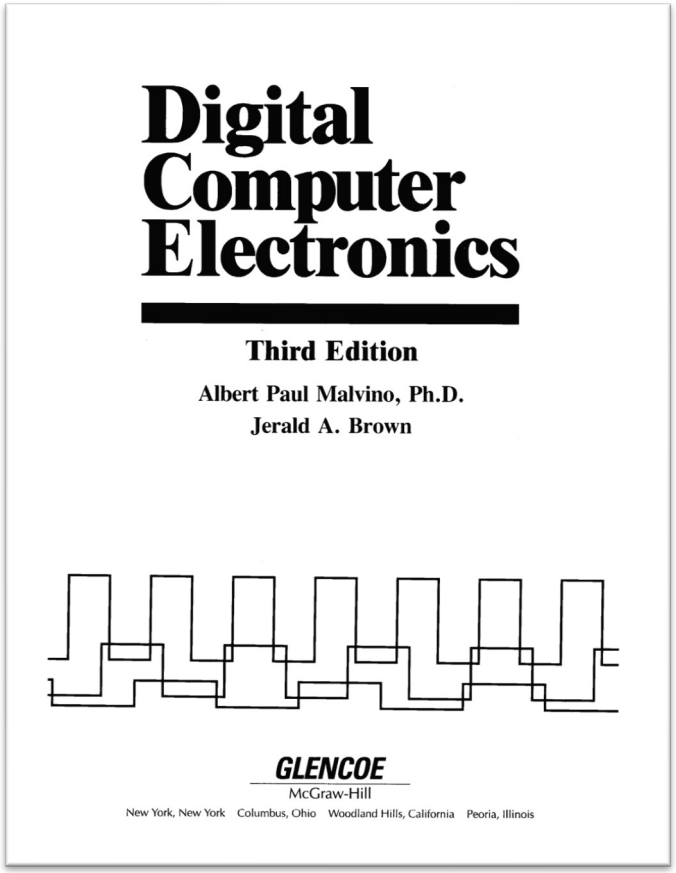
The ASAP CPU

Breadboard to Silicon

by Aaron Libokuohai Bovensiepen

Goal



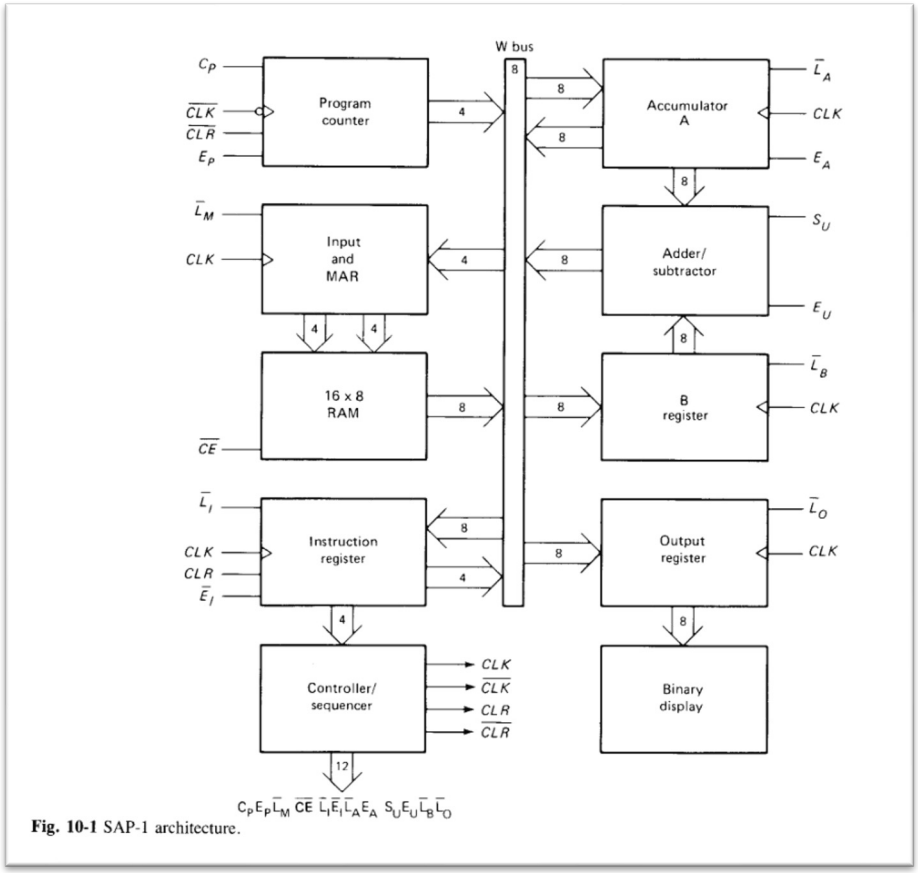
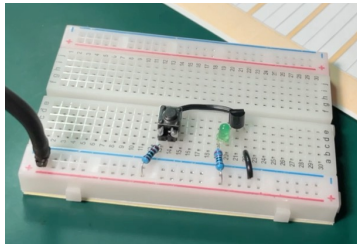


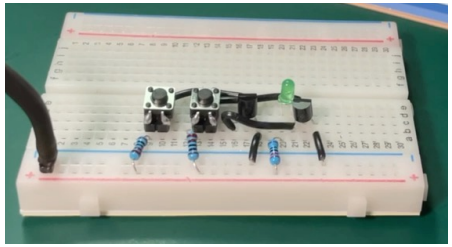
Fig. 10-1 SAP-1 architecture.

Building it from Scratch

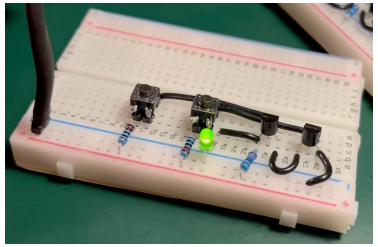
Logic Gates



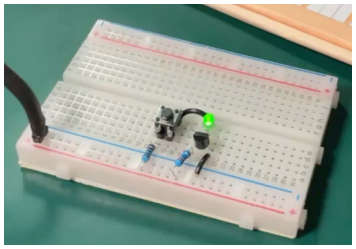
Buffer



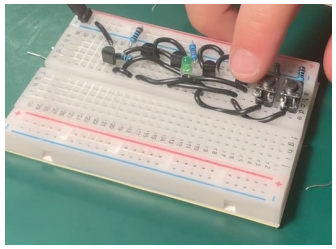
OR



NOR



Inverter

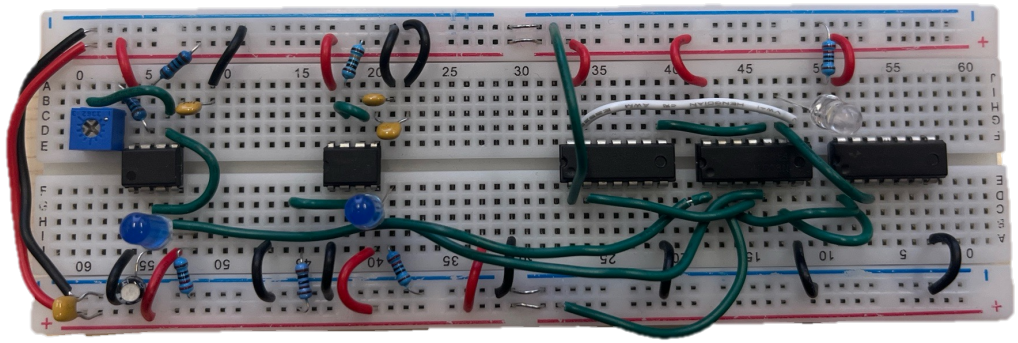


XOR

Clock

Determines when each step of an instruction happens.

