



What it was like programming  
a first-generation computer

Ed Barnard (Ed#1)  
@ewbarnard  
InboxDollars

## Computing Past: Mel, the Realest Programmer of All

Photo: Computer History Museum, <http://www.computerhistory.org/revolution/early-computer-companies/5/116>

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

1

Welcome! Today we'll be learning what it was like programming a first-generation computer in the 1950s. This is the Librascope LGP-30, first shipped in September of 1956. **By the way, we'll** be counting Eds, and I'm Ed #1.

Thanks to our  
Sponsors!



AUTOMATTIC

salesforce

Magento®

MailChimp

CISCO DEVNET

SensioLabs  
Creator of  Symfony

Microsoft

# Our Timeline

- Hacker Folklore (1982-83)
- Vacuum Tube Computer Programming (1956)
- Modern Times (2017)
- Video: Warming Up the LGP-30 (6:51 run time)

LGP-30 Schematics: [https://archive.org/details/bitsavers\\_royalPreciatics1959\\_26037699](https://archive.org/details/bitsavers_royalPreciatics1959_26037699)

From the Librascope LGP-30 Schematics, "drawn 4-12-56"

TOLERANCES	FILE	DATE	REVISION
RESISTORS - 1% CAPACITORS - 5% FILAMENT - 10% VOLTAGE - 10%	LGP-30 MATRIX DRIVER	4-12-56	1

3

Computing Past: Mel, The Realtest Programmer of All – #phptek 2017 by Ed Barnard @ewbarnard

Was anyone here NOT programming computers 35 years ago? **We'll** first take a step back 35 years to see the context of our story. **Then** we'll step back 60 years to see a first-generation computer and what it was like to program the thing. **Then** we'll apply this perspective to our own careers today. **Finally**, assuming WiFi works for us, I have a seven-minute video.

## Our Timeline

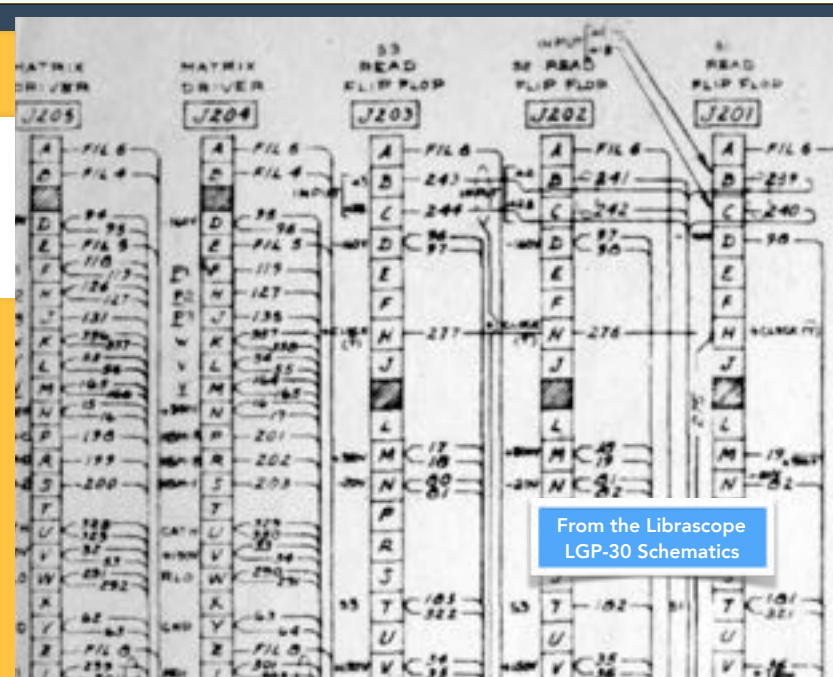


### Hacker Folklore (1982-83)

- Vacuum Tube Computer Programming (1956)
- Modern Times (2017)
- Video: Warming Up the LGP-30 (6:51 run time)

LGP-30 Schematics: [https://archive.org/details/bitsavers\\_royalPreciatics1959\\_26037699](https://archive.org/details/bitsavers_royalPreciatics1959_26037699)

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard



First up: A bit of official hacker folklore.

## Hacker Folklore: *Dune*

*A beginning is the time for taking the most delicate care that the balances are correct. This every sister of the Bene Gesserit knows. To begin your study of the life of Muad'Dib, then, take care that you first place him in his time: born in the 57th year of the Padishah Emperor, Shaddam IV. And take the most special care that you locate Muad'Dib in his place: the planet Arrakis. Do not be deceived by the fact that he was born on Caladan and lived his first fifteen years there. Arrakis, the planet known as Dune, is forever his place.*

—from “Manual of Muad'Dib” by the Princess Irulan



Images scanned from my personal collection

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

5

I'm going to hit you with some obscure cultural references. That's part of hacker culture. Don't worry, this is the only one that takes up a whole slide! So, in talking about hacker folklore, we need to start with the opening words of *Dune* by Frank Herbert. The explanation is important for our story.

## Hacker Folklore (our story)

- Our story begins:
  - Not 60 years ago with vacuum tubes, but
  - A mere 35 years ago with a Best-Selling book
  - *A beginning is the time for taking the most delicate care that the balances are correct*

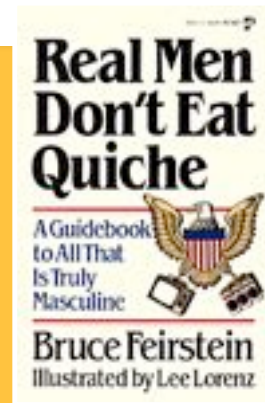


Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

6

In the same way, it's important to recognize that our story begins not 60 years ago with vacuum tubes, but a mere 35 years ago with a best-selling book. Do take the most delicate care to understand the balances as we step back in time.

## Real Men Don't Eat Quiche (1982)



- In 1982, the “battle of the sexes” was raging
- This book poked fun at the macho concept of “Real Men,” contrasting them to men getting more out of life, the “quiche eaters”
- Quiche eater: *A man who is a dilettante, a trend-chaser, an over-anxious conformist, one who eschews (or merely lacks) the traditional masculine virtues*

35 years ago, back in 1982, the “battle of the sexes” was raging. This book was a best-seller all summer. This book poked fun at the macho concept of “Real Men,” contrasting them to men getting more out of life, the “quiche eaters.” Quiche eater: A man who is a dilettante, a trend-chaser, an over-anxious conformist, one who eschews (or merely lacks) the traditional masculine virtues.

## Hacker Folklore: Real Programmers Don't Use Pascal (1983)

- When Ed Post of Tektronix (Ed #2) continued the joke,
- The phrase "Real Men" became "Real Programmers"
- Those who used friendlier programming languages such as Pascal became the "Quiche Eaters"
- Published in *Datamation* magazine, July 1983

Back in the good old days, the "Golden Era" of computers, it was easy to separate the adults from the children, sometimes called "Real Men" and "Quiche Eaters" in the literature...

Ed Post of Tektronix (Ed #2) continued the joke. He changed the phrase from "Real Men" to "Real Programmers" and spun a delightful tale in honor of computer programmers. Those who used friendlier programming languages such as Pascal became the "Quiche Eaters." His essay became a Letter to the Editor of *Datamation* magazine. We passed the magazine around the office, as I'm sure other people did too, and it became part of our hacker folklore.



## University of Washington (1977)



Photo: University of Washington School of Public Health, <http://deohs.washington.edu/seattle-how-get-here-what-do>

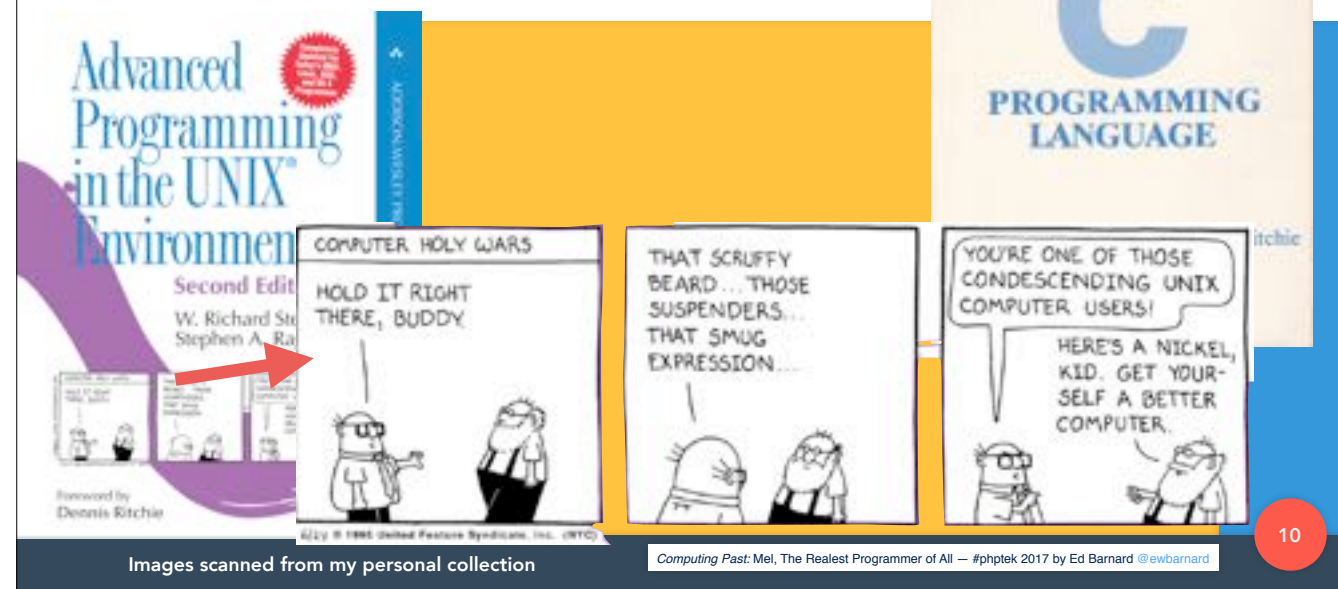
- Department of Engineering: I began with FORTRAN (tested-out), continued with CDC assembly language
- Department of Computer Science: I began with Pascal
- Of the two, it's clearly the Department of Engineering that existed to get the job done

Computing Past: Mel, The Realist Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

9

There really was a distinction between Real Programmers and Quiche Eaters at the time. Here's the University of Washington as an example. I transferred in to the Department of Engineering and later into Computer Science. I already knew FORTRAN, so they let me take the final exam to prove it, and enroll in Assembly Language. The Computer Science department began with Pascal. Of the two, it's clearly the Department of Engineering that existed to get the job done.

## Culture Wars Continue (2005)



The culture wars continue to this day, as you know. UNIX dates from the 1960s, and actual books from the 1970s. The definitive book on UNIX programming, here on the left, came in 2005. There's a Dilbert cartoon on the front. I enlarged it so we can read the story.

## Radio Shack TRS-80 (1977)

- Job fears were real
- This was the first generation of children with access to computers
- Teenagers potentially had equal footing with adults in the job market



Photo: You Tube: <https://i.ytimg.com/vi/wgKWV8C3e7M/maxresdefault.jpg>

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

11

This is the **Radio Shack TRS-80**, which we all called the Trash 80. You can see the **software library** came in a collectible vinyl binder. That's a **floppy-disk drive** on the right.

**Job fears** were real because **this was** the first generation of children with access to computers. **Teenagers** potentially had equal footing with adults in the job market.

## Washington State Data Processing Service Center: A Mainframe Shop (1977)

IBM

Problem and Change Control  
at the State of Washington  
Data Processing Service Center



- IBM came visiting to see how they closely-coupled their dual IBM 370/158 mainframes
- IBM produced a Case Study brochure to honor the occasion



Images scanned from my personal collection

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

12

Centralized computing was a necessity, since IBM and their room-sized mainframes were the major player. You're looking at the Washington State Data Processing Service Center with a pair of IBM 370/158s in the background. This was the computing resource for all of Washington State government.

The system programmers on the right did such an amazing job of closely-coupling their IBM 370 systems together, that IBM itself came to find out how they did it. This is from the advertising brochure that IBM produced as a result. That's the data center director on the left, my Dad.

## CRAY-1 arrives at Daresbury Laboratory, UK (1979)

- During this period, the Real Programmers were the ones who understood computer programming, and the Quiche Eaters were the ones who didn't.
- The Real Programmer is in danger of becoming extinct, being replaced by high-school students playing Pac-Man with TRASH-80s!
- Understanding these differences gives these kids something to aspire to, a role model.



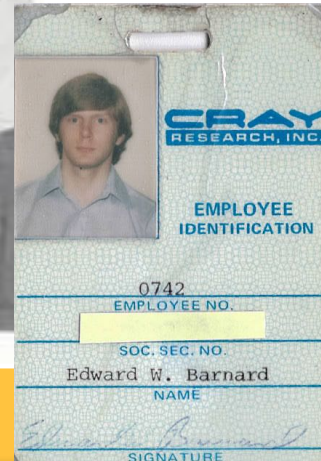
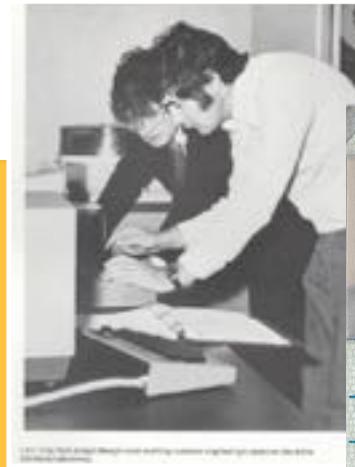
L to r: Phil, Andy Roberts, Director of Daresbury Laboratory; Seymour Cray, founder of Cray Research, Inc., and Dr. Peter Swales, Head of Daresbury's Computer Systems and Electronics Division

Computing Past: Mel, The Realist Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

13

Remember, the fears were real. We had the room-size mainframes and supercomputers on the one hand, and home computers on the other. That is Seymour Cray in the center, the Father of Supercomputing. This is the background as Ed continues the story... **During this period**, the Real Programmers were the ones who understood computer programming, and the Quiche Eaters were the ones who didn't. **The Real** Programmer is in danger of becoming extinct, being replaced by high-school students playing Pac-Man with TRASH-80s! **Understanding** these differences gives these kids something to aspire to, a role model.

