Notes on HP’s Classic Calculators

T Nixon 2019
Light Emitting Diodes (LEDs) are a great and easy way to display numerical or alphanumerical information.

LEDs require current passing through them to create light. This current is usually in the 10 to 20 millamp (mA) range and when this current flows, the LEDs will develop a voltage drop across them. For red LEDs, this voltage is about 1.7 volts. Most circuits that have LEDs are powered by a voltage higher than this and if you try to drive a LED directly from this higher voltage it will be damaged from too much current flowing through it. One common method to reduce the current is to place a resistor in series with the LED.

If we decide that the LED requires 20mA to be bright enough for an application powered by a 3.75 volt supply then by Ohms Law we can decide the resistor value. If the LED drops 1.7 volts, then the resistor will have 2.05 volts across it. (3.75 – 1.7 = 2.05) The current flowing through the resistor generates a small amount of heat which could be considered a waste of energy.

\[
R = \frac{V}{I} = \frac{2.05}{0.02} = 103 \text{ ohms}
\]

As you can see, this simple LED circuit consumes little power, but if you consider a 15 digit, 7 segment LED display which could have up to 87 LEDs on at any time then the energy consumed becomes a problem because the battery will not last long trying to drive that sort of load. (87 x 20mA is 1.74 amps)

It was found that supplying a short duration pulse of high current at regular intervals could also drive the LEDs at the required brightness. This current could be in the 100’s of milliamps, but is only for very short periods of time and does not cause any detrimental effects on the LED. In fact by using this method, the LEDs are mostly turned off. With some clever electronic design, a multiple LED display could be driven like this and is how the Classic display works.

It takes 280\mu S (millions of a second) to refresh the display and out of this, each LED is on for about 5\mu S. In other words, each LED is turned on for approximately 1/56th of the display refresh time. The refresh process that controls which LEDs are on or off happens so fast that the human eye cannot detect any flicker. The result is a bright display with a much smaller amount of power being used to drive it.

Further power savings were realised by charging small inductors and then discharging them into the LEDs to light them. This meant that the series resistors were not required and thus the power they consumed was eliminated. Another feature is that the displays are powered directly from the battery which means the power supply circuit does not have to be beefed up enough to supply the extra power.

Two specialist integrated circuits were developed to drive the display in this manner. One is called the Anode Driver and the other is the Cathode Driver. These match the connections to the LEDs which also have an anode and a cathode.

The cathode driver sequentially activates each of the 15 display digits one at a time. It can do this because the cathodes of each display LED are connected together. The anode driver then sequentially turns on each of the appropriate LED anodes for that digit. Each digit is active for 20\mu S and therefore a full display refresh takes 280\mu S. (But 15 digits = 300\mu S ???) This anomaly is due to the fact that while the decimal point is displayed in a separate digit, it is actually lit up during the same 20\mu S for the digit that it follows.
The Arithmetic & Register Circuit (ARC) has 5 data lines connected to the anode driver. These lines labelled A, B, C, D and E transfer partially decoded display information. The anode driver fully decodes this information into eight output lines which connect to the 7 LED segments of each display and one to the decimal point. The reason for the partial decoding is to help reduce power consumption and probably simpler design.

The quad inductor packages have different colours because of the decimal point LED. The 7 LED segments of each digit require one 130uH inductor each to store the charge. The decimal point LED being lit for half the time of the others, only requires a 68uH inductor. Therefore one quad package has 4 x 130uH inductors and the other has 3 x 130uH and 1 x 68uH inductors*. 

The quad inductor packages have different colours because of the decimal point LED. The 7 LED segments of each digit require one 130uH inductor each to store the charge. The decimal point LED being lit for half the time of the others, only requires a 68uH inductor. Therefore one quad package has 4 x 130uH inductors and the other has 3 x 130uH and 1 x 68uH inductors*.
ELECTRICAL PATHS FOR THE LED DISPLAYS

In the following diagram, the anode switch for LED segment (a) is closed. The cathode driver has selected Digit 1 to display information. The inductor (L1) charges from the battery, through the anode switch and back to battery again. The charge process is allowed to continue for 2.5μS for LED segments a – g, and 1.25μS for the decimal point.

After the time interval, the anode switch opens and the cathode switch stays closed. The inductor stops charging and now discharges through the cathode transistor and the LED segment which briefly lights it up. The anode switches open and close at spaced intervals of 1.25μS. The LEDs are diodes so they help to isolate each inductor from discharging into each other. The discharge time is about 5μS for segments a – g and 2.5 seconds for the decimal point. The inductors have to be discharged before the cathode driver selects the next digit or those LEDs may light briefly causing a bleed effect from one digit to the next. There is not much time available for the decimal point to discharge before the following digit needs to light which explains the faster timing requirement.

