CSCI 6461 Fall 2023 Group 10 Project 1 – Hand Guide

Project 1:

To run the project refer to "Running the input file" section in the page number 3.
To know how to run the simulator follow this document from start.
Input File standard name : **ProgramLoadFile.txt**

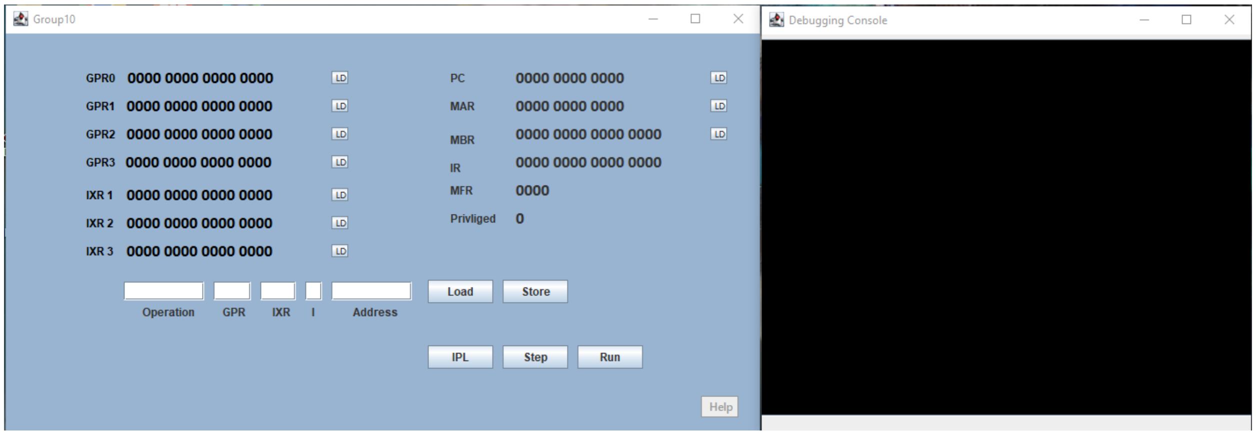
JDK Compiler compliance level : 17
JRE version : jre.full.win32.x86_64_17.0.8.v20230831-1047

Console of the Project :
When you run the project using the command

```
java -jar CSCI6461F23G10Project1.jar
```

You'll see two windows. One is our simulator console and other is our debugger.
Console is where you interact with the simulator and debugger window is a non editable text area to show user the internal operations.

Note :
Always press IPL button when running the simulator to initialize the memory.



```
GPR0   0000 0000 0000 0000      PC        0000 0000 0000
GPR1   0000 0000 0000 0000      MAR       0000 0000 0000
GPR2   0000 0000 0000 0000      MBR       0000 0000 0000 0000
GPR3   0000 0000 0000 0000      IR        0000 0000 0000 0000
IXR 1  0000 0000 0000 0000      MFR       0000
IXR 2  0000 0000 0000 0000      Privliged 0
IXR 3  0000 0000 0000 0000
```

There are several registers in our console. They are GPR0, GPR1, GPR2, GPR3,IXR1, IXR2,IXR3,PC,MAR,MBR,IR,MFR,PRIVILIGE.

Description of the console components :

The "**IPL**" button is to load the input file into the memory.

The "**LD**" buttons are to load the user input entered through the text fields into the corresponding registers.

The **text fields** are for the user to enter the instruction word.

The "**Store**" button is to store the MBR value into the Memory at location MAR.

The “**Load**” button is to retrieve the MBR value from memory specified at location MAR.

The “**Run**” button is to run the program at one go.

The “**Step**” button is run the program instruction by instruction.

The “Help” button is to show the help text for the user to operate the simulator. Currently it is disable as part of project phase 1.

Limitation of the input fields :

The input fields are expected to receive a certain number of bits, although it may take as many digits as the user enters. To execute the user instruction safely follow the notation.

Opcode field contains only 6 bits

GPR field contains only 2 bits

IXR field contains only 2 bits

I field contains only 1 bit

Address field contains only 5 bits

Loading the values into the registers manually:

To load a value in the register manually enter the instruction in the text fields and click the corresponding “LD” button.

For example enter “000001 00 00 0 00011” in the input and click “LD” right next to the GPR0, the GPR0 value will be set to this value.

The above steps hold true for all the registers, except the length being different for few of the fields.

To set PC and MAR, enter “000000 00 00 0 XXXXX”. The XXXXX part is a binary string that we will be setting in the PC and MAR field.

Note :

The setting of PC and MAR is limited to 5 bits when given through user input text fields.

MFR and privilege are fields that are present on console but doesn’t server any purpose as part of Project Phase 1.

Loading value from a specific location in the Memory by querying manually(Functionality of “Load” button)

To do so,

First you need to click the IPL button on the console.

Set MAR to a certain address. Now, press the “Load” button on the screen. Then the MBR value is set to the field corresponding to the address in the memory.

For example, if the Memory has 0000 0000 0001 1100 at location 2, then

Enter 000000 00 00 0 00010 in the user input.

Click “LD” next to MAR

Click “Load” at the below.

Storing user input value at a specified location in the Memory manually(Functionality of Store Button):

To do so,

Run the jar, click the IPL button.

Set the MAR to the location value by clicking “LD” button next to MAR after inputting the address.

Set the MBR with the value you want to store and the click “LD” button next to MBR.

Click the Store button in the below. Look in the debugger console to see the operation performed.

Verify by setting the MAR to some other valid address press “Load”(Just to change the MBR value). And then set MAR to the specified location and click “Load” below. You can see the MBR value is populated with the user given address.

