

# Table of Contents

Stone Knives and Bearskins .....	i
About Your Kit.....	iii
Safety, Component Inventory, and Soldering .....	iv
1: System Power Supply .....	1
2: Data and Address Inputs and Displays .....	3
3: RAM Memory .....	7
4: EEPROM Memory .....	13
5: Input/Output: LCD, Keyboard, and Serial Port Circuits .....	19
6: Reset and System Clock Circuitry .....	23
7: CPU Bus Drivers, Address Decoder, and Debugger.....	27
8: Interface Ports, Final Testing, and Remote Operation .....	33
9: Z80 Machine Language Programming.....	39
10: Loops .....	45
11: Subroutines and the Machine Stack .....	49
12: Controlling Real-World Devices - Traffic Light Controller.....	55
13: Digital to Analog Conversion.....	65



## Stone Knives and Bearskins

In this course we are going to build a microcomputer from scratch and then program this computer to do useful tasks. This will be quite the adventure. It will require you to exercise skill, judgment, and patience to be successful.

You may have friends that have built their own personal computers. When you tell these friends that you're building a computer, they say it's easy. Just buy a motherboard, a case, some RAM, and a hard drive and you're good to go. Your friends only know of pre-made circuit boards, power supplies, and the like. They've never built a computer from actual electronic parts. You mention that it involves soldering. What? Who needs soldering?

In reality, *someone* must design and manufacture computers of all types. And someone must assemble and test machines in order for them to be integrated within systems or sold to end-users. Someone must write the operating system and application software that will make these computers run. Electronics and software professionals are these "someones." At the conclusion of this semester, you will have the experience under your belt to build and troubleshoot simple computer hardware and software.

Your studies will build on the fundamentals of digital logic. Computers are machines that work in an orderly fashion, just like the combinatorial logic and counter circuits you already know about. Everything in the computer world is very systematic and logical. It must be so, because computers are demanding taskmasters. Given inconsistent instructions and data, a computer produces unreliable results. That's not acceptable - especially if that computer is making life-and-death decisions such as in an aircraft autopilot, or in a biomedical device such as an implantable cardiac pacemaker.

We will be using a legacy microprocessor, the Zilog Z80, as our workhorse. This is an 8-bit chip that can access 64k bytes (kB) of memory. That's about 0.000064 GB. And oh yes, this computer has only 32 kbytes of RAM (0.000032 GB). It will run at about 4 MHz (that's 0.004 GHz). Its single accumulator register is 8 bits wide (modern PCs feature multiple accumulators 64 bits or more in size). Yet it will run software very efficiently and reliably.

Our software will be written in the low-level native language of the Z80 CPU. This language uses binary numbers and is called *machine language*. No C++, C#, Java, Visual BASIC, or Javascript for us! All computers require machine language instructions in order to operate; higher-level languages use a program called a *compiler*<sup>1</sup> to translate program instructions into the machine language of the computer's CPU.

The fact is, we don't want to use the latest CPU to build our "scratch" computer. The hardware is too unforgiving. We are using simple chips for that reason. We can repair our computer if we make a mistake. When your friends visit and ask what you're doing with the soldering iron, simply answer the same as Spock in the Star Trek episode *The City on the Edge of Forever*. Spock said that he was "endeavoring ... to construct a mnemonic memory circuit using stone knives and bearskins."<sup>2</sup>

---

<sup>1</sup> Some languages such as BASIC and Javascript use an interpreter program to carry out their instructions. Javascript interpreters are implemented within web browsers.

<sup>2</sup> See <https://www.youtube.com/watch?v=F226oWBHvvI>

## Tips for Success

It will take time and careful work to complete the computer. Almost everything you need is in this kit, and it is supplied with a modern, high-quality circuit board.

**It is very important to follow the instructions exactly as they are shown in this laboratory manual.**

Some important points:

- Use the provided IC sockets for installation of all chips in the kit. Soldering the ICs directly onto the board is a very bad idea. It makes it very hard to replace them in the event of trouble.
- Do not assemble the kit using lead-free solder. Lead-free solder requires a higher soldering iron tip temperature. It is also difficult to remove should you make a mistake during assembly. **Use only 60-40 or 63-37 rosin core solder on the kit.**
- Practice soldering on an old circuit board until you feel confident in your abilities. Learning to solder while building a complicated kit is not a good idea.
- Use anti-static precautions when assembling and working with the computer. An anti-static wrist strap is essential. An errant static discharge can destroy sensitive components.
- Make very sure that parts are in the exact locations called for. This kit uses a wide variety of components, just like any other modern electronic product. *Double-check each component to ensure that it's in the correct orientation and printed circuit board (PCB) location.*
- Do not put in parts until they are called for in the assembly instructions. Otherwise, you won't be able to successfully test the computer stage-by-stage.
- If something doesn't check out, fix it immediately. Don't wait until a later stage; as you add parts to the board, troubles will become harder to identify.
- Watch the provided instructional videos prior to each step. We have provided videos that go beyond the written instructions. These will help you better understand how the computer should be working at each step in the process.

## About Your Kit

### Overview

The EZ-80 is a microcomputer designed around the Zilog Z80400 (CMOS Z80) CPU. The entire computer is designed using generic (and readily-available) CMOS support chips, so it consumes very little power. With the status bar-LEDs installed, it draws about 80 mA; removing these LEDs drops current consumption to less than 30 mA. It will run for many hours from a single 9V transistor radio battery. The Z80 was chosen as it's a legacy processor that still finds application in a wide variety of applications. For example, a derivative of the Z80 is still used as the core processor in many Texas Instruments calculators.

The machine uses a standard 16 character by two-line LCD with software-controlled backlight and has a local keypad for interacting with it directly (no PC is required). Additionally, the unit features an onboard TTL serial port that enables full remote operation, including one-click Intel Hex downloads from a PC computer running the *MiniTerminal* package.

For interfacing to the outside world, the EZ-80 includes two 8-bit output ports and a single 8-bit input port -- that's 24 bits of uncommitted I/O.

The computer features buffered address and data busses. To minimize electrical noise on all busses, a full ground plane is incorporated on the back of the PC board.

The operating system and monitor are contained in 8 KiB<sup>3</sup> of EEPROM. The monitor supports memory editing, software breakpoints, Z80 register editing, a disassembler, and a hardware single-step debugger to allow tracing code one instruction at a time. The user interface runs simultaneously and transparently on both the local machine and a personal computer (if connected). The firmware also includes a rich (and documented) API that allows programmers easy access to all the machine's resources.

### Kit Design Philosophy

This computer is designed for students in high school and undergraduate computer courses that are new to computer technology. As such, the unit is designed to be built and tested one stage at a time. Students learn first-hand how each portion of a classic microcomputer operates -- and at the same time, gain valuable experience with troubleshooting.

The circuit board itself is designed for easy soldering. All components use through-hole technology. Extensive effort has been made to mark all items and sections of the board clearly to minimize the opportunities for "rookie mistakes" and also help students associate the real-world computer components with the theory they're learning in the classroom. Additionally, the board contains only one jumper (EEPROM write enable) -- which makes assembly much less confusing than other kits.

The digital integrated circuits (ICs) are all socketed. This allows easy removal of components should troubleshooting be required -- and also allows graceful recovery in the event an IC is placed in the wrong location. It happens!

Comprehensive assembly, test and troubleshooting information is given at each assembly step to boost success and confidence. For a student new to electronics, identifying components can be a challenge, so we include quality photos and drawings wherever possible.

Minimal equipment is required to conduct most of the tests. A common digital multimeter is all that is needed for most tests; one section makes optional use of an oscilloscope. The remaining test equipment is on the board itself, consisting of two DIP switches for manual bus manipulation and two LED bar displays that reflect status of the address and data busses. The operating system firmware provides a comprehensive power on self test (POST); POST error codes are displayed by flashing the LCD backlight to aid in troubleshooting.

---

<sup>3</sup> The *kibibyte*, or binary kilobyte, symbol KiB (IEC 80000-13), represents 1024 bytes.

## Safety, Component Inventory, and Soldering

Building a computer from the ground up is a complex project. We will walk you through kit construction one stage at a time so that you can be confident that your machine will work when you're done.

We want you to work with safety in mind at all times, so we be providing guidance for protecting yourself and others around you. If you are unsure how to safely perform any procedure described in this instruction manual, stop what you are doing and seek the guidance of an experienced person.

In order to build and test this computer, you will need the following tools and accessories at a minimum:

- An electronics-grade soldering iron that's rated between 20 and 30 watts. The tip should be a chisel shape that's no more than 0.100" (2.54 mm) in width. A temperature-controlled iron is best; set the tip temperature to 400° C (752 °F). Your soldering iron should include a proper rest so that it can be laid down without burning the work surface.
- A sponge pad (lightly dampen with water) for periodically wiping the soldering iron tip. You must keep the soldering iron tip clean to produce good solder joints.
- Electronics grade rosin-core solder. For best results, use so-called "60/40" or "63/37" solders. These solders are an alloy of 60 or 63 percent tin, and 40 or 37 percent lead. Lead-free solder will work, but we don't recommend it for kit building. It can be very difficult to remove lead-free solder in the event that you must undo a mistake. Use small-caliber (0.050" / 1.25 mm or less) solder. You'll find it works much better on the small solder joints of this kit!
- An anti-static wrist or ankle grounding band. This item is absolutely necessary for you to wear while handling any electronic components in the kit.
- Personal protective equipment. You must wear ANSI Z87.1 (2020) rated safety glasses (or equivalent) whenever you are soldering, cutting, or performing any other construction tasks on the kit. Additionally, you must work in a well-ventilated environment when soldering so that you don't inhale solder fumes, which are toxic. You may also use a desktop exhaust fan employing an activated-carbon scrubber.
- Adequate lighting. If you can't see what you're doing, you will make mistakes. You may also endanger yourself or others.
- A safe environment for soldering. Such an environment has adequate lighting and ventilation, as well as a safe, non-flammable surface upon which soldering will be performed.
- Basic hand tools (pliers, screwdrivers, electronics (fine) wire cutter, and so forth).
- A general purpose digital multimeter for checking continuity and DC voltages. You can often obtain a very serviceable meter for under \$10 at hardware and surplus stores.

The following items are nice to have, but not necessarily essential:

- An illuminated magnifier. This helpful for working with small components and soldering tiny joints (there are about 500 solder joints in your kit).
- A "third hand." This is a special vise designed to hold electronic components such as circuit boards while they are being worked on.

Finally, to load the 28C64 EEPROM with the EZ-80 operating system, you will need access to a suitable programmer, such as the [True-USB PRO GQ-4X](https://mcumall.com/store) available from <https://mcumall.com/store>. The binary ROM image may be downloaded from our website at <http://n0gsg.com/EZ80>. If you are building this computer as a school project, your department should be able to help you program the EEPROM.

## Safety Practices

Before we delve into the components of the kit, let's talk about a few important safety practices. **Please don't skip this section.**

### Component Safety - Electrostatic Discharge (ESD)

Did you know that most of the semiconductors (chips) in your kit can be damaged by simply touching them? This is because your body normally carries thousands of volts of electric charge. This is why you often get a shock in winter by touching a doorknob or other metal object.

To neutralize this charge, you must wear an anti-static wrist band that is connected to Earth ground. These bands are usually available for under ten dollars.

The parts of your kit that are sensitive are the integrated circuit "chips" (ICs) that form the bulk of its circuitry. Some chips that have been "zapped" by ESD may survive and work normally, only to fail in random fashion at a later time. Others may immediately die. And a small fraction of chips will survive and never fail.

If you don't wear an anti-static wrist or ankle band while building this kit you are gambling with your success. Please wear one!

Keep the electronic parts in your kit in the original packaging until you need them. Wrapping them in a sheet of tin foil, or placing them in a box lined with tin foil, is also excellent protection.

Oh yes, just in case your buddies want to admire your work, make sure they are grounded before they touch your precious board!

### Soldering Safety

A soldering iron is very hot. Hot enough to easily burn you (duh!), but also hot enough to start a fire if it contacts flammable materials. Use a proper rest for the iron to prevent work surface damage.

Always unplug or turn off your iron when it's not in use. And NEVER walk away from a workstation when the iron is still on!

NEVER "flip" the iron to remove excess solder from the tip. Carefully wipe the tip across the dampened pad.

ALWAYS wear your eye protection when soldering. Solder can unexpectedly bubble and splash when it's hot.

*By the way, if it takes more than about two seconds to solder any joint, it's taking too long.* Leaving the iron on the board more than two seconds or so will overheat the delicate circuit pads and traces. It's counterintuitive, but a "hot" iron (running at the temperature we recommend, 400° C (752 °F), is less likely to burn the circuit board. This is because the "hot" iron transfers heat more readily to the board, so joints are soldered more quickly.







When you leave the work area, wash your hands thoroughly. The residues of soldering are toxic and you do not want to ingest them.

## Component Inventory

Before we light up that soldering iron, let's get familiar with the components in our kit. That way we will know what the parts look like before we are in the middle of a procedure.

*Remember to use ESD-safe practice any time you're working with the parts in the kit. Check off each component as you go. Sometimes you will see the same component in many places in your kit.*

### **Mechanical Components**

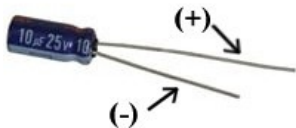

No.	Component	Picture and/or Explanation	Quantity in Kit	Present?
1	Printed Circuit Board X-EZ80100		1	
2	Header, 16 position	 This item has 16 pins that are either silver or gold. It will be used to connect the LCD display to the main PC board. <u>The short pins will be soldered to the LCD.</u>	1	
3	Header, 6 position right-angle	 This six-pin connector will be used to attach the USB TO TTL serial adapter. <u>The short pins will be soldered to the main PC board.</u>	1	
4	Header, 1 position	These two single-pin headers will be soldered to the 5V and GROUND test points on the circuit board. Your kit may contain a two-position header; cut it into two pieces.	2	
5	Barrel Connector, PJ-102A		1	
6	AC Adapter, 9~12VDC output @ 500 mA		1	
7	Heatsink, TO220 clip-on	 The heatsink will clip onto U22, the voltage regulator, to help keep it cool.	1	



## Capacitors

Your kit uses capacitors for two purposes, filtering the DC power supplied to each IC chip (to ensure that it's pure and free of noise), and to control timing either in conjunction with a resistor or crystal. It's very important that you correctly identify capacitors and install them in the proper locations.

Most of the capacitors have no polarity and can be installed either way. Electrolytic capacitors look like tiny "cans" and must be installed only one way. Electrolytics are marked with a "+" or "-" on the case to show polarity. *The longer lead is always the "+" or positive connection.* **Electrolytic capacitors that are installed backwards may get very hot and explode.** You have been warned!

No.	Component	Picture and/or Explanation	Quantity in Kit	Present?
8	Electrolytic capacitor, 10 $\mu$ F / 25 V		3	
9	Disc or Mylar capacitor, 0.1 $\mu$ F / 50 V	 These parts are not polarized and may be installed in either direction. May be marked "0.1" or "104"	19	
10	Disc capacitor, 22 pF / 50 V	These look identical to the other disc capacitors but will be marked "20," "22," or "22J". <u>It is very important to separate these two parts from the other capacitors as they are critical to the operation of your kit.</u>	2	

## Integrated Circuit (IC) Sockets



Your kit uses sockets for all IC chips except the voltage regulator. IC sockets have an indentation at the end that will be aligned with the marked outline on the circuit board. Please do not solder the ICs directly onto the PC board. It is very difficult to replace ICs that are not socketed.

No.	Component	Circuit Locations	Quantity in Kit	Present?
11	DIP Socket, 14 pin	U5, U6, and U8	3	
12	DIP Socket, 16 pin	U1, U3, U7, U17, S32, S33, SO1, SO2, SO3	9	
13	DIP Socket, 20 pin	U4, U11, U12, U13, U14, U16, U18, U19, U20, U21, D2, D4	12	
14	DIP Socket, 28 pin	U9, U10	2	
15	DIP Socket, 40 pin	U2 (Z80 CPU)	1	

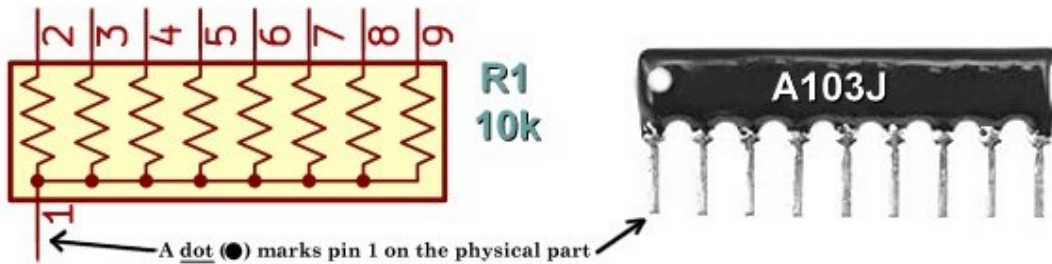
## Resistors

Resistors are used to control the flow of electrical current for many purposes. Your kit has two styles of resistors, axial and single-inline-package (SIP).

Axial resistors have a body containing a color code with leads at the ends. They have no polarity, but it is very important to install the correct value at each location on the board. *Use an ohmmeter to confirm their value if you're in doubt!*




SIP resistors contain multiple identical resistors in a single package. You'll find them extensively used in the kit because they're compact, reliable, and easy to work with. All the resistors in a SIP have one end connected to a "common" terminal as shown on the schematic symbol. On the component itself, there is a dot marking this common pin as well. We have placed a dot next to the SIP resistors on the PC board to help you identify the correct direction.





On the SIP resistor above, the value is given as "103." The first two digits ("10") are part of the value; the third digit ("3") gives the number of zeros to write after the value. Each of the eight resistors inside this part has a value of 10,000  $\Omega$  (10 k $\Omega$ ). You can verify this with an ohmmeter, of course.

**IMPORTANT**  
**For your kit to work, you must have all SIP resistors placed in the correct direction, with correct values. Please double-check each SIP resistor as you install it to be sure.**

No.	Component	Circuit Locations	Quantity in Kit	Present?
16	SIP Resistor, 10k	R1, R10 (Marked "103")	2	
17	Axial Resistor, 4.7M	R2 (Yellow-Violet-Green)	1	
18	Axial Resistor, 10 k $\Omega$	R3, R4, R16 (Brown-Black-Orange)	3	
19	SIP Resistor, 47 k $\Omega$	R5, R9, R14 (Marked "473")	3	
20	Axial Resistor, 22 $\Omega$	R6, R7, R8 (Red-Red-Black)	3	
21	Axial Resistor, 100 $\Omega$	R11 (Brown-Black-Brown)	1	
22	Potentiometer, 10 k $\Omega$	 R12 (Marked "103"; used to set LCD contrast.)	1	
23	SIP Resistor, 470 $\Omega$	R13, R15 (Marked "471")	2	
24	Axial Resistor, 100 k $\Omega$	R17 (Brown-Black-Yellow)	1	
25	SIP Resistor, 4.7 k $\Omega$	R18, R19 (Marked "472")	2	


## Switches

We will provide input to our computer via various switches. There are two styles of switches in the kit, pushbutton and dual-inline-package (DIP). The DIP switches contain eight individual switches in each package. They allow us to put digital data (1s and 0s) on the data and address busses during testing.

No.	Component	Circuit Locations	Quantity in Kit	Present?
26	DIP Switch 8 Station	 S32, S33	2	
27	Tactile Switch SPST	 S1~S31	31	

## Timing Crystal

Your computer requires an accurate timing source. The crystal works like a clock pendulum; it vibrates at a very precise frequency, 3,579,545 oscillations per second (3.579545 MHz). This synchronizes all the computer's circuits. The crystal has no polarity and may be installed in either direction. Handle it gently!

No.	Component	Circuit Locations	Quantity in Kit	Present?
28	Crystal, 3.579545 MHz Parallel Mode	 Y1	1	

## Semiconductors (Integrated Circuits and Diodes)

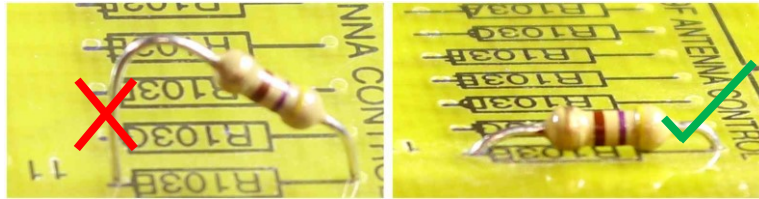
The ICs in the kit form the circuitry of the computer. In particular, U2, the Z80 CPU, is the workhorse. It carries out each instruction in our programs. Semiconductor diodes D1 and D3 are used to control the direction of current flow -- they form a "one way street." Light emitting diodes (LEDs D2 and D4) convert electrical current to light so that we can see the status of various digital signals in the circuit. Be sure to follow ESD precautions when handling all semiconductors.

No.	Component	Circuit Locations	Quantity in Kit	Present?
29	74HC138	U1, U3, U17	3	
30	Z80400 CMOS CPU	U2	1	
31	74HC245	U4, U20, U21	3	
32	74HCT08 / 74HC08	U5	1	
33	74HCT00 / 74HC00	U6	1	
34	74HC4017	U7	1	
35	74HC04	U8	1	
36	CY62256-70PC	U9 (RAM)	1	
37	28C64	U10 (EEPROM)	1	
38	74HCT574	U11, U13, U14	3	
39	74HCT573	U12, U16, U18, U19	4	
40	GM1602	U15 (LCD Display)	1	
41	L7805CV	U22 (Voltage Regulator)	1	
42	1N4001	D1, D3	2	
43	LTA-1000E	D2, D4 (LED Light Bar Array)	2	

## How to Solder Correctly

Your kit uses a printed circuit (PC board). A PC board contains all the components along with the copper conductors that complete the circuit. Through-hole components are used in this kit.

Through-hole components are mounted to a PC board by inserting their leads through the board's predrilled holes. Most components are generally seated flush with the board. Below we show how resistors should be mounted. The right-hand picture is correct: Both leads are of equal length, centering the part on the board.

