PDP-8/L Minicomputer Restoration and Programming
William Minshew and Eric Schwarzenbach '13

Submitted to the
Department of Mechanical and Aerospace Engineering
Princeton University
in partial fulfillment of the requirements
of Undergraduate Independent Work

Final Report

May 2, 2013

Michael Littman
Daniel Nosenchuck
MAE 440D
34 pages
Color photos included

I hereby declare that this Independent Work report represents my own work in accordance with University regulations.

x [illegible]

Eric Schwarzenbach

x [illegible]

William Minshew

ACKNOWLEDGMENTS

We would first like to acknowledge our advisor Michael Littman for providing weekly advice and support throughout the year. Advice given by Michael Thompson and Warren Stearns of the Rhode Island Computer Museum (RICM) has been a great help, as they were often able to provide guidance in some of the more technical aspects of diagnosing and debugging. Special thanks to Brian Shelburne of Wittenberg University for creating the PDP-8 emulator we used extensively. Assistance provided by Jo Ann Love and Cynthia Menkes for helping us stay on track with deadlines and answering any and all questions, no matter how silly, throughout the year was greatly appreciated. Finally, we would like to thank Princeton's School of Engineering and Applied Science, and Dean Bogucki personally, for supplying the financial funds with which our beloved PDP-8/L was purchased.

Table of Contents

I. Introduction 7
II. Power supply 9
III. Front panel 11
IV. Core memory 13
  A. How core memory works 13
  B. Diagnosis and repair 16
V. Programming 19
  A. Instruction set 19
    1. Memory reference instructions (MRI) 19
    2. I/O transfer instructions (IOT) 20
    3. Microinstructions (OPR) 21
  B. Emulator 22
  C. Diagnostic programs 22
VI. Current loop to RS-232 converter 24
  A. Designing the circuit 24
VII. Transmitting programs 26
VIII. Future work 27
IX. Works cited 28

Table of Figures

Figure 1. General layout of our PDP-8/L 8
Figure 2. Side view from the outside of the three electrolytic capacitors in the power supply 9
Figure 3. The front panel 11
Figure 4. Illustration of 4 x 4 core memory plane 14
Figure 5. Important flip chip modules in memory functions 16
Figure 6. Core memory stack with attached sense and inhibit wires 18
Figure 7. Layout of MRI instructions 19
Figure 8. Layout of IOT instructions 20
Figure 9. Visual display of Brian Shelburne's PDP-8 emulator 22
Figure 10. Layout of group 1 microinstructions 30
Figure 11. Layout of group 2 microinstructions 31

ABSTRACT

In this project, we successfully restored a Digital Equipment Corporation (DEC) PDP-8/L minicomputer and interfaced it to a modern laptop. During its restoration, we repaired the power supply, replaced a defunct cooling fan, fixed the front panel, repaired the core memory, created a library of basic programs on an emulator in PDP-8 assembly language (PAL), ran extensive diagnostics on both the CPU logic and memory stack, and designed an RS-232 to current loop interface to allow the PDP-8 to communicate with the terminal of a modern laptop.

I. Introduction

The PDP-8 computer, a table-top machine designed in 1965 by Digital Equipment Corporation (DEC), was the world's first true minicomputer. The machine was packaged with a teletype for basic input/output and priced at a mere $18,000, allowing thousands to be sold worldwide to all kinds of manufacturing plants, offices, universities, and scientific laboratories,[1] leading to its nickname as the "model T of the industry."[2] Composed of discrete flip chip boards, the computer used diode-transistor logic (DTL) and was roughly equivalent to a small refrigerator in size.[3]

In 1968 the PDP-8/L was introduced; it was the smallest and least expensive model in the PDP family. The computer was built with medium-scale integration (MSI) transistor-transistor logic (TTL) integrated circuit modules and uses magnetic cores for memory—the same kind of memory used in the early space program. It is a single-address machine with a word size of 12 bits and a memory of 4096 (4k) words, subdivided into 32 pages of 128 words each. The computer has four main registers: the accumulator (AC), program counter (PC), memory address (MA), and memory buffer (MB). The PDP-8 is considered a "load and store" computer; the 12-bit word size only allows one address to be referenced per instruction. Therefore, the accumulator is always assumed to be the other operand in any instruction requiring two values. This simplistic design requires that the accumulator is often overwritten by subsequent instructions. The system runs with a cycle time of approximately 1.6 us, determined largely by memory access; therefore, two cycle instructions such as addition take roughly 3.2 us (~0.313 MIPS). Programming is done either manually with switches on the front panel or by reading in perforated paper tape.[4] Figure 1 shows the general layout of the inside of our machine, and further detail is depicted in Appendix A.

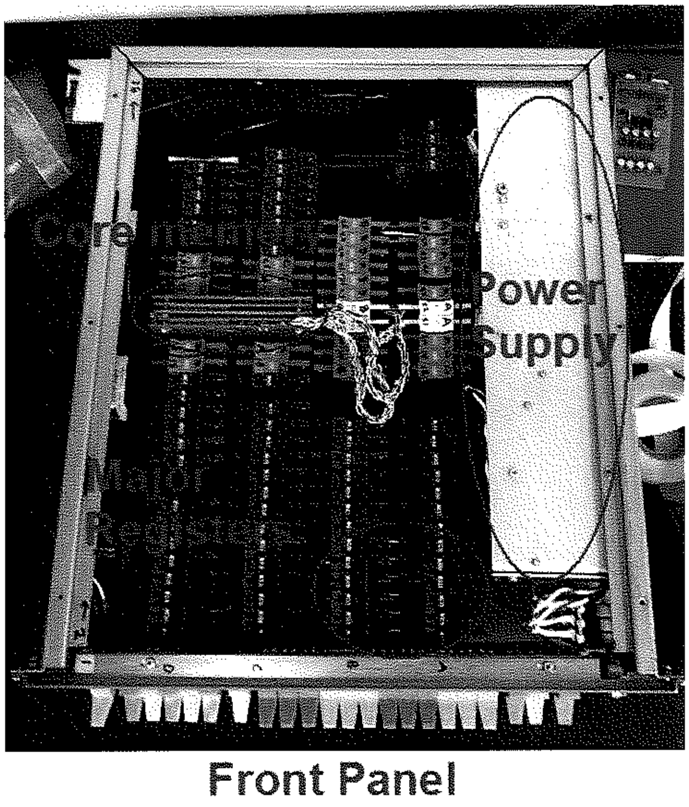


Figure 1. General layout of our PDP-8/L

Over the summer Professor Michael Littman acquired an old PDP-8/L and teletype, which we restored to working order over the course of the year. After performing some simple diagnostic tests on the instruction set and memory, we subsequently designed and constructed an interface which allows it to communicate with a modern laptop. Using this interface, we were able to upload more advanced diagnostic programs onto the machine to further test the integrity of its components.

