AccessionIndex: TCD-SCSS-T.20121208.016
Accession Date: 8-Dec-2012
Accession By: Prof.J.G.Byrne
Object name: HP9100A Programmable Calculator
Vintage: c.1968
Synopsis: HP's first calculator. S/N: 816-02612.

Description:
The HP9100A was Hewlett Packard's first calculator, an electronic programmable reverse Polish scientific calculator, a famous product. It was the intellectual and philosophical precursor to the equally famous HP35, their first handheld calculator.

The prototype, a floating-point algebraic calculator called the *Green Machine*, was begun privately c.1964 by Tom Osbourne, based on algorithmic state machine concepts from his PhD. Once working, he prepared patent documents and then conducted demonstrations over the next year to many companies (including HP) without eliciting interest until HP was prompted by an ex-colleague, and took on both the project and Osbourne. See his recollections in this catalog's related folder. The Green Machine and its design documentation are now in the Smithsonian Institute.

Within HP this design was combined with work on a fixed-point design by the mathematician Malcolm McMillan, and evolved to a microprogrammed reverse-Polish machine with a 3-level stack, that used a high-density inductive printed-circuit ROM for algorithms, core memory for storage, a CRT display and a magnetic card reader and writer. It did not use any integrated circuits (which were not sufficiently low-power then), instead using a form of dynamic logic that Osbourne called "power gating" to reduce logic complexity and considerably reduce power consumption (to 70W max), and it was notably robust, a feature of HP products of that time.

The architecture could perform several operations in parallel and included multi-way branching. It had three stack levels, X, Y and Z, all visible on the display. The result of two operand functions was stored in Y. There were 16 storage registers in core memory, and their contents survived power cycling. Memory was shared between programs and data (each register could hold 14 program steps), allowing self-modifying code. The reader/writer could store 196 program steps on each of two tracks on cards somewhat smaller than credit cards.

The keys were in four groups: scientific functions, stack and memory functions, digits and basic math, and programming functions. There was a thumb wheel for selecting the number of digits displayed after the decimal point, and switches for degrees or radians, fixed or floating point, on/off and program/run. It performed many scientific calculator functions like $\log_e$, $\log_{10}$, antilog, square-root, trigonometrics, hyperbolics, their inverses, coordinate conversions, etc, see the brochure in this catalog's related folder. Internally numbers had 12 digits, but users were presented with a 10-digit mantissa and 2-digit exponent, with a floating point range of 10^{-98} to 10^{+99}.

Although these machines are often called RPN (along with the HP9810 which also employs the three-register, result-to-Y system), there is no automatic stack lift/drop, and they don't behave as any other RPN system, and their behaviour doesn't match modern mathematical definitions of RPN.

The HP9100A should not to be confused with the HP9100B, which had more storage registers, more program steps, and subroutines.

Mechanically, the HP9100A top cover hinges at the rear. The power supply and CRT display are mounted on the underside of the top cover. The base unit contains the logic, with a large motherboard (board 6) that is mostly a matrix of diodes, which form AND and OR logic gates (which do not need power) for things like the incrementing and decrementing. Underneath are the 512 x 64-bit program ROM boards 11-14. There are seven plug-in boards, listed from left to right in the table below: one small one at each side (9 & 8), two larger boards (1 & 2) at the left and three larger boards (3, 4 and 5) in the middle (all boards numbered as per HP9100B schematics; the HP9100A may be different in this regard, but it is unlikely).

Board No.	Function
9	Microcode branch logic and error flip-flop
1	Control logic and control ROM
2	20 J-K flip-flops
3	2208-bit Core memory
4	Core memory sense and inhibit circuits
5	20 J-K flip-flops
8	Program ROM instruction decoder

Table 1: HP9100A plugin boards

Board 9 is the microcode branch logic and error flip-flop. Board 1 contains the control logic (CPU clock and control ROM), which clocks at 1.21 MHz (825 nS cycle). Boards 2 and 5 each contain 20 J-K flip-flops. Board 8 is the 64-bit program ROM instruction decoder. Board 3 is the 2208-bit core memory board (in the HP9100B this was increased to 3840-bits).

Board 4 is the core memory sense and inhibit circuits, essentially the 'data' interface to the core memory plane, the address drivers being on board 3 with the core plane. This is another difference between the HP9100A and HP9100B. In the HP9100B, board 3 has address drivers only, while board 4 contains the core memory unit and the sense and inhibit circuits. Also, in the HP9100B the address drivers are matched to the core memory plane, so boards 3 & 4 are joined together by double-width handles and the boards are swapped as a set.

There are actually two ROMs in a HP9100A. One is a ‘core-on-a-rope’ (also called ‘woven-wire’ or ‘braided-wire’) *control ROM* on the control logic PCB (board 1). This is essentially storage for the low-level microcode instruction expansion, see further below. It was a non-traditional form of read-only core memory. It consisted of 29 ferrite cores, with a sense wire through each of the 29 cores, and 64 drive wires through some cores (representing a logic one) and around the outside of other cores (representing a logic zero). A current pulse in the drive wire inductively generated all 29 bits of the expanded microcode instruction simultaneously.

The other is the *program ROM*, which is a 14-layer (some say 16-layer) PCB in the middle of the bottom of the machine. It has 256 x 64 bits in a HP9100A (512 x 64 bits in a HP9100B), and works by inductive coupling between PCB tracks. The track linewidth and separation are both only 0.010”, five times smaller than usual at that time. This program ROM is 64-bits wide, but that does not make it a 64 bit machine.

The HP9100A architecture is not conventional, and rarely fully explained. HP state that it had a very-long-instruction-word (VLIW) architecture, with highly nested VLIW programming, but this is very debateable. There was no general-purpose arithmetic/logic unit (ALU) which can operate on various registers. Osbourne states that its arithmetic section was distributed throughout the system by assigning small, but specialized, tasks to the various working registers within the system. Separate buses then interconnected selected pairs of these registers. So there were a number of working registers, some with their own associated increment, decrement or shift logic blocks. These associated logic blocks were tantamount to concurrent functional units, as for example, one working register could be incremented and another decremented in a single microcode step. Some operations, like shifts, worked on an entire register. There did not seem to be any two-operand operations making use of two registers. Osbourne states that typically three to seven operations were executed simultaneously (i.e. in parallel) during each microcode step.

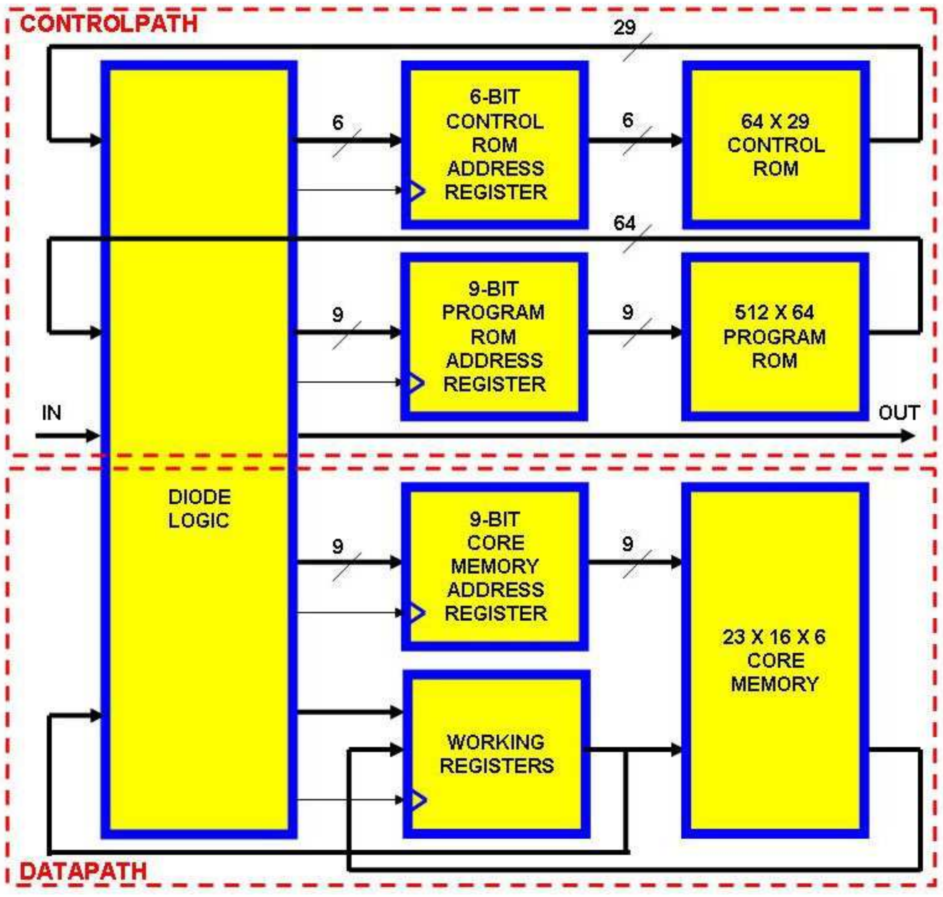

Figure 1: HP9100A processor architecture

There were no programming tools, no assembler, compiler, or linker, because there was no processor in the classical sense. The processor was a microprogrammed state machine that had no classical instruction set or fetch/execute cycle, and did not run software in the conventional sense, but employed the program ROM to direct the machine’s operations during each state, to test branch conditions and specify a next state. The state machine was programmed using graphical flow charts (ASM charts).

The absence of a general-purpose ALU and the graphical programming underscores that the design was pure ASM, with no attempt to present a traditional programming model. Rather it was just a bunch of datapath units sequenced by a controlpath finite state machine, see Figure 1.

ASM charts differ from software flow charts by the incorporation of time, so that multiple register operations can be executed subject to a multiple conditions all within a single state. This especially facilitates iterative operations. In the HP 9100A, loops of this type frequently consisted of a single program word to be repeated until the exit conditions were met, at which time the microinstructions were inhibited and the next program word selected. Half of the 64-bit program word was used to encode these microinstructions. The other half of the 64-bit ‘program word’ was used for addressing and to test conditions of both internal registers and asynchronous external signals generated by I/O devices. The results of each test determined the next program ROM address and whether or not to suppress the action of the microinstructions in the other half of the program word.

