the computer.

Each card can hole only one line of information.

The standard IBM card is 80 characters long.

Transistors

Integrated circuits placed all components in one chip, drastically reducing the size.

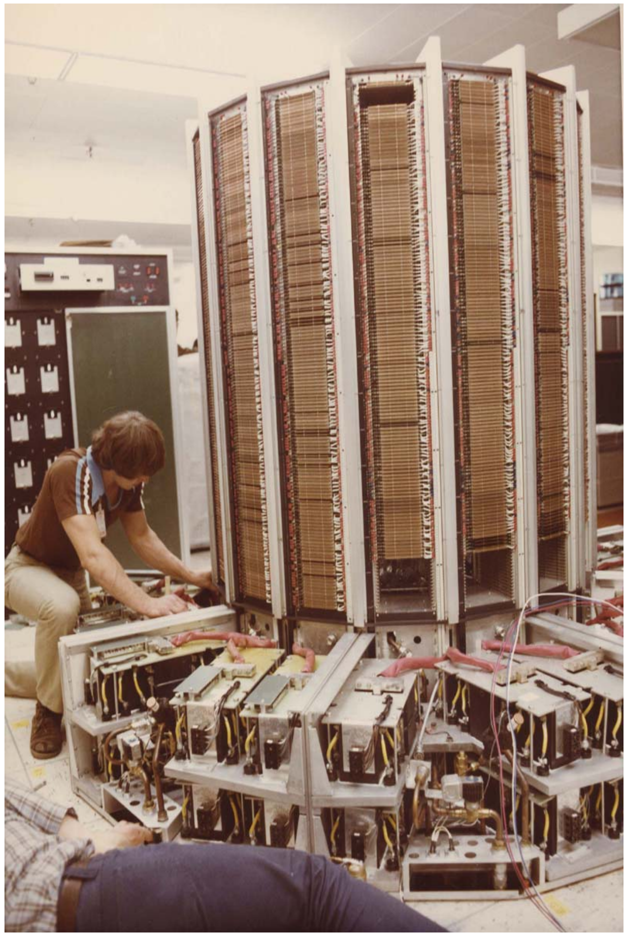


In 1947, John Bardeen and Walter Brattain invented the transistor, which quickly replaced the vacuum tube technology. Initially, electronic devices are made of individual components.

Vax-11 from Digital Equipment

Vax-11 is popular in universities in the early 1980s.

Cray Supercomputer



One of the first so-called supercomputer built around 1976. It was the fastest and also most expensive.

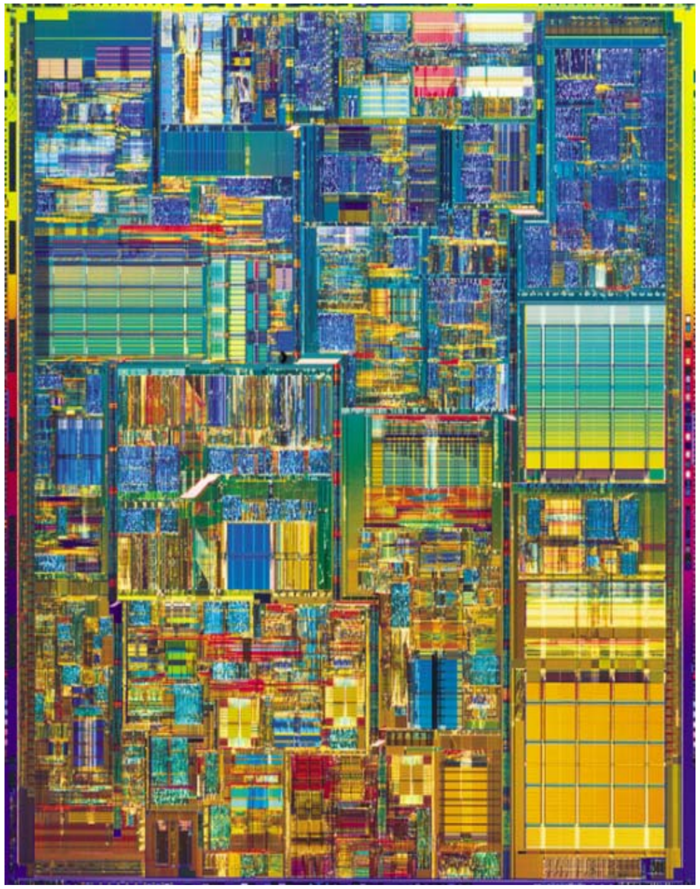
IBM PCs (1981)

The IBM Personal Computer started a revolution for computing by the common folks.

