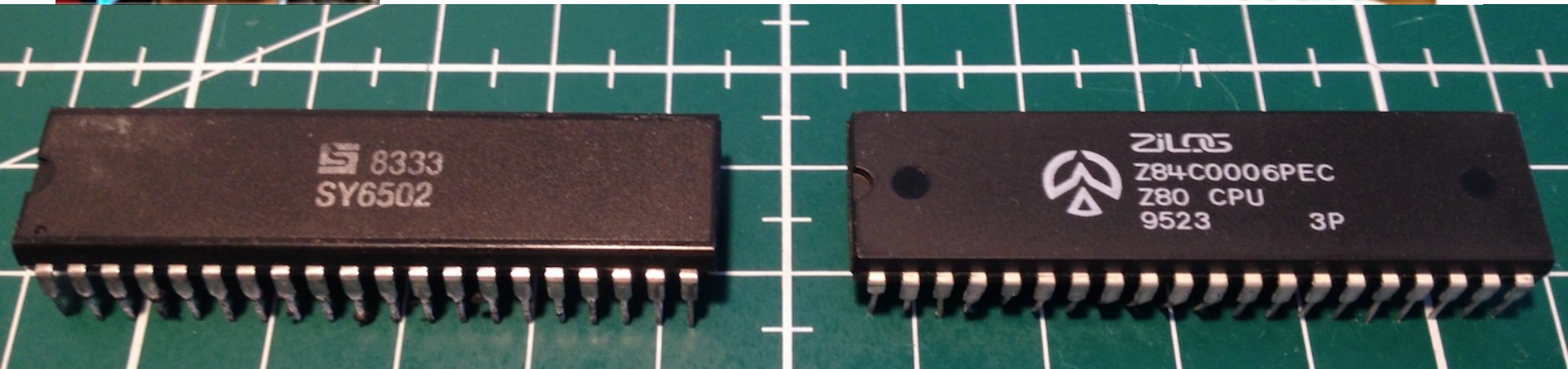
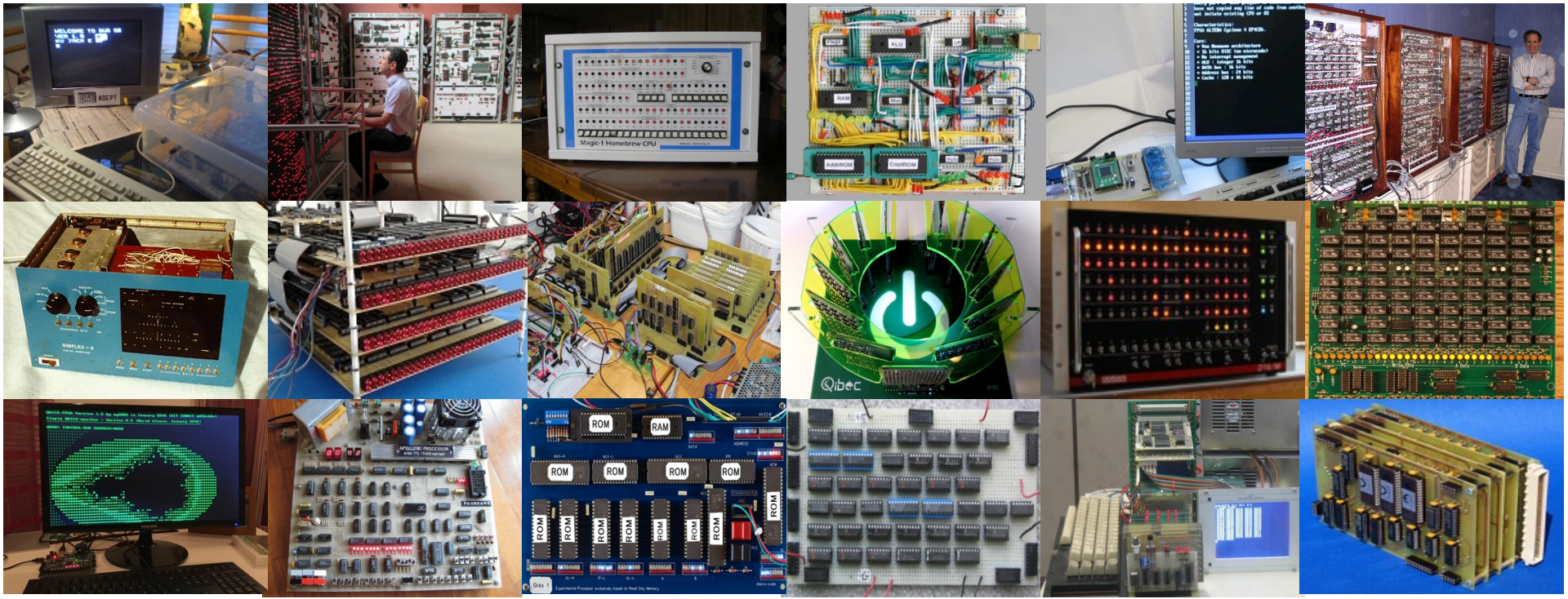
Gigatron TTL microcomputer
“Brand new vintage”

VCF Berlin 2019



Original idea: build our own CPU that can play Tic-Tac-Toe



https://www.homebrewcpuring.org

Before you begin

What core building blocks?

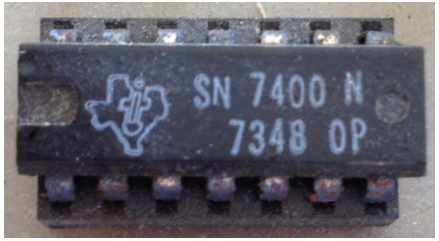
FPGA, SSI logic chips, NAND gates, discrete transistors, tubes, relays, steam punk, …

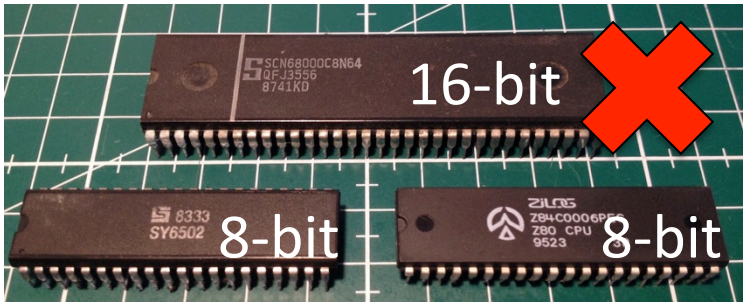
Data path size?

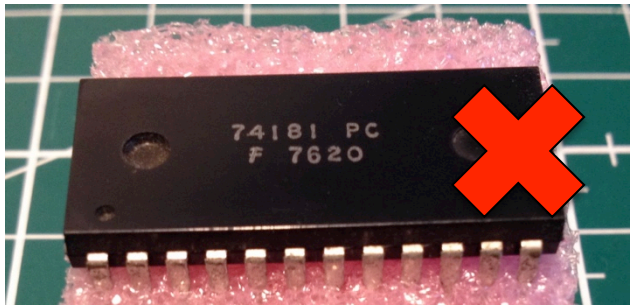
64 bits, 32 bits, 16 bits, 8 bits, 4 bits, 1 bit, other…

Standard ALU chips or custom?

74181 chips (4-bit ALU)?







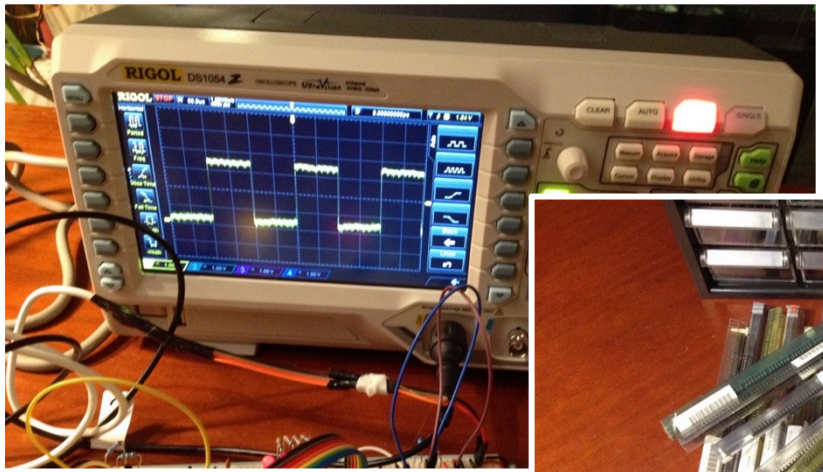
Our choices:

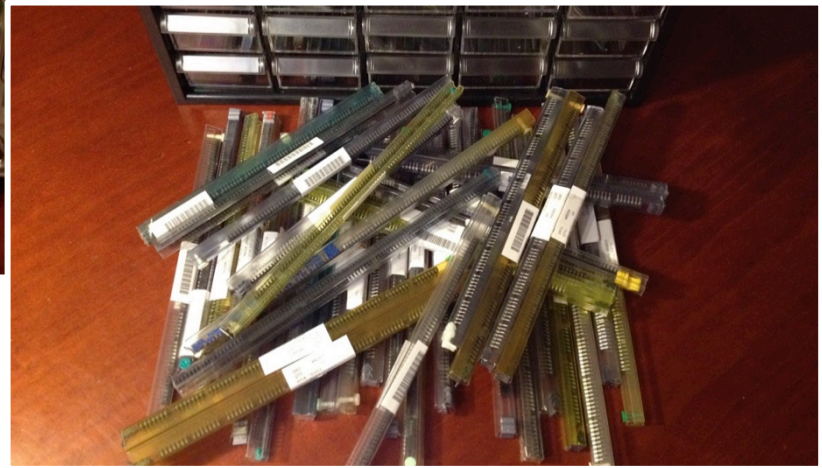
7400 series logic "TTL"

