CSCI-6461

Class Project

User’s Guide for Project Phase 2

Roger Zhang
Guanyu Zuo
Yi Zheng
Alan Yang

1. User’s Guide Overview

This User’s Guide for the Class Project Phase 1 consists of six main parts:

- User’s Guide Overview
- Main GUI
- Buttons
- Indicators
- Cache indicator
- Virtual Keyboard
- Run Test Program 1
- FAQ

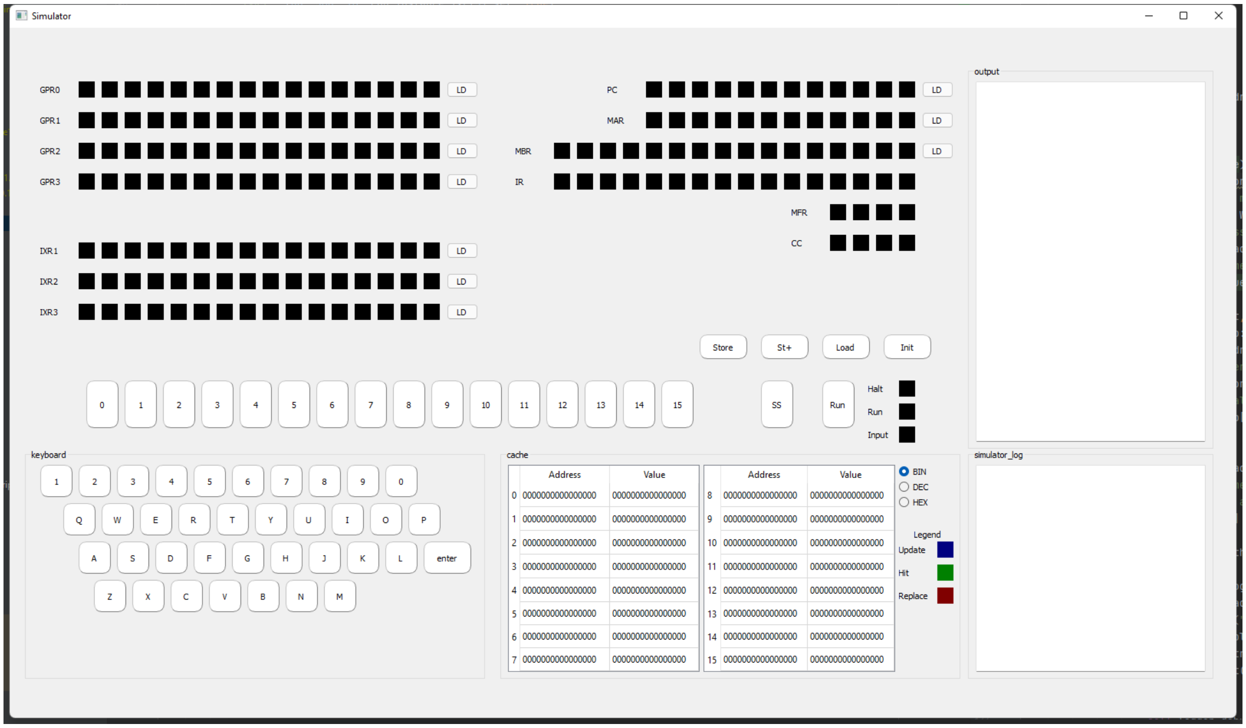
2. Main GUI

How to open the main GUI?

In order to open the main GUI, please open the **run_gui.bat** in the folder [delivery].

run_gui.bat

The main GUI consists of register indicator, cache indicator, switch, signal, virtual keyboard, output box and logging bux, the simulator interface and the debug console which will be showing below.



