CS 382 Project 2 User Manual

Breona Pizzuta and Ben Carpenter
Group 32 on Canvas
“I pledge my honor that I have abided by the Stevens Honor System”

1. Introduction

CPU Name: Shudong-INATOR

We are pleased to introduce the Shudong-INATOR, a state-of-the-art CPU that combines outstanding performance with a lighthearted homage to our distinguished professor Shudong, in the spirit of innovation and a little comedy. The term "Shudong-INATOR" adds a playful, memorable twist to the strength and accuracy of a high-performance machine. The Shudong-INATOR is designed to handle the most difficult chores with ease, speed, and efficiency, much as our professor has helped us understand difficult subjects with clarity and insight. We hope that this CPU will demonstrate our technical proficiency as well as our capacity to take pleasure in the learning process.

Job Description:
Ben: Worked mainly on the CPU architecture and various testing. Helped Breona with the debugging of code to get it to run properly.
Breona: Worked mainly on writing the code and writing the User Manual. Helped Ben fix errors within the CPU.
Overall we worked on most sections of the project at the same time, together in a collaborative environment.

2. Assembling Instructions

How to use your assembler program (e.g., what command, what library to link, etc), and how to load it to the instruction and/or data memory:

Running the Assembler:
To assemble an assembly program into a machine code memory image:

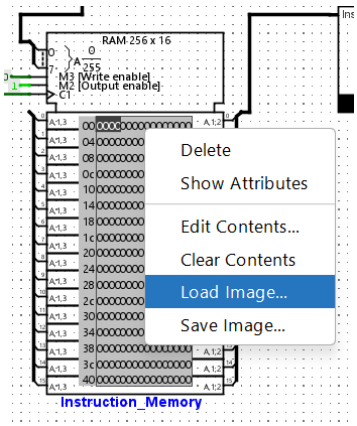
- Create a .txt file
    - use a .data section to define any variables in binary in this format:
        - variableName: 01
    - use a .text section to write your instructions
        - Instruction format shown in section 4.
- Save your Shudong-INATOR assembly code in a .txt file, named instructions.txt
- Open a terminal or command prompt.
- Run the assembler program: python assembler.py instructions.txt

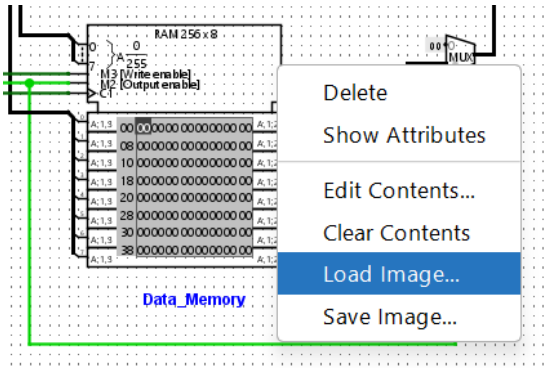
The program will output two files:

- code_image.txt: Contains the hexadecimal representation of the instructions (machine code) for the .text section.
- data_image.txt: Contains the hexadecimal representation of the initialized data for the .data section.

Loading Memory Images

- To use the image files they need to be stored in the correct RAM.
- Instruction Memory (code_image.txt): Load this file into the CPU's instruction memory.
- Data Memory (data_image.txt): Load this file into the CPU's data memory.4