II. Power supply

Our first order of business was to repair the power supply to allow us to safely turn on the computer and begin our diagnosis. The PDP-8 uses three main large electrolytic capacitors (see Figure 2), which after several decades of nonuse are not safe to immediately power on. This is because the thin film of oxide used as an insulator between the electrodes requires a small leakage current to prevent degradation. After a long period of inactivity, this layer may break down completely, causing the capacitor to form a short. When suddenly powered on to full voltage, the heat generated from excessive current flow through the short may produce a gas inside the capacitor that ultimately causes an explosion.[5]

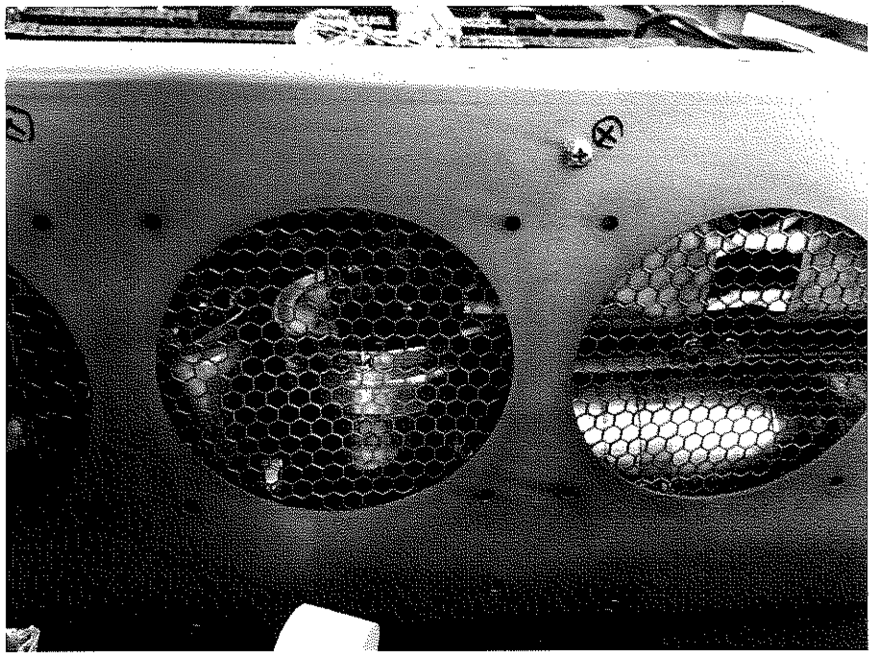


Figure 2. Side view from the outside of the three electrolytic capacitors in the power supply

In order to avoid this safety hazard, the three capacitors had to be reformed—a process which attempts to slowly bring the capacitors back up to operating voltage with minimal current in order to carefully re-oxidize the insulator. We were able to accomplish this with a standard laboratory power supply, which allowed us to set a maximum allowable current of 10 mA and then slowly ramp up the voltage over time.

To access the power supply, we first needed to properly document and remove all of the flip chip modules (see Figure 1). After examining the power supply schematics, we decided it would be safe to attempt reformation without removing the capacitors from the rest of the power supply.[6] Unfortunately, this process was only successful with one of the capacitors and failed to reach operational voltage on the other two. We then carefully disassembled the remaining capacitors for future replacement, but decided it would be prudent to re-try the process now that they were fully isolated from the system. Fortunately, this time we successfully reformed the remaining two electrolytics.

With the capacitors now in safe working condition, we plugged the machine in and turned our PDP-8 on for the first time in over a decade. We noticed immediately that one of the cooling fans did not work properly. After removing the defunct fan and soldering on a replacement, we briefly re-tested the power supply. When everything appeared to be in working condition, we decided it was safe to re-insert the flip chip boards and attempt to power the whole computer on. The process was successful—our PDP-8/L had life at last.

III. Front panel

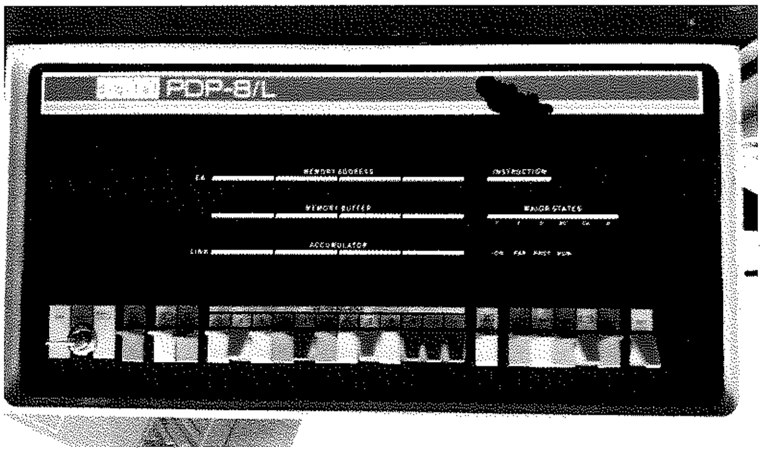


Figure 3. The front panel

The front panel of the PDP-8/L is particularly important for the initial debugging process, as the manual switches are the only way to test the memory and various functionalities until the system is in good enough condition to read input from perforated tape.

After powering the system on and briefly testing random inputs to memory, we discovered two problems with our front panel. First, many of the switches seemed spotty at best—sometimes working, often not. Because of the sporadic nature of the issue, we thought it might have to do with faulty mechanical contacts; after all, the system had been dormant for quite some time. Fortunately, we were correct and the problem was quickly rectified with the application of contact cleaner. The second problem was with the incandescent lamps used to display the contents of the accumulator, memory address, and memory buffer. Specifically, the lights representing bits 8, 9, 10, and 11 of the AC, bits 0 and 3 of the MA, and bit 0 of the MB were not working. After inspecting the back of the panel, we noticed that the previous owner had already replaced several bulbs, and in the process, had damaged many of the traces; of the 7 broken lights, only the bulbs on AC bit 11 and MA bit 3 were actually blown. We diagnosed the remaining bits and ended up replacing transistors on bits 8, 9, and 10 of the AC. The malfunction on bit 0 of the MB was due to a bad solder joint—a result of the shoddy prior repair job—

and bit 0 of the MA was simply a missing bulb which we subsequently replaced. With all the lights restored, our front panel was brought back to working order, allowing us to focus on the most important part of our restoration—the core memory.