8-bit system

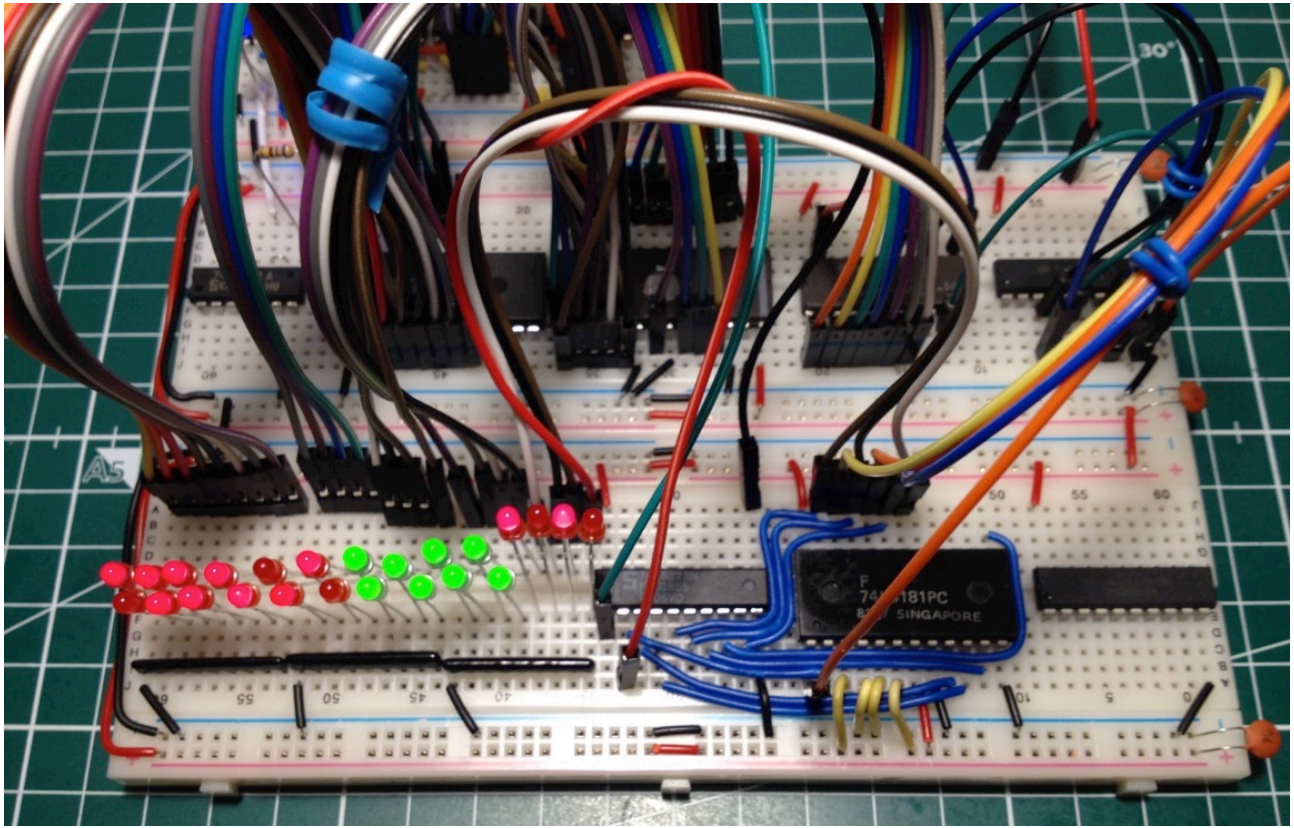
No complex chips

Buy books, tools and hundreds of 7400-series chips …

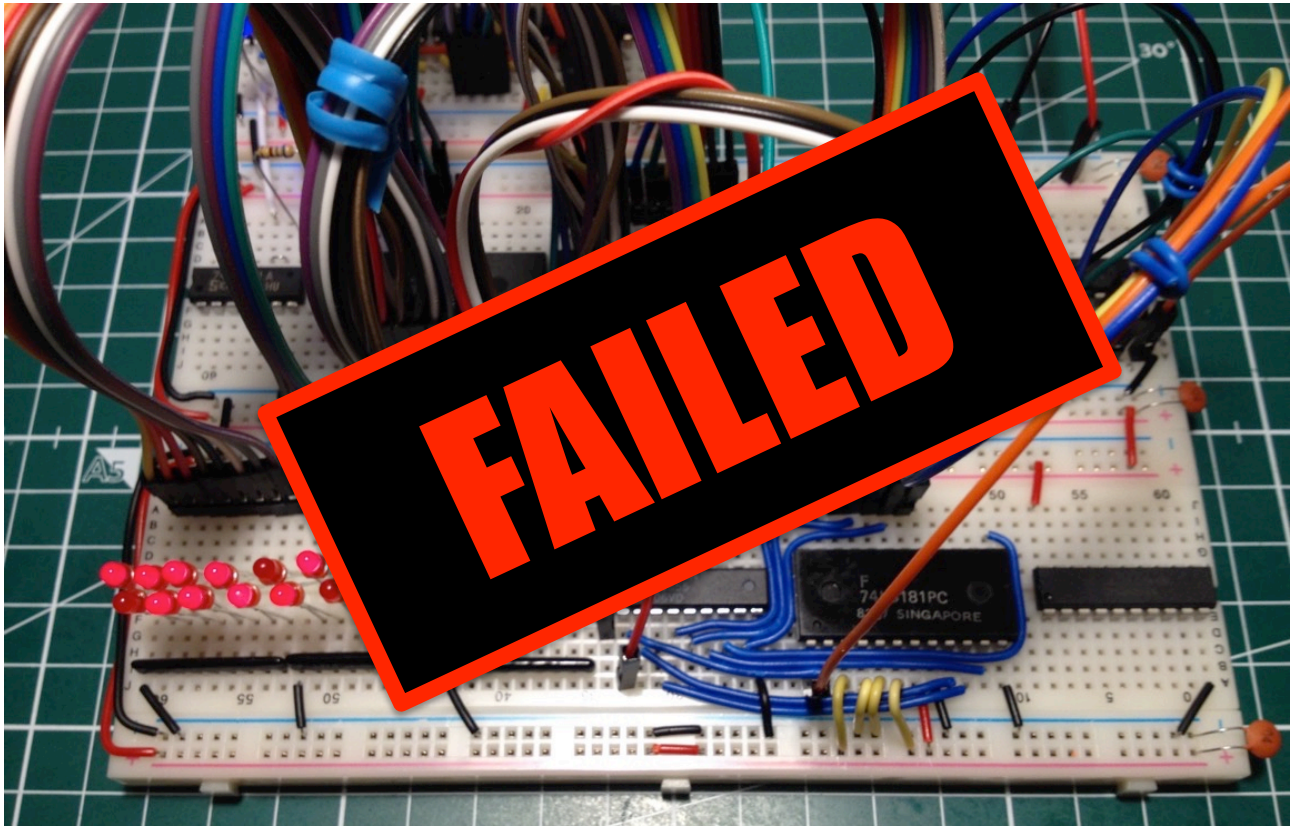




… and you can build a 4-bit computer!



… and you can build a 4-bit computer!

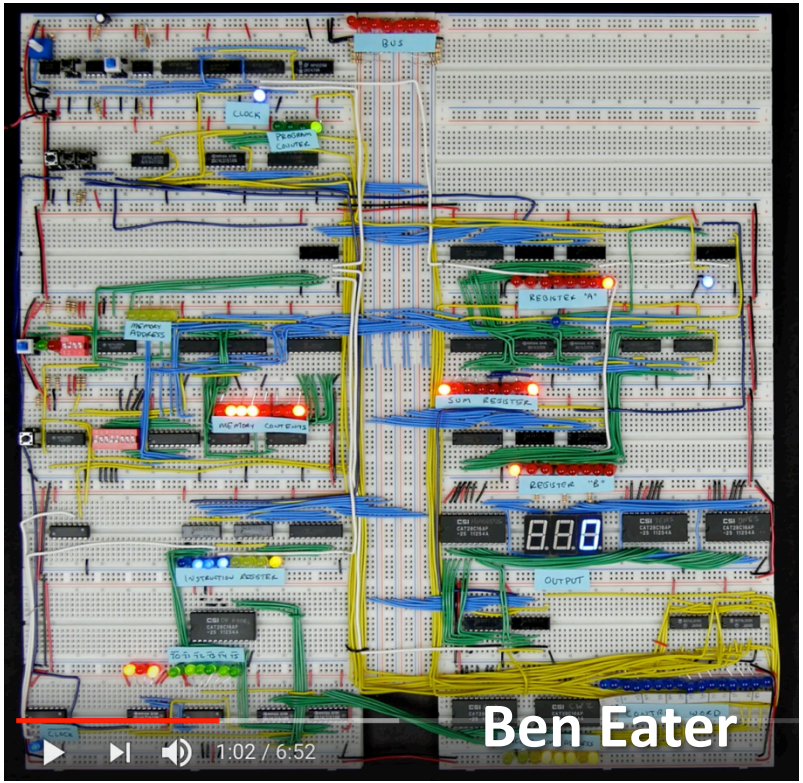


Look around for inspiration

Breadboard computer based on textbook SAP-1 design (“Simple As Possible”).

Great educational YouTube series for the 7400-series

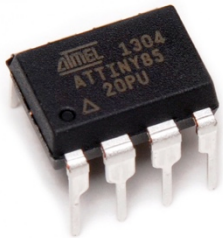
This might be pushed to play Tic Tac Toe on a 8x8 LED matrix



But then we also saw this! Quark-85

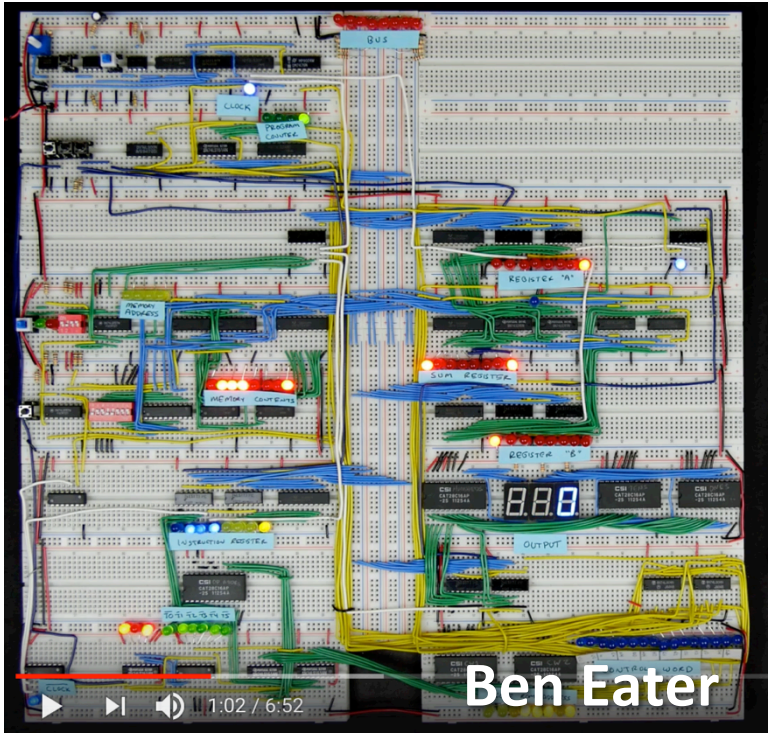
A simple ATtiny85 microcontroller with 8-bits
with 5 usable I/O lines,
8 kB EEPROM, 512 bytes RAM:

Can do color VGA, with
stereo sound and joystick input

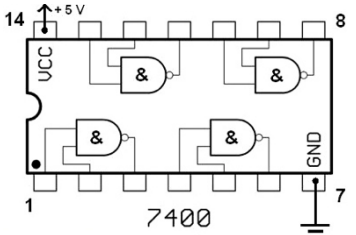


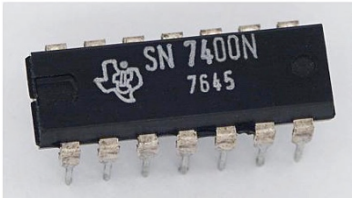
Software can create VGA signals?!?!?

A crazy idea is born: can we combine these?



Our new quest: our computer as an exercise in *minimalism*





Rule 1 Only allow the simplest of logic chips

74HC595 shift-register is “borderline OK”: ALUs, UARTs, are a no-go

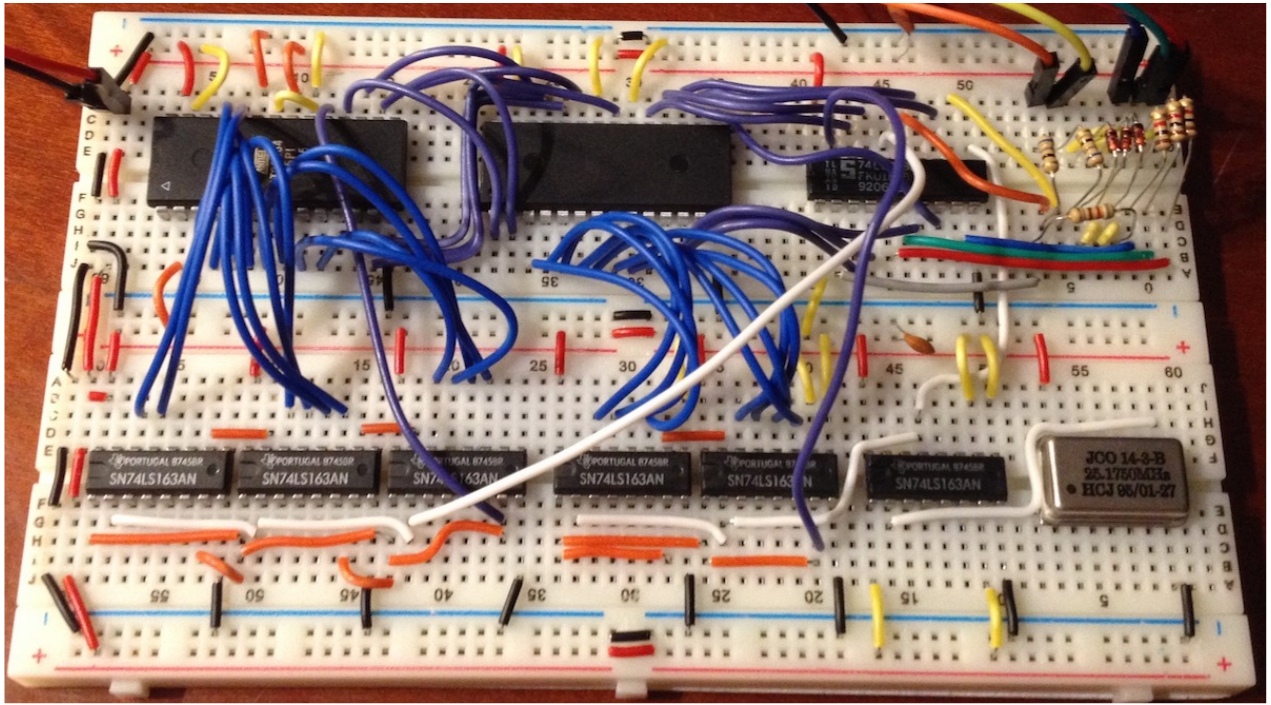
Rule 2 Must be single board: <40 chip count

Same ballpark as Wozniak’s Break Out, early PC video cards or the “Ben Eater” breadboard type of computers

