

The School of Engineering and Applied Science

of The George Washington University

CSCI 6461

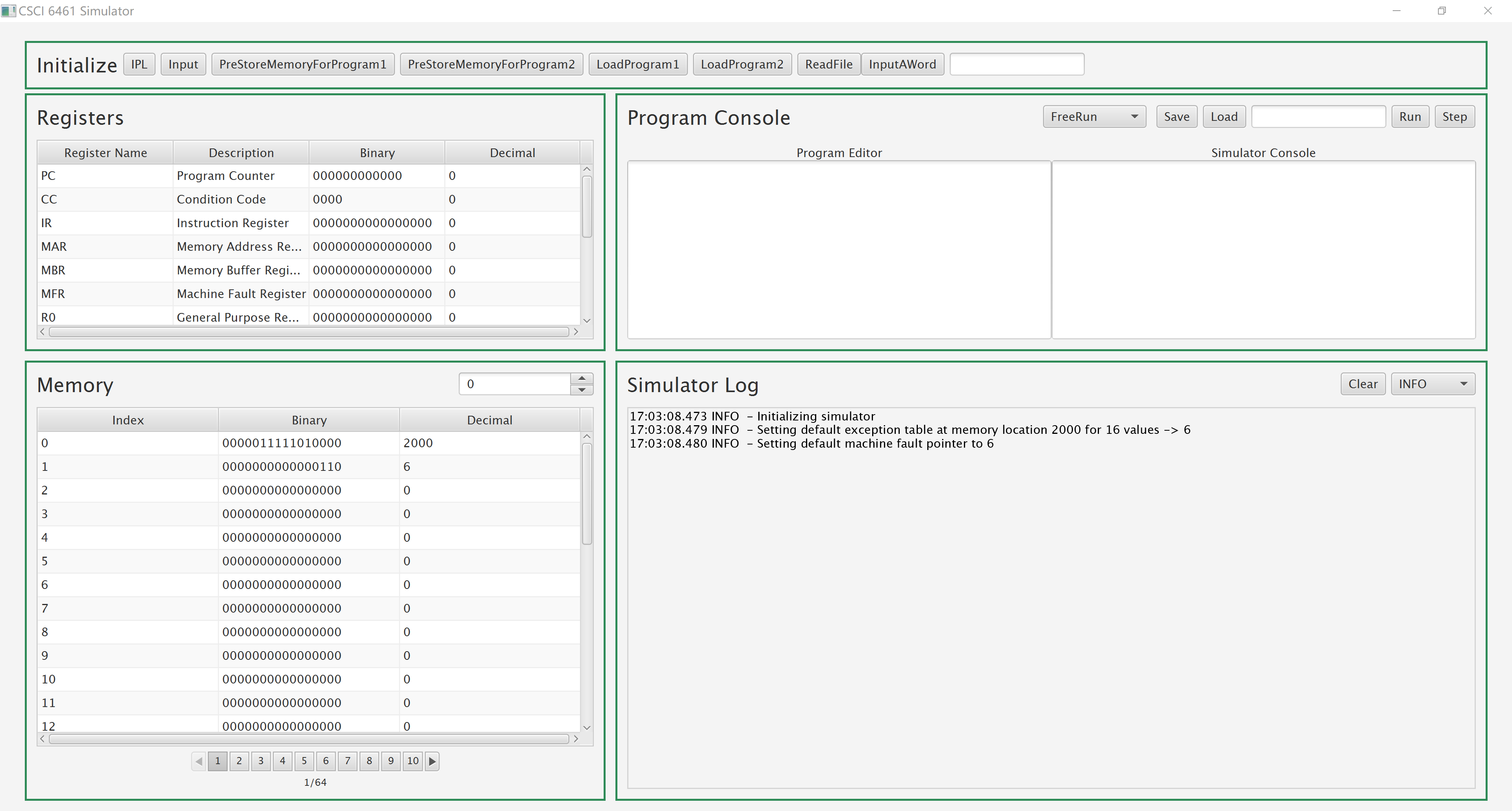
Fall 2018

Project 1, Group 9

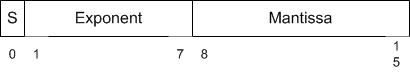
Floating Point and Vector Operations

|  |  |  |
| --- | --- | --- |
| Michael Frank | m.afrank@yahoo.com | Team Leader |
| Chuchu Jin | mayflywing@gwmail.gwu.edu | Programmer |
| Minghui Xu | mhxu@gwmail.gwu.edu | Programmer |
| Naim Merheb | naim@gwmail.gwu.edu | Documentation |

# Design Note



## Floating Point Number



Floating point number = (-1)^S \* (1.M) \* 2^(exponent-bias)

bias = 2^(exponent\_bits -1) -1 = 2^6 – 1 =63

For example,

0 1000001 01100000

S = 0, 1+M = 1.375, exponent - bias = 65 - 63 = 2

Value = (-1)^0 \* 1.375 \* 2^2 = 5.5

## Pipeline

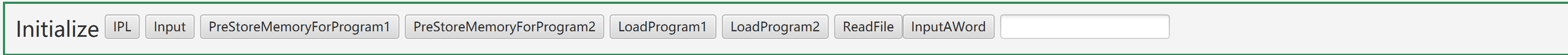
Floating point addition and subtraction is divided into four steps:

1. Match: Shift the decimal point of the smaller number to the left until the exponents are equal.
2. Calculate: Add the numbers with decimal points aligned.
3. Normalize: Normalize the result to normal form.
4. Result: Get the result.

Here is an example to execute six floating point addition/subtraction operations in pipeline:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Result |  |  |  | R1 | R2 | R3 | R4 | R5 | R6 |
| Normalize |  |  | N1 | N2 | N3 | N4 | N5 | N6 |  |
| Calculate |  | C1 | C2 | C3 | C4 | C5 | C6 |  |  |
