A Brief History of Punched Cards: The Era of Programming on Paper

8 min read · Aug 10, 2025

Ryena Dhingra Follow

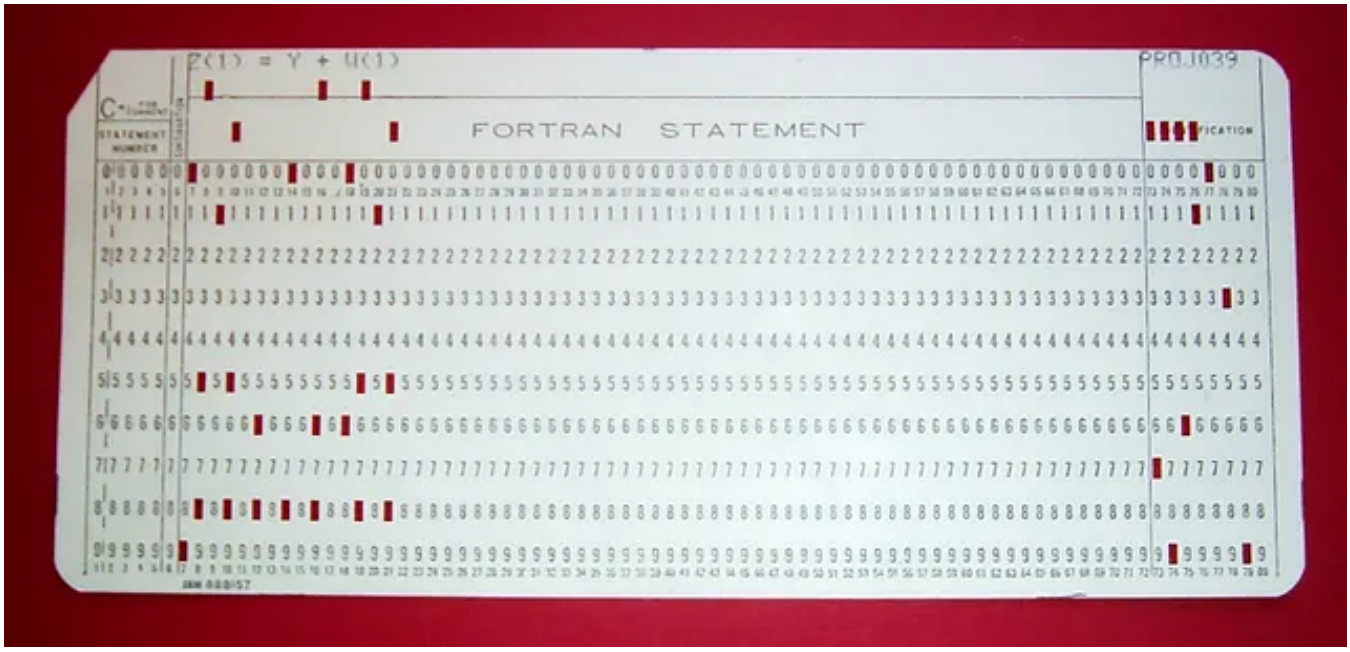
Listen Share

Margaret Hamilton (Draper Laboratory; restored by Adam Cuerden, edited by author, 1969, PD-US)

This is Margaret Hamilton, a lead computer scientist at NASA during the 60's, and next to her is the pile of code, typed by hand on paper, that landed the

first humans on the moon as part of the Apollo Space Program. While multiple remarkable stories are wrapped up in this single sentence, a question that might stand out is 'Why was this code written on paper?' To answer this we must trace the origins of programming and go back roughly 60 years to an era when computing did not exist as we know it today.

Early computers like ENIAC (1945), UNIVAC 1 (1951), IBM 1401 (1959) did not have keyboards to input code or screens to view output. These are components that are now taken for granted in any programming environment but were developed much later than when we began programming computers. One must wonder how computers were fed code back then. The answer lies in the image attached below.