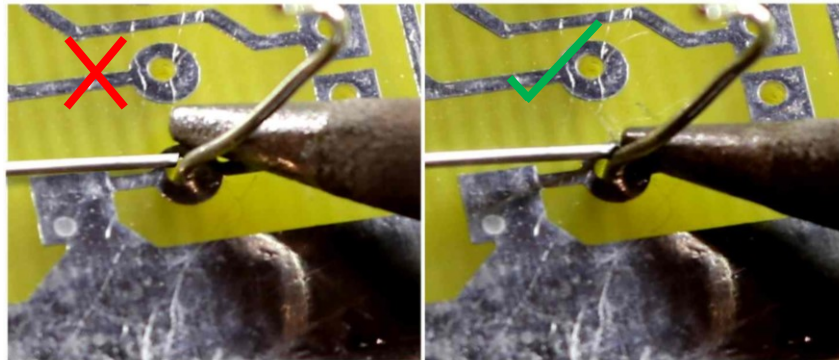


When populating a board, seat, then solder, just one or two components at a time. Always double-check that you are placing the correct component and value in the correct position before soldering. Some parts such as electrolytic capacitors, diodes, and transistors are polarized; mount these in agreement the PC board's silkscreen pattern. Mount all components flush with the board unless you are otherwise instructed.

### Soldering Technique

Soldering parts on a board is a very fast operation. In general, each joint should take three seconds or less to complete. Spending more time does not create better solder joints - - in fact, overheating the board may cause the copper to delaminate, or separate, from the board, which will require further repair.

The basic technique is shown below. The most important principle is that the component lead and copper pad be heated simultaneously. All parts of the joint must be heated for effective soldering.



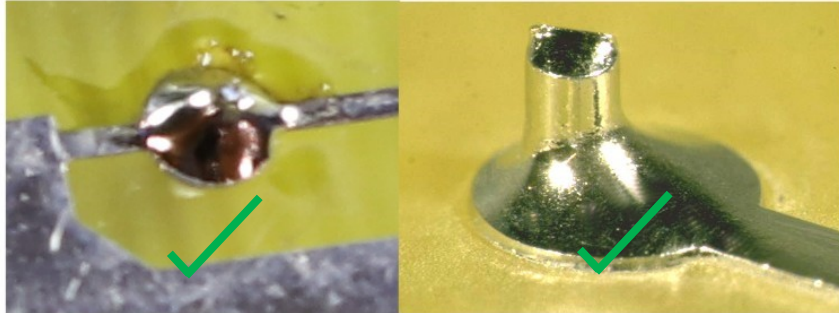
The left-hand picture is the wrong way. The tip of the iron is touching the component's lead wire, but not the copper pad on the board. Only the component lead will be heated. This will produce a poor solder joint.

On the right, the soldering iron touches both the pad and lead; this is the correct method. The tip is pressed firmly but lightly to make good thermal contact with both pad and lead, and the solder is introduced (from left in the figure) slowly. It's okay to touch the solder to both the tip and joint to get things moving, but you should feed solder primarily to the joint, not the tip, in order to ensure full surface wetting. Make sure you don't spend more than three seconds heating and soldering each joint.

The circuit board in your kit is of high quality and is very rugged. It will withstand most mistakes as long as its copper circuit pads are not overheated. Remember, most joints should be completed in 3 seconds or less.

### *Appearance of Good Solder Joints*

Two good solder joints are shown below. Their surfaces appear smooth, shiny, and continuous. Both have just the right amount of solder - - just enough to form a rounded shape, called a *fillet*, where the lead passes through the hole. If your joints look like this, good work!



Below are some bad solder joints. Both joints at left (top and bottom) are *cold joints* with *solder voids*. The solder and joint were not heated sufficiently for the solder to wet both the PC board conductor and component lead. Cold joints can be corrected by simply reheating the joint and adding a small amount of fresh solder until the molten solder flows smoothly and fills any voids.



The top middle joint has *insufficient solder* - - the solder is not contacting both the lead and the circuit pad. This joint will be mechanically weak and may not even be electrically connected. To correct this joint, reheat and add solder until a proper fillet has formed around the lead. At top right, we see a *solder ball*. This joint also has excess solder. This is typically caused by not heating both the circuit pad and component lead, causing the solder to adhere only to the component lead. To correct, reheat the joint, making sure to heat both the component lead and solder pad. If excess solder remains, you may need to remove it using solder wick or a solder vacuum tool. The joint at bottom right is a *cold joint caused by component lead movement during cooling*. To fix this problem, simply reheat the joint until the solder flows, then allow it to cool, making sure not to allow any movement.

## *How to Fix Solder Bridges*



A problem you may eventually run into is the *solder bridge* shown above. These are easy to create when soldering closely-spaced pins.

Your circuit board has green *solder resist* printed between all adjacent solder points to minimize the chance of creating solder bridges.

If you create a solder bridge, don't panic. Simply position a bit of solder wick over the bridge and heat the solder wick with your iron. The solder wick will magically soak up the solder from the bridge. Be sure not to overheat the board or components when correcting solder bridges.

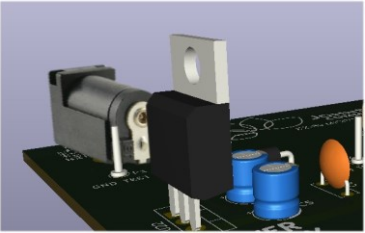
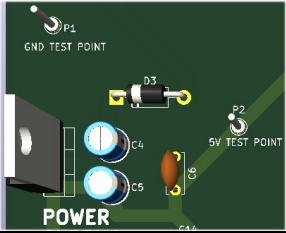
**Using too much solder is a good recipe for solder bridges; use only the amount of solder required to get a clean, smooth joint. No more and no less!**

## 1: System Power Supply

The power supply section regulates the incoming DC from the "wall wart" power pack to a steady 5 volts. It's very important for the proper operation of your computer!

### Assembly Procedure

We recommend that you place a check mark by each step as you complete it. All components in this kit mount on the top (component) side of the circuit board.

Step No.	Description	Completed
1	Clean both the top and bottom of the PC board using isopropyl alcohol (90% or higher concentration). This will eliminate any residues that could interfere with soldering. <u>From here on, try to avoid touching the PC board surfaces.</u>	
2	Install the <u>power connector J1</u> . You will find it easy to install if you proceed as follows:  a) Insert the connector into the printed circuit from the top (component) side.  b) Holding the connector in place, turn the board upside down and lay it on the worktable. (The connector will be contacting the work surface and will stay in place.)  c) Solder the three connector pins.	
3	 <p>Install the voltage regulator U22 (L7805V) as shown in this picture. Be sure to seat it fully on the board before soldering.</p>	
4	 <p>Install 10 µF capacitors C4 and C5. Note polarity - - the (-) negative terminal faces right as shown in the picture.</p>	
5	Install the 0.1 µF capacitor C6. This part may be marked "104" and has no polarity, so it may be installed in either direction.	
6	Install the test points P1 and P2 as shown in the picture. You may need to carefully cut these from a larger header (2 or 18 pin) as supplied in the kit.	
7	Install diode D3 (1N4001). <u>This part is polarized</u> - - the line on the part must face left as shown on the PC board silk screen and image above.	

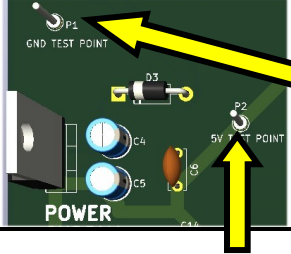
Congratulations! You've completed the power supply. Now let's test it to see if it's working.

## Test Procedure

Name \_\_\_\_\_

Please don't connect the board to power until instructed to do so.

Sign-Off \_\_\_\_\_

Step No.	Description	Check or Measurement Result
1	Power up your digital multimeter (DMM) and set it to measure DC VOLTS. We recommend that you attach alligator clip leads to the meter.	
2	 <p>Connect the black (GROUND) lead of the DMM to the P1 GND TEST POINT on the board.</p> <p>Be careful not to short the lead against anything else on the board!</p>	
3	Connect the red (HOT) lead of the DMM to the P2 5V TEST POINT.	
4	Connect the AC adapter to the board and plug it into an outlet.	
5	<p>Observe the meter reading. It should be <math>5\text{ V} \pm 0.25\text{ V}</math>. Recording your reading at right.</p> <p><b><u>If this reading is correct, congratulations! Your power supply section is in good working order.</u></b></p>	

## Troubleshooting Hints

Please check the following if your power supply section doesn't seem to be working correctly.

### Getting a zero or very low reading?

1. Is the DMM set to the correct range (DC volts)?
2. Is the AC power adapter plugged into a live outlet?
3. Is the DMM healthy? Connect it to a known good DC source to check it.
4. Are all the components soldered in place correctly? (For example, inserting D3 backwards will stop this section from working.)

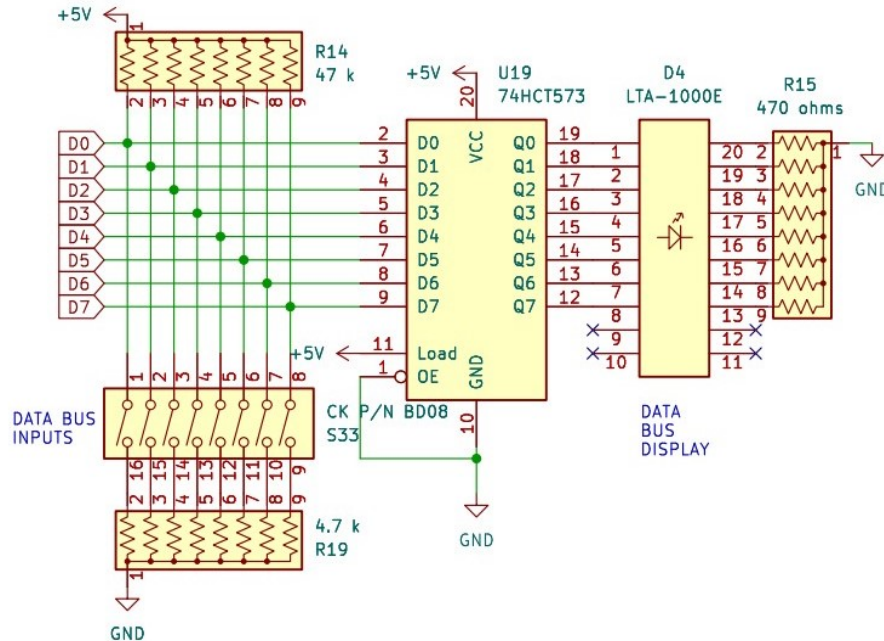
### Getting a reading that's too high?

1. Is U22 installed in the correct direction? (It won't work as a regulator if it's installed backwards!)
2. Are all three leads (especially the center leg, ground) soldered correctly?



## 2: Data and Address Inputs and Displays

The data and address inputs and displays allow us to test our computer one section at a time. These two sections are nearly identical. Each consists of an 8-station DIP switch, pull-up and pull-down resistors, and a display driver. The schematic of the data section is shown below.



### Input Section Operation

Resistor R14 is a pull-up resistor. Because it's a high value (47 kΩ), it can only supply a very small current to the circuit. That's okay, because U19 and the other ICs in our computer are CMOS chips. They draw almost zero current at their inputs! When any of the eight switches in S33, the DIP switch, are in the OPEN position (as the schematic shows), the resistors in R14 pull up the associated data line to a 5 volt level, making a logic "1." The voltage for logic HIGH should be very close to 5 volts.

When a switch within S33 is CLOSED, R19 is introduced into the circuit to produce a logic LOW. The corresponding resistors in R14 and R9 form a voltage divider. The output voltage for the logic LOW is:

$$V_{\text{LOW-LEVEL}} = V_{CC} \times \left( \frac{R19}{R19 + R14} \right) = 5V \times \left( \frac{4.7 \text{ k}\Omega}{4.7 \text{ k}\Omega + 47 \text{ k}\Omega} \right) = \underline{\underline{455 \text{ mV}}}$$

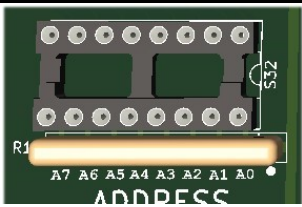
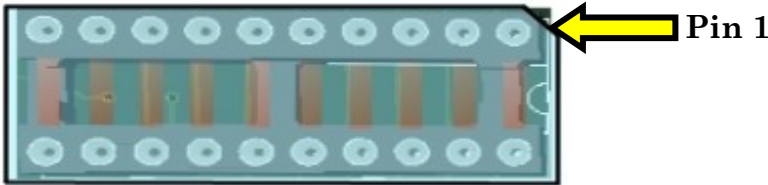
The CMOS chips within our computer will accept anything less than 1.5 V as a logic LOW, so this circuit works just fine. Later on, we will install the CPU (U2) and bus driver chips (U4, U20, and U21). These ICs will have outputs that are much "stronger" (lower output resistance) than any of the resistors in the data and address entry sections. This way, when we install the CPU and bus drivers, the DIP switches will no longer have any effect on the computer.

### LED Driver

The signals on the data and address busses are too weak to drive LEDs directly. U19 boosts the current on each bus line to a level that can operate the LEDs without loading down the bus. R15 limits the current in each LED to around 20 mA.

## Assembly Procedure

We recommend that you place a check mark by each step as you complete it.

Step No.	Description	Completed
1	Locate and install R5, R9, and R14, the 47k SIP resistors. These will be marked "473." <b>BE ABSOLUTELY SURE YOU ARE INSTALLING THESE RESISTORS IN THE CORRECT LOCATIONS WITH CORRECT ORIENTATION (DOT ON EACH PART MATCHES DOT ON PC BOARD).</b>	
2	Locate and install R18 AND R19, the 4.7k SIP resistors. These will be marked "472."	
3	Locate and install R13 AND R15, the 470Ω SIP resistors. These will be marked "471."	
4	 <p>Install 16-pin IC sockets in the S32 and S33 positions on the board. Note that the notch in the socket must align with the PC board silkscreen legend as shown.</p>	
5	Install 20-pin IC sockets in the U18, U19, D2, and D4 positions on the board.	
6	 <p>Install the two LTA-1000E LED Bar Arrays in the D2 and D4 sockets. <b>Note the orientation very carefully!</b> There is an <b>index mark</b> on the corner adjacent to pin 1 on these parts. This mark is very subtle - - look carefully.</p>	
7	<p>Install U18 and U19, the 74HCT573 octal transparent latch ICs into their sockets. Be sure you've aligned the notch on each IC with the PC board legend. <b>Also, make sure you are installing the 74HCT573 chips in these positions, and not the 74HCT574s that are used in subsequent steps!</b></p> <p><i>TIP: You'll find it much easier to install ICs if the pins are straightened! At your ESD-safe workstation, simply lay each IC on each SIDE, then gently push the IC onto the surface to straighten the pins.</i></p>	
8	<p>Install the two DIP switches into the S32 and S33 sockets.</p> <p>The word "ON" on the switches should face towards the legends "DATA INPUT" and "ADDRESS INPUT." (When a switch is ON, it is CLOSED, making a logic LOW.)</p>	

You've completed the data and address input section. Now let's test it to see if it's working.

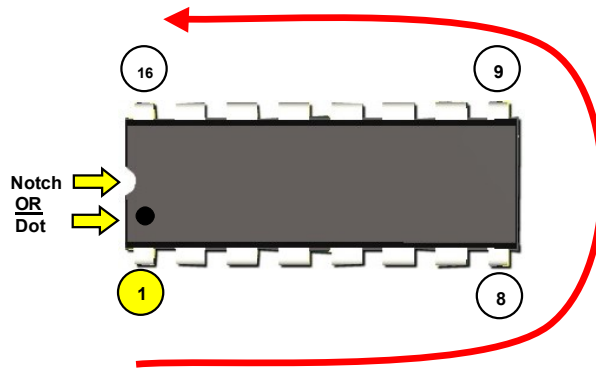
## Test Procedure

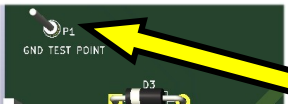
Name \_\_\_\_\_

Please don't connect the board to power until instructed to do so.

Sign-Off \_\_\_\_\_

*TIP: During the test procedure you'll be reading voltages on integrated circuit pins. IC pins are numbered as shown below for a 16-pin DIP package. Pin 1 is always adjacent to the notch or dot as shown.*



Step No.	Description	Completed																		
1	Power up your digital multimeter (DMM) and set it to measure DC VOLTS. We recommend that you attach an alligator clip lead to the meter negative input, and a "needle" probe to the positive or hot input.																			
2	 <p>Connect the black (GROUND) lead of the DMM to the P1 GND TEST POINT on the board.</p> <p>Be careful not to short the lead against anything else on the board!</p>																			
3	<b>Apply power to the board.</b> Exercise all DIP switches and verify that the associated LEDs turn off and on to match the DIP switch inputs. <i>(If this is not working, check the <b>Troubleshooting Hints</b> on the next page.)</i>																			
4	<p>Verify the logic voltage level for a logic LOW for all DATA inputs. Move all DATA input switches to the ON position, then measure the inputs at U19. <u>These should measure less than 0.5 V.</u></p> <p><i>TIP: Refer to the schematic diagram to find all U19 input pin numbers.</i></p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">U19 Pin #</th> <th style="text-align: left;">Logic "0" Voltage Measured</th> </tr> </thead> <tbody> <tr><td>D0 - 2</td><td></td></tr> <tr><td>D1 -</td><td></td></tr> <tr><td>D2 -</td><td></td></tr> <tr><td>D3 -</td><td></td></tr> <tr><td>D4 -</td><td></td></tr> <tr><td>D5 -</td><td></td></tr> <tr><td>D6 -</td><td></td></tr> <tr><td>D7 -</td><td></td></tr> </tbody> </table>	U19 Pin #	Logic "0" Voltage Measured	D0 - 2		D1 -		D2 -		D3 -		D4 -		D5 -		D6 -		D7 -		
U19 Pin #	Logic "0" Voltage Measured																			
D0 - 2																				
D1 -																				
D2 -																				
D3 -																				
D4 -																				
D5 -																				
D6 -																				
D7 -																				

*(Test procedure is continued on the next page.)*

## Test Procedure (Continued)

5	Verify the logic voltage level for a logic HIGH for all DATA inputs. Move all DATA input switches to the OFF position, then measure the inputs at U19. <u>These should measure <math>5V \pm 0.25 V</math>.</u>  <table border="1" data-bbox="326 401 1027 737"><thead><tr><th data-bbox="326 401 678 432"><u>U19 Pin #</u></th><th data-bbox="678 401 1027 432"><u>Logic "1" Voltage Measured</u></th></tr></thead><tbody><tr><td data-bbox="326 432 678 464">D0 - 2</td><td data-bbox="678 432 1027 464"></td></tr><tr><td data-bbox="326 464 678 495">D1 -</td><td data-bbox="678 464 1027 495"></td></tr><tr><td data-bbox="326 495 678 527">D2 -</td><td data-bbox="678 495 1027 527"></td></tr><tr><td data-bbox="326 527 678 558">D3 -</td><td data-bbox="678 527 1027 558"></td></tr><tr><td data-bbox="326 558 678 590">D4 -</td><td data-bbox="678 558 1027 590"></td></tr><tr><td data-bbox="326 590 678 621">D5 -</td><td data-bbox="678 590 1027 621"></td></tr><tr><td data-bbox="326 621 678 653">D6 -</td><td data-bbox="678 621 1027 653"></td></tr><tr><td data-bbox="326 653 678 684">D7 -</td><td data-bbox="678 653 1027 684"></td></tr></tbody></table>	<u>U19 Pin #</u>	<u>Logic "1" Voltage Measured</u>	D0 - 2		D1 -		D2 -		D3 -		D4 -		D5 -		D6 -		D7 -		
<u>U19 Pin #</u>	<u>Logic "1" Voltage Measured</u>																			
D0 - 2																				
D1 -																				
D2 -																				
D3 -																				
D4 -																				
D5 -																				
D6 -																				
D7 -																				
6	Repeat the tests of Steps 4 and 5 to confirm operation of the ADDRESS inputs on U18.																			

If it all works...fantastic! You're ready to move on to bigger and better things!

## Troubleshooting Hints

Please check the following if your data and address input section doesn't seem to be working correctly.

### LEDs not Lighting?

1. Is the power supply good? (Repeat the +5V measurement of the prior section.)
2. Are the LED bar arrays (D2 and D4) inserted in the correct direction?
3. Are the correct ICs installed for U18 and U19?
4. Are all the SIP resistors of the correct value and oriented correctly? (We sure hope so!)

### LEDs Stuck On?

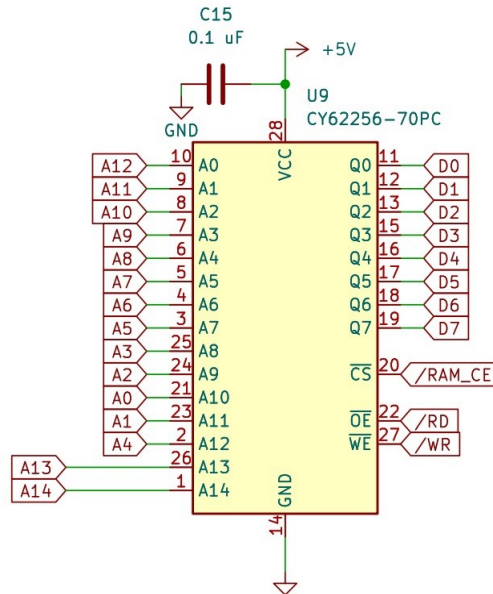
1. Check for a solder ball short in the LED and/or U18/U19 areas.
2. Are the correct ICs installed for U18 and U19?

### Logic Voltage Levels Wrong?

1. Are R14, R19, R5, and R18 in the correct positions?
2. Are the solder joints on S32 and S33 okay?

### 3: RAM Memory

The random access memory, or RAM, is a key part of a microcomputer. This high-speed memory holds the variable data needed by programs running on the computer. It also contains the stack, which is a reserved area for temporary information. Finally, your own programs will be downloaded into the RAM to be run on the computer.



#### How the RAM Works

The operation of the RAM in your computer is very simple. In the schematic above, you can see that the RAM chip, U9 has several groups of signals connected to it. These groups are power, address, data, and control signals.

#### Power

5 volt power is supplied on Pin 28 and this power is filtered by C15 to keep it clean. Every logic IC in your computer has a 0.1  $\mu\text{F}$  capacitor nearby to keep its supply free of AC noise. (*Oh, that's why there are so many of these capacitors in the kit!*)

#### Address

The 32 KiB RAM has  $2^{15}$  or 32,768 individual bytes that the CPU can access. The address inputs A0 through A14 come from the CPU and drive the address inputs of the RAM chip. You will notice that the address signals seem a bit "scrambled." For example, the CPU's A12 connects to A0 (Pin 10) of the RAM. This seems rather bizarre! We designed the RAM section this way so that your computer can work with either the 8 KiB part (HM6264) or the 32 KiB part (CY62256) in U9's place.