## Mostyn Lewis at Daresbury (1979)



- There is a clear need to point out these differences:
- Help employers of Real Programmers realize why it would be a mistake to replace us

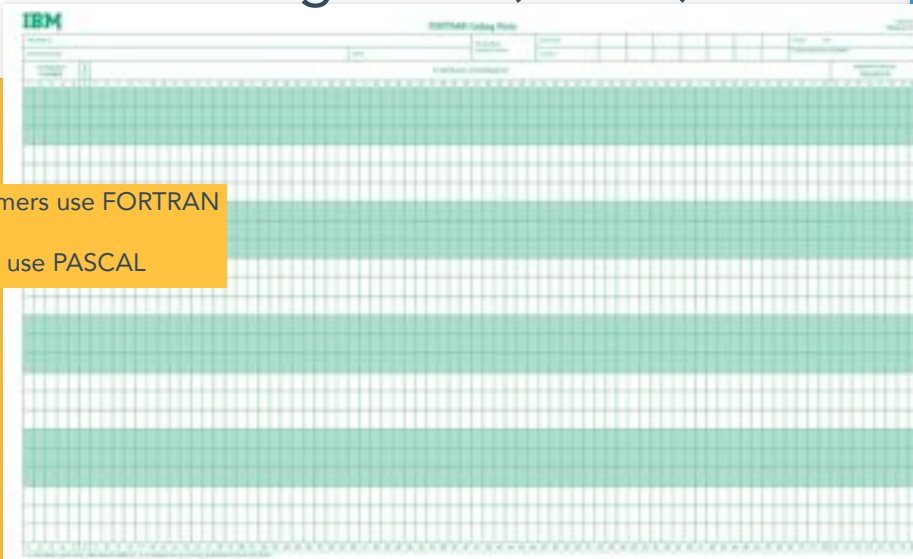
Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

14

Ed continues: There is a clear need to point out these differences: Help employers of Real Programmers to realize why it would be a mistake to replace the Real Programmers on their staff with 12-year-old Pac-Man players (at a considerable salary savings). That's me in the middle. I stayed in contact with Mostyn Lewis, on the far left, for many years.

## FORTRAN Coding Form (1970s)

- Real programmers use FORTRAN
- Quiche Eaters use PASCAL



Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

15

The easiest way to tell a Real Programmer from the crowd is by the programming language he or she uses. **Real Programmers** use FORTRAN. **Quiche Eaters** use PASCAL. (Real Programmers actually talked in capital letters you understand.)

This is a pad of FORTRAN coding forms. You would write your program, in pencil, on the form and send it off for keypunching. Each row becomes one punch card.

## Pascal (1978)

- Niklaus Wirth: How do you pronounce your name?
- By name, "Veert"
- Or by value, "Worth"
- Quiche Eater



**Niklaus Wirth**, the designer of PASCAL, was asked, "How do you pronounce your name?" **He replied**, "You can either call me by name, pronouncing it 'Veert,' or call me by value, 'Worth.'" **One can tell** immediately from this comment that Niklaus Wirth is a Quiche Eater. In other words, Real Programmers are serious, and don't do cute.





## Punch Card (1950s-1970s)

Real Programmers don't need abstract concepts to get their jobs done. They are perfectly happy with a keypunch, a FORTRAN IV compiler, and some pizza.

This is a FORTRAN punch card. **You can see** the one line of code on the left, and the **sequence number**, that is, the line number, in the eight columns on the right. That's why we still try to keep the line length to 72 or 80 characters to this day. Real Programmers don't need abstract concepts to get their jobs done. They are perfectly happy with a keypunch, a FORTRAN IV compiler, and some pizza.

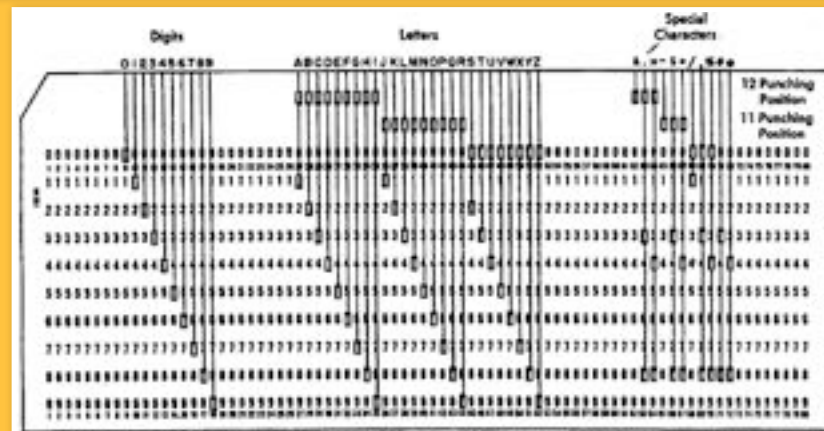


Figure 3. Punching Positions in Card

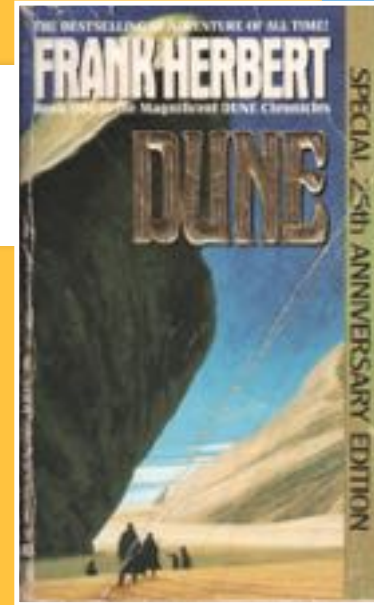
## Punch Card Layout (1949)

*Digits use one hole per character, letters use two, and special characters mostly use three*

**Digits** use one hole per character, **letters** use two, and **special** characters mostly use three. So, remember: If you can't do it in FORTRAN, do it in assembly language. If you can't do it in assembly language, it isn't worth doing.

## *Hacker Folklore: Our Context*

- Fears for our jobs/occupations
- Culture wars
- What it means to be elite, a Real Programmer



Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

19

Here's the context for our story: We had fears for our jobs and occupations. The culture wars continued. We addressed those fears by explaining what it means to be elite, a Real Programmer.

# The Story of Mel

Posted to Usenet by its author, Ed Nather (Ed #3), May 1983

A recent article devoted to the *macho* side of programming  
made the bald and unvarnished statement:  
**Real Programmers write in FORTRAN.**



LIBRASCOPPE'S MURAL ROOM became a study hall for neophyte LFG-30 programmers the week of July 14. Students participating in this first training school for LFG-30 customers included (seated l. to r.) Bill Hopper, Mary Cornell and Chuck Roe, Convair-Pomona; John Corkhill, Convair-San Diego; R. J. Bibbia, Lark Aviation; K. A. Hurst, D. D. Parkhurst, C. S. K. Kishimoto and Ed J. Rasmussen, Convair-San Diego; George Kendrick, Convair-Pomona; Chuck Ray, Caltech; and William Clayton, National Security Agency. Standing (l. to r.) are Fred Flannell, class instructor and assistant sales manager of Royal-McBee; and Royal-McBee Applications Engineers Bud Hazlett, Jack Behr and Mel Kaye. (Photo by DUGGAN)

## Mel Kaye (1956)

- Back in the Good Old Days...
- The term "software" sounded funny
- Real Computers were made out of drums
- Real Programmers wrote in machine code

Maybe they do now, in this decadent era of Lite beer, hand calculators, and "user-friendly" software but back in the Good Old Days, when the term "software" sounded funny and Real Computers were made out of drums and vacuum tubes, Real Programmers wrote in machine code. **Mel is** on the back right.

## Coding Sheet (1956)

WPM-10 CODING SHEET Page 1 of 6

Job No. \_\_\_\_\_ Program No. 11.0K Prepared by Mr. Kaya Date 10/24/56

Problem DATA INPUT #1 SUBROUTINE Track \_\_\_\_\_

Program Input Codes	Location	Instruction Op.	Address	Contents of Address	Notes
<u>0.0.0.0</u>	<u>1</u>				
<u>1.0.0.0</u>	<u>1</u>				
	<u>0.0.0.0</u>	<u>B</u>	<u>0.1.1.5</u>	<u>Addr</u>	
	<u>0.1</u>	<u>C</u>	<u>0.0.4.4</u>		
	<u>0.2</u>	<u>P</u>	<u>0.0.3.8</u>		<u>Input I.D. word</u>
	<u>0.3</u>	<u>I</u>	<u>0.0.0.0</u>		
	<u>0.4</u>	<u>H</u>	<u>6.3.5.4</u>		<u>Orig I.D. word</u>
	<u>0.5</u>	<u>T</u>	<u>0.0.2.2</u>		<u>→ P&gt;7</u>
	<u>0.6</u>	<u>V</u>	<u>0.0.0.7</u>		
	<u>0.7</u>	<u>S</u>	<u>0.0.2.7</u>	<u>1020</u>	
	<u>0.8</u>	<u>T</u>	<u>0.0.0.0</u>	<u>Exit</u>	<u>Out if I.D. word = 0</u>
	<u>0.9</u>	<u>A</u>	<u>0.1.9.5</u>	<u>1020</u>	
	<u>1.0</u>	<u>N</u>	<u>0.1.3.2</u>	<u>1020</u>	<u>Shift left 2</u>

Computing Past: Mel, The Realist Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

Yes, Real Programmers wrote in machine code. Not FORTRAN. Not, even, assembly language. Machine Code. Raw, unadorned, inscrutable hexadecimal numbers. Directly.

# Coding Sheet (1956)

104-30 CODING SHEET Page 2 of 6

Job No. \_\_\_\_\_ Program No. 11.00 Prepared by M. K. L. Date 10/24/56

Problem DATA INPUT #1 SUBROUTINE Track \_\_\_\_\_

Program Input Codes	Location	Instruction Op. Address	Contents of Address	Notes
	0.0.3.2	S.0.2.4.7	107	
	3.3	T.0.2.2.0		→ positive p
	3.4	F.0.2.1.3	xxxxxxxx	here for neg. p. Drop all
	3.5	X.C.6.3.5.7		Complement p.
	3.6	X.S.6.3.5.7		
	3.7	M.0.0.5.9	1014	Shift to 29
	3.8	X.H.6.3.0.5	p.0.2.9	
	3.9	X.B.6.3.2.4	Orig. I.O. word	
	4.0	F.0.2.1.7	xxxxxxxx	Drop all but P102
	4.1	I.0.0.1.4		
	4.2	B.0.1.2.6	exit 2104	here for positive p.

Lest a whole new generation of programmers grow up in ignorance of this glorious past, I feel duty-bound to describe, as best I can through the generation gap, how a Real Programmer wrote code. I'll call him Mel, because that was his name.

# Our Timeline

- Hacker Folklore (1982-83)

## Vacuum Tube Computer Programming (1956)

- Modern Times (2017)
- Video: Warming Up the LGP-30 (6:51 run time)

From the Librascope  
LGP-30 Schematics

LGP-30 Schematics: [https://archive.org/details/bitsavers\\_royalPreciatics1959\\_26037699](https://archive.org/details/bitsavers_royalPreciatics1959_26037699)

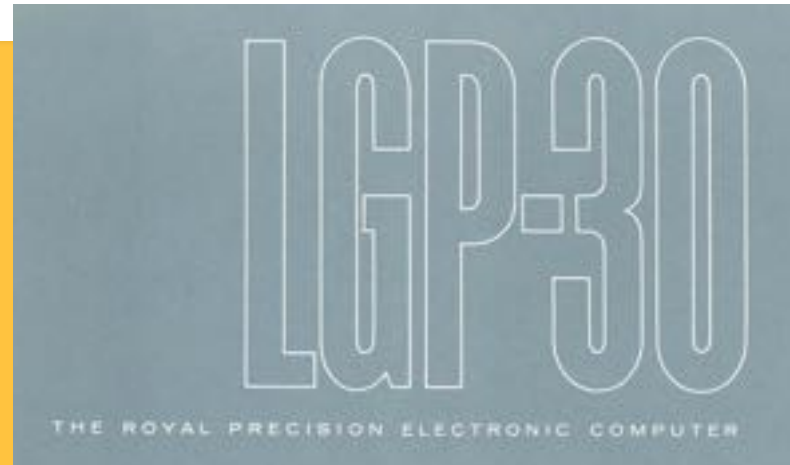
Computing Past: Mel, The Realist Programmer of All – #phptek 2017 by Ed Barnard @ewbarnard

24

Now we can get real.



## LGP-30: Royal Precision Electronic Computer (1955)



<https://www.pinterest.com/mattarpen/back-to-the-future-stuff/>

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

25

The year is 1955, and it's time to introduce the Royal Precision Electronic Computer, the LGP-30.