Fortran punch card (Arnold Reinhold, 2006, CC BY-SA 2.5)

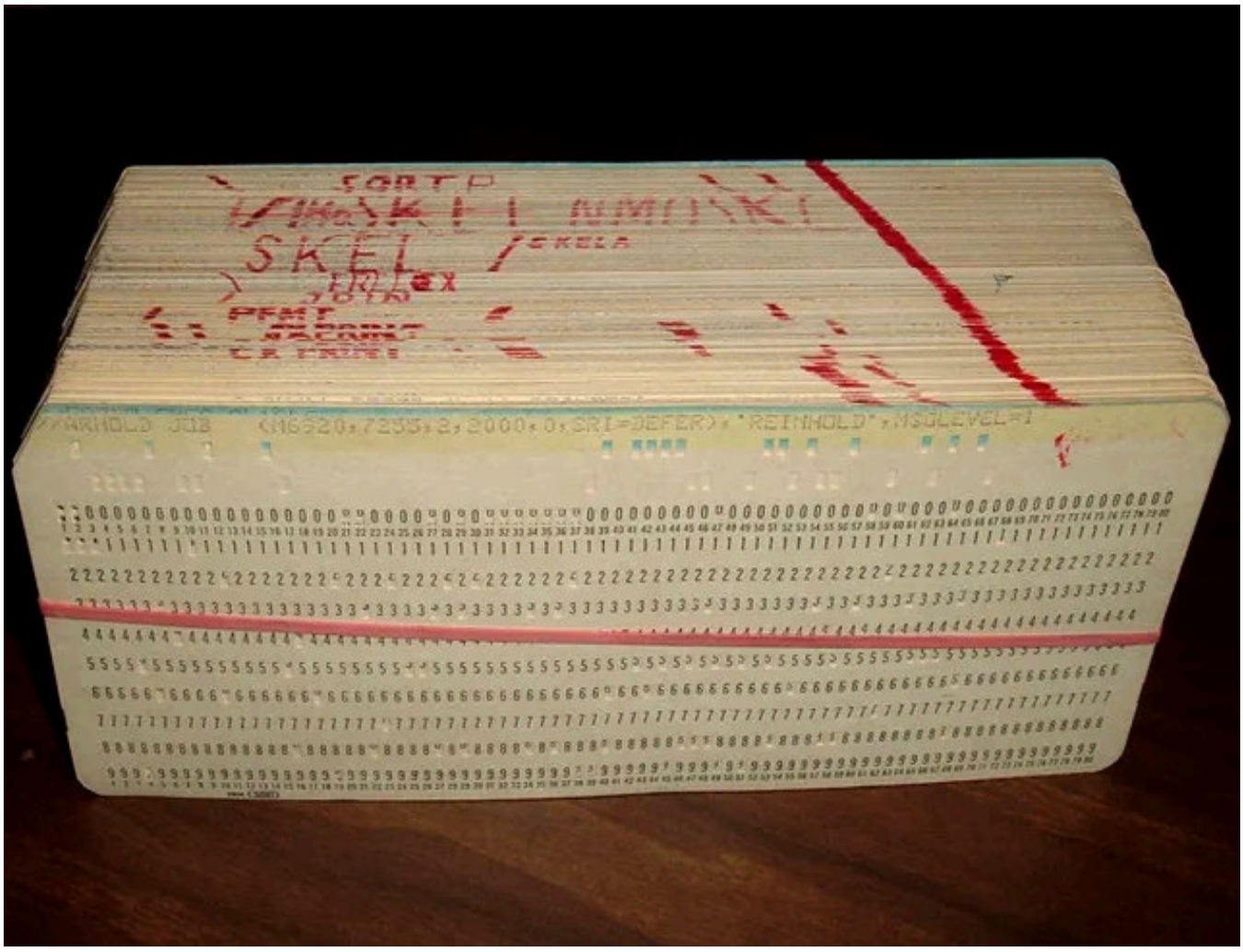
This is a punched card, made of paper. The one shown here is a standard IBM card with 80 columns and 12 rows. Each column represents one character and that character is determined by the specific pattern of holes punched in that column. When put into a card reader, the machine could understand what the card said by shining light over it. If a hole was encountered, light would pass through it, triggering the photoelectric sensors on the other side and thus passing the information to the computer as electrical signals. Common languages used during the punched card era were COBOL, ALGOL and Assembly Languages. The card shown here has a Fortran statement punched onto it.