| Match | M1 | M2 | M3 | M4 | M5 | M6 |  |  |  |
| Clock | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

## Initialize Button



**Button [IPL]**

By Clicking IPL, registers, memory and CPU will be set initially.

**Button [Input]**

By Clicking Input, the number in the Simulator Console will be input to CPU.

**Button [PreStoreMemoryForProgram1]**

By Clicking PreStoreMemoryForProgram1, it will store some values into memory, which helps run the program1:

|  |  |  |  |
| --- | --- | --- | --- |
| memory[8] | 0000000001000000 | 64 | for X1 |
| memory[9] | 0000000010101010 | 170 | for X2 |
| memory[85] | 0111111111111111 | 32767 | for compare |
| memory[86] | 0000000001000000 | 64 |  |
| memory[87] | 0000000001010100 | 84 |  |

**Button [PreStoreMemoryForProgram2]**

By Clicking PreStoreMemoryForProgram2, it will store some values into memory, which helps run the program2:

|  |  |  |  |
| --- | --- | --- | --- |
| memory[12] | 0000001000000000 | 512 | Word store start |
| memory[13] | 0000000001000000 | 64 | for X1 |
| memory[14] | 0000000001100000 | 96 | for X2 |
| memory[15] | 0000000000000001 | 1 | sentence number |
| memory[16] | 0000000000000001 | 1 | word number |

**Button [LoadProgram1]**

That will load instructions used in program1and store them into memory[126-185]. Set the PC =126.

**Button [LoadProgram2]**

That will load instructions used in program2 and store them into memory[64-10]. Set the PC =64.