IV. Core memory

A. How core memory works

The main memory in the PDP-8 consists of a 12 by 64 by 64 matrix of small toroidal ferrite-cores. Threaded through the center of each core are X-selection, Y-selection, sense, and inhibit wires, each with a specific direction (see Figure 4) and wound in a clever 3-D manner. Each magnetic core represents a single bit with the magnetic state of the core representing the binary value of the bit. The wires are wound in a way that is easily relatable to the Cartesian coordinate system: the x- and y-coordinates specify the address in memory and the z-coordinate corresponds to the specific bit in that address—hence, the 12 by 64 by 64 matrix is equivalent to 4096 12-bit words. Each X-selection wire is threaded through every core in a given x-plane for a total of 64 wires, while each Y-selection wire is threaded through every core in a given y-plane, again for a total of 64 wires. Thus, the proper X- and Y-selection wires will intersect only at the 12 cores corresponding to the 12-bit word we wish to access in memory. Each sense and inhibit wire is threaded through every core in a given z-plane for a total of 12 wires each. So by using an X-selection, Y-selection, and either sense or inhibit wire we can access any particular bit in memory.[7]

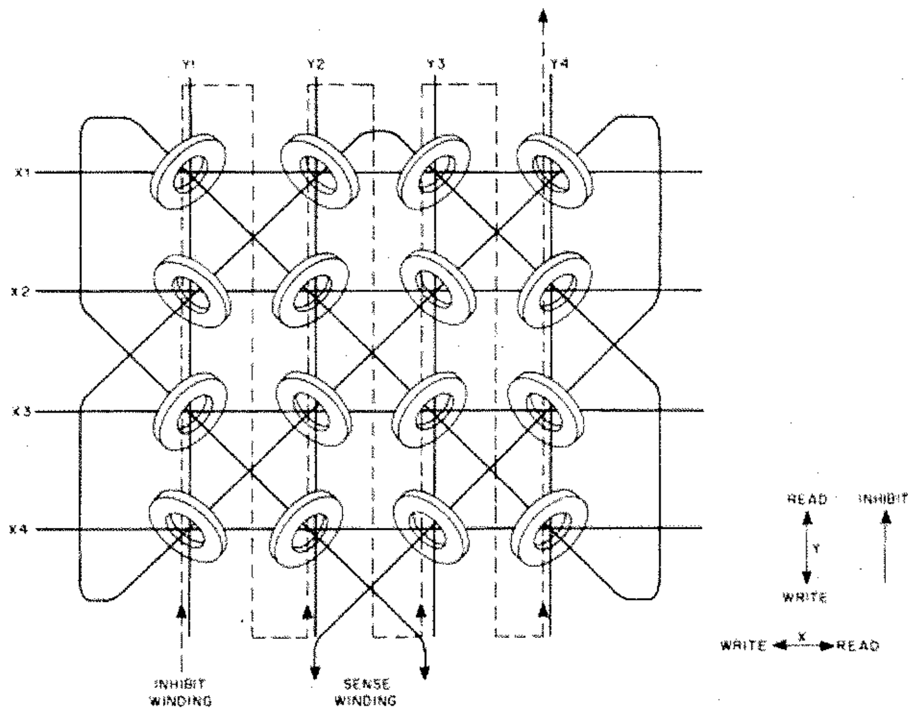


Figure 4. Illustration of 4 x 4 core memory plane[8]

Memory operations on the PDP-8/L consist of five major functions: address selection, read, sense, inhibit, and write—all five of which must be performed regardless of whether we intend to read or write. To understand why, we will first define the magnetic state in the ferrite core induced by current flowing in the "write" direction (see Figure 4) to correspond to a bit with a value of 1 (conversely, the state induced by current flowing in the "read" direction corresponds to a value of 0). In order to read the memory, the computer first erases the memory by driving a current that sets all of the values to 0. While it does this, the sense wire detects which bits change magnetic flux and which bits don't—those that do were previously 1s and those that don't were already 0s. Thus, after we are done we must write the value back into memory (after all, a read function that erased the value permanently would be quite useless). We will discuss how the computer writes data to the memory after address selection.[9]

To select an address, the memory address register is first decoded and the passed along to the X- and Y-Diode Selection Matrices, which cause current to flow through the correct X- and Y-selection wires, thereby selecting an entire 12-bit word. The direction of this current is dependent on whether the computer intends to read or write to the address as depicted in Figure 4. The decoding is performed by the G221 flip chip modules surrounding the memory stack (see Figure 5), where the appropriate X-selection wire is determined by bits 0 to 5 and the Y by bits 6 to 11.[10]

To perform any operation on memory, first the address is selected, and current is run in the "read" direction. It is important to note at this time that every wire passing through a core is limited to a low enough current level, known as the "half-select value," such that no single wire may produce a strong enough magnetic field to induce a magnetic state change in any core. Thus, in order to change the magnetic state of any ferrite core, current must be flowing through both the X- and Y-selection wires. The magnetic states of other cores in a given X- or Y-plane, which have only a single selection wire with current, are left unaltered. The system detects which bits were 1s and 0s and uses this information to write the value back into memory. Now that the read function has set each bit equal to 0, we only need to fix the bits that were previously 1s. This is where the inhibit wire comes in. We perform address selection again, this time running current in the "write" direction. For the bits that we wish to leave as 0, the computer runs a current through the inhibit wire, which flows anti-parallel to the Y-selection wire. This produces a net effect of no current in that direction, leaving only the X-selection wire with current, which is not enough to bring the ferrite core out of the "1" state. Note, however, that this write process only works when all of the cores begin in the "0" state. This is why it is necessary to read (and therefore erase) the memory first, even if our ultimate goal is to write a new value.

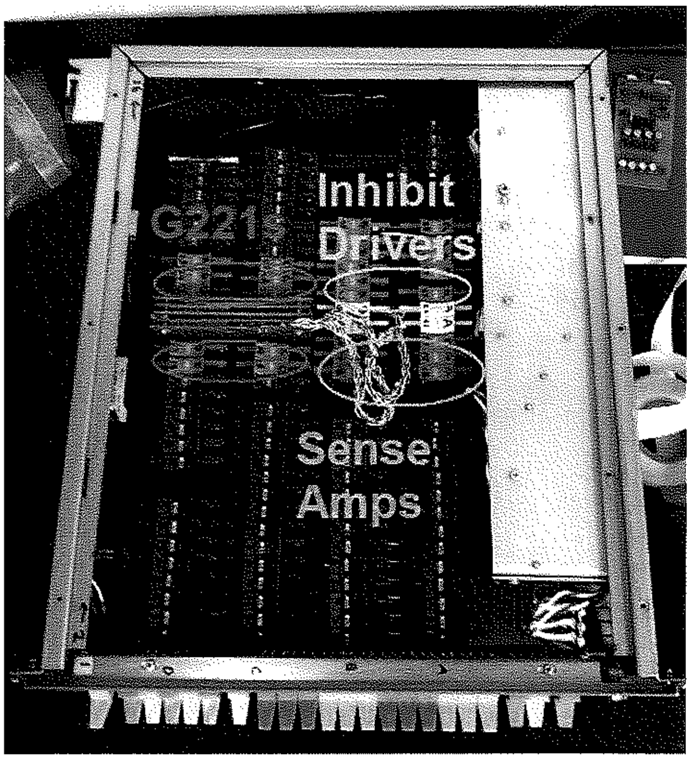


Figure 5. Important flip chip modules in memory functions

B. Diagnosis and repair

In order to diagnose our core memory, we first needed to ensure the PDP-8 was correctly selecting the specified addresses. The eight G221 modules—organized in 4 pairs—facilitate this action via the Diode Selection Matrices on the core stack. When working properly, each pair only outputs one high line at a time. This was easily verified with the use of an extender board and oscilloscope.