## LGP-30: Royal Precision Electronic Computer (1955)

- Low Cost
- Small size... Mobile
- Simplified Programming
- Large Memory
- Reliable Performance



[http://archive.computerhistory.org/resources/text/Royal\\_McBee/RPC.LGP-30.1956.102646223.pdf](http://archive.computerhistory.org/resources/text/Royal_McBee/RPC.LGP-30.1956.102646223.pdf)

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

26

It was low cost, with a retail price of \$47,000. It was small, about the size of a desk. It weighed 800 pounds. It was mounted on heavy-duty castors so it could be rolled across the floor. That made it portable. Wait until you find out what "simplified programming" means! The large memory was 4,096 words of spinning drum memory.

## LGP-30 (1956)



Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

It came with a chair, but the guy was probably optional. On the right is the main memory.

## Museum LGP-30 (since ca. 1980)



Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

28

This in the Computing History Museum. The only I/O device was the electric typewriter you see here.

## Coding Sheet (1956)

**B: Bring:** Bring contents of location 0115 to the accumulator

**C: Clear:** Store contents of the accumulator in memory, clearing the accumulator

**P: Print:** Print an electric typewriter symbol.  
The symbol is denoted by the track number part of the address in an instruction word.  
When executing this command, the computer has complete control over the typewriter functions, including decimal digits, letters, punctuation marks, shifts, tabs, carriage return and any other operation the typewriter can perform.  
This permits complete flexibility in the format of the output, including simultaneous punching on tape, if desired

**I: Input:** From the typewriter

**H: Hold:** Store to drum

**T: Test:** If-test jump

**U: Unconditional jump:**  
The "else" part of the "if"

**S: Subtract:** Subtract the contents of location 0029 from the contents of the accumulator, and retain the difference in the accumulator

**N: Multiply Lower:** Multiply the number in the accumulator by the number in location 0132, retaining the least-significant bits in the accum

https://archive.org/details/bitsavers\_royalPreciutimeManualOct60\_3213373

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

Here is a library routine provided by the vendor. Yes, library software was published on hand-written coding forms. That's how it was done with the first generation of commercial computers.

**B is the Bring instruction.** Bring the contents of location 0115 to the accumulator.

**C is Clear.** Store the contents of the accumulator in memory, clearing the accumulator.

**P is Print.** Print an electric typewriter symbol. The symbol is denoted by the track number part of the address in an instruction word. When executing this command, the computer has complete control over the typewriter functions, including decimal digits, letters, punctuation marks, shifts, tabs, carriage return and any other operation the typewriter can perform. This permits complete flexibility in the format of the output, including simultaneous punching on tape if desired.

**I is Input** from the typewriter.

**H is Hold**, that is, store the value to drum memory.

**T is Test**, an if-test jump.

**U is Unconditional** jump. In this case it's the "else" part of the if.

**S is Subtract.** Subtract the contents of location 0029 from the accumulator, and retain the difference in the accumulator.

**N is Multiply Lower.** When you multiply a five-digit number by another five-digit number, you get a ten-digit number. It's the same thing here. When you multiply a 30-bit number by another 30-bit number, the result is up to 60 bits wide. The accumulator is only 31 bits wide (counting the sign bit), so the M instruction keeps the upper half of the result, and the N instruction keeps the lower half of the result. So, here, we multiply the number in the accumulator by the number in location 0132, retaining the least-significant bits in the accumulator.

DATA INPUT #1 SUBROUTINE  
( PROGRAM 11.0)

**FUNCTION:**

To read a decimal number from tape, convert to binary, scale to the proper binal point location, and store the word in a specified drum location. For each number the following is punched on tape:

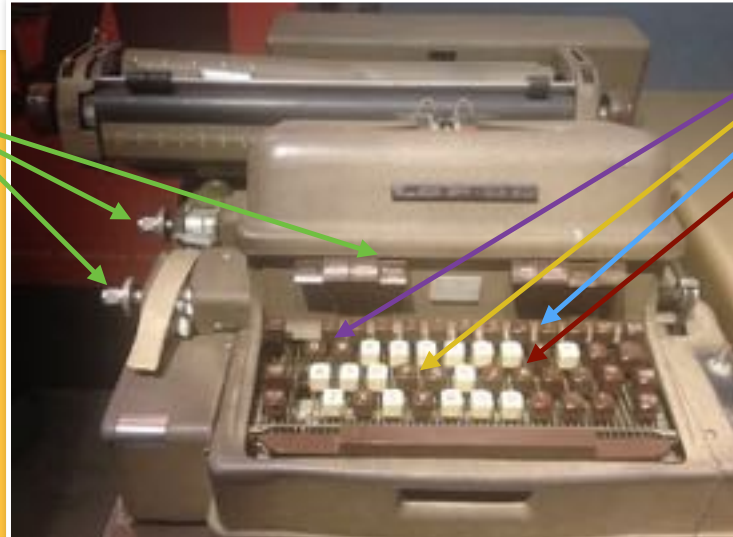
1. The decimal point location of the number on tape, counting from right to left. (One decimal digit designated as "P").
2. The binal point location desired for the number to be placed on drum. (Sign and two decimal digits designated as "q").
3. The drum location to which the number is to be sent. (2 decimal digits for track and 2 decimal digits for sector).
4. The number to be entered (Seven decimal digits plus sign).

B 0.1.1.5	A0030	
C 0.0.4.4		
P 0.0.3.8		{ Input I.D. word
I 0.0.0.0	X	
H 6.3.5.4	Orig. I.D. word	
T 0.0.2.2		→ P27
V 0.0.0.7		
S 0.0.2.7	X 1030	
F . . .	Exit	Out if I.D. word = 0
A 0.1.4.5	1030	
N 0.1.3.2	1027	Shift left 2
E 0.2.3.3	X 05NKNW41	Drop P = from I.D. word
H 6.3.5.5	N <sub>2</sub>	
V 0.0.1.7		
A 0.1.3.6	1024	Shift left 24 bits

Here's the documentation for the library routine from the previous slide. The code on the left is copied from the previous slide. **This is a data input subroutine... You can see the decimal point is designated as P.**

## Flexowriter (1956)

Paper Tape  
Reader/Punch



Hexadecimal Notation is  
0-9, F-G, J-K, Q-W  
(not 0-9, A-F)

"Simplified Programming"  
means  
"type directly to drum memory"  
or  
"type directly to paper tape"

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

31

The **paper tape punch** and reader are on the left. **Use the switches** to turn the punch on or off. Hexadecimal notation was not yet standardized. The numbers were 0-9, **f-g, j-k, q-w**, not a-f. You can see the letter sequence on the keyboard. **There is no one key**. A **lower-case L** was used instead. And, by the way, "simplified Programming" means "type directly to drum memory" or "type directly to paper tape."

# Hexadecimal Digits (1956)

Decimal and Hexadecimal  
Equivalents of Binary Numbers

Binary

0 0 0 0  
0 0 0 1  
0 0 1 0  
0 0 1 1  
0 1 0 0  
0 1 0 1  
0 1 1 0  
0 1 1 1  
1 0 0 0  
1 0 0 1  
1 0 1 0  
1 0 1 1  
1 1 0 0  
1 1 0 1  
1 1 1 0  
1 1 1 1

Hexadecimal

0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
f  
g  
j  
k  
q  
w

Decimal

0  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15

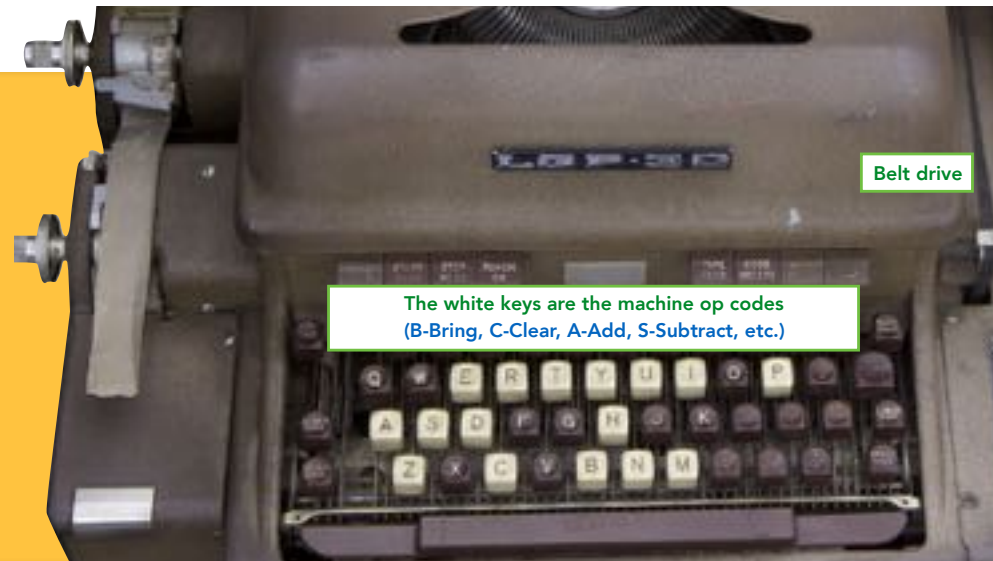
Computing Past: Mel, The Realist Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

32

You thought you knew hex? Think again! The numbers were f, g, j, k, q, w. We'll be seeing that a lot.



## Flexowriter (1956)



Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

33

As you can see, this is quite developer-friendly. The machine instruction codes are the white keys.

# Punching Tapes From Coding Sheets (1956)

1/2-30 ORDIN. SHEET

Job No. \_\_\_\_\_ Program No. 11.28 Prepared by DET

Problem DATA INPUT #1 SUB

Program Input Codes	Location	Instruction	Address
0.0.0.0	1		
0.0.0.0	2		
	0.0.0.0	B.0.0.0	
	0.1	C.0.0.0	
	0.2	D.0.0.0	
	0.3	E.0.0.0	
	0.4	F.0.0.0	
	0.5	G.0.0.0	
	0.6	H.0.0.0	
	0.7	I.0.0.0	
	0.8	J.0.0.0	
	0.9	K.0.0.0	
	1.0	L.0.0.0	
	1.1	M.0.0.0	

1. Only the "Program Input Codes" and "Instruction" columns of the coding sheet are to be punched, with appropriate stops. Never punch "Location," "Contents of Address," or "Notes" columns.

2. Each entry on a line must be followed by a conditional stop code-- "Stop" column, symbol ('). A line left blank must have the stop code punched.

3. Punch the "Program Input Codes" column only when there is an entry in the column. The "Program Input Codes" must be followed by the stop ('). This punching must precede the punching of the "Instruction" column on the same line of the coding sheet.

4. Leading zeros need not be punched. All other zeros must be punched. E.G., 00013086' only 13086' need be punched. ,0000017' must be punched ,0000017'. For T0059' punch T0059'.

5. Consider brackets as containing zeros. E.G., for [...]' : B[00000000]' , only the stop code need be punched. For R[...] : B[0000]' punch B0000'.

Here is how you get your program, that you wrote out on paper, into the computer. **Only the “program input codes” and...**

# Punching Tapes From Coding Sheets (1956)

100-30 CODING SHEET

No. \_\_\_\_\_ Program No. 1000 Prepared by Dr. L. J. Fogel

Item DATA INPUT #1 SUBROUTINE

Program Input Codes	Location in	Instruction Op. Address	Contents Address
0.0.0	1		
0.0.0	2		
	0.0.0.0	B. 0.1.1.5	ADD 20
	0.1	C. 0.0.0.4	
	0.2	A. 0.0.0.8	
	0.3	E. 0.0.0.0	01
	0.4	E. 0.0.0.4	Orig. I.D.
	0.5	T. 0.0.0.2	
	0.6	H. 0.0.0.7	
	0.7	S. 0.0.0.3	0100
	0.8	T. 0.0.0.7	Exit
	0.9	A. 0.1.5.5	1000
	1.0	A. 0.1.5.2	1000
	1.1	E. 0.0.0.3	0100000001 Dep. P + from I.D. card

6. All punching may be done in lower case. B0627' will appear as b0627'.
7. The placing of carriage returns is left to the discretion of the person preparing the tape. Carriage returns do not affect the input operation. We have arbitrarily placed a carriage return (␣) after every 4 words on each coding sheet.
8. A heading may precede a punched program to identify the tape. Anything except a stop code may be punched as a header. Then as the tape is fed through the input reader the heading will print but will not affect the operation of the computer.
9. Each tape should be verified after punching. This can be done by placing the punched tape in the reader and "listing" the tape by the following process.
- a. Make sure "Cond. Stop" button on reader is up.
  - b. Depress "Start Read" button.
  - c. When printing stops, depress space bar.
  - d. Repeat steps b and c until entire tape is printed.

Then the printing may be visually checked against the coding sheets for correctness and presence of stop codes.

## LGP-30 Instruction Set (1956)

THE COMMAND TABLE	
code letter of command	command
B	Bring
A	Add
S	Subtract
M	Multiply, fractional
N	Multiply, integral
D	Divide
H	Hold

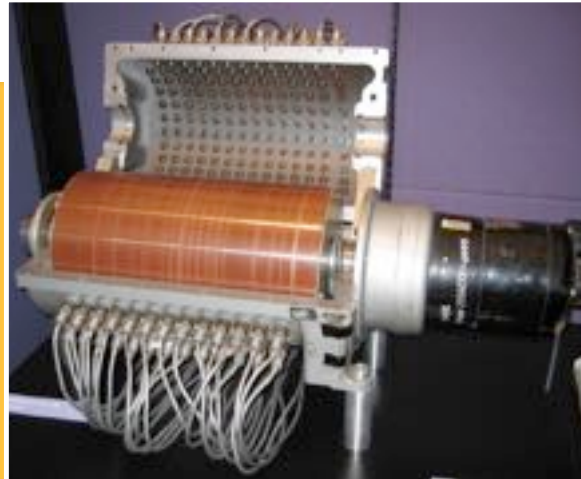
C	Clear
Y	Store address
R	Return address
E	Extract
U	Transfer control
T	Test
I	Input
P	Print
Z	Stop

- 16 hardware instructions
- Stored in drum memory

The machine had 16 hardware instructions. All instructions were stored in memory. This was actually a big deal, and the sales brochures called this a stored-memory computer.

