

# Cool Pookie Unit (CPU)

---

Created By:

James Schorle and Lucas Vanzelli

Version 1.0

December 10th, 2024

I pledge my honor that I have abided by the Stevens Honor System.

Welcome to the Cool Pookie Unit... read further if you dare.....



# Table of Contents

I.	Breakdown of Responsibilities	pg. 3
II.	How to use the Assembler	pg. 4-5
III.	Architecture Description	pg. 5-10
IV.	Instruction Format	pg. 11-12

## **I. Breakdown of Responsibilities**

### **1.1. The Responsibilities of Sir James Schorle**

Sir James Schorle is a second-year Computer Science student at Stevens Institute of Technology. His favorite animal is a whale. He loves to dance. I forgot this was a breakdown of responsibilities and not a biography. In the Cool Pookie Unit, Sir James Schorle, son of Mr. Logisim, ingeniously spearheaded the wiring protocol of the CPU, invented the system for the Control Unit, developed the storing and loading features, and properly connected the wiring of every component in a way Steve Jobs would shed a tear to. Additionally, James assisted in debugging Sir Lucas Vanzelli's Register File, ALU, and Python Assembler.

### **1.2. The Responsibilities of Sir Lucas Vanzelli**

My money don't jiggle, jiggle, it folds... oh wait... I forgot I wasn't at karaoke. In the Cool Pookie Unit, Sir Lucas Vanzelli (second (maybe third?)-year CS student also at SIT) invented the ALU and collaborated with Sir James Schorle to create the CPU. He aided in the Control Unit's debugging process and expertly developed and wrote almost everything in the Python Assembler Program for the Cunita Pookie Unit. Sir Lucas Vanzelli also dedicated precious hours of his time to brainstorming the project as a whole, in and out of office hours. Lastly, Sir Lucas Vanzelli wrote the crankypants.pookie program to test the CPU.

## II. How to Use the Assembler

### Assembler 101

The Assembler is written in Python, in a file called “assembler.py.” It decodes instructions to a hexadecimal representation compatible with the CPU (Cool Pookie Unit). To use the assembler, a user must include a piece of sample code to compile. The assembler takes an input file called “crankypants.pookie” to accomplish this. Reference the image below.

```
SNATCH_IMM X1, 1
LOG X1, 0x00
SNATCH_IMM X2, 2
KISS X1, X1, X2
SNATCH_ADR X3, 0x00
SNATCH_IMM X4, 6
DIVORCE X4, X4, X3
KISS X1, X1, X2
LOG X4, 0x08
```

This is a sample input file for the CPU.

```
ubuntu@ubuntu:~/shared/cs382/project2$ python3 assembler.py
SNATCH_IMM X1, 1
0804
LOG X1, 0x00
1800
SNATCH_IMM X2, 2
0809
KISS X1, X1, X2
0404
SNATCH_ADR X3, 0x00
1002
SNATCH_IMM X4, 6
081B
DIVORCE X4, X4, X3
0C3B
KISS X1, X1, X2
0404
LOG X4, 0x08
1823
```

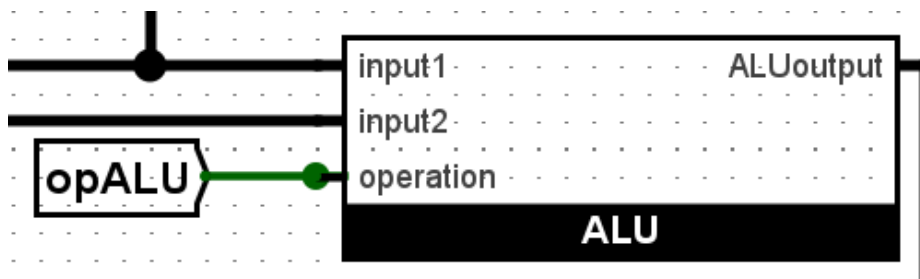
To use this assembler in the Virtual Machine, one must use the command “python3 assembler.py.” This will print out the instructions of the sample code and their corresponding hexadecimal values that are compatible with CPU.