After we confirmed our address selection modules worked properly, we moved on to the reading and writing processes. We began testing by manually storing the address of each location in that location by each individual octal digit (e.g. 0000o in 0000o, 0001o in 0001o, … , 0007o in 0007o, 0010o in 0010o, 0020o in 0020o, …, 6000o in 6000o, 7000o in 7000o). The test was largely successful except for one glaring problem: bit 2 of the memory buffer (MB) always read high, no matter what address we examined and no matter what value had been previously stored.

In order to diagnose where the problem was located, we did our best to isolate each phase of memory access for testing. Since each memory address contains 12 bits, all stages of memory operation employ several flip chip modules that perform the same function on a 3 or 4 bit subset of the word. To exploit this redundancy, we re-arranged the boards in our PDP-8 one at a time to see if the error predictably jumped around to different bits. For example, if the problem were located in the memory buffer, then we would expect to see bit 0 read high consistently and bit 2 behave normally if we switched the M220 boards at AB06 (responsible for bits 0 and 1 of all major registers, see Figure 1 and Appendix A for physical location) and AB07 (responsible for bits 2 and 3).[11] Unfortunately, after swapping several pairs, the problem persisted. We returned each module to its original location and moved on to the next phase—sensing.

When the cores are erased during the "read" process, a small pulse is generated on the sense wire from the change in magnetic flux if the core were previously in the "1" state. This pulse is then amplified by a G020 module and is used by the computer to determine what value was stored at that memory location.[12] To see if we had a problem with our sense amplifiers, we again took advantage of the redundancy across boards by swapping A18 (responsible for sensing bits 0 and 1) and A19 (responsible for bits 2 and 3).[13] The problem remained—our A19 G020 appeared functional.

As explained above, inhibit wires are used to write memory. The wires themselves are controlled by inhibit drivers on the G228 flip chip modules. Similar to previous tests, we swapped the boards at A23 (responsible for bits 0 to 3) and B23 (responsible for bits 4 to 7).[14] After swapping the boards, bit 2 still remained high across all tested memory addresses.

By process of elimination, we determined the problem was likely either a broken inhibit or sense wire within our memory stack. By testing continuity on both the sense and inhibit wires of bit 2, we were able to determine that the problem was a break in the sense wire. Our next task was finding exactly where the break had occurred in the wire. If the break lay outside the stack, we would simply need to solder a new wire on at the break and connect it to the sense wire board.

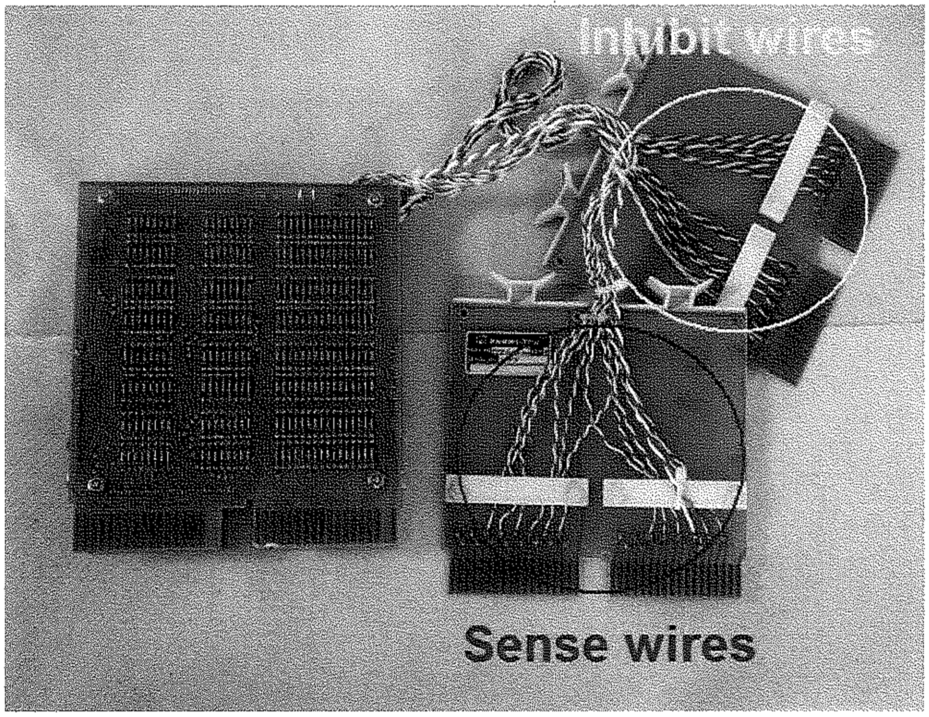


Figure 6. Core memory stack with attached sense and inhibit wires

If the break were inside the stack, we would need to physically dismantle the core memory to replace the wire. Fortunately, the break lay just outside the cores, and we were able to replace the wire successfully. With the sense wire repaired, the memory stack was fully operational again, which meant we could begin running diagnostic programs on the PDP-8.

V. Programming

A. Instruction set

The instruction set for the PDP-8 is extremely simplistic, yet remains powerfully flexible. The machine only has 8 different opcodes, corresponding to the first 3 bits of any instruction. However, to allow for more functionality, these opcodes are subdivided into three groups: memory reference instructions (opcodes 0-5), input/output transfer (opcode 6), and microinstructions (opcode 7).

1. Memory reference instructions (MRI)

```
   0   1   2   3   4   5   6   7   8   9   10  11
 +---+---+---+---+---+---+---+---+---+---+---+---+
 |   OpCode  |IA |MP |       Offset  Address     |
 +---+---+---+---+---+---+---+---+---+---+---+---+

Bits 0 - 2   Operation Code (opcode)
Bit    3     Indirect Addressing Bit (0:Direct/1:Indirect)
Bit    4     Memory Page (0:Zero Page/1:Current Page)
Bits 5 - 11  Offset Address
```

Figure 7. Layout of MRI instructions[15]

Memory reference instructions act on a specified location in memory, denoted by the indirect bit, page bit, and offset (see Figure 7). These various bits may be combined in a number of manners to allow any instruction to access any location in memory. The instructions are as follows:

Opcode	Instruction	Function
000_2	AND	Logical AND
001_2	TAD	Two's Complement Add
010_2	ISZ	Increment and Skip if Zero
011_2	DCA	Deposit and Clear the Accumulator
100_2	JMS	Jump to Subroutine
101_2	JMP	Unconditional Jump[16]

2. I/O transfer instructions (IOT)

```
  0   1   2   3   4   5   6   7   8   9   10  11
+---+---+---+---+---+---+---+---+---+---+---+---+
| 1 | 1 | 0 |  device    number       |  function |
+---+---+---+---+---+---+---+---+---+---+---+---+

Bits  0 - 2  : Opcode 6
Bits  3 - 8  : Device Number
Bits  9 - 11 : Extended function field (operation specification bits)
```

Figure 8. Layout of IOT instructions[17]

Input/output transfer instructions are used to interface the PDP-8 with peripheral devices. Though the PDP-8 can interface with up to 64 different external devices, the most common two are the teletype (TTY) keyboard and printer. In order to transfer information between the fast CPU and slow peripherals, the PDP-8 utilizes flags to signify when both components are ready. To input or output data, the accumulator transfers data with the proper I/O device buffer, described in more detail below.

