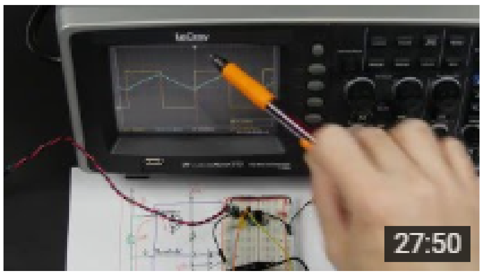
Build an 8-bit computer from scratch

Clock module

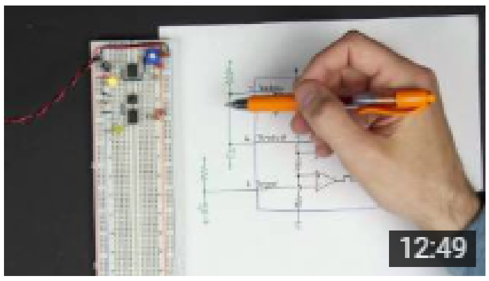
The computer's clock is used to synchronize all operations. The clock we're building is based on the popular 555 timer IC. The videos go into some detail on the operation of the 555 and use it in three different ways.

Our clock is adjustable-speed (from less than 1Hz to a few hundred Hz). The clock can also be put into a manual mode where you push a button to advance each clock cycle. This will be a really useful feature for debugging the computer later on.

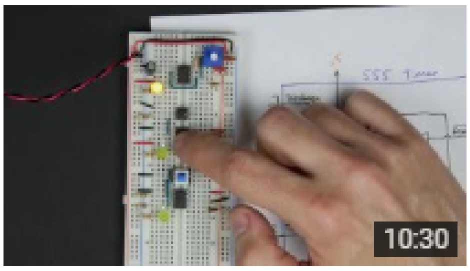
Videos



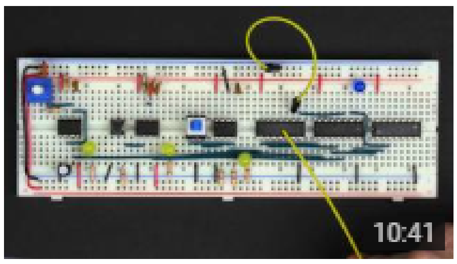
Astable 555 timer - 8-bit computer clock - part 1



Monostable 555 timer - 8-bit computer clock - part 2



Bistable 555 - 8-bit computer clock - part 3

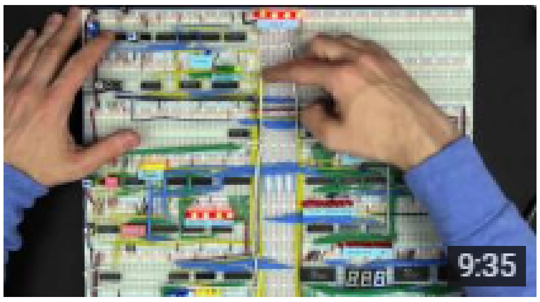


Clock logic - 8-bit computer clock - part 4

Registers

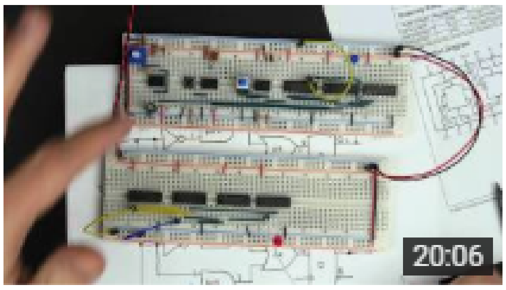
Most CPUs have a number of registers which store small amounts of data that the CPU is processing. In our simple breadboard CPU, we'll build three 8-bit registers: A, B, and IR. The A and B registers are general-purpose registers. IR (the instruction register) works similarly, but we'll only use it for storing the current instruction that's being executed.