If the cathode driver is faulty and the cathode switch is not turned ON for a digit, then when the anode switch opens, the inductor discharge current has nowhere to go. That current will still try to flow and most likely it will push through the junctions of the just turned off anode driver transistor. Eventually, if not straight away, the transistor will be damaged and the corresponding display segment may stay on permanently or not come on at all.
The anode driver provides the master clock pulses (Ø1 and Ø2) for all the calculator circuits, including the ARC chip. The ARC chip is responsible for partially decoding the required display information and sends it the anode driver to decode it fully for the 7 segment displays. However, the ARC chip and the anode driver don't have a complete display timing reference between each other.

<table>
<thead>
<tr>
<th>T1</th>
<th>T2</th>
<th>T3</th>
<th>T4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Q2</td>
<td>Q3</td>
<td>Q4</td>
</tr>
<tr>
<td>Ø1</td>
<td>Ø2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The timing diagram shows Q1 and Q2 repeat the same sequence during the T1 – T4 phases, but Q3 and Q4 have different bit patterns. These are the ones that could be out of sync with the ARC chip. If the T1 – T4 cycles are not synchronised, the anode driver cannot decode the patchy information and the display will show garbage. The method chosen to do the synchronisation is quite clever as it requires no extra data lines and is quite transparent to the display operation.

When the B and D lines from the ARC chip are logic HI and Ø1 is logic LO, the anode driver circuit detects this and sets Q3 HI and Q4 LO before the rising edge of the Ø1 pulse at T3. This then puts those clock lines in the correct logic state during the T2 cycle. The dotted lines on the timing diagram reflect this. Now if nothing changes that timing should stay synchronised until power off, but if there is a "glitch in the matrix", then the display will briefly show garbage until the timing is reset automatically during digit updates. The reset will occur on any digit that requires the B and D lines to go HI. ie. Digits 0, 2, 3, 8, 9 and ‘d’. (As in HP-65 “Crd”)

At switch on the display on the Classics always has a [0] displayed so I assume then, that on the very first display refresh the digits will most likely show garbage until that first [0] is decoded, however this will happen too fast for the eye to see. The diagram shows the digital reset path when B and D go HI. Flip flops B1, B3 and B4 receive the reset pulse. Red represents Logic HI, green represents Logic LO.
The HP-45 patent document (4,001,569) shows an image of the required data for the anode driver to decode the digit ‘9’. The image on the right shows the corresponding trace from an oscilloscope. The time base is 10uS. The yellow trace is set to 2V/div.

The first digit to be sent to the anode driver from the ARC is the mantissa sign, followed by the exponent ones, exponent tens, exponent sign, and then the mantissa digits 10 down to 1. The first mantissa digit (9 in this case) is the last digit to be sent to the anode driver before the cathode driver is reset to start the next display scan. This reset is caused by the falling edge of the RCD pulse.

This next trace shows the voltage appearing on the anode driver pins for LED segments A and B while the LED display is showing 0.00. The timing discrepancy is due to the calculator RC timing which can vary due to temperature, component tolerances and aging effects. (20us designed vs 22us actual)
After the power is turned on, these images show a Classic display showing 0.00, plus the trace on an oscilloscope as it monitors the A B C D and E inputs to the anode driver, and the Reset Cathode Display (RCD) line from the ARC to the cathode driver.
As mentioned, not all LED segments are lit at any one time, however the display refresh process is so fast that the eye only sees whatever is meant to be displayed. The following single digit \( .8 \) sequence with a display showing \( .823 \) takes about 20\( \mu \)S, including the decimal point. By staggering the time that each LED turns on, the current inrush that would occur if all segments were turned on together is reduced. Notice that only 1 digit is active at any one time and the LEDs in all the other digits are turned off. They will be excited in turn as the cathode driver sweeps across the display.
The schematic of the cathode driver chip shows that on reset, digit 15 is first out. This is the Mantissa Sign.

This logic trace shows the A – E pulses, Ø1, Ø2 and the anode to cathode driver step signal. (C. CL) The time base is 10μS. The yellow line is the RCD pulse - 5V/div. The display is showing 0.00.
This next trace shows the 56 clock pulses between each of the RCD pulses. This represents one Word (or instruction) time for the classic calculator and takes 280uS.

First out is Mantissa Sign (20uS)                Last out is Mantissa Digit 1 (20uS)

The time base is 50uS. The yellow trace is set to 2V/div, the purple trace is set to 5V/div.

See if you can decipher the digits shown on the display 😊

Hint, if you can count to 10 you are on the right track.
The Classic display driver LEDs are connected together in a multiplexed method and the Anode and Cathode drivers are responsible for sending the display information to the correct LED segment.

The Anode driver, as discussed, decodes the five display lines into 7 segment information and outputs that data to the LED anodes. The Cathode driver is responsible for selecting the correct LED digit and does this by sequentially enabling each LED digit in turn. At the start of a display sequence, the RCD line is cycled to reset the Cathode driver to start a new display sequence.

Fourteen pulses are sent to the Cathode driver from the Anode driver every display sequence. One RCD pulse is sent from the ARC on the main CPU board, for every display sequence. You might notice that the display refresh for this calculator is 310µS, not the specified 280µS. This is most likely due to changes over time with the oscillator components. The following image shows this process. The horizontal scale is 50µS/Div. The vertical scale is 2V/Div.

