Gigatron TTL microcomputer
“Brand new vintage”

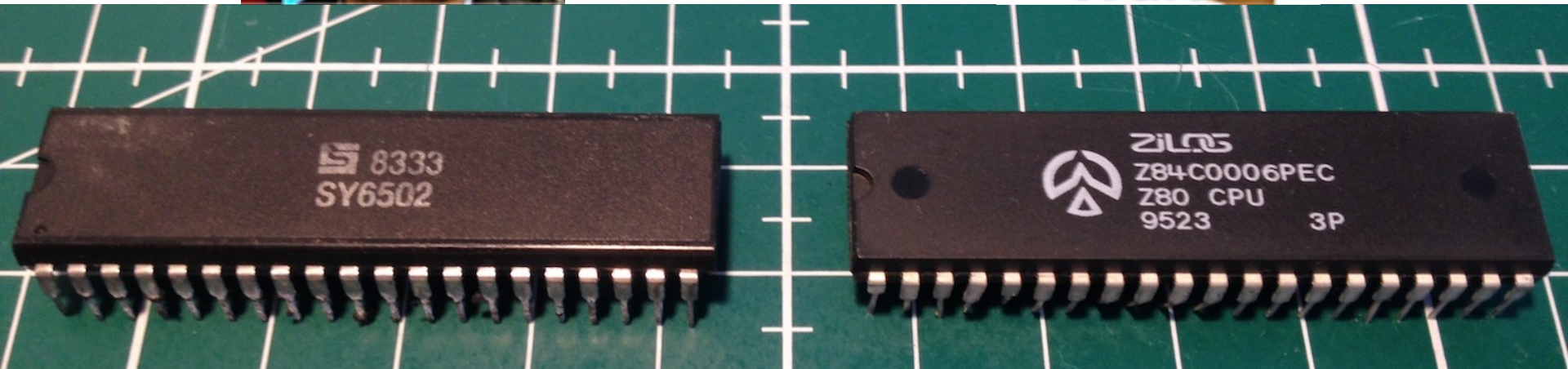
VCF Zürich 2018

Marcel van Kervinck

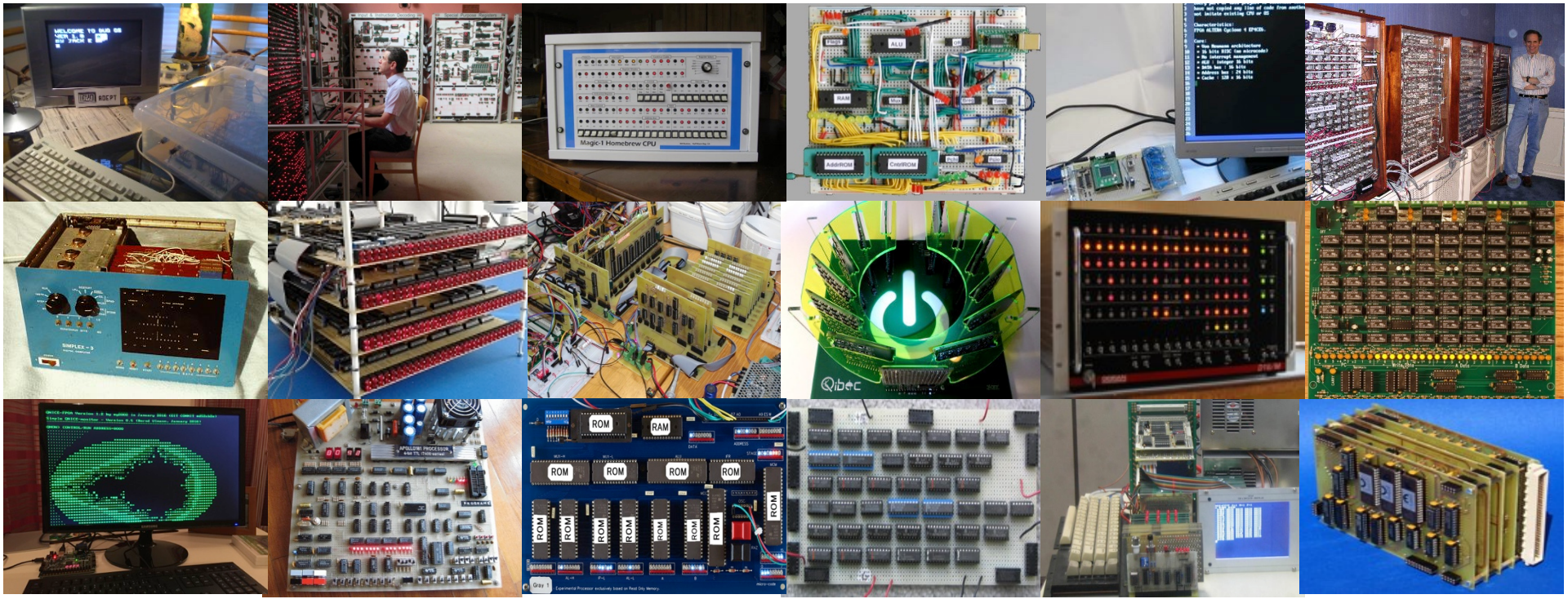
About us


Marcel


Walter


8333
SY6502
ZiLOG
Z84C0006PEC
Z80 CPU
9523
3P

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



https://www.homebrewcpuring.org

Before you begin

What core building blocks?

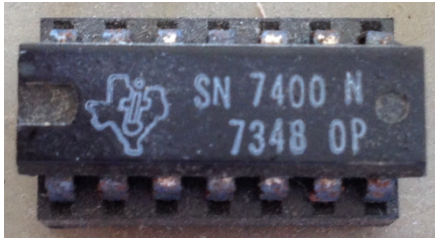
FPGA, SSI logic chips, NAND gates, discrete transistors, tubes, relays, steam punk, …

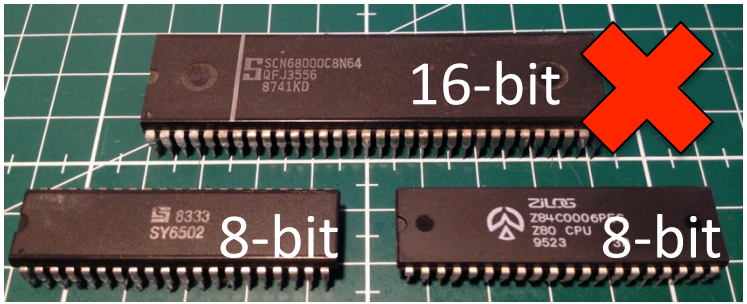
Data path size?

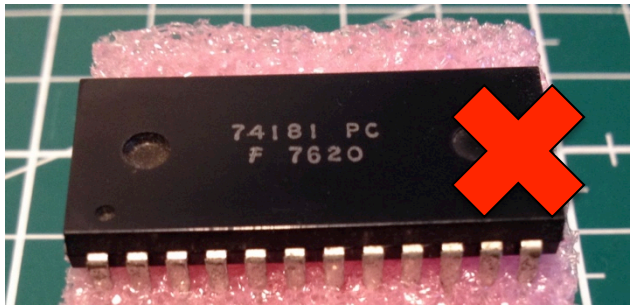
64 bits, 32 bits, 16 bits, 8 bits, 4 bits, 1 bit, other…

Standard ALU chips or custom?

74181 chips (4-bit ALU)?







Our choices:

7400 series logic "TTL"

8-bit system

No complex chips

Much more to consider

Harvard or Von Neumann?

Microprogramming or RISC?

Pipelining yes or no?

Existing instruction set or own?

Peripherals, extendibility, power, ..

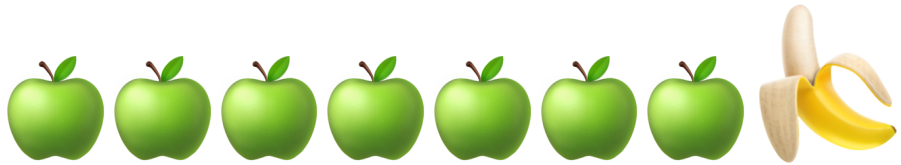
Time and budget?

