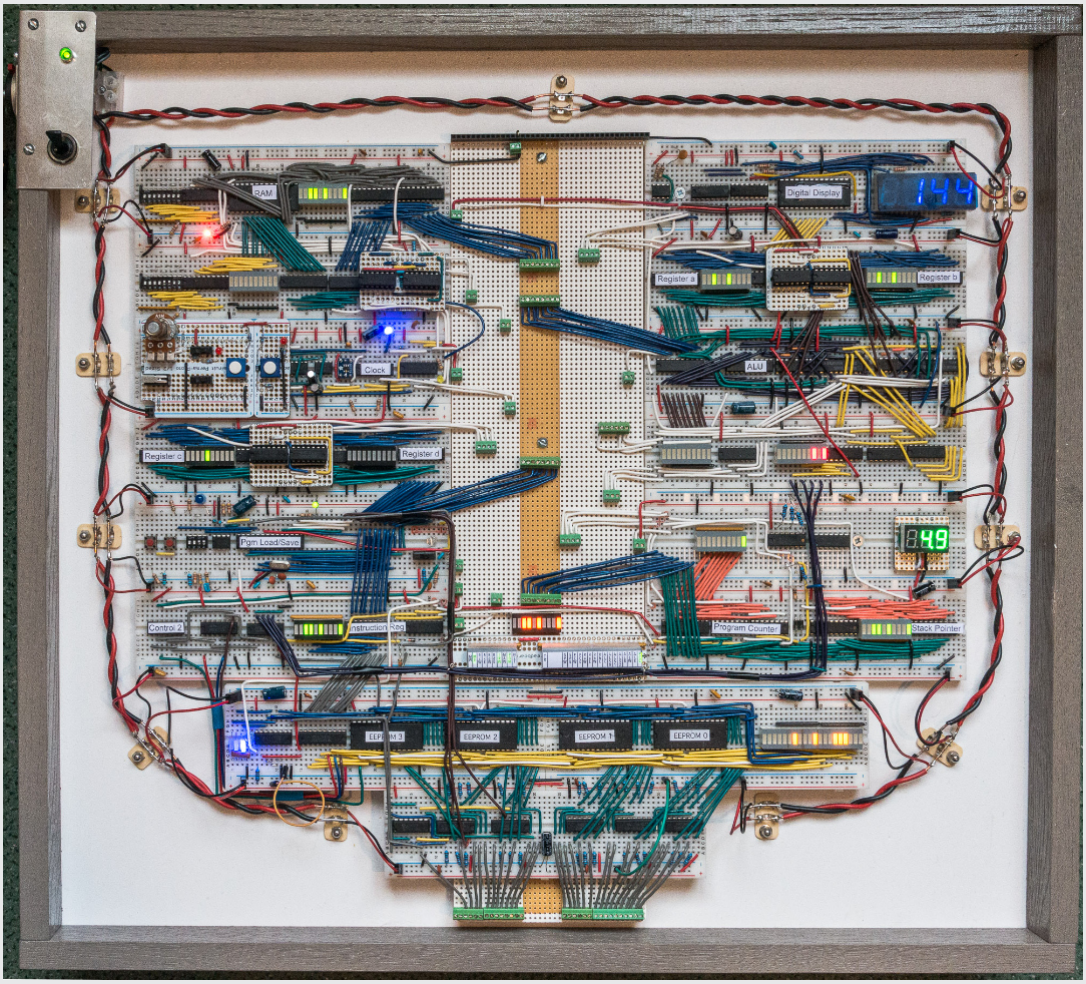
8-Bit Breadboard Computer

My Build

Work in progress



August 2020

Table of Content

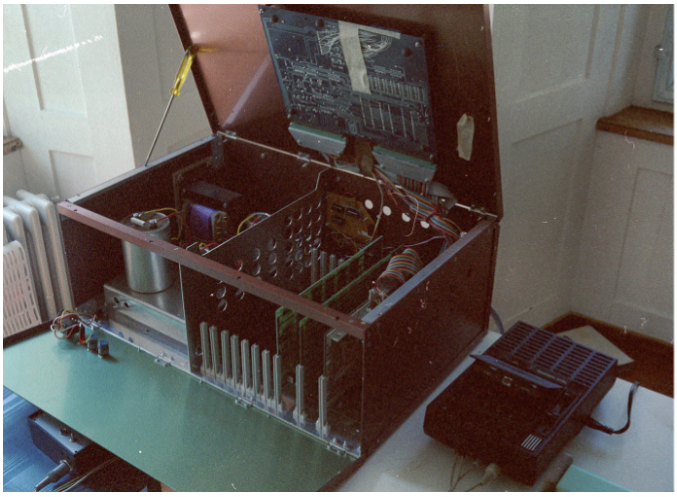
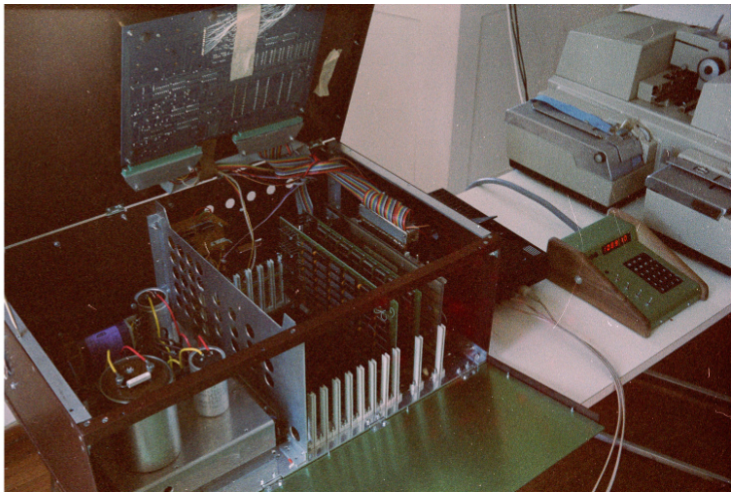
Baseline

I discovered Ben Eater's now almost famous videos on YouTube in the closing months of 2019. These videos have inspired many hobbyists to build their own versions of a SAP computer. SAP stands for **Simple as possible**, as described by **Albert Paul Malvino** in his book „Digital Computer Electronics" (c) 1977.
For myself, I decided to follow closely James Bates videos, whose computer has 8-bit addresses instead of 4-bit addresses used by Ben Eater. Thanks to both for their work and excellent didactic skills in explaining their respective builds.