The five instructions for input from the TTY keyboard are:

Instruction	Value	Function
KCF	6030o	Clear keyboard flag
KSF	6031o	Skip on keyboard flag
KCC	6032o	Clear keyboard flag and AC
KRS	6034o	Read keyboard buffer static; keyboard buffer is OR'd with bits 4 to 11 of the AC
KRB	6036o	Read keyboard buffer dynamic; combination of KCC and KRS

Despite having five separate instructions, in practice only two are used, KSF and KRB. The five instructions for output to the TTY printer are:

Instruction	Value	Function
TFL	6040o	Sets printer flag
TSF	6041o	Skip on printer flag
TCF	6042o	Clear printer flag
TPC	6044o	Load printer buffer and print
TLS	6046o	Load printer sequence; combination of TCF and TPC

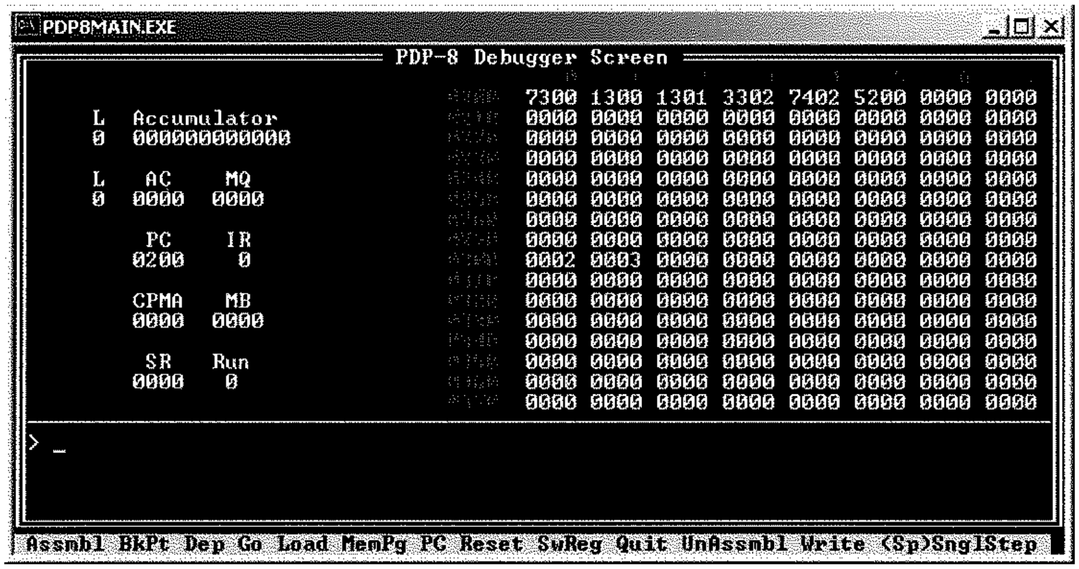
Again, only TSF and TLS are commonly used.[18]

3. Microinstructions (OPR)

Microinstructions use the accumulator-link pair as the sole operand, freeing up the remaining instruction bits for functional use. This set can be further subdivided into two groups based on bit 3: one that acts on the AC-link pair (group 1), and another that focuses on conditional branching based on the AC-link pair (group 2). Because each bit represents a distinct microinstruction, often multiple microinstructions can be combined into one instruction. For example, in group 1, bit 4 clears the AC (7200o) and bit 5 clears the link (7100o); therefore, when both bits are high (7300o) both the AC and link are cleared. See Appendix B for detailed diagrams of bit functionality in each instruction group.

B. Emulator

While restoring our PDP-8, we simultaneously familiarized ourselves with the architecture by using an emulator created by Brian Shelburne. The program came with a user's manual and a series of labs designed to introduce students to the PDP-8 Assembly Language (PAL) and the unique coding process involved with a "load and store" computer.[19]



```
                          PDP-8 Debugger Screen

                                            7300 1300 1301 3302 7402 5200 0000 0000
   L  Accumulator                           0000 0000 0000 0000 0000 0000 0000 0000
   0  000000000000                          0000 0000 0000 0000 0000 0000 0000 0000
                                            0000 0000 0000 0000 0000 0000 0000 0000
   L   AC    MQ                             0000 0000 0000 0000 0000 0000 0000 0000
   0  0000  0000                            0000 0000 0000 0000 0000 0000 0000 0000
                                            0000 0000 0000 0000 0000 0000 0000 0000
       PC    IR                             0000 0000 0000 0000 0000 0000 0000 0000
      0200    0                             0002 0003 0000 0000 0000 0000 0000 0000
                                            0000 0000 0000 0000 0000 0000 0000 0000
      CPMA   MB                             0000 0000 0000 0000 0000 0000 0000 0000
      0000  0000                            0000 0000 0000 0000 0000 0000 0000 0000
                                            0000 0000 0000 0000 0000 0000 0000 0000
       SR   Run                             0000 0000 0000 0000 0000 0000 0000 0000
      0000   0                              0000 0000 0000 0000 0000 0000 0000 0000
                                            0000 0000 0000 0000 0000 0000 0000 0000

> _

Assmbl BkPt Dep Go Load MemPg PC Reset SwReg Quit UnAssmbl Write <Sp>SnglStep
```

Figure 9. Visual display of Brian Shelburne's PDP-8 emulator[20]

The emulator is fairly straightforward to use. Programs can be written in PAL and then assembled, or can be entered manually with machine code on the debug screen shown in Figure 9. The debug screen displays the contents of the major registers on the left half of the screen and the contents of a specific memory page on the right half. Programs may be run continuously or single-stepped through with the space bar. Working with this emulator prior to the restoration of the physical computer helped expedite future programming on our PDP-8.

C. Diagnostic programs

With the sense wire repaired, we were finally able to run programs on our PDP-8. However, before we could fully diagnose the state of the core memory, we had to verify that the PDP-8 could still properly execute its entire instruction set. In order to accomplish this, we used the switches to input small programs testing the various instructions one at a time (see Appendix C for a list of test programs). Once we verified the CPU was fully functional, we moved on to memory diagnostics. First, we ran a basic memory test that checked each memory address one at a time, storing all 1s or 0s in each location and then verifying the contents. Next, we began testing for interference between adjacent locations by alternating between all 1s or 0s in every location, and then checking a single location to make sure it hadn't been affected. If it encountered any contaminated memory locations, the program would store the address in a previously hand-tested section of memory. The test ran flawlessly—our memory proved functional. With considerable confidence in our core memory, we began designing an interface to allow the PDP-8 to communicate with a modern laptop in order to facilitate the loading of longer, more advanced diagnostics.

