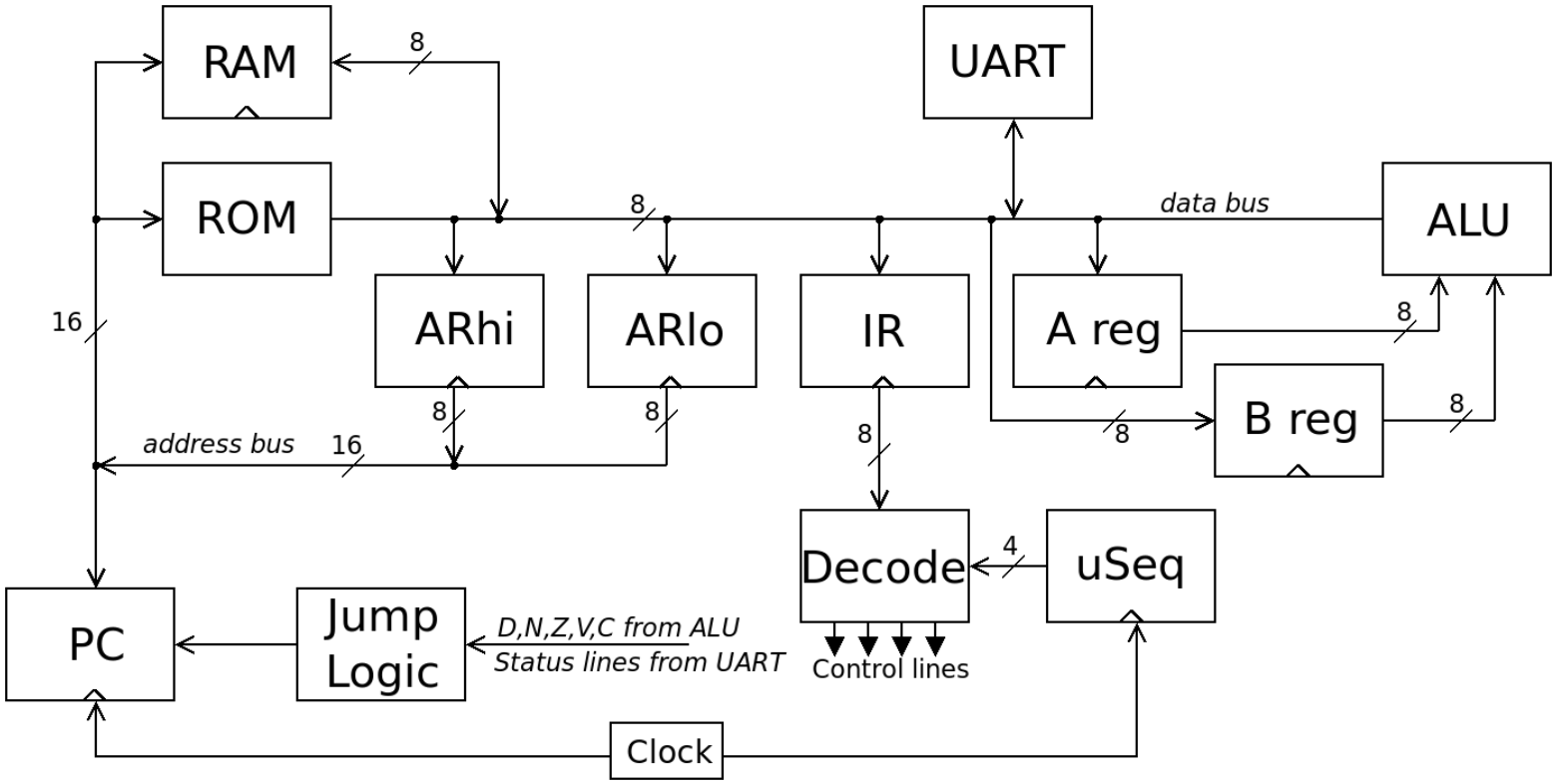


These are the schematics for the CSCvon8 CPU.
See https://github.com/DoctorWkt/CSCvon8/ for details.