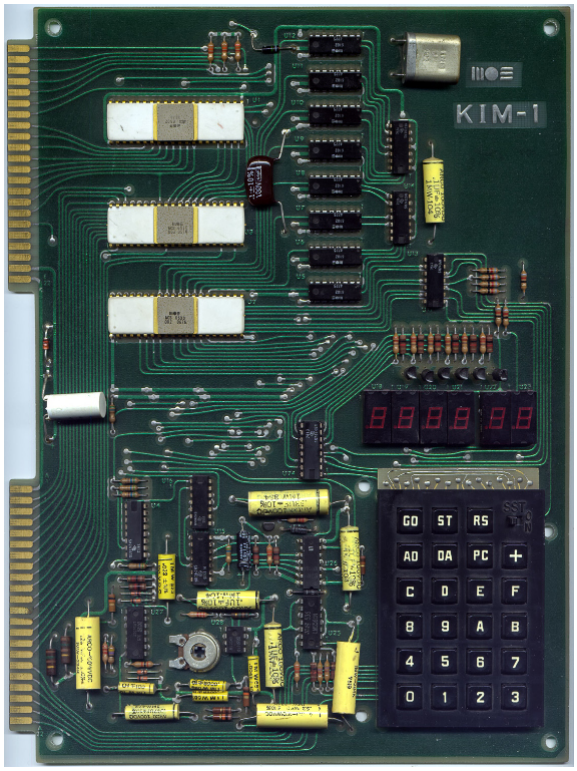
Ben Eater on YouTube
James Bates on YouTube
James Bates on GitHub

1. Looking back to 1977

About 45 years ago, between 1973 and 1978 I was an avid reader of magazines like **Kilobaud** and **Byte**. This was the time when the Altair 8800 came out (1975) and hobbyist in USA and Germany were building their own home computers. For me, the **KIM-1 single board computer** was very attractive and I expanded it with an S-100 bus adapter board and a 12 kB memory board. I added a beefy power supply and a big aluminum case. A simple Philips cassette recorder was used to save and load programs. I also owned a heavy teletype machine with paper tape reader. Well, these were the days well before IBM put the first PC on the market (1981). The famous Apple 2 reached the market also in 1977. Unfortunately, my machine did not make into the twenty first century.

My DIY home computer in 1977

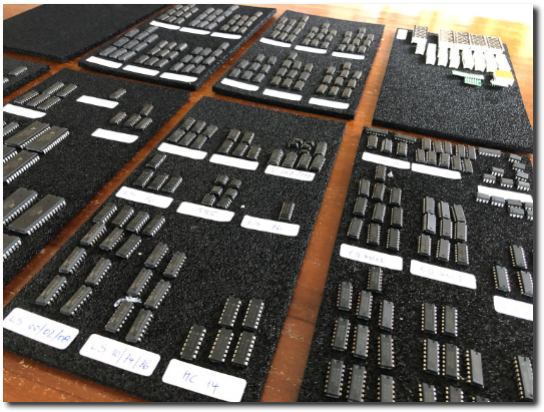


The famous KIM-1 single board computer with 6502 processor, 1kB of ram.

KIM-1 on Wikipedia.

2. Start of Project

After carefully studying James Bates videos and schematics (on GitHub) I started ordering components from different sources, mainly Mouser.ch.
I also followed Ben Eater's advice concerning breadboards: do not use cheap ones, spend a little more and spare yourself trouble along the way. I assembled a large collection of chips, mosty 74HCTxxx.
This is what my collection looked like:

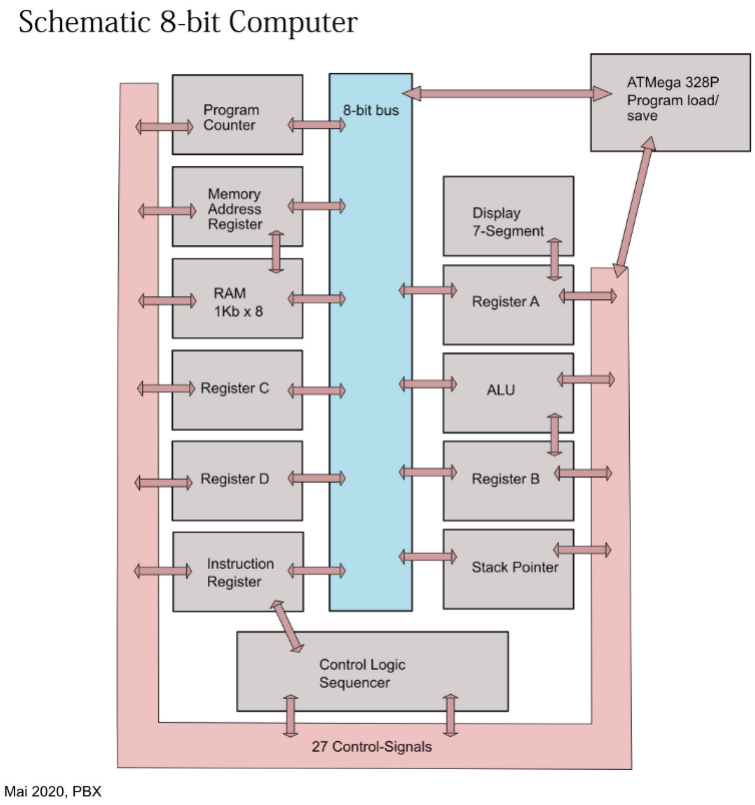


Collection of chips for the DIY computer

3. Architecture and Components

The computer has these components:

- 4 8-Bit general purpose registers
- 256 Bytes of RAM
- 8-Bit Program Counter
- 8-Bit Stack Pointer
- ALU based on the chip 74LS382
- Control Unit with 4 EEProms AS6C6264 generating 27 control signals.
- Decimal display (7-Segmant display) connected to register a
- Unit to load and save programs to/from EEPROM 24c256. This unit is built around an ATMega328p (sort of like a bare bones Arduino).



Components of my 8-bit Computer

Importance of the Control Logic

Every computer consists of such components - these components are like musicians in an orchestra: without an able conductor they are not going to produce a coherent piece of music.
The Control Logic (control unit) is this conductor.
It tells the components what do to, or, in other words: at what time they should do what they are designed to do. The drummer in an orchestra only hits his drums when the conductor gives him a signal. Likewise for all other players.

In this and other computers, the control logic asserts control lines (high or low), every component has two ore more control lines and they perform a function according to the signals the read.
The computer clock is like a metronome: it synchronizes all activites. On the falling end of the clock edge a new set of control lines are asserted and on the next rising clock edge the components do their thing. In ths way the computer marches through a program (see chapter clock).
This computer has 27 control lines. See list in the attatchment.

4. Thoughts on how to do it

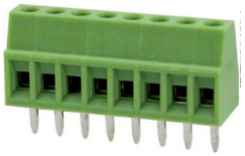
Ben Eaters computer has only 4-bit adresses and therefore only 16 bytes of ram are possible. In my build I follow James Bates who has also very interesting videos showing his design in great detail. He chose to have 8 bit addresses and also he distinguishes between **program memory** and **data memory**. Fortunately James Bates published his design with complete schematics on GitHub.That set me going with a good start.
James Bates on GitHub.

For my build I set theses goals:

- only use led bars for 8-bit displays and use single leds only sporadically
- use a perfboard (stripboard) for bus lines and the 27 control lines
- generally use more breadboards for easy wiring
- use HCT chips, as James Bates does (this allows mixing them with LS chips)
- build registers using 74HCT574 and 74HCT 245 chips - they have input/output clean on on side of the chips - this avoids crossover wires.
- heavy gauge wires for power distribution bringing 5 volts to the breadboards.

