20180514

NLUUG

gigatron

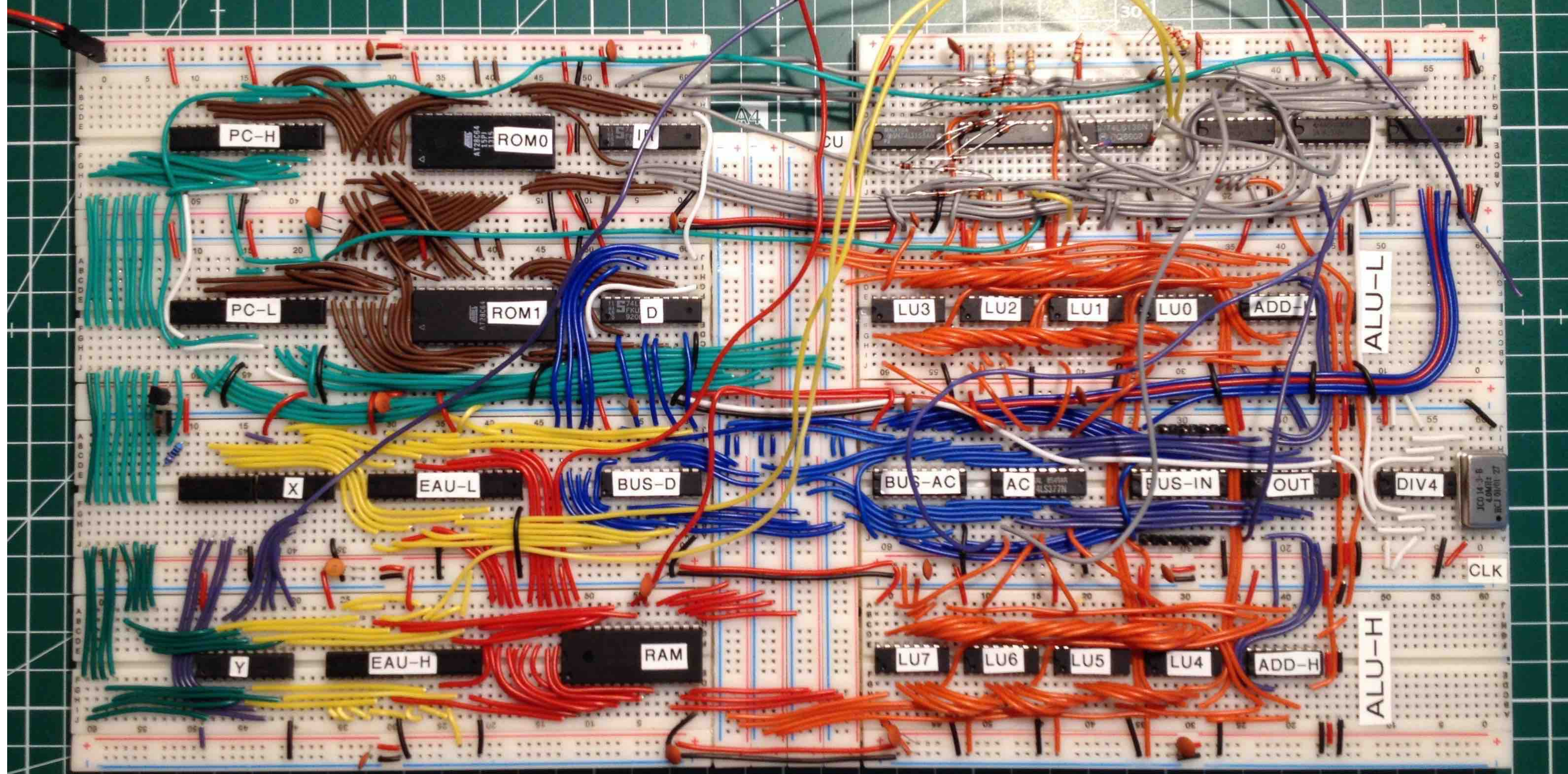
the TTL microcomputer

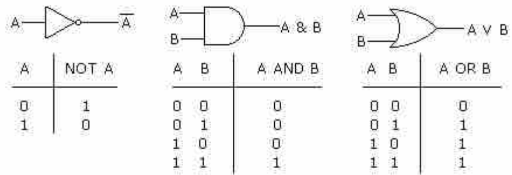
marcelk@gigatron.io

walter@gigatron.io

- Walter Belgers
- NLUUG speaker since 1994, honorary member
- Soldering since 1978
- Programming since 1978
- Collector of Sun/UNIX systems

gigatron

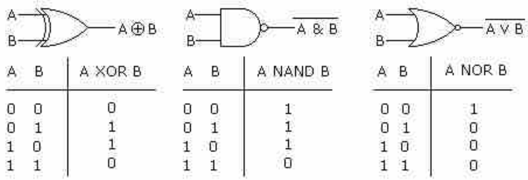




A	NOT A
0	1
1	0

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

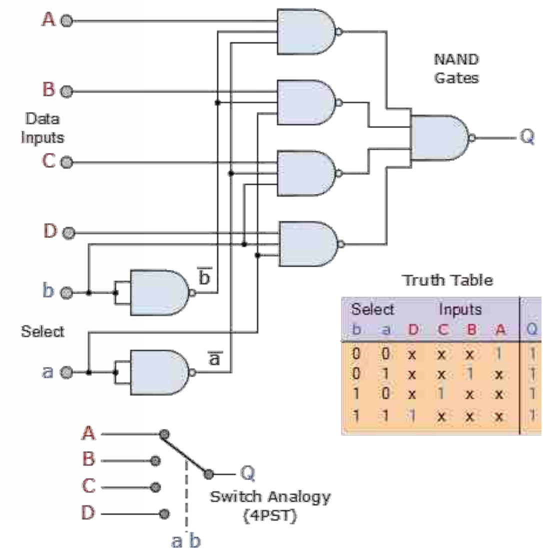
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1



A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

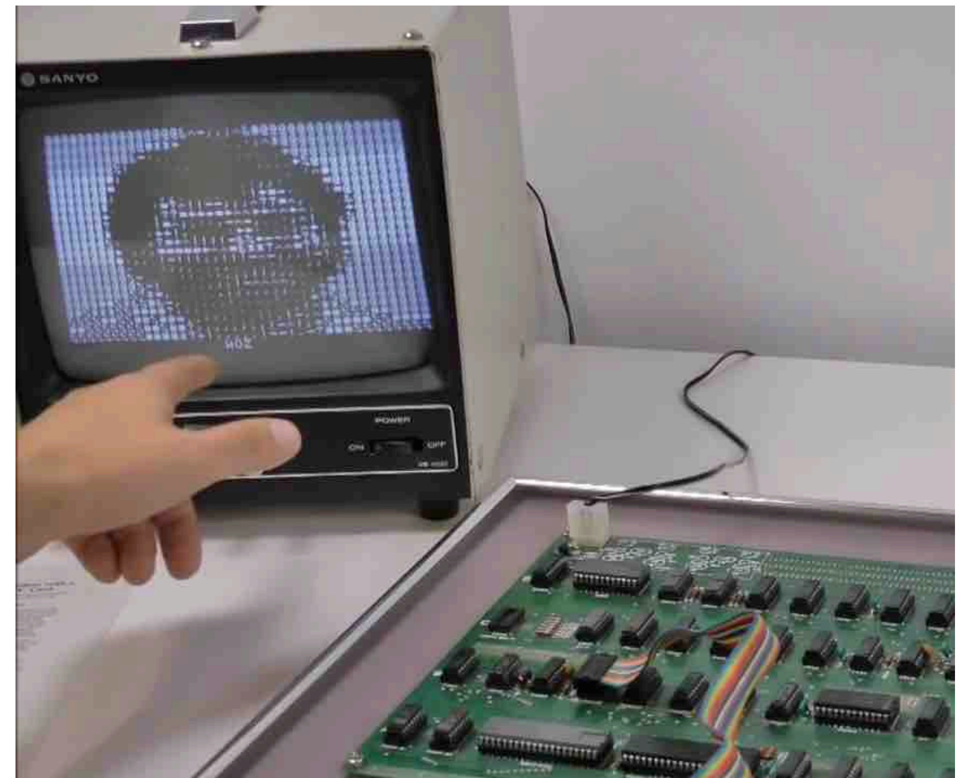
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0



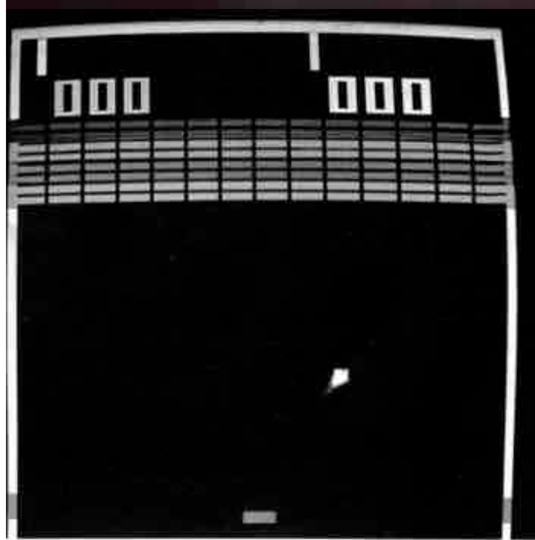
Select		Inputs				Q
b	a	D	C	B	A	Q
0	0	x	x	x	1	1
0	1	x	x	1	x	1
1	0	x	1	x	x	1
1	1	1	x	x	x	1

Apple 1

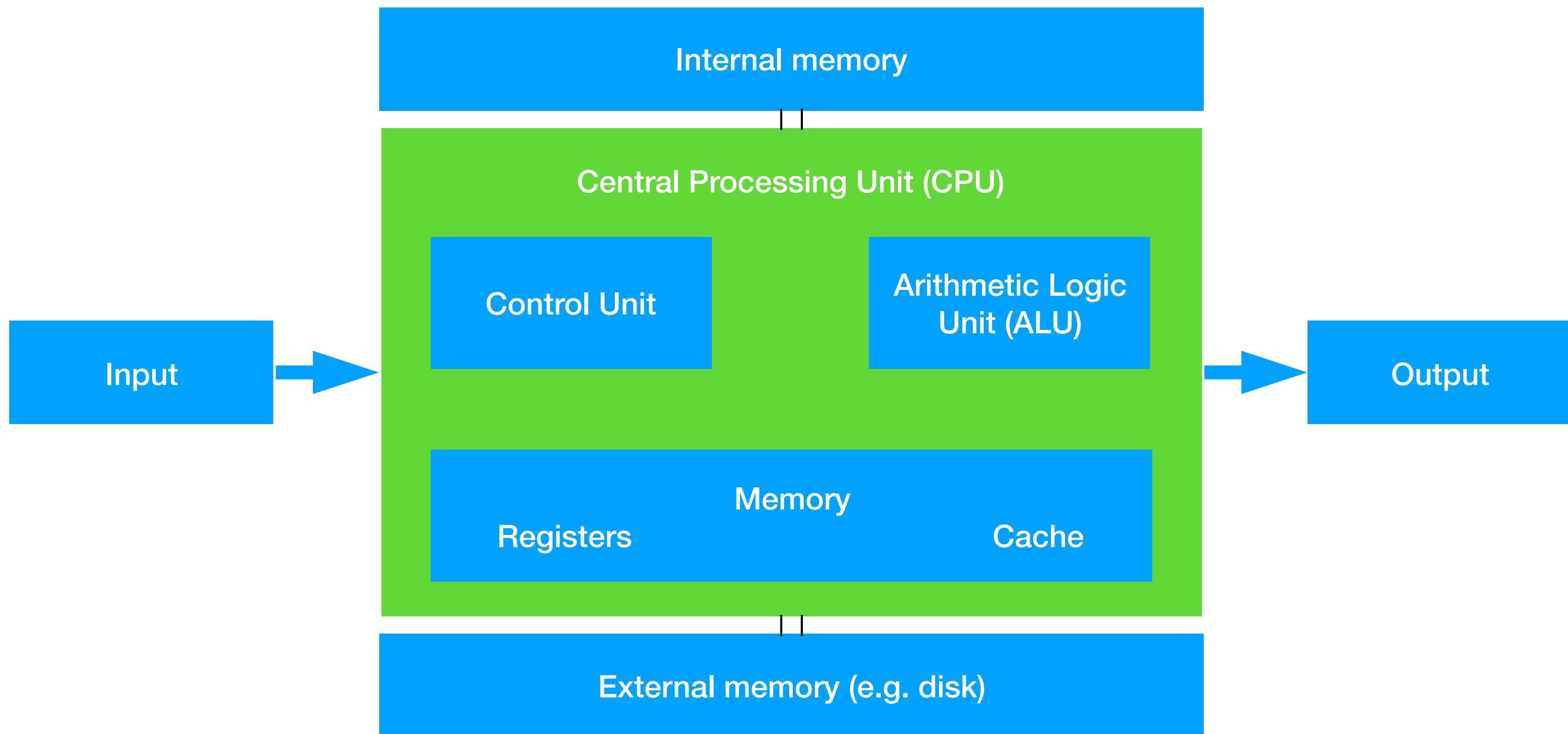
- 6502 microprocessor, 1MHz
- 4kB ROM
- 4kB RAM
- Monochrome output (composite)
  - 280x192 or 40x24 text



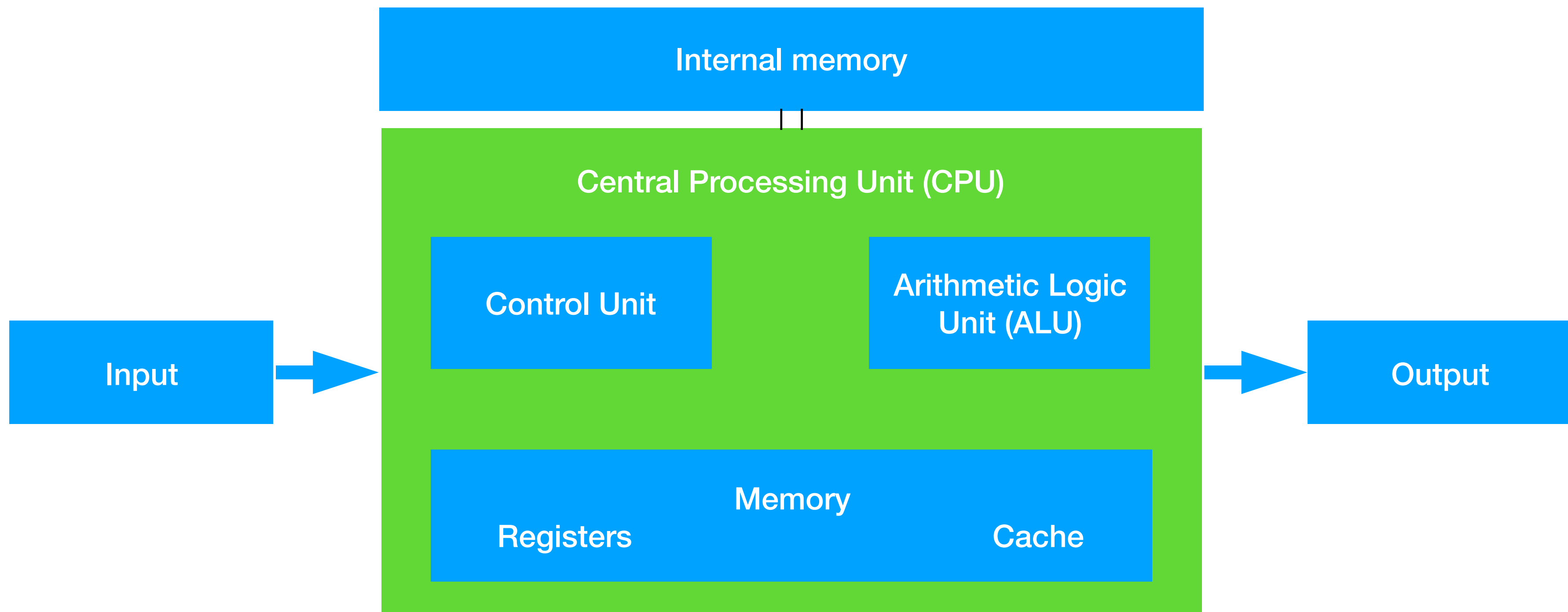
https://www.youtube.com/watch?v=4I8i_xOBTPg