1-2-3 days per week for 3m-6m-1yr

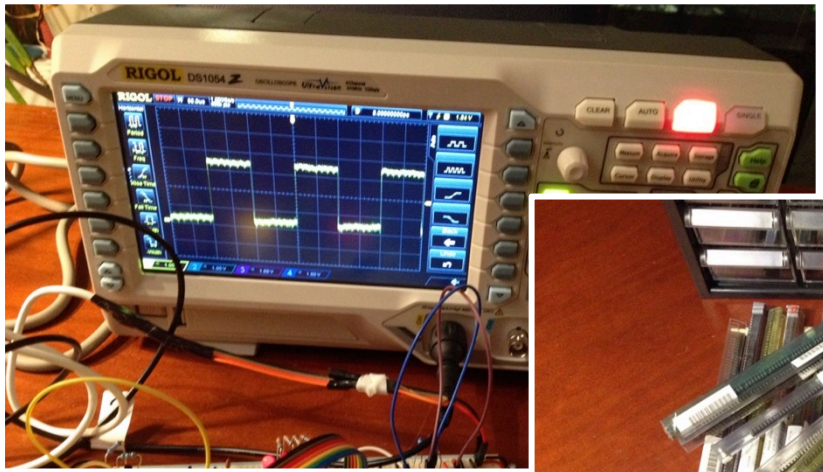
700–1000 euro to first PCB

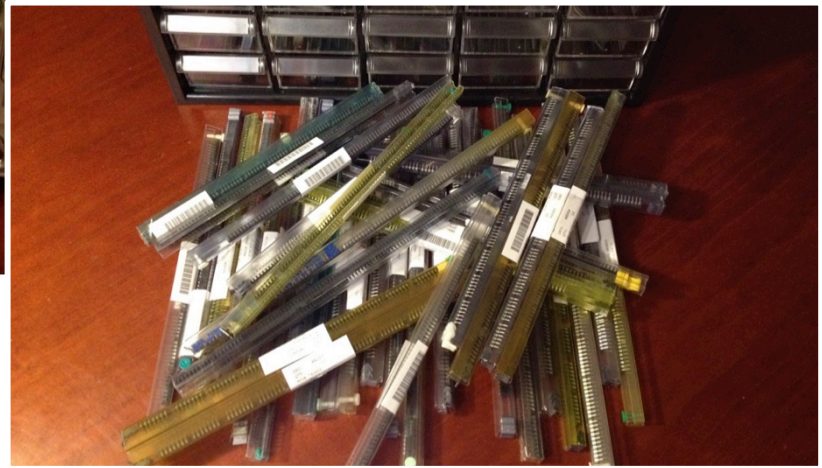
It better be fun

Most important: what makes yours *unique?*

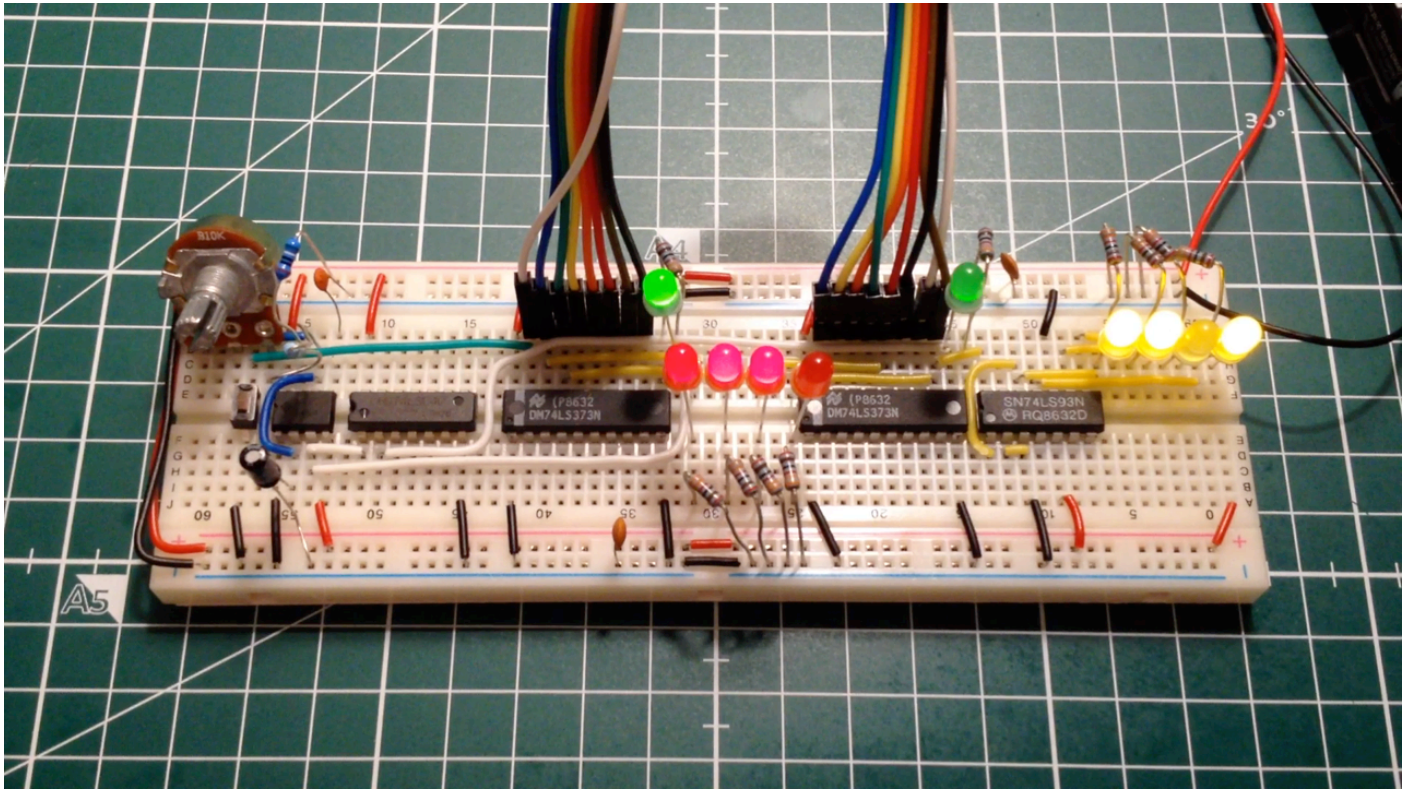


Buy books, tools and hundreds of 7400-series chips …

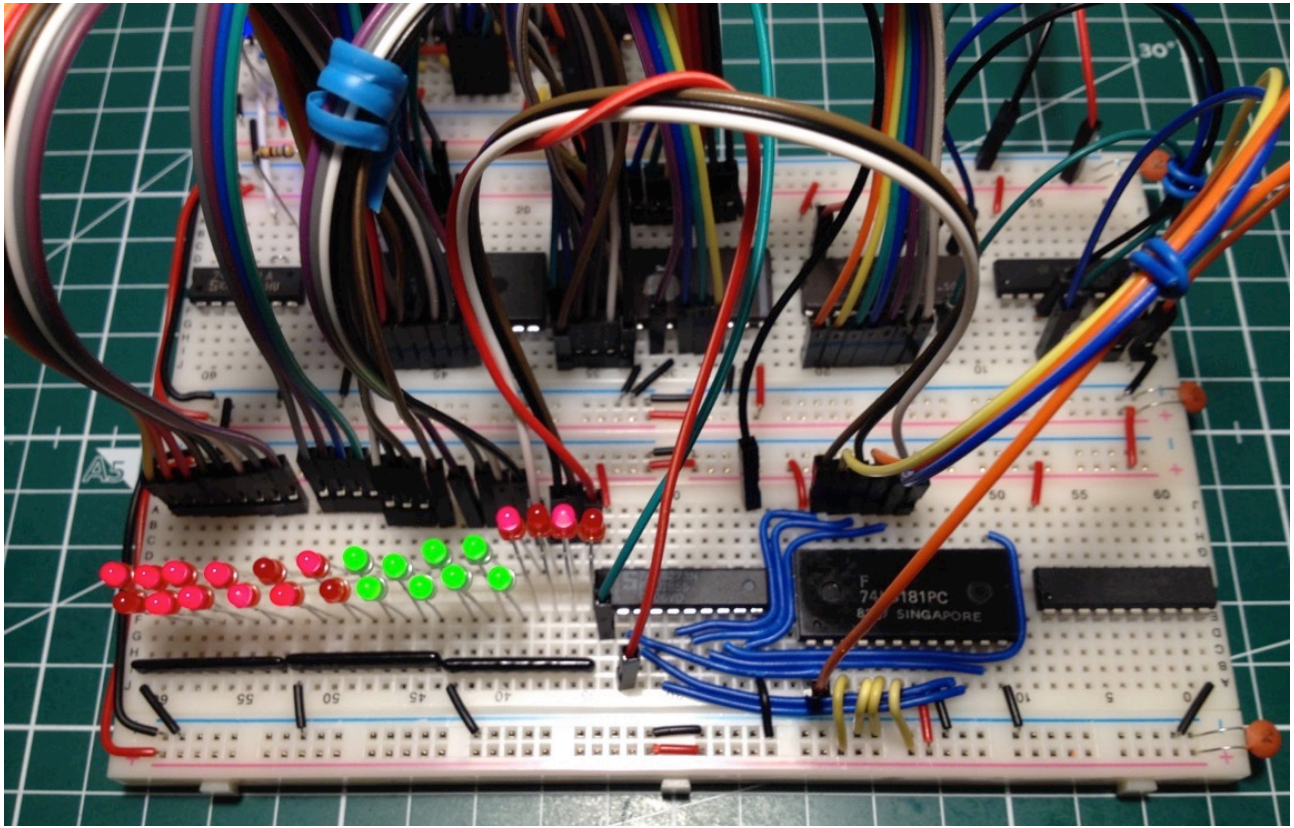




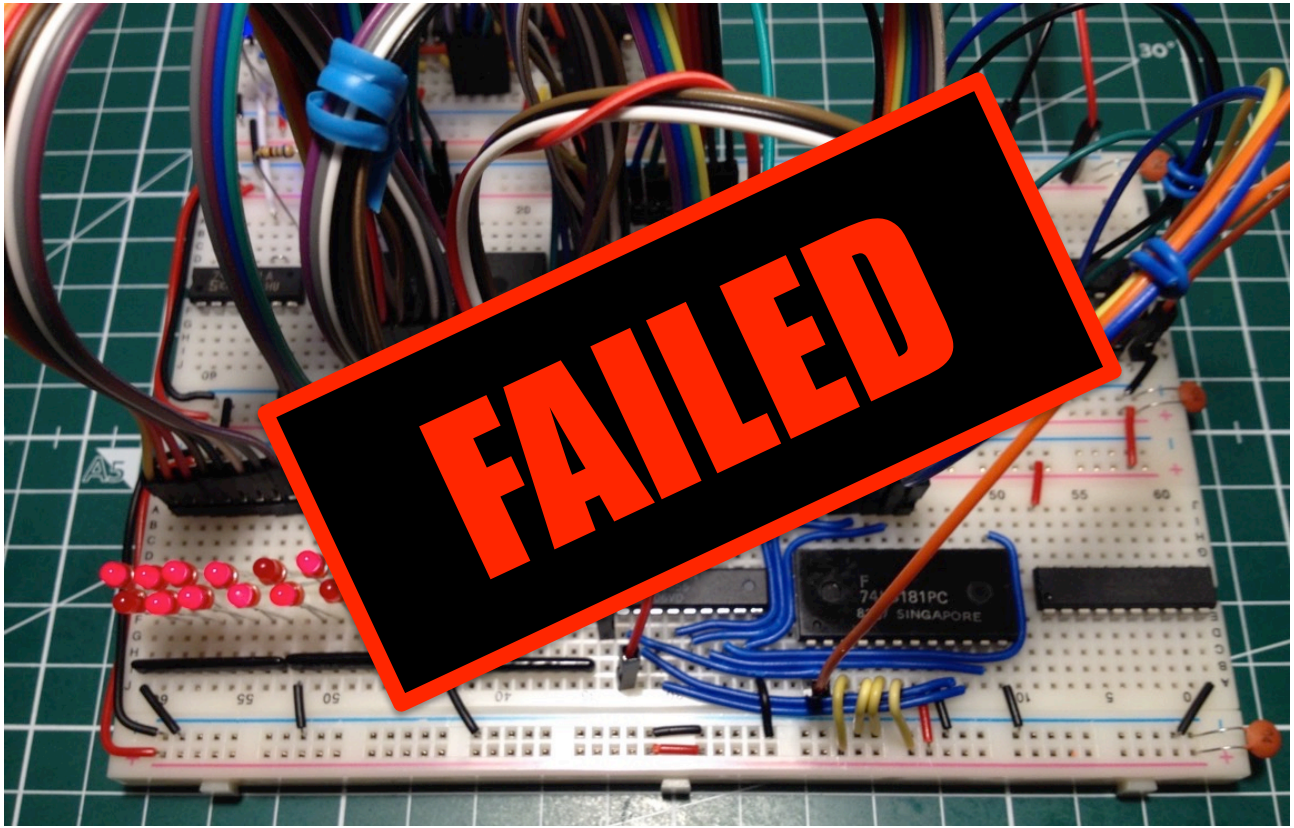
… learn how the components work …



… and you can build a 4-bit computer!



… and you can build a 4-bit computer!

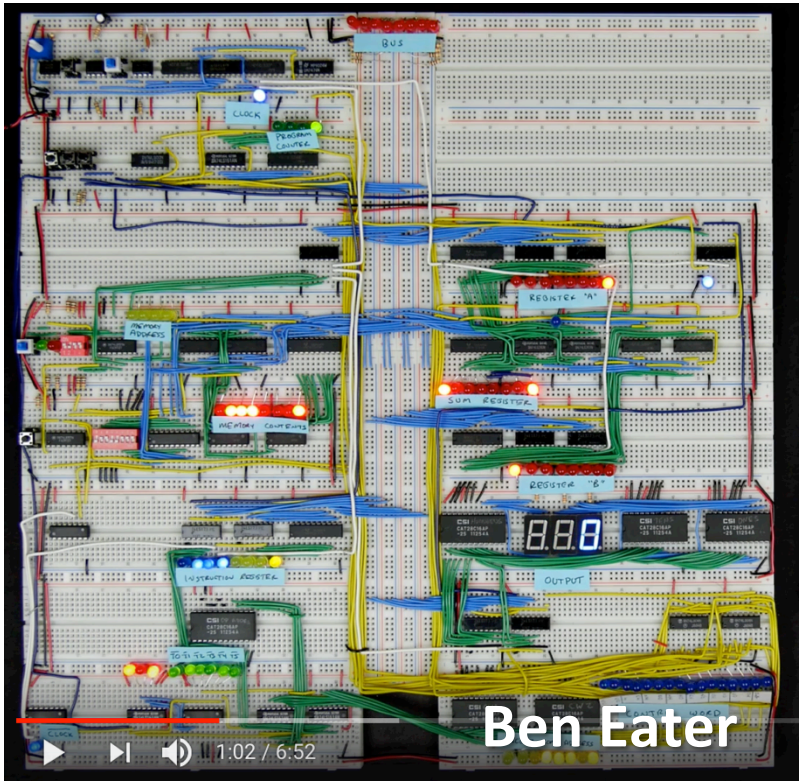


Look around for inspiration

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

Great educational YouTube series for the 7400-series

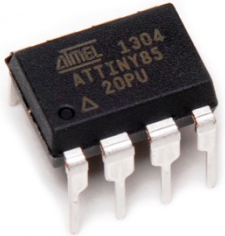
This might be pushed to play Tic Tac Toe on a 8x8 LED matrix



But then we also saw this! Quark-85

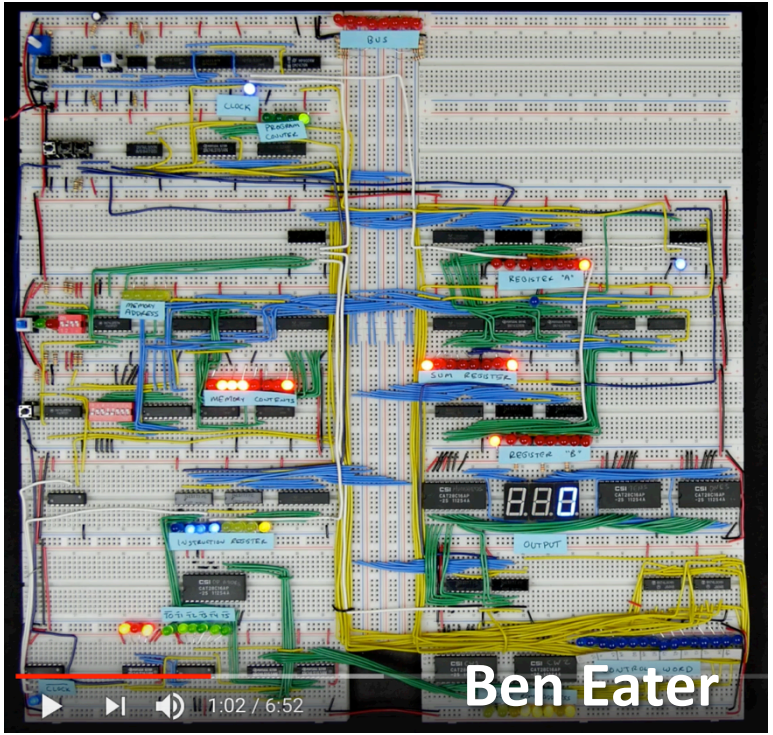
A simple ATtiny85 microcontroller with
8-bits with 5 usable I/O lines,
8 kB EEPROM, 512 bytes RAM :

Can do color VGA, with
stereo sound and
joystick input

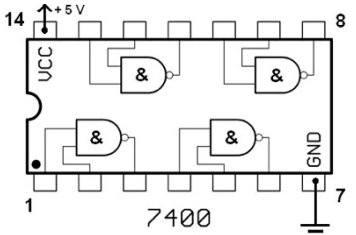


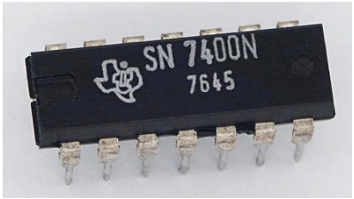
Software can bit-bang VGA?!?!?

A crazy idea is born: can we combine these?



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





Rule 1 No complex logic chips

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

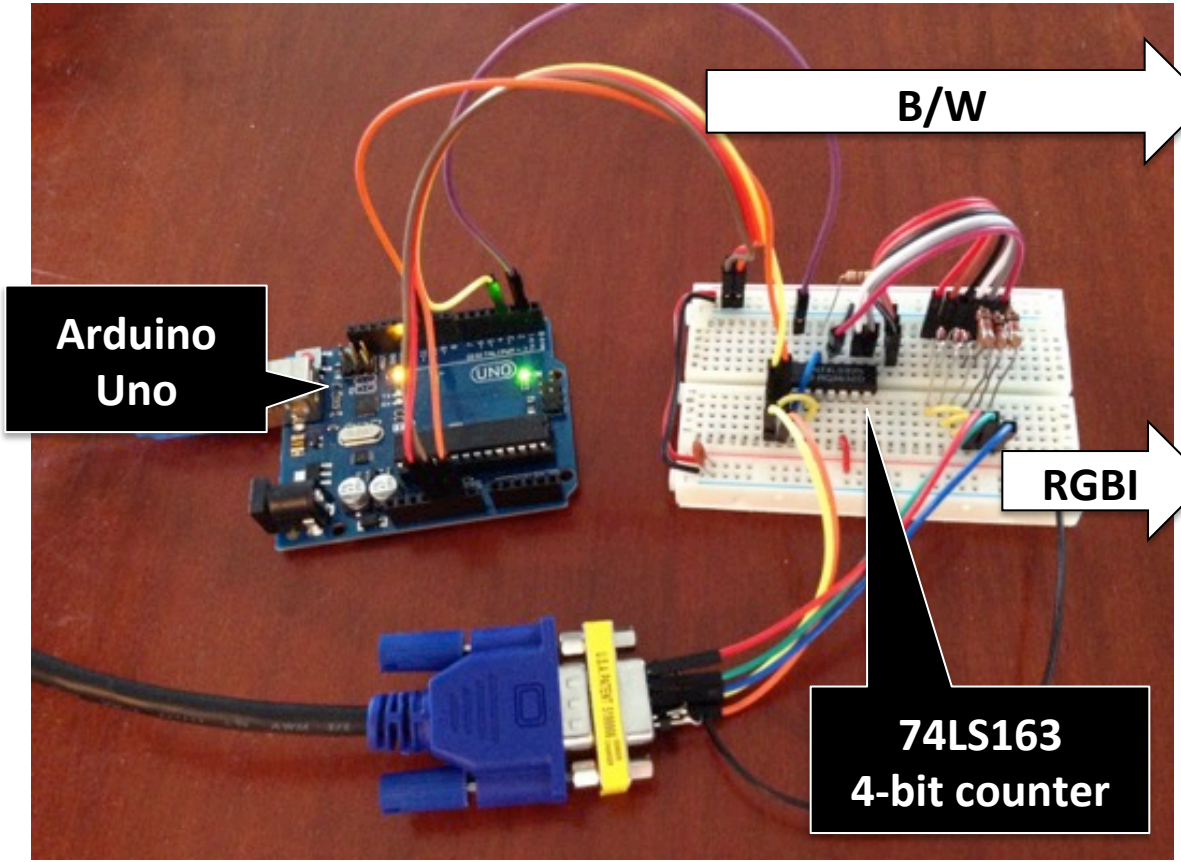
Rule 2 Single board, 30-40 chip count

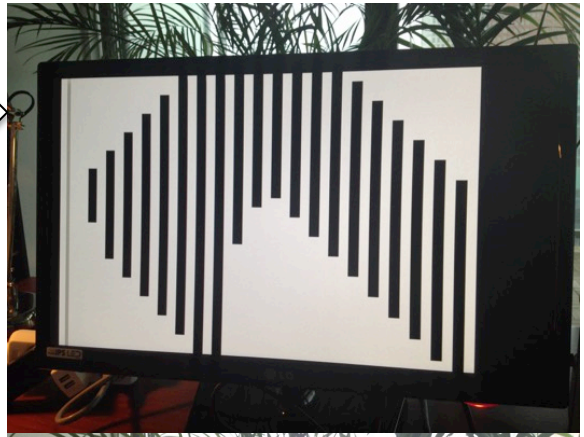
Same ballpark as Wozniak’s Break Out, early PC video cards or the “Ben Eater” breadboard type of computers

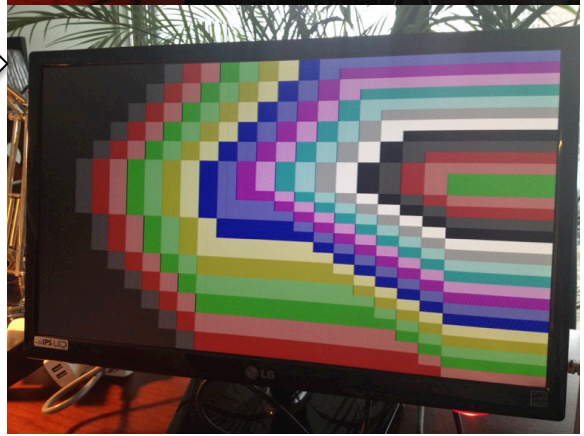
Rule 3 Capable of video games with sound

Let software do the job of complex video and sound ICs

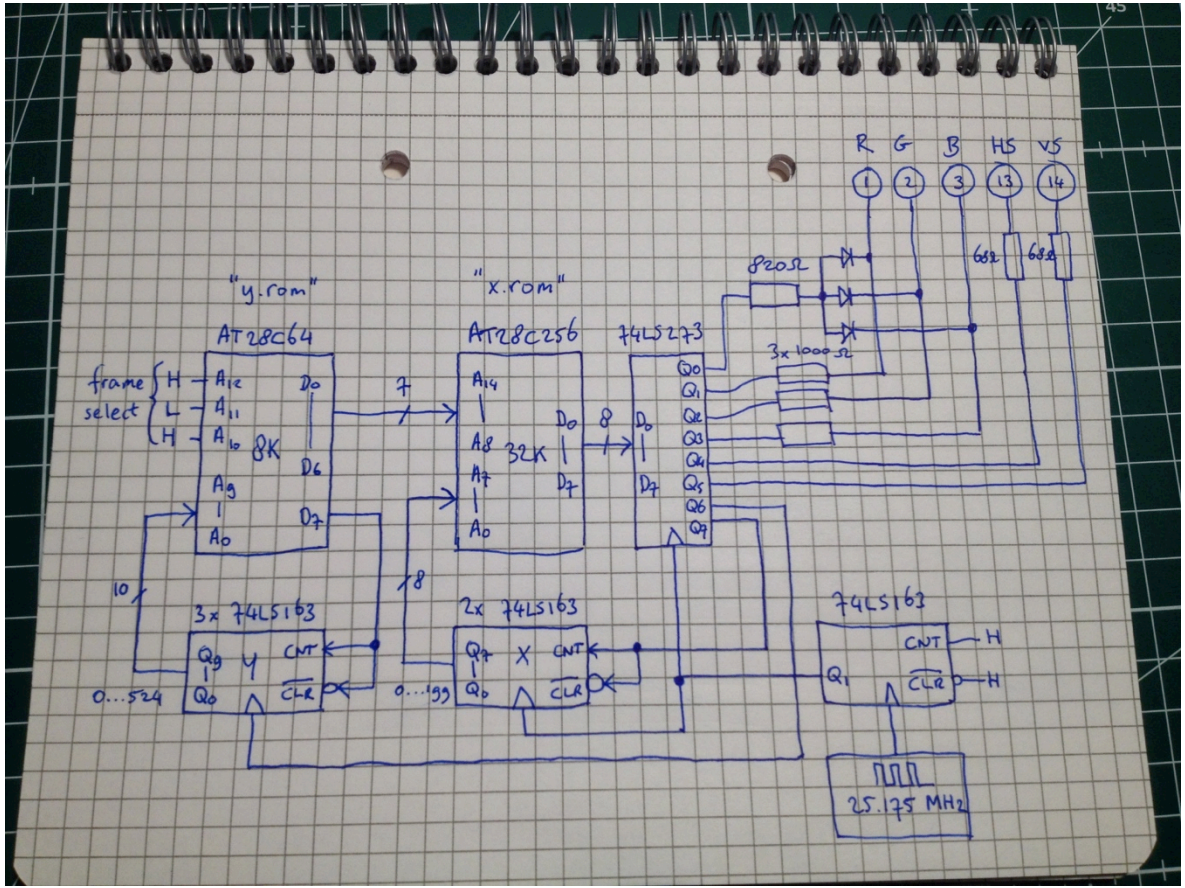
VGA first. Test signals from Arduino Uno