Think of punched cards as a physical medium to store data and program before magnetic tapes and disks took over. These had to be fed to the computer as input

and the tasks were performed in strict sequence, with each card representing a specific instruction or piece of data. This meant that each card stored one line of code from the program.

Let me help you visualise the huge number of cards a typical program would require. A single simulation program run by Margaret Hamilton's team for testing spacecraft systems might require around 5,000–8,000 lines of code for the program and around 10,000 more for the data to be processed by the program. This would require 15,000–20,000 punched cards for a single simulation!

Also, since these cards had to be fed into the card reader in a strict order, imagine the catastrophe should a programmer drop his bundle of cards! To prevent this from happening, programmers would often number their cards or tip the cards to draw a slant line, as shown in the picture below, so they could be rearranged if an unfortunate disaster were to occur. Some cards, like the one having the Fortran statement above, were also numbered by punching holes on the right side of the card, so they could be sorted later by a card sorter machine.



Punched card program deck (Arnold Reinhold, 2006, CC BY-SA 3.0)

I'll show you how to read a punched card later in this blog.

Tracing the Origins of Punched Cards

Quite interestingly, punched cards have their origins in the textile industry. Their design was inspired by the Jacquard Loom which used stiff cards with holes in them to control the pattern that was to be weaved onto the cloth.

This inspiration led Herman Hollerith to conceive the idea of punched cards which he invented to record the data of individuals during the 1890 United States census. The data from the punched cards was analysed by an electro-mechanical tabulator (note that it was not a computer) which was also devised by Hollerith. Hollerith's invention reduced the duration of the census from 7 years to 2 years.

Hollerith founded the Tabulating Machine Company in 1896 to commercialise his invention. The company sold punch machines and tabulators. In 1911, the company merged with the Computing Scale Company of America and the International Time Recording Company to form what would later be known as International Business Machines (IBM).

Why Were They Needed?

The two main applications of punched cards were in

1. Business, for maintaining records and processing data
2. In Computer Science, as input and storage medium for programs and data required for programs

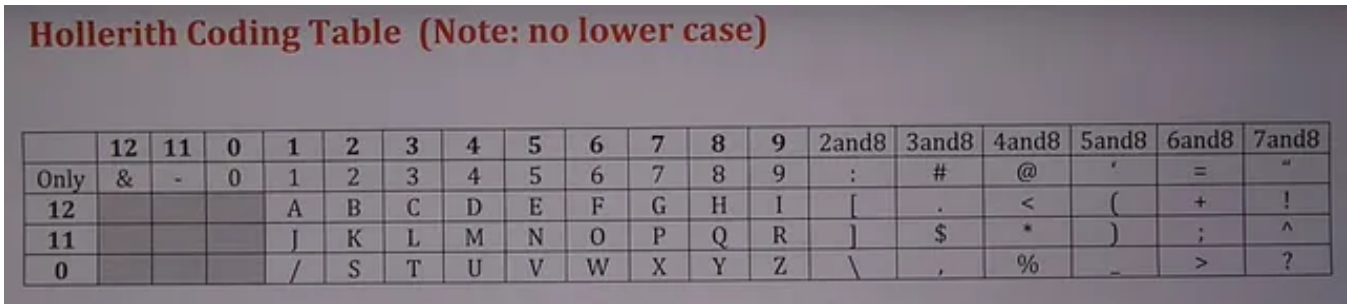
Before the integration of computers and punched cards, payroll processing, inventory management, billing and invoicing, record maintenance, insurance and banking were all done manually by people or by tabulators like the one Hollerith had invented. It was at this point in history that the gap between mechanical and digital computing was bridged. Punched cards became widely used for storing and analysing data within businesses. By the 1950's, punched cards had made their way into the world of Computer Science by becoming the main medium used for programming.