000
000

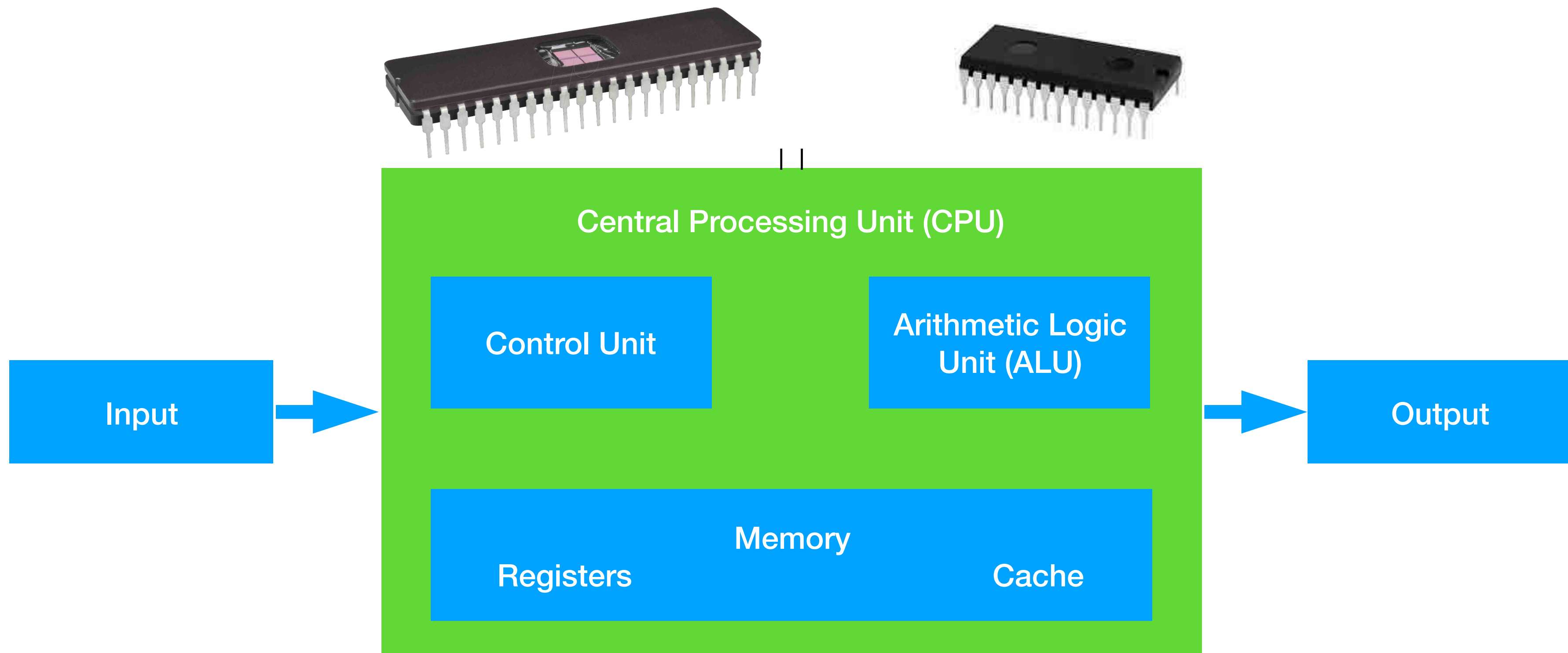
A computer



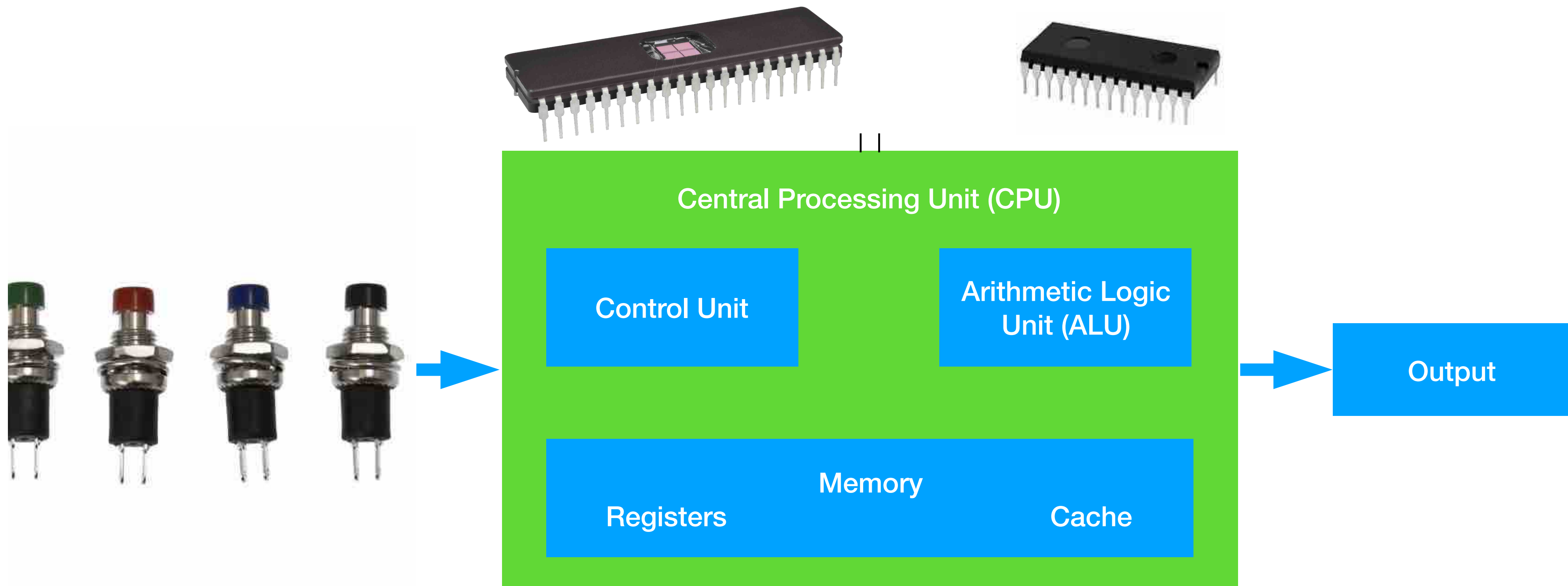
A computer



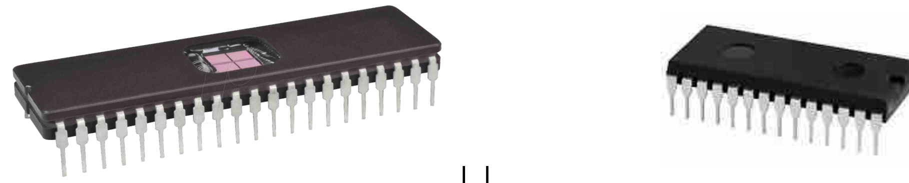
A computer

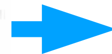


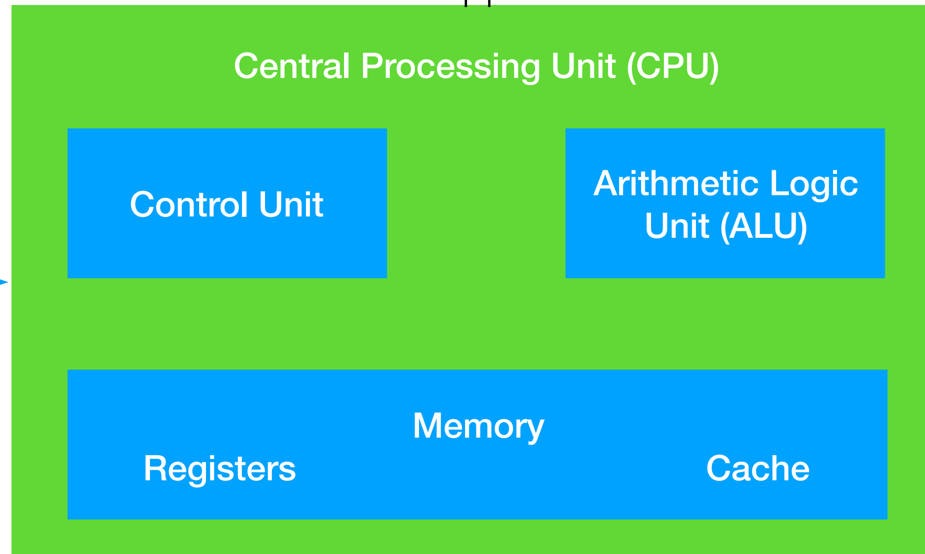
A computer

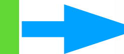


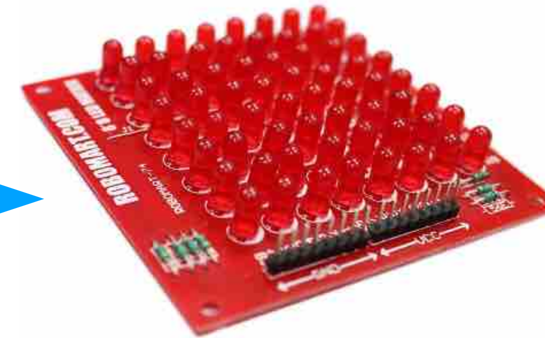
A computer



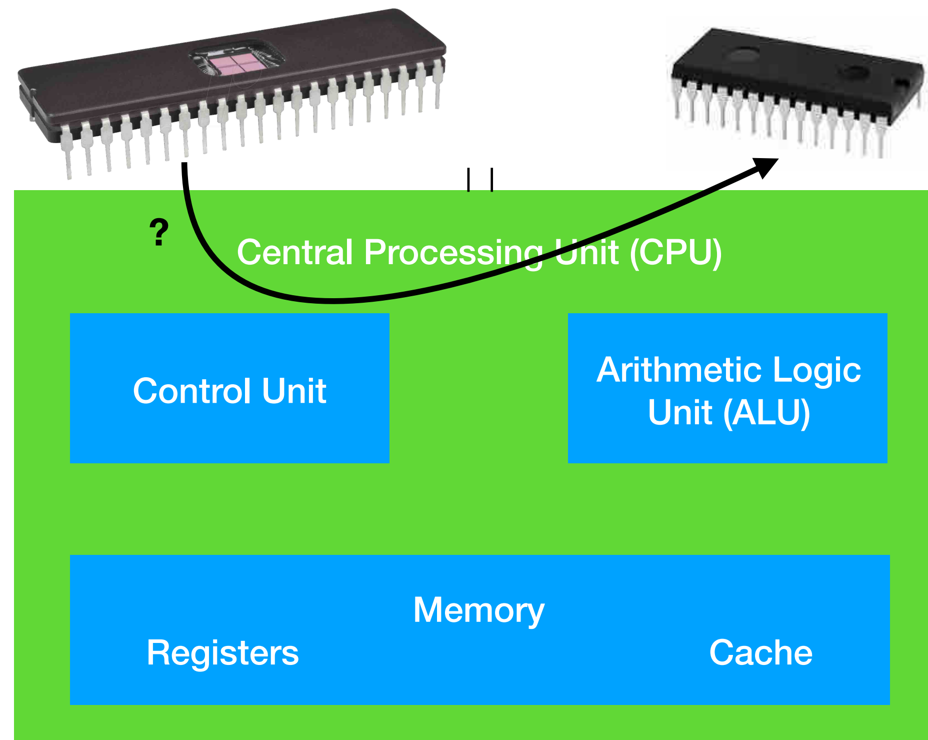




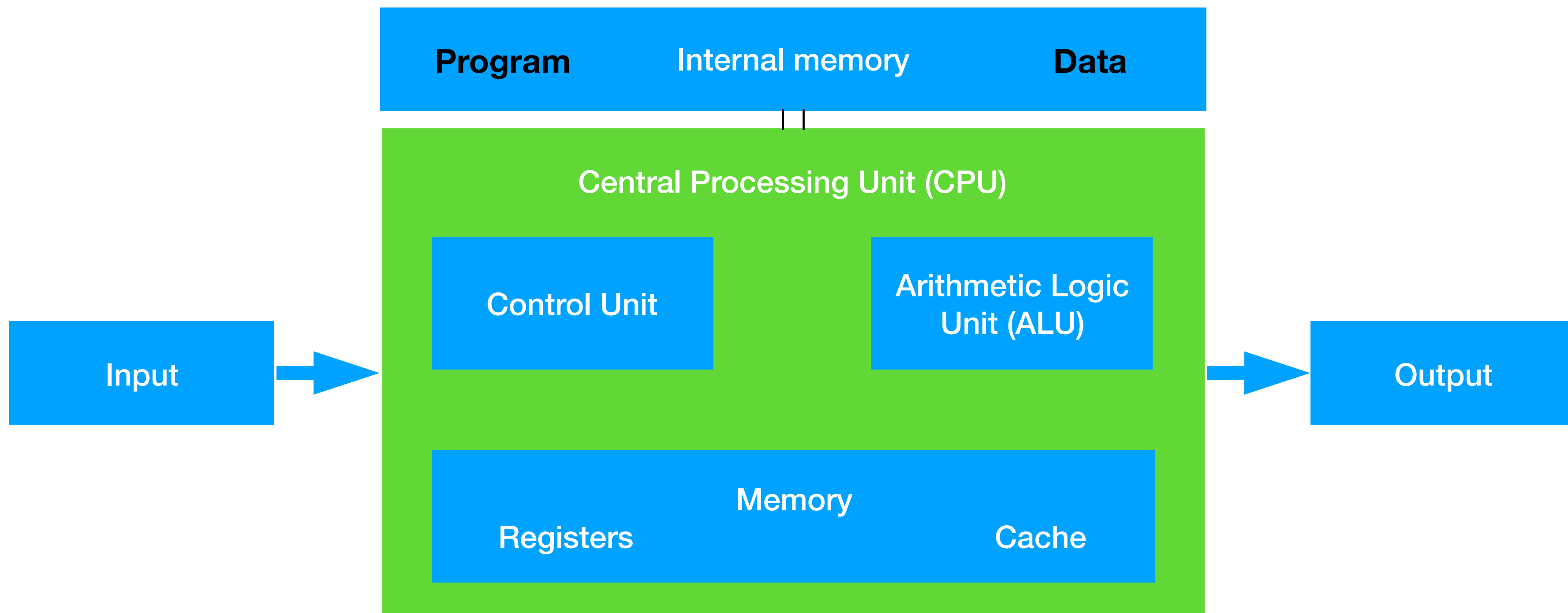




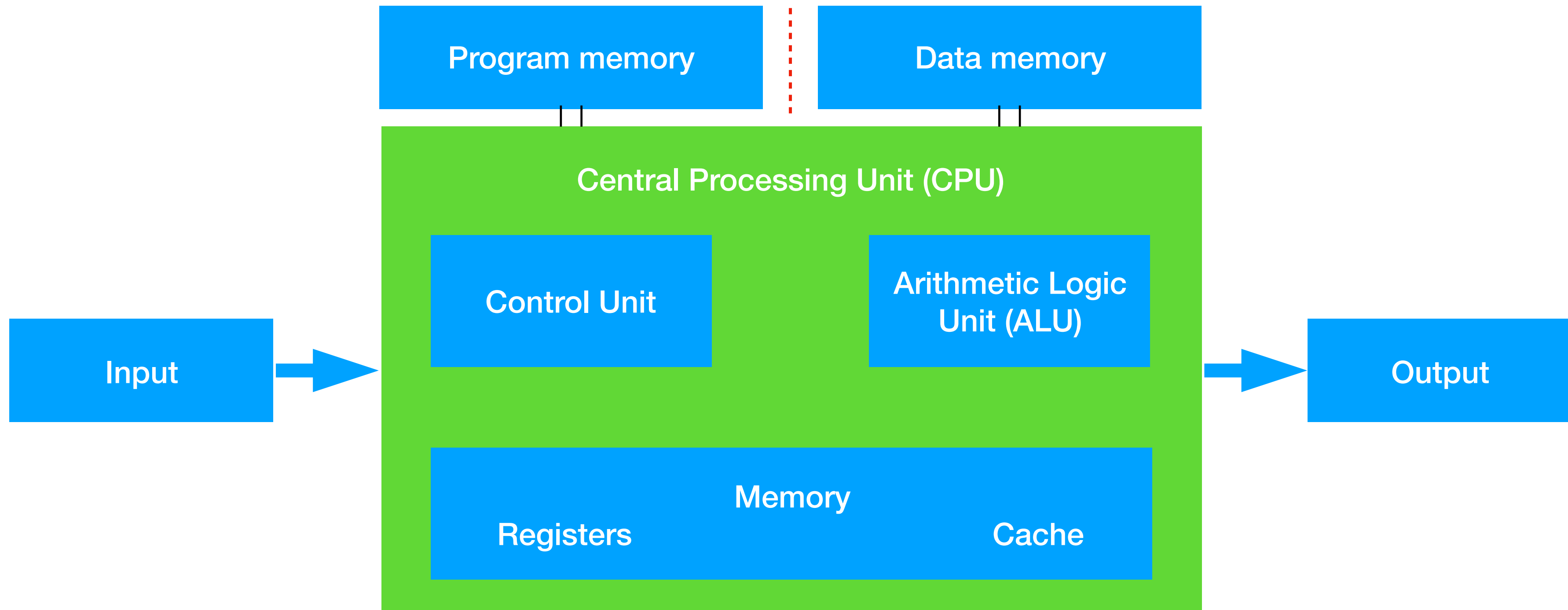
A computer



Von Neumann



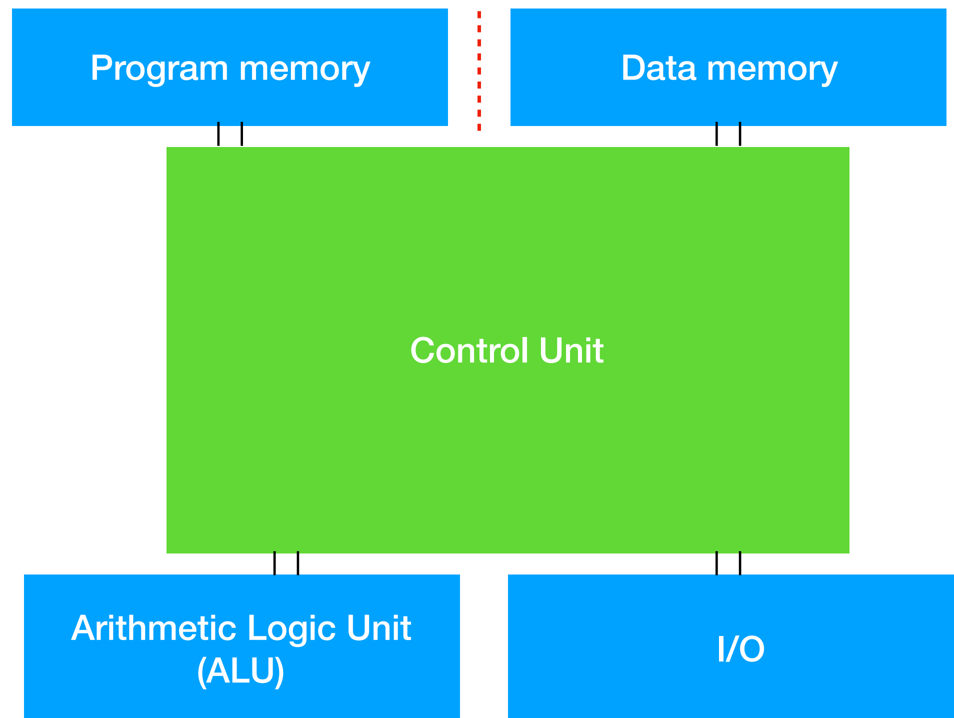
Harvard



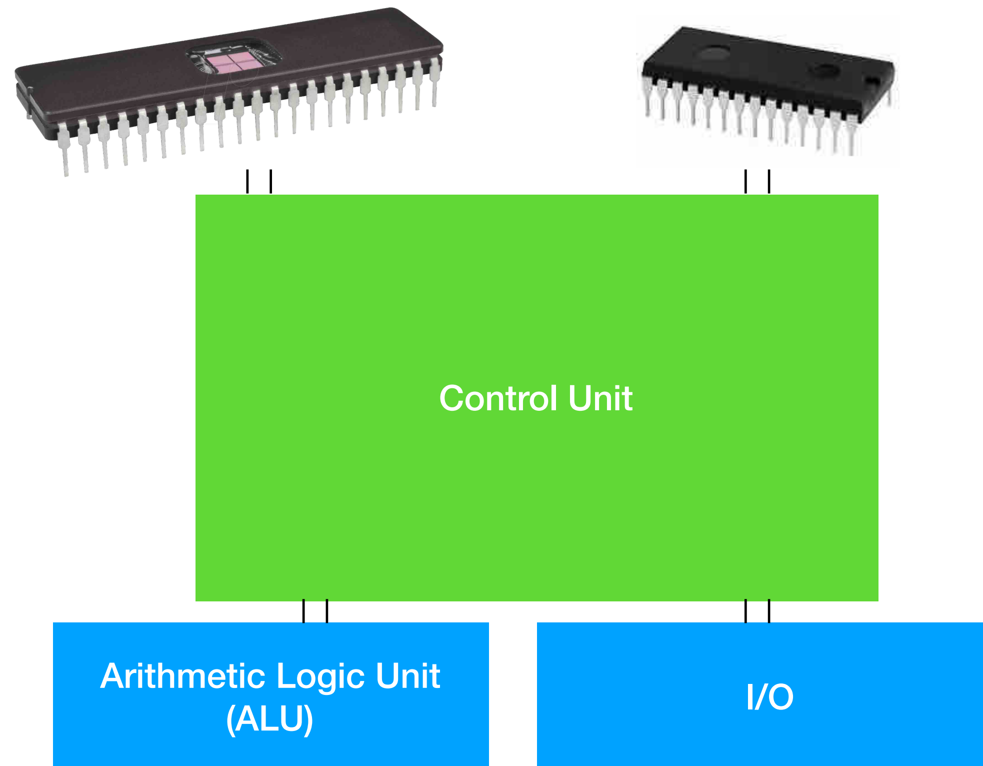
Von Neumann vs Harvard

- The first of many choices…
- Adding hardware complexity for added functionality?
- Goal: **keep the hardware as simple as we can, at the cost of more complex software**