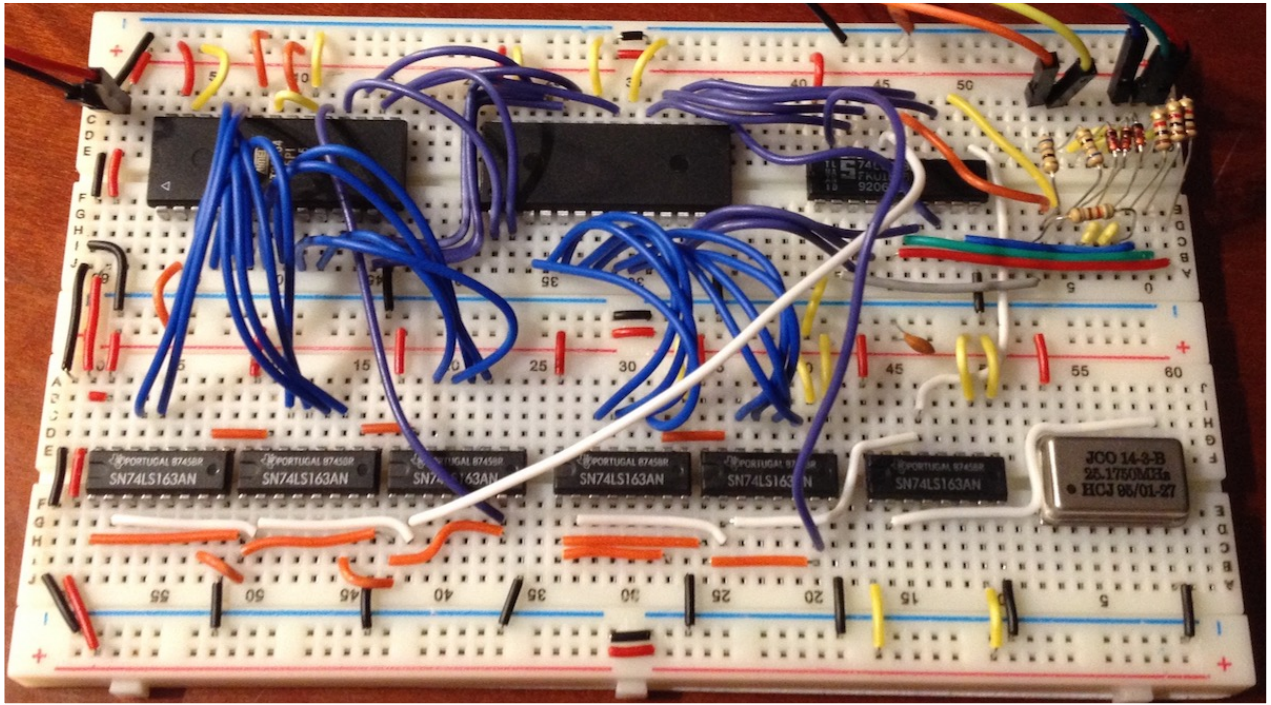




Documenting helps you think



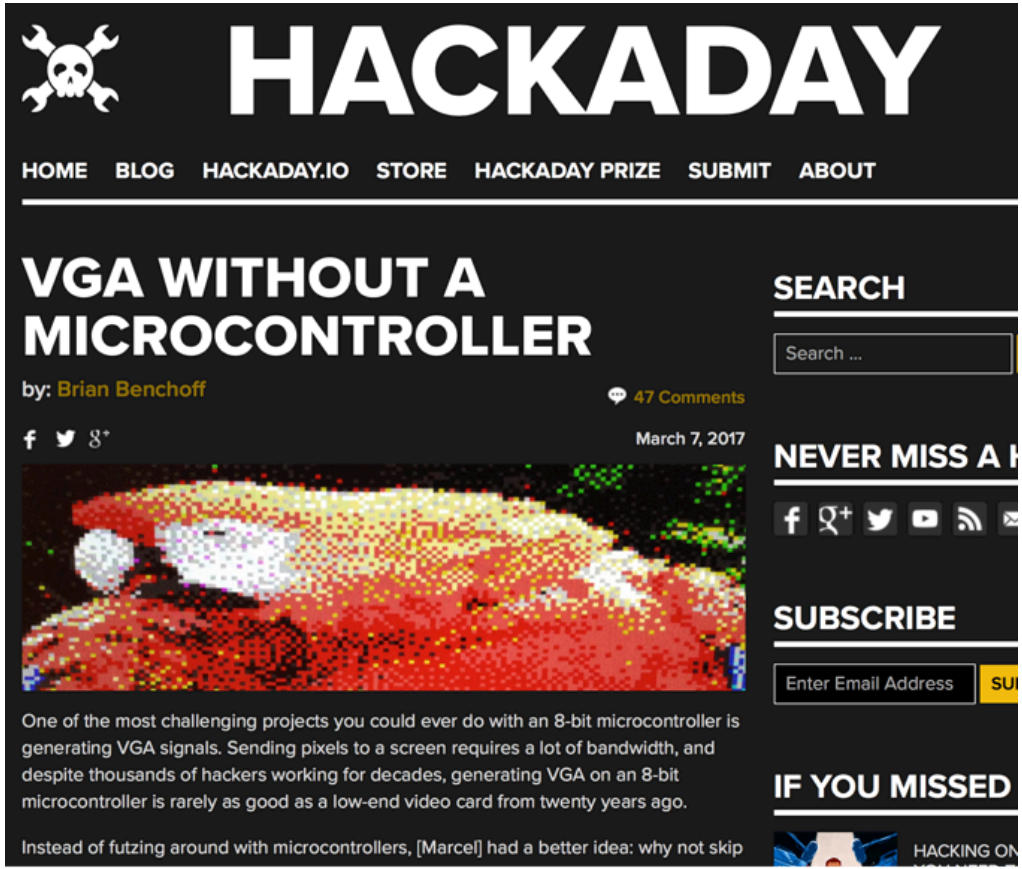
Now remove the microcontroller



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

Look ma, no microcontroller!

Hackaday took notice



HACKADAY

HOME BLOG HACKADAY.IO STORE HACKADAY PRIZE SUBMIT ABOUT

VGA WITHOUT A MICROCONTROLLER

by: Brian Benchoff

47 Comments

March 7, 2017

One of the most challenging projects you could ever do with an 8-bit microcontroller is generating VGA signals. Sending pixels to a screen requires a lot of bandwidth, and despite thousands of hackers working for decades, generating VGA on an 8-bit microcontroller is rarely as good as a low-end video card from twenty years ago.

Instead of futzing around with microcontrollers, [Marcel] had a better idea: why not skip

SEARCH

Search ...

NEVER MISS A

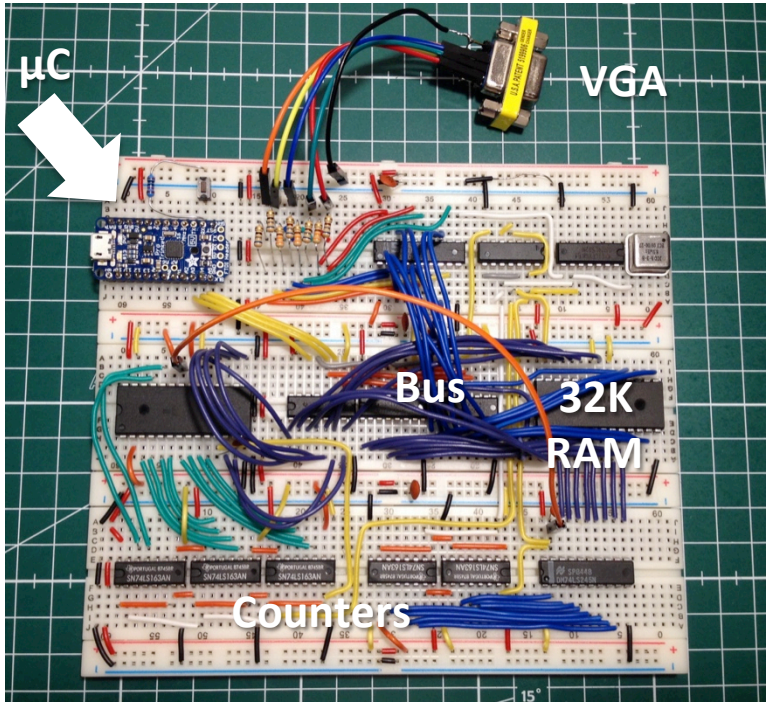
SUBSCRIBE

Enter Email Address

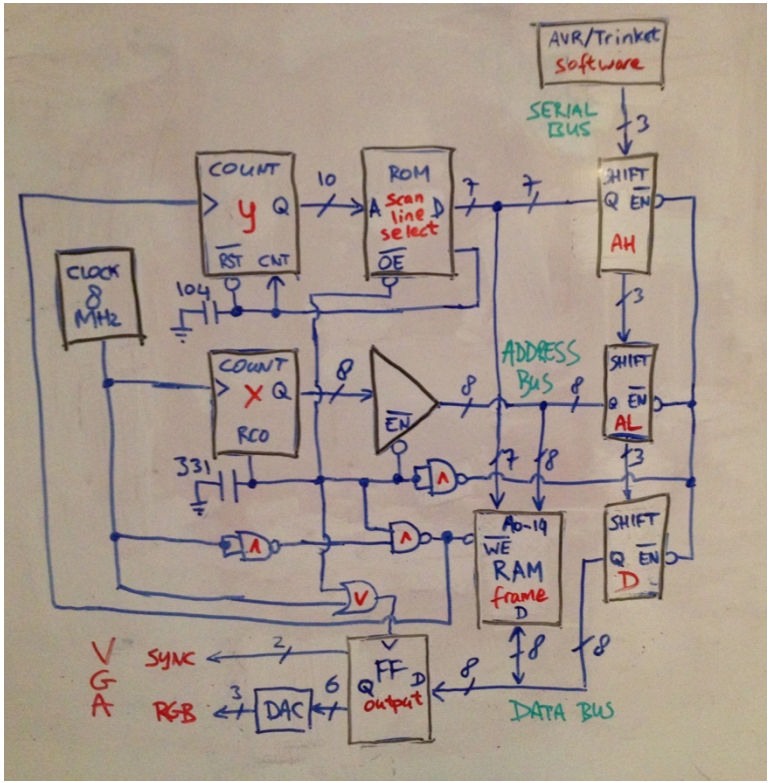
IF YOU MISSED

HACKING ON

Try the same with a RAM (and a microcontroller again)

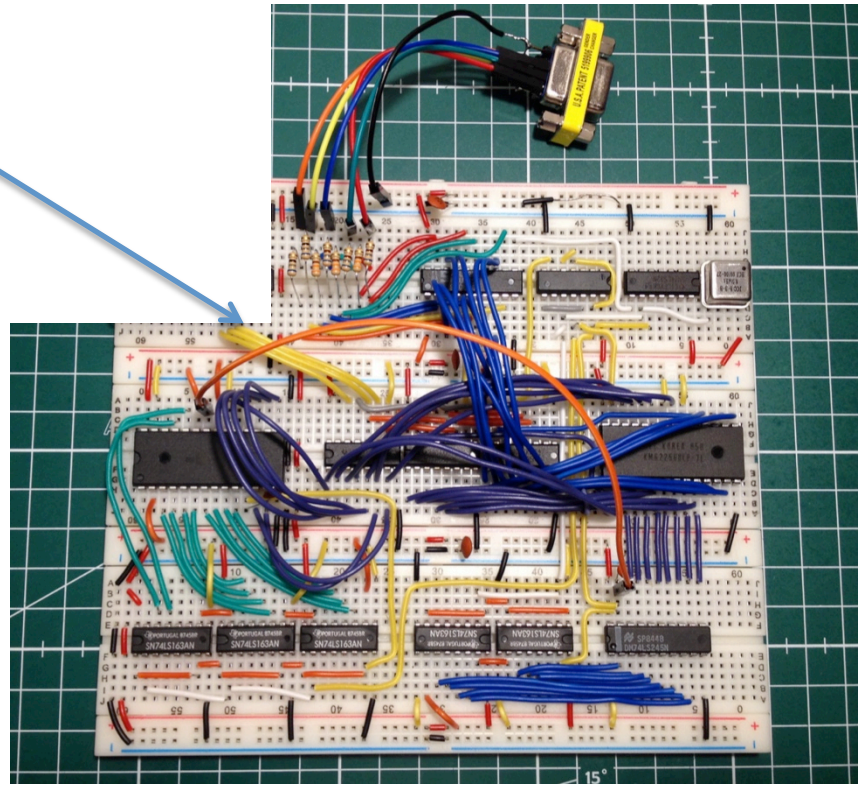


8 MHz breadboard dynamic VGA
from TTL logic and a 32K RAM.
A microcontroller to setup the RAM



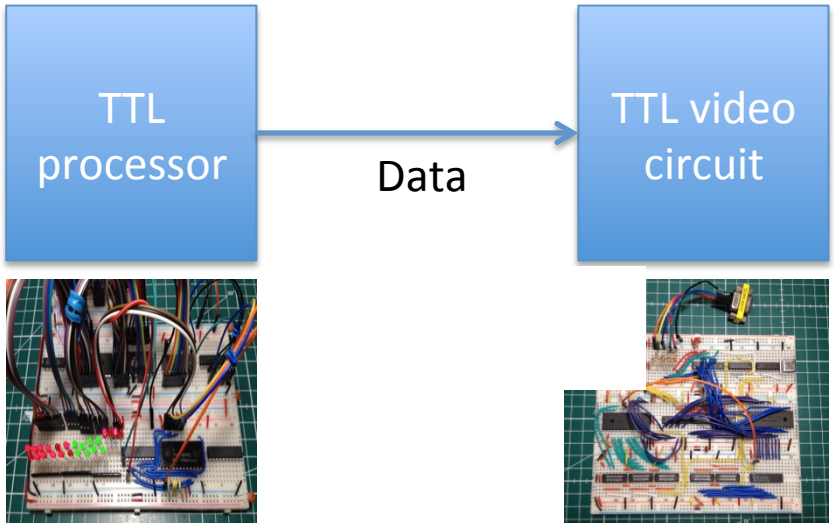
This is basically a video card

Data



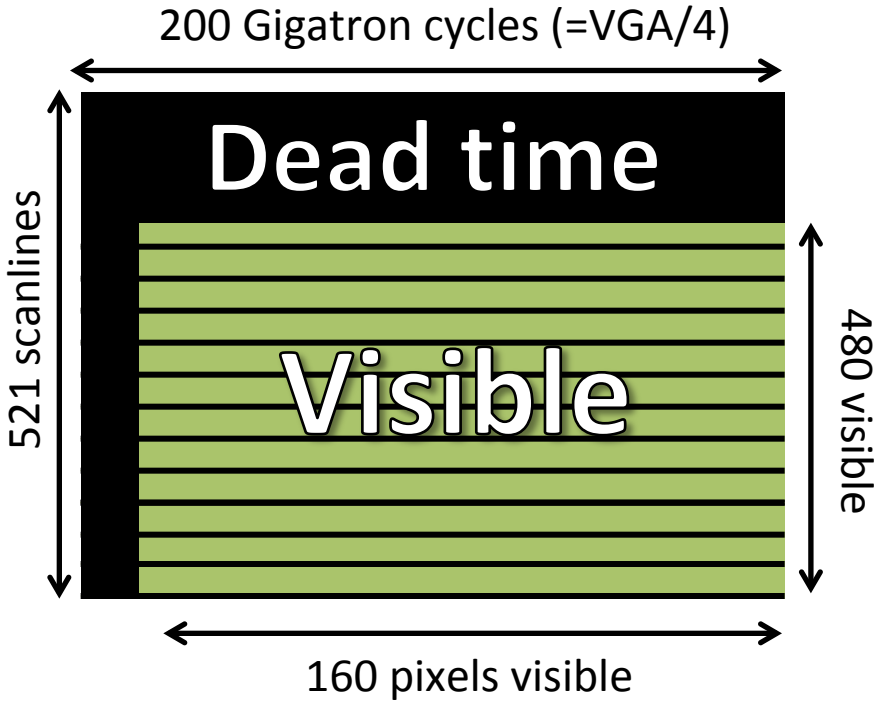
The next design step defines the Gigatron

This is what we wanted to avoid ending up with:

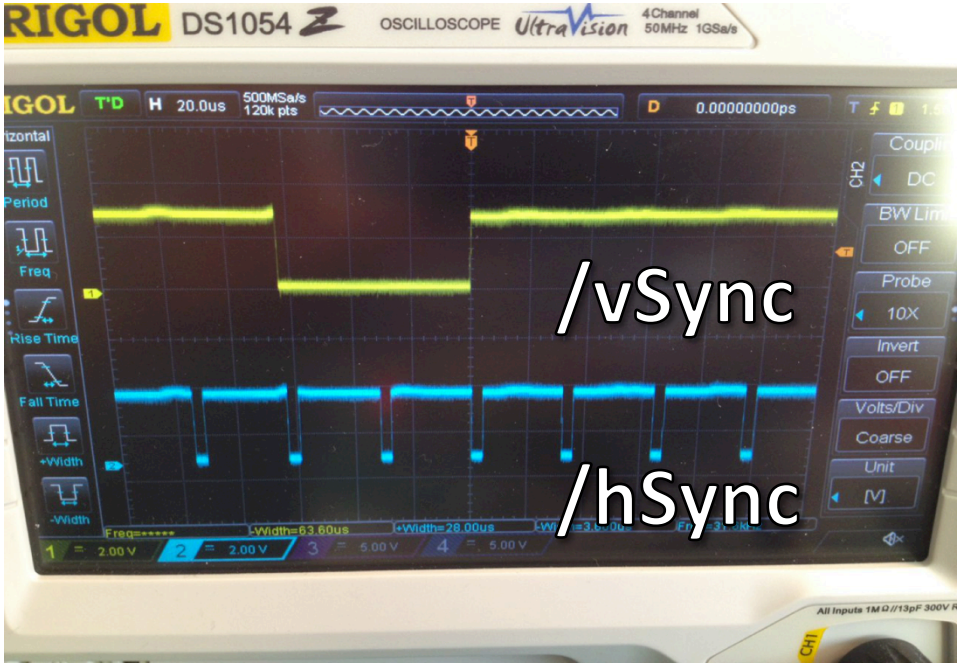


To achieve a low chip count, we must attempt to merge both functions into one

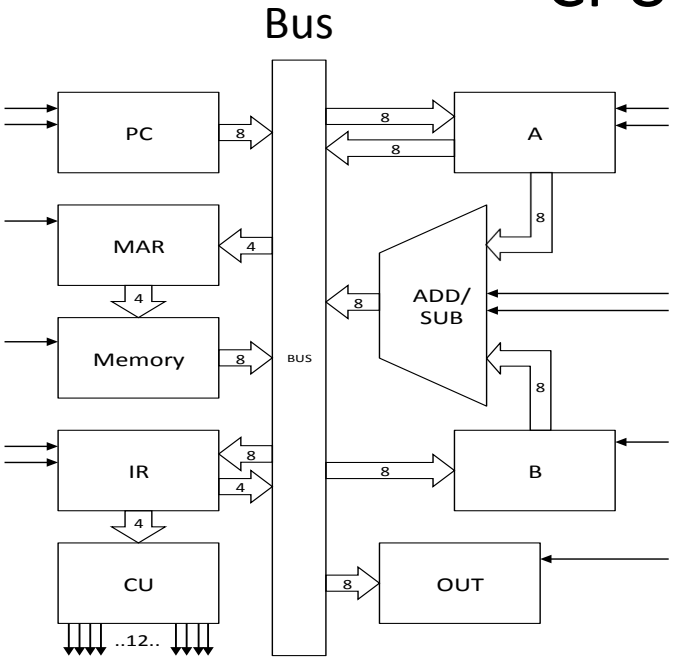
Remember Quark-85: video signals and dead time

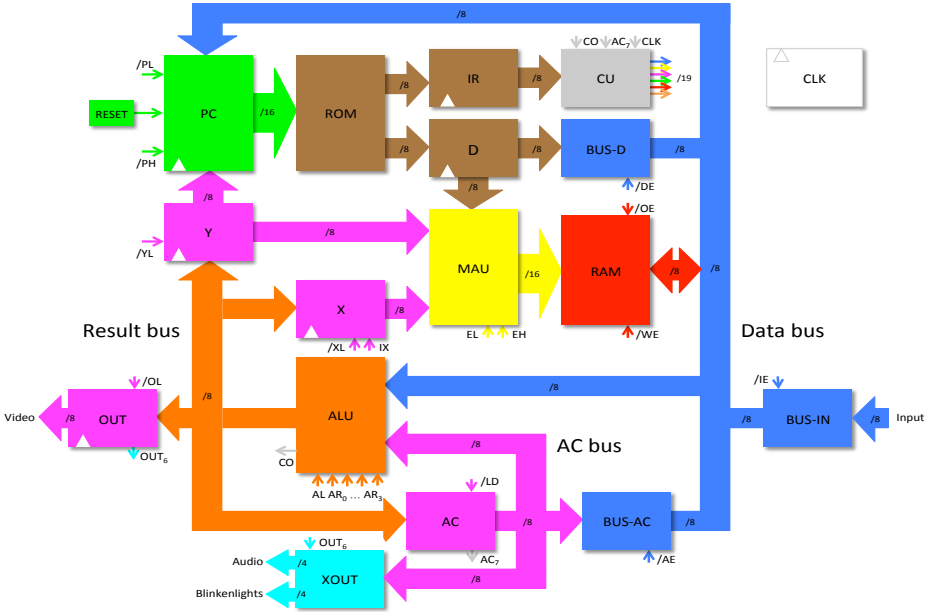


1 video frame
drawn 60 times per second



CPU that can do all



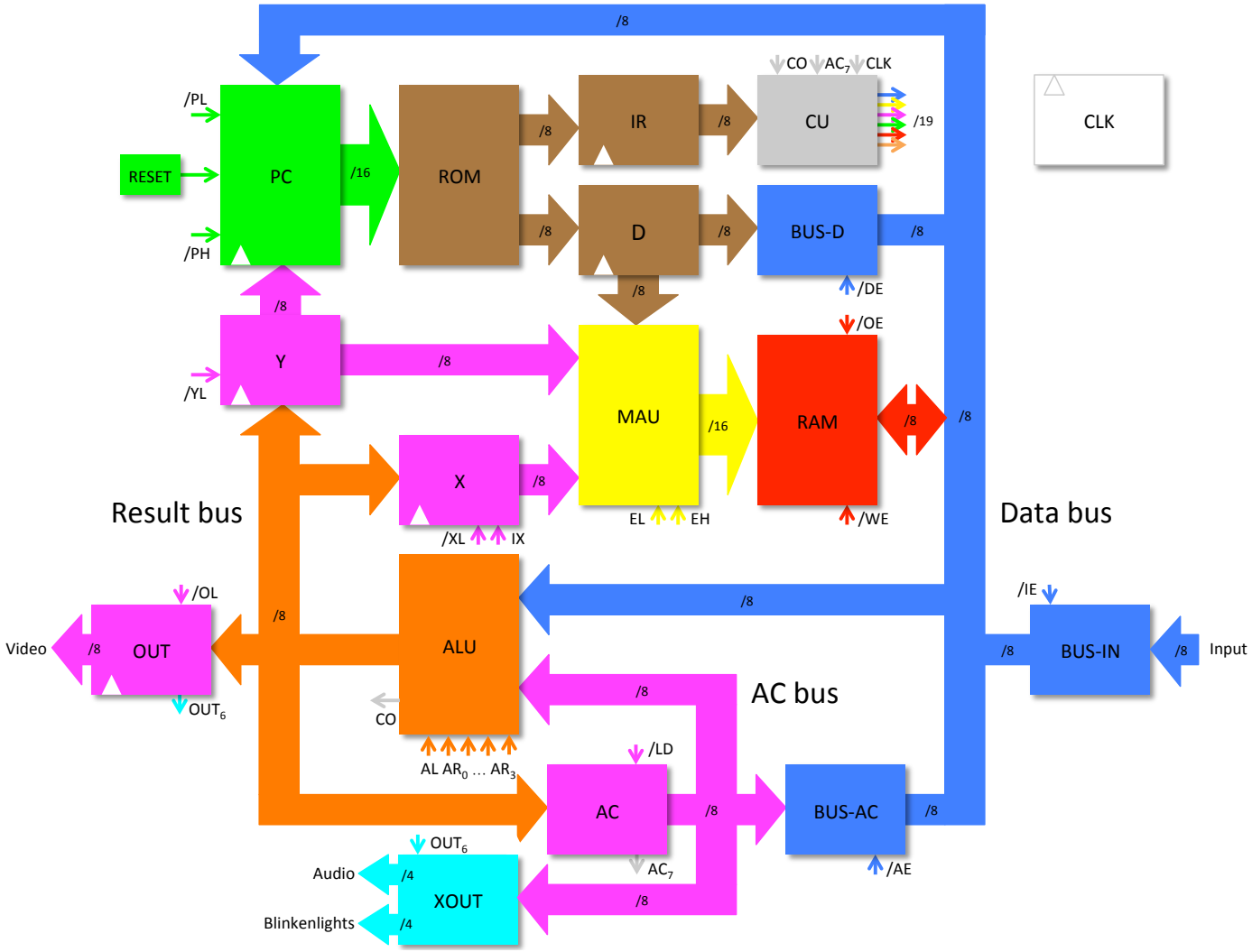


Reference CPU design (SAP-1)

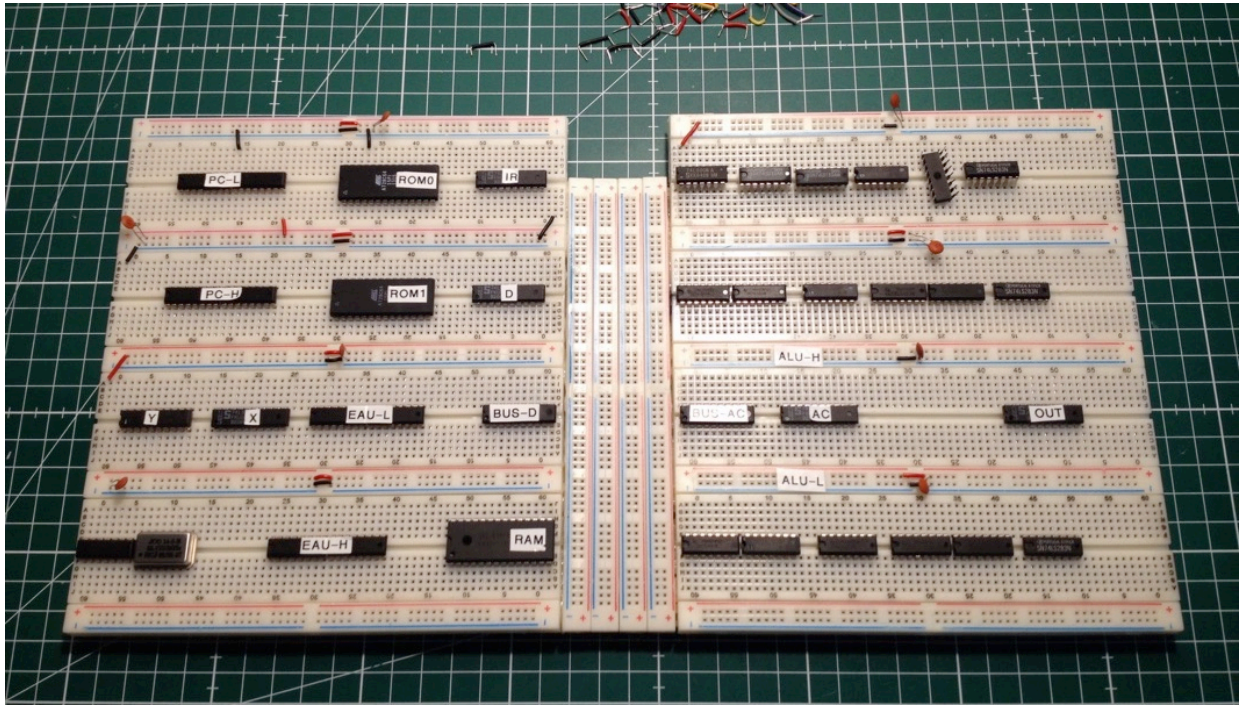
- Von Neumann architecture
- 1 central bus is bottleneck: complexity↑
- Must be microcoded: speed↓↓

Our design

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

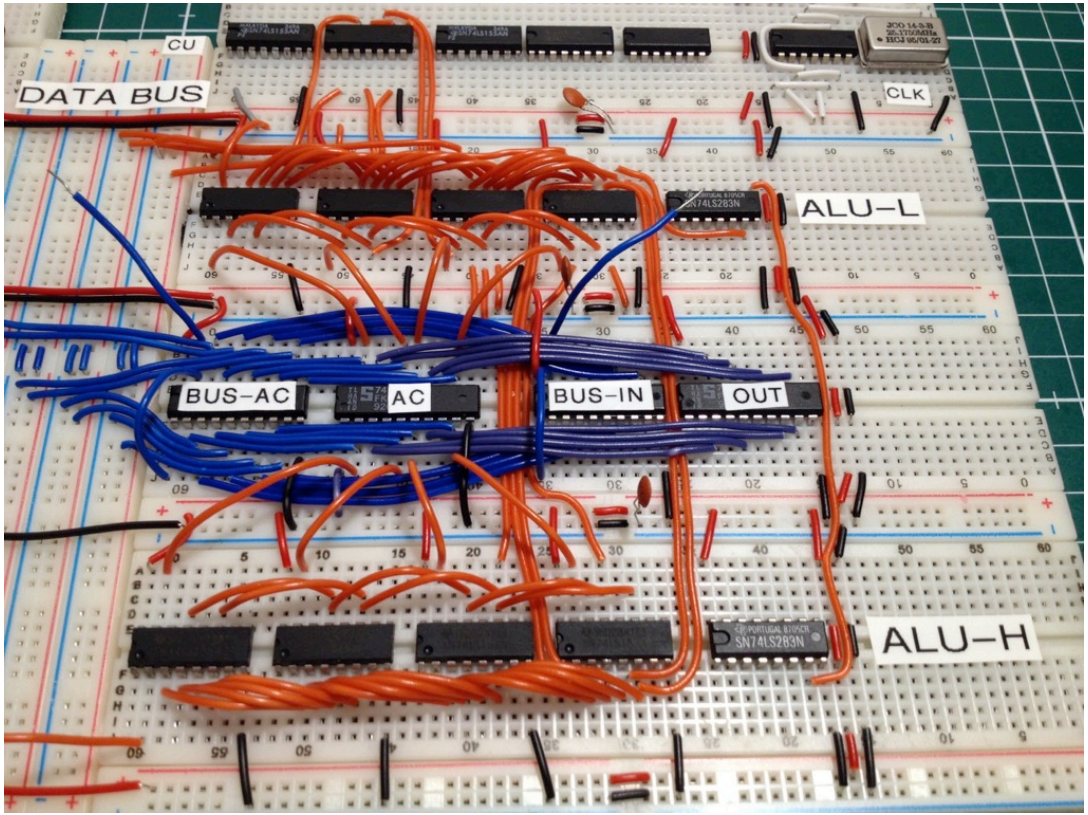

/8
CO
AC7
CLK
CLK
/PL
RESET
/PH
PC
/16
ROM
/8
IR
/8
CU
/19
/8
D
/8
BUS-D
/8
/DE
/8
Y
/YL
/8
MAU
/16
/OE
RAM
/8
/8
/WE
EL
EH
X
/8
/XL
IX
Result bus
Data bus
/OL
Video
/8
OUT
OUT6
/8
ALU
CO
AL AR0 … AR3
/8
AC bus
/IE
/8
BUS-IN
/8
Input
/LD
AC
/8
BUS-AC
/8
/AE
AC7
OUT6
Audio
/4
Blinkenlights
/4
XOUT
/8

2017-03-31: Start building it on a breadboard



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

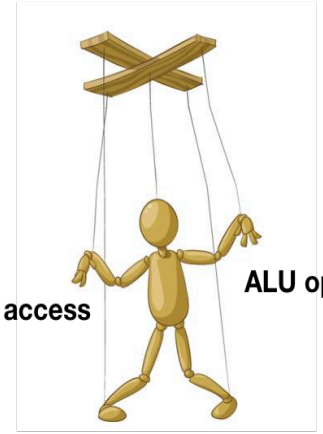
2017-04-08: ALU from multiplexers and adders



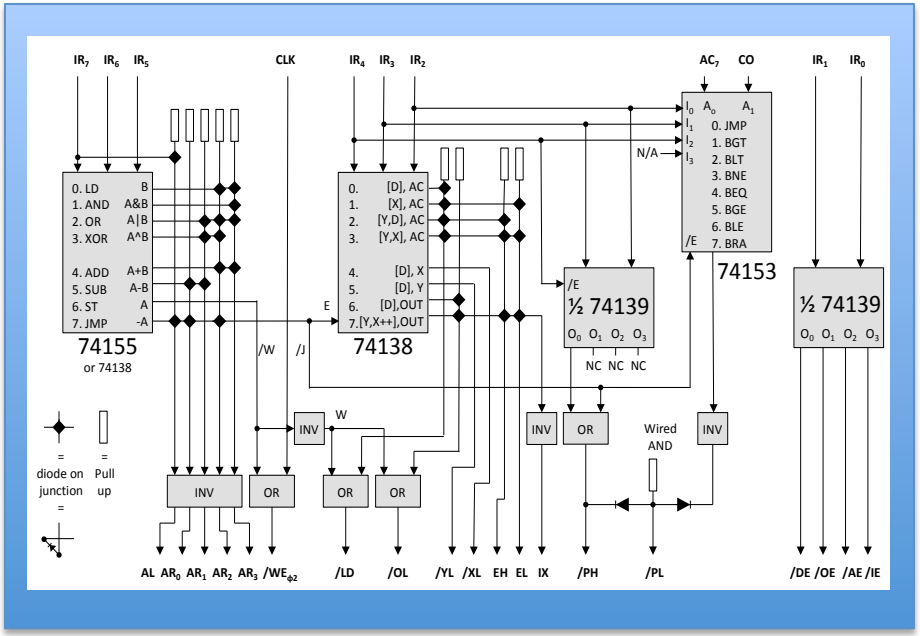
The last detail is … the instruction set

