Tomasulo-Softcore Library

Yihao Liu & Stephen Longofono

December 13, 2018

Introduction

This document outlines the Tomasulo-Softcore library, a Python library written to model a single-core software CPU based on Tomasulo's Algorithm. The source code is presented as an open-source library, and is available on Github.com under the MIT license. This library was developed as an academic exercise for the graduate-level computer architecture course at the University of Pittsburgh in Pennsylvania, U.S.A.

The library demonstrates all of the major facets of the algorithm, including register renaming, out-of-order execution, memory disambiguation, speculative execution.

Please reference the Github repository for the most up-to-date code, instructions, and test cases. The repository is hosted at:
https://www.github.com/SLongofono/Tomasulo-Softcore.git/

1 Building & Execution

1.1 System Requirements

The library was developed for Python 3.6+. No external libraries were used to develop the core library. The project was developed and tested in a Linux environment, in a directory with write and execute privileges. Note that the output files will be placed in the same directory as the input files passed in; in order to correctly generate an output file, the user must have write privileges to the directory with the input files.

Included in the repository is a shell script used to generate UML diagrams from the class hierarchy. If you choose to modify or extend the library, the script can be run again from the root directory of the repository to generate fresh images of the UML class diagrams. Running this script requires Pylint 2.2.1+, but is not otherwise necessary for the use of the core library.

1.2 Build Instructions

There are no build instructions for the library, as it is a standalone Python module. The entry point to execution is Tomasulo.py. To start the library and print usage instructions, simply invoke your Python interpreter with Tomasulo.py as an argument from the command line.

1.3 Running Test Inputs

All test inputs are passed as command line arguments. The test inputs all follow a rigid and ordered format, as demonstrated by the file "example_input.txt" in the "testCases/" directory of the repository root. For convenience, all of our test cases are located in this "textCases/" directory. To execute the library on a test input file, type:

```
<Python3.6> Tomasulo.py <filepath>
```

where Python3.6 is the CLI invoker for your Python 3 interpreter, and filepath is the absolute filepath of the desired test input.

There is also a test script which will run all tests and report the results using diff. Navigate to the testCases directory and run it with the shell:

```
./runTests.sh
```

1.4 Defining System Inputs

The system inputs follow a rigid and specifically ordered format. All inputs should be UTF-8 compliant, printable text format files. The specifications are presented in order below:

- The first line of the file is ignored. In our sample inputs, the first line is used to label the table columns that the following 4 lines compose.
- The next four lines begin with a label, followed by a space, and then 4 entries of either a natural number or a space (only permissible in the third position). The labels are "Integer adder", "FP adder", "FP multiplier", and Load/store unit".
- The next line begins with the label "ROB entries = ", followed by a natural number.
- The next line begins with "CDB buffer entries = ", followed by a natural number.
- The next line is optional. It consists of comma-delineated patterns of the form A=B. The pattern A is the letter "R" or the letter "F" concatenated with exactly one of the numbers 0-31. The pattern B is an integer value which can be represented in 32 bits.
- The next line is optional. It consists of comma-delineated patterns of the form Meme[A]=B. The pattern A is a single natural number between 0 and 255 inclusive. The pattern B is an integer or floating point value which can be represented in 64 bits.
- The next line is blank. This line is not optional.
- The remainder of the lines in the file are instructions. All instructions are represented as shown in the rubric. Replace immediate values and offsets by their integer equivalents. In the case of a branch instruction, express the offset in terms of instructions to move relative to PC-next.
- The last line of the file should be the last instruction.

1.5 System Outputs

The library will output execution information in two ways; during execution, progress and the contents of various system components are displayed to **stdout**. Upon completion, an output file will be written to the directory containing the input file. The output file will be the same name as the input file, with "_output" appended just before the file extension.

2 Verification & Testing

In order to verify a proper and complete implementation of Tomasulo's algorithm, we developed a series of progressively more complex test inputs. All of the test cases used are provided in the repository under the "testCases/" directory. For convenience, they are repeated here along with a short description of which feature they verify.