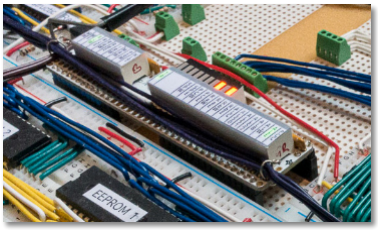
4.1 Basic Layout

This is the physical layout that I defined early on. The 39 line stripboard carries 27 control signals, 8 bus signals, 5 Volt, ground, clock and reset. Left and right side is spray painted white.
Degson terminal blocks in various length are soldered onto the stripboard
I also opted for a less dense wiring by allocating more breadboard space for the whole layout. Power distribution was also an issue that I pondered for some time. A good solution was found after some time..

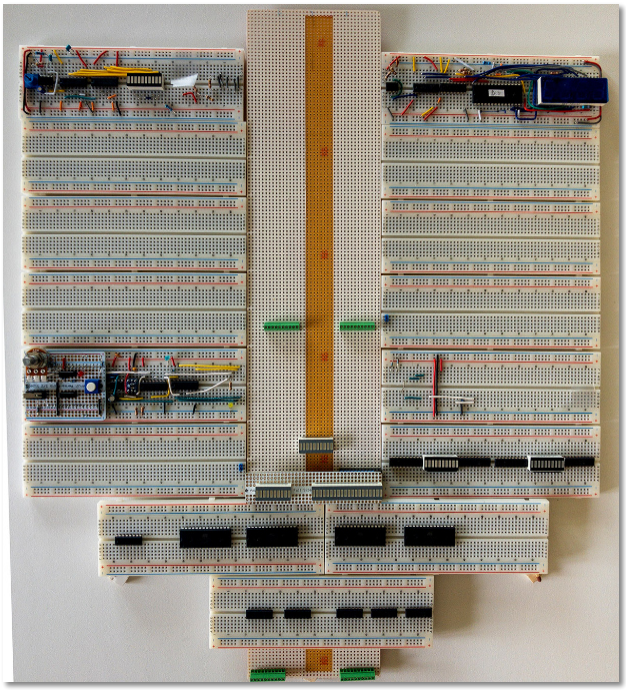


Degson Terminal Blocks

The Control Unit is placed on the lower end of the stripboard which runs below the EEProm breadboards. A bridge carrying led bars is used to display the control signals.



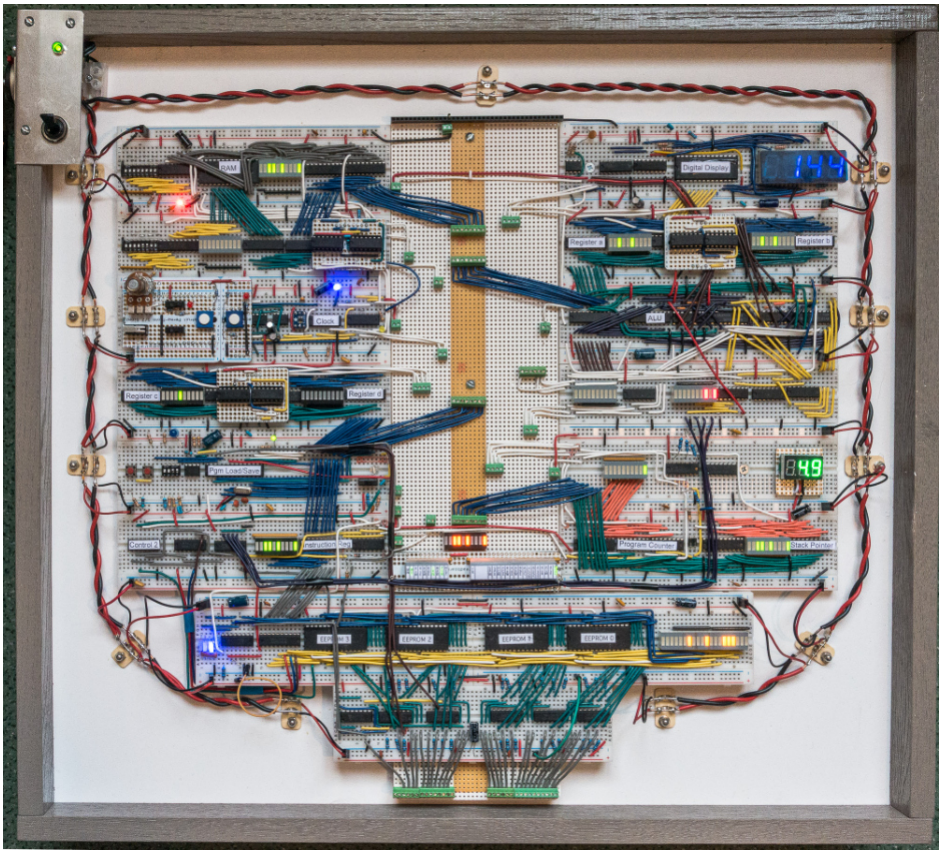
Bridge with led bars



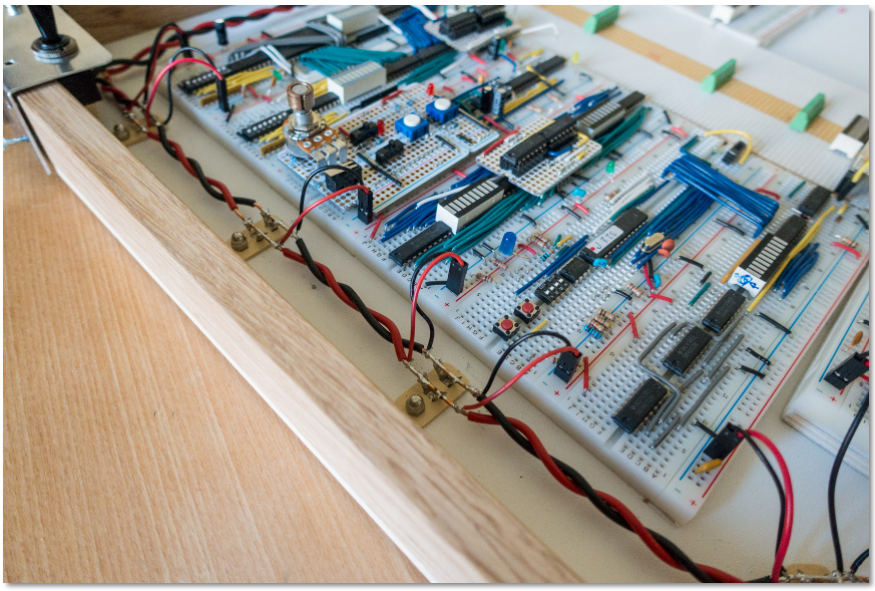
Clock, Decimal display already done at this point