In short, the CPU will send the proper binary address out on A0~A14 when it's accessing memory. The RAM's job is to serve up the proper data location when that address is received.

#### Data

The RAM receives from the CPU (or sends it back to the CPU) on the eight bi-directional, tri-state data signals D0 through D7. These eight data signals are shared with every device in your computer.

### *Control Signals*

The control signals tell the RAM what to do. First and most important, the **/RAM\_CE** signal on Pin 20 tells the RAM if the CPU wishes to talk with it. The **/RAM\_CE** signal arises from the **address decoder** circuit of your computer. The address decoder enables /RAM\_CE only when the CPU emits a memory address we've allocated for the RAM. In your computer, the RAM uses the 32,768 memory locations from 8000H (H means hexadecimal) to FFFFH. If the CPU asks for any address outside this range, **/RAM\_CE** will remain inactive (high). In turn, the RAM chip ignores requests for addresses that are not assigned to it.

The **/RD** (read) and **/WR** (write) signals come from the CPU and tell the RAM what the CPU needs to do with the RAM's data. If the CPU wishes to read data from the RAM, it will output a proper address causing **/RAM\_CE** to become active, *and* it will activate the **/RD** signal. The RAM's **/OE** (output enable) signal on Pin 22 is connected to **/RD**, so the RAM activates its tri-state data outputs, which pushes the selected RAM data onto the data bus for the CPU to read.

Writing data works almost identically. The CPU will send out an address like before, but for writing, the CPU will place the data to be stored on the data bus, then activate the **/WR** signal, which activates the **/WE** input of the RAM on Pin 27. This tells the RAM to "listen" on the data bus and accept the CPU's data.

To summarize, **/RAM\_CE** tells the RAM it's part of a conversation with the CPU. **/RD** tells the RAM to talk to the CPU, and **/WR** tells the RAM to listen and accept data. The address lines tell the RAM where to store or retrieve the data.

Your kit includes three pushbutton switches for simulating the RAM's control signals. These are marked **/RAM\_CE**, **/WR**, and **/RD** on the board. They will allow us to confirm the operation of the RAM without the CPU and address decoder circuits installed.

### *RAM Data are Volatile*

The RAM in your computer needs a constant supply of power to retain its data. When power is removed, data in the RAM will be lost. This characteristic is common for almost all RAM memories.

## Assembly Procedure

We recommend that you place a check mark by each step as you complete it.

Step No.	Description	Completed
1	Install a 28-pin socket for the RAM in U9's position.	
2	Locate and install R1, a 10k SIP resistor. This will be marked "103." <b>Be sure to align the dot on the part with the dot on the board.</b>	
3	Locate and install C2 AND C3, the 22 pF capacitors. These will be marked "22" or "20." These capacitors are not part of the RAM circuit; installing them is a good way to prevent them from being lost.	
4	Install all the remaining 0.1 $\mu$ F capacitors in positions C7, C8, C9, C10, C11, C12, C13, C14, C15, C16, C17, C18, C19, C20, C21, C22, C23, and C24. These parts may be marked "0.1" or "104." <u>This should account for all ceramic disc capacitors in your kit.</u>	
5	Install pushbutton switches S2, S3, and S4.	
6	Carefully install the CY62256 RAM IC into the socket in position U9.  TIP: This IC may be marked "AS6C62256" or something similar. If you see "62256" as part of the number of the IC, you are installing the correct part.  <u>Straighten the pins on this IC before inserting</u> (remember, press the IC sideways against your ESD-safe worksurface.)  <b>Although it will take a bit more force to install the 28-pin RAM, try not to force the IC into the socket. <u>You can easily bend or break off IC pins.</u></b>  <b>Make sure all pins are mating correctly before pressing the chip into position.</b>	

You've completed the RAM circuitry. That was pretty easy, wasn't it? Now let's test it!

## Test Procedure

Name \_\_\_\_\_  
Sign-Off \_\_\_\_\_

Step No.	Description	Completed
1	<b>Apply power to the board.</b> Exercise all DIP switches and verify that the associated LEDs match the DIP switch inputs. ( <i>If this is not working, check the <b>Troubleshooting Hints</b> on the next page.</i> )	
2	Apply the address [0000 0000]. All the ADDRESS BUS LEDs should be off.	
3	Apply the data pattern [0000 0000]. All the DATA BUS LEDs should be off.	
4	Select the RAM by holding down the <b>/RAM_CE</b> switch, S4.  While holding down S4, activate the read signal ( <b>/RD</b> ) by holding down switch S3.  Because you haven't stored anything in RAM, and RAM is volatile, you're like to see a random byte displayed. Record what appears on the DATA BUS during the read operation.  Data Bus Pattern: [                    ]  <i>Release S3 when you're done with this step.</i>	
5	Write your data into the RAM by holding down the <b>/RAM_CE</b> switch, S4, then momentarily press <b>/WR</b> (S2). You've just written data into RAM location [0000 0000].  <i>Don't forget to release the <b>/WR</b> switch!</i>	
6	Recall the data in location [0000 0000] as you did in Step 4. Record the result below.  Data Bus Pattern: [                    ]	
7	Release the pushbutton switches, then enter the next address, [0000 0001].  Enter the data [0000 0001] into this RAM address.	
8	Enter unique data into addresses [0000 0010] through [0000 1000]	
9	Verify the data held in each address by entering the address onto the bus, then activating both <b>/RAM_CE</b> and <b>/RD</b> .	
10	Remove power from the computer (wait about a minute with power off), then reapply power. Repeat Step 9. What happened to the data?	

Did everything work correctly? If so, congratulations! You've just verified that the data, control, and address busses as well as the RAM memory are healthy in your computer. That should make you feel great as this is a major accomplishment. Onward we go!



## **Troubleshooting Hints**

*Please check the following if your data and address input section doesn't seem to be working correctly.*

### LEDs not Lighting?

1. Is the power supply good? (Repeat the +5V measurement of the prior section - if it's not good, check that the power adapter is in a live outlet, and that there are no solder shorts on the back of your board.)

### RAM Data aren't Recalling Correctly

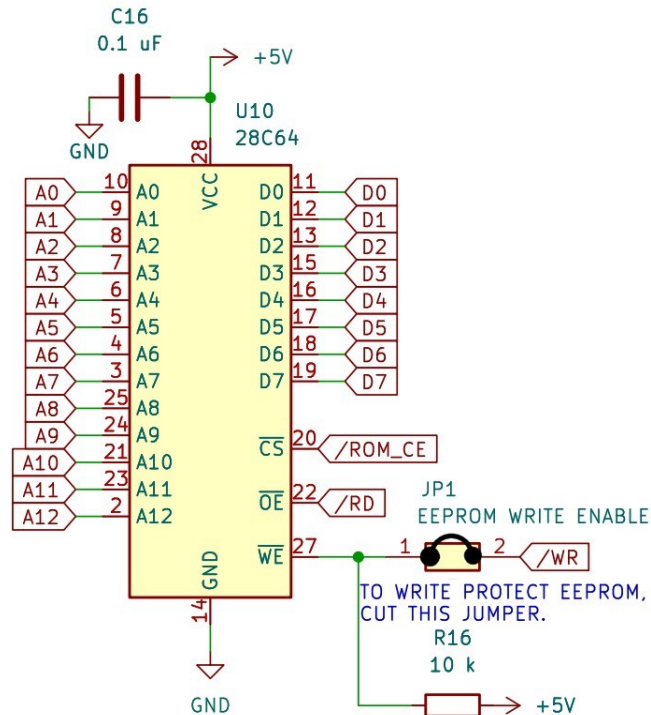
1. Double-check that you've written the correct data into the memory.
2. Did you install the correct IC in U9's position?
3. Does U9 have good Vcc (Pin 28) and GROUND (Pin 14)? Measure with a DMM to be sure.
4. Are all solder joints on U9 okay?
5. Make sure you don't have any solder ball shorts on any ICs you've installed so far.



## 4: EEPROM Memory

From our last experience, we know that the RAM memory in our computer is volatile. That is, the RAM loses its information when the power is removed. Because we only use the RAM for temporary information, this is not a problem.

However, when your computer is first turned on, it needs to have a reliable way of knowing what to do. Without available instructions, a computer is just a fancy paperweight. The EEPROM (Electrically Erasable Programmable Read-Only Memory) is a special chip that retains information permanently - - even when the power is turned off. We say that the EEPROM is a non-volatile form of memory. It's very similar to the flash memory used in USB memory sticks.



### How the EEPROM Circuitry Works

Just like the RAM, the EEPROM is connected to four groups of signals - - power, address, data, and control signals.

#### Power

5 volt power is supplied on Pin 28 and this power is filtered by C16.

#### Address

The EEPROM has 8 KiB ( $2^{13}$  or 8,192 individual bytes) that the CPU can access. The address inputs A0 through A12 are provided by the CPU. When your computer is reset, the CPU fetches the instruction at location 0000H (remember, H means hexadecimal). The EEPROM is mapped by the address decoder to respond on addresses 0000H through 7FFFH, so it will supply the instruction data to the CPU.

Note that the EEPROM is only an 8 KiB device (0000H ~ 1FFFH are valid EEPROM addresses), but it is enabled over a 32 KiB range. This causes *address foldover* to occur; the EEPROM data will repeat four times over the 32 KiB lower address range.

## Data

Just like the RAM, the EEPROM communicates information with the CPU on the eight bi-directional, tri-state data signals D0 through D7.

## Control Signals

The **/ROM\_CE** signal on Pin 20 tells the EEPROM if the CPU wishes to talk with it. This signal arises from the **address decoder** circuit of your computer. The address decoder enables /ROM\_CE only when the CPU emits a memory address in the range 0000H to 7FFFH. If the CPU asks for any address outside this range, **/ROM\_CE** will remain inactive (high). The EEPROM chip is de-selected for addresses that are not assigned to it.

The **/RD** (read) and **/WR** (write) signals come from the CPU and tell the EEPROM what the CPU needs to do with the RAM's data. If the CPU wishes to read data from the EEPROM, it will output a proper address causing **/ROM\_CE** to become active, *and* it will activate the **/RD** signal. The EEPROM's **/OE** (output enable) signal on Pin 22 is connected to **/RD**, so the EEPROM activates its tri-state data outputs, which pushes the selected EEPROM data onto the data bus for the CPU to read.

Writing data works almost identically. The CPU will send out an address like before, but for writing, the CPU will place the data to be stored on the data bus, then activate the **/WR** signal, which activates the **/WE** input of the EEPROM on Pin 27. This tells the EEPROM to store the CPU's data.

To summarize, **/ROM\_CE** tells the EEPROM it's part of a conversation with the CPU. **/RD** tells the EEPROM to send data to the CPU, and **/WR** tells it to accept data. The address lines tell the EEPROM where to store or retrieve the data.

## EEPROM Write Protect Circuit

The EEPROM will eventually be programmed with a large machine-language program called the *operating system*. We will need this program to remain intact so that we can use it to enter and run programs on the computer. The *monitor*, which is part of the operating system, provides a user interface for us to interact with. Normally, we don't want the CPU to be able to erase this information, so we have provided jumper JP1, *EEPROM WRITE ENABLE*, on the circuit board.

Jumper JP1, when installed, allows the active-low **/WR** signal to reach the EEPROM. This will allow us to write data into the EEPROM for testing purposes. Once we've tested the EEPROM circuit, we will cut this jumper. That will effectively write-protect the EEPROM, as R16 will then pull-up the **/WR** signal to 5 volts at all times, preventing EEPROM data from being overwritten.

## Pushbutton Switches

Because we haven't yet installed the address decoder, we will add the **/ROM\_CE** pushbutton switch to the board. The existing **/WR** and **/RD** switches will work together with the **/ROM\_CE** button to allow us to test the EEPROM.

## EEPROM Data are Non-Volatile

When power is removed, the EEPROM will retain its information. This is very important! Personal computers also have an EEPROM (flash) memory that holds the instructions for the BIOS (Basic Input Output System). The BIOS of a PC is what activates when a computer boots; it is the BIOS that loads the PC's operating system from disk into RAM, then transfers control to that operating system. Our little computer could certainly be expanded to include disks and other storage devices if we so wished.

## Assembly Procedure

*We recommend that you place a check mark by each step as you complete it.*

Step No.	Description	Completed
1	Install a 28-pin socket for the EEPROM in U10's position.	
2	Locate and install R16, a 10k axial resistor (color code BROWN-BLACK-ORANGE).	
3	Install a short (0.5", 12.5 mm) wire in the JP1 EEPROM WRITE ENABLE location. This will enable writing data into the EEPROM chip.	
4	Install pushbutton switch S5 ( <b>/ROM_CE</b> ).	
5	Carefully install the 28C64 EEPROM IC into the socket in position U10.  <u>Straighten the pins on this IC before inserting</u> (press the IC sideways against your ESD-safe worksurface.)  <b>Make sure all pins are mating correctly before pressing the chip into position.</b>  <b>Remember, parts with many pins are easily damaged, so don't force them into sockets.</b>	

You've completed the EEPROM circuitry. Ready to try it out? I knew you were.

## Test Procedure

Name \_\_\_\_\_  
 Sign-Off \_\_\_\_\_

Step No.	Description	Completed
1	<b>Apply power to the board.</b> Exercise all DIP switches and verify that the associated LEDs match the DIP switch inputs. <i>(If this is not working, check the <b>Troubleshooting Hints</b> on the next page.)</i>	
2	Apply the address [0000 0000]. All the ADDRESS BUS LEDs should be off.	
3	Apply the data pattern [0000 0000]. All the DATA BUS LEDs should be off.	
4	Select the EEPROM by holding down the <b>/ROM_CE</b> switch, S5.  While holding down S5, activate the read signal ( <b>/RD</b> ) by holding down switch S3.  Because you haven't stored anything in EEPROM, and the EEPROM has arrived unprogrammed, you're like to see a random byte displayed. Record what appears on the DATA BUS during the read operation. <i>(Don't be surprised if you see all zeroes.)</i>  Data Bus Pattern: [                    ]  <i>Release S3 when you're done with this step.</i>	
5	Write your data into the RAM by holding down the <b>/ROM_CE</b> switch, S5, then momentarily press <b>/WR</b> (S2). You've just written data into EEPROM location [0000 0000].  <i>Don't forget to release the <b>/WR</b> switch!</i>	
6	Recall the data in location [0000 0000] as you did in Step 4. Record the result below.  Data Bus Pattern: [                    ]	
7	Release the pushbutton switches, then enter the next address, [0000 0001].  Enter the data [0000 0001] into this EEPROM address.	
8	Enter unique data into addresses [0000 0010] through [0000 1000]	
9	Verify the data held in each address by entering the address onto the bus, then activating both <b>/ROM_CE</b> and <b>/RD</b> .	
10	Remove power from the computer (wait about a minute with power off), then repeat Step 9. Are the data still present?	

Did everything work correctly? We hope so, because now all the memory in your computer is working. The only remaining items will be the input-output sections, the address decoder, and CPU group.

## **Troubleshooting Hints**

*Please check the following if your data and address input section doesn't seem to be working correctly.*

### LEDs not Lighting?

1. Is the power supply good? (Repeat the +5V measurement of the prior section - if it's not good, check that the power adapter is in a live outlet, and that there are no solder shorts on the back of your board.)

### EEPROM Data aren't Recalling Correctly

1. Double-check that you've written the correct data into the memory.
2. Did you install the correct IC in U10's position?
3. Does U10 have good Vcc (Pin 28) and GROUND (Pin 14)? Measure with a DMM to be sure.
4. Are all solder joints on U10 okay?
5. Make sure you don't have any solder ball shorts on any ICs you've installed so far.





## 5: Input/Output: LCD, Keyboard, and Serial Port Circuits

The input and output sections of a computer allow it to gather input from the outside world, and in turn generate output that we can read, such as on the LCD screen. Our computer has an on-board keypad for user input, a 2 line by 16 character liquid crystal display (LCD), and a TTL-level serial port for connection to a personal computer. All of this is accomplished with just two IC chips! The schematic of this area is on the adjacent page.

### Output Port 82H

An eight-bit data latch, U14, is used to capture CPU data written to I/O port 82H. This address is signaled from the **address decoder** section by examining CPU data lines A0~A7 and CPU control line **/IORQ** (I/O request). When the CPU writes to I/O port 82H, the data will be captured and output on pins Q0~Q7 of U14. This as an *output port* mapped in the *I/O plane* of the Z80.

### LCD Interface

The LCD backlight LED current is set by R11. Software can control the backlight as its cathode is connected to Q6 of the U14 data latch. Writing 1 turns the light off, and 0 turns it on.

The LCD has eight data input lines (D0 ~ D7), four of which are grounded (D0 ~ D3), and four which connect to U14, the keyboard / LCD / serial port data latch. The **E** (enable, actually a *data clock*) and **RS** (register select) signals also pass to U14. You might wonder how eight data bits get to the LCD when four of the data inputs are grounded. The system software simply passes data to the LCD's HD44780 controller chip *four bits at a time*. Therefore, to send eight bits, *two* four-bit transactions must take place. The **RS** signal tells the LCD if the data is a command (1) or displayed data (0). These signals are all managed in software.

### Keyboard Interface

Pins Q0~Q4 of U14, the output port, are shared with keyboard row scan signals **KR0** through **KR4**. These pins serve a dual purpose - - they convey data and control information to the LCD and also are used to poll the rows of the keyboard. The system firmware manages these resources so that they do not clash. This is quite simple; we simply don't write data to the LCD and scan the keyboard at the same time. You may know this by another name, *multiplexing*.

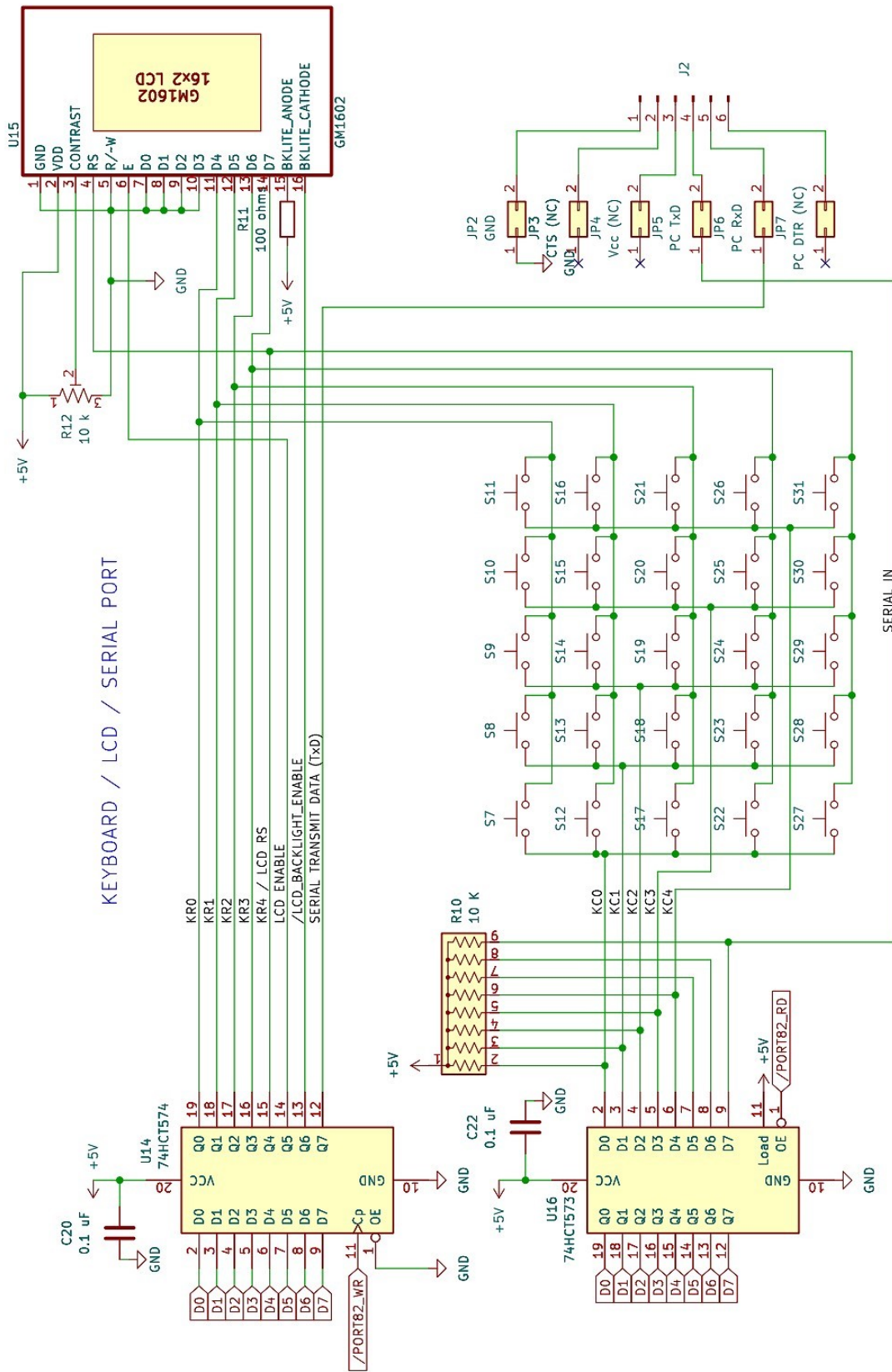
The system software scans the keyboard by writing a logic "0" the row it wishes to check. For example, suppose that S7 is pressed, and the signal **KR0** is low (logic 0). You can see that S7 physically bridges **KR0** to **KC0**, which is connected to the D0 input to U16, the port 82H input register. When the software reads port 82H, it will see that D0 has gone low, signaling that S7 has been pressed.

If S7 is not pressed, then the resistor on Pin 2 of R10 will pull line **KC0** to 5 V (high); the software will then see a logic 1 for D0 upon reading input port 82H.

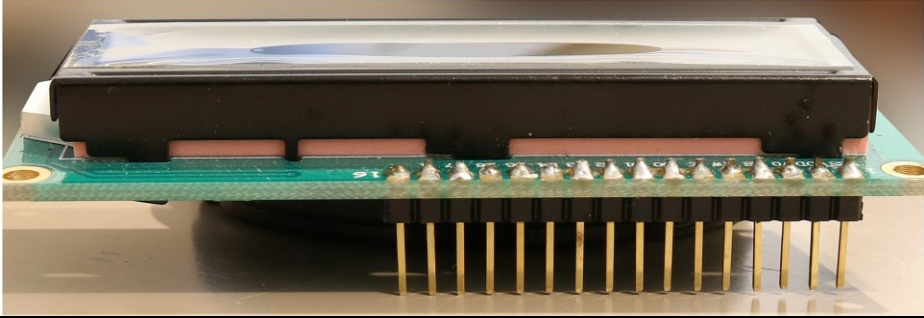

The software simply checks each row one at a time, and looks for any resulting logic zero results on **KC0** ~ **KC4**; these signal a pressed key, which is then acted on by the software. Since mechanical switches require debouncing, this is also handled by the software.