	Address	Value
0	0000000000000000	0000000000000000
1	0000000000000000	0000000000000000
2	0000000000000000	0000000000000000
3	0000000000000000	0000000000000000
4	0000000000000000	0000000000000000
5	0000000000000000	0000000000000000
6	0000000000000000	0000000000000000
7	0000000000000000	0000000000000000

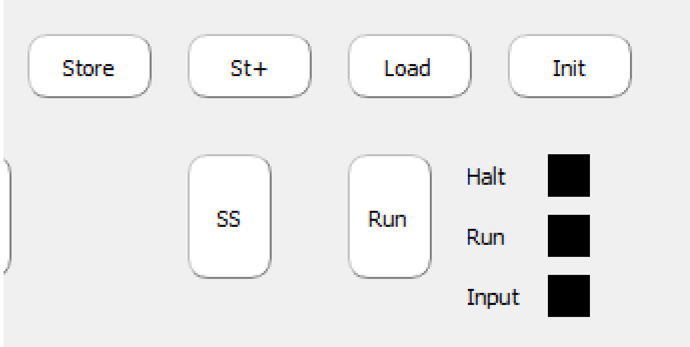
	Address	Value
8	0000000000000000	0000000000000000
9	0000000000000000	0000000000000000
10	0000000000000000	0000000000000000
11	0000000000000000	0000000000000000
12	0000000000000000	0000000000000000
13	0000000000000000	0000000000000000
14	0000000000000000	0000000000000000
15	0000000000000000	0000000000000000

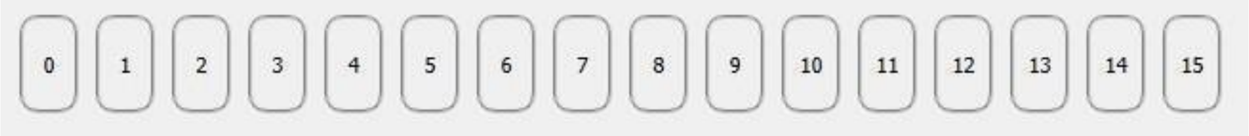
This is what the main GUI looks like upon start. We use Pyqt5 library in Python as the course recommended to construct the whole user's interface.

As we can see, the user interface of our emulator is functionally consistent with the requirements. Also, there is a debug console for the output. For those buttons and indicators, this user's guide will cover below.

3. Buttons

Our emulator has few major control buttons and ,few LD buttons for each registers and few number buttons to store numbers as shown below:





These buttons above are used to control the emulator.

For the detail functions of major control buttons:

[Store]:

This button stores the content of the MBR register to memory at the address specified by the content of the MAR register.

[St+]:

This button does what the “Store” button does and increments the MAR register by one.

[Load]:

This button loads the memory content at the address specified by the content of the MAR register to the MBR register.

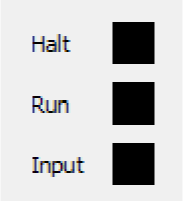
[Init]:

This button clears the contents of all registers.

[SS]:

This button runs a single stage (fetch / decode / execute).

4. **Indicator**



[Halt]:

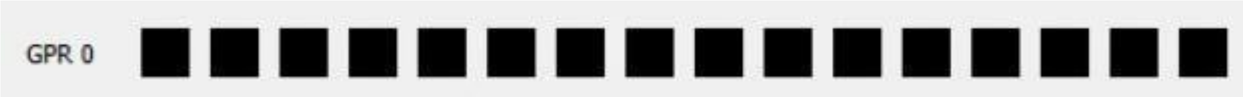
When this indicator is checked, the emulator is in halt.

[Run]:

When this indicator is checked, the emulator is in running.