Warren Toomey

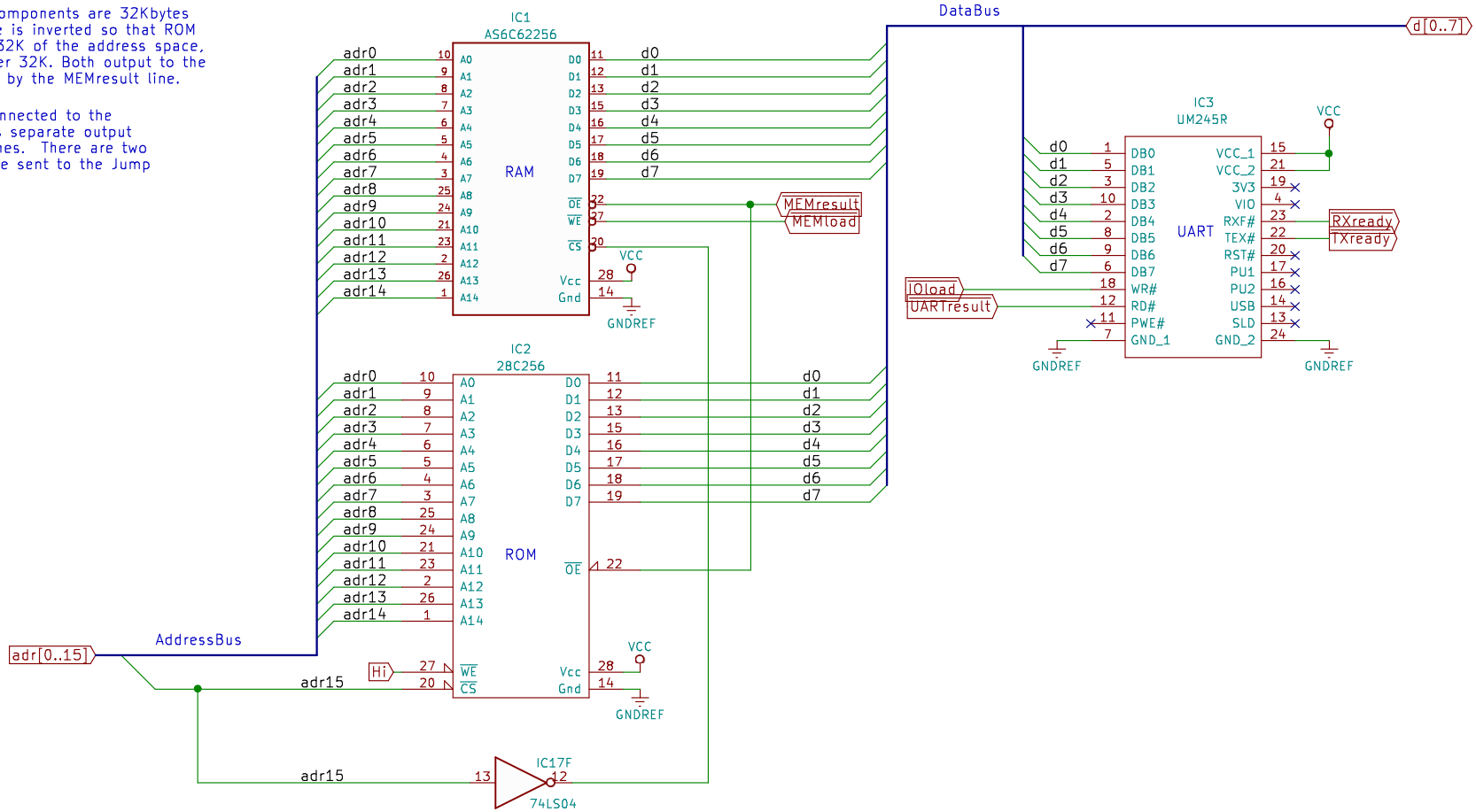
Sheet: /
File: Schematic.sch

Title: CSCvon8 CPU

Size: A4	Date:	**Rev: 1.2**
KiCad E.D.A. kicad 5.1.2-f72e74a84ubuntu18.04.1		Id: 1/6

The RAM and ROM components are 32Kbytes each. The adr15 line is inverted so that ROM occupies the lower 32K of the address space, and RAM in the upper 32K. Both output to the data bus, controlled by the MEMresult line.

The UART is also connected to the data bus, and it has separate output and input control lines. There are two ready lines which are sent to the Jump logic.