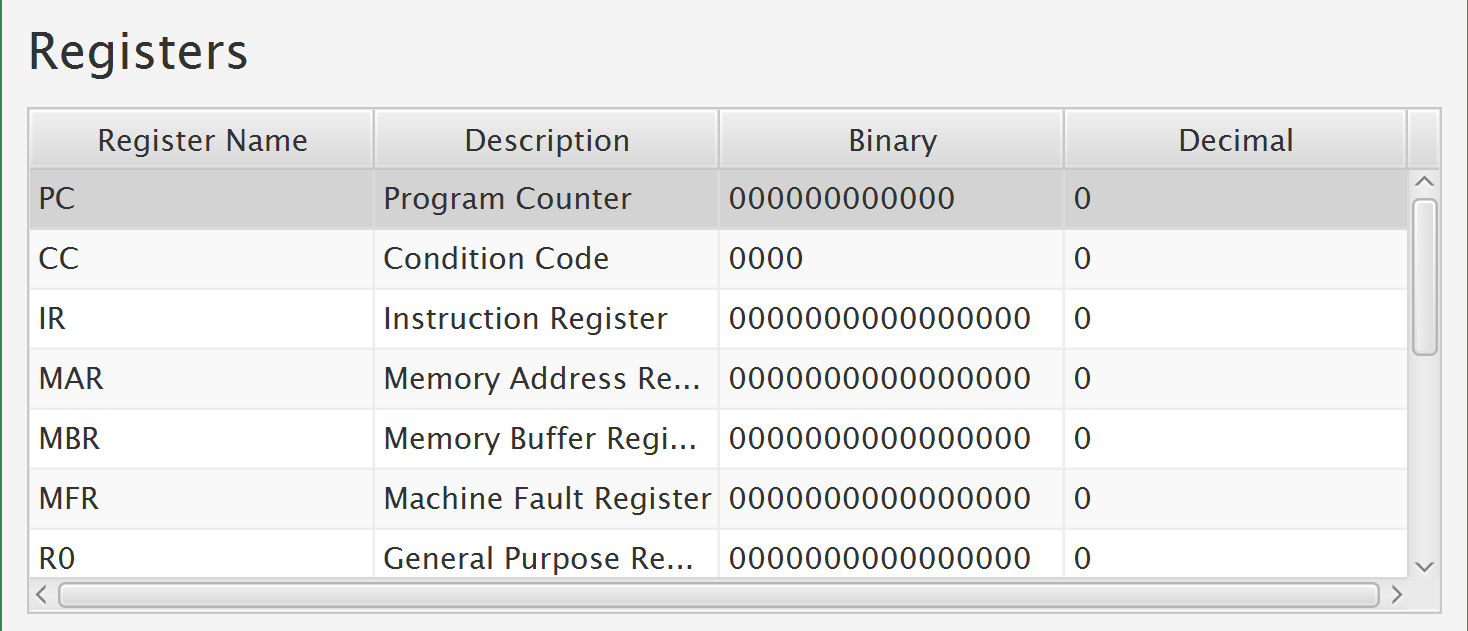
**Button [ReadFile]**

That will read “**program2\_paragraph.txt**” and store each word into memory.

**Button [InputAWord]**

This is the word you want to find in paragraph. The word will be stored in CPU.

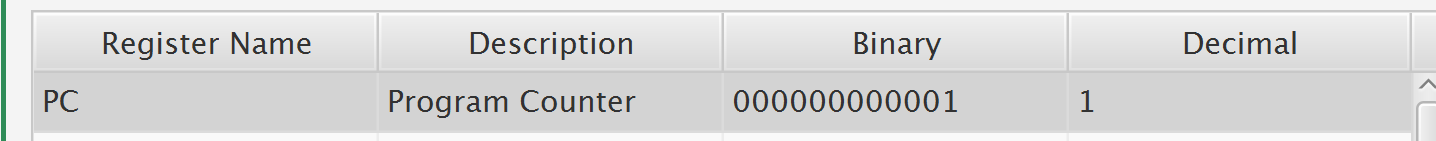
## Registers



1.We have implemented the following registers:

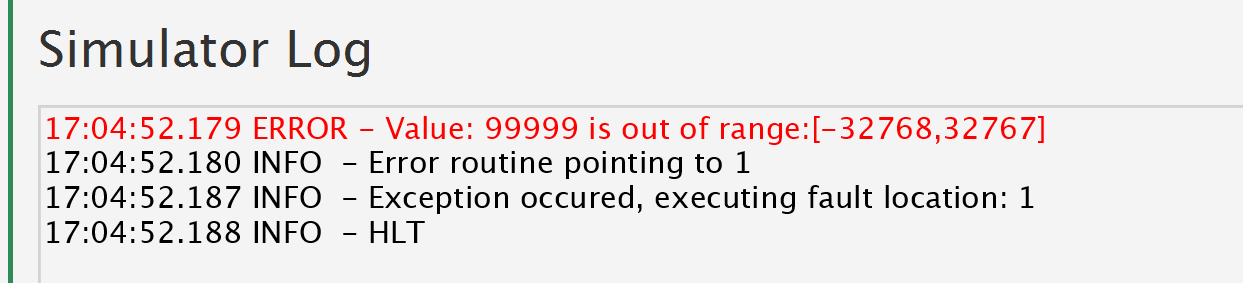
|  |  |
| --- | --- |
| **Mnemonic** | **Size** |
| PC | 12 bits |
| CC | 4 bits |
| IR | 16 bits |
| MAR | 16 bits |
| MBR | 16 bits |
| MFR | 4 bits |
| R0…R3 | 16bits |
| X1…X3 | 16 bits |
| FR0…FR1 | 16bits |

2. You can deposit a value to the register directly by double clicking a cell in “Binary” or “Decimal” column and setting a value.