Field	Function
0-5	Microinstruction
[x:x+?]	Condition code
[y:y+8]	Branch target 1
[z:z+8]	Branch target 2

Table 2: HP9100A 64-bit program word format

HP state that the HP 9100A processor design had 64 different instructions that were encoded in a 6-bit field within each program word. The ‘core-on-a-rope’ control ROM expanded that 6-bit instruction field to a 29-bit microcode instruction ‘control word’. Nowadays such an approach is now called vertical microprogramming, as distinct from horizontal microprogramming where the 29 bits would be embedded in a wider program word. While the control ROM could be thought just an efficient substitute for random logic, Osbourne also states that the control ROM could alter its address, so, for example, it could sequence through several control ROM addresses during a single microprogram step. Nowadays this would be called a sequence of nanocode. However, Osbourne did not know about even microprogramming when the HP9100A was designed, let alone vertical microcoding or nanocoding.

Both the program and control words were input to the diode logic on the motherboard. Each program word included the next control ROM address, modified by logic gates on the motherboard in response to conditional branching, etc.

An interesting technique was used to test conditions. All of the individual conditions to be tested were encoded, then compared to a condition subfield of the program word, and the result determined which of two next-program-address fields was selected. Osbourne states this technique is free from synchronization failures, but without substantiation. Board 9 contains the microcode branch logic and error flip-flop, but a single flip-flop would only reduce, not eliminate, the probability of synchronization failure.

In all there were 9 program ROM address flip-flops, 6 control logic address flip-flops, and 9 core memory address flip-flops, plus 16 flip flops for the working registers, a total of 40 flip-flops, spread over the two flip-flop boards. Although the core memory was 6-bits wide, not all the working registers were 6-bit. Some had associated logic blocks, and one register appeared to be particularly important, with the full 6 bits, and interfaces to the keyboard, card reader and I/O bus, etc. The diode logic determined whether or not a flip-flop was clocked at the end of each 825 nS clock cycle.

The random-access core memory held the X, Y, and Z and sixteen general registers (not to be confused with the working registers). The X, Y and Z registers could only represent floating-point values, with a 10-digit signed mantissa and a 2-digit signed exponent. The 16 general registers were 6-bits wide per digit, nominally a BCD digit (but hex F represents -1), sign bit and blanking bit, or a 6-bit program code, equivalent to one keycode, as there were 62 keys. Thus there were 19 accessible registers in the core memory. However, it actually implemented 23 registers. The remaining four registers were not user-accessible, being used for internal storage of intermediate values and machine state.

Maths operations were performed one digit at a time, least significant digit first, first retrieving an operand digit from core memory, then operating on it, then saving the resulting digit back to core memory. Addition was iterative, incrementing one of the digits, decrementing the other, until the latter reached zero. Multiplication was by repeated addition, division by repeated subtraction.

The maths functions were in the 64-bit program ROM. Osbourne and Cochran stated that Malcolm McMillan's implementation of Jack Volder’s CORDIC algorithms (which use iterative shifts and adds) were the basis for the transcendental functions in the calculator (the provision of inc/dec/shift fits this assertion). It also employed natural logarithms as well as J.E.Meggitt’s algorithms for pseudo-division and pseudo-multiplication to calculate logarithms. It could add or subtract two floating-point numbers in 2 mS, multiply two numbers in 35 mS, and perform a trigonometric calculation in 350 mS.

User 'programs', a sequence of ‘instructions’ equal to keycodes, were stored in the general registers (registers were sacrificed to the program). Each of the 16 registers could represent either a floating-point value or up to 14 user instructions (for a total capacity of 196 instructions). External memory could be added by connecting the HP9101A Extended Memory, which added 248 general registers or 3472 instructions. Normally the microcode responded to keystrokes, and branched as needed, but when a program was to be stepped or run, the microcode looked to the general registers for the next instruction. This is the effective user-level programming model.

Thus in modern terms the HP9100A user-level architecture and programming model is an ‘RPN’ floating-point processor with instruction opcodes equal to keycodes, while its low-level microarchitecture is a vertically microprogrammed algorithmic state machine (ASM) capable of executing sequences of nanoinstructions.

Key	(octal) Opcode	Operand	Instruction
0..9	00..09		0..9
e, a. b, f, c, d	12..17		e, a. b, f, c, d
CLEAR	20		x, y, z, R(e), R(f), prefix, FLAG ← 0
•	21		Decimal point
ROLL ↑	22		z ← y, y ← x, x ← z
x → ( )	23	n	Save x → R(n)
y ⇆ ( )	24	n	Swap y ⇆ R(n)
↓	25		x ← y, y ← z, z ← z
ENTER EXP	26		Begin exponent
↑	27		z ← y, y ← x, x ← x
x ⇆ y	30		Swap x ⇆ y
ROLL ↓	31		x ← y, y ← z, z ← x
CHG SIGN	32		Change sign of mantissa or exponent
+	33		y ← y + x
−	34		y ← y − x
÷	35		y ← y ÷ x
×	36		y ← y × x
CLEAR x	37		x ← 0
y → ( )	40	n	Save y → R(n)
STOP	41		Halt and wait for manual instruction
FMT	42	k	For use with peripherals
IF FLAG	43	p, q	If false, PC ← PC+3 Elseif p = (0-9,a-f), then PC ← $(pq)_{14}$ Else PC ← PC+1
GO TO ( ) ( )	44	p, q	PC ← $(pq)_{14}$
PRINT	45		For use with peripherals
END	46		PC ← $(00)_{14}$ and end program
CONTINUE	47		Start program execution
IF x = y	50	p, q	If false, PC ← PC+3 Elseif p = (0-9,a-f), then PC ← $(pq)_{14}$ Else PC ← PC+1
	51		
IF x < y	52	p, q	If false, PC ← PC+3 Elseif p = (0-9,a-f), then PC ← $(pq)_{14}$ Else PC ← PC+1
IF x > y	53	p, q	If false, PC ← PC+3 Elseif p = (0-9,a-f), then PC ← $(pq)_{14}$ Else PC ← PC+1
SET FLAG	54		FLAG ← true
‖y‖	55		x ← magnitude(x)
π	56		x ← π
PAUSE	57		Pause for ~150 mS
ACC +	60		R(f) ← R(f) + x, R(e) ← R(e) + y
RCL	61		x ← R(f), y ← R(e)
TO POLAR	62		x ← $\sqrt{(x^2 + y^2)}$, y ← $\tan^{-1}(y/x)$
ACC -	63		R(f) ← R(f) - x, R(e) ← R(e) - y
int x	64		x ← integer(x)
ln x	65		x ← $\log_e(x)$
TO RECT	66		x ← x·cos(y), y ← x·sin(y)
hyper	67	sin | cos | tan	Hyperbolic prefix

sin x	70		x ← {prefix}sin(x)
tan x	71		x ← {prefix}tan(x)
arc	72	sin | cos | tan | hyper	Inverse prefix
cos x	73		x ← {prefix}cos(x)
e^x	74		x ← e^x
log x	75		x ← $\log_{10}(x)$
√x	76		x ← √x
	77		

Table 3: HP9100A instruction set
*The program counter (PC) is base-14, i.e. (**pq**)$_{14}$ is of the form 01-09,0a-0d, … dd*
*There are 16 general registers R(**n**) where **n** = 0-9,a-f*
*The FMT operand **k** is any octal key code (00-77)$_8$*

This machine was a considerable achievement. The emergence of microprocessors and semiconductor memories allowed the algorithms to be transferred out of the microarchitecture and into conventional memory, so subsequent HP calculators were able to adopt a more traditional architecture.

Accession Index	Object with Identification
: TCD-SCSS-T.20121208.016	HP9100A Programmable Calculator. S/N: 816-02612. Core memory label: 810680-A01

Many thanks to Dr.Tony Duell for technical detail, review and corrections.

Trivia1: Supposedly the first machine to be referred to as a "personal computer".
Trivia2: Tom Osbourne is the inventor of Algorithmic State Machines.
Trivia3: He is also the inventor of "Chip Select", a derivative of power gating.
Trivia4: He also led the HP35 calculator project.

References:

1. Osbourne, T.E., *Hewlett-Packard Calculator Architectures*, Chapter 50 in: Siewiorek, D.P., Bell, C.G., Newell, A., *Computer Structures: Principles and Examples*, McGraw-Hill, 1982.

2. *HP Journal*, Hewlett-Packard, September 1968.

3. *HP Journal*, Hewlett-Packard, October 1970.

4. HP9100 Documentation: http://hpmuseum.net/exhibit.php?hwdoc=50

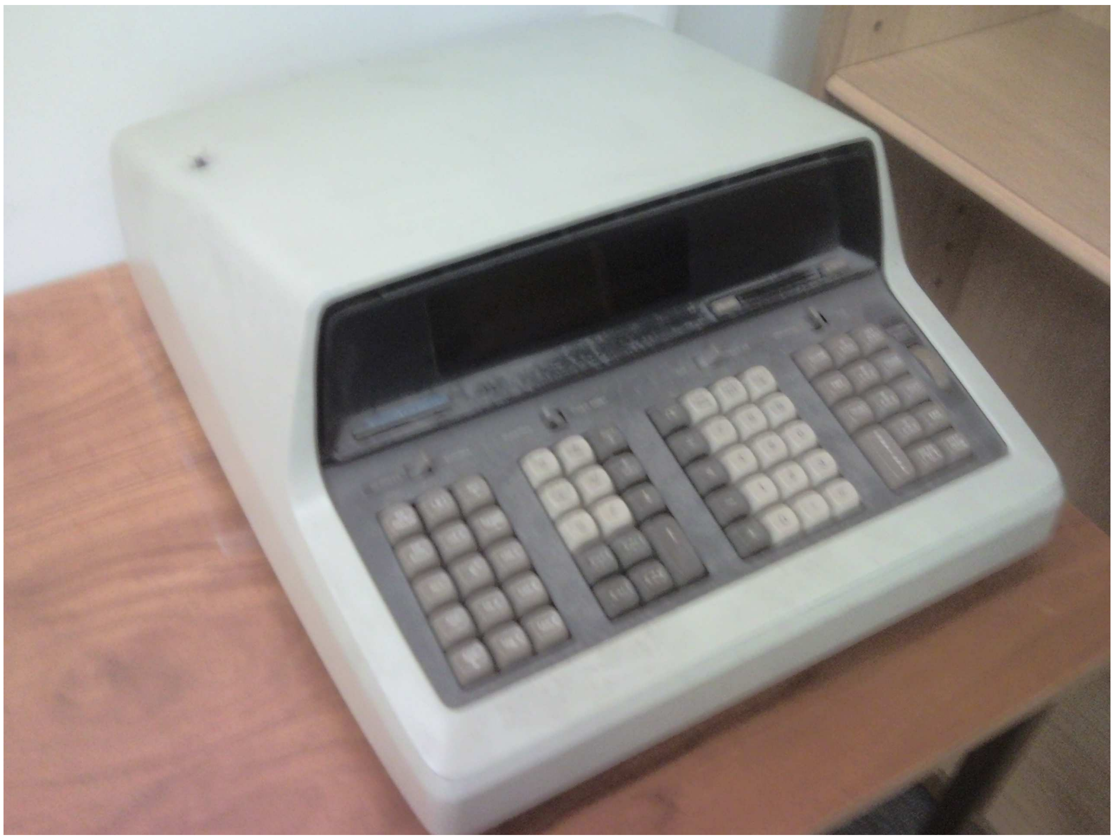
Figure 2: HP9100A Programmable Calculator three-quarter view