2.1 simple1.txt

This test was designed to test a pair of independent integer ALU instructions, as a verification that our system could correctly process both types of Integer ALU instructions in isolation. It also serves as a verification of the general flow of the program, excluding the memory stage.

Input

```
  # of rs Cycles in EX  Cycles in Mem # of FUs
Integer adder 2 1   1
FP adder  3 3   1
FP multiplier 2 20    1
Load/store unit 3 1 4 1
ROB entries = 128
CDB buffer entries = 1
R1=10,R2=20,R3=30,F2=30.1
Mem[4]=1,Mem[8]=2,Mem[12]=3.4

ADD R1,R2,R3
SUB R4,R2,R3
```

Output

```
================================Instruction Completion Table====================
ID | IS  EX  MEM  WB  COM
0 | 0  1  None  2  3
1 | 1  2  None  3  4
2 | 3  4  None  5  6

================================Integer ARF=====================================
R0 : 0              R1 : 50             R2 : 20             R3 : 30
R4 : -10            R5 : 99             R6 : 0              R7 : 0
R8 : 0              R9 : 0              R10: 0              R11: 0
R12: 0              R13: 0              R14: 0              R15: 0
R16: 0              R17: 0              R18: 0              R19: 0
R20: 0              R21: 0              R22: 0              R23: 0
R24: 0              R25: 0              R26: 0              R27: 0
R28: 0              R29: 0              R30: 0              R31: 0

================================Floating Point ARF==============================
F0 : 0.000000                           F1 : 0.000000
F2 : 30.100000                          F3 : 0.000000
F4 : 0.000000                           F5 : 0.000000
F6 : 0.000000                           F7 : 0.000000
F8 : 0.000000                           F9 : 0.000000
F10: 0.000000                           F11: 0.000000
F12: 0.000000                           F13: 0.000000
F14: 0.000000                           F15: 0.000000
F16: 0.000000                           F17: 0.000000
F18: 0.000000                           F19: 0.000000
F20: 0.000000                           F21: 0.000000
F22: 0.000000                           F23: 0.000000
F24: 0.000000                           F25: 0.000000
F26: 0.000000                           F27: 0.000000
```

```
F28: 0.000000                          F29: 0.000000
F30: 0.000000                          F31: 0.000000


================================Memory Unit=====================================
Word 01: 1.000000                      Word 02: 2.000000
Word 03: 3.400000
```

2.2 simple2.txt

This test was designed to test a pair of independent floating point ALU instructions, as a verification that our system could correctly process both types of floating point ALU instructions in isolation. It also serves as a verification of the general flow of the program, excluding the memory stage.

Input

```
  # of rs Cycles in EX  Cycles in Mem # of FUs
Integer adder 2 1   1
FP adder  3 3   2
FP multiplier 2 20    1
Load/store unit 3 1 4 1
ROB entries = 128
CDB buffer entries = 1
F1=10.10,F2=20.20,F3=30.30,F4=30.1

ADD.D F1,F2,F3
SUB.D F4,F2,F3
```

Output

```
================================Instruction Completion Table====================
ID | IS  EX  MEM  WB  COM
0 | 0  1  None  4  5
1 | 1  2  None  5  6

================================Integer ARF=====================================
R0 : 0              R1 : 0              R2 : 0              R3 : 0
R4 : 0              R5 : 0              R6 : 0              R7 : 0
R8 : 0              R9 : 0              R10: 0              R11: 0
R12: 0              R13: 0              R14: 0              R15: 0
R16: 0              R17: 0              R18: 0              R19: 0
R20: 0              R21: 0              R22: 0              R23: 0
R24: 0              R25: 0              R26: 0              R27: 0
R28: 0              R29: 0              R30: 0              R31: 0

================================Floating Point ARF==============================
F0 : 0.000000                          F1 : 50.500000
F2 : 20.200000                         F3 : 30.300000
F4 : -10.100000                        F5 : 0.000000
F6 : 0.000000                          F7 : 0.000000
F8 : 0.000000                          F9 : 0.000000
F10: 0.000000                          F11: 0.000000
F12: 0.000000                          F13: 0.000000
F14: 0.000000                          F15: 0.000000
```

```
F16: 0.000000                          F17: 0.000000
F18: 0.000000                          F19: 0.000000
F20: 0.000000                          F21: 0.000000
F22: 0.000000                          F23: 0.000000
F24: 0.000000                          F25: 0.000000
F26: 0.000000                          F27: 0.000000
F28: 0.000000                          F29: 0.000000
F30: 0.000000                          F31: 0.000000


================================Memory Unit=====================================
```