Ed continues the story: I first met Mel when I went to work for the Royal McBee Computer Corp., a now-defunct subsidiary of the Royal typewriter company. The firm manufactured the LGP-30, a small, cheap (by the standards of the day) drum-memory computer, and had just started to manufacture the RPC-4000, a much-improved, bigger, better, faster--drum-memory computer.

## Magnetic Drum Storage (1951)



- Engineering Research Associates (ERA) of St. Paul, Minnesota
- Elite navy code breakers in World War II
- Navy Lt. Bill Norris chosen CEO
- Founders in 1957 formed Control Data Corporation (CDC)
- Seymour Cray became CDC chief designer

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

37

This is drum memory. The read heads are along the outside, and obviously don't move. It's the drum that moves, bringing its data to the read head, not the other way around. This example was built in St. Paul by a company that could have become Silicon Valley but didn't. At the end of World War II, an elite group of Navy code breakers created a company whose top-secret work helped to launch the world's computer industry. The company was called Engineering Research Associates, and very few people knew its secrets. Most still don't. One of those code breakers was Bill Norris, a Navy Lieutenant. He became CEO. After the company was bought out, Bill Norris and the other founders created Control Data. Seymour Cray joined a year later.

## Magnetic-Core Memory (1955-1975)



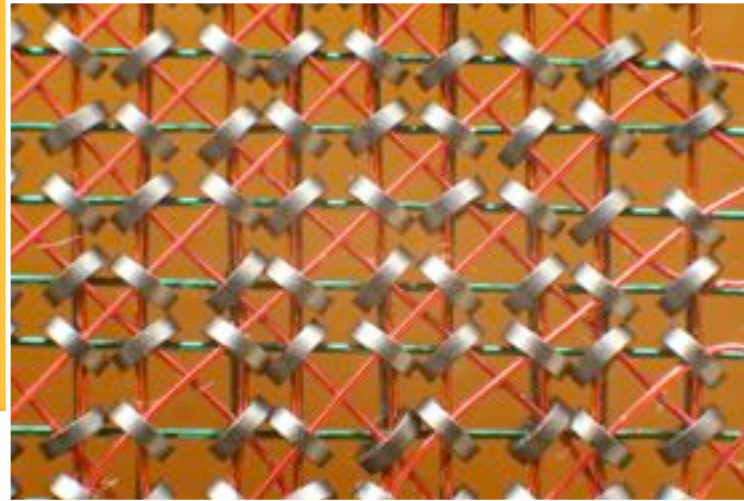
"Cores cost too much, and aren't here to stay, anyway."  
That's why you haven't heard of the company, or the computer.

38

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

This is a full rack of core memory. There is a coin in front for reference. The coin is about the size of a quarter. Ed continued, "Cores cost too much, and aren't here to stay, anyway." That's why you haven't heard of the company, or the computer.

- Core memory:
  - Quite expensive during Mel's time
  - Hand-manufactured by skilled workers
- Drum memory was simpler to manufacture
  - Therefore cheaper
  - But *much* slower than Core Memory



## Magnetic-Core Memory (1955-1975)

Core memory was quite expensive during Mel's time because it had to be hand-manufactured by skilled workers. Drum memory was simpler to manufacture, and therefore cheaper. But it was much slower than core memory.



## RPC 4000 Electronic Computer System (1960)

I had been hired to write a FORTRAN compiler for this new marvel and Mel was my guide to its wonders. Mel didn't approve of compilers.



Computing Past: Mel, The Realist Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

40

This looks to me like a Star Wars droid unit. In fact it looks like a droid that just survived a battle and hasn't been cleaned up yet. **The Librascope logo** is on the left, and then **Control Data** slapped their own name on the front. That probably means that Control Data bought a used droid, refurbished it, and sold it for \$10,000 or so, which was a huge discount. Ed continues: I had been hired to write a FORTRAN compiler for this new marvel and Mel was my guide to its wonders. Mel didn't approve of compilers.



INSTRUCTION WORD

command | --- track --- | --- sector --- |

address

"If a program can't rewrite its own code," he asked, "what good is it?"

The LGP-30 gives the programmer a unique interlaced pattern of word addresses which greatly simplify the reduction of memory access time. The stored program operation of the computer makes it possible for a program to be self-modifying, thus increasing flexibility still further.

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

41

"If a program can't rewrite its own code," he asked, "what good is it?" At the time, programs were supposed to be self-modifying. **Here's what the sales brochure says:** The stored program operation of the computer makes it possible for a program to be self-modifying, thus increasing flexibility still further.

## Most Popular Program

- Hexadecimal
- Most popular program the company owned
- Played blackjack with potential customers at computer shows

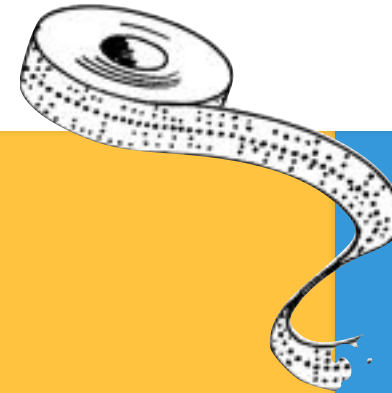


Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

42

Mel had written, in hexadecimal, the most popular program the company owned. It ran on the LGP-30 and played blackjack with potential customers at computer shows.

# Blackjack



Opening instructions of the LGP-30  
Blackjack Program as typed by Mel  
(Recovered from paper tape)

Note: Hexadecimal digits are  
0-9, f-g, j-k, q-w

```
v0402k00'k2w8j'k3278'f31j4'  
q2k98'22k78'q2qwj'22k54'22k58'
```

[ftp://ftp.informatik.uni-stuttgart.de/pub/cm/lgp30/papertapes/Games/bkjc.ktx](http://ftp.informatik.uni-stuttgart.de/pub/cm/lgp30/papertapes/Games/bkjc.ktx)

Here's what the beginning of the blackjack program looks like. Remember, hexadecimal notation was different then. Those are not the digits you seek.

## More Blackjack

v042k00'

k2w8j'k3278'f31j4'q2k98'22k78'q2qw'22k54'22k58'  
22k6j'332w0'f32k8'931w0'w31q0'q2k24'q2q8j'j2k94'  
q2k0j'22k68'22k70'22k50'12k18'92k4q'w2k4q'g2k68'  
f2q68'12k0k'12k18'q2k4q'j2k8f'k2k88'12k14'k2k8j'  
33300'f3300'56f40656'q06627q'f3200'11w20806'  
56140656'561j0656'56140656'56130656'56340656'56440656'564j0656'  
280j0406'10640806'10750806'1076j0806'q06f77q'q06f27q'q06f77q'q06f67q'  
1000000'800000'10000'8000'800000004'q7q7q7q7'7wwwwwwj'400000'

j36w5j4j'  
v0402q00'  
12k94'w13wj'g2q24'w329j'g2q1j'12wj4'f2q28'12k94'  
f2q28'12k0w'f2wq0'12q78'22q38'22q40'12k90'92k4w'  
k2k8j'g2q8j'12q38'22q40'w2k64'q2k68'k2w8j'  
k2w8j'f2q68'332w0'f33j'j2q'f2k0j'12k9j'131k4'  
w2w8j'800q2q2j'81000'4'81800'32q28'f2q68'k33q4'  
80j00'58'81800'32q28'f2q68'k3274'33300'f3300'  
20187q7q'81800'1327j'q2q8j'k327j'f2qk8'32q28'f2q68'  
q2w2j'q33q4'j33q4'92kw8'w2qf4'g2w60'80j00'10'  
2216w45q'  
v0402w00'  
81800'12w5j'q2q8j'k2w5j'32q28'f2q68'g2w3j'q3274'  
j3274'q3068'f2w08'133q4'92kw8'q2k0w'f2q88'13274'  
92kw8'q2k0w'f2w0'13274'92kw8'q2k0w'f2w88'  
33300'f3300'20106f00'712f12f0'2608067q'80000'40000'92qwj'  
w2qwj'q2w90'f2qj4'2k9j'81800'30'133q4'g2wf4'  
f2wj0'92kw8'j33q4'w2w94'g2w8'f2wj0'q2qf4'k33q4'  
80j00'28'81800'12w5j'w2q8j'k2w5j'32q28'f2q68'  
q2w4j'q3274'j3274'92kw8'w2qf4'g3004'80j00'133q4'  
3f4w87wE'  
v0403000'  
f3074'q2qwj'q3010'f2w08'q2q8j'g303j'332w0'f32k8'  
92w94'w2w94'g2w88'81000'14'81800'f2w34'q3030'  
q304j'f302j'211j589f'q2qwj'q302j'13274'g3060'f302j'  
q2w4'f2wq8'q2w4'k3274'f2w08'92kw8'j33q4'w2qf4'  
q30f0'33300'f3300'2030055f'7f5f77q7'k33q4'k33q4'f3130'  
q2q8j'q30k4'12q8j'w327j'q30k4'33300'f3300'203000f0'  
726f6j64'726f6j7'12q78'k2w8j'f30w4'1327j'w2qwj'g3110'  
33300'f3300'20302j06'f6721f2f'7f7q7q7q'12qf4'k33q4'13278'  
07q49gk1j'

v0430100'  
q2q8j'k3278'f3130'100'33300'f3300'20305f46'5f720j06'  
q067q7q'13334'33298'f3280'13274'92k98'k3274'w2qf4'  
q3204'33330'f3300'30180f52'75f8106q'60606q'k3274'k3274'  
13274'w334'q3268'13278'w2q8j'k3278'33300'f3300'  
307f6e46'1f4f6010'1608067q'13278'g31kj'j2k94'33300'f3300'  
10340807'k2l94'33298'f3280'33300'f3300'2q04047q'13278'  
q31w8'81000'k327j'k327j'k2w5j'68'f32w4'80j00'  
j'k3278'w3278'80700'3j'f3194'80j00'8'  
19j56j2'  
v0430200'  
f31j4'q2q8j'g3240'12q8j'w2w5j'g3240'33300'f3300'  
30180f0j'726f66j4'726f66j7q'80j00'13324'w2qf4'g33w4'f33q8'  
33300'f3300'30185f46'5f720j06'q067q7q'13274'3298f'f3280'  
80j00'f3160'13278'q2q8j'f3174'50'wwwqw8q'  
w2w4j'g32f0'w2w4j'g32qj'w2w4j'g32j4'80j00'24'  
q2w5j4'6310j'g32q4'232q0'8f100'200'f31g0'80600'  
f329j'80f00'f329j'12w8j'q328j'k2w8j'13048'63048'  
j3048'72q64'q3048'j3048'f2k2j'400'81000'f2q78'  
w26k030'  
v0430300'  
131qj'933q0'f330j'w339j'g3350'f3344'7q000000'47q0000'  
4000'3w00'80068'j'4'4'13300'q33j3j'  
f33f8'133j3'q3300'f33q0'g3318'1331j'73320'93324'  
q3328'k33qj'f3338'g3318'1331j'13320'93324'g3328'  
k33qj'w3330'q3334'g3300'f33qj'133j'10'7q000000'  
13300'f33q0'23300'f33q8'7q7q7q7q'4j'1332j'80268'  
j3334'1331j'f33k0'4'63398'w339j'g336j'f339j'  
q33q2'f31qj'13278'w2q8j'k3278'12q78'k2w8j'f316j'  
00q0j242'

Address	Disassembly	Comment
00003400	CALL 00003400	
00003401	CALL 00003400	
00003402	CALL 00003400	
00003403	CALL 00003400	
00003404	CALL 00003400	
00003405	CALL 00003400	
00003406	CALL 00003400	
00003407	CALL 00003400	
00003408	CALL 00003400	
00003409	CALL 00003400	
0000340A	CALL 00003400	
0000340B	CALL 00003400	
0000340C	CALL 00003400	
0000340D	CALL 00003400	
0000340E	CALL 00003400	
0000340F	CALL 00003400	
00003410	CALL 00003400	
00003411	CALL 00003400	
00003412	CALL 00003400	
00003413	CALL 00003400	
00003414	CALL 00003400	
00003415	CALL 00003400	
00003416	CALL 00003400	
00003417	CALL 00003400	
00003418	CALL 00003400	
00003419	CALL 00003400	
0000341A	CALL 00003400	
0000341B	CALL 00003400	
0000341C	CALL 00003400	
0000341D	CALL 00003400	
0000341E	CALL 00003400	
0000341F	CALL 00003400	
00003420	CALL 00003400	
00003421	CALL 00003400	
00003422	CALL 00003400	
00003423	CALL 00003400	
00003424	CALL 00003400	
00003425	CALL 00003400	
00003426	CALL 00003400	
00003427	CALL 00003400	
00003428	CALL 00003400	
00003429	CALL 00003400	
0000342A	CALL 00003400	
0000342B	CALL 00003400	
0000342C	CALL 00003400	
0000342D	CALL 00003400	
0000342E	CALL 00003400	
0000342F	CALL 00003400	
00003430	CALL 00003400	
00003431	CALL 00003400	
00003432	CALL 00003400	
00003433	CALL 00003400	
00003434	CALL 00003400	
00003435	CALL 00003400	
00003436	CALL 00003400	
00003437	CALL 00003400	
00003438	CALL 00003400	
00003439	CALL 00003400	
0000343A	CALL 00003400	
0000343B	CALL 00003400	
0000343C	CALL 00003400	
0000343D	CALL 00003400	
0000343E	CALL 00003400	
0000343F	CALL 00003400	
00003440	CALL 00003400	
00003441	CALL 00003400	
00003442	CALL 00003400	
00003443	CALL 00003400	
00003444	CALL 00003400	
00003445	CALL 00003400	
00003446	CALL 00003400	
00003447	CALL 00003400	
00003448	CALL 00003400	
00003449	CALL 00003400	
0000344A	CALL 00003400	
0000344B	CALL 00003400	
0000344C	CALL 00003400	
0000344D	CALL 00003400	
0000344E	CALL 00003400	
0000344F	CALL 00003400	
00003450	CALL 00003400	
00003451	CALL 00003400	
00003452	CALL 00003400	
00003453	CALL 00003400	
00003454	CALL 00003400	
00003455	CALL 00003400	
00003456	CALL 00003400	
00003457	CALL 00003400	
00003458	CALL 00003400	
00003459	CALL 00003400	
0000345A	CALL 00003400	
0000345B	CALL 00003400	
0000345C	CALL 00003400	
0000345D	CALL 00003400	
0000345E	CALL 00003400	
0000345F	CALL 00003400	
00003460		

Here's more of the blackjack program. Move along, there's nothing to see here.