Rule 3 Capable of video games with sound

Let software do the job of complex video and sound ICs

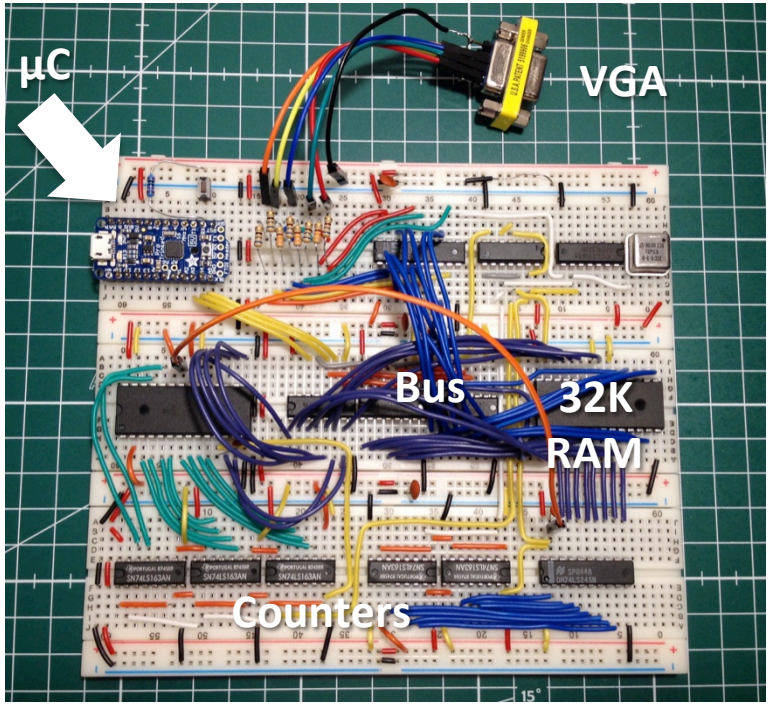
Our first working circuit



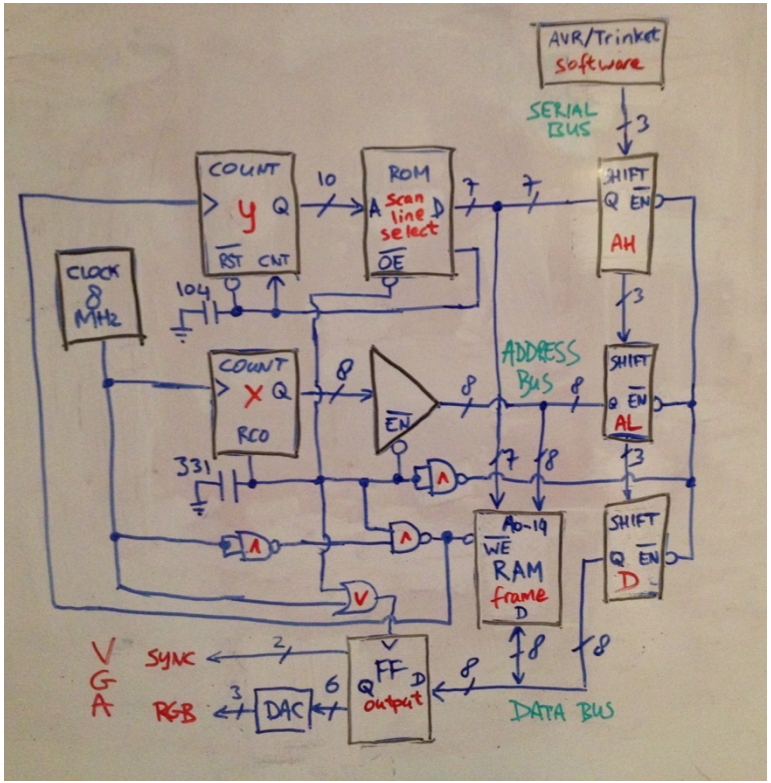
1 oscillator (25.175 MHz), 6 counters (4-bits),
2 EEPROM (8K + 32K) and 1 register (8-bits)

Look ma, no microcontroller!

Try the same with a RAM (and a microcontroller)

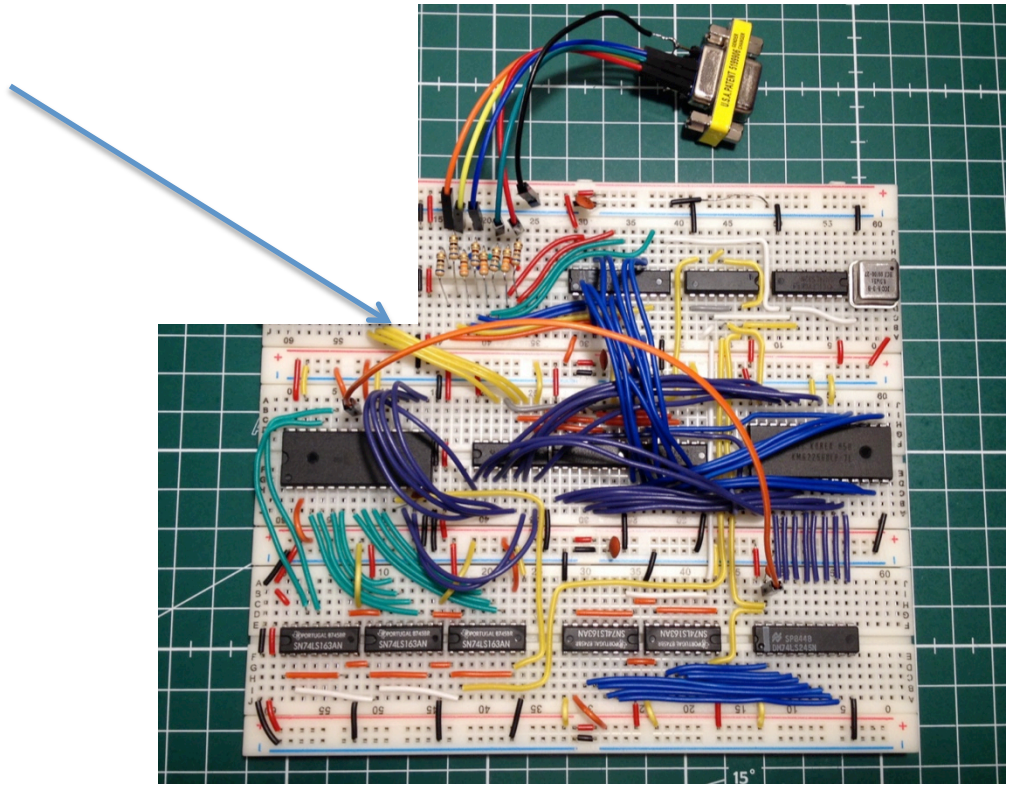


8 MHz breadboard dynamic VGA
from TTL logic and a 32K RAM.
A microcontroller to setup the RAM



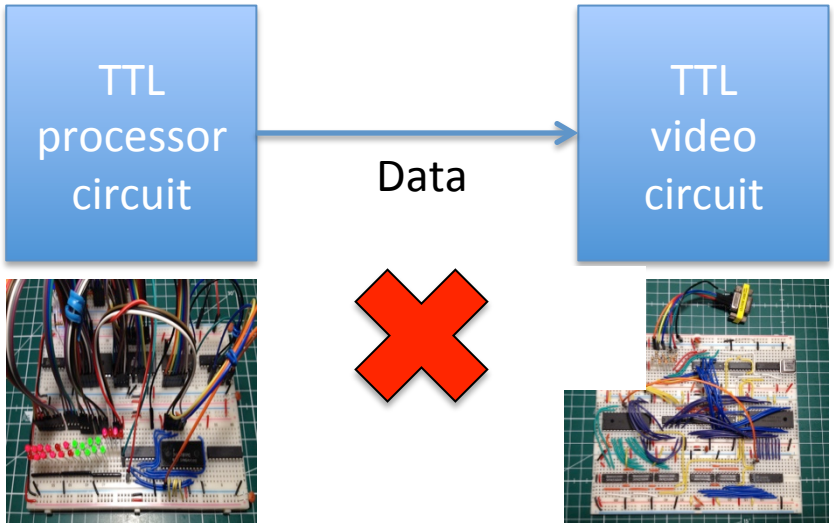
This is basically a video card

Data



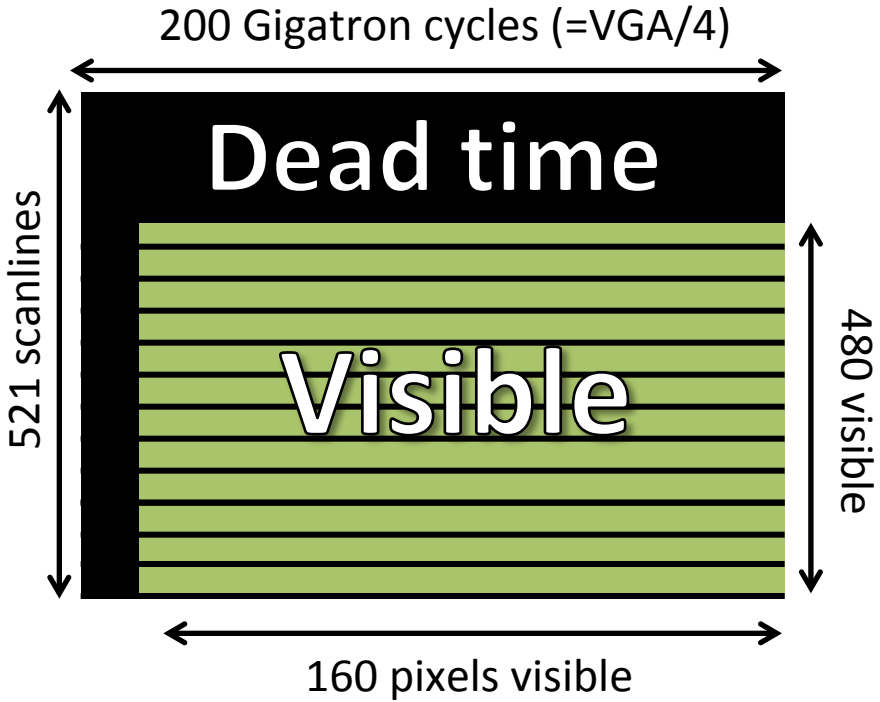
The next design step *defines* the Gigatron

This is what we really wanted to avoid ending up with:

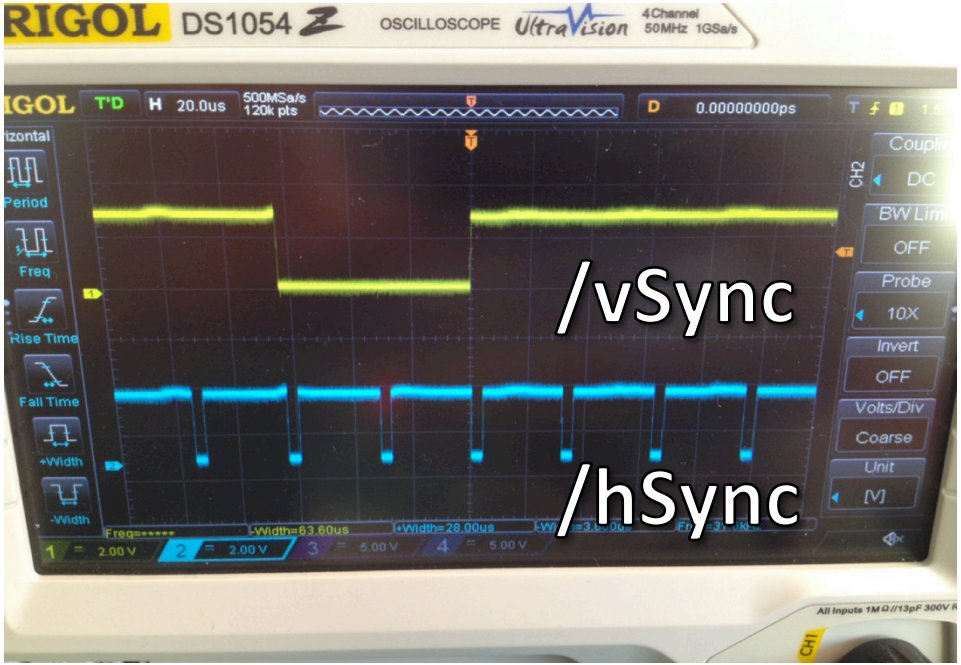


To achieve a low chip count, we must attempt to merge both functions into one

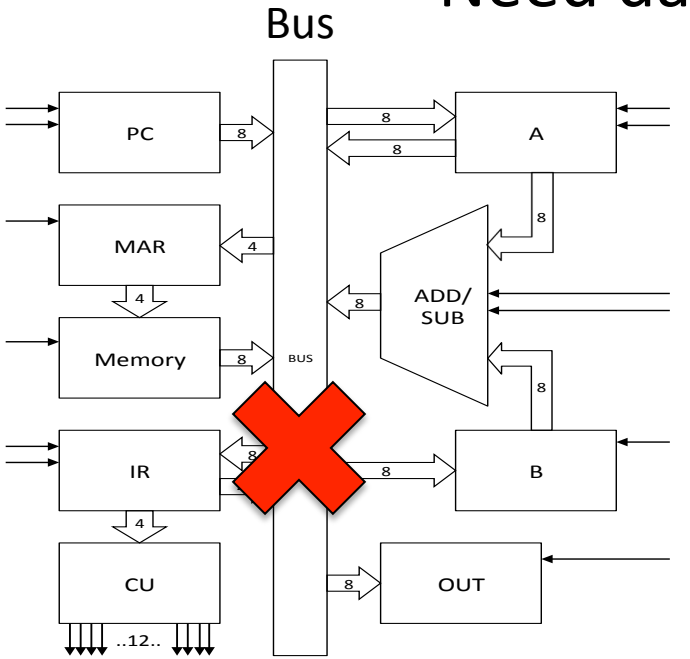
Remember Quark-85: video signals and dead time

