The Forge's Manual

Nicholas Fontana and Dylan McGrory

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

Explanation of name:

The Forge: We picked this name because it symbolizes our intense effort, resilience, and craftsmanship required to create our CPU, similar to a blacksmith forging steel through fire and hammer.

1. Roles:

Nicholas Fontana-

1. Hardware implementation (Logisim)
2. Documentation

Dylan McGrory-

1. Assembler
2. Hardware implementation (Logisim)

2. How to use Assembler:

1. To start, begin by opening up the .txt file named "assembly"
2. Write your desired Assembly code within the text, save, and close.

Format for writing Assembly code:

1. It's important to note that the assembly code should only be written BELOW the .text segment.
2. One line for each instruction and or declaration of a segment
3. Write your data in the .data section, only the .int data type can be supported.

Here is an example of how you could set up your assembly file:

```
.text
LDR X3, X0
ADD X2, X3, X3
ADD X0, X3, X2
LDR X1, X3
```

```
STR X2, X3
STR X0, X2
STR X1, X0
ADD X2, X1, X2
SUB X2, X2, X3

.data
.int 1
.int 4
.int 2
```

Running the assembler:

1. Open up the Python file named assembler.py
2. Click run and your instructions will be found in the .txt file named, instructions_image.txt.
3. Data will be found in the data_image.txt file

3. Using data files in Logisim:

Now that you have your image files let's use them in Logisim!
Steps on how to do so:

1. First, right-click on instruction memory and click “load image”.
2. Once this is done, select the “instructions_image.txt” for the instructions memory and close out of the window.
3. Repeat the same process with data memory but use “data_image.txt” instead.
4. **EXTREMELY IMPORTANT:** The files must use the v3.0 Hex format for loading both data and instruction memory, so make sure you click that box when loading both.

4. Running “The Forge”

Here comes the best part! You get to see the output of your instruction. Here is how to do it:

1. Simply click on the simulate tab on the top left and enable auto ticks to begin the simulation.
2. Press the pause button after your final instruction executes.

3. Look at your beautiful results in our displays on the right!

5. Architecture

Registers:

4 general purpose registers: X0, X1, X2, X3

Functions:

ADD Function:

1. Syntax: ADD Rd, Rn, Rm
2. Adds two 8 bits numbers (Rn + Rm) and stores the result in Rd

SUB Function:

1. Syntax: SUB Rd, Rn, Rm
2. Subtracts two 8 bits numbers (Rn - Rm) and stores the result in Rd

LDR Function:

1. Syntax: LDR Xt, Xn
2. Loads data at address Xn from data memory to register Xt

STR Function:

1. Syntax: STR Xt, Xn
2. Stores data from register Xt to address Xn in data memory

Encodings:

Instruction	Opcode (7-6)	Rm (5-4)	Rn (3-2)	Rd (1-0)
Add Rd, Rn, Rm	00	2nd register	1st Register	Dest register
SUB Rd, Rn, Rm	01	2nd register	1st Register	Dest register
LDR Xt, Xn	10		Adr register	Dest register
STR Xt, Xn	11	Data register	Adr register	