Note that in this image, there are no decimal points being shown on the display. The calculator is a HP-65 and the PRGM/RUN switch has been placed in the PRGM mode with the display showing 00 00.

When the PRGM/RUN switch is placed in the RUN position with the display showing 0.00, you can now see the decimal point has been decoded and the extra step has been added by the Anode driver.
The next image is a close up of the relationship between the RCD pulse and the first step pulse. The horizontal scale is 5uS/Div. The vertical scale is 2V/Div.

At the hardware level, the decimal point will be included on the display when any of the Register B digits have bit 2 set to 1. That is, any digit 2, 3, 6 or 7. This is a snapshot of the HP-65 signals while running the small program listed below. While the program is running, the HP-65 internal microcode transfers much of the 2367236723 number to Register B. This scope trace shows that there are multiple decimal points lit on the display.

```
LBL A
  2367236723
  ENTER
  GTO A
```
HP-55 Oscillator waveform captured on the crystal connection to the anode driver

≈ 1.3μS = 748KHz

≈ 3V

Block Diagram of the Classic Series
There are 8 ROMs available in the HP-45, and each is selected by the instruction `select rom (x)` where `(x)` can only be a value between 0 and 7. On initial power up, ROM 0 is selected by default. That way the calculator will always start executing code from ROM 0, address 0. If the microcode needs to begin executing in a different ROM then appropriate `select rom` instructions will be executed. There are 256 instructions stored in each ROM, and that requires an 8 bit address be used to access them all. Each instruction is 10 bits wide. The ROM chips used in the HP-45 are called quad ROMs meaning that 4 ROMs are packaged in one chip.

Shown below is a nice image of the HP-35 ROM chip taken by Peter Monta. The red highlight shows the Gnd trace around the chip.

Each instruction requires 56 clock cycles to execute. This image taken from the HP-45 patent document shows the IA and the IS busses. The IA bus is used to transfer the instruction address which is used by the ROMs to get the next instruction. This instruction is then sent out from the active ROM on the IS bus. After the address is accepted by the ROM, it takes a few clock cycles to process the address and fetch the instruction before it can be sent out. You can see this delay in the diagram.
This image shows the various busses at switch on. The HP-45 patent document states that the first instruction that executes must be a JSB (Jump to Subroutine) instruction so that the address decoder circuits are initialised properly. You can see below that the IA line is low for most of the initialization. During this time the ROM address is continually set at 0 during clock pulses 19 to 26. The WS line is also low because there is no WS information encoded in JSB instructions.

The first 10 bit instruction is accessed here

Here is first line of microcode shown in the HP-45 patent document. The dots represent 0’s.

```
0 L00000: ..1.1.11.1  ->  L0053    PW01   JSB PW02
```

Looking at the instruction shown in the image we get 0010 (2) 1101 (B) 01 (JSB), in other words JSB $2B. As you can see the first instruction is indeed a JSB. During initialization, this single instruction executes continually. After initialization has completed the IA bus becomes active and the next address on this bus will be 2B hex which came from the JSB instruction.

<table>
<thead>
<tr>
<th>ROM ADDR</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>pwo1: jsb pwo23</td>
</tr>
<tr>
<td>$002B</td>
<td>pwo2: go to pwo3</td>
</tr>
</tbody>
</table>

The image below starts from the 2nd instruction (marked above) and shows the next address on the IA bus is 2B hex.
Next we see the first four instruction execution sequences. Instruction 0 (jsb pwo2) has already executed. The yellow bars indicate the 56 clock cycles for each instruction.

<table>
<thead>
<tr>
<th>ROM ADDR</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000</td>
<td>pwo1: jsb pwo2</td>
</tr>
<tr>
<td>$002B</td>
<td>pwo2: go to pwo3</td>
</tr>
<tr>
<td>$0014</td>
<td>pwo3: select rom 5</td>
</tr>
<tr>
<td>$0515</td>
<td>pwo2z0: go to pwo2</td>
</tr>
</tbody>
</table>

The master clock is 784KHz divided by 4 which gives 196KHz. Divided by 56 bits per instruction, this is about 286uS per instruction, or about 3500 instructions per second. You can see that this particular calculator is executing instructions at around 320uS as shown by the 100uS scale graticule.

Notice that the Program Counter shown on the IA bus has incremented normally from $14 to $15 after the select rom 5 instruction, but now the code is being fetched from that address in ROM 5.

You can also see that an address is set and then the instruction there is fetched from ROM during a 56 bit cycle, but that instruction is not executed until the next 56 bit cycle. In that way, one instruction is being executed while the next instruction is being fetched.

The logic analyser image below is showing the instruction flow for a HP-45 in the code loop that waits for a key press. This code is being executed from ROM 3 which would have been selected from the following instruction which is located in ROM 6 at address $FE.