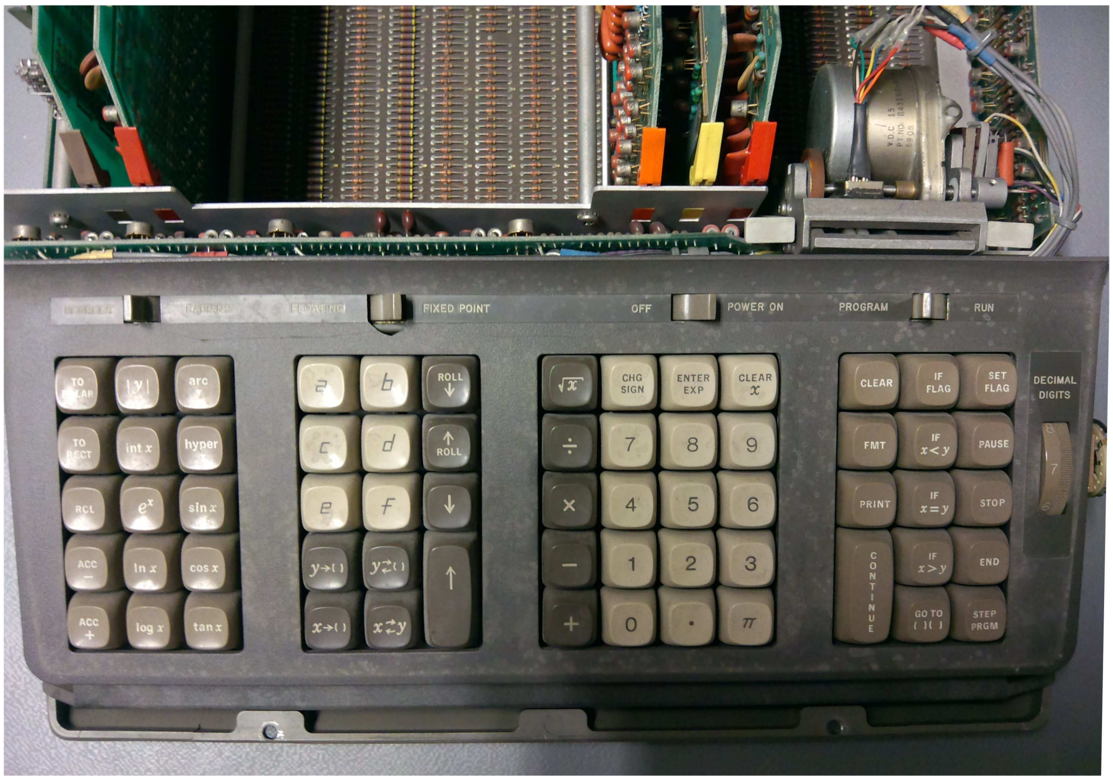

Figure 3: HP9100A keyboard

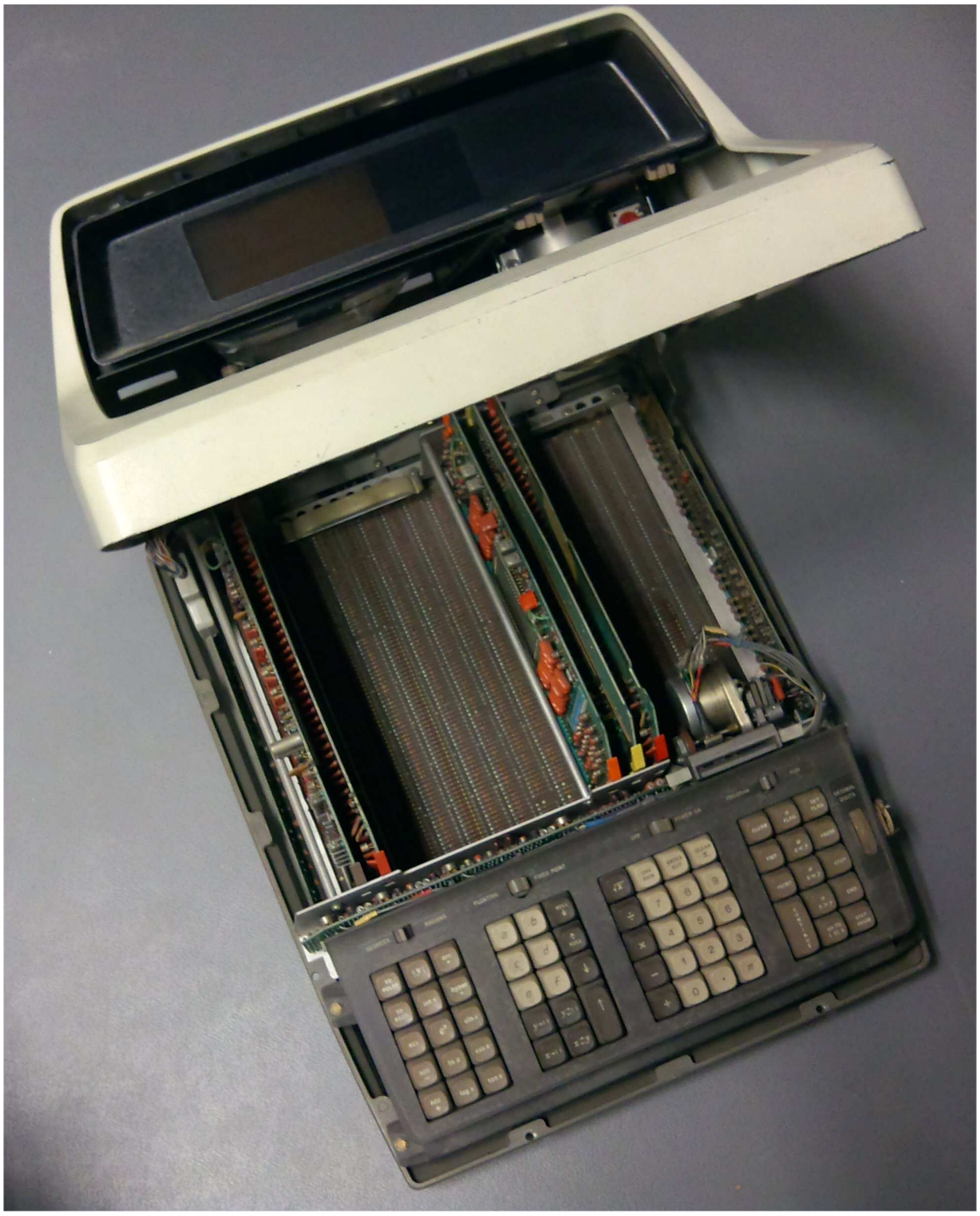
Figure 4: HP9100A internal view with top lifted