Videos

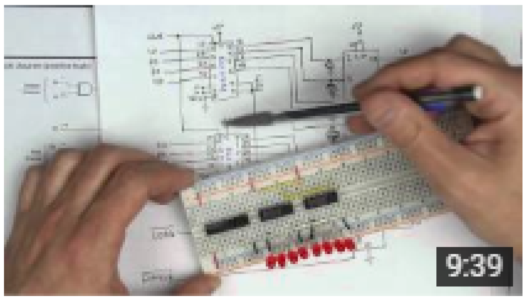


Bus architecture and how register transfers work - 8 bit register - Part 1

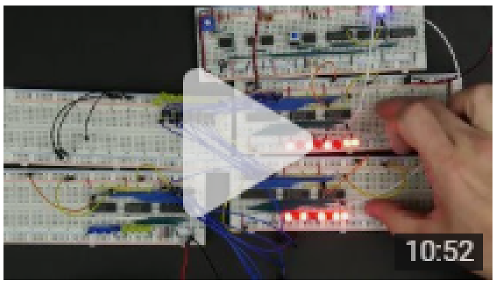
Tri-state logic: Connecting multiple outputs together - 8 bit register - Part 2



Designing and building a 1-bit register - 8 bit register - Part 3



Building an 8-bit register - 8-bit register - Part 4



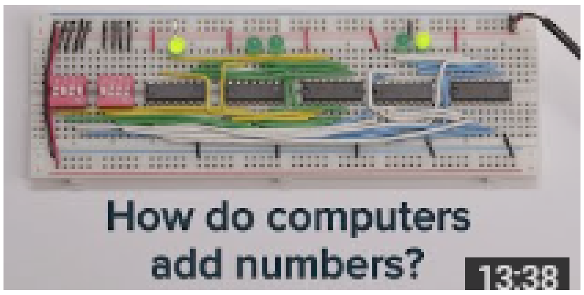
Testing our computer's registers - 8-bit register - Part 5

Arithmetic logic unit (ALU)

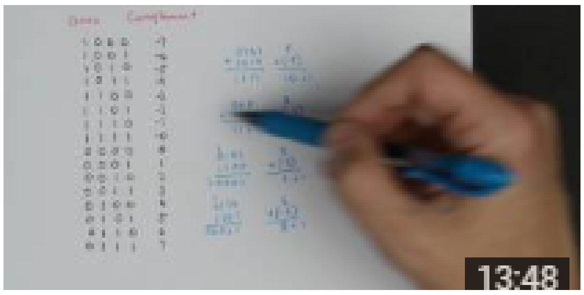
The arithmetic logic unit (ALU) part of a CPU is usually capable of performing various arithmetic, bitwise, and comparison operations on binary numbers. In our simple breadboard CPU, the ALU is just able to add and subtract. It's connected to the A and B registers and outputs either the sum of A+B or the difference of A-B.

Theory

Before diving into building the ALU, check out these two videos. The first shows how we can use hardware to add numbers in binary. The second video explains how negative numbers work. Between the two videos, you'll know how to add negative numbers, which means you'll know how to subtract!

