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

CUPPY THE CPU

Overview
Cuppy the CPU takes the instructions ADD, SUB, or LDR to add, subtract, and load with three operands, then performs actions accordingly.

Allowed Instructions:
These actions are add, sub, and load.
ADD destination register *(space)* 1st register *(space)* second register
ADD Rm Rn Rt
SUB destination register *(space)* 1st register *(space)* subtracting register
SUB Rm Rn Rt
LDR destination register *(space)* 1st register *(space)* offset
LDR Rm Rn Rt

ADD will add the two operand registers together and store their sum in the destination register. SUB will subtract the second register from the first one, then store their difference in the destination register. LDR will load the value stored in the first operand register to the destination register.

Behind the Scenes:
The assembler takes an instruction file and converts each instruction to a binary opcode, which is then converted to a hexadecimal value and written to an image file. The 8 bit opcode has the instruction ID in the first two bits. "00" is the first instruction, which is addition, "ADD". "01" is the second instruction option, which is subtraction, "SUB". Finally "LDR" is the load instruction, which loads the value in the first operand register to the destination one.

The opcode looks like this:

Binary	XX	XX	XX	XX
Instruction	instruction ID	destination register	operand I	operand II

Since 2 bits can represent the values 0-4 and we only have 4 registers and 3 instructions, Cuppy only requires 2 bits for each segment of the instruction.

Specifically broken down these are the opcode options

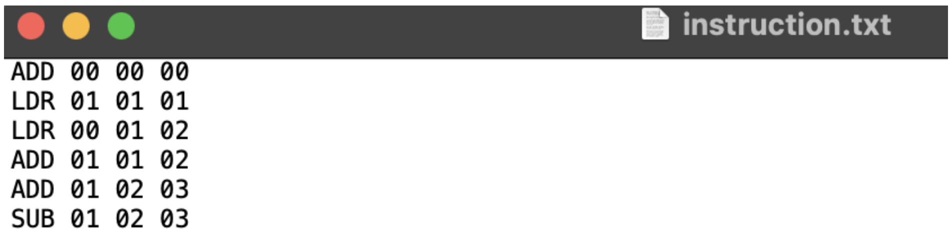
Bits 0-1	Instruction	Bits 2-3	Destination Register	Bits 3-4	1st Operand	Bits 5-6	Second Operand
00	ADD	00	Reg 0	00	Reg 0	00	Reg 0
01	SUB	01	Reg 1	01	Reg 1	01	Reg 1
10	LDR	10	Reg 2	10	Reg 2	10	Reg 2
		11	Reg 3	11	Reg 3	11	Reg 3

Architecture:
This CPU uses 4 general purpose registers that are 2-bit addressed. They are addressed as "00", "01", "10", and "11." The assembler reads each instruction from an instruction file and converts it to a binary opcode based on the instruction given and the registers it pertains to. After assembling this binary code, it converts each instruction to a hexadecimal value which is written to the image file. This way the instruction memory can store each instruction as a short hexadecimal value.

How to Use:
In order for this program to function the CPU (.circ file), assembler (.py file), and instruction (.txt file) must all be stored in the same directory. Write the instructions in the syntax given above, making sure it's exact since the assembler does not have an error checking function. Right click the IM, Instruction Memory (RAM), select image.txt, and load it as a v3.0 hex file. Then add values to the registers in Data Memory in the registers that you are using instructions with. After that it is ready to run! Click the clocks or enable auto-click to watch the values fill in according to the instructions!

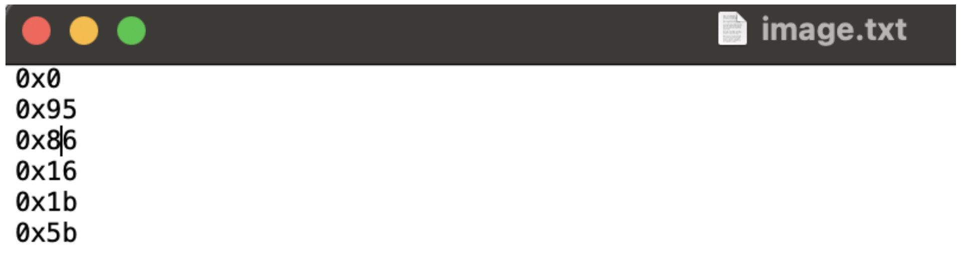
Example:

1. Create an instruction set



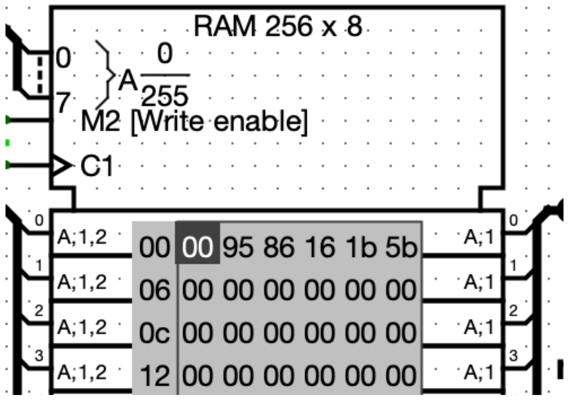
```
ADD 00 00 00
LDR 01 01 01
LDR 00 01 02
ADD 01 01 02
ADD 01 02 03
SUB 01 02 03
```

2. Run the assembler program. It must be run through the folder, rather than just the file (on a mac at least). This will create an image file, which has these results.



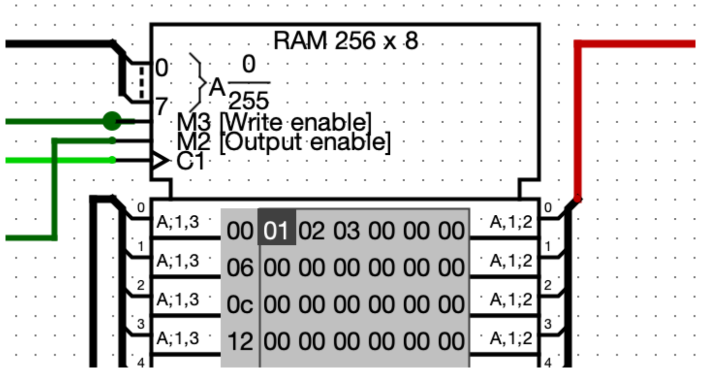
```
0x0
0x95
0x86
0x16
0x1b
0x5b
```

3. Make sure the image file is loaded into the Instruction Memory. You will see the same hexadecimal digits in the first slots as you do in the image file.



```
00 00 95 86 16 1b 5b
06 00 00 00 00 00 00
0c 00 00 00 00 00 00
12 00 00 00 00 00 00
```

4. Click the hand tool at the top of the screen and select the registers you would like to change, then enter a number. Make sure it can be represented in less than 2 bits, so only registers 0-3.



```
00 01 02 03 00 00 00
06 00 00 00 00 00 00
0c 00 00 00 00 00 00
12 00 00 00 00 00 00
```

5. Finally click the clocks or auto-simulate the program by enabling the auto-ticker in the clock simulator tab.

To reset the program you click “reset simulation” under the simulate tab and either run it again immediately or reassemble the program with different instructions. Hopefully Cuppy and this explanation enable you to successfully fill instructions and see them in action.