The assembler also generates an image file named “poopypants.pookieout” that holds the hexadecimal value of the instructions. This image file can be loaded into the Instruction Memory to be processed by the CPU, and eventually stored into the Data Memory. To store values into the data memory, a user must use the instruction

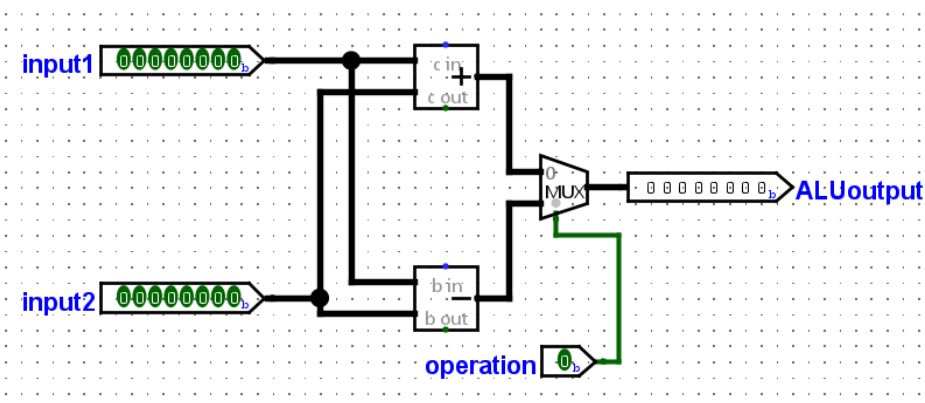
LOG [register], [address] **NOTE: Each address is 8 bits**

### III. Architecture Description

ALU



The ALU component  
in the CPU.

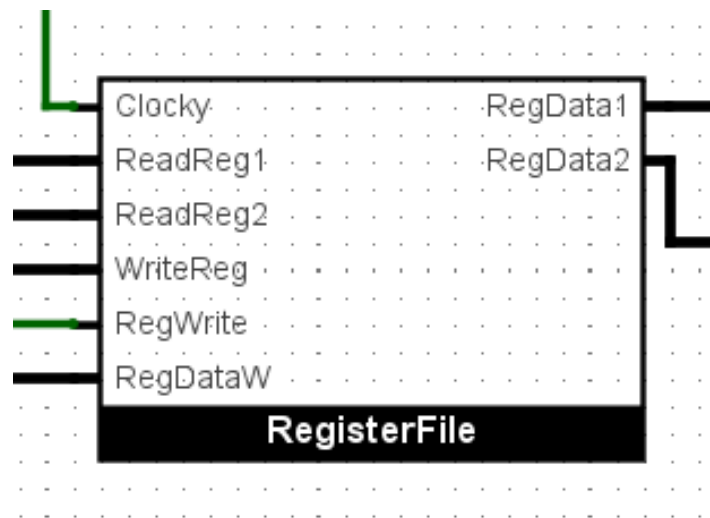


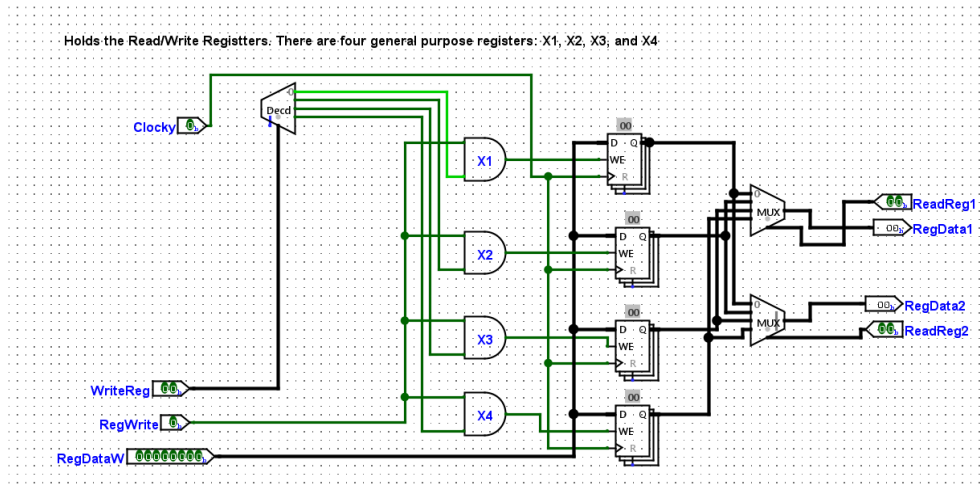
The entire ALU  
component.

CHANGE

The ALU stands for “Arithmetic Logic Unit” and this is the component of the CPU that processes what arithmetic instruction will be used. The ALU receives two inputs—input1 and input2— which then will be added/subtracted. The operation performed is determined by the hexadecimal translated from an instruction. The Control Unit has a tunnel named “opALU” and “ALUsed” that corresponds with a bit in an instruction. The bit that corresponds with ALUsed will only ever be 1 if the instruction being processed is either addition or subtraction. opALU will only receive a signal if the ALUsed bit is 1. The bit that corresponds with opALU will be 0 if the arithmetic is addition, or 1 if the arithmetic operation is subtraction. Consequently, this signal from opALU will feed into a multiplexor in the ALU to let the CPU know if it should perform addition or subtraction.

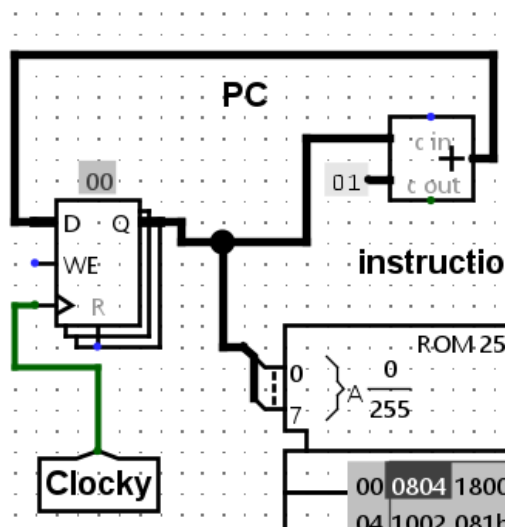
## Register File





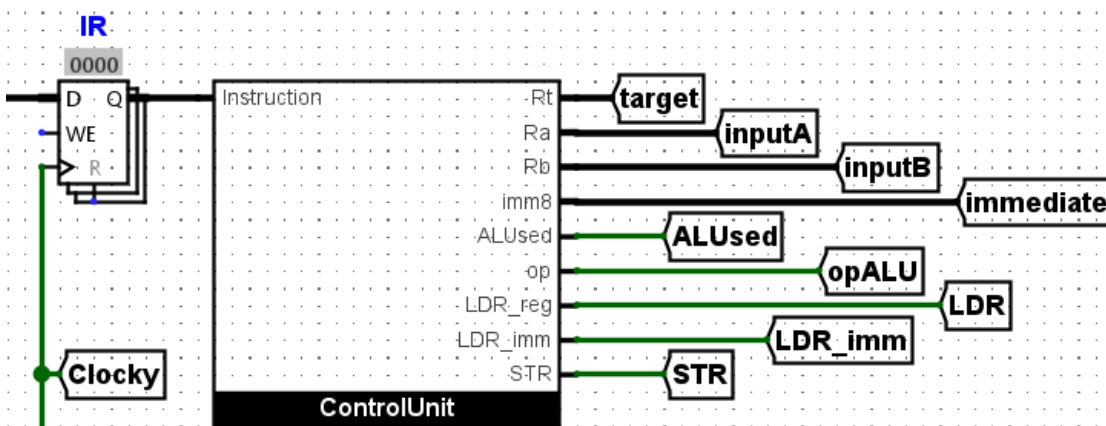
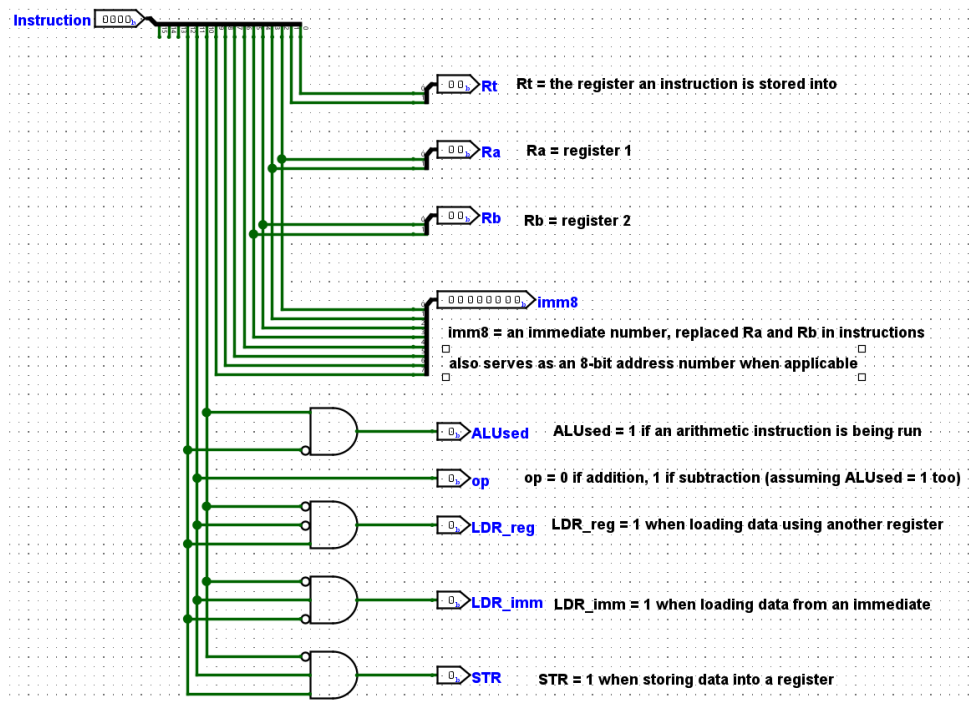
The Register File stores the general-purpose registers in the CPU. There are four general-purpose register files, X1, X2, X3, and X4 that store intermittent instruction data that is not (yet) stored into memory.