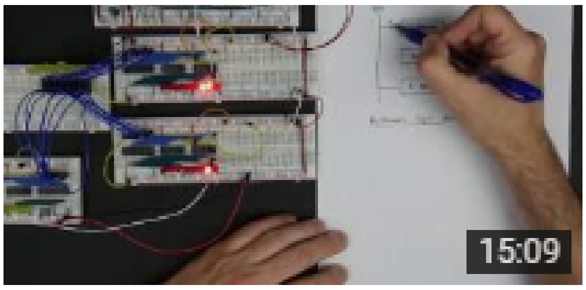


Learn how computers add numbers and build a 4 bit adder circuit

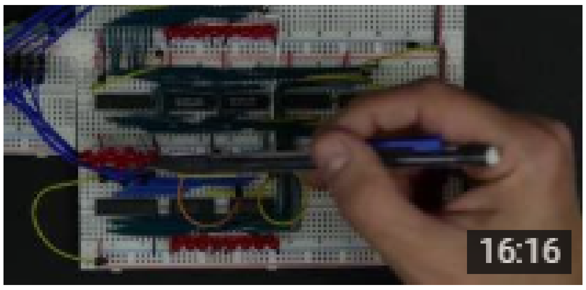


Twos complement: Negative numbers in binary

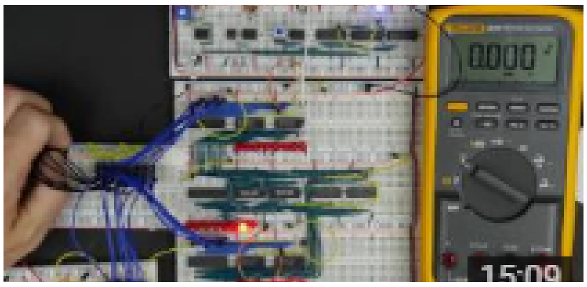
ALU design and construction



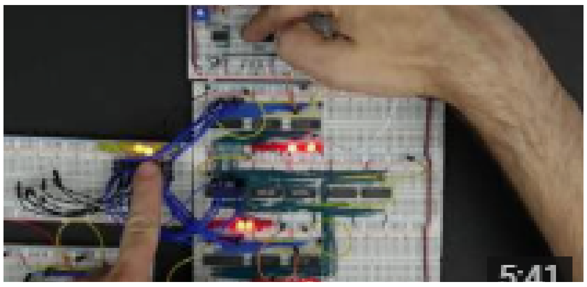
ALU Design



Building the ALU



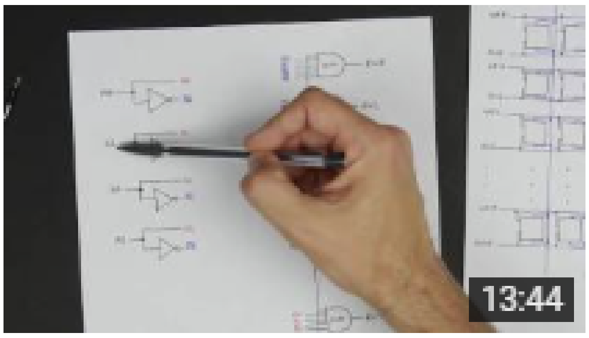
Troubleshooting the ALU



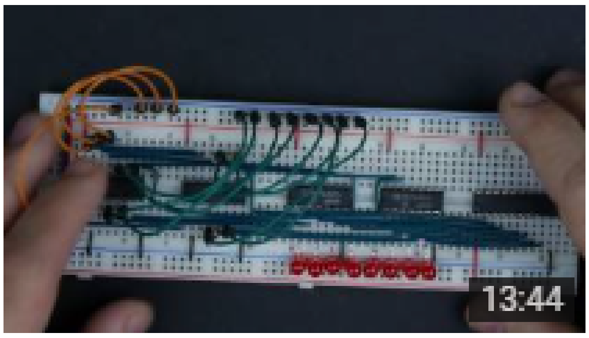
Testing the computer's ALU

Random access memory (RAM) module

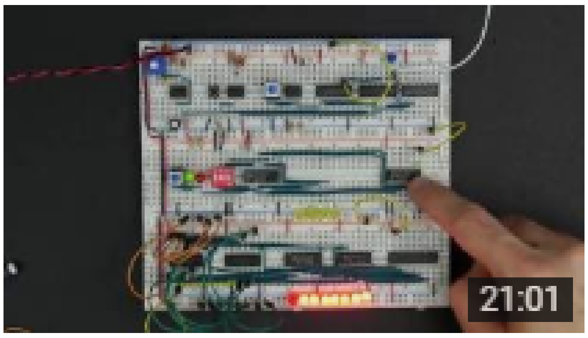
The random access memory (RAM) stores the program the computer is executing as well as any data the program needs. Our breadboard computer uses 4-bit addresses which means it will only have 16 bytes of RAM, limiting the size and complexity of programs it can run. This is by far its biggest limitation.