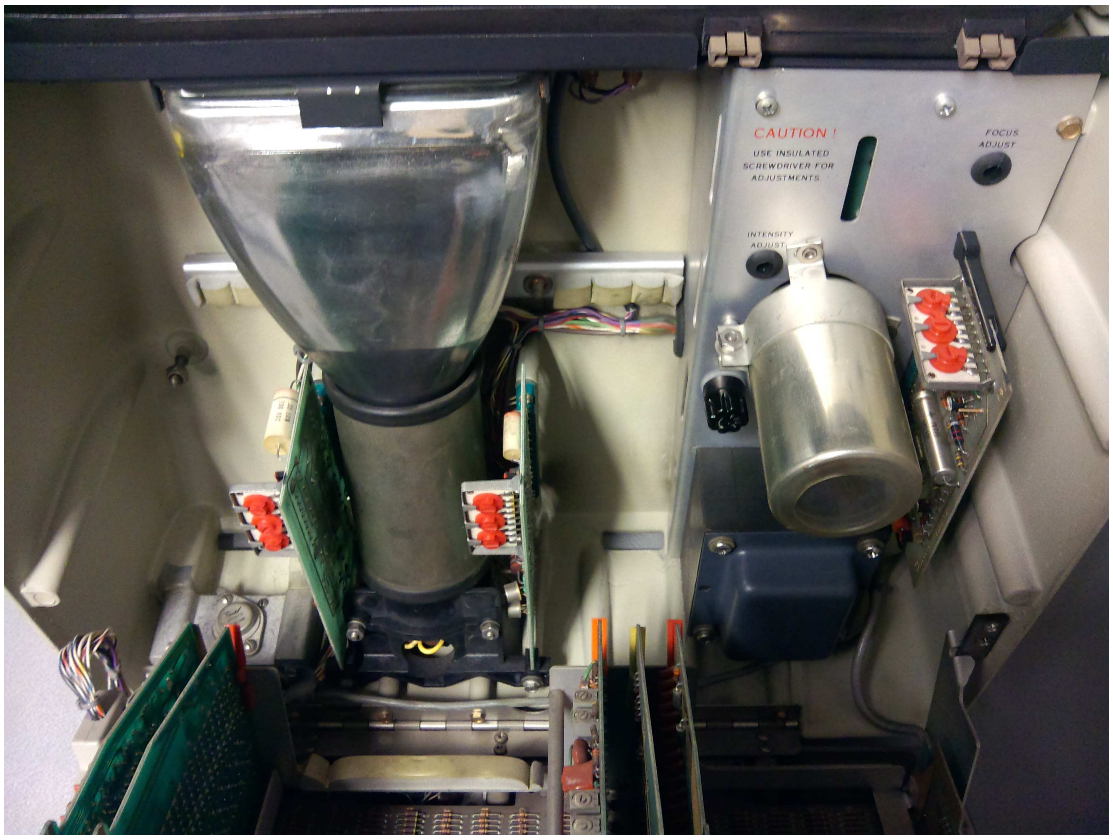

Figure 5: HP9100A CRT display and power supply

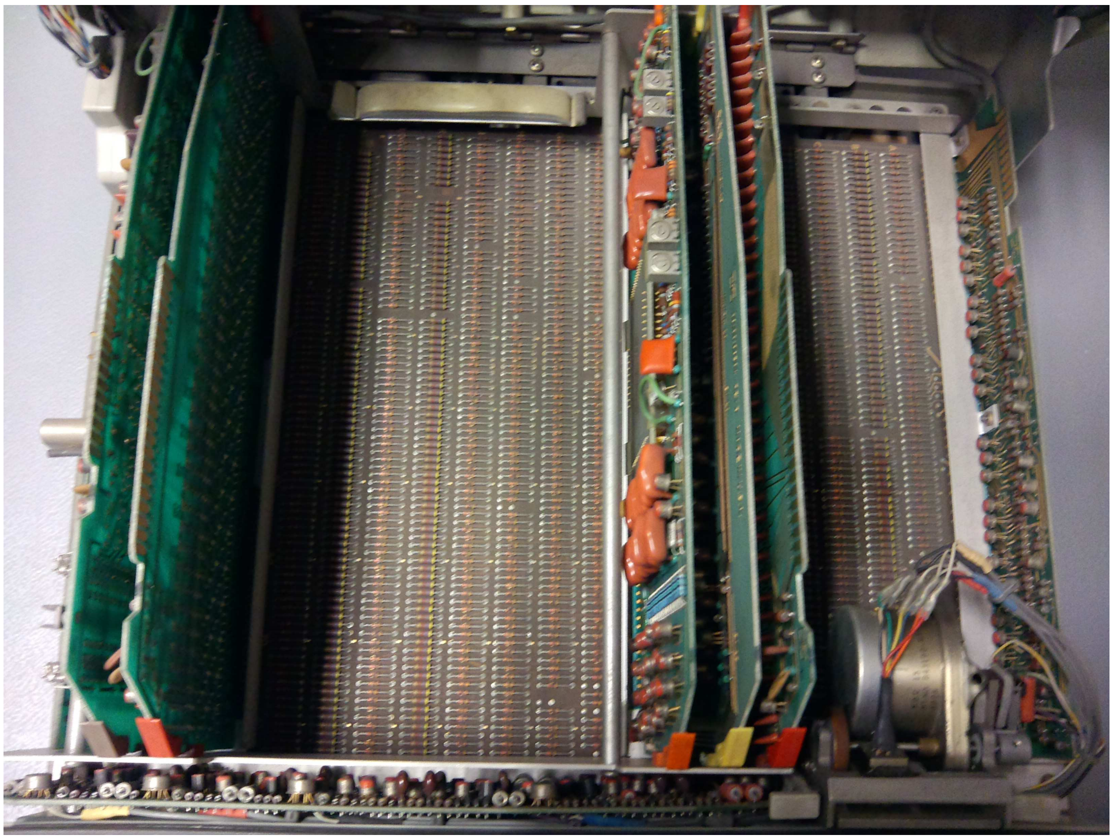
Figure 6: HP9100A top view of motherboard and plug-in boards

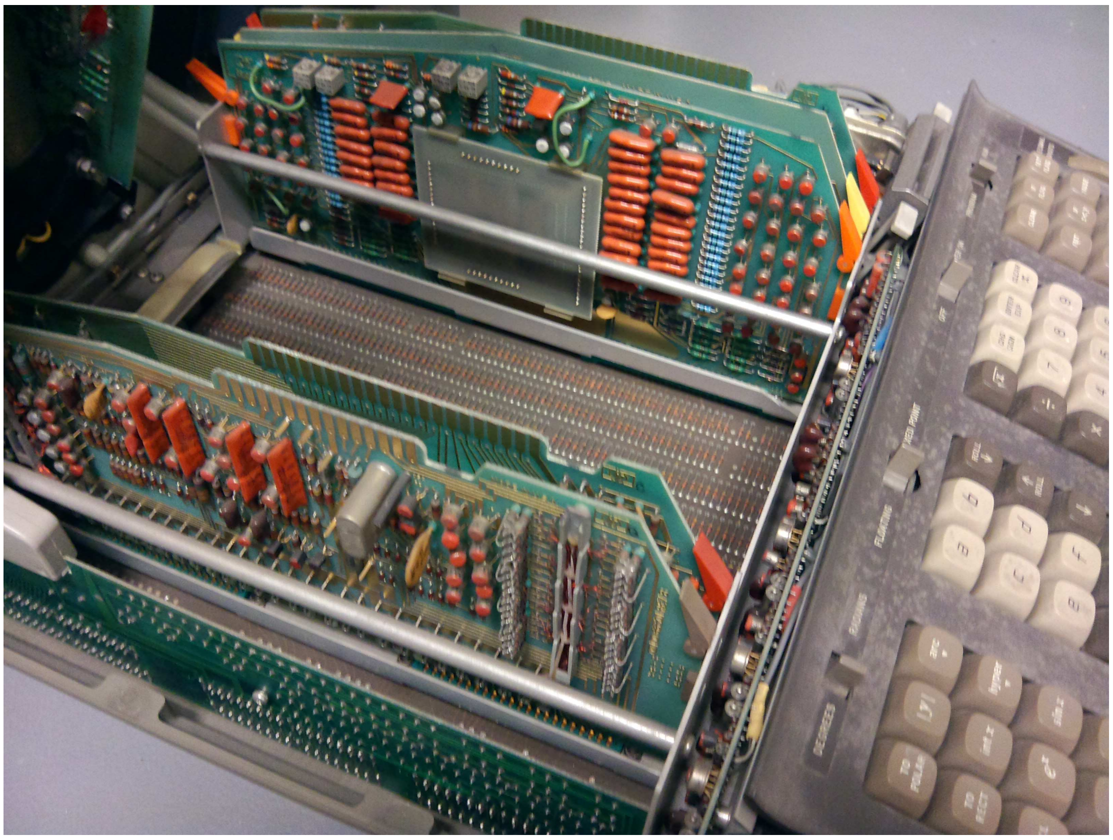
Figure 7: HP9100A three-quarter view of motherboard and plug-in boards

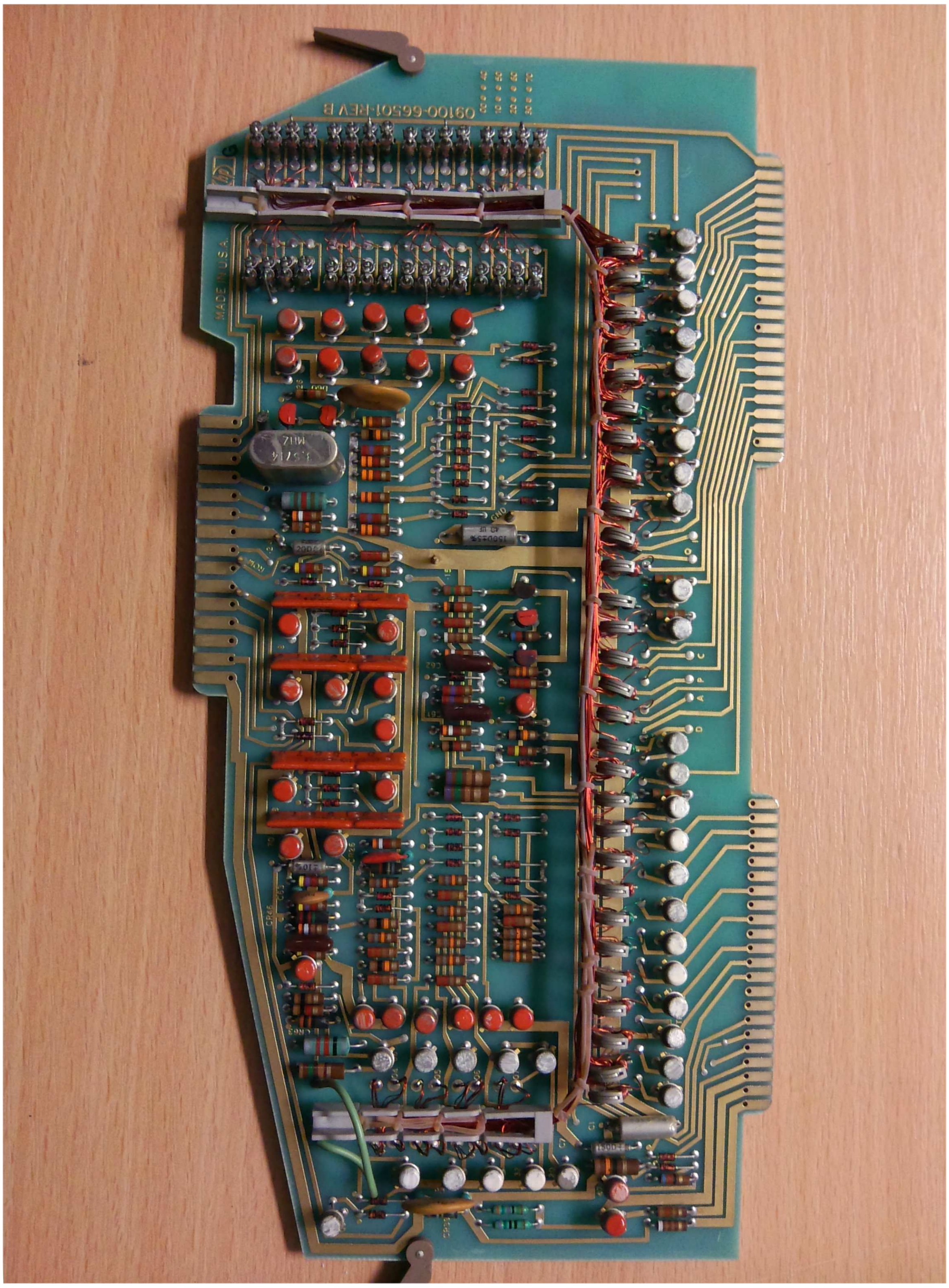

Figure 8: HP9100A control logic board 1 top view
The 'core-on-a-rope' control ROM is the column of ferrite cores at right