VI. Current loop to RS-232 converter

Having finished our own memory diagnostic tests, we then moved onto our final task—interfacing our PDP-8 with a modern computer. To do this, we needed to convert the PDP-8's output into a format readable by a modern laptop. The PDP-8 uses a 20 mA current loop interface; however, this is no longer a viable form of communication with modern day electronics, which require voltage based interfaces such as RS-232.

As its name implies, a current loop interface uses the presence of current rather than voltage levels for signaling. The absence of current signifies high (space), and the presence of current within the loop signifies low (mark). The PDP-8 receives and transmits data by sending or reading incoming current through these loops. RS-232, on the other hand, uses voltage levels to determine data bits. Any value between +3V and +15V is interpreted as a logic zero (space), while any voltage between -3V and -15V is interpreted as logic one (mark). Voltage levels between -3V and +3V are not a valid signal. This standard is now outdated due to its low transmission speeds and high voltage requirements. However, the low transmission speed that makes this technology obsolete is also what makes it so useful for communicating with the PDP-8, which operates at a 110 baud rate. While RS-232 is seldom used in modern laptops, many older computers still have a COM port which use this standard.

A. Designing the circuit

We began designing our circuit by further researching the technology standards online. We quickly realized it would be necessary to electrically isolate the RS-232 voltages from the current loop inputs and outputs, and began designing a circuit based on one we found online utilizing four optoisolators. After constructing the circuit, we began testing it and noticed several key flaws: First, our assumed pinouts were wrong. We had based our design off of pinout schemes located online, which incorrectly attributed DCE/DTE roles to our laptop and PDP-8. Furthermore, the layout of the circuit did not provide enough current to fully unlock all of the optoisolators, causing our data to be lost in transmission. After much frustration and considerable effort to rectify our initial circuit, we decided to scrap it entirely and produce our own design from scratch.

We began by running our own tests to determine which pinouts from the laptop and PDP-8 were responsible for sending and receiving data. Once correctly identified, we moved on to our second issue of fully unlocking the transistors within our optoisolators. Our new design utilized two optoisolators and a MAX232 with an intermediary current amplifier to ensure full transmission of the data. As a simple way to test our circuit without involving the PDP-8, we initially constructed it in a loopback format (see Appendix D).

In the loopback circuit, RS-232 data output from the laptop's TX pin is converted to TTL by the MAX232. Next, the TTL logic is run through a current amplifier because our MAX232 was unable to source enough current to fully unlock the optoisolator. This output acts as a switch to open or close the current loop, which would normally be sent to the PDP-8. In our loopback design, however, the output drives another optoisolator via a second current amplifier—which is where the PDP-8 would normally input its data. The output of the second optoisolator is then connected to a pull-up resistor to produce TTL voltages. The final signal is then inverted and fed back into the MAX232, which subsequently outputs RS-232 data to the RX pin of our laptop.

The circuit was successful—we were able to echo data from a windows terminal on our laptop. We then split our loopback into "send" and "receive" components with the PDP-8 now connected (see Appendix D). To confirm that our circuit still functioned properly, we toggled in simple character input and output programs. Both programs were able to successfully send and receive data to and from the PDP-8—our computers could now communicate.

VII. Transmitting programs

With some minor tweaking to the parameters of our windows terminal (see Appendix E for specific settings), we discovered we were now able to continuously transmit data between our machines. All PDP-8s begin with a clean memory stack, meaning an initial program must be toggled into the machine by hand in order to receive data from a perforated tape, or in our case a laptop. The two most basic forms of data are Read-in-Mode (RIM) and Binary (BIN). Since the PDP-8 uses 12-bit words, but the I/O buffer is only capable of sending 8 bits at a time, each transmission consists of 6 data bits with a 2 bit header. In RIM, the absolute address is sent first, followed by the content of said address. Therefore, each word sent requires four 8-bit transmissions—a pair each for the address and content. In BIN, only the starting address is specified, and then subsequent 8-bit pairs transmit the contents of consecutive addresses. This means that a program in BIN format uses as little as half the tape and can therefore be sent twice as fast. Consequently, programs are usually stored in BIN format. Because of this, the standard BIN loader program is larger and more robust with a checksum error included. Since the PDP-8 starts off with a clean slate, it is common practice to toggle in a simple RIM loader, which is then subsequently used to upload the more rigorous BIN loader via paper tape.

We located a wealth of DEC PDP-8 programs—in both RIM and BIN formats—in the "Software Archive" of www.bitsavers.org, as well as other online sources (see Appendix C).[21] After toggling in the RIM loader from the PDP-8 handbook, we then used it to successfully upload the BIN loader. With the BIN loader in place, we were then able to complete our PDP-8 diagnostics using a number of DEC's official testing programs, including rigorous instruction tests and checkerboard memory diagnostics with varying patterns.

VIII. Future work

With a fully functional PDP-8 and robust I/O interface, it is now possible to install more advanced operating systems and language interpreters. Future projects should seek to upload and utilize systems such as BASIC, FORTRAN, and FOCAL-69, which greatly expand the programming capabilities of the PDP-8. These languages also come with their own loaders; however, the relay control line is required for their use. Thus, our RS-232 to current loop converter could be further developed to interface CTS and DTR lines of the RS-232 with the relay control of the PDP-8.

---

[1] "Internet History" 1965
[2] Jones FAQ
[3] McMurran 88
[4] PDP-8/L Maintenance Manual Vol I 1-1
[5] Hazell
[6] PDP-8/L maintenance Manual Vol II D-CS-718-0-1
[7] PDP-8/L maintenance Manual Vol I 4.12
[8] Ibid. Figure 4-8
[9] Ibid. 4.12
[10] PDP-8/L maintenance Manual Vol II D-BS-8L-0-16
[11] Ibid. D-BS-8L-0-8
[12] PDP-8/L maintenance Manual Vol I 4.12
[13] PDP-8/L maintenance Manual Vol II D-BS-8L-0-14
[14] Ibid. D-BS-8L-0-14
[15] Shelburne 1.4.4
[16] Introduction to Programming 2-8
[17] Shelburne 1.4.4
[18] Shelburne 6.2
[19] Shelburne Home Page
[20] Ibid.
[21] Thompson, Michael and Warren Stearns

IX. Works cited

Gesswein, David. Online PDP-8 Home Page, Run a PDP-8. N.p.. Web. <http://www.pdp8.net/index.shtml>.

Hazell, Mike. "Capacitor Reforming." VMARS. The Vintage and Military Amateur Radio Society, Apr 2000. Web. <http://www.vmars.org.uk/capacitor_reforming.htm>.