### Serial Port

Serial data is sent one bit at a time. It's a very standard way of sending information between computers. Your machine has a two-wire serial port that will enable it to communicate freely with a personal computer. The transmit data signal **TxD** arises from U14 output Q7 and passes to a TTL-to-USB serial port interface connected to connector J2. The system software converts transmitted data to serial, then outputs the serial data via U14 output Q7. Serial received data signal **SERIAL\_IN** passes from J2 back to input D7 of U16, where the system software detects the incoming data and converts it back to parallel form. If you believe that the system software is very busy when talking to the keyboard, LCD, and serial port, you are right!



## Assembly Procedure

Step No.	Description	Completed
1	Install a 20-pin socket for U14.	
2	Locate and install R11, a 100 $\Omega$ axial resistor (color code BROWN-BLACK-BROWN).	
3	Locate and install R12, the 10 k $\Omega$ LCD contrast potentiometer. This part will be marked "103." Look carefully as the lettering may be very small.  Adjust it so that it's halfway between minimum and maximum positions.	
4	Install the 16x1 header onto the LCD display. <u>Carefully note that the short pins on this header should be towards the LCD display board as shown in the picture below.</u> 	
5	Carefully solder the LCD assembly to the main circuit board as shown below. <u>Use a minimum of heat and trim the excess length from the protruding header pins under the board.</u> The LCD should be almost flush with the board and will be spaced from the board by the header insulator and LCD metal frame tabs. 	
6	Locate and install R10, a 10k SIP resistor. This will be marked "103."  <b><u>BE ABSOLUTELY SURE YOU ARE INSTALLING THIS RESISTOR IN THE CORRECT LOCATION WITH CORRECT ORIENTATION (DOT ON PART MATCHES DOT ON PC BOARD).</u></b>	
7	Solder in the remaining pushbutton switches in positions S1 (/PORT82_WR), S4 (RESET) and S7 through S31 (keypad).	
8	Install U14, the 74HCT574 data latch, in its socket.	

Your computer is beginning to look pretty complete now, isn't it? Let's see how things are working.

## Test Procedure

Name \_\_\_\_\_  
Sign-Off \_\_\_\_\_

Step No.	Description	Completed
1	<b>Apply power to the board.</b> The LCD may or may not illuminate as the data in the output latch U14 will be random on power up.	
2	Apply the data pattern [0000 0000].	
3	Momentarily press S1, the /PORT82_WR switch. The LCD backlight should immediately illuminate.	
4	Change the data pattern to [0100 0000] and again momentarily press S1. The LCD backlight will turn off. Pretty cool, right?	
5	<p>We can devise a test to see that the other 7 data bits from U14 are working properly. We can turn each bit on or off by changing the input data, then writing it to the port. From the schematic we know what each bit is connected to. <u>Work this out.</u></p> <p>Identify the signal and function of each output pin on U14 and record this information below. Then test each of these signals as you did in Steps 1 through 4. You will need measure each of these signals with a DMM set to DC volts. <u>Connect the black (ground) lead of the DMM to the GROUND test point on the board for all the measurements. Be careful not to short any adjacent pins while you're doing these tests!</u></p> <p><u>U14 Output Signal - Function - Point(s) of Test on Board</u>            Q0 - <b>Keyboard KR0</b> and <b>LCD D4</b> - U14 Pin 19            Q1 -            Q2 -            Q3 -            Q4 -            Q5 -            Q6 - <b>LCD Backlight</b> - U14 Pin 13 and U15 (LCD) Pin 16            Q7 - <b>Serial Output</b> - U14 Pin 12 and <b>PC RxD</b> at JP6.</p> <p>Note: We haven't installed the jumpers JP2 ~ JP7 for the serial port, but you should be able to measure the Q7 output at the <b>PC RxD</b> test point on JP6.</p>	

All should be working perfectly. If not, these suggestions may help.

### Troubleshooting Hints

#### LCD Backlight not Illuminating (or is Stuck On)

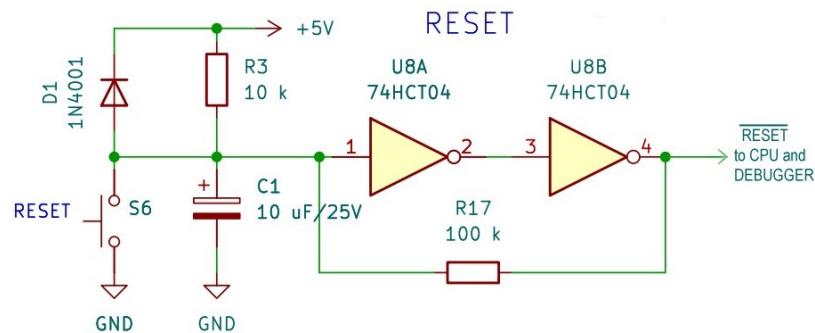
1. Is the power supply okay?
2. Is the correct IC chip (74HCT574) installed for U14?
3. Are the solder joints on U14 and S1 good?

#### LCD Backlight is Okay (Turns On and Off), but can't Control other Bits in Port 82H

1. Double-check solder joints on U14. This is the most likely problem.
2. Make sure you're measuring in the right places!

## 6: Reset and System Clock Circuitry

The reset and system clock are two vital sections within your computer. Both of these sections are implemented using a single IC chip, U8, a 74HCT04 hex inverter.



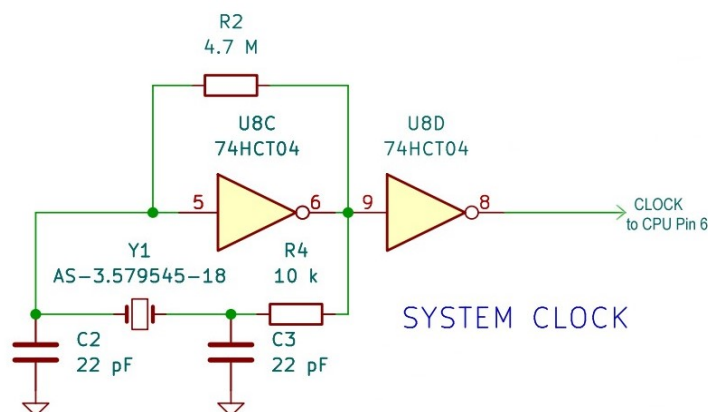
### Reset Circuit

A digital circuit will be in an unknown state (condition) when the power is first applied. For a computer, this is unacceptable. We need the computer to operate correctly every time it's powered up. The job of the reset circuit is to generate a momentary pulse that brings all the computer's circuits into a known state on power up.

The circuit has an RC time-constant of about 100 ms (that's 0.1 second) formed by R3 and C1. On power-up, C1 is totally discharged. It has zero voltage, so U8A pin 1 sees a logic LOW. In turn, U8A inverts this LOW to a HIGH, which then passes to U8B, which converts it back to a LOW again. The LOW coming from U8B passes to the CPU and DEBUGGER sections, resetting them to their proper, initial states.

C1 then begins charging through R3. After about 1/4 of the RC time-constant (25 ms) passes, the voltage on C1 rises through 1.5 volts, which is large enough to be recognized as a logic HIGH by U8A. The output of U8B also goes high, which removes the RESET signal from the computer's circuits; the computer then begins running.

You might wonder why we inverted the signal twice in this circuit. That's a little strange! We did this so that we could form a Schmitt trigger from U8A and U8B. Positive feedback from U8B, Pin 4 is fed back to U8A, Pin 1. This provides positive "snap action" on the RESET signal output.



### System Clock Oscillator

The other half of U8, together with Y1, the crystal, forms the *system clock oscillator*. An oscillator converts DC energy from the power supply into alternating current (AC). U8C is biased into its linear region by R2 - - so U8C functions as an analog amplifier. Signal from crystal Y1 passes into U8 Pin 5,

where it is amplified and given a 180° phase shift. The outgoing signal from U8C Pin 6 passes through R4 back through the crystal circuit (C2, Y1, and C3), where it again receives a 180° phase shift. The signal is now in phase so it provides *positive feedback*, which sustains the oscillation. The result is the same as placing a microphone in a PA system too close to a speaker, except that in our circuit, the oscillation frequency is very precise. This is because Y1 provides the correct phase shift of 180° *only at exactly its resonant frequency*, which is 3.579545 MHz.

U8D serves to square up the AC signal from U8C (you can think of U8D as an analog-to-digital converter), resulting in a very clean digital clock signal that can operate the CPU.

### Assembly Procedure

Step No.	Description	Completed
1	Install a 14-pin socket for U8.	
2	Install C1, a 10 µF capacitor. <b><u>Make sure you install this component with correct polarity as marked on the PC board. The longer lead is positive.</u></b>	
3	Install diode D1. <b><u>The cathode side of this part has a line and must face towards the line (and capacitor C1) on the circuit board.</u></b>	
4	Install the following components. <b><u>CAUTION: Be very careful not to confuse R3/R4 and R17. They look very similar! Use an ohmmeter to confirm their values if you have trouble seeing the color bands.</u></b> R3 and R4, 10k axial resistors (Brown-Black-Orange) R17, 100k axial resistor (Brown-Black-Yellow) R2, 4.7M axial resistor (Yellow-Violet-Green)	
5	Install Y1, the timing crystal. Y1 has no polarity and may be inserted in either direction. Mount Y1 flush with the board. <b><u>Treat the leads of Y1 gently! Do not bend the leads unnecessarily. They are easily broken off.</u></b>	
6	Install U8, the 74HCT04 integrated circuit, into its socket.	

## Test Procedure

Name \_\_\_\_\_  
 Sign-Off \_\_\_\_\_

*We recommend using an oscilloscope with a 10:1 probe for measuring the CPU clock signal. If you do not have this tool available, you can skip that portion of the test procedure.*

Step No.	Description	Completed
1	Connect the GROUND ( <b>black</b> ) lead of a digital multimeter set for DC volts to the GROUND test point on the board. (Use an alligator test clip).	
2	<b>Apply power to the board.</b>	
3	Measure the DC voltage at U8B, Pin 4. It should be 5 volts $\pm$ 0.25 V.	
4	Press and hold the RESET button and again measure the DC voltage at U8B, Pin 4. You should now observe 0 volts. (You may see 25 to 50 mV which is okay.)	
5	While keeping the meter on U8B Pin 4, release the RESET button. You should see the voltage immediately jump back to 5 volts as in Step 3.	
6	Move the DMM lead to U8C Pin 6. You should see between 1 and 2 volts. This verifies that the "amplifier" formed by U8C is indeed correctly self-biased.	
<p>If you have an oscilloscope available, you can measure the clock oscillator output to confirm that it's working correctly.</p> <p>Set up your oscilloscope as follows to begin. You can fine-tune these settings as required to get a good picture of the clock signal:</p> <ul style="list-style-type: none"> <li>• Auto Trigger</li> <li>• DC coupling</li> <li>• 2 volts/vertical division</li> <li>• 10:1 probe in use (see your scope's instruction manual)</li> <li>• 0.1 <math>\mu</math>s (100 ns)/horizontal division</li> </ul>		
7	Connect the GROUND lead of the scope 10:1 probe to the GROUND test point on the circuit board.	
8	<p>Touch U8D Pin 8 with the tip of the 10:1 probe. You should observe the system clock signal.</p> <p><u>This signal should be 5 volts peak-to-peak (0 V minimum, 5 V maximum) with a period of 0.279 <math>\mu</math>s (279 ns).</u></p> <p>Your scope may be able to read the frequency directly. As long as it reads close to 3.58 MHz (scope frequency readings vary in accuracy), the circuit is in great shape.</p> <p>NOTE: The exact frequency should be 3.579545 MHz <math>\pm</math> 100 Hz. Your oscilloscope may not be able to resolve the frequency this accurately.</p>	

All should be working perfectly. If not, see the next page for troubleshooting hints.

## ***Troubleshooting Hints***

### RESET Signal Stuck High or Low - Does not Respond to RESET Switch Presses

1. Is the power supply okay?
2. Is the correct IC chip (74HCT04) installed for U8?
3. Is D1 installed in the correct direction? (The line marking the cathode should point towards C1 and the RESET switch.)
4. Is C1 installed correctly? (The negative lead of C1 should face U8.)
5. Are the values of R3 and R17 correct?

### Oscillator is not Running - but RESET Circuit Works

1. Check your oscilloscope probe by touching the scope probe to the CALIBRATOR output of the scope. You may have a bad probe or other problem. (You can also touch the probe to the 5V TEST POINT on the board; the trace should jump up about 2.5 divisions from its ground baseline.)
2. Double-check the values of C2 and C3 (installed previously). Both C2 and C3 should be marked "20" or "22." Incorrect values for these capacitors will stop the oscillator dead in its tracks!
3. Check the value of R2, the bias resistor. Its color code should be Yellow-Violet-Green.
4. Inspect all solder joints surrounding U8 and the oscillator components.
5. Check that U8C is properly self-biased by reading the voltage on U8 Pin 6 with a DMM. You should observe between 1 and 2 volts at this point (74HCT04 installed), or 2 and 3 volts (74HC04 installed).

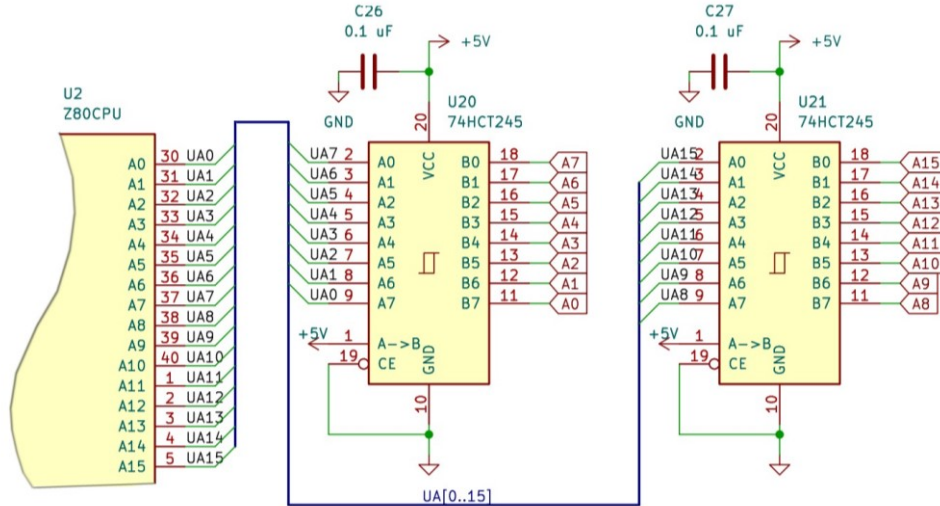


## 7: CPU Bus Drivers, Address Decoder, and Debugger

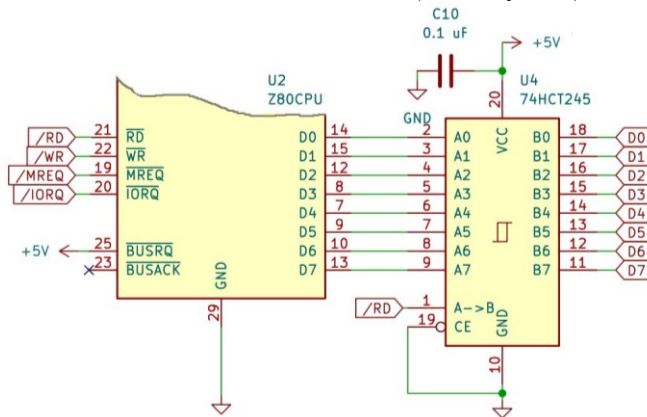
These three sections are the last we need to assemble to put your computer in running order. After this, we will only need to add the last I/O chips and serial port to allow it to talk to the outside world.

### Bus Drivers

There are two bus driver *buffer* groups. Buffers are circuits used to amplify and stabilize digital signals. The Z80 CPU bus pins don't have sufficient current capacity to reliably operate all the parts of the computer (memory and I/O). The buffers boost the current from the CPU to overcome this problem.



The address bus is buffered by U20 and U21. The unbuffered address signals from the CPU, UA0~UA15, enter the "A" inputs of U20 and U21 (Pins 2~9 on each device). U20 and U21 are bi-directional buffers; the direction is hard-wired in the A-to-B direction by connecting Pin 1, the direction input, to 5 volts. Therefore, the address bus signals within your computer may only originate at the CPU. The buffered address bus connects to all other sections (memory, I/O) as required by each section.

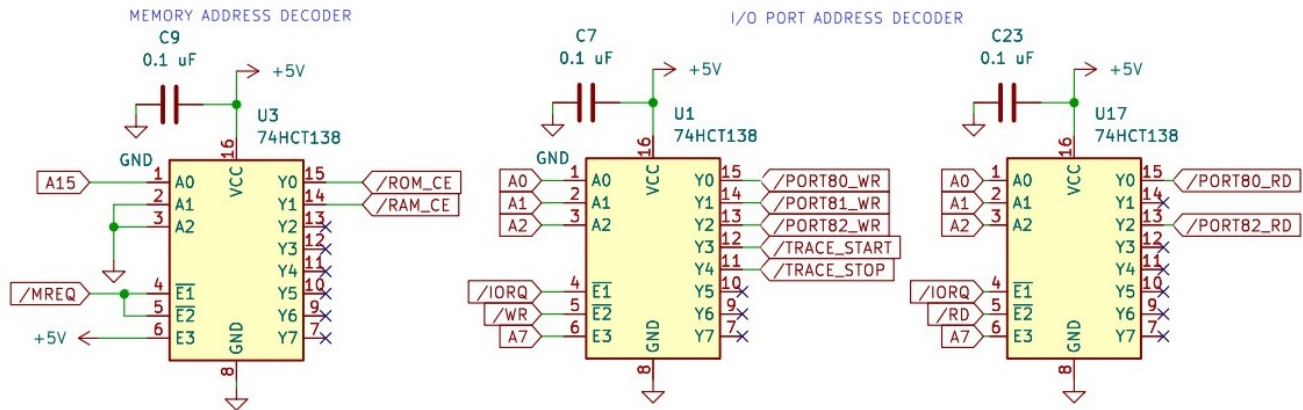


### Data Bus Transceiver and Driver

U4 is the data bus transceiver and driver. The circuit is very similar to the address bus driver, except that since only eight bits are on this bus, only one chip is required. Additionally, the data bus is bi-directional; you can see that Pin 1 of U4 is tied to the CPU /RD signal. When the CPU is reading

data,  $\overline{RD}$  will be low. This will cause U4 to move signals in the B to A direction. D0 ~ D7 of the computer will contain data arriving at the CPU from memory or I/O.

When  $\overline{RD}$  is high (inactive), the data bus flow is now CPU to memory and I/O. With U4 Pin 1 high, U4 now routes data from the CPU to the data bus (A to B direction).



### Address Decoder

The address decoder selects *one* memory or I/O device based on the address and control bus inputs presented to it. This section consists of U3, U1, and U17. All three ICs are identical three-to-eight line encoders of type 74HCT138. The binary selection code input is on Pins 1, 2, and 3 of each IC. These are labeled internally as A0, A1, and A2 -- be careful not to confuse these with the computer's address line signal names! In turn, depending on the binary code provided to Pins 1 through 3, one of the eight outputs, Y0 through Y7, will go low. All other outputs will remain high.

In order for *any* output to go low, the device must be *enabled*. That's the function of Pins 4, 5, and 6. Pins 4 and 5 must be low, and Pin 6 must be high; otherwise all the Y-outputs are inactive (high).

For the *memory address decoder*, U3,  $\overline{MREQ}$  from the CPU is tied to both  $\overline{E1}$  and  $\overline{E2}$  (active-low) enable inputs, and E3 is tied high. Therefore, U3 can select a memory device only when the CPU wants to read or write from memory. The only possible decode conditions are A15=1, which selects the RAM (8000H ~ FFFFH), and A15=0, which selects the ROM (0000H ~ 7FFFH). The signals  $\overline{ROM\_CE}$  and  $\overline{RAM\_CE}$  connect to the ROM and RAM respectively to enable them.

The *I/O port address decoder* consists of U1 and U17. Both of these chips are wired almost identically; both are enabled only when  $\overline{IORQ}$  from the CPU is low (Pin 4 of both ICs) and A7 is high. U17 is active only during I/O reads as the CPU  $\overline{RD}$  signal activates it on Pin 5. U1 is activated by  $\overline{WR}$ , so its outputs trigger only during I/O writes. The respective outputs of U1 and U17 are selected by A0 ~ A3; since A7 is required to be high, the active decoding range is 80H ~ 87H. Because A4 through A6 are not decoded, the outputs of U1 and U17 will also activate in the same sequence for addresses 88H through FFH. This is called *I/O address foldover*; it does not impact the operation of the computer since we don't need more I/O address decodes beyond what is provided by U1 and U17.

### Debugger Circuit

U5, U6, and U7 form the *debugger* circuit (refer to the computer main schematic). U6A and U6B form an S-R latch; during reset ( $\overline{RESET}=0$ ), or when a  $\overline{TRACE\_STOP}$  is issued by software through the address decoder on port 84H, this latch is cleared, forcing counter U7 Pin 15 (RESET) high. U7 output Q3 is forced low by the reset. When the system software begins a trace operation, it pulses  $\overline{TRACE\_START}$  momentarily by writing to port 83H. In turn, the latch formed by U6A and U6B is set, causing U7 Pin 15 (RESET) to go low. Now U7 can count  $\overline{M1}$  instruction fetches which are inverted by U6D and passed through U5B to U7's clock input. The client target instruction executes after three  $\overline{M1}$  fetches, where U7 Q3 output goes high on Pin 7; this is inverted by U6C to generate a non-maskable interrupt (NMI), which is serviced by the software to report the results.