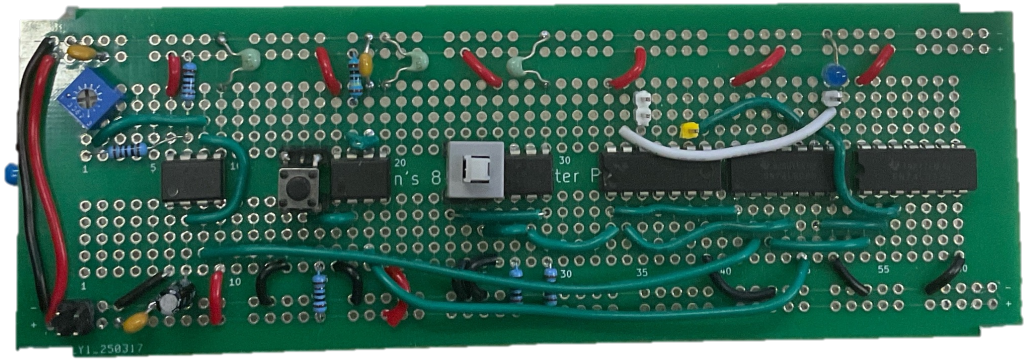
Type	Quantity
comparators	4
NOT	3
AND	3
OR	1
SR Latch	3



Clock

Determines when each step of an instruction happens.

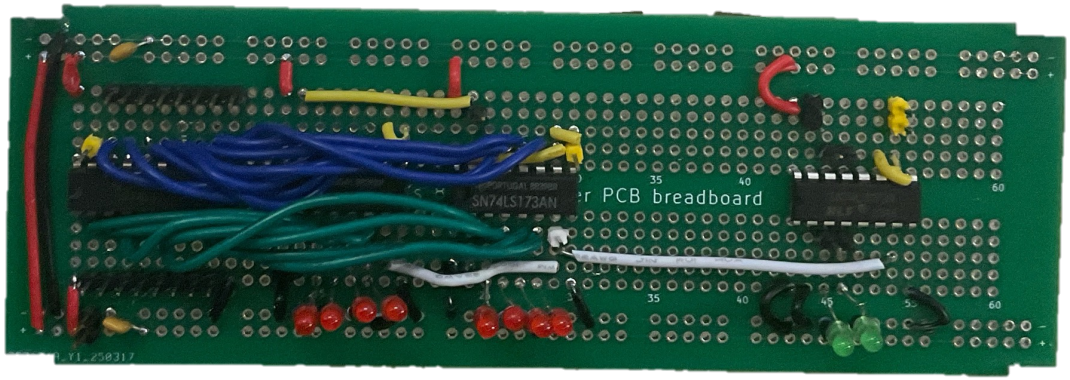
Type	Quantity
comparators	4
NOT	3
AND	3
OR	1
SR Latch	3



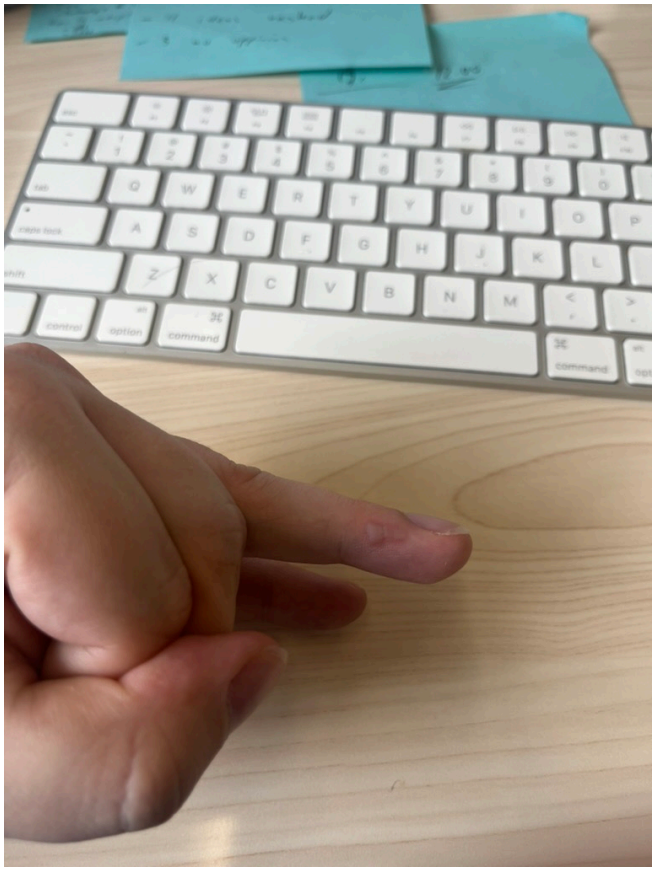
Register (A and B)

Stores data for processing or exchange between modules.

Type	Quantity
NOT	16
AND	32
OR	16
D Flip Flops	16
Tristate Buffer	16



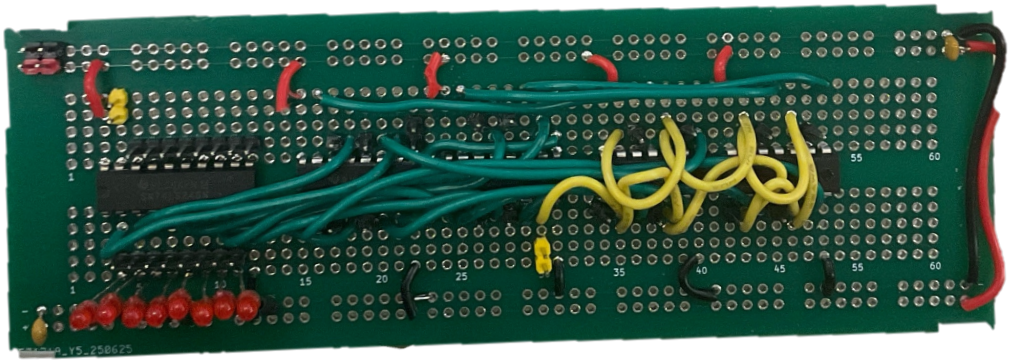
Stable and Dangerous



ALU

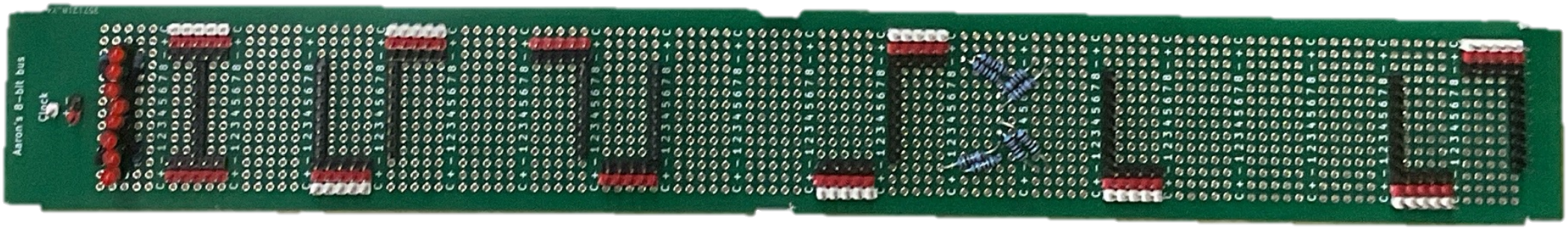
Executes addition and subtraction on top of the A and B register.