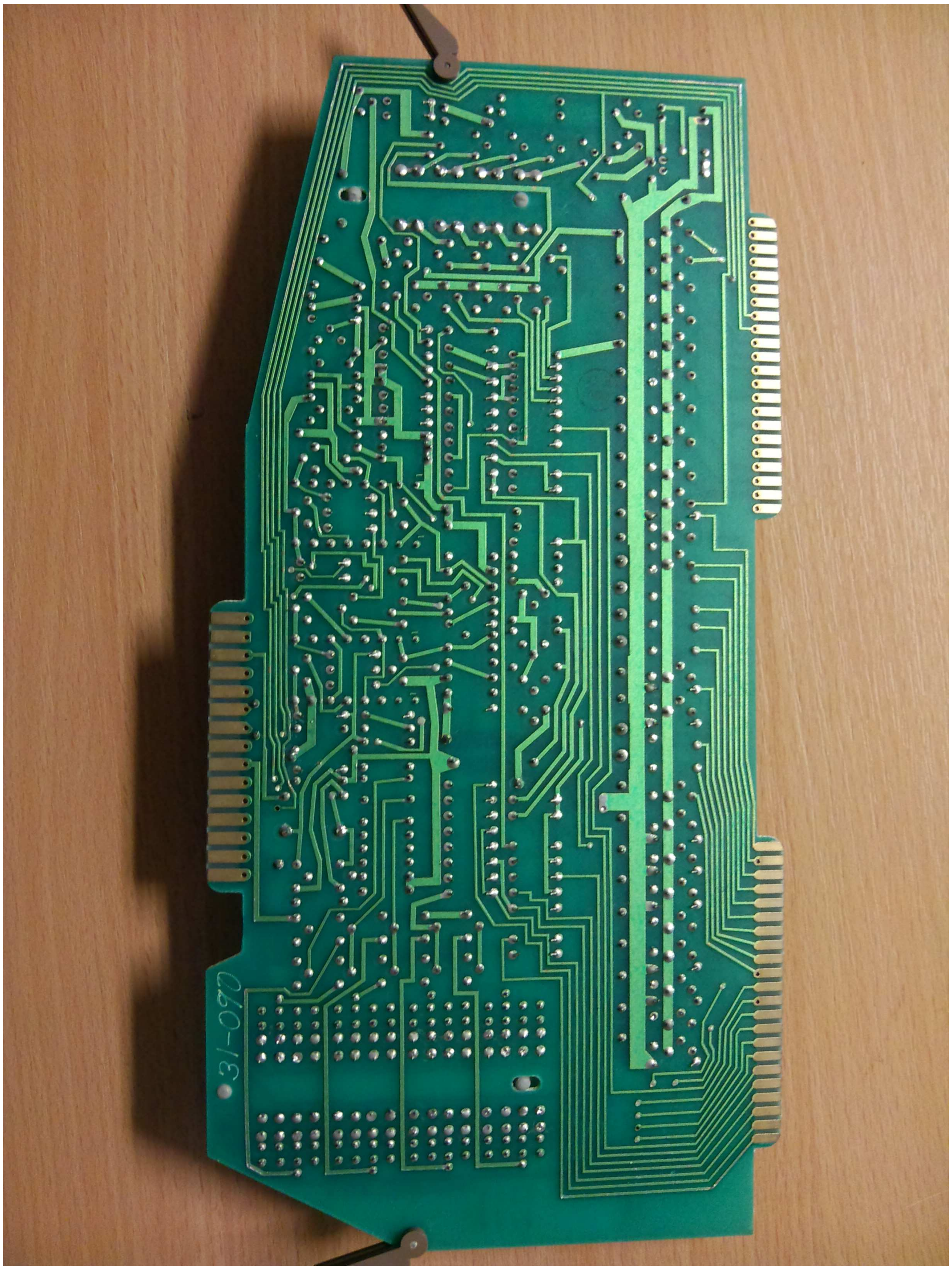

Figure 9: HP9100A control logic board 1 rear view

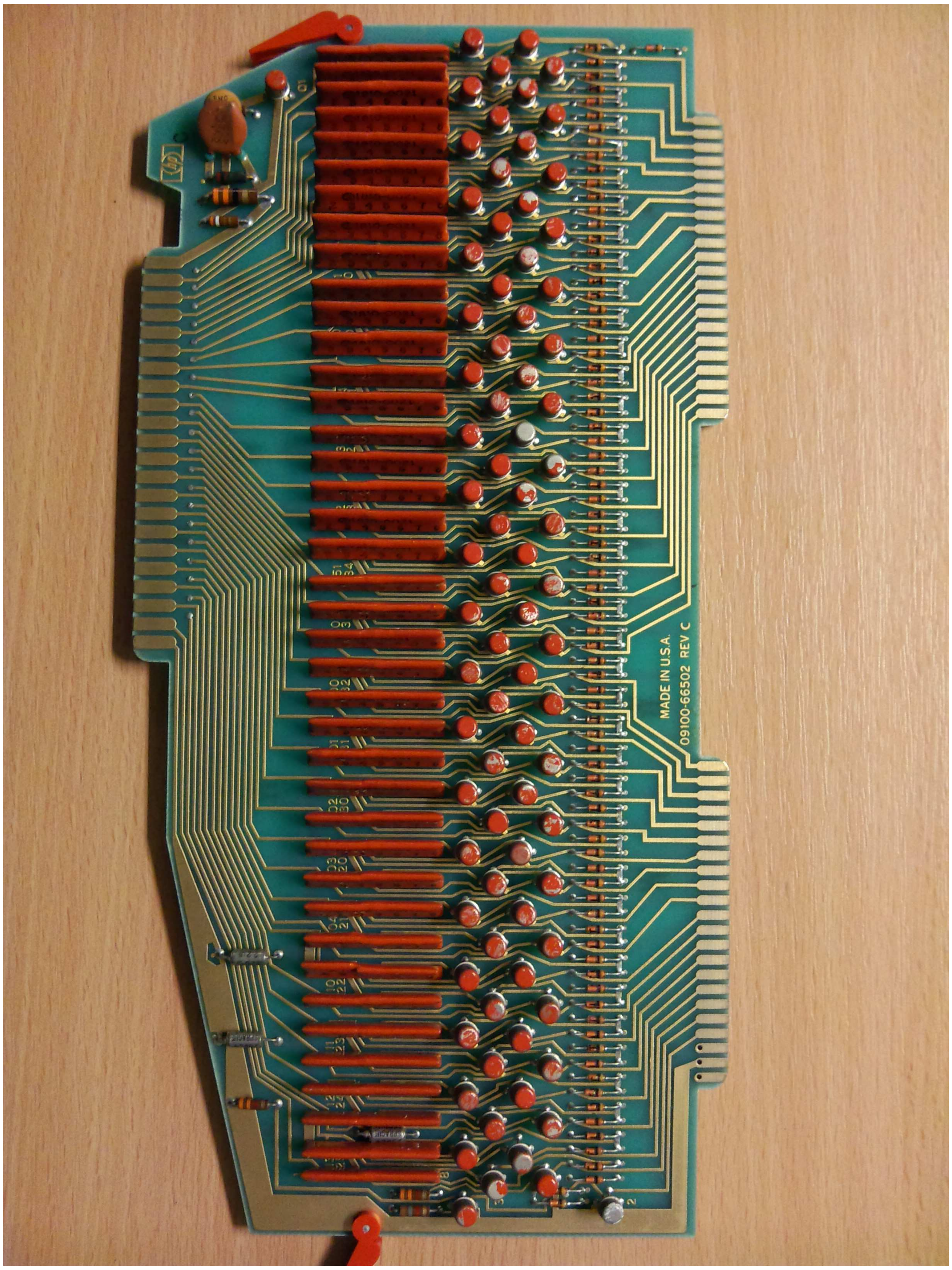

Figure 10: HP9100A flip-flop board 2 top view

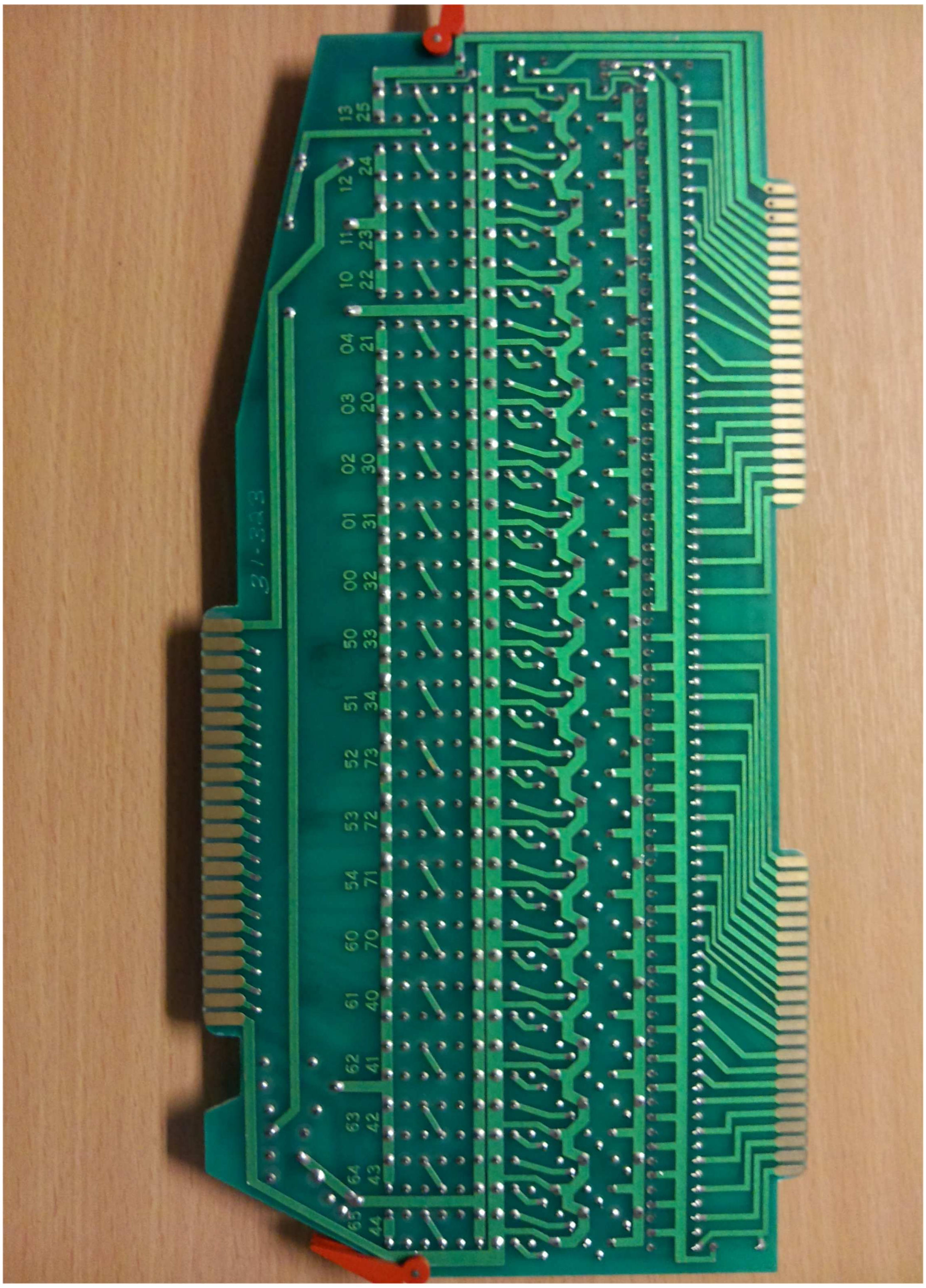
Figure 11: HP9100A flip-flop board 2 rear view