2.3 simple3.txt

This test was designed to test a pair of independent floating point multiplier instructions, as a verification that our system could correctly process floating point multiplier instructions in isolation.

Input

```
  # of rs Cycles in EX  Cycles in Mem # of FUs
Integer adder 2 1   1
FP adder  3 3   1
FP multiplier 2 20    2
Load/store unit 3 1 4 1
ROB entries = 128
CDB buffer entries = 1
F1=0.2345,F2=30.1,F3=3.14159,F4=-999.999,F5=2000

MULT.D F1,F2,F3
MULT.D F4,F2,F5
```

Output

```
================================Instruction Completion Table====================
ID | IS  EX  MEM  WB  COM
0 | 0  1  None  21  22
1 | 1  2  None  22  23

================================Integer ARF=====================================
R0 : 0              R1 : 0              R2 : 0              R3 : 0
R4 : 0              R5 : 0              R6 : 0              R7 : 0
R8 : 0              R9 : 0              R10: 0              R11: 0
R12: 0              R13: 0              R14: 0              R15: 0
R16: 0              R17: 0              R18: 0              R19: 0
R20: 0              R21: 0              R22: 0              R23: 0
R24: 0              R25: 0              R26: 0              R27: 0
R28: 0              R29: 0              R30: 0              R31: 0

================================Floating Point ARF==============================
F0 : 0.000000                          F1 : 94.561859
F2 : 30.100000                         F3 : 3.141590
F4 : 60200.000000                      F5 : 2000.000000
F6 : 0.000000                          F7 : 0.000000
```

```
F8 : 0.000000                          F9 : 0.000000
F10: 0.000000                          F11: 0.000000
F12: 0.000000                          F13: 0.000000
F14: 0.000000                          F15: 0.000000
F16: 0.000000                          F17: 0.000000
F18: 0.000000                          F19: 0.000000
F20: 0.000000                          F21: 0.000000
F22: 0.000000                          F23: 0.000000
F24: 0.000000                          F25: 0.000000
F26: 0.000000                          F27: 0.000000
F28: 0.000000                          F29: 0.000000
F30: 0.000000                          F31: 0.000000


================================Memory Unit=====================================
```

2.4 simple4.txt

This test case was designed to test the load and store instructions in isolation. It also serves as a verification of the general flow of the program, excluding the branch unit.

Input

```
  # of rs Cycles in EX  Cycles in Mem # of FUs
Integer adder 2 1   1
FP adder  3 3   1
FP multiplier 2 20    1
Load/store unit 3 1 4 1
ROB entries = 128
CDB buffer entries = 1
R1=10,R2=20,R3=30,F2=30.1
Mem[4]=1,Mem[8]=2,Mem[12]=3.4

SD F2, 8(R0)
LD F1, 4(R0)
```