The “PC” comes with 64kilobyte of memory, 5.25 inch floppy disk drive. It runs at 4.7mega-Hertz.

The whole operating system, the Microsoft’s DOS, is on one floppy.

Very Large Scale Integrated (VLSI) Circuits



Modern computers are based on technology of very large number of components on a small silicon chip.

Portable Computing

Copyright (C), Yoko Katagiri & NihongoWeb

Nowadays in 2006, laptop of 1.2kg in weight is common place.

It runs at 1.8 GegaHertz speed with 512 MegaByte of RAM and 40 Gbyte of internal hard disk, plus DVD drive etc.

Characteristics of Commercial Computers

Year	Computer Name	Power (Watts)	Performance (adds/sec)	Memory (kByte)	Price (US dollars)
1951	UNIVAC I	124,500	1,900	48	$1,000,000
1964	IBM S360	10,000	500,000	64	$1,000,000
1965	PDP-8	500	330,000	4	$16,000
1976	Cray-1	60,000	166,000,000	32,768	$4,000,000
1981	IBM PC	150	240,000	256	$3,000
1991	HP 9000	500	50,000,000	16,384	$7,400
2006	ThinkPad T43 notebook	20	1,000,000,000	512,000	$1,900

TIME on Turing and von Neumann

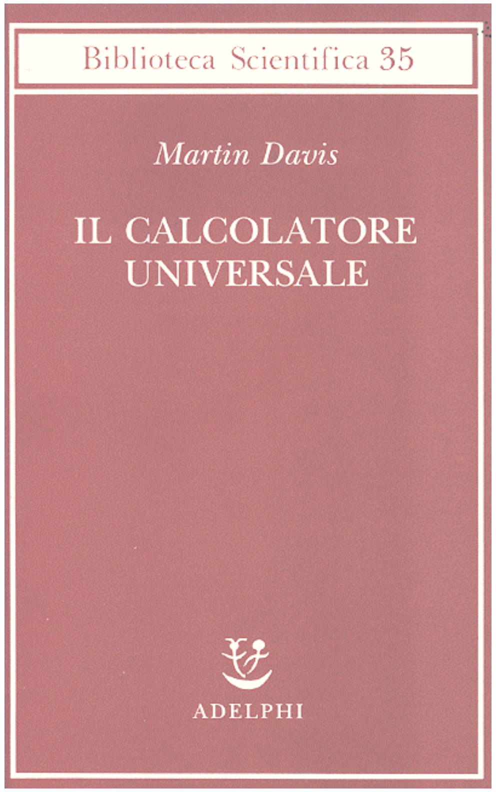
Alan Turing’s name was on the list of the twenty greatest “scientists and thinkers” of the twentieth century as proposed by TIME magazine (in its March 29, 1999 issue). Said ***TIME***:

“So many ideas and technological advances converged to create the modern computer that it is foolhardy to give one person the credit for inventing it. But the fact remains that everyone who taps at a keyboard, opening a spreadsheet or a word-processing program, is working on an incarnation of a Turing machine.”

And here is what ***TIME*** had to say about von Neumann:

“Virtually all computers today from $10 million supercomputers to the tiny chips that power cell phones and Furbies, have one thing in common: they are all “von Neumann machines,” variations on the basic computer architecture that John von Neumann, building on the work of Alan Turing, laid out in the 1940s.”

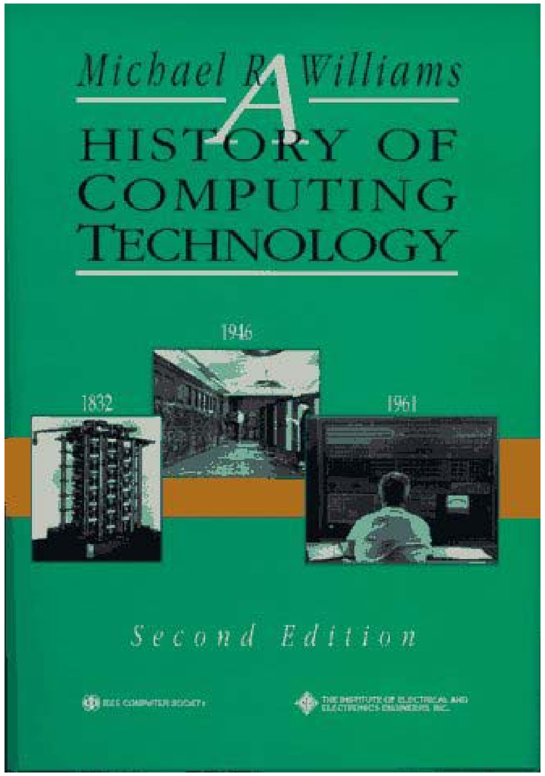
Per approfondire….