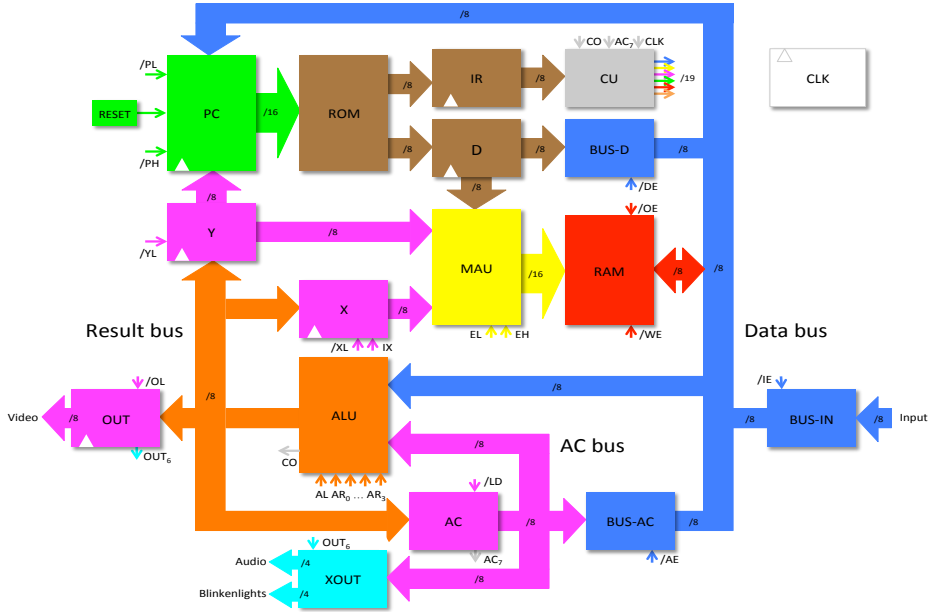


1 video frame
drawn 60 times per second



Need data path that can do it all





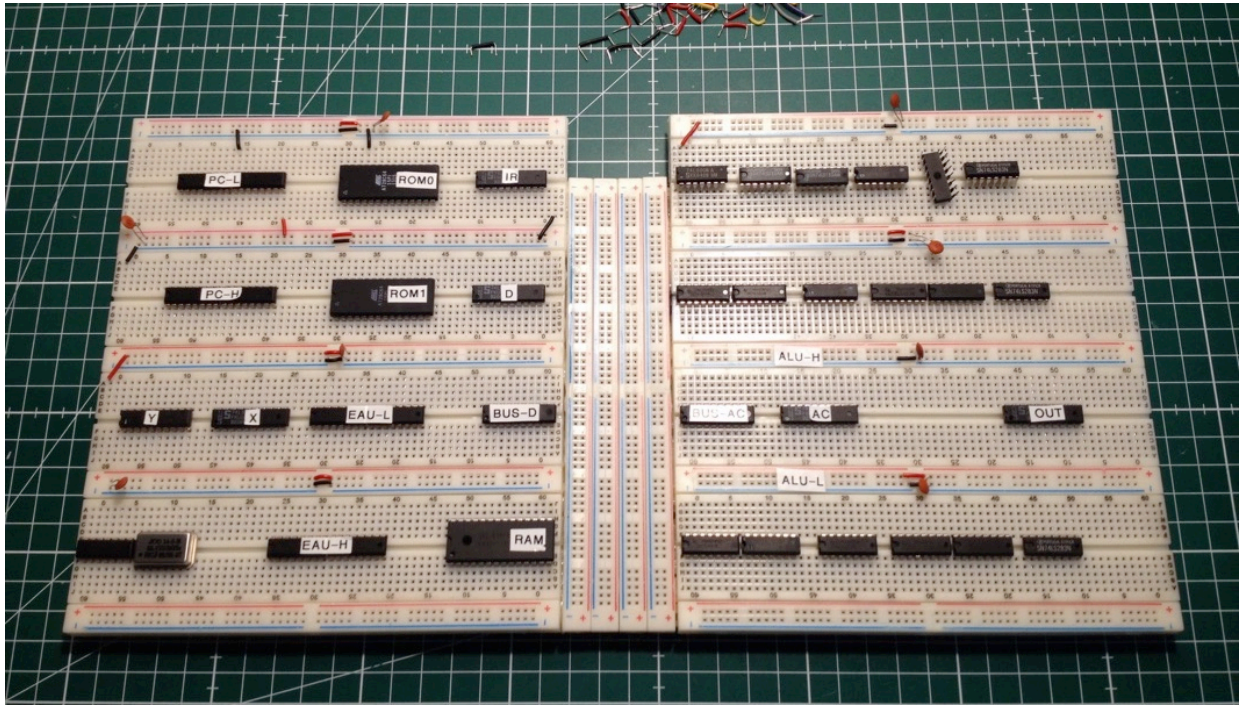
Reference CPU design (SAP-1)

- Von Neumann architecture
- 1 central bus is bottleneck: complexity↑
- Must be microcoded: speed↓↓

Our design

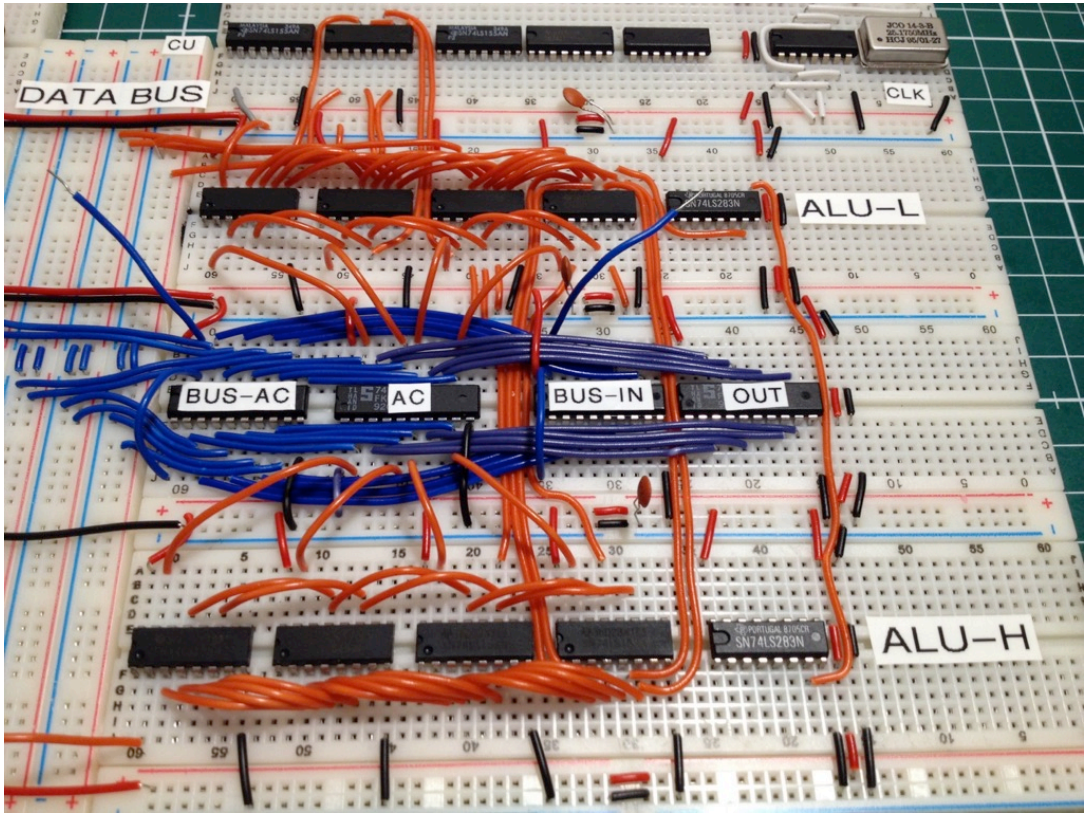
- Harvard architecture
- Split bus for efficiency: chip count↓
- Can do 1 instruction per cycle: speed↑↑

March 2017: Start building it on a breadboard



Data paths are known. The design details can be worked out as we go

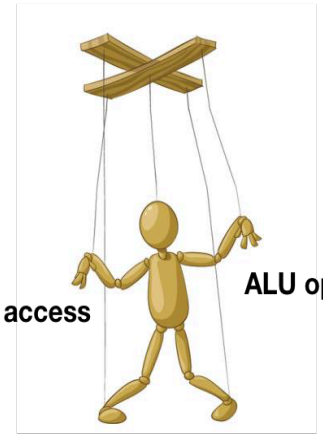
April 2017: ALU from multiplexers and adders



The last detail is … the instruction set

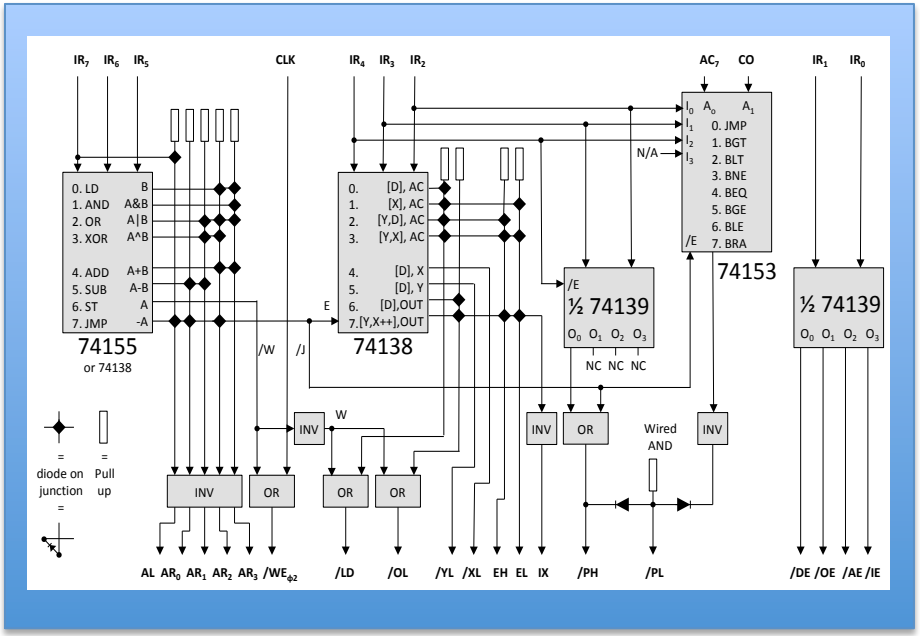
0	1	0	1	0	1	0	0

instruction
fixed length



Instruction and mode define what all units do

Map 8 instruction bits …

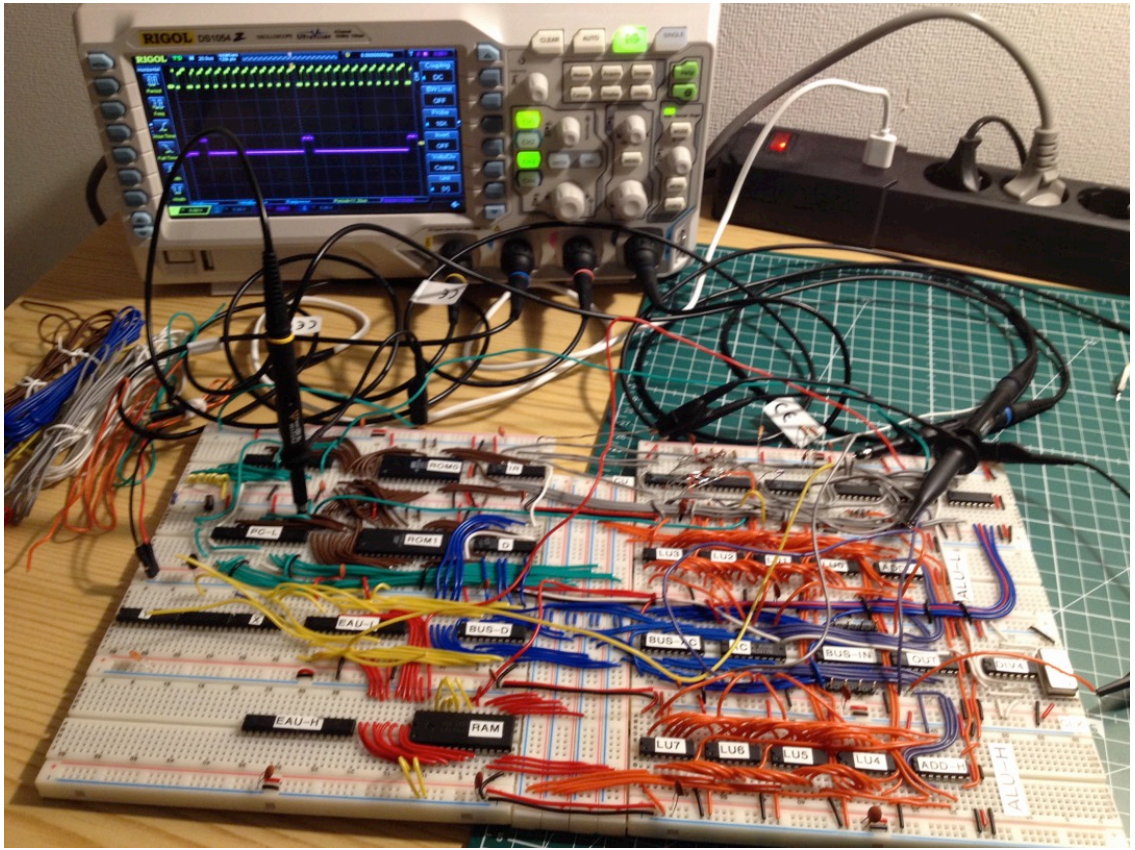


… to 19 control signals with
6 logic chips and 30 diodes

NOP	JMP
LD	BGT
AND	BLT
OR	BNE
XOR	BEQ
ADD	BGE
SUB	BLE
ST	BRA

16 native instructions,
32 modes (not all are useful)

