HOW TO USE:

PLEASE READ THE README.TXT

HOW TO RUN (STEP BY STEP)

Always run commands from inside the automatedTesting/ folder.

Linux:
```
 cd automatedTesting
```
Windows:

 (open terminal inside automatedTesting)

A) Run ONLY assembler tests

Linux:
```
  python3 src/main.py --no-sim --linux
```
Windows:
```
 python3 src\main.py --no-sim --windows
```
B) Run ONLY simulator tests

Linux:
```
 python3 src/main.py --no-asm --linux
```
Windows:
```
 python3 src\main.py --no-asm --windows
```
C) Run BOTH assembler and simulator tests

Linux:
```
 python3 src/main.py --linux
```
Windows:
```
 python3 src\main.py --windows
```

Guide to the automatic infrastructure:

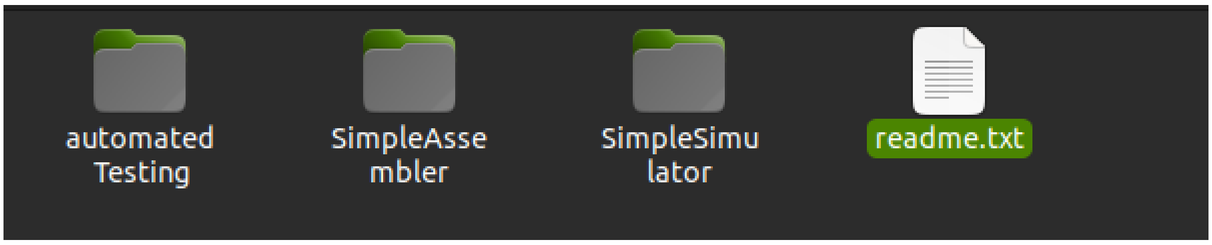
This project is an automated grading framework for a simplified RISC-V–like assembler and simulator. The Students should implement their own Assembler and Simulator. The framework runs the student code on predefined test cases. Outputs produced by the student programs are compared against reference ("golden") outputs. Marks are assigned based on the number of tests passed. Make sure the name of the Assembler and Simulator as **"Assembler.py"** and **"Simulator.py".**

Overall Flow:

1. Test inputs are provided by the framework.
2. The student code is executed automatically with these inputs.
3. Student outputs are saved in designated user_* folders.
4. The grader compares student outputs with reference outputs.
5. Pass/fail is computed per test case.
6. Final marks are reported.

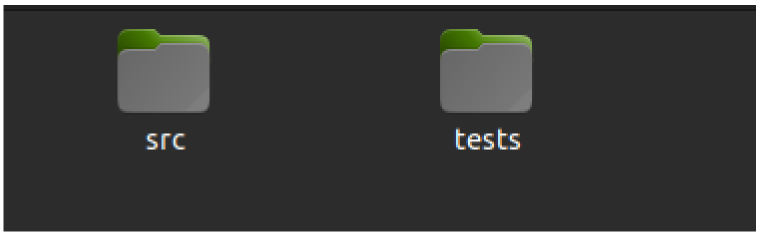
input → **execution** → **student output** → **reference output** → **comparison** → **marks**.

Folder Organization:



1. Automated Testing- contains the program that will evaluate the assembler and simulator.
2. SimpleAssembler- Place your assembler code in this folder.
3. SimpleSimulator- Place your simulator code in this folder.

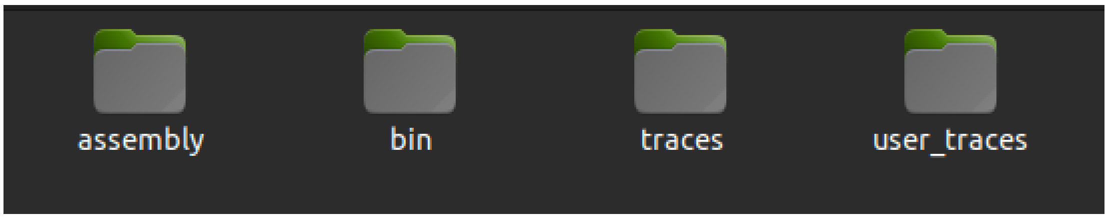
Automated Testing Folder



In this folder, you will find **src** and **tests.** The **src** contains the Python code for grading of both your assembler and simulator.

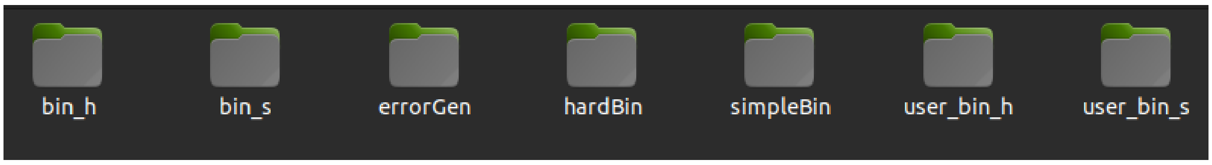
Students must not modify anything inside automatedTesting/src.

Inside the tests folder, you will find the following folders:



Assembler Grading/Evaluation:
Some predefined assembly code examples are present in **`automatedTesting/tests/assembly/simpleBin/`** these are the *txt files. These files are the assembly code that will be used by the Assembler to generate the corresponding binary code and store it in **`automatedTesting/tests/assembly/user_bin_s/`** for (simple tests). Then this code is compared with the Reference Outputs - Correct (golden) outputs that are stored in: **`automatedTesting/tests/assembly/bin_s/`** to do the grading. On the project evaluation day, you will be given a superset of test cases that you will be evaluated on.