## Assembly Procedure

Step No.	Description	Completed
1	Install 16-pin sockets for U7, U17, U1, and U3.	
2	Install 14-pin sockets for U5 and U6	
3	Install 20-pin sockets for U4, U20, and U21.	
4	Install a 40-pin socket for U2, the CPU.	
5	Install U7, the 74HC4017 counter.	
6	Install U4, U20, and U21, the 74HCT245 bus transceivers.	
7	Install U5, a 74HCT08 Quad-AND gate.	
8	Install U6, a 74HCT00 Quad-NAND gate.	
9	Install U1, U3, and U17, the 74HCT138 3-to-8 decoders.	
10	Install U2, the Z80400 CPU.  Be sure to carefully straighten the pins of U2 and <b>do not force it into the socket</b> . You can easily bend a pin under, or outright break it off. <b>Take your time and inspect your work carefully as you insert this large component.</b>	
11	Install a 20-pin socket for U16.	
12	Install U16, a 74HCT573 octal latch. This is the keyboard input port.	
13	<b>Carefully cut and remove the WRITE ENABLE jumper JP1</b> adjacent to U10, the EEPROM. <i>This will prevent EEPROM data from being corrupted.</i>	
14	<b>Remove U10, the EEPROM and place it into anti-static packaging.</b>	

## Programming the EEPROM

Your computer needs the operating system file written into the EEPROM. Without an operating system, you won't be able to interact with your computer to program it (though it might make a nice paperweight or wall hanging!)

The operating system is contained in a binary image file that you may download from the following web address:

<http://n0gsg.com/EZ80/>

Additionally, you'll find the *MiniTerminal* applet for Windows, along with the schematic diagram and other information (such as sample test programs) on this page. There is no charge for downloading information from this website.

Once you've downloaded the binary file, use a suitable programmer such as the [True-USB PRO GQ-4X](#) available from <https://mcumall.com/store>. **Once you've programmed the EEPROM, install it back into the U10 socket.** Your computer is now ready to test!

## Test Procedure

Name \_\_\_\_\_

Sign-Off \_\_\_\_\_

Our test procedure will consist of two parts, a power-on test and programming test.

### Power-On Test

Step No.	Description	Completed
1	<b>Apply power to the board.</b>	
2	<p>After a short delay (less than 1 second), you should see the following prompt on the LCD display:</p> <p><b>Monitor Ready</b> &gt;</p> <p>If you see this prompt, <i>Congratulations!</i> All of the assembled portions of your computer work properly. <i>Way to go!</i></p> <p><b>We know that all is OK at this point because the operating system performs a fairly complete power on self test, or POST, each time the computer is powered up or reset.</b></p> <p>The <b>Monitor Ready</b> prompt says that your computer is in good health and ready for use.</p>	

If you see the **Monitor Ready** prompt, go to the next page to perform the programming test.

### TROUBLESHOOTING TIPS

If you don't see the **Monitor Ready** prompt, try adjusting **R12**, the **LCD CONTRAST** control. Adjust the control to obtain the clearest text on the display.

Is the backlight of the LCD flashing on and off? If so, the computer is trying to tell you it has detected trouble by flashing a POST (power on self-test) code. The troubleshooting hints provided at the end of this section will help you untangle POST errors.

### Programming Test

Step No.	Description	Completed																				
1	<b>Apply power to the board.</b>																					
2	<p>At the <b>Monitor Ready</b> prompt, press the following key sequence on the keypad:</p> <p><b>MEM</b> 8200 <b>ENTER</b></p> <p>The computer will respond:</p> <p><b>Edit Memory</b> 8200: XX &gt;_</p> <p><u>This is the prompt for the Memory Editor.</u> "XX" will be a random value, depending on what is in memory address 8200H on power-up.</p> <p>Type the following:</p> <p>CF <b>ENTER</b></p> <p>The monitor will respond: 8200: CF 8201: XX &gt;_</p> <p><i>The monitor is confirming that the data byte CF was written into memory location 8200 on the first line, and is waiting for your input on the second. Go to Step 3.</i></p>																					
3	<p>Type 21 <b>ENTER</b> to enter data into memory location 8201.</p> <p>TIP: If you make a mistake, simply press <b>&lt;-</b> to go back.</p>																					
4	<p>Complete entry of the test program by entering data bytes into each of the remaining memory locations. <u>Type these carefully!</u></p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr> <td>8202: 00</td> <td>8203: 00</td> <td>8204: 3E</td> <td>8205: 0C</td> </tr> <tr> <td>8206: D7</td> <td>8207: 3E</td> <td>8208: 20</td> <td>8209: D7</td> </tr> <tr> <td>820A: CD</td> <td>820B: 41</td> <td>820C: 00</td> <td>820D: 01</td> </tr> <tr> <td>820E: 05</td> <td>820F: 00</td> <td>8210: CD</td> <td>8211: 50</td> </tr> <tr> <td>8212: 00</td> <td>8213: 23</td> <td>8214: 18</td> <td>8215: F1</td> </tr> </table>	8202: 00	8203: 00	8204: 3E	8205: 0C	8206: D7	8207: 3E	8208: 20	8209: D7	820A: CD	820B: 41	820C: 00	820D: 01	820E: 05	820F: 00	8210: CD	8211: 50	8212: 00	8213: 23	8214: 18	8215: F1	
8202: 00	8203: 00	8204: 3E	8205: 0C																			
8206: D7	8207: 3E	8208: 20	8209: D7																			
820A: CD	820B: 41	820C: 00	820D: 01																			
820E: 05	820F: 00	8210: CD	8211: 50																			
8212: 00	8213: 23	8214: 18	8215: F1																			
5	<p>AFTER you've pressed <b>ENTER</b> for location 8215, press <b>STOP</b> to return to the <b>Monitor Ready</b> prompt. Then run your program by typing:</p> <p><b>GO</b> 8200 <b>ENTER</b></p> <p>The computer will begin counting upward on the LCD display, two counts per second. It will continue doing this until you press the <b>RESET</b> button.</p> <p>When you press <b>RESET</b> the program will remain in memory -- you can run it as many times as you desire.</p> <p><i>If this works, your computer is in perfect working order. The only things remaining to be assembled are the I/O ports and serial port.</i></p>																					

## Troubleshooting Hints

### POST Error Code

*The operating system checks the RAM, EEPROM, keyboard and serial port circuits each time the system is powered up or reset. If you see the backlight of the LCD flashing on power up, this means that POST has detected a problem with your computer. The number of sequential flashes gives the POST error code.*

*Getting a POST code is encouraging - - the CPU is running at least some of the operating system code!*

1. Code 1: RAM ERROR. **Check:** Solder joints around U9 as well as the memory address decoder, U3. Make sure there are no bent pins on either of these ICs.
2. Code 2: ROM CHECKSUM INVALID. **Check:** Solder joints around U10 as well as the memory address decoder, U3. Make sure there are no bent pins on either of these ICs.
3. Code 3: Keyboard KEY STUCK. **Check:** Solder joints around the keypad (look for solder splashes or solder balls), the 10k SIP resistor R10, as well as around U16, the keyboard input port. Make sure that U16 is the proper IC. (You did install U16 and R10, right!?)
4. Code 4: Serial Input STUCK LOW. **Check:** Solder joints around U16 and on R10, the 10k SIP resistor.

### LCD Has No Message (and may not be lit), and NO POST ERROR CODE

*Something is stopping the computer dead in its tracks! Check the simple items first.*

*Note that we have not yet installed components in the U11, U12, and U13 positions. These are not required for the computer to boot and complete the self-test. We've deliberately omitted these three chips to make testing simpler.*

*Also, please note that the positions of the DATA and ADDRESS DIP switches will no longer have any effect on the computer as the respective bus buffers are electrically "stronger" than the DIP switch circuits. So don't worry about the DIP switches!*

1. Check the power supply at the **5V TEST POINT**. (Don't skip this step.)
2. Make sure you've installed the proper ICs in the U4, U20, U21, U7, U5, U6, U1, U3, and U17 positions. It doesn't hurt to look carefully for bent pins.
3. Remove U10, the EEPROM, and verify that it contains good data. Use the VERIFY feature of your EEPROM programmer.
4. Inspect all solder joints surrounding U8 and the oscillator components.
5. Check that the CPU RESET is being released. Measure the DC voltage at U2 Pin 26 (**/RESET**) with the RESET button pressed and released. It should measure close to zero volts (< 50 mV) during reset, and close to 5 volts ( $5\text{ V} \pm 0.25\text{ V}$ ) when the RESET button is released.
6. Use a scope with 10:1 probe to verify that the CPU clock is OK; measure at U2 Pin 6. You should measure a signal that's  $3.579545\text{ MHz} \pm 100\text{ Hz}$ , 5 volts peak-to-peak.
7. If you suspect a short on the data or address bus (not likely as we tested these sections earlier), remove power from the computer and use the continuity check feature of your DMM to "buzz" between adjacent bus pins (A0 to A1, A1 to A2, and so forth). Do the same for the data bus.

#### TIPS:

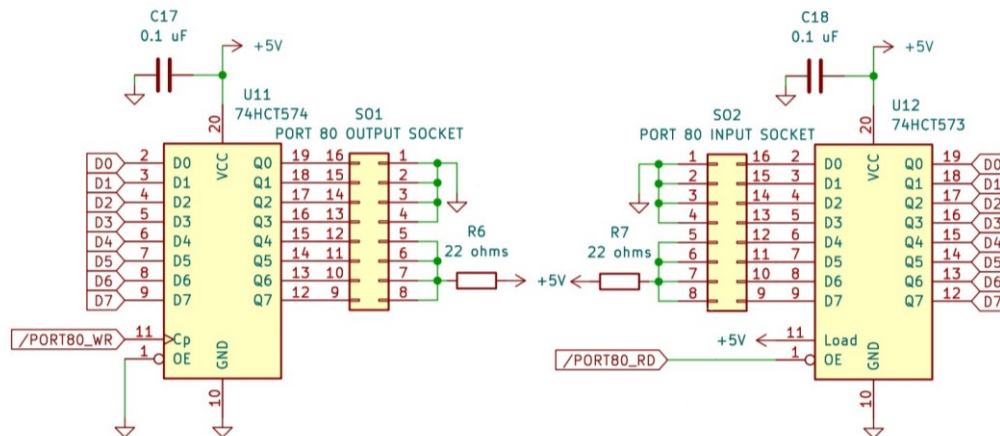
- More than 90% of problems with electronic kits can be traced to problems with solder joints. It can pay to have a second person inspect your board if you can't readily locate the trouble.
- If you have access to a friend's working machine, you can substitute suspect components into their computer to test them. Remember to use ESD-safe component handling practices.

## 8: Interface Ports, Final Testing, and Remote Operation

The circuits in this section consist of input and output ports and a TTL serial port. These will allow your computer to connect to external circuitry and a personal computer.

### *Input and Output Ports*

There are two eight-bit output ports and one eight-bit input port. The schematic of the input and output port mapped to address 80H is below.



One of the output ports is created using U11, a 74HCT574 eight-bit (octal) flip-flop. The clock input to this latch is supplied by the **address decoder**. When the CPU performs a write on I/O port 80H, the **/PORT80\_WR** signal will go low. In turn, data from the system address bus on **D0 ~ D7** will be captured within the eight individual D flip-flops within the chip and will instantly appear on chip outputs **Q0 ~ Q7**. These outputs are connected to Pins 9 through 16 of **SO1**, the Port 80H output socket, where they can be received by external circuitry. A second eight-bit output is provided by U13; its output appears at SO3.

Note how Pins 1 through 4 of SO1 are at ground potential - - these are a handy way to ground your external device to the computer's ground. Pins 5 through 8 of SO1 receive 5 volts from the computer's power supply through R6, a 22  $\Omega$  current limiting resistor. R6 provides some measure of protection against short circuits on the power bus. (If you need higher current at the output port and are confident in your connections, you can replace R6 with a wire jumper to get full current capability -- about 500 mA -- at the output port socket.)

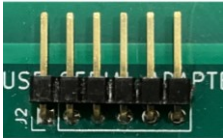
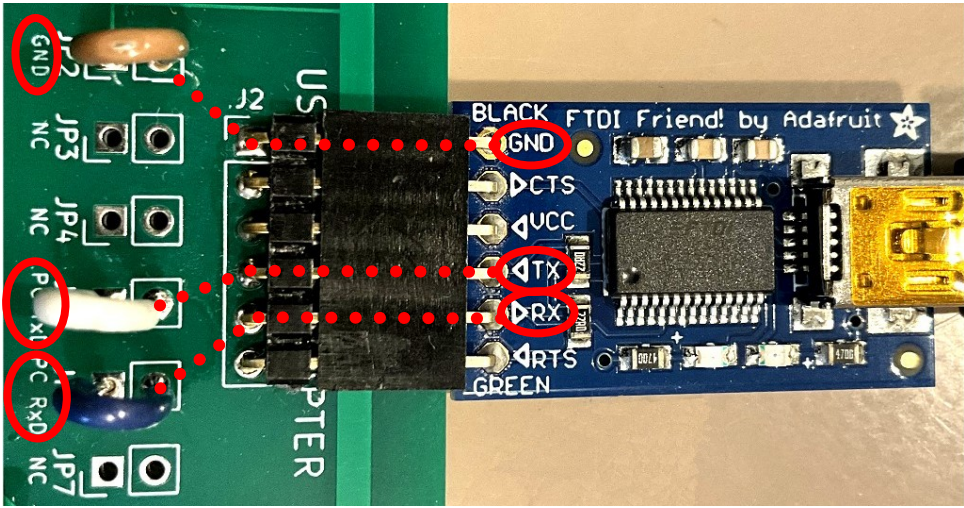
The input port is built around U12, a 75HCT573 octal latch. Input information on the Port 80H input socket, **SO2**, is connected to Pins 2 through 9 of U12. These are the data input pins. When the CPU reads from I/O port 80H, **/PORT80\_RD** goes low. This signal enables the output of U12 on Pin 1, **/OE**; U12 then pushes the input information onto **Q0 ~ Q7**, its outputs, which then flows onto the data bus where it can be read by the CPU.

### *Serial Port*

U14, output **Q7** and U16, input **D7**, form the *serial port*. This port allows your computer to talk to a personal computer or other device that communicates serially (one bit at a time) at TTL voltage levels. Instead of using a separate chip (such as universal asynchronous receiver transmitter, or UART), your computer's operating system performs the serial-to-parallel conversion using a software method called *bit banging*.

To connect your computer to a PC, you must purchase a USB to TTL Serial adapter such as the AdaFruit FTDI Friend or Inland FTDI Adapter. Many companies make these boards. We've provided jumpers JP2 ~ JP7 to allow you to configure your computer board for any USB adapter. You can download the *MiniTerminal* applet (Windows only) free from <http://n0gsg.com/EZ80>.

## Assembly Procedure


Step No.	Description	Completed
1	Install 16-pin sockets for SO1, SO2, and SO3.	
2	Install 20-pin sockets for U11, U12, and U13.	
3	Install 22 $\Omega$ axial resistors (marked RED-RED-BLACK) for R6, R7, and R8.	
4	Install 74HCT574 octal flip-flop ICs into U11 and U13 sockets.	
5	Install a 74HCT573 octal latch IC into the U12 socket.	
6	Install the six-pin right-angle header into the J2 position as shown in the picture. <u>The short, straight pins are soldered to the board and the right-angle pins must face towards the edge of the board.</u> 	
7	<p>Temporarily connect your USB to TTL Serial Adapter board to J2 as shown in the picture.</p>  <p>a) Note the naming of the pins on your adapter and the pin names shown by each configuration jumper of JP2 through JP7.</p> <p>b) Solder three jumpers (GND, PC TxD, and PC RxD) so that the signals match up as shown in the picture.</p> <p><b>IMPORTANT: <u>Connect only the three signals shown.</u></b> There is no need to connect anything to the pads marked "NC." These signals are not connected to anything on your computer.</p>	
8	<b>Be sure to set your USB to TTL adapter for 5 volt operation. This is very important – see the documentation that comes with your unit.</b>	



## Test Procedure - Personal Computer Connection

Name \_\_\_\_\_  
 Sign-Off \_\_\_\_\_

The *MiniTerminal* applet, available free of charge from <http://n0gsg.com/EZ80>, is highly recommended. Simply download the applet and place it on your desktop. There's no installation procedure; double-click the applet's icon to launch it.

Step No.	Description	Completed
1	<p>Connect your USB to TTL serial adapter to your personal computer and install device drivers as required.</p> <p>TIP: On <i>Windows</i>, use <i>Device Manager</i> to check the installation of the USB to TTL adapter, which may be accessed as follows:</p> <ul style="list-style-type: none"> <li>• Hold down the Windows key , and while doing so, type <b>R</b>. The <i>Run</i> dialog will appear. Release the Windows and R keys.</li> <li>• Type <b>devmgmt.msc</b> into the box and press <b>ENTER</b>.</li> <li>• Device Manager will open. Drill down into <i>Ports (COM &amp; LPT)</i>. You should see your adapter and its COM port number. <u>Take note of the COM port number.</u></li> </ul>	
2	Connect the USB to TTL board to J2 on your EZ-80 computer. Power up your computer.	
3	<p>Open the <i>MiniTerminal</i> applet. Click the purple <b>Settings...</b> button and choose the COM port number you noted in Step 1.</p> <p>TIP: Windows may block <i>MiniTerminal</i> the first time you try to run it. Click "Advanced" and follow the dialogs to enable it.</p>	
4	<p>Press the RESET button on your computer. After POST has completed, you should see the sign on prompt in <i>MiniTerminal</i> as shown:</p> <pre>EZ-80 Resident Monitor V1.01 (c) 2023 Tom A. Wheeler Monitor Ready &gt;</pre> <p>You will also see "Monitor Ready" on the computer's LCD display just as before.</p> <p><b>TIP: <u>Your EZ-80 trainer always works with both the remote PC MiniTerminal and the local LCD and keypad simultaneously.</u></b> You will always see output on both. <i>You may freely enter commands from either the PC or the local keypad at any time.</i></p>	
5	<p>Press the <b>ENTER</b> key on your PC from within <i>MiniTerminal</i>. You should see your computer immediately respond with another <b>Monitor Ready</b> prompt.</p> <p>If you see this, congratulations! You're now connected remotely to the computer and everything is working perfectly.</p>	

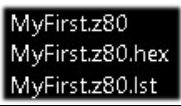
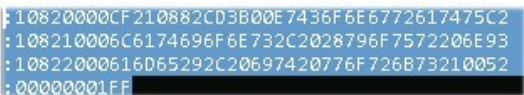
## Test Procedure - I/O Port Operation

In this procedure we'll check the operation of the I/O ports you've built in this final assembly stage. You will need a digital multimeter (DMM) to make test measurements as well as a short AWG 24 jumper wire. If you have DuPont jumpers (male to male) these are best.

Step No.	Description	Completed
1	Connect your EZ-80 computer to a PC running <i>MiniTerminal</i> as you did in the previous procedure. Press RESET on the EZ-80 to obtain a sign-on message.	
2	Use an alligator clip lead to connect the black (ground) of your DMM to the GND TEST POINT on the EZ-80 board. Set the DMM to measure DC VOLTS.	
3	<p>At the Monitor Ready prompt enter the following command:</p> <p><code>O 80 00</code> <b>ENTER</b></p> <p><b>TIP: This command starts with the letter 'O', and not the numeral 0, as it's the <u>Output to Port</u> operation.</b></p> <p><i>This command outputs (O) the byte 00H to Port 80H.</i></p> <p>With your DMM, measure the voltage at Pins 9 through 16 of SO1. All pins should read 0V.</p>	
4	<p>Repeat the command of Step 3 to write the following hex values to Port 80: 01, 02, 04, 08, 10, 20, 40, 80. For example, to write 01 to Port 80:</p> <p><code>O 80 01</code> <b>ENTER</b></p> <p>Since this sets D0 HIGH, and all other bits LOW, you should read <math>5V \pm 0.25V</math> at Pin 16 of SO1, and 0V at all other data pins (9 through 15). When you write 02 to the port, you should see Pin 15 HIGH, and the other pins LOW.</p> <p><i>Refer to the Schematic Diagram. It will help you understand this procedure, which is known as a "walking bit" test.</i></p>	
5	Repeat Steps 3 and 4 to test Port 81. You'll measure at SO3.	
6	Use your jumper wire to short Pins 9 and 1 of SO2. This produces a logic LOW at Input Port 80 bit D7 (Pin 1 is GROUND).	
7	<p>At the Monitor Ready prompt, enter the following command to query the input port:</p> <p><code>I 80</code> <b>ENTER</b></p> <p>The monitor will respond with:</p> <p><b>Port 80 data 7F</b></p> <p>The returned value MUST be less than 80H to pass this test. It may be anything from 00 through 7F.</p>	
8	<p>Repeat Steps 6 and 7, but move the jumper to Pins 9 and 8 of SO2. This gives a logic 1 at D7 (Pin 8 is 5V), so the response should be 80 through FF.</p> <p><i>If all these steps are successful, all the external I/O ports on your EZ-80 are fully functional. <b>Remove all test jumpers from the I/O sockets.</b></i></p>	

## Test Procedure - Remote Software Development and Downloading

Your EZ-80 computer supports any Z80 development system that can produce Intel Hex files. These files are used to interchange binary information between computers. A great resource for developing Z80 software in conjunction with *MiniTerminal* and your EZ-80 is [asm80.com](http://asm80.com), created by Martin Maly of Prague, Czech Republic.

Step No.	Description	Completed
1	Connect your computer and PC; launch <i>MiniTerminal</i> and navigate to the <b>Monitor Ready</b> prompt.	
2	Navigate your web browser to <a href="http://ASM80.COM">ASM80.COM</a> . Start a new Z80 project by clicking the <b>New File</b> button at right. We recommend "MyFirst" as your first project name. Select type as "Zilog Z80."  <b>IMPORTANT: Your project name must not contain any spaces.</b>	
3	Type (or copy) the following code into the ASM80 workspace window.  <pre> ;MyFirst.ASM - - My first Z80 assembler program  CR      EQU      0DH      ;CARRIAGE RETURN CHARACTER PRTSTG  EQU      3BH      ;PRINT ASCII-Z STRING TO CONSOLE                  ORG 8200H      ;EZ-80 PGMS SHOULD START @ 8200H  START:  LD      HL,MSG1      ;ADDR OF MSG TO PRINT         CALL   PRTSTG         RST    20H          ;RETURN TO EZ-80 MONITOR  MSG1:   DB      'Congratulations, (your name), it works!',CR,CR,0          END </pre>	
4	Press the <b>COMPILE [F9]</b> button at right to assemble your program.   On the left under <i>Workspace in your browser</i> you will now see three files as shown here. Click MyFirst.z80.hex to navigate to the IntelHex output.	
5	Click anywhere within the Intel Hex output and copy it to the Clipboard with the following key sequence:  <b>CTRL-A</b> (select all of the Intel Hex data) and then <b>CTRL-C</b> (copy the data to the Clipboard).	
6	Switch to the <i>MiniTerminal</i> applet and click the green <b>Upload Hex File</b> button. You will observe the following: <ul style="list-style-type: none"> <li>A dialog will appear as data are sent to the EZ-80 board.</li> <li>At the end of the upload, you'll see the following report on the screen:</li> </ul> <pre> Upload to remote host GOOD. Use G (address) to run and test your code.  &gt; </pre>	
7	Type <b>G 8200</b> <b>ENTER</b> to run your program. The "congratulations" message will appear, followed by a <b>BREAK at:8206</b> message as control returns to the Monitor.	