5. The final computer

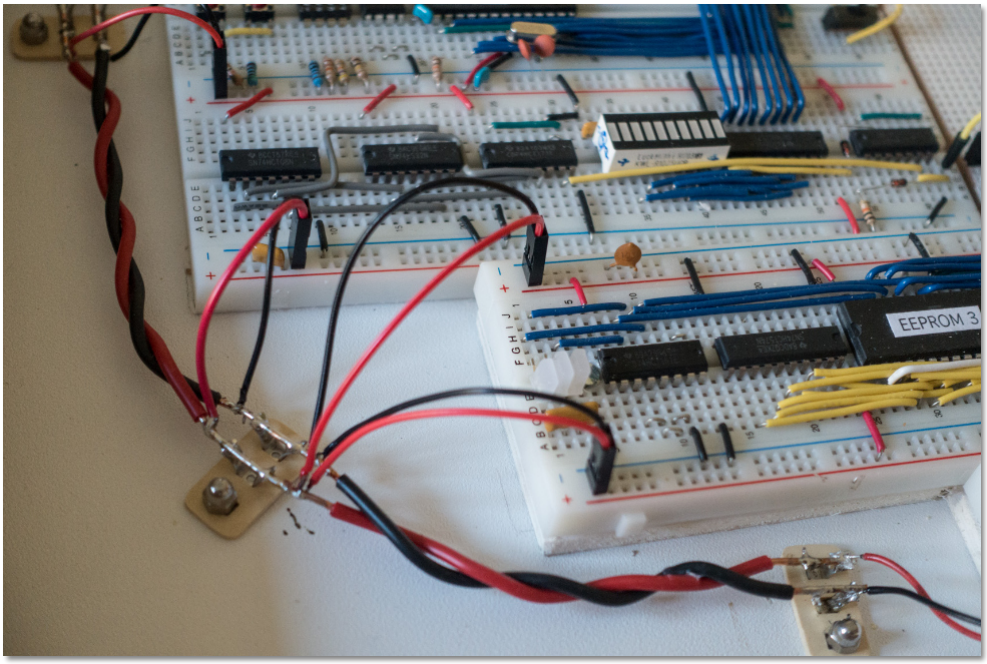
Here is how the machine looks:



Finished...

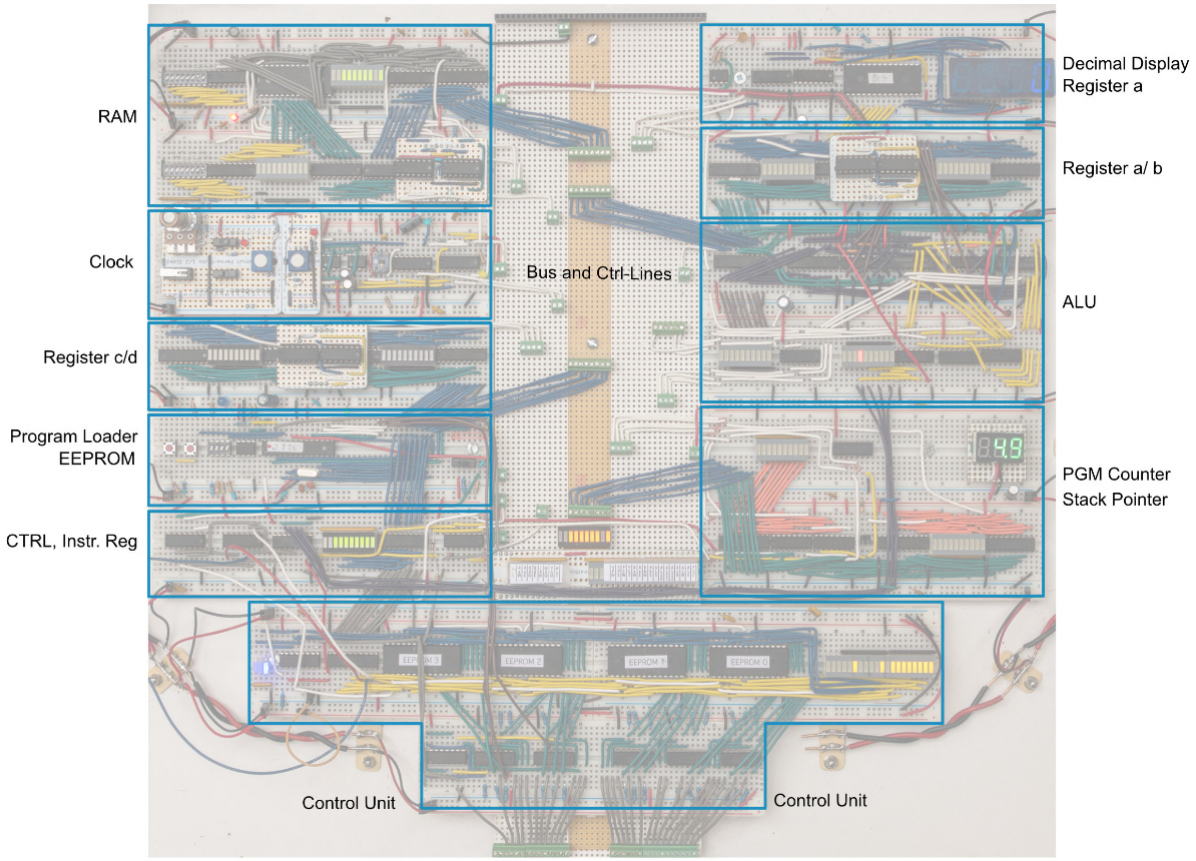


Side view showing power distribution



Details of power connections

This is a component overview



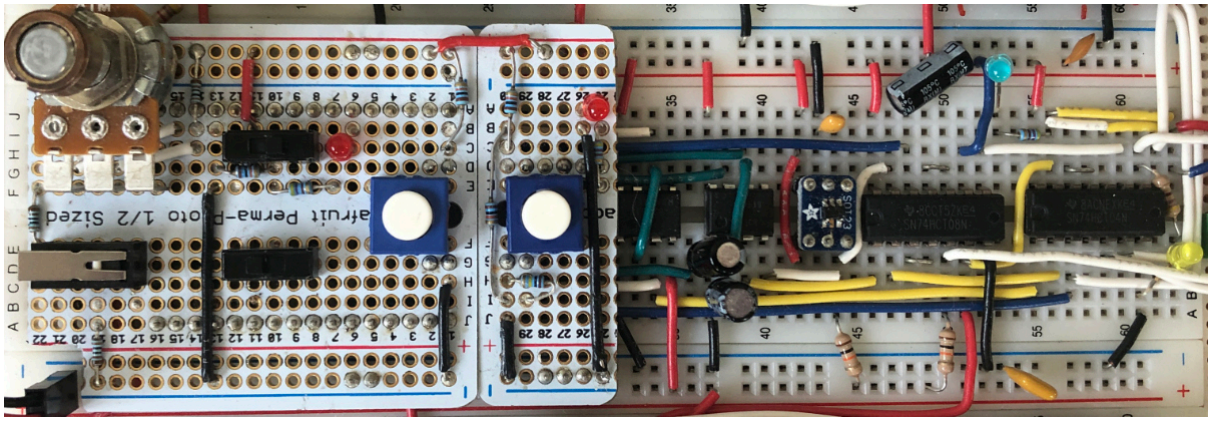
6. Components

6.1 Clock Module

I, as everbody does, started with the clock module. This was not done in Ben Eater's style, but rather using a simpler circuit around the 555 chip I also wanted a power-on reset circuit and switches and pushbuttons that were somehow soldered on rather than simply stuck into a breadboard. I decided to build a 'bridge' that is put into the breadboard. I used an Adafruit Proto Board to do this. The power-on reset circuit (the small breakout board with the star) is using a small SMD chip LP3470 Voltage Supervisor.
The clock module has these controls:

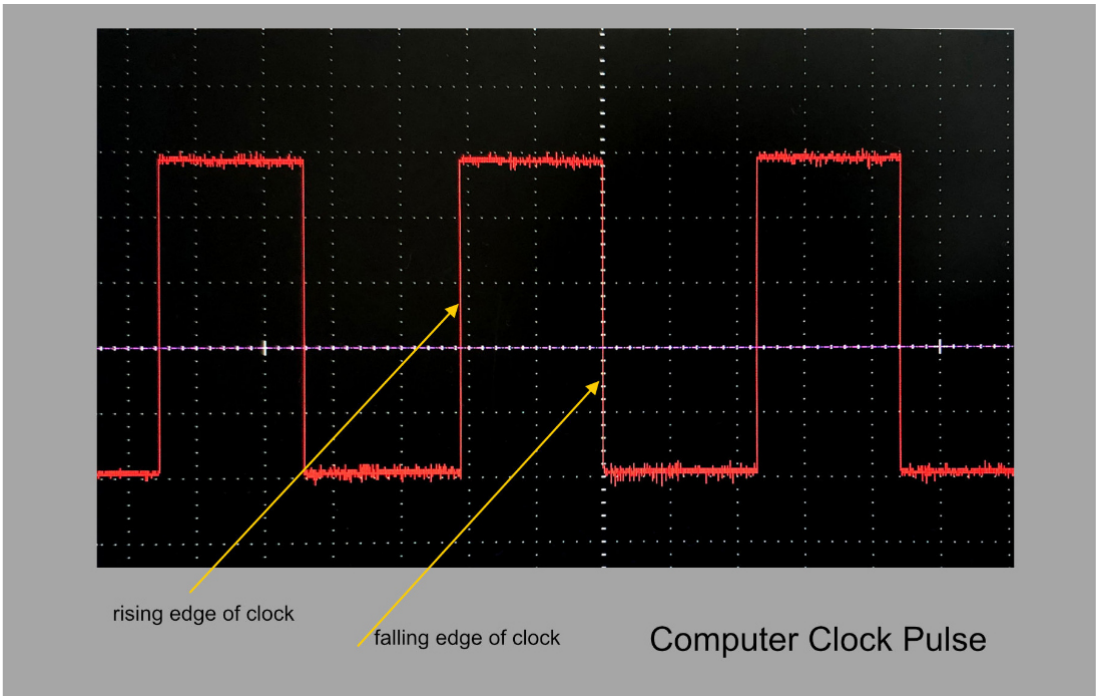
- potentiometer for clock frequency
- slide switch for programming the ram using dip switches
- slide switch for manual/automatic clock
- pushbutton for computer reset
- pushbotton (toggle) to quickly inhibit clock
- pushbutton (microswitch) single step the clock

This is what the final clock unit looks like:



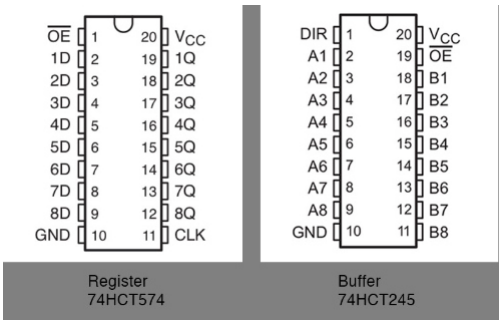
Clock module with 'Bridge' on the left.

The clock pulse is shown here. At the rising edge of the clock the next microinstruction is executed and on the falling edge of the clock the control logic changes the control signals in preparation of the next microinstruction.



6.2 General Purpose Registers

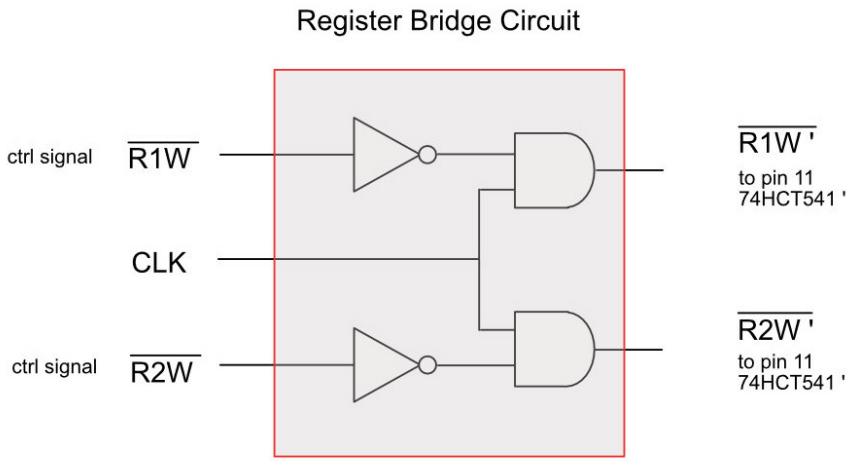
Two general purpose registers (GPR) are placed on one breadboard. I choose to use chips that have their input/output pins on one side for cleaner wiring. For the actual register I use the 74HCT574 chip and for the output buffer (to bus) I use the 74HCT245 chips.



Two chips for one GPR

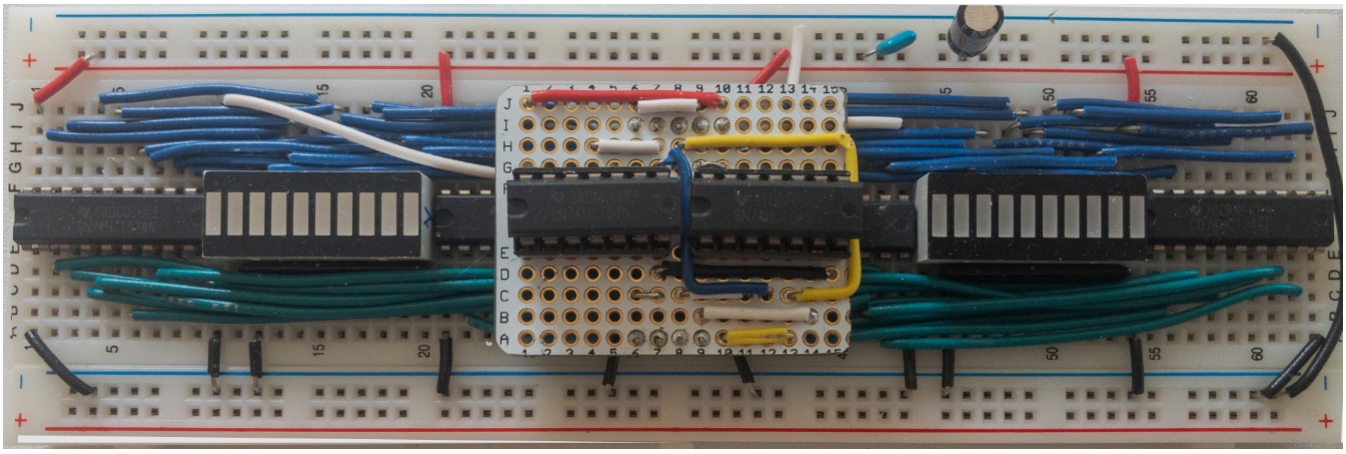
Using the 574 chip has a small drawback since it has no input enable pin. The existing enable pin (pin 1)is for the output lines only (3-state logic). Because of that I built a small circuit serving the two registers on a breadbaord. This circuit is mounted on what I call a 'register bridge' - there was simply not enough space on the breadboard to fit 2 additional chips (INV and AND).

