v.20180217

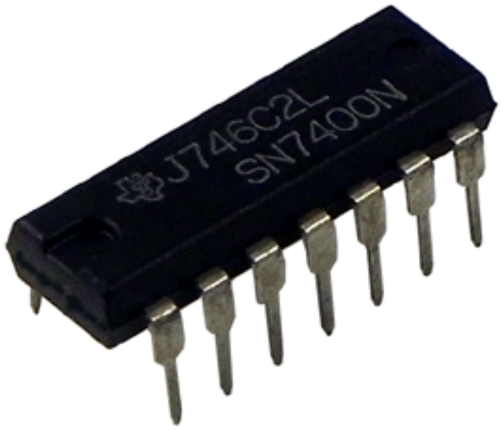
HACKER HOTEL

gigatron

the TTL microcomputer

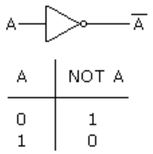
marcelk@gigatron.io

walter@gigatron.io

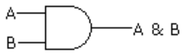

J746C2L
SN7400N


PU 81

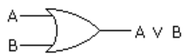

RIGOL
DS1054 Z
OSCILLOSCOPE
UltraVision
4Channel
50MHz 1GSa/s
MENU
CLEAR
AUTO
RUN STOP
SINGLE
Measure
Acquire
Storage
Cursor
Display
Utility
Help
IO Setting
Beeper
Off
Language
English
Pass/Fail
System
Zoom:100.0ns
CH1
CH2
CH3
CH4
MATH
REF
MODE
FORCE



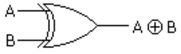
A	NOT A
0	1
1	0



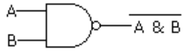
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1



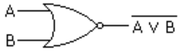
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1



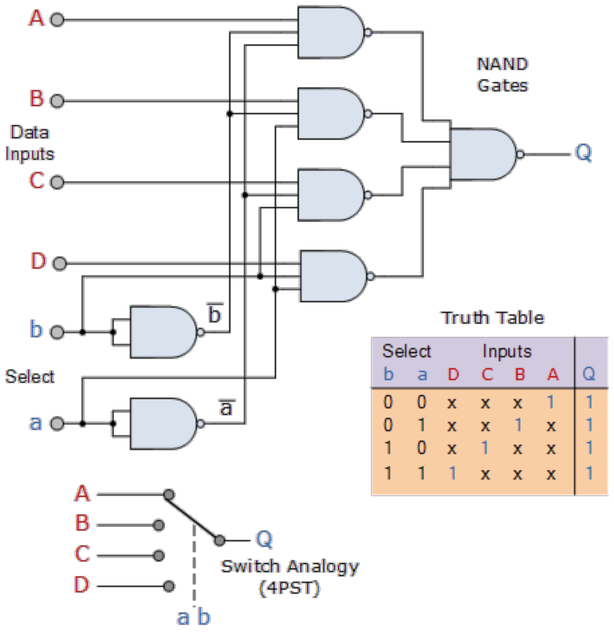
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



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

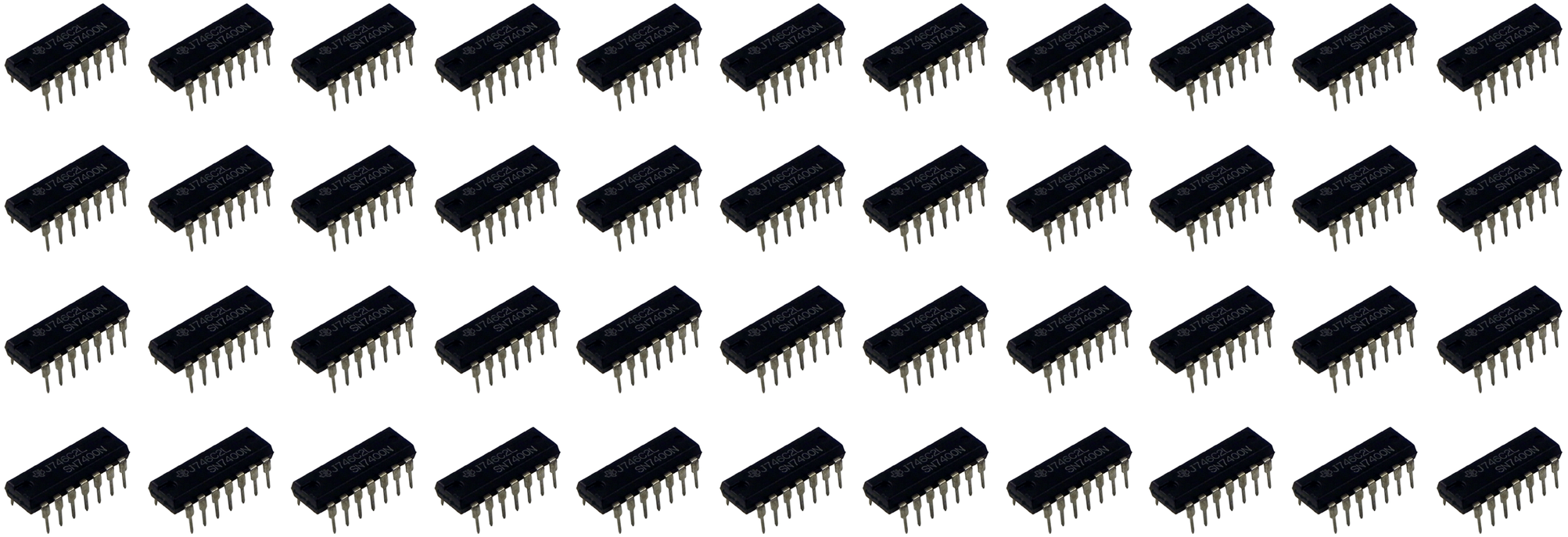


A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

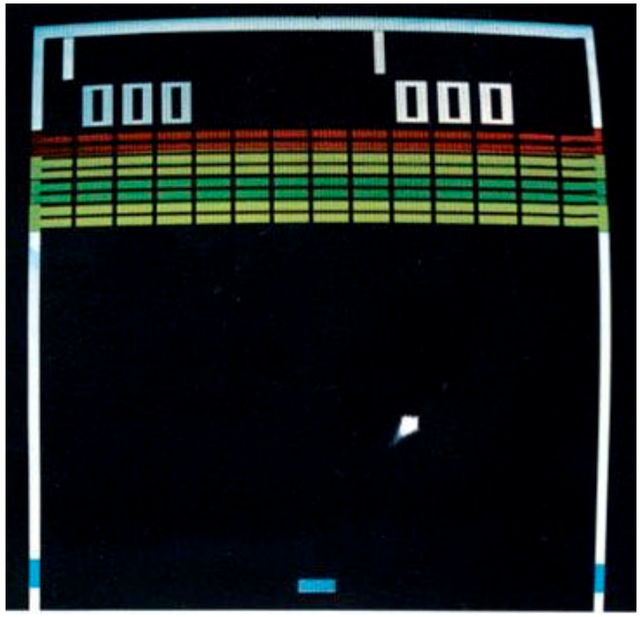


Truth Table

Select b	Select a	Inputs D	Inputs C	Inputs B	Inputs 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

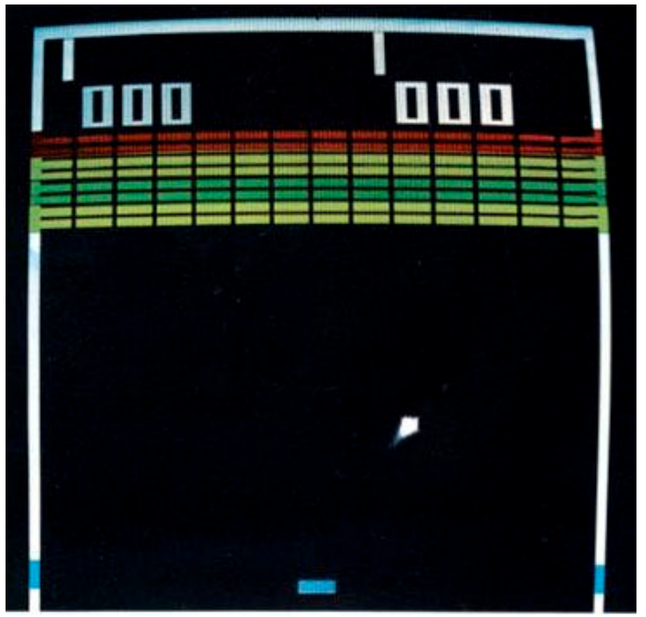

SN7400N


BREAKOUT


000
000

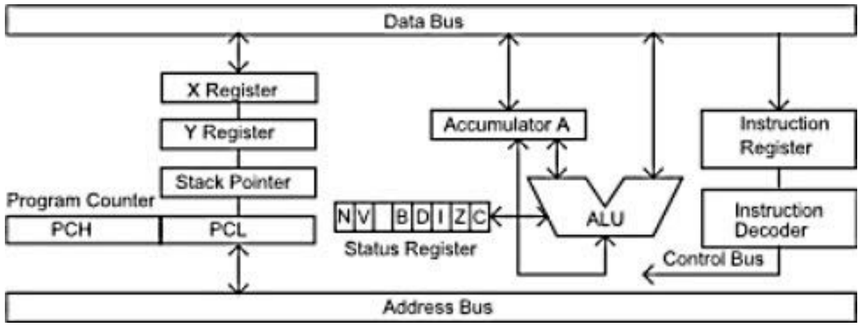

PLAYER UP
BALL IN PLAY
382
000

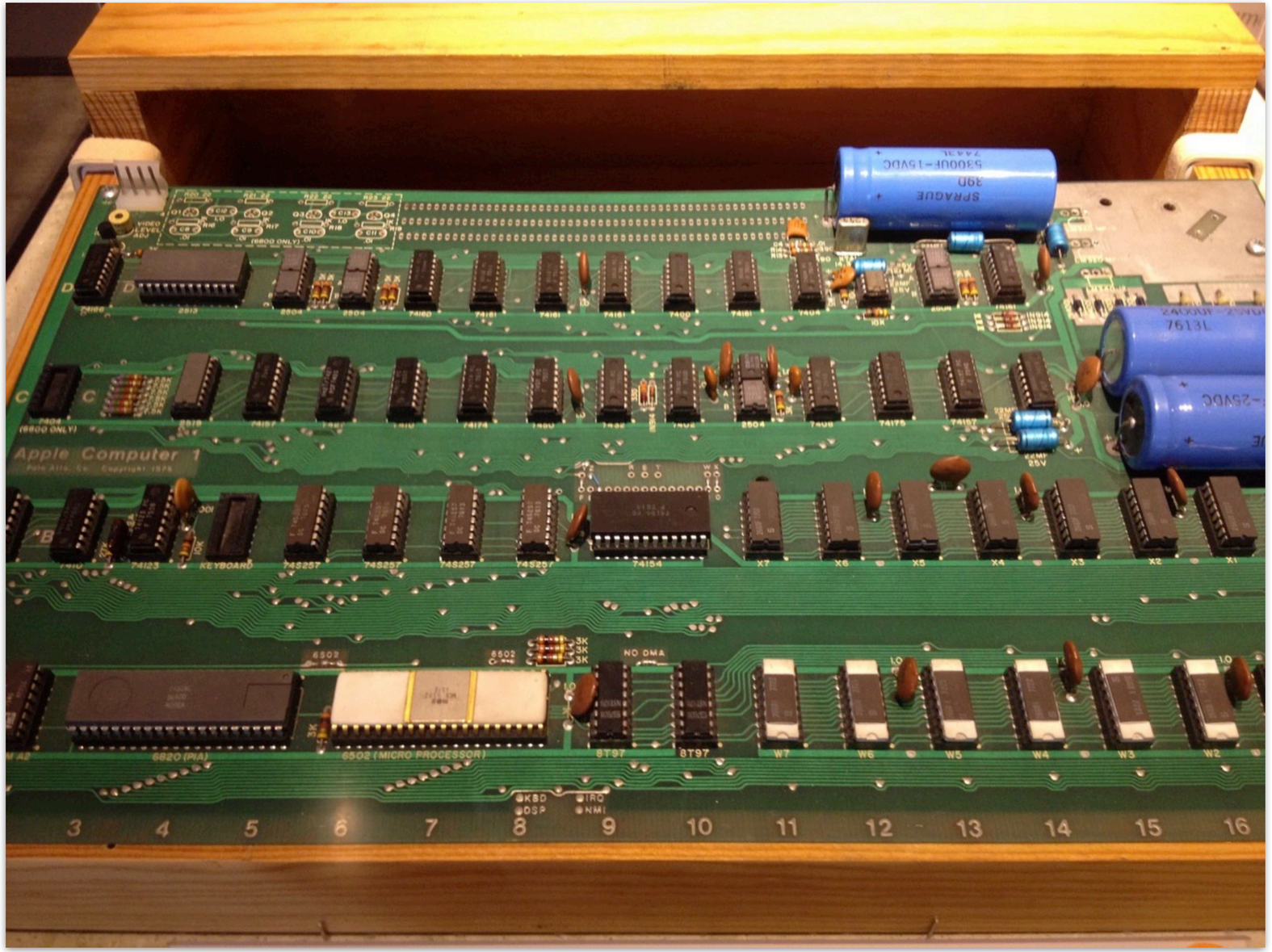

BREAKOUT


000
000


PLAYER UP
BALL IN PLAY
382
000

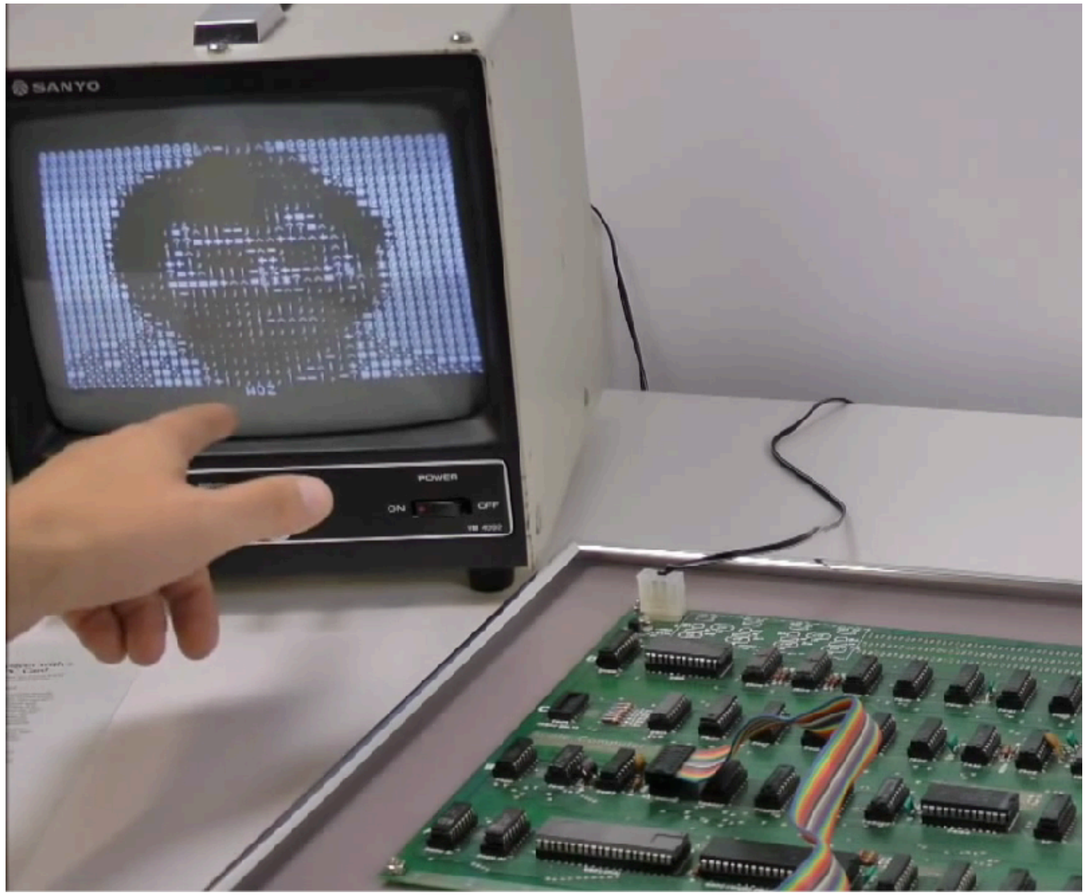
Picture: Larry Nelson



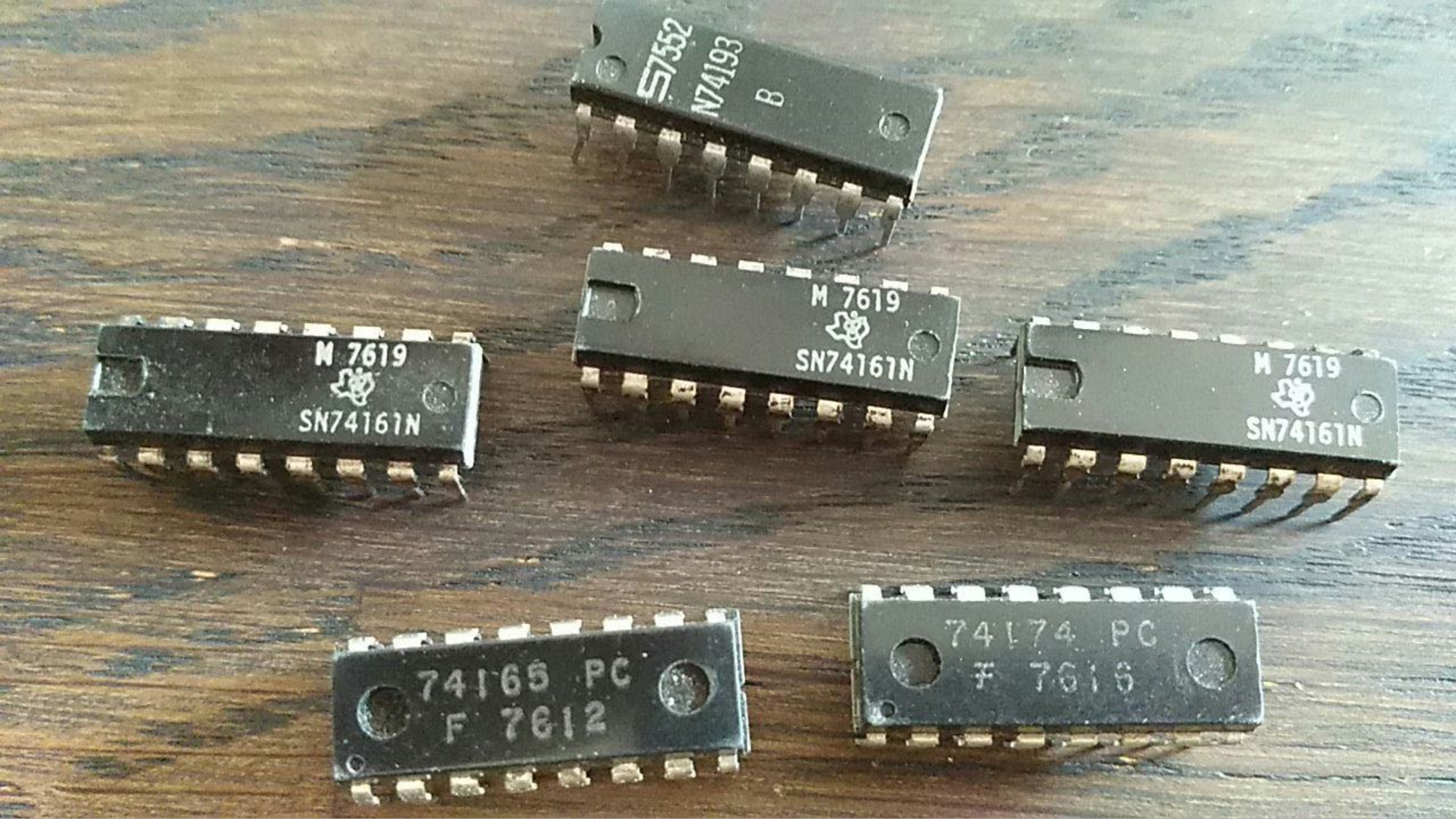


Apple 1

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

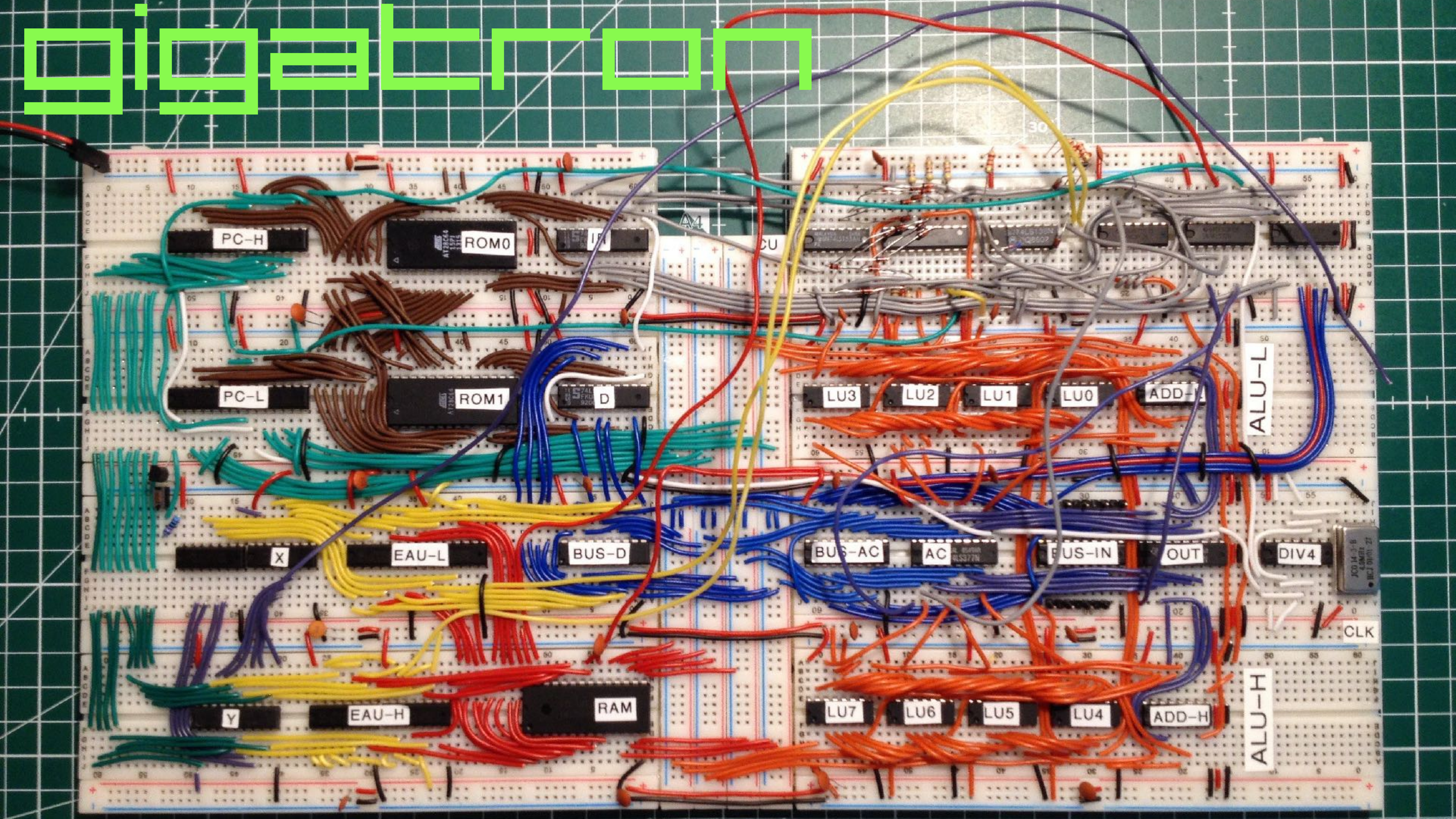


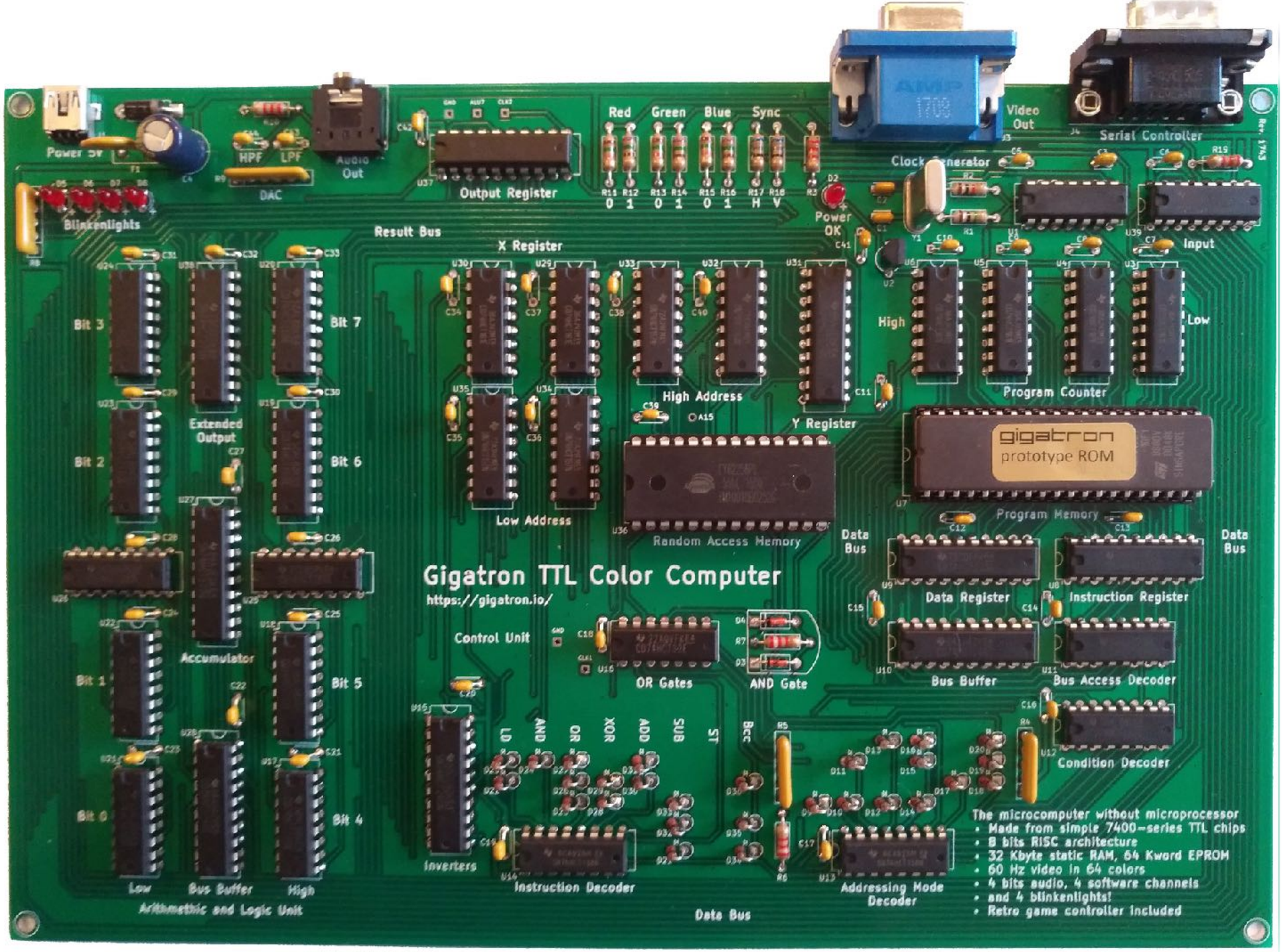
https://www.youtube.com/watch?v=4l8i_xOBTPg