Creating your own test case:



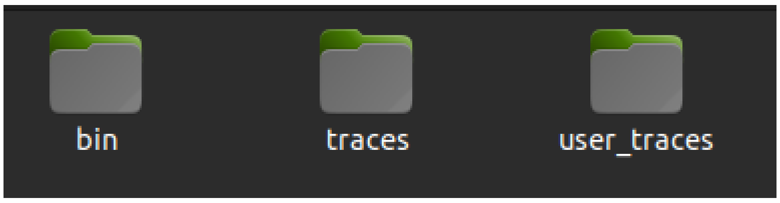
You can create your own for a testing case. Create an assembly code and put it in one of the folders.
`automatedTesting/tests/assembly/simpleBin/`
`automatedTesting/tests/assembly/hardBin/`
When you put it in one of these folders, the framework will create output in the following folders, respectively:
`automatedTesting/tests/assembly/user_bin_s/`
`automatedTesting/tests/assembly/user_bin_h/`
Make sure you design your own reference code (do it manually, since you are comparing the generated binary) and place it in the following folder.
`automatedTesting/tests/assembly/bin_s/`
`automatedTesting/tests/assembly/bin_h/`
Make sure the test case names are the same; otherwise, it will result in an error.

If you want to make sure your simulator can evaluate illegal cases, put it in
`automatedTesting/tests/assembly/errorGen/`
Here, there will be no comparison.

Simulator Grading/Evaluation

The simulator is evaluated in a manner similar to the assembler, but instead of assembly code, it operates on binary (machine-code) input files and produces execution traces. Some predefined binary (machine-code) test cases are present in
`automatedTesting/tests/bin/simple/` — these are the *txt files.
These files are the binary programs that will be used by the Simulator to generate the corresponding execution trace and store it in
`automatedTesting/tests/user_traces/simple/` (for simple tests). Then this output is compared with the Reference Outputs (correct "golden" traces) that are stored in **`automatedTesting/tests/traces/simple/`** to perform grading. On the project evaluation day, you will be given a superset of test cases that you will be evaluated on.

Creating your own test case:



You can create your own test case for testing your simulator. Create a binary (machine-code) file and put it in one of the following folders:
`automatedTesting/tests/bin/simple/`
`automatedTesting/tests/bin/hard/`
When you put it in one of these folders, the framework will create output in the following folders, respectively:
`automatedTesting/tests/user_traces/simple/`
`automatedTesting/tests/user_traces/hard/`

Make sure you design your own reference trace (do it manually, since you are comparing the simulator output) and place it in the following folder:
`automatedTesting/tests/traces/simple/`
`automatedTesting/tests/traces/hard/`
Make sure the test case names are exactly the same; otherwise, it will result in an error during comparison.

File tree:
```
├── automatedTesting
│   ├── src
│   │   ├── AsmGrader.py
│   │   ├── colors.py
│   │   ├── Grader.py
│   │   ├── main.py
│   │   ├── __pycache__
│   │   │   ├── AsmGrader.cpython-310.pyc
│   │   │   ├── AsmGrader.cpython-38.pyc
│   │   │   ├── colors.cpython-310.pyc
│   │   │   ├── colors.cpython-38.pyc
│   │   │   ├── Grader.cpython-310.pyc
│   │   │   ├── Grader.cpython-38.pyc
│   │   │   ├── Results.cpython-310.pyc
│   │   │   ├── Results.cpython-38.pyc
│   │   │   ├── SimGrader.cpython-310.pyc
│   │   │   └── SimGrader.cpython-38.pyc
│   │   ├── Results.py
│   │   └── SimGrader.py
│   └── tests
│       ├── assembly
│       │   ├── bin_h
│       │   ├── bin_s
│       │   │   ├── test_case_1.txt
│       │   │   ├── test_case_2.txt
│       │   │   ├── test_case_3.txt
│       │   │   └── test_case_4.txt
│       │   ├── errorGen
│       │   │   └── error.txt
│       │   ├── hardBin
│       │   ├── simpleBin
│       │   │   ├── test_case_1.txt
│       │   │   ├── test_case_2.txt
│       │   │   ├── test_case_3.txt
│       │   │   └── test_case_4.txt
│       │   ├── user_bin_h
│       │   └── user_bin_s
│       ├── bin
│       │   ├── hard
│       │   │   ├── hard_1.txt
│       │   │   ├── hard_2.txt
│       │   │   ├── hard_3.txt
│       │   │   ├── hard_4.txt
│       │   │   └── hard_5.txt
│       │   └── simple
│       │       ├── simple_1.txt
│       │       ├── simple_2.txt
│       │       ├── simple_3.txt
│       │       ├── simple_4.txt
│       │       ├── simple_5.txt
│       │       ├── test_case_1_asm.txt
│       │       ├── test_case_1.txt
│       │       └── test_case_2.txt
│       ├── traces
│       │   ├── hard
│       │   │   ├── hard_1.txt
│       │   │   ├── hard_2.txt
│       │   │   ├── hard_3.txt
│       │   │   ├── hard_4.txt
│       │   │   └── hard_5.txt
│       │   └── simple
│       │       ├── simple_1.txt
│       │       ├── simple_2.txt
│       │       ├── simple_3.txt
│       │       ├── simple_4.txt
│       │       ├── simple_5.txt
│       │       ├── test_case_1_asm.txt
│       │       ├── test_case_1.txt
│       │       └── test_case_2.txt
│       └── user_traces
│           ├── hard
│           └── simple
├── readme.txt
├── SimpleAssembler
│   └── readme.txt
└── SimpleSimulator
    └── readme.txt
```

NB. All the *.txt files, except the readme.txt, are the examples