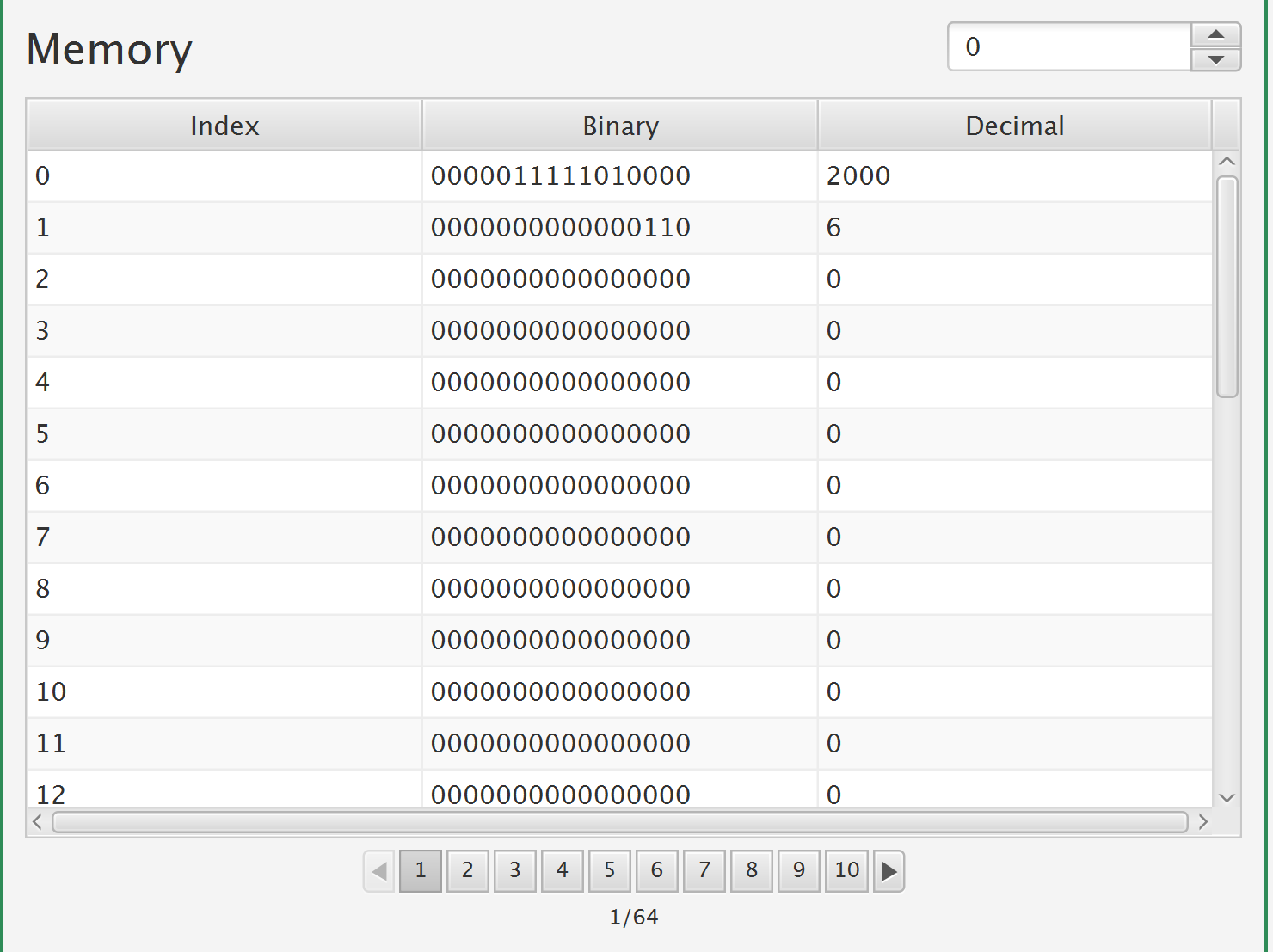




3.If the value is out of range, the simulator will throw Illegal Value Exception.



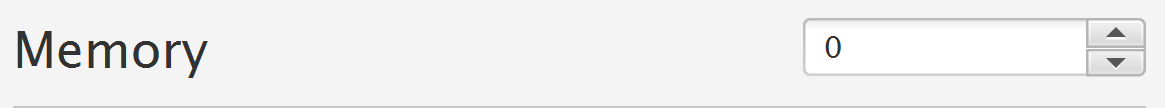
## Memory and Cache



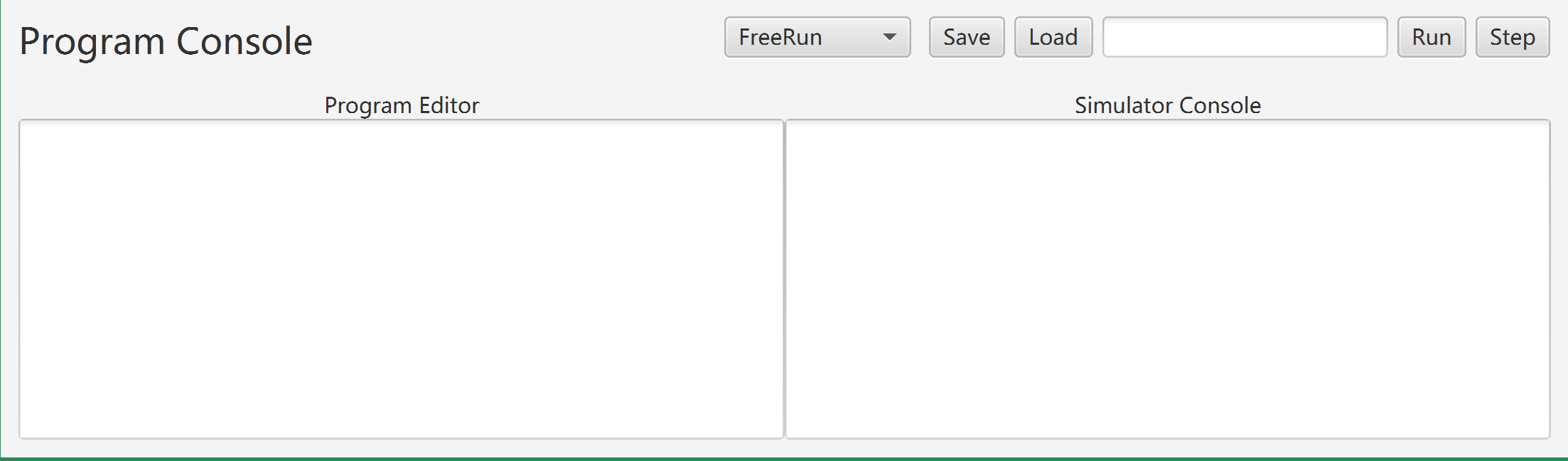
1. Memory is 2048 words size, 16-bit words.
2. Use **“BitSet”** to store data.
3. Set **MAR**, **MBR** when store and fetch.
4. Cache is implemented in **“src.Memory.MemoryCache.Java”**. For every store and fetch, we search the cache first. If the corresponding index is found in the cache, then simulator will logger “hit cache”. If the corresponding index is not found in the cache, put the index to the cache. When the cache is full, we use FIFO and remove the first.
5. We have stored some value in memory for **TRAP** and **Machine Fault**