## ***Troubleshooting Hints***

### **Unable to See USB-to-Serial Adapter in Device Manager**

1. Check to make sure you've loaded the correct device driver.
2. With Device Manager open, remove then reinsert the USB adapter. This forces any Windows driver code to unload, then reload as the adapter is reinserted. This may fix the problem.
3. Make sure your USB port is working. Connect a known-working device to check it.

### **Not Seeing Sign-On Message or Monitor Ready in MiniTerminal**

1. Press the RESET button on your computer.

IMPORTANT: If you power your computer off while it's connected to the USB adapter on J2, it may not reset properly due to power feedthrough from this connector. *If this is a problem, install a diode (cathode line pointing towards the USB adapter) in place of the TxD jumper (JP5) adjacent to J2.*

You should always see information such as **Monitor Ready** on the LCD screen of your computer.

2. Check the 5V test point to be sure that your computer has power.
3. Perform a loopback test with the USB-to-Serial Adapter:
  - a) Disconnect the adapter from your EZ-80, but leave it connected to the PC.
  - b) Use a jumper (DuPont types work best) to short the TxD and RxD pins on the adapter connector.
  - c) Type into the MiniTerminal window:
    - \* If you see what you're typing, the USB-to-Serial Adapter is healthy. Check that you've installed the jumpers JP2 ~ JP7 correctly. Also check that you've selected the correct data rate (4800 bps, the default) in the MiniTerminal settings dialog box.
    - \* If you don't see your typing, check the device driver and COM port settings for your adapter.
4. Inspect the soldering around U14 and U16. Also make sure that there are no bent pins on these two ICs.

### **I/O Ports are not Working Correctly**

1. Be sure you've installed U11, U12, and U13 correctly - - each should be the correct part number and oriented correctly in the sockets (no bent pins, etc.).
2. Check the solder joints around U1 and U17, the port address decoders.

#### TIPS:

- More than 90% of problems with electronic kits can be traced to problems with solder joints. It can pay to have a second person inspect your board if you can't readily locate the trouble.
- If you have access to a friend's working machine, you can substitute suspect components into their computer to test them. Remember to use ESD-safe component handling practices.

## 9: Z80 Machine Language Programming

### Introduction

The Zilog Z80 was quite popular in its time, and is still used extensively today, primary as a foundational processor platform. By "foundational" we mean that internal design of the Z80 is used inside newer microprocessor chips as a functional building block. The actual (physical) Z80 chip (as originally used in the 1970s and 1980s) is still used, primarily in industrial automation products.

Spin-offs of the Z80's internal design are used in everything from TI calculators to Internet of Things (IoT) devices such as home appliances. We can still purchase new Z80 chips today. This is amazing since the chip is nearly 50 years old.

Historically, the Z80 was one of the first widely-adopted 8-bit microprocessors. It was used in the Radio Shack TRS-80 computers for many years. Bill Gates of Microsoft got started in the computer business by co-writing (with his business partner Paul Allen) a BASIC language interpreter for the MITS Altair 8800 computer, which used an 8080 processor. Yes, this was Microsoft's first commercial product - - one that would eventually be sold to other computer manufacturers including Radio Shack, Apple, Commodore, and others. The Z80 is compatible with the 8080 and 8085 chips at the machine language level, which allowed Gates and Allen to easily port their BASIC interpreter to the TRS-80.

We will be writing programs on our computer using machine language. Writing code this way is arduous and slow, but also results in programs that run as fast as possible on just about any hardware.

To write programs in machine language we must know a great deal about the computer's internal workings, and in particular its CPU chip.

No soldering is required for this lab!

### Instructions

1. Use the Z80 datasheet or your class notes to answer the following questions:

a) How many memory addresses can the Z80 access?

---

b) Explain what each Z80 register below is used for:

Register	Typical Use
A	
F	
BC	
DE	
HL	
PC	
SP	

c) What is held in register PC when the system is reset? Why is this important?

---

2. Hand-assemble the following assembly language program. Use the website ASM80.COM to verify the correctness of your work. Make sure you start your program at address 8200H by including an "ORG 8200H" at the top of your code.

Address	Machine Code	Source Code
8200		LD A, 03H
		LD B, 02H
		ADD A, B
		RST 20H

Explain what you believe this program does - and what contents should be held in registers A and B when it completes. (The **RST 20H** instruction returns control back to the Monitor so that we can view the results of the program's run.)

---

3. Enter your program into the EZ-80 system as follows:

- a) Power up your EZ-80. The **Monitor Ready** prompt should appear on the LCD.
- b) Enter the following command: **M 8200** from either the local keyboard or a connected PC.

TIP: Press the **MEM** key, the digits 8200, then the **ENTER** key to enter this command on the EZ-80 keyboard.

- c) Enter the hexadecimal Machine Code bytes into each memory address as you determined in Step 2. (For example, location 8200 should hold 3EH and 8201 should hold 03H.)

TIP: Press **ENTER** after each entry. You can press the **<-** key to go back to a prior memory location if you make a mistake in typing.

- d) After you've entered all the bytes, press **STOP** to return to the **Monitor Ready** prompt.

4. Your EZ-80 computer includes a Z80 disassembler. This utility is handy for verifying the correctness of machine language programs we've entered into it. Let's use the disassembler to check our program now:

- a) At the **Monitor Ready** prompt, press the **UNASM** key. Type in the first memory address of your program (8200), then press **ENTER**.

- b) The screen should display the memory address 8200, the bytes 3E 03 (which are the opcode and operand of the LD instruction), and the mnemonic "**LD A, 03**" just as shown in the Source Code above.

- c) View subsequent instructions in the program by pressing either **ENTER** or **->** to scroll through them. Press **STOP** when you're done to return to the **Monitor Ready** prompt.

5. Run this program to check the results:

- a) Press the **GO** key, the first address of your program (8200), then the **ENTER** key.
- b) The code runs. Since our program has only four instructions, and the CPU is running at 3.58 MHz, it takes less than 4  $\mu$ s to complete the run!

You should see the following message on the LCD:

```
BREAK at: 8205  
>_
```

The BREAK message is the result of the RST 20H instruction. This instruction completes the run of your program by returning control to the EZ-80 Monitor. The BREAK message essentially says "Boss, I've finished your program run and saved all the results."

6. The result of the program should be stored in the A register. To inspect the CPU registers, press the **REGS** key, followed by **ENTER**.

The EZ-80 Monitor displays CPU registers one pair at a time on the LCD. (If you are running the Monitor remotely from a personal computer using the *Mini Terminal* applet, it's smart enough to know this - - and will show you *all* the registers at once.)

If you're working with the local computer keyboard, you can use the **->** and **<-** keys to scroll through the registers. Write down what you find below.

Register	Contents (Hex)	Notes
AF		Consists of Accumulator and Flags
BC		Consists of B and C registers as a "register pair"
DE		Consists of D and E registers as a "register pair"
HL		Consists of H and L registers as a "register pair"
PC		Program Counter register - this contains the address of the next instruction to be executed by the CPU

Explain what you believe this program accomplished. What actions occurred? Did the outcomes agree with your expectations from Step 2?

---

---

7. Hand-assemble the following program as you did in Step 2, and verify what it does.

Address	Machine Code	Source Code
8200		LD HL, 0102H
		LD DE, 0304H
		ADD HL, DE
		RST 20H

Explain what you believe this program does - and what should be in registers HL and DE when it completes. Of course, load it onto the EZ-80 and run it to confirm the results.

---

Program Results of Step 7:

Register	Contents (Hex)	Notes
AF		
BC		
DE		
HL		
PC		

## Sign Off

Have your instructor review your work for correctness in order to obtain a sign-off.

Sign Off \_\_\_\_\_



## Questions

1. Explain the difference between "source code" and "machine code." Which of these can actually run on a computer?

---

---

---

2. Referring to the Z80 datasheet (or instruction set reference), what is the difference between the addition operations that were carried out in Step 2 and Step 7?

---

---

---

3. What was the effect of the RST 20H instruction we placed at the end of each program?

---

---

---

4. Summarize what you have learned in this experience. Do you have suggestions to improve it?

---

---

---

---

---

---

---



## 10: Loops

### Introduction

A loop is a section of program code that is used to repeat an action or set of actions. Everyday life is full of loops. For example, suppose that you are having four new tires put on your car. The service technician must pull each wheel off the car, remove the old tire, get a new one from the rack, mount and balance it, and reinstall the wheel. He or she must complete these steps four times to complete the work. Yes, that's a loop!

It works exactly the same way inside a computer. Whenever we want a computer to do the same thing over and over, we program it to loop.

Loops can be used as time-delay devices. In microcomputers and microcontrollers this is a common practice, though not necessarily a good one<sup>4</sup>. We are going to demonstrate a time delay by making the computer display a digital count. Without the time delay, the count would just whiz by and we'd be unable to see it.

In a later course, we will discuss how to use interrupts to create accurate timing while at the same time allowing a programmatic task to continue undisturbed.

### Instructions

1. The code below is a simple delay loop. Enter the code below and run it. Use ASM80.COM to do the assembly for you, but please write in the machine code you generated.

Address	Machine Code	Source Code	Comment
8200		LD HL,0FFFFH	Starting count=65535
8203		DEC HL	Subtract 1 from count
8204		LD A,H	
8205		OR L	See if (HL == 0)
8206		JP NZ,8203H	If not zero, keep counting downward
8209		RST 20H	Done!

How long did the delay last? (How much time passed time between starting the program with the GO command and seeing the **Monitor Ready** prompt)?

---

<sup>4</sup> It's a poor practice in a multi-tasking environment such as Android, Linux or Windows. Delay loops make programs dependent upon the CPU's clock for accurate timing - usually a very bad idea unless you know that the CPU clock frequency is not going to change. Our little microcomputer won't tell on us if we burn a few of its CPU cycles for a good cause.

2. We can make delays last longer by nesting the loops. A nested loop is one that has a loop inside of another. Going back to our service technician, suppose that there is a line of ten cars waiting to have new tires installed. That's an outer loop - - inside this outer loop, the tech will replace the four tires on each car (the inner loop). The technician will replace a total of 40 tires.

We can make time delays run longer by *nesting* loops. Try the one below.

Address	Machine Code	Source Code	Comment
8200		LD B,18H	Outer loop - count = 26
8202		LD HL,0FFFFH	Inner loop starting count=65535
8205		DEC HL	
8206		LD A,H	
8207		OR L	
8208		JP NZ,8205H	Keep going till inner loop runs out
820B		DEC B	Count down on outer loop
820C		JP NZ,8202H	Repeat inner loop until outer count reaches 0
820F		RST 20H	How long did it take?

How long did the delay last?

---

3. As you've seen the computer is very fast. Delay loops can be used to slow things down so that we can see the results. The program below counts starting with 1 and uses a delay loop at the end to allow us to see the numbers. This program uses a subroutine from the operating system kernel to display the numbers<sup>5</sup>. Type it in carefully. You may want to enter it into ASM80 and simply download it directly from your personal computer<sup>6</sup>. That's the easy way!

Please adjust this program so that it accurately times to the second. You'll need to adjust the value loaded into B at the instruction located at 8206H. You can do it!

---

<sup>5</sup> The operating system kernel contains the BIOS and Monitor. It's contained in the 8 KiB flash (28C64 EEPROM) on your computer. The kernel contains many useful subroutines we can use in our own programs.

<sup>6</sup> See the section *Test Procedure - Remote Software Development and Downloading* in Lab Experience 8 if you need a refresh on this topic.

Address	Machine Code	Source Code	Comment
8200		LD HL,0000H	Initial display = 0
8203		LD (8280H),HL	Save display value in RAM
8206		LD B,1	How much time delay? Adjust highlighted value to adjust speed of "clock."
8208		LD HL,8F00H	Inner loop count
820B		DEC HL	Inner countdown
820C		LD A,H	Test for inner loop
820D		OR L	count reaching zero
820E		JP NZ,820BH	Not zero? Continue inner loop.
8211		DEC B	Outer loop countdown
8212		JP NZ,8208H	Repeat outer loop if B > 0
8215		LD A,0CH	Clear screen command
8217		RST 10H	Output command to LCD
8218		LD HL,(8280H)	Retrieve displayed count (number)
821B		INC HL	Add 1 to this number
821C		LD (8280H),HL	Save it back to RAM
821F		CALL 0041H	Kernel: Print as a number to LCD
8222		JP 8206H	Back to top to wait another second

## Sign Off

When you've completed Step 3, show your results to the instructor.

Sign Off \_\_\_\_\_

## Questions

1. What is the definition of a loop in computer programs? What were loops used for in this laboratory experience?

---

---

---

2. Explain what is meant by a "nested" loop. Give an everyday example (not the one from the introduction) of a nested loop you have observed in everyday life.

---

---

---

3. In Step 3, explain how you adjusted the nested loop parameters to get a one-second delay.

---

---

---

4. Summarize what you have learned in this experience. Do you have suggestions to improve it?

---

---

---

---

---

---

---

## 11: Subroutines and the Machine Stack

### Introduction

Often we've written a bit of code that we'd like to reuse in different parts of a program. For example, we might have written a bit of logic to make a robot take one complete step forward. It would be silly to repeat this logic every place in a program where the robot needed to take a step. That would use up a lot of space and make our robot program hard to maintain.

A section of code that can be reused by different portions of a program is called a *subroutine*. In assembly language, we use the CALL instruction to turn control over to subroutines.

Suppose that we have a main program in our robot with three instructions, A, B, C and D. They might look like this:

```
0001  A      (Do something: SMILE)
0002  B      (Walk forward one step: CALL WALK)
0003  C      (Do something: WAVE)
0004  D      (Walk forward again: CALL WALK)
...

0050  WALK: (Logic for walking forward one step)
```

(Individual steps needed to walk a step are here in Steps 51 through 58)

```
0059  RETURN (Command to return to the caller)
```

The numbers at left represent the addresses of instructions. We know that the CPU tracks this with the *program counter* (PC) register. When the CPU is executing Step A, the PC register holds 0001. When Step B is active, PC holds 0002.

When Step B executes, the CPU must turn control over to the WALK subroutine. But because the WALK subroutine could be called from any place in the program, the CPU needs to know what comes next when the WALK subroutine completes. What comes next? Of course, you're right. Step C must come next. That means that the CPU needs to know how to get back to Step C after the CALL of Step B.

When the CALL of Step B is executed, the CPU stores the address of Step C on the stack first. We say that the address of Step C has been *pushed onto the stack*. It then passes control to the WALK routine. The WALK routine completes its instructions, then returns control with the RET instruction. RET tells the CPU to *pop the last address off the stack* and load that into the PC counter. (In other words, the CPU "jumps" to the last address pushed onto the stack.)

The same action happens in Step D. The WALK subroutine is again called; the machine stack will hold the address of the instruction after Step D while the WALK subroutine runs so that control will be returned to the correct place after WALK completes, just as before.

The stack makes all kinds of things possible besides subroutines. We can push registers onto the stack to temporarily save their contents while they're used for other purposes. Many things can be achieved, some quite strange. In this experience you'll see the stack in action. We'll use the EZ-80 integrated debugger to trace through a short program and observe what's happening with the CPU registers and stack.

## Actions in the Stack Demo Program

The demonstration program used during this laboratory experience is designed to let us see how the machine stack operates within our computer.

The program itself is relatively simple. It contains a main routine and an UNKNOWN subroutine that we will ask you to analyze.

We will use the TRACE command to single-step through our program one instruction at a time. This can be done directly from the EZ-80 computer's keyboard, or from a remotely-connected personal computer running the *MiniTerminal* package.

It's easier to see things from *MiniTerminal*, so we prefer you use this method.

## Just Take One Walk Around the Block

The end of the demo program contains a JP START instruction. You do not need to follow any action past this point as the program's instructions will repeat indefinitely.

## Taking Things a Step at a Time

The EZ-80 trainer includes a powerful hardware-based *trace* function. When we use Trace mode, we get to see exactly what happens as each instruction in a program is completed.

The trace function is hardware-based. This means that you can trace into any memory location on the machine, including the operating system routines in flash (EEPROM).

When running a trace from the MiniTerminal applet on your personal computer, you will see a complete report of the CPU registers after each instruction.

If you don't have a personal computer available, no problem. You can run a trace from the EZ-80's local (built-in) keypad. To see what's in the CPU registers during a local trace, simply press the **REGS** key to enter the Register Editor. Press **STOP** to exit the Register Editor and return to the trace.

Because trace also utilizes the keyboard and character-display (LCD / serial) subroutines in the operating system, tracing into certain areas of the EEPROM software will probably deliver unpredictable results (sort of like a cat chasing its own tail).

## Just in Case You Get Lost. . .

You can always press **RESET** to get the system under control. Unless your program destroyed itself in a crash (this can happen), code loaded into RAM is normally preserved during a reset.



## Instructions

1. Compile and download the following program into your computer.

```
;
;STACK AND SUBROUTINES
;
;DATE: 12/7/2023
;

        ORG 8200H

START: LD    SP,9000H    ;SET STACK TO POINT AT RAM

        NOP            ;(1) NOTE THE INITIAL SP VALUE HERE

        LD     C,2      ;Counter for program main loop

LOOP:   LD     B,4
        CALL  0053H    ;CALL KERNEL "SHORT DELAY" ROUTINE
                          ;(2) NOTE SP VALUE AFTER CPU REACHES 0053H

        LD     HL,1234H ;(3) NOTE SP VALUE WHEN CPU REACHES THIS INSTR
        LD     DE,5678H

        CALL  UNKNOWN

        NOP            ;(8) WHAT IS SP HERE AND WHY?

        JP     START

;
;UNKNOWN ROUTINE
;

UNKNOWN:
        NOP            ;(4) WHAT IS SP HERE AND WHY?

        PUSH  HL
        PUSH  DE

        NOP            ;(5) SP VALUE HERE?

        POP   HL
        POP   DE
        NOP            ;(6) EXPLAIN VALUES NOW IN DE & HL
        NOP            ;(7) WHAT IS SP HERE AND WHY?
        RET

        END
```

- Trace through this program by entering the following command at the remote console (*MiniTerminal* window) at the Monitor Ready prompt:

T 8200 **ENTER**

You will see a display of the CPU registers, followed by the first instruction of the program. The screen output should look like this:

```
Monitor Ready
>T 8200

AF  BC  DE  HL  IX/Y PC/SP
8EA0 2988 233C 266A E2EA 8200
0882 0004 8CB6 AA08 3EEA 0000
C=0  N=0  V=0  H=0  Z=0  S=1

8200:31 00 90 LD SP,9000
[ENTER]=STEP, [STOP]=EXIT, [M]=MULTISTEP, [R]=EDIT REGISTERS
>_
```

*Since your program hasn't loaded any CPU registers yet, the only values that will match the sample above will be PC and SP.*

Press **ENTER** on the PC to execute this first instruction, and you'll see new output. The SP register will now hold 9000H since this first instruction was completed.

You can now walk through each instruction of the program and take careful note of what's happening. Press **ENTER** to execute each subsequent instruction.

TIP: If you miss something, just reset the computer and start the trace command again.

- Record the Stack Pointer (SP) register contents at each of the nine (9) test points in the program, and comment on each as needed.

Test Point	Stack Pointer (SP) Value	Explanation
(1)		
(2)		
(3)		
(4)		
(5)		
(6)		
(7)		
(8)		

4. From what you've observed, what do you suppose the purpose of the UNKNOWN subroutine is?

---

---

## Sign Off

When you've completed Step 4, show your results to the instructor. On to the questions!

Sign Off \_\_\_\_\_

## Questions

1. What is the purpose of the machine stack?

---

---

2. What CPU register is used to manage the operation of the machine stack? What happens to this register's contents when an item is (a) put on the stack, and (b) removed from the stack.

---

---

---

3. In the test program, what is the importance of the first instruction (LD SP, 9000H)?

---

---

4. Explain how the UNKNOWN subroutine swaps the HL and DE register contents.

---

---

---

5. Why must each subroutine end with a RET (return) instruction?

---

---

6. Summarize what you have learned in this experience. Do you have suggestions to improve it?

---

---

---

## 12: Controlling Real-World Devices - Traffic Light Controller

### Introduction

Up to now, all of our efforts have been theoretical. We've built the computer and loaded programs into it. As the computer ran our programs, we watched it produce results - - at least on paper. We will now interface our computer with the outside world. It will perform useful work for us.

In this experiment we'll build a traffic light controller with the computer. It will operate just like a traffic light sequencer at a real intersection, just in scaled-down form.

Real-world computer applications require both hardware and software to work together. We'll do it this way here. First we'll construct and test the hardware. Once the hardware is working, we will then configure the software to run correctly. This is how computers are set up in industry, so it will be our first "real life" experience.

### Materials Needed

- EZ-80 computer in good working order with USB Serial Adapter
- *Mini Terminal* software (installed on JCCC laboratory computers)
- 16-pin DIP jumper cable, 18" length
- Solderless breadboard and electronic parts per the included schematic diagram
- EZ-80 Schematic Diagram

### Part 1 - Hardware

1. Construct the circuit of Figure 1 on a solderless breadboard. It will be helpful to first install one end of the DIP cable onto your breadboard. Pay close attention to the orientation of the pins on the cable end (they're marked).