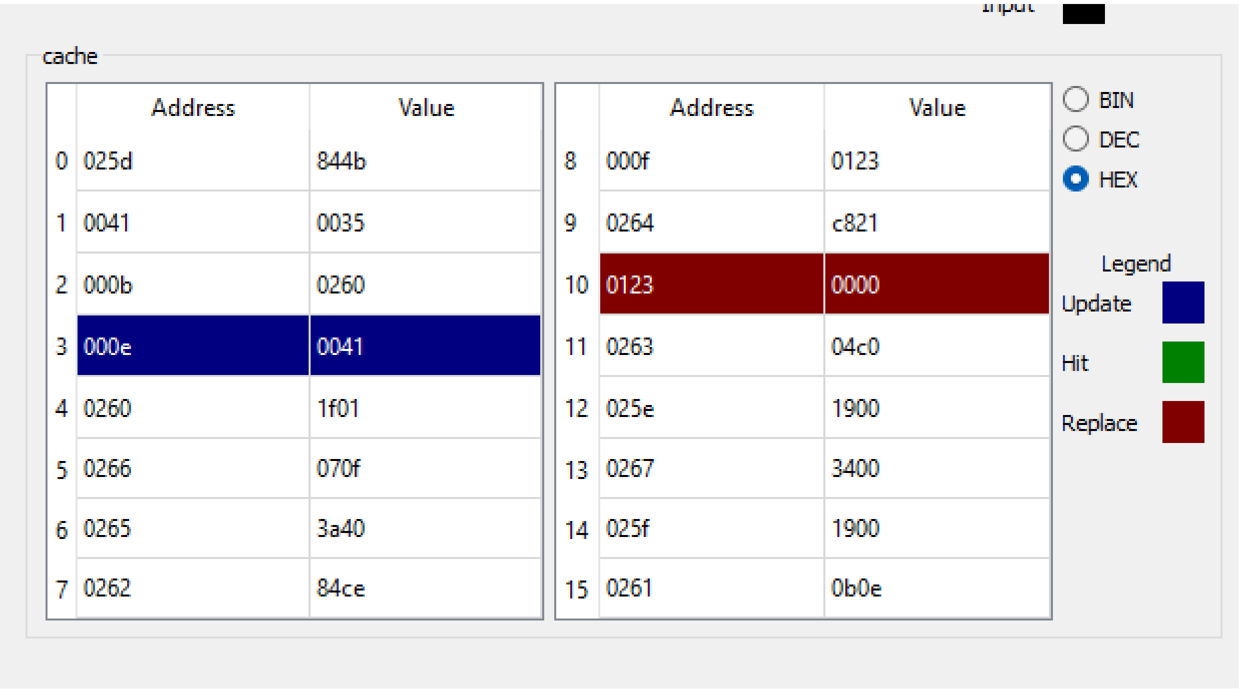
[Input]:

When this indicator is checked, the emulator is waiting for user input from keyboard.



Indicators like this show the memory address of each certain register.

5. Cache Indicator



	Address	Value
0	025d	844b
1	0041	0035
2	000b	0260
3	000e	0041
4	0260	1f01
5	0266	070f
6	0265	3a40
7	0262	84ce

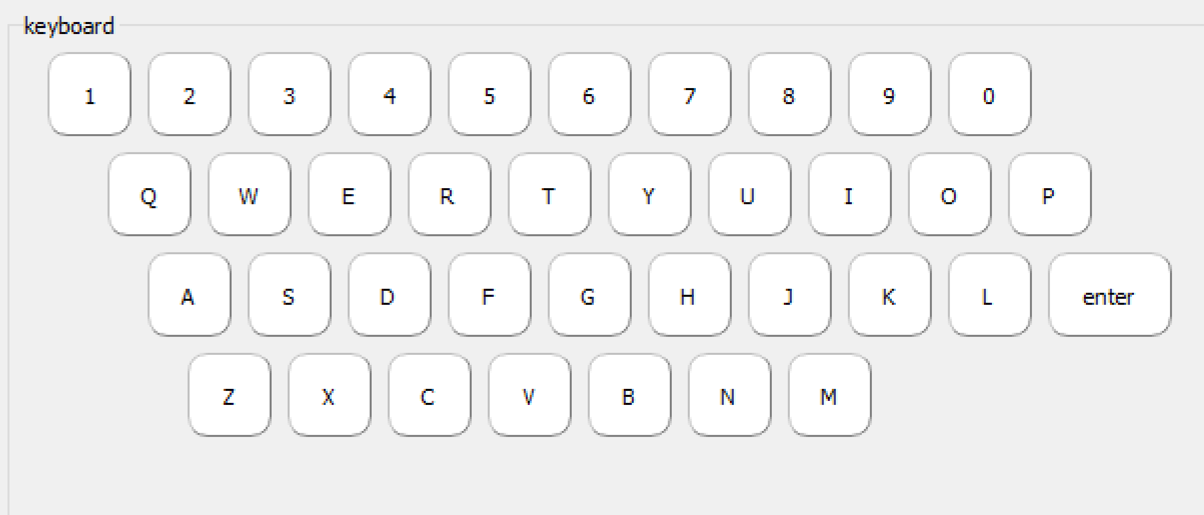
	Address	Value
8	000f	0123
9	0264	c821
10	0123	0000
11	0263	04c0
12	025e	1900
13	0267	3400
14	025f	1900
15	0261	0b0e