May 2017: First Fibonacci series computed



Fibonacci program

```
0000 0000  ld   $00    ; outer loop
0001 c200  st   [$00]  ; a=0
0002 0001  ld   $01    ; b=1
0003 fc0a  bra  $0a
0004 0200  nop         ; (pipelining)
0005 0100  ld   [$00]  ; inner loop
0006 c202  st   [$02]  ; tmp=a
0007 0101  ld   [$01]
0008 c200  st   [$00]  ; a=b
0009 8102  adda [$02]
000a c201  st   [$01]  ; b+=tmp
000b 1a00  ld   ac,out ; emit next Fibonacci number
000c f405  bge  $05    ; repeat if bit7 is still 0
000d 0200  nop         ; (pipelining)
000e fc00  bra  $00    ; start over again
000f 0200  nop         ; (pipelining)
```

F(n) = F(n-2) + F(n-1)

0

1

1

2

3

5

8

13

21

34

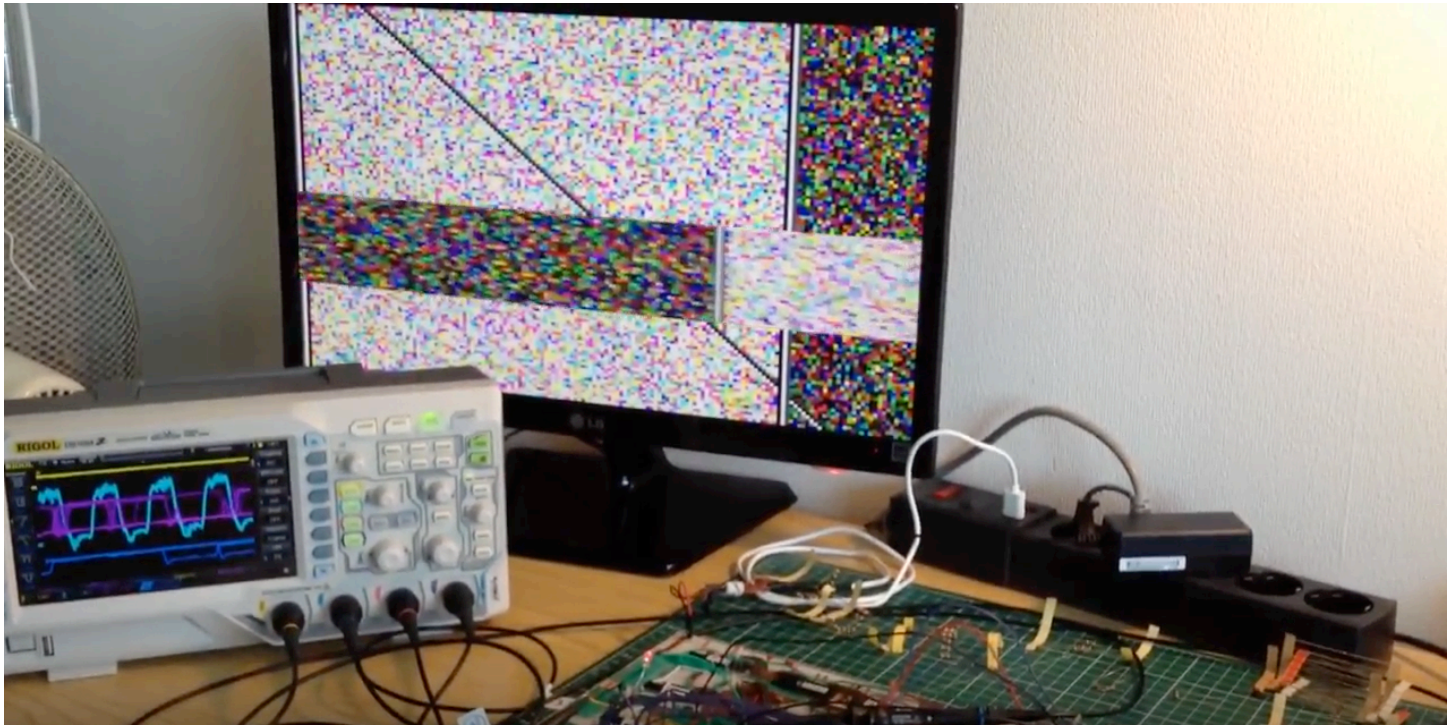
55

89

144

…

May 2017: First moving video

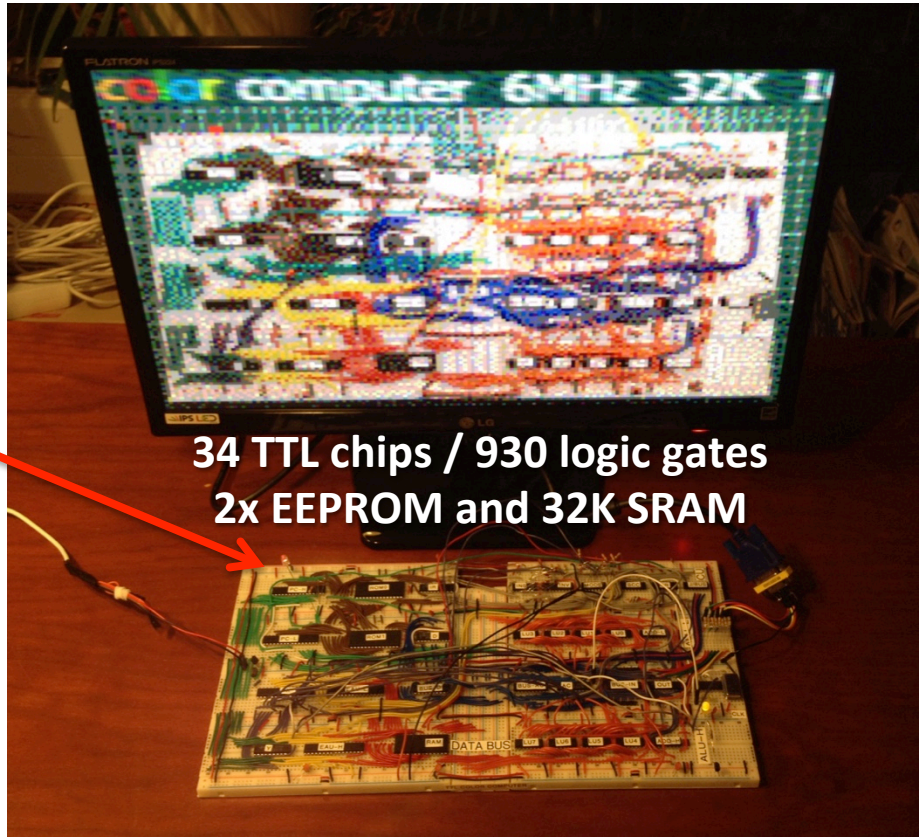


First moving video from my breadboard TTL color computer. Again, the test image is just initialized SRAM garbage with some lines drawn over it.

https://www.youtube.com/watch?v=MHs7bQgqABM

June 2017: The breadboard becomes self-aware

And it blinks an LED



Minimalism

No standard instruction set

No interface adapter chips

No linear address space

No relative addressing

No flags register

No register file

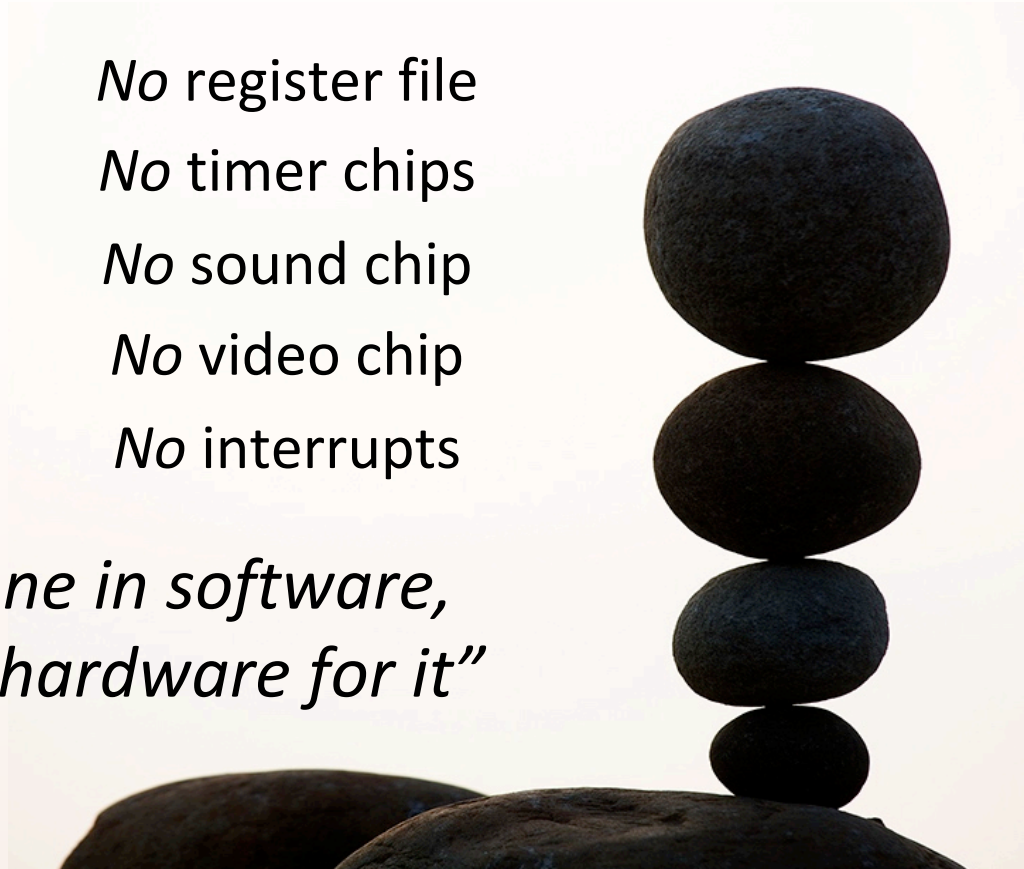
No timer chips

No sound chip

No video chip

No interrupts

“If it can be done in software,
you don’t need hardware for it”



In reality we could only do very simple demos this way

Game with bricks, ball and bat

Application logic mixed with video generation loop.
No interrupts, so we must count every instruction to keep VGA in sync. This is tedious.

We need a higher programming level: 16-bit virtual CPU

34 instructions, 16-bits

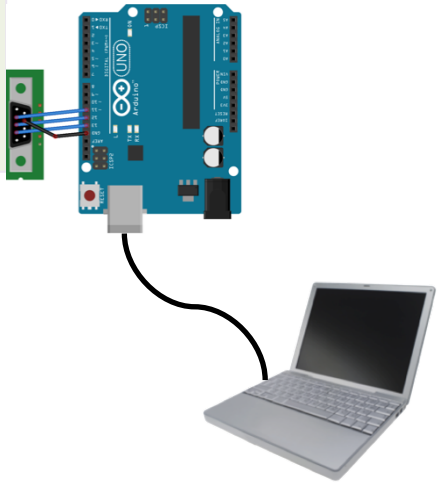
```
ADDI ADDW ALLOC
ANDI ANDW BCC
BRA  CALL DEEK
DEF  DOKE INC
LD   LDI  LDLW
LDW  LDWI LSLW
LUP  ORI  ORW
PEEK POKE POP
PUSH RET  ST
STLW STW  SUBI
SUBW SYS  XORI
XORW
```

High level

vCPU runs application code <u>from RAM</u>

SWEET16 inspired, Von Neumann!