0	1	0	1	0	1	0	0

instruction
fixed length



Map 8 instruction bits …



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

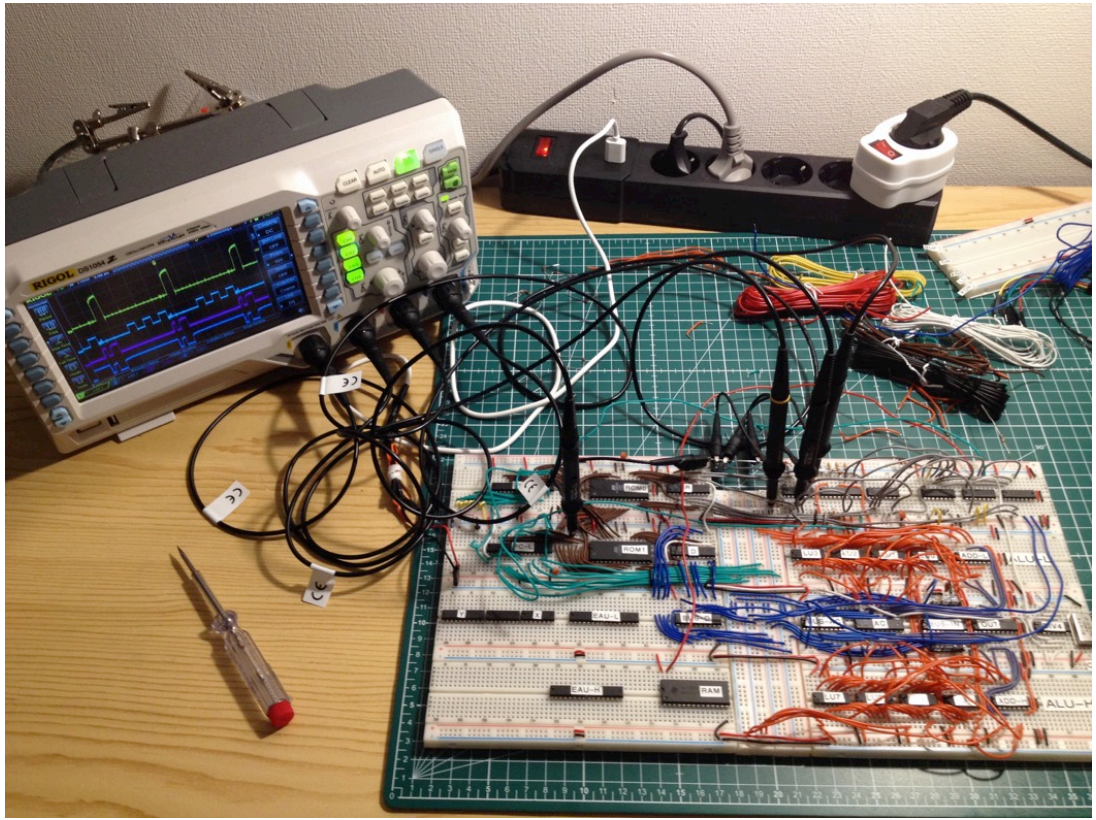
Instruction and mode define what all units do

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

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

2017-05-02: First simple program loop

```
address
|     encoding
|     |     opcode
|     |     |     operand
|     |     |     |
V     V     V     V
0000 0000  ld    $00
0001 0001  ld    $01
0002 0002  ld    $02
0003 0003  ld    $03
0004 0004  ld    $04
0005 0005  ld    $05
0006 0006  ld    $06
0007 fc00  bra   $00
0008 0008  ld    $08
```



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

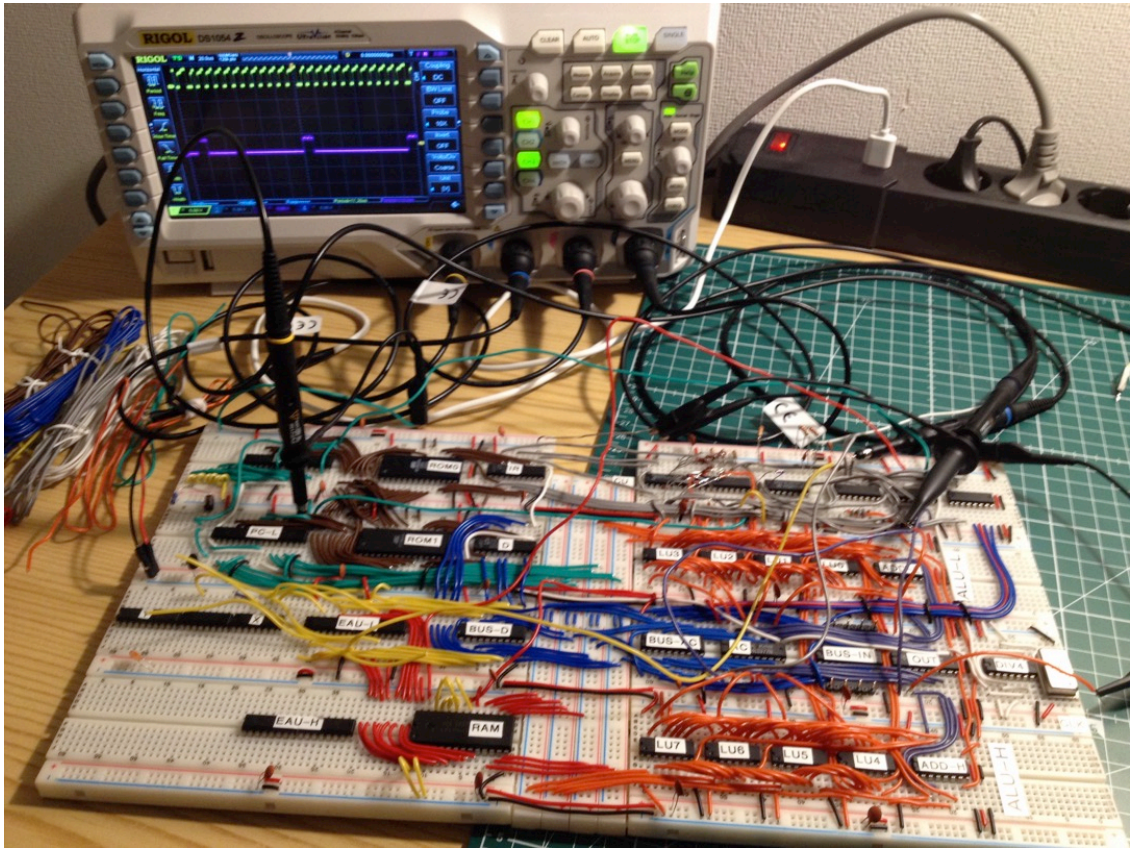
55

89

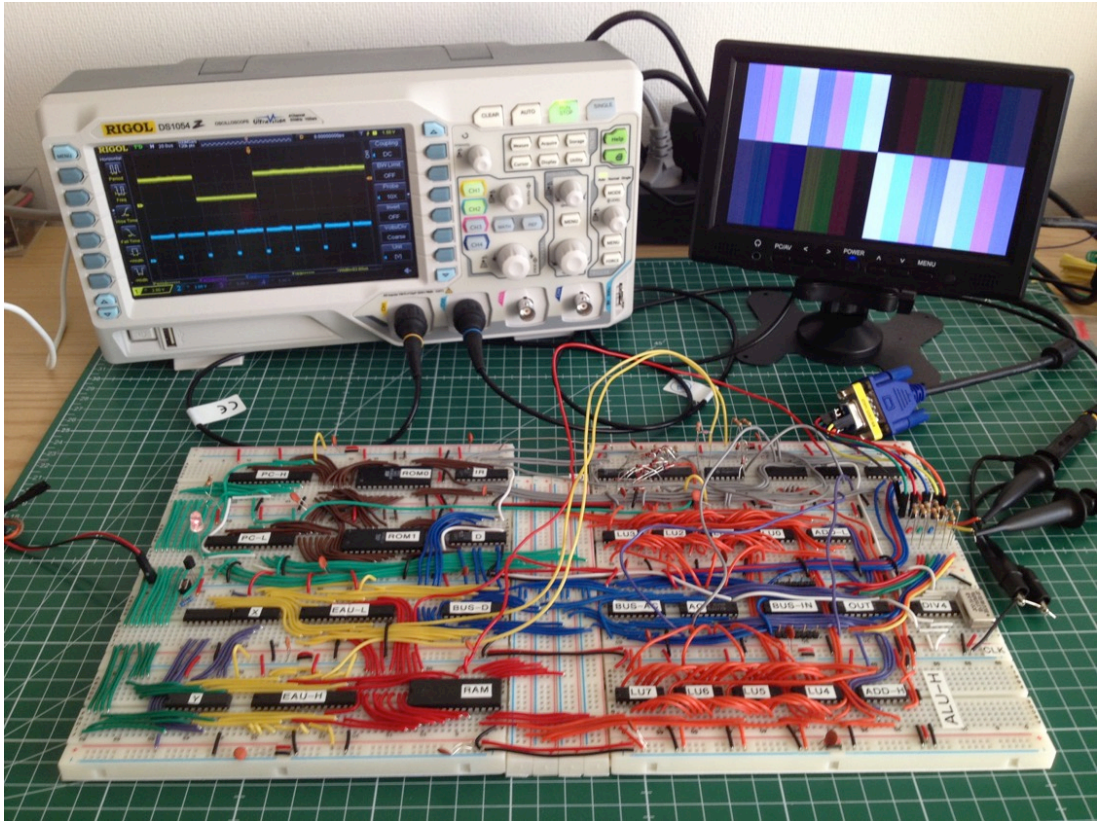
144

…

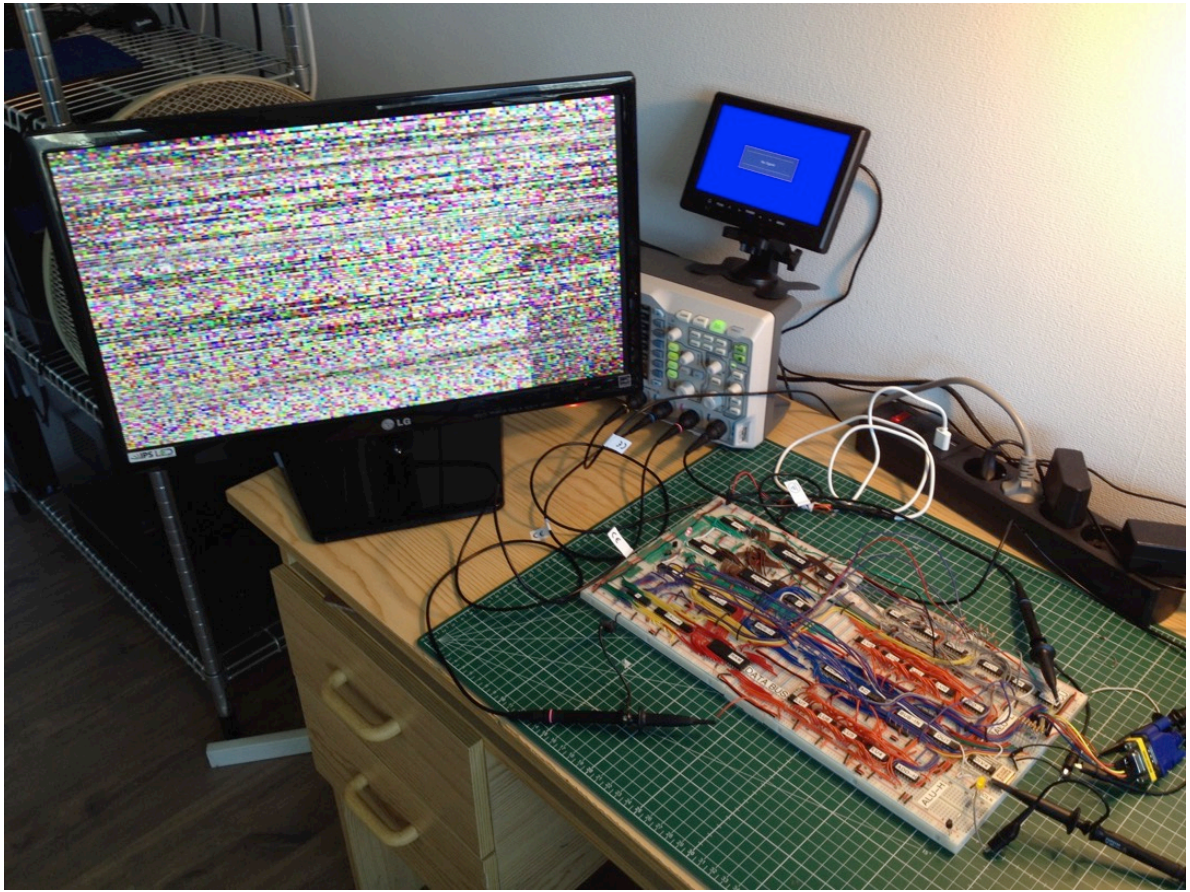
2017-05-11: First Fibonacci series computed



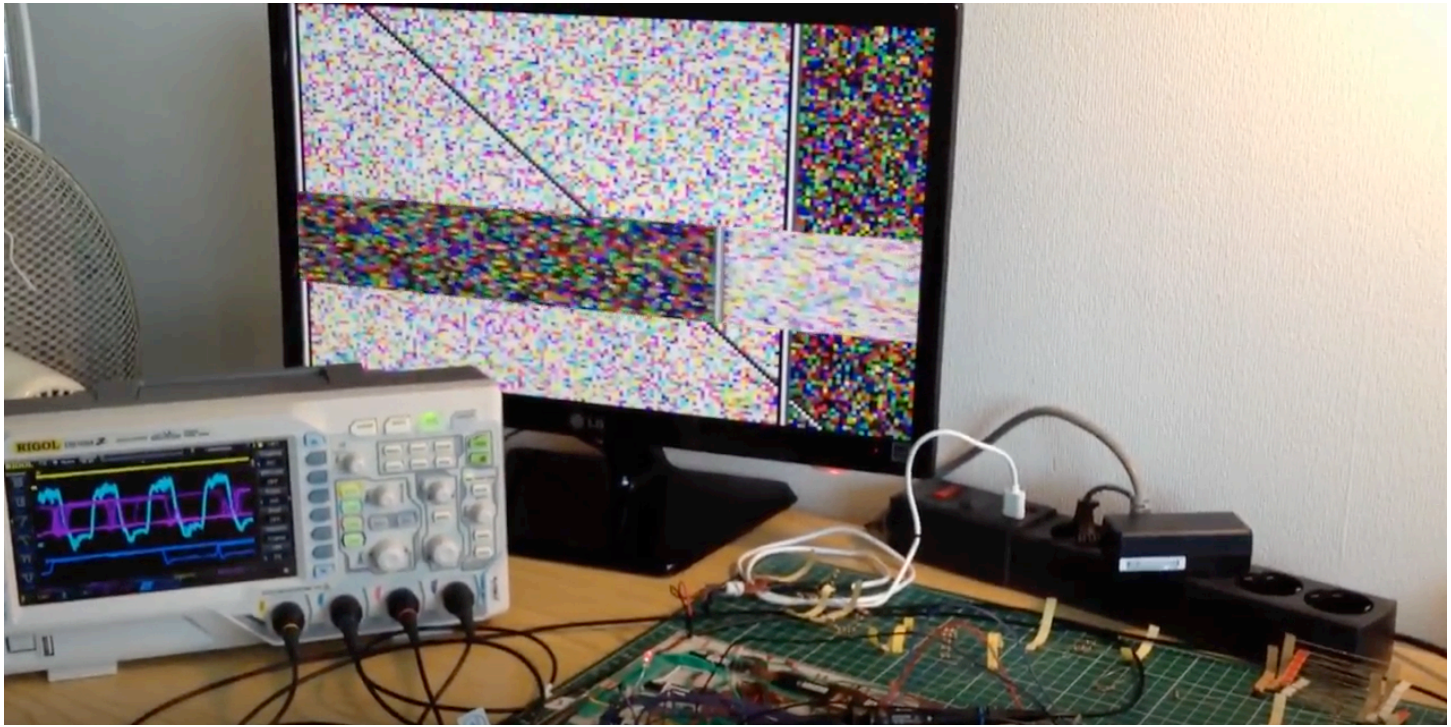
2017-05-13: First video signals from software



2017-05-25: First pixels from RAM



2017-05-28: First moving video

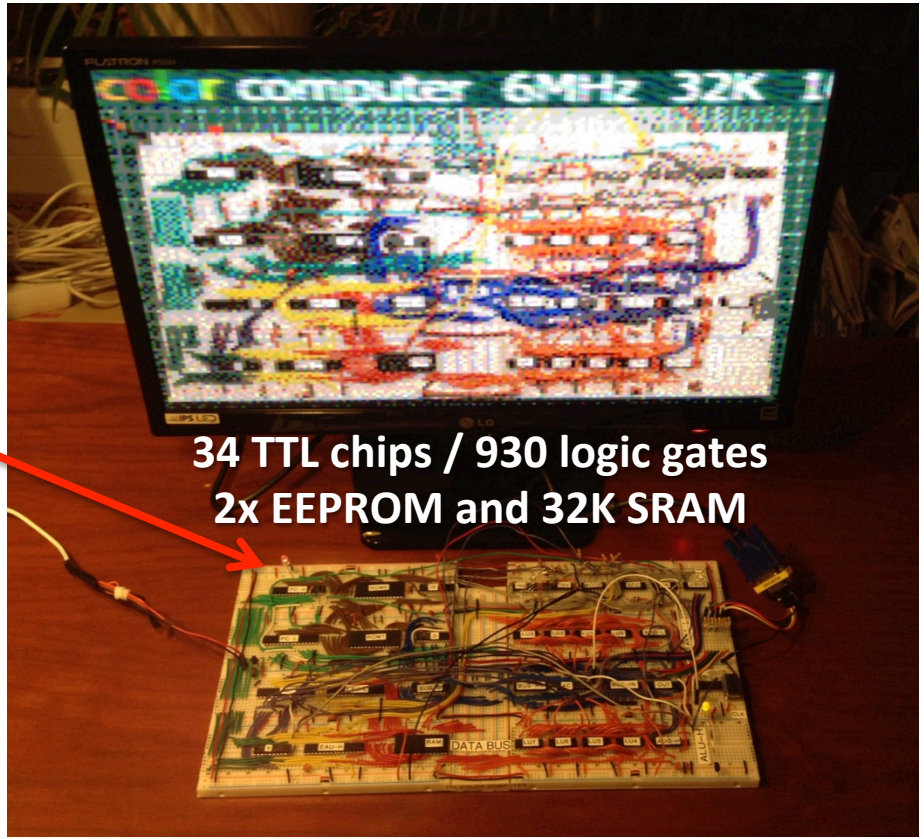


First moving video from my breadboard TTL color computer. Again, the test image is just initialized SRAM garbage with some lines drawn over it.