How Do You Read a Punched Card?

Let us go back to the punched card that we saw at the beginning of this blog. We can make out that each column has been encoded to represent a character. The

encoding method used for this particular card is called Hollerith Encoding (named after the same Hollerith mentioned earlier).

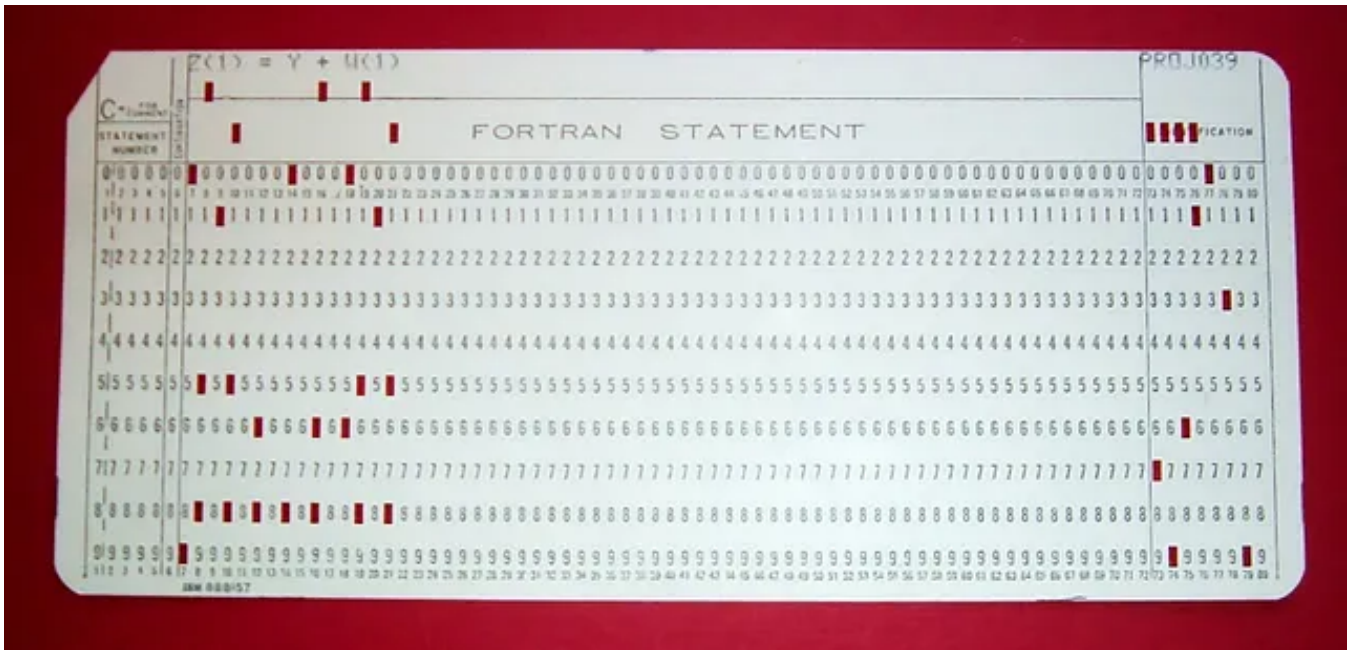
The card has 80 columns and 12 rows and each column equals one character. The Fortran statement has been typed on the top for the ease of the programmer. Notice how each column has a maximum of 3 punches. Let us refer to this table to learn how Hollerith Encoding was performed.