## Playing Blackjack



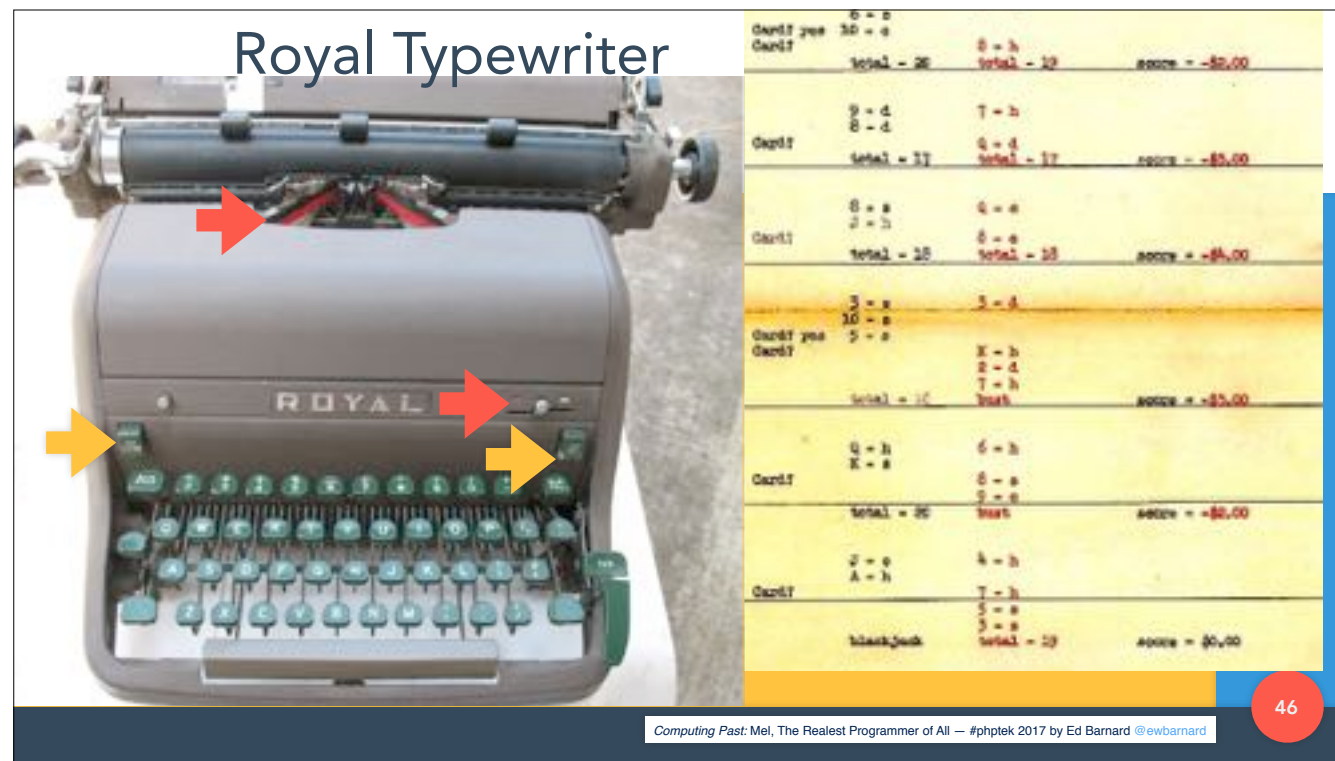
Card?	2 - s	A - s	
	10 - d		
Card? yes	5 - s		
Card?		2 - c	
		6 - c	
	total - 17	total - 19	score = -\$4.00
	5 - d	4 - c	
Card? yes	5 - c		
Card?	K - d	3 - s	
		J - c	
	total - 20	total - 17	score = -\$3.00
	4 - h	A - c	
Card? yes	6 - s		
Card?	10 - c	8 - h	
	total - 20	total - 19	score = -\$2.00

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

45

The blackjack program's effect was always dramatic. The LGP-30 booth was packed at every show, while the IBM salesmen stood around talking to each other. Whether or not this actually sold computers was a question we never discussed.

## Royal Typewriter

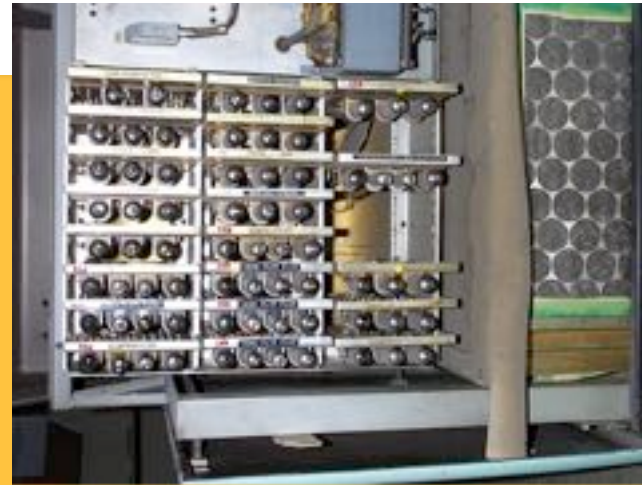


Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

46

This is a standard Royal typewriter from the same time period. **Notice the fabric ribbon has both black and red.** That's why there's both **red and black typing on the output.** The typist manually **moves the selector** from black to red and back. Also notice the keys to **set and clear the hardware tabs.** That's how the text on the right is **lined up into columns.** Also, if you ever get in an argument about "spaces versus tabs," do remember that THAT battle was decided more than a century ago. Business professionals used tabs and tab stops.

## LGP-30 CPU Vacuum Tubes (1956)



Mel's job was to re-write the blackjack program for the RPC-4000.  
(Port? What does that even mean?)

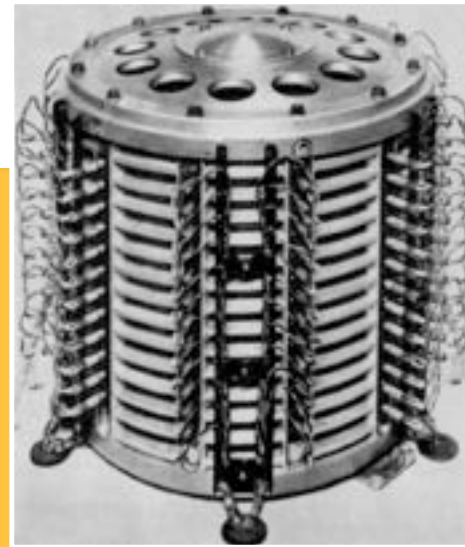
Debit	B = \$ J = \$	C = \$ S = \$	
	total = 50	total = 18	assets = -88,00
Debit plus Credit	J = \$ S = \$	S = \$ E = \$	
	total = 32	total = 25,00	assets = -25,00
Debit	C = \$ E = \$	S = \$ J = \$	
	total = 80	total = 80,00	assets = -80,00
Debit	J = \$ A = \$	C = \$ S = \$	
liability	total = 12	total = 30,00	assets = 30,00

- One-plus-one addressing
- Each machine instruction contains:
  - Operation code (e.g., *Add*)
  - Address of operand
  - Address of next instruction to execute
- This means every single instruction was followed by a GO TO!
- Try explaining *that* to a PASCAL programmer



The new computer had a one-plus-one addressing scheme, in which each machine instruction, in addition to the operation code and the address of the needed operand, had a second address that indicated where, on the revolving drum, the next instruction was located. In modern parlance, every single instruction was followed by a GO TO! Try explaining that to a Pascal programmer.





## Drum Is Not RAM!

- 1930s technology (invented 1932 in Austria)
- Unlike RAM (Random Access Memory), you can't simply read from any location at any time
- Unlike a spinning disc, read/write heads do not move
- The CPU loads your instructions directly from the spinning drum

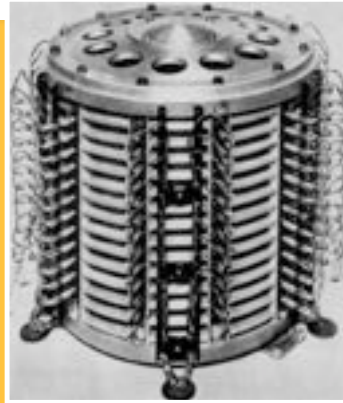
[https://en.wikipedia.org/wiki/Drum\\_memory](https://en.wikipedia.org/wiki/Drum_memory)

Computing Past: Mel, The Realist Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

49

I hope you're getting the idea that drum memory was quite different. **It was** 1930s technology, invented in Austria in 1932. **Unlike** RAM (which is Random Access Memory), you can't simply read from any location at any time. **Unlike** a spinning disc, the read/write heads do not move. **The CPU** loads your instructions directly from the spinning drum.

## Read Out-Of-Sequence (1)



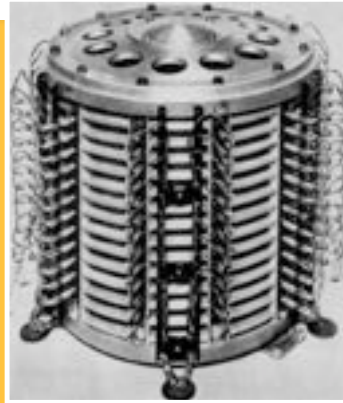
- RPC-4000 drum speed is 3600 RPM (or 17 ms per revolution)
- CPU loads instruction from drum once it is under read head
- CPU then loads operand from drum (as specified in the instruction)
- CPU then executes the instruction, which takes about the same time as either of the above two steps

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

50

The RPC-4000 drum speed is 3600 RPM (or 17 ms per revolution). The CPU loads each instruction from the drum once it is under the read head. The CPU then loads that instruction's operand from drum. The CPU then executes the instruction, which takes about the same time as either of the above two steps.

## Read Out-Of-Sequence (2)

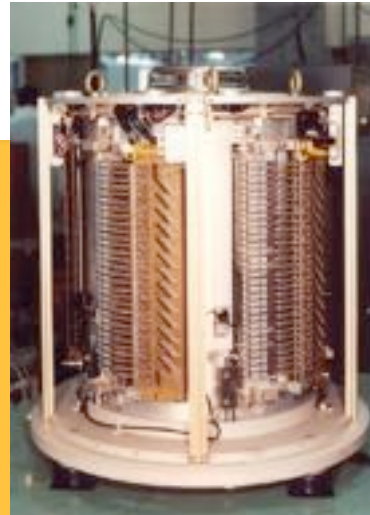


- If instructions and data were sequential, we'd be missing the "window of opportunity" to read in the next item
- No such things as instruction or data buffers
- Drum is hard-wired to the CPU
- Rather than "miss revolutions" when trying to read data, we spread things around the drum based on revolution time

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

51

If the instructions and data were a sequential stream on the drum, they would go by too fast for the CPU to handle. We'd be missing the "window of opportunity" to read in the next item, because that next item would have already spun past the read head. There were no such things as instruction buffers or data buffers. That drum is hard-wired to the CPU. So, rather than missing revolutions when trying to read data, we spread things around the drum based on the drum's revolution time.