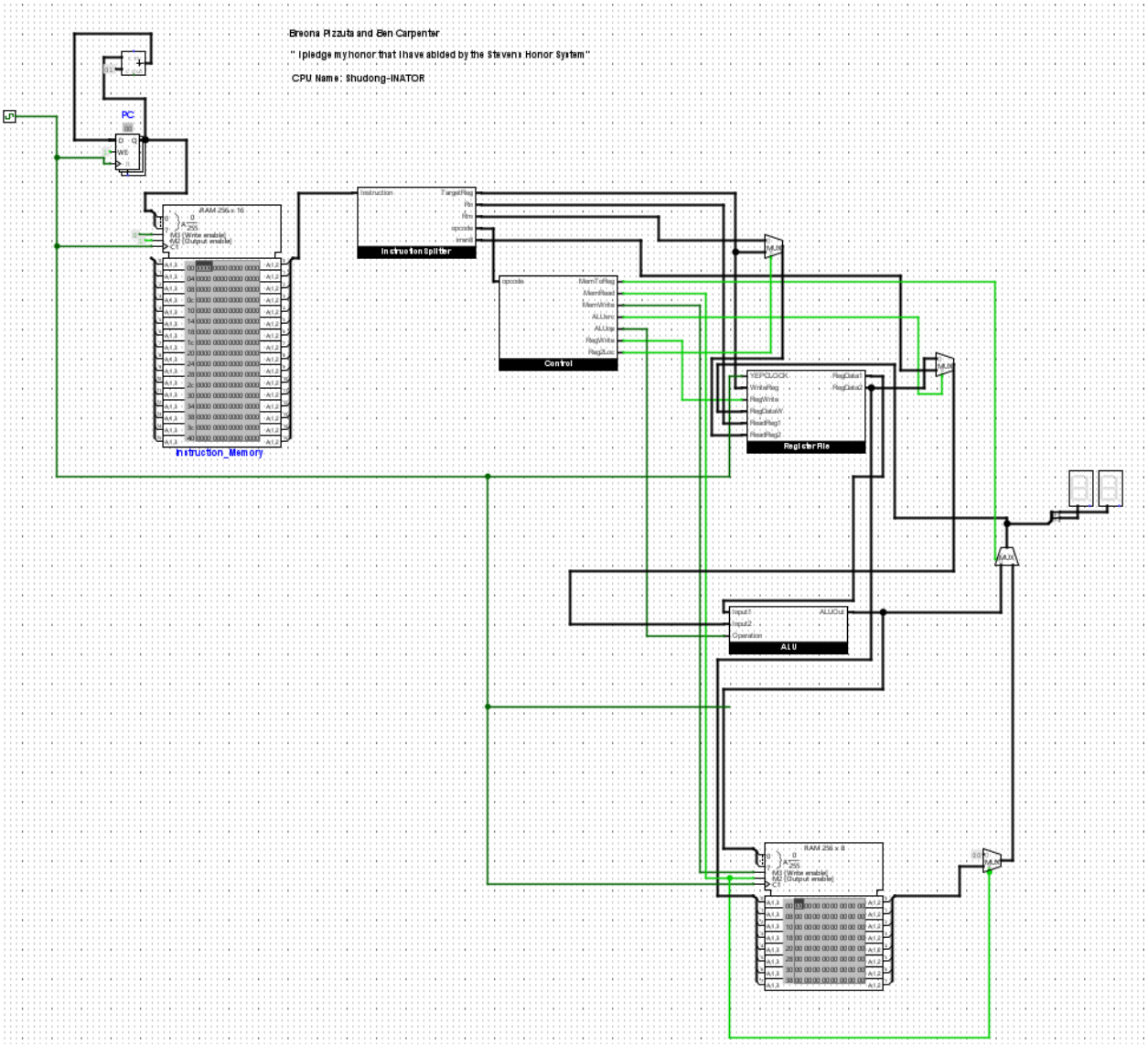




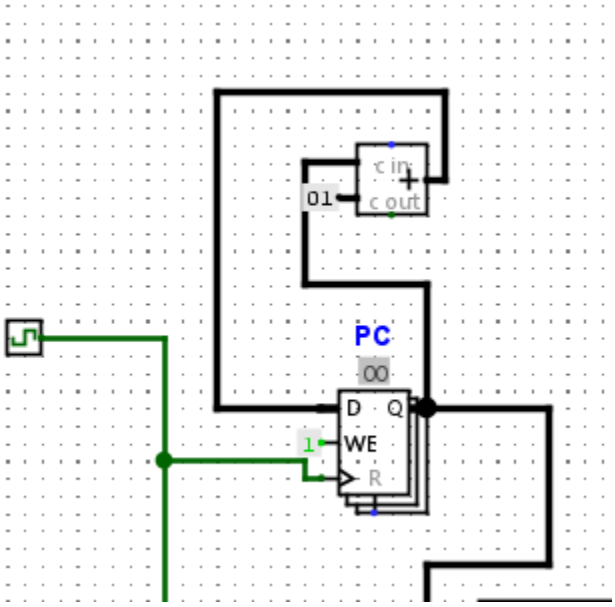
3. CPU Architecture

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:

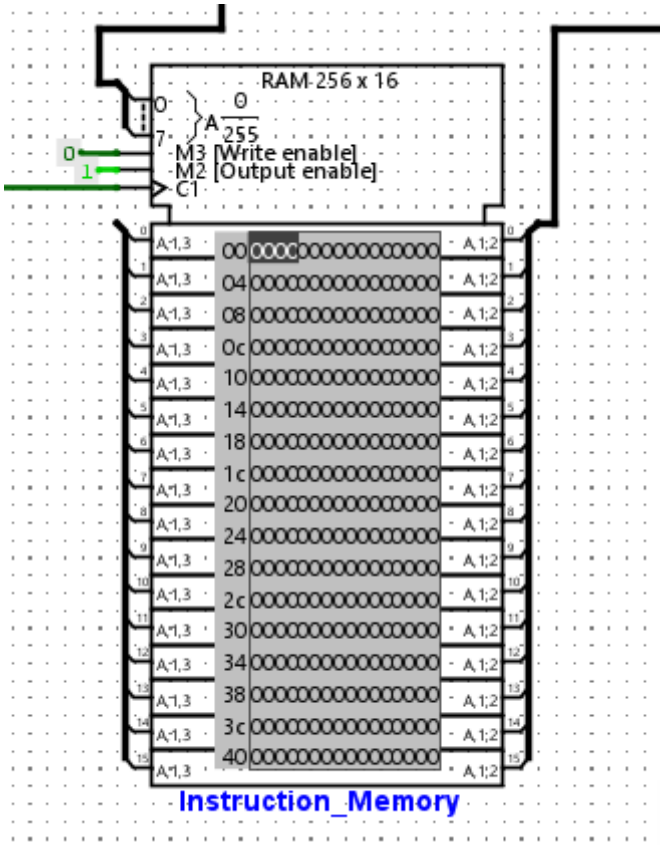
Overview:
Our CPU is composed of a PC, Instruction_Memory, Instruction Splitter, Control Unit, Register File, ALU, and Data_Memory. This is shown below:



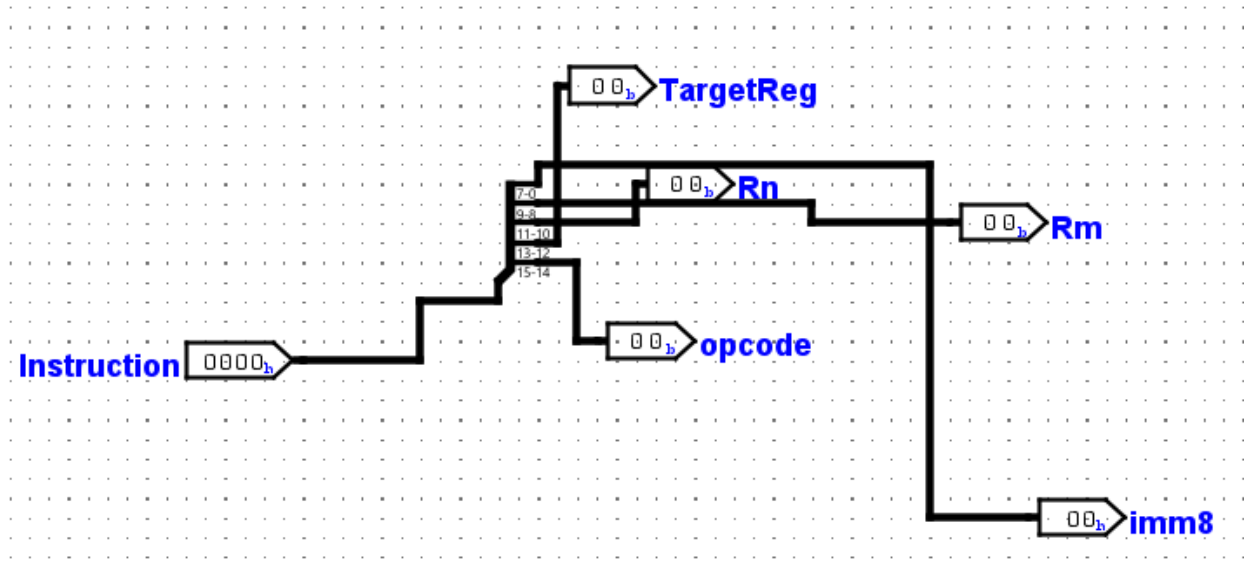
PC:
Used to move through the instructions.



Instruction_Memory:
Holds the instructions. 16 bits per instruction or 4 hex digits long.



Instruction Splitter:
It takes the 4-hexadecimal-digit instruction and splits it into two bits for the opcode to determine the operation, two bits for the target register (TargetReg), two bits for the first register called in the instructions (Rn), two bits for the second register on the instructions (Rm), and an immediate number of 8 bits (imm8). We use the second register for ADD and SUB, and the immediate number is used for LDR and STR.



(ignore the mess of wires)

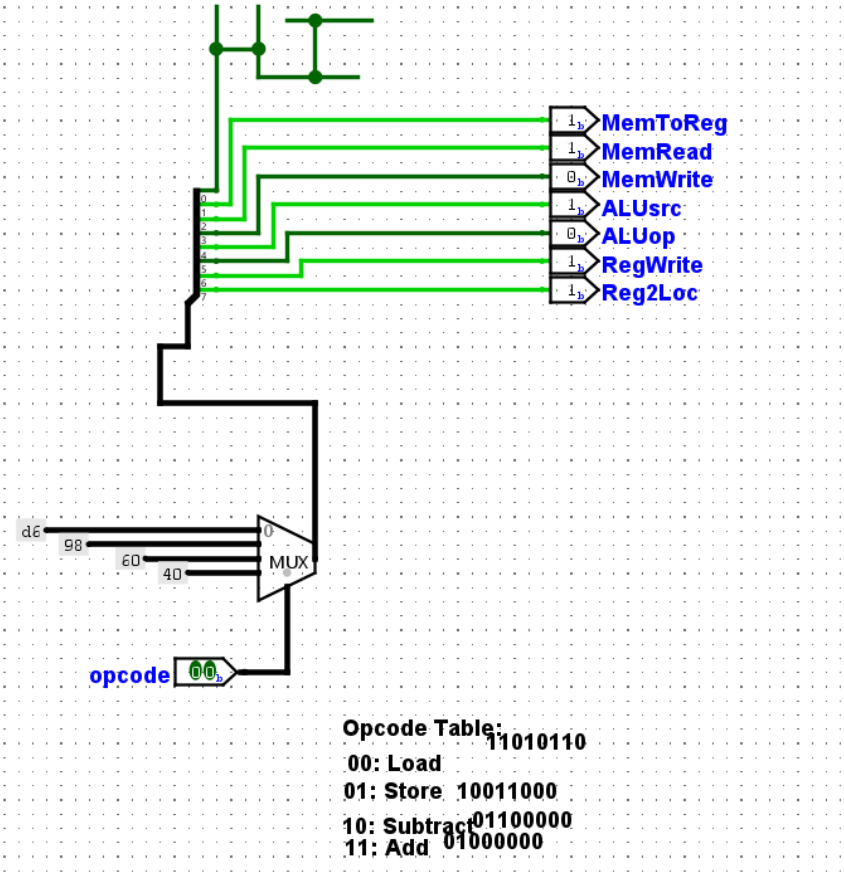
Control Unit:
The control unit translates the opcode to determine what operation we will be performing. This will determine whether or not we use the ALU.

The opcode for each instruction type is as follows:
00: Load (LDR)
01: Store (STR)
10: Subtract (SUB)
11: Add (ADD)

The outputs used here are: MemToReg, MemRead, MemWrite, ALUsrc, ALUop, RegWrite and Reg2Loc.

The table below shows the outputs used to execute each instruction:

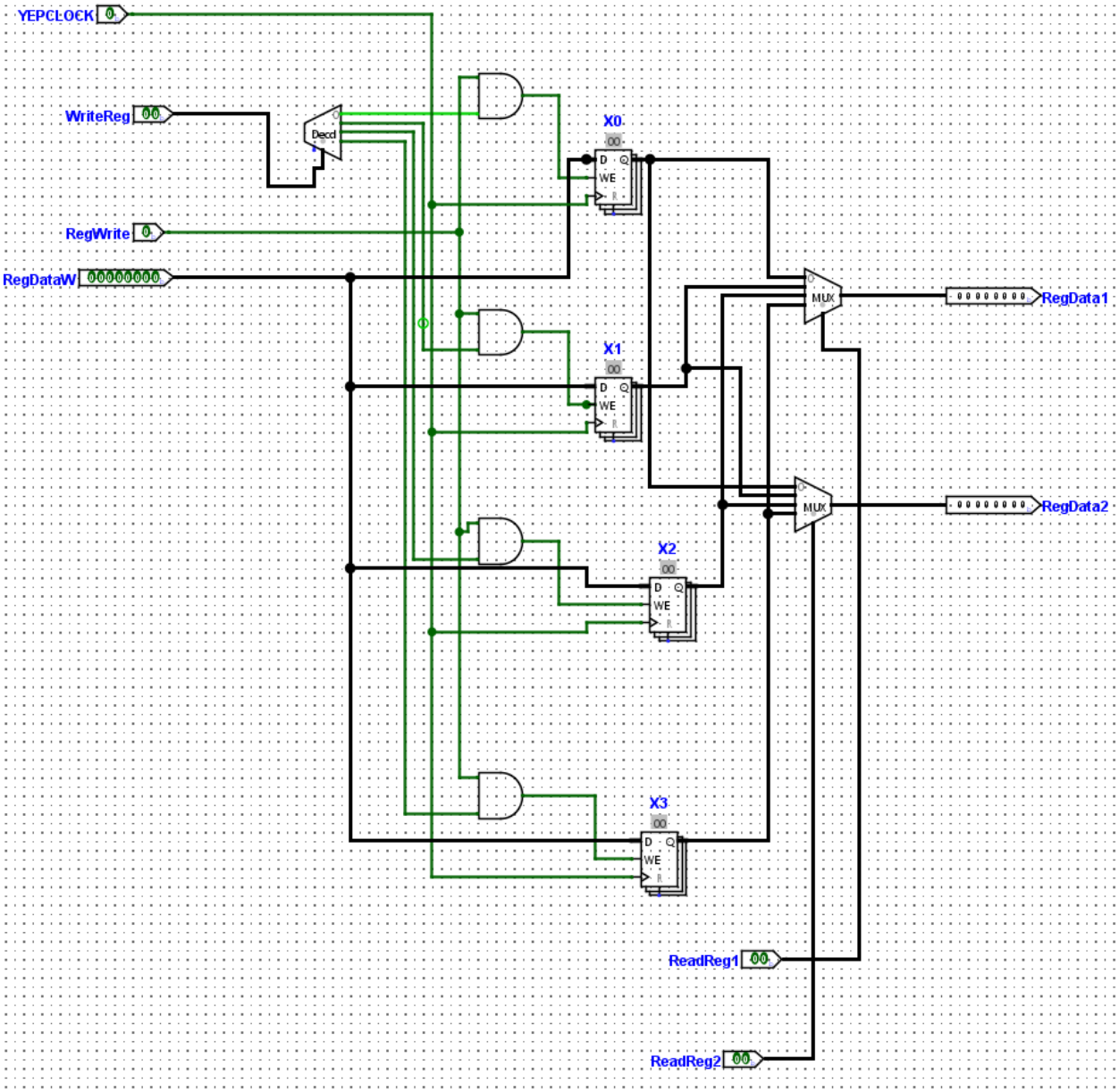
		LDR	STR	ADD	SUB
Reg2Loc	0 for Rm or 1 for Rt	1	1	0	0
RegWrite	Writes data to register	1	0	1	1
ALUop	Opcode for + (0) or - (1)	0	0	0	1
ALUsrc	imm or second reg	1	1	0	0
MemWrite	Writes data to memory	0	1	0	0
MemRead	Reading from memory	1	0	0	0
MemToReg	Deciding if you take calculate from ALU (0) or ReadDatam (1)	1	0	0	0
Extra bit	0 as a buffer to get an even 8 bits	0	0	0	0
Full Opcode	Full output for each	11010110	10011000	01000000	01100000