○ BIN
○ DEC
● HEX

Legend	
Update	🟦
Hit	🟩
Replace	🟥

This is what cache indicator will look like after the test program. You can choose the indicator's format by selecting through ratio button to change format between hexadecimal, decimal and binary. Also, the last hit, updated and replaced cache will be marked with different color.

6. Virtual Keyboard



This virtual keyboard serves as the input for simulator's program. Please note that this keyboard is only effective when the Input indicator is on.

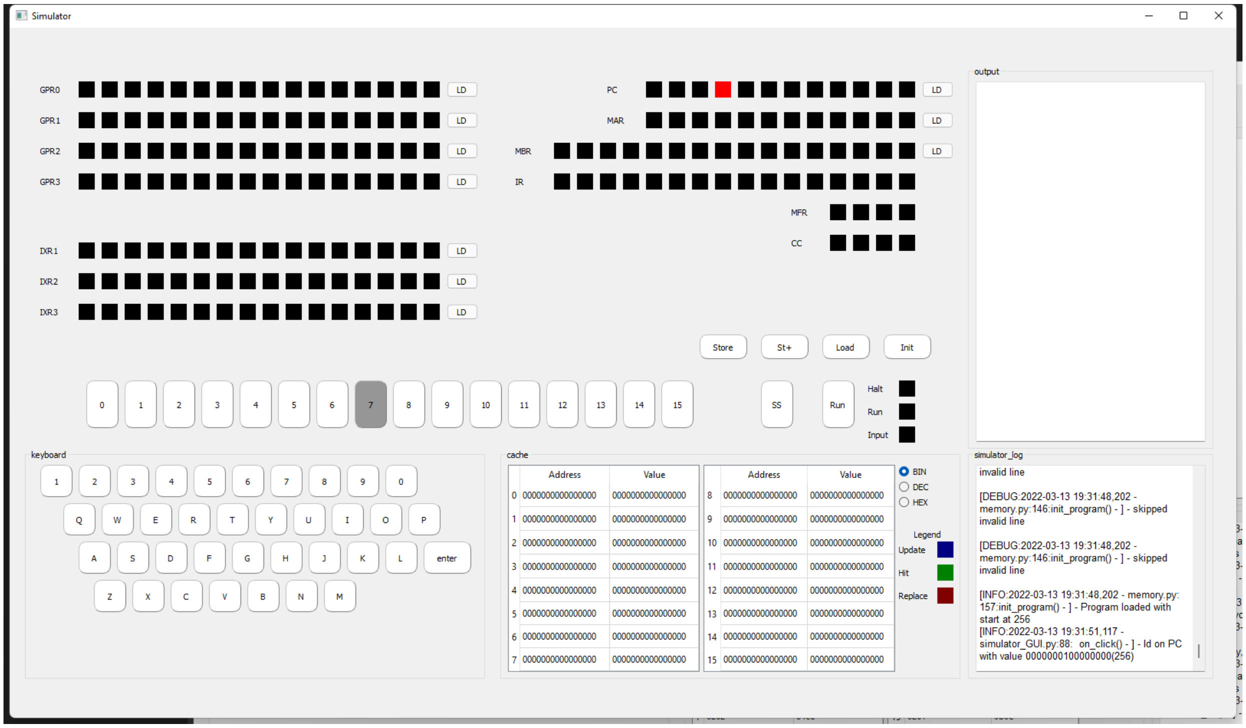
When Input indicator is on, the program will pause and wait for user's keyboard input.