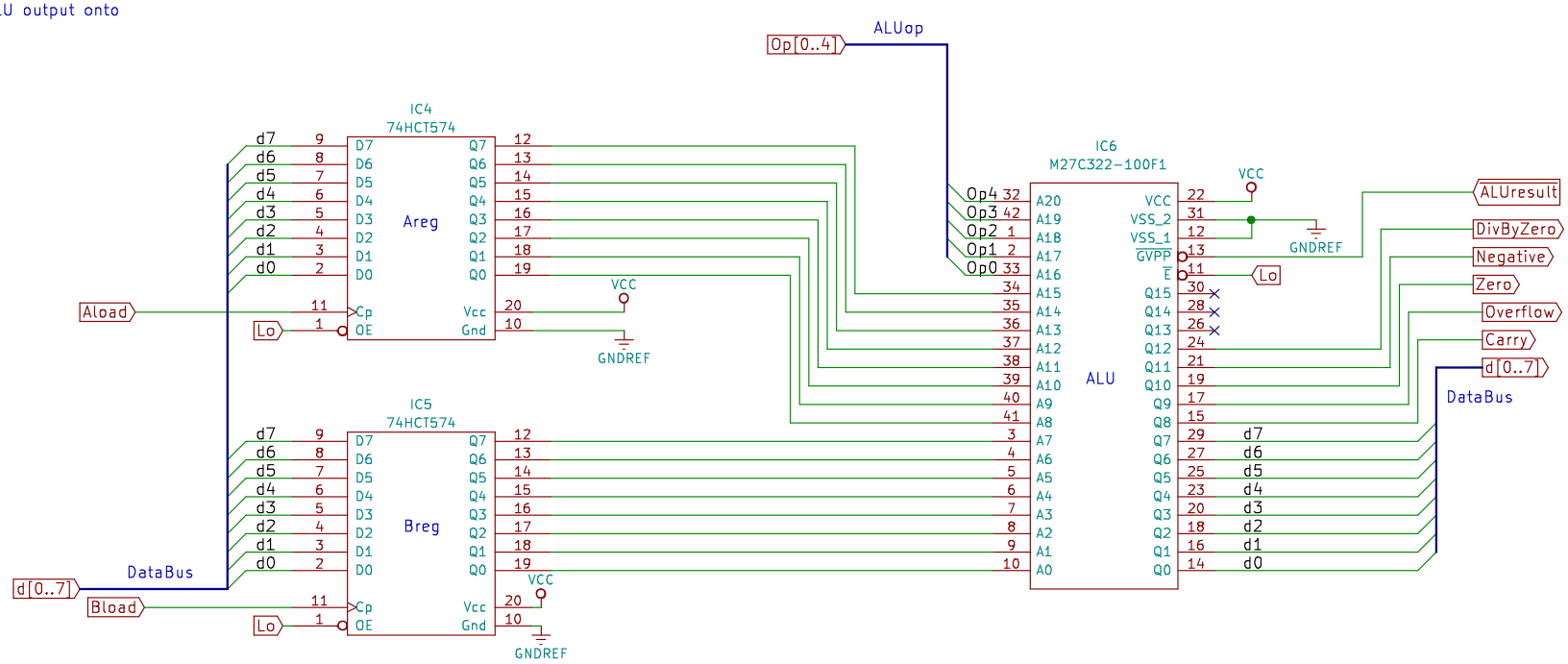
Sheet: /Memory and I/O/		
File: MemoryIO.sch		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. kicad 5.1.2-f72e74a84ubuntu18.04.1		Id: 2/6

The A and B registers can load from the data bus. Their outputs are directly connected to the ALU.

The ALU also receives the 5-bit ALU operation from the instruction decoder logic.

The ALU outputs the 8-bit result plus five flags that describe the type of output.