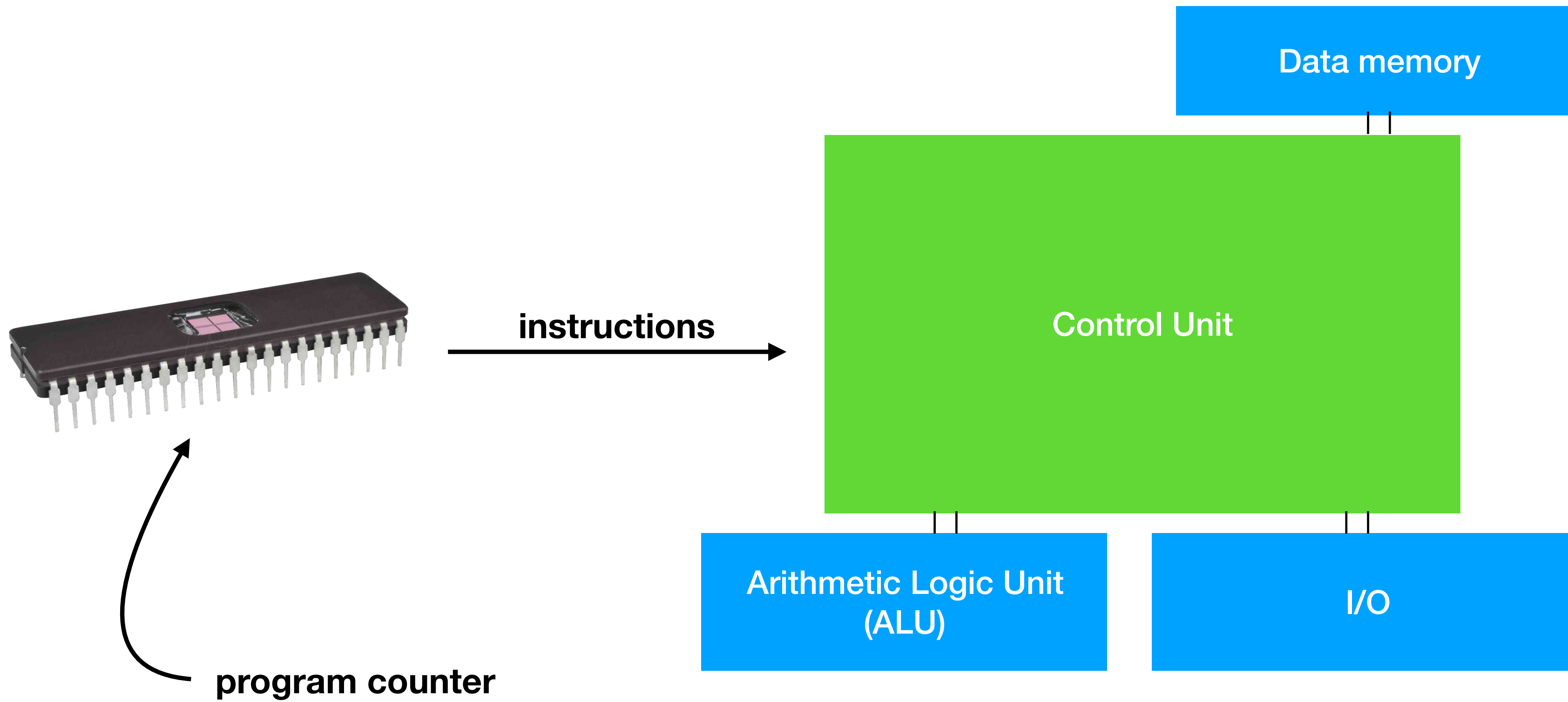
Harvard

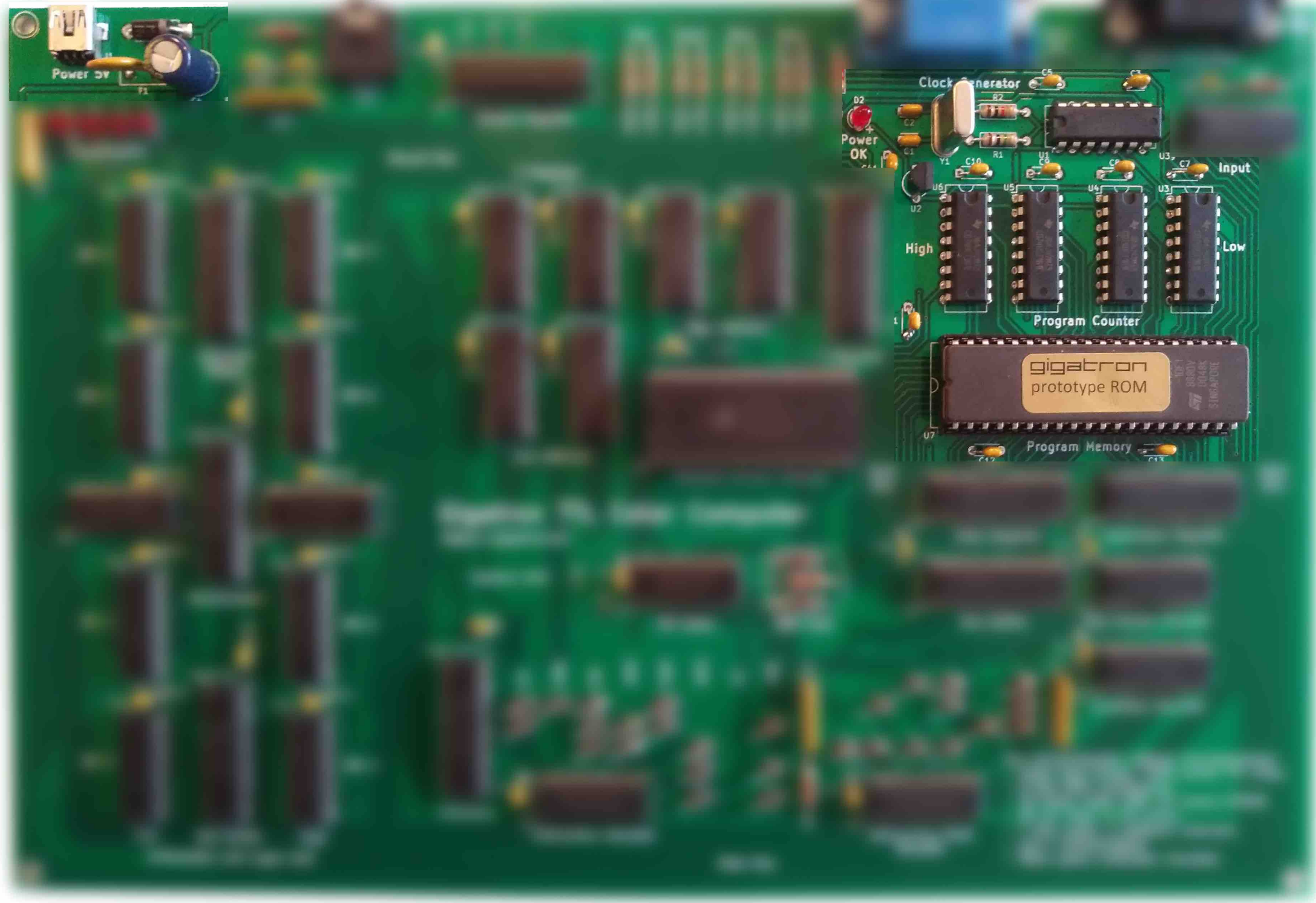


Harvard

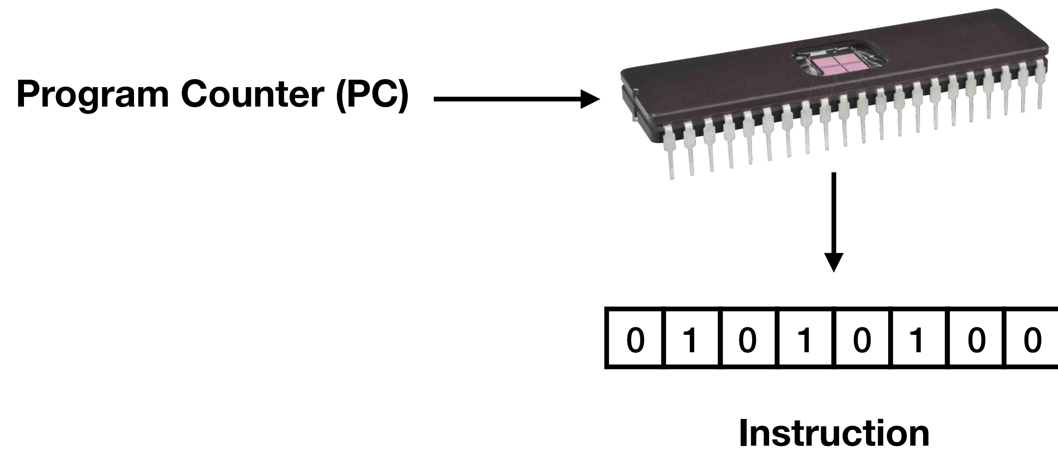


Harvard

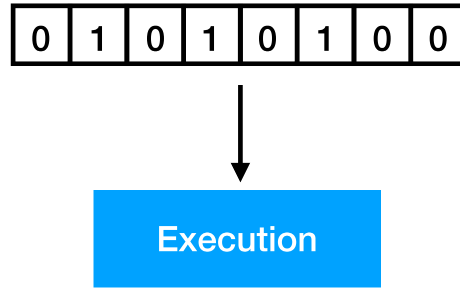



Power 5V
Clock Generator
Power OK
Input
High
Low
Program Counter
gigatron
prototype ROM
Program Memory

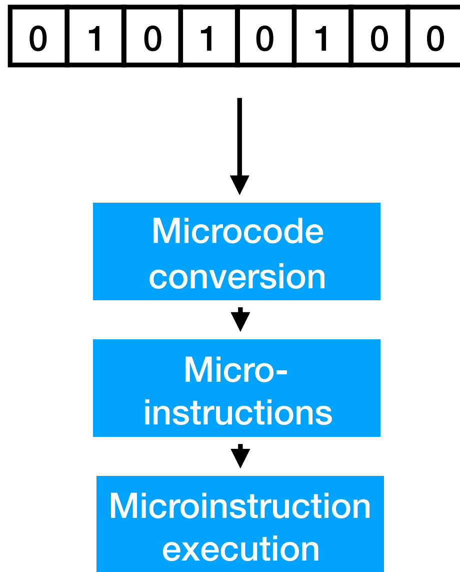
Instructions and operands



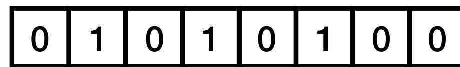
RISC



CISC



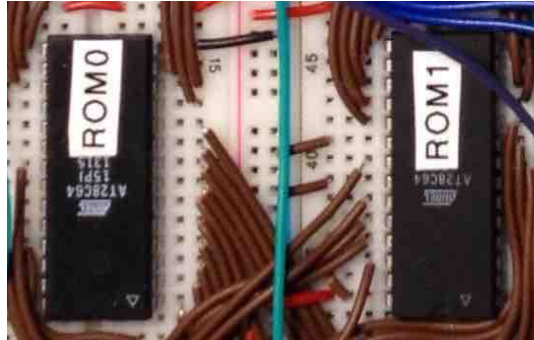
RISC

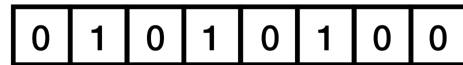


Execution

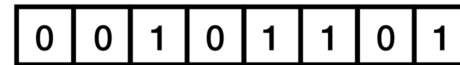
Instructions and operands

Program Counter (PC)