S7552
N74193
B
M 7619
SN74161N
M 7619
SN74161N
M 7619
SN74161N
74165 PC
F 7612
74174 PC
F 7616

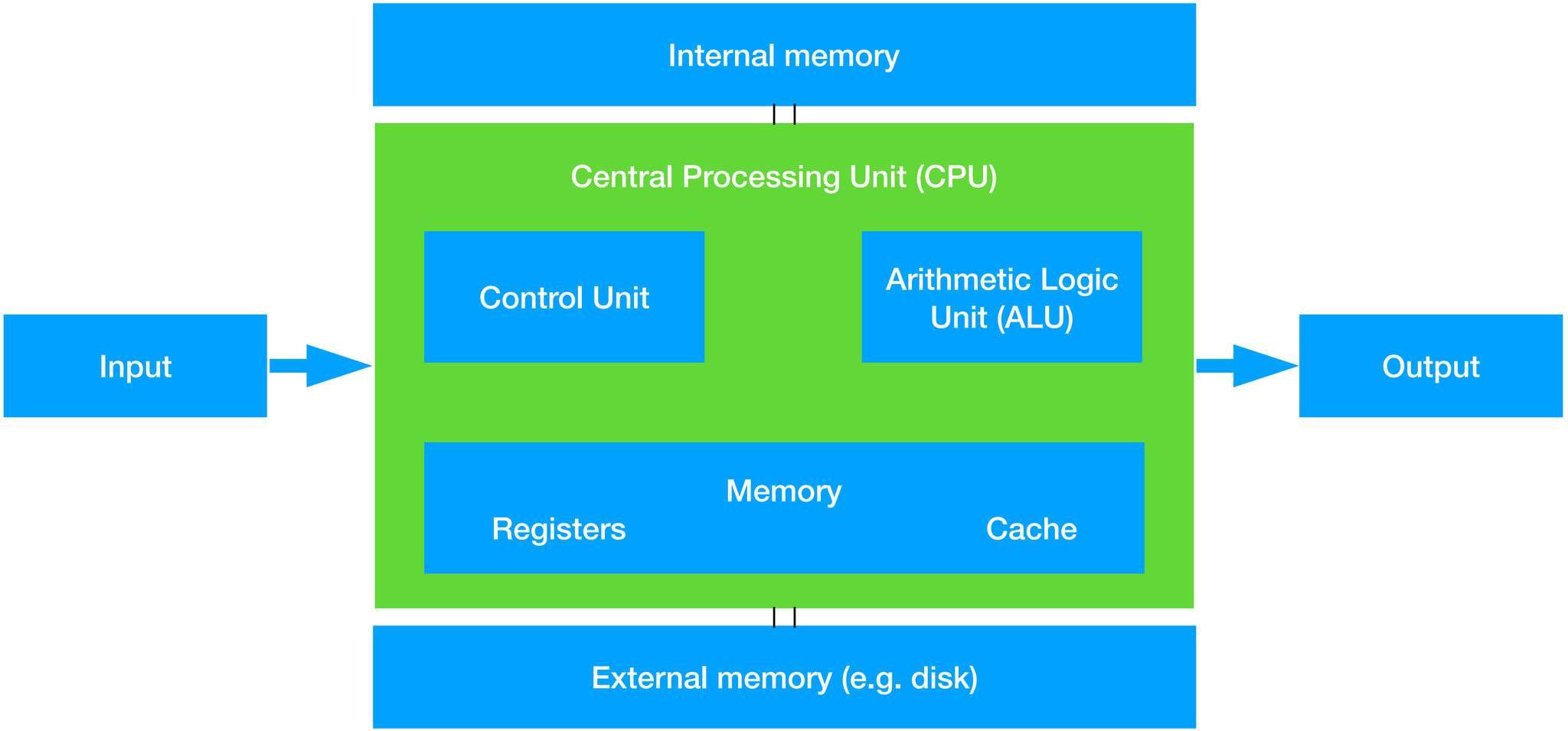
Us in that era..

gigatron

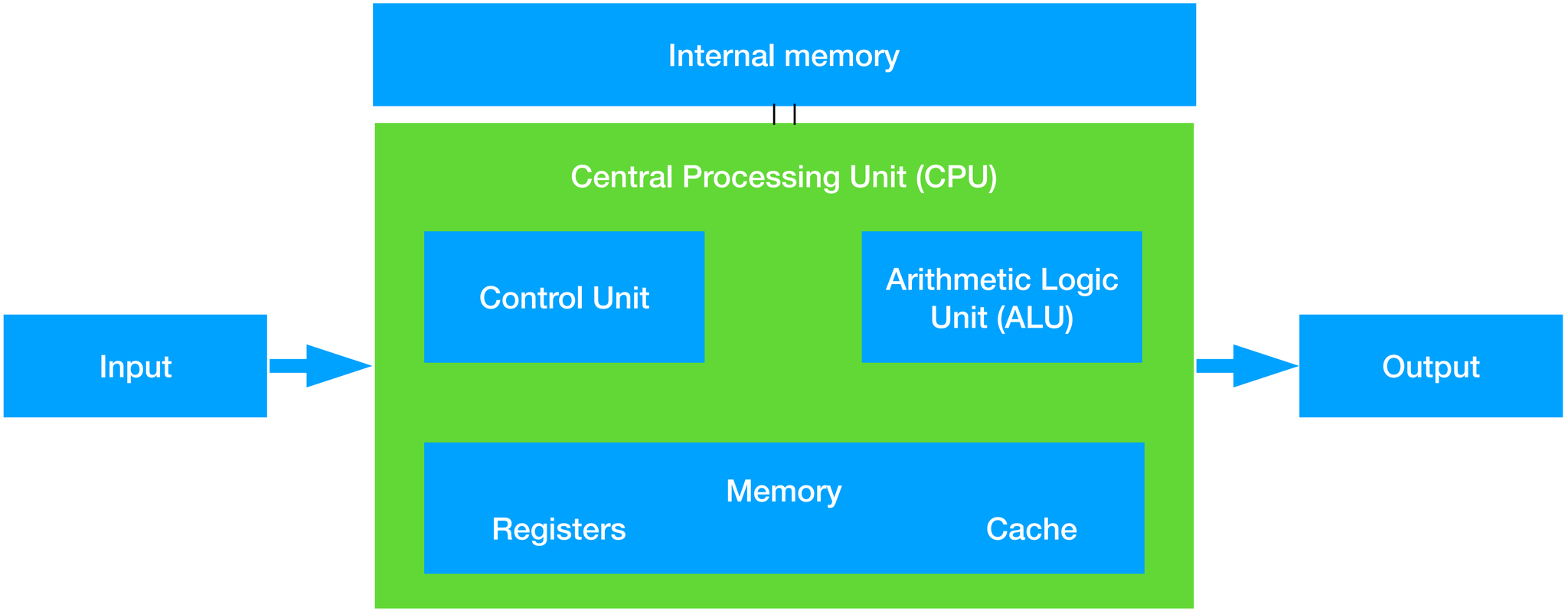

PC-H
ROM0
PC-L
ROM1
D
X
EAU-L
BUS-D
Y
EAU-H
RAM
LU3
LU2
LU1
LU0
ADD-L
ALU-L
BUS-AC
AC
BUS-IN
OUT
DIV4
CLK
LU7
LU6
LU5
LU4
ADD-H
ALU-H


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

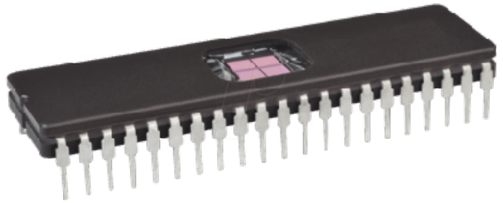
A computer

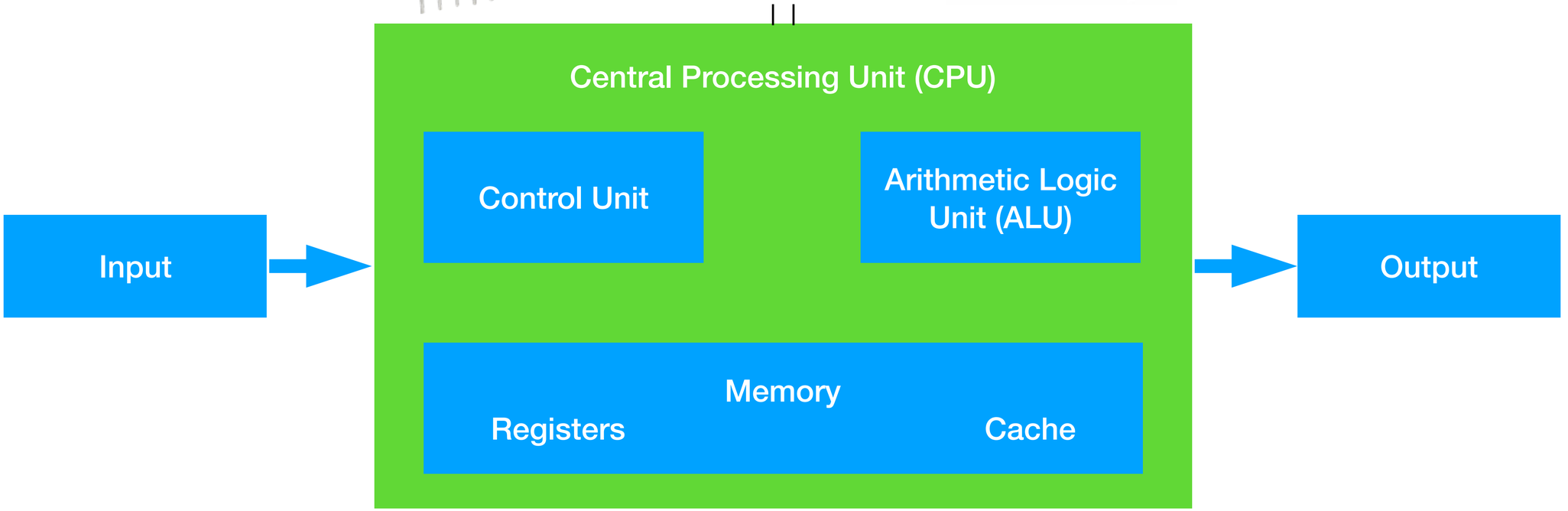

Internal memory
Central Processing Unit (CPU)
Control Unit
Arithmetic Logic Unit (ALU)
Input
Output
Memory
Registers
Cache
External memory (e.g. disk)

A computer

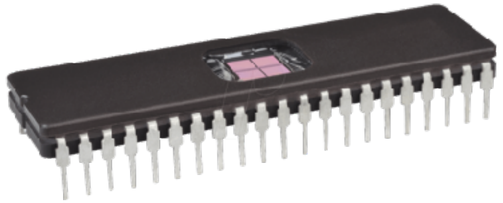

Internal memory
Central Processing Unit (CPU)
Control Unit
Arithmetic Logic Unit (ALU)
Input
Output
Memory
Registers
Cache

A computer




Central Processing Unit (CPU)
Control Unit
Arithmetic Logic Unit (ALU)
Input
Output
Memory
Registers
Cache

A computer



Central Processing Unit (CPU)

Control Unit

Arithmetic Logic Unit (ALU)

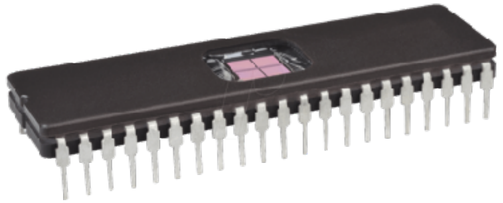
Memory

Registers

Cache

Output

A computer



Central Processing Unit (CPU)

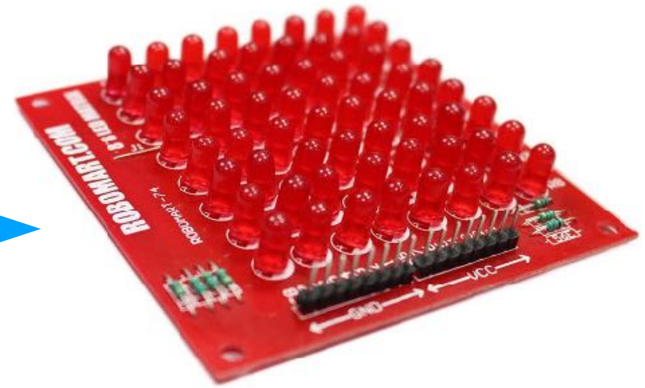
Control Unit

Arithmetic Logic Unit (ALU)

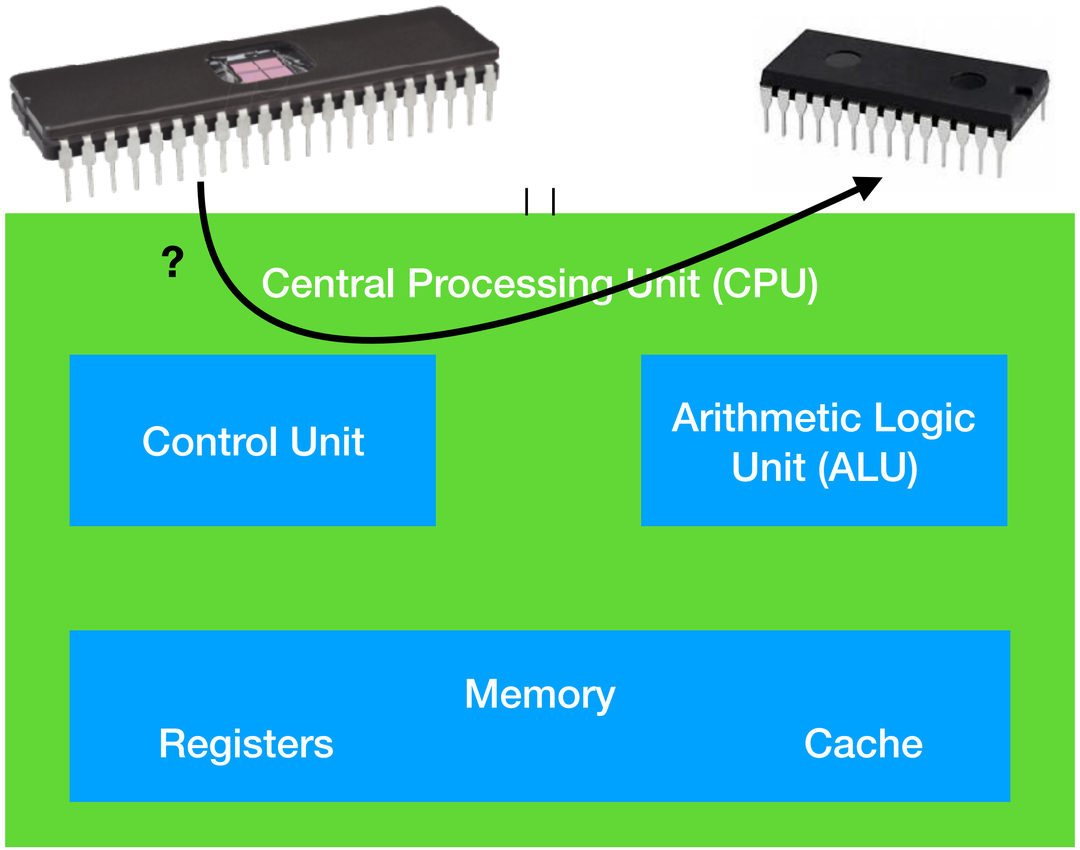
Memory

Registers

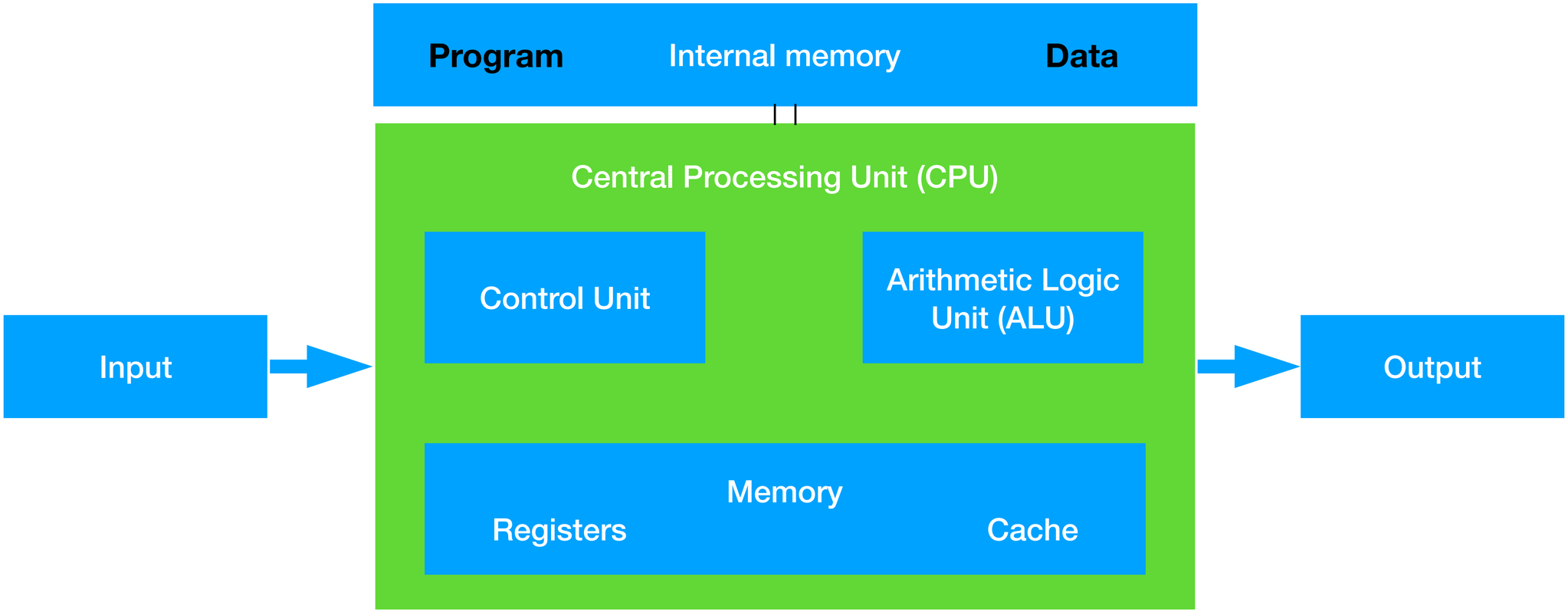
Cache



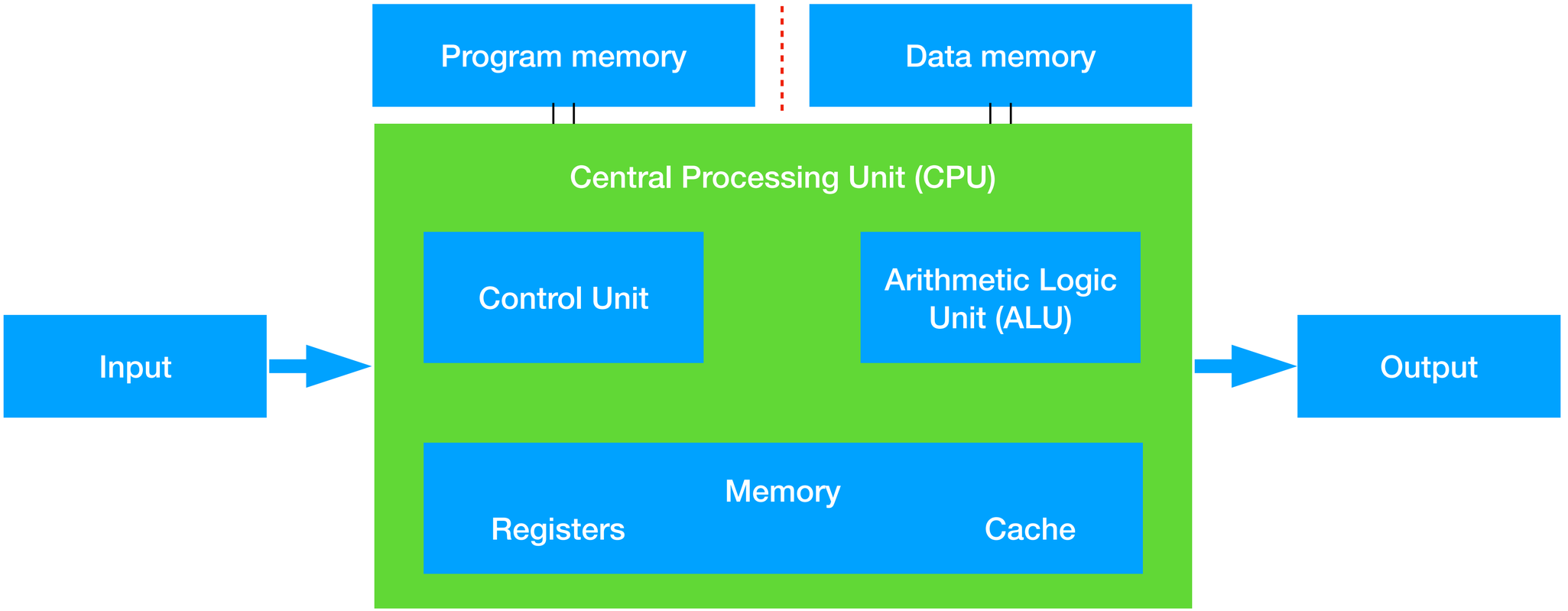
A computer


?
Central Processing Unit (CPU)
Control Unit
Arithmetic Logic Unit (ALU)
Memory
Registers
Cache

Von Neumann


Program
Internal memory
Data
Central Processing Unit (CPU)
Control Unit
Arithmetic Logic Unit (ALU)
Input
Output
Memory
Registers
Cache

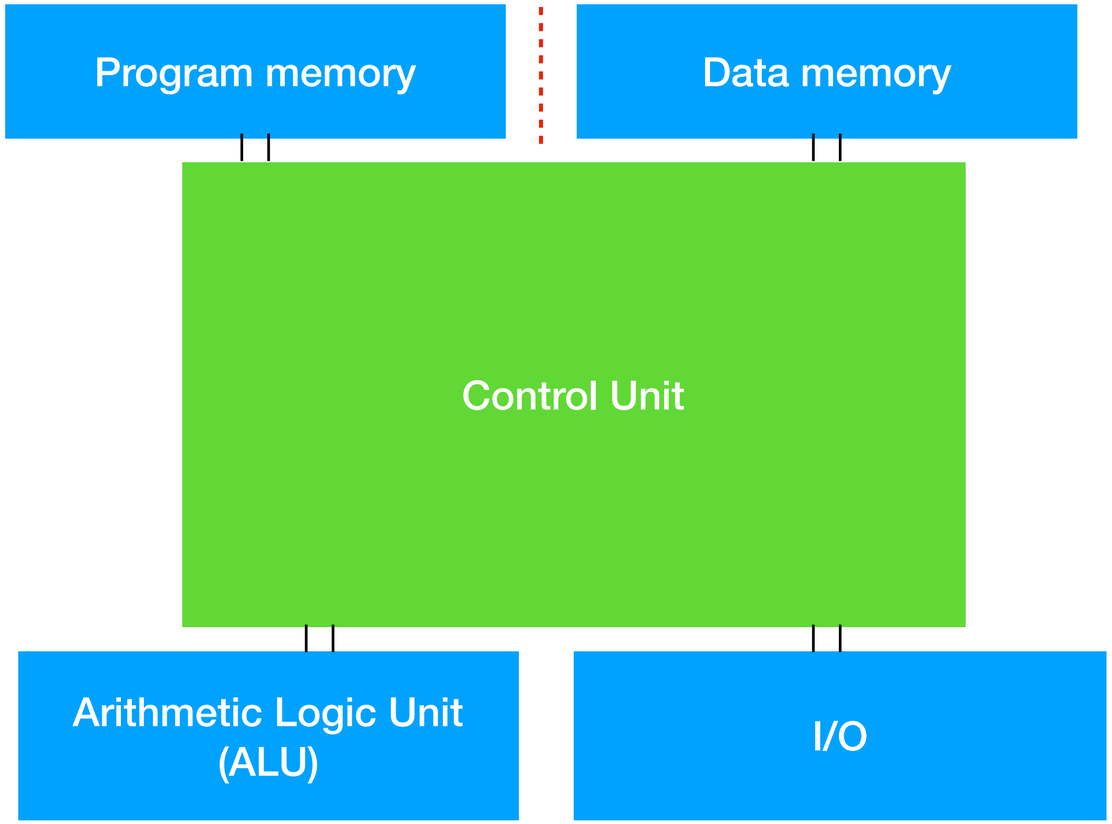
Harvard


Program memory
Data memory
Central Processing Unit (CPU)
Control Unit
Arithmetic Logic Unit (ALU)
Input
Output
Memory
Registers
Cache