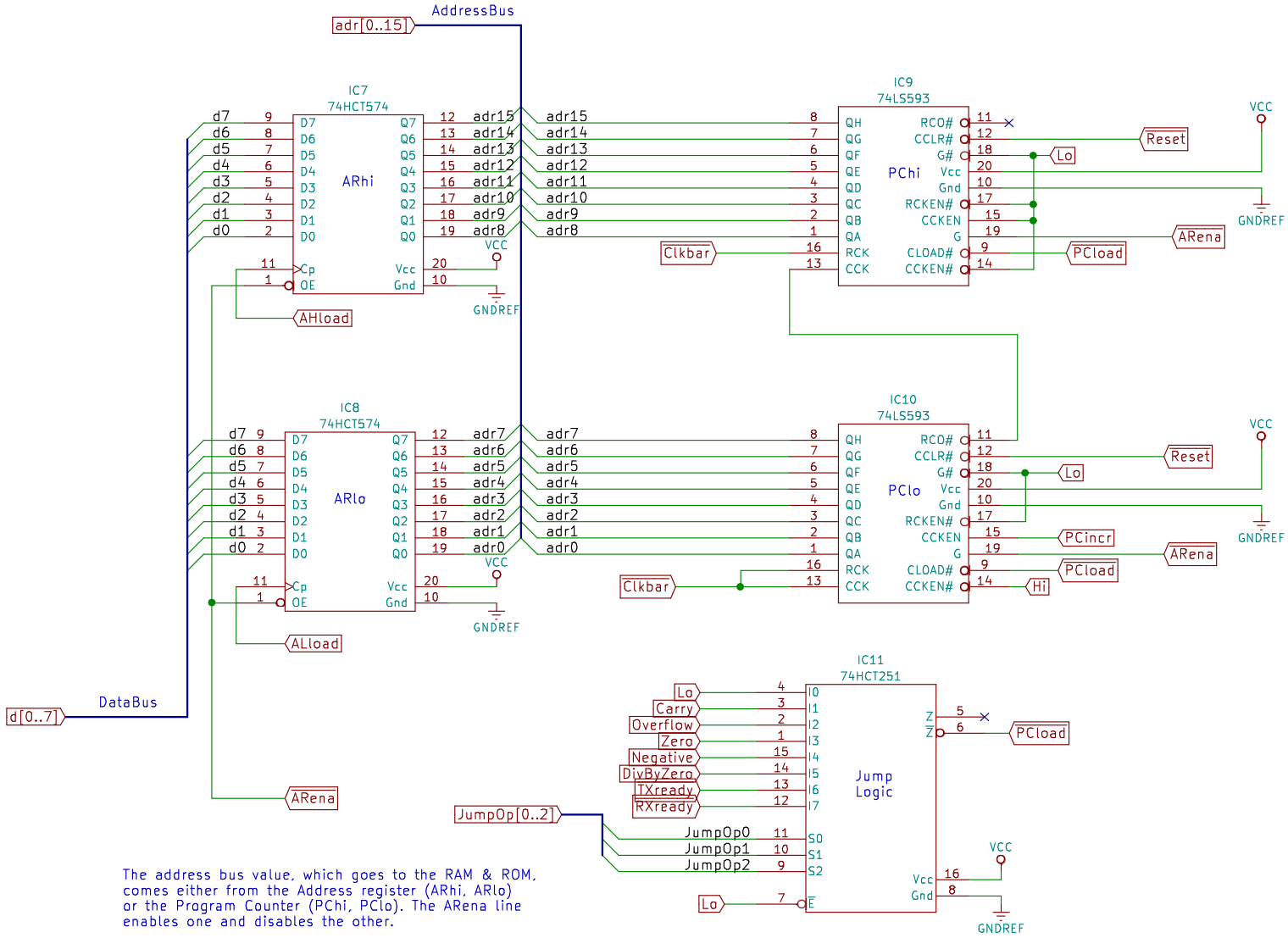
The ALUresult control line enables the ALU output onto the data bus.



Sheet: /ALU and Data Registers/
File: ALU_DataRegs.sch

Title:

Size: A4	Date:	**Rev:**
KiCad E.D.A. kicad 5.1.2-f72e74a84ubuntu18.04.1		Id: 3/6



The address bus value, which goes to the RAM & ROM, comes either from the Address register (ARhi, ARlo) or the Program Counter (PChi, PClo). The ARena line enables one and disables the other.

The address register loads from the databus, using the AHload and ALload control lines.