PC



The PC, aka “Program Counter”, is how the Cool Pookie Unit keeps track of the next instruction in the program. Every time the clock circuit activates, the program counter is incremented by one.

## Control Unit

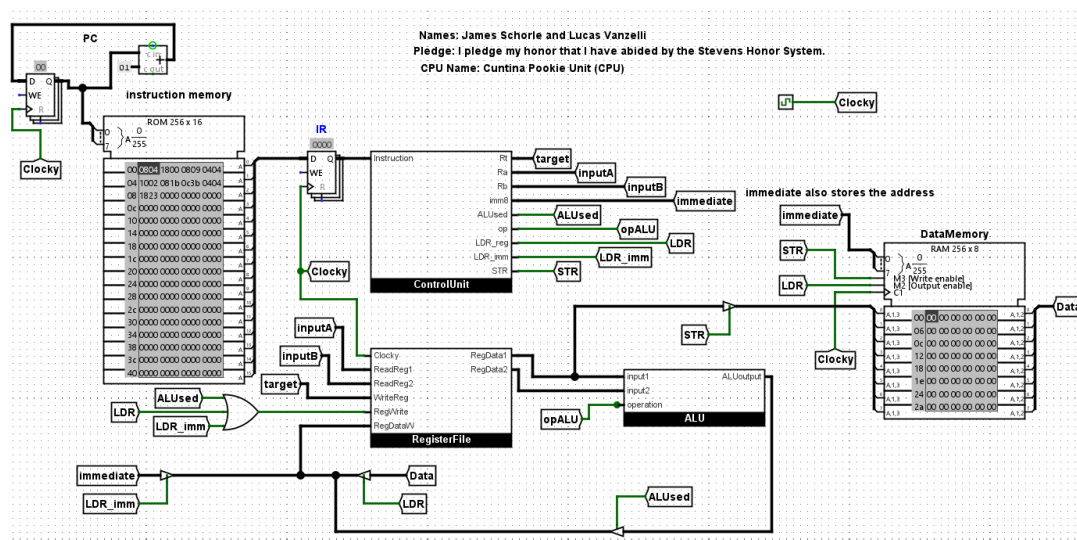


The Control Unit, put bluntly, is a big combinational logic circuit that translates a written instruction in the assembler language into the language our CPU can understand. More specifically, though, the Control Unit takes in a 16-bit hexadecimal instruction and breaks it apart piece by piece into several individual pins, along with its corresponding registers and immediate/address values. The Control Unit breaks apart the hexadecimal into its binary form to



differentiate between instructions. Aside from the first three digits of this instruction which are left circuitless for conciseness, the next three digits (which we consider the most-significant bits) are what is called the operation code, determining which instruction will be used. The first of these digits determines if data within a register will be accessed; the second determines whether addition or subtraction will undergo (if applicable); and the third digit is officially known as ALUsed, which, when active, indicates that an arithmetic operation is happening. The remaining ten digits are used to store any and all registers, 8-bit immediates, or 8-bit addresses. Refer to the instruction breakdown below on how this is possible along with the circuiting behind it.