|  |  |  |  |
| --- | --- | --- | --- |
| memory[0] | 0000000000000111 | 2000 | Trap table start |
| memory[1] | 0000000000000110 | 6 | Machine Fault Trap instruction location |
| memory[6] | 0000000000000000 | 0 | HLT |
| memory[2000] | 0000000000010100 | 6 | Routine 0 |
| memory[2001] | 0000000000010100 | 6 | Routine 1 IllegalMemoryAccess |
| memory[2002] | 0000000000010100 | 6 | Routine 2 IllegalTrapCode |
| memory[2003] | 0000000000010100 | 6 | Routine 3 IllegalValue |
| memory[2004] | 0000000000010100 | 6 | Routine 4 IllegalOpcode |
| memory[2005] | 0000000000010100 | 6 | Routine 5 IllegalRegisterAccess |
| memory[2006] | 0000000000010100 | 6 | Routine 6 |
| memory[2007] | 0000000000010100 | 6 | Routine 7 |
| memory[2008] | 0000000000010100 | 6 | Routine 8 MemoryOutOfBounds |
| memory[2009] | 0000000000010100 | 6 | Routine 9 |
| memory[2010] | 0000000000010100 | 6 | Routine 10 |
| memory[2011] | 0000000000010100 | 6 | Routine 11 |
| memory[2012] | 0000000000010100 | 6 | Routine 12 |
| memory[2013] | 0000000000010100 | 6 | Routine 13 |
| memory[2014] | 0000000000010100 | 6 | Routine 14 |
| memory[2015] | 0000000000010100 | 6 | Routine 15 |