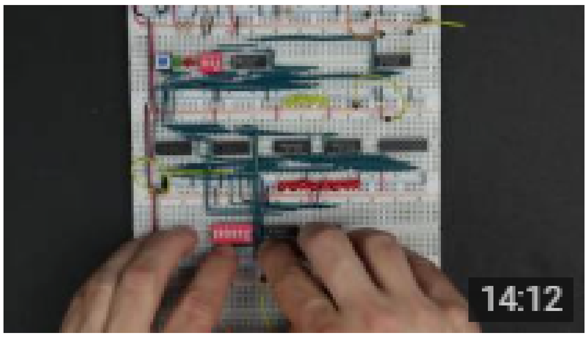
8-bit computer RAM intro



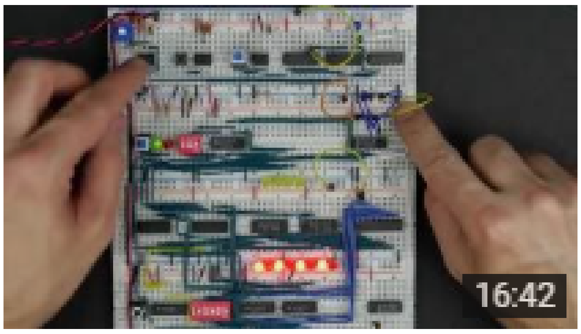
RAM module build - part 1



RAM module build - part 2



RAM module build - part 3



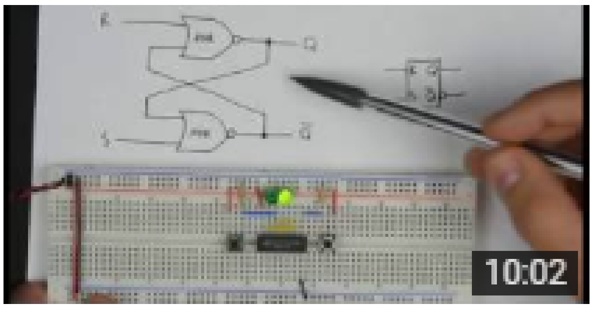
RAM module testing and troubleshooting

Program counter

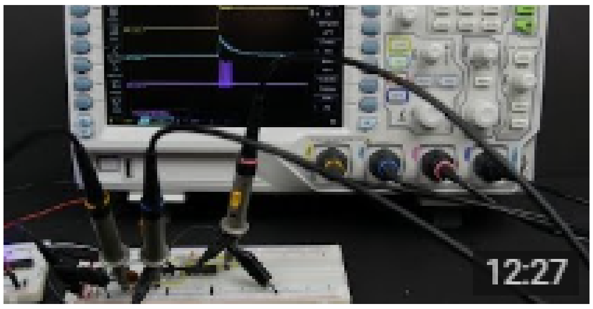
The program counter (PC) counts in binary to keep track of which instruction the computer is currently executing.

Theory

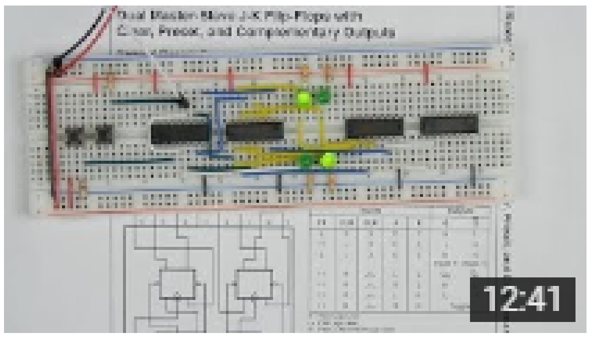
This first set of videos explain how counting in binary works:



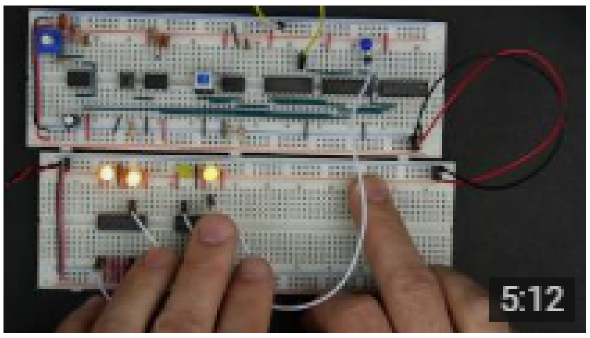
[JK flip-flop]



[JK flip-flop racing]

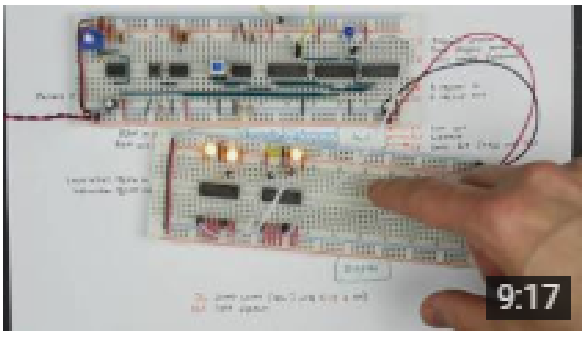


[Master-slave JK flip-flop]