Output

```
================================Instruction Completion Table====================
ID | IS  EX  MEM  WB  COM
0 | 0  1  None  None  2
1 | 1  2  6  10  11

================================Integer ARF=====================================
R0 : 0              R1 : 10             R2 : 20             R3 : 30
R4 : 0              R5 : 0              R6 : 0              R7 : 0
R8 : 0              R9 : 0              R10: 0              R11: 0
R12: 0              R13: 0              R14: 0              R15: 0
R16: 0              R17: 0              R18: 0              R19: 0
R20: 0              R21: 0              R22: 0              R23: 0
R24: 0              R25: 0              R26: 0              R27: 0
R28: 0              R29: 0              R30: 0              R31: 0
```

```
================================Floating Point ARF================================
F0 : 0.000000                              F1 : 1.000000
F2 : 30.100000                             F3 : 0.000000
F4 : 0.000000                              F5 : 0.000000
F6 : 0.000000                              F7 : 0.000000
F8 : 0.000000                              F9 : 0.000000
F10: 0.000000                              F11: 0.000000
F12: 0.000000                              F13: 0.000000
F14: 0.000000                              F15: 0.000000
F16: 0.000000                              F17: 0.000000
F18: 0.000000                              F19: 0.000000
F20: 0.000000                              F21: 0.000000
F22: 0.000000                              F23: 0.000000
F24: 0.000000                              F25: 0.000000
F26: 0.000000                              F27: 0.000000
F28: 0.000000                              F29: 0.000000
F30: 0.000000                              F31: 0.000000


================================Memory Unit=====================================
Word 01: 1.000000                          Word 02: 30.100000
Word 03: 3.400000
```

2.5 simple5.txt

This test case was designed to test the branch unit with a single branch instruction. It serves to test the simplest functions of the branch unit, including storing copies of the RAT and correct branch prediction.

Input

```
  # of rs Cycles in EX  Cycles in Mem # of FUs
Integer adder 2 1   1
FP adder  3 3   1
FP multiplier 2 20    1
Load/store unit 3 1 4 1
ROB entries = 16
CDB buffer entries = 1
R1=10,R2=20,R3=30,R4=20,F2=30.1
Mem[4]=1,Mem[8]=2,Mem[12]=3.4

SUB R4,R4,R2
BNE R4,R0,-2
```

Output

```
================================Instruction Completion Table====================
ID | IS  EX  MEM  WB  COM
0 | 0  1  None  2  3
1 | 1  3  None  None  4

================================Integer ARF=====================================
R0 : 0              R1 : 10             R2 : 20             R3 : 30
R4 : 0              R5 : 0              R6 : 0              R7 : 0
R8 : 0              R9 : 0              R10: 0              R11: 0
```

```
R12: 0              R13: 0              R14: 0              R15: 0
R16: 0              R17: 0              R18: 0              R19: 0
R20: 0              R21: 0              R22: 0              R23: 0
R24: 0              R25: 0              R26: 0              R27: 0
R28: 0              R29: 0              R30: 0              R31: 0

================================Floating Point ARF==============================
F0 : 0.000000                           F1 : 0.000000
F2 : 30.100000                          F3 : 0.000000
F4 : 0.000000                           F5 : 0.000000
F6 : 0.000000                           F7 : 0.000000
F8 : 0.000000                           F9 : 0.000000
F10: 0.000000                           F11: 0.000000
F12: 0.000000                           F13: 0.000000
F14: 0.000000                           F15: 0.000000
F16: 0.000000                           F17: 0.000000
F18: 0.000000                           F19: 0.000000
F20: 0.000000                           F21: 0.000000
F22: 0.000000                           F23: 0.000000
F24: 0.000000                           F25: 0.000000
F26: 0.000000                           F27: 0.000000
F28: 0.000000                           F29: 0.000000
F30: 0.000000                           F31: 0.000000


================================Memory Unit=====================================
Word 01: 1.000000                       Word 02: 2.000000
Word 03: 3.400000
```

2.6 simple6.txt

This test case was designed to test corner cases of the system configuration. This test will issue instructions to saturate the capacity of all functional units by filling the output buffers and the reservation stations. The goal is to prove that the library can make forward progress in the face of a demanding load.

Input

```
  # of rs Cycles in EX  Cycles in Mem # of FUs
Integer adder 1 1   1
FP adder  1 4   1
FP multiplier 1 4    1
Load/store unit 1 1 4 1
ROB entries = 128
CDB buffer entries = 2
R1=1,R2=2,R3=3,F1=1.1,F2=2.1
Mem[0]=0.0,Mem[4]=1.1,Mem[8]=2.2,Mem[12]=3.3

LD F3, 0(R0)
LD F4, 4(R0)
LD F5, 8(R0)
LD F6, 12(R0)
SD F1, 0(R0)
```

```
SD F2, 4(R0)
ADDD.D F7,F3,F4
ADDD.D F8,F3,F4
MULT.D F9,F3,F4
MULT.D F10,F3,F4
SUB R9,R3,R4
SUB R10,R3,R4
```