34 self-timed instructions



Low level

Native code for hardware functions

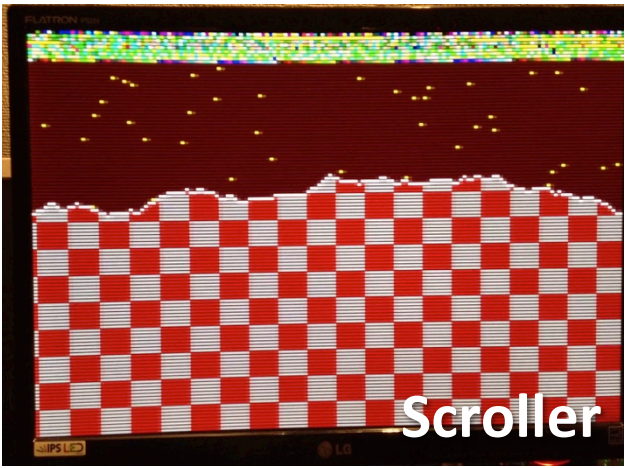
VGA compatible signals

4 channel sound, I/O

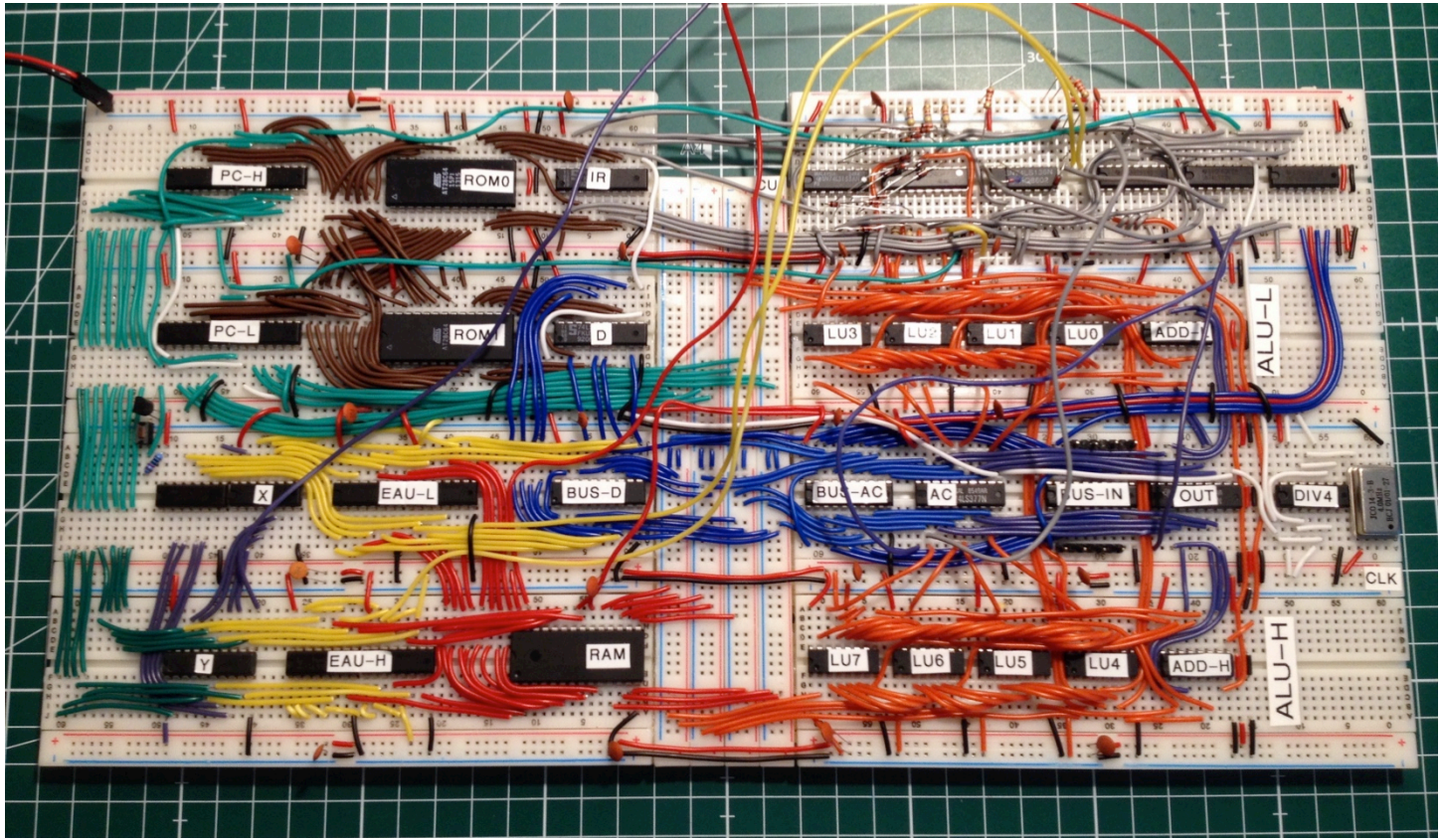
ROM disk

an interpreter: **vCPU**

December 2017: first programs with vCPU



Wise people stop here



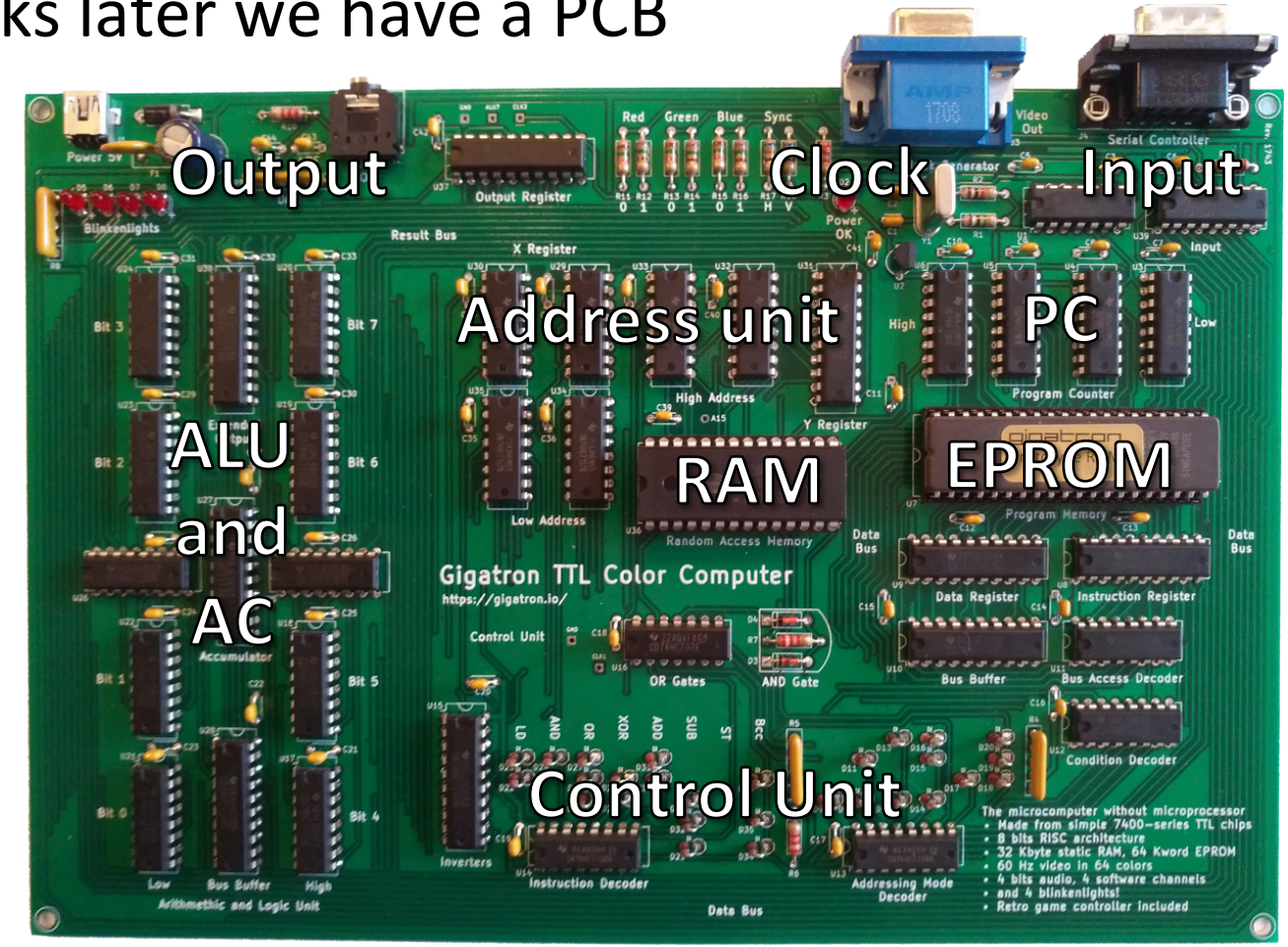
Wise people stop here

“There is no product obscure enough that people are not interested in it.”

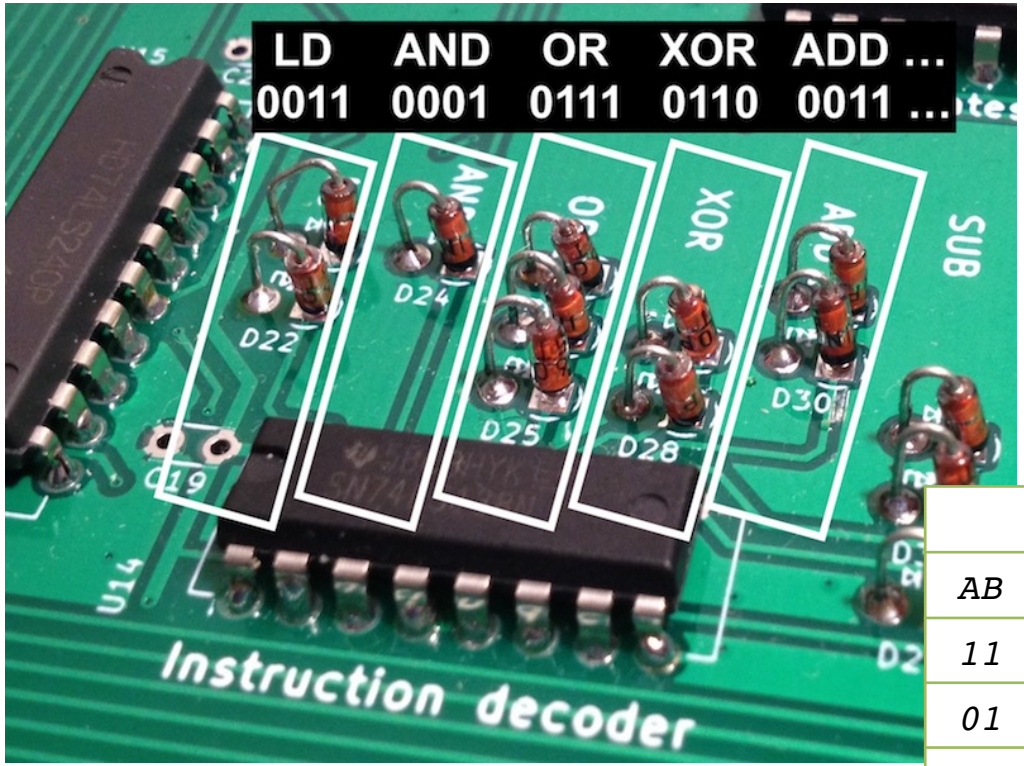
Oscar “Obsolescence Guaranteed” Vermeulen