## Read Out-Of-Sequence (3)

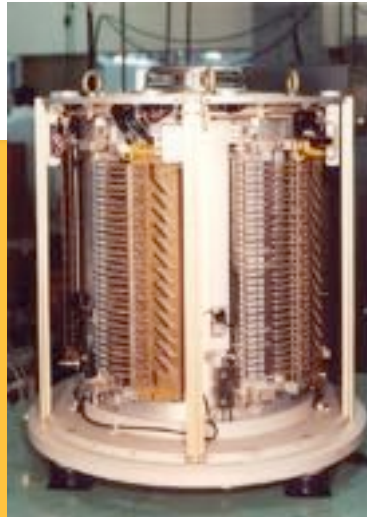
- Example: Read instruction from Track 0, Sector 0
- By time we extract the data address from that instruction, the read heads are coming up on Sector 3 of all tracks (all tracks have read heads)
- Therefore Mel (who knows the timing) might specify Track 1, Sector 3 as the data address

[http://museum.ipsj.or.jp/computer/device/magnetic\\_drum/images/0006\\_02\\_1.jpg](http://museum.ipsj.or.jp/computer/device/magnetic_drum/images/0006_02_1.jpg)

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

52

Let's take an example to see how this works. **For example**, let's read the instruction from Track 0, Sector 0. **By the time** we extract the data address from that instruction, the drum has kept spinning along. The read heads are coming up on Sector 3 of all tracks. Remember, all tracks have read heads, so we're coming up on Sector 3 of any track. **Therefore Mel**, who knows the timing, might specify Track 1, Sector 3 as the data address.



## Read Out-Of-Sequence (4)

- By time the CPU finishes executing the instruction, drum sector 12 (let us suppose) is arriving at the read heads
- This is sector 12 of *any* track, because *every* track has its own fixed, unmoving, read head
- Mel, knowing this, would have set the “next instruction” address to Track 0, Sector 12

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

53

**By time** the CPU finishes executing the instruction, drum sector 12 (let us suppose) is arriving at the read heads. **This is** sector 12 of any track, because every track has its own fixed, unmoving, read head. **Mel**, knowing this, would have set the “next instruction” address to Track 0, Sector 12.

## That's Why We Can't Have Nice Things!

- This is why we can't have nice normal sequential programs with drum memory
- The drum is like getting on a ski lift that never stops: You time it right, or you splash and wait to try again



Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

54

**This** is why we can't have nice normal sequential programs with drum memory. **The drum** is like getting on a ski lift that never stops: You time it right, or you splash and wait to try again.

## (New) Blackjack

# RPC-4000

Program W1-01.0



BLACKJACK GAME  
(By Mel Kaye of Librascope Inc.)

This program is designed to simulate a game of Blackjack between one player (the machine operator) and a dealer (the computer). This write-up is intended to provide the player with the information necessary to play the game.

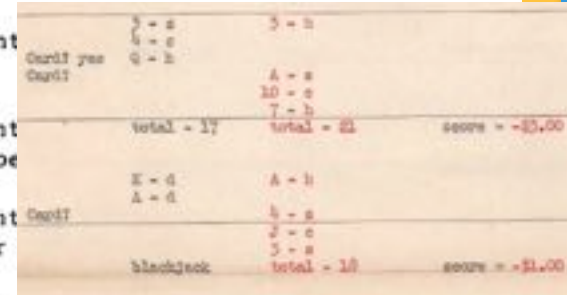
# Blackjack (1960)



BLACKJACK GAME  
(By Mel Kaye of Librascope Inc.)

Before playing Blackjack, set 4 typewriter tab stops to provide for 4 columns of printing. The following is the suggested column lengths (left to right) and their content.

1. 15 spaces. This column will contain the player's responses.
2. 12 spaces. This column will contain their numeric total or an alphabetic response.
3. 12 spaces. This column will contain the cards and their numeric total or a message.
4. 12 spaces. This column will contain the score at the end of the hand.

A screenshot of a typewritten Blackjack game printout. It shows two hands. The first hand has cards 5, 4, and Q, with a total of 19. The second hand has cards K, A, 4, 2, and 5, with a total of 22, which is a blackjack. The scores are calculated as -25.00 for the first hand and -21.00 for the second hand. The printout is formatted with tabs to align the cards, totals, and scores.

5 - 5	5 - 5	
4 - 4		
Q - 10	A - 1	
	10 - 10	
	7 - 7	
total = 19	total = 22	score = -25.00
K - 10	A - 1	
A - 1	4 - 4	
	2 - 2	
	5 - 5	
blackjack	total = 22	score = -21.00



# Blackjack User Interface (1960)



BLACKJACK GAME  
(By Mel Kaye of Librascope Inc.)

The standard RPC-4000 bootstrap loading procedure will load the hexadecimal program tape (no check sum) into locations 00000 through 00928. The program begins at location 00000. After the tape is input, the program selects the typewriter for input and output and then prints "How much do you bet?" Type the amount of your bet in pennies and depress the stop code. e.g., 150\* bets \$1.50.

Next, the program prints "Shuffling". Depress SENSE SWITCH 1 to terminate the shuffling procedure. The program then prints "Cut" and simulates cutting the deck until SENSE SWITCH 1 is raised. The program then deals cards and the game proceeds.

# Blackjack User Interface (1960)



BLACKJACK GAME  
(By Mel Kaye of Librascope Inc.)

All questions from the program must be answered on the typewriter keyboard, and must be followed by depressing the stop code (\*) key. Permissible affirmative answers are: yes\*, ok\*, si\*, ja\*, oui\*. Permissible negative answers are: no\*, non\*, nein\*, nope\*, \* (only the stop code).

If the player's first two cards total 11, the program prints "Press?" An affirmative answer will cause the program to deal the player one card only, and to double the amount of bet for this hand. Any non-affirmative answer to Press causes the program to proceed normally and ask whether the player wants a card.

# Blackjack Program's Rules of Play (1960)

## PLAYING CONVENTIONS

1. Player's Blackjack pays double.
- 
- 
6. The amount of bet will be SWITCH 8 is depressed. And
7. Blackjack by either dealer or player will cause the hand to end and the new score to be printed in column 4.
8. If SENSE SWITCH 32 is depressed, there is a better than normal chance of an ace being dealt as the player's first card.

***Cheat Mode:  
Remember this, it's part of the story!***



Like any game, the blackjack program had its own rules of play. Take a look at the final rule: If SENSE SWITCH 32 is depressed, there is a better than normal chance of an ace being dealt as the player's first card. We'll be coming back to this.

## LGP-30 (1956)



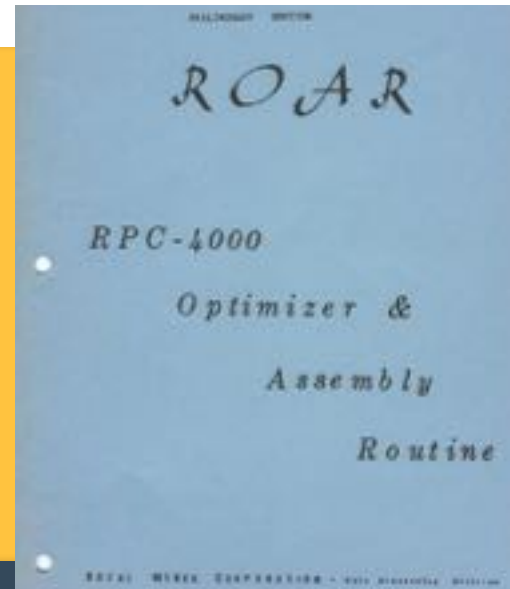
- On RPC-4000, Mel could optimize his code:
- Locate instructions on the drum
- As one instruction finished,
- Next just arriving at read head

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

60

This is the **drum memory** on the far left. **The CPU** is adjacent, as close as possible. Here is the **oscilloscope** tube. You can see the electric typewriter, the **Flexowriter**. Over on the right is optional equipment, the **high speed tape punch**. Mel loved the RPC-4000 because he could optimize his code: that is, locate instructions on the drum so that just as one finished its job, the next would be just arriving at the “read head” and available for immediate execution.

# ROAR: Royal Optimizer & Assembly Routine



There was a program to do that job, an "optimizing assembler," but Mel refused to use it.

"You never know where it's going to put things," he explained, "so you'd have to use separate constants." It was a long time before I understood that remark.

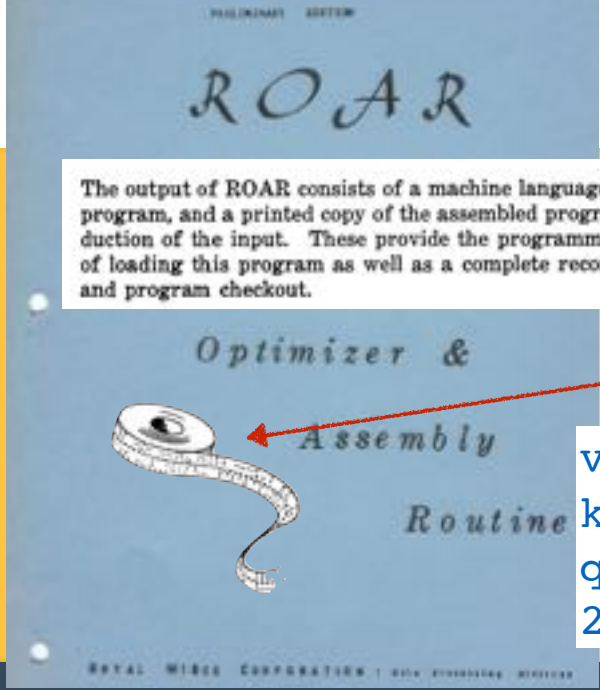
Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

61

There was a program to do that job, an "optimizing assembler" called ROAR, but Mel refused to use it.

"**You never** know where it's going to put things," he explained, "so you'd have to use separate constants." It was a long time before I understood that remark.

## ROAR Output



The output of ROAR consists of a machine language tape of the assembled program, and a printed copy of the assembled program along with a reproduction of the input. These provide the programmer with a ready means of loading this program as well as a complete record for error correction and program checkout.

ROAR output is a hole-punched paper tape and machine-language printout. That is how you watch for errors and do program checkout.

```
v0402k00'k2w8j'  
k3278'f31j4'  
q2k98'22k78'q2qwj'  
22k54'22k58'
```

ROYAL WISE CORPORATION • 1000 BROADWAY • NEW YORK

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

62

ROAR output is a hole-punched paper tape and machine-language printout. THAT is how you watch for errors and do program checkout. Getting your program to run was a slow process.

## RPC-4000 Instruction Set (1960)

ORDER SYMBOL	ORDER NUMBER	D ADDRESS		OPERATION	EFFECT ON		
		TRACK	SECTOR		U	L	X
HLT	00	000	Any	HALT	N.U.	N.U.	Index
SNS	00	≠000	Any	SENSE No operation. Turn the Branch Control on if a track bit (or more) corresponds to a depressed sense switch on the console. The track 64 bit will always turn the Branch Control on.	N.U.	N.U.	Index
CXE	01	A	B	COMPARE X EQUAL 1. Turn the BC off. 2. Compare the bits of the D address with the corresponding bits of the X register. 3. If equal, turn BC on.	N.U.	N.U.	D addr. compared Indexing is redundant
RAU	02	A	B	RESET - ADD UPPER Replace the contents of U with the contents of memory location A B	C(AB)→U	N.U.	Index
RAL	03	A	B	RESET - ADD LOWER Replace the contents of L with the contents of memory location AB	N.U.	C(AB)→L	Index

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

63

Since Mel knew the numerical value of every operation code, and assigned his own drum addresses, every instruction he wrote could also be considered a numerical constant.

## RPC-4000 Instruction Set (1960)

ORDER SYMBOL	ORDER NUMBER	D ADDRESS TRACE SECTOR		OPERATION	U	L	X
STU	24	A	B	STORE UPPER Replace the contents of memory location AB with the contents of the upper accumulator; leave the upper unchanged.	Stored, unaltered	N.U.	Index
STL	25	A	B	STORE LOWER Replace the contents of memory location AB with the contents of the lower; leave the lower unchanged.	N.U.	Stored unaltered	Index
CUU	26	A	B	CLEAR UPPER Replace the contents of memory location AB with the contents of the upper accumulator, then clear the upper to zeroes.	$C(U) \rightarrow C(AB)$ $zero \rightarrow C(U)$	N.U.	Index
CLL	27	A	B	CLEAR LOWER Replace the contents of memory location AB with the contents of the lower, then set the lower to zeroes	N.U.	$C(L) \rightarrow C(AB)$ $zero \rightarrow C(L)$	Index
ADU	28	A	B	ADD UPPER Add algebraically the contents of memory location AB to the contents of the upper, leaving the sum in the upper. An overflow will turn BC on.	$C(U) + C(AB) \rightarrow C(U)$	N.U.	Index
ADL	29	A	B	ADD LOWER Add algebraically the contents of memory location AB to the contents of the lower, leaving the sum in the lower. An overflow will turn BC on.	N.U.	$C(L) + C(AB) \rightarrow C(L)$	Index

Computing Past: Mel, The Realist Programmer of All — #phpteck 2017 by Ed Barnard @ewbarnard

64

He could pick up an earlier “add” instruction, say, and multiply by it, if it had the right numeric value. His code was not easy for someone else to modify.






## Hand Optimized

- Mel's always ran faster
- "Top-down" design not invented yet
- Optimize innermost parts of loops first, first choice of optimum addresses on the drum
- Optimizing assembler not smart enough

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

65

I compared Mel's hand-optimized programs with the same code massaged by the optimizing assembler program, and Mel's always ran faster. That was because the "top-down" method of program design hadn't been invented yet, and Mel wouldn't have used it anyway. He wrote the innermost parts of his program loops first, so they would get first choice of the optimum address locations on the drum. The optimizing assembler wasn't smart enough to do it that way.