Output

```
================================Instruction Completion Table====================
ID | IS  EX  MEM  WB  COM
0 | 0  1  2  6  7
1 | 7  8  9  13  14
2 | 14  15  16  20  21
3 | 21  22  23  27  28
4 | 28  29  None  None  30
5 | 34  35  None  None  36
6 | 35  36  None  40  41
7 | 41  42  None  46  47
8 | 42  43  None  47  48
9 | 48  49  None  53  54
10 | 49  50  None  51  55
11 | 52  53  None  54  56

================================Integer ARF=====================================
R0 : 0              R1 : 1              R2 : 2              R3 : 3
R4 : 0              R5 : 0              R6 : 0              R7 : 0
R8 : 0              R9 : 3              R10: 3              R11: 0
R12: 0              R13: 0              R14: 0              R15: 0
R16: 0              R17: 0              R18: 0              R19: 0
R20: 0              R21: 0              R22: 0              R23: 0
R24: 0              R25: 0              R26: 0              R27: 0
R28: 0              R29: 0              R30: 0              R31: 0

================================Floating Point ARF==============================
F0 : 0.000000                           F1 : 1.100000
F2 : 2.100000                           F3 : 0.000000
F4 : 1.100000                           F5 : 2.200000
F6 : 3.300000                           F7 : 1.100000
F8 : 1.100000                           F9 : 0.000000
F10: 0.000000                           F11: 0.000000
F12: 0.000000                           F13: 0.000000
F14: 0.000000                           F15: 0.000000
F16: 0.000000                           F17: 0.000000
F18: 0.000000                           F19: 0.000000
F20: 0.000000                           F21: 0.000000
F22: 0.000000                           F23: 0.000000
F24: 0.000000                           F25: 0.000000
F26: 0.000000                           F27: 0.000000
F28: 0.000000                           F29: 0.000000
F30: 0.000000                           F31: 0.000000
```

```
================================Memory Unit=====================================
Word 00: 1.100000                       Word 01: 2.100000
Word 02: 2.200000                       Word 03: 3.300000
```

2.7 simple7.txt

This test case was designed to verify the absence of WAW hazards, a behavior that the algorithm was designed to remove.

Input

```
  # of rs Cycles in EX  Cycles in Mem # of FUs
Integer adder 2 1   1
FP adder  3 3   1
FP multiplier 2 20    1
Load/store unit 3 1 4 1
ROB entries = 128
CDB buffer entries = 1
R1=10,R2=20,R3=30,F2=30.1

ADD R1,R2,R3
SUB R1,R2,R3
```

Output

```
================================Instruction Completion Table====================
ID | IS  EX  MEM  WB  COM
0 | 0  1  None  2  3
1 | 1  2  None  3  4

================================Integer ARF=====================================
R0 : 0              R1 : -10            R2 : 20             R3 : 30
R4 : 0              R5 : 0              R6 : 0              R7 : 0
R8 : 0              R9 : 0              R10: 0              R11: 0
R12: 0              R13: 0              R14: 0              R15: 0
R16: 0              R17: 0              R18: 0              R19: 0
R20: 0              R21: 0              R22: 0              R23: 0
R24: 0              R25: 0              R26: 0              R27: 0
R28: 0              R29: 0              R30: 0              R31: 0

================================Floating Point ARF==============================
F0 : 0.000000                           F1 : 0.000000
F2 : 30.100000                          F3 : 0.000000
F4 : 0.000000                           F5 : 0.000000
F6 : 0.000000                           F7 : 0.000000
F8 : 0.000000                           F9 : 0.000000
F10: 0.000000                           F11: 0.000000
F12: 0.000000                           F13: 0.000000
F14: 0.000000                           F15: 0.000000
F16: 0.000000                           F17: 0.000000
F18: 0.000000                           F19: 0.000000
F20: 0.000000                           F21: 0.000000
```

```
F22: 0.000000                          F23: 0.000000
F24: 0.000000                          F25: 0.000000
F26: 0.000000                          F27: 0.000000
F28: 0.000000                          F29: 0.000000
F30: 0.000000                          F31: 0.000000


================================Memory Unit=====================================
```

2.8 simple8.txt

This test case was designed to verify the absence of WAR hazards, a behavior that the algorithm was designed to remove.