After hitting a button in keyboard, the program will read the input into desired register and resume the program.

7. Run program 1

- In the delivery folder, you can double click on **run_gui.bat** to start the GUI.

- After GUI starts, either choose to use switch and LD button to load custom data or use **init** button to load the IPL.txt in the same directory. You can simply change the IPL.txt to load your own program.
- After hitting the init button, enter the start address using the LD button on the left of the PC indicator.(The program starts at 0x0100, 256)
- Now the simulator will look like below and ready to run.



● BIN ○ DEC ○ HEX

	Address	Value		Address	Value
0	0000000000000000	0000000000000000	8	0000000000000000	0000000000000000
1	0000000000000000	0000000000000000	9	0000000000000000	0000000000000000
2	0000000000000000	0000000000000000	10	0000000000000000	0000000000000000
3	0000000000000000	0000000000000000	11	0000000000000000	0000000000000000
4	0000000000000000	0000000000000000	12	0000000000000000	0000000000000000
5	0000000000000000	0000000000000000	13	0000000000000000	0000000000000000
6	0000000000000000	0000000000000000	14	0000000000000000	0000000000000000
7	0000000000000000	0000000000000000	15	0000000000000000	0000000000000000

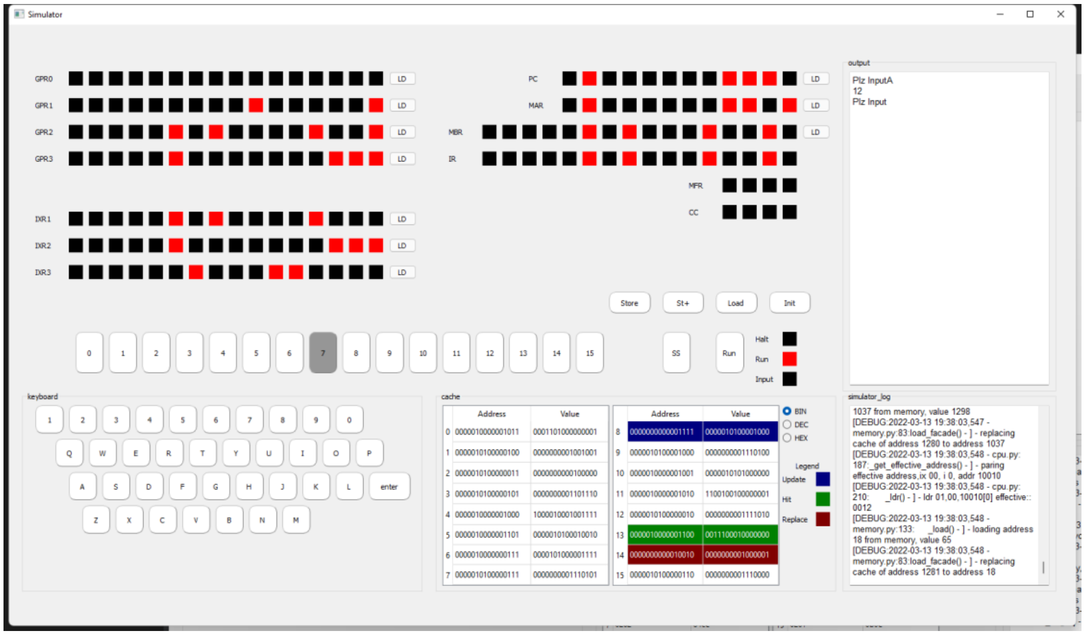
```
invalid line

[DEBUG:2022-03-13 19:31:48,202 -
memory.py:146:init_program() - ] - skipped
invalid line

[DEBUG:2022-03-13 19:31:48,202 -
memory.py:146:init_program() - ] - skipped
invalid line

[INFO:2022-03-13 19:31:48,202 - memory.py:
157:init_program() - ] - Program loaded with
start at 256
[INFO:2022-03-13 19:31:51,117 -
simulator_GUI.py:88:  on_click() - ] - ld on PC
with value 0000000100000000(256)
```