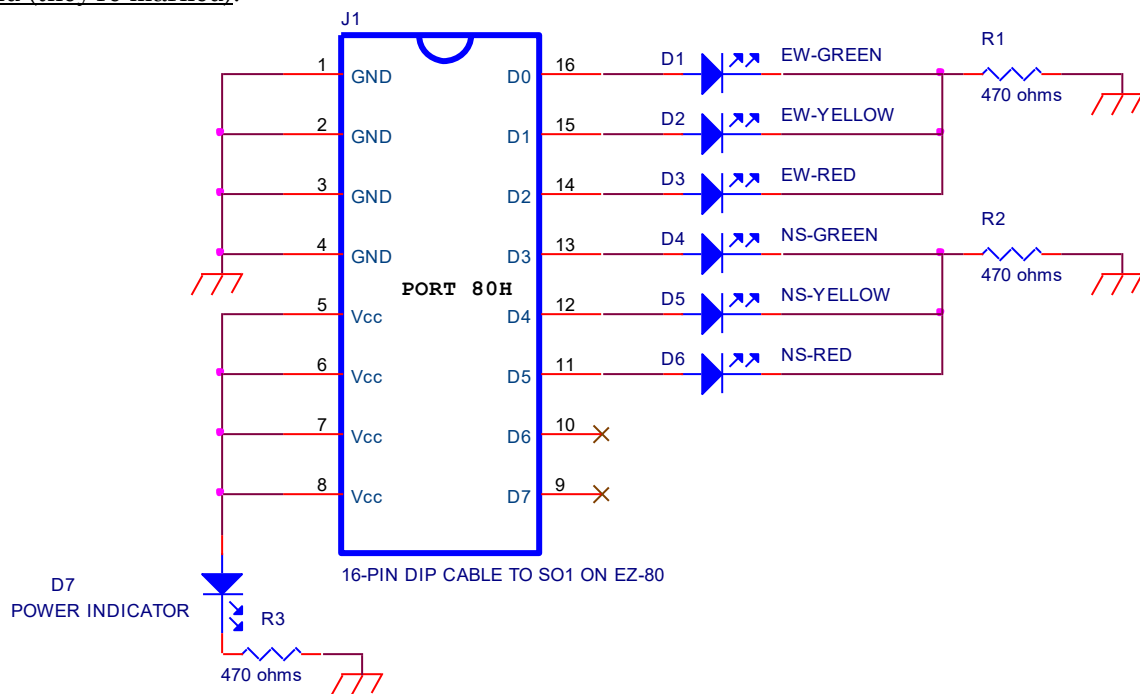


Figure 1: Traffic Light Circuitry

- From the schematic diagram we know that the LEDs of the traffic light are connected to Output Port 80H pins D0 through D5. Therefore, any data written to Output Port 80H will show up on our LEDs. A "1" in each position turns on the respective LED and a "0" turns it off. *The EZ-80 will be supplying the power that operates the LEDs.*

With the power to the EZ-80 OFF, connect the breadboarded circuit to the computer. Have your instructor review this work before proceeding.

- Attach the PC and serial cable to the computer. Launch the *Mini Terminal* software. Don't forget to select the correct COM port number.

Apply power to the EZ-80 computer. Press **ENTER** on the PC keyboard. You should see the familiar **Monitor Ready** prompt.

If you don't get **Monitor Ready**, stop and determine what's wrong before proceeding any further.

- The Monitor has the ability to read and write data from I/O ports on the computer. We're going to use this ability to test our hardware. The "O" (Output to port) command will be used to test each LED - - we know how to test this because each LED is connected to a different bit as shown in the table below.

For example, to turn on the EW GREEN LED, we would issue the following command at the Monitor Ready Prompt:

o 80 01 **ENTER**

In response, the EW GREEN LED should illuminate.

Complete the binary and hexadecimal values in the table, then use the Output command to verify that each LED works correctly.

LED Description	Port 80H Bit#	Binary Value	Hexadecimal Value
EW GREEN	0	0000 0001	01H
EW YELLOW	1		
EW RED	2		
NS GREEN	3		
NS YELLOW	4		
NS RED	5	0010 0000	20H

**TIP: If you're unable to get the LEDs to light, check the following:**

- √ Are the ground (GND) pins from the SO1 cable connected to your breadboard?
- √ Are the LEDs installed in the correct direction? (Yes, it happens to the best of us.)
- √ Use a multimeter to check the voltage on each SO1 output pin (socket pins 9 through 16). You should see 5V when 1s are written to the respective pins. If that is okay, you can simply follow the circuit through to the LEDs.

## Sign Off - Hardware Working Correctly

If you're successful in Step 4, congratulations - your hardware is working correctly. Have the instructor sign you off.

Sign Off \_\_\_\_\_

## Part 2 - Software

Now that we've got working hardware, it's time to get the software in gear. With this step completed our system will be complete. In the real world, we are often configuring software to do our bidding - and often changing it to meet customer needs. It will be helpful to understand what the software is doing before tweaking on it.

### Software Analysis

It will be helpful for you to follow and mark-up Listing 1 as we walk through the description of the code. This may look forbidding at first, but it's really quite simple when divided into sections.

The software can be divided into four portions - initialization, run-time routine, delay subroutine, and lookup tables. During initialization, the program prints a sign-on message to the serial console, then initializes the `CYCLE_COUNTER` variable in RAM to 1. The `CYCLE_COUNTER` is a 16-bit unsigned integer variable, so it has a modulus of  $2^{16}$  (or 65,536) and can count from 1 to 65,535.

The run-time routine is a loop. This loop turns on the signal lights appropriate to each state possible for the intersection. There are six states defined by the `LIGHT_TABLE`, which is a lookup table in RAM. The `LIGHT_TABLE` contains the binary codes to operate the correct lamps during each state of the intersection. This table is terminated with a zero byte to mark its end. This is how the run-time routine knows the end of the table has been reached. The HL register pair is used to address each entry in the `LIGHT_TABLE`.

For example, as shown in the table on the next page, the first state of the `LIGHT_TABLE` represents a GREEN light for the East-West (EW) portion of the intersection, and a RED light for the North-South (NS) portion.

This binary value is **0010 0001**, or **21H**. Writing this value to Output Port 80H will light the correct LEDs.

After the correct LEDs have been turned on, the run-time code then looks in the `DELAY_TABLE` to determine how long this combination of lamps must be lit. The DE register pair is used to access the delay table entries. A subroutine called `DELAY_SECONDS` is called to carry out the necessary delay.

After the delay, the run-time routine simply increments HL and DE to move to the next entries in the `LIGHT` and `DELAY` tables. When all the entries have been acted upon, the run-time increments the `CYCLE_COUNT` variable, then branches back to `NEW_CYCLE` to begin the next cycle for the intersection.

## Part 2 Procedure

1. Using the procedures we've previously developed for remote operation, enter the software of Listing 1 into a new project on the *ASM80.COM* website.
2. We need to figure out the correct binary patterns for each of the six traffic light states. We're going to load these into the `LIGHT_TABLE`. Don't panic!

These binary values are easily obtained by performing a bitwise-OR on the appropriate values from the table you completed in Step 4 of Part 1.

There are six states for the intersection, so we need to compute six binary values.

For example, in State 1, the EW GREEN and NS RED lights must be on, and all others must be off. According to the table of Step 2 we know that the required bit patterns are as follows:

EW GREEN = **0000 0001**                      and                      NS RED=**0010 0000**

We simply OR these two values together to get the pattern that will light the correct LEDs. This pattern is **0010 0001**, or in hexadecimal, 21H. So the first entry in the LIGHT\_TABLE is 21H.

You can test this by using the `O` command of the Monitor. Simply write 21H to Port 80H:

```
O 80 21
```

You'll see the correct lights illuminated.

Complete the table below, then transfer your results to the `LIGHT_TABLE` entries in the program source code. *Don't program the DURATION values just yet. Leave them at 1 second each.*

State	Binary Value	Hexadecimal Value	Duration (Seconds)
1 - EW GREEN, NS RED	<b>0010 0001</b>	<b>21H</b>	<b>10</b>
2 - EW YELLOW, NS RED			<b>5</b>
3 - EW RED, NS RED			<b>4</b>
4 - EW RED, NS GREEN			<b>20</b>
5 - EW RED, NS YELLOW			<b>7</b>
6 - EW RED, NS RED			<b>4</b>

3. Run the program by issuing a `G` command to address 8200H. You should see the intersection cycle through smoothly, with one second allotted for each state. If you see this, congratulations - - you're almost there! (You'll need to reset the computer with the **RESET** button to regain control.)



5. To complete the software configuration, program the DURATION values from the table of Step 3 into the DELAY\_TABLE.
6. Run it again - and give a holler if it works! Of course, don't forget to get a sign off.
7. Please complete the Conclusions page prior to turning this lab in.

## Sign Off

Hardware and software working together correctly - intersection cycles correctly with delays as specified.

Sign Off \_\_\_\_\_

## Listing 1 - Traffic Light Application Code

```
;
;TRAFFIC LIGHT APPLICATION
;
;AUTHOR: WHEELER, T
;DATE: 12/10/2023
;
;REVISION HISTORY: 12/10/2023 ADAPTED TO EZ-80 HOST
;TARGET SYSTEM: EZ-80 WITH MONITOR V1.0 OR LATER
;

LED_PORT      EQU 80H      ;Port 80H, S01 ON EZ80

;
;DIRECT CALLS TO KERNEL I/O ROUTINES
;

PRTSTG        EQU    3BH      ;Print string addressed by <HL>
DELAY_100MS   EQU    50H      ;Delay for number of 100 ms periods in <BC>
PRINT_INT     EQU    41H      ;Print integer held in <HL>
LCD_CHROUT    EQU    44H      ;Print to LCD only
CLS           EQU    0CH      ;Character code to clear LCD

;
;BEGIN TRAFFIC LIGHT CODE
;

                ORG 8200H

;
;INITIALIZATION STEPS
;
;1. PRINT SIGN-ON MESSAGE
;2. SET CYCLE COUNTER (IN B-REGISTER) TO 1
;

START:         LD      HL,MSG1      ;Sign-on message
               CALL   PRTSTG

               LD      HL,1         ;Initialize CYCLE counter (CYCLE = 1)
               LD      (CYCLE_COUNTER),HL

;EACH TIME THE TRAFFIC LIGHT BEGINS A NEW CYCLE, CONTROL PASSES HERE.
;THE SYSTEM PRINTS A CONSOLE MESSAGE DETAILING THE CYCLE STARTING, ALONG WITH
;THE CYCLE NUMBER.

NEW_CYCLE:

               LD      A,CLS
               CALL   LCD_CHROUT    ;CLEAR LCD SCREEN
               LD      A,13
               CALL   LCD_CHROUT    ;FORCE LCD CURSOR TO TOP LINE

               LD      HL,MSG4
               CALL   PRTSTG        ;ANNOUNCE NEW CYCLE
               LD      HL,(CYCLE_COUNTER)
               CALL   PRINT_INT     ;PRINT CYCLE #

;
;SETUP FOR THE LOOKUP TABLES (LIGHTS AND DELAYS) NEEDED TO OPERATE THE TRAFFIC LIGHT.
;<HL> POINTS TO THE LIGHTS TABLE, <DE> ADDRESSES THE DELAY TABLE.
;

               LD      HL,LIGHT_TABLE
               LD      DE,DELAY_TABLE
               LD      C,1          ;STATE COUNTER
```

```

;
;FOR EACH STATE IN LIGHT_TABLE, TURN ON THE INDICATED TRAFFIC LIGHTS
;
LP1:
        LD      A, (HL)          ;PULL VALUE FROM (HL) WHICH POINTS TO THE LIGHTS TABLE
        OUT    (LED_PORT),A     ;WRITE BINARY DATA TO TRAFFIC LIGHT LEDs

;
;WRITE STATUS MESSAGE WITH STATE NUMBER TO SERIAL PORT EACH TIME THE TRAFFIC LIGHT
;CHANGES STATE. THIS CAUSES DETAILS TO APPEAR ON THE MINI TERMINAL.
;

        PUSH   HL                ;Save POINTER to LIGHT_TABLE

        LD     HL,MSG2           ;Print STATE message
        CALL  PRTSTG            ;Print STATE message
        LD     L,C              ;GET STATE #
        LD     H,0
        CALL  PRINT_INT        ;PRINT STATE #

        LD     HL,MSG3

        CALL  PRTSTG            ;Print DELAY message
        LD     A, (DE)          ;Retrieve DELAY parameter in seconds from DELAY table
        LD     L,A
        LD     H,0
        CALL  PRINT_INT        ;PRINT DELAY IN SECONDS
        LD     A, (DE)          ;PULL DELAY FROM DELAY TABLE
        CALL  DELAY_SECONDS

        POP   HL                ;Restore LIGHT_TABLE pointer

        INC   HL
        INC   DE                ;Point to next LIGHT and DELAY information
        INC   C                ;Get next STATE number for display routine

        LD     A, (HL)          ;Check for END OF LIGHT_TABLE which is marked
        OR    A                ;with a ZERO (00H) byte.
        JP    NZ,LP1           ;If not at END OF TABLE, go back to loop LP1

;
;UPDATE CYCLE COUNTER (16-BITS)
;
        LD     HL, (CYCLE_COUNTER)
        INC   HL
        LD     (CYCLE_COUNTER),HL

        JP    NEW_CYCLE        ;GOTO NEW_CYCLE

;
;SUBROUTINE TO DELAY IN SECONDS, BASED ON VALUE IN A-REGISTER
;
;THIS DELAY IS BASED ON A CLOCK OF 3.579545 MHz AND THE VALUE
;IS APPROXIMATE (WITHIN 1%). YOU CAN TWEAK THE TWO <HL> LOAD VALUES
;TO BETTER APPROXIMATE 1 s TIMING IF NEEDED.
;

DELAY_SECONDS:
        PUSH  AF
        PUSH  BC
        LD    B,A
DELAY1S_LOOP:
        PUSH  BC
        LD    BC,10            ;PARAMETER: WAIT FOR TEN (10) 100 ms INTERVALS
        CALL  DELAY_100MS     ;USE KERNEL ROUTINE TO WAIT 1 s
        POP   BC
        DJNZ DELAY1S_LOOP
        POP   BC
        POP   AF
        RET

```

```

;
;LIGHT_TABLE
;
;THIS TABLE HOLDS THE BIT-MAPPED PATTERNS THAT REPRESENT THE ON-OFF CONDITION
;OF INDIVIDUAL LAMPS FOR THE INTERSECTION. THESE ARE MAPPED AS FOLLOWS:
;
;D0 - EW GREEN
;D1 - EW YELLOW
;D2 - EW RED
;D3 - NS GREEN
;D4 - NS YELLOW
;D5 - NS RED
;
;EACH ENTRY IN THE TABLE IS ONE STATE FOR THE INTERSECTION, AND A
;00H BYTE MARKS THE END OF THE TABLE.
;
;YOU MUST DETERMINE THE VALUES FOR THESE BYTES. SEE THE INSTRUCTIONS OF PART 2.

LIGHT_TABLE:
    DB  XXH           ;STATE 1 - EW GREEN, NS RED
    DB  XXH           ;STATE 2 - EW YELLOW, NS RED
    DB  XXH           ;STATE 3 - EW RED, NS RED
    DB  XXH           ;STATE 4 - EW RED, NS GREEN
    DB  XXH           ;STATE 5 - EW RED, NS YELLOW
    DB  XXH           ;STATE 6 - EW RED, NS RED
    DB  0             ;Marks END OF TABLE and END OF CYCLE

;
;DELAY_TABLE
;
;THIS TABLE HOLDS THE VALUE, IN SECONDS, THAT EACH STATE WILL LAST.
;IT MUST HAVE THE SAME NUMBER OF ENTRIES AS THE LIGHT_TABLE. NO 00H
;END-OF-TABLE MARK IS REQUIRED AS THAT'S ALREADY INCLUDED IN THE LIGHT_TABLE.
;
;YOU MUST DETERMINE THE VALUES FOR THESE BYTES. SEE THE INSTRUCTIONS OF PART 2. LEAVE
;THESE VALUES INTACT FOR INITIAL TESTING.

DELAY_TABLE:
    DB  1             ;STATE 1 TIME IN SECONDS
    DB  1             ;STATE 2 TIME
    DB  1             ;STATE 3 TIME
    DB  1             ;STATE 4 TIME
    DB  1             ;STATE 5 TIME
    DB  1             ;STATE 6 TIME

;
;STRINGS
;

MSG1:      DB 13,13,'Traffic Light Simulation',13,0
MSG2:      DB 13,'State:',0
MSG3:      DB  ',Secs: ',0
MSG4:      DB 13,'CYCLE #',0

;DATA SEGMENT -- VARIABLES

CYCLE_COUNTER:  DW      0

END

```

## Conclusions

Summarize what you've learned in this experiment.

---

---

---

---

---

Do you have any suggestions to improve this experience?

---

---

---

---

---



## 13: Digital to Analog Conversion

### Introduction

It's an analog world. Most of the quantities we would like to control with a digital computer are continuously-variable analog quantities. These include motor speed, temperature, brightness, sound, and so forth. *Digital-to-analog converters* (DACs) translate computer binary values into analog voltages and currents. In this experiment we're going to build and test a simple DAC called an R2R Ladder DAC. With this DAC in the circuit our computer will be able to do many analog things. We'll even get to see it make synthetic sine, triangle and square waves. Exciting stuff!

### Materials Needed

- Working EZ-80 system with connected Personal Computer
- One 16-pin DIP jumper cable, 18" length
- Solderless breadboard and electronic parts per the included schematic diagram
- Digital Multimeter and Digital Storage Oscilloscope (at least 50 MHz bandwidth)

### Part 1 - Hardware Construction and Test

1. Construct the circuit of Figure 2 (overleaf) on a solderless breadboard. It will be helpful to first install one end of each DIP cable onto your breadboard. Pay close attention to the orientation of the pins on the cable end (they're marked). R17 can be fitted under the cable.

Your circuit should be constructed in a manner very similar to Figure 1. Since there are a lot of resistors in the D-A converter, use extra care to locate each value in the right place. Cutting the resistor leads short will also help prevent short circuits. *If you space R8~R16 as shown (every other hole), your work will be much easier.*

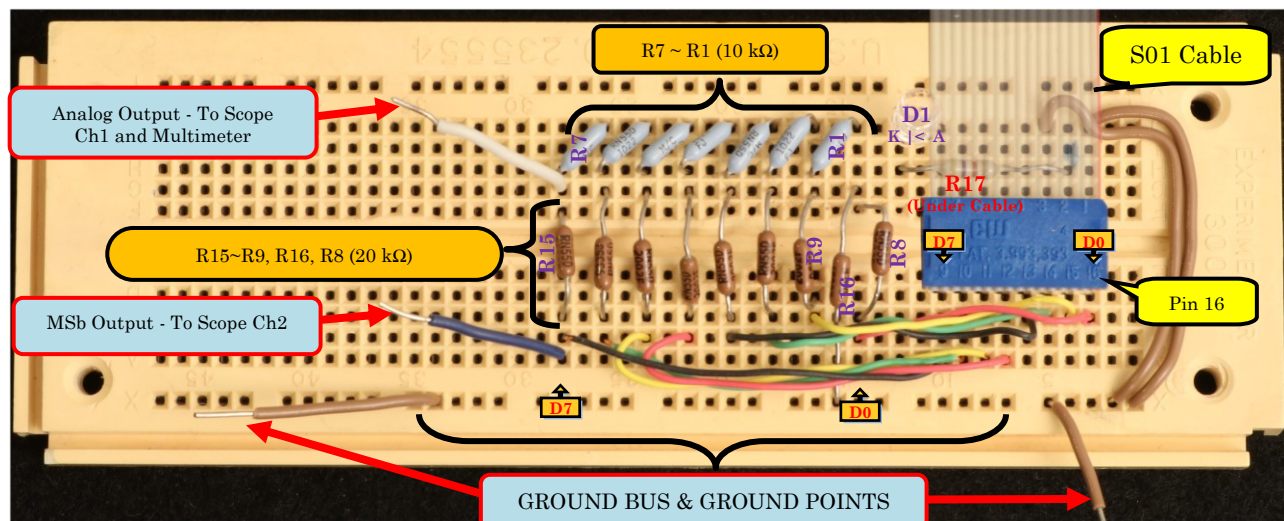
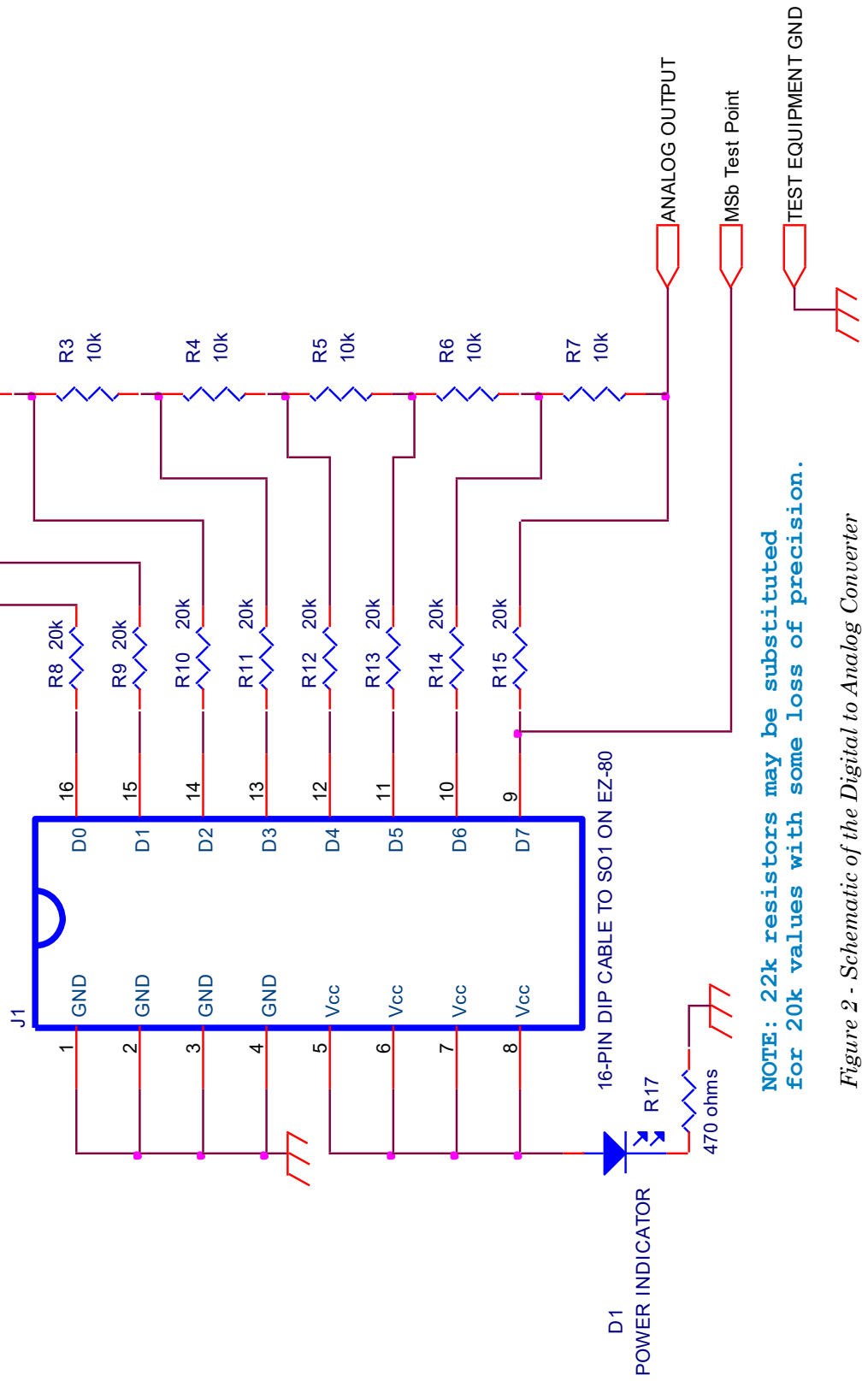


Figure 1: Suggested Layout of the Digital to Analog Converter



**NOTE: 22k resistors may be substituted for 20k values with some loss of precision.**

*Figure 2 - Schematic of the Digital to Analog Converter*



- Using 10:1 probes in the 10X position, connect the oscilloscope channels to the circuit as indicated in Figures 1 and 2.