Input

```
  # of rs Cycles in EX  Cycles in Mem # of FUs
Integer adder 4 3   3
FP adder  3 3   1
FP multiplier 2 20    1
Load/store unit 3 1 4 1
ROB entries = 128
CDB buffer entries = 1
R1=1,R2=1,R3=1,R1=2,F2=30.1
Mem[4]=1,Mem[8]=2,Mem[12]=3.4

SUB R3,R3,R1
SUB R3,R4,R1
BNE R3,R0,-2
```

Output

```
================================Instruction Completion Table====================
ID | IS  EX  MEM  WB  COM
0 | 0  1  None  4  5
1 | 1  2  None  5  6
2 | 2  6  None  None  9

================================Integer ARF=====================================
R0 : 0              R1 : 1              R2 : 1              R3 : 0
R4 : 1              R5 : 0              R6 : 0              R7 : 0
R8 : 0              R9 : 0              R10: 0              R11: 0
R12: 0              R13: 0              R14: 0              R15: 0
R16: 0              R17: 0              R18: 0              R19: 0
R20: 0              R21: 0              R22: 0              R23: 0
R24: 0              R25: 0              R26: 0              R27: 0
R28: 0              R29: 0              R30: 0              R31: 0

================================Floating Point ARF==============================
F0 : 0.000000                          F1 : 0.000000
F2 : 30.100000                         F3 : 0.000000
F4 : 0.000000                          F5 : 0.000000
```

```
F6 : 0.000000                          F7 : 0.000000
F8 : 0.000000                          F9 : 0.000000
F10: 0.000000                          F11: 0.000000
F12: 0.000000                          F13: 0.000000
F14: 0.000000                          F15: 0.000000
F16: 0.000000                          F17: 0.000000
F18: 0.000000                          F19: 0.000000
F20: 0.000000                          F21: 0.000000
F22: 0.000000                          F23: 0.000000
F24: 0.000000                          F25: 0.000000
F26: 0.000000                          F27: 0.000000
F28: 0.000000                          F29: 0.000000
F30: 0.000000                          F31: 0.000000


================================Memory Unit====================================
Word 01: 1.000000                      Word 02: 2.000000
Word 03: 3.400000
```

2.9 simple9.txt

This test case was designed to verify that the system is capable of handling RAW dependencies. Several truly dependent instructions are issued for each of the types of functional units, to test the ability to complete execution as prescribed by Tomasulo's algorithm.

Input

```
  # of rs Cycles in EX  Cycles in Mem # of FUs
Integer adder 2 1   1
FP adder  3 3   1
FP multiplier 2 20    1
Load/store unit 3 1 4 1
ROB entries = 10
CDB buffer entries = 1
R1=10,R2=20,R3=30,F2=30.1
Mem[4]=1,Mem[8]=2,Mem[12]=3.4

ADD R1,R2,R3
SUB R4,R1,R3
```

Output

```
================================Instruction Completion Table====================
ID | IS  EX  MEM  WB  COM
0 | 0  1  None  2  3
1 | 1  3  None  4  5

================================Integer ARF=====================================
R0 : 0              R1 : 50             R2 : 20             R3 : 30
R4 : 20             R5 : 0              R6 : 0              R7 : 0
R8 : 0              R9 : 0              R10: 0              R11: 0
R12: 0              R13: 0              R14: 0              R15: 0
R16: 0              R17: 0              R18: 0              R19: 0
R20: 0              R21: 0              R22: 0              R23: 0
```

```
R24: 0              R25: 0              R26: 0              R27: 0
R28: 0              R29: 0              R30: 0              R31: 0

================================Floating Point ARF==============================
F0 : 0.000000                           F1 : 0.000000
F2 : 30.100000                          F3 : 0.000000
F4 : 0.000000                           F5 : 0.000000
F6 : 0.000000                           F7 : 0.000000
F8 : 0.000000                           F9 : 0.000000
F10: 0.000000                           F11: 0.000000
F12: 0.000000                           F13: 0.000000
F14: 0.000000                           F15: 0.000000
F16: 0.000000                           F17: 0.000000
F18: 0.000000                           F19: 0.000000
F20: 0.000000                           F21: 0.000000
F22: 0.000000                           F23: 0.000000
F24: 0.000000                           F25: 0.000000
F26: 0.000000                           F27: 0.000000
F28: 0.000000                           F29: 0.000000
F30: 0.000000                           F31: 0.000000


================================Memory Unit=====================================
Word 01: 1.000000                       Word 02: 2.000000
Word 03: 3.400000
```