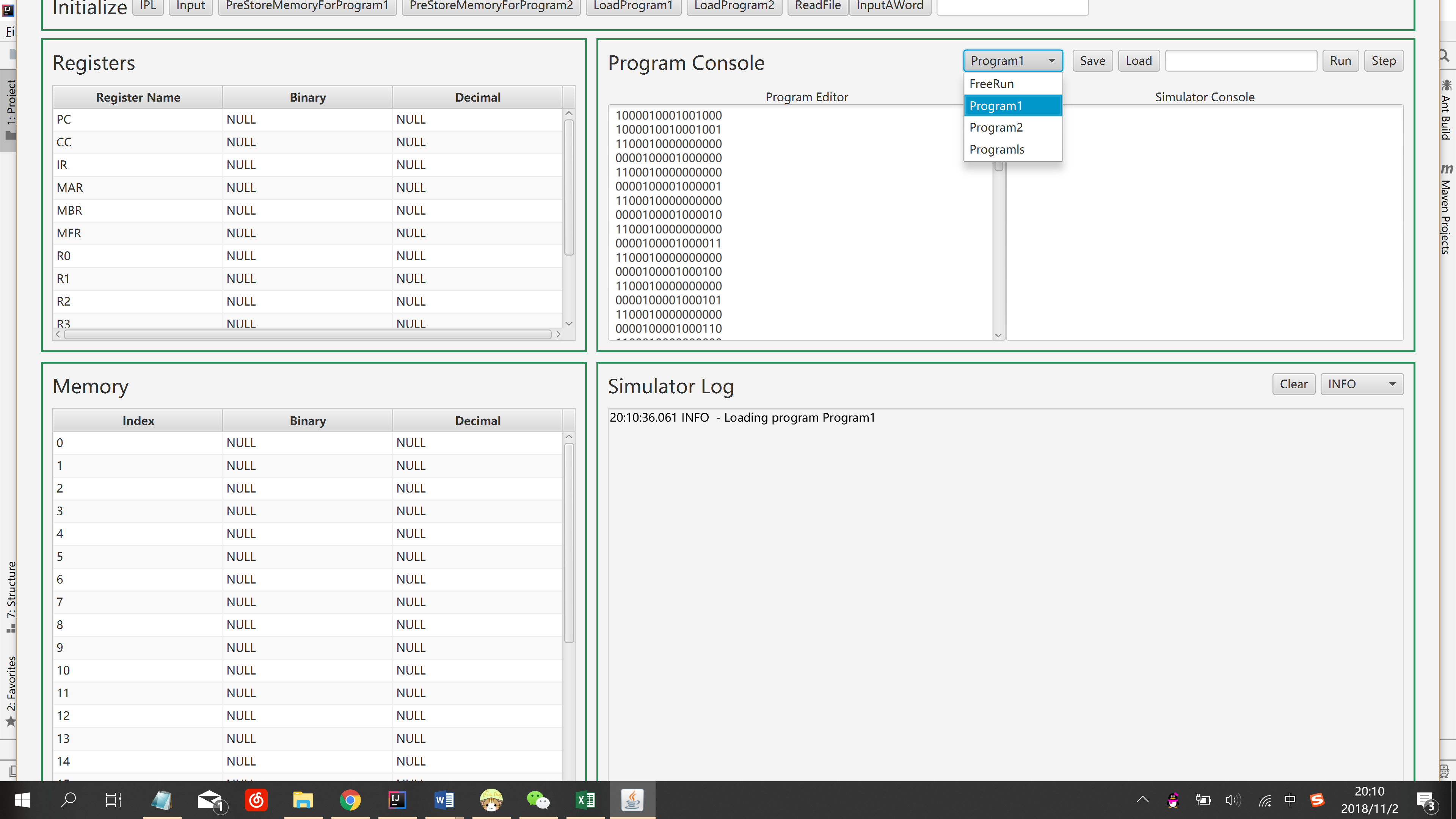
1. Memory can be changed directly by double clicking cells and setting a different value, which is analogous to what we have done with registers
2. You can go to any location in memory by inputting an index here.



## Program Console



***ComboBox [programNameSelector]***



You can select some programs that are already stored. And it will automatically set instructions in Program Editor (***TextArea [programContents])***.

***Button [Save]***

By clicking Save, it will save the current instructions of the selected program in our Program Editor.

***Button [Load]***

By clicking Load, it will load the instructions of selected program and store them into memory.

If **TextField [StartIndex]** is empty, the default location is memory[32], the program will be load to memory start from 32 and set PC = 32;

If you input a number in **TextField [StartIndex]**, the program will be load to memory start from the input number and set PC = input number.

***Button [Run]***

Run the program until HLT.