So…. we make it a kit! It sounds like fun and our friends ask for one…

10 weeks later we have a PCB



Manually routing for interesting layout

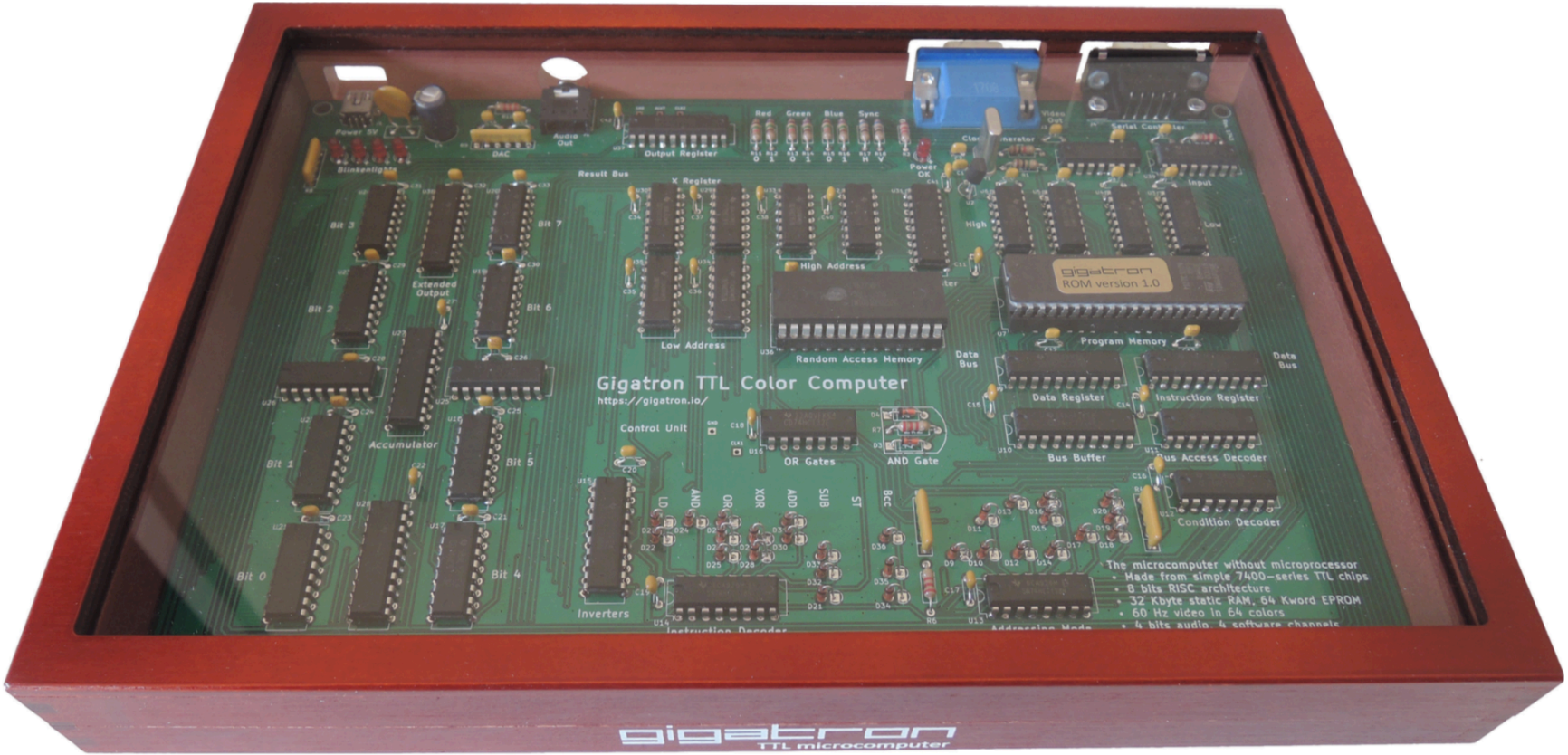


For example: here the diodes visualize the truth tables for each operation

No diode = 0, Diode = 1

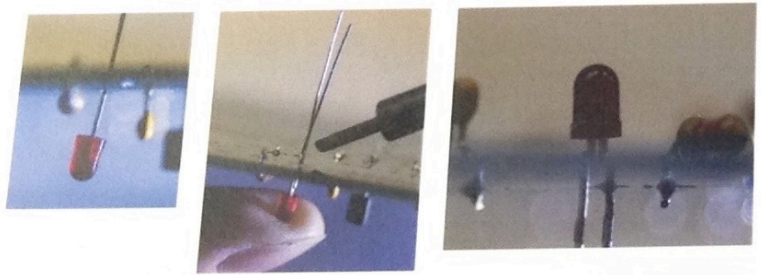
	LD	**AND**	**OR**	**XOR**	**ADD**	**SUB**
AB	B	A∧B	A∨B	A≠B	B	~B
11	**1**	**1**	**1**	0	**1**	0
01	**1**	0	**1**	**1**	**1**	0
10	0	0	**1**	**1**	0	**1**
00	0	0	0	0	0	**1**

Search for a nice enclosure



Fool proof manual

and solder one lead with just a little bit of solder. It is now probably not where you want the LED to be.



Next, apply heat from the soldering iron while, again **gently**, pushing the LED in the correct position. When you are satisfied, solder the other lead and re-heat the first one to make sure both leads are soldered well.

The same goes for ICs and IC sockets: solder one pin with a little solder, then the opposite pin. Next, see if it is placed correctly. If not, re-heat and **gently** push the

5

Assembly and testing

How to build your gigatron

This chapter explains how to build the kit, part by part. Before building, make sure that you know how to solder (see chapter 4) and have checked that you have all the components (chapter 3). There's quite a lot of components, but don't be intimidated, we will be soldering them part by part. Building the gigatron will take about 3 to 4 hours.

1. To practice with soldering, we start by soldering **40 ceramic 100nF capacitors**, marked C5 through C44. There are at least 40 provided in the kit. Do not confuse them with the three 47pF capacitors. Only put in from C5

Tedious logistics

1+ hour effort per kit shipped

Buying parts

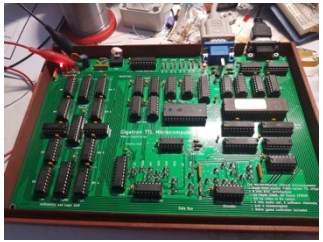
Our TTL computer is now a DIY soldering kit!

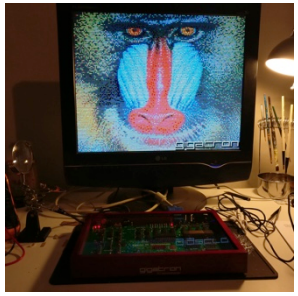
Just need a soldering iron, a multi-meter and 3-4 hours of time to build. No oscilloscope needed

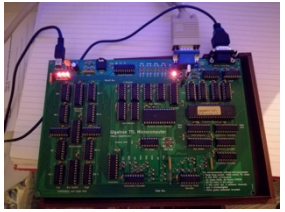
gigatron

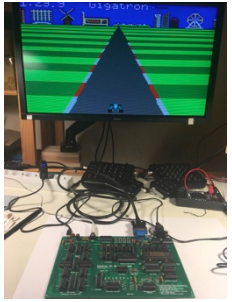
TTL microcomputer

A new one is born every day now

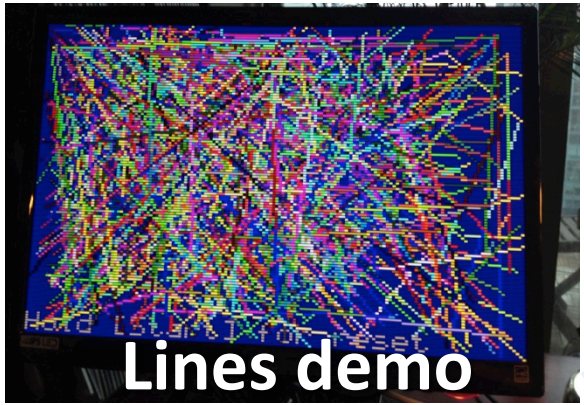


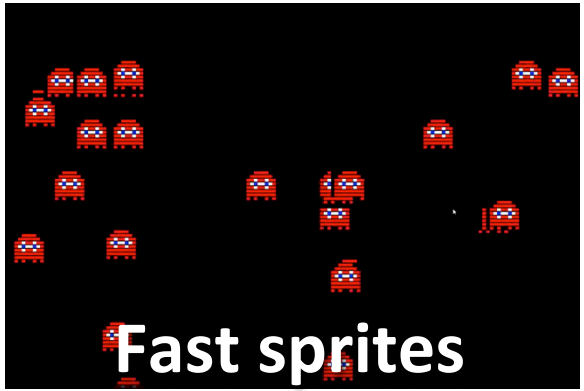


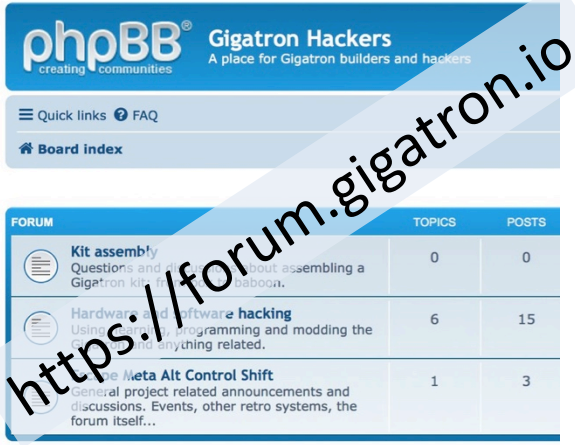


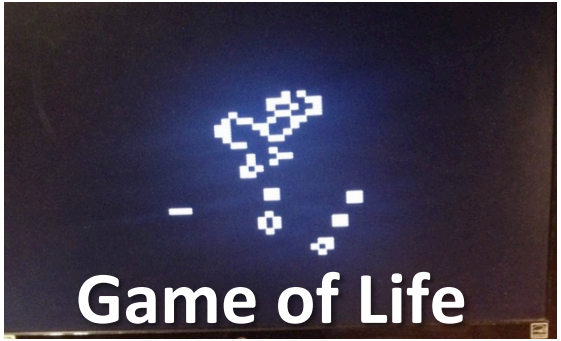


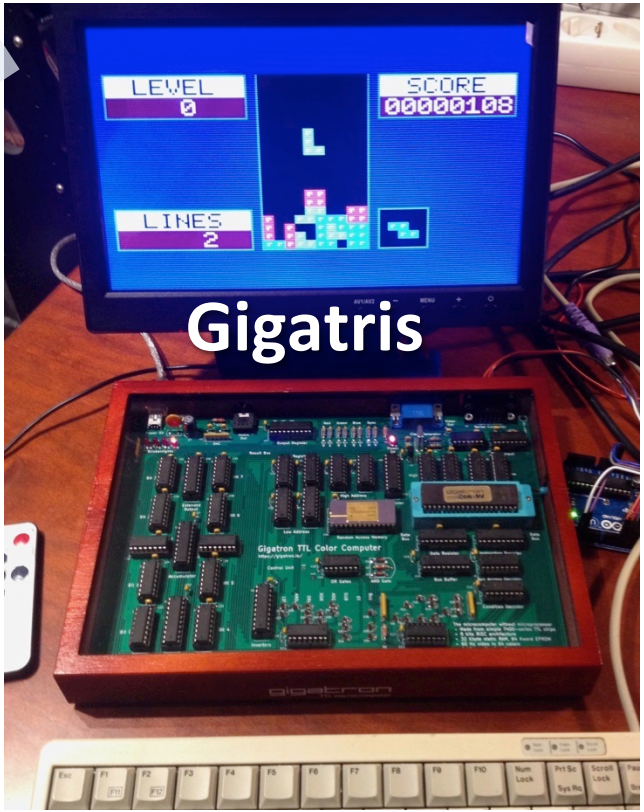
Community after first month











https://forum.gigatron.io

YouTube spreads the word

Not all ideals made it into “v1”

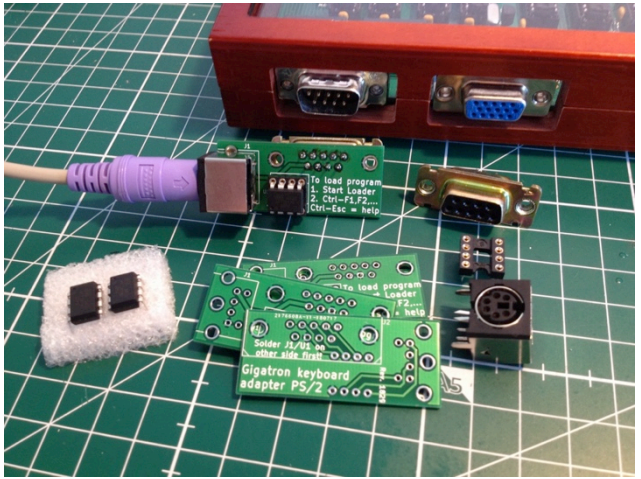
There was no keyboard hookup

Even the PS/2 protocol turns out to be an ugly beast.