This is the next instruction that executes which is a return from subroutine.

Some other instructions will be executed before finally ending up in this code loop which continually scans to see if a key has been pressed.
The ROM address is presented on the IA bus during the eight Ø1 and Ø2 clock cycles between 19 and 26.

The address data is clocked from the Ø1 signal and the IA data becomes \textbf{10001100} which is \textbf{8C} in hex.

Notice the IA bus only uses 8 bit addresses to access each of the 256 memory locations in the selected ROM.

The data on the WS bus depends on what part of a register needs to be acted on when those particular instructions are being executed. The WS information is embedded into the 10 bit instruction in bits 2, 3, and 4.

The storage format of each 56 bit register is shown below. In this case the C register is shown.

<table>
<thead>
<tr>
<th>Mantissa Sign</th>
<th>Mantissa</th>
<th>Exp Sign</th>
<th>Exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1234567890</td>
<td>912</td>
<td></td>
</tr>
</tbody>
</table>

(Decimal = -1.234567890 E-12)

The digits are Mantissa Sign (13) down to Exponent Units (0).

The P register is 4 bits wide and can hold a number from 0 to 15. It is used as a pointer to a single digit, or multiple digits, within a register. In microcode, it is also used as a delay counter. The maximum number that the P register holds depends on the calculator model. For the HP-45, the P register maximum value can be 15. As there are 14 digits in each register, numbered 0 to 13, when the P register is used as a digit pointer, its value will be limited to between 0 and 13. Here, it is set to 4. This can be accomplished by the \textbf{4 -> P} instruction.

P register

4

We will use a “clear register c” type of instruction here. This is written as \textbf{0 -> c}. The WS information is written into instructions that use it by enclosing the identifier inside brackets.

Instruction

\textbf{0 -> c [?] (? represents one of the WS types)}
Using the value above, clearing the C register using the different WS methods available results in these values.

<table>
<thead>
<tr>
<th>Type</th>
<th>Value</th>
<th>Use</th>
<th>Instruction</th>
<th>10 bit Binary 10 bit Binary 10 bit Binary 10 bit Binary 10 bit Binary</th>
<th>New C Register value</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>000</td>
<td>P digit</td>
<td>0-&gt;c[p]</td>
<td>001100000010</td>
<td>9 1234567800 912</td>
</tr>
<tr>
<td>ms</td>
<td>001</td>
<td>Mantissa</td>
<td>0-&gt;c[ms]</td>
<td>001100001110</td>
<td>9 0000000000 912</td>
</tr>
<tr>
<td>x</td>
<td>010</td>
<td>eXponent</td>
<td>0-&gt;c[x]</td>
<td>001100101010</td>
<td>9 1234567890 900</td>
</tr>
<tr>
<td>w</td>
<td>011</td>
<td>Word</td>
<td>0-&gt;c[w]</td>
<td>001100111010</td>
<td>0 0000000000 000</td>
</tr>
<tr>
<td>wp</td>
<td>100</td>
<td>Digit 0 up to P</td>
<td>0-&gt;c[wp]</td>
<td>001101001010</td>
<td>9 1234567800 000</td>
</tr>
<tr>
<td>ms</td>
<td>101</td>
<td>Mantissa + Sign</td>
<td>0-&gt;c[ms]</td>
<td>001101011010</td>
<td>0 0000000000 912</td>
</tr>
<tr>
<td>xs</td>
<td>110</td>
<td>eXponent Sign</td>
<td>0-&gt;c[xs]</td>
<td>001101111010</td>
<td>9 1234567890 012</td>
</tr>
<tr>
<td>s</td>
<td>111</td>
<td>Sign</td>
<td>0-&gt;c[s]</td>
<td>001101111010</td>
<td>0 1234567890 912</td>
</tr>
</tbody>
</table>

The fifth instruction that executes from switch on starts to make use of the WS bus. This part of the microcode is preparing to clear the HP-45 memory to make sure the 10 available storage registers are all set to zero.

There are 56 clock cycles for each instruction.
There are 14 digits in each register.
Therefore, there are (56 / 14 = 4) clock cycles used to process each digit.

The WS bus will cycle 4 times for each digit that is included in the [WS] part of the instruction and will stay logic low for those digits that are not. This image, taken from the HP-45 patent document, shows how the WS bus interacts with the 56 clock cycles.
You can see while $0 \rightarrow c[w]$ is executing the WS bus is cycling 4 times for each of the 14 register digits and while $0 - 1 \rightarrow c[s]$ is executing only the last 4 cycles are active for the mantissa sign digit.

The P register is located inside the Arithmetic and Control chip (ARC). As it has a role to play in the word select function, it is responsible for providing the signals for the WS bus when the P register is used to determine which digits are to be acted on.