Type	Quantity
XOR	24
AND	16
OR	8
Tristate Buffer	8

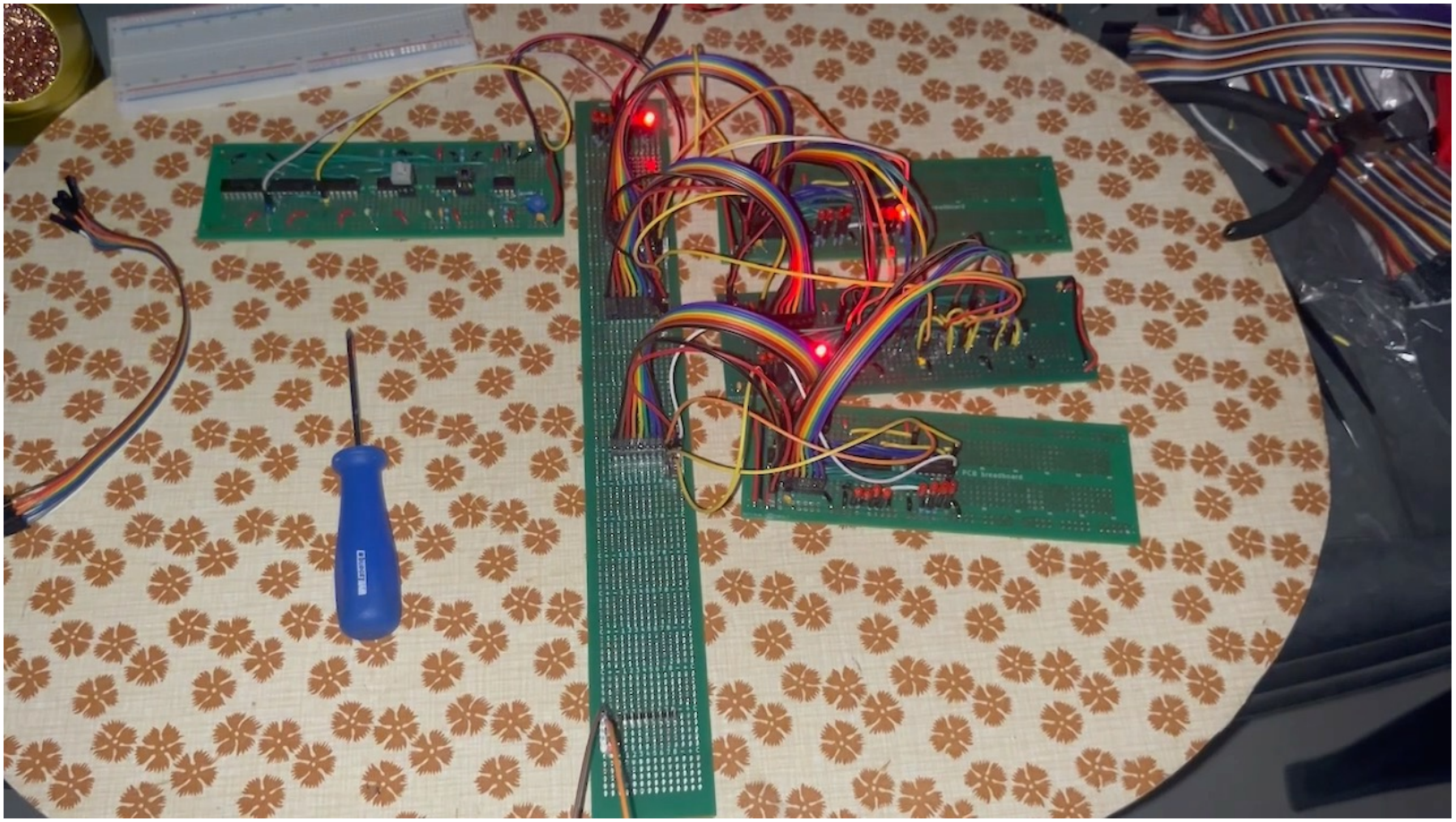


8-bit bus

Transfers path between the different modules.