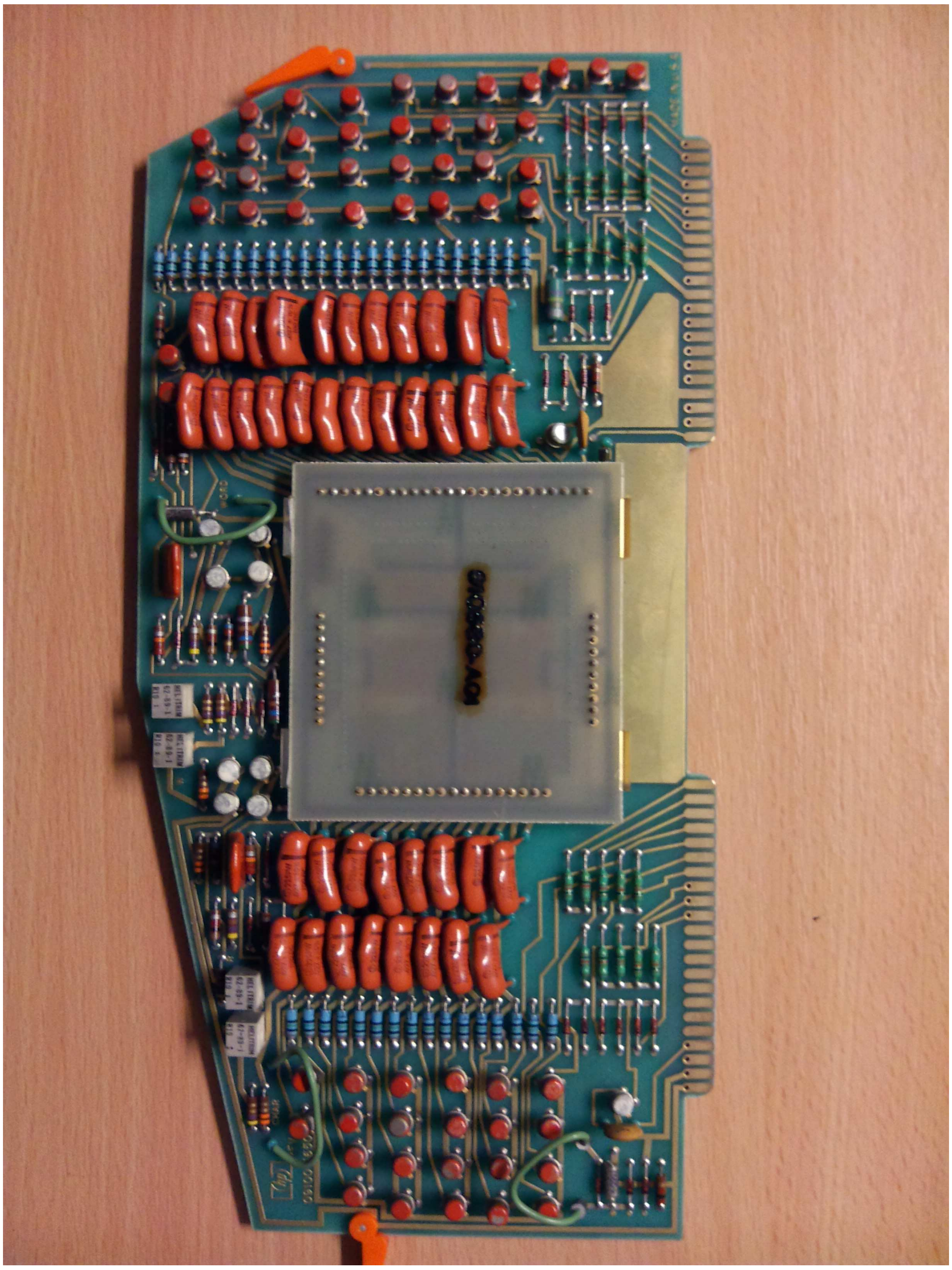
Figure 12: HP9100A core memory board 3 top view