The other types of word select functions are controlled from an unlikely place. The WS bus signals for these come from circuitry from inside the ROM chips. It makes sense to let the ROM chips produce the WS signals because they are where the instructions are fetched from. As the instructions are read from ROM memory they are checked for WS information and if so, the ROM chip will output the required information onto the WS bus. In the logic circuit diagrams below you can see that the WS bus signals exit from each of these chips.
The code shown below is part of the clearing code for the ten memory registers and executes just after switch on. You can see that the WS bus activity coincides with the instruction flow.

```
$05EE   a exchange c[w]
$05EF   c -> stack
$05F0   c -> data
$05F1   a exchange c[w]
$05F2   c + 1 -> c[p]
$05F3   c + 1 -> c[p]
$05F4   if no carry go to clr3
$05EC   clr3:  c - 1 -> c[p]
$05ED   c -> data address
$05EE   a exchange c[w]
$05EF   c -> stack
```

This code repeats until the ten registers are cleared.
The card reader relies on mechanical switches for its operation. When the card is inserted into the slot, the first switch to close is the motor switch (MTRS). This alerts the Card Reader Controller (CRC) chip that a card is present and to start reading or writing. If the PRGM/RUN switch is in the PRGM position, the CRC will start a write process, or if the switch is in the RUN position, it will start a read process.

The switches are activated by the card rubbing over small nylon balls which rest against spring contacts. The card presses down on the ball forcing the contact to connect to the circuit board pads beneath them. This effectively shorts the contact to GND.

There are three switches listed in order of operation.

- **MTRS** Motor Switch: Alerts the CRC that a card is present in the slot and to start the motor.
- **HDS** Head Switch: Alerts the CRC that the card is sufficiently in the slot to decide what process to begin, either read or write.
- **WPS** Write Protect Switch: Alerts the CRC of the magnetic card is write protect status.

A new magnetic card has square ends on it. If the user wants to write protect the card, then the corner that is inserted first can have one corner cut as shown below. Normally the HDS and the WPS will short to GND together. However, if the card corner is cut, the WPS operation is delayed momentarily because the card edge takes a tiny bit longer to reach the ball. The CRC detects that the WPS was not connected to GND when the HDS connected to GND and thus determines that the card is write protected. In this situation, even though the card passes through the calculator, the write process is inhibited.
The trace below shows a small program being written for a HP-65 to a non write protected card. The PRGM/RUN switch is in the PRGM position.

When the card entered the slot, the MTRS switch closed pulling the MTRS line LO. The CRC chip detected the change and turned on the motor to pull the card through the calculator. The WPS switch then closed and pulled the WPS line LO. A short time later the HDS switch closed and pulled the HDS line LO. Because the WPS line was LO when this happened, the CRC chip knows the card is not write protected and it then set the WE line HI. This condition enables the card write circuitry to become active in the sense chip.

As the card passes over the R/W head the CRC send the program data bits to the sense chip on the WA and WB lines. After the 600 data bits are written to the card, the CRC tests for the MTRS to open again. This will happen after the card end passes the nylon ball and releases the MTRS switch allowing the line to return to a HI state. The WE line was released back into a HI state turning off the sense chip write circuits. The WA and WB lines also went HI at this point. Soon after, the card end will pass by the WPS nylon ball and the WPS line will go HI, shortly followed by the HDS line. You will notice that both these lines appeared to go HI simultaneously. This is because the card was quickly removed from the card slot after the motor stopped and the scope could not see the difference in the switches working. If the card is not manually removed from the slot after the motor stops, then the HDS line will stay LO.

If there are any switches that do not operate or they operate outside time limits determined by the CRC circuitry, then the motor will be turned off and an error will be flagged to the ARC chip which will make the LED display flash.

The card motor was on for about 2 seconds. You can see that there is still some area behind the WB data that could be used to store more information. This data could still be written until just before the CRC chip put the WE line back to LO. This would indicate that the card is moving a little bit slow through the reader resulting in data that is bunched up a bit.

If the card is write protected by cutting the card corner off, then it takes a little bit longer for the WPS to be operated by the nylon ball. This means the HDS will operate first and bring the HDS line LO before the WPS line. The CRC will interpret this condition as a write protected card and will not put the WE line HI. Even if the 600 data bits were transferred to the sense chip, the data will not be written to the card.
If the PRGM/RUN switch is in the RUN position when the card is placed in the slot, the CRC will initiate a card read process. Regardless of whether the card is write protected or not, the WE line will remain LO disabling the sense chip write circuitry and allow the data to be read from the magnetic strip on the card. The sense chip will amplify and condition the signals from the read/write head and send the data out on the RA and RB lines as shown below. All other switch signals work as in the write process.

This card was recorded with the following small program. Each key press is saved as a 6 bit value.

<table>
<thead>
<tr>
<th>#</th>
<th>Key</th>
<th>Value</th>
<th>6 Bit Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>000100</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>3</td>
<td>000011</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>000010</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>20</td>
<td>010100</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>19</td>
<td>010011</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>18</td>
<td>010010</td>
</tr>
<tr>
<td>7</td>
<td>ENTER</td>
<td>62</td>
<td>111110</td>
</tr>
<tr>
<td>8</td>
<td>R/S</td>
<td>34</td>
<td>100010</td>
</tr>
</tbody>
</table>

The data bits that were read from the card are expanded below.