- Hit the **run** button to run the whole program or **SS** button for single step, The program will output Plz Input[A-U] to ask user input a number, This process will loop for 20 times and ask the user to enter a final number(Plz inputU)
- Please use digit button in virtual keyboard to input number digit and use enter to confirm a valid input. For example, after input indicator is on, click 1. And wait for the input indicator. Then click 2. And wait for the input indicator again. Then click enter. With this procedure, you have entered 12 as the first input into the program. The screenshot below shows how the GUI looks like after entering 12

- 

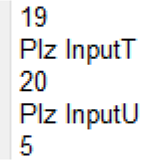
```
Plz InputA
12
Plz Input
```

	Address	Value
0	0000010000001011	0001101000000001
1	0000010100000100	0000000001001001
2	0000010100000011	0000000000100000
3	0000010100000101	0000000001101110
4	0000010000001000	1000010001001111
5	0000010000001101	0000010100010010
6	0000010000000111	0000101000001111
7	0000010100000111	0000000001110101

	Address	Value
8	0000000000001111	0000010100001000
9	0000010100001000	0000000001110100
10	0000010000001001	0000010101000000
11	0000010000001010	1100100100000001
12	0000010100000010	0000000001111010
13	0000010000001100	0011100010000000
14	0000000000010010	0000000001000001
15	0000010100000110	0000000001110000

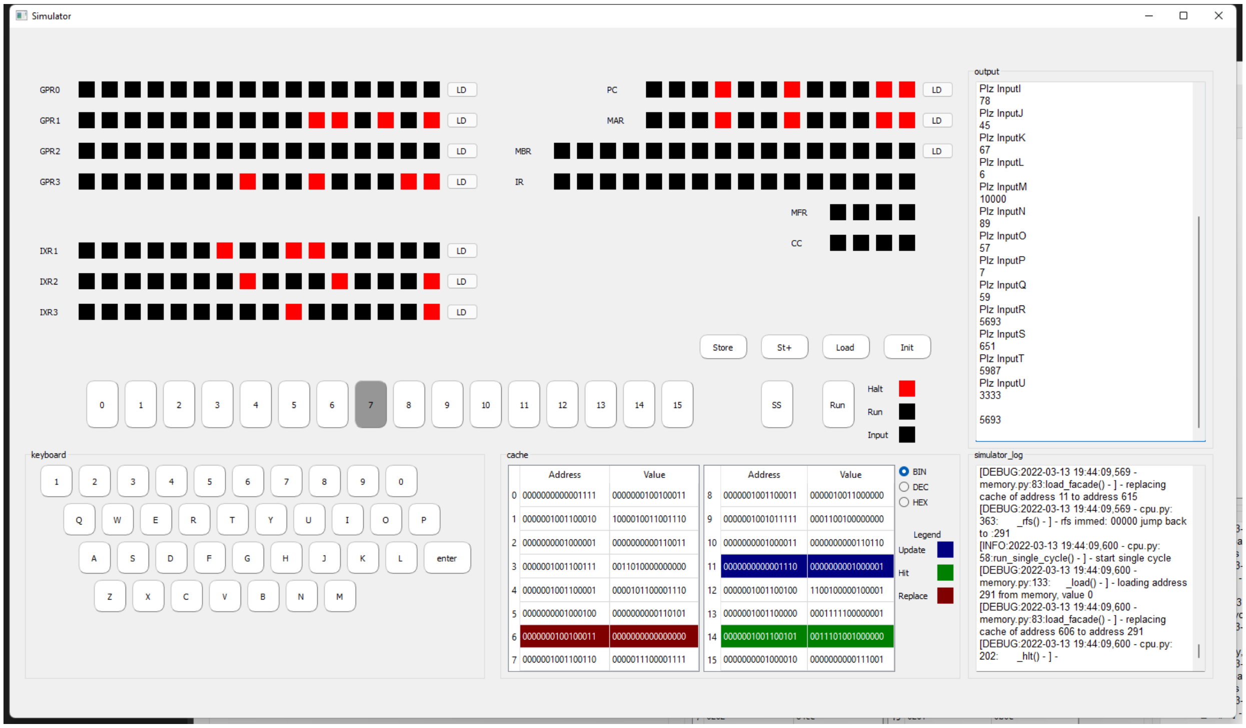
```
1037 from memory, value 1298
[DEBUG:2022-03-13 19:38:03,547 -
memory.py:83:load_facade() - ] - replacing
cache of address 1280 to address 1037
[DEBUG:2022-03-13 19:38:03,548 - cpu.py:
187:_get_effective_address() - ] - paring
effective address,ix 00, i 0, addr 10010
[DEBUG:2022-03-13 19:38:03,548 - cpu.py:
210:      _ldr() - ] - ldr 01,00,10010[0] effective::
0012
[DEBUG:2022-03-13 19:38:03,548 -
memory.py:133:      _load() - ] - loading address
18 from memory, value 65
[DEBUG:2022-03-13 19:38:03,548 -
memory.py:83:load_facade() - ] - replacing
cache of address 1281 to address 18
```