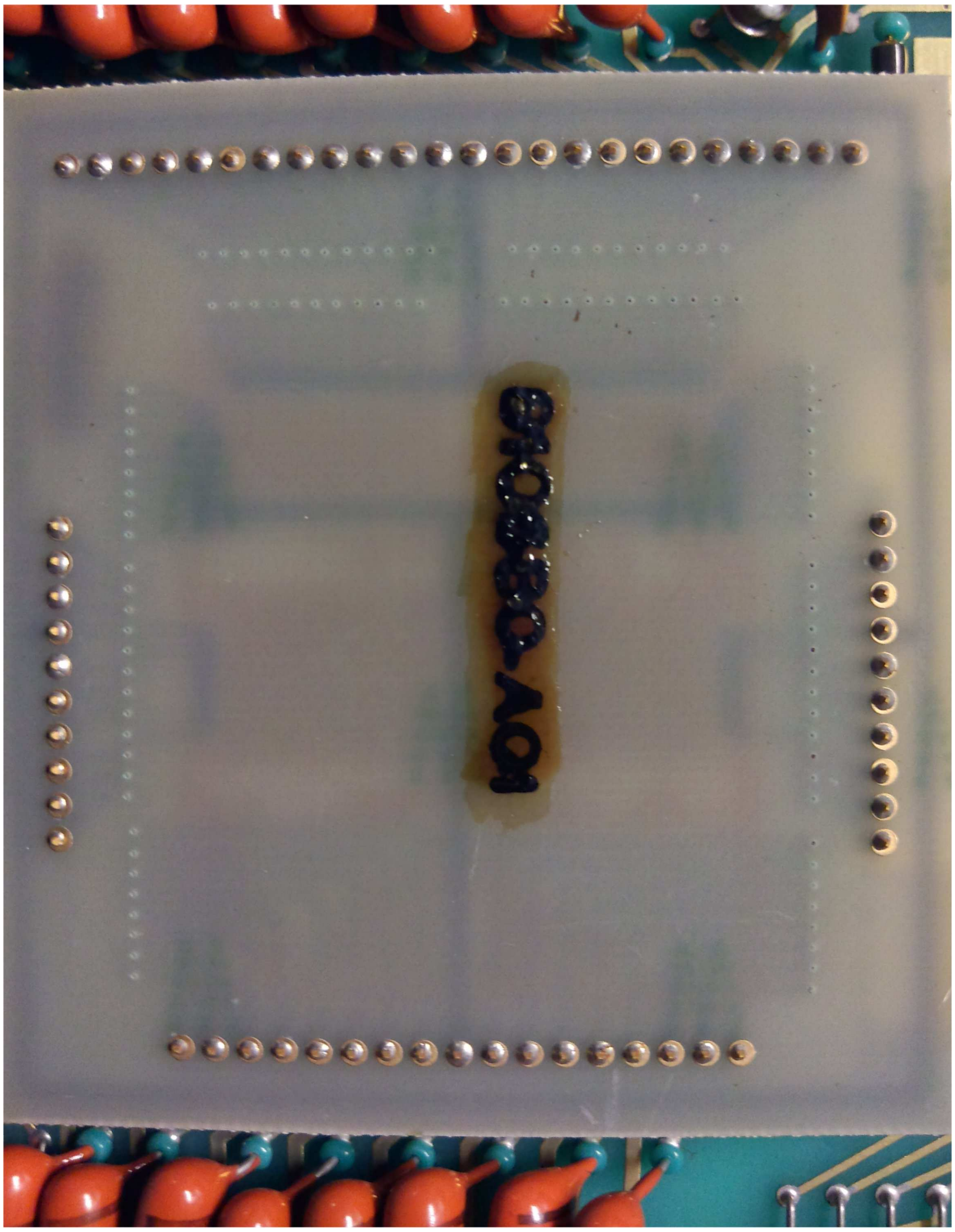

Figure 13: HP9100A core memory board 3 closeup
Label: 810680-A01

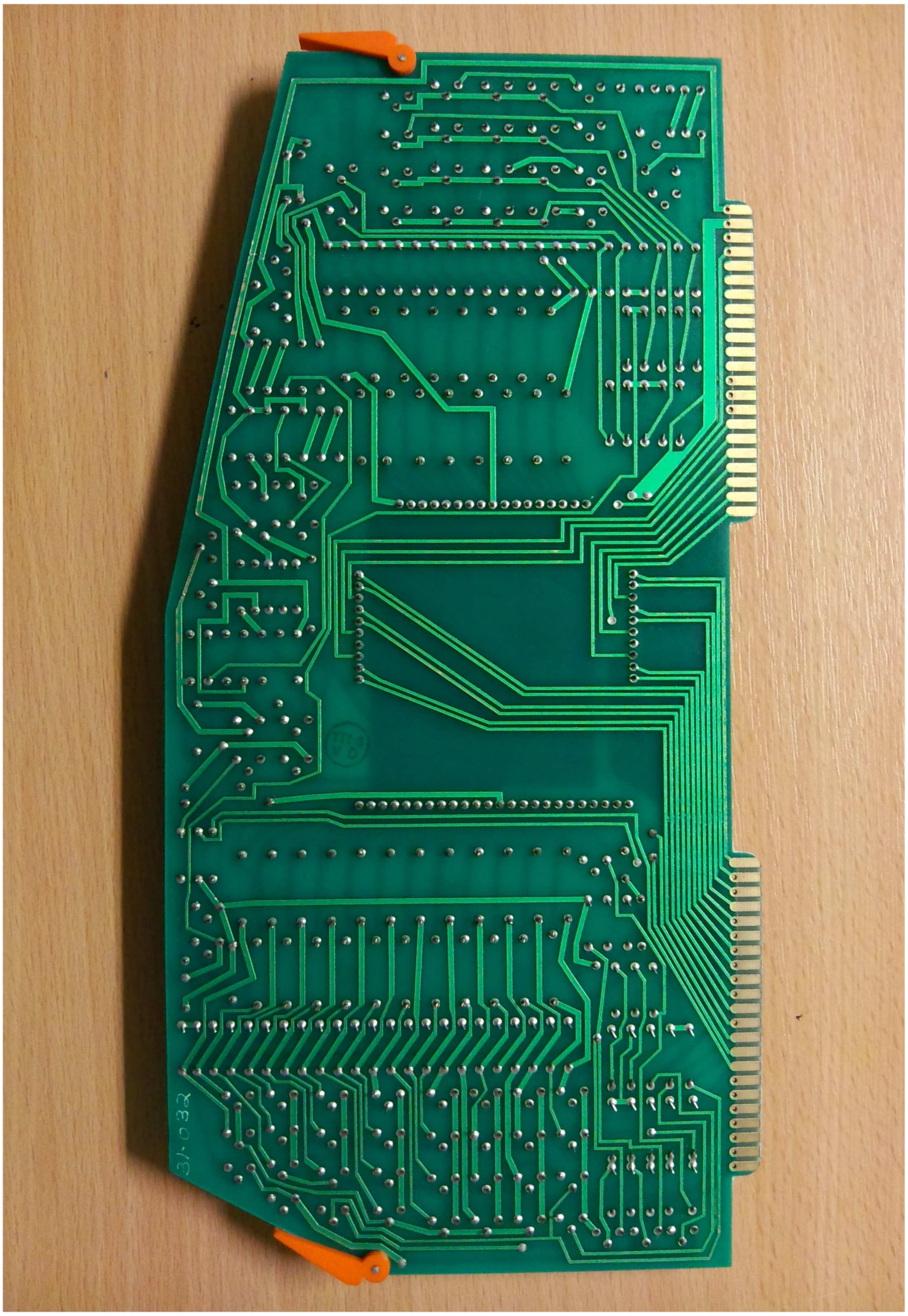

Figure 14: HP9100A core memory board 3 rear view

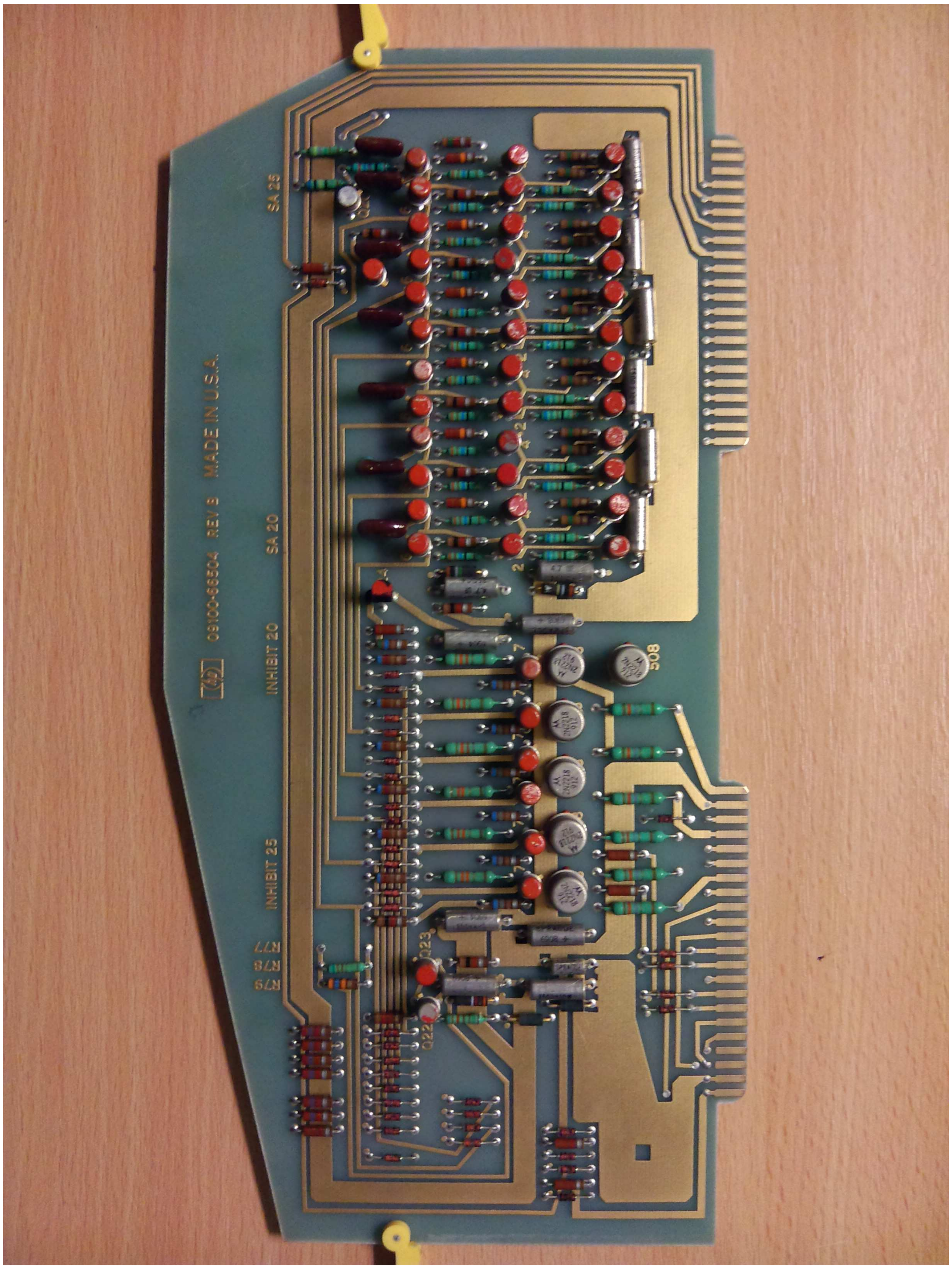

Figure 15: HP9100A core memory sense and inhibit circuits board 4 top view

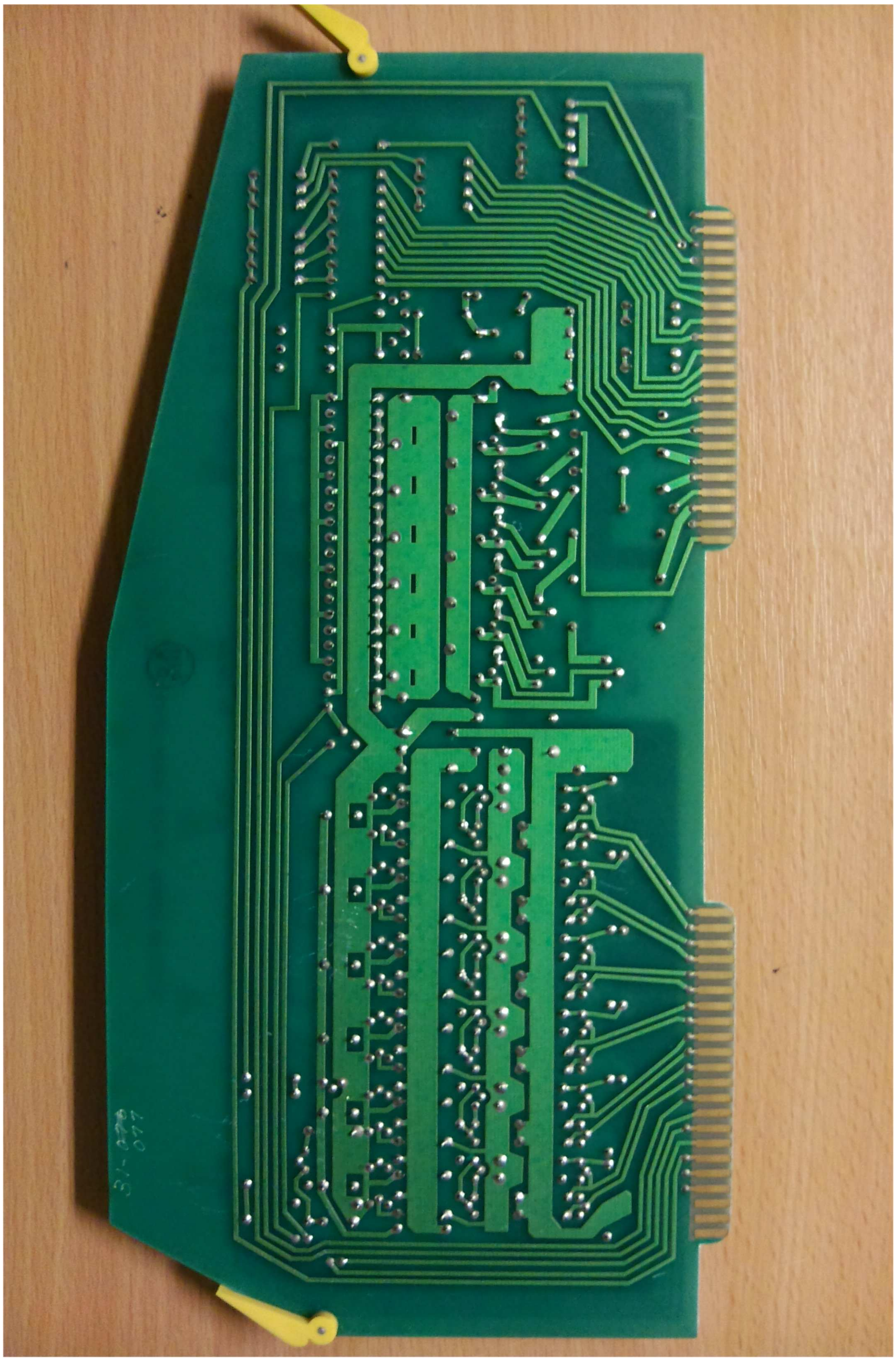
Figure 16: HP9100A core memory sense and inhibit circuits board 4 rear view