This circuit and the RxW signal (register write) with the clock is shown in this picture. Works just fine.



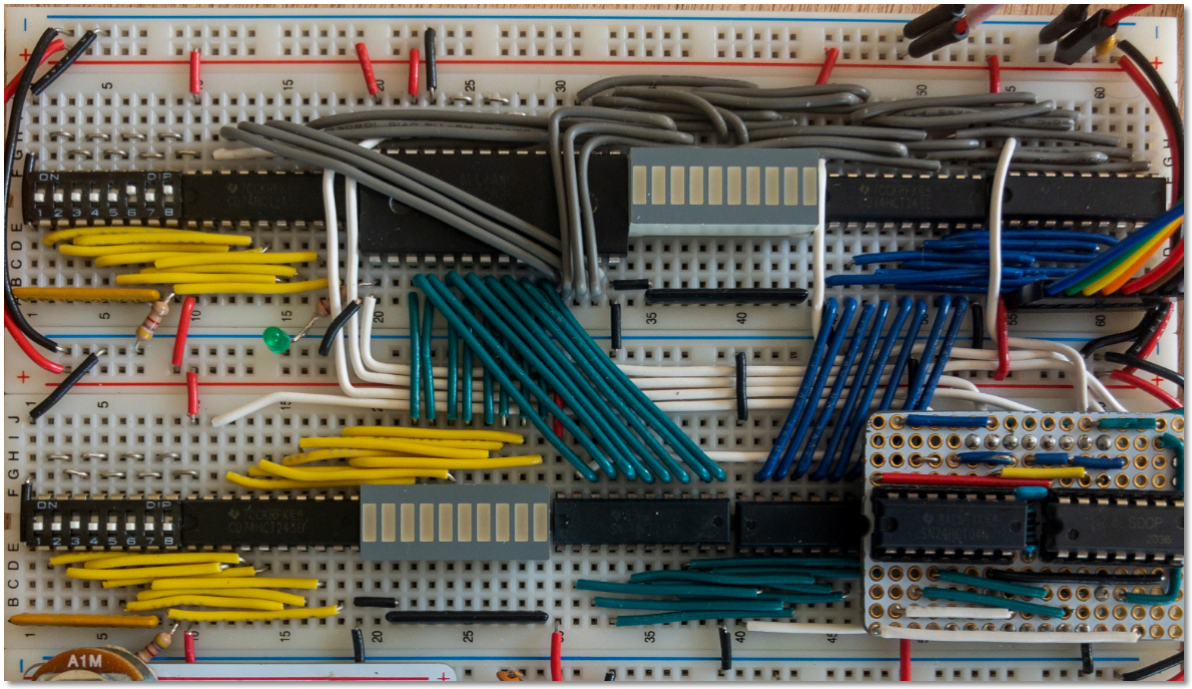
Register circuit

A breadboard with two GPR's look like this (register bridge in the middle of the picture):



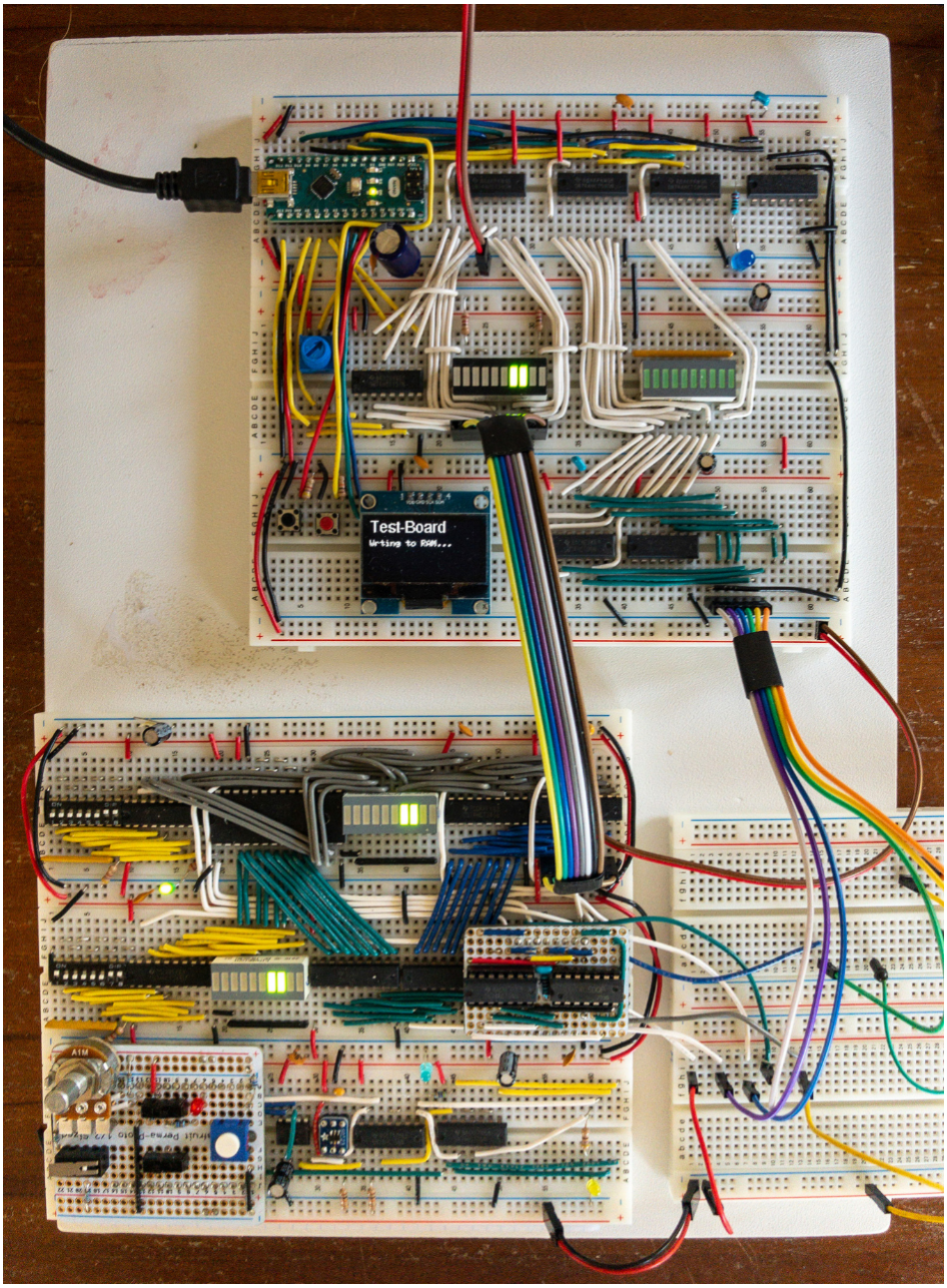
6.3 RAM Module

The Ram module is built on two breadboards. It consists of the actual SRAM chip AS6C6264 the memory address register and various other components. The SRAM chip has 12 address lines and line 12 is used to address the program memory (fetching instructions). Two dip switches (address/data) allow for entering data into ram. Upper led bar shows memory content, lowerled bar shows address.