The data is stored Least Significant Bit (LSB) first which is why it looks opposite to the 6 bit binary above. All unused program codes are set to the 6 bit NOP value of – 000000.
The card reader circuit for the HP-65 and HP-67.

This trace shows the steady state voltage at the read write head. It shows some power supply hash which is about +/- 200mV.
This trace is showing a HP-65 write operation looking at the logic data coming into the WA pin and the AC data being fed into the write head from the sense chip. The vertical scale is 2V/Div. The horizontal scale is 1mS per division showing that the data pulses are about 1.2mS in duration.

This trace is showing a HP-65 read operation looking at the logic data coming out of the RA pin and the AC data coming from the read/write head to the sense chip. The HAC and HA vertical scale is 0.5 V/Div and the RA vertical scale is 2V/Div.

Relative to ground, the negative switching threshold appears to be about 1.3V and the positive switching threshold appears to be about 1.5V. This equates to a 200mV signal.
This trace shows the write enable input (WE) in blue with the CA pin. The initial pulses on the CA pin correspond to the program data pulses being written to the card. Most of the program memory is empty.

This is a trace of the motor voltage when reading a card is about 2.4V.

This trace is looking at the voltage across the 4K64 trim resistor connected to the sense chip pin 2. It changes from 0V to about 0.5V and shows some power supply hash superimposed on top.
The material used to grip the cards and enable the drive roller to pull them through the reader does its job quite well. However, over time it deteriorates into a messy blob of goo and the card reader stops working as a result.

There are a few ways to repair this problem but this discussion will be about using easily available o-rings.

There is plenty of information on how to do this type of repair on the web so it is not worth re-inventing the wheel here.

However, one thing is worth mentioning and that is the o-rings sometimes have a small lip on the outer circumference of the ring and is the result of the manufacturing process. Unfortunately this widens the diameter of the ring slightly and can cause the o-ring to put too much pressure on the little nylon idler roller. This little piece of the reader is quite important as it provides a rolling surface for the card to glide over while the drive roller moves the card. If the drive roller diameter is too wide, as in using an oversized o-ring, then too much pressure will be placed on the card and rollers as it is squeezed between these two parts.

The o-ring outer circumference can be cleaned up using the following simple method and should guarantee roundness of the parts when complete. Slide two o-rings on to the shaft of a 9/32” drill bit. Push them on from the end that goes into the drill chuck, not along the flutes. Place the drill into the chuck and tighten it so that the o-rings are about ½ inch from the lower chuck jaws. Then using a moderate drill speed, hold a flat file at right angles to the side of the o-rings and apply some pressure to trim the outer surface down. Move the file against the direction of the drill, then move the file out and away then back and repeat.
This process should be undertaken on a trial and error basis so that when the O-ring are fitted to the drive wheel, without glue, the drive wheel starts to spin freely or with a tiny amount of friction. The drive pin should be at the minimum pressure position (as explained below) while doing this test. Now when the card is inserted into the reader it should fit nicely when travelling through. Also, as the pressure adjustment was set at the minimum, there is plenty of adjustment available if the card slips a bit.

The drive pin that the drive roller spins on is supposed to be eccentric so that there is a small amount of adjustment that can be made to alter the pressure or “grip” on the card as it passes through the reader. After repairing a few, it struck me that I could not figure out how this worked. The drive pin looks like a normal straight pin shaft so turning it would not affect the positioning of the drive roller. Even if it was offset, turning it would result in the shaft end rotating in an eccentric manner, not the shaft itself.

On closer inspection however, one end of the shaft has a smaller diameter which mates with a support hole on the inner surface of the card reader moulding. It is this tiny part of the shaft that is offset slightly and provides the eccentricity and in effect changes the angle of the shaft slightly between the two support holes.

Looking at the position of the offset in Fig A you can see that it is towards the bottom of the shaft. When in this position, the drive wheel will have less pressure applied to the card and idler wheel. This is where it should be if the O-rings need trimming.

In Fig B you can see that the screw driver adjustment slot is vertical and the position indicator is facing forward. This is how it would look in the HP-65 and 67 while looking down at the card reader.

In the HP-65 and HP-67, with the position indicator pointing towards the sense chip, the lightest pressure is applied. If the protrusion is towards the DC motor, then the maximum pressure is applied. There is a procedure for adjusting this pressure in the HP-97 service manual which is freely available on the web.
Note that when the two o-rings are fitted, the sides of the rings should not extend past the end of the drive wheel shaft. This will cause the drive wheel to be too tight when it is placed in position in the card reader frame. If this happens the sides of the o-rings can be wiped against the flat edge of a file to flatten them a tiny bit. If the o-rings are pressed too hard against the flat edge of the drive wheel, then they can be squashed. If this happens, the outer diameter can increase in size and add to the drive pressure.