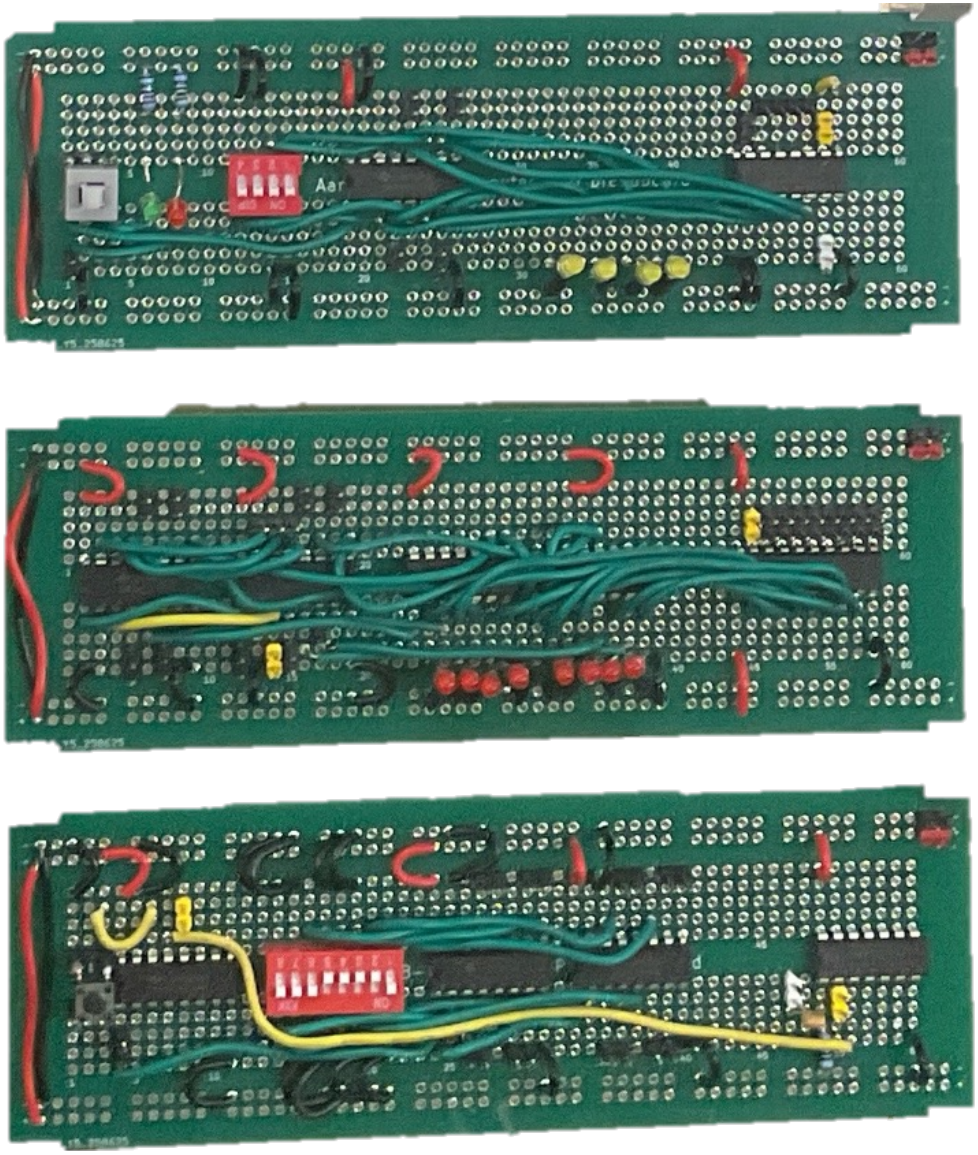
Counting



RAM

Stores data and instructions.

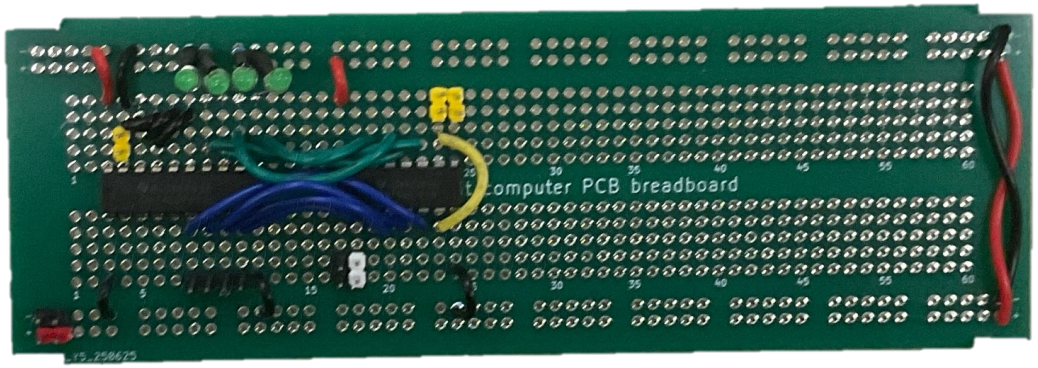
Type	Quantity
NOT	153
AND	322
OR	145
D flip-flop	132
Tristate Buffer	8
NAND	1



Program Counter

Keeps track of the next instruction address to run.

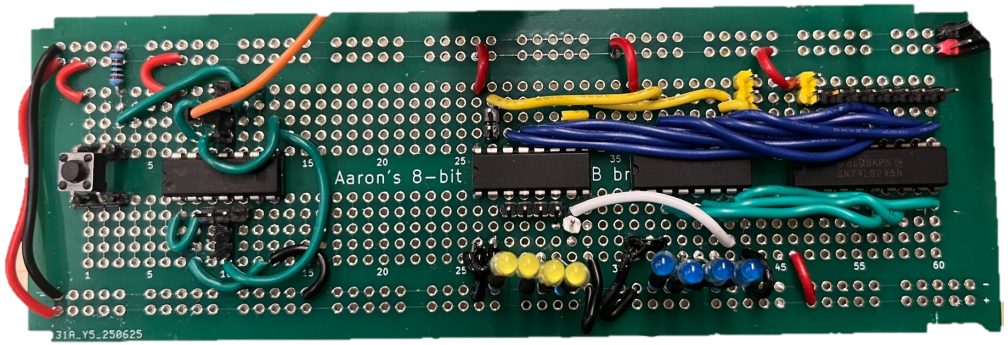
Type	Quantity
JK flip-flop	4
NOT	3
Buffer	1
AND	16
NAND	1
NOR	10
Tristate Buffer	4



Instruction Register

Holds the current instruction being executed.

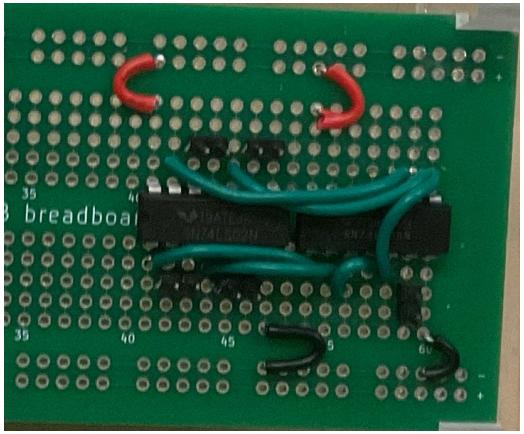
Type	Quantity
NOT	16
AND	32
OR	16
D flip-flop	16
Tristate Buffer	4

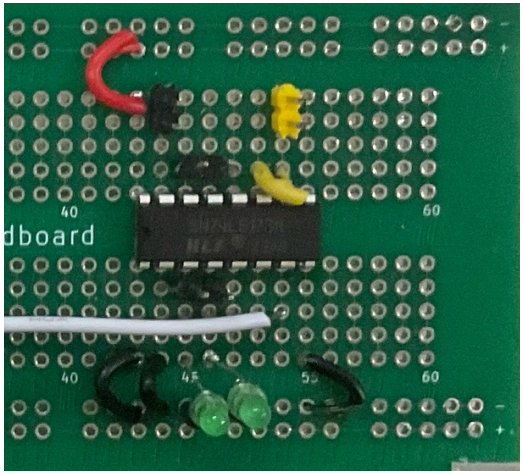


Flags Register

Stores status bits, such as the carry and zero flags, from the ALU's operation.