Output

2.10 complex6.txt

This test case was designed to test our model of system memory (RAM). Here, we write a series of patterns to memory to verify that loads and stores are executing in the correct order, and that data is written to the correct location.

Input

```
  # of rs Cycles in EX  Cycles in Mem # of FUs
Integer adder 2 1   1
FP adder  3 3   1
FP multiplier 2 20    1
Load/store unit 3 1 4 1
ROB entries = 128
CDB buffer entries = 1
R1=10,R2=16,R3=30,F1=111.111,F2=222.222
Mem[4]=1,Mem[8]=2,Mem[12]=3.4

ADD R1,R0,R0
SD F1, 0(R1)
ADDI R1,R1,4
BNE R1,R2,-3
SD F2, 0(R1)
LD F3, 0(R1)
```

```
LD F4, -8(R1)
```

Output

```
================================Instruction Completion Table====================
ID | IS  EX  MEM  WB  COM
0 | 0  1  None  2  3
1 | 1  3  4  8  9
2 | 2  3  None  4  10
3 | 3  5  None  None  11
4 | 4  5  6  10  12
5 | 5  6  None  7  13
6 | 6  8  None  None  14
7 | 7  8  9  13  15
8 | 8  9  None  11  16
9 | 9  12  None  None  17
10 | 10  12  13  17  18
11 | 12  13  None  14  19
12 | 13  15  None  None  20
13 | 17  18  19  23  24
14 | 18  19  23  27  28
15 | 19  20  27  31  32

================================Integer ARF=====================================
R0 : 0              R1 : 16             R2 : 16             R3 : 30
R4 : 0              R5 : 0              R6 : 0              R7 : 0
R8 : 0              R9 : 0              R10: 0              R11: 0
R12: 0              R13: 0              R14: 0              R15: 0
R16: 0              R17: 0              R18: 0              R19: 0
R20: 0              R21: 0              R22: 0              R23: 0
R24: 0              R25: 0              R26: 0              R27: 0
R28: 0              R29: 0              R30: 0              R31: 0

================================Floating Point ARF==============================
F0 : 0.000000                           F1 : 111.1100
F2 : 222.2220                           F3 : 111.1110
F4 : 111.1110                           F5 : 0.000000
F6 : 0.000000                           F7 : 0.000000
F8 : 0.000000                           F9 : 0.000000
F10: 0.000000                           F11: 0.000000
F12: 0.000000                           F13: 0.000000
F14: 0.000000                           F15: 0.000000
F16: 0.000000                           F17: 0.000000
F18: 0.000000                           F19: 0.000000
F20: 0.000000                           F21: 0.000000
F22: 0.000000                           F23: 0.000000
F24: 0.000000                           F25: 0.000000
F26: 0.000000                           F27: 0.000000
F28: 0.000000                           F29: 0.000000
F30: 0.000000                           F31: 0.000000


================================Memory Unit=====================================
```

```
Word 01: 111.1110                    Word 02: 111.1110
Word 03: 111.1110                    Word 04: 111.1110
Word 05: 111.1110
```

2.11 complex7.txt

This final test case was designed to show that our system can do real work. We execute a sum over some data in memory to induce a chain of dependencies and memory latencies.