Channel 1 should be connected to the **ANALOG OUTPUT**.

Channel 2 should be coupled to the **MSb Test Point**.

Set the scope to trigger from Channel 2.

Connect both scope ground leads to the ground bus on the breadboard.

- Connect the digital multimeter ground lead to the ground bus on the breadboard. (The scope should also be grounded in this same location.)

**Connecting the instrumentation ground leads to the breadboard ground maximizes the accuracy of measurements by eliminating the small voltage drops inherent in the DIP jumper cable.**

- Have your instructor check your work before powering up the EZ-80.

- Launch the *Mini Terminal* application. Power up the EZ-80 and sign into the system (press **ENTER** on the *Mini Terminal* screen; you should see the **Monitor Ready** prompt).

*If you don't see Monitor's sign-on message, determine the fault before continuing.*

- We need to determine the approximate DC reference voltage  $V_{REF}$  of the DAC. To do so, proceed as follows:

- Connect the DC multimeter temporarily to the **MSb Test Point**.

- Using the **O (Out Port)** command, write FFH to the SO1 output port register at address 80H:

**O 80 FF** **ENTER**

- Record the DC multimeter reading from the **MSb Test Point**. This is the approximate value of the DAC reference voltage - - it should be  $5\text{ V} \pm 20\text{ mV}$ . Record to an accuracy of four significant digits.

$V_{REF} =$  \_\_\_\_\_

- For convenience, we will use the *DAC Test* program of Listing 1 to exercise and test the operation of the DAC. Assemble, load, and execute this program.

- Move the DC voltmeter positive lead to the **ANALOG OUTPUT**. We are going to calculate and measure the DC performance of the DAC.

9. We can test the DAC by measuring its resolution (smallest possible voltage step) and DC output voltage range.

The resolution (smallest possible voltage step) of a DAC is computed by using Equation 11-1:

$$(11-1) \Delta V = \frac{V_{REF}}{2^N} \text{ (Note that } \Delta V \text{ is pronounced "delta vee.")}$$

where  $V_{REF}$  is the DC reference voltage, and  $N$  is the number of bits in the DAC. Since your DAC has 8 bits,  $N=8$ .

Using your estimated measurement of  $V_{REF}$  from Step 6, compute the theoretical resolution of your DAC below. Show all work.

Theoretical Resolution  $\Delta V$ : \_\_\_\_\_

10. We also know that the minimum and maximum DC voltages of a DAC are easy to calculate as shown in Equations (11-2) and (11-3):

$$(11-2) V_{MIN} \approx 0V$$

$$(11-3) V_{MAX} = V_{REF} \left( \frac{2^N - 1}{2^N} \right) = V_{REF} - \Delta V$$

Calculate the theoretical  $V_{MIN}$  and  $V_{MAX}$  values below; show your work.

Theoretical  $V_{MIN}$  = \_\_\_\_\_

Theoretical  $V_{MAX}$  = \_\_\_\_\_

TIP: Your value for  $V_{MAX}$  should agree with Equation 11-3 - - just one " $\Delta V$ " below  $V_{REF}$ .

11. Using the *DAC Test* application, exercise the DAC as follows to obtain actual performance values.

TIP: You can press **ENTER** to directly enter hexadecimal write values.

a) Write 00H to the DAC and record the actual  $V_{MIN}$ .

Actual  $V_{MIN}$  = \_\_\_\_\_

b) Write FFH to the DAC and record the actual  $V_{MAX}$ .

Actual  $V_{MAX}$  = \_\_\_\_\_

12. You can check the actual resolution of the DAC as well. Devise your own method for doing this and report the results you obtain.

---

---

13. How do the values you obtained compared to the theoretical values from Steps 9 and 10?

---

---

14. What could your DAC circuit be used for? (Is there a real-life application you can think of?)

---

---

## Sign Off

Have your instructor review your results from Steps 10 through 14. Congratulations - your DAC is operational!

Sign Off - DAC Operational \_\_\_\_\_

## Part 2 - DAC AC Operation

In this section we're going to make the DAC produce synthetic AC waveforms. In fact, the DAC we have just constructed is the equivalent circuit of one channel of a conventional PC sound card. With the right software (and some additional memory), our circuit can produce music, speech, and other sounds.

We are going to watch the DAC produce triangle, square, and sine waves. The method we are going to use is exactly what is used in a modern synthesized signal generator - the only difference is that the processor in a signal generator is much faster than our humble little Z80 microprocessor.

1. Assemble, load and execute the *Waveform Synthesis* application.

a) Choose 1 - SQUARE. You should see a square wave on both Channel 1 and 2.

**Adjust the scope to capture a couple cycles of this output and print this output.**

b) Reset the computer and again run the Waveform Test, but choose 2 - TRIANGLE.

**Again adjust the scope to capture a couple of cycles and print this output.**

2. The *monotonicity* of a DAC can be assessed by testing it with a triangle wave. A monotonic DAC produces a very straight, linear slope when reproducing a triangle wave.

Comment on the appearance of the triangle wave your DAC is generating. How monotonic does it appear to be? Can you see the "steps" in the DAC output?

---

---

3. Return to the Waveform Synthesis main menu and choose 3 - SINE WAVE - FUNDAMENTAL FREQUENCY. This program uses the DAC to produce an artificial (synthesized) sine wave.

**Adjust the scope to capture a couple of cycles and print this output.**

Comment on the appearance of the sine wave. How does it compare to the sine wave from the benchtop signal generator, for example?

---

---

4. By selectively discarding samples (data points) from a digitized waveform, we can alter its frequency without altering the fundamental sample rate. This process is called *decimation*. (This term comes from the word *decimal*, meaning ten. In modern usage, when a group is decimated, its numbers are greatly reduced.)

By choosing 4 - SINE WAVE - SECOND HARMONIC, we can see what happens to the frequency of a sine wave when we discard every other data point. We call this two-fold ( $M=2$ ) decimation. We can create any frequency in this manner from a single sine wave - - even the tones of a musical scale. But we have to be careful. We are, after all, throwing away information!

**Adjust the scope to capture a couple of cycles and print this output.**

Comment on the appearance of the sine wave. How does it compare to the sine wave from the prior step? How does its frequency compare to the original sine wave?

---

---

5. If we go really nuts, we can try throwing out three of every four data points. 5 - SINE WAVE - FOURTH HARMONIC does just that. This is a four-fold ( $M=4$ ) decimation of the original sine wave. It only requires 1/4th of the original information since only one in four data points is utilized. But when you examine the waveform, you'll notice that it's now distorted in a new way.

What is the nature of the "distortion" on the reproduced waveform? What would you suggest doing to the circuit to reduce this distortion? (Think in particular how L-C and R-C filter circuits could help.)

---

---

---

---

## Sign Off

Have your instructor review your output from Step 5. You're almost done!

Sign Off - DAC AC Test is Operational \_\_\_\_\_

## Conclusions

Summarize what you've learned in this experiment.

---

---

---

---

---

Do you have any suggestions to improve this experience?

---

---

---

---

---

```

;
;LISTING 1
;
;DACTEST APPLICATION
;
;AUTHOR: WHEELER, T
;DATE: 12/11/2023
;
;REVISION HISTORY: NONE
;TARGET SYSTEM: EZ-80 SYSTEM WITH MONITOR VERSION 1.01 OR LATER
;

DAC_PORT      EQU 80H      ;EZ-80 Port 80H, SO1

BUFFER_SIZE   EQU 40      ;SIZE OF INPUT/OUTPUT BUFFER IN BYTES

;
;
;DIRECT CALLS TO KERNEL I/O ROUTINES
;

PRTSTG        EQU 3BH     ;Print string addressed by <HL>
DELAY_100MS   EQU 50H     ;Delay for number of 100 ms periods in <BC>
DELAY_SHORT   EQU 53H     ;Short delay held in B
PRINT_INT     EQU 41H     ;Print integer held in <HL>
LCD_CHROUT    EQU 44H     ;Print to LCD only
PARSE_HEX     EQU 79H     ;Convert ASCII hex number pointed by HL to binary in DE
LINE_INPUT    EQU 6DH     ;Input to buffer @ (HL), maxlen=B;
                                ;Return: C=length, CY=1 means ESCAPE
PRHEX         EQU 82H     ;Print 8-bit HEX value in A
PR2HEX        EQU 85H     ;Print 16-bit HEX value in DE

;ASCII KEY VALUES

ESC           EQU 27      ;[ESC] KEY
ENTER_KEY    EQU 13      ;[ENTER] KEY
CLS          EQU 0CH      ;Character code to clear LCD

;
;BEGIN APPLICATION CODE
;

                ORG 8200H

START:         LD      C,80H      ;INITIAL VALUE FOR DAC

                LD      HL,APP_MSG
                CALL   PRTSTG

LOOP:         LD      HL,MSG2      ;MESSAGE: DAC VALUE
                CALL   PRTSTG
                LD      A,C
                CALL   PRHEX      ;OUTPUT VALUE IN HEX
                LD      A,C
                OUT    (DAC_PORT),A ;WRITE TO DAC ON PORT 80H (SO1)

;GET COMMAND FROM USER

WAIT_CMD:     LD      HL,CMD_MSG
                CALL   PRTSTG
W1:          RST      18H      ;CHRIN -> GET CHAR FROM USER
                JR      Z,W1      ;WAIT IF NO CHARACTER TYPED

                CP      ESC
                JP      Z,EXIT

                CP      '1'
                JP      Z,UP

```

```

                CP      '2'
                JP      Z,DOWN

                CP      ENTER_KEY
                JP      Z,ENTER

                LD      HL,WHAT_MSG      ;UNKNOWN COMMAND - PRINT 'WHAT?' AND SHOW MENU
                CALL    PRTSTG
                JP      WAIT_CMD

;
;ENTER KEY - GET DIRECT HEX VALUE FROM USER AND WRITE IT TO DAC
;
ENTER:          LD      HL,MSG3
                CALL    PRTSTG
                CALL    INPUT_HEX_VALUE
                LD      C,E              ;GET LSB OF VALUE TO SEND TO DAC
                JP      LOOP

;
;UP AND DOWN INCREMENT ROUTINES
;
DOWN:          DEC     C
                JP      LOOP

UP:            INC     C
                JP      LOOP

EXIT:          RST     20H              ;BACK TO MONITOR

;
;SUBROUTINE: TOUPPER
;
;CONVERTS CHARACTER IN .A TO UPPERCASE
;
TOUPPER:       CP      'a'              ;< 'a' -- NO CORRECTION NEEDED
                RET     C
                CP      'z'+1          ;> 'z' -- NO CORRECTON NEEDED
                RET     NC
                XOR     20H            ;FLIP BIT TO FORCE TO UPPERCASE
                RET

;
;SUBROUTINE: UCASE
;
;CONVERTS STRING @ HL TO UPPERCASE
;
UCASE:         LD      A,(HL)
                OR      A
                RET     Z
                CALL   TOUPPER
                LD      (HL),A
                INC     HL
                JR      UCASE

```



```

;
;SUBROUTINE: INPUT HEX VALUE
;
;PARAMETERS: NONE
;RETURNS: CY=1, ESCAPE OR STOP PRESSED
;          DE=NUMBER ENTERED, BINARY
;
INPUT_HEX_VALUE:
    LD     HL,BUFFER
    LD     B,BUFFER_SIZE - 2
    CALL  LINE_INPUT      ;GET INPUT FROM CONSOLE (ETC.)
    RET   C                ;CY=1, ESCAPE PRESSED
    CALL  UCASE           ;FORCE BUFFER TO UPPERCASE
    LD     HL,BUFFER
    CALL  PARSE_HEX       ;EVALUATE NUMBER ENTERED (HEX)
    XOR   A               ;CY=0, ANSWER IS IN DE
    RET

;
;STRINGS
;
WHAT_MSG:    DB 13,'WHAT?',13
APP_MSG:     DB 13,'DACTEST Application',13
             DB '1:UP',13,'2:DOWN',13,'[ENTER] :DIRECT ENTRY OF VALUE TO DAC',13,0

CMD_MSG:     DB 13,'CMD >',0

MSG2:        DB 13,'DAC VALUE: ',0
MSG3:        DB 13,'ENTER NEW DAC VALUE >',0

;I/O BUFFER FOR GETTING INFORMATION FROM USER

BUFFER:      DEFS     BUFFER_SIZE

                END

```

```

;
;LISTING 2
;
;WAVEFORM SYNTHESIS APPLICATION
;
;AUTHOR: WHEELER, T
;DATE: 12/11/2023
;
;REVISION HISTORY: NONE
;TARGET SYSTEM: EZ-80 WITH MONITOR VERSION 1.01 OR LATER
;

DAC_PORT EQU 80H          ;DAC Output Port

;
;DIRECT CALLS TO KERNEL I/O ROUTINES
;

PRTSTG      EQU      3BH          ;Print string addressed by <HL>
DELAY_100MS EQU      50H          ;Delay for number of 100 ms periods in <BC>
DELAY_SHORT EQU      53H          ;Short delay held in B
PRINT_INTEQU 41H          ;Print integer held in <HL>
LCD_CHROUT  EQU      44H          ;Print to LCD only

;ASCII KEY VALUES

ESC          EQU      27          ;[ESC] KEY
ENTER_KEY    EQU      13          ;[ENTER] KEY
CLS          EQU      0CH          ;Character code to clear LCD

;
;BEGIN APPLICATION CODE
;

                ORG 8200H

;GET COMMAND FROM USER

MENU:          LD      HL,MENU_MSG
                CALL   PRTSTG

WAIT_CMD:     RST      18H          ;CHRIN-READ CHAR FROM INPUT STREAM
                JR      Z,WAIT_CMD  ;WAIT FOR A CHAR TO BE TYPED

                CP      '1'
                JP      Z,SQUARE

                CP      '2'
                JP      Z,TRIANGLE

                CP      '3'
                JP      Z,SINE_FUNDAMENTAL

                CP      '4'
                JP      Z,SINE_SECOND_HARMONIC

                CP      '5'
                JP      Z,SINE_FOURTH_HARMONIC

                CP      '6'
                JR      NZ,WHAT_ERR
                RST      20H          ;RETURN TO MONITOR

```

```

WHAT_ERR:      LD      HL,WHAT_MSG
               CALL    PRTSTG
               JP      MENU

;ROUTINE TO ANNOUNCE BEGINNING OF EACH ROUTINE
;
;HL HOLDS 1st MESSAGE, DE HOLDS 2nd MESSAGE
;

ANNOUNCE:     PUSH    DE
               CALL    PRTSTG
               POP     HL
               JP      PRTSTG

;
;SQUARE WAVE ROUTINE
;

SQUARE:       LD      HL,SQUARE_MSG
               LD      DE,KEY_MSG
               CALL    ANNOUNCE

SQ_LOOP:      LD      A,0FFH
               OUT    (DAC_PORT),A
               PUSH   AF
               RST    18H           ;KEY PRESSED TO EXIT?
               JR     NZ,SQ_EXIT
               LD      B,0FFH
               CALL   DELAY_SHORT
               LD      B,0FFH
               CALL   DELAY_SHORT
               POP    AF
               CPL                    ;1s COMP A -> A TO GEN
                                       ;TOP & BOTTOM OF WAVEFORM
               JP     SQ_LOOP

SQ_EXIT:      POP    AF
               JP     MENU

;
;TRIANGLE WAVE ROUTINE
;

TRIANGLE:    LD      HL,TRIANGLE_MSG
               LD      DE,KEY_MSG
               CALL    ANNOUNCE

T_LOOP1:     XOR     A                    ;INITIAL VALUE
               OUT    (DAC_PORT),A
               INC    A
               JR     NZ,T_LOOP1       ;OVERFLOW --> REVERSE DIRECTION
               DEC    A                    ;FF-->A
               EX     AF,AF'
               RST    18H
               JR     NZ,MENU
               EX     AF,AF'

T_LOOP2:     OUT    (DAC_PORT),A
               DEC    A
               JR     NZ,T_LOOP2
               RST    18H           ;KEY PRESSED TO EXIT?
               JR     NZ,MENU
               JR     T_LOOP1

SINE_FUNDAMENTAL:
               LD      HL,SINE_MSG
               LD      DE,KEY_MSG
               CALL    ANNOUNCE
               LD      HL,SINE_LUT       ;WARNING: MUST RESIDE ON PAGE BOUNDARY

```

```

SINE_LOOP:      LD      A, (HL)
                OUT    (DAC_PORT), A
                INC    L                      ;WRAPS AROUND 256-SAMPLE BUFFER
                RST    18H
                JP     NZ, MENU
                JR     SINE_LOOP

SINE_SECOND_HARMONIC:
                LD      HL, SINE_MSG2
                LD      DE, KEY_MSG
                CALL   ANNOUNCE
                LD      HL, SINE_LUT          ;WARNING: MUST RESIDE ON PAGE BOUNDARY

SINE_LOOP2:    LD      A, (HL)
                OUT    (DAC_PORT), A
                INC    L                      ;WRAPS AROUND 256-SAMPLE BUFFER
                INC    L                      ;DECIMATE SAMPLE DATA (M=1:2)
                ;      (DOUBLE OUTPUT FREQUENCY)
                RST    18H
                JP     NZ, MENU
                JR     SINE_LOOP2

SINE_FOURTH_HARMONIC:
                LD      HL, SINE_MSG4
                LD      DE, KEY_MSG
                CALL   ANNOUNCE
                LD      HL, SINE_LUT          ;WARNING: MUST RESIDE ON PAGE BOUNDARY

SINE_LOOP4:    LD      A, (HL)
                OUT    (DAC_PORT), A
                INC    L
                INC    L
                INC    L
                INC    L                      ;DECIMATE SAMPLE DATA (M=1:4)
                ;      (QUADRUPLE OUTPUT FREQUENCY)
                RST    18H
                JP     NZ, MENU
                JR     SINE_LOOP4

SINE_MSG4:     DB      13, 'SINE MODE - FOURTH HARMONIC...', 0
SINE_MSG2:     DB      13, 'SINE MODE - SECOND HARMONIC...', 0
SINE_MSG:      DB      13, 'SINE MODE - FUNDAMENTAL FREQUENCY...', 0
TRIANGLE_MSG:  DB      13, 'TRIANGLE MODE...', 0
SQUARE_MSG:    DB      13, 'SQUARE WAVE MODE...', 0
KEY_MSG:       DB      13, 'PRESS ANY KEY TO EXIT.', 0

MENU_MSG:      DB      13, 13, 'WAVEFORM DEMO', 13, 13
                DB      '1=SQUARE WAVE', 13, '2=TRIANGLE WAVE', 13
                DB      '3=SINE WAVE - FUNDAMENTAL FREQUENCY', 13
                DB      '4=SINE WAVE - SECOND HARMONIC', 13
                DB      '5=SINE WAVE - FOURTH HARMONIC', 13, '6=EXIT', 13, '>', 0

WHAT_MSG:      DB      13, 13, 'WHAT?', 13, 0

```

```

;
;SINE WAVE LOOKUP TABLE (256 ENTRIES)
;
;THESE ARE COMPUTED TO GIVE A SINE WAVE WITH A PERIOD OF 256 SAMPLES AND
;AN AMPLITUDE OF 127.49 UNITS PEAK. WE ADD 128 TO THIS TO SUPERIMPOSE A DC LEVEL
;OF Vcc/2 -- ABOUT 2.5V -- ONTO THE SINE WAVE SO THAT WE CAN REPRODUCE BOTH THE
;POSITIVE AND NEGATIVE PEAKS WITHOUT DISTORTION. MULTIPLYING N BY PI/128 GIVES US
;A SINE PERIOD OF 256 SAMPLES (256*PI/128 = 2*PI RADIANS).
;
;FORMULA: y(N) = int( 127.49*sin(N*PI/128) + 128 )
;
;WARNING: THIS TABLE MUST RESIDE ON A PAGE BOUNDARY!
;
;THE TWO-STEP CALCULATION BELOW FORCES THIS TO HAPPEN (AT LEAST ON THE ASM80 ASSEMBLER)
;

```

```

PC_VAL      EQU ($/256)
TBL_START   EQU (PC_VAL+1)*256

                ORG TBL_START

```

```

SINE_LUT:
DB 128,131,134,137,140,143,146,149,152,155,158,162
DB 165,167,170,173,176,179,182,185,188,190,193,196,198
DB 201,203,206,208,211,213,215,218,220,222,224,226,228
DB 230,232,234,235,237,238,240,241,243,244,245,246
DB 248,249,250,250,251,252,253,253,254,254,254,255
DB 255,255,255,255,255,255,254,254,254,253,253,252,251
DB 250,250,249,248,246,245,244,243,241,240,238,237,235
DB 234,232,230,228,226,224,222,220,218,215,213,211
DB 208,206,203,201,198,196,193,190,188,185,182,179,176
DB 173,170,167,165,162,158,155,152,149,146,143,140
DB 137,134,131,128,124,121,118,115,112,109,106,103,100
DB 97,93,90,88,85,82,79,76,73,70,67,65,62,59,57,54
DB 52,49,47,44,42,40,37,35,33,31,29,27,25,23,21,20,18
DB 17,15,14,12,11,10,9,7,6,5,5,4,3,2,2,1,1,1,0,0,0
DB 0,0,0,1,1,1,2,2,3,4,5,5,6,7,9,10,11,12,14,15,17,18
DB 20,21,23,25,27,29,31,33,35,37,40,42,44,47,49,52
DB 54,57,59,62,65,67,70,73,76,79,82,85,88,90,93,97,100
DB 103,106,109,112,115,118,121,124

```

```

END

```