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

2017-06-13: The breadboard becomes self-aware

And it blinks an LED



And Hackaday covers it again ☺

Minimalism at work

No standard instruction set

No interface adapter chips

No linear address space

No relative addressing

No flags register

No register file

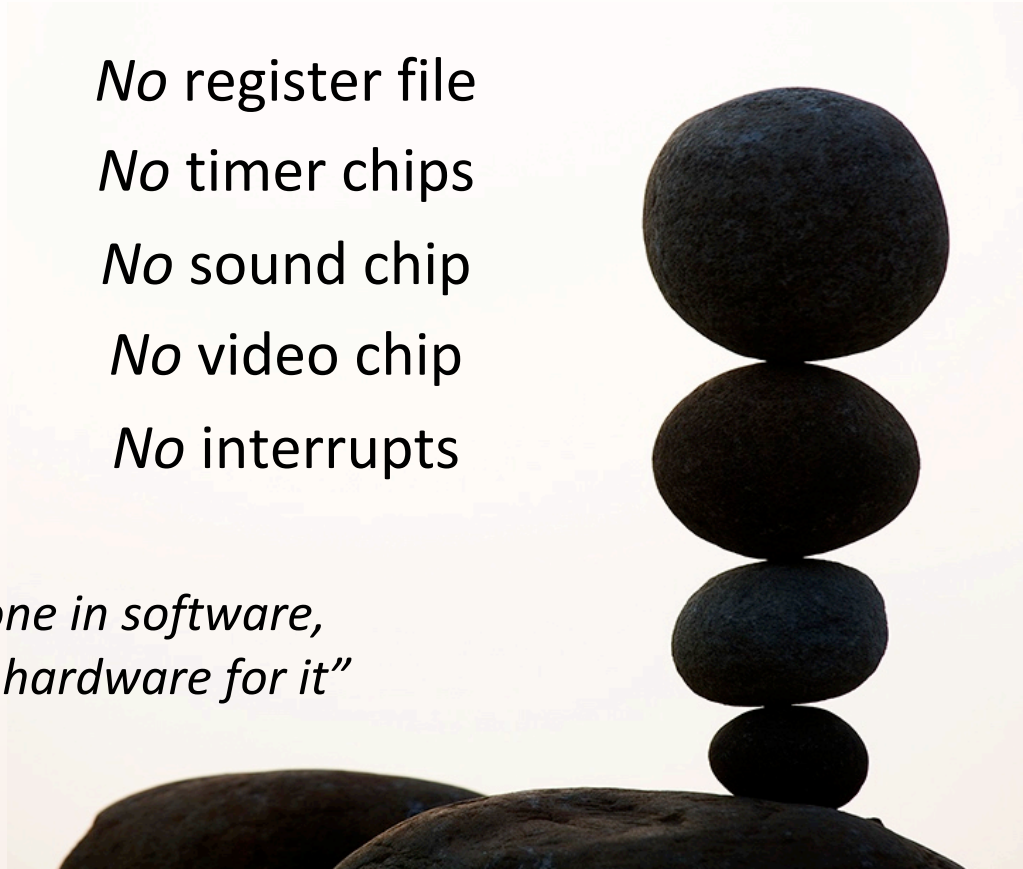
No timer chips

No sound chip

No video chip

No interrupts

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



But how do you program that?

Native code for hardware functions

Bit-bang VGA compatible signals,

4 channel sound, I/O, blinkenlights, reset, ROM as disk,

and applications …

EPROM

Use Python *as* offline assembler

E.g.: def nop(): ROM.append((0x02, 0x00))

Not an assembler *in* Python!

🤭 Python syntax for assembly

😀 Get a macro assembler for free

16 instructions, 8-bits

Memory load/store:	`ld st`
Logical operations:	`anda ora xora`
Arithmetic operations:	`adda suba`
Unconditional jumps:	`jmp bra`
Conditional jumps:	`bgt beq bge`
	`blt bne ble`
No operation:	`nop`

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.