## Most Pessimum

- Flexowriter required a delay between output characters
- Placed instructions just *past* the read head
- “Optimum” is absolute term
- These locations are “most pessimum”

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

66

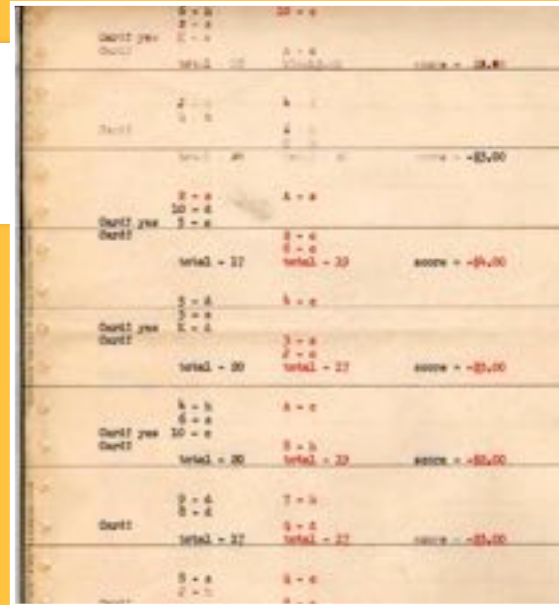
**Mel never** wrote time-delay loops, either, even when the balky Flexowriter required a delay between output characters to work right.

**He just** located instructions on the drum so each successive one was just past the read head when it was needed; the drum had to execute another complete revolution to find the next instruction. He coined an unforgettable term for this procedure.

**Although** “optimum” is an absolute term, like “unique,” it became common verbal practice to make it relative: “not quite optimum” or “less optimum” or “not very optimum.” **Mel** called the maximum time-delay locations the “most pessimum.”

# Times Change

- These days, the scarce resource is *developer time*
- We pay attention to software quality, maintainability, etc.
- We use CPUs rated at billions of instructions per second
- Mel used a single CPU rated at 500 instructions per second: To accomplish *anything* required the *most* hardware performance possible
- Modern values simply did not apply, in the vacuum-tube era



## Change Request

- Finished and got it to run
- Change Request from sales department
- Sometimes the customers lost
- Change the odds and let the Wookie win

Computing Past: Mel, The Realist Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

68

After he finished the blackjack program and got it to run (“Even the initializer is optimized,” he said proudly), he got a Change Request from the sales department.

The program used an elegant (optimized) random number generator to shuffle the “cards” and deal from the “deck,” and some of the salesmen thought it was too fair, since sometimes the customers lost.

They wanted Mel to modify the program so, at the setting of a sense switch on the console, they could change the odds and let the Wookie win.

# Cheat Mode

*The others came out of the ship, on my advice: evidence of any mistrust at this point would humiliate the Karhidish escort, impugning their shifgrethor.*  
—The Left Hand of Darkness by Ursula K. Le Guin (1969)

- Mel balked
- Impugned his shifgrethor, so refused
- Mel gave in, but got the test wrong
- Program would cheat, winning every time
- Mel was delighted with this and refused to fix it

```
8.  If SENSE SWITCH 32 is depressed, there is a better than normal chance  
    of an ace being dealt as the player's first card.
```

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

69

**Mel** balked. He felt this was patently dishonest, which it was, and that it **impinged** on his personal integrity as a programmer, which it did, so he refused to do it.

The Head Salesman talked to Mel, as did the Big Boss and, at the boss's urging, a few Fellow Programmers. **Mel** finally gave in and wrote the code, but he got the test backwards, and, when the switch was turned on, **the program** would cheat, winning every time.

**Mel** was delighted with this, claiming his subconscious was uncontrollably ethical, and adamantly refused to fix it.

## Greener Pa\$tture\$

- Mel left the company for greener pa\$tture\$
- Big Boss asked me to look at the code
- See if I could find the test and reverse it
- Tracking Mel's code was a real adventure



Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

70

After Mel had left the company for greener pa\$tture\$, the Big Boss asked me to look at the code and see if I could find the test and reverse it. Somewhat reluctantly, I agreed to look. Tracking Mel's code was a real adventure.

## Programming Is An Art Form



Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

71

I have often felt that programming is an art form, whose real value can only be appreciated by another versed in the same arcane art; there are lovely gems and brilliant coups hidden from human view and admiration, sometimes forever, by the very nature of the process.

## You Can Learn A Lot About An Individual



Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

72

You can learn a lot about an individual just by reading through his or her code, even in hexadecimal. Mel was, I think, an unsung genius.



## An Innocent Loop



Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

73

Perhaps my greatest shock came when I found an innocent loop that had no test in it. No test. None.

## Infinite Loop

```
1  <?php
2  $array = [3.1, 41, 59, 26, 53];
3  $total = 0;
4  $index = 0;
5  while (true) { $total += $array[$index++]; }
```

Common sense said it had to be a closed loop, where the program would circle, forever, endlessly. Program control passed right through it, however, and safely out the other side. It took me two weeks to figure out. The RPC-4000 computer had a really modern facility called an index register. It allowed the programmer to write a program loop that used an indexed instruction inside; each time through, the number in the index register was added to the address of that instruction, so it would refer to the next datum in a series. In other words, we are indexing through an array.

# Index Register

## THE INDEX REGISTER

The INDEX Register performs several important functions in the RPC-4000. Its primary use is for address modification and, for this purpose, bits 5 thru 17 of the INDEX Register serve to hold a value by which the Data-address Field of an instruction may be incremented. This incremental value may be placed in the INDEX Register by means of a Load Index (LDX) instruction, and can be used by including an Index Tag in the appropriate instruction.

He had only to increment the index register each time through. Mel never used it.

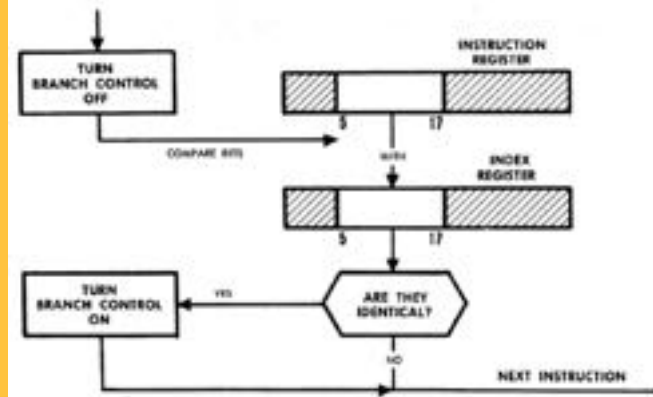
Instead, he would pull the instruction into a machine register, add one to its address, and store it back. He would then execute the modified instruction right from the register.

The loop was written so this additional execution time was taken into account—just as the instruction finished, the next one was right under the drum's read head, ready to go. But the loop had no test in it.

# Hardware Overflow

If indexing is specified for this instruction, a Data Value of zero will turn on the Branch Control, regardless of the index value. This occurs because the zero Data Value, when indexed, becomes identical with the Index Value. Conversely, any Data Value other than zero will turn off the Branch Control, regardless of the Index Value, inasmuch as any non-zero value, when indexed, becomes greater than the Index Value.

Minimum time-----4 word times  
Overflow-----Not a factor  
Branch Control-----Conditionally set "On" or "Off"  
Registers affected---None



- Index flag set
- Yet Mel never used the Index Register
- The carry adds one to the operation code
- Turns the instruction into a Jump instruction
- Next instruction at 000 00

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

76

The vital clue came when I noticed the index register bit, the bit that lay between the address and the operation code in the instruction word, was turned on—yet Mel never used the index register, leaving it zero all the time. When the light went on it nearly blinded me.

He had located the data he was working on near the top of memory—the largest locations the instructions could address—so, after the last datum was handled, incrementing the instruction address would make it overflow.

The carry would add one to the operation code, changing it to the next one in the instruction set: a jump instruction. Sure enough, the next program instruction was in address location zero, and the program went happily on its way.

## An Insane Optimization

- Mel clearly understood the inner workings
- He undoubtedly learned timings with scope and circuit diagrams
- Likely built a collection of tricks
- Even through the 1980s, a Real Programmer could beat every compiler or assembler



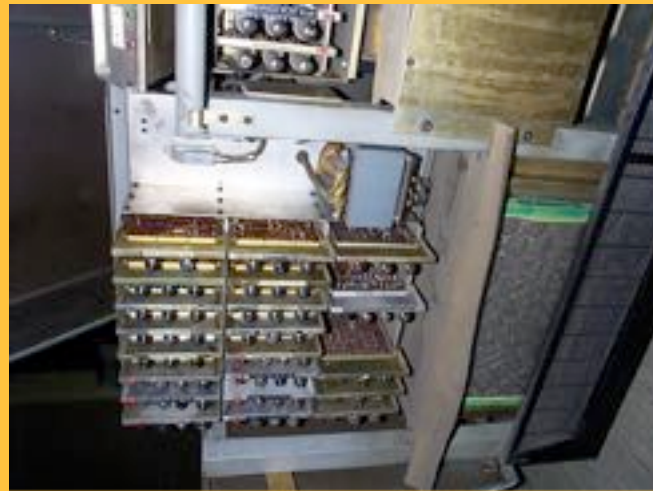
[https://upload.wikimedia.org/wikipedia/commons/0/07/Oscilloscopic\\_tube.jpg](https://upload.wikimedia.org/wikipedia/commons/0/07/Oscilloscopic_tube.jpg)

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

77

Mel clearly understood the inner workings of drum memory and the processor extremely well. He undoubtedly learned the timings with an oscilloscope and circuit diagrams and a lot of poking around. He likely built a collection of tricks to squeeze maximum performance. Even through the 1980s, a Real Programmer could beat every compiler or optimizing assembler.

## LGP-30 Restoration in Stuttgart (1999)



Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

78

I haven't kept in touch with Mel, so I don't know if he ever gave in to the flood of change that has washed over programming techniques since those long-gone days. I like to think he didn't.

## LGP-30 Restoration in Stuttgart (1999)



Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

79

In any event, I was impressed enough that I quit looking for the offending test, telling the Big Boss I couldn't find it. He didn't seem surprised.

## Librazette (August 1956)



LIBRASCOPE'S MURAL ROOM became a study hall for neophyte LFG-30 programmers the week of July 16. Students participating in this first training school for LFG-30 customers included (seated l. to r.) Bill Hopper, Mary Cornell and Chuck Rue, Convair-Pomona; John Corkhill, Convair-San Diego; R. J. Bibbins, Link Aviation; K. A. Hurst, D. D. Parkhurst, C. S. Kikushima and Ides J. Romero, Convair-San Diego; George Kendrick, Convair-Pomona; Chuck Ray, Caltech; and William Clayton, National Security Agency. Standing (l. to r.) are Fred Flannell, class instructor and assistant sales manager of Royal-McBee; and Royal-McBee Applications Engineers Bud Hazlett, Jack Behr and Mel Kaye. (Photo by Duggan)



### Welcome

Librascope welcomes the following new employees who joined us during July:

#### ACCOUNTING—

Robert McMullen  
Arlene Swanson

#### ENGINEERING—

ADMINISTRATIVE—  
Ella Maribach  
Raymond Marinello

#### ENGINEERING—

COMMERCIAL—  
Melvin Kaye

#### ENGINEERING—

SPECIAL DEVICES—  
Melvin Smokler

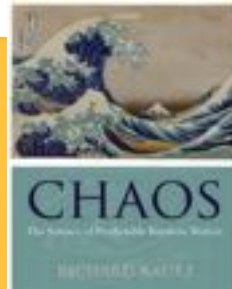
#### MACHINE SHOP—

Howard Cleveland

When I left the company, the blackjack program would still cheat if you turned on the right sense switch, and I think that's how it should be. I didn't feel comfortable hacking up the code of a Real Programmer.



## Side Note: *The Butterfly Effect* (1961)



Edward Lorenz (Ed #4):  
*The Butterfly Effect*  
was created on a  
Royal-McBee LGP-30

In 1961, Lorenz was running a numerical computer model to redo a weather prediction from the middle of the previous run as a shortcut. He entered the initial condition 0.506 from the printout instead of entering the full precision 0.506127 value. The result was a completely different weather scenario.



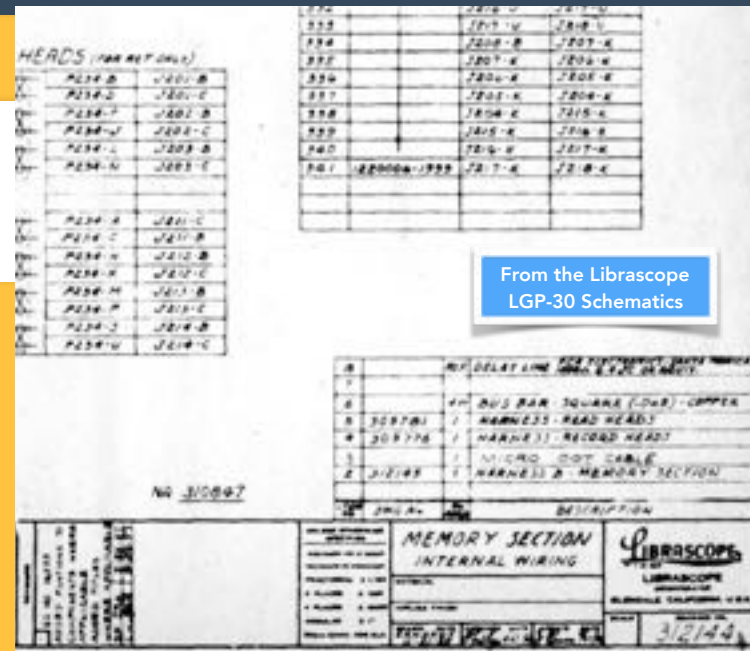
FIG. 10.1 Edward Lorenz, 1957.  
(Courtesy of Edward Lorenz.)