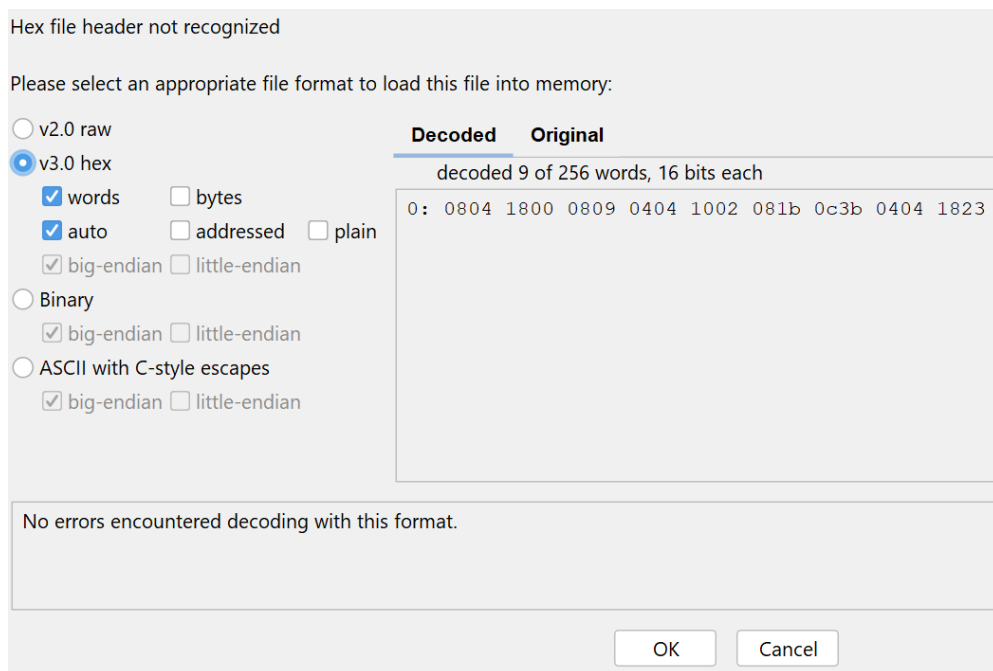
## Entire CPU



The Cool Pookie Unit uses the components above in combination to function. This allows the CPU to process and execute the KISS (ADD), DIVORCE (SUB), LOG(STR), SNATCH\_IMM (LDR\_IMM), and SNATCH\_ADR (LDR\_ADR) instructions. Sir James Schorle and Lucas Vanzelli used numerous tunnels in the Cool Pookie Unit to keep the unit organized and easily

digestible. This is also the reason why the creators used separate circuits to create the CPU, ALU, Control Unit, and Register File.


When loading a set of instructions into the CPU, the user must generate the “poopypants.pookieout” file from the assembler and load it into the Instruction Memory. Once loaded, make sure that the file format 3.0. hex is selected. Then, the user can press “OK” and run the simulation.



## IV. Instruction Format

### NOTE:

All register fields require two bits, and all immediates or addresses require eight bits. The Operator Code (or opcode for short) requires three bits, for reasons explained within the Control Unit section. Additionally, to keep things clean and neat, we added an extra three empty bits to the front of every instruction so that its total bit number would add up to sixteen.

 <- How the bits of each instruction are delegated.

### INSTRUCTIONS:

**KISS Rt, Ra, Rb**                      000|001|0000|Ra|Rb|Rt

KISS joins the values stored within two registers Ra and Rb and stores their sum in Rt, just as a kiss joins together two people.

**DIVORCE Rt, Ra, Rb**                      000|011|0000|Ra|Rb|Rt

DIVORCE, unlike KISS, does not join two people together; it tears them apart, subtracting the value stored within Rb from Ra, storing the result in Rt. When Ra and Rb fall through, Rt is left with the remains. :(

**LOG Rt, addr8**                      000|110|addr8|Rt

LOG acts as a captain's log to a pirate after a great plunder, yarghhh!!! This instruction saves the stored value of Rt into memory at address addr8, just as a pirate would bury treasure where X marks the spot.

**SNATCH\_IMM Rt, imm8**      000|010|imm8|Rt

SNATCH\_IMM snatches the 8-bit immediate number imm8 and saves it into Rt, which can then be used later.

**SNATCH\_ADR Rt, addr8**      000|100|addr8|Rt

SNATCH\_ADR snatches the data in memory at addr8 and subsequently stores it into Rt for it to potentially use later.