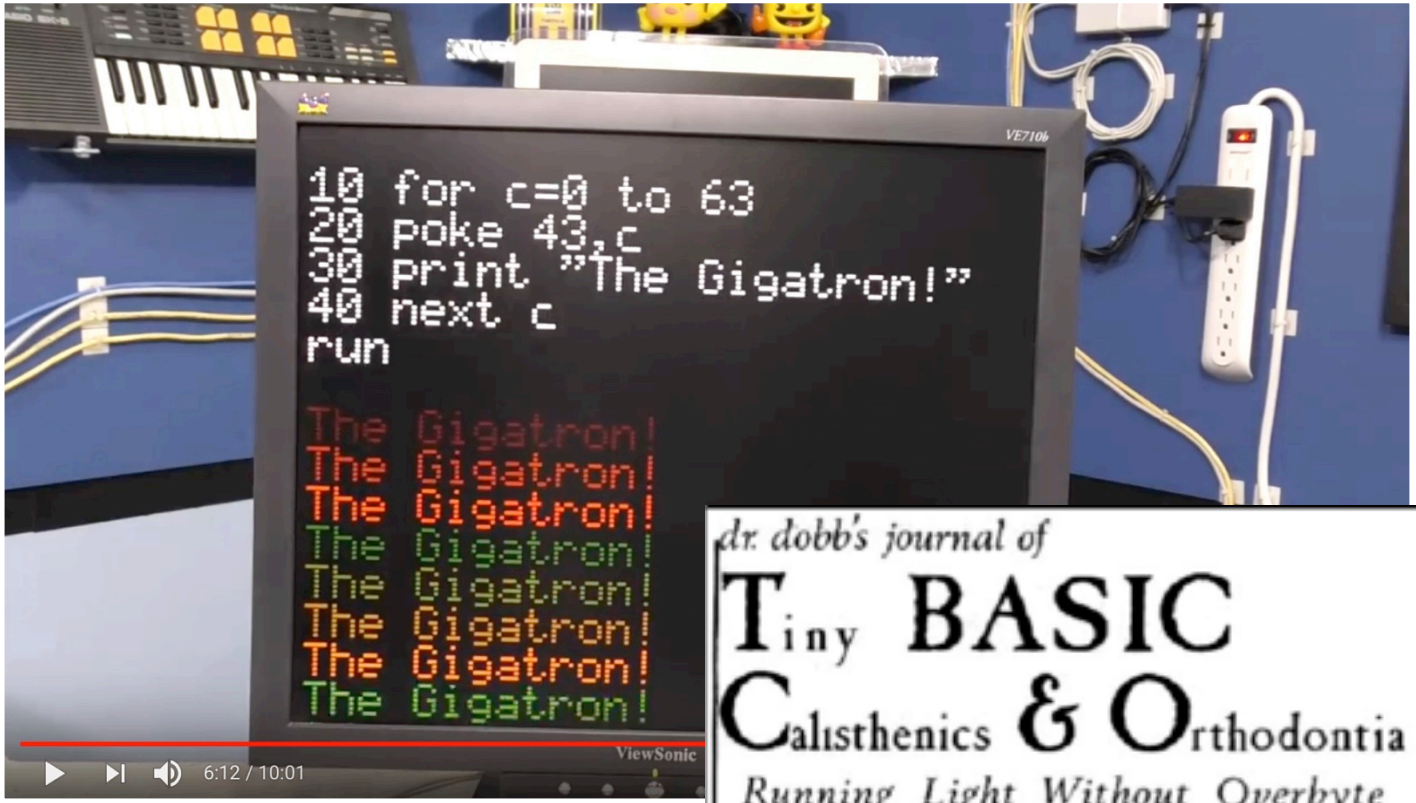
There was no built-in BASIC

Bill Gates doesn’t respond to our e-mails and it takes many weeks to write a BASIC.

June 2018: PS/2 keyboard adapter



We ported Tiny BASIC to the Gigatron



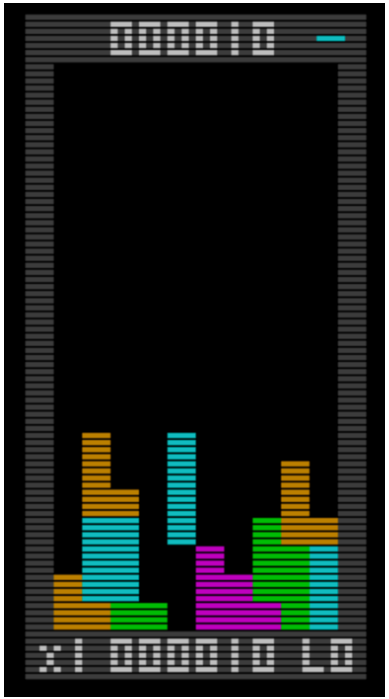
Fibonacci (again), in BASIC

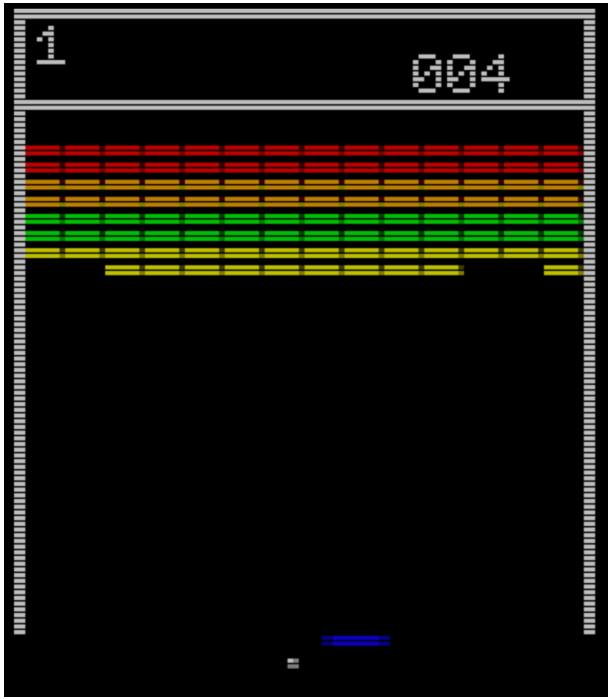
```
1 'Fibonacci with bignums
2 a(0)=0:b(0)=1:c(0)=0
3 n=n+1:printn;": ";
4 fori=0toj:put48+b(j-i)
5 nexti:print:fori=0toj
6 c=a(i)+b(i)+c:a(i)=b(i)
7 b(i)=c:ifc<10 c=0:goto9
8 b(i)=c-10:c=1
9 nexti:ifc=0 goto12
10 j=j+1
11 a(j)=0:b(j)=c:c(j)=0
12 print:goto 3
9120 bytes free
Ok
```

This time with bignums

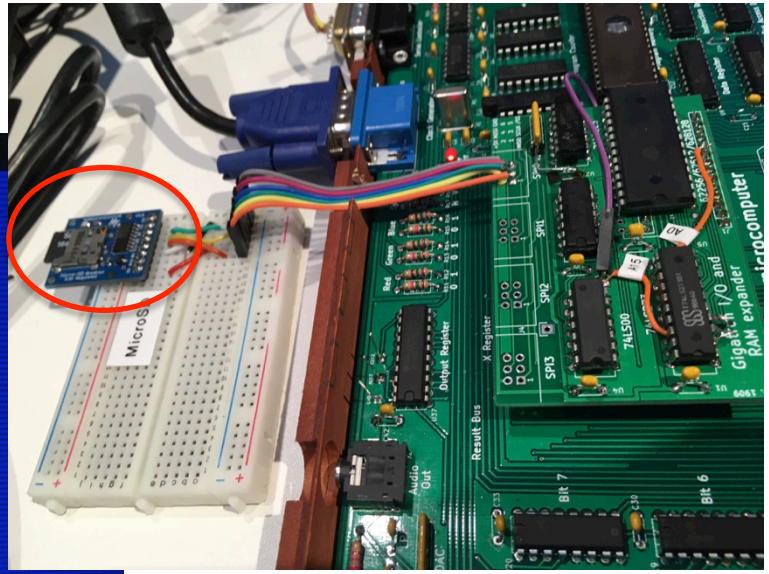
```
93: 126643214342800811782

94: 204913025252120171200

95: 331556239594009829982

96: 536469264845211541182

97: 868025504439221371164

98: 1404494769284432913346
?Break error in 9
Ok
```

In the meantime, users have written more cool games

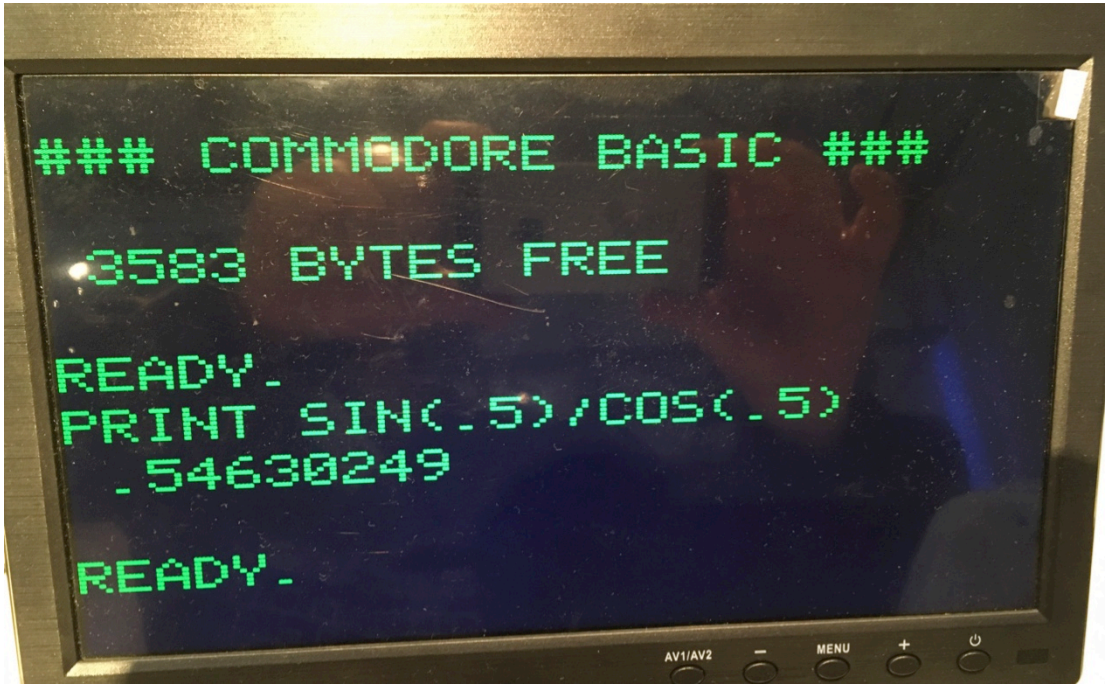




New development: SPI port gives SDC/MMC access



New development: v6502 gives Micro-Soft BASIC



Second virtual CPU

Now we're multi-core ☺

Runs 6502 code

Effective speed 125 kHz

+ Apple-1 software

+ Original Micro-Soft BASIC

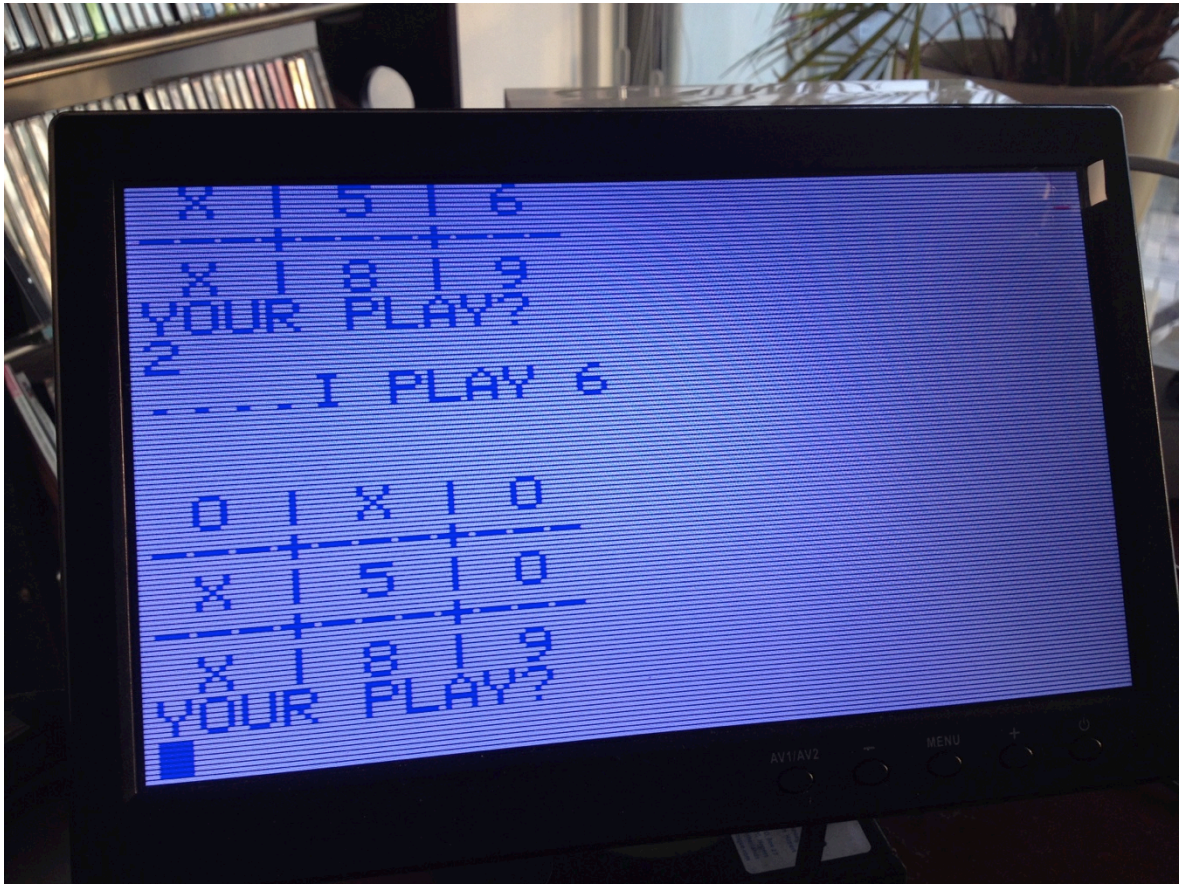
→ floating point!

And oh, remember our original goal?

Tic Tac Toe?

Source: http://www.talkingelectronics.com

Tom Pittman’s 1977 Tic-Tac-Toe BASIC program works



Tom Pittman’s 1977 Tic-Tac-Toe BASIC program works