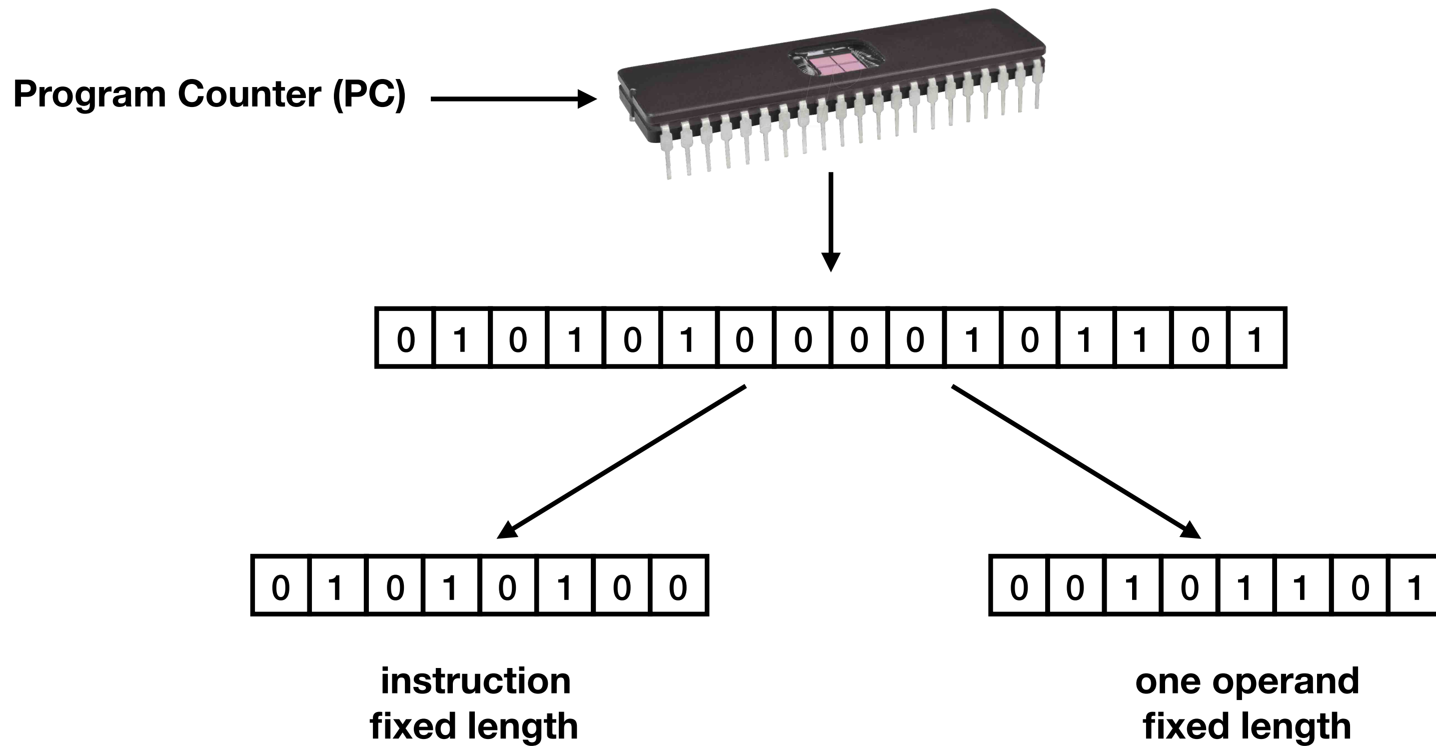


instruction
fixed length



one operand
fixed length

Instructions and operands

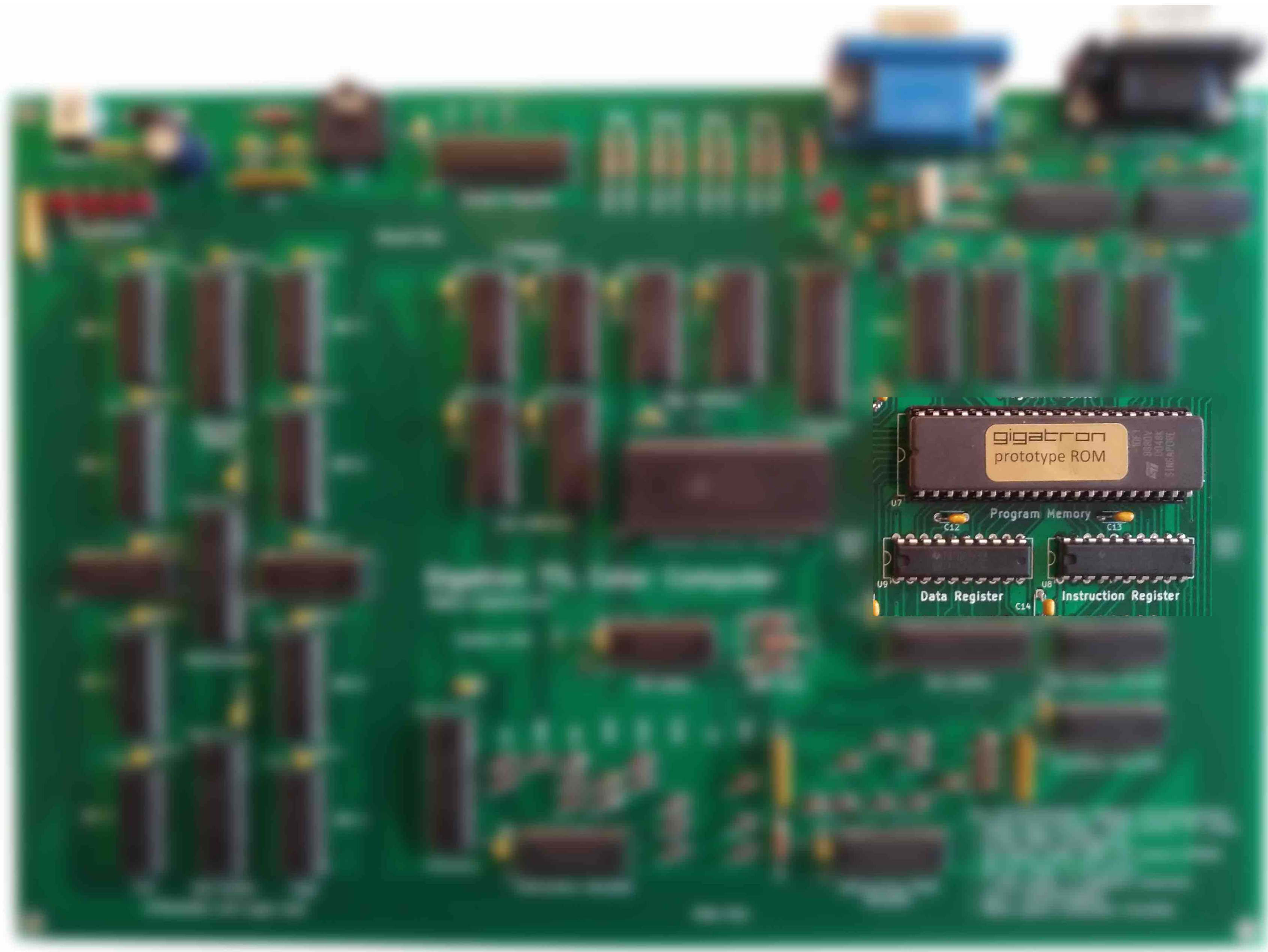


Pipelining

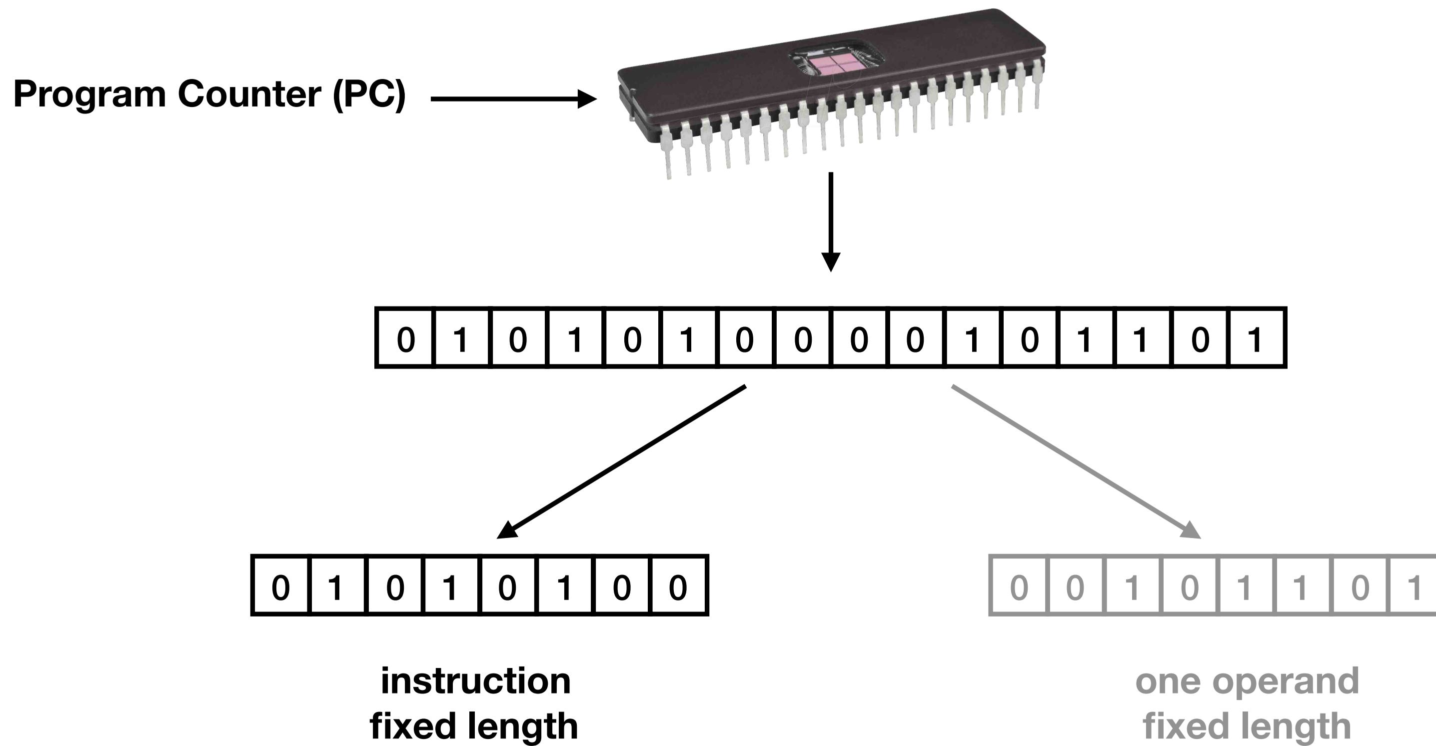
- Each instruction is one byte with one byte of data
  - Even if data is not used
- No microcode: one opcode, one cycle, one action
- When instruction is working on the opcode and data, the next instruction will already be fetched (pipelining)

Pipelining

- Step one: fetch instruction and operand from the EPROM and store them in an instruction register and data register
- Step two: execute the instruction
- These steps are done simultaneously


gigatron
prototype ROM
Program Memory
Data Register
Instruction Register

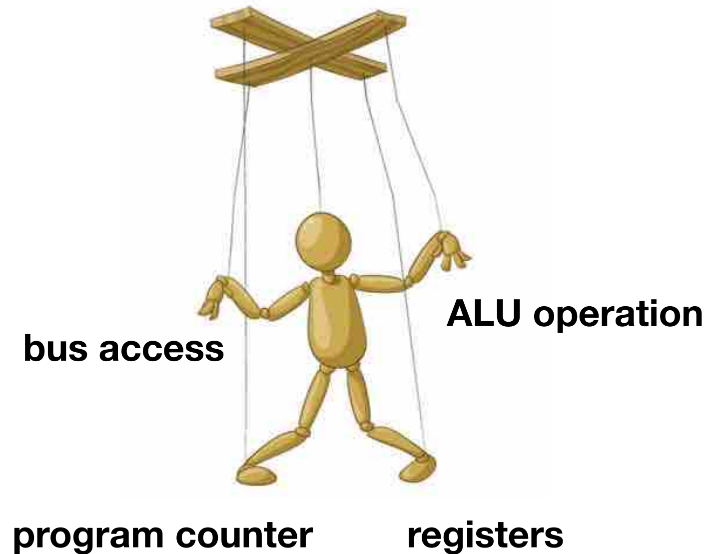
Instructions and operands



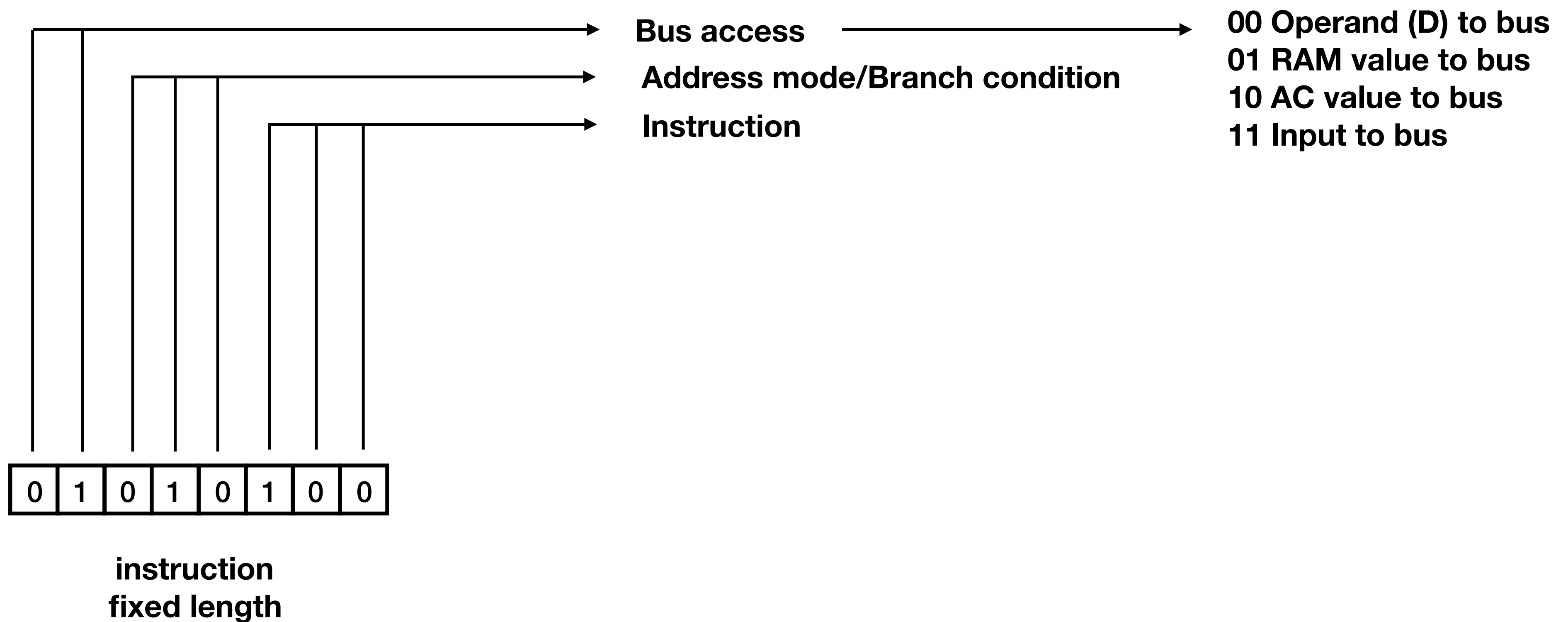
Instructions and operands

0	1	0	1	0	1	0	0

instruction
fixed length



Instructions and operands



Input

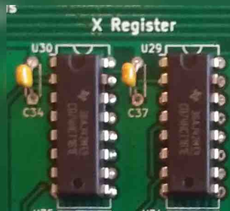
U36
Random Access Memory

U10
Bus Buffer
U11
Bus Access Decoder

Bus Buffer

Registers

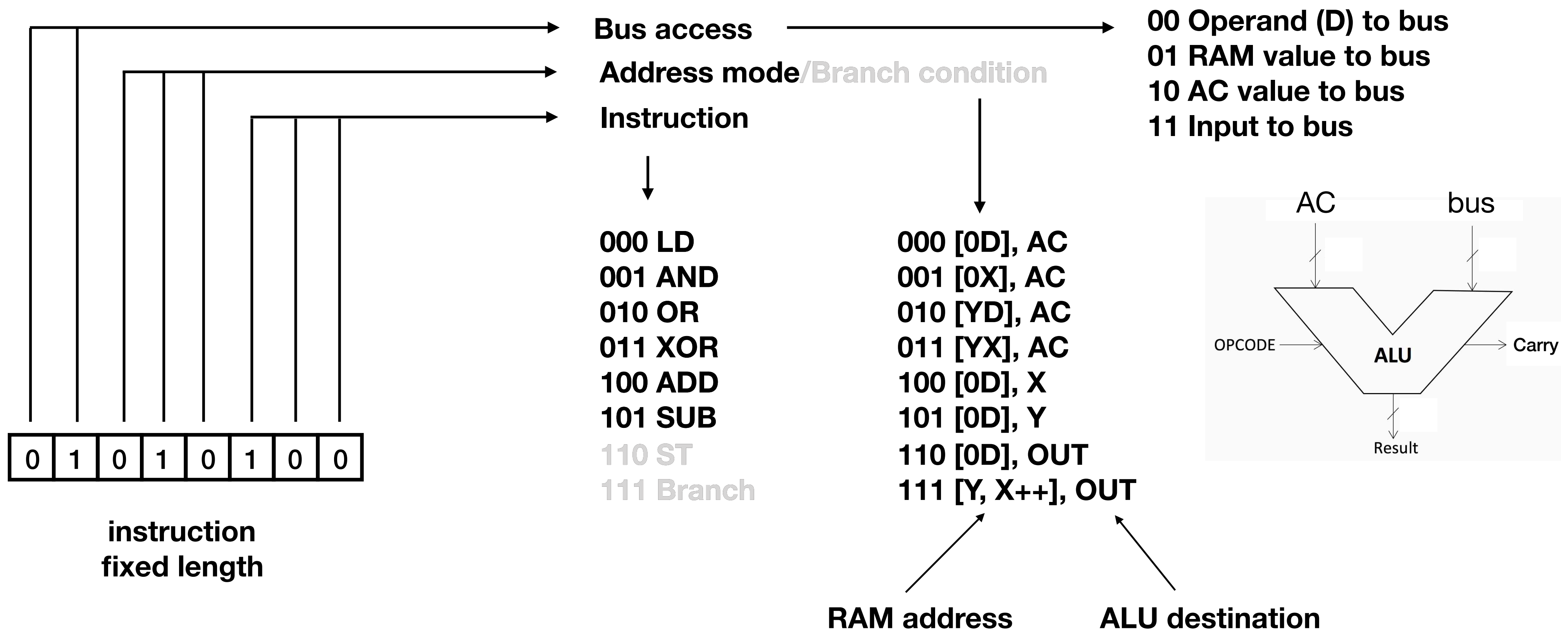
- Program counter (**PC**): can be set using branch instruction, incremented via the clock en set to 0 at boot time
- **IR**, **D** buffer: buffer between EPROM and bus
- Accumulator (**AC**): contains ALU result, R/W
- **X** register: can be used for the low bits of the RAM address, counter, write only
- **Y** register: can be used for the high bits of the RAM address, buffer, write only
- **OUT** register: buffer, write only
- **IN** register: buffer, read only

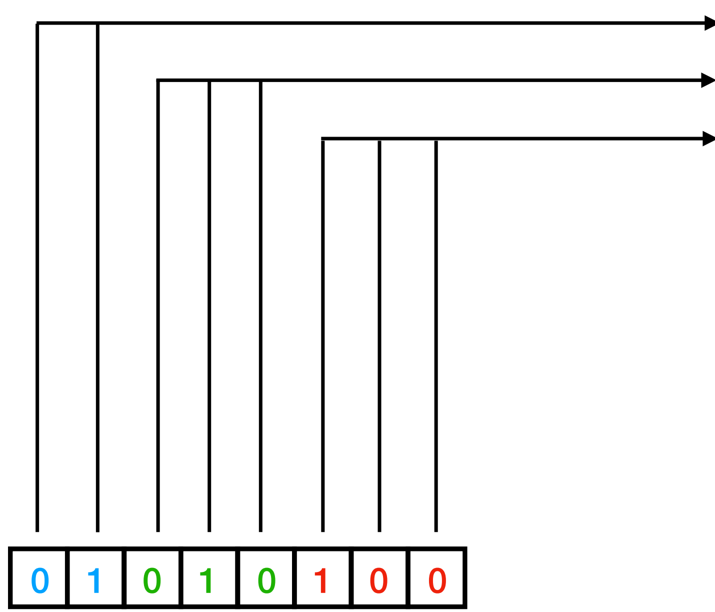

X Register
U30
U29
C34
C37

U31
Y Register

C27
U27
U25
Accumulator

Instructions and operands





instruction
fixed length

Bus access →

00 Operand (D) to bus
01 RAM value to bus
10 AC value to bus
11 Input to bus

Address mode/Branch condition

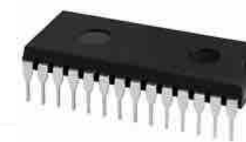
000 [0D], AC
001 [0X], AC
010 [YD], AC
011 [YX], AC
100 [0D], X
101 [0D], Y
110 [0D], OUT
111 [Y, X++], OUT

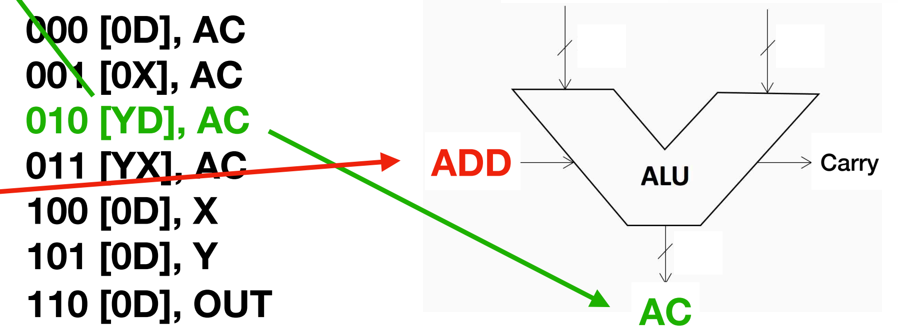
Instruction

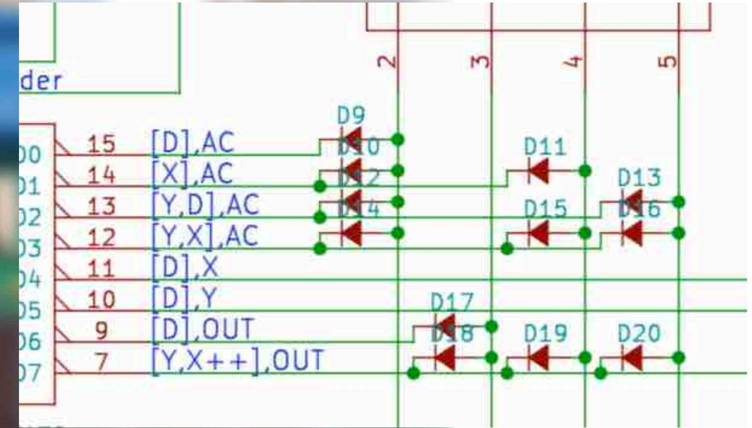
000 LD
001 AND
010 OR
011 XOR
100 ADD
101 SUB
110 ST
111 Branch

RAM address

ALU destination

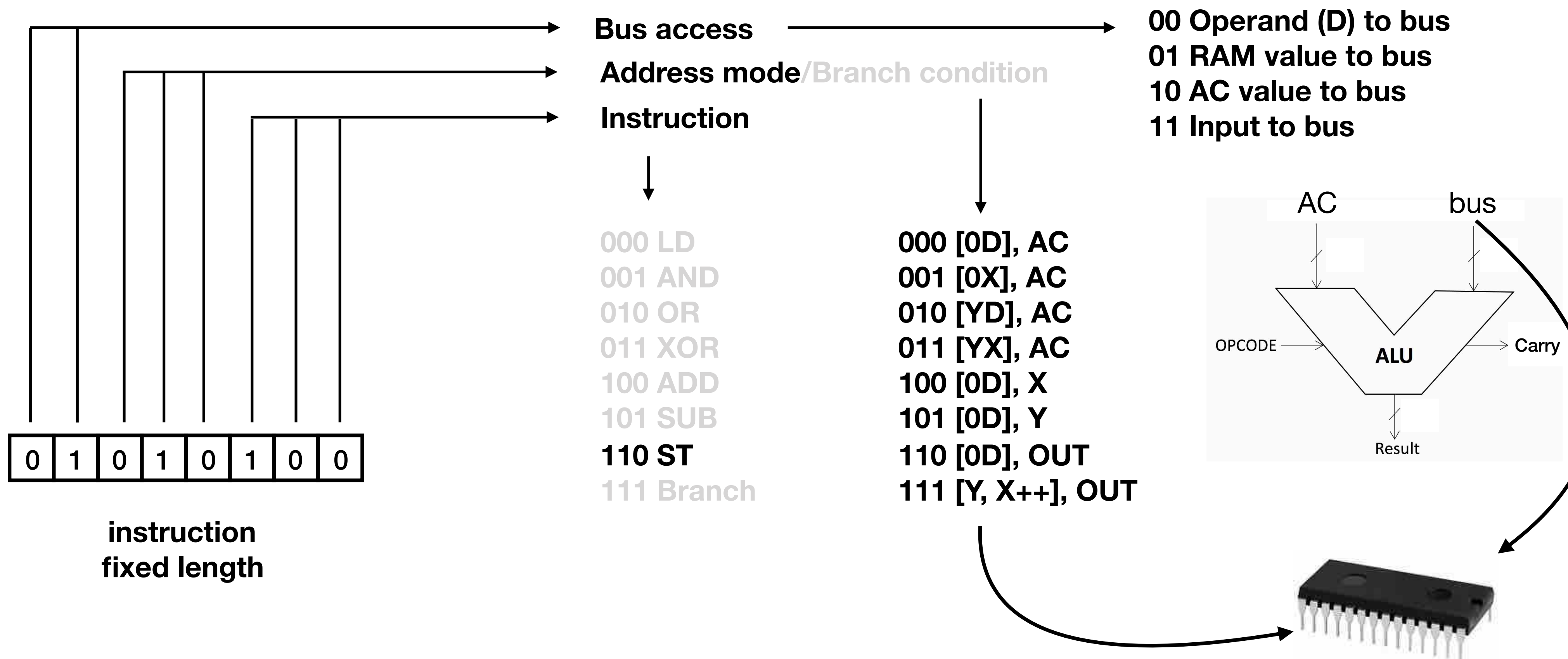




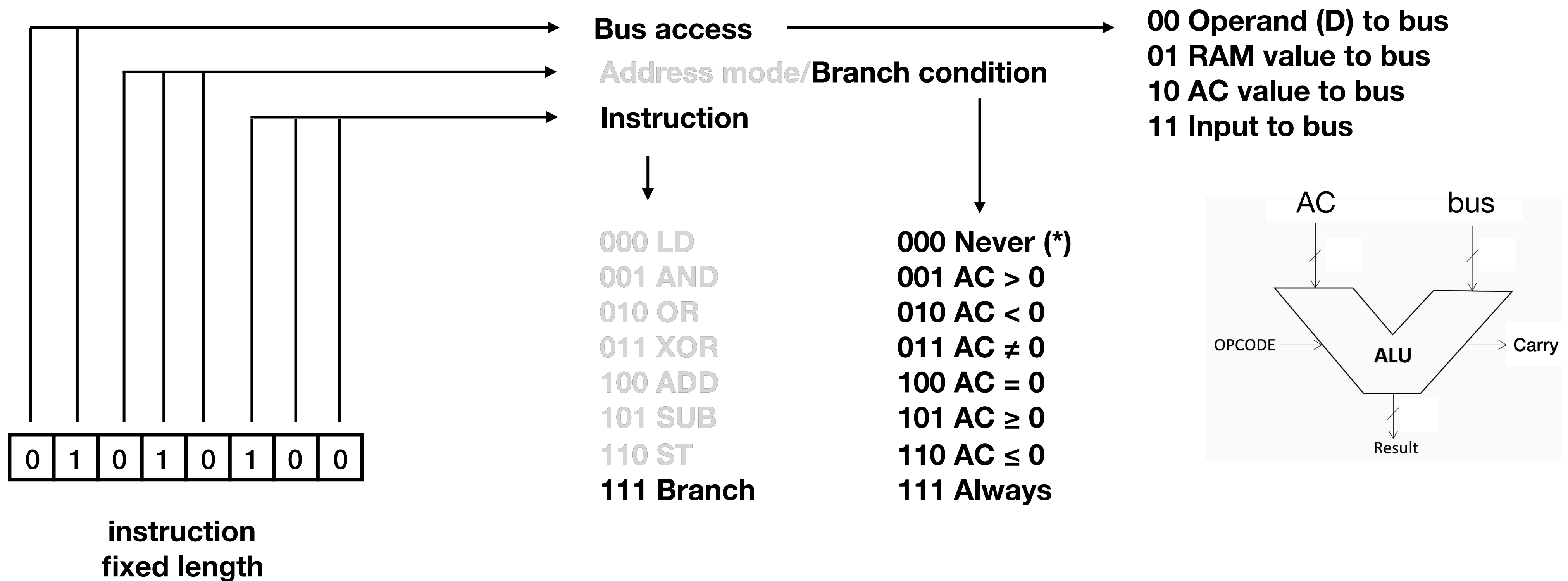


der
2
3
4
5
D9
D10
D11
D12
D13
D14
D15
D16
D17
D18
D19
D20
15 [D],AC
14 [X],AC
13 [Y,D],AC
12 [Y,X],AC
11 [D],X
10 [D],Y
9 [D],OUT
7 [Y,X++],OUT

Instructions and operands



Instructions and operands



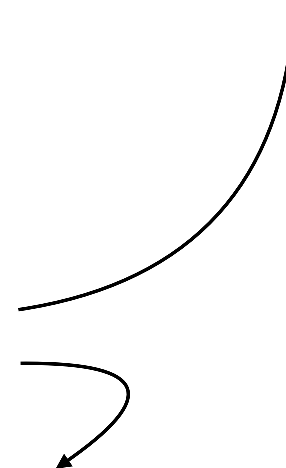
Conditional branches

	AC = 0	AC < 0	AC > 0	COMBINED
000 Never (*)	x	x	x	never
001 AC > 0	x	x	√	AC > 0
010 AC < 0	x	√	x	AC < 0
011 AC ≠ 0	x	√	√	AC ≠ 0
100 AC = 0	√	x	x	AC = 0
101 AC ≥ 0	√	x	√	AC ≥ 0
110 AC ≤ 0	√	√	x	AC ≤ 0
111 Always	√	√	√	Always

Conditional branches

	Carry(-AC)=1 AC = 0	AC[7]=1 AC < 0	AC[7]=0 && Carry(-AC)=0 AC≥0 && AC≠0	COMBINED
000 Never (*)	x	x	x	never
001 AC > 0	x	x	√	AC > 0
010 AC < 0	x	√	x	AC < 0
011 AC ≠ 0	x	√	√	AC ≠ 0
100 AC = 0	√	x	x	AC = 0
101 AC ≥ 0	√	x	√	AC ≥ 0
110 AC ≤ 0	√	√	x	AC ≤ 0
111 Always	√	√	√	Always

Instruction	Mnemonic	What the ALU calculates
000	**LD**	Bus
001	**AND**	AC AND Bus
010	**OR**	AC OR Bus
011	**XOR**	AC XOR Bus
100	**ADD**	AC + Bus
101	**SUB**	AC - Bus
110	**ST**	AC
111	**Branch**	- AC

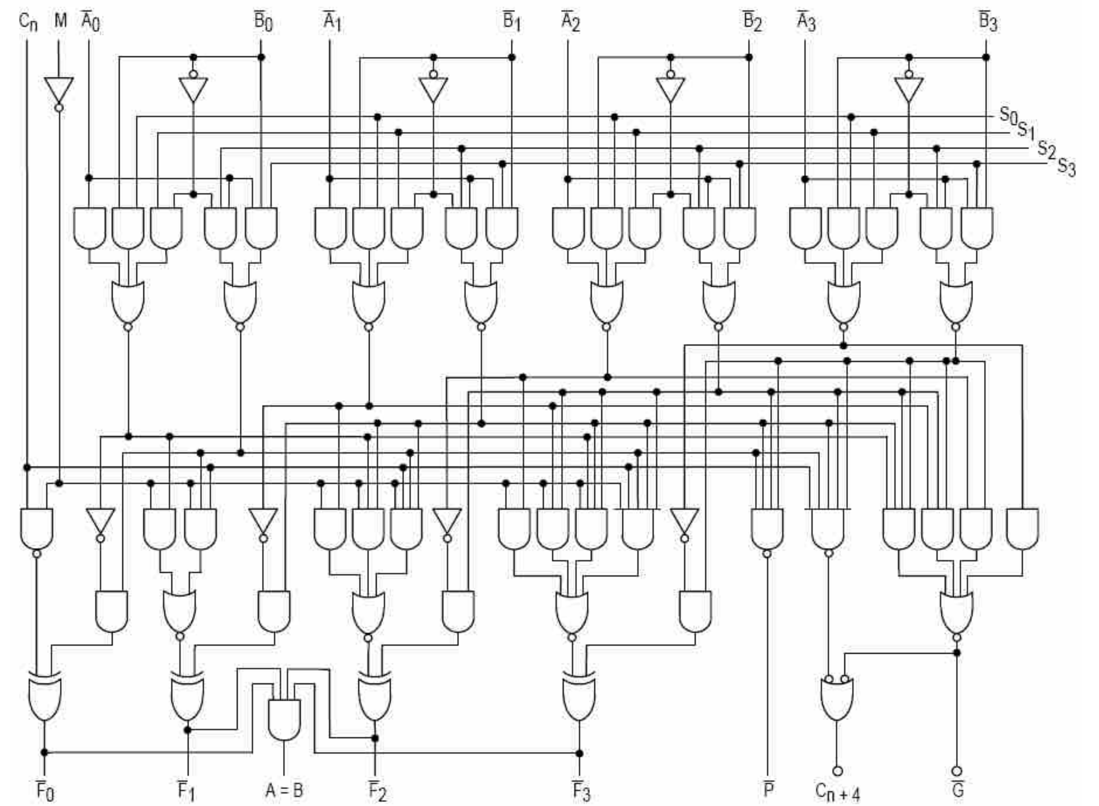


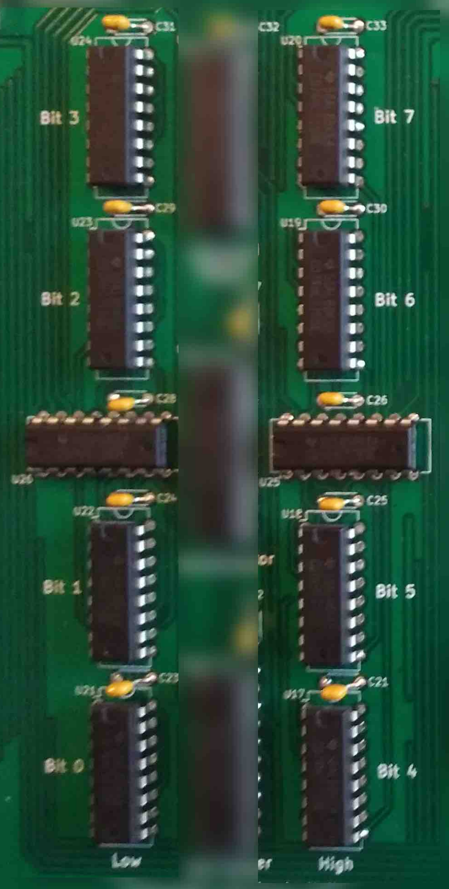
ST is used to write data to RAM. But, in the same cycle, the AC can now be copied to X, Y by using the correct address mode

The carry bit of the ALU can now be used to determine the branch condition

7352
SN74S181N

7352
SN74S181N


Cn
M
A0
B0
A1
B1
A2
B2
A3
B3
S0
S1
S2
S3
F0
F1
A = B
F2
F3
P
Cn + 4
G

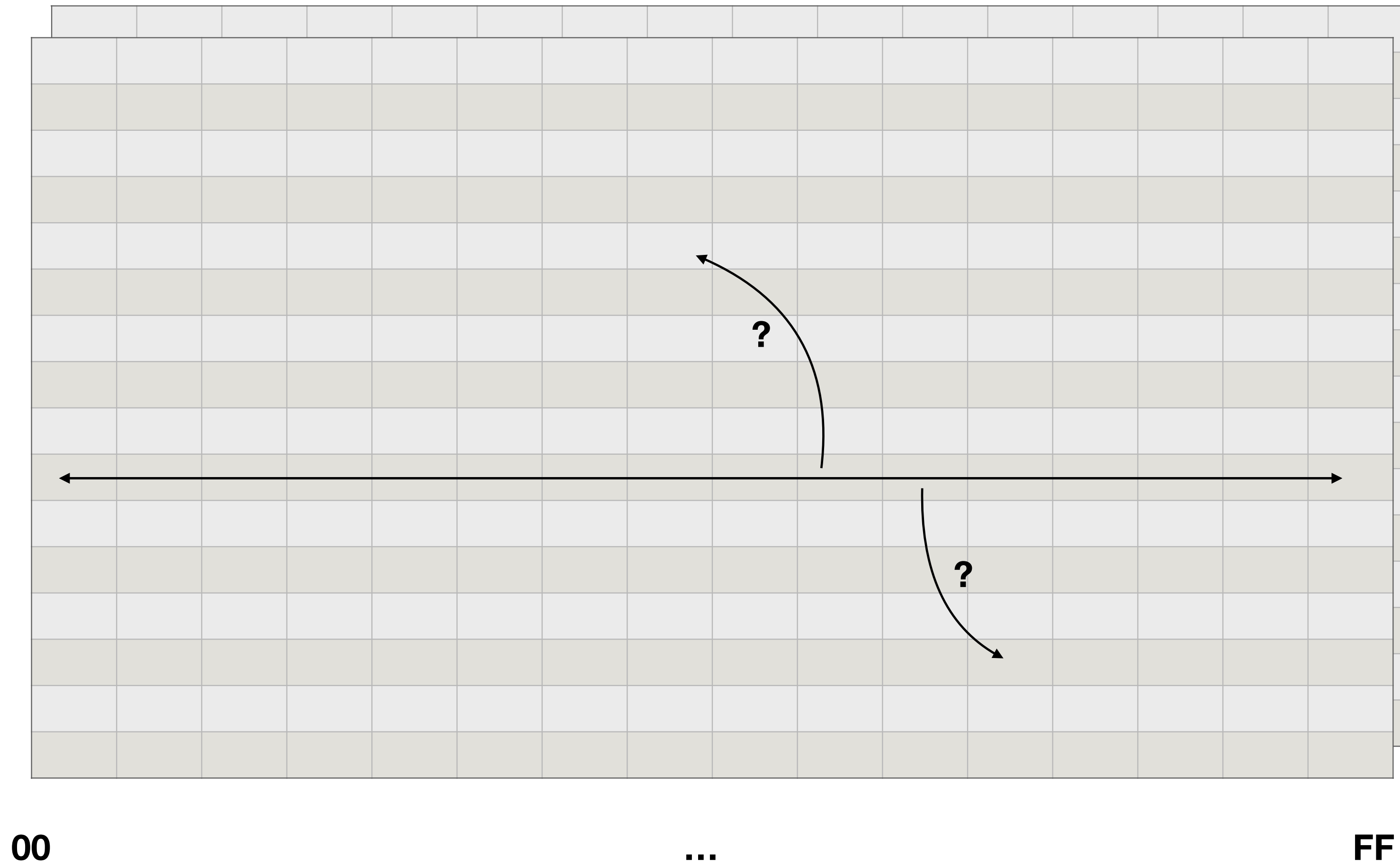

C31
C33
Bit 3
Bit 7
C29
C30
Bit 2
Bit 6
C28
C26
Bit 1
Bit 5
C25
C21
Bit 0
Bit 4
Low
High

U33
U32
C38
C40
High Address
C39

U35
U34
C35
C36
Low Address

Branch addressing

Branch addressing



Far jump

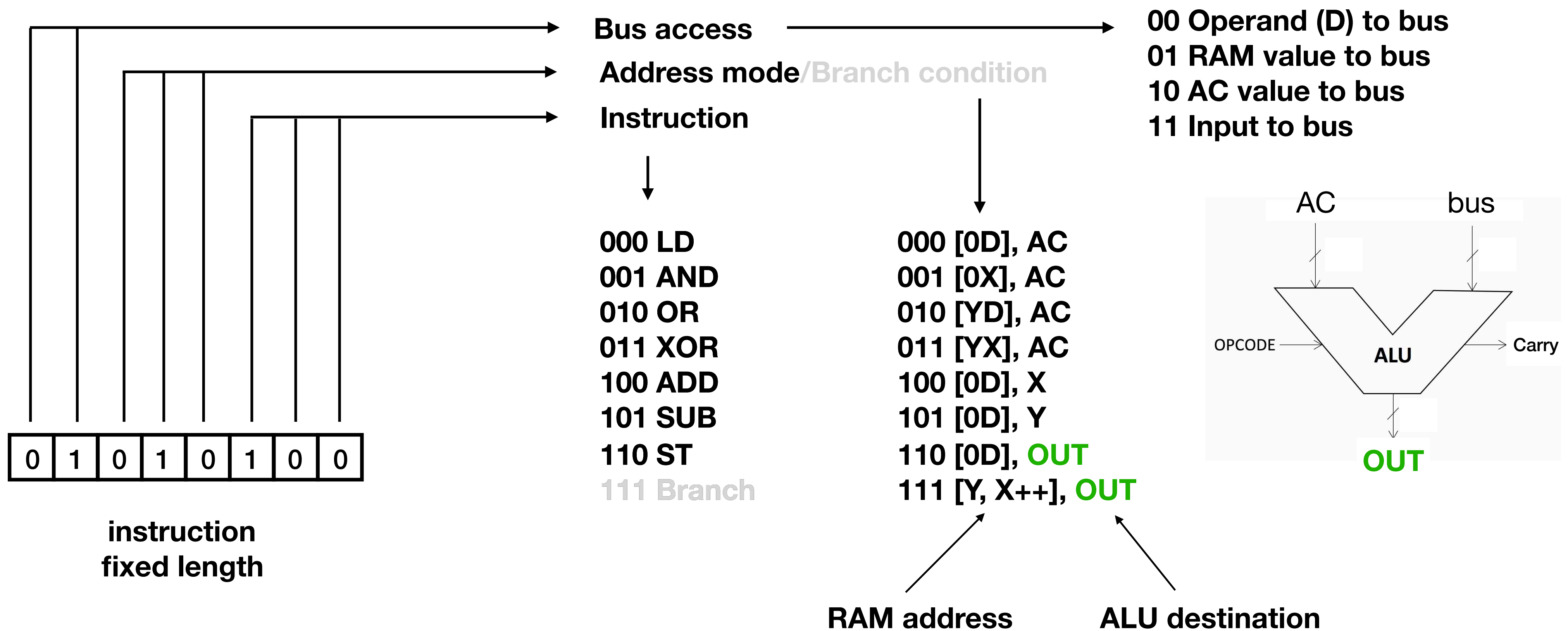
000 Far jump
001 AC > 0
010 AC < 0
011 AC $\neq 0$
100 AC $= 0$
101 AC ≥ 0
110 AC ≤ 0
111 Jump

- One branch is on the condition of “never”
- Added some hardware to make it useful: far jump
  - Use Y register contents as high bits

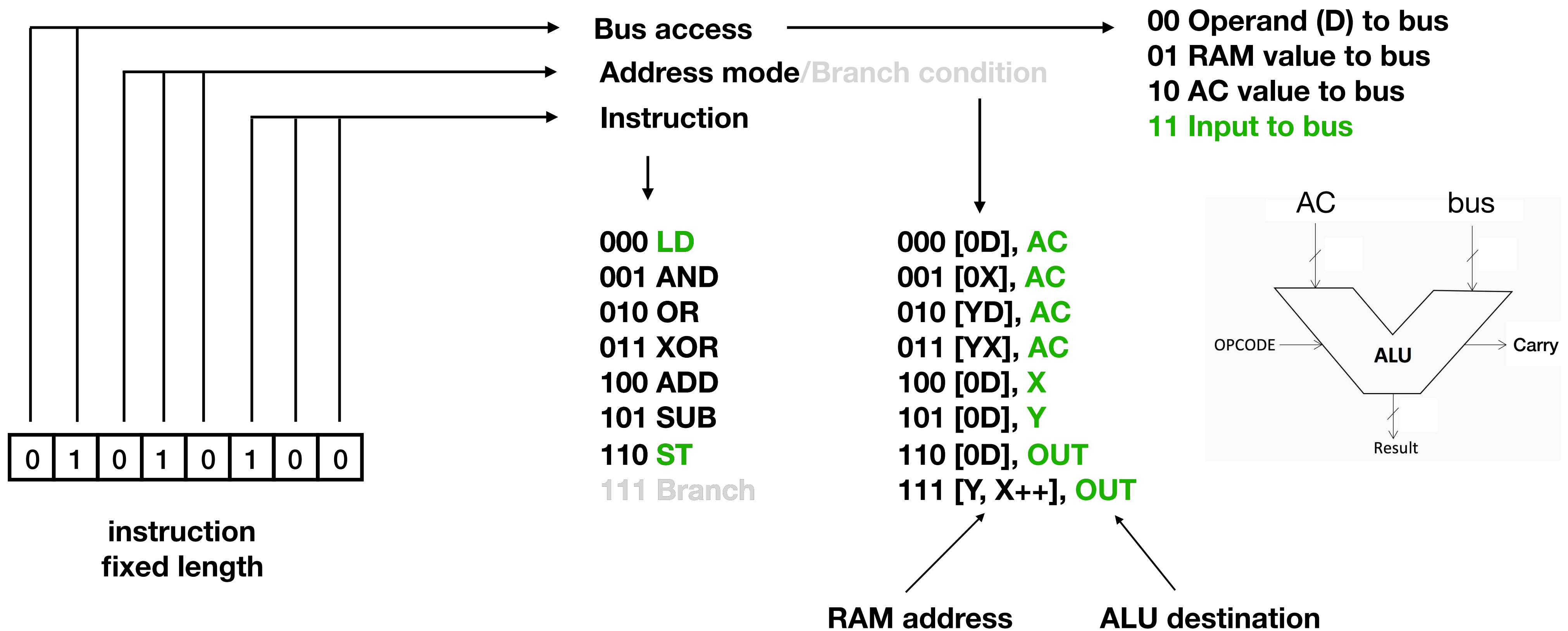
Input and Output

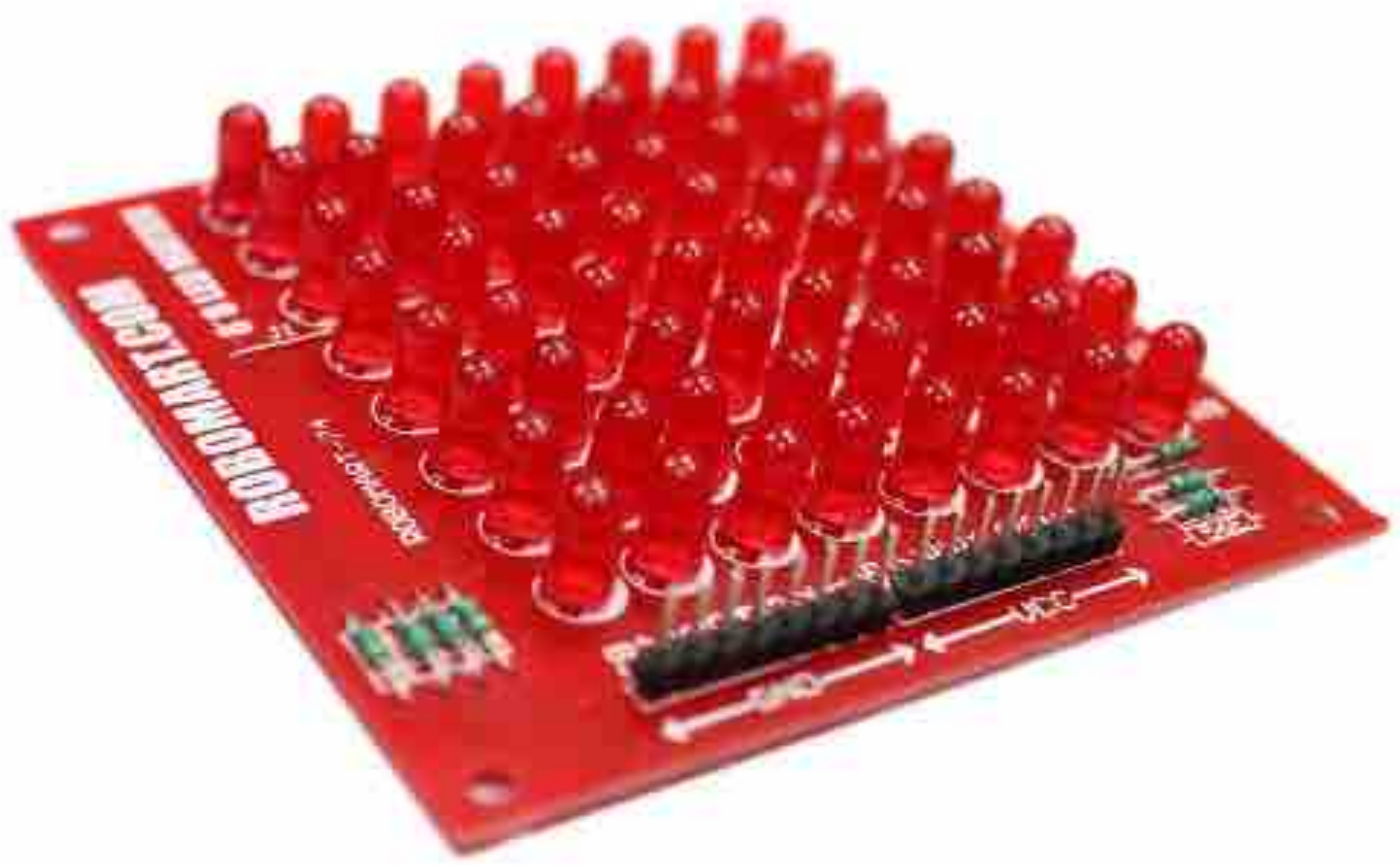
- In principle, fairly simple
- Output: copy ALU result to output using addressing mode
- Input: copy input to the bus
  (and from there to AC, X, Y, RAM or OUT)

Output



Input

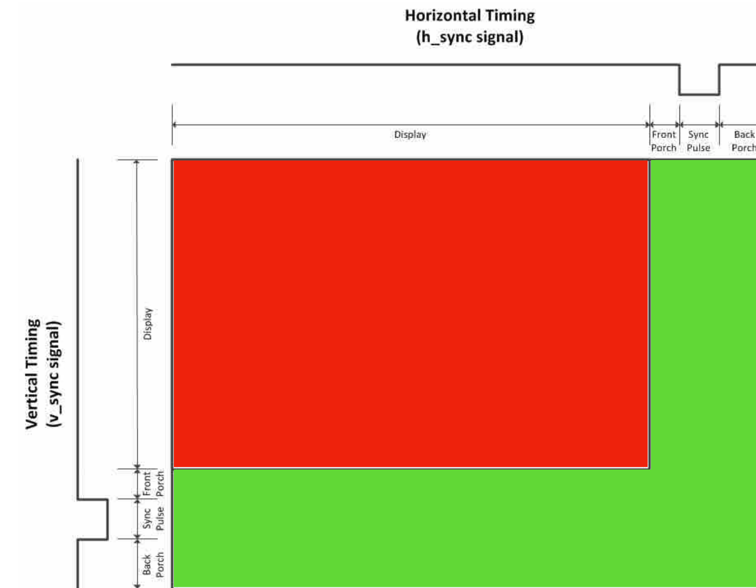



ROBOMART.COM

VGA

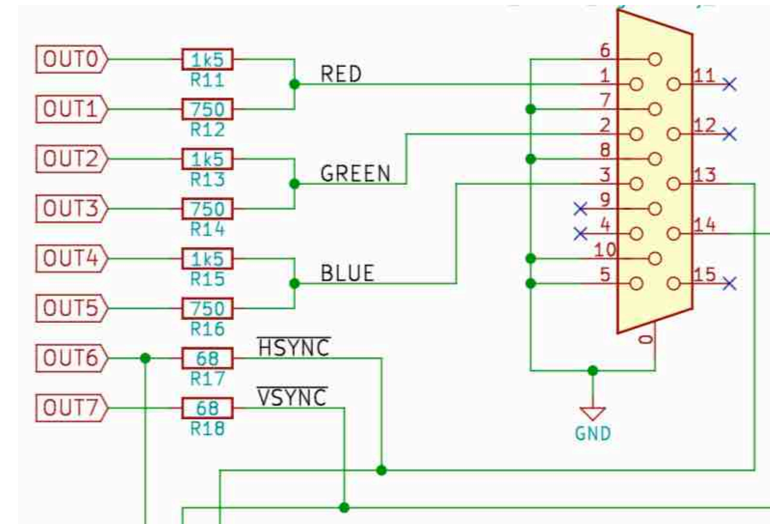
- Not as difficult as you might think..
- 640x480 EGA with 64 colours (6-bit RGB)
- VGA output needs HSYNC and VSYNC
- Since we use RISC, it should be fairly easy to bitbang the signal to the VGA output
- Hence the 6.25MHz clock and the fact that the X register is a counter

VGA

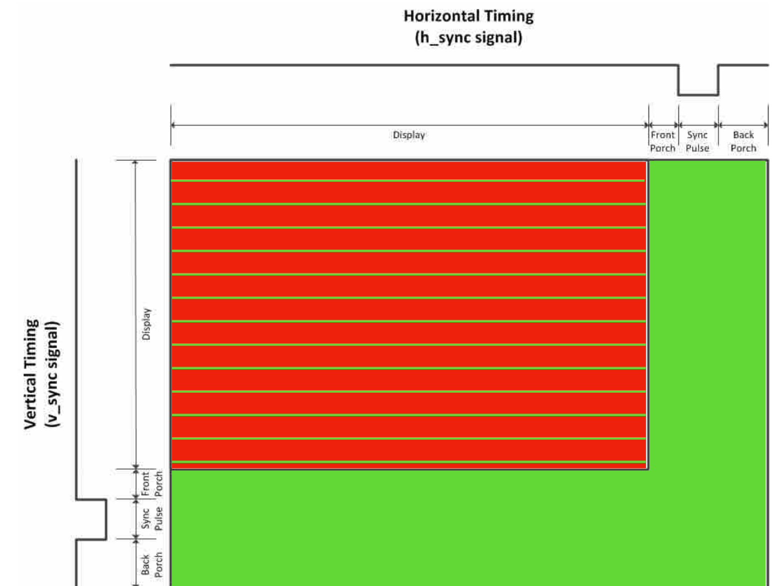


- A lot of time goes into doing the VGA signal, but the system still is fast enough
- Timing is critical
  - Solved by using an interpreter that runs a few opcodes at a time and checks if VGA output needs to be done before running the next few opcodes
    including all the VGA output and timing

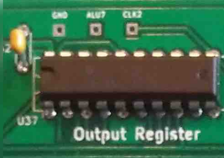
VGA

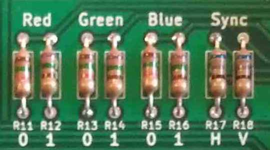


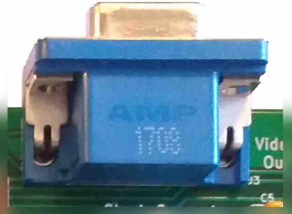
- Because of the speed of the RAM, we use 160 pixels in each scanline instead of 640
- Because of the amount of memory needed for a full image, we repeat each scanline 4 times
  - Or 3, followed by a blank line, for retro effect
- Effective resolution 160x120



LG

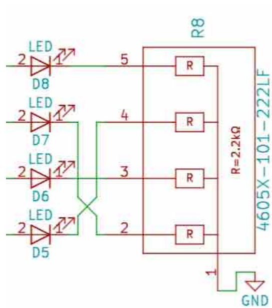

GND
ALU7
CLK2
U37
Output Register

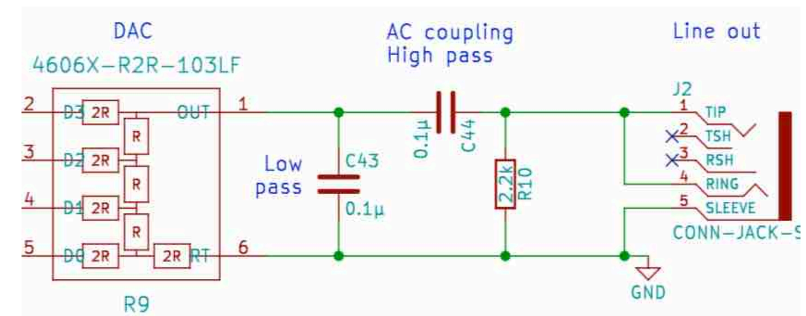

Red
Green
Blue
Sync
R11 R12 R13 R14 R15 R16 R17 R18
0 1 0 1 0 1 H V

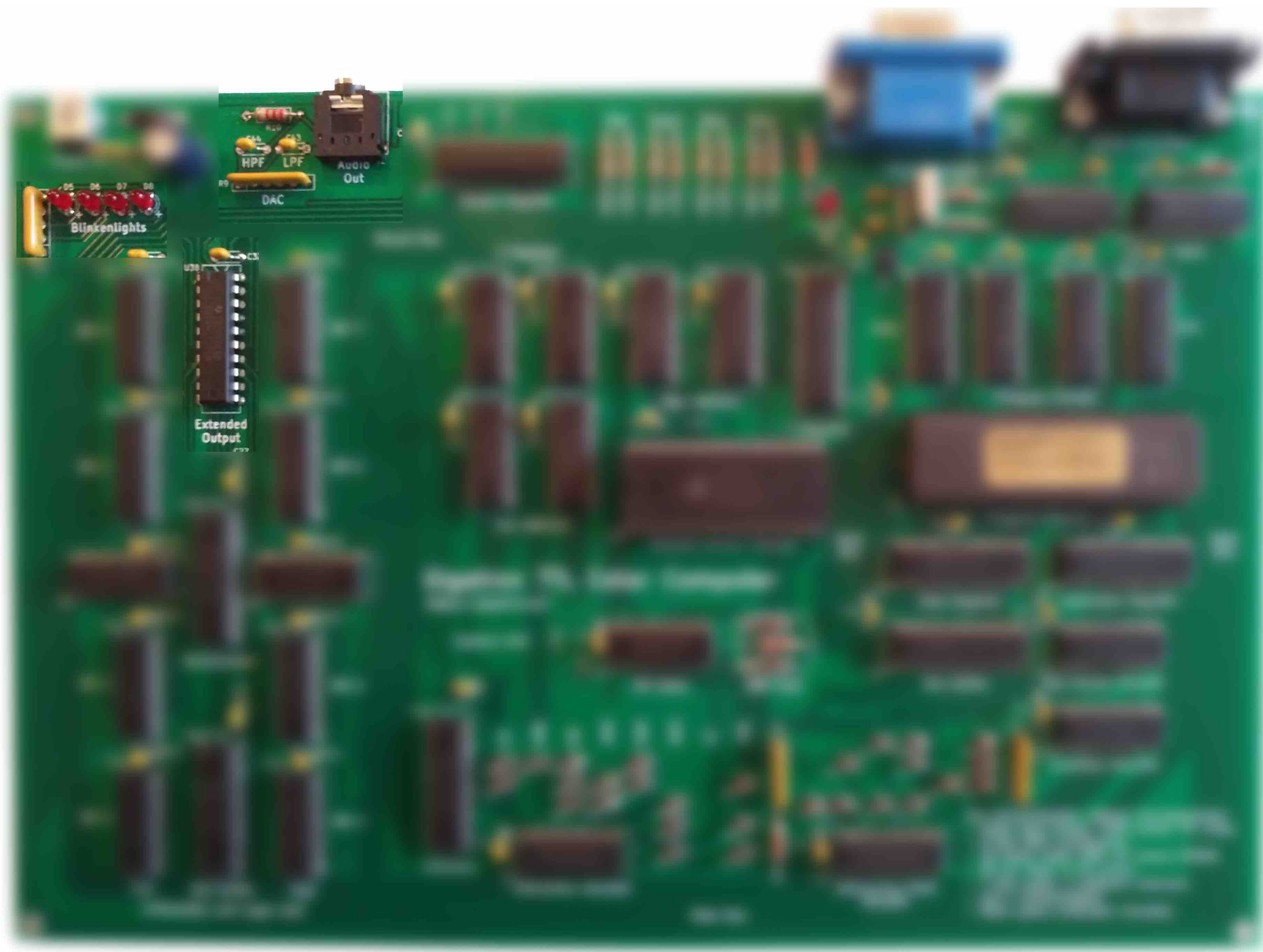

AMP
1708

Can we do more output?

- During SYNC, no RGB screen output is needed
- Reroute output from VGA to something else during SYNC!
- Use the 8 data bits during HSYNC to drive 4 LEDs and a 4-bit audio DAC



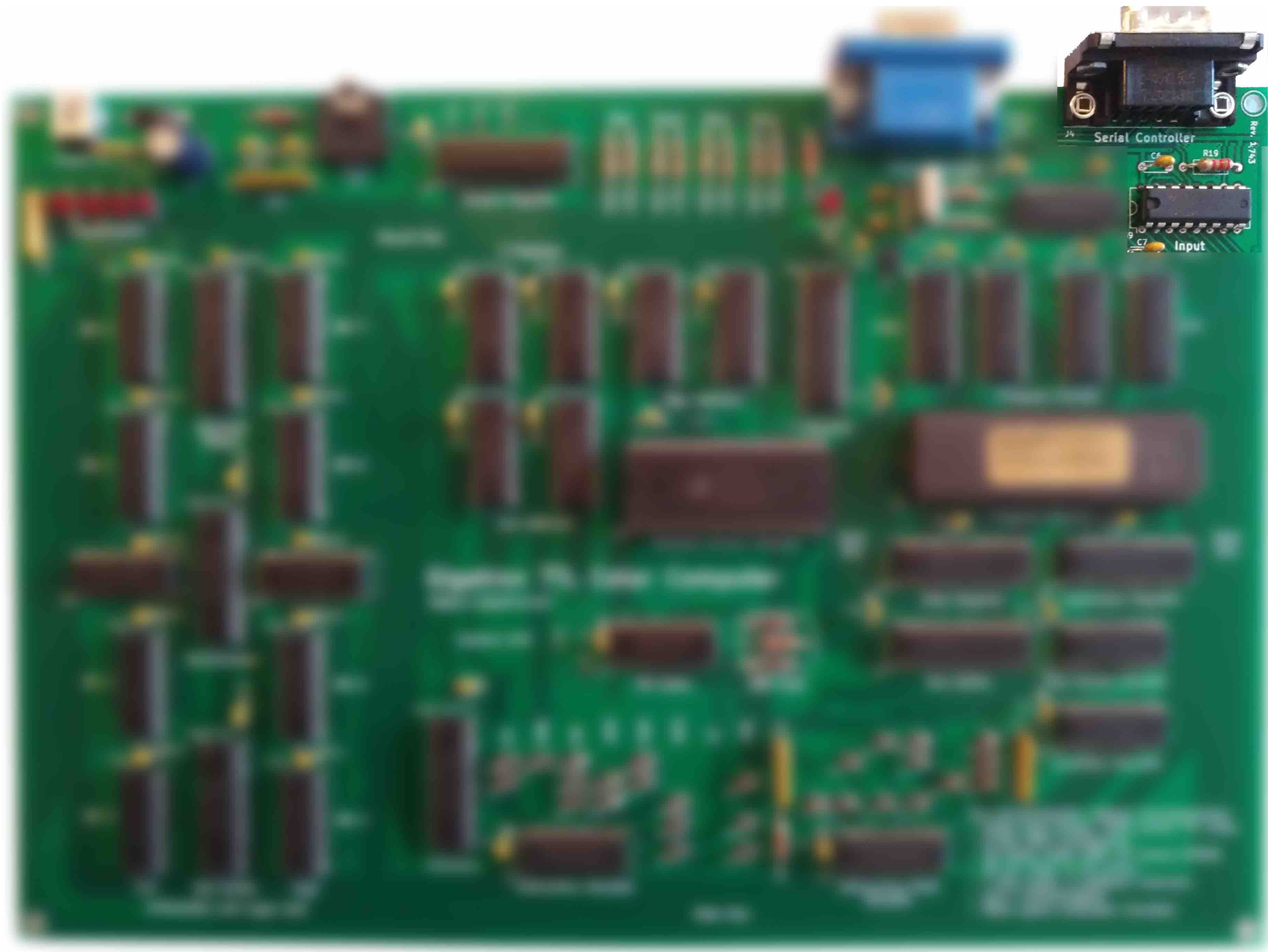



HPF
LPF
Audio
Out
DAC
Blinkenlights
Extended
Output

Input

- Famicom (NES) controller
  - Uses simple DB-9 connector
  - Contains some electronics…
  - A bit per button needs to be read
- Piggyback onto SYNC signals, too

- VSYNC starts Famicom polling
- HSYNC polls one bit into a shift register
- Software needs to read this shift register after 8 HSYNCs have passed


J4
Serial Controller
Rev. 1743
C6
R19
C7
Input

What have we got?

- CPU without a microprocessor chip
- RISC, running at 6.25MHz
- 32kB RAM
- VGA output, sound, blinkenlights, controller input
- Interpreter takes away the burden of getting timing right

SOFTWARE

Programming

- ~~Approach 1: write assembly and burn it to EPROM~~
- ~~Approach 2: compile software written in a higher language to assembly and burn it to EPROM~~
- Approach 3: compile software written in a (somewhat) higher language to an intermediate format and burn it to EPROM, use an interpreter to interpret and run the code

Programming

- Somewhat high-level language: Gigatron Control Language (GCL)

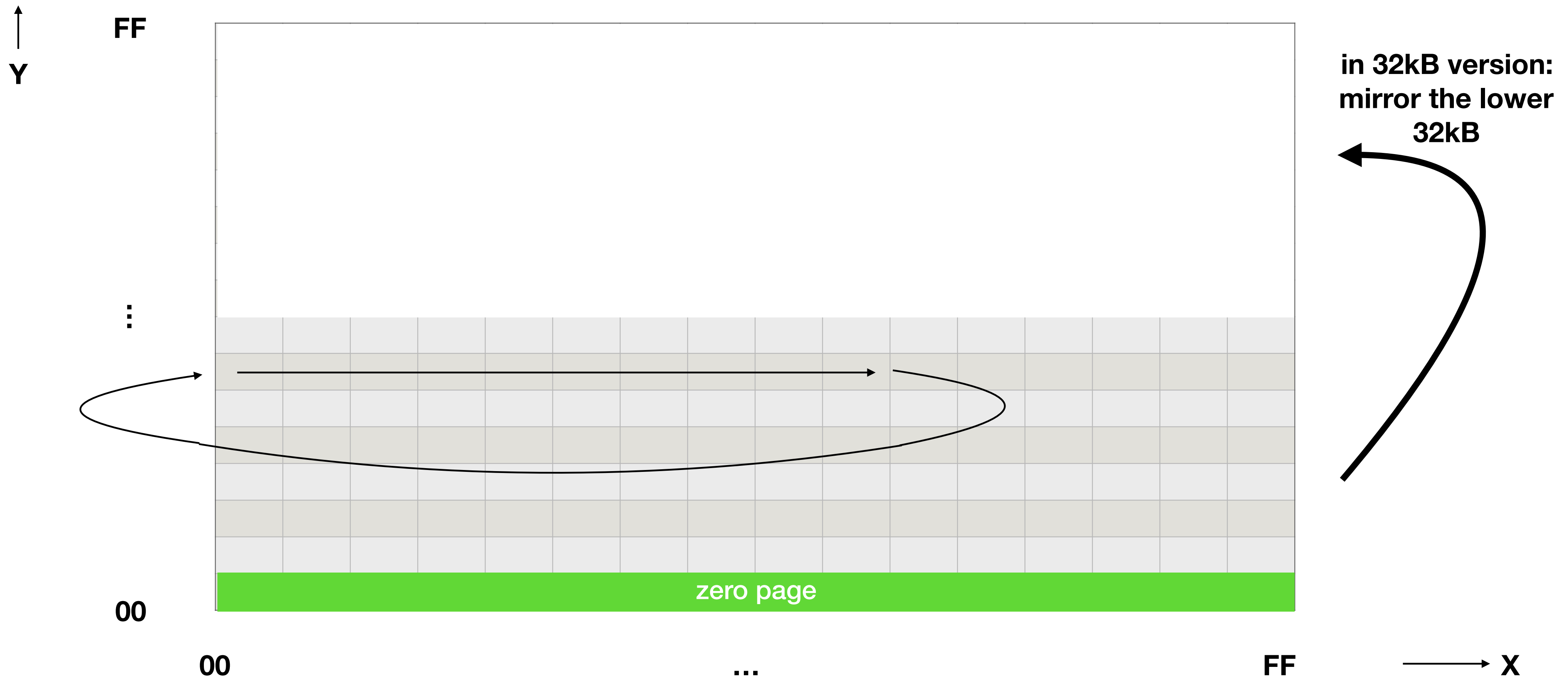
```
gcl1 {GCL version}

{Function to draw binary value as pixels}

[def                                {10 GOTO 80}
  $4448 D= {Middle of screen}       {20 D=$4448: REM MIDDLE OF SCREEN}
  [do
    C [if<0 15 else 5] D.           {30 IF C<0 POKE D,15 ELSE POKE D,5}
    C C+ C=                         {40 C=C+C}
    D 1+ D=                         {50 D=D+1}
    -$4458 D+ if<0 loop]            {60 IF D<$4458 THEN 30}
  ret                               {70 RETURN}
] Plot=
```

Programming

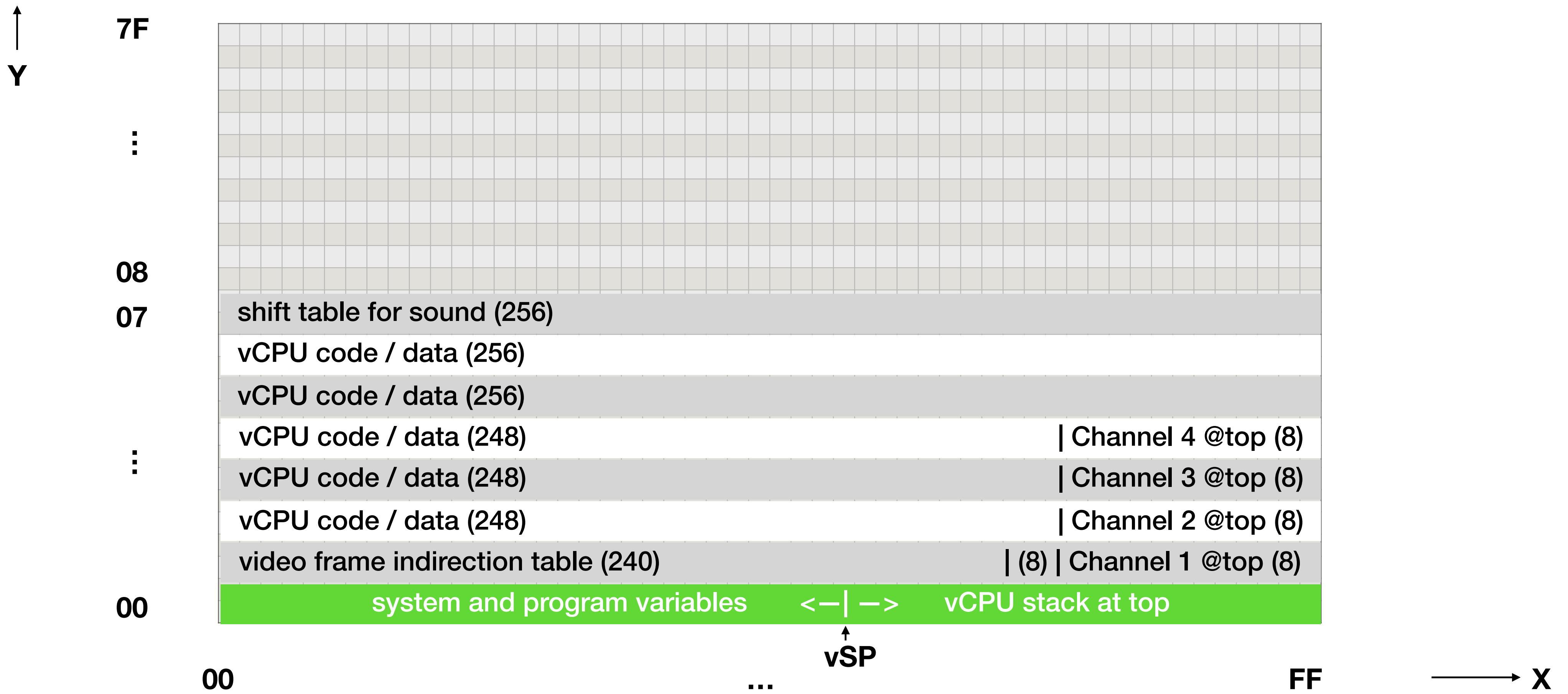
{Compute largest 16-bit Fibonacci number and plot it on screen}

```
[do
  0 A=             {80 A=0}
  1 B=             {90 B=1}
  [do
    A B+ C=        {100 C=A+B}
    B A= C B=      {110 A=B: B=C}
    if>0 loop]     {120 IF B>0 THEN 100}
  Plot!            {130 GOSUB 20}
  loop]            {140 GOTO 80}
```

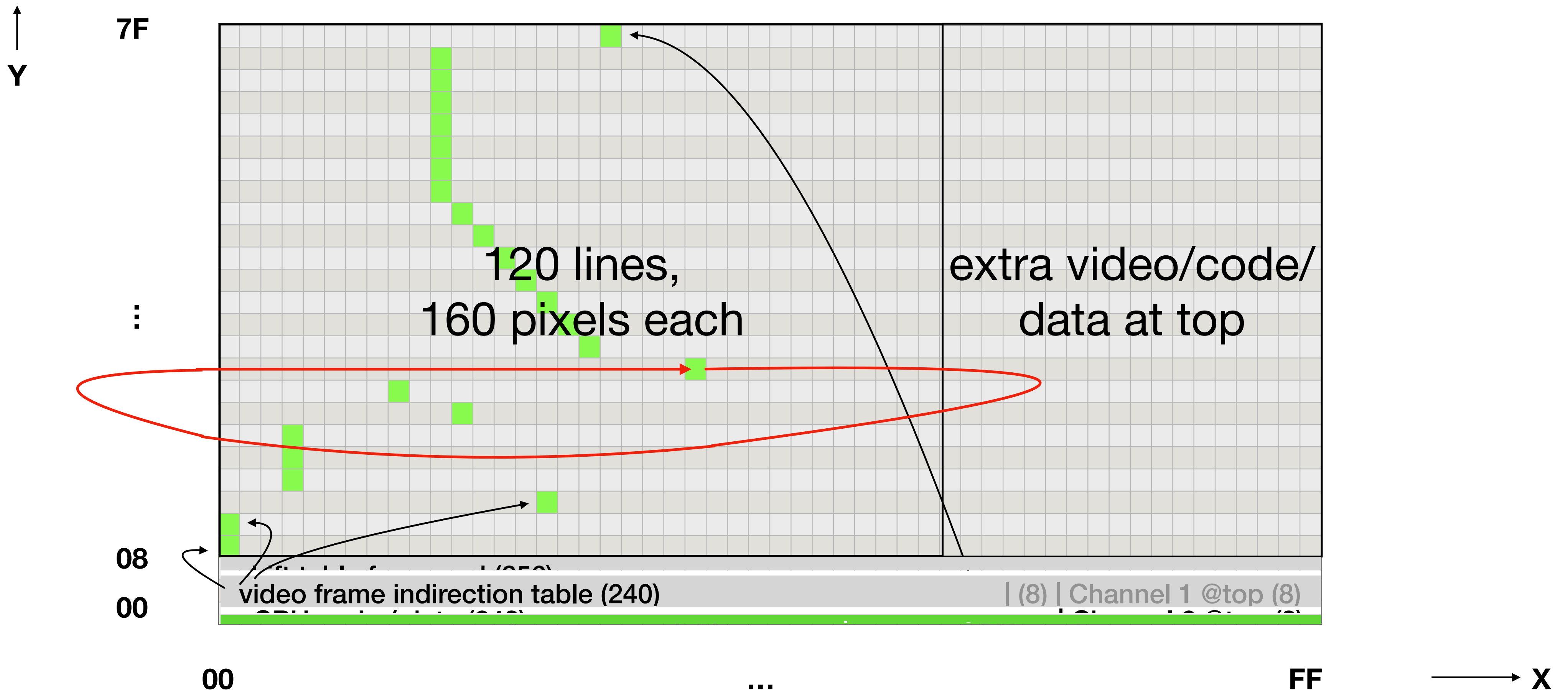
RAM Memory map and [Y, X++]



RAM Memory map



RAM Memory map

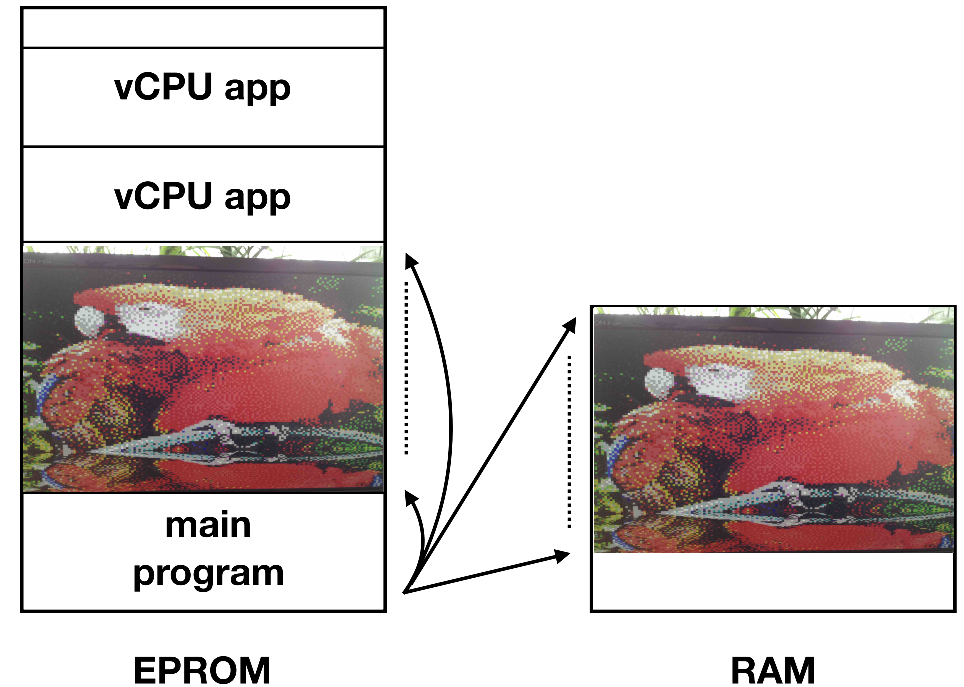


Interpreter

- Emulates a virtual PC
  - Runs vCPU instructions from RAM (using a vPC)
  - Emulates a 16-bit architecture (with a 16-bit vAC, vSP)
- Each instruction is emulated, track is kept of the amount of clock cycles
- A bit like SWEET16 on Apple

Result

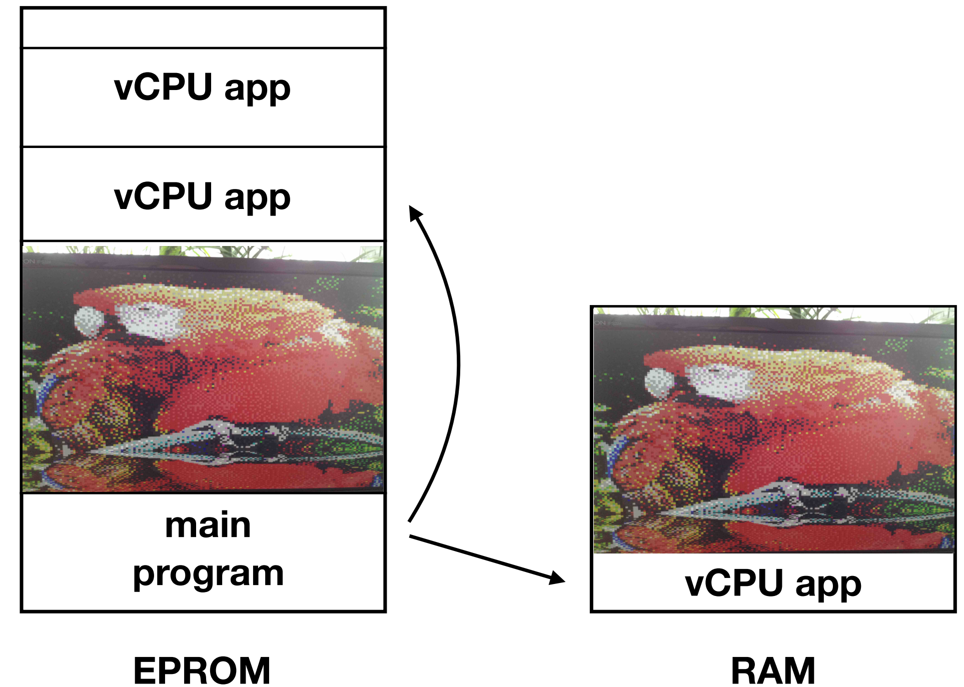
- 16 bit CPU
- Von Neumann architecture
- On an 8 bit CPU
- With Harvard Architecture



Result

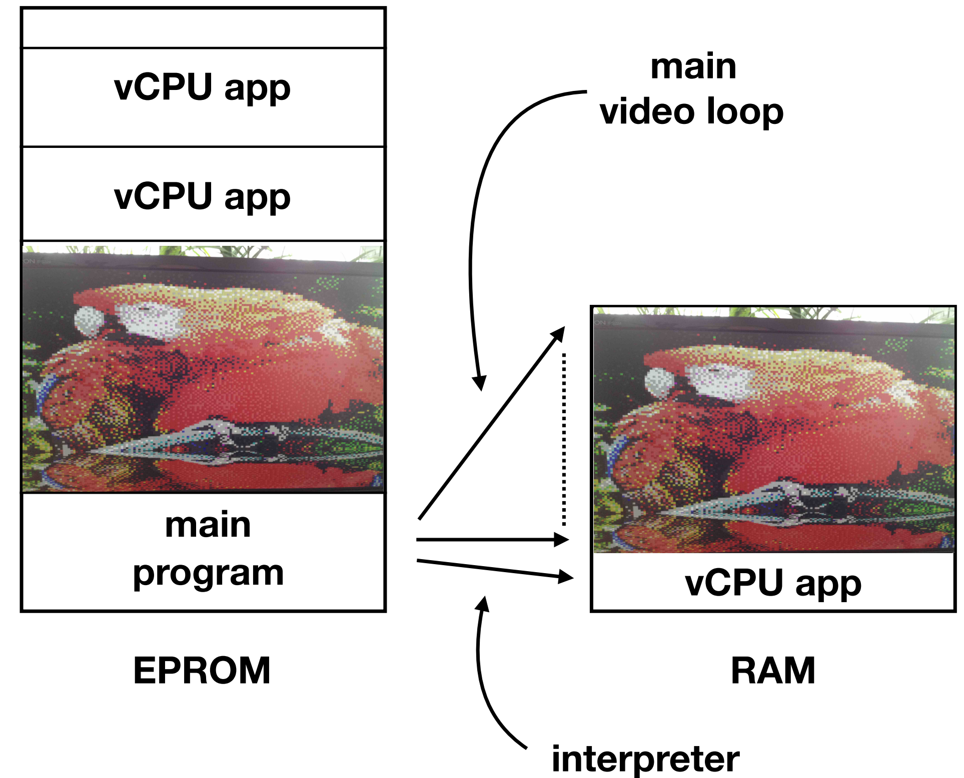
- 16 bit CPU
- Von Neumann architecture

- On an 8 bit CPU
- With Harvard Architecture

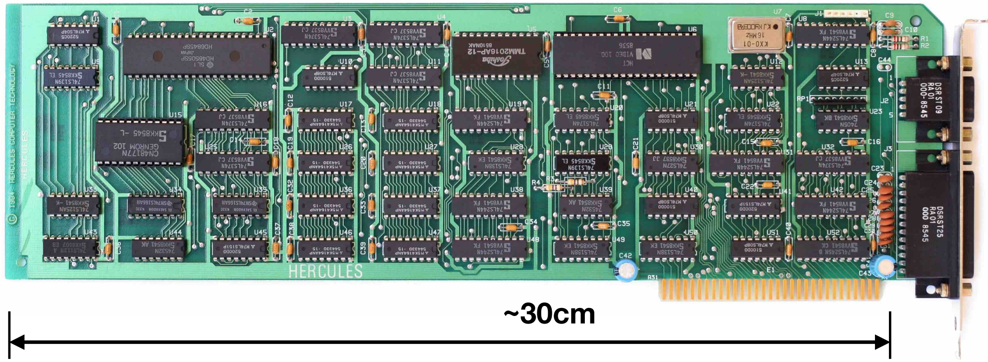


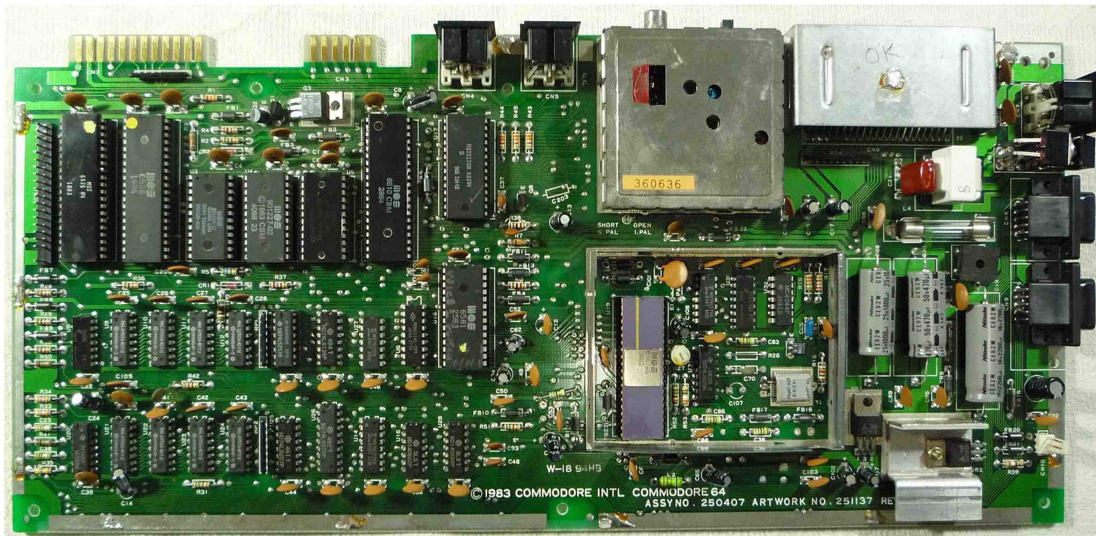
Result

- 16 bit CPU
- Von Neumann architecture
- On an 8 bit CPU
- With Harvard Architecture

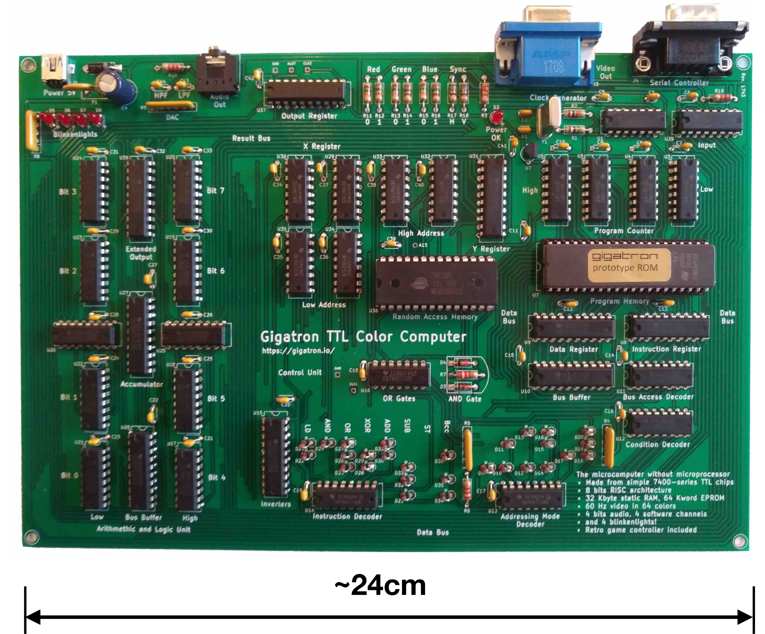


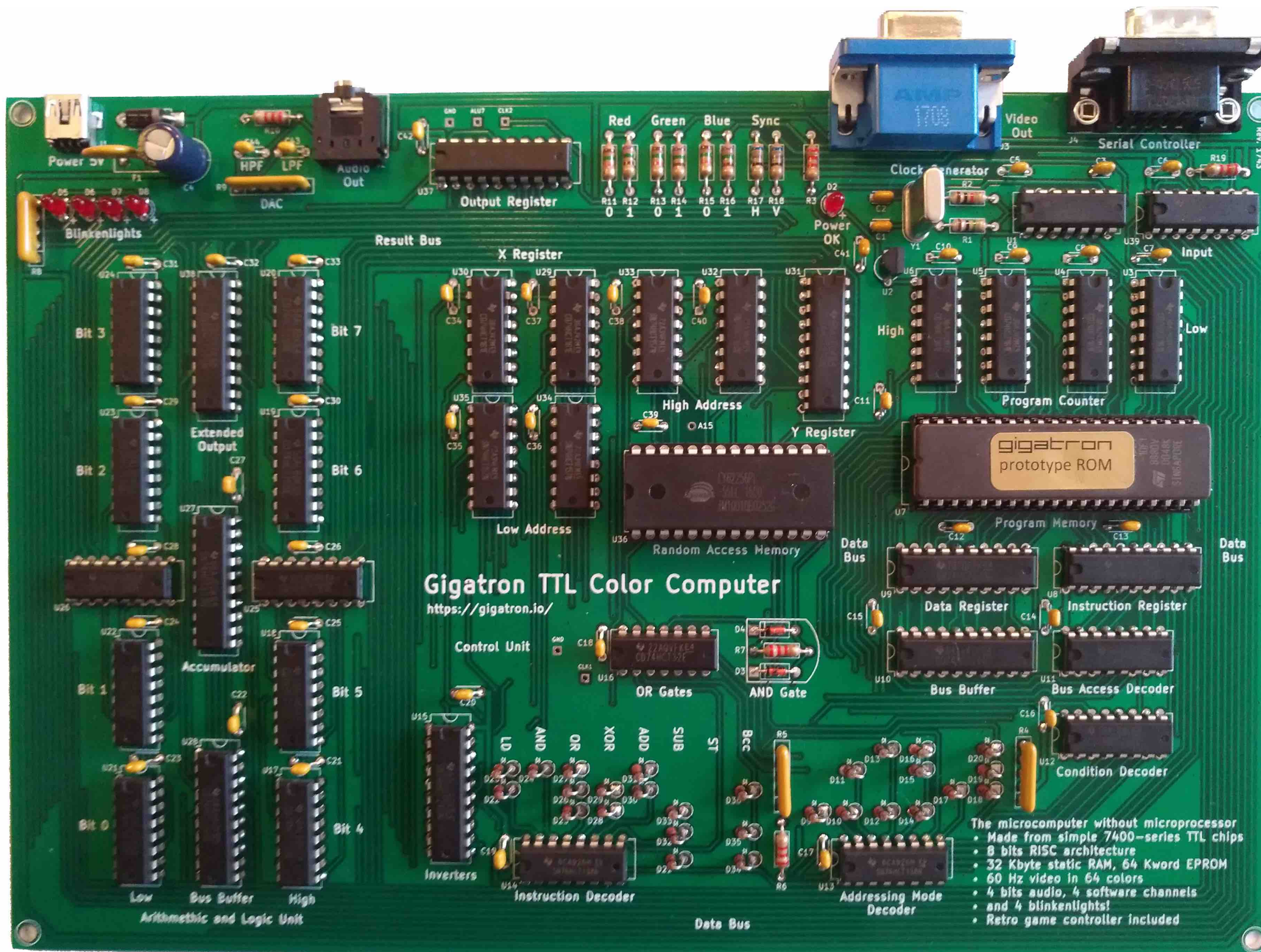
https://gigatron.io/emu/

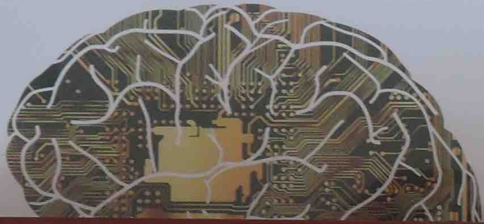




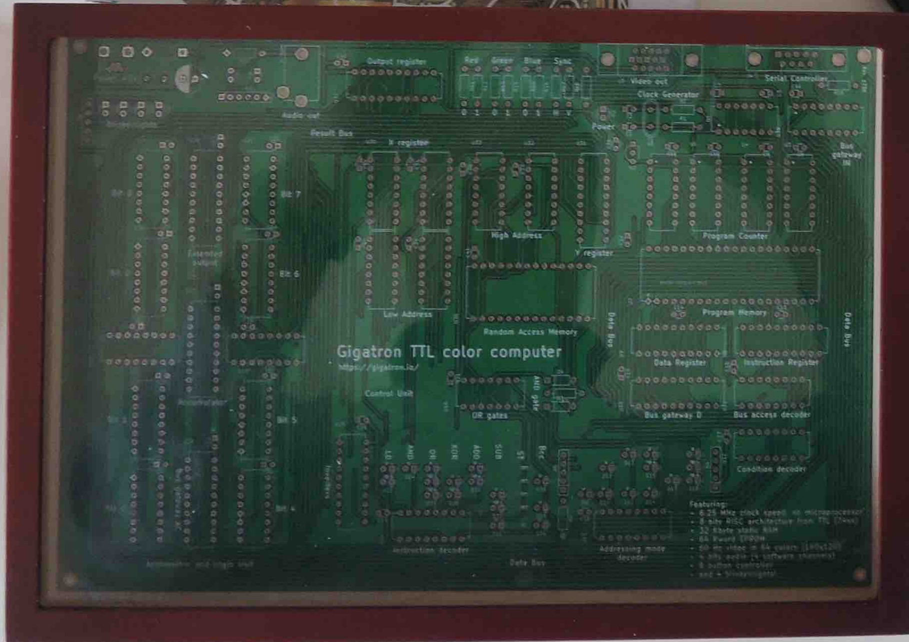
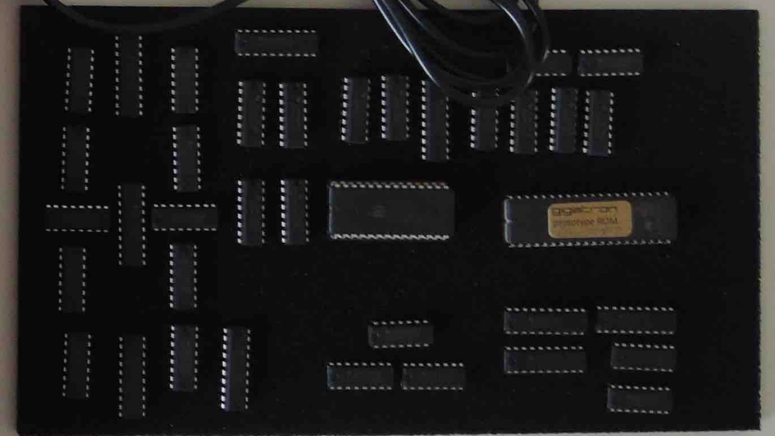
~39cm



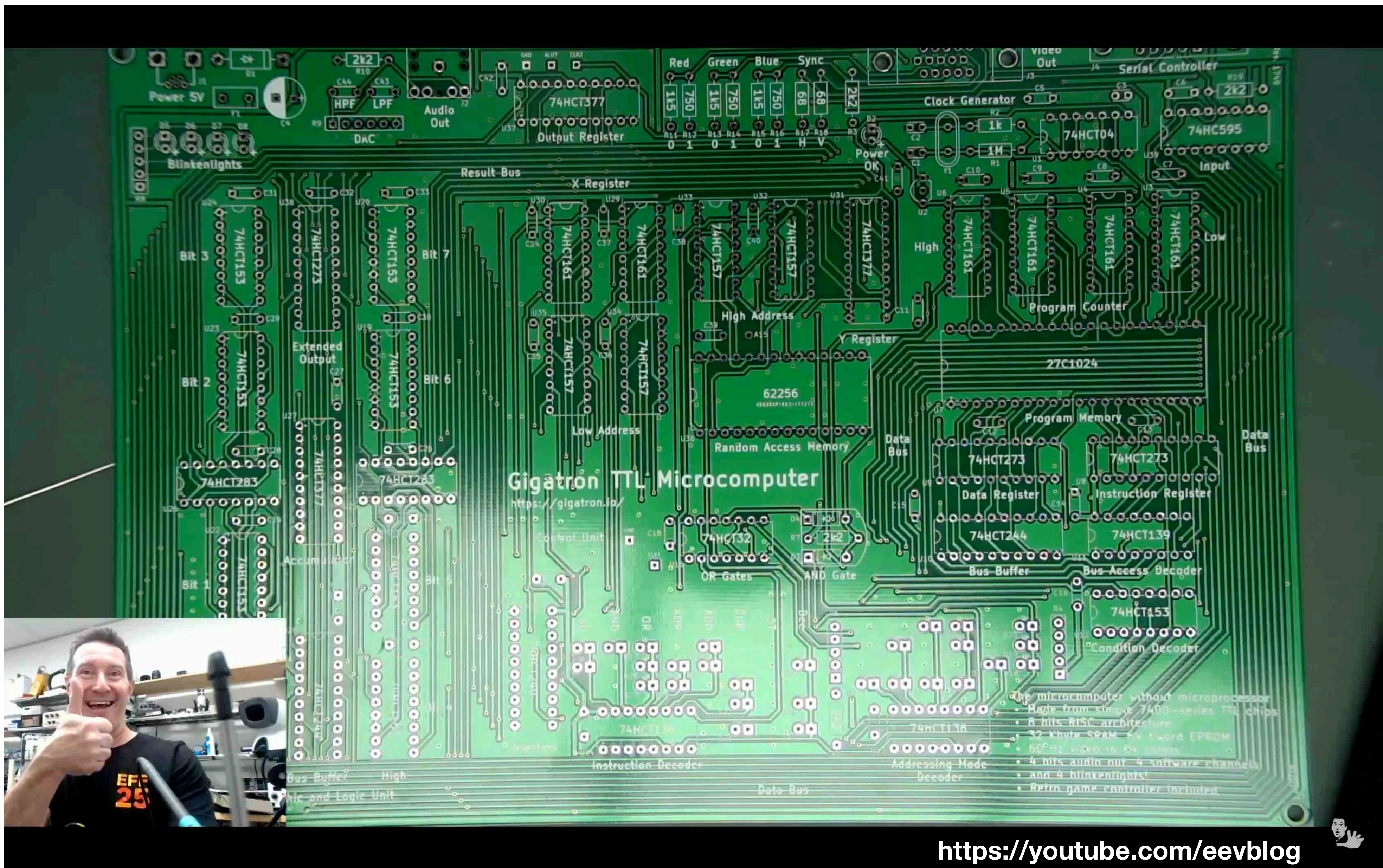

Gigatron TTL Color Computer
https://gigatron.io/
Power 5V
HPF
LPF
Audio Out
DAC
Output Register
Red
Green
Blue
Sync
Video Out
Serial Controller
Clock Generator
Power OK
Blinkenlights
Result Bus
X Register
High Address
Low Address
Y Register
High
Low
Input
Program Counter
gigatron
prototype ROM
Program Memory
Random Access Memory
Data Bus
Data Register
Instruction Register
Bus Buffer
Bus Access Decoder
Condition Decoder
Bit 3
Bit 7
Bit 2
Bit 6
Bit 1
Bit 5
Bit 0
Bit 4
Extended Output
Accumulator
Low
Bus Buffer
High
Arithmethic and Logic Unit
Control Unit
OR Gates
AND Gate
LD
AND
OR
XOR
ADD
SUB
ST
Bcc
Inverters
Instruction Decoder
Addressing Mode Decoder
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


gigatron
The TTL based
microcomputer system

SELECT
START


Gigatron TTL color computer
Output register
Program Counter
Random Access Memory
Program Memory
Data Register
Instruction Register

gigatron

gigatron
The TTL based
microcomputer system
ASSEMBLY MANUAL
It comes in a very neat looking little box,
and a fantastic manual.
https://youtube.com/adric22


Gigatron TTL Microcomputer
https://gigatron.io/
Power 5V
Blinkenlights
DAC
HPF
LPF
Audio Out
74HCT377
Output Register
Red
Green
Blue
Sync
Result Bus
X Register
Power OK
Clock Generator
74HCT04
Serial Controller
74HC595
Input
Video Out
Bit 3
Bit 2
Bit 1
Bit 7
Bit 6
74HCT153
74HCT273
74HCT283
74HCT377
Extended Output
Accumulator
74HCT161
74HCT157
Low Address
High Address
Y Register
62256
Random Access Memory
High
Low
Program Counter
27C1024
Program Memory
Data Bus
74HCT273
Data Register
Instruction Register
74HCT32
OR Gates
AND Gate
74HCT244
Bus Buffer
74HCT139
Bus Access Decoder
74HCT153
Condition Decoder
74HCT138
Instruction Decoder
Addressing Mode Decoder
Data Bus
https://youtube.com/eevblog

GIGATRIS

LEDS:
FPS: 60.00
XOUT: 08 IN: FF
Load: 0200 Vars: 0030

blinky.gt1
checkerboard.gt1
checkerboard.vasm
clearscreen.gt1
clearscreen1.vasm
dots.gt1
dots.vasm
dots2.gt1
dots2.vasm
life.gt1
life.vasm
life1.gt1
life1.vasm
life2.gt1
life2.vasm
life3.gt1
life3.vasm
life4.vasm
lines.gt1
lines.vasm
rom_test.vasm
sprites_test0.vasm
sprites_test1.vasm
sprites_test2.vasm
sys_test.gt1
sys_test.vasm
sys_test1.vasm

48 08 00 08 7C 00 85 00
B2 00 B3 00 E2 00 E3 00

Mode: Load
Ver: 0.3.6

24 Sprites, 10x10 pixels, 8x8 data, 1 pixel border, no backround restore or save, no mask, no trasparency, only works on solid colour backgrounds...still very cool!

https://github.com/at67/gigatron-rom/tree/master/Emu

Ready
hello
PS/2 keyboard!
LG
Gigatron TTL Color Computer

gigatron.io

walter@gigatron.io