- This procedure will loop for 20 times and ask user to enter a final number(Plz Input U)

- 

```
19
Plz InputT
20
Plz InputU
5
```

- After Taking the final number, the program will calculate the closest number in the previous 20 numbers to the final number and print the closest number.



```
Plz InputI
78
Plz InputJ
45
Plz InputK
67
Plz InputL
6
Plz InputM
10000
Plz InputN
89
Plz InputO
57
Plz InputP
7
Plz InputQ
59
Plz InputR
5693
Plz InputS
651
Plz InputT
5987
Plz InputU
3333

5693
```

	Address	Value
0	0000000000001111	0000000100100011
1	0000001001100010	1000010011001110
2	0000000001000001	0000000000110011
3	0000001001100111	0011010000000000
4	0000001001100001	0000101100001110
5	0000000001000100	0000000000110101
6	0000000100100011	0000000000000000
7	0000001001100110	0000011100001111

	Address	Value
8	0000001001100011	0000010011000000
9	0000001001011111	0001100100000000
10	0000000001000011	0000000000110110
11	0000000000001110	0000000001000001
12	0000001001100100	1100100000100001
13	0000001001100000	0001111100000001
14	0000001001100101	0011101001000000
15	0000000001000010	0000000000111001

```
[DEBUG:2022-03-13 19:44:09,569 -
memory.py:83:load_facade() - ] - replacing
cache of address 11 to address 615
[DEBUG:2022-03-13 19:44:09,569 - cpu.py:
363:      _rfs() - ] - rfs immed: 00000 jump back
to :291
[INFO:2022-03-13 19:44:09,600 - cpu.py:
58:run_single_cycle() - ] - start single cycle
[DEBUG:2022-03-13 19:44:09,600 -
memory.py:133:      _load() - ] - loading address
291 from memory, value 0
[DEBUG:2022-03-13 19:44:09,600 -
memory.py:83:load_facade() - ] - replacing
cache of address 606 to address 291
[DEBUG:2022-03-13 19:44:09,600 - cpu.py:
202:      _hlt() - ] -
```

- The simulator output will look like below after inputting some random number and 3333 as final number. As can be seen the program successfully calculated the closest number 5693 and output. The complete output is like this.

```
Plz InputA
12
Plz InputB
233
Plz InputC
908
Plz InputD
609
Plz InputE
56900
Plz InputF
7777
Plz InputG
6790
Plz InputH
54
Plz InputI
78
Plz InputJ
45
Plz InputK
67
Plz InputL
6
Plz InputM
10000
Plz InputN
89
Plz InputO
57
Plz InputP
7
Plz InputQ
59
Plz InputR
5693
Plz InputS
651
Plz InputT
5987
Plz InputU
3333

5693
```

FAQ:

If the GUI failed to start using the run_gui.bat, you can choose to run the GUI using **simulator_GUI/simulator_GUI.exe**

If you are running the GUI from here, please replace the **simulator_GUI/IPL.txt** for loading custom programs.