Fortunately, Lorenz had an ace in the hole: a primitive personal computer called a Royal-McBee LGP-30. This desk-sized behemoth was built from vacuum tubes and had a computing power comparable to a modern programmable pocket calculator. All the same, the Royal-McBee was capable of solving a set of weather equations involving a dozen or more variables, time step by time step, printing out a list of updated variables every 10 seconds. In principle, the calculation was just

## Our Timeline

- Hacker Folklore (1982-83)
- Vacuum Tube Computer Programming (1956)
- ▶ ***Modern Times (2017)***
- Video: Warming Up the LGP-30 (6:51 run time)

LGP-30 Schematics: [https://archive.org/details/bitsavers\\_royalPreciatics1959\\_26037699](https://archive.org/details/bitsavers_royalPreciatics1959_26037699)



Finally, that brings us up to today.

## Counting Eds

- Let's put our Eds together: Ed #1, Ed #2, Ed #3, and Ed #4
- One of us (at least!) was a Real Programmer...
- What more can you learn from *me*?



## “The Story of Mel” Explained

**The first thing I do in interviewing a candidate is determine what type of engineer they are. Not that there is anything wrong with journeyman programmers who write the glue for existing API's. But someone who gets it fundamentally is a huge asset to any organization.**

–James Seibel (shared with permission)  
Author of “The Story of Mel” Explained”  
[https://en.wikipedia.org/wiki/The\\_Story\\_of\\_Mel](https://en.wikipedia.org/wiki/The_Story_of_Mel)

Computing Past: Mel, The Realist Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

84

James Seibel wrote the official explanation for The Story of Mel and inspired this talk. He’s now a project manager. This is what he explained to me, and gave me permission to share with all of you.

## The Preparatory Path

- Parisa Tabriz (seated) heads Google's "Department of Chromeland Security"
- She wrote "So you want to work in security?" article
- In explaining "There is no single, standard, or preparatory path" she writes...



Parisa Tabriz is head of the security team for Google Chrome. A lot of people ask her how to get into "infosec," so she wrote an essay, "So you want to work in security?" She explains that there is no single, standard, or preparatory path.

# How Computers and Software Work

**Independent of how you acquire it, you'll benefit from having a strong understanding of applied computer science, or how computers and software work.**

–Parisa Tabriz

## Layers of Abstraction

**Much of applied computer science is about solving problems with layers of abstraction, and security is often about finding the flawed assumptions in those abstractions... and then figuring out how to best fix (or exploit) them.**

–Parisa Tabriz (shared with permission)

Computing Past: Mel, The Realist Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

87

In other words, there's real value in knowing your stuff.

## My Own Example: From the Inside Out

- Cray mainframe being wired together by hand
- CRAY-1 has 60 miles of twisted-pair wire
- Each wire is 1, 2, or 3 feet long
- CRAY-1 built in a circle to reduce wire lengths across the backplane



All Cray Research images scanned from my personal collection

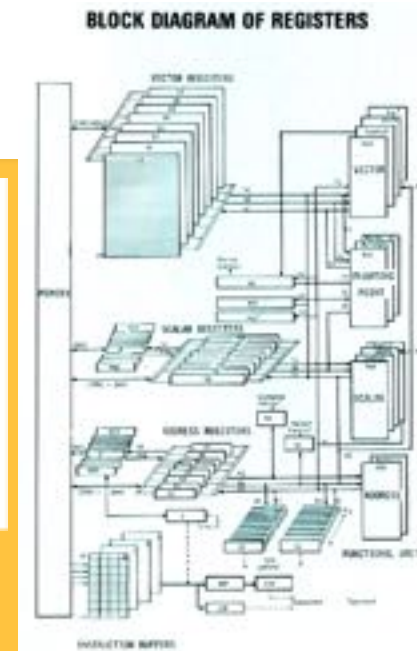
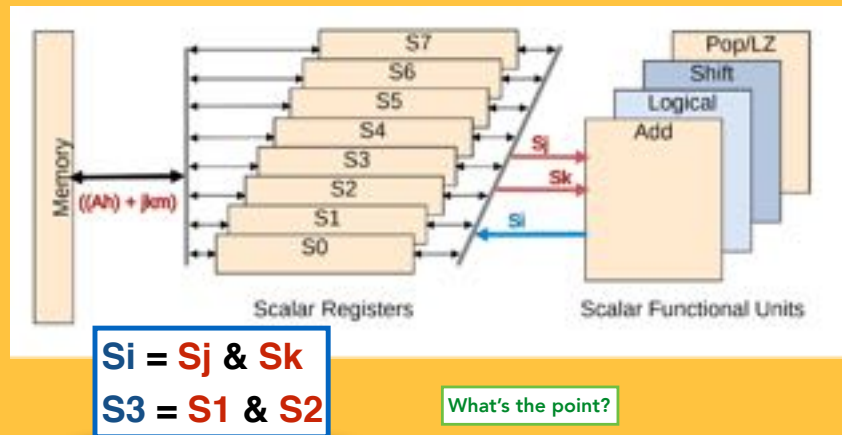
*Computing Past: Mel, The Realest Programmer of All* — #phptek 2017 by Ed Barnard @ewbarnard

88

Let's finish with my own example. This is the CRAY-1 supercomputer, from the inside out. This the Cray mainframe being wired together by hand. The CRAY-1 has 60 miles of twisted-pair wire. Each wire is precisely 1, 2, or 3 feet long. The CRAY-1 was built in a circle to reduce wire lengths across the backplane.



## CRAY-1 CPU (1977)



It's too small to read, but the CRAY-1 CPU block diagram is on the right. Most of us could draw the thing from memory, any time. I took the middle part and made the diagram on the left. So what's the point?

## Logical Product (AND)

**PHP equivalent: `$s3 = $s1 & $s2;`**

A logical product is the AND function:

operand one	1010
operand two	<u>1100</u>
result	1000

When you understand computers (hardware) *and* software, you'll intuitively understand the subtleties of PHP and MySQL

- **Examine each bit position for operand 1 and operand 2**
- **When both are a 1, result is a 1**
- **Otherwise result is a 0**

The CRAY-1 scalar logical units do the same thing as the PHP bitwise operators. If you know one, you know the other. When you understand computers, that is, hardware AND software, you'll intuitively understand the subtleties of PHP and MySQL.

# CRAY-1 Functional Unit Times

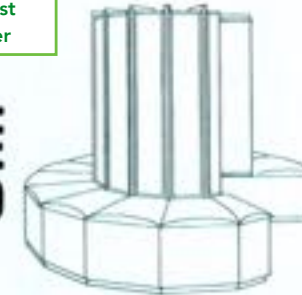
FUNCTIONAL UNITS		
Functional Unit	Unit Time (Clock Periods)	Instructions
Address integer add	2	030, 031
Address integer multiply	6	032
Scalar integer add	3	060, 061
Scalar logical	1	042 - 051
Scalar shift	2	052 - 055
Scalar pop/parity <sup>†</sup>	3	056, 057
loading zero	4	026
Vector integer add	5	027
Vector logical	2	154 - 157
Vector shift	4	140 - 147, 175
Vector pop/parity <sup>†</sup>	6	150 - 153
Floating point add	6	174k1, 174k2
Floating point multiply	7	062, 065, 170 - 173
Floating point reciprocal	14	064 - 067, 160 - 167
Memory (scalar)	11 <sup>††</sup>	070, 178k1, 0
Memory (vector)	7 <sup>††</sup>	100 - 150
		178, 177

<sup>†</sup> Only with vector pop/parity  
<sup>††</sup> For Serial 1 - scalar 10

Knowing the CRAY-1 Functional Unit times, in clock periods, meant that we could beat the FORTRAN compiler every time, just like Mel beat the optimizing assembler

One technique was to take the FORTRAN compiler output and rewrite the innermost loops to take better advantage of the CRAY-1 CPU's resources

CAL  
REFERENCE  
CARD

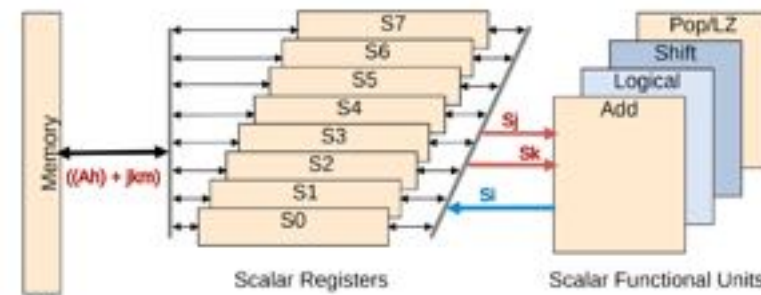


BLOCK DIAGRAM OF REGISTERS

Remember we looked at the tricky access times with drum memory? The CRAY-1 has something similar. There are a dozen functional units, not to mention memory bank access times, that all have different timing. The CRAY-1, which is a single CPU, could literally be running a half-dozen instructions simultaneously, with instruction results arriving at different times. **Knowing** the CRAY-1 Functional Unit times, in clock periods, meant that we could beat the FORTRAN compiler every time, just like Mel beat the optimizing assembler. **One technique**, for example, was to take the FORTRAN compiler output and rewrite the innermost loops to take better advantage of the CRAY-1 CPU's resources.

# CRAY-1 Scalar Functional Units

"All functional units can operate concurrently so that in addition to the benefits of pipelining (each unit can be driven at a result rate of 1 per clock period), there is also parallelism across the units."



Scalar Add is 3 Clock Period (CP):

CP 1:  $S0 = S1 + S2$  (Operands arrive at Scalar Add Functional Unit)

CP 2:  $S3 = S4 + S5$  (Scalar Add takes new operands every CP)

CP 3:  $S6 = S1 - S5$  (Operands immediately available for next instruction)

CP 4: Result of  $S1 + S2$  available in  $S0$

CP 5: Result of  $S4 + S5$  available in  $S3$

CP 6: Result of  $S1 - S5$  available in  $S6$

This helps understand when results arrive out of order, and asynchronous programming in general

Computing Past: Mel, The Realist Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

92

Here's an example of overlapping add instructions. Yes, this is machine-language programming, just like Mel. This kind of experience helps you understand pipelining. **This helps** you understand what's happening when results arrive out of order, or even asynchronous programming in general.

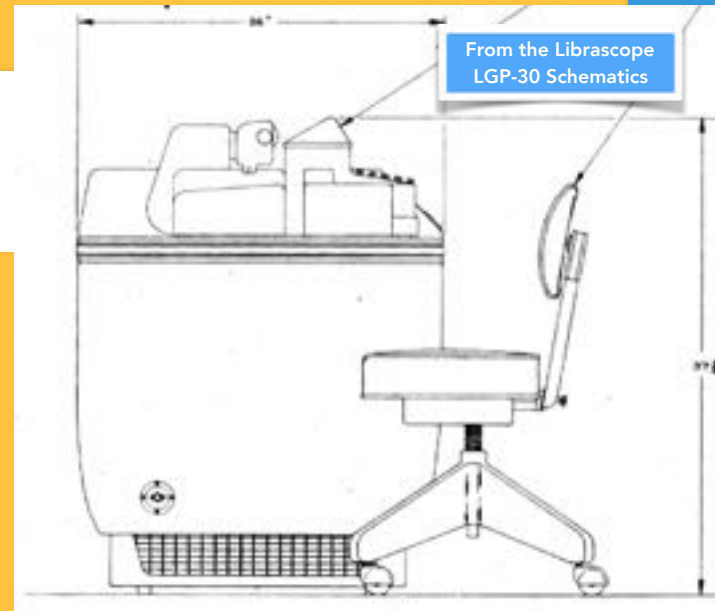
## Our Timeline

- Hacker Folklore (1982-83)
- Vacuum Tube Computer Programming (1956)
- Modern Times (2017)



**Video: Warming Up the LGP-30  
(6:51 run time)**

LGP-30 Schematics: [https://archive.org/details/bitsavers\\_royalPreciatics1959\\_26037699](https://archive.org/details/bitsavers_royalPreciatics1959_26037699)



# Warming Up The LGP-30



Uploaded on Dec 8, 2008  
The LGP 30 computer is a computer made of valves, diodes and a drum memory. This machine is from 1958. It is located at the museum of the computer science faculty of the university of Stuttgart, Germany. The computer is in a fully functional state.

In this clip, the machine is shown starting up and doing some work. Just to get an imagination what that kind of machine looked and feelled like.

For further information (in German), go to:  
<http://computermuseum.informatik.uni-stuttgart.de/>

Category Science & Technology  
License Standard YouTube License

<https://www.youtube.com/watch?v=7WaYYNUCWMY>

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

## Summary

- Computers have changed, thank goodness
- Computer programming is still an art—and a science
- I've beaten the CRAY-1 assembler with hand coding; those lessons carry forward
- Find your awesome: I just showed you mine!



CRAY-1 in Deutsches Museum  
by Clemens Pfeiffer

Computing Past: Mel, The Realest Programmer of All — #phptek 2017 by Ed Barnard @ewbarnard

95

**Computers** have changed, thank goodness. **Computer** programming is still an art—and a science. **This** is not about me, but I have beaten the CRAY-1 assembler with hand coding; those lessons do carry forward. **Find** your awesome: I just showed you mine!



## Thank You

- Ed Barnard, [InboxDollars.com](http://InboxDollars.com)
- Twitter: @ewbarnard
- Slide deck and links to more reading: <http://otscripts.com/computing-past-mel-the-realest-programmer-of-all/>
- Please rate this talk: <https://joind.in/talk/c19c4>