Need a higher abstraction 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
```

On system

High level

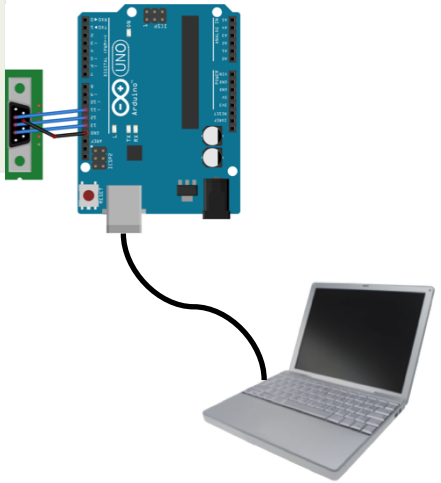
vCPU runs application code <u>from RAM</u>

- SWEET16 inspired, Von Neumann!
- 34 self-timed instructions
- Can mix with native 8-bit code
- Load from ROM disk or input port

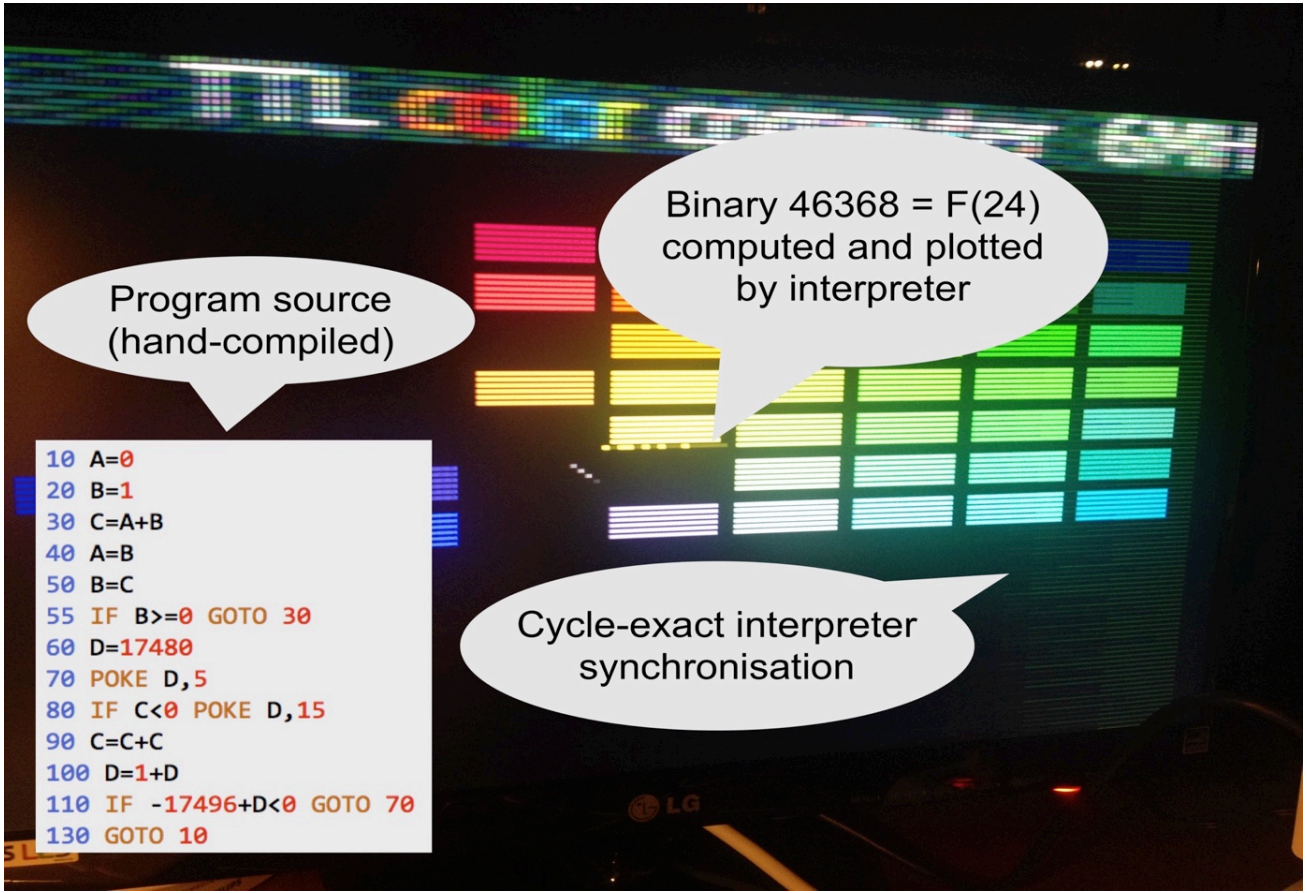
Low level

Native code for hardware functions

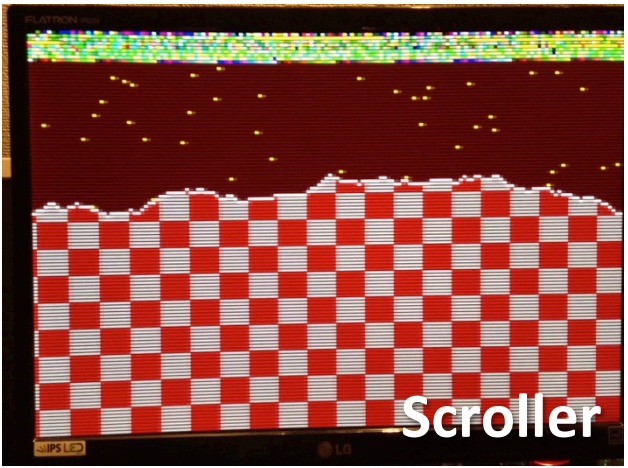
- Bit-bang VGA compatible signals,
- 4 channel sound, I/O, blinkenlights, reset, ROM as disk, and …
- an interpreted virtual CPU: **vCPU**



2017-11-12: vCPU interpreter works



Some first programs we wrote with vCPU



Wise people stop here

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

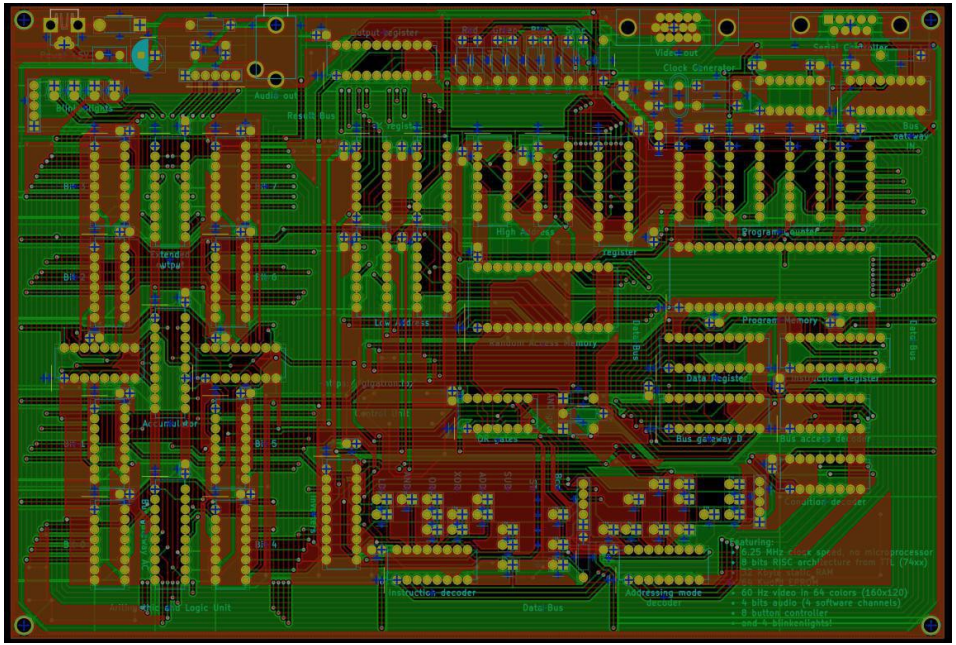
Oscar "Obsolescence Guaranteed" Vermeulen

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

- Focus all efforts on 1st time right builds: assembly manual, videos, website
- The hardest part: stop working on new features for a while
- Talk a lot with other kit makers and potential users
- Find suppliers for quality parts
- All details matter
- Run beta-tests

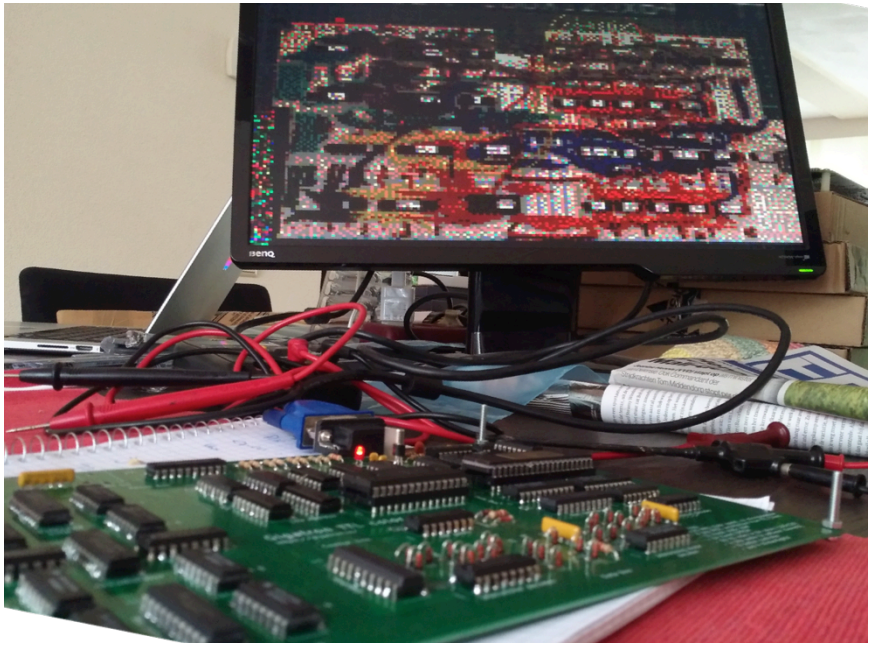
Next phase: learn to make a printed circuit board

About 10 weeks work

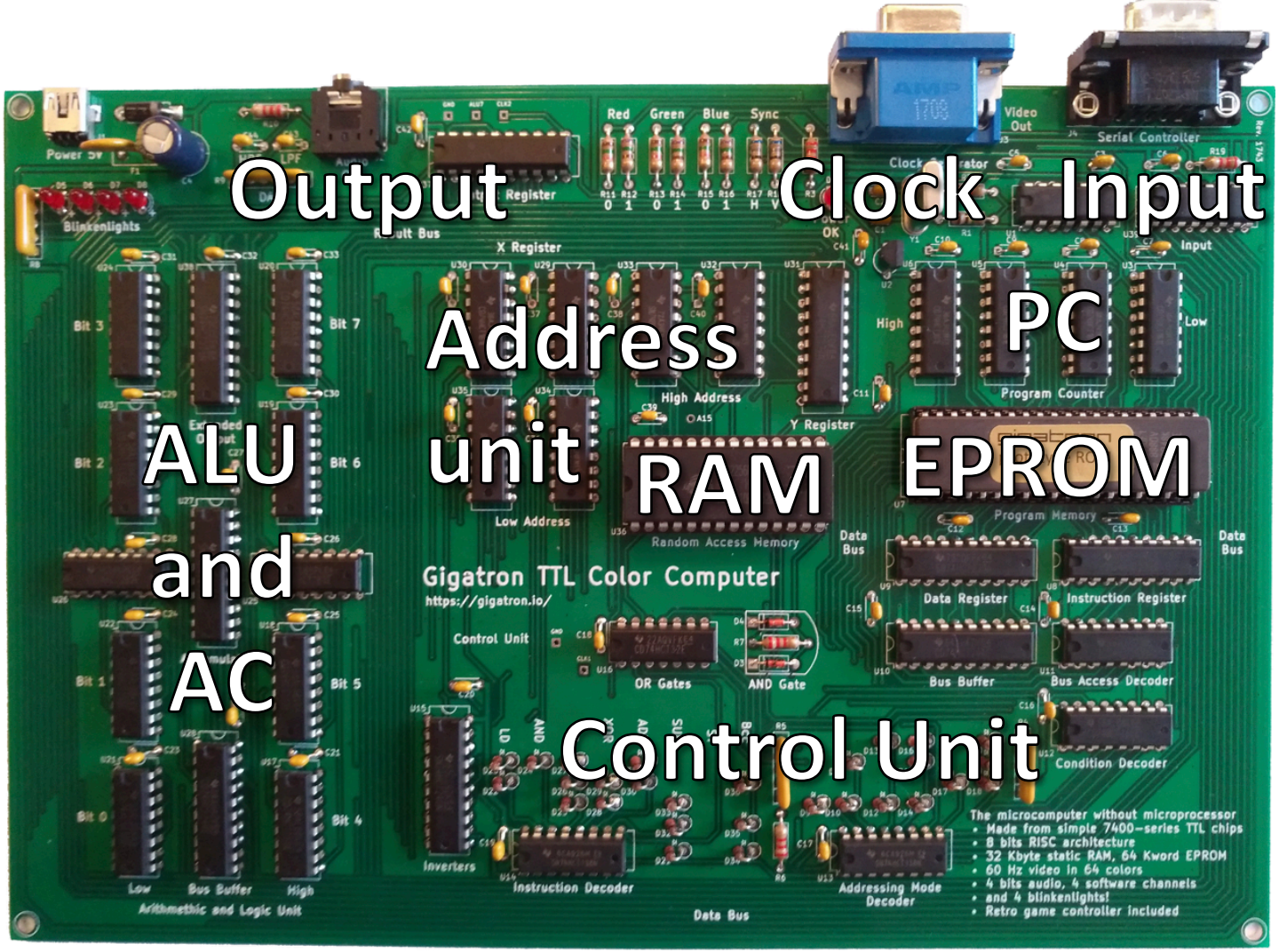


Done in Kicad4

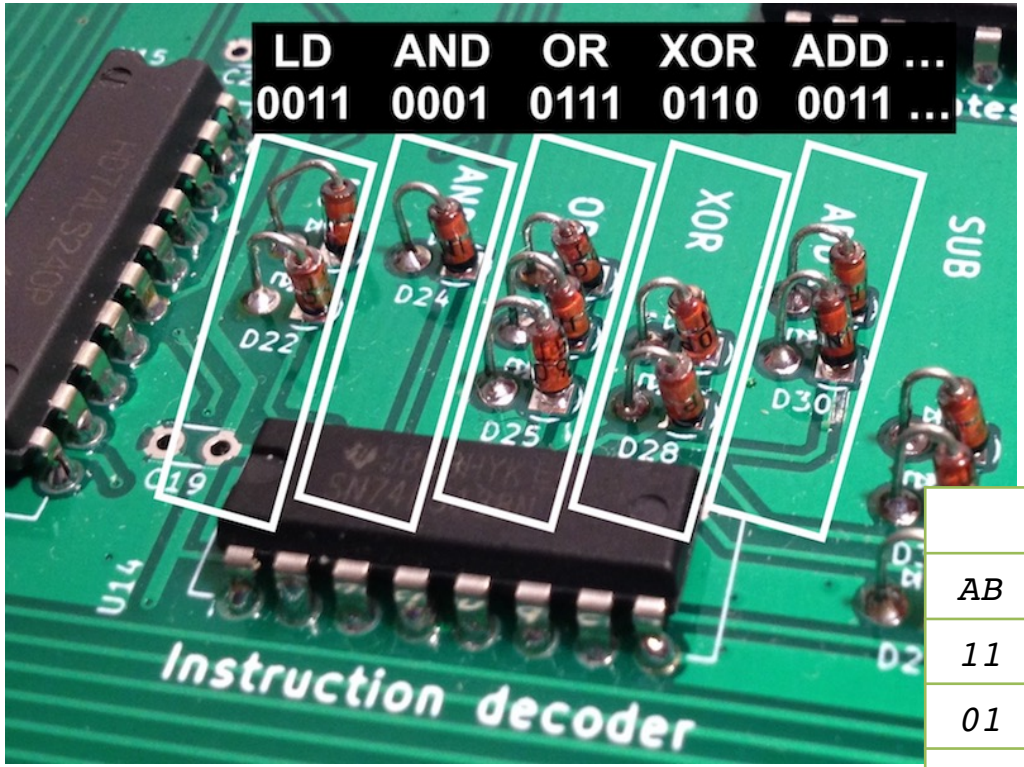
First one could be brought to life \o/



PCB displaying an image of its prototype


Output
Clock
Input
Address
unit
PC
ALU
and
AC
RAM
EPROM
Control Unit
Gigatron TTL Color Computer
https://gigatron.io/
Red
Green
Blue
Sync
Video Out
Serial Controller
Power 5V
Blinkenlights
Result Bus
X Register
High Address
Low Address
Y Register
Program Counter
Random Access Memory
Program Memory
Data Bus
Data Register
Instruction Register
Bus Buffer
Bus Access Decoder
Condition Decoder
Control Unit
OR Gates
AND Gate
Inverters
Instruction Decoder
Addressing Mode Decoder
Arithmetic and Logic Unit
Bit 0
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
Low
Bus Buffer
High
The microcomputer without microprocessor
• Made from simple 7400-series TTL chips
• 8 bits RISC architecture
• 32 Kbyte static RAM, 64 Kword EPROM
• 60 Hz video in 64 colors
• 4 bits audio, 4 software channels
• and 4 blinkenlights!
• Retro game controller included

Manually routing for interesting layout

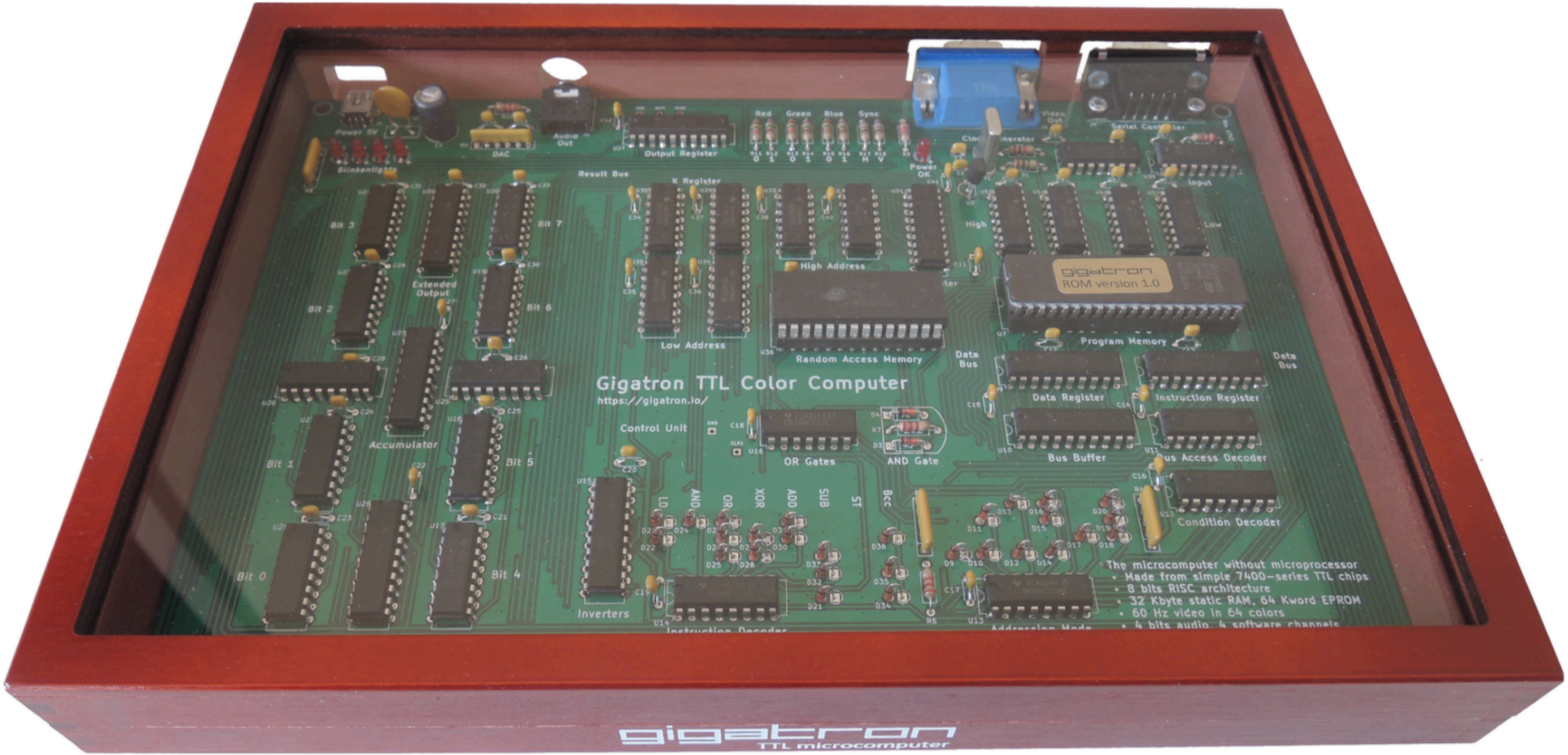


For example: here the diodes visualize the truth tables for each operation