Type	Quantity
NOT	2
AND	7
OR	2
D flip-flop	2
NOR	4

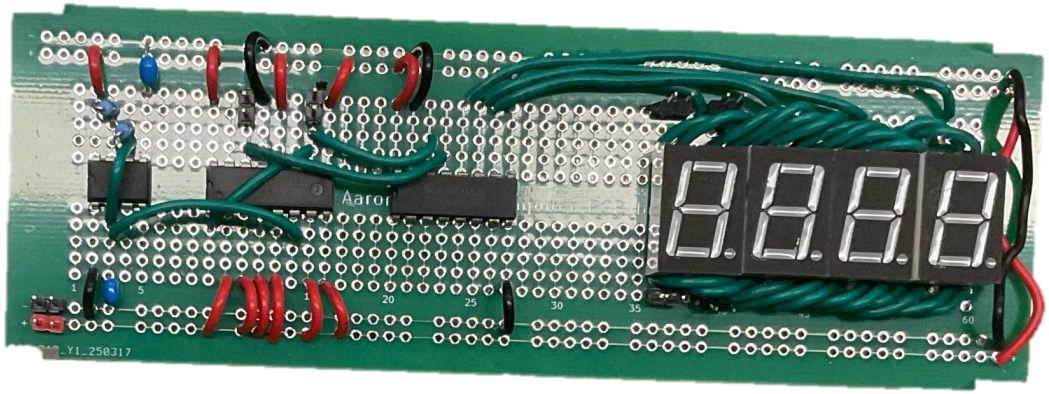


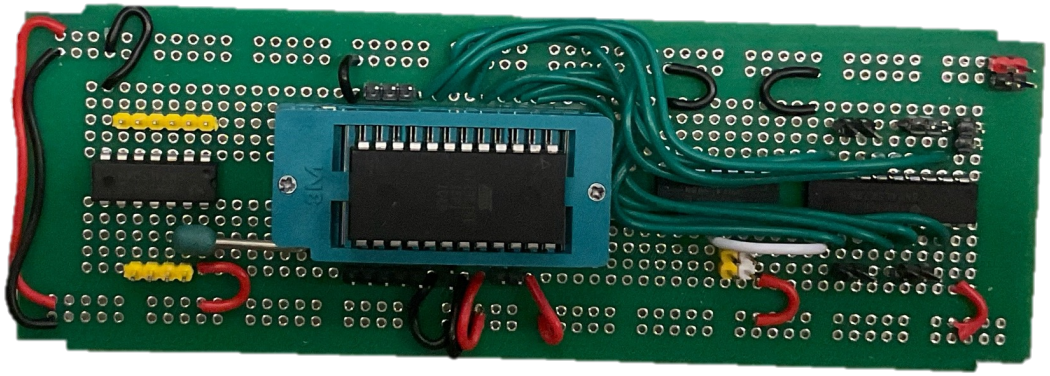


Output

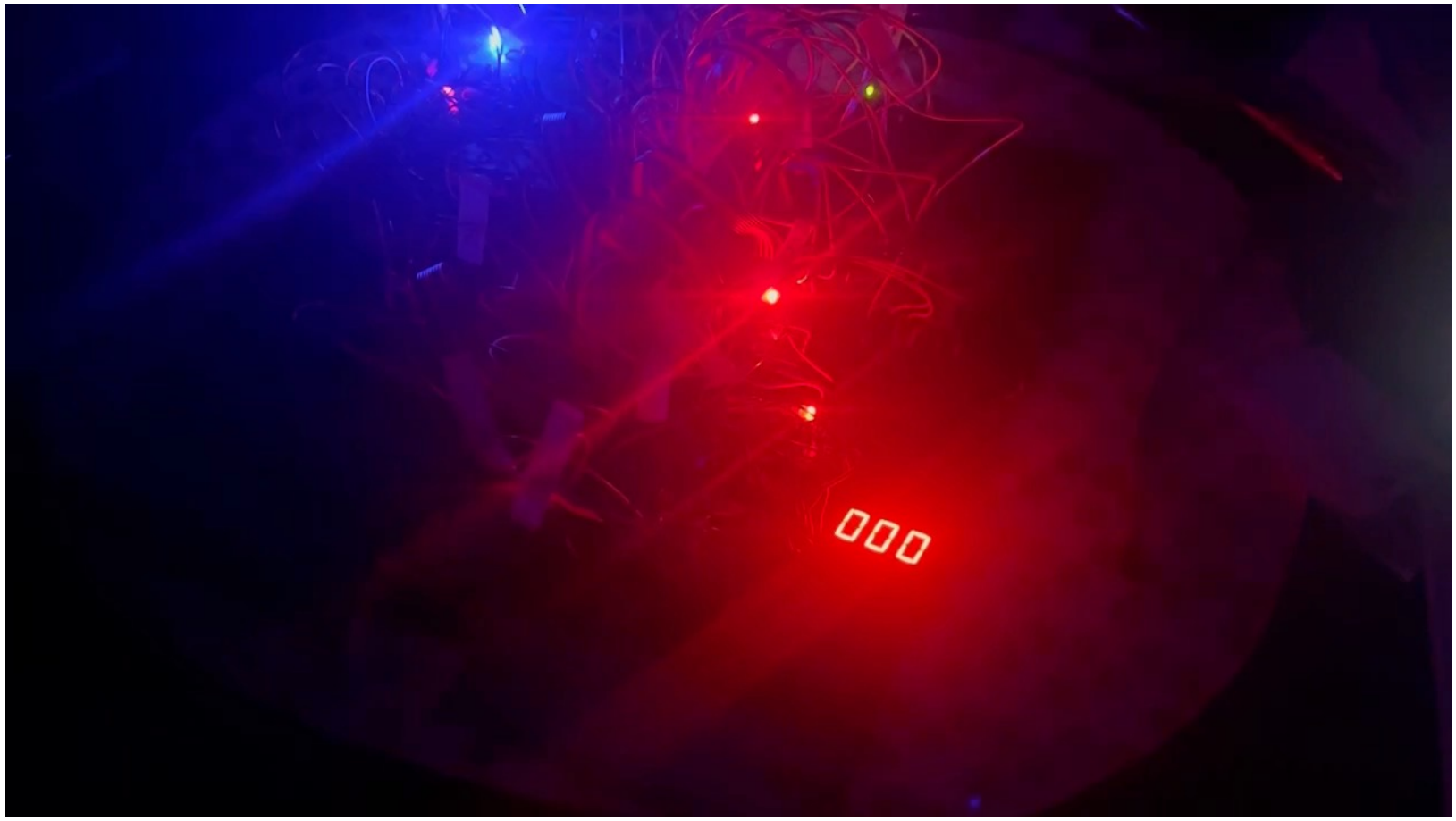
Stores and displays output data.

Type	Quantity
EEPROM	1
NAND	8
NOT	10
JK flip-flop	2
SR Latch	1
comparators	2
D flip-flop	8
AND	1