Hollerith Coding Table (Note: no lower case)

	12	11	0	1	2	3	4	5	6	7	8	9	2and8	3and8	4and8	5and8	6and8	7and8
Only	&	-	0	1	2	3	4	5	6	7	8	9	:	#	@	'	=	"
12				A	B	C	D	E	F	G	H	I	[	.	<	(	+	!
11				J	K	L	M	N	O	P	Q	R	]	$	*	)	;	^
0				/	S	T	U	V	W	X	Y	Z	\	,	%	_	>	?

IBM Punched Cards Explained — Video 207 (Anthony Francis-Jones, YouTube)



Fortran punch card (Arnold Reinhold, 2006, CC BY-SA 2.5)

The top three rows, 12, 11 and 0, act as the zone punches. They divide the entire set of characters into 4 zones (when 12 is punched, when 11 is punched, when 0 is punched and when no zone punch row is punched). This allows the rows 1–9 to represent a different set of characters depending on what zone the given column is in. For instance, in the card shown here, for the first character 'Z', a combination of 0 and 9 is punched. You can match this with the table. For the next character '(', the zone punch is 11 and additionally, rows 5 and 8 have been punched. See the table.

Every column can have a minimum of 1 punch and a maximum of 3 punches. Thus, Hollerith Encoding provided an efficient encoding method and also ensured that the paper wasn't made flimsy or difficult to read by making too many punches. A common saying during this time was 'do not fold, spindle or mutilate' as that could damage the cards!

Understanding the Setup and the Workflow

During the era that we have stepped into, computers didn't occupy just a quarter of your desk. Instead there were entire rooms dedicated to the setup. Remarkably, these setups weren't just single computers but entire ecosystems of machines and workstations. So far, we have understood that programmers couldn't simply write their code into the computer and wait for the output. Let us now also understand the workflow of how all this happened.

A programmer would typically begin writing the code on special paper forms called coding forms. This code was now needed to be punched onto a deck of cards line-by-line. This was done with the help of a typewriter-like machine called a keypunch. Each key on the keyboard of the keypunch represented an alphabet, number or special character. When a key was pressed, the pattern of holes corresponding to that character would be punched onto a given column. A widely used keypunch was the IBM 029 Punch Machine which has been shown below. The Fortran statement card shown earlier was also punched using this keypunch.

IBM card punch station 029 (waelder, 2007, CC BY-SA 3.0)

The next step involved feeding the deck of cards into a card reader. It was an input device that enabled the mainframe computer to read the program and the input. If the keypunch was the keyboard, the card reader was the USB port of the punched card era! The card reader would pass the deck of cards one-by-one through the reading mechanism discussed earlier and convert the hole patterns into electrical signals that the computer could process.

This was followed by a mainframe computer like the IBM 1401. Once the card reader fed the deck into the mainframe's memory, the CPU would begin executing instructions from the punched cards. The program could analyse datasets, perform calculations or control machinery in specialised systems.

The result was usually printed onto paper using a line printer like the one shown below. In some computers, the result could also be punched onto new cards using a card punch connected to the mainframe. These output cards could then be used as input for some other program.

Because batch processing was done, any error produced during the compilation was revealed at the end of the loop. Since there was no editing inside the computer, each fix meant replacing the faulty card with a newly punched one. After that, the entire process had to be done again.

A Remarkable End

I hope you have now found the answer to the question I proposed at the beginning of this blog, 'Why was the code for the Apollo Space Program written on paper?' Margaret Hamilton's team worked with the custom-built Apollo Guidance Computer (AGC) on which programs were written in the AGC Assembly Language. Punched cards were likely used as the main input medium for ground operations!

Gradually, punched cards went into oblivion as they were replaced by magnetic tapes and disks which offered higher storage capacity, faster access and easier editing. Nonetheless, it is extremely fascinating to understand their working and contribution to modern day programming.

Programming Technology Computer Science Software Engineering Software Development

Follow

Written by Ryena Dhingra

33 followers · 11 following

Responses (3)

Write a response