RAM Unit

The Ram unit was unit tested using the test setup prior to the assemby of the wohle computer. (see chapter Test Circuit). This shows the ram module under test:



Testing the ram module: test circuit above, ram module (and clock) below

6.4 Control Logic

The control logic is to the computer what a conductor is to an orchestra: it tells the components what to do. It does this by asserting various control signals. This computer has 27 control signals (see list in attachment). The control logic has a counter (74HCT161) called the **microtimer**, this counter is incremented at the falling edge of the clock pulse. The 3-bit output of the microtimer is input to the 4 EEPROMS.

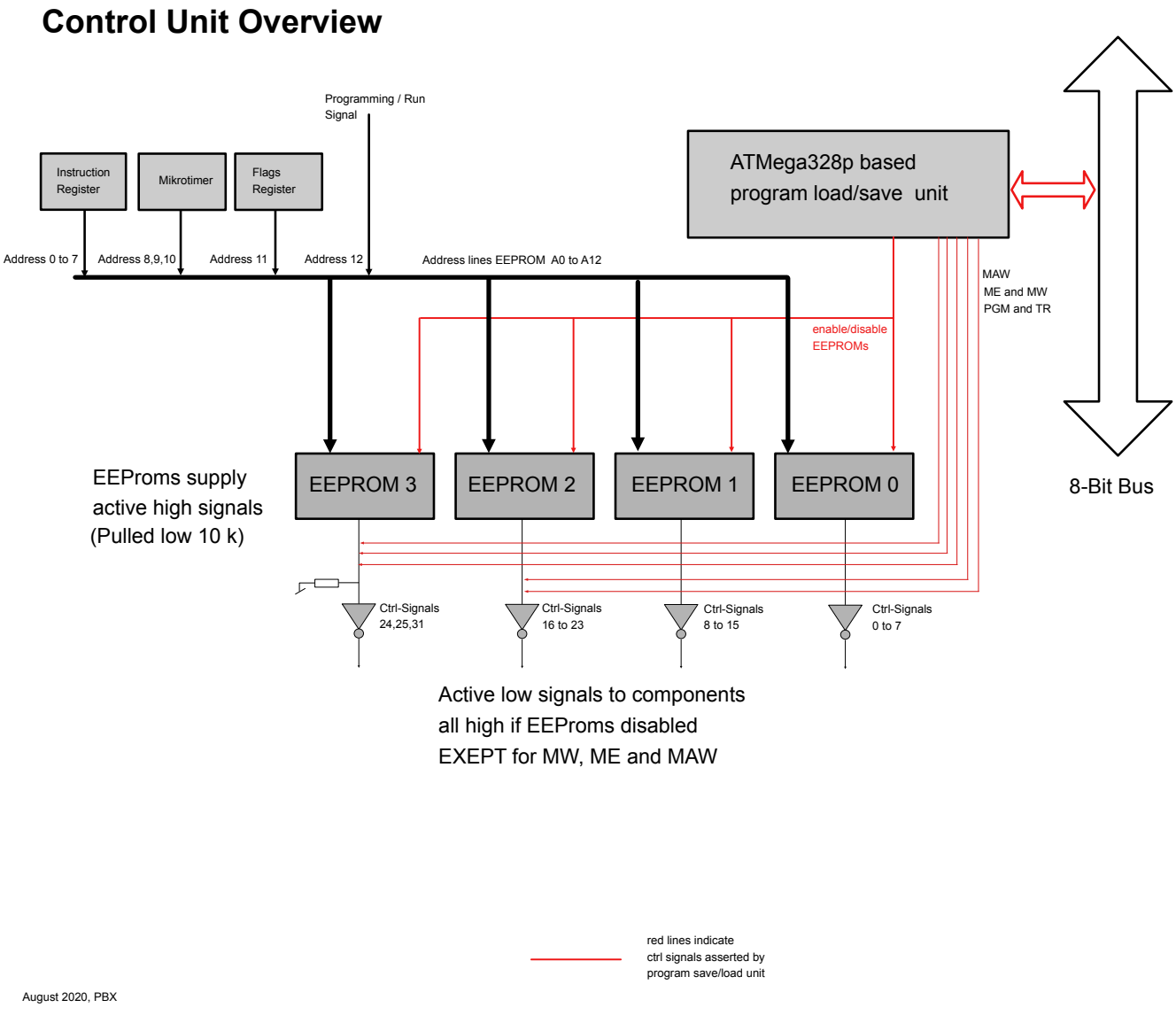
The 12 address (input) lines to the 4 EEPROMs are:

- Bit 0 to 7 from the instruction register
- Bit 8,9 and 10 from the microtimer

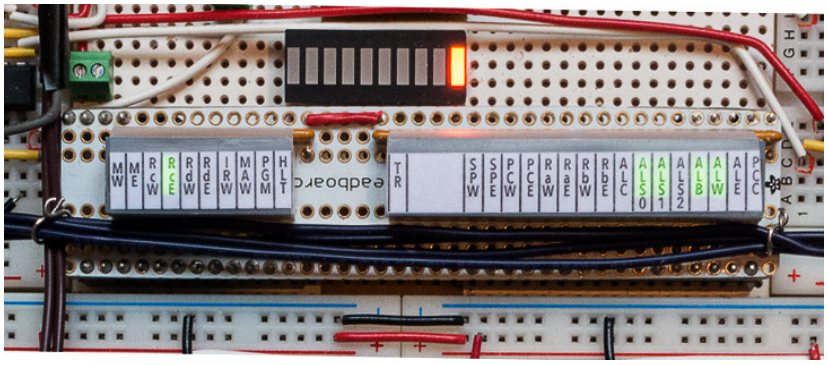
- Bit 11 from the flags logic
- Bit 12 from control line PGM

The EEPROMs can be **enabled** or **disabled** (3-state output) - for **normal run** EEPROMs are **enabled**. When the program load/save unit takes control of the computer the EEPROMs are **disabled**. **All** output lines from the EEPROMs are pulled low by 10 k resistors. All 27 control-signals are inverted, so we have active low signals fed to the components.

This is an overview of the control logic:

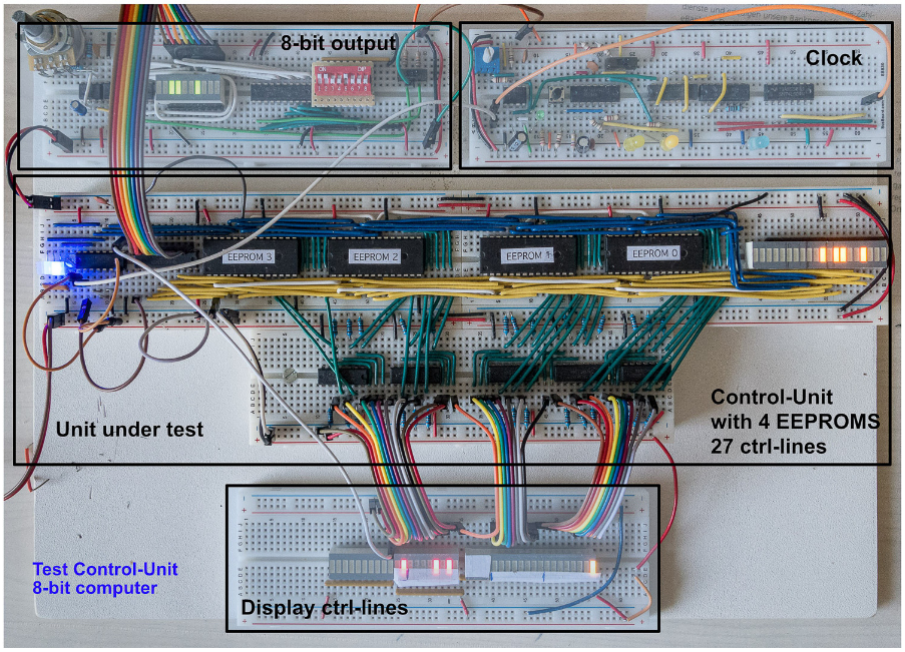


Control Logic Overview



View of display of the 27 control lines (green) and bus (orange)