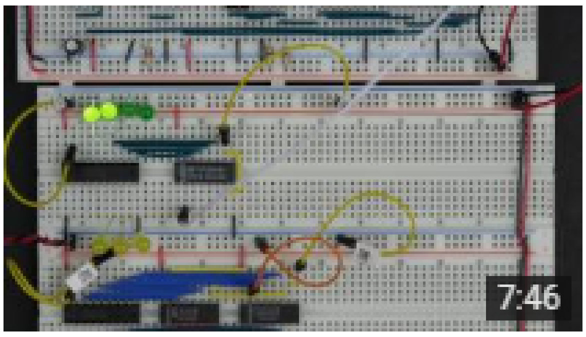


[Binary counter]

Program counter design and build



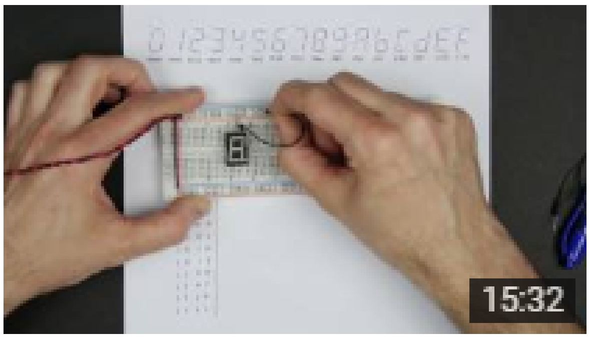
[Program counter design]



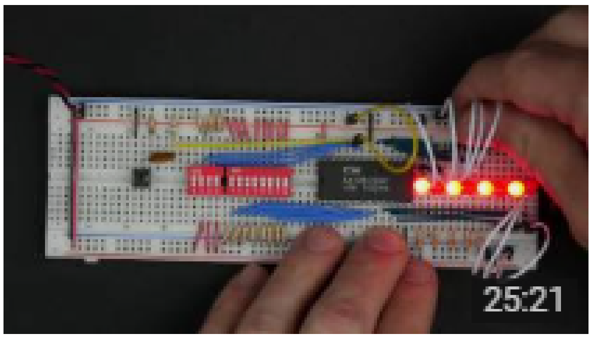
[Program counter build]

Output register

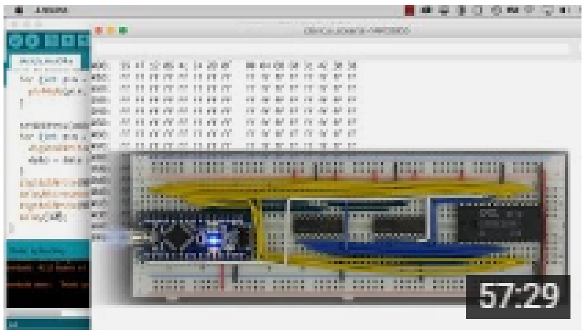
The output register is similar to any other register (like the A and B registers) except rather than displaying its contents in binary on 8 LEDs, it displays its contents in decimal on a 7-segment display. Doing that requires some complex logic; luckily there's an easier way:



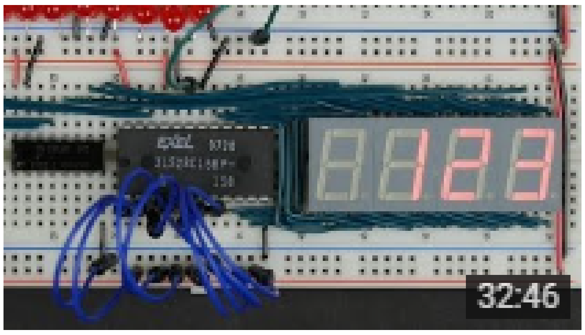
Designing a 7-segment hex decoder



Using an EEPROM to replace combinational logic



Build an Arduino EEPROM programmer

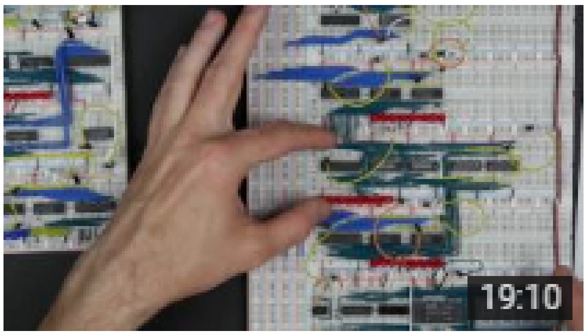


Build an 8-bit decimal display for our 8-bit computer

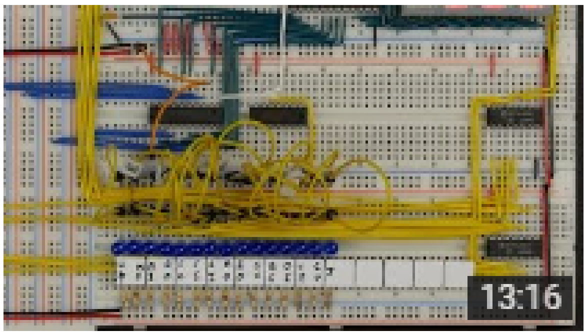
Bringing it all together

Before building the control logic, we want to connect all of the modules to a shared bus and test things out. The modularity of the design makes it easier to test each module by itself so we won't ever get to a point where we put it all together and nothing works.

Videos



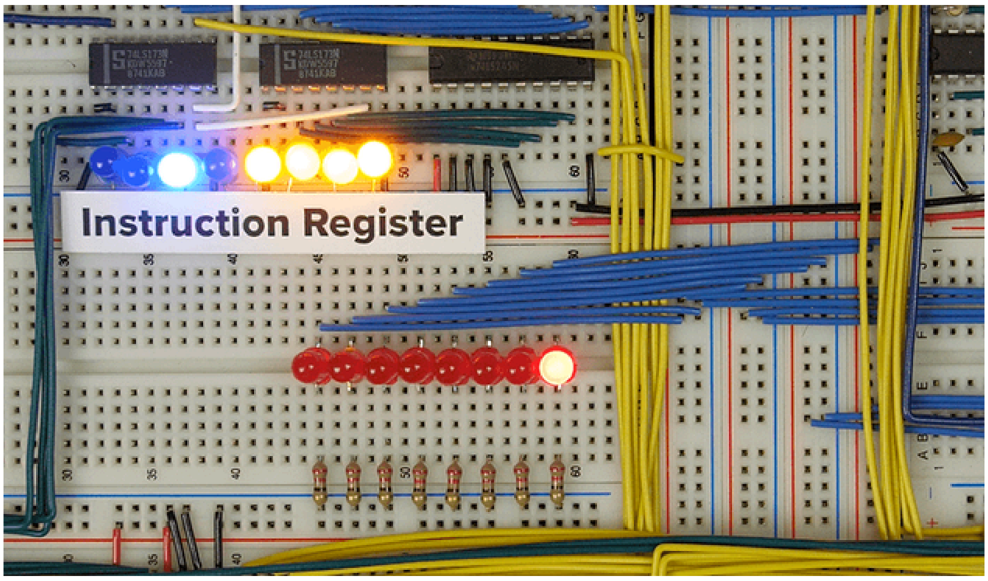
8-bit computer build: Connecting the bus



8-bit CPU control signal overview

Notes

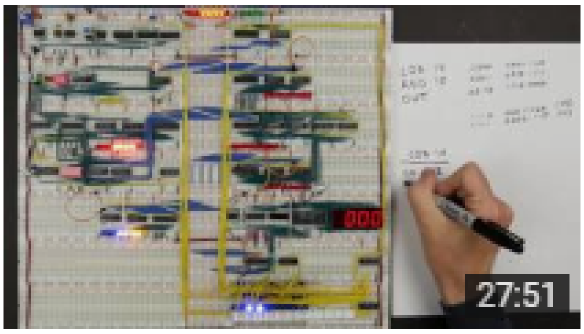
Rather than soldering the LEDs for the bus together as shown in the videos, you can put them in an open space on the breadboard. There will be ample free space on the right side of the breadboard just below the instruction register. Here's what that looks like:



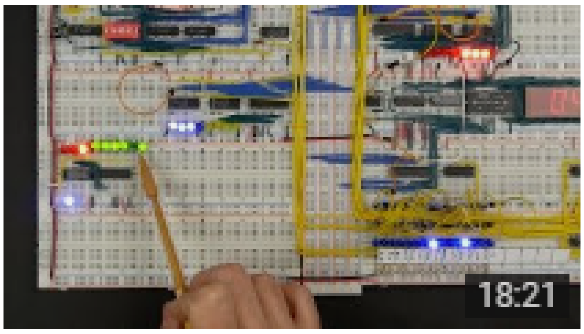
CPU control logic

The control logic is the heart of the CPU. It’s what defines the opcodes the processor recognizes and what happens when it executes each instruction.

