Name of the CPU: **Apex(Legends) Processing Unit**
Joshua Bekkerman Tristan Mikiewicz
“I pledge my honor that I have abided by the Stevens Honor System”

Work done by Joshua Bekkerman:

Joshua designed the instruction language and was responsible for translating the language into an image file (containing only hexadecimals and addresses). This was done in python.

Work done by Tristan Mikiewicz:

Tristan's main contribution was in designing the CPU using Logisim. This included architecture planning, circuit layout, and ensuring functionality alignment with the project's objectives.

• How to use your assembler program:

We have assembler.py, a python program that assembles our newly created assembly language. We put this program in the same folder as a .txt file, this .txt file contains our assembly program(written in our new language). Then we run the assembler.py program. This program outputs a .txt file containing the hexadecimal encodings. In order to load the image file into the cpu, we right click on the Instruction Memory and click load image.

• The architecture description of your CPU, e.g., how many general purpose registers and how can we refer them in an assembly program, what functions your CPU can do, etc;

Our cpu has 8 general purpose registers (r0, r1, r2, r3, r4, r5, r6, r7). Each register is 8 bits, so we can hold numbers up to 255. Our cpu can add and subtract, load memory into registers and store register data into memory.

• For each instruction implemented, write the general format (such as ADD Rm,Rn,Rt), and its binary encoding. You can follow a similar description as in Chapter 3.2 in textbook. When describing binary encoding, you need to specify why you need this number of bits for that field.

<dest> add <arg1> <arg2>
<dest> sub <arg1> <arg2>
<dest> load <adr>
<dest> store <adr>

Our binary encodings are 32 bits.

Bits	31	28-30	27	26-24	18	17-15	9	2	1			
Signal	write/ read	Dest. reg	imm1	arg1	imm2	arg2	operation	memR	memW			

(19-26 for number 1) (17-10 for number 2)