To ensure that the control logic is working properly - asserting the correct control signals for all instructions, it was tested prior to the assembly of the whole computer. This is the test setup:



Testing the control logic

6.5 ALU Unit

The ALU unit uses two SN74LS382 chips to perform arithmetic and logic functions. One operand is always taken from register Rb. These are the ALU instructions (from James Bates):

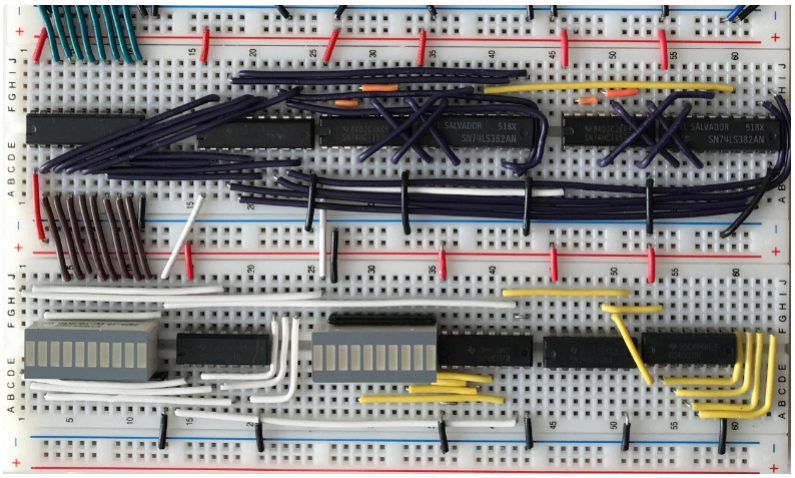
- ADD Rx,Rb
- ADC Rx,Rb
- SUB Rx,Rb
- SBC Rx,Rb
- CMP Rx,Rb
- AND Rx,Rb
- OR Rx,Rb

- XOR Rx,Rb
- NOT RX
- INC Rx
- DEC Rx
- PUSH Rx
- POP Rx

An operation in the ALU can result in one or more of these flags (stored in the flag register:

- Carry flag
- Zero flag
- Negative flag
- Overflow flag

This is the ALU in an early stage of construction:



Led bar left: ALU result register - Led bar right: flags register

6.6 5 Volt power

5 Volt is distributed through thick twisted copper wires to all breadbords - see pictures. The clock board contains a **power-on-reset** circuit using a LP3470 Voltage Supervisor chip to generate the power-on-reset signal. This signal is combined (OR gate) with the manual reset.

While executing the Fibonacci program the computer draws about 127 mA from the external 5 Volt power supply.

7. Program Load/Save

Entering machine code into ram with dip switches is tediuos at best - just as it was on the Altair 8800 45 years ago..
I came accros a clever circuit persented on hackster.io by David Hansel.
David Hansel on hackster-io

David uses a ATMega328p microprocessor with a small EEPROM attached to save a program sitting in ram into one of 14 slots in the EEPROM. Likewise the circuit reads program code from the EEPROM and writes it into program memory in ram (starting at address zero.
I put this circuit onto a breadbord, adapted the sketch and bingo: it works well.: it can write a program into ram after power-on and this program will then run. In other word: I can switch power on, the circuit puts a program into ram and this program then runs. I very neat solution. Thanks David.
Loading /Saving programcode to and from ram is done by highjacking control of the computer:

- disable the EEPROM's of the control logic, their output is 3-state logic and pulldown resistors make sure that all 27 control signals are off
- The PGM and the TR signals are set (access program memory and inhibits the microtimer)
- Ram is read by setting the MAW and the ME signal
- Ram ist written to by setting the MAW and the MW signal.

To load the sketch into the ATMega328p I use a FTDI-adapter. This part is removed once the microprocessor ist functioning ok.

8. Conclusion

It was fun to build. At several stages I found that I need to bring in my oscilloscope to check on the quality of signals. Mostly, however, a Logic Probe is all that I used to trace signals from here to there. Two times I needed to reorder a set of hookup wires (Adafruit Hook-up Wire Spool Set - 22AWG Solid).

9. Attatchment

9.1 List of Control Lines

This list includes the 8 bus lines (which are NOT control lines) - it is a list off all the signal lines on the perfboard.

Control Lines

EEProm	Bit Pos	OutputPin	PinNumer	Leiter-Nr.	Signal	Signal semantic
0	7	IO/7	19	1	PCC	Programcounter count
0	6	IO/6	18	2	ALE	ALU Output Enable
0	5	IO/5	17	3	ALW	ALU write result
0	4	IO/4	16	4	ALB	ALU use register b
0	3	IO/3	15	5	ALS2	ALU S2
0	2	IO/2	13	6	ALS1	ALU S1
0	1	IO/1	12	7	ALS0	ALU S0
0	0	IO/0	11	8	ALC	ALU carry in
1	15	IO/7	19	9	RbE	Register b enable
1	14	IO/6	18	10	RbW	Register b write
1	13	IO/5	17	11	RaE	Register a enable
1	12	IO/4	16	12	RaW	Register a write
1	11	IO/3	15	13	PCE	Programcounter enable
1	10	IO/2	13	14	PCW	Programcounter write
1	9	IO/1	12	15	SPE	Stackpointer enable
1	8	IO/0	11	16	SPW	Stackpointer write
Bus0				17	Bus0	
Bus1				18	Bus1	
Bus2				19	Bus2	
Bus3				20	Bus3	
Bus4				21	Bus4	
Bus5				22	Bus5	
Bus6				23	Bus6	
Bus7				24	Bus7	
GND				25	GND	
2	23	IO/7	19	26	HLT	Halt
2	22	IO/6	18	27	PGM	Use program memory
2	21	IO/5	17	28	MAW	Memory addr. reg write
2	20	IO/4	16	29	IRW	Instruction reg write
2	19	IO/3	15	30	RdE	Register d enable
2	18	IO/2	13	31	RdW	Register d write
2	17	IO/1	12	32	RcE	Register c enable
2	16	IO/0	11	33	RcW	Register c write
3	31	IO/7	19	34	TR	Inhibit microtimer
3	25	IO/1	12	35	ME	Memory enable (ram)
3	24	IO/0	11	36	MW	Memory write (ram)
				37	CLK	clock
				38	RST	Reset
				39		

Control Lines (bus in red)

end of document