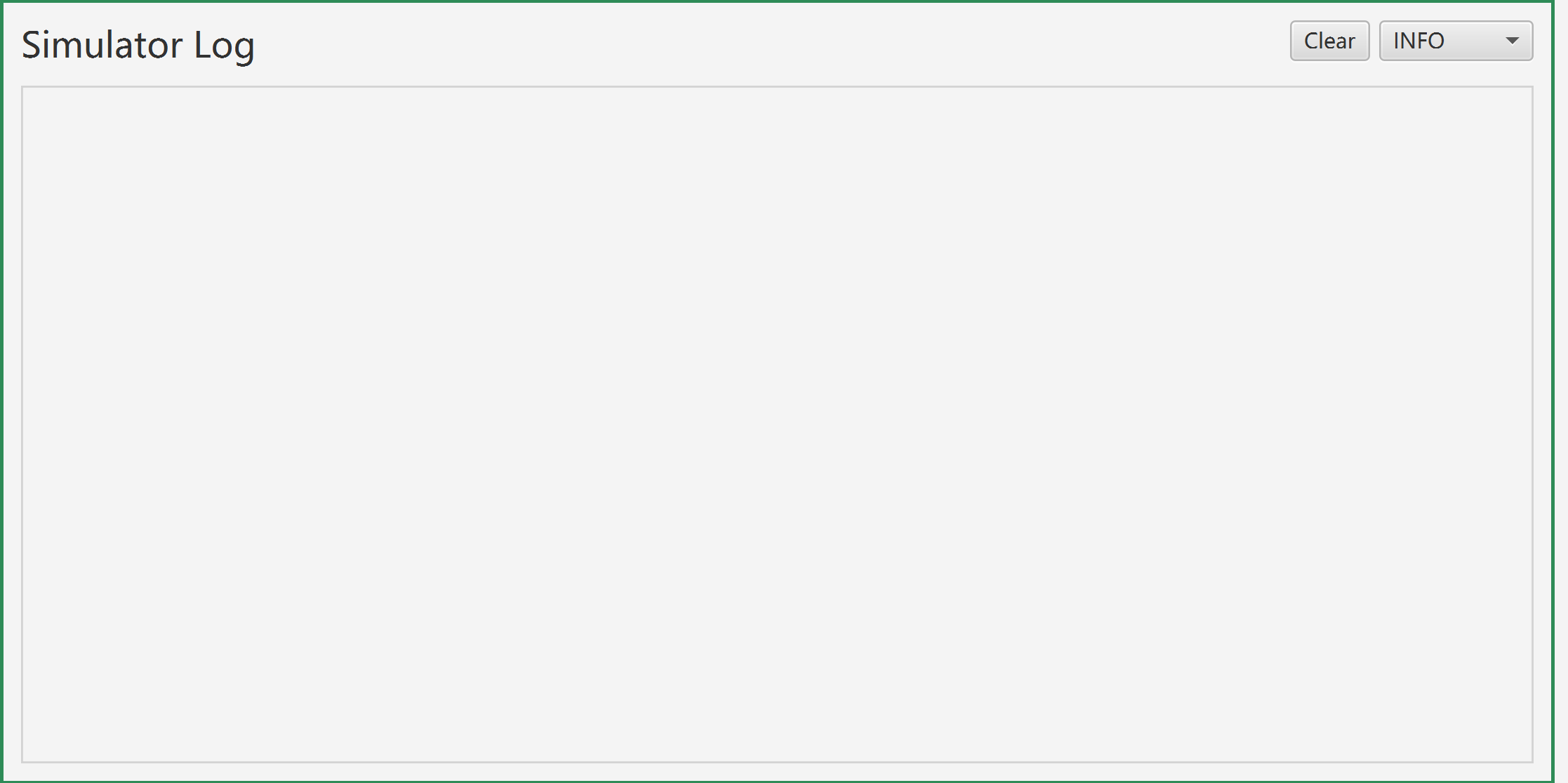
***Button [Step]***

Run single Step.

***TextArea [console]***

If the simulator executes instruction OUT, the output will show in the ***TextArea [console].***

## Simulator Log



***Button [Clear]***

Clear the contents in the log.

***ComboBox [logLevels]***

Select different level of log to show.