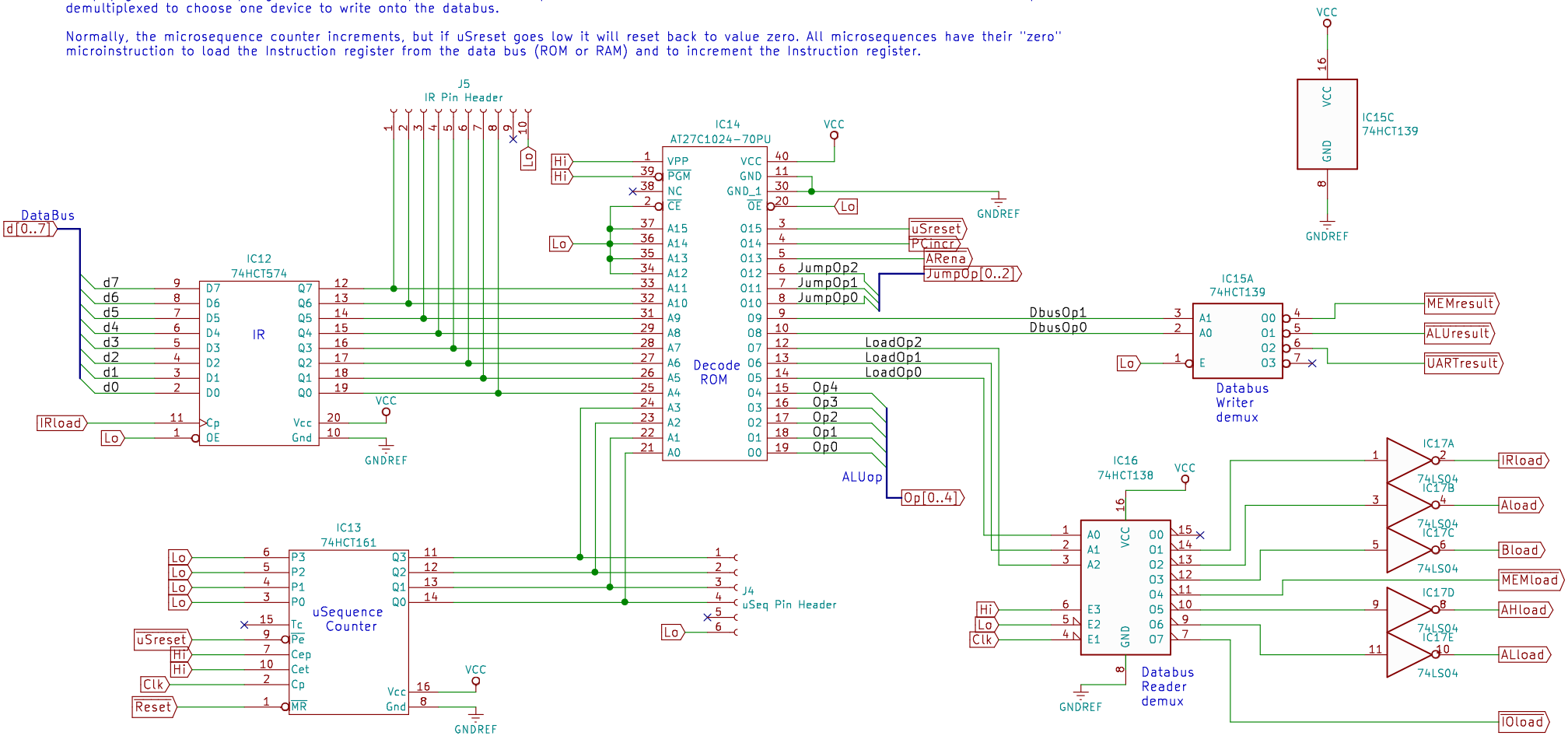
The PC can keep its current value, increment on PCincr or load when the /Z line (pin 6) from the Jump logic goes low. On a jump, the PC loads from the Address register.

There are eight jump options: never (Lo), five ALU flag results and two UART status results. The JumpOp control lines choose how to jump.

Sheet: /Addressing/		
File: Addressing.sch		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. kicad 5.1.2-f72e74a84ubuntu18.04.1		Id: 4/6

CSCvon8 is a microsequenced CPU. The Instruction register, combined with the microsequence counter, selects the next microinstruction to perform. This is "looked up" in the Decode ROM, which produces the sixteen control lines for the specific microinstruction. The ALUop goes to the ALU. The JumpOp goes to the Jump logic. The three LoadOp lines are demultiplexed to choose one device to load from the databus. The two DbusOp lines are demultiplexed to choose one device to write onto the databus.