Input

```
  # of rs Cycles in EX  Cycles in Mem # of FUs
Integer adder 2 1   1
FP adder  3 3   1
FP multiplier 2 20    1
Load/store unit 3 1 4 1
ROB entries = 128
CDB buffer entries = 1
R1=4,R2=20,R3=16,F2=30.1
Mem[0]=1.01,Mem[4]=2.01,Mem[8]=3.01,Mem[12]=4.01

ADD.D F2,F0,F0
ADD R2,R0,R0
LD F1,0(R2)
ADDI R2,R2,4
ADDI R1,R1,-1
ADD.D F2,F2,F1
BNE R0,R1,-5
SD F2,0(R3)
```

Output

```
================================Instruction Completion Table====================
ID | IS  EX  MEM  WB  COM
0 | 0  1  None  4  5
1 | 1  2  None  3  6
2 | 2  3  4  8  9
3 | 3  4  None  5  10
4 | 4  5  None  6  11
5 | 5  9  None  12  13
6 | 6  7  None  None  14
7 | 7  8  9  13  15
8 | 8  9  None  10  16
9 | 9  10  None  11  17
10 | 10  14  None  17  18
11 | 11  12  None  None  19
12 | 12  13  14  18  20
13 | 13  14  None  15  21
14 | 14  15  None  16  22
15 | 15  19  None  22  23
16 | 16  17  None  None  24
17 | 17  18  19  22  25
18 | 18  19  None  20  26
19 | 19  20  None  21  27
```

```
20 | 20  24  None  27  28
21 | 21  22  None  None  29
22      | 24              28              29              33              34

================================Integer ARF====================================
R0 : 0              R1 : 0              R2 : 16             R3 : 16
R4 : 0              R5 : 0              R6 : 0              R7 : 0
R8 : 0              R9 : 0              R10: 0              R11: 0
R12: 0              R13: 0              R14: 0              R15: 0
R16: 0              R17: 0              R18: 0              R19: 0
R20: 0              R21: 0              R22: 0              R23: 0
R24: 0              R25: 0              R26: 0              R27: 0
R28: 0              R29: 0              R30: 0              R31: 0

==============================Floating Point ARF===============================
F0 : 0.000000                           F1 : 4.010000
F2 : 10.40000                           F3 : 0.000000
F4 : 0.000000                           F5 : 0.000000
F6 : 0.000000                           F7 : 0.000000
F8 : 0.000000                           F9 : 0.000000
F10: 0.000000                           F11: 0.000000
F12: 0.000000                           F13: 0.000000
F14: 0.000000                           F15: 0.000000
F16: 0.000000                           F17: 0.000000
F18: 0.000000                           F19: 0.000000
F20: 0.000000                           F21: 0.000000
F22: 0.000000                           F23: 0.000000
F24: 0.000000                           F25: 0.000000
F26: 0.000000                           F27: 0.000000
F28: 0.000000                           F29: 0.000000
F30: 0.000000                           F31: 0.000000


================================Memory Unit====================================
Word 00: 1.010000                       Word 01: 2.010000
Word 02: 3.010000                       Word 03: 4.010000
Word 04: 10.40000
```

3 Division of Labor

3.1 Responsibilities

We developed our code and test cases together. Stephen was more experienced with Python, so for productivity's sake he wrote much of the code. Yihao helped by developing the general flow of the program, writing effective test cases, identifying areas for improvement in the design, and general debugging.

3.2 Code Contribution

Yihao developed the floating-point functional units and the memory unit. Stephen developed the other main components of the library. Stephen wrote the first pass of the five execution stages, and then worked with Yihao to debug them as each test case was visited. More specific details on the day-to-day code contributions are available in the commit history of the Github repository.

3.3 Test Contribution

Stephen wrote the simpler test cases as a litmus test for the first pass of the data path. Stephen helped come up with the ideas for the more exhaustive tests. Yihao wrote the bulk of the test input files. Both team members completed all tests by hand to verify that our expected results were consistent.

3.4 Meetings & Participation

Since there were only two members on our team, it was easy to work together and hold productive meetings. We spent several weekends coding side-by-side, and maintained effective communication throughout the project.