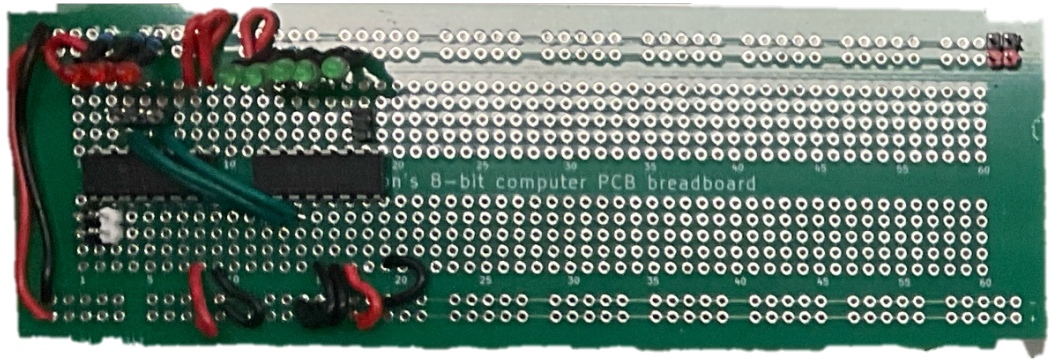
Counting with Output

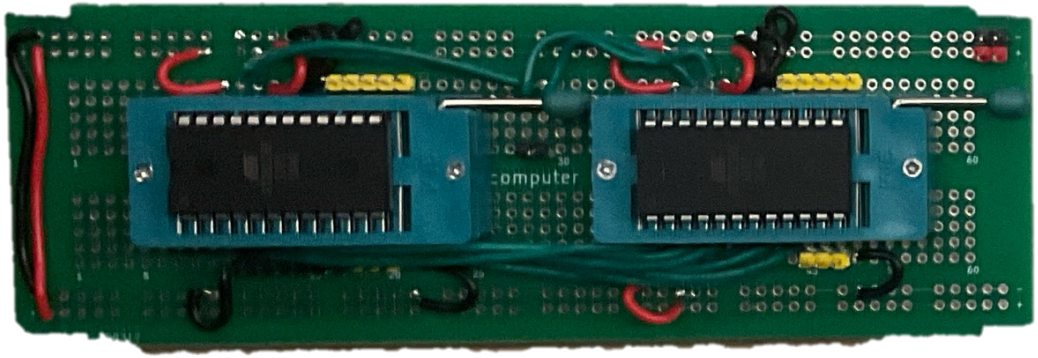


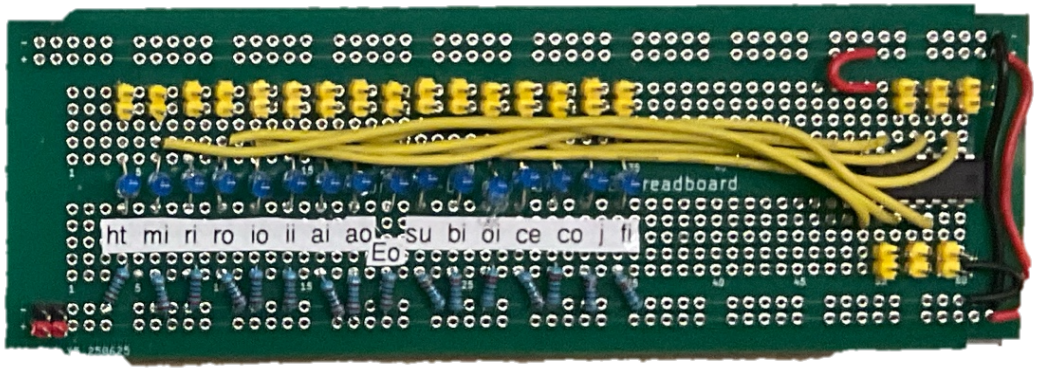
Control logic

Generates the signals that control all modules to perform each step of an instruction.

Type	Quantity
NOT	23
AND	14
NOR	2
NAND	8
JK flip-flop	4
EEPROM	2







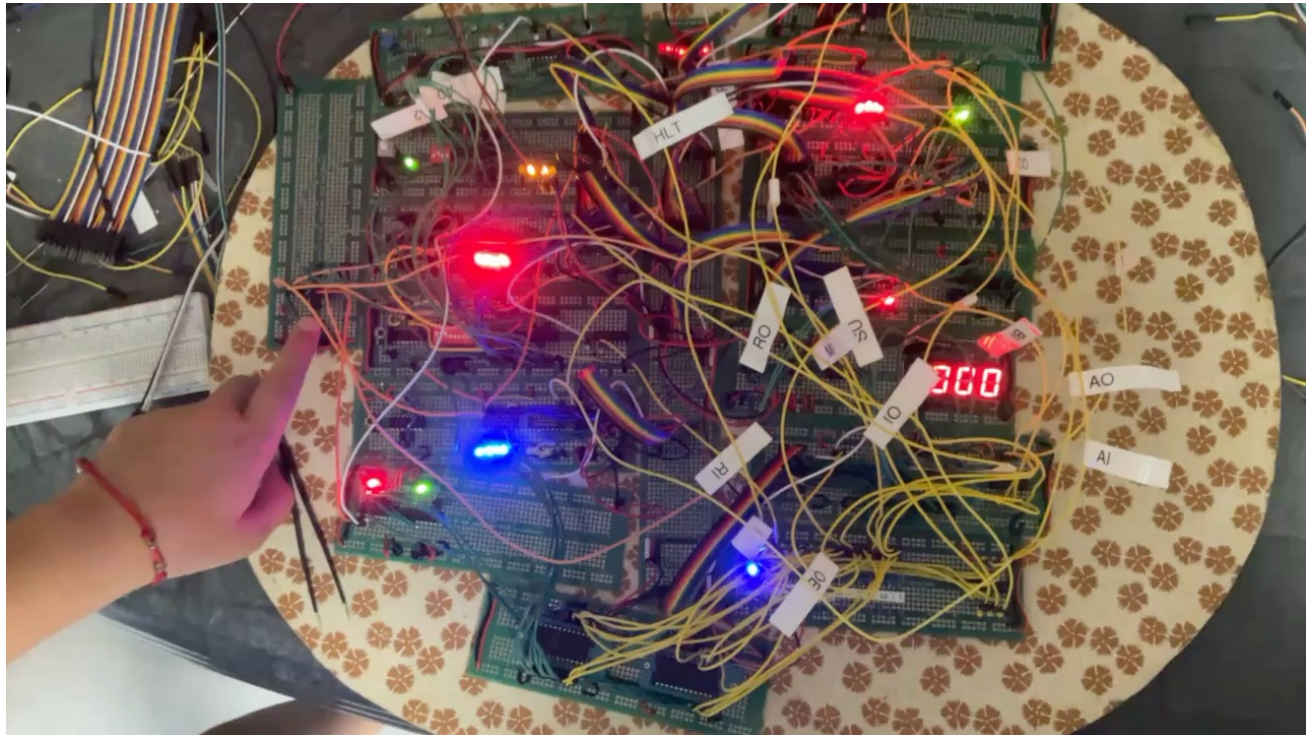
Operations and Microcode

Instruction	Microcode				
	Step 0	**Step 1**	**Step 2**	**Step 3**	**Step 4**
NOP	FETCH #1	FETCH #2			
LDA	FETCH #1	FETCH #2	Instruction Out, Memory Address in	RAM Out, A Register In	
ADD	FETCH #1	FETCH #2	Instruction Out, Memory Address in	RAM Out, B Register In	ALU Out, A Register In, Flags In
SUB	FETCH #1	FETCH #2	Instruction Out, Memory Address in	RAM Out, B Register In	ALU Out, A Register In, ALU Substract, Flags In
STA	FETCH #1	FETCH #2	Instruction Out, Memory Address in	A Register Out, RAM In	
LDI	FETCH #1	FETCH #2	Instruction Out, A Register In		
JMP	FETCH #1	FETCH #2	Instruction Out, Program Counter in		
JC	FETCH #1	FETCH #2			
JZ	FETCH #1	FETCH #2			
OUT	FETCH #1	FETCH #2	A Register Out, Output Register In		
HLT	FETCH #1	FETCH #2	Halt Clock		

FETCH #1: Memory Address Register In, Program Counter Out
FETCH #2: RAM Out, Instruction In, Program Counter Enable