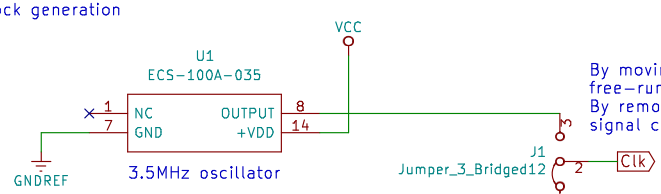
Normally, the microsequence counter increments, but if uSreset goes low it will reset back to value zero. All microsequences have their "zero" microinstruction to load the Instruction register from the data bus (ROM or RAM) and to increment the Instruction register.



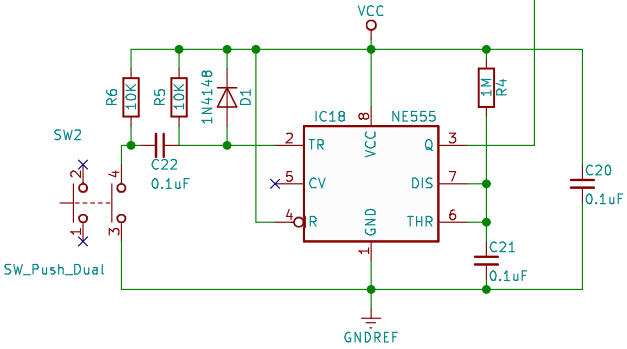
It's so annoying that the 74HCT574 has an active high load line, and the RAM and UART have active low load lines. One extra chip just to deal with that, sigh.

Sheet: /Instruction Decode/		
File: IR_Decode.sch		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. kicad 5.1.2-f72e74a84ubuntu18.04.1		Id: 5/6

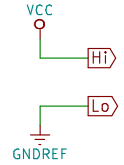
Clock generation



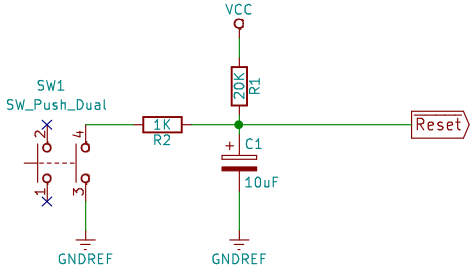
By moving the jumper, the CPU can operate in free-running mode or in single-step mode. By removing the jumper, an external clock signal can be applied to jumper pin 2.



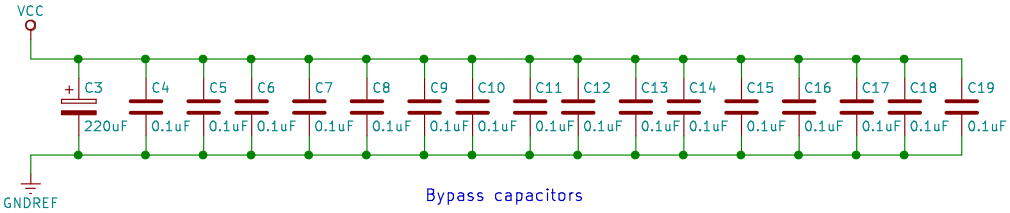
555 circuit from
https://electronics.stackexchange.com/questions/180716/555-timer-one-shot-trigger



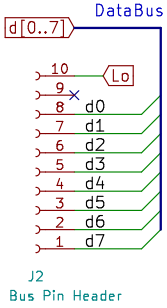
Logic levels



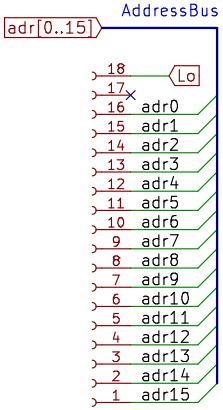
Reset generation



Bypass capacitors

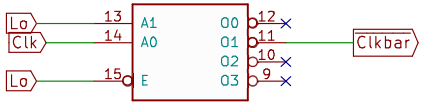


J2
Data Bus Pin Header



J3
Address Bus Pin Header

IC15B
74HCT139



Clkbar is an inverted Clk signal, used to initiate events on the falling edge of Clk. We use the spare 139 half as an inverter.

Sheet: /Analog/ File: Analog.sch		
Title:		
Size: A4	Date:	**Rev:**
KiCad E.D.A. kicad 5.1.2-f72e74a84ubuntu18.04.1		Id: 6/6