"Internet History." Computer History Museum. Web. 1965. <http://www.computerhistory.org/internet_history/>.

Introduction to Programming: PDP-8 Family Computers. Digital Equipment Corporation, 1969. Print.

Jones, Douglas. The Digital Equipment Corporation PDP-8. University of Iowa Department of Computer Science. Web. <http://homepage.cs.uiowa.edu/~jones/pdp8/>.

McMurran, Marshall. Achieving Accuracy: A Legacy of Computers and Missiles. Xlibris Corporation, 2009. 88.

McQuiggan, Kevin, ed. Highgate's PDP-8 Page. N.p., 10 Jan 2003. Web. <http://highgate.comm.sfu.ca/pdp8/>.

PDP-8/L Maintenance Manual. 1. Digital Equipment Corporation, 1968. Print. <http://bitsavers.trailing-edge.com/pdf/dec/pdp8/pdp8l/DEC-8L-HR1B-D_8LmaintVol1.pdf>.

PDP-8/L Maintenance Manual. 2. Digital Equipment Corporation, 1968. Print. <http://bitsavers.trailing-edge.com/pdf/dec/pdp8/pdp8l/DEC-8L-HR2A-D_8Lschem_Feb70.pdf>.

"PDP-8/L Restoration." Rhode Island Computer Museum, 18 Nov 2012. Web. <http://www.ricomputermuseum.org/Home/equipment/pdp-8-l/pdp-8-l_blog>.

Shelburne, Brian. Brian Shelburne's PDP-8 Home Page. University of Wittenberg. Web. <http://www4.wittenberg.edu/academics/mathcomp/bjsdir/PDP8HomePage.htm>. Emulator.

Shelburne, Brian. The PDP-8 Emulator Program User's Manual. Web. <http://www4.wittenberg.edu/academics/mathcomp/bjsdir/pdp8.zip>.

The Digital Small Computer Handbook. Digital Equipment Corporation, 1967.

Thompson, Michael and Warren Stearns. "PDP-8/L Reconstruction Help." Message to William Minshew and Eric Schwarzenbach. E-mail.

Slyngstad, Vince. "Repair Stuff." PDP-8 Stuff. N.p., 20 12 2011. Web. 11 Jan 2013. <http://so-much-stuff.com/pdp8/repair/repair.php>.

Appendix A – Documented layout of flip chip modules

SLOT	A	B	C	D
1	G921 – PDP-8/L control panel			
2	[illegible]		[illegible]	[illegible]
3	[illegible]		[illegible]	
4	[illegible]		[illegible]	[illegible]
5	[illegible]		[illegible]	[illegible]
6	[illegible]		[illegible]	[illegible]
7	[illegible]		M111	[illegible]
8	[illegible]	[illegible]	[illegible]	[illegible]
9	[illegible]	[illegible]	[illegible]	[illegible]
10	[illegible]	[illegible]	[illegible]	
11	[illegible]	[illegible]	[illegible]	
12	[illegible]	[illegible]	[illegible]	[illegible]
13	[illegible]	[illegible]	[illegible]	[illegible]
14	[illegible]	[illegible]	[illegible]	[illegible]
15	M162	M119	[illegible]	[illegible]
16	M162	M162	M111	[illegible]
17	G020	M162	[illegible]	[illegible]
18	G020	G020	G221	G221
19	G020	G020	G221	G221
20	G020	G020	G610	
21	W825		Memory Stack	
22	W825		G611	
23	G228	G228	G221	G221
24	G228	G228	G221	G221
25	G624	G624	G220	G220
26	G624	G624	M002	
27	G826		[illegible]	[illegible]
28	G785		[illegible]	[illegible]
29	M715		[illegible]	[illegible]
30	M795		[illegible]	[illegible]
31	M718		[illegible]	
32	[illegible]		M706	-
33	[illegible]		[illegible]	M076 cable
34	M111	M983 (?)	M901 cable	M903 cable
35	[illegible]	M983 (?)	M903 cable	M908 (?) cable
36		M983 (?)	M903 cable	M903 cable

Key:

	Optional data break interface
X###	Optional memory parity
X###	Optional power fail
X###	Optional high speed read and/or punch
▨	Green-tabbed flip chip
■	Magenta-tabbed flip chip
░	Flip chip not in machine
	White-tabbed/non-tabbed flip chip

Appendix B – Microinstruction subgroups

Group 1

```
  0   1   2   3   4   5   6   7   8   9   10  11
+---+---+---+---+---+---+---+---+---+---+---+---+
| 1 | 1 | 1 | 0 |CLA|CLL|CMA|CML|RAR|RAL|0/1|IAC|
+---+---+---+---+---+---+---+---+---+---+---+---+
                                          ••
                                 Rotate 1 Position if 0
                                        2 Positions if 1
```

Figure 10. Layout of group 1 microinstructions

Instruction	Value	Function
NOP	7000o	No operation
CLA	7200o	Clear AC
CLL	7100o	Clear link
CMA	7040o	Complement AC
CML	7020o	Complement link
IAC	7001o	Increment AC
RAR	7010o	Rotate AC-link right
RTR	7012o	Rotate AC-link right twice
RAL	7004o	Rotate AC-link left
RTL	7006o	Rotate AC-link left twice

Group 2

```
  0   1   2   3   4   5   6   7   8   9   10  11
+---+---+---+---+---+---+---+---+---+---+---+---+
| 1 | 1 | 1 | 1 |   |SMA|SZA|SNL| 0 |   |   | 0 |
+---+---+---+---+---+---+---+---+---+---+---+---+
                                  ••
                    Bit 8 = 0 : SMA or SZA or SNL

  0   1   2   3   4   5   6   7   8   9   10  11
+---+---+---+---+---+---+---+---+---+---+---+---+
| 1 | 1 | 1 | 1 |   |SPA|SNA|SZL| 1 |   |   | 0 |
+---+---+---+---+---+---+---+---+---+---+---+---+
                                  ••
                 Bit 8 = 1 : SPA and SNA and SZL

  0   1   2   3   4   5   6   7   8   9   10  11
+---+---+---+---+---+---+---+---+---+---+---+---+
| 1 | 1 | 1 | 1 |CLA|   |   |   |   |OSR|HLT| 0 |
+---+---+---+---+---+---+---+---+---+---+---+---+
```

Figure 11. Layout of group 2 microinstructions

Instruction	Value	Function
SMA	7500o	Skip on AC < 0
SZA	7440o	Skin on AC = 0
SNL	7420o	Skin on link = 1

If multiple microinstructions from this set are combined, the individual results are ORed

SPA	7510o	Skip on AC >= 0
SNA	7450o	Skip on AC != 0
SZL	7430o	Skip on link = 0

If multiple microinstructions from this set are combined, the individual results are ANDed