Example Program: countdown and halt

00	LDA	15
01	OUT	
02	SUB	14
03	JZ	6
04	OUT	
05	JMP	2
06	HLT	
…		
14	1	
15	15	



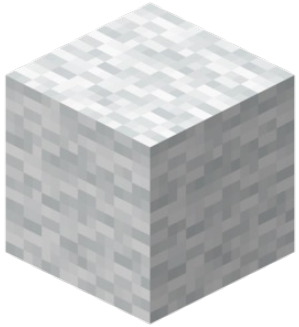
Playing Minecraft

to learn Verilog

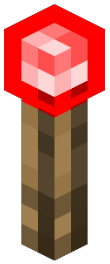
Minecraft electronical components

Lever
Input

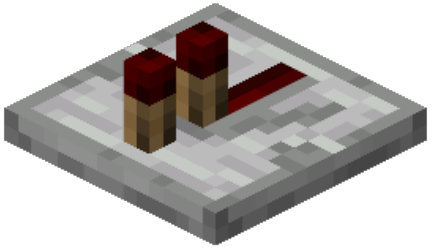
Redstone dust
Wire



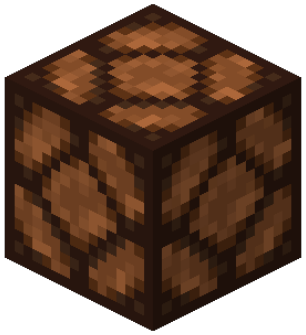
Block
Circuit Shape



Redstone torch
Inverter

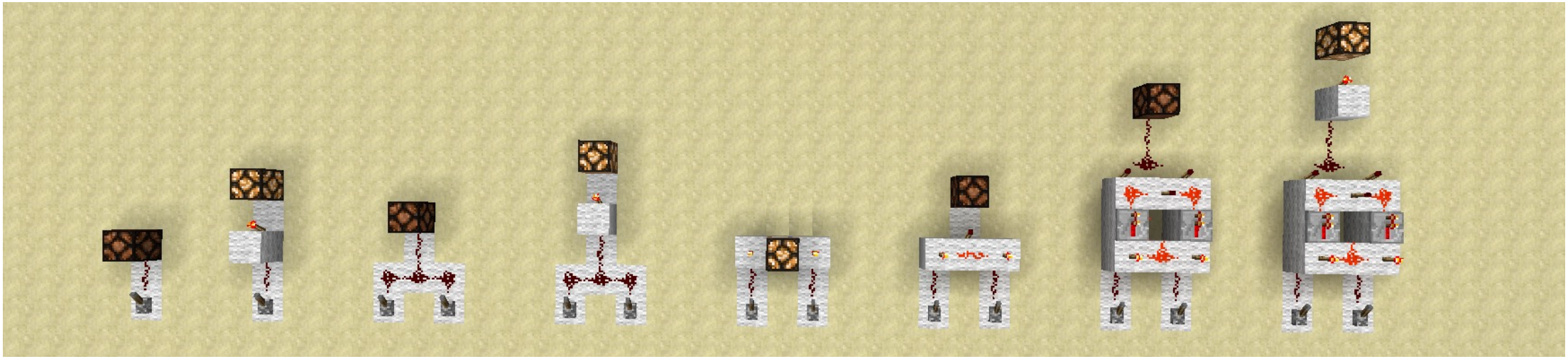


Repeater
Diode, Booster



Redstone lamp
Output

Minecraft Logic Gates



Buffer **Inverter** **OR** **NOR** **NAND** **AND** **XOR** **XNOR**

Minecraft Logic Gates (AND)

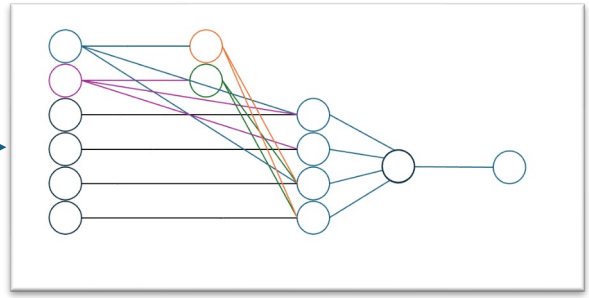
MinecraftHDL

```
module mux (a, b, c, d, s, o);
  input a,b,c,d;
  input [1:0] s;
  output o;
  reg o;

  always @(a or b or c or d or s)
  begin
    if (s == 2'b00)        o  = a;
    else if (s == 2'b01)   o = b;
    else if (s == 2'b10)   o = c;
    else                   o = d;
  end
endmodule
```

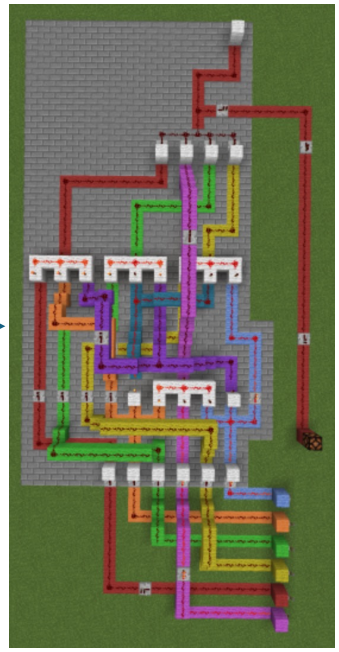
Verilog

Yosys →



Graph Representation

MinecraftHDL →



In-game circuit

Verilog to Minecraft Synthesis

```
module fulladder
(
   input x, y, cin,
   output A, cout
);

   assign {cout, A} = cin + y + x;

endmodule
```

Build it with Verilog

Verilog Design of the ASAP CPU

Module	Verilog LOC
Register A & B	2
ALU	29
Bus	8
RAM	12
Program Counter	9
Instruction Module	18
Flag Register	10
Output Register	91
Control Signals	203

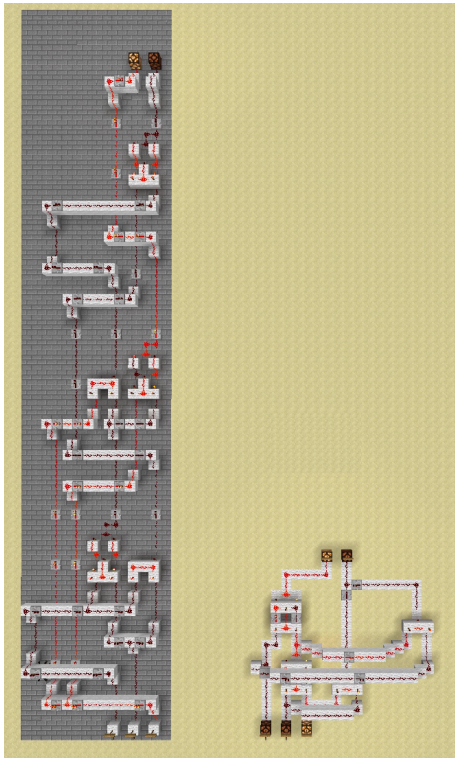
```
mem[0]  = { OP_LDA, 4'b1111 };
mem[1]  = { OP_OUT, 4'b0000 };
mem[2]  = { OP_SUB, 4'b1110 };
mem[3]  = { OP_OUT, 4'b0000 };
mem[4]  = { OP_JZ,  4'b0110 };
mem[5]  = { OP_JMP, 4'b0010 };
mem[6]  = { OP_OUT, 4'b0000 };
mem[7]  = { OP_HLT, 4'b0000 };
mem[8]  = { OP_NOP, 4'b0000 };
mem[9]  = { OP_NOP, 4'b0000 };
mem[10] = { OP_NOP, 4'b0000 };
mem[11] = { OP_NOP, 4'b0000 };
mem[12] = { OP_NOP, 4'b0000 };
mem[13] = { OP_NOP, 4'b0000 };
mem[14] = { 8'b00000001 };
mem[15] = { 8'b00001111 };
```