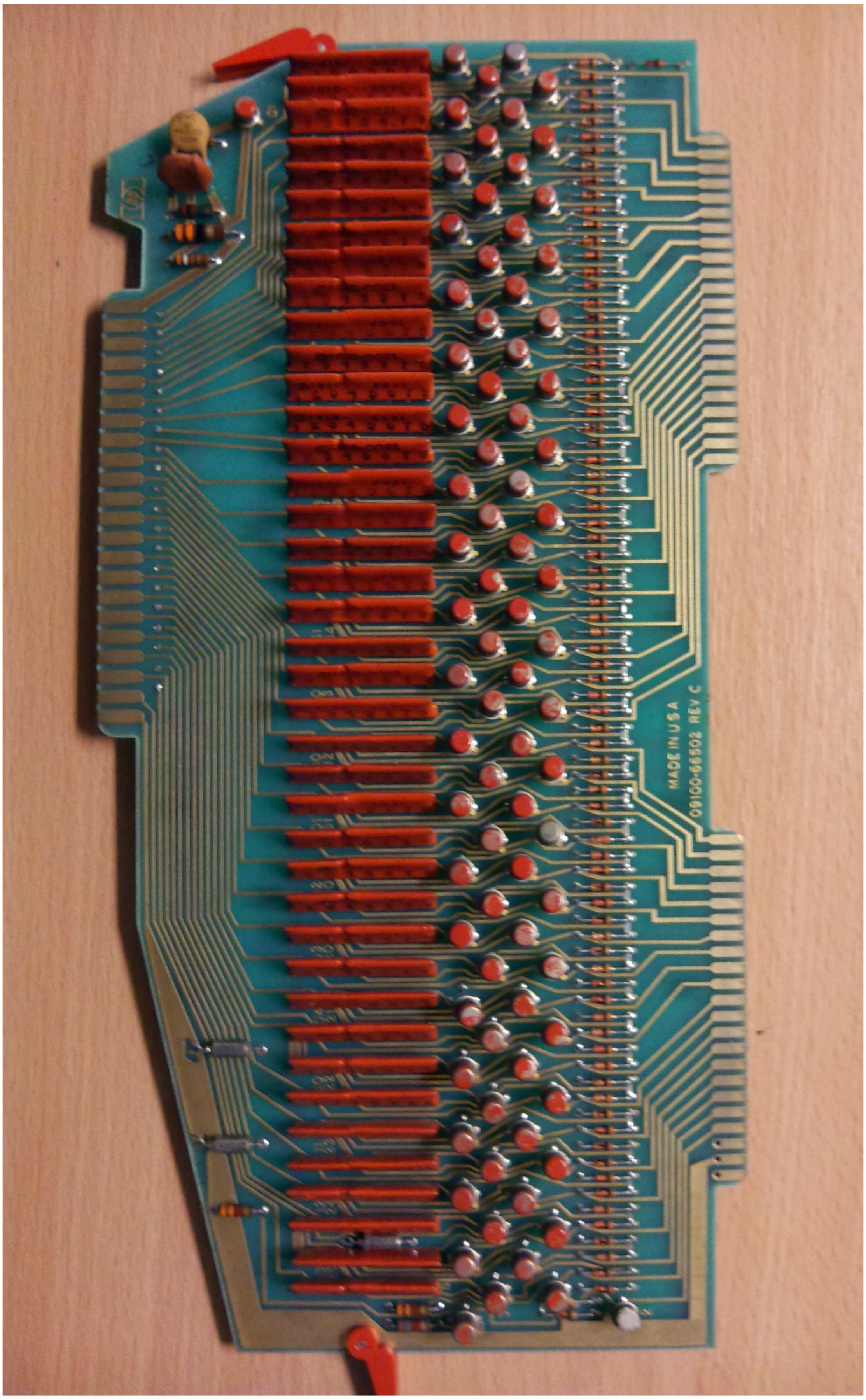

Figure 17: HP9100A flip-flop board 5 top view

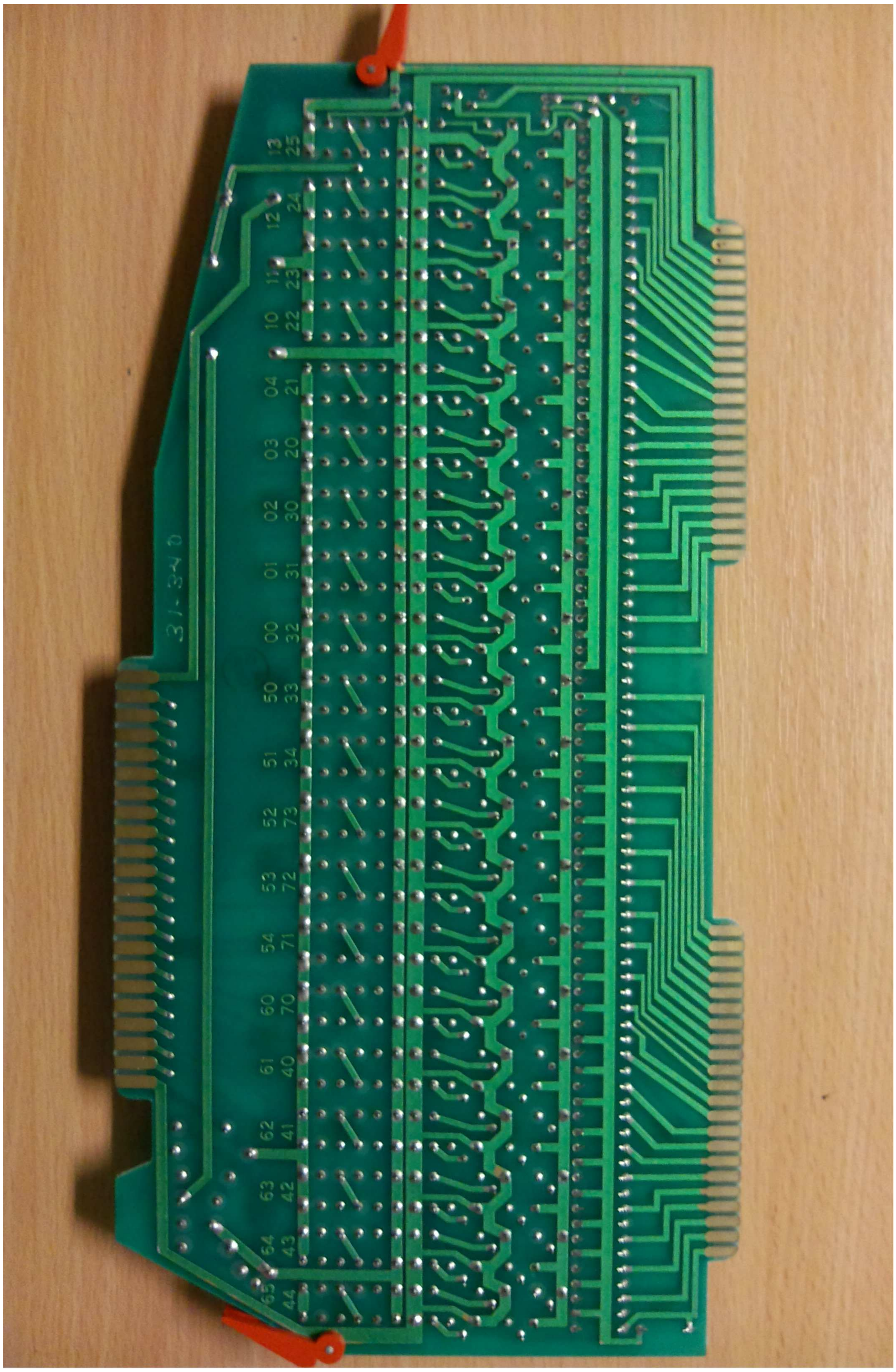
Figure 18: HP9100A flip-flop board 5 rear view

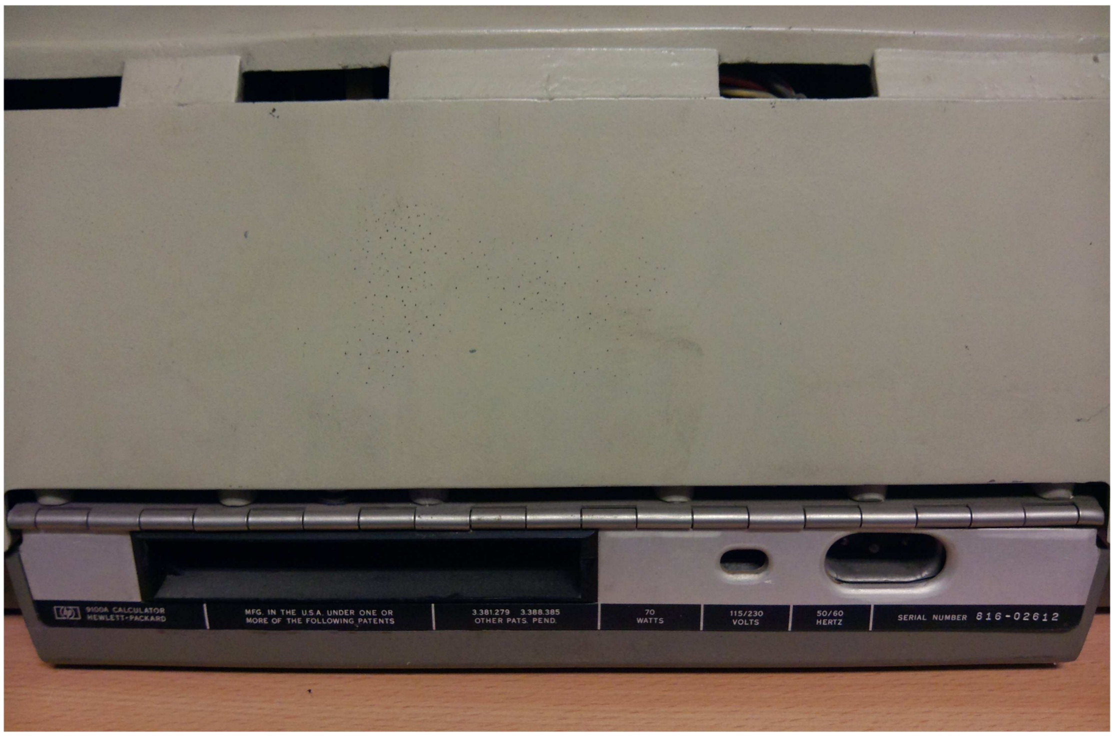

Figure 19: HP9100A rear view

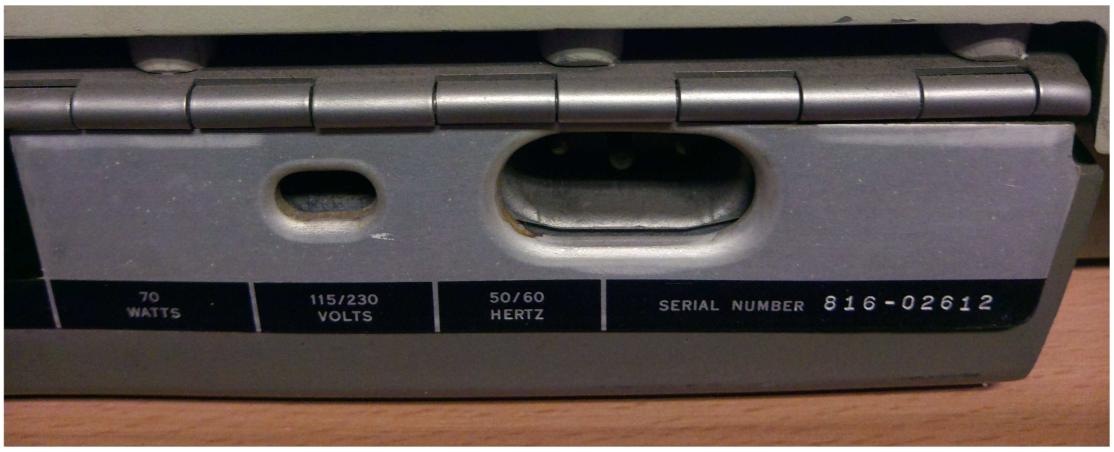

Figure 20: HP9100A manufacturing label
S/N: 816-02612