Super glue works well to secure the o-rings to the drive wheel. It only needs to be applied to the flat surface of the drive wheel as indicated by the green arrow. Use the glue sparingly. A drop of glue can be placed on a piece of paper and a small pin can be used to scoop some up and apply to the wheel surface. Once the o-ring has been slid into place, a similar amount of glue can be applied to the side of that o-ring and the next one put in place.

The other item that often needs attention after years in service is the coupler that connects the DC motor shaft to the helical gear. This part looks like a small aluminium cylinder between the motor and the helical gear. The inside of the coupler is made out of a material that deteriorates over time and crumbles and in doing so can no longer work in the way intended. With the motor removed from the card reader frame, you should be able to move the helical gear back and forth a small amount. If you gently pull the helical gear away from the motor hopefully it will stay in place. If it slides off the motor shaft easily, then a repair is required. If left in this condition, the motor shaft will spin inside the coupler and the card will travel too slowly through the reader or not at all.

There are a few methods mentioned on the web to fix this issue. The method described here is quite easy and cheap to do. Note before disassembly that the helical gear has some sideways movement.

Note: The card reader has many small parts that can disappear into oblivion if not handled with care. Try to keep your fingers away from the circuit board and its components.

All you need is the following...

- A small length of 75 ohm RG-58 coax
- Stanley (box cutter) blade
- Side cutters
- Pliers
- 5/64 drill bit
- Bench drill
- Small piece of timber about ½ inch thin
- Super glue

This procedure may sound long winded on paper, but in practice is easy to do. Please read through all of the notes before trying it out. It assumes the DC motor has already been removed from the card reader.

Start off by removing the coupler from the helical gear. It might be a bit tight so gentle twisting pressure from some pliers may help. Clean up any remaining coupler material from the helical gear and wipe with Isopropyl alcohol or other cleaning agent. Keep the aluminium cover as it will be used later on. Put the helical gear and motor screws somewhere safe.
Strip about 3 inches of the outer sheath and underlying wire braid from the RG-58 cable. There will be a translucent inner core remaining with a copper wire in the centre.

Using some side cutters, cut into the translucent material and down to, but not through, the copper wire. The idea is to remove some of the sheath from the copper wire.

Use the box cutter to trim one end so it has a clean edge. This will be the end of the sheath that you need to drill into.

The sheath needs to be centre drilled using the 9/64 drill bit and will ensure that it fits snugly over the end of the helical gear shaft. The hole needs to be drilled as close as possible to the centre of the sheath and ideally this job would be performed using a lathe. Unfortunately not many people have access to one of these so it can be done using a bench drill. If this is not available either, do not attempt to do the drilling by hand. If the hole is off centre, even by a tiny bit, the shaft will vibrate in use and may stop the card reader from working properly and could also damage the motor bearings.

Put the drill in the drill press chuck leaving about 1 ½ inches exposed. Move the drill press table so that it is about 2 inches below the drill tip. The table usually has a hole in the middle so make sure the drill can pass through this. Tighten the table securely so it won’t move.

Secure the piece of timber to the drill table. Using one or two bolts through it and the table is preferable to using clamps.

Drill a 9/64 hole straight through the timber. While the drill is spinning, you should move the chuck up and down a few times to make sure the hole is nice and clean. Turn the drill off.

From this point on, the table and timber cannot move in relation to the drill centre line. If it does, even a tiny bit, you will have to start the procedure over from scratch.

Undo the chuck and open it wide enough so the drill can be removed without upsetting anything.

Turn the drill tip side upwards and push it up through the hole from underneath the timber. It will be a tight fit (which is good) so don’t be tempted to use a hammer or similar to tap it in as this could move the table or timber. You can push the drill upwards using the side face of some pliers or grip the drill shaft with the pliers being careful not to bend or break it.

Insert the translucent sheath into the chuck with the trimmed edge facing towards the drill. Tighten the chuck but only finger tight. It should be tight enough so the sheath won’t slip but won’t be crushed also.
Start the drill and move the chuck down so that the drill bit penetrates about ¼ inch into the sheath and bring the drill back out. If all is well you should have a nicely centred hole in the sheath.

Remove the sheath from the chuck and using the box cutter blade, cut of the end that was drilled to a length of about ½ inch.

If you want to, you can now repeat the process and make a few more coupler parts for spares.

You should be able to see how far the drill bit went into the coupler. Place the coupler beside the helical gear and line it up so the bottom of the drill hole lines up with the end of the helical gear. Place the box cutter blade at right angles across the coupler at the point where it lines up with the shoulder.

Once this has been cut to match, line up the coupler with the original aluminium retainer sheath and trim the end of the coupler to match its length.
If you make a mistake cutting the new coupler to size, use another of the spares and start again. If the drill is still set up it will be easy to make some more couplers.

Without glue, put the coupler onto the helical gear end and push the coupler all the way onto the motor shaft. You should notice that the motor shaft has some back and forth movement. This is ok.