No diode = 0, Diode = 1

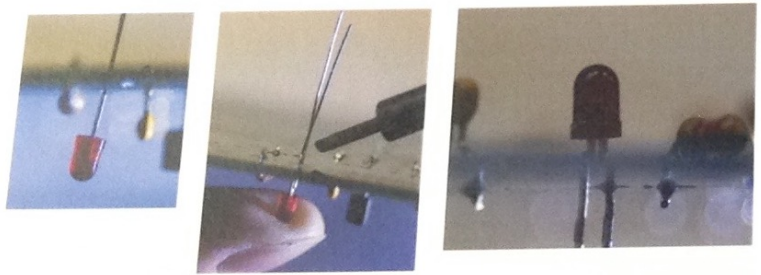
	LD	AND	OR	XOR	ADD	SUB
AB	B	A∧B	A∨B	A≠B	B	~B
11	**1**	**1**	**1**	0	**1**	0
01	**1**	0	**1**	**1**	**1**	0
10	0	0	**1**	**1**	0	**1**
00	0	0	0	0	0	**1**

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

Buying parts

Tedious logistics

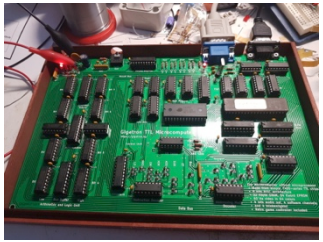
Computer as a DIY soldering kit

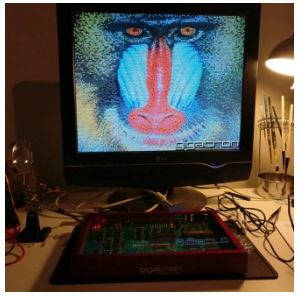
Just need a soldering iron, a multi-meter and 3-4 hours of time to build. No oscilloscope needed

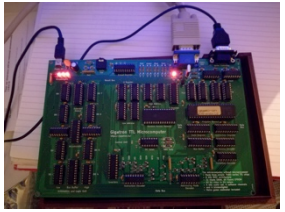
gigatron

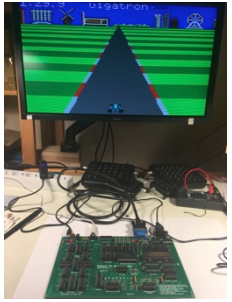
TTL microcomputer

A new one is born every day now

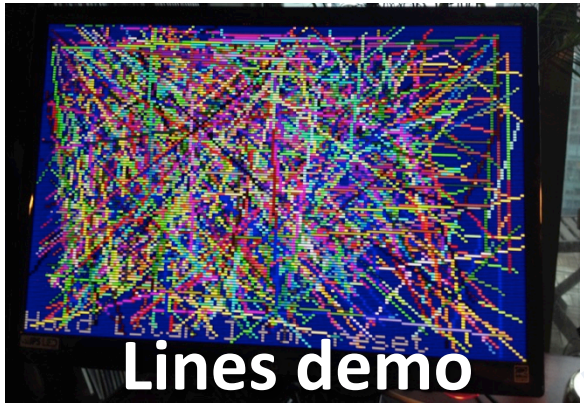


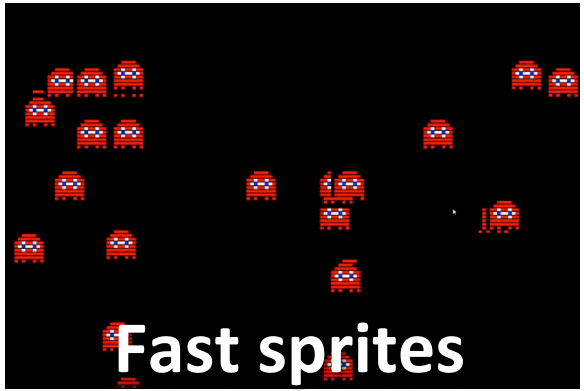


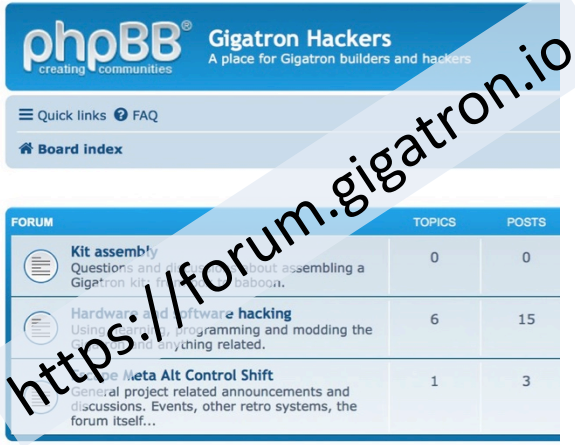


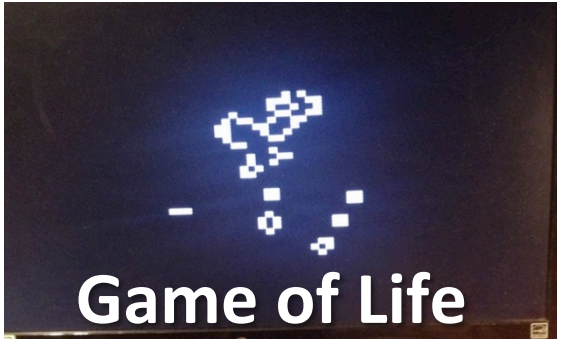


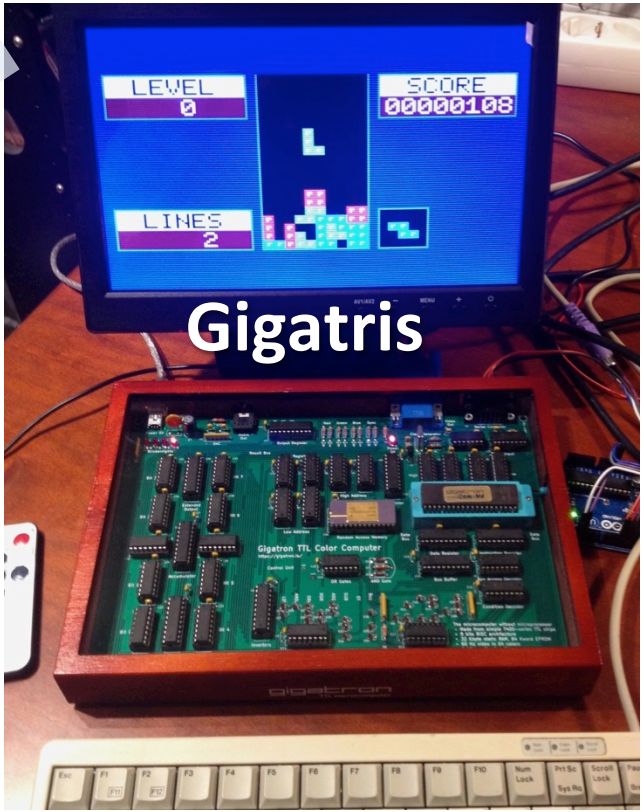
Community after first month











YouTube spreads the word

Not all ideals made it into “v1”

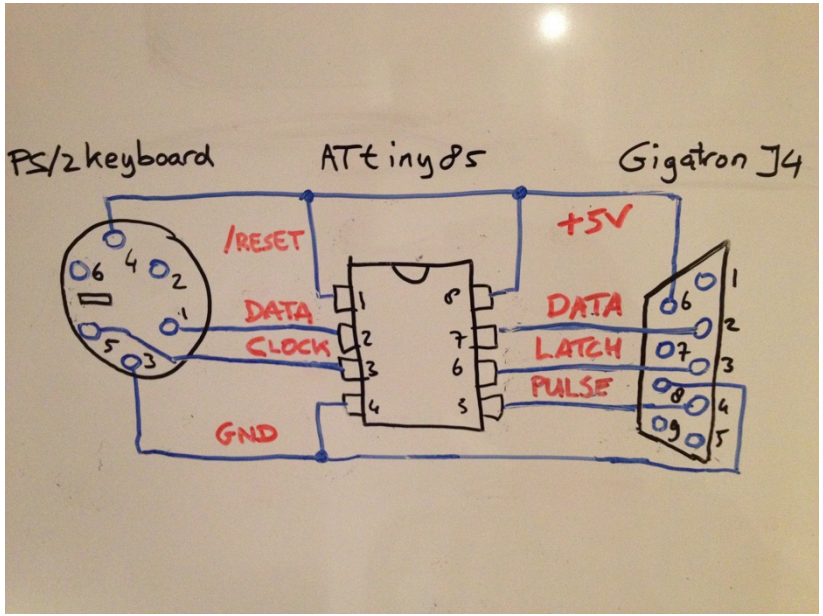
There was no keyboard hookup

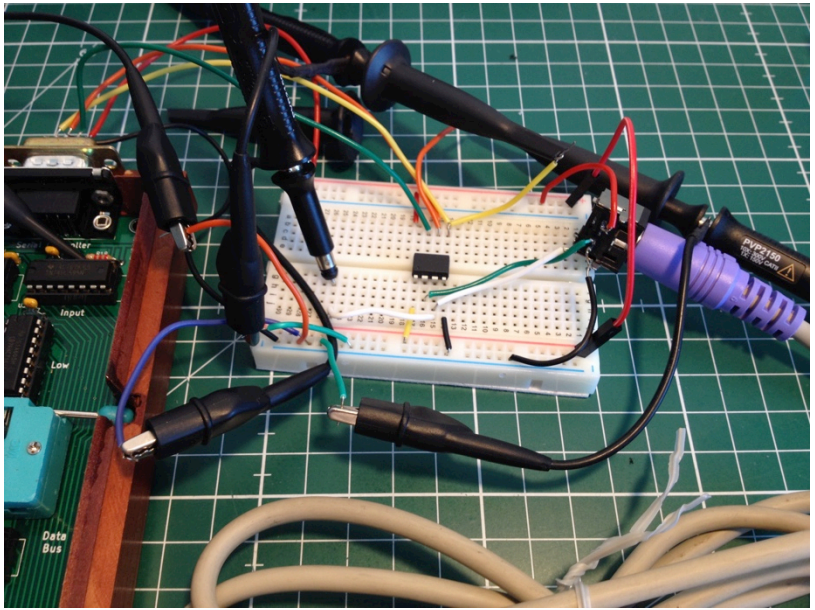
Even the PS/2 protocol turns out to be an ugly beast.

There was no built-in BASIC

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

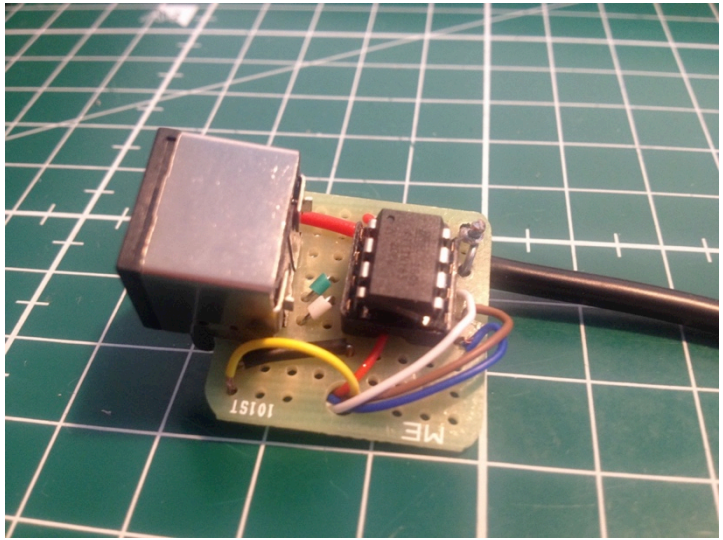
2018-06-05 So we carried on



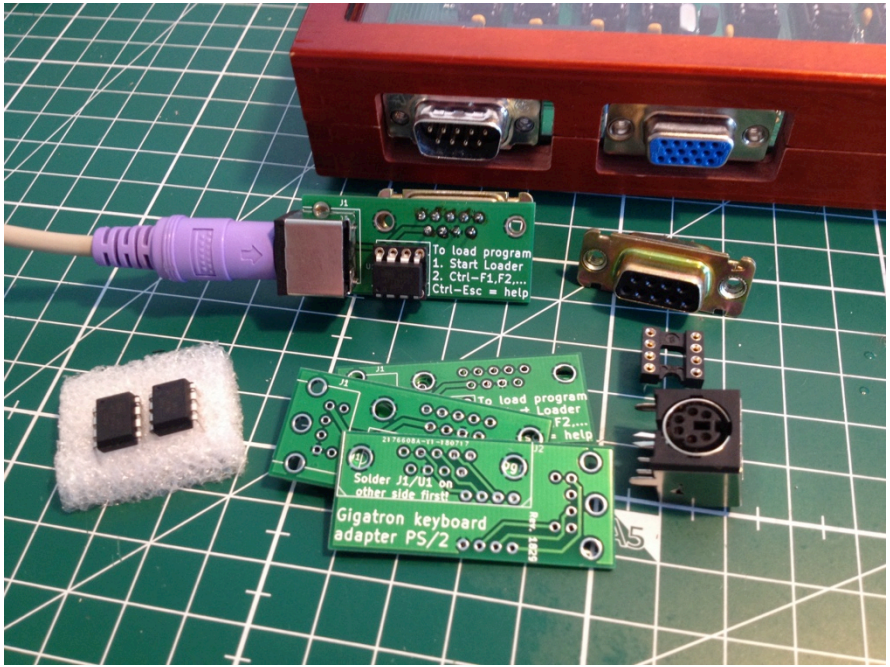


Can you can guess what this is?

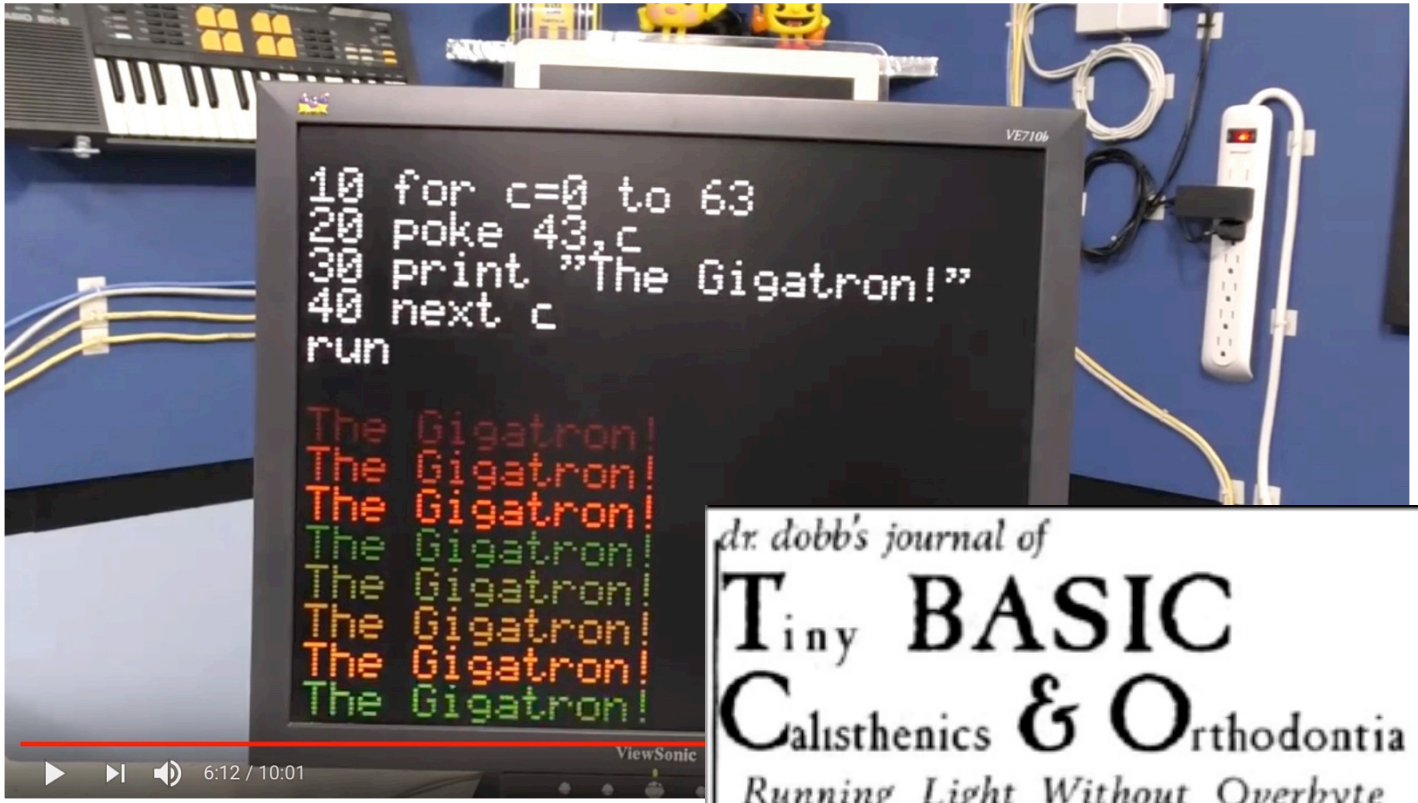
2017-06-07 Prototype PS/2 keyboard adapter



2018-07-24 Pluggy McPlugface



This summer we ported Tiny BASIC to the Gigatron



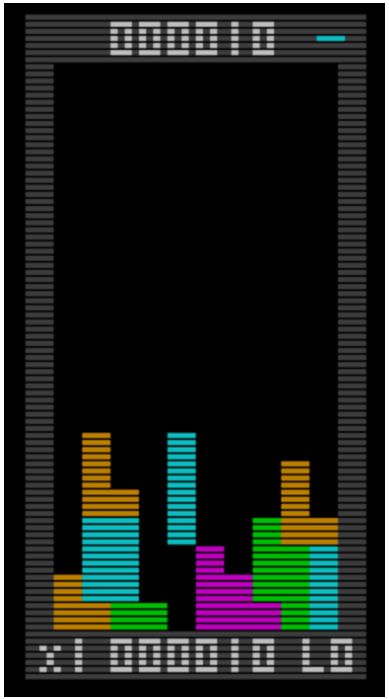
Fibonacci (again), in BASIC

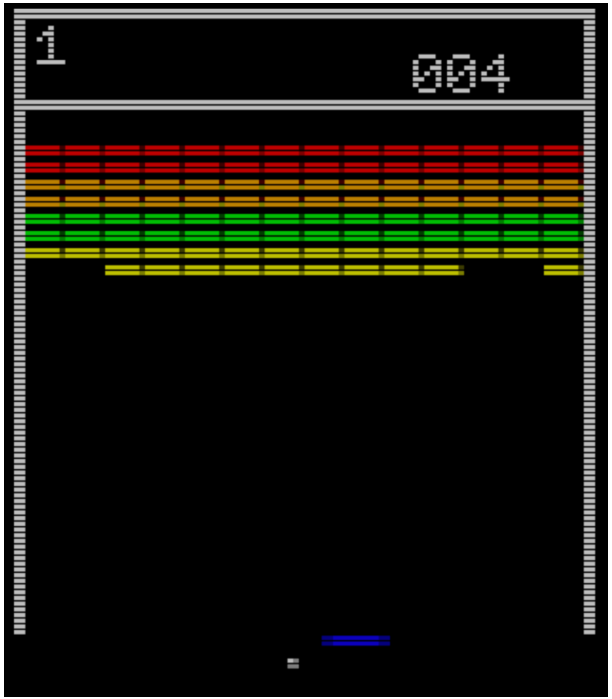
```
1 'Fibonacci with bignums
2 a(0)=0:b(0)=1:c(0)=0
3 n=n+1:printn;": ";
4 fori=0toj:put48+b(j-i)
5 nexti:print:fori=0toj
6 c=a(i)+b(i)+c:a(i)=b(i)
7 b(i)=c:ifc<10 c=0:goto9
8 b(i)=c-10:c=1
9 nexti:ifc=0 goto12
10 j=j+1
11 a(j)=0:b(j)=c:c(j)=0
12 print:goto 3
9120 bytes free
Ok
```

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



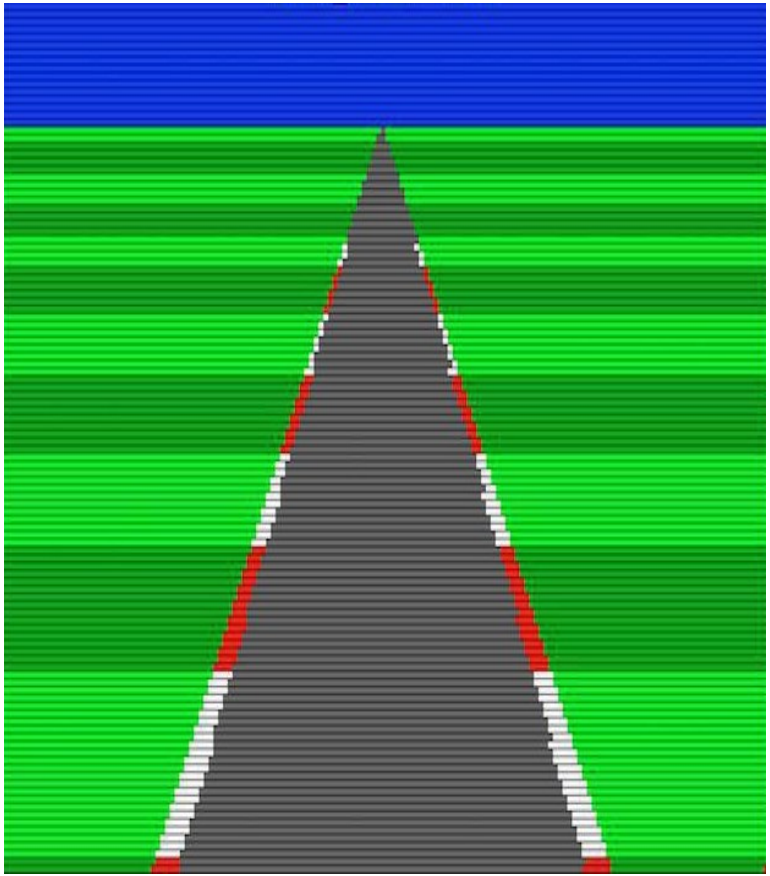


Another episode by the 8-Bit Guy just released

What’s at the horizon?

Gigatron team

- Tutorials
- Support
- Bug fixes
- …



Community

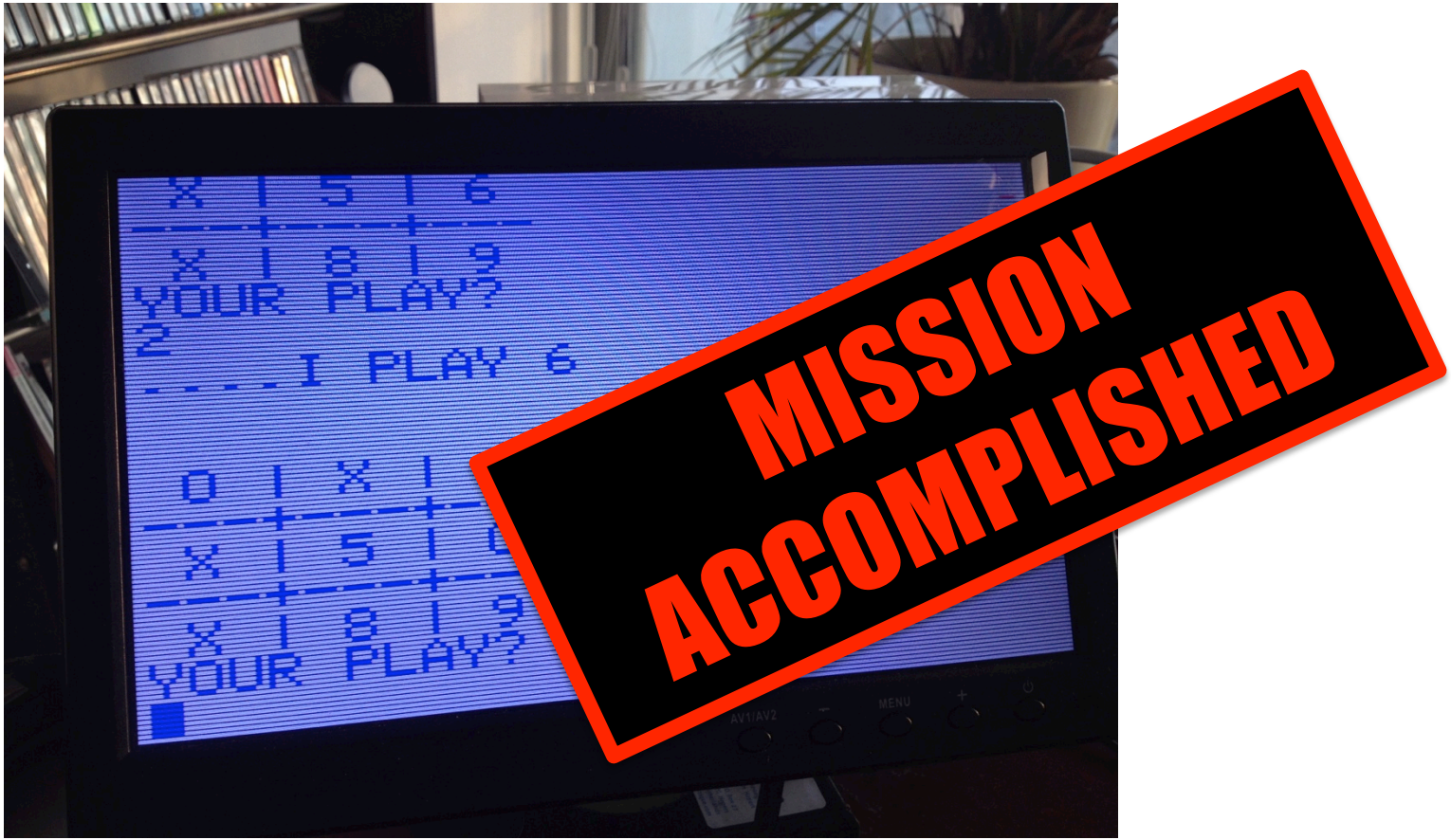
- Games (Galaga?)
- More tooling?
- Hardware hacks?
- 6502 emulator?
- FORTH?
- …

And oh, remember our original goal?

Tic Tac Toe?

Source: http://www.talkingelectronics.com

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



Thank you for your attention!