Running the Input file :

Run the jar file and make sure you click the IPL button.

The PC value is set to 0000 0000 0000 by default. And our start address is also 0000000000000000.

Enter 000000 00 00 0 00000, click “LD” next to PC.

Pressing Run Button : If you would like to run program wholly, Press “Run” button.

Expected output:

The program execute each of the instruction present in the memory. You can eventually see the debugger console to know the performed actions.

At last , the window pop ups showing “HALT Triggered. Program is halted. End of the Program” or similar text. It means that program is executed fully.

Pressing the Single Step Button : If you want to perform actions instruction by instruction.

You just need to click “Step” button continuously to run the program till you see the HALT window.

Detailed description to run the input file is given as follows.

Instructions to run the input file :

step 1 : Loading the "ProgramFileLoad.txt" input file
Press IPL button on the screen

step 2: Setting the PC to start address
Enter 000000 00 00 0 00000 in the input
Press LD button next to PC

step 3: Run the program instruction by instruction

Press Step button on the screen
Instruction executed :
(0510)16 = (000001 01 00 0 10000)2
LDR GPR2,IXR0,EA[(10000)2]

Press Step button on the screen
Instruction executed :
(A451)16 = (101001 00 01 0 10001)2
LDX GPR0,IXR1,EA[(10001)2]

Press Step button on the screen
Instruction executed :
(0F16)16 = (000011 11 00 0 10110)2
LDA GPR3,IXR0,EA[(10110)2]

Press Step button on the screen
Instruction executed :
(0910)16 = (000010 01 00 0 10000)2
STR GPR2,IXR0,EA[(10000)2]

Enter the value 000001 00 00 0 000011 in the input
Press LD button next to IXR1
Press Step button on the screen
Instruction executed :
(A891)16 = (101010 00 01 0 10001)2
STX GPR0,IXR1,EA[(100011)2]

Press Step button on the screen
Instruction executed with indexing
(A4BD)16 = (101001 00 10 1 11101)2
LDX GPR0,IXR3,EA[(11101)2]

Press Step button on the screen
(0B24)16 = (000010 11 00 1 00100)2

STR GPR3,IXR0,EA[(00100)2]

Press Step button on the screen
(0501)16 = (000001 01 00 0 00001)2
LDR GPR1,IXR0,EA[(00001)2]

Press Step button on the screen
(0B03)16 = (000010 11 00 0 00011)2
STR GPR3,IXR0,EA[(00011)2]

Press Step button on the screen
(0404)16 = (000001 00 00 0 00100)2
LDR GPR0,IXR0,EA[(00100)2]

Press Step button on the screen
(0904)16 = (000010 01 00 0 00100)2
STR GPR1,IXR0,EA[(00100)2]

Press Step button on the screen
Instruction executed with indexing
(0481)16 = (000001 00 10 0 00001)2
LDR GPR0,IXR3,EA[(00001)2]

Press Step button on the screen
(0001)16 = (000000 00 00 0 00001)2
HALT GPR0,IXR0,EA[(00001)2]

***** Program HALTs here The rest will be executing in mid of the program*****

(0401)16 = (000001 00 00 0 00001)2
LDR GPR0,IXR0,EA[(00001)2]

(0904)16 = (000010 01 00 0 00100)2
STR GPR1,IXR0,EA[(00100)2]

(0510)16 = (000001 01 00 0 10000)2

LDR GPR1,IXR0,EA[(10000)2]

Instruction executed with indexing

(A451)16 = (101001 00 01 0 10001)2

LDX GPR0,IXR1,EA[(10001)2]

Exceptions :

The program is designed to show the exceptions that you may encounter during execution of the program.

A pop up window is shown with the exception message in the window.

Try giving empty input and click on any LD, Load, Store buttons to see the exception.

Design Notes of the Code :

The program's entry point is in FrontPanel.java

FrontPanel.java : is the class where we declare, define components like buttons, labels, text inputs. We set default actions for the buttons, default text to labels, positioning the components. We also set values to the registers and get values from memory and load registers.

We also show exception pop ups from this file.

BinaryOperations.java : In this file we perform binary operations like converting hexadecimal to binary, decimal to binary, binary addition.

This file follows singleton pattern.

Constants.java : In this file, we deal with setting default values that we will be using frequently to set few registers.

FileHandler.java : FileHandler class is there for to perform different file operations like loading the file, checking if the file exists, reading from the file and converting the hexadecimal to binary and store it into the memory.

Instructionword.java : In this file, we define Instruction word components; opcode, gpr, ixr, indirect addressing, address. We also identify opcodes, gpr/ixr registers in the instruction word.

Memory.java : One of the key class, where we store our program as a instance which was loaded from the input file. We also perform Get from memory and store into memory in this class.

Opcodes.java : In this enum we define several opcodes that we use during our program.

OutputConsole.java : This is the class responsible to show the debugger console.

PopUps.java : This file is responsible for showing messages or errors in a new popup window.

Registers.java : This enum file is to identify various registers in our program.

Simulator.java : In this file, we perform fetch execution cycle. We decode instruction perform the instruction as it dictates, sets few registers as per the instructions or load/store memory into, from respectively.

UserInputReader.java : In this file, we interpret the user input and validate with the different conditions, like checking if all the input is in binary string, whether the user had given exact number of input in the text field as it meant to be.

UtilClass.java : In this class we define few methods that we use frequently during our program execution like getting string in a certain format, appending string with zeroes at beginning.

***** End Of File *****