Insert the motor back into the card reader frame and make sure the tip on the right side of the helical gear is aligned to the bearing end in the card reader frame. Insert and tighten the two motor screws just enough to hold the motor in place. You will probably find that the helical gear will not move back and forth now. You can use some needle nose pliers or a flat bladed screw driver to gently push the helical gear and coupler closer to the motor.

Keep tightening the motor screws and moving the coupler making sure the helical gear always has some back and forth movement. If not by the time the motor is fully tightened, you will have to remove the motor, trim the motor side of the couple a bit more and then reassemble. If this shaft has no side play because the coupler is too long then the motor shaft will be pushed too hard into the motor casing and could damage the bearings and other motor internals.

When satisfied that the coupler is ok, disassemble it and place a small amount of super glue around the helical gear drive end being careful not to put too much on. Overdoing the glue may cause the glue to dribble into the centre hole or out over the gear.

Insert the coupler onto the helical gear.

Place the gear and coupler onto a flat surface and roll it with light pressure along the length of your finger. This will help centre the coupler onto the shaft. Leave it alone now while the glue sets.

**Note:** There is no need to glue the coupler to the motor shaft.

When the glue has cured properly slide the coupler all the way onto the motor shaft as discussed previously and reassemble the motor and tighten the screws, again making sure there is a small amount of side play.

When fiddling with a disassembled card reader it is possible that some of the wiring gets stressed from all the movement. This is especially so where wires are soldered to the circuit boards. These cables are made up of many tiny wires which can individually break. This may limit the amount of current that can pass through them and could upset the calculator operation. Check the wiring at these points to make sure some or all of the wires have not broken. If any have you will need to re-solder the connection.
Idler Roller Dimensions
Emulator Card File Format

The card files are text files but have a simple (encoding) method and this was used on some other programs I have done in the past but the method was changed at some point. I added a header to tell the difference.

The first line in the file is "NeWe"

The second line is the character count of the following text, and can be used for initial verify of the contents.

The following simple text encryption is each text character XOR'd with 55hex.

The following text data is as follows...

Calculator ID:
   67 or 97
Card bitmap file name:
   Must be in same directory as the card file
Card name:
   Limited to ASCII character values 0x20 - 0x7F
   20 characters maximum

Card Data:

HP registers contain 7 bytes or 56 bits.
Card #1 storage is from HP memory $2F -> $20
Card #2 storage is from HP memory $1F -> $10

Each original card has 34 records of data each containing 28 bits of information.

This comprises a STATUS record, 32 data records and a Checksum record.

This gives a total count of 952 bits per card.

The first of the card file data is 21 0's. (3 records). This is not normally needed, or stored on real cards, but early on I didn't fully understand the way the HP microcode interacts with the CRC. You can ignore this data for reading, but must be inserted for writing if the file format is to stay compatible. For the same reason, the Checksum record is duplicated at the end of the file.
The next 7 nibbles of data are the card STATUS information

<table>
<thead>
<tr>
<th>N0</th>
<th>N1</th>
<th>N2</th>
<th>N3</th>
<th>N4</th>
<th>N5</th>
<th>N6</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>SCI</td>
<td>0</td>
<td>DSPx</td>
<td>0</td>
<td>DEG</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>ENG</td>
<td>-</td>
<td>1</td>
<td>RAD</td>
<td>1234 Flags</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>FIX</td>
<td>9</td>
<td>2</td>
<td>GRD</td>
<td>1=Set</td>
</tr>
</tbody>
</table>

For example, if the calculator is set to:

```
FIX 3, RAD Mode, Flag 1 = 1, Single Sided, Program Card
2
3
1
8  (4 bits - Flag1 = Set, bit 3 = 1)
1
3
```

The next data is the 112 steps of code.

Consider this data is in register $2F (47dec):

```
DB55DC542D1BFA  (Nibble 13 - Nibble 0)
```

The data for the card file is as follows...

```
5 - nibble 7
C
D
5
5
B
D - nibble 13
A - nibble 0
F
B
1
D
2
4 - nibble 6
```

Continue same sequence down to RAM address $20 (32dec)
The next data is 7 nibbles of checksum

The checksum is the sum of the STATUS and program nibbles

Example: A 67 program is created as follows:

```
FIX 2, DEG, No Flags, Single Sided, Program Card #1
7 presses of Key 1
(Note: The program code for Key [1] is 11)
```

Register $2F: 11111111111111

Card data: (Ignoring blank zeros’ mentioned above)

```
2220013      STATUS
1111111      Reg $2F
1111111
0000000      all other registers -> $20
0000000
4442235      checksum
```

The checksum is stored in the file as...

```
4
4
4
2
2
3
5
4
4
4
2
2
3
5
```

Notice that the checksum data is duplicated, however as mentioned previously, only one checksum record is required for the actual card.

If there is data for card #2 then that card is stored in the same format but the STATUS bits will be adjusted for Card #2.

Data cards are stored the same except that the memory storage registers are used.

Data Card #1 from RAM memory $0F down to $00
Data Card #2 from RAM memory $3F down to $30