Verilog Design for Minecraft

Limitations of the MinecraftHDL environment:

- circuits are larger than necessary
- high delay
- only combinational logic

One Bit Full Adder

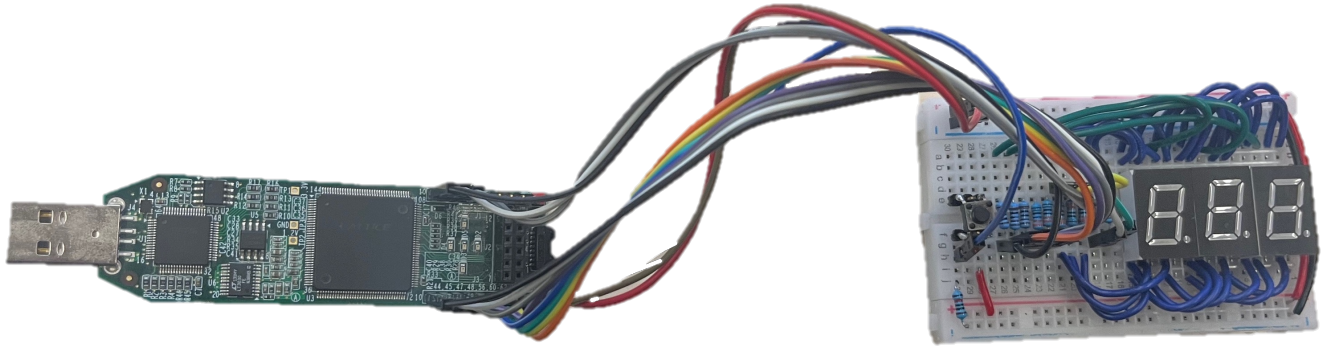


Verilog Manual

Verilog Design on FPGA

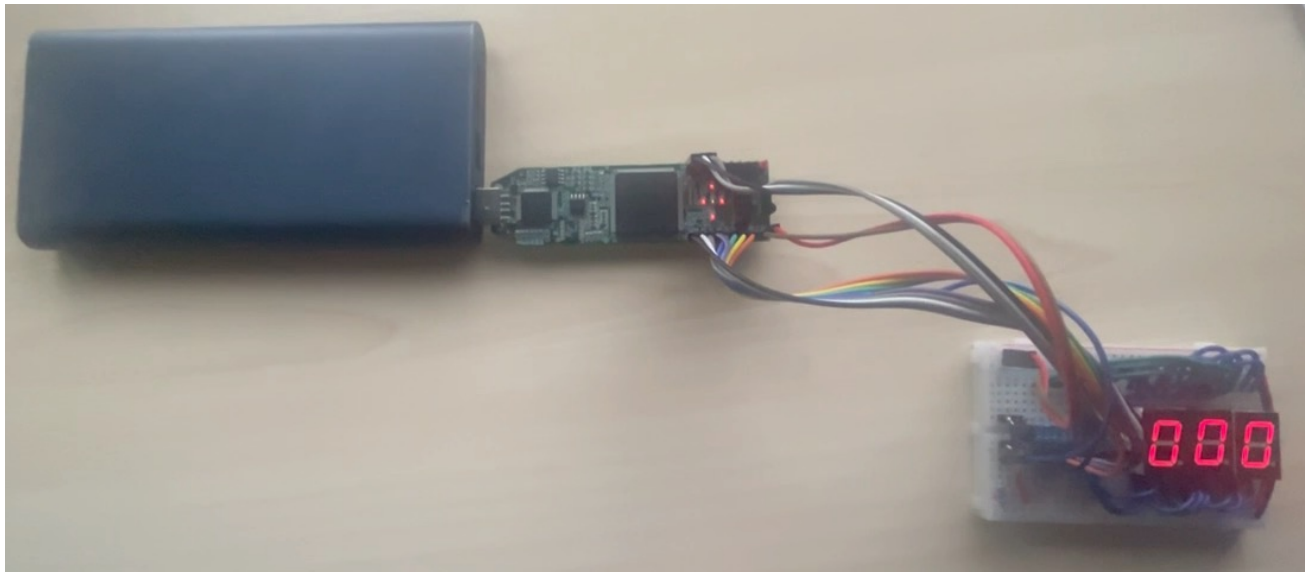
Usage of:
- Lattice FPGA
- apio tooling to make synthesis and flashing simple

Attach external output, everything else (including code) fixed inside of FPGA.



Example Program: Fibonacci sequence

00	LDA	15
01	ADD	14
02	STA	15
03	JC	11
04	OUT	
05	LDA	14
06	ADD	15
07	STA	14
08	JC	11
09	OUT	
10	JMP	0
11	HLT	
...		
14	1	
15	0	

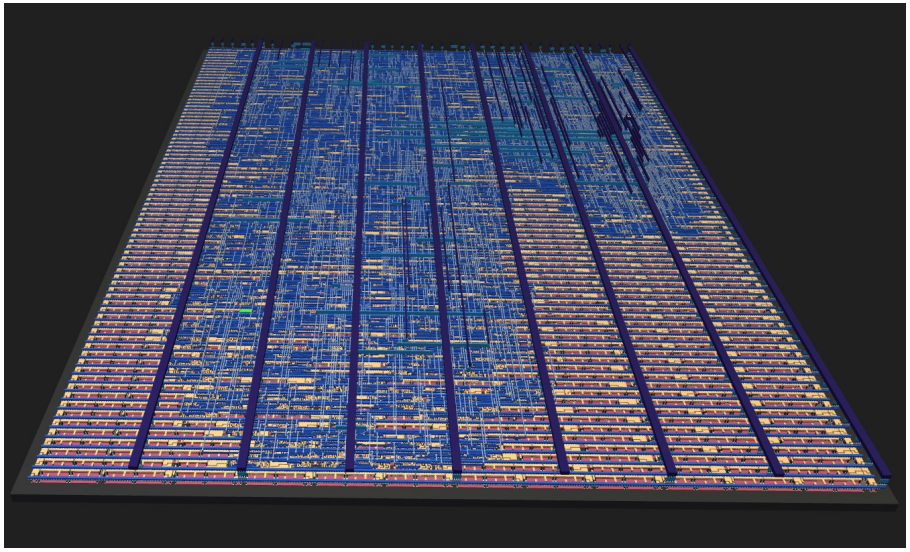


Verilog Design on ASIC

TinyTapeout 7 already has Tiny Eater but I could not make it work.

Next Goal:
SKY25b shuttle closes in 10 days

Wish me luck! 🥰



Thank you for listening.
Any Questions?

References

- **Ben Eater YouTube Channel**
  https://www.youtube.com/@BenEater
- **MinecraftHDL**
  https://github.com/itsfrank/MinecraftHDL
- **apio**
  https://github.com/FPGAwars/apio
- **SAP-1 Design**
  Digital Computer Electronics by A. P. Malvino and J. A. Brown
- **Austin Morlan's FPGA work**
  https://austinmorlan.com/posts/8bit_breadboard_fpga/
- **TinyTapeout**
  https://tinytapeout.com
- **Jason Kaufmann's Tiny Eater 8 Bit**
  https://github.com/jasonkaufmann/tt07-beneater8bit