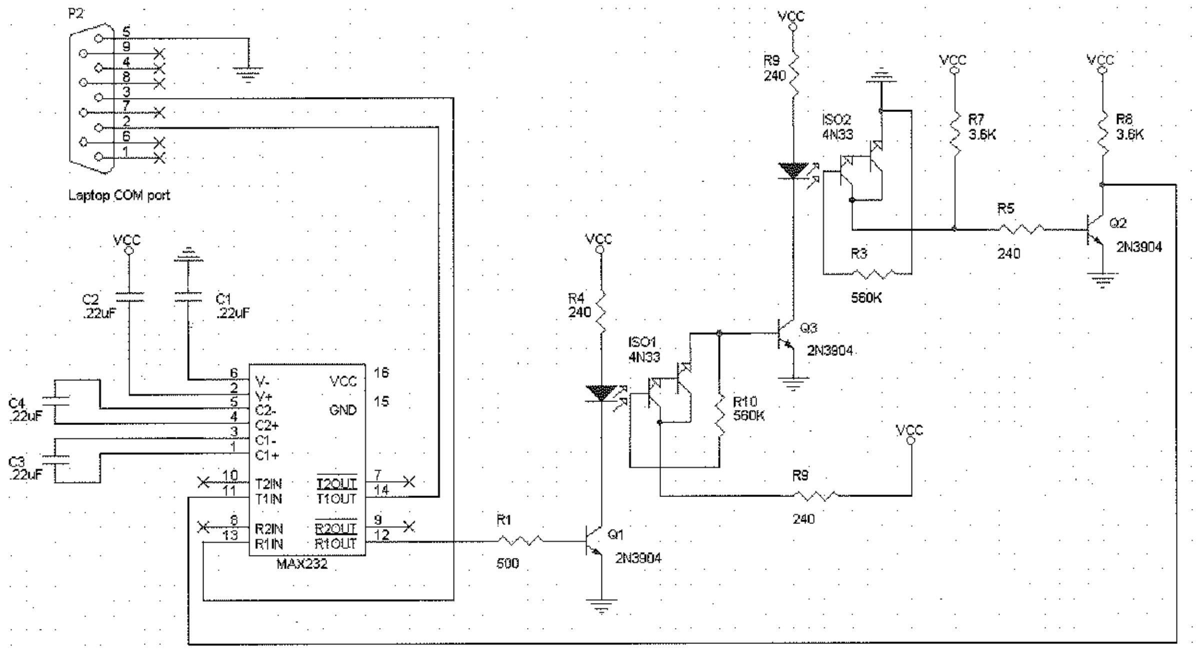
SKP	7410o	Skip always
CLA	7600o	Clear AC
OSR	7404o	OR switch register with AC
HLT	7402o	Halt program

Appendix C – List of programs

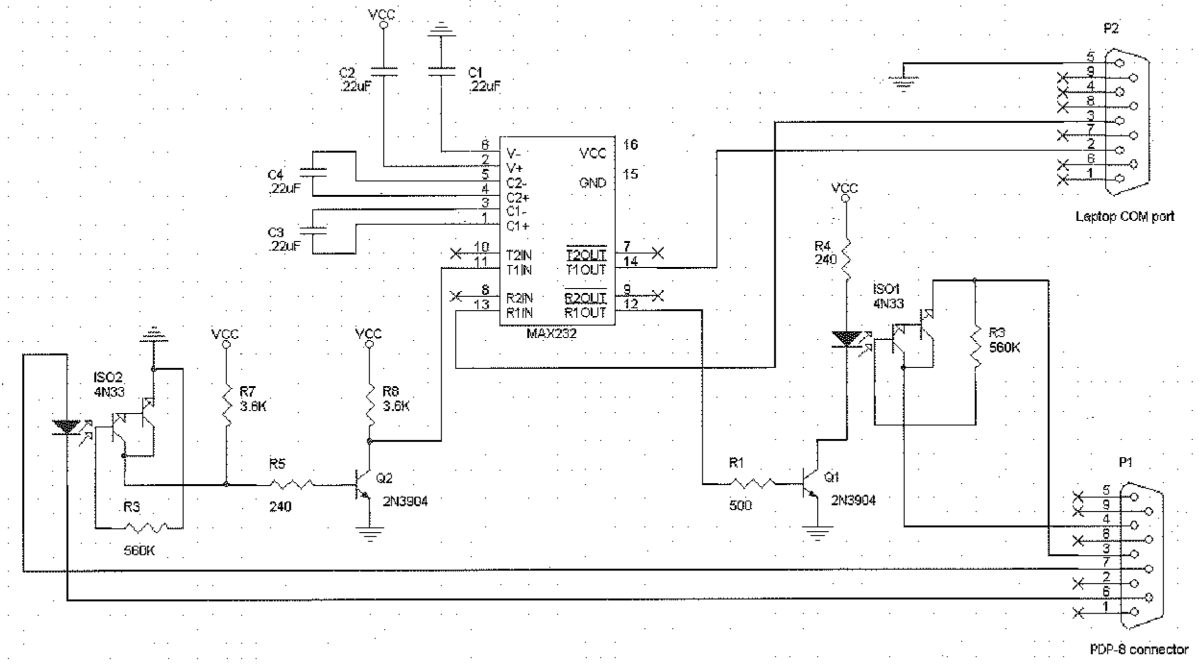
1) Coded for emulator
   a) Counter
   b) Addition A + B
   c) Subtraction A – B
   d) Multiplication (loops)
   e) Division
   f) Absolute value |A – B|
   g) GCD (Euclid's algorithm)
   h) Array summation (indirect addressing)
   i) Unpacking octal digits (shifts / rotations)
   j) Read character, display back
   k) "Hello, world"
   l) Reads string, displays back
2) Coded for PDP-8 diagnostics
   a) Instruction tests
      i) Group 1 microinstructions
      ii) Group 2 microinstructions
      iii) MRI instructions
      iv) Single character input
      v) Single character output
   b) Self-coded diagnostic programs
      i) Continuous character output
      ii) Basic memory diagnostic
      iii) Advanced memory diagnostic (checkerboard – single run and continuous)
      iv) RIM loader from own core memory
      v) ASCII loader from modern computer
      vi) I/O echo with modern computer
   c) DEC official diagnostic programs
      i) Instruction tests 1-3
      ii) Checkerboard memory tests
      iii) Memory protect test
3) DEC programs
   a) Symbolic editor
   b) FOCAL

More official DEC programs can be found online at:
http://bitsavers.informatik.uni-stuttgart.de/bits/DEC/pdp8/
http://www.dbit.com/pub/pdp8/paper/

Appendix D – RS-232 to current loop circuit



Loopback schematic



Final design

Appendix E – Terminal settings

1) General settings
    a. COM1
    b. 110 baud
    c. 8 data bits
    d. No parity
    e. 2 stop bits
2) HyperTerminal
    a. Flow control: none
    b. Settings
        i. Emulate TTY
        ii. Force incoming data into 7 bit ASCII
3) Warren's modified MTTTY windows terminal
    a. Flow control: none, with disabled CTS and DTR control