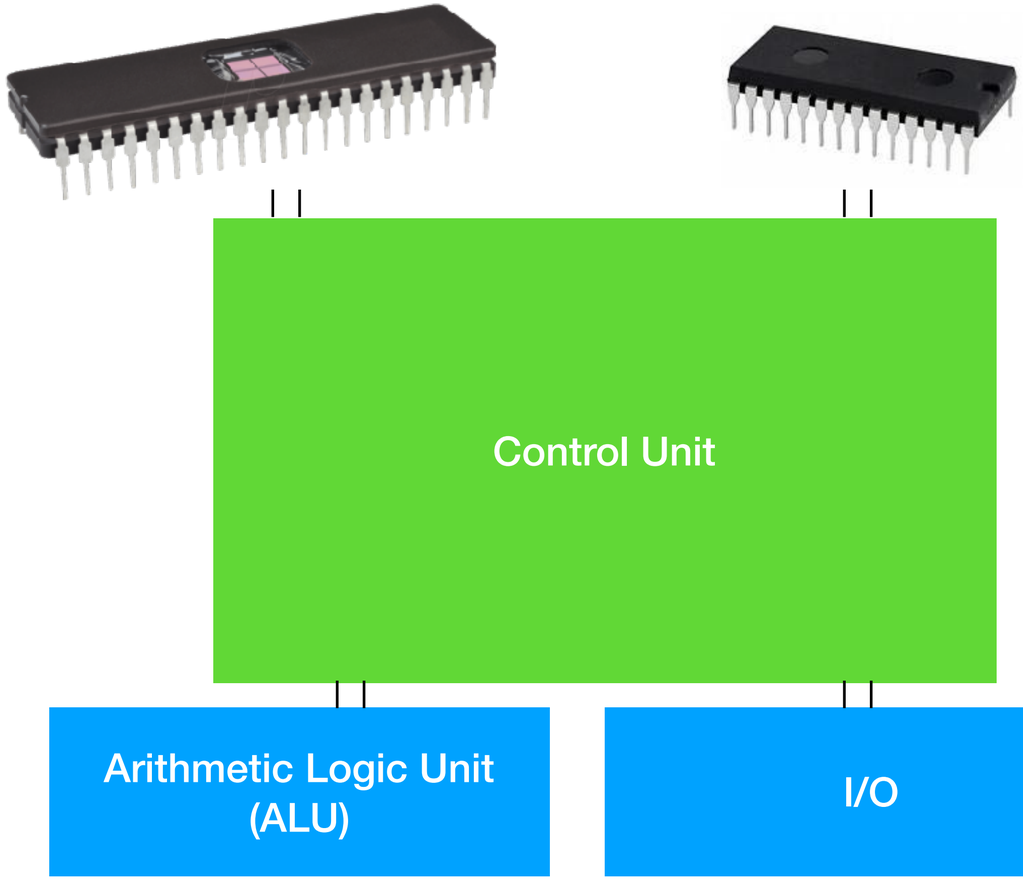
Von Neumann vs Harvard

- 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**

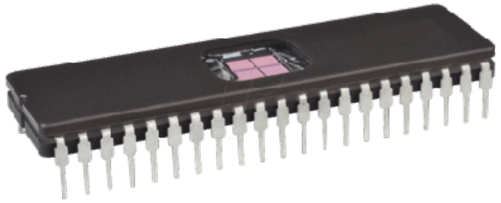
Harvard


Program memory
Data memory
Control Unit
Arithmetic Logic Unit
(ALU)
I/O

Harvard



Harvard



instructions

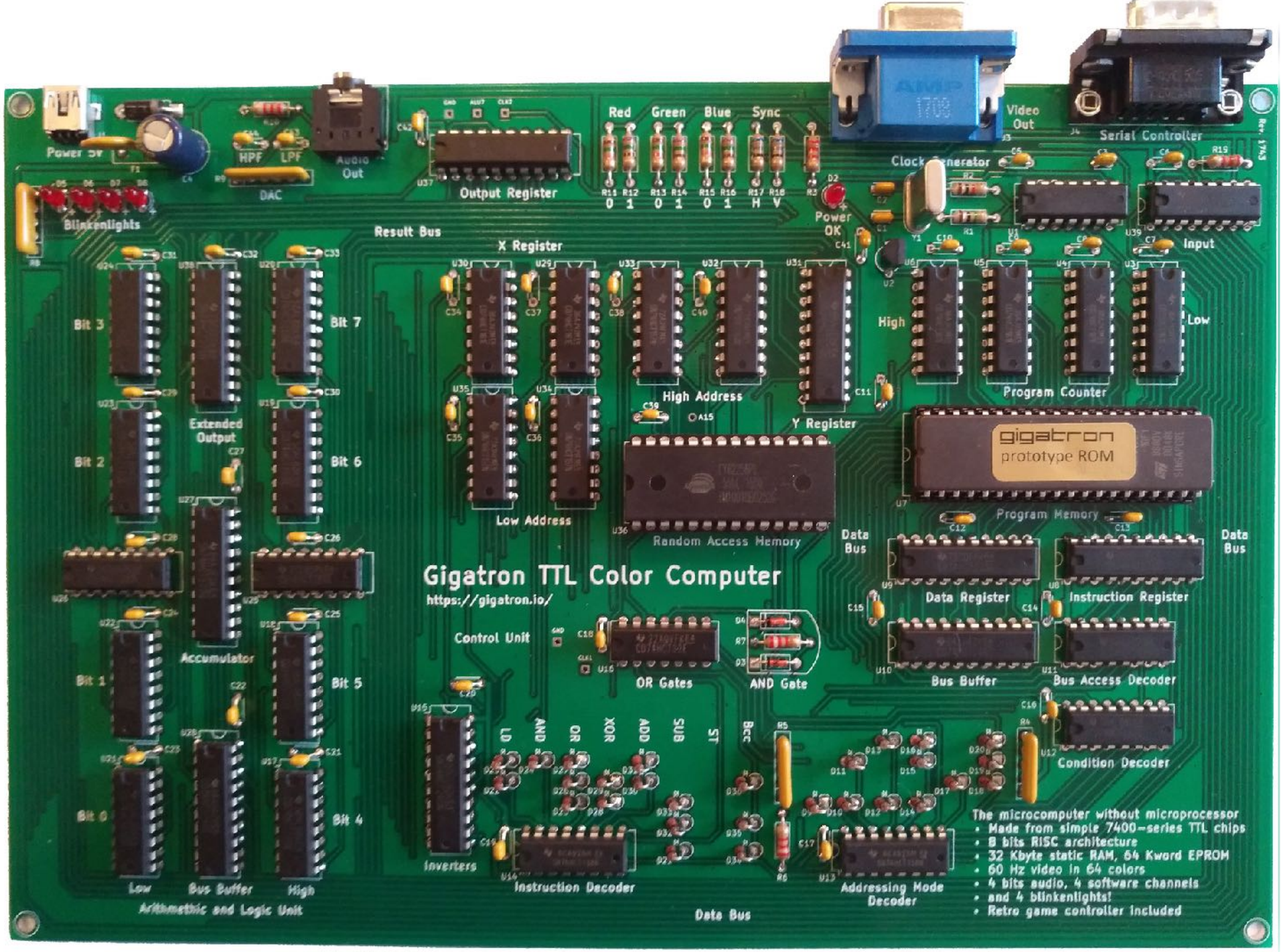
program counter

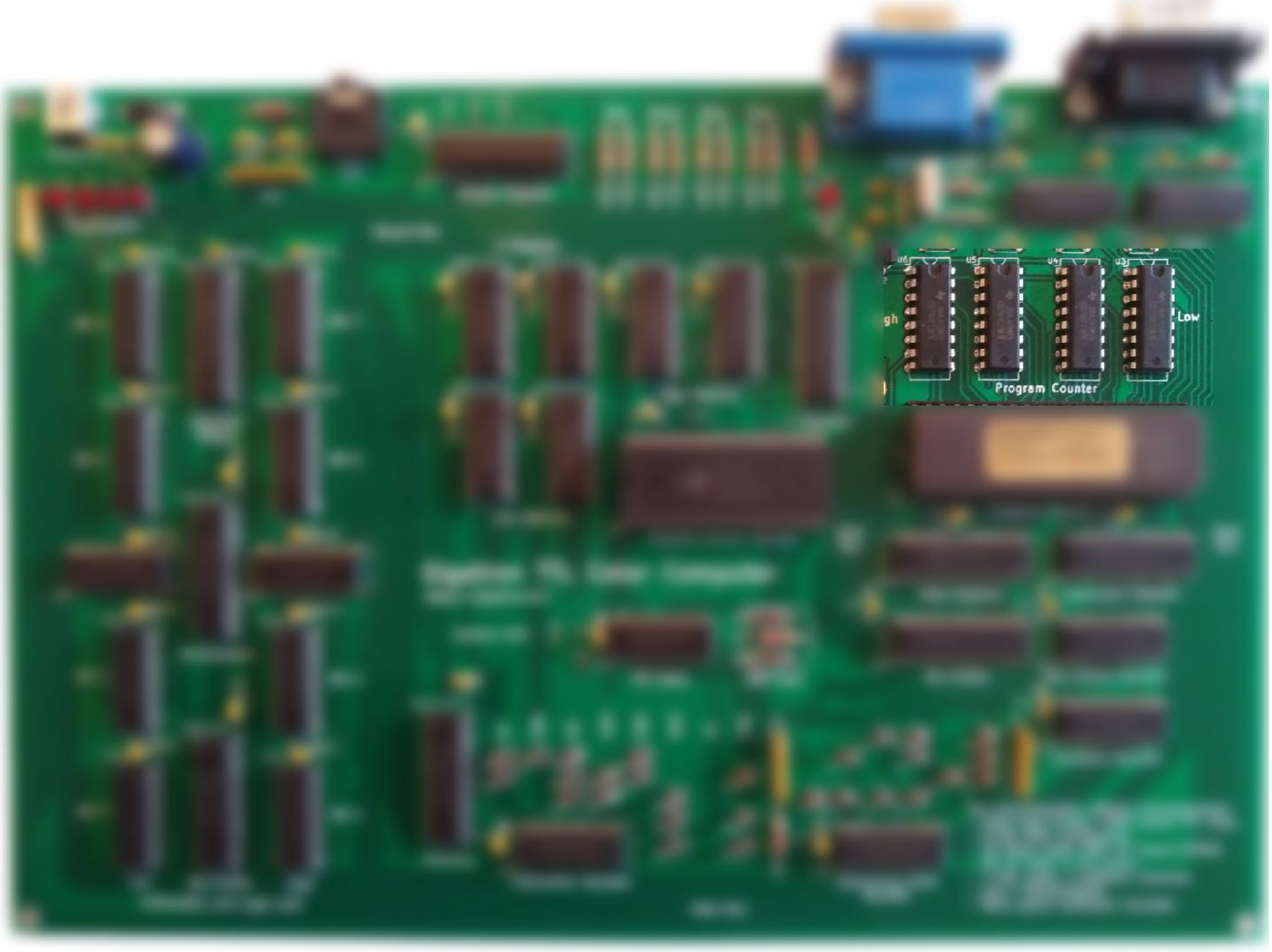
Data memory

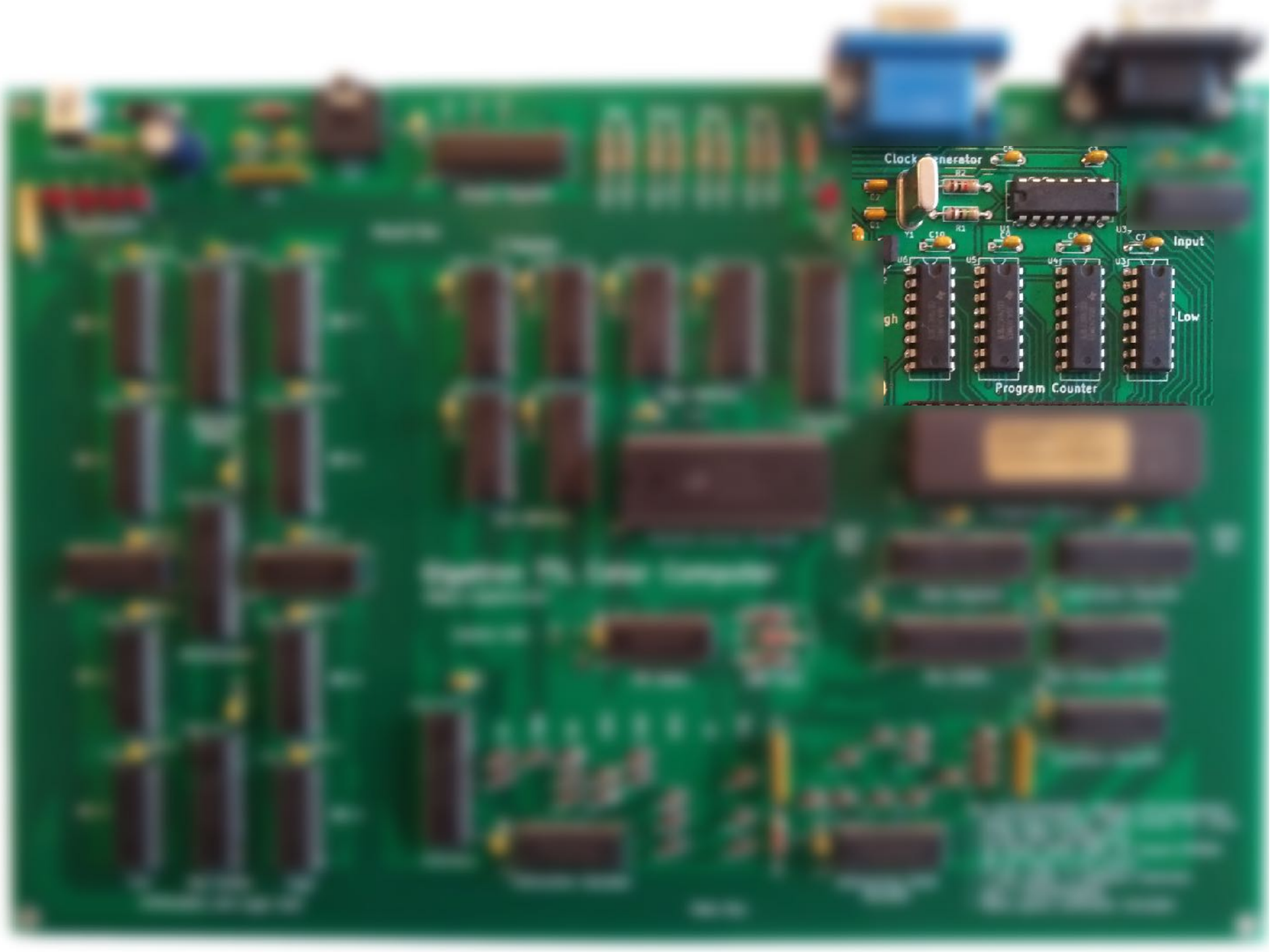
Control Unit

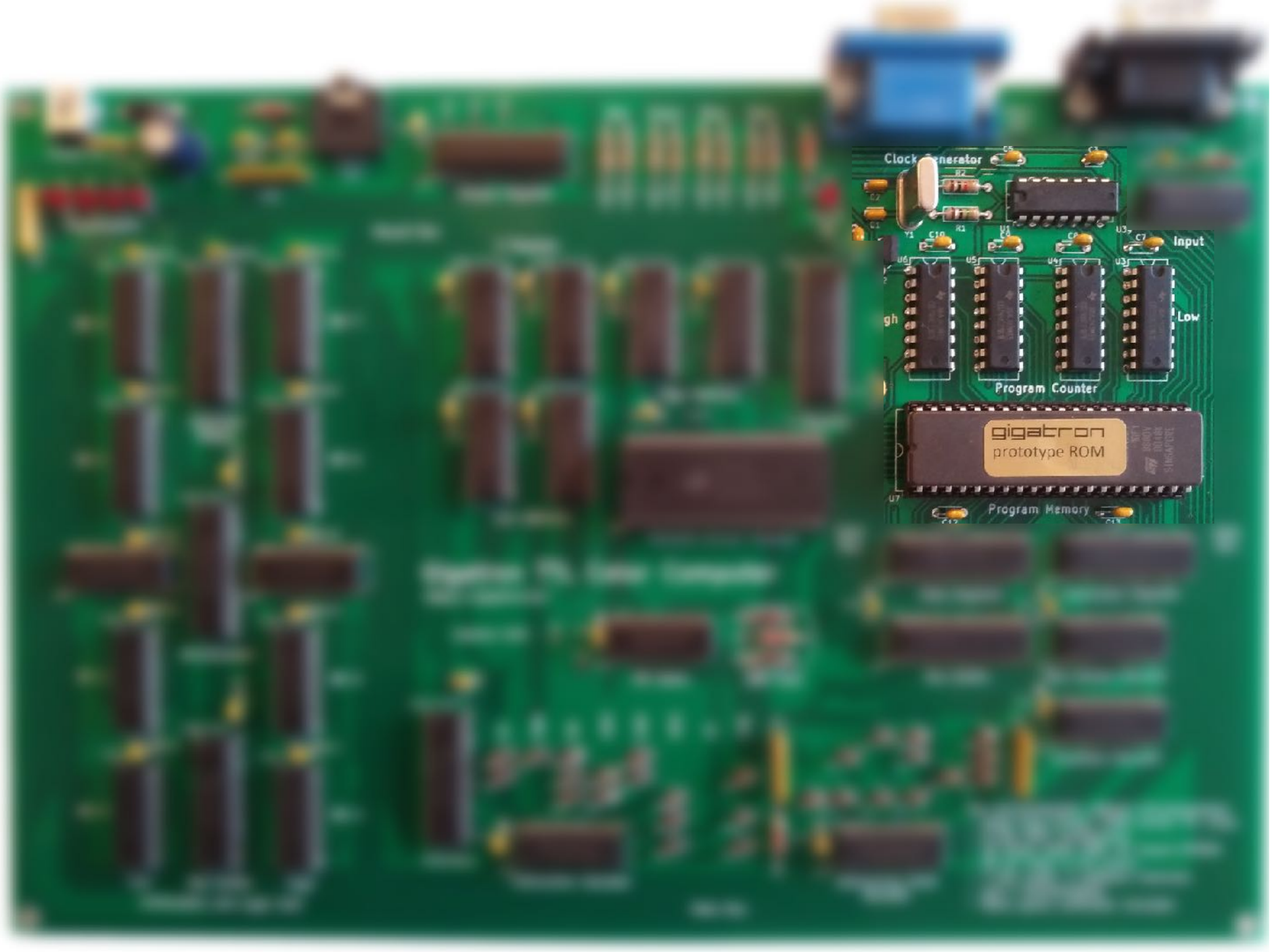
Arithmetic Logic Unit
(ALU)

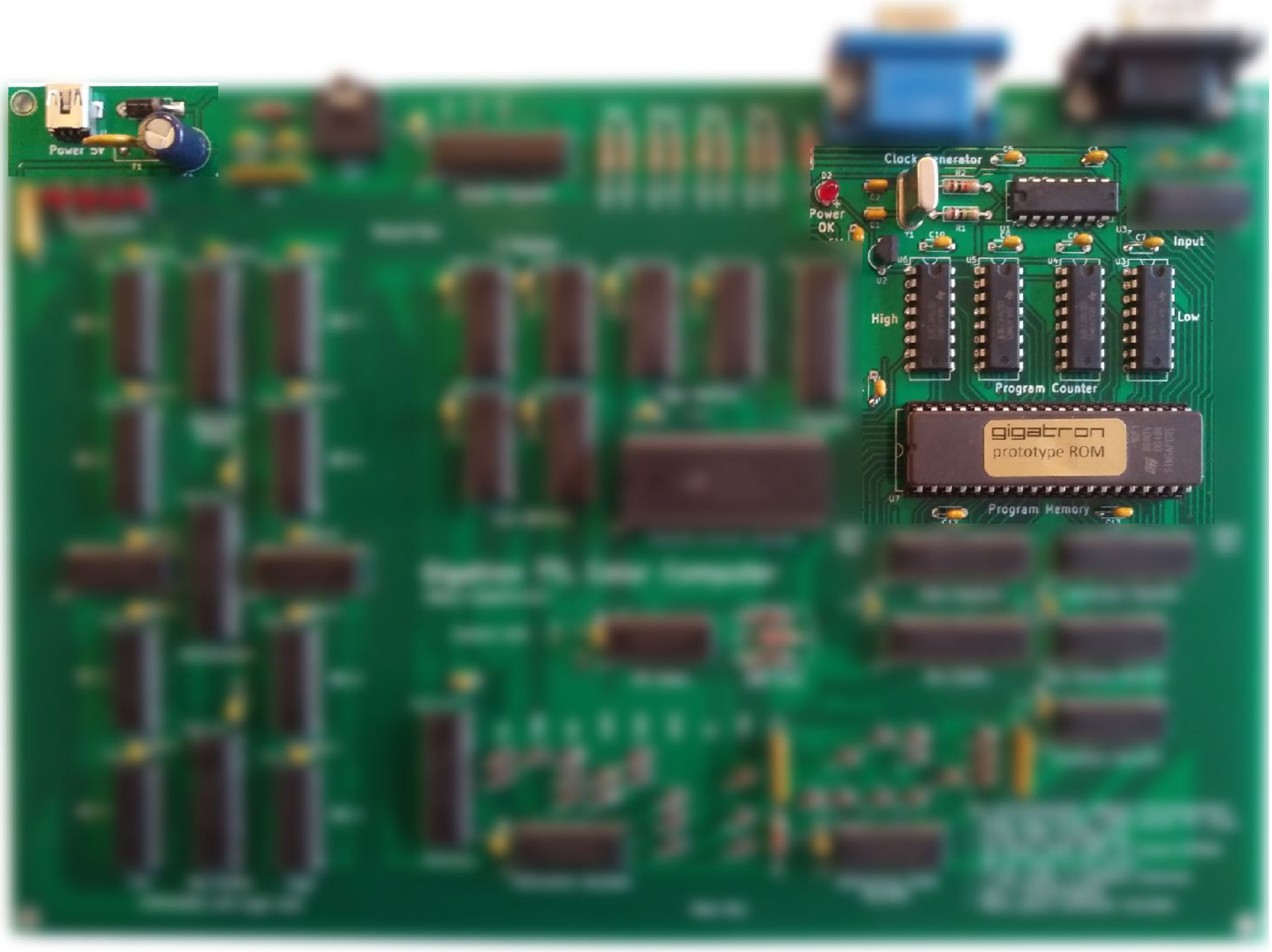
I/O


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

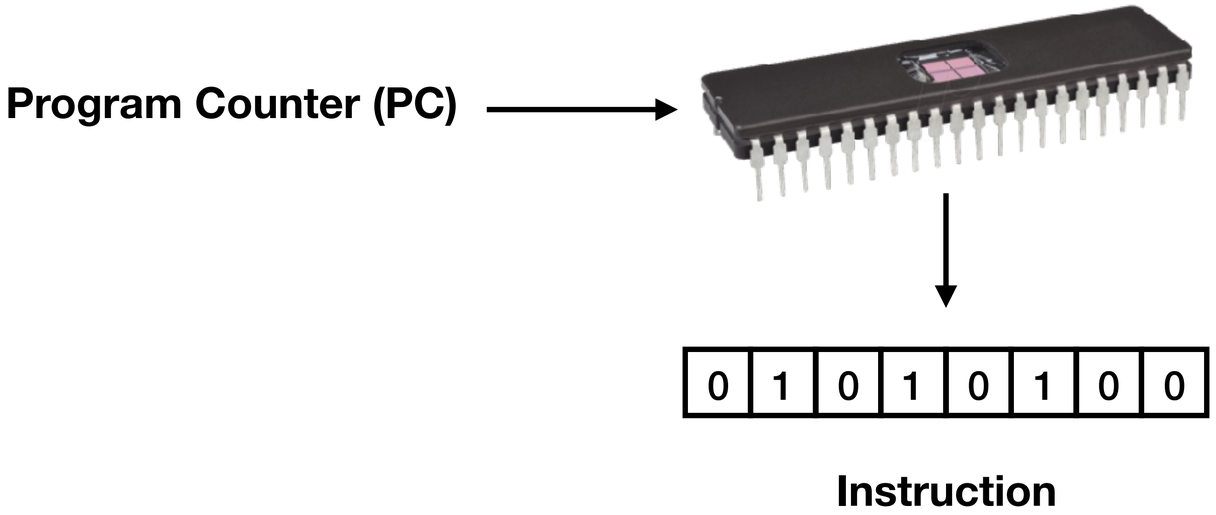

Low
Program Counter


Clock Generator
Input
Low
Program Counter

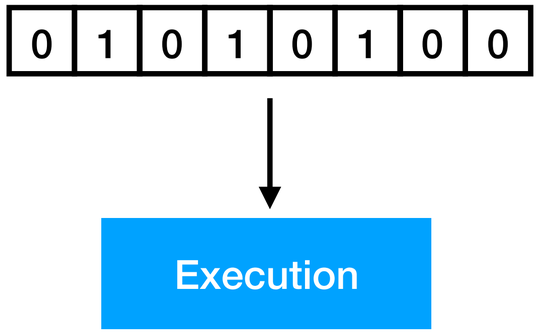

Input
Low
Program Counter
gigatron
prototype ROM
Program Memory


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

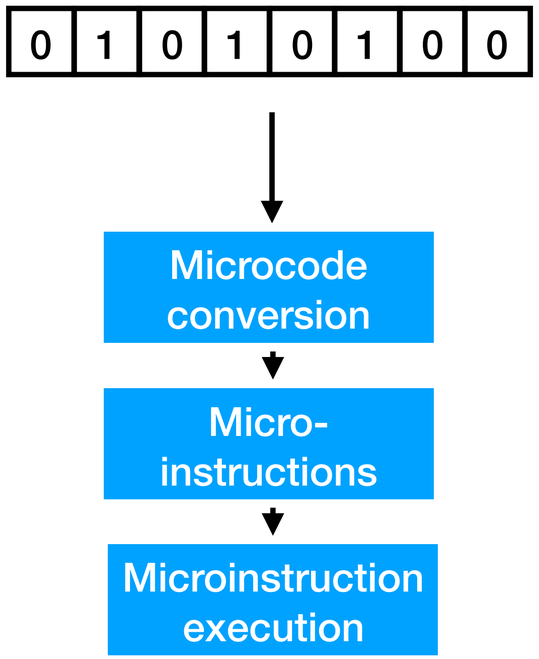
Instructions and operands



RISC


0 1 0 1 0 1 0 0
Execution

CISC


0 1 0 1 0 1 0 0
Microcode conversion
Micro-instructions
Microinstruction execution

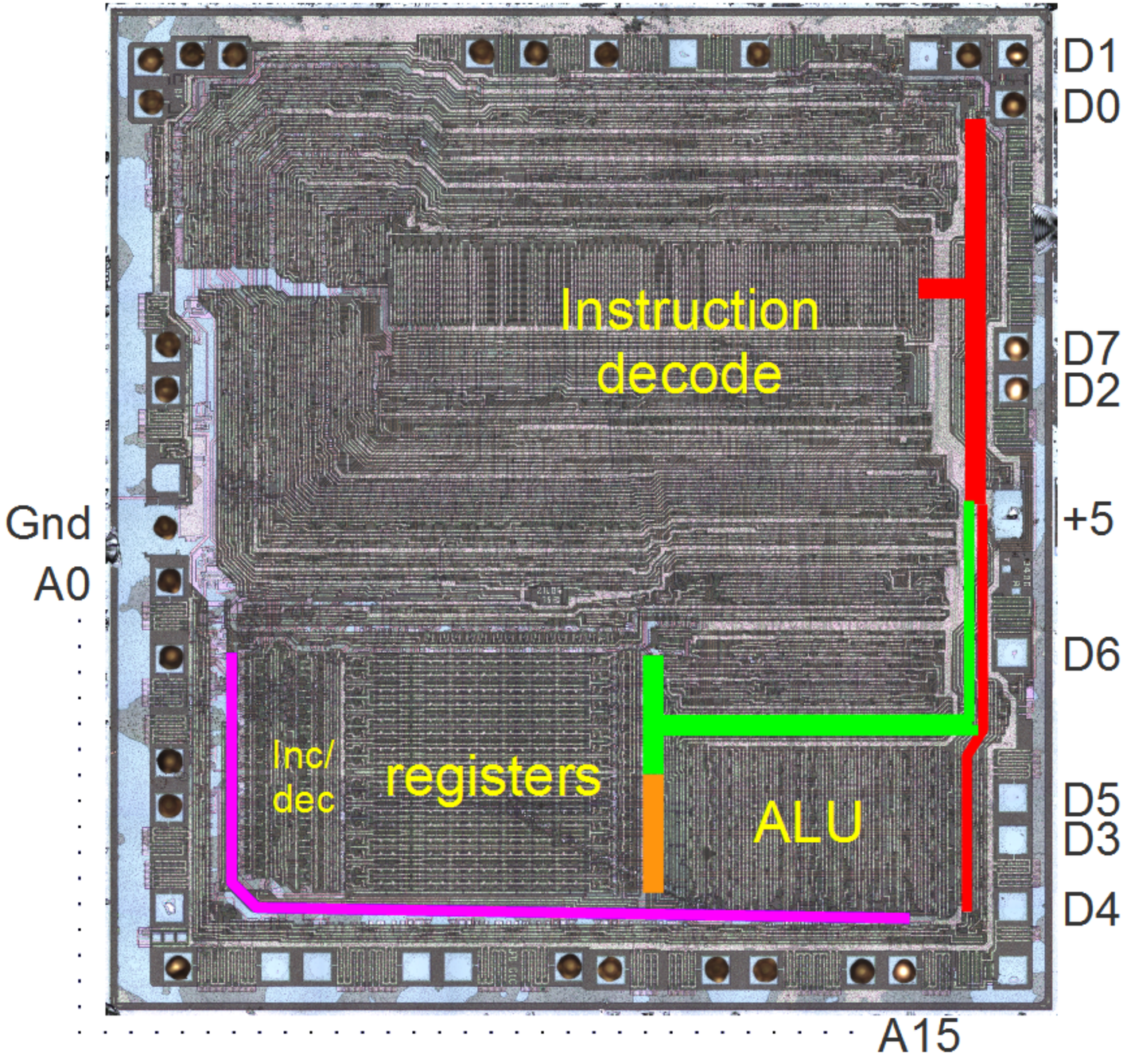
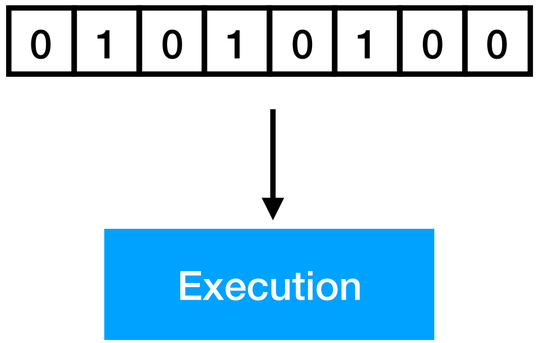

D1
D0
Instruction
decode
D7
D2
Gnd
+5
A0
D6
Inc/
dec
registers
ALU
D5
D3
D4
A15

TABLE 10-2: PIC12F629/675 INSTRUCTION SET

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected
BYTE-ORIENTED FILE REGISTER OPERATIONS								
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff	
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff	
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z
MOVWF	f	Move W to f	1	00	0000	lfff	ffff	
NOP	-	No Operation	1	00	0000	0xx0	0000	
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff	
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z
BIT-ORIENTED FILE REGISTER OPERATIONS								
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff	
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff	
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff	
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff	
LITERAL AND CONTROL OPERATIONS								
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk	
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk	
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk	
RETFIE	-	Return from interrupt	2	00	0000	0000	1001	
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk	
RETURN	-	Return from Subroutine	2	00	0000	0000	1000	
SLEEP	-	Go into Standby mode	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z

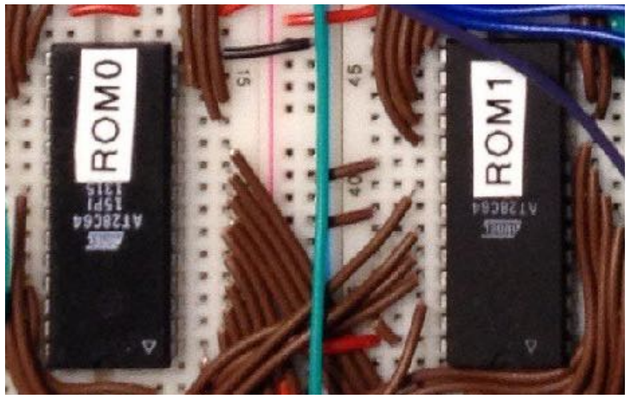
```
LOC     OBJ CODE    STMT  SOURCE STATMENT
0000    222600      23    SORT:   LD      (DATA), HL    ;SAVE DATA ADDRESS
0003    CB84        24    LOOP:   RES     FLAG, H       ;INITIALIZE EXCHANGE FLAG
0005    41          25            LD      B,C           ;INITIALIZE LENGTH COUNTER
0006    05          26            DEC     B             ;ADJUST FOR TESTING
0007    DD2A2600    27            LD      IX, (DATA)    ;INITIALIZE ARRAY POINTER
000B    DD7E00      28    NEXT:   LD      A,(IX+0)      ;FIRST ELEMENT IN COMPARISON
000E    57          29            LD      D, A          ;TEMPORARY STORAGE FOR ELEMEN
000F    DD5E01      30            LD      E, (IX+1)     ;SECOND ELEMENT IN COMPARISON
0012    93          31            SUB     E             ;COMPARISON FIRST TO SECOND
0013    3008        32            JR      NC, NOEX-$    ;IF FIRST> SECOND, NO JUMP
0015    DD7300      33            LD      (IX), E       ;EXCHANGE ARRAY ELEMENTS
0018    DD7201      34            LD      (IX+1), D
001B    CBC4        35            SET     FLAG H        ;RECORD EXCHANGE OCCURRED
001D    DD23        36    NOEX:   INC     IX            ;POINT TO NEXT DATA ELEMENT
001F    10EA        37            DJNZ    NEXT-$        ;COUNT NUMBER OF COMPARISONS
                                                        ;REPEAT IF MORE DATA PAIRS
0021    CB44        39            BIT     FLAG, H       ;DETERMINE IF EXCHANGE OCCURRE
0023    20DE        40            JR      NZ, LOOP-$    ;CONTINUE IF DATA UNSORTED
0025    C9          41            RET                   ;OTHERWISE, EXIT
                    42    ;
0026                43    FLAG:   EQU     0             ;DESIGNATION OF FLAG BIT
0026                44    DATA:   DEFS    2             ;STORAGE FOR DATA ADDRESS
                    45            END
```

RISC



Instructions and operands

Program Counter (PC)



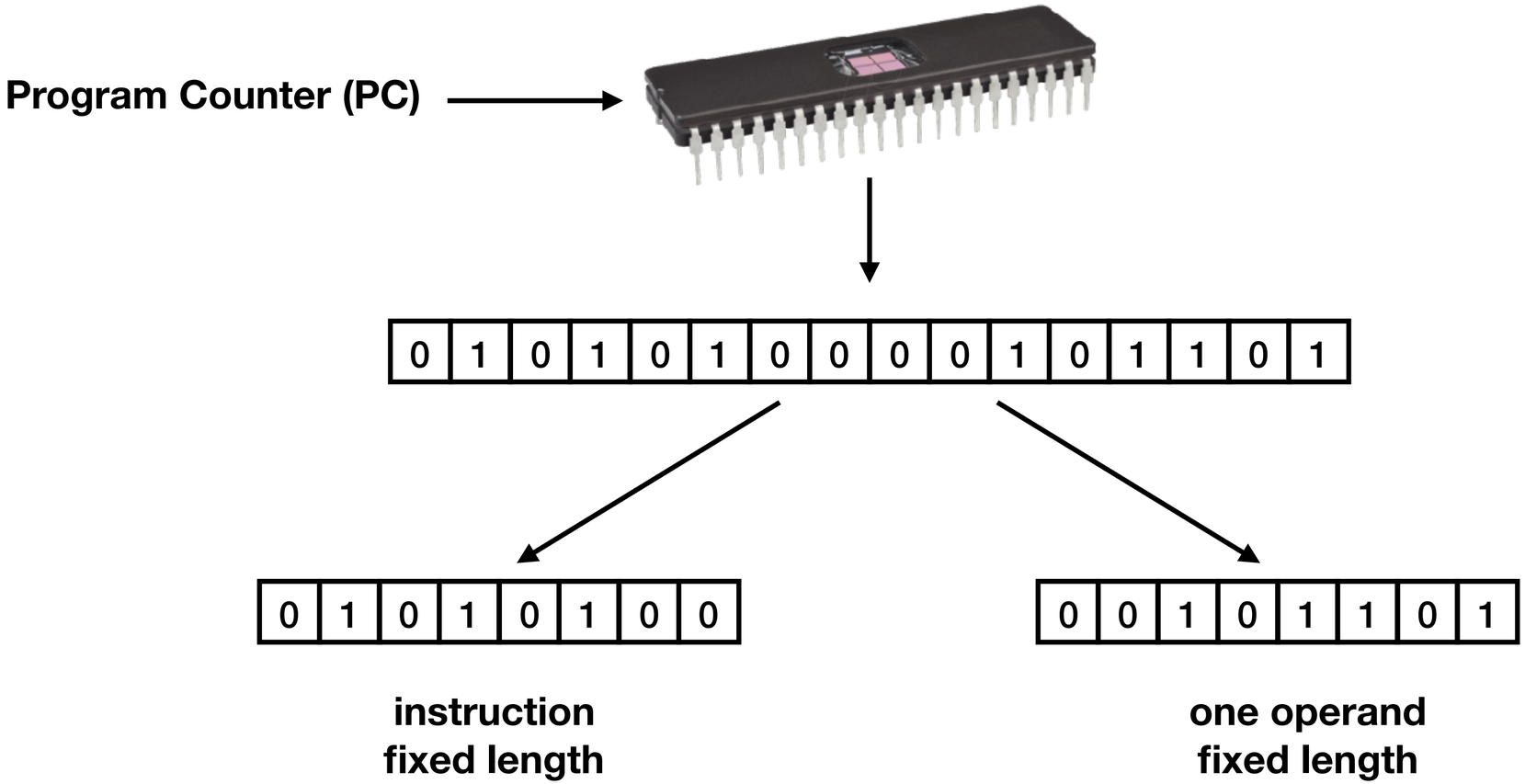
0	1	0	1	0	1	0	0

instruction
fixed length

0	0	1	0	1	1	0	1

one operand
fixed length

Instructions and operands

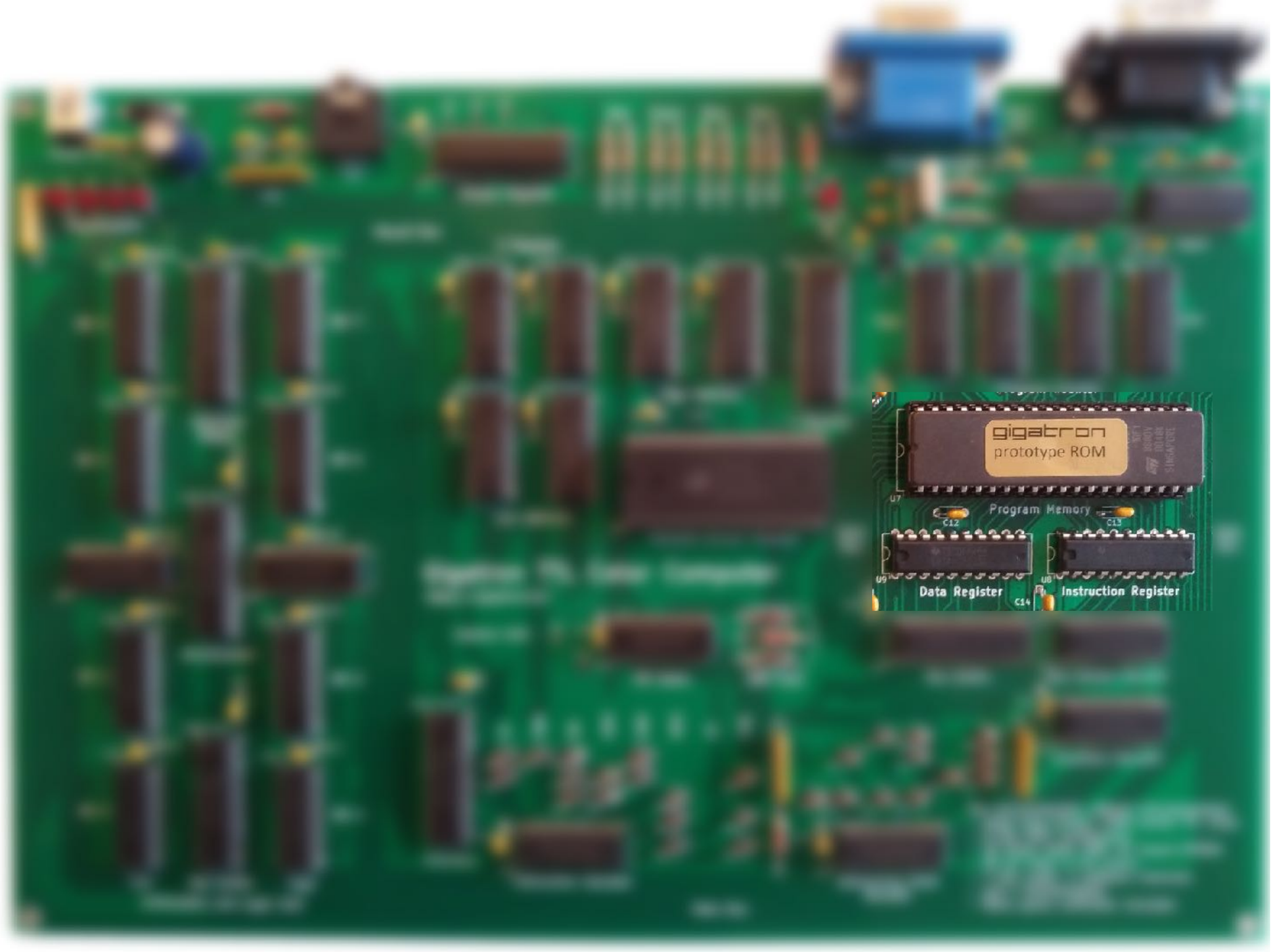


Pipelining

- 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
U7
Program Memory
C12
C13
U9
Data Register
U8
Instruction Register
C14

Branch delay slot

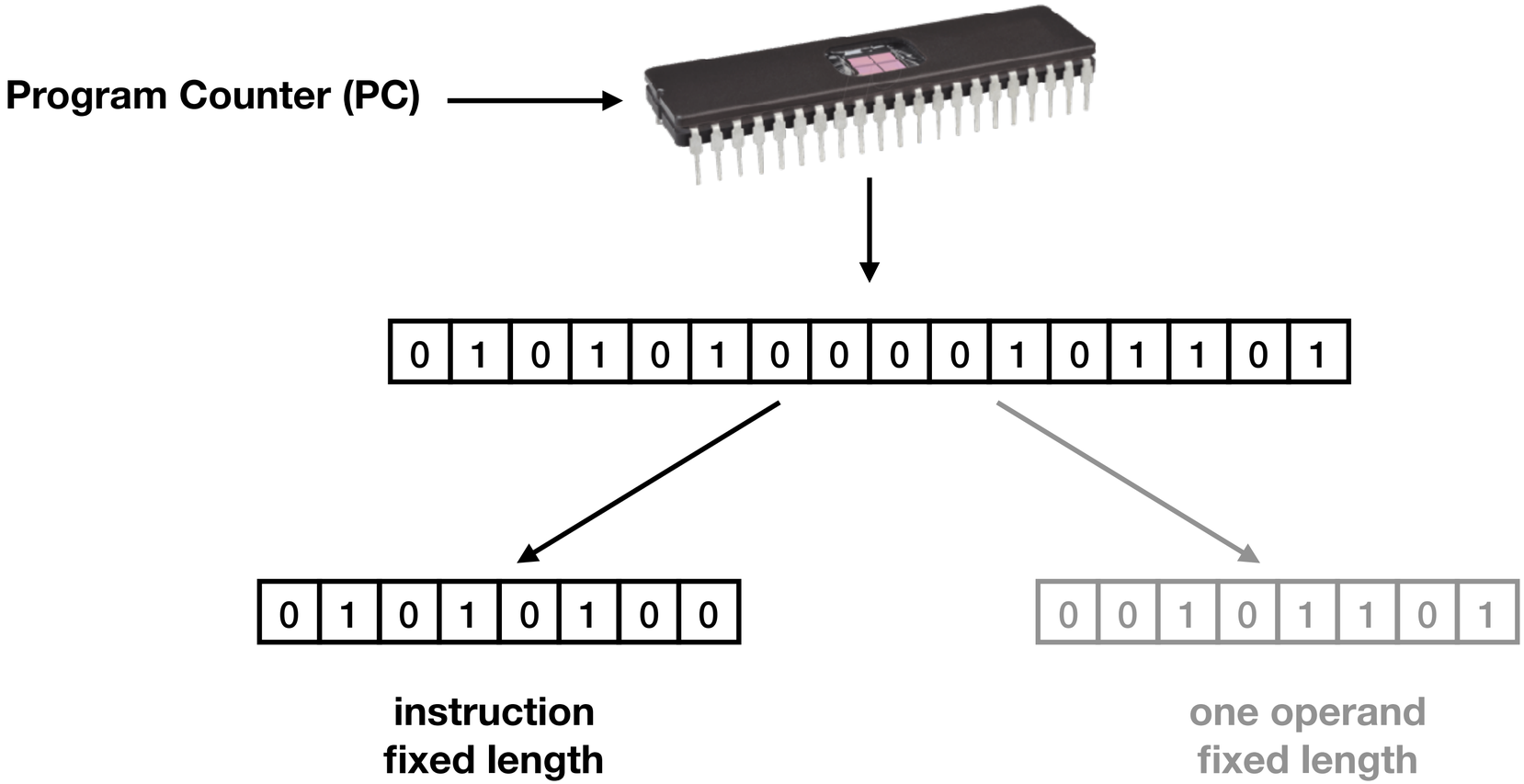
- When a branch instruction is done, the next instruction was already fetched from memory, even if the branch is made!

```
0000 0000  ld   $00
[..]
000e fc00  bra  $00
000f 0200  nop
```

- Can be turned into:

```
0000 0000  ld   $00
[..]
000e fc01  bra  $01   # was: bra $00
000f 0000  ld   $00   # instruction on address 0
```

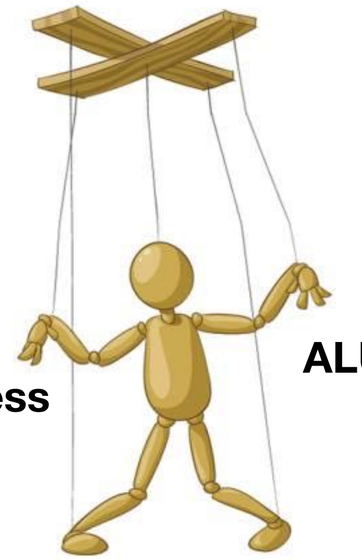
Instructions and operands



Instructions and operands

0	1	0	1	0	1	0	0

instruction
fixed length



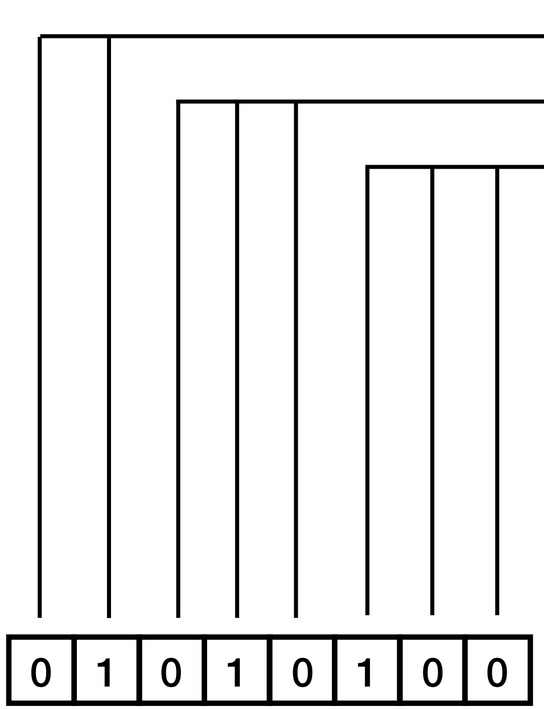
bus access

ALU operation

program counter

registers

Instructions and operands



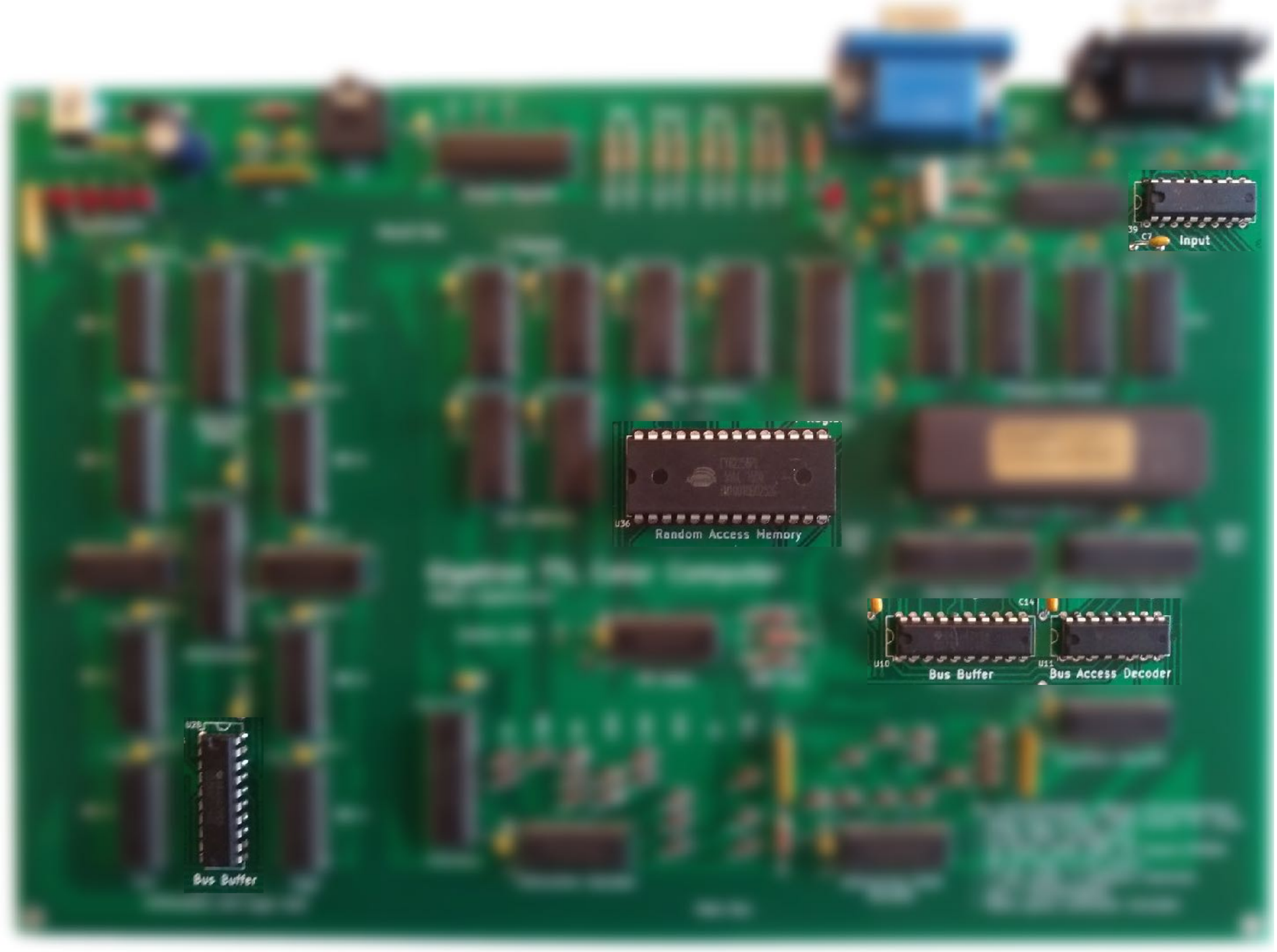
instruction
fixed length

Bus access

Address mode/Branch condition

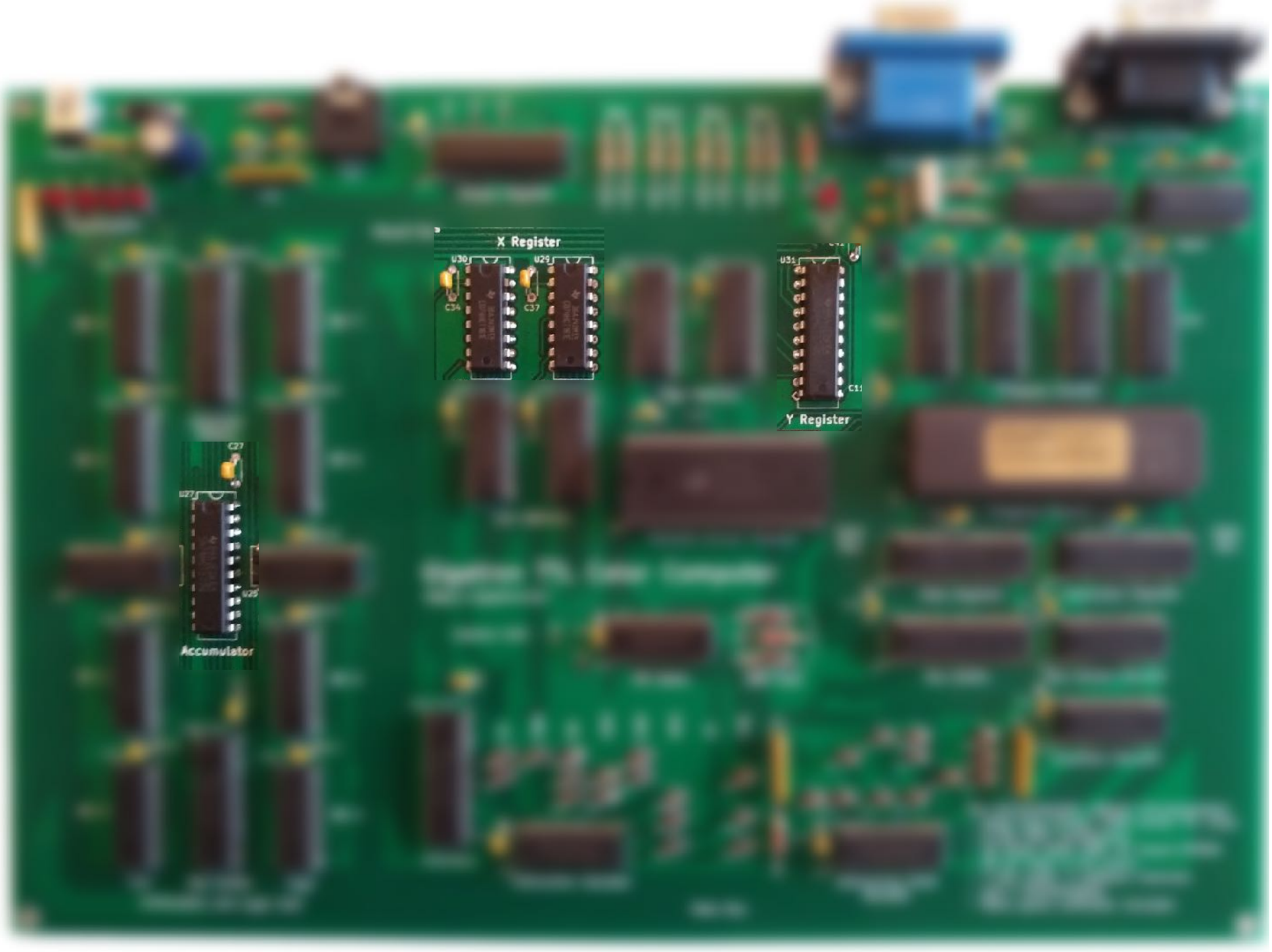
Instruction

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

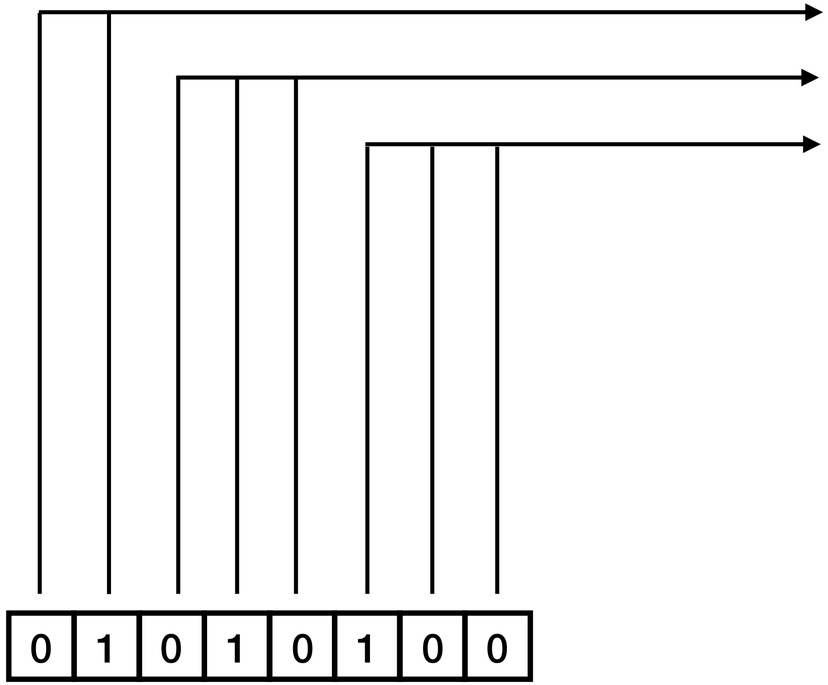

Input
Random Access Memory
Bus Buffer
Bus Access Decoder
Bus Buffer

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
Y Register
U31
C11
Accumulator
C27
U27
U25

Instructions and operands



instruction
fixed length

Bus access

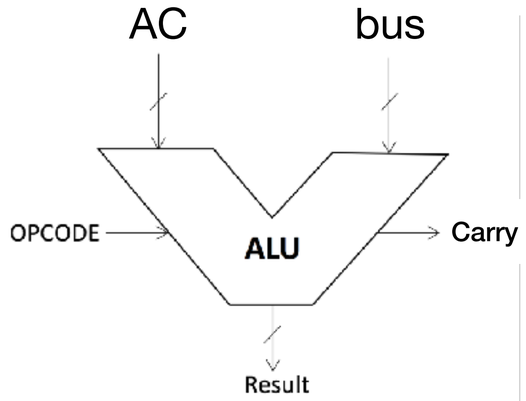
Address mode/Branch condition

Instruction

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

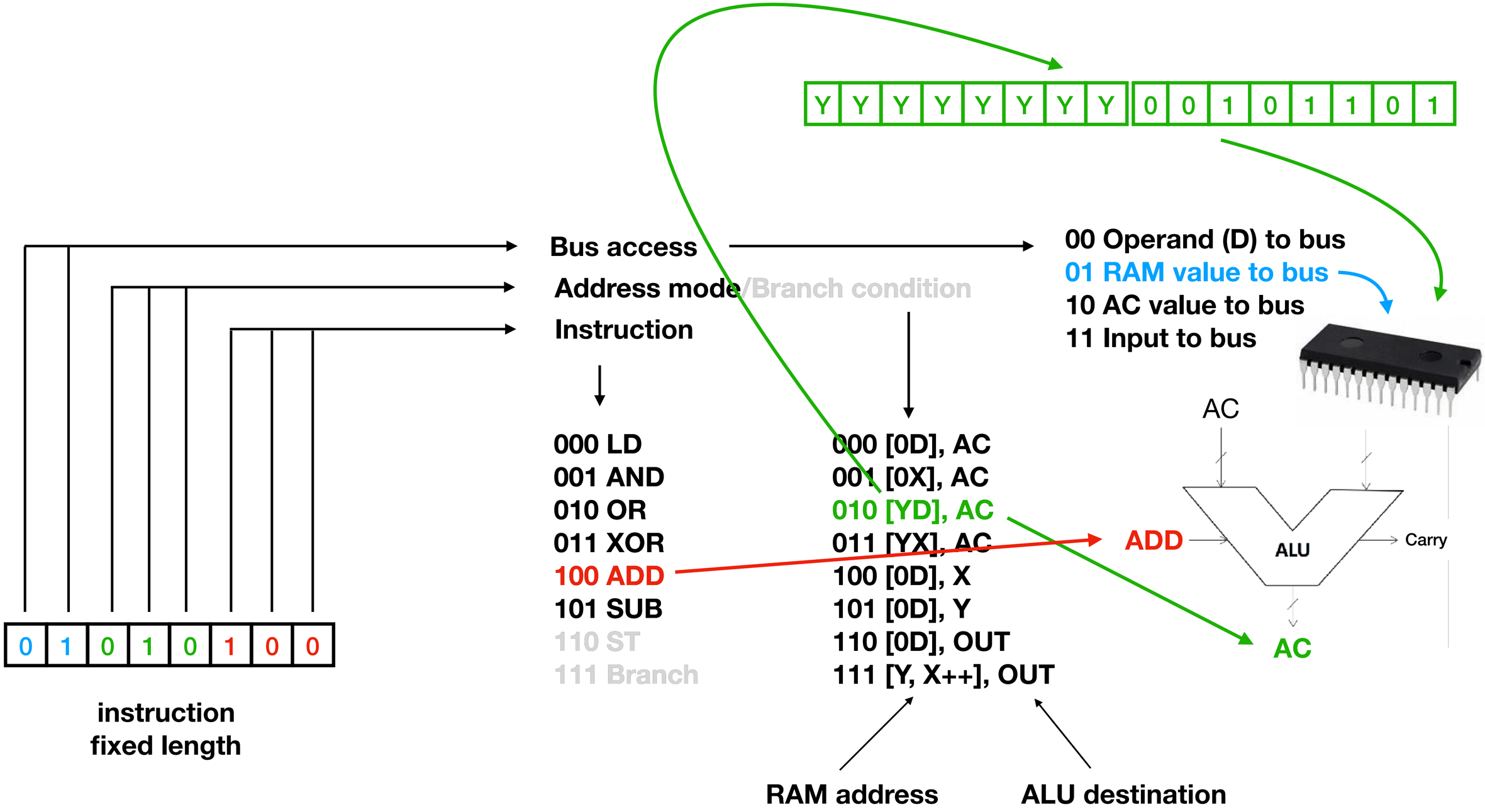
000 LD
001 AND
010 OR
011 XOR
100 ADD
101 SUB
110 ST
111 Branch

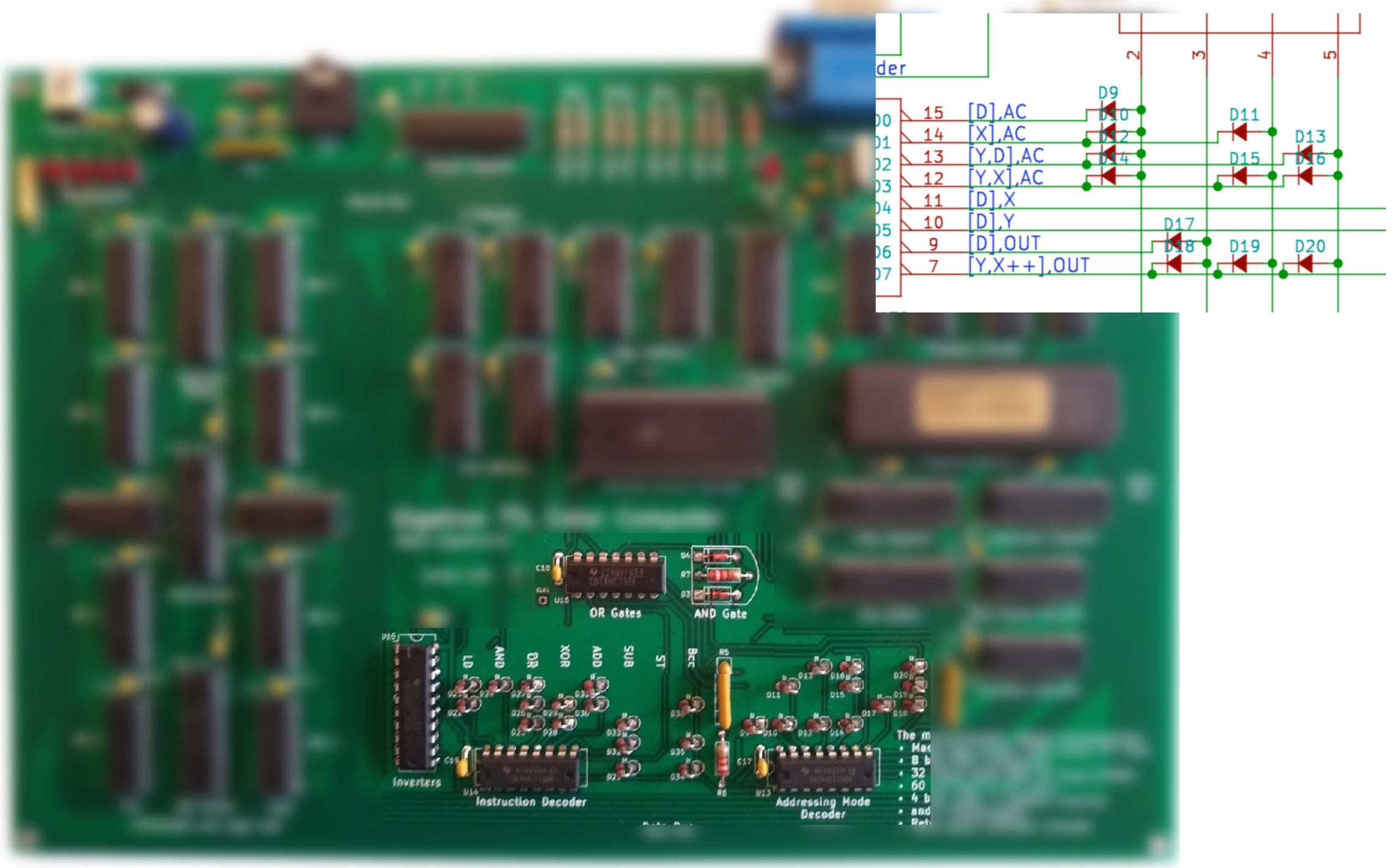
000 [0D], AC
001 [0X], AC
010 [YD], AC
011 [YX], AC
100 [0D], X
101 [0D], Y
110 [0D], OUT
111 [Y, X++], OUT



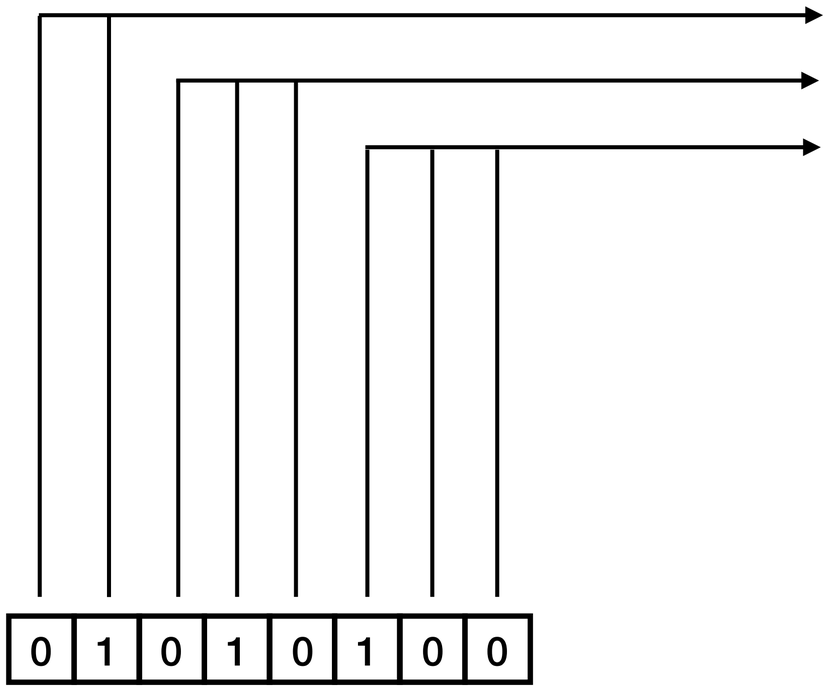
RAM address

ALU destination


Y Y Y Y Y Y Y Y 0 0 1 0 1 1 0 1
Bus access
Address mode/Branch condition
Instruction
00 Operand (D) to bus
01 RAM value to bus
10 AC value to bus
11 Input to bus
000 LD
001 AND
010 OR
011 XOR
100 ADD
101 SUB
110 ST
111 Branch
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
AC
ADD
ALU
Carry
AC
0 1 0 1 0 1 0 0
instruction
fixed length
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
OR Gates
AND Gate
LD
AND
OR
XOR
ADD
SUB
ST
Bcc
Inverters
Instruction Decoder
Addressing Mode Decoder

Instructions and operands



instruction
fixed length

Bus access

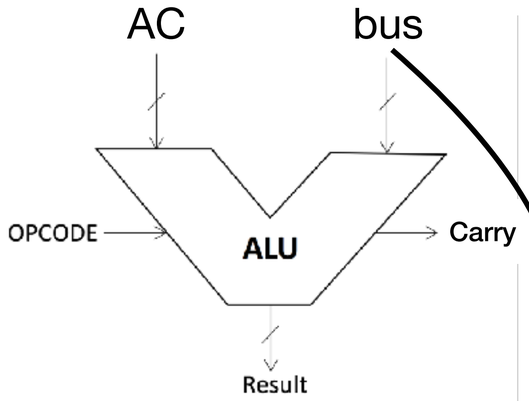
Address mode/Branch condition

Instruction

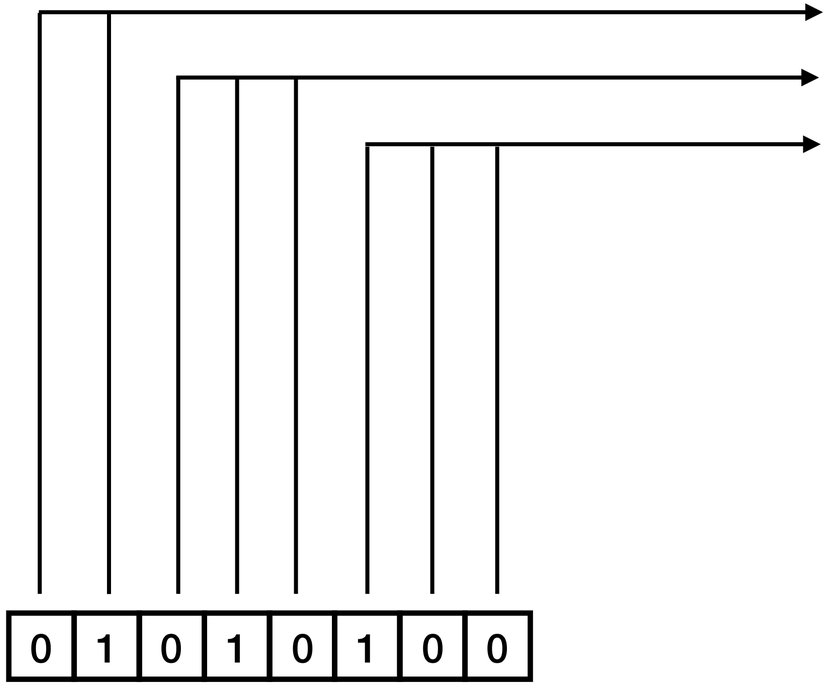
000 LD
001 AND
010 OR
011 XOR
100 ADD
101 SUB
110 ST
111 Branch

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

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



Instructions and operands



instruction
fixed length

Bus access

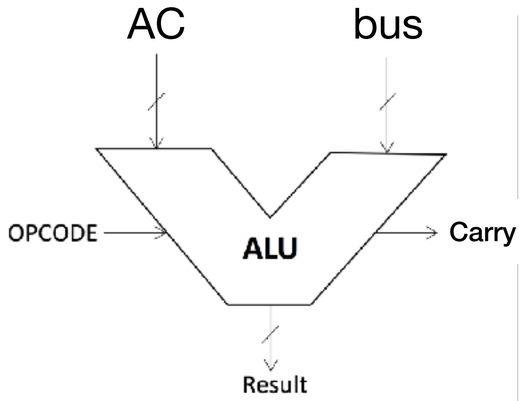
Address mode/Branch condition

Instruction

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

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

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

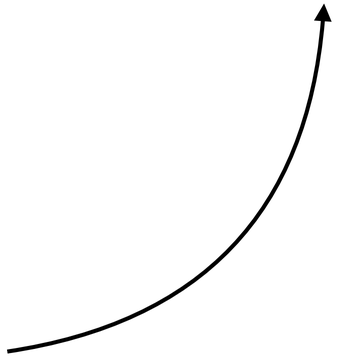


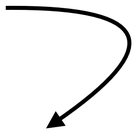
Conditional branches

	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

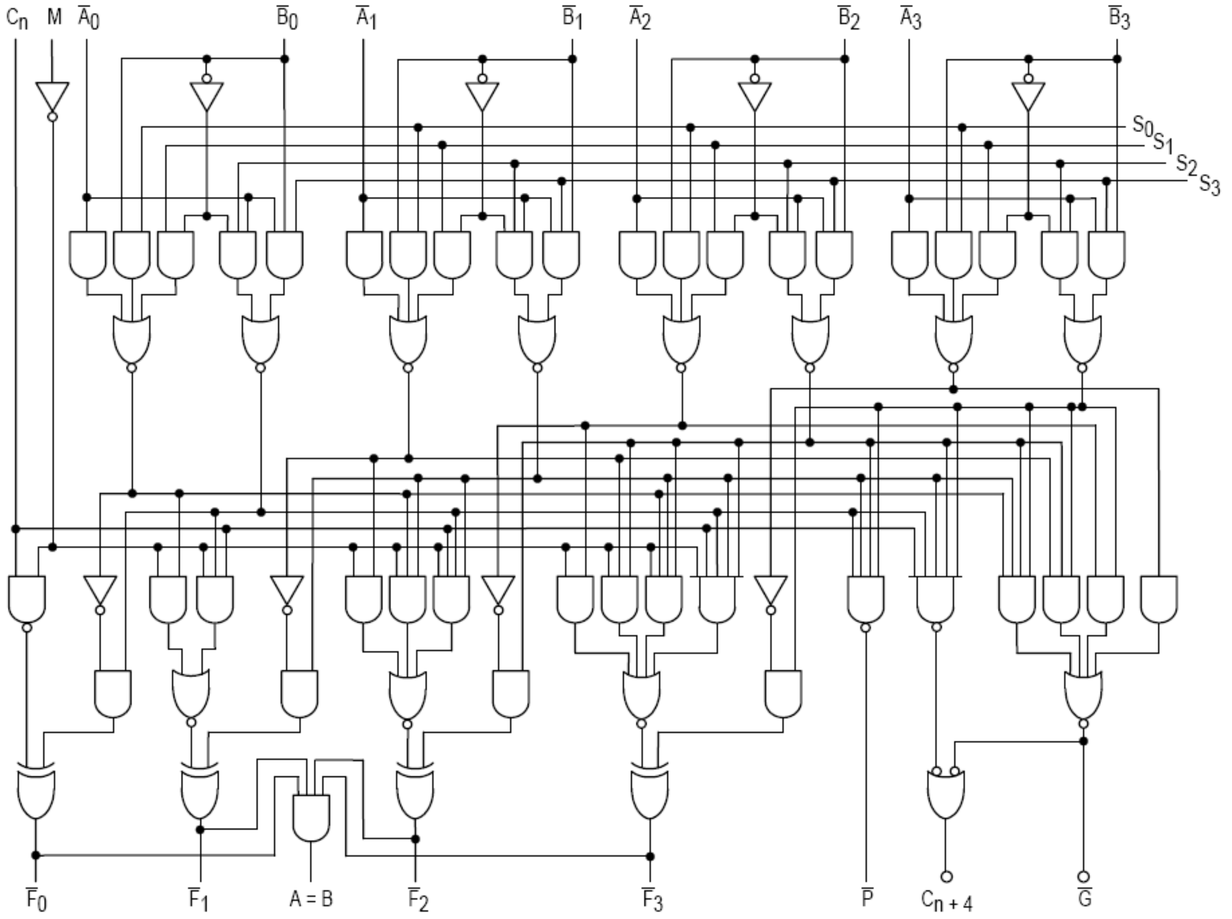


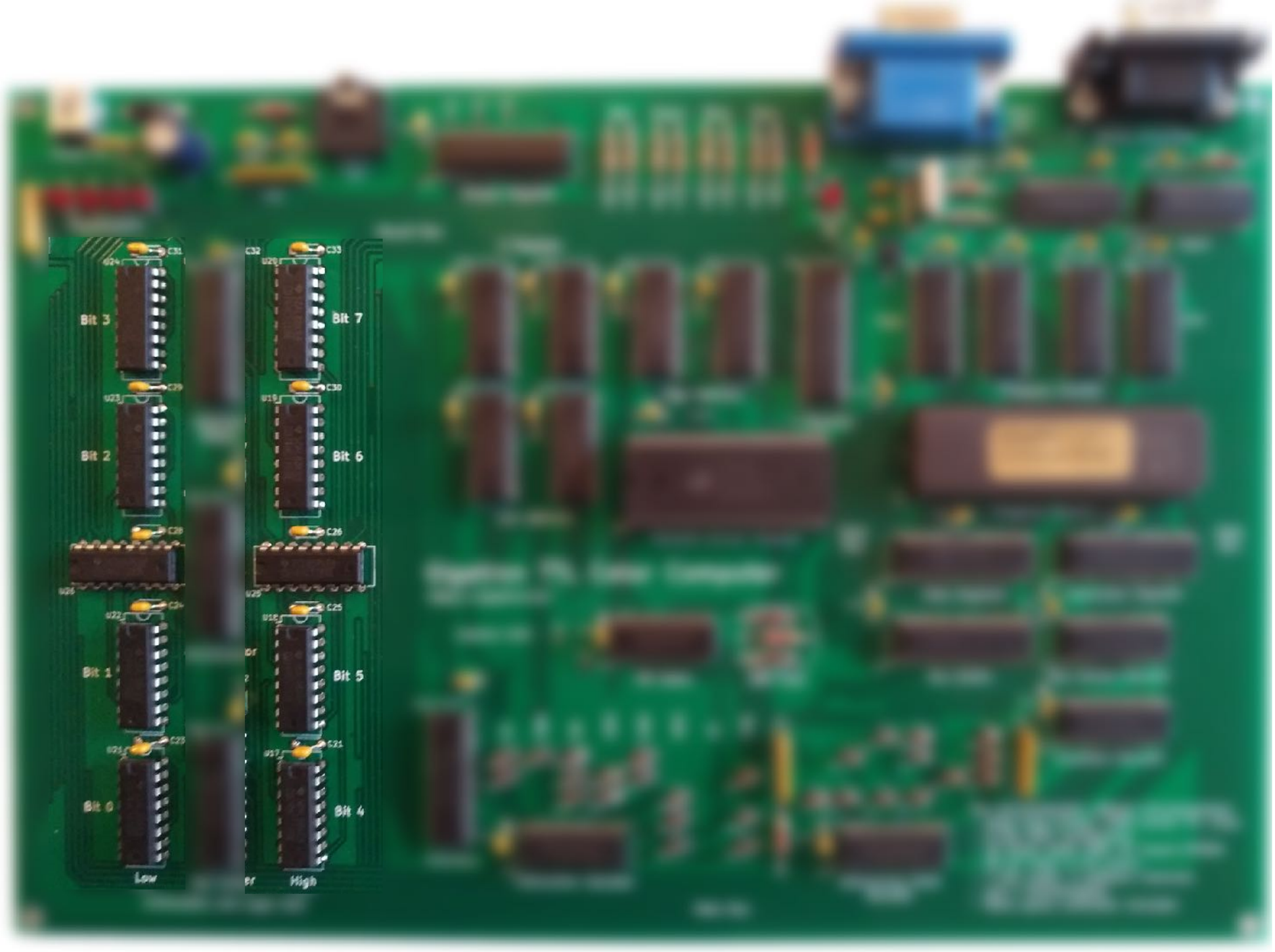


The carry bit of the ALU can now be used to determine the branch condition


7352
SN74S181N


7352
SN74S181N

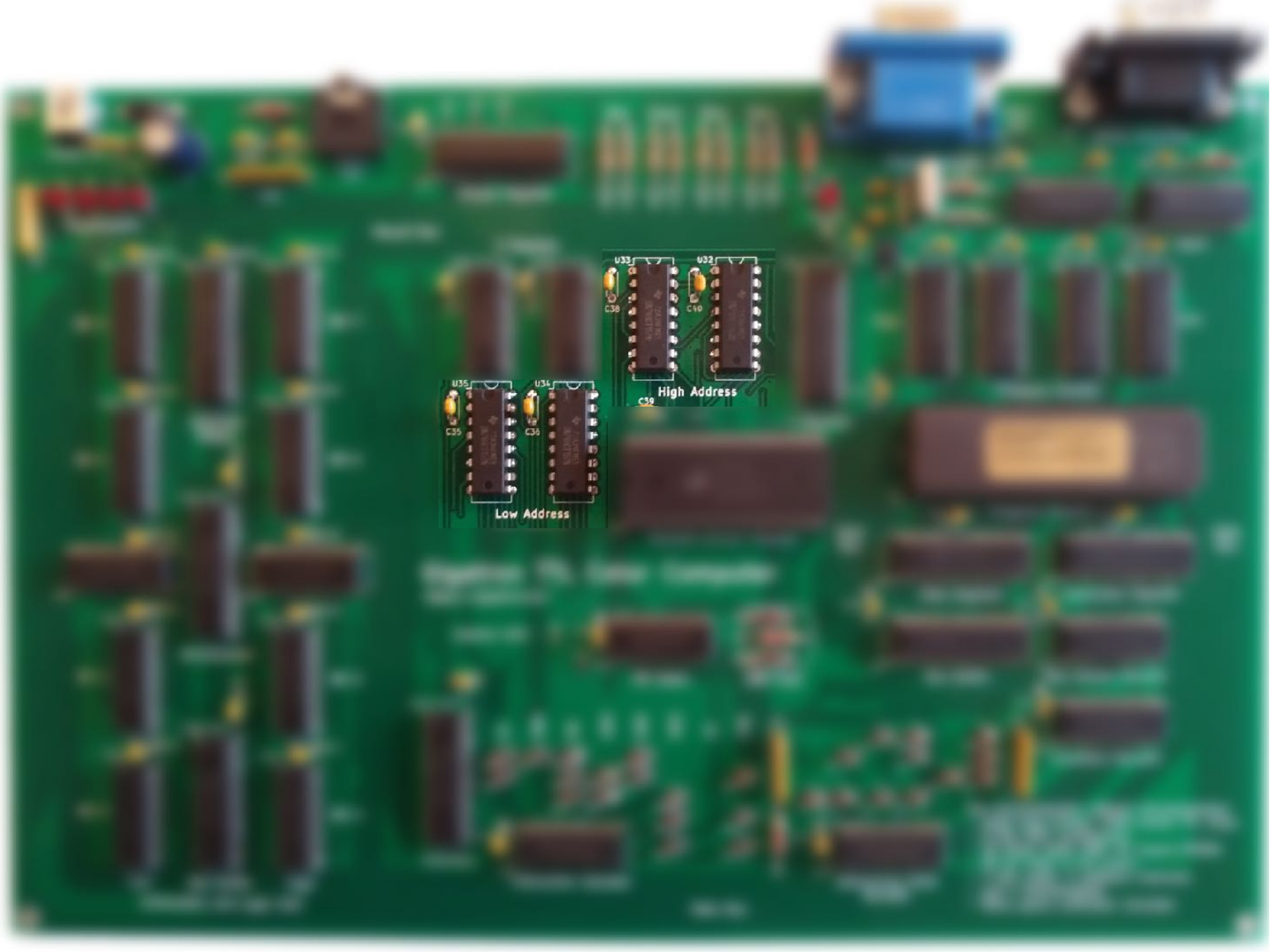

Cn
M
A0
B0
A1
B1
A2
B2
A3
B3
S0
S1
S2
S3
F0
F1
A = B
F2
F3
P
Cn + 4
G


Bit 3
Bit 2
Bit 1
Bit 0
Low
Bit 7
Bit 6
Bit 5
Bit 4
High

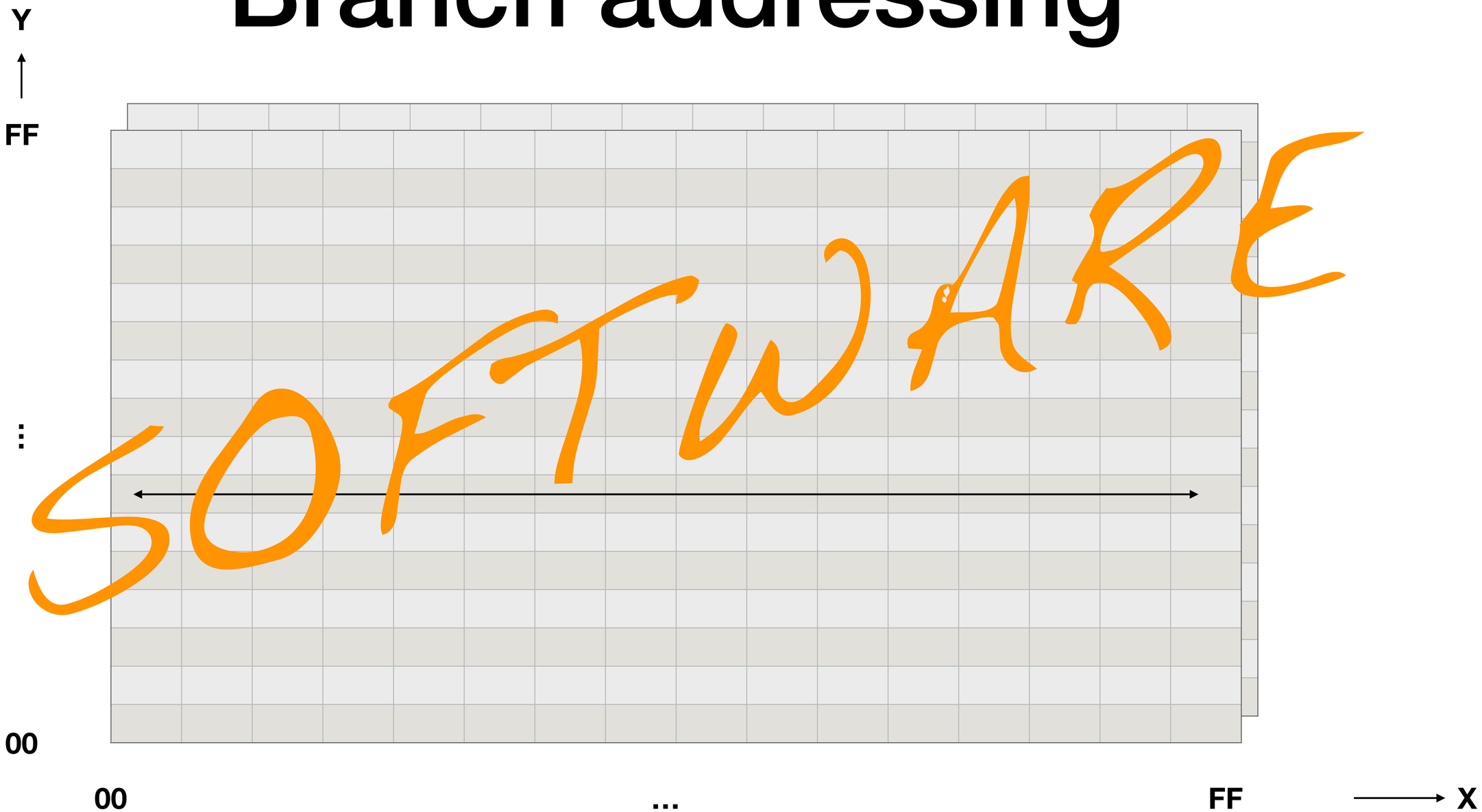
Address mode

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

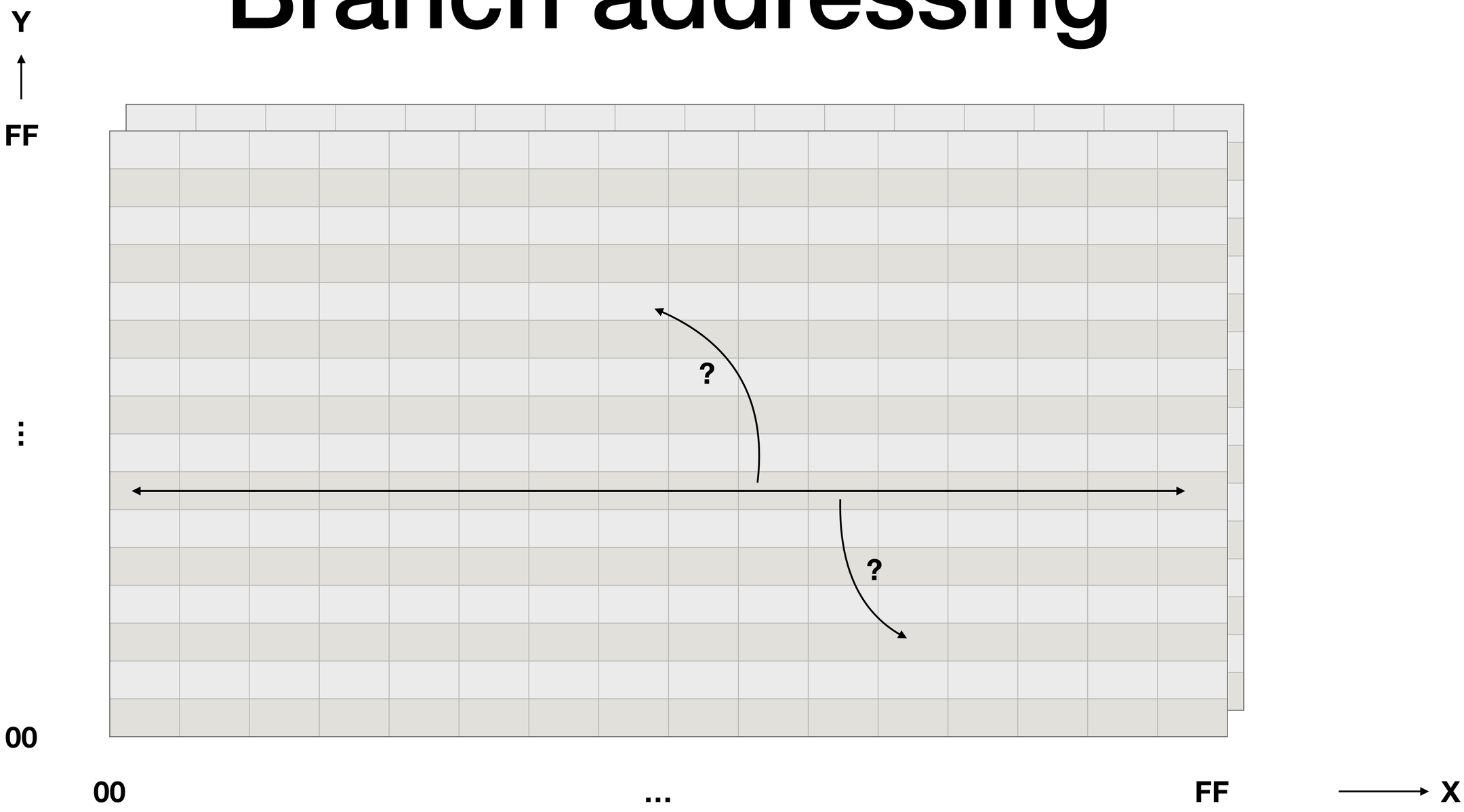
Mode	Address high	Address low	Use
0D	00000000	DDDDDDDD	zero page direct
0X	00000000	XXXXXXXX	zero page indirect via X
YD	YYYYYYYY	DDDDDDDD	page Y direct
YX	YYYYYYYY	XXXXXXXX	page Y indirect via X


U33
U32
C38
C40
High Address
C39
U35
U34
C35
C36
Low Address

Branch addressing



Branch addressing



Far jump

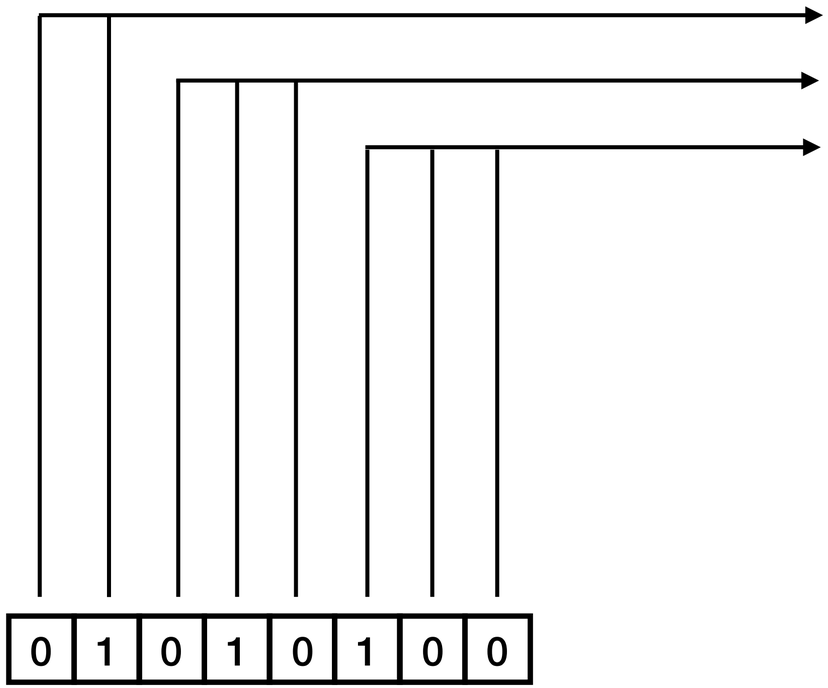
000 Far jump
001 AC > 0
010 AC < 0
011 AC ≠ 0
100 AC = 0
101 AC ≥ 0
110 AC ≤ 0
111 Jump

- 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



instruction
fixed length

Bus access

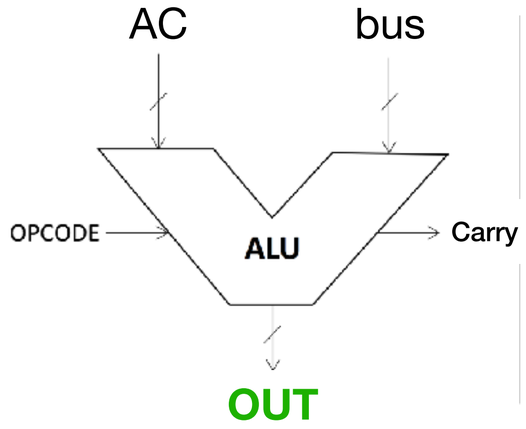
Address mode/Branch condition

Instruction

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

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

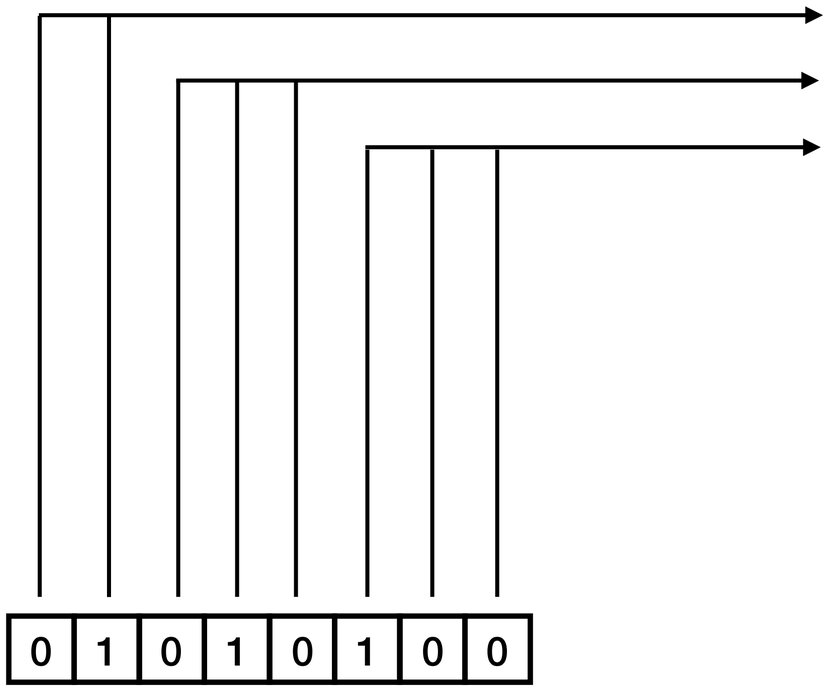
000 [0D], AC
001 [0X], AC
010 [YD], AC
011 [YX], AC
100 [0D], X
101 [0D], Y
110 [0D], OUT
111 [Y, X++], OUT



RAM address

ALU destination

Input



instruction
fixed length

Bus access
Address mode/Branch condition
Instruction

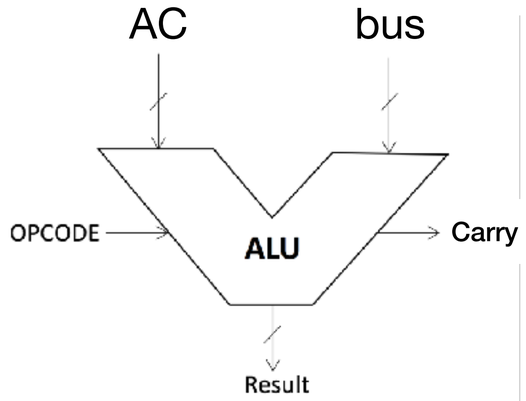
00 Operand (D) to bus
01 RAM value to bus
10 AC value to bus
11 Input to bus

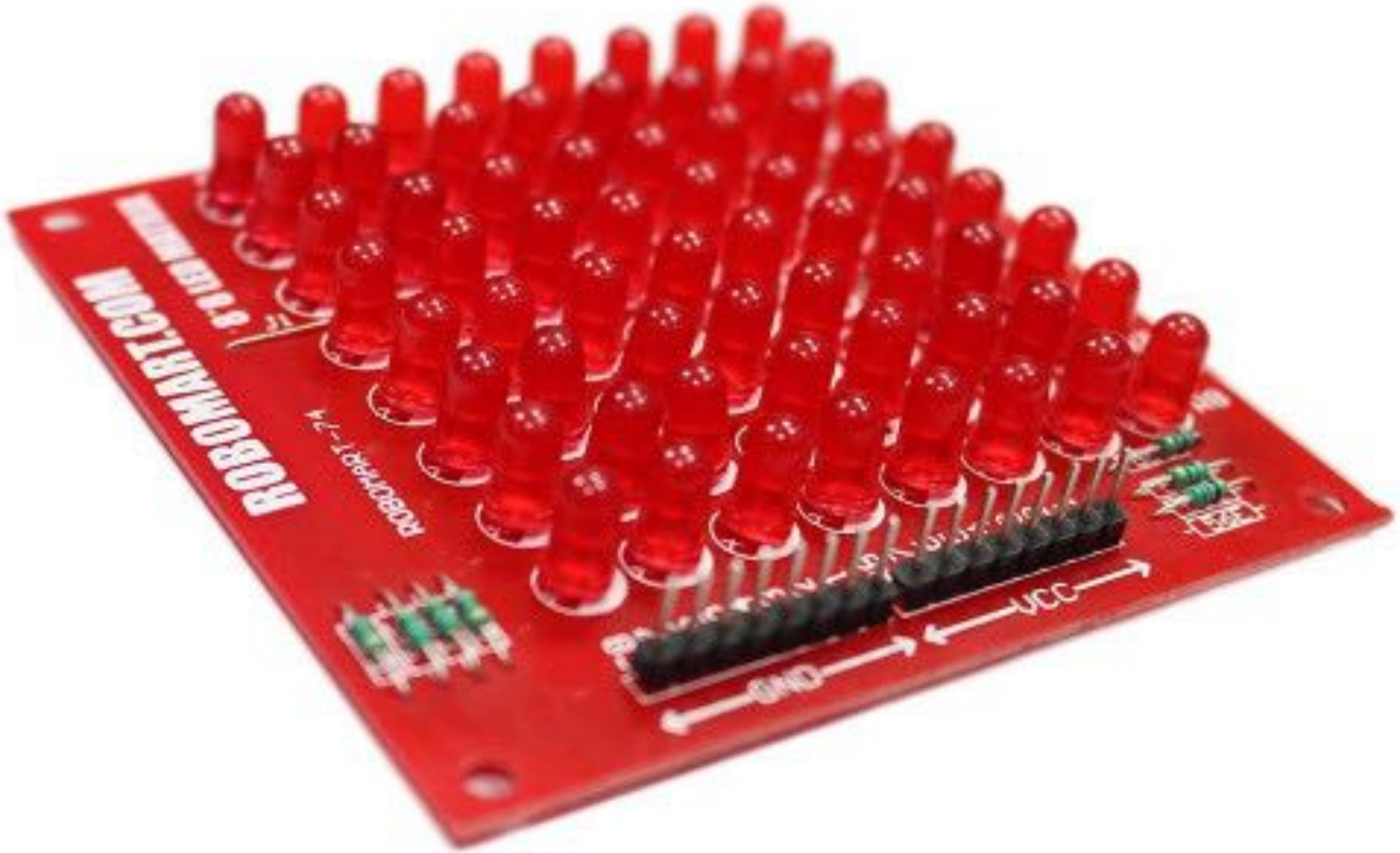
000 LD
001 AND
010 OR
011 XOR
100 ADD
101 SUB
110 ST
111 Branch

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

RAM address

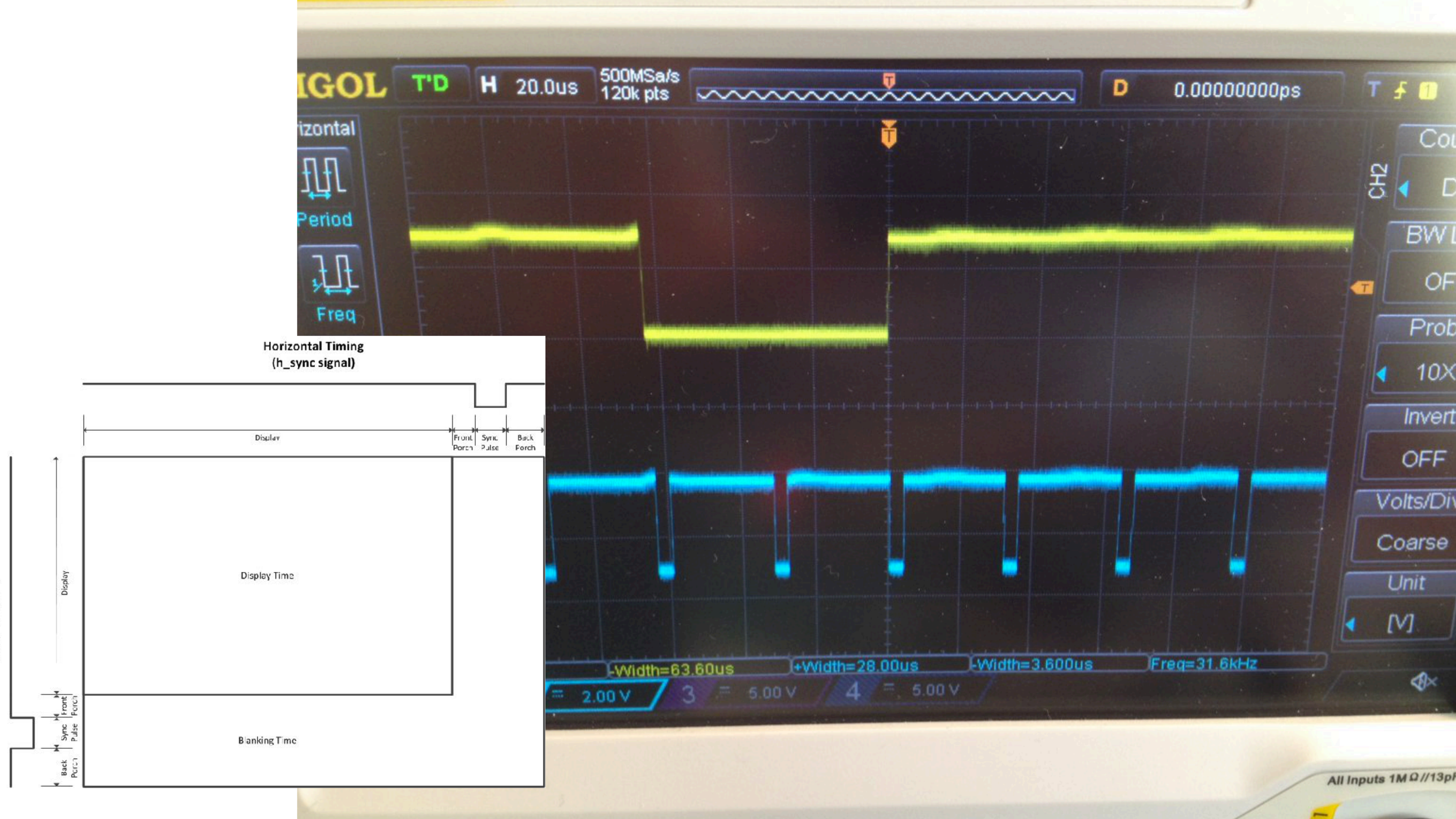
ALU destination



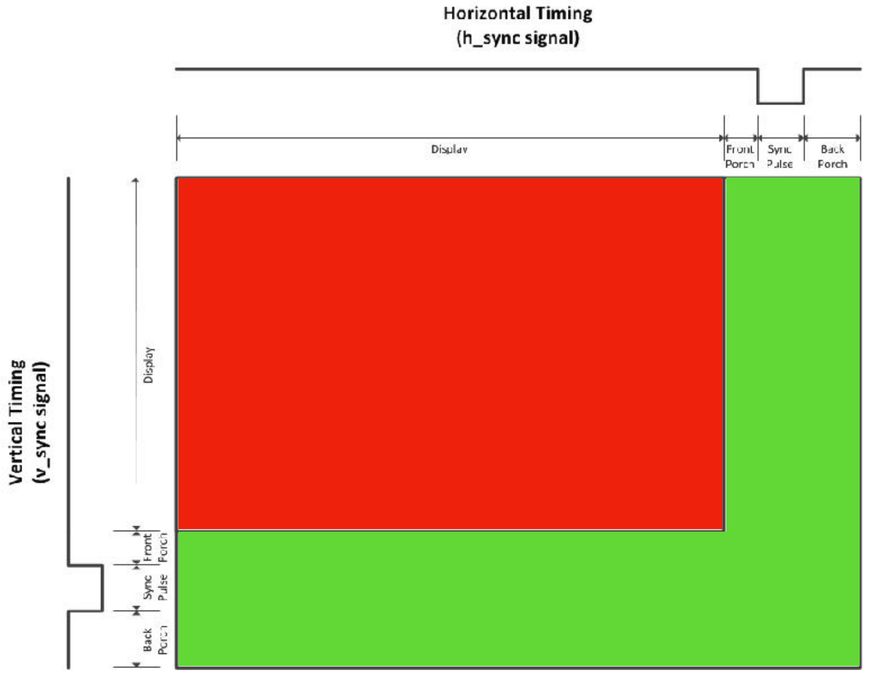

ROBOMARTCOM
ROBOPRO1-24
GND
VCC

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

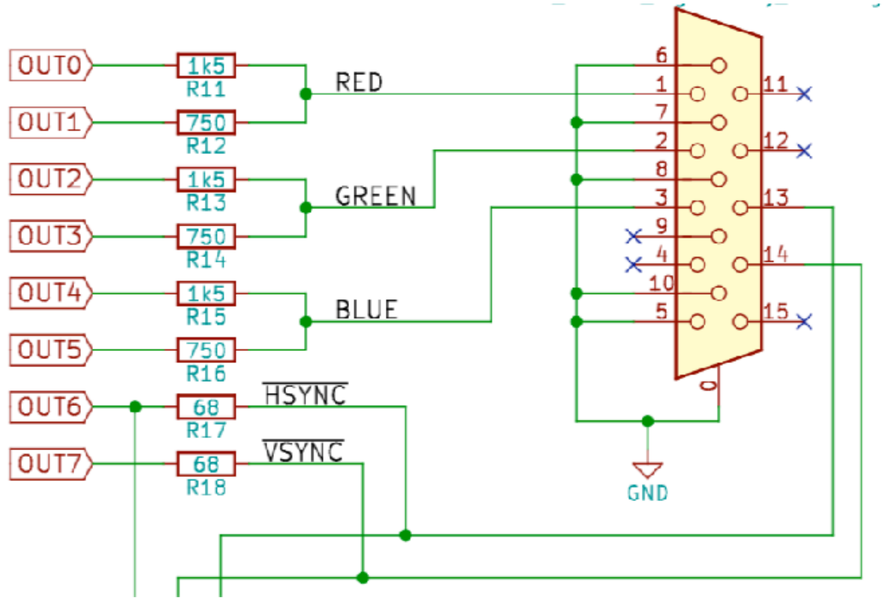

IGOL
T'D
H 20.0us
500MSa/s
120k pts
D 0.00000000ps
izontal
Period
Freq
CH2
Cou
BW L
OF
Prob
10X
Invert
OFF
Volts/Div
Coarse
Unit
[V]
-Width=63.60us
+Width=28.00us
-Width=3.600us
Freq=31.6kHz
2.00 V
5.00 V
5.00 V
All Inputs 1MΩ//13pF
Horizontal Timing
(h_sync signal)
Display
Front Porch
Sync Pulse
Back Porch
Display
Display Time
Blanking Time
Front Porch
Sync Pulse
Back Porch

VGA

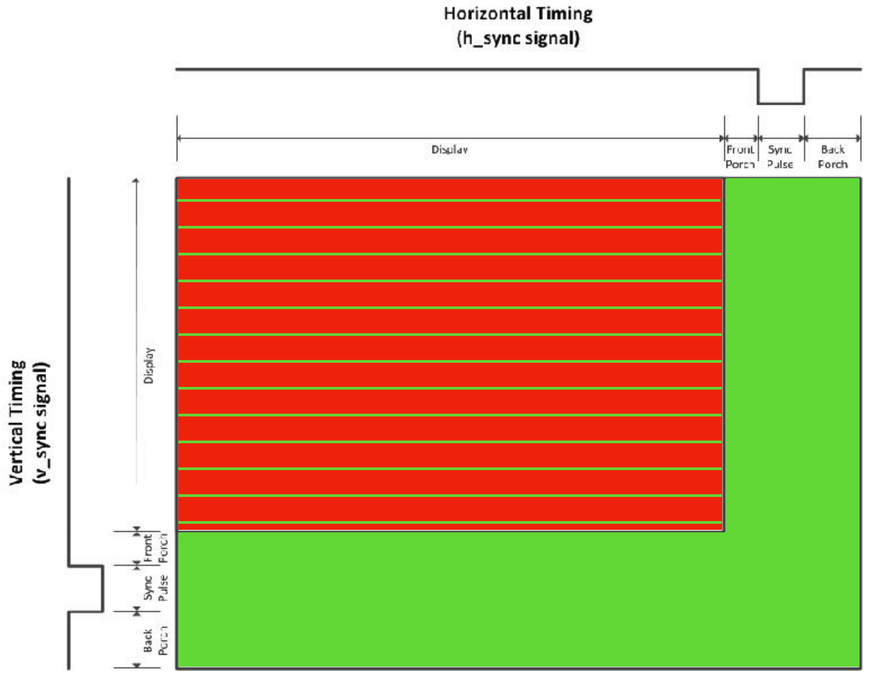


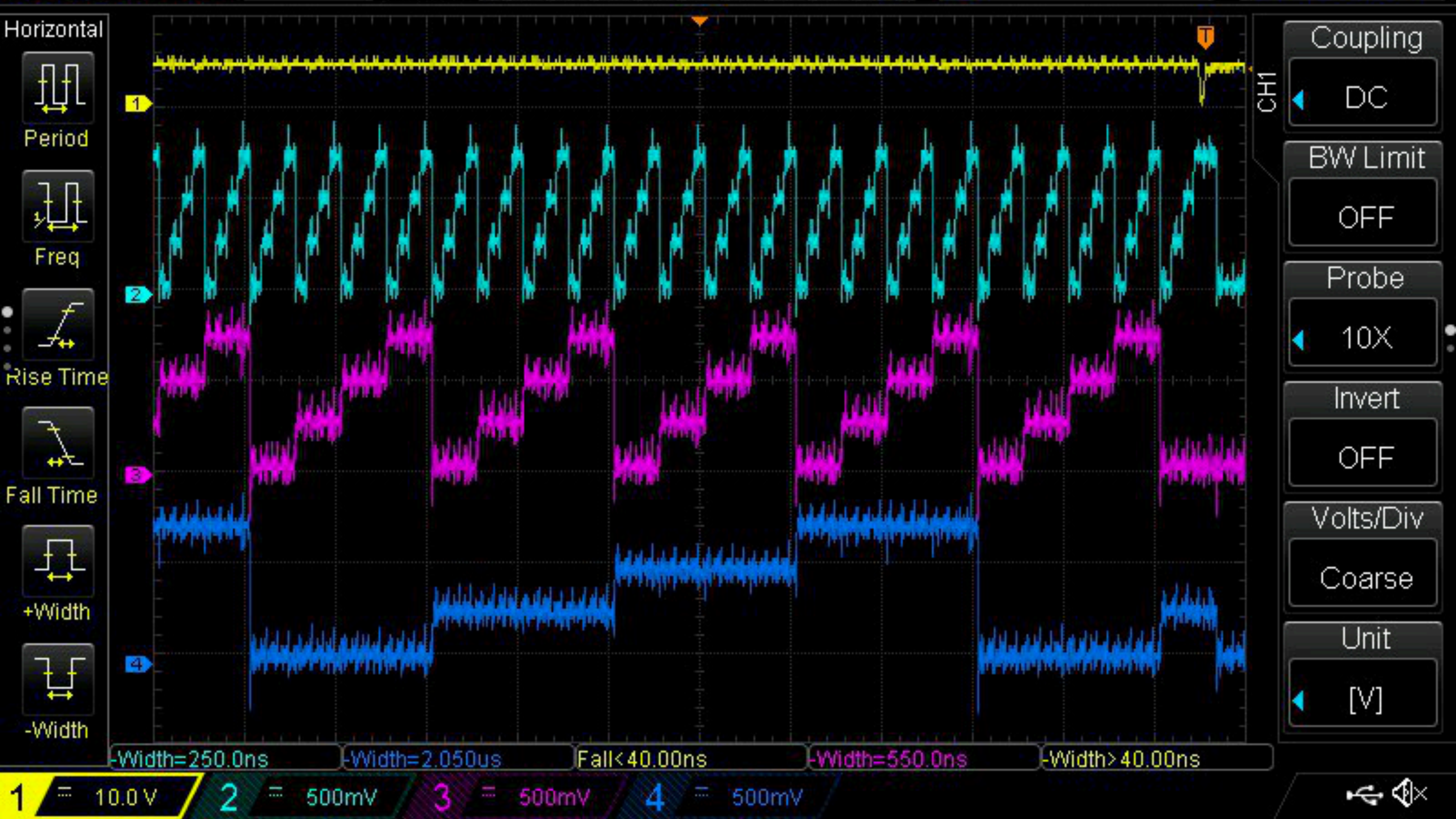
- A lot of time goes into doing the VGA signal, but the system still is fast enough
- Timing is critical
  - Solved by using an interpreter that runs a few opcodes at a time and checks if VGA output needs to be done before running the next few opcodes
    including all the VGA output and timing

VGA

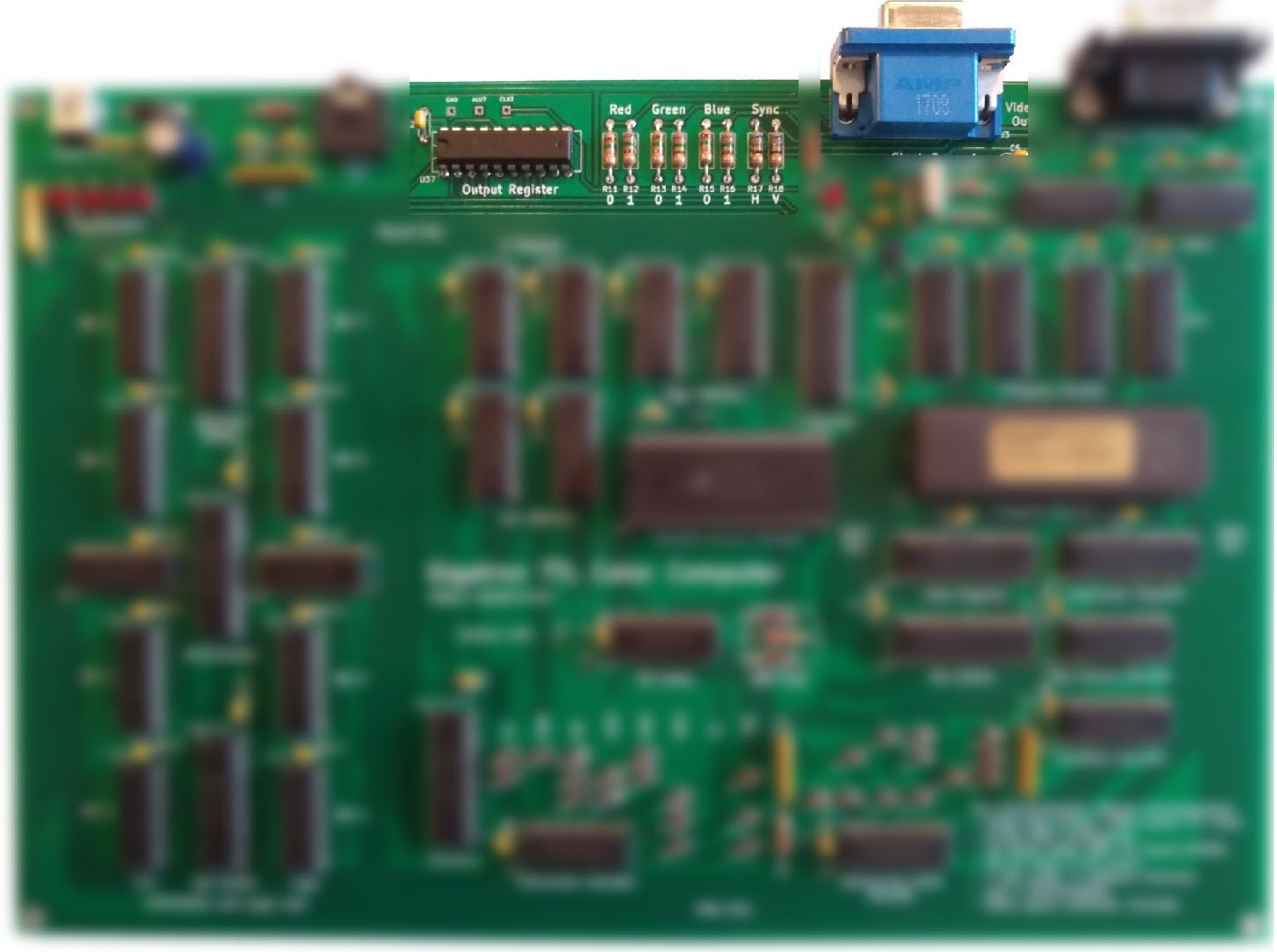


- Because of the speed of the RAM, we use 160 pixels in each scanline instead of 640
- Because of the amount of memory needed for a full image, we repeat each scanline 4 times
  - Or 3, followed by a blank line, for retro effect
- Effective resolution 160x120



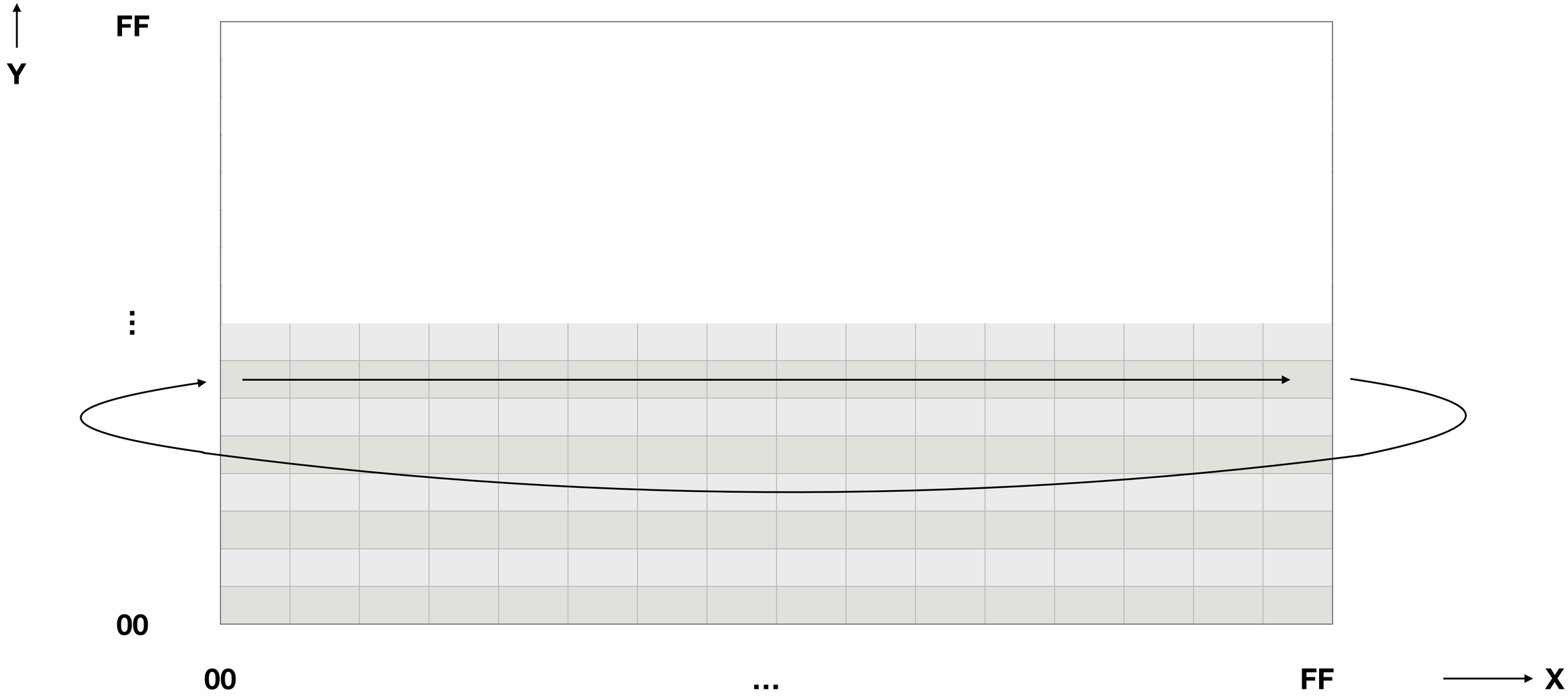

Horizontal
Period
Freq
Rise Time
Fall Time
+Width
-Width
CH1
Coupling
DC
BW Limit
OFF
Probe
10X
Invert
OFF
Volts/Div
Coarse
Unit
[V]
-Width=250.0ns
-Width=2.050us
Fall<40.00ns
-Width=550.0ns
-Width>40.00ns
1 10.0 V
2 500mV
3 500mV
4 500mV


IPS LED
LG

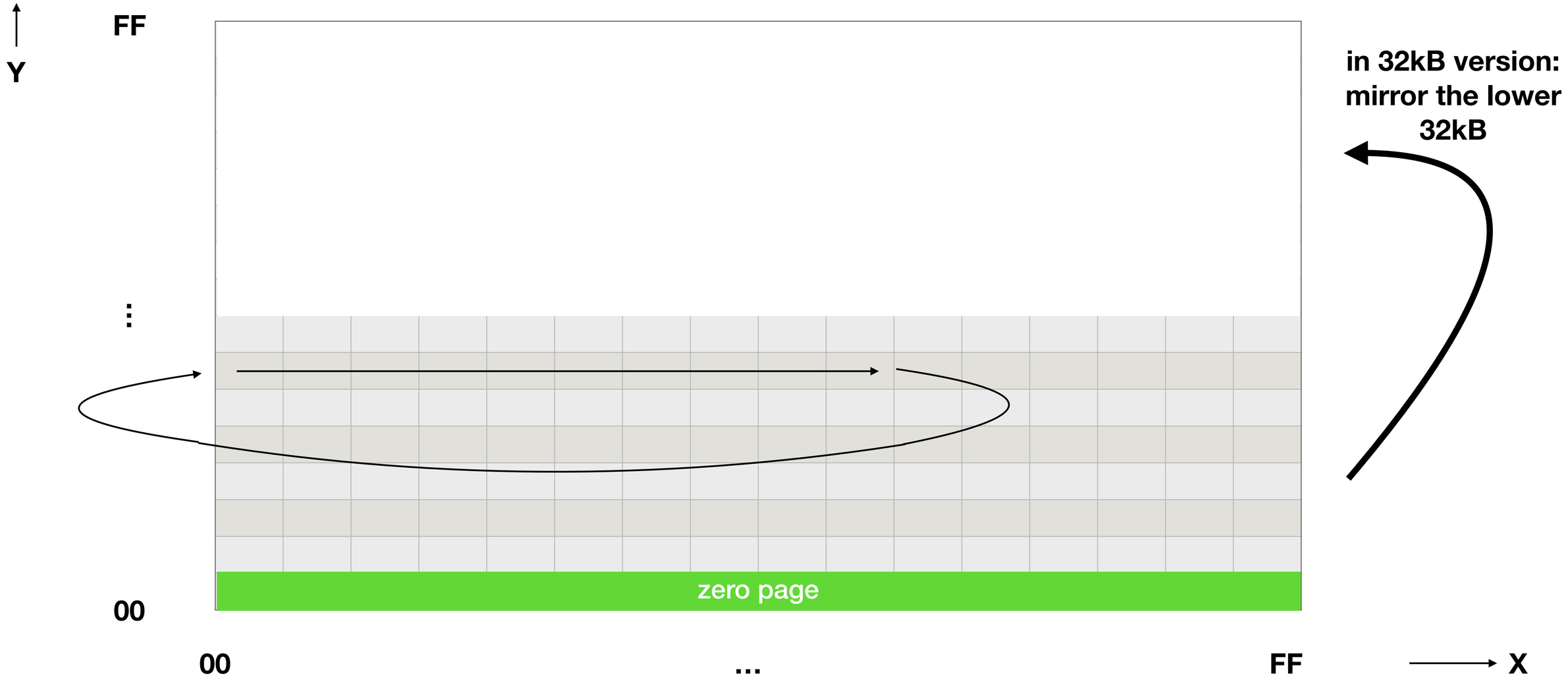

Red
Green
Blue
Sync
R11 R12 R13 R14 R15 R16 R17 R18
0 1 0 1 0 1 H V
Output Register
U37
AMP
1709


BenQ
PHILIPS

RAM Memory map and [Y, X++]

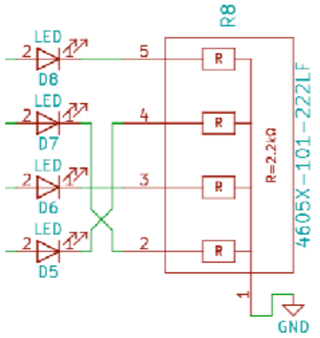


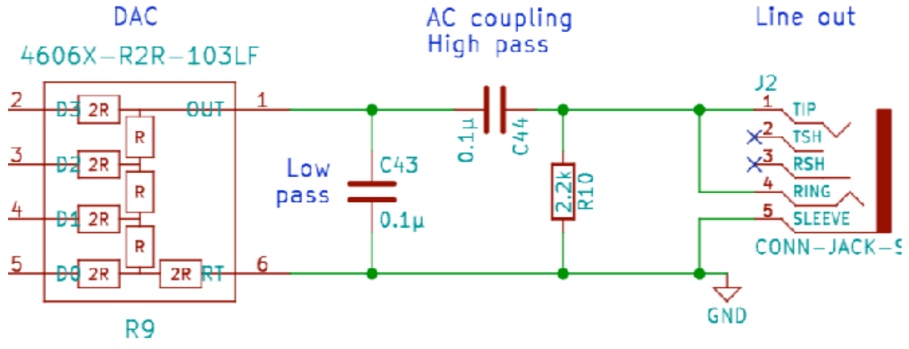
RAM Memory map and [Y, X++]

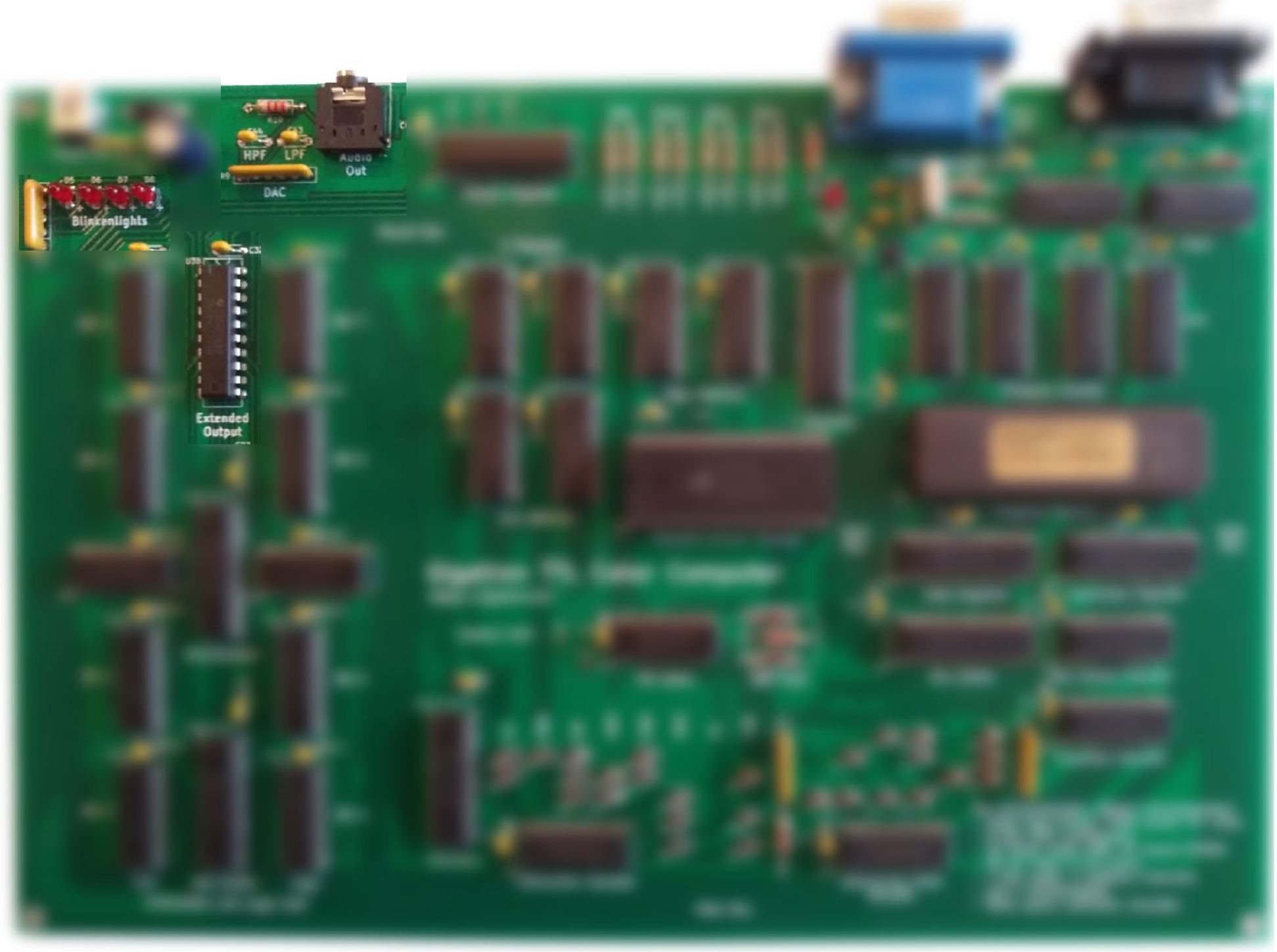


Can we do more output?

- 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

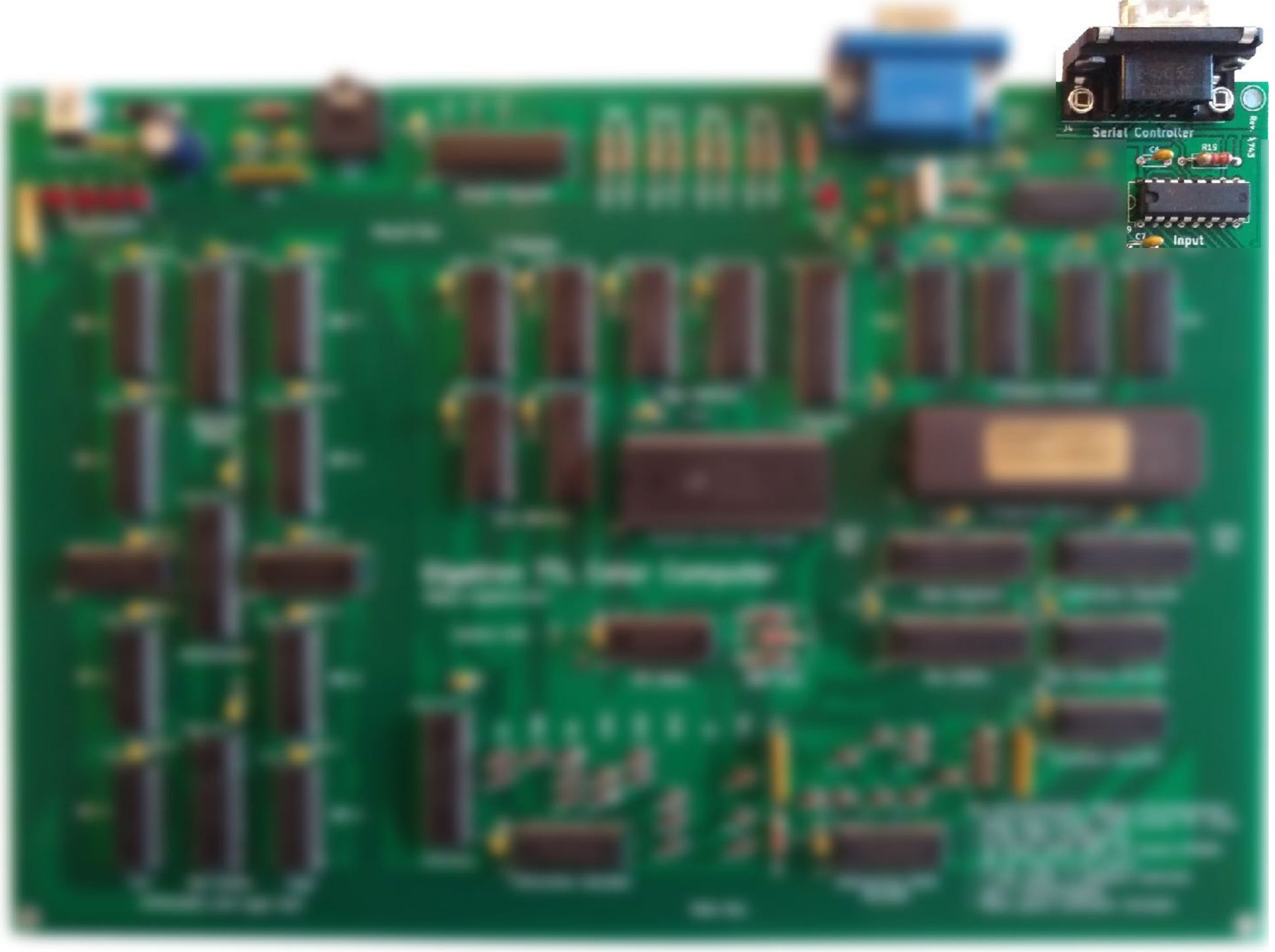





HPF
LPF
Audio
Out
DAC
Blinkenlights
Extended
Output

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


Serial Controller
R19
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}

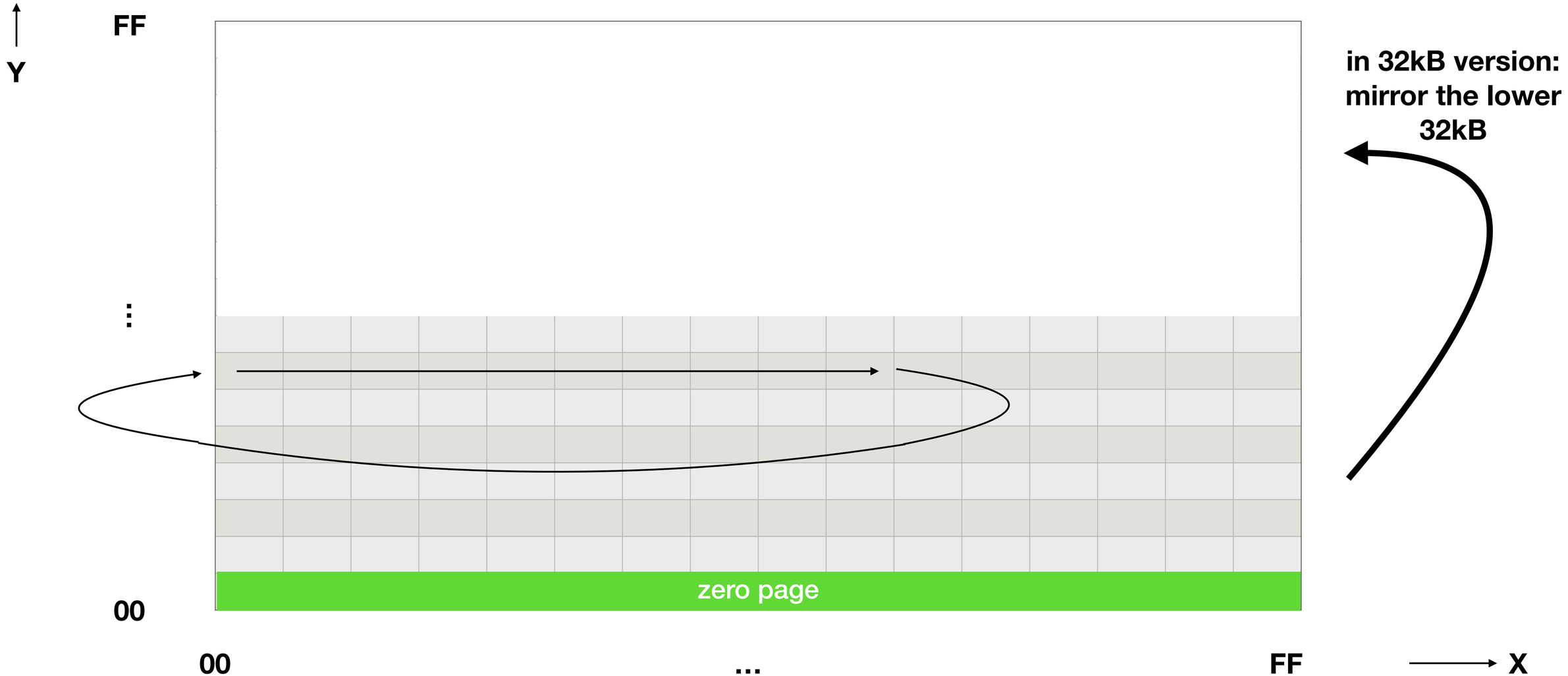
```
[def                                    {10 GOTO 80}
  $4448 D= {Middle of screen}           {20 D=$4448: REM MIDDLE OF SCREEN}
  [do
    C [if<0 15 else 5] D.               {30 IF C<0 POKE D,15 ELSE POKE D,5}
    C C+ C=                             {40 C=C+C}
    D 1+ D=                        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

Programming

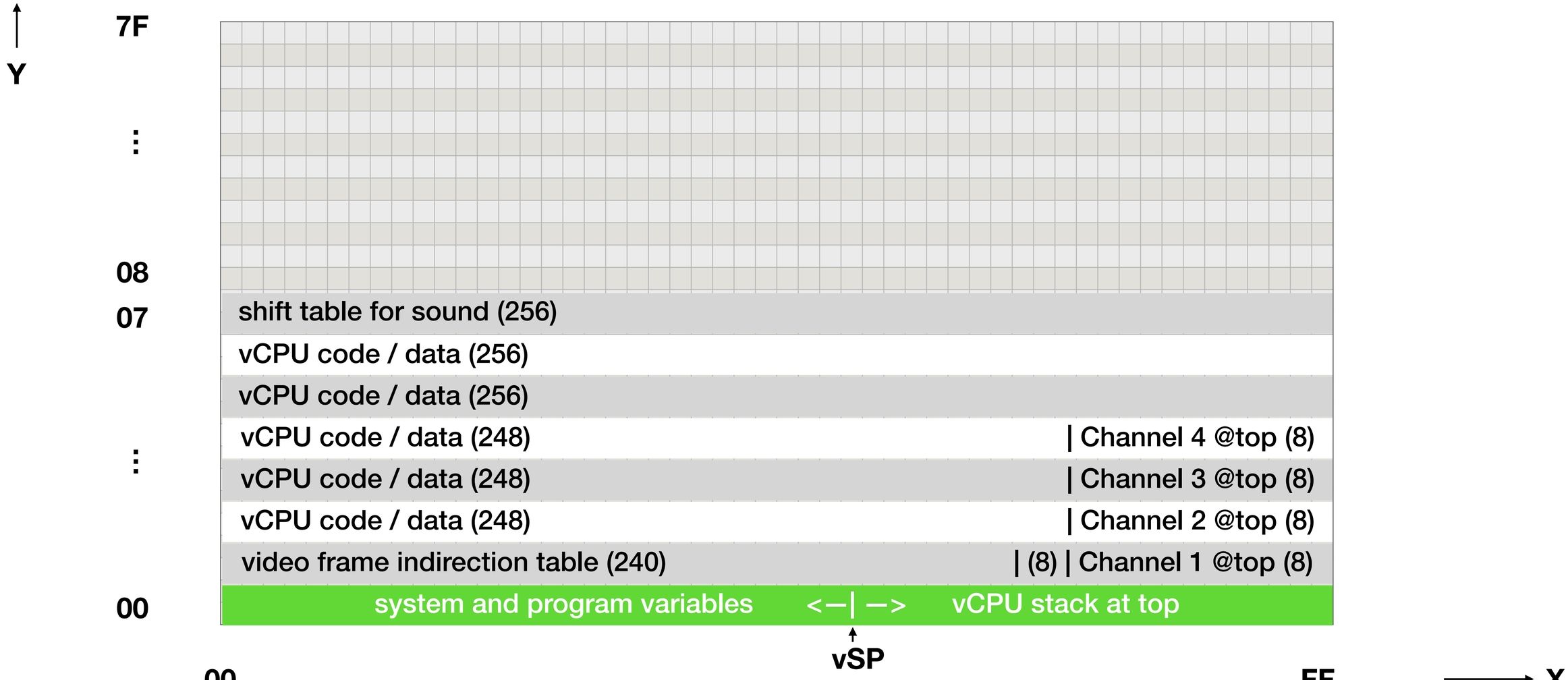
```
{Compute largest 16-bit Fibonacci number and plot it on screen}

[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}
```

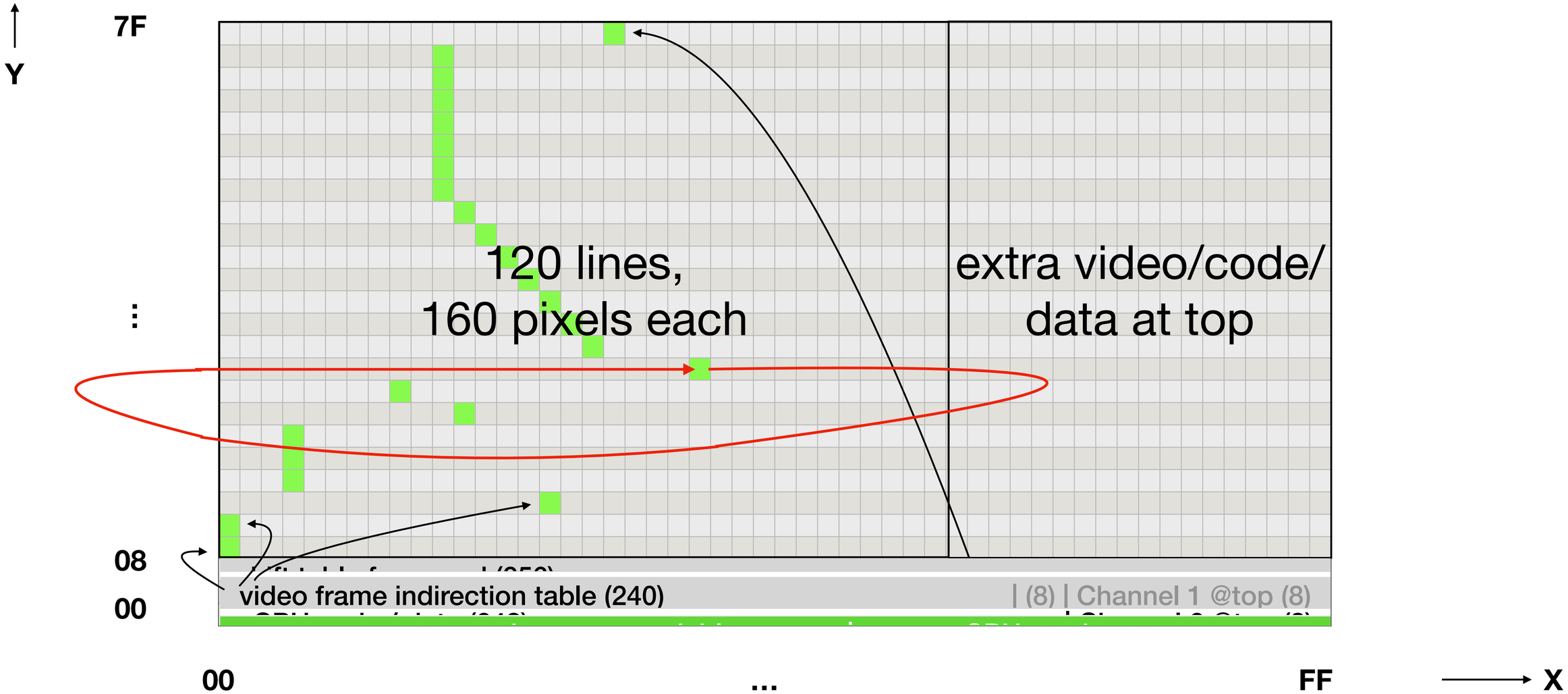
RAM Memory map and [Y, X++]



RAM Memory map



RAM Memory map



Interpreter

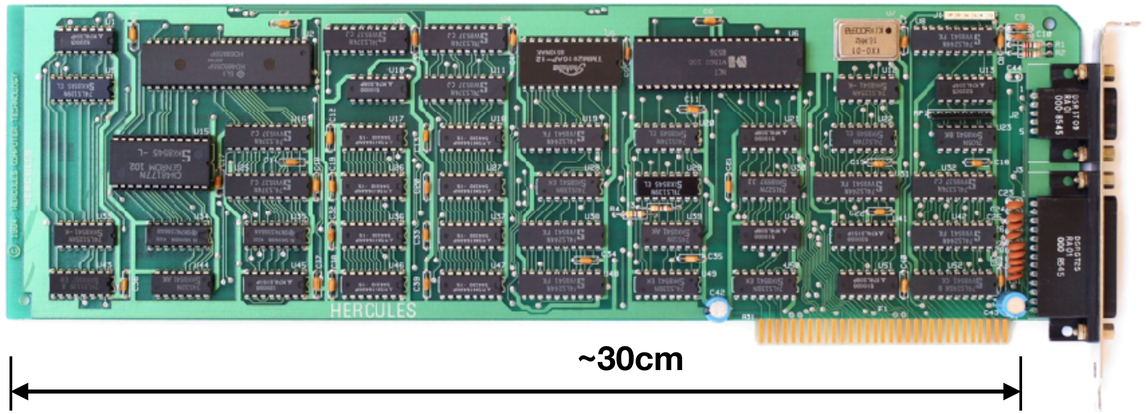
- Emulates a virtual PC
  - Runs vCPU instructions from RAM (using a vPC)
  - Emulates a 16-bit architecture (with a 16-bit vAC, vSP)
- Each instruction is emulated, track is kept of the amount of clock cycles
- A bit like SWEET16 on Apple

Result

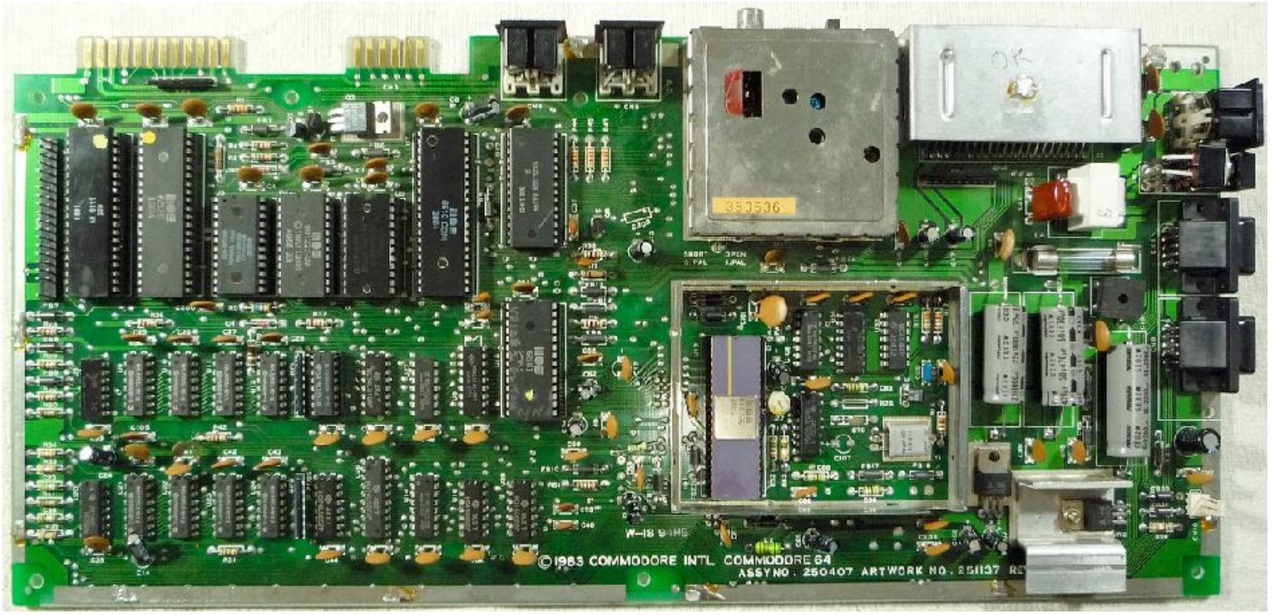
- 16 bit CPU
- Von Neumann architecture

- On an 8 bit CPU
- With Harvard Architecture

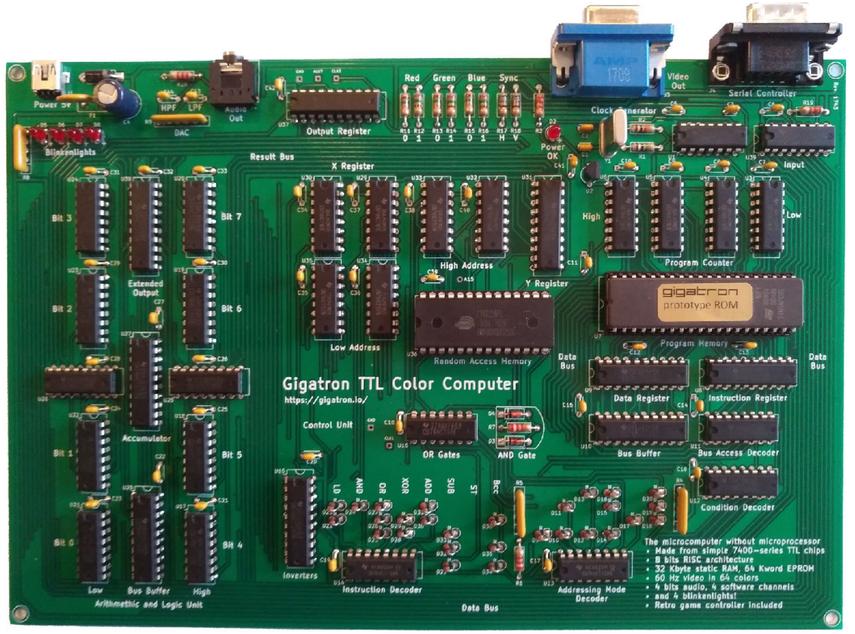

TRS color computer 6MHz



~30cm

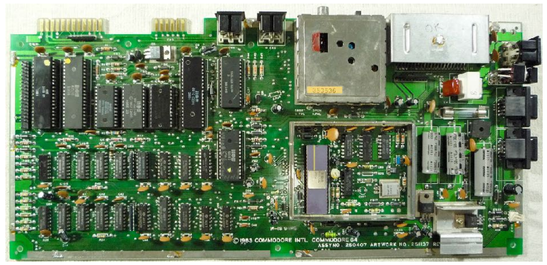


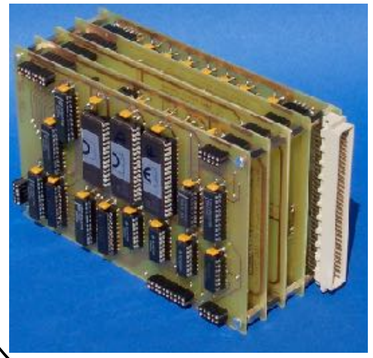
~39cm

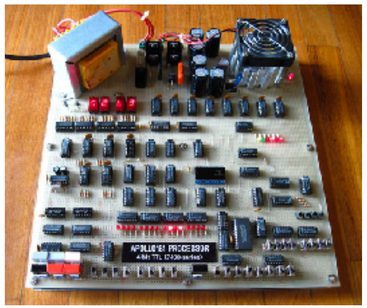


~24cm

Needs a microprocessor chip

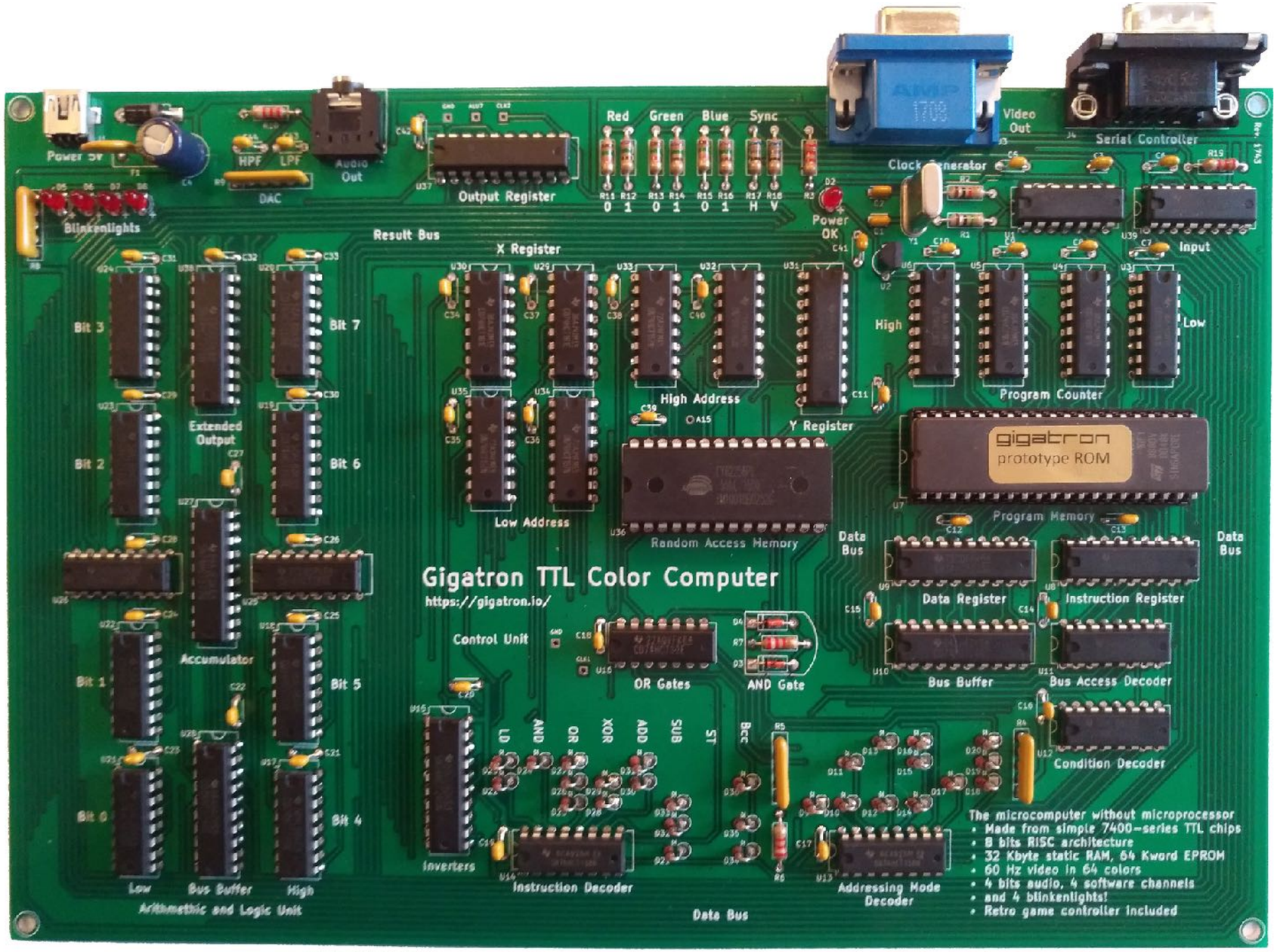


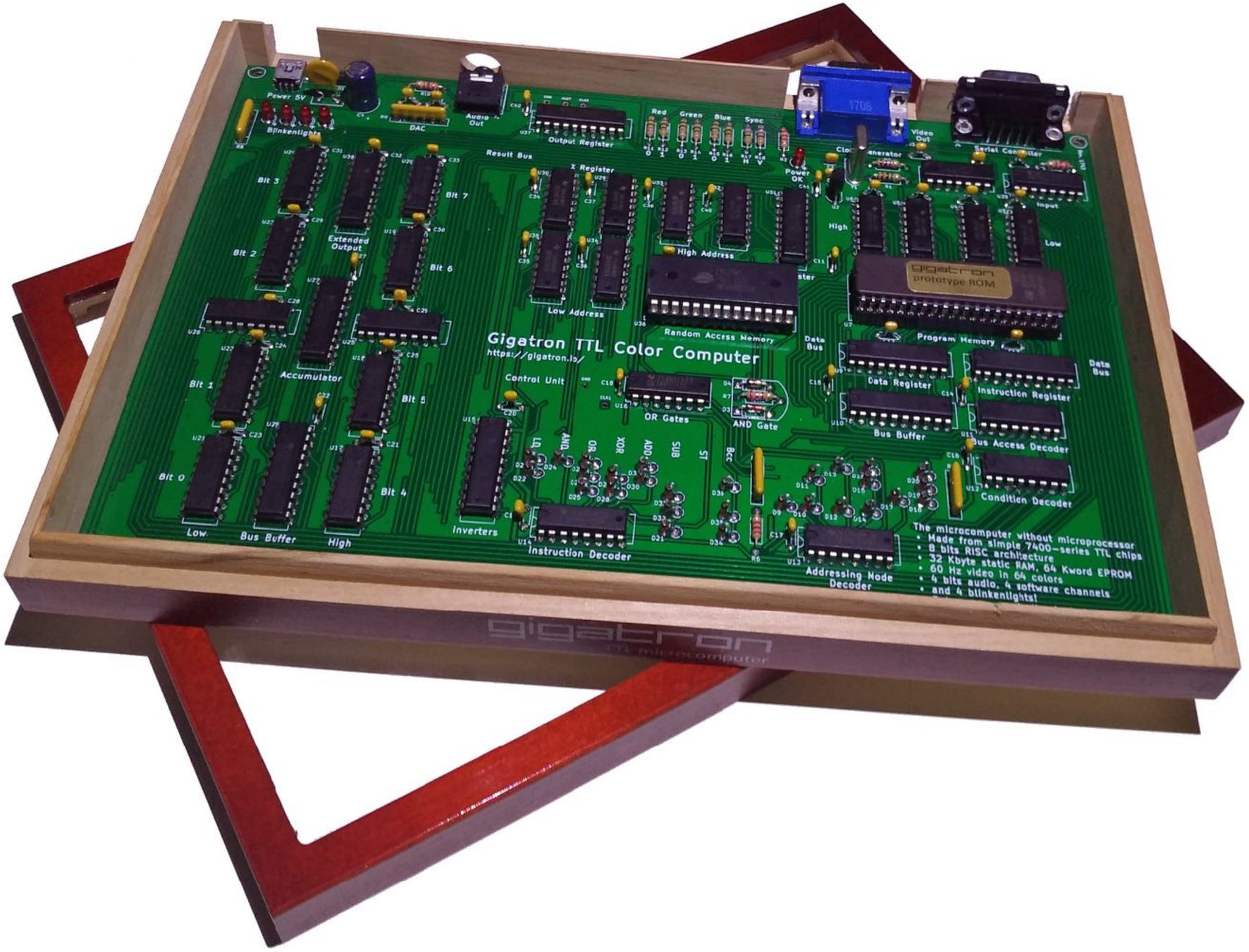


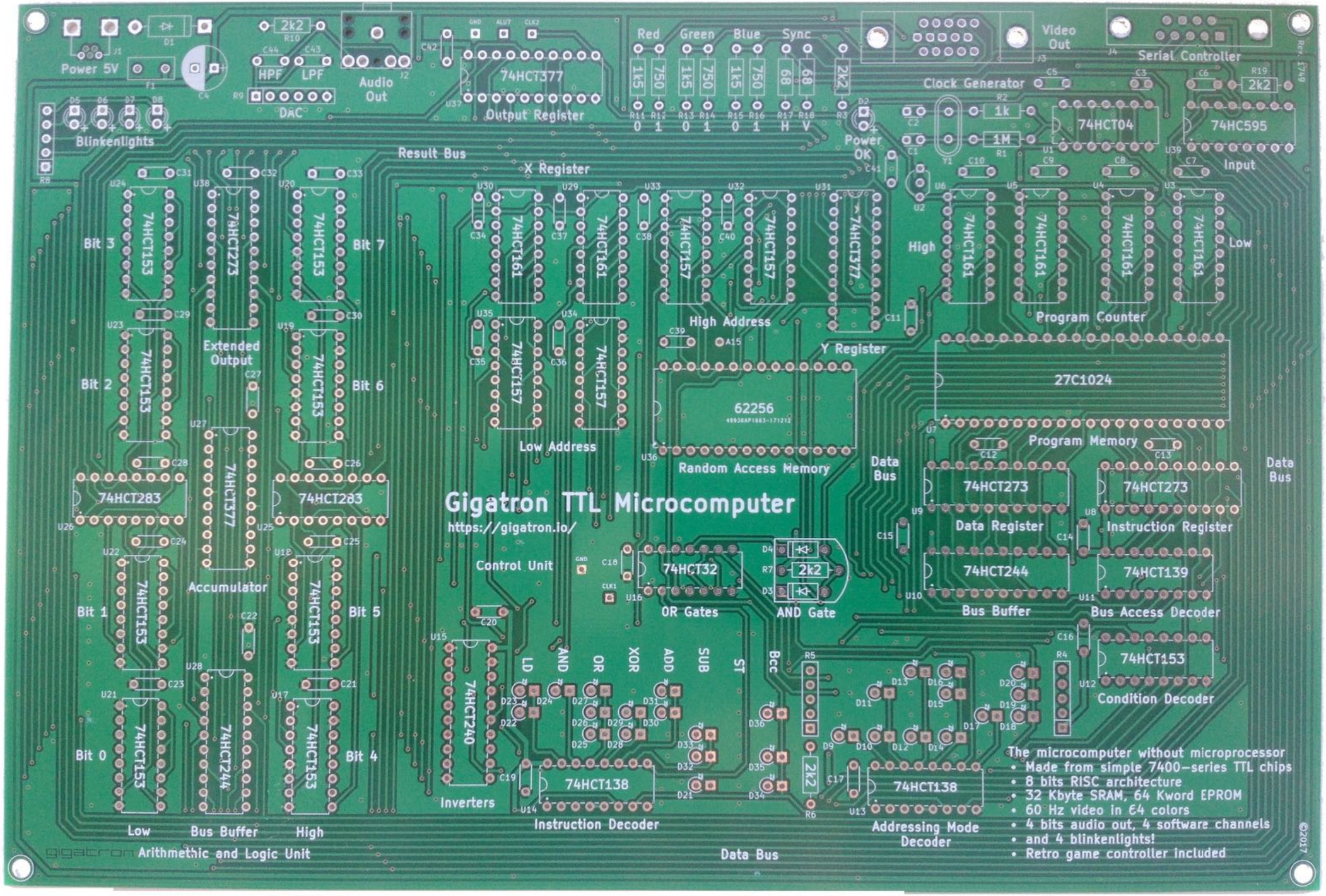


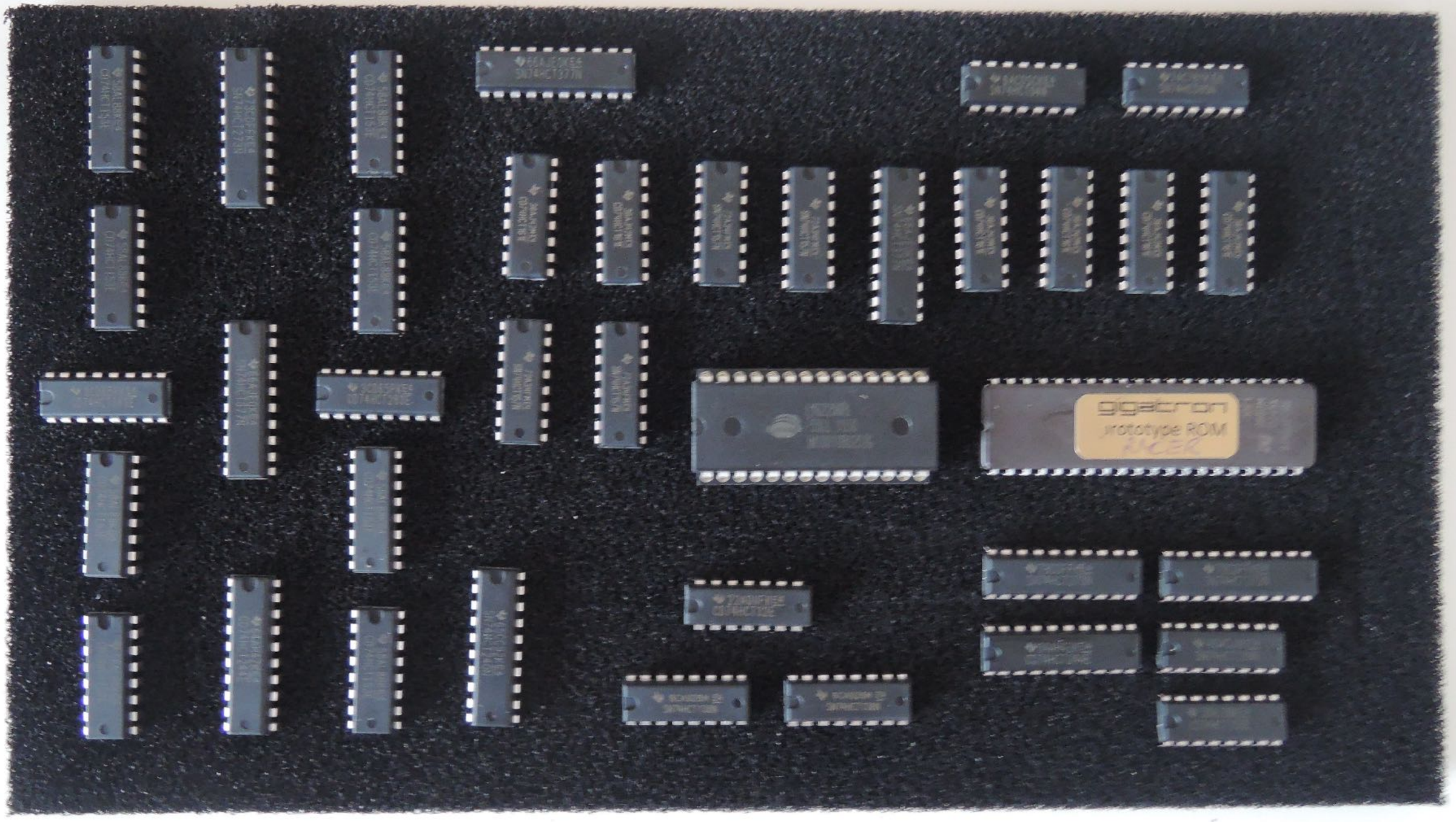
Needs multiple boards

Does not have audio/video


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


Gigatron TTL Microcomputer
https://gigatron.io/
Power 5V
Audio Out
HPF
LPF
DAC
74HCT377
Output Register
Red
Green
Blue
Sync
Video Out
Serial Controller
Clock Generator
74HCT04
74HC595
Input
Blinkenlights
Result Bus
X Register
Y Register
High Address
Low Address
74HCT161
74HCT157
74HCT377
Program Counter
High
Low
27C1024
Program Memory
62256
Random Access Memory
Data Bus
74HCT273
Data Register
Instruction Register
74HCT244
Bus Buffer
74HCT139
Bus Access Decoder
74HCT153
Condition Decoder
Control Unit
74HCT32
OR Gates
AND Gate
74HCT240
Inverters
74HCT138
Instruction Decoder
Addressing Mode Decoder
LD
AND
OR
XOR
ADD
SUB
ST
Bcc
Bit 0
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
Extended Output
Accumulator
74HCT283
74HCT273
74HCT244
Low
Bus Buffer
High
Arithmetic and Logic Unit
The microcomputer without microprocessor
• Made from simple 7400-series TTL chips
• 8 bits RISC architecture
• 32 Kbyte SRAM, 64 Kword EPROM
• 60 Hz video in 64 colors
• 4 bits audio out, 4 software channels
• and 4 blinkenlights!
• Retro game controller included
©2017


gigatron
prototype ROM

gigatron

The TTL based microcomputer system

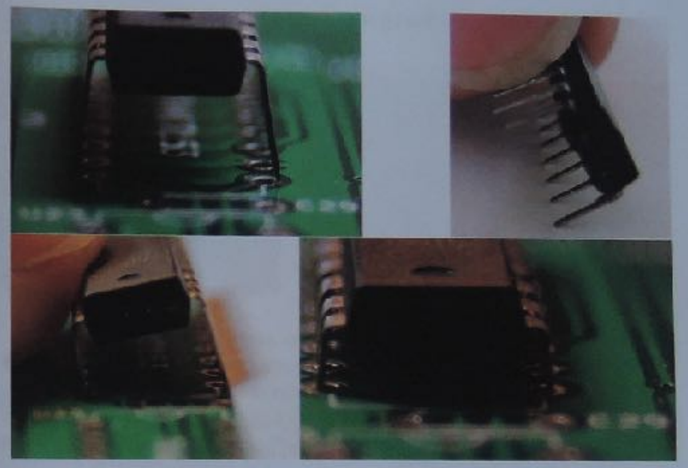
ASSEMBLY MANUAL

hen you turn the PCB

ature and be
a pen, just
ther hand to
the PCB, do
g iron.

enough
solder

IC
e



Take care as this requires only a gentle push. The little push on both sides should straighten the pins so you can insert it easily into the PCB.

So, the order is:

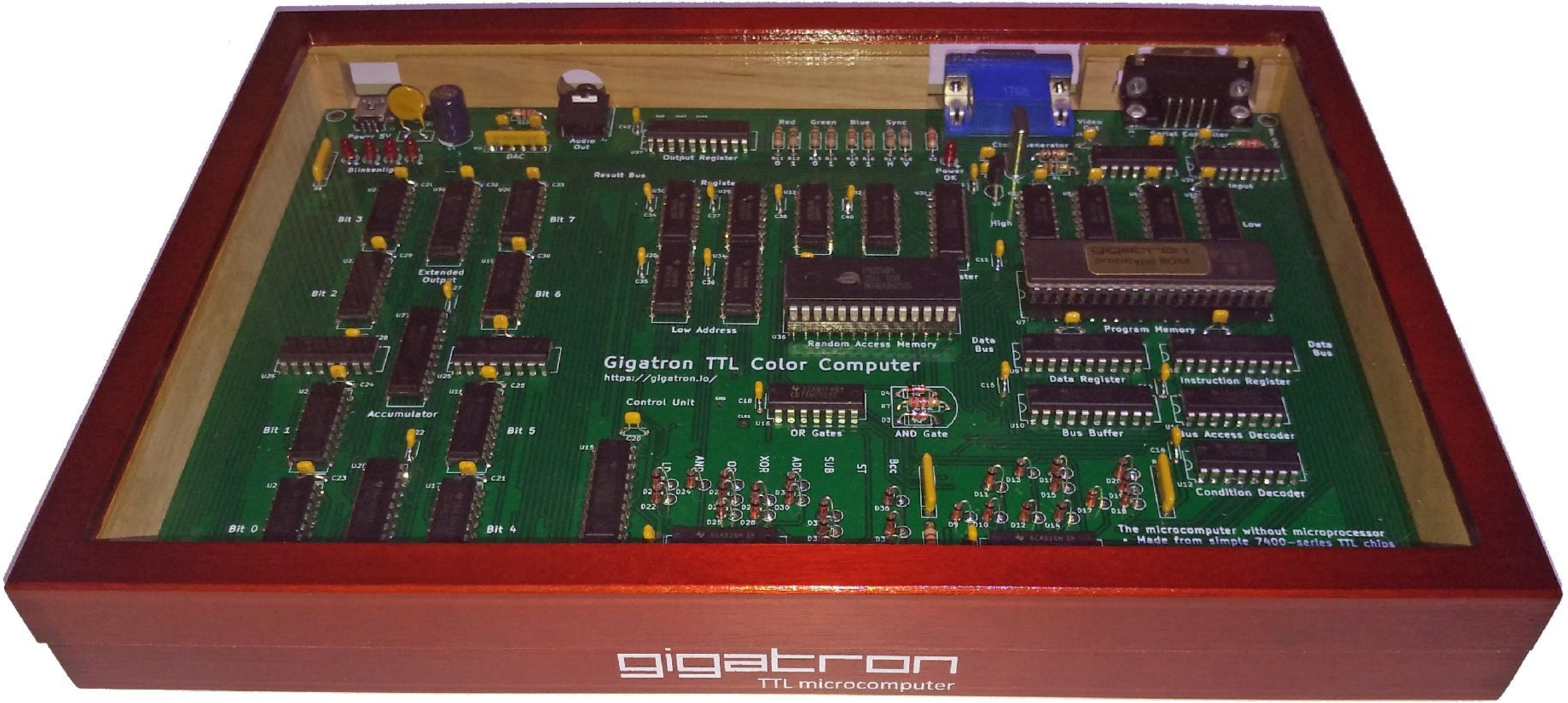
- briefly heat up both the PCBs metal ring and the wire or pin
- apply solder
- remove solder
- remove soldering iron

You should now have successfully soldered the component onto the PCB! Make sure the solder joint looks smooth. There should not be any holes. Those happen when you do not use enough solder or when the temperature is too low. Neither should there be too much, this could cause electrical shorts with adjacent components. Cut away the excess wire, if there is any.

Next, check if the component is placed correctly, i.e. they sit on the PCB. If not, reheat the faulty joint and **gently** push the component in when the solder is liquid. If you push too hard, you will damage the PCB as the traces will come loose!

This can be handy for some components like LEDs that, in most cases, will never end up in the correct position on the first try. Here's the trick: you insert the LED into the PCB


gigatron
The TTL based
microcomputer system
Gigatron TTL color computer
https://gigatron.io/


Gigatron TTL Color Computer
https://gigatron.io/
Control Unit
Accumulator
Bit 0
Bit 1
Bit 2
Bit 3
Bit 4
Bit 5
Bit 6
Bit 7
Extended Output
Low Address
Random Access Memory
Program Memory
Data Register
Instruction Register
Bus Buffer
Bus Access Decoder
Condition Decoder
OR Gates
AND Gate
Output Register
Result Bus
Data Bus
Red
Green
Blue
Sync
Video
Audio Out
DAC
Power 5V
Power OK
High
Low
LD
AND
OR
XOR
ADD
SUB
ST
Bcc
The microcomputer without microprocessor
• Made from simple 7400-series TTL chips
gigatron
TTL microcomputer

gigatron.io

- Mailing list:
  https://mm.gigatron.io/lijsten/listinfo/announce

- Source code:
  https://github.com/kervinck/gigatron-rom

- Visualizer by Martin Sedlák:
  https://www.crabaware.com/Test/gigatron_emu.zip

- Assembly walk-through videos:
  https://goo.gl/TtqqQR

gigatron.io

walter@gigatron.io