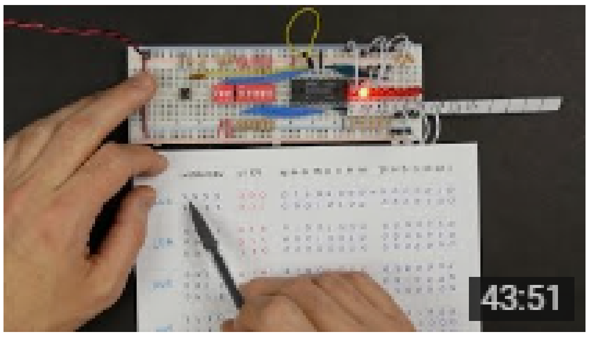
Videos



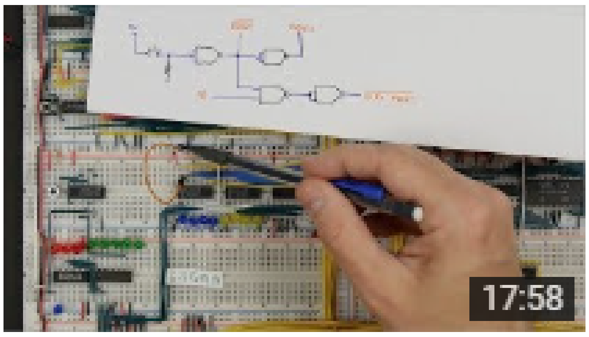
[8-bit CPU control logic: Part 1]



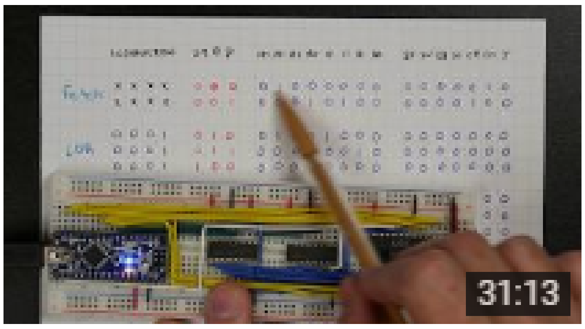
[8-bit CPU control logic: Part 2]



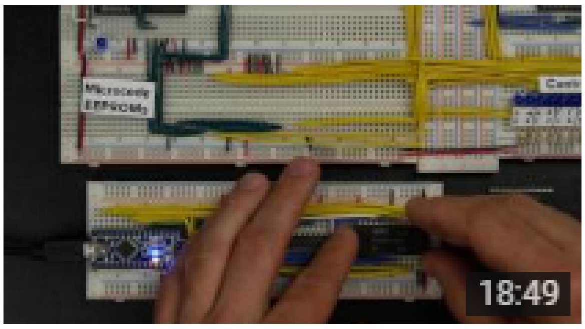
[8-bit CPU control logic: Part 3]



[8-bit CPU reset circuit and power supply tips]

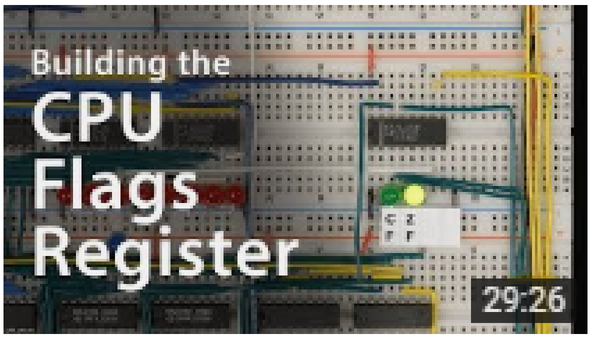


[Reprogramming CPU microcode with an Arduino]



[Adding more machine language instructions to the CPU]

Making a computer Turing complete



CPU flags register

Conditional jump instructions

Notes

An alternative to soldering a USB cable for power is to use a DC wall plug with a screw terminal adapter (If you bought Kit #1 from me, you already have these). To create a more robust connection as described in the video above about power supply tips, you can use multiple wires like this: