DESIGN NOTES PROJECT-2

BY-TEAM 1

Sai Sharath Chandra Mahankali G21123208

Sayyam Rakesh Jain G30483119

In this analysis, we are required to build a simple CISC-based computer simulator that aims to be implemented and used as a tool to illustrate how instructions are processed and stored.

It has the following characteristics-

- 4 General Purpose Registers (GPRs) – each 16 bits in length
- 3 Index Registers – 12 bits in length
- 16-bit words
- Memory of 2048 words, expandable to 4096 words
- Word addressable

The four GPRs are numbered 0-3 and can be mnemonically referred to as R0 – R3. They may be used as accumulators. The index registers are mnemonically referred to as IXR1 or IXR2 or IXR3. The machine has other registers Program Counter (PC), Memory Address Register (MAR), Memory Buffer Register (MBR), Instruction Register (IR), and Machine Fault Register (MFR).

INSTRUCTIONS REFERENCE-

Opcode	Instruction	Description
01	LDR r, x, address[,I]	Load Register From Memory, r = 0..3 r <– c(EA)
02	STR r, x, address[,I]	Store Register To Memory, r = 0..3 Memory(EA) <– c(r)
03	LDA r, x, address[,I]	Load Register with Address, r = 0..3 r <– EA
41	LDX x, address[,I]	Load Index Register from Memory, x = 1..3 Xx <- c(EA)
42	STX x, address[,I]	Store Index Register to Memory. X = 1..3 Memory(EA) <- c(Xx)

DEBUGGING PANEL-

The Registers Panel Area displays the values of all kinds of registers-

- R0-R3:16-bit General Purpose Registers.
- IXR1-IXR3:16-bit Index Registers used for pointing to operand addresses during the running of the program.
- MAR:12-bit Memory Address Register.
- MBR:16-bit Memory Buffer Register.
- MFR: 4-bit Machine Fault Register.
- PC: 12-bit Program counter, a register in the computer processor that has the address of the next instruction which is to be executed from memory.
- IR: 16-bit Instruction Register that holds the instructions that are being executed currently.
- Assembler:

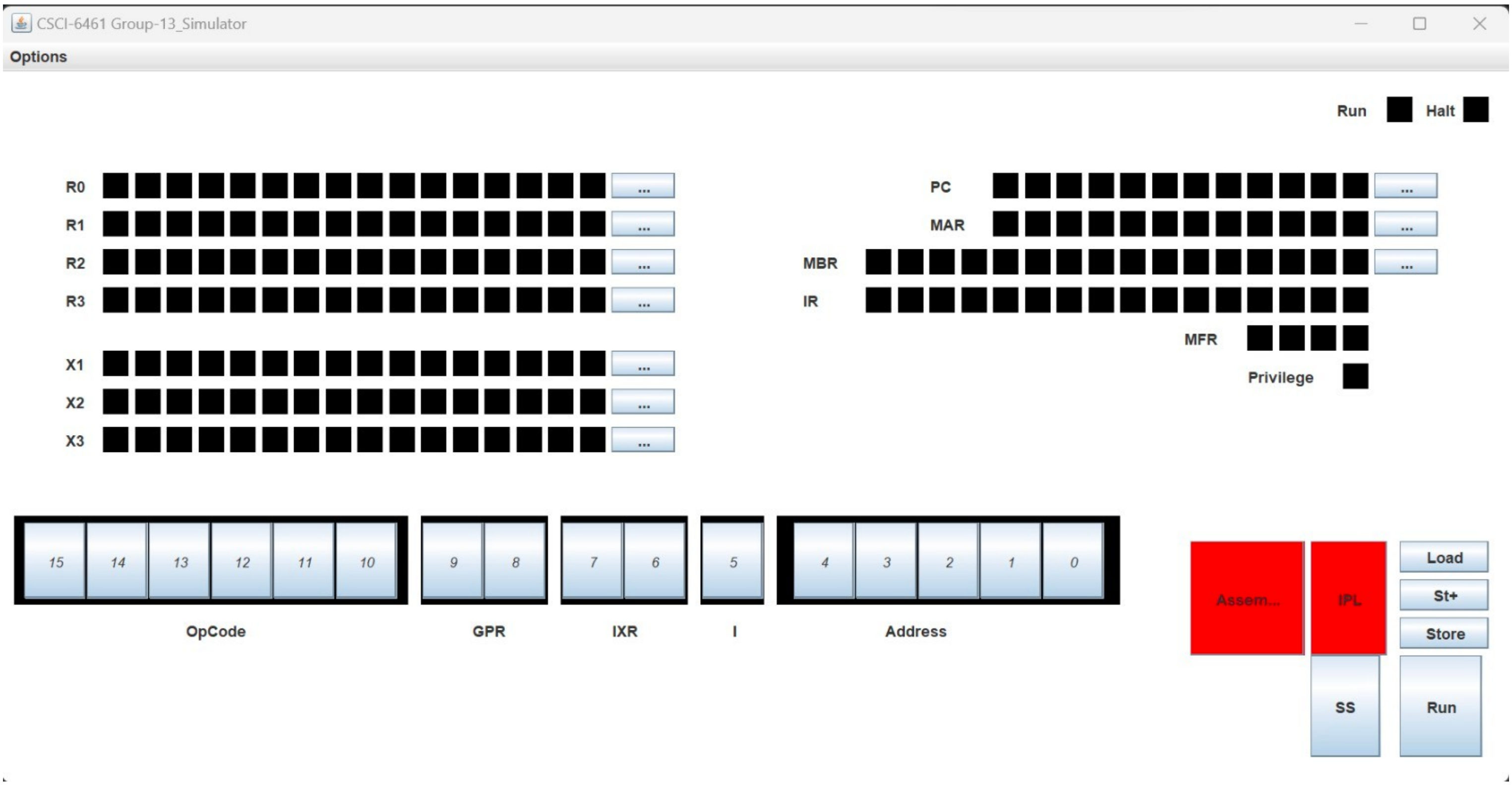
The Panel has several control buttons-

- IPL: The IPL button initializes the memory, instructions and PC with the value loaded in a common bus.
- Run: The run button will execute all the instructions in the input file and will provide the final output
- SS: Single Step button which is used to execute one step at a time to determine how it is functioning.
- Load: Executes load operation which loads the value of the address in MAR into MBR.
- Store: Executes store operation which stores the value in MBR into the address of MAR.
- Additionally we have added Assembler button to our stimulator that takes the source code as an input text file and translates into a file that uses hexadecimal locations

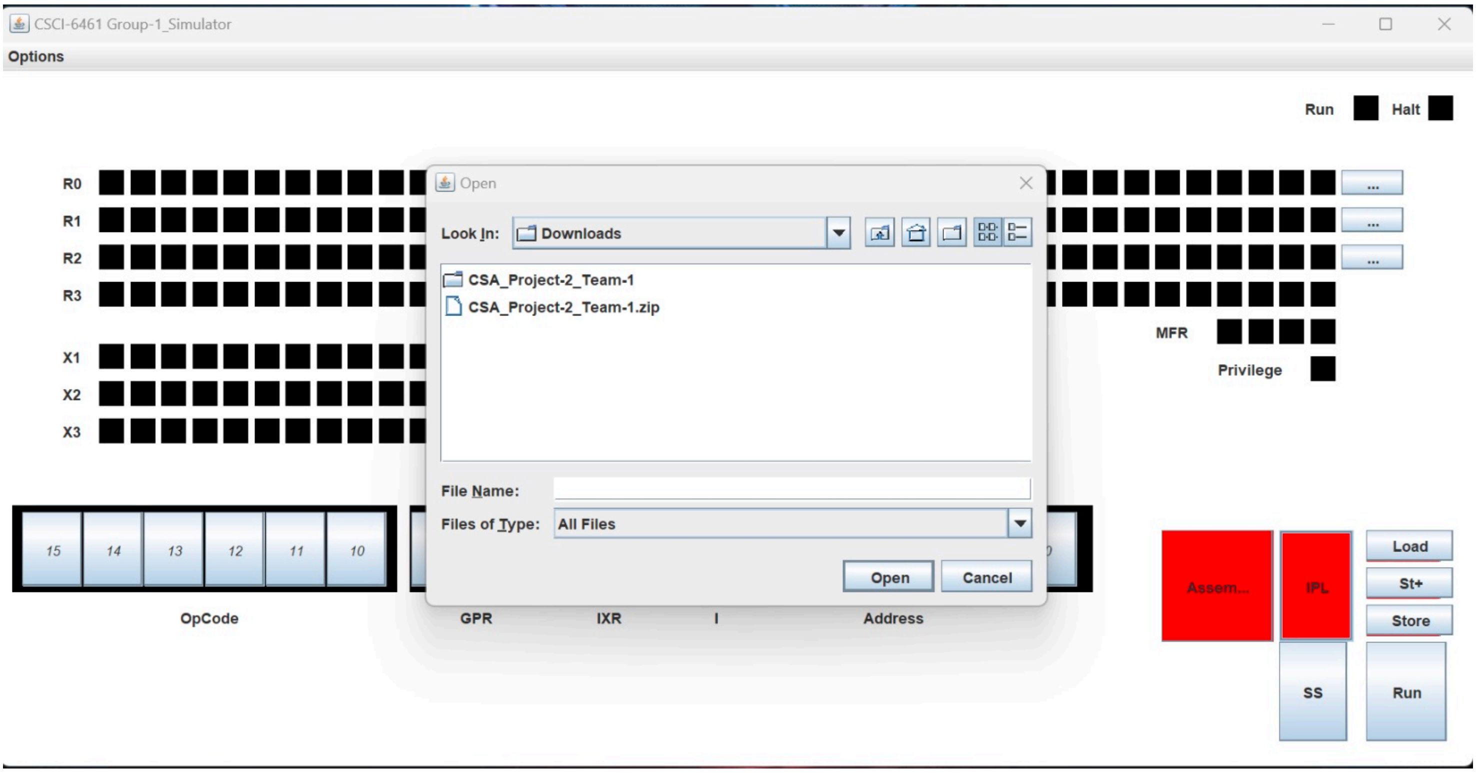
Our Objective in this project is to transform the code into a file that closely resembles the format of the initialization file required in the project-1

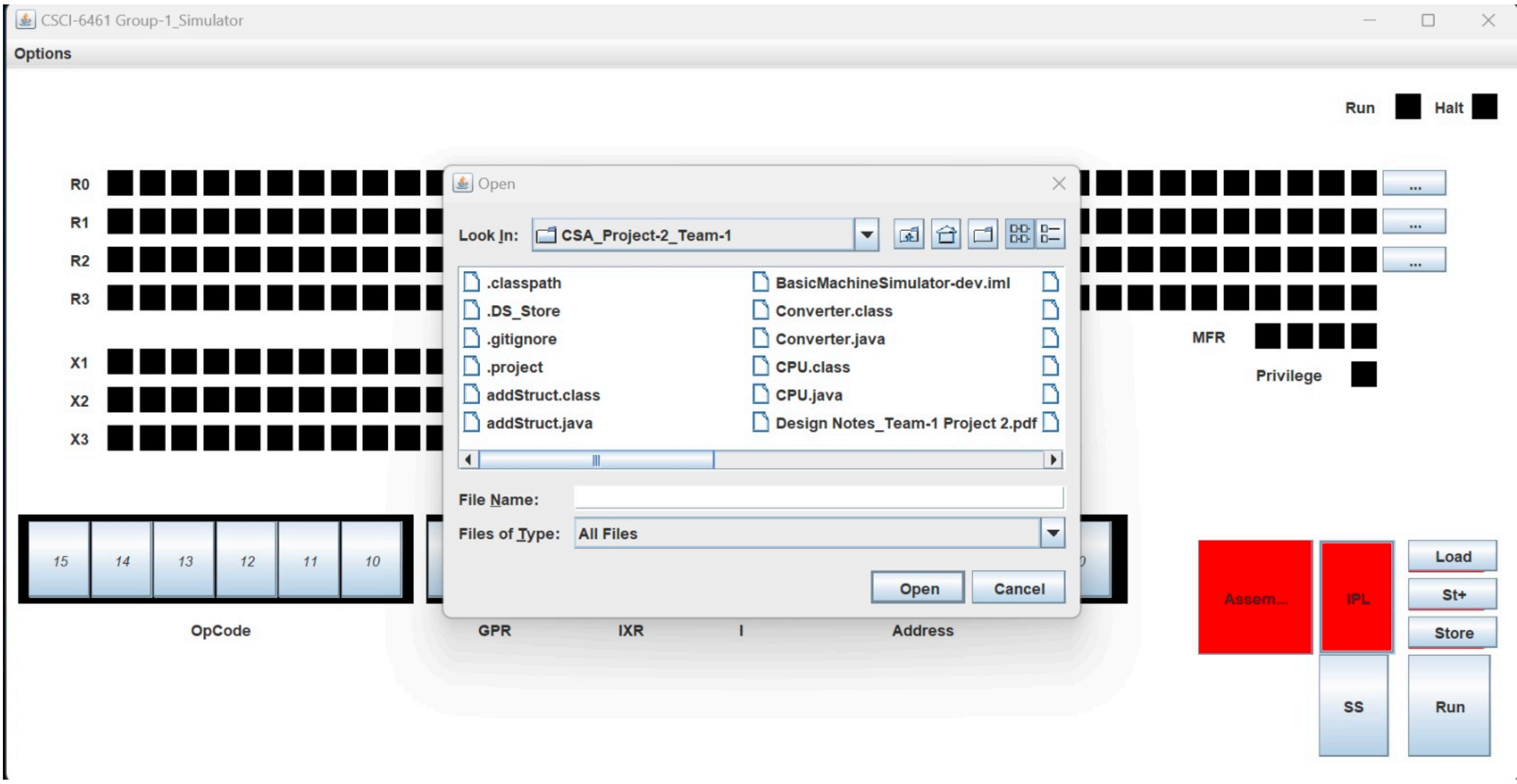
Steps to run the Assembler:

Step-1: Click the “Assembler” button , and select the desired file(i.e example input.txt)



Step-2 : On clicking assembler button it takes input text f



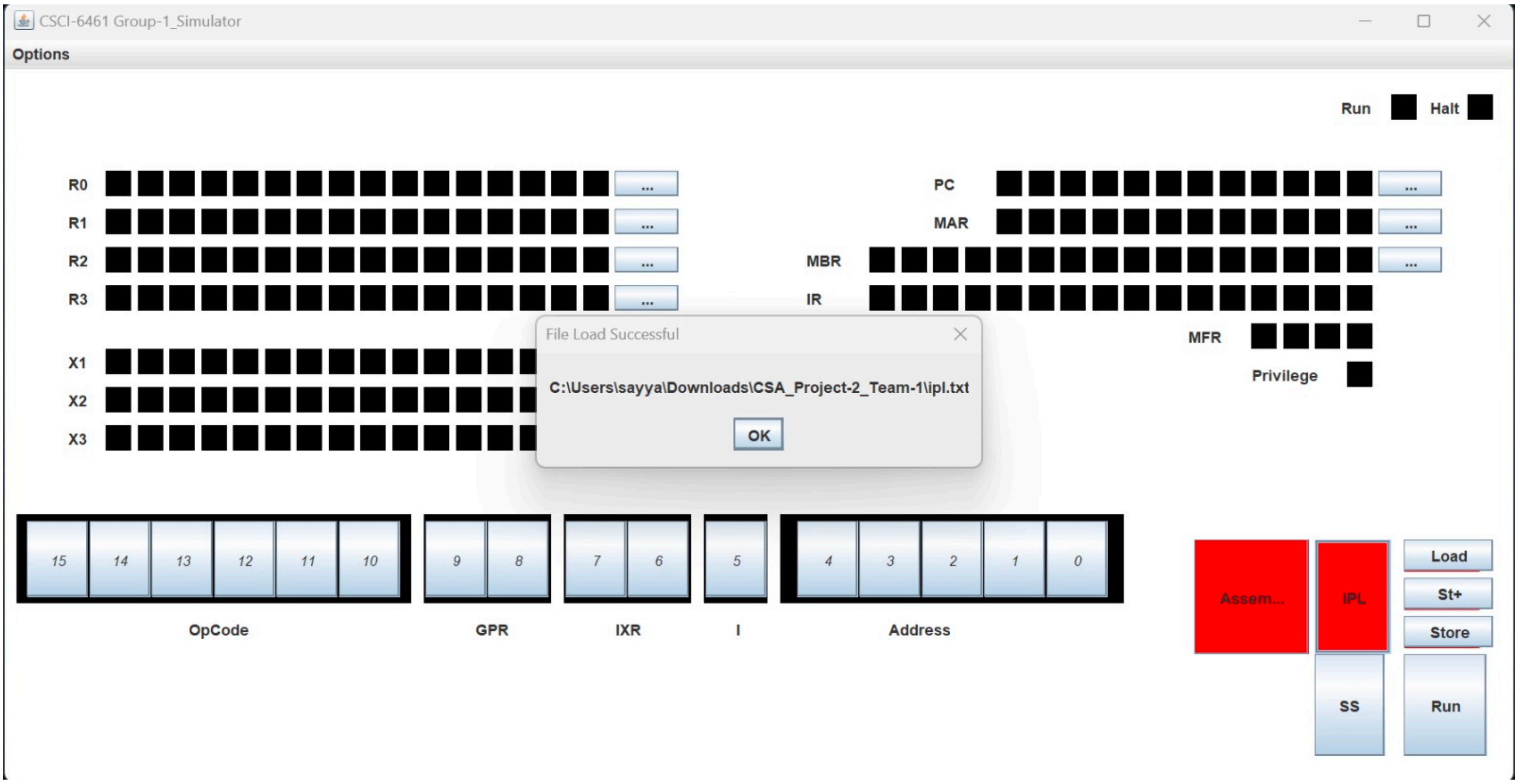


Look In: CSA_Project-2_Team-1

.classpath
.DS_Store
.gitignore
.project
addStruct.class
addStruct.java
BasicMachineSimulator-dev.iml
Converter.class
Converter.java
CPU.class
CPU.java
Design Notes_Team-1 Project 2.pdf

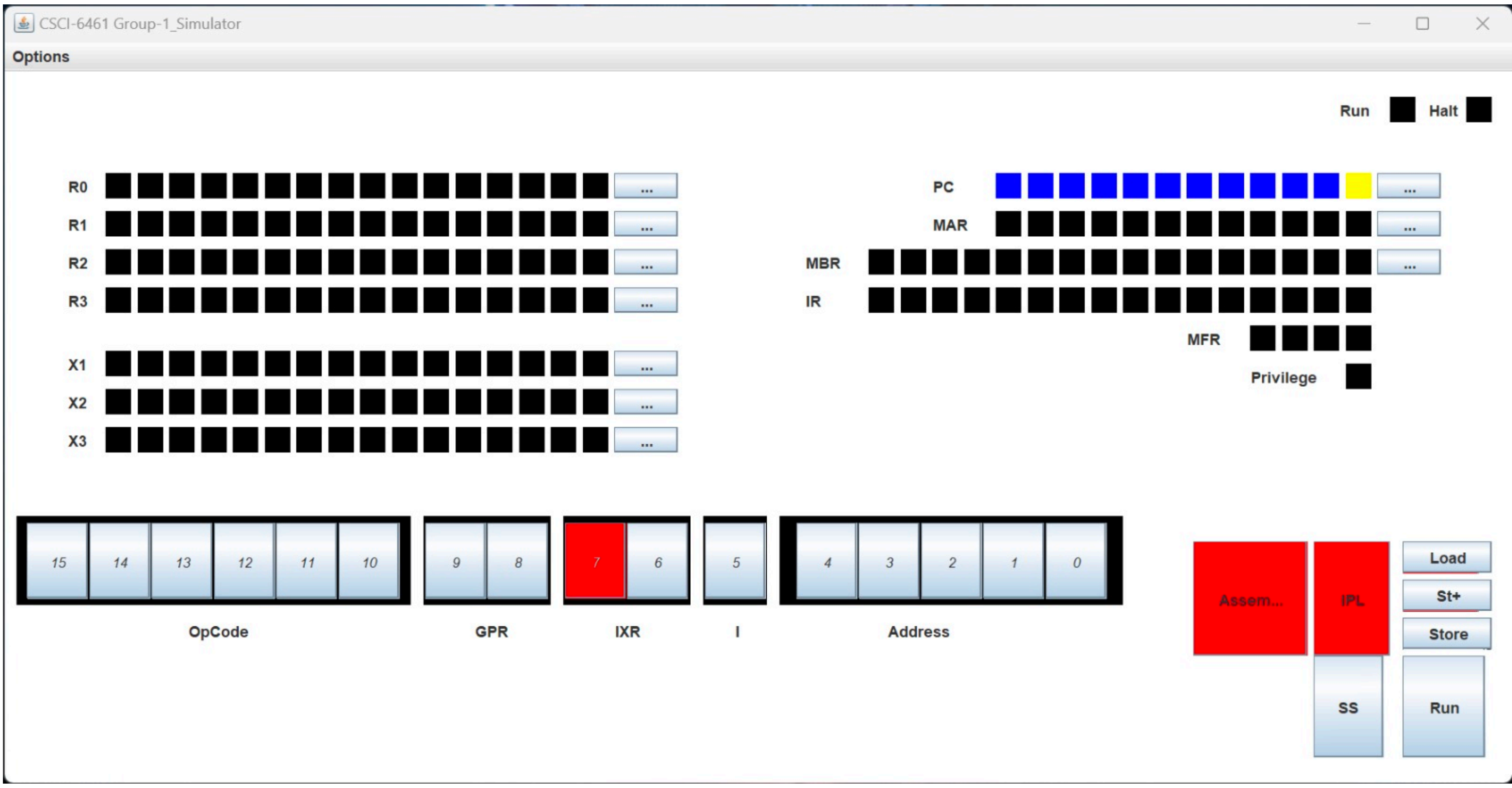
File Name:

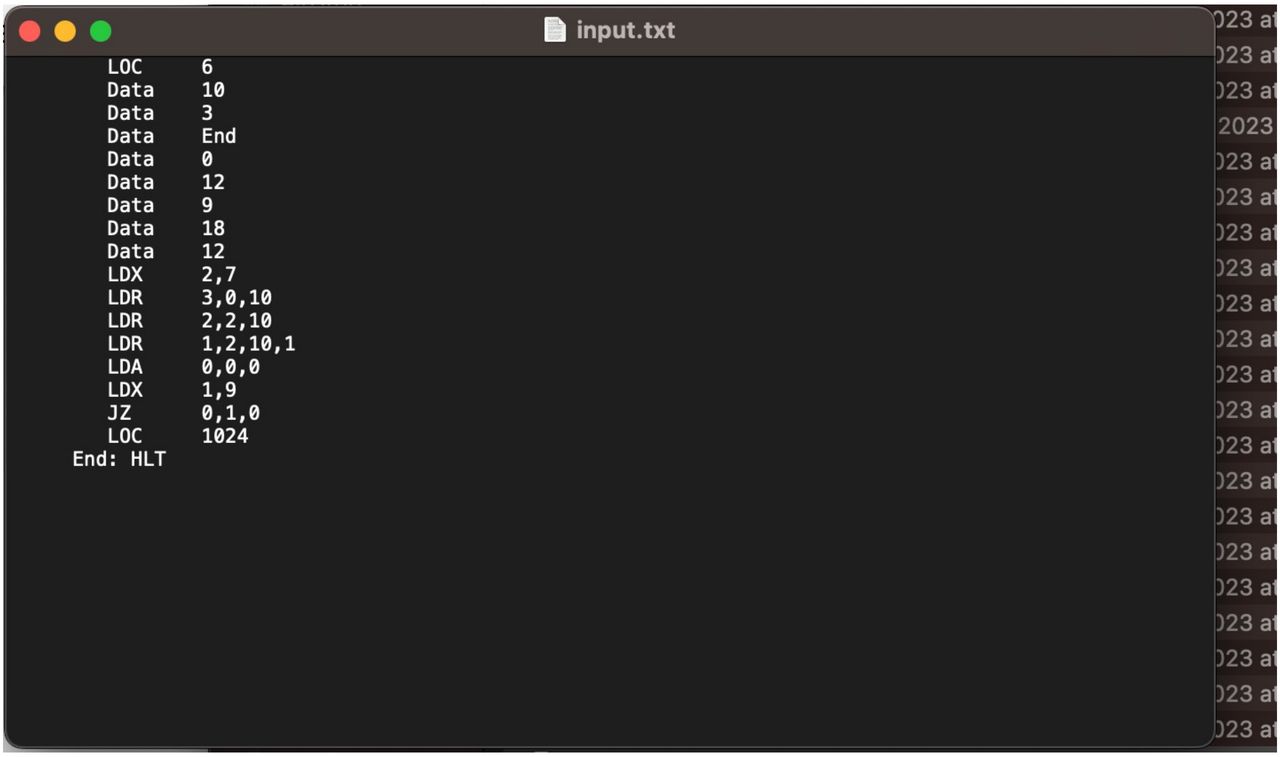
Files of Type: All Files



File Load Successful

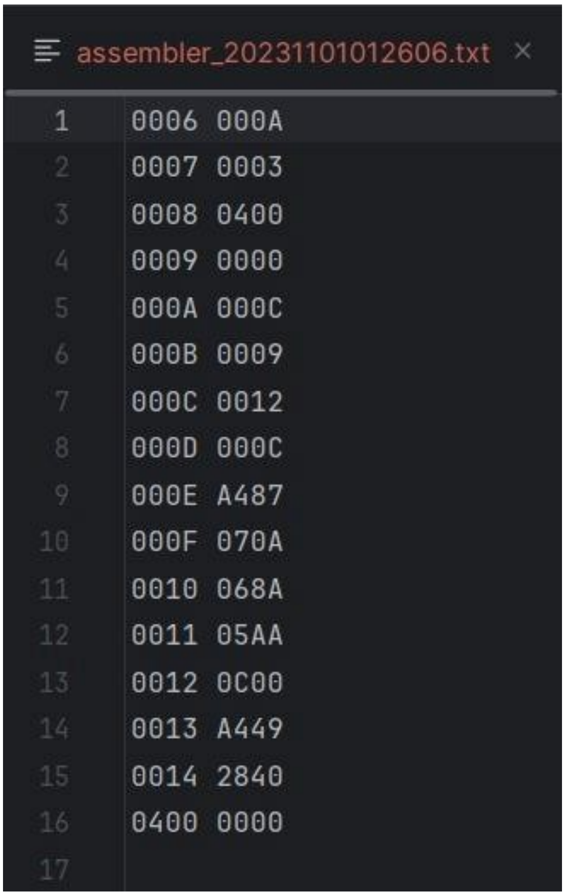
C:\Users\sayya\Downloads\CSA_Project-2_Team-1\ipl.txt





```
      LOC     6
      Data    10
      Data    3
      Data    End
      Data    0
      Data    12
      Data    9
      Data    18
      Data    12
      LDX     2,7
      LDR     3,0,10
      LDR     2,2,10
      LDR     1,2,10,1
      LDA     0,0,0
      LDX     1,9
      JZ      0,1,0
      LOC     1024
   End: HLT
```

The Assembler is downloaded as a.txt file and uses a table or directory to transform the text op code portion to hexadecimal.



```
0006 000A
0007 0003
0008 0400
0009 0000
000A 000C
000B 0009
000C 0012
000D 000C
000E A487
000F 070A
0010 068A
0011 05AA
0012 0C00
0013 A449
0014 2840
0400 0000
```