Biblioteca Scientifica 35

Martin Davis

IL CALCOLATORE UNIVERSALE

ADELPHI


Michael R. Williams

A HISTORY OF COMPUTING TECHNOLOGY

Second Edition

IEEE COMPUTER SOCIETY

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.

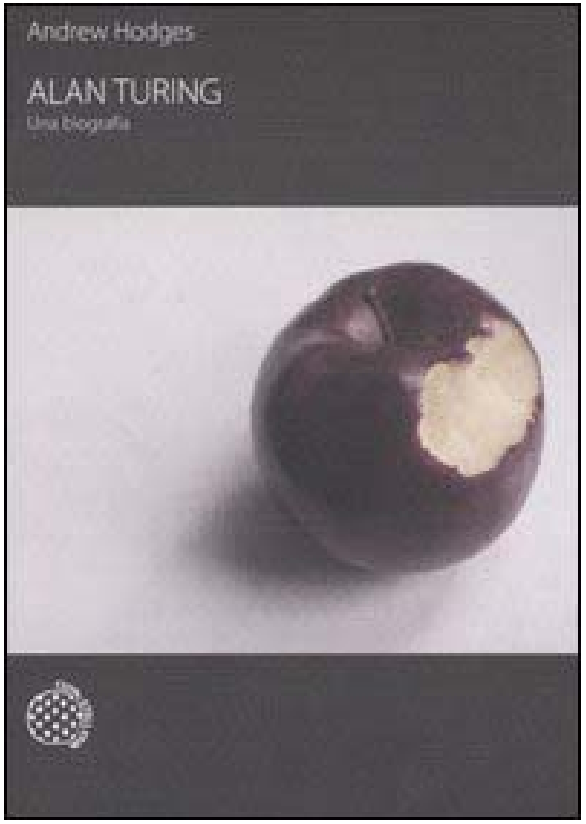
Paul E. Ceruzzi

Storia dell'informatica

Dai primi computer digitali all'era di Internet

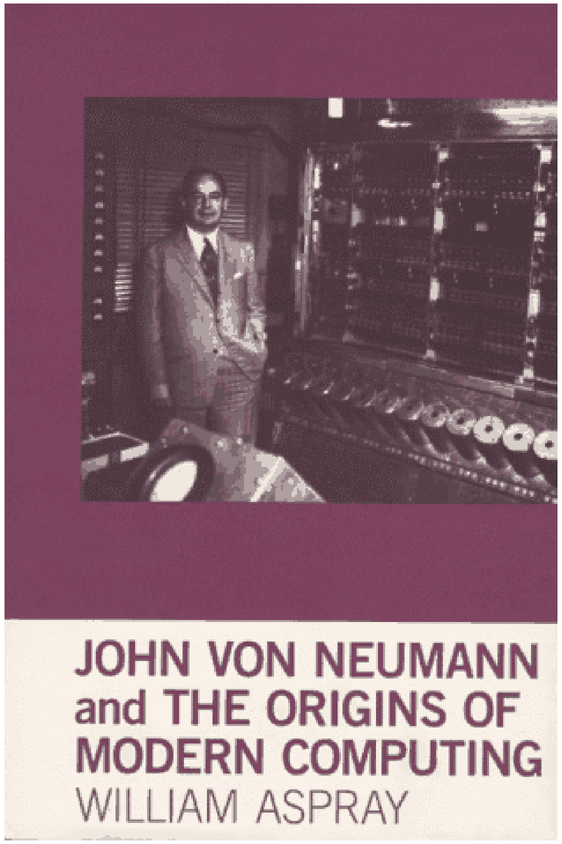
APOGEO

Su Turing e von Neumann


Andrew Hodges

ALAN TURING

Una biografia


JOHN VON NEUMANN and THE ORIGINS OF MODERN COMPUTING

WILLIAM ASPRAY