# CUP

(Don’t worry, the coffee is made out of mineral oil! 😉)

By: Adib Osmany & Neel Meyyur

# Job Description

Adib Osmany:

- Created Register File
- Created “ALU”
- Designed the main function in the python file
- Came up with project name and outline of manual

Neel Meyyur:

- Created Instruction Memory
- Worked on data memory segment
- Designed the pad function in the python file
- Worked on architecture description & general format of the assembler.

---

# How To Use Our Assembler Program

1) **Prepare the assembly code**: Write your assembly code in a file named "CUP.txt", where each line contains a valid assembly instruction. Do not include any other lines or content that aren't assembly instructions! Make sure this file is in the same folder as the other files mentioned in this procedure.
2) **Run the program:** Execute the assembler program "CUP.py".
3) **Check the output file:** Open the output file that was generated after the assembler program was generated. This file should be called "image file.txt".
4) **Prepare the digital circuit file:** Make sure that "CUP.circ" is opened up in Logisim evolution. Set the autotick rate for the simulator to 4 Hz. (slow enough to see what is happening, but not too slow where it becomes boring)
5) **Load the assembler code into the RAM:** Load the code generated from "image file.txt" into the RAM. This should be done by right clicking the RAM and clicking "Load image…".
6) **Run the simulation:** Enable auto-tick so that the simulation runs. Be amazed at the results of the CPU crafted by two noble CS382 students!

---

# Architecture Description

The CPU comprises a Program Counter (PC), an instruction memory, a register file, and an Arithmetic Logic Unit (ALU). The CPU circuit is equipped with four general-purpose registers labeled X0 to X3, and an additional constant register, X31, designed to hold a constant value of 0. Each of the general-purpose registers can store up to 24 bits of data. The constant 0 register, X31, serves the purpose of preventing unintended changes to other registers after completing a set of assembly instructions.

The CPU supports four fundamental functions, including addition, multiplication, storing a value into a register, and loading a value from a register. These functions allow the utilization of values from both registers and immediate values. The design ensures flexibility and control over data manipulation while incorporating the constant 0 register, as a safety measure, to maintain the integrity of other registers.

# General Format

| Instruction | opcode | Rd | Rm | Rn | MEM |
|---|---|---|---|---|---|
| ADD Rd, Rm, Rn | 0001 | Register | Register or Constant | Register or Constant | N/A |
| MUL Rd, Rm, Rn | 0010 | Register | Register or Constant | Register or Constant | N/A |
| STR Rm, Rd | 0001 | Register | Register or Constant | N/A | N/A |
| STR Rd, MEM | 0011 | Register | N/A | N/A | Memory |
| LDR Rm, Rd | 0001 | Register | Register or Constant | N/A | N/A |
| LDR Rd, MEM | 0100 | Register | N/A | N/A | Memory |

# Example Format

| Instruction | opcode | Rd | Rm | Rn | MEM |
|---|---|---|---|---|---|
| ADD X0, 5, 6 | 0001 | X0 | 5 | 6 | N/A |
| STR 2, X1 | 0001 | X1 | 2 | N/A | N/A |
| MUL X2, X1, X0 | 0010 | X2 | X1 | X0 | N/A |
| LDR X3, X2 | 0001 | X2 | X3 | N/A | N/A |
| STR X1, MEM | 0011 | X1 | N/A | N/A | Memory |
| LDR X2, MEM | 0100 | X2 | N/A | N/A | Memory |

–

**Note:** The opcodes for ADD Rd, Rm, Rn; STR Rm, Rd; and LDR Rm, Rd are the same. This is something that has been done intentionally. Here is an explanation:

Let's say that we have the line "STR 5, X1" in our assembly program. We are essentially adding 5 to 0, and storing that result in X1. This is the same idea for "LDR 5, X1" except we are loading the result in X1, not storing.

–

**Note:** We also have two sets of instructions for STR and LDR. This has been done to provide more flexibility with our CPU, making it more user friendly!

---