Dominick Battinelli & Zachary Rimshnick
CS 382 - Project 2
I pledge my honor that I have abided by the Stevens Honor System.

CPU NAME: ONPAR CPU

Job Descriptions

Zack:

- Wired the Instruction Memory and Register file portion of the CPU. Then connected both of our work.
- Wrote the python assembler program to translate our own assembly code to an image file with hexes and addresses.
- Wrote assembler program description

Dominick:

- Wired the ALU and Data Memory portion of the CPU. Then connected both of our work.
- Created the machine code and assembly language
- Wrote the instruction manual for our CPU
- Wrote architecture description of our CPU

Assembler Program

To use our python assembler program to translate our assembly code into a usable image file on Logisim, all you need is to run the program. It will ask for the **name of your Assembly Instructions file (.txt)**, in our assembly language, and then the intended **name for the translated Image file (.txt)**. The image file can either be a pre existing .txt file, or you can enter a new file name and the program will create a new .txt file. The program will read the given instructions file and translate it to hexadecimal and address in an image file, which is to be loaded as the **Instruction Memory** input in Logisim. Our program does not affect data memory, and therefore must be created separately.

Architecture Description

Our CPU contains 4 general purpose registers which can each store a hex value. These registers can be referred to by X0, X1, X2, and X3 in our assembly language. Our CPU can perform 3 functions, it can load data from the memory into a register, and it can add or subtract the values of 2 registers and store the result into a register. In order to load a value into a register from memory the user must use the **`LOD`** instruction. The **`LOD`** instruction can be written in our assembly language as **`LOD imm4 Rd`**. Our data memory can hold 16 hex numbers which can be referred to by the **imm4**. **Rd** is the register in which the data will be loaded into. In order to add or subtract numbers in our CPU the user can use the **ADD** and **SUB** instructions. The format for these instructions is **ADD Rn Rm Rd** and **SUB Rn Rm Rd**, where **Rn** is the first register, **Rm** is the second register, and **Rd** is the register where you want the result to be stored. Our CPU also contains an LED display which displays the hex number being stored into a register for each instruction.

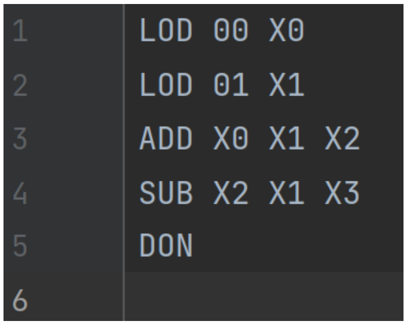
Instruction Manual

LOD imm4 Rd: Loads the value stored in the imm4 data address into register Rd
ADD Rn Rm Rd: Adds the value stored in Rn to the value stored in Rm and stored it in Rd
SUB Rn Rm Rd: Subtracts the value stored in Rn from the value in Rm and stores it in Rd
DON: Signifies the end of the program

Instructions	OPCODE	**imm4** (Memory address)		**Rd** (Destination)
LOD imm4 Rd	00	4 bits		2 bits
		Rn (1st register)	**Rm** (2nd register)	
ADD Rn Rm Rd	01	2 bits	2 bits	2 bits
SUB Rn Rm Rd	10	2 bits	2 bits	2 bits
DON				

Notes

- Registers must be a register from X0 to X3
- Data memory only holds 16 bytes of data so **imm4** must be a number from 0 to 15
- The OPCODE of each instruction takes 2 bits as there are 3 instructions to choose from
- **Rn**, **Rm**, and **Rd** all take 2 bits in machine code as there are 4 registers to choose from
- The data address takes 4 bits as there are 16 memory addresses to choose from



```
LOD 00 X0
LOD 01 X1
ADD X0 X1 X2
SUB X2 X1 X3
DON
```

sample program

Extra Credit: *Added an LED display which displays the hex number being stored into a register for each instruction.*