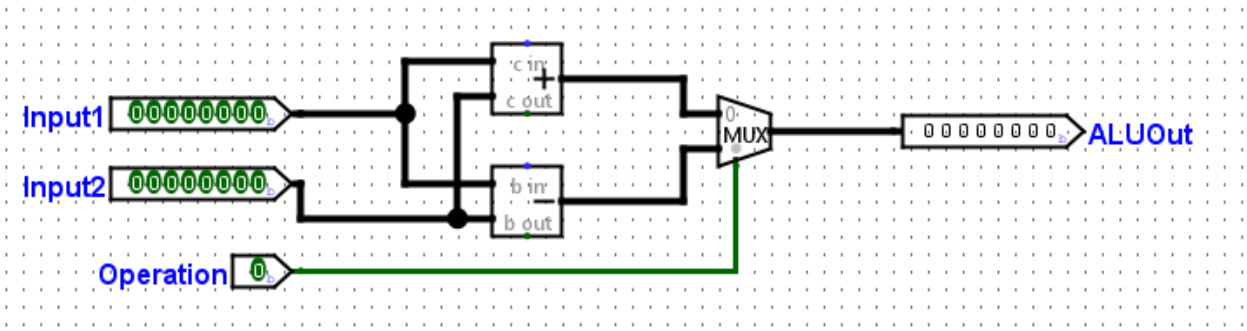
Register File:

Four General-Purpose Registers

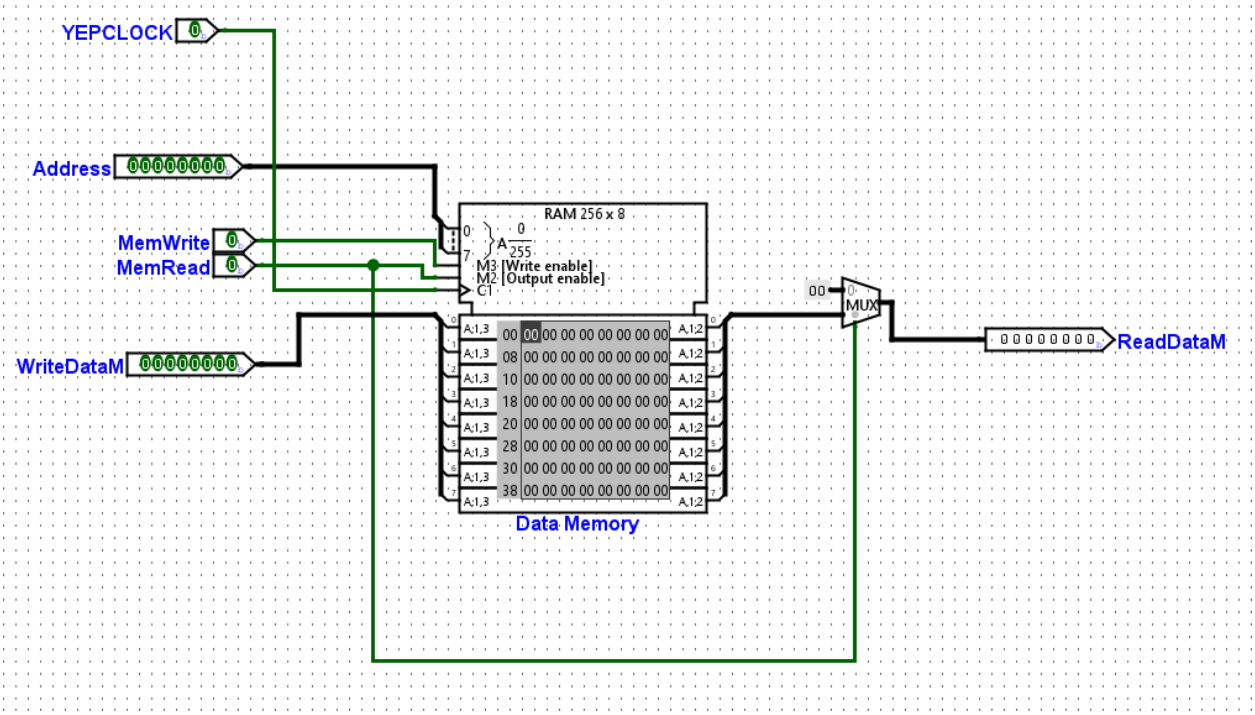
- The register file holds the registers X0-X3 which store data for reading and writing
- The CPU contains 4 general-purpose registers, referred to as X0, X1, X2, and X3.
- Registers can be used as sources or destinations in instructions.



ALU:
The Arithmetic Logic Unit, or ALU does the computations for the instructions ADD and SUB. In the ALU, ADD is represented as 0 and SUB is represented as 1.



Data_Memory:
The Data Memory is used when we use the instruction LDR to load from memory or when we use the instruction STR to write/store in memory.



4. Instruction Format

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

Each instruction follows the format: Opcode (2 bits) + Target Register (Rd) (2 bits) + Source Register 1 (Rn) (2 bits) + Source Register 2 (2 bits) + Immediate (Rm/Imm) (8 bits)

Supported Instructions

- ADD: Adds two register values, Rn and Rm and stores the result in a destination register Rd.
    - ADD Rd, Rn, Rm
    - Binary encoding: 01000000
- SUB: Subtracts one register value (Rn) from another (Rm) and stores the result in a destination register (Rd).
    - SUB Rd, Rn, Rm
    - Binary encoding: 01100000
- LDR: Loads a value from memory from location (Rn) and Offset of Imm8 into a register (Rd).
    - LDR Rd, [Rn, Imm8]
    - Binary encoding: 11010110
- STR: Stores the value of a register (Rd) at a memory location (Rn) with a specified offset of Imm8.
    - STR Rd, [Rn, Imm8]
    - Binary encoding: 10011000

Why This Number of Bits?

- Opcode (2 bits): Allows for up to 4 instructions, sufficient for this CPU design.
- Registers (2 bits): Addresses up to 4 general-purpose registers.
- Immediate (8 bits): Supports values between 0–256, sufficient for simple programs.

5. Demo Program

```
.data
hi: 11
.text
LDR X0, [X0, 0]
ADD X0, X0, X0
ADD X1, X0, X0
SUB X0, X1, X0
STR X0, [X0, 20]
```

This program just quickly samples several of the different functionalities of the CPU, including every unique instruction, utilizing multiple registers, and using the same register multiple times in one instruction.

6. Extra Credit

We completed two extra credit assignments:

- Making your assembler recognize data and text segment so it can generate two image files. They will be loaded into instruction and data memory separately